Titre: Title:	Deep Representation Learning of 3D Human Motion with Recurrent Neural Networks
Auteur: Author:	Félix Gingras Harvey
Date:	2021
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Gingras Harvey, F. (2021). Deep Representation Learning of 3D Human Motion with Recurrent Neural Networks [Thèse de doctorat, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/6261/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Deep Representation Learning of 3D Human Motion with Recurrent Neural Networks

FÉLIX GINGRAS HARVEY
Département de génie informatique et génie logiciel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie informatique

Avril 2021

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Deep Representation Learning of 3D Human Motion with Recurrent Neural Networks

présentée par **Félix GINGRAS HARVEY**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Thomas HURTUT, président
Christopher J. PAL, membre et directeur de recherche
Lama SÉOUD, membre
Michiel VAN DE PANNE, membre externe

DEDICATION

To Joachim "grand-papa Jo" Gingras.

ACKNOWLEDGEMENTS

First, I want to offer my deepest thanks to Stephanie Saint-Pierre, who met me at the very beginning of this long journey, and has been by my side ever since, through all the obstacles, and uncertainties, through the highs and the lows of the Ph.D. road. In the words of Alex Ebert and Jade Castrinos, *home is wherever I'm with you.*

A huge thank you to Julien Roy and Paul Barde for all those conversations over coffee (or beer) and all the laughs that were inevitably part of it. Thank you for your constant help and support. I am infinitely grateful that you have been part of my journey during these last years, and can't wait to witness the next amazing steps in your respective paths.

Thanks to my parents Pierre Harvey and Louise Gingras, who always encouraged me in my studies and my sister Mariane Gingras Harvey, the ultimate gift expert, despite my few calls. One may not choose one's family, but I wouldn't choose any other.

Thanks to Michel Gagnon for encouraging me and convincing me some years ago to pursue graduate studies. You might have seen in me more than I did myself.

Many thanks to Christopher Pal for taking me under his supervision in 2014 and introducing me to the Mila lab. Having you on my side has been very rewarding, and you were always a great partner to rationally remind me how Reviewer 2 was wrong. Please keep giving your black hole speech on demand!

I also thank Daniel Holden and Mike Yurick for their valuable help and guidance throughout my Ph.D., as well as for being inspiring colleagues.

Thanks also to Yves Jacquier and Olivier Pomarez from Ubisoft La Forge, who have built over the last years a truly great research lab and have given me the chance to be part of it from the start.

Finally, thanks to all my friends and relatives who know nothing about machine learning and who helped me during all these years to keep a necessary balance.

RÉSUMÉ

Introduction

Que ce soit au cinéma ou dans les jeux vidéos, une expérience immersive, dans laquelle un spectateur ou un utilisateur peut se perdre pendant des heures, est souvent gage de succès. L'immersion est souvent grandement influencée par le réalisme du contenu présenté, même lorsque ce contenu fait partie d'une histoire imaginaire ou fantastique. Plusieurs facettes du contenu contribuent au réalisme de l'expérience, et le mouvement des personnages est l'un de ces éléments importants. Par exemple, des mouvements s'éloignant trop des lois de la physique, ou qui ne respectent pas l'intuition établie par la physionomie d'un personnage pourront briser l'illusion et nuire à l'expérience utilisateur. En animation, plusieurs règles existent afin de clarifier et éviter certains de ces problèmes, et les technologies ne cessent d'être améliorées pour continuellement produire des animations de plus en plus réalistes.

Parmi ces technologies, celles de la capture de mouvements (ou MOCAP pour *motion capture*) sont devenues des standards dans plusieurs industries pour recréer les mouvements humains dans un environnement 3D avec une grande fidélité. Ces approches sont souvent basées sur un ensemble de caméras infra-rouge à haute fréquence suivant plusieurs marqueurs réflectifs dans l'espace avec une grande précision. Le mouvement de ces marqueurs peut ensuite contrôler le mouvement du squelette d'un personnage dans un environnement 3D virtuel. Cette approche est très utile pour l'animation de personnages de formes humaines ou humanoïdes, pour lesquels nous sommes en général très sensibles aux artefacts. Le problème majeur de la capture de mouvement, et qui reviendra comme motivation dans chaque chapitre de cette thèse, est l'ensemble des coûts qui en découlent. Effectivement, les technologies de capture de mouvements sont très dispendieuses à utiliser, et requièrent une panoplie d'étapes longues et coûteuses de post-traitement pour arriver à une animation finale. Nous nous penchons donc dans les travaux présentés ici sur des techniques qui permettront de réduire les coûts associés à la création d'animations, soit par la facilitation de la ré-utilisation de séquences de MOCAP, soit par la génération directe de mouvements. Malgré leur grande demande en ressources, la qualité supérieure des systèmes de capture de mouvements a tout de même poussé plusieurs studios de jeux et de films à les utiliser massivement dans les dernières années, menant à de vastes bases de données de mouvements. Cela ouvre donc la porte aux techniques basées sur les données et plus précisément à l'apprentissage machine.

Effectivement, une vague de progrès incessante des approches d'apprentissage machine a pu être observée durant la dernière décennie, en partie causée par l'augmentation de la puissance

de calcul des ordinateurs et à l'accessibilité à de plus en plus de données. À ces éléments clés se sont ajoutés plusieurs progrès dans les méthodes utilisées, améliorant davantage les résultats. Les progrès les plus marquants en apprentissage profond ont d'abord été concentrés surtout sur la vision par ordinateur appliquée aux images et le traitement du langage naturel, mais s'appliquent de plus en plus dans d'autres sphères, comme le traitement audio et vidéo, la modélisation de protéines, les prédictions climatiques, la prédiction de risque d'infection et plusieurs autres. Ces techniques s'appliquent aussi au domaine de l'animation qui est le domaine d'intérêt ici. Nous verrons dans cette thèse comment l'apprentissage profond, par l'entremise de réseaux de neurones artificiels récurrents, nous permet l'obtention de représentations latentes du mouvement humain permettant plusieurs tâches utiles, comme la classification, la décompression, la reconstruction, le dé-bruitage, ou la génération de mouvements. Les réseaux de neurones récurrents sont en effet un choix naturel pour la modélisation du mouvement humain, puisqu'ils sont spécifiquement conçus pour traiter des séquences temporelles. Ces réseaux possèdent effectivement des connexions récurrentes (temporelles) leur permettant ainsi de compresser et transmettre l'information du contexte passé dans leur représentation interne. Ils peuvent donc modéliser les dépendances temporelles de séquences de longueurs arbitraires dans leur mémoire interne et d'effectuer plusieurs tâches en aval qui utilisent ces informations. Dans les différents chapitres de cette thèse, nous nous concentrons principalement sur deux de ces tâches possibles, soit la reconnaissance de mouvements (Chapitre 3) et la génération de transition (Chapitres 3, 4, 5). Une motivation commune à ces tâches, et donc générique à cette thèse, est la réduction de coûts élevés liés à la capture de mouvements. La reconnaissance de mouvements permet d'automatiser la description sémantique de séquences existantes, qui sont souvent entièrement décrites par un nom de fichier lié à une production précise. La classification d'actions peu donc faciliter grandement la recherche et la ré-utilisation d'actions déjà capturées. La génération de transitions permet quant à elle de produire le mouvement liant un contexte passé à une trame-clé visée. En utilisant un système d'apprentissage machine, basé sur des données réelles de mouvements humains, cela permet d'atteindre une qualité digne de la capture de mouvement dans un contexte d'animation par trame-clés et donc de faciliter la tâche d'animateurs. Nous résumons sommairement ici nos approches liées à ces efforts, groupées par publication.

Classification semi-supervisée et génération adverse de mouvements avec contraintes (Chapitre 3)

Dans cet article de journal, nous explorons principalement l'apprentissage multi-tâches semi-supervisé de séquences de positions de marqueurs 3D issus de la capture de mouvements.

Le but principal est d'y effectuer la reconnaissance de mouvement. Notre formulation semi-supervisée émerge de notre formulation de fonctions de perte multiples qui peuvent s'appliquer en partie sur des données non-annotées. En effet, notre fonction de perte optimise plusieurs objectifs simultanés pour chaque séquence. Certains de ces objectifs sont optionnels et ne requiert pas d'annotations (non-supervisé), comme la reconstruction par trame, et la reconstruction par séquence. Un autre objectif facultatif, la classification par trame requiert des annotations (supervisé), tout comme notre objectif principal: la classification de séquence. Nos objectifs non-supervisés nous permettent donc l'utilisation de bases de données sans annotation, qui sont souvent moins coûteuses à produire et donc plus facile à acquérir. Cela nous permet donc en pratique d'améliorer la performance de classification de mouvements obtenue avec peu de données annotées en utilisant des données additionnelles non-annotés, plus faciles d'accès.

Notre architecture contient plusieurs modules: un encodeur de trame, un décodeur de reconstruction de trame, un décodeur de classification de trame, un encodeur séquentiel (récurrent), un décodeur de reconstruction de séquence et un décodeur de classification de séquence. Certains de ces modules sont activés seulement si la fonction de coût contient un objectif qui les utilise. Notre architecture permet donc l'apprentissage semi-supervisé et ne requiert pas de séparer l'entraînement en deux phases distinctes (non-supervisée, puis supervisée).

Nos expériences principales sont conduites sur la base de données de mouvements HDM05, contenant 65 actions différentes, sur laquelle plusieurs travaux montrent des résultats qui peuvent parfois être trompeurs. En effet, nous montrons dans l'article comment plusieurs travaux précédents sur cet ensemble de données donnent une impression inexacte que la tâche de classification d'action est résolue grâce à une séparation des données peu réaliste. Nous ré-implémentons certains de ces travaux en utilisant une séparation basée sur les acteurs pour représenter une capacité de généralisation plus plausible, ce qui amène les meilleurs résultats de classification sur l'ensemble de test à passer de 96.92% à 81.64% de précision (Table 3.3), que nous utilisons comme nouvelle base de référence.

Nos résultats empiriques nous montrent que lorsque nous utilisons uniquement la base de données annotées HDM05 (Section 3.7.1), la reconstruction de trame comme seul objectif additionnel est le meilleur régularisateur, menant la précision à 86.14% de précision sur l'ensemble de test. Lorsque nous ajoutons aux données d'entraînement la base de données non-annotées CMU (Section 3.7.2), qui est beaucoup plus volumineuse, activer en prime la régularisation par reconstruction de séquence nous permet d'obtenir de meilleures représentations latentes menant à une précision de 87.40% sur l'ensemble de test HDM05. Notre approche, basée sur la reconstruction et la classification nous permet également le regroupe-

ment automatique de séquences d'actions similaires à un niveau sémantique (Section 3.7.3), permettant l'accélération d'annotations de données en grappe. Par contre, nous montrons aussi que les avantages de notre approche semi-supervisée sont moins importants lorsque l'ensemble de données annotées est beaucoup plus vaste qu'HDM05 (Section 3.7.4), notamment lorsque nous utilisons la très volumineuse base de données NTU-RGB+D.

Finalement, nous effectuons également dans ce chapitre une exploration préliminaire de la génération automatique de transitions, qui sera le sujet principal de la suite de la thèse. En utilisant une architecture similaire au modèle avec module de reconstruction de séquences (et sans couche de classification), nous explorons la prédiction de mouvements basée sur un contexte passé et conditionné sur une trame-clé visée dans un intervalle de temps défini. Nous ajoutons à l'approche une composante antagoniste avec l'utilisation d'un réseau discriminant pour donner aux séquences générées plus de réalisme, et afin d'éviter l'attraction vers la pose moyenne souvent observée avec des réseaux récurrents. Par contre la combinaison des réseaux récurrents aux réseaux antagonistes, deux types de modèles connus pour être difficiles à entraîner, mène vers des problèmes de stabilité. Nous proposons donc deux fonctions de pertes basée sur des contraintes physiques du mouvement (Section 3.5.6, Section 3.5.7) qui mènent vers de meilleures transitions (Tableau 3.7.5), ainsi qu'une stabilité accrue pour l'entraînement du générateur (Figure 3.6).

Réseaux récurrents de transitions pour la locomotion de personnages (Chapitre 4)

Dans cette note technique de conférence, nous explorons plus en profondeur la génération de transitions d'animations pour des caractères humains. Nous détaillons une architecture uniquement basée sur la reconstruction, similaire à celle présentée au Chapitre 3, avec l'ajout d'encodeurs à propagation avant pour chaque trame, et sans l'utilisation du vecteur latent complémentaire à l'état interne de l'encodeur récurrent. Nous augmentons également l'architecture avec un réseau d'initialisation d'état caché de la couche récurrente (Section 4.4.6) pour mieux performer avec un contexte passé restreint. Nous proposons aussi l'enseignement forcé probabiliste avec probabilité constante comme stratégie d'entraînement pour la gestion d'accumulation d'erreurs. Nous présentons l'apport de chacune de nos contributions dans une étude d'ablation (Tableau 3.5). De plus, nous définissons une approche de perception locale du terrain pour de meilleurs transitions sur terrain accidenté (Section 4.6.2). La performance de notre architecture nous permet également d'effectuer de la décompression temporelle d'animation (Section 4.6.4, en appliquant le réseau transitionnel en série pour chaque trame-clé sauvegardée. Notre approche comporte cependant plusieurs limites

l'empêchant de devenir un véritable outil pour un animateur. Premièrement, cette étude est effectuée sur des données de positions uniquement, limitant l'applicabilité à un personnage animé avec un maillage d'apparence associé. Deuxièmement, l'architecture n'obtient en entrée aucune information temporelle et doit effectivement apprendre une sorte de compteur interne pour savoir quand atteindre la trame clé. Cela n'offre donc pas de souplesse face à la longueur de la transition désirée. Troisièmement, la trame-clé cible dans nos expérience contient non seulement les positions visées, mais aussi l'information de vélocité de chaque joint, ce qui réduit l'ambiguïté sur le mouvement à produire pour atteindre la cible. Finalement, notre approche est par nature déterministe et ne permet donc pas l'échantillonnage de transition différentes pour un contexte donnée. Nous nous attaquons à toutes ces limitations dans le Chapitre 5.

Interpolation robuste du mouvement (Chapitre 5)

Dans cet article de conférence, nous montrons d'abord les limites de l'adaptation naïve des approches de prédiction de mouvement pour la génération de transition. Nous effectuons dans ce Chapitre notre analyse la plus poussée sur le sujet en commençant par l'élaboration d'une architecture basée sur les progrès récents des meilleures approches pour la prédiction de mouvement avec des réseaux récurrents. Ici, toutes nos expériences sont effectuées sur des données d'animation *complètes*, c'est-à-dire sur des données de rotations et de déplacement global, qui peuvent donc être utilisées pour des personnages de jeux vidéos avec maillage. Nous montrons comment l'ajout d'information de trame-clés cibles en entrée à un réseau de prédiction n'est pas suffisant pour produire un outil utile en pratique, mettant en lumière les limitations énumérées dans section ci-haut. Nous proposons donc deux différents modificateurs latents additifs pour s'adresser à ces limitations.

Notre premier modificateur est un code de délai restant (Section 5.3.3) permettant à la couche récurrente d'avoir une notion du nombre de trames restantes à générer avant la fin de la transition. Cela permet donc de moduler la vitesse ou la trajectoire du mouvement afin de respecter les contraintes de temps imposées par l'utilisateur. Notre code de délai restant est inspiré des réseaux de types *Transformateurs* largement utilisés dans le traitement du langage naturel, où un code est nécessaire pour indiquer la position du mot dans une phrase. Nous utilisons les même calculs pour notre code de délai restant, mais basés dans notre cas sur le nombre de trames restantes avant la cible au lieu de la position dans une phrase. Ce code est donc additionné aux représentations latentes fournies par les différents encodeurs à propagation avant du modèle. Ces représentations modifiées sont ensuite fournies à la couche récurrente. Cela permet donc de moduler la représentation latente des différentes entrées de

manière unique pour chaque stade (trame) de la transition. Ce signal est donc difficile à ignorer par le réseau de neurones et mène à de meilleurs résultats que l'ajout aux vecteurs d'entrée d'une dimension temporelle (Figure 5.4). Notre code de délai restant est donc un élément clé de l'architecture qui permet au modèle de bien gérer des transitions de longueurs différentes, un requis nécessaire à l'utilisation de l'outil en pratique.

Notre deuxième modificateur latent est un bruit de cible programmé (Section 5.3.4) appliqué seulement sur les représentations utilisant la trame-clé cible. Ce bruit est un vecteur échantillonné d'une distribution gaussienne multi-dimensionnelle sphérique, centrée à l'origine. Ce vecteur est échantillonné une seule fois par transition, mais un facteur de mise à l'échelle variable y est appliqué durant la transition. Ce facteur est égale à 1.0 lorsque le délai restant est assez grand, et réduit jusqu'à 0.0 lorsque le délai restant de transition est inférieur à 5 trames. En conséquence, les représentations touchées sont affectées proportionnellement à la durée de transition restante. Intuitivement cela peut représenter un agent avec une vue déformée de la cible à atteindre, mais qui se précise en s'y rapprochant. Le bruit de cible joue deux rôles simultanés. Premièrement, il permet d'avoir une meilleure robustesse face aux déformations des cibles et force le réseau de neurones à s'appuyer davantage sur le contexte passé de l'animation que sur la cible en début de transition. Deuxièmement, il permet l'échantillonnage stochastique de transitions pour un contexte et une cible donnés. Effectivement, garder le bruit de cible programmé activé durant l'utilisation de l'outil permet de moduler de façon aléatoire la perception de la cible en début de transition et donc de générer différentes transitions pour la même cible. Le niveau de stochasticité peut être contrôlé facilement en ajustant le facteur de mise à l'échelle, qui lorsque nul résulte en un modèle déterministe. Encore une fois, la modulation des représentations internes de l'architecture par l'addition a un impact considérable sur les résultats et empêche le réseau d'ignorer facilement la source de bruit aléatoire. Nous montrons comment une méthode plus simple d'ajout du bruit de cible aux entrées du réseaux de neurones a beaucoup moins d'impact sur la stochasticité des transitions générées (Figure 5.5).

Dans cet article, nous ajoutons également une fonction de perte antagoniste à l'entraînement afin d'améliorer les résultats visuels obtenus en réduisant les artefacts liés aux objectifs de reconstruction. Nous utilisons deux discriminateurs à propagation avant (Section 5.3.5) appliqués sur des fenêtres glissantes de transitions générées. Les différents discriminateurs prennent en entrées des fenêtres de différentes largeurs. Nous utilisons la formule antagoniste des moindres carrés (*Least-Square GANs*) et améliorons non seulement les résultats visuels, mais aussi les résultats quantitatifs de reconstruction.

Nos analyses empiriques sont présentées en trois parties. Nous explorons premièrement de

manière quantitative la performance de notre architecture de base, utilisant plusieurs avancées récentes, sur la tâche répandue de prédiction de mouvements (Section 5.4.1). Nous utilisons la base de données *Human 3.6M* (H36M) ainsi que le banc de tests commun de la littérature sur ces données. Nous validons que notre modèle de base, ainsi que notre représentation de données basée sur les quaternions mènent à des résultats compétitifs avec l'état de l'art en prédiction (Tableau 5.1). Deuxièmement, nous testons les performances de l'architecture dans le cadre de génération de transitions sur un sous-ensemble de H36M plus approprié pour des transitions de plus d'une demie seconde (Section 5.4.2). Nous analysons quantitativement ici l'ajout de nos différentes contributions sur trois différentes mesures de qualité, et notons des améliorations dans toutes les catégories de tests, sur toutes les longueurs de transitions analysées (Tableau 5.2). Dans ces tests, nous explorons la capacité des différentes variantes du modèle à généraliser à des transitions plus longues que celles vues durant l'entraînement. Cela est dans le but de ne pas imposer de limites strictes aux utilisateurs de l'outil face à la longueur des transitions. Finalement, nos résultats quantitatifs et qualitatifs les plus représentatifs de vrais scénarios d'animation sont présentés sur une nouvelle base de données de haute qualité appropriée pour la tâche de génération de transitions. Cette base de données maison, appelée LaFAN1 (Section 5.4.3), publiée avec l'article, est accessible gratuitement en ligne, avec le code nécessaire pour reproduire les résultats associés à nos bases de références dans le but de faciliter les comparaisons futures. Avec ces données plus complexes, nous voyons ici aussi que nos contributions mènent vers des améliorations quantitatives considérables (Tableau 5.3). Notre exploration qualitative est effectuée grâce à l'implémentation d'une extension personnalisée à l'application d'animation *Motion Builder*, largement utilisée dans l'industrie (Section 5.4.4). À travers cette extension un animateur peut facilement générer les trames manquantes d'une animation avec une interface usager simple et complète.

Discussion et directions futures

L'ensemble de nos travaux vise à réduire les problèmes liés à la dépendance des studios de divertissements sur la dispendieuse capture de mouvements. Ces problèmes sont pertinents pour plusieurs entreprises étant donnés les coûts considérables associés à la capture de nouvelles séquences ainsi qu'aux post-traitements nécessaires pour chacune d'elles. Nous nous attaquons d'abord aux difficultés liées à la réutilisation de telles séquences, engendrées souvent par des descripteurs (ex. noms de fichier) souvent trop limitées, peu descriptifs et suivant des conventions changeantes pour chaque production. Nous explorons donc une architecture permettant la classification de mouvements pouvant utiliser de grandes quantités de données non-annotées pour mieux performer avec peu d'annotations, un cas de figure courant en

réalité. Un tel système permet aussi l'accélération de d'annotation groupes de séquences par regroupement de clips similaires. Ensuite, à travers plusieurs travaux, nous nous attaquons à la génération automatique de mouvements. Plus précisément nous explorons différentes techniques et stratégies d'entraînement spécifiques à l'interpolation automatique entre trames-clés fournies par des animateurs, rappelant le flux de travail en animation classique. Le but ici est d'alléger le travail des animateurs en permettant la génération de transitions entre trame-clés éloignées dans le temps avec une qualité pouvant égaler celle obtenue par la MOCAP. Plusieurs leçons peuvent être tirées des études présentées ici.

D'abord, dans nos travaux au sein d'Ubisoft sur la classification de mouvements, nous avons rapidement été témoins des défis associés à définir une taxonomie adéquate des mouvements dans un contexte de production. Effectivement, trouver un vocabulaire satisfaisant pour tous les animateurs s'est avéré difficile. De plus, lorsque l'on considère tous les mouvements utiles dans la production de jeux vidéos variés, imposer une seule classe à chaque trame d'animation mène vers un vocabulaire trop volumineux. En effet, les combinaisons de mouvements possibles (par exemple *marcher* + *tourner* + *fusiller*) deviennent trop nombreuses. Nous avons donc modifié notre méthode en pratique pour effectuer de l'annotation multi-classes, ou la probabilité de chaque classe est prédite indépendemment des autres.

Ensuite, traiter des données de positions est considérablement plus facile pour un modèle que traiter des données angulaires. Effectivement, entre l'élaboration des systèmes présentés au Chapitre 4 et au Chapitre 5, un temps et des effort importants furent nécessaires pour passer à une représentation de mouvement angulaire et obtenir des résultats visuellement semblables sur un squelette sans maillage. Il semble en effet plus simple de prédire les déplacements dans l'espace euclidien 3D défini par les positions que l'espace défini par la représentation angulaire choisie (dans notre cas les quaternions). Nous émettons les hypothèses que cette différence est causée par (1) le simple fait que les données de rotation contiennent plus d'information que les positions et (2) que le mouvement des positions est plus *linéaire* que le mouvement angulaire, c'est-à-dire que le positions agissent de façon linéaire sur une plus grande échelle temporelle.

Nous avons également été témoins à travers nos travaux de la difficulté de s'adresser à un problème où les données critiques à l'apprentissage sont aussi les plus rares. En effet, dans le cas de la génération automatique de transitions, les séquences d'animations disponibles sont souvent dominées par des mouvements de locomotion simples (ex. *marcher* ou *courir*), qui sont aussi les plus commun dans la production d'un jeu vidéo. Cette disponibilité de telles actions réduit l'intérêt de leur production automatique. Des séquences difficiles à animer, ou peu communes dans les productions passées (ex. *monter une échelle à une main*) rendent la

génération automatisée plus intéressante, mais aussi beaucoup plus difficiles pour un système d'apprentissage machine. Des techniques d'équilibrage de données ou de sélection intelligente des exemples d'entraînement deviennent ici très pertinentes, et établissent des directions à explorer dans des travaux futurs.

Finalement, la nature continue et *boîte-noire* des réseaux de neurones peuvent limiter leur application à des domaines pratiques ou du moins requérir des traitements additionnels. Par exemple, l'un des artefacts communs et facilement reconnu par des humains dans une animation générée est le patinage des pieds en contacts avec le sol (*foot-slides* ou *foot-skate*). Par contre, avec une animation générée par un réseau de neurones, aucun moyen connu permet d'éliminer ce problème sans post-traitement de cinématiques inverses. Un réseau performant saura générer des vélocités de pieds très basses, mais jamais nulles. La nature *boîte-noire* des réseaux de neurones joue aussi un rôle dans la difficulté à contrôler ce qui est généré, ce qui peut rendre leur adoption par des artistes ou des animateurs beaucoup plus difficile. Effectivement, malgré la possibilité que nous offrons aux utilisateurs dans le Chapitre 5 d'échantillonner plusieurs transitions pour un même contexte, fournir des contrôles plus fins quant au style d'animation serait un avantage considérable. Ceci est en soit une direction de recherche active et intéressante.

ABSTRACT

Realism in video games and in movies with Computer Generated Imagery (CGI) is often key for a positive, immersive experience. When animating computer-generated characters on screen, motion capture (MOCAP) is often considered the highest standard to obtain such realism. Indeed, MOCAP technologies excel at accurately reproducing human movements in a 3D environment as they are based on precise high-speed and high precision 3D marker tracking in space and time. Although some recent techniques rely less on marker-based capture (mostly in cinema) the data used in this thesis mostly comes from marker-based motion capture using a Vicon system[1]. Given such a widespread use motion capture over the years, companies have gathered large amounts of such high quality data, opening the door to data-driven approaches for analysis, classification, generative modeling and others.

Meanwhile, over the last decade, deep learning approaches have shown that they are able to surpass many other data-driven approaches for a myriad of tasks when the available data is sufficient. Naturally, interest for applying such approaches on 3D motion data has thus emerged in the animation domain.

In this thesis, we investigate the representational power of a special type of artificial deep neural networks, called Recurrent Neural Networks (RNNs), to solve different tasks on animation data. RNNs offer a natural choice of architecture for motion modeling, as they are explicitly designed for modeling temporal dependencies in sequences by having an internal memory of past context and recurrent connections in time. This allows them in theory to effectively handle sequences of arbitrary lengths and to compress relevant past information in their internal state.

More precisely, we first investigate in this work the representational power of RNNs for action classification. As motion capture became a common tool for video game and visual effects (VFX) studios, large datasets of motion data became available inside those companies, but not always in a structured manner. Capture sessions are usually tied to a particular production, and are often only described by filenames that are standardized on a per-production basis. This can be cumbersome to explore or simply insufficient to find precise actions for a novel production. The task of action classification aims at better describing the possible different motions inside each capture sequence in order to foster reuse of existing motion capture. To this end, we propose in Chapter 3 a multi-module, multi-objective architecture for semi-supervised learning of marker-based motion data. Optional modules, such as the

[1] https://www.vicon.com/

frame reconstruction decoder, the frame classifier and the sequence reconstruction decoder are activated with respect to the loss signals that are used. We show that semi-supervised learning, where large amounts of unlabeled sequences can be used with reconstruction losses to learn better representations, lead to improved classification performance on small labeled datasets. We further show that such a semi-supervised approach can lead to strong clustering capabilities, which can in turn accelerate labeling of the unlabeled sequences.

At the end of Chapter 3, we also investigate what will be the focus of the remainder of this thesis : motion transition generation. This task consists of generating the missing motion linking an input sequence to a target keyframe. In classical animation pipelines it was common to have keyframe artists draw principal keyframes describing the motion of characters, along with timing information for these movements. Interpolation artists, or *in-betweeners*, would then draw the missing motion frames with the right timing. Computer animation software systems offer similar interpolation capabilities as they allow one to fit different types of curves to animation keyframes. However, these interpolation strategies remain very limited and keyframes must often be as temporally dense as the final playback framerate for good results. In this work we investigate ways to perform transition generation from sparser keyframes, similarly to classical pipelines. We also aim at bringing motion capture quality in keyframe-based animation by using deep recurrent networks to learn motion patterns from real data. At the end of Chapter 3 we investigate an adversarial approach to generating marker positions between keyframes, and show that physically-based loss functions can stabilize training and improve results. In Chapter 4, we present a better, specialized architecture for transition generation of bone positions, along with training strategies that lead to better results. Our system can further use local terrain information to modulate the motion, and can perform high-quality temporal decompression for human locomotion. Finally, in Chapter 5, we address important limitations of our previous generators by improving robustness to time and target variations through latent modifiers that also enable stochasticity in the generated transitions. In that last study, we operate on bone rotations that allow for practical use in an animation software, with skinned characters. We present qualitative results through a Motion Builder plugin that can easily be used by animators.

In Chapter 6, we then further discuss common observations from these different works, report additional challenges related to applying such methods in a production environment and highlight relevant, interesting future research directions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

AdaIN	Adaptive Instance Normalization
CNN	Convolutional Neural Network
CGI	Computer Generated Imagery
CVPR	Computer Vision and Patter Recognition
DoF	Degree of Freedom
DRL	Deep Reinforcement Learning
FPS	Frames Per Second
FSM	Finite State Machine
GAN	Generative Adversarial Network
GMM	Gaussian Mixture Model
GP	Gaussian Process
GPU	Graphical Processing Unit
GRU	Gated Recurrent Unit
HMM	Hidden Markov Model
LERP	Linear Interpolation
LSGAN	Least Square Generative Adversarial Network
LSTM	Long Short Term Memory
MAP	Maximum A Posteriori
MDP	Markov Decision Process
ML	Machine Learning
MLP	Multi-Layer Perceptron
MOCAP	Motion Capture
NARA	Not A Real Acronym
NF	Normalizing Flow
NLL	Negative Log-Likelihood
NLP	Natural Language Processing
PCA	Principal Component Analysis
PD	Proportional-Derivative
PPO	Proximal Policy Optimization
RED	Recurrent Encoder-Decoder
ReLU	Recitfied Linear Unit
RGB	Red Green Blue
RGB+D	Red Green Blue + Depth

RNN	Recurrent Neural Networks
RTN	Recurrent Transition Networks
SLERP	Spherical Linear Interpolation
SVM	Support Vector Machine
VAE	Variational Auto-Encoder
VFX	Visual Effects
WGAN	Wasserstein Generative Adversarial Network

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

1.1 Animating Characters and Immersion

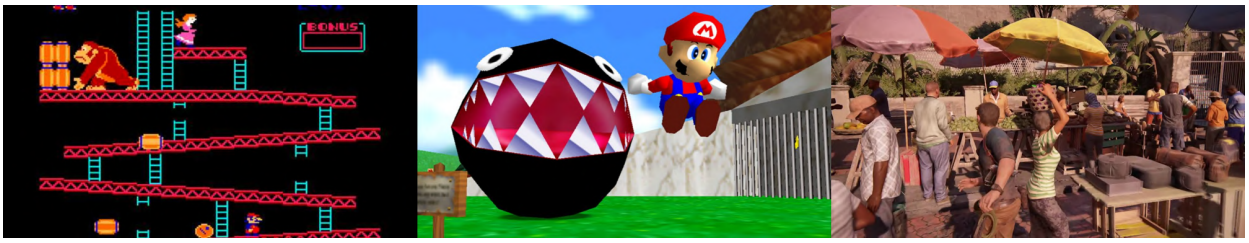


Figure 1.1 **The evolution of animation in games.** LEFT: Donkey Kong (1981) was one of the first games to use basic animations using very few keyframes. CENTER: Super Mario 64 (1996) was one of the first widespread home console games with 3D graphics and animation. RIGHT: Uncharted 4 (2016) was praised for its animation quality, mixing motion capture and physics-based animation.

Animation was first associated exclusively with moving drawings, primarily used for children entertainment. With landmark animations such as Steamboat Willie[1] in 1928, these 2D illustration-based animations kept gaining popularity during decades and laid the grounds for a a new branch of the movie industry. Nowadays, such hand-made animation methods are often considered as *traditional* or *classical*, given the rise of computer-assisted and computer-generated animations. More than 50 years after Steamboat Willie, in the arcade era of the 1980s, the first primitive versions of computer animations were seen in games such as Donkey Kong, Pac-Man or the Legend of Zelda. In these, only a few character poses were alternating as a way of showing movement or actions. In the 1990s, the advent of 3D graphics and home consoles, with the feature film Toy Story, and games such as Super Mario 64 really brought forward 3D animation and its impact on quality. In parallel, Motion Capture (MOCAP) technologies - that we cover in the next section - were starting to make their way in video games such as Virtua Fighter 2 and Soul Edge. With computational power always increasing, the availability of home consoles and personal computers, and improvements of computer generated graphics and visual effects, the overall quality of immersive experiences has kept improving since then.

Indeed, humans tend to seek experiences that can transport them mentally into new universes and many aspects of the stories we create play a role in that immersion. One of the key aspects for such a feat is the quality of character motion, which needs to keep up with the

[1] https://www.youtube.com/watch?v=BBgghnQF6E4

ever increasing quality of the accompanying visuals. Bad animation in a visual story can break the illusion of realism - even with imaginary creatures - and will thus hinder the feeling of immersion.

The work presented here was done in collaboration with Ubisoft, one of the world's leading AAA video game companies, which strives to deliver evermore immersive experiences to its users. There will thus be a focus through this thesis towards animation for video games, while most of the approaches and lessons presented here may be applied to other industries relying on MOCAP data. Given the high-standards for big studios in terms of animation quality and the capacity of MOCAP to re-create high-fidelity motions, these technologies have become a standard for creating high-quality, life-like animations that have been used in a myriad of well-known titles over the years.

As we will discuss below, the aim of this thesis is to address, through machine learning approaches, problems emerging from the very high costs of using MOCAP. Namely, we first tackle the challenges of reuse of motion data in the absence of clear motion descriptors through semi-supervised motion classification. We then aim at bridging the quality gap between keyframe animation and MOCAP through data-driven constrained motion generation with recurrent neural networks. In the rest of this introduction, we will first introduce MOCAP technologies and reasons behind their high costs. We will then briefly review the recent wave of deep learning approaches, their successes and their applicability to motion. Finally, we will present the main points of our work through an outline of the rest of the thesis.

1.2 Motion Capture

Motion Capture, or MOCAP technologies allow for capturing human motions realistically by tracking body movements or facial expressions of an actor with high precision, both temporally and spatially. In this work, we focus on body movement, and mostly use marker-based data, where actors wear special suits with reflective markers. In this setting, a rig of high-speed infrared cameras is set up around the capture volume, as seen in the left image of Figure 1.2.

The markers reflect the infrared light sent by the cameras and 3D tracking is done through the aggregation of the multiple viewpoints. The final results are a fully-defined 3D skeleton animation replicating closely the recorded motion (Figure 1.2, RIGHT). Such an animation is then usable and modifiable inside animation softwares. Multiple cleanup and processing steps, sometimes manual, must be performed in order to get the final motion on a usable character (Figure 1.3), while further retargeting may be necessary to apply motion on a

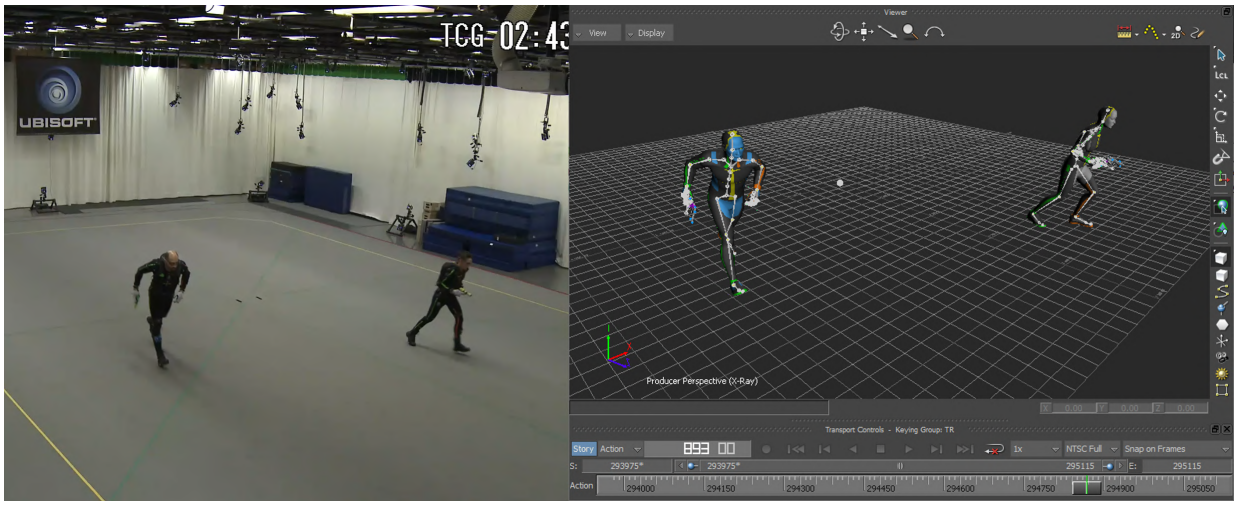


Figure 1.2 **Motion Capture**. LEFT: Capture session with actors wearing MOCAP suits and a Vicon camera rig. RIGHT: The corresponding processed sequence applied on skinned characters viewed inside the Motion Builder software. Courtesy of Ubisoft Montreal.

specific production character, or to reuse the motion on a novel character.

The equipment for MOCAP, the required actors, as well as all the processing required in order to get the final animation on the right character makes the technology very resource intensive, which is its main disadvantage. Indeed, the planning and processing of a motion sequence can take days or weeks to completion, depending on the complexity of the sequence, with the work of multiple people needed. However, given its high realism and quality, MOCAP is still often the default choice of big studios or companies that can afford it. Therefore, the amount of MOCAP data available inside such studios have grown over the years. Consequentially, this opens the door to approaches such as machine learning, and more specifically deep learning to be applied on such data.

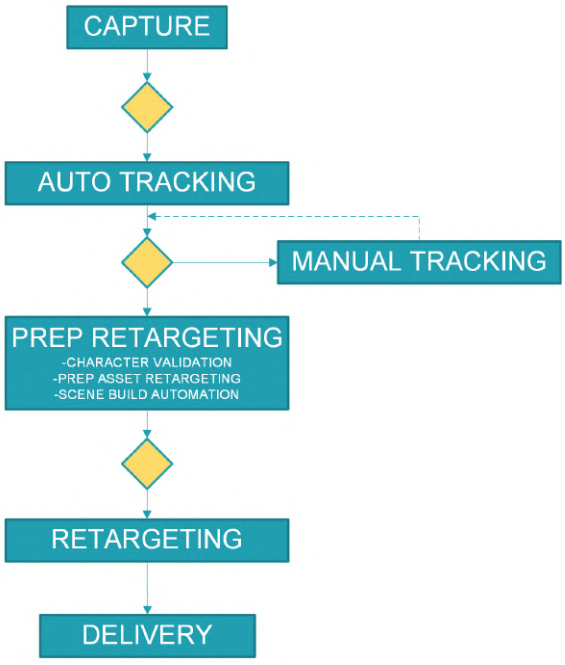


Figure 1.3 **Processing overview for MOCAP**. Courtesy of Ubisoft.

1.3 AI and the Rise of Deep Learning

Artificial Intelligence (AI) has captured human imagination and interest for decades, while its definition has always been somewhat volatile. Indeed, AI seems to have a definition intimately tied to the comparison between technology and magic made by Sir Arthur C. Clarke :

"*any sufficiently advanced technology is indistinguishable from magic*"

and one could argue that any sufficiently advanced *computer program* can be indistinguishable from *intelligence*. In 1959, Arthur Samuel brought impressions of computer *intelligence* with his pioneering work on Machine Learning (ML) to play checkers. Today, his program might not be considered intelligent by many, as the algorithms it used are now well-known and their computation requirements insignificant. Recent astonishing performances from language models such as GPT-3 [1] give rise today to similar debates about machine intelligence and even general intelligence, while some scholars still consider most of modern machine learning as mere curve fitting and not very intelligent[2].

In any case, an important milestone in the general field of AI was made with the advent of Machine Learning (ML), in which decision rules, or program parameters are not all coded by hand anymore, but rather adjusted, or *learned* through examples. These examples are often called the *training* data, and the term generalization refers to the setting where a learned algorithm is deployed, or evaluated on novel, *test* data. A core principle of ML is that test data cannot be present in the training data nor used to select the best model. ML methods are thus data-driven approaches that rely on the availability of sufficient data from which to learn those rules. Such a framework then enables addressing problems that were previously too complex for humans to code completely. A canonical example of this is for the classification of pictures of dogs and cats, for which writing a robust program is a very hard task. However, given an appropriate ML model with enough training data, this task becomes suddenly possible and is now considered easy for many recent techniques in many different settings. ML algorithms will often fall into one of three common categories, i.e. supervised learning, unsupervised learning, and Reinforcement Learning (RL), while boundaries between those might not always be clear and methods may fit in multiple categories.

While we won't delve into RL [2] in this thesis, it is a wide and active area of research that offers a well principled, generic way of approaching learning for an agent performing actions

[2]https://www.quantamagazine.org/to-build-truly-intelligent-machines-teach-them-cause-and-effect-20180515/

in an environment. It has seen a resurgence of popularity in research in the recent years, which was directly related to its possible coupling with deep learning approaches.

Supervised learning techniques rely on the availability of *labels*, which in general represent the correct outputs for any given training sample. The training of an ML model then consists of finding the set of parameters that will minimize some distance between its outputs and the true labels or their respective probability distributions. The presence of discrete labels is often framed as a *classification* problem, while continuous labels are used in the *regression* setting. Algorithms such as Nearest Neighbors, Gaussian Processes, Decision Trees, Naive Bayes are all supervised algorithms.

Unsupervised algorithms don't rely on labels, but only on data samples. Therefore, they are often used as a way to transform or augment the data in a useful manner for downstream tasks. Unsupervised methods include clustering algorithms, (e.g. K-means), component decomposition (e.g. PCA), dimensionality reduction algorithms and density estimation algorithms. Generative modeling is a type of unsupervised task that relies on density estimation of the data probability distribution in order to generate new samples.

An important family of ML algorithms so far left out and that can be used in the three categories described above is the family of artificial neural networks (or simply neural networks in this thesis). Neural nets are based on a crude approximation of biological neurons as building blocks [3]. Indeed, an artificial neuron is a computational unit that receives multiple signals, combines them through some simple transformation, and outputs a new signal. These transformations are usually a sum of the weighted input signals and a bias term, followed by a non-linear transformation of the result. The sum is weighted by the *synaptic weights* of the neuron that represent, along with the bias, the learned parameters of a neural net. Multiple neurons sharing the same inputs are called layers, and thus a minimal neural network will have one layer with one neuron. Fully connected, feed-forward neural networks are often called Perceptrons [4] or Multi-Layer Perceptron (MLP) when having more than one layer. The MLP is a canonical example of a deep neural network, as it uses hidden layers that do not produce final outputs, but rather intermediate latent representations that feed other layers. Such networks lead to high-level representation learning in a hierarchical manner throughout the layers, which was shown to be a key aspect of approximating complex functions on high-dimensional data [5]. The term Deep Learning (DL) thus represents ML approaches based on neural networks with multiple layers. Such deep networks however, with ever growing number of neurons and parameters presented the challenge of defining a good update rule for the synaptic weights. Rumelhart *et al.* [6] showed that the reverse-mode, automatic differentiation [7–9] popularized as *backpropagation* was an effective algorithm for such an

update. Nowadays, most neural networks are trained on variants of this algorithm, such as Stochastic Gradient Descent (SGD), RMSProp [10], Adam [11] and others. Deep learning libraries, such as Tensorflow [12] and pyTorch [13] automate automatic differentiation and related optimizers, making it almost trivial for users to apply them.

In the last decade, an explosion of deep learning successes have contributed to foster even more research, in a snowballing effect that keeps going to this day. These initial successes however were not necessarily justified by novel theory or algorithms only but mostly by other important factors. First, the release of free, public large-scale datasets such as the MNIST[3] database and more recently Imagenet [14] fostered ML research and showcased the superiority of deep models, given enough data. Second, the coupling of the training of deep models with available Graphical Processing Units (GPU) that offered high parallelism capacity was a practical achievement that opened the door to wider and deeper architectures to be trained efficiently on common hardware. A now symbolic combination of those factors was achieved in 2012 by Krizhevsky *et al.* [15] who were the first to use a deep Convolutional Neural Network (CNN) [16, 17] for the Imagenet classification task, trained on two GPUs, and the first to reach an error rate below 25%. This was a landmark achievement from which followed multiple DL approaches that kept reducing the error rate in the following years, and played a significant role in the deep learning renaissance of the last decade. It seems indeed, that even though the novelty of the algorithms and theory are sometimes modest, we might be witnessing since then the impacts of winning the *hardware* and *software lotteries* [18].

As suggested above, computer vision and CNNs played a major role in the development of deep learning techniques and interest. This has lead to many important advances in domains like medical imaging [19], self-driving cars [20], movie production [21, 22], and many others. Other important branches of deep learning played significant roles in other domains. Namely, Recurrent Neural Networks [23, 24], and their improved counter-part, Long-Short-Term-Memory (LSTM) networks [25] have been used in a wide range of applications, such as handwritten character recognition [26] and prediction [27], statistical machine translation [28, 29], speech recognition [30, 31], text generation [27], motion prediction [32, 33] and others. More recently, large-scale Natural Language Processing (NLP) systems tend to favor more and more attention-based systems [34] that can remove the need for recurrent connections in the models [35–37]. The recent GPT-3 architecture [1] uses such an approach, scaled to previously unseen size and data, to produce a single system capable of tackling multiple tasks such as question-answering or translation all framed as text completion. Graph-based DL [38] can also help solve different problems, such as protein [39] or document classification

[3] http://yann.lecun.com/exdb/mnist/

[40], drug side-effects prediction [41] or user recommendations [42]. Another important axis of development in DL is the one of generative modeling, where models aim at learning the data distribution in order to produce new samples. Within that realm came in 2014 the formulation of Generative Adversarial Networks (GANs) [43], which provide a way to efficiently transform a simple distribution into a complex data distribution and to perform direct sampling of new data points. GANs have become widely popular because of their high-quality realistic results, despite the fact that they do not allow for inference and thus do not permit likelihood estimation. Their training scheme, based on learned loss from a discriminator network, optimizes for realism of the samples, at the cost of often ignoring modes of the data distribution (mode collapse). GANs have also been used in multiple areas to produce better results, such as super-resolution [44], image completion [45], sound generation [46], image translation [47], text-to-image translation [48], and motion completion [49]. Note that all these methods differ from the original GAN formulation in that they use *conditional* GANs, where the generator does not only use a noise sample as input, but other conditioning signals. Indeed, unconditional GANs can be very impressive [50] but may have limited applications without conditioning capabilities.

1.4 Motivation

As described above, the goal of this work is to reduce the reliance on novel recording sessions of MOCAP sequences, as they are very costly and time-consuming to produce and to post-process. This thesis focuses on two different solutions for such problems, via 3 different works published as a journal article (Chapter 3), a conference technical brief (Chapter 4), and a conference proceeding (Chapter 5). The different approaches proposed share a core component, which is the use of LSTM layers in the architecture as a way to model temporal dependencies in motion.

The first problem we tackle is the one of motion recognition in MOCAP sequences in order to foster reuse of existing motion data. Indeed, many production studios have gathered over the years large amounts of high quality motion data that are unfortunately often hard to re-purpose simply because of a lack of structure and well-defined descriptors for such assets. In large game companies, different productions may have different naming conventions for motion files, and this is often the only descriptor available. We thus wish to improve existing methods for 3D motion classification with the goal of automatically classifying motion clips for easier retrieval. We investigate whether a specific modular recurrent architecture, within a semi-supervised learning setting, can improve motion classification performance on small labeled datasets by leveraging larger amounts of unlabeled data.

The second problem we investigate is the one of automatic motion in-filling or keyframe interpolation, where missing animation between clips or keyframes is generated to save animators' time or prevent new MOCAP sessions. Indeed, even though keyframe animation requires less resources than MOCAP, it is also time-consuming for animators as current tools impose to provide good keyframes that are temporally dense. This is due to the limited interpolation strategies currently available in animation softwares. The resulting quality is also often lower than with motion capture. We wish to address those problems by allowing for sparser keyframes with a better in-betweening technology, while bringing MOCAP quality to keyframe animation. We thus use a recurrent motion prediction architecture that we constrain in various ways to improve the generated transitions so that they are plausible and respect spatio-temporal constraints, while trying to reach MOCAP quality.

1.5 Outline

The rest of the thesis is separated in the following way:

Chapter 2 : Literature Review

We review in this chapter the relevant literature, starting with some background on RNNs, CNNs, auto-regressive models, and GANs. These are the basic DL building blocks that we will use in most of the following articles. We then review several aspects of the area of motion modeling, starting with preliminaries such as the question of data representation, especially for motion generation, error metrics, and motion datasets. We then delve deeper into the subject by reviewing works on motion classification and motion generation, our two main tasks. Finally, we touch on the subject of conditioning in deep neural networks.

Chapter 3: Recurrent Semi-Supervised Classification and Constrained Adversarial Generation with Motion Capture Data

This chapter is an article published in the Image and Vision Computing journal in which we focus on semi-supervised action recognition and perform preliminary investigations on the task of transition generation.

We tackle the task of action recognition on a popular motion capture dataset [51] and show how the commonly used benchmark can lead to misleading results because of its unrealistic data partitioning (Section 3.3). We then show how State-Of-The-Art (SOTA) methods get significantly lower accuracies when using a plausible actor-based partitioning, as we do.

We then present in Section 3.4 a modular, recurrent architecture where different decoder modules can be used with respect to optional objective functions. Those objectives correspond to the tasks of classification and reconstruction, both on a per-frame or per-sequence bases. Since the main goal is to perform sequence classification, this is the only mandatory objective. We then perform an empirical study on which decoders are beneficial to the final task, in supervised and semi-supervised settings. Our study suggests that adding frame reconstruction to a sequence classifier helps the most when working with labeled data only (Section 3.7.1), while both frame and sequence reconstruction losses lead to the best performance in a semi-supervised context, i.e. when a larger amount of unlabeled sequence is also used for representation learning (Section 3.7.2). Additional experiments on a much larger, labeled dataset [52] suggest that reconstruction objectives do not help as much to increase the performance of our architecture with the same capacity (Section 3.7.4). As demonstrated in Section 3.7.3, using such an approach also allows for accurate clustering of large unlabeled datasets towards semantically meaningful actions, accelerating the labeling of new data.

We then perform a preliminary exploration of constrained motion prediction (Section 3.5) to reach target keyframes based on a motion context, a task on which we focus in the rest of the thesis. In this setup, we replace our sequence reconstruction module with a motion prediction module simply by modifying the loss function and training setup. We also add, as conditioning information, a target pose to reach to produce transitions. To prevent collapse to an average pose, or an average interpolation, we make use of the adversarial learning framework by adding a motion discriminator network and a discriminator loss to train our generator. As training LSTMs and GANs are both hard in their own rights, combining the two approaches is particular challenge. We show in this study how the use of additional physically-based loss functions that act as soft constraints help stabilize the training of our LSTM-GAN and improves the generated transitions.

Chapter 4: Recurrent Transition Networks for Character Locomotion

In this SIGGRAPH Asia 2018 Technical Brief, we investigate in more details the challenges of transition generation with recurrent neural networks. In Section 4.4, we simplify the architecture used in Chapter 3 by removing the latent vector used between the sequence encoder and decoder, yielding an architecture more similar to highly performing motion predictor networks used for motion. Here, the notion of encoding and decoding is solely based on whether the network is running in auto-regressive mode, i.e. using its previous outputs in the absence of ground-truth context. We also use MLPs as frame encoder and decoders [33] and use a residual formulation of the RNN where only offsets from the previous

frame are predicted [32]. As opposed to motion prediction architectures, we augment the inputs of the LSTM at each time frame with keyframe pose and velocity information to smoothly reach the desired state.

We also propose ways to accelerate training and improve final results. First, we augment the architecture with a recurrent hidden state initializer (Section 4.4.6) that infers a good hidden state from the first frame of past context with which the LSTM is initialized. This allows the recurrent layer to start in a good latent region with respect to the context. We also propose to use a fixed-probability teacher forcing strategy during training (Section 4.5) that outperforms free-running, teacher forcing and scheduled teacher forcing. We test these contributions in an ablation study (Section 4.6.3).

Finally we also explore terrain conditioning (Section 4.6.2) for generating motion on uneven terrain, as feet artifacts are often the most noticeable in 3D animation, and ground contacts play a major role in the plausibility of movement. In this case, we augment the architecture with a convolutional neural network that processes local terrain information presented as a heightmap centered on the character. The output of the CNN is used as extra conditioning information. We show that the impact of such conditioning is limited on short transitions, but becomes significant on longer ones (e.g. 2 seconds), and will reduce the eventual Inverse-Kinematics (IK) post-process impact on the output.

Compared to our previous attempt at transition generation from Chapter 3, this work leads to visually convincing results that are often hard to distinguish from real transitions and allow the model to be used for temporal super-resolution of motion (Section 4.6.4). However, multiple implementation shortcuts were taken to reach those results, limiting the applicability of the system in real-world scenarios. Such limitations include the fact that the network uses keyframe velocities as conditioning information, that all outputs contain positional information only, and that the transition length is fixed for a particular trained network. In the next article, we address those limitations and more.

Chapter 5: Robust Motion In-betweening

In this SIGGRAPH 2020 conference proceeding and Transaction On Graphics (TOG) journal paper, based on our work on recurrent transition networks, we address the limitations of Chapter 4 in order to produce a system that can be used in practical scenarios by animators. This is again part of our continued effort to bridge the gap of quality between MOCAP and keyframe animation. We achieve this first through the design of a strong motion generation baseline based on SOTA advances in the domain of unconstrained prediction [32, 33, 53–55]. This motion predictor, as opposed to our previous work, outputs fully defined animation data,

with global displacement and rotation information, to be applied on skinned characters.

We then add conditioning information about the target state to build a transition network (Section 5.3.2), and address the inherent limitations of this naive approach for robust motion transitions. Namely, we address the problems of brittleness to the transition lengths and to target noise or modifications, as well as the inherent determinism of the system. We apply these modifications with two different additive latent modifiers applied on the LSTM inputs.

The first latent modifier is a *time-to-arrival* embedding (Section 5.3.3) similar to the positional encoding used in Transformer networks [35], which in our case indicates the number of timesteps left before reaching the target. This embedding is added to all LSTM inputs and provides a dense, continuous code that shifts the input representations uniquely for each time to arrival, effectively modulating the model's behavior.

Our second modifier is called *scheduled target noise* (Section 5.3.4), and is a random noise vector sampled from a zero-centered spherical Gaussian once per transition. This embedding is added only to target representations and is scaled down to zero during the generation. It is thus a random modifier that diminishes in time, forcing the network to rely more heavily on the past context at the beginning of the transition while being robust to distorted targets. An additional scaling factor for the whole noise sequence allows users to determine the level of stochasticity desired at runtime when using the system. Therefore, the model can provide multiple transitions for a fixed context, and the trade-off between quality and variety is controllable with this stochasticity parameter.

We quantitatively evaluate our contributions on a commonly used dataset (Section 5.4.1, Section 5.4.2) that is however ill-suited for transitions that go beyond a very short length. We thus also evaluate our approach on a higher quality, custom motion dataset (Section 5.4.3) that we released with the article, along with code to replicate the data processing and baseline results to which we compare.

We qualitatively evaluate our approach via a custom plugin for Motion Builder (Section 5.4.4), an animation software commonly used in production. Our plugin allows animators to generate high-quality transitions for their character in a few clicks and to control the level of stochasticity in the resulting animation.

Chapter 6 General Discussion

We discuss in this chapter shared observations through the different works done, the challenges and the surprises encountered, and elaborate on interesting future directions. Namely we touch on subjects such as the challenges associated with action taxonomies in production

(Section 6.1), practical limitations of RNNs (Section 6.2), the challenges of rotations over positions (6.3), the critical data problem in production (Section 6.4) and the problem of style and control with neural networks (Section 6.5). We also analyze the impact of our work so far in Sections 6.6 and 6.7.

CHAPTER 2 BACKGROUND AND LITERATURE REVIEW

2.1 Deep Learning Background

2.1.1 Recurrent Neural Networks

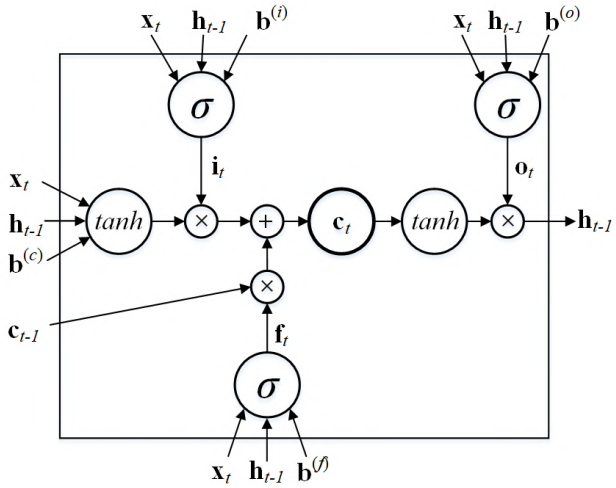
The main architectures for motion generation presented in this thesis are built with Recurrent Neural Networks (RNNs) at their core, which were derived from the works of Hopfield [23] and Rumelhart *et al.* [6]. RNNs are an extension of artificial neural networks that have at least one layer with *recurrent* connections to itself, allowing for propagation - and backpropagation - of information between timesteps of a series. For a simple linear feed-forward network with a single hidden layer, the forward pass in the hidden layer can be written in the form

$$\mathbf{h}_t = \mathrm{W}\mathbf{x}_t + \mathbf{b} \tag{2.1}$$

whereas the RNN equivalent can be written as

$$\mathbf{h}_t = \mathrm{W}\mathbf{x}_t + \mathrm{U}\mathbf{h}_{t-1} + \mathbf{b} \tag{2.2}$$

where W and U are the feed-forward and recurrent weight matrices respectively and $\mathbf{b}$ is a bias vector. The vector $\mathbf{x}_t$ is the current input at time t and $\mathbf{h}$ is the output vector of the hidden layer. The added recurrent connections U allow $\mathbf{h}$ to propagate information through time, making it act as a kind of memory, and leaves the architecture compatible with back propagation algorithms. Back-propagation through U is often referred to as *back-propagation through time*. RNNs are powerful sequence learning networks in theory, that are however limited in their capacity to retain information and back-propagate error signals correctly over long periods of time. This can be in part explained by their fully connected recurrent weights that can amplify or reduce the norm of the back-propagated signal at each timestep, leading to exploding or vanishing gradients. To counter exploding gradients, simple strategies, such as gradient clipping can be used, while the vanishing gradient problem was tackled with a novel recurrent layer architecture [25] called the Long-Short-Term-Memory cell (LSTM). LSTM layers allow the network to control updates (and back-propagation) to its internal state through smooth, differentiable gates managed by additional weights. An overview of the LSTM cell is shown in Figure 2.1, where the σ symbol represents the sigmoid non-linearity, while $\bigoplus$ and $\bigotimes$ represent element-wise addition and multiplication respectively.



$$\mathbf{i}_t = \text{sigmoid}(\mathrm{W}^{(i)}\mathbf{x}_t + \mathrm{U}^{(i)}\mathbf{h}_{t-1} + \mathbf{b}^{(i)}) \quad (2.3)$$

$$\mathbf{o}_t = \text{sigmoid}(\mathrm{W}^{(o)}\mathbf{x}_t + \mathrm{U}^{(o)}\mathbf{h}_{t-1} + \mathbf{b}^{(o)}) \quad (2.4)$$

$$\mathbf{f}_t = \text{sigmoid}(\mathrm{W}^{(f)}\mathbf{x}_t + \mathrm{U}^{(f)}\mathbf{h}_{t-1} + \mathbf{b}^{(f)}) \quad (2.5)$$

$$\hat{\mathbf{c}}_t = \tanh(\mathrm{W}^{(c)}\mathbf{x}_t + \mathrm{U}^{(c)}\mathbf{h}_{t-1} + \mathbf{b}^{(c)}) \quad (2.6)$$

$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \hat{\mathbf{c}}_t \quad (2.7)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \quad (2.8)$$

Figure 2.1 LSTM cell.

The corresponding equations 2.3 to 2.8 show the calculations of the forward pass at a given timestep t. LSTM layers thus provide a hidden state $\mathbf{h}_t$ just as standard RNNs, but also keep an internal *memory state* $\mathbf{c}_t$ which is of the same dimension as the hidden state. The different sigmoid-based gates allow to control the impact of new information on $\mathbf{c}_t$ and $\mathbf{h}_t$, allowing increased stability in signal propagation. For great overviews of LSTMs and applications, please refer to [24] and [27]. More recently, Cho *et al.* [29] have proposed another gated recurrent architecture, known as the Gated Recurrent Unit (GRU) that solves the same problems as LSTMs and uses less parameters. There is no consensus in the literature about which of the two architectures performs best, but in our empirical tests applied to our tasks, LSTMs performed slightly better, motivating our use of this higher-capacity structure.

RNNs are often considered as sequence encoders, since their last hidden state $\mathbf{h}_T$ contains a representation of the whole sequence. As such, RNNs can encode variable length sequences into fixed-sized *encodings*, which can in turn be used for different tasks, such as classification, prediction or reconstruction. These Recurrent Encoder-Decoders (RED) architectures have been used successfully in Natural Language Processing (NLP) for machine translation [28,29] or sentiment analysis [56], and in the video domain for unsupervised video representation learning [57], as well as many other areas.

2.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [16,17] are neural networks that use a special type of architecture based on the convolution operation.[1] Convolutions are local computations based on filters that are repeatedly applied on restrained parts of the input while moving across its dimension(s). In other DL terms, those local filters are equivalent to linear perceptrons with limited receptive fields duplicated across the input. The weights of the those filters are the learned parameters of a convolutional layer, while the optional padding on the input, the stride of the filters, their number and their receptive field's size are all user-defined hyper-parameters. For a complete introduction to CNNs, we recommend the CS231 Stanford course page[2] on the matter. CNNs are roughly inspired by the processes happening in the visual cortex of humans [16] by doing local computations duplicated across spatial dimensions. This gives CNNs the property of shift invariance, as the same filters are applied across spatial locations. It also leads to another practical advantage; convolutional layers offer a high level of parallelism as filters are applied on different input regions independently. It is thus easy to distribute the computations on different input regions across multiple cores of a GPU. This architecture, as with RNNs represents a way to inject some sort of domain knowledge in a neural network and has been an important pillar in many recent vision-based deep learning successes.

In in most image-based CNNs, 2D convolutions are used, where the filters are repeated across two input dimensions. In Chapter 5, our critic networks use 1D convolutions, where the spatial receptive field includes all skeleton joints, and filters are repeated only across the temporal dimension.

2.1.3 Generative Models

A big part of this thesis touches on some kind of generative modeling, since our models have the task of generating new data, albeit conditioned on other data. Generative modeling is often described as density estimation, where the goal is to learn the data distribution p_{data}, which then enables the creation of new data points or the completion and modification of existing samples. It is often associated with some *understanding* of the data generated processes, as it requires for a system to discover important latent factors of a data distribution. A great overview of generative models from the point of view of maximum likelihood is given by Ian Goodfelllow in [58], where different families are derived. In this research, we combine two important families of generative models : auto-regressive models, and Generative Adversarial

[1] In practice, convolutions in CNNs are implemented as *cross-correlations*.

[2] https://cs231n.github.io/convolutional-networks/

Networks (GANs) that we discuss below, while recent development in machine learning gave rise to other important families of models, such as Variational Auto-Encoders (VAE) [59], normalizing flows [60] and more recently denoising diffusion probabilistic models [61].

Auto-regressive models

We refer here to auto-regressive models as models predicting sequential components based on all previous or a subset of previous components. Note that the sequence does not need to be a temporal sequence, and can represent any tensor with an arbitrarily defined component order that assumes some dependence between those components. A simple linear auto-regressive model with one step look-back can be expressed as

$$x_n = \omega x_{n-1} + a \tag{2.9}$$

where x_n is an element of the sequence, and ω and a are parameters of the model. As learning a sufficiently expressive distribution of high-dimensional data samples can be intractable, many probabilistic auto-regressive models aim at simplifying this task by decomposing a data sample vector $\mathbf{x}$ into its components x_i or a sample sequence X into its timesteps vectors $\mathbf{x}_t$. Learning conditional probabilities for those elements is often much simpler and the joint probability of the whole sample can be retrieved through the chain rule of probability.

$$\begin{aligned} P(X) &= P(\mathbf{x}_0)P(\mathbf{x}_1|\mathbf{x}_0)P(\mathbf{x}_2|\mathbf{x}_1,\mathbf{x}_0)... \\ &= P(\mathbf{x}_0)\prod_{t=1}^{n} P(\mathbf{x}_t|\mathbf{x}_0,...,\mathbf{x}_{t-1}) \end{aligned} \tag{2.10}$$

RNNs are good examples of auto-regressive models, as they can naturally maximise the likelihood of a slice $\mathbf{x}_t$ of a sequence X given all the previous timesteps. A prior for $P(\mathbf{x}_0)$ can be given or learned. PixelRNN [62] is a good example of such an approach for generating images pixel by pixel, as well as many RNN-based motion prediction methods that we will discuss below. When training RNNs as auto-regressive models, some care must be taken to make them robust to their own runtime predictions. Indeed, in tasks where the RNN will have to predict long sequences, training it to do prediction only from past ground-truth information (*teacher forcing*) may lead to unstable results when the output distribution deviates from the training distribution. On the other hand, training a model to produce long-sequences based only on its own prediction (*free-running*) may severely hinder the training and lead to averaged outputs easily. Scheduled teacher forcing [63] and professor forcing [64] are known approaches to RNN training to address these issues. On the animation side particularly, Li

et al. [65] proposed Auto-Conditioned LSTMs for long-term generation, while we opt for a fixed-probability teacher forcing approach in Chapter 4 and a curriculum learning strategy in Chapter 5 to address the same problems.

In many settings, a fixed local window of context is used as an estimation of $P(\mathbf{x}_t|\mathbf{x}_0, ..., \mathbf{x}_{t-1})$, with the assumption that the current element is independent of distant ones. PixelCNN [66] uses such assumptions. Other architectures such as the convolution-based WaveNet [67] for audio or the feed-forward Phase-Functioned Neural Network [68] for animation are auto-regressive networks applied on temporal data that also use a fixed number of conditioning elements to generate the next elements of a sequence. The less information - or past context - one uses, the more ambiguity there will be about the possible future motion. Auto-regressive models with one-step look-back will often use additional signals, such as phase information to disambiguate the current motion and improve predictions.

Generative Adversarial Networks

Generative Adversarial Networks (GANs) [43] are a family of generative models where a generator network uses a loss function that is not user-defined, but learned by another network called a discriminator (or sometimes a critic). The generator (G), parameterized by ϕ learns a mapping from a simple distribution $p_{\mathbf{z}}$ (usually a multivariate normal ditribution) to an output distribution p_G, while the discriminator (D), parameterized by θ learns to assess the realism of the generated samples. More precisely, G is trained to have its output distribution p_G match the true data distribution p_{data}, while D is trained to differentiate real data samples from generated ones. In the original GAN formulation, D is trained as a binary classifier with a single sigmoid output. The generator learns from the error signal of the differentiable discriminator as it can back-propagate through D's network. This results in a minimax game $\min_G \max_D V(D, G)$ between G and D where the value function $V(D, G)$ is as follows:

$$V(G, D) = \mathbb{E}_{\mathbf{x}\sim p_{data}}\Big[\log D_\theta(\mathbf{x})\Big] + \mathbb{E}_{\mathbf{z}\sim p_z}\Big[\log\Big(1 - D_\theta(G_\phi(\mathbf{z})\Big)\Big] \tag{2.11}$$

In practice, this minimax game can be translated into the following loss signals for G and D:

$$L\phi = \hat{\mathbb{E}}[-\log(D_\theta(G_\phi(\mathbf{z})))] \tag{2.12}$$

$$L\theta = \hat{\mathbb{E}}[-\log D_\theta(\mathbf{x}) - \log(1 - D_\theta(G_\phi(\mathbf{z})))] \tag{2.13}$$

where $\hat{\mathbb{E}}$ represents the empirical expectation over a training minibatch. This has been shown to minimize the Jensen-Shannon (JS) divergence between p_G and p_{data}, for which the global minimum implies that G provides indistinguishable samples from the real ones. GANs

became popular in the vision community for their direct and scalable sampling capabilities as well as their visually appealing results. Compared to auto-encoders, VAEs and other generative models, GANs have been shown to provide crisp samples, reducing averaging effects often produced by likelihood or reconstruction based methods. GANs however, present a few disadvantages. They can't model the probabilities of given samples as they do not model likelihood or an estimate of it. They are also known to be hard to train, as (1) they do not provide a meaningful measure of convergence, given that the loss function isn't fixed and (2) it is hard to keep G and D balanced so that D provides useful gradients to G throughout training. Many approaches have been proposed to tackle these issues, with Wasserstein GANs (WGAN) [69, 70] being one of the most popular ones. Throughout our research however, comparisons between WGAN and another GAN variant called Least Square GAN (LSGAN) [71], which is simpler to implement, always favored the latter in terms of stability and qualitative results.

Least Square GANs use different optimization objectives to train G and D:

$$L_\theta = \frac{1}{2}\hat{\mathbb{E}}[(D_\theta(\mathbf{x}) - b)^2] + \frac{1}{2}\hat{\mathbb{E}}[(D_\theta(G_\phi(\mathbf{z})) - a)^2] \tag{2.14}$$

$$L_\phi = \frac{1}{2}\hat{\mathbb{E}}[(D_\theta(G_\phi(\mathbf{z})) - c)^2] \tag{2.15}$$

in which a, b, c are fixed scalars. With the constraints $b-c = 1$ and $b-a = 1$ the authors show that this optimization minimizes the Pearson χ^2 divergence between p_G and p_{data} instead of the JS divergence. However, authors also suggest using $a = 0$, $b = 1$ and $c = 1$, keeping the labels 0 and 1 for fake and real labels respectively, and pushing G to provide realistic samples. This is what we used in our LSGAN implementation in Chapter 5. LSGANs address vanishing gradient issues that often arise in regular GANs, making the learning process more efficient.

2.2 Motion Modeling

We refer to motion modeling as any technique that allows to automatically extract a motion representation useful for classification (Section 2.2.4) or other data-generating downstream tasks such as reconstruction, prediction, or generation of novel movements (Section 2.2.5). We restrain in this work the data domain of *motion* to sequential 3D skeletal data, even though a vast amount of work has been done in motion analysis in RGB or RGB+D videos. Note however that in these latter cases, the first step of the analysis is often to estimate 3D skeletal data before applying subsequent processing.

2.2.1 Data Representation

In animation pipelines such as for video games or in animation authoring softwares, visual animation of a character mesh is usually driven by an animated skeleton, as seen in Figure 2.2, through skinning weights that specify the joints' influences on mesh vertices. This allows animators to manipulate a skeleton object, which is simpler than a mesh, and for which multiple character meshes may be skinned. Bone transformations can be *joint-local*

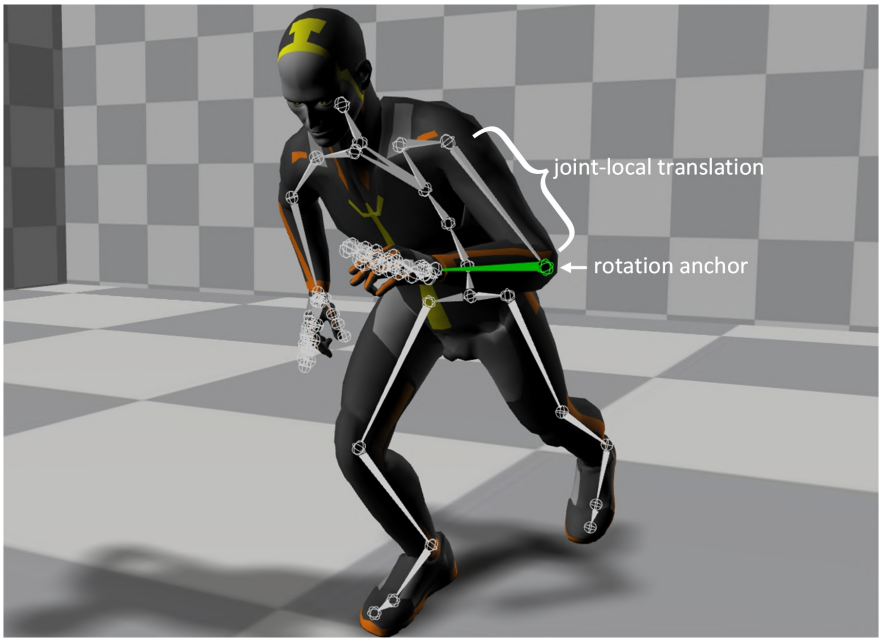


Figure 2.2 **Skeleton-driven kinematic animation.** The underlying skeleton drives the mesh animation. Each joint is defined by a translation and a rotation. Joint-local translations are offsets from parent bones in the hierarchy. The endpoint of that offset is the rotation point.

when expressed as relative to the parent bone's frame of reference, *character* or *root-local* when expressed relative to the root bone, or *global* when relative to the world's frame of reference. Scaling of bones is expressed as joint-local translations and are usually constant in any given motion sequence. Note however that animation data, as expressed through skeletal information is the final representation obtained in the MOCAP pipeline, whereas the initial format in this case usually contains 3D marker positions. We work with a subset of markers under that representation in Chapter 3.

A common and practical way to represent 3D bone data is through a 4x4 transformation matrix T containing rotation and translation information. Such a transformation matrix can

thus be separated into a 3x3 rotation matrix R, and a 3D translation vector, or positional vector $\mathbf{p}$:

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}; R = \begin{bmatrix} a & b & c \\ c & d & d \\ e & f & g \end{bmatrix}; T = \begin{bmatrix} a & b & c & x \\ c & d & d & y \\ e & f & g & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The matrix R must be part of the *special orthogonal group* SO(3) in order to represent a valid rotation, meaning it is an orthogonal 3x3 matrix that has a determinant of 1. Another common representation for the rotation information in computer graphics are quaternions, which define a number system that extends complex numbers:

$$\mathbf{q} = w + xi + yj + zk\,, \text{ with } i^2 = j^2 = k^2 = ijk = -1$$

Quaternions can be internally represented as normalized 4D vectors $\mathbf{q} = [w\, x\, y\, z]^T$, useful for computations. It is also common for animation software to expose XYZ Euler angles as a user-facing representation. Indeed Euler angles are easier to grasp and visualize, while transformation matrices and quaternions have better mathematical properties for computations.

A long-lasting question in motion modeling and deep learning thus regards which angular representation one should use. Any task, such as classification, that doesn't require producing skinned animation can be performed with positional data only. For example, in Chapter 3, we improve action classification with auxiliary reconstruction tasks that do not require a meshed character. We use only positional data from either MOCAP markers or Kinect-inferred joints. Positional data is often easier for neural network to learn and to generate. In Chapter 4, the model generates high-quality positional data only without any post-processing and with minimal pre-processing. To reach similar levels of quality in Chapter 5, considerable additions such as new training strategies, computations, pre-processes and post-processes had to be used. This was necessary since production-ready motion generation for skinned characters requires angular information. Different works have used different angular representations such as rotation matrices (e.g. [72]), quaternions (e.g. [53]), exponential-maps (e.g. [33]), and others (e.g. [73, 74]), while Euler angles are rarely used because of their limitations such as singularities and gimbal lock. If one wants to avoid singularity issues with Euler angles, exponential maps and axis-angles arising near the usual $[-\pi, \pi]$ limits around an axis, one must use an unbounded domain, hindering the learning process and/or generalization. Note however that when modeling angular velocities at high enough frequencies, these types of singularities can be avoided if the angular data is processed to be smooth (no flipping around singularity points). The outputs of the model will then have a very limited domain centered

at 0, a desirable setting for deep learning. Quaternions are good candidates for modeling angular data as they do not suffer from singularities, are compact, easily composable, and can be easily and smoothly interpolated through spherical linear interpolation (SLERP). However, similarly to rotation matrices that need to be orthogonalized, quaternions must be normalized to represent valid rotations. This can be done as a post-process on model outputs and smoothly encouraged by a regularization loss. As reported by Pavllo *et al.* [53], quaternions also offer *antipodal* representations, where $\mathbf{q}$ and $-\mathbf{q}$ represent the same orientation, potentially causing discontinuous sequences. This can be easily fixed with a smoothing preprocessing phase, and antipodal representations can be used as data augmentation. We use such pre-processed quaternions in Chapter 5.

2.2.2 Error Metrics and Loss Functions for Motion

In the classification domain, loss functions and error metrics are usually straightforward, with Negative Log-Likelihood (NLL) loss and measures such as classification accuracy, precision, recall and F1 score being commonly used. We follow this path in Chapter 3 to train and evaluate our classification models. However, when generating motion, there is no consensus yet on how to best evaluate animation in a way that correlates well with human appreciation, and on which losses better capture those criteria. Early work on spacetime constraints [75] proposed an algorithmic approach to keyframe animation of an articulated, physically-simulated character that naturally imposes - through physics - some of the 12 principles of animation [76], such as *anticipation*, *squash-and-stretch*, *follow-through* and *timing*. These principles have been well-established guidelines for improved quality in classical hand-drawn animation for decades. Hodgins *et al.* [77] included in their system passive secondary motions to add the effect of *follow-through*. Although they didn't use human reference motions in their algorithm, they used human recordings to evaluate their results both qualitatively and quantitatively. This is an important step towards closing the gap between generated and recorded movements. Indeed, for complex 3D humanoid characters, these high-level animation principles, approximated through physical or kinematic rules are not complete enough to produce satisfactory, realistic movements. This is why using recorded reference motions inside the generating system or learning algorithm quickly became a widely adopted approach [78–81] that outpaced handcrafting of new rules for movement.

For machine learning systems, a straightforward way to capture human motion dynamics is to train with a loss signal directly tied to ground-truth motion. NLL and any reconstruction losses directly optimize for realism of the generated animations as they push models towards mimicking humans. For example, given constant bone offsets, any angular reconstruction

loss that is perfectly minimized to zero will lead to perfect reconstruction of local motion on training data. However, when generating novel constrained motion or when the task is simply under-specified, it is unclear which loss signal is best to minimize so that errors from the model have the least impact on human judgement. In other words, how can the missing capacity of a neural network or under-determined inputs be framed in a way that doesn't hinder qualitative results? Some hints can be extracted from the generative modeling literature where GANs often offer samples that are preferred to those given by a VAE decoder with the same capacity. When the goal is to generate quality samples, it can be preferred to produce crisp, realistic samples that ignore some modes of the data distribution over blurry samples that average different modes of the distribution. Of course, the goal of improving the architecture, capacity and learning mechanisms of models is to eliminate that trade-off, but it is often still present. Simplifying and disambiguating the task can also be a shortcut to augment the relative capacity of a network. A good example is found in the work of Martinez *et al.* [32] where velocities are predicted instead of poses in order to significantly reduce the range of expected outputs of the model as well as having its mean prediction correspond to a good baseline (zero-velocity). Another example is the motion quality obtained from small feed-forward, single-frame, auto-regressive networks for online control (e.g. [68,74,82–84]) compared to large recurrent, longer-term predictors without dense control signals [32, 85, 86]. The control models use various ways to disambiguate the task, such as the common use of temporally dense trajectory or phase information at each frame. Multi-frame prediction models on the other hand often rely on longer past motion context coupled with large recurrent architectures in order to disambiguate the task and thus require much more capacity.

In animation, short-term predictions (<0.5 seconds) tend to be deterministic given some conditions such as seed frames. This is why most reconstruction-based methods can be satisfactory for such short-term tasks. Numerous works on motion prediction have used some angular reconstruction loss while reporting on a common Euler angles benchmark [32, 53, 55, 87–89]. However, human motion is inherently stochastic for longer time horizons and reconstruction losses for those tasks quickly show limitations. It is harder in these cases to specify a good loss function as the objective often becomes to produce realistic motion, regardless of the ground-truth sequence. Li *et al.* [65] showed that a fixed scheduled sampling rate [63] can allow LSTMs to produce very long sequences of plausible motion, while not being able to quantitatively establish their results. Ghosh *et al.* [88] leveraged a pre-trained action classifier to evaluate long-term generation under the hypotheses that motion deterioration should impact negatively classification performance, and that classification accuracy should correlate with human judgement. Barsoum *et al.* [85] use a disconnected discriminator -

not used in the generator loss - in order to keep track of their best model during training. This idea is also used by Corona *et al.* [86], who also extend the Inception score [90] used in the image domain to motion, as well as a pair-wise distance metric on their generated motions in order to assess diversity. While these advances are all useful steps into finding the right metrics for capturing motion quality, it seems that this research topic is still open and relevant. Moreover, depending on which metric captures best human preferences, it might not be straightforward to find which losses functions to minimize as not all quality heuristics allow backpropagation. Pure GAN-based approaches tend to avoid this question by assuming that a discriminator will consider the same features as a human to score samples, which is not always the case.

2.2.3 Motion Datasets

We list in Table 2.1 the different public motion datasets we use in the following chapters. The LaFAN1 motion dataset is one of our contributions in Chapter 5 and was released alongside the article in an effort to foster future comparisons on the task of transition generation. We give further details about how we used these datasets below.

Table 2.1 Overview of different 3D motion datasets used in this research. Different versions of these datasets can be accessible online. We report some details specific to the version used in our research.

Identifier	**Year**	**Subjects**	**Categories**	**Labels**	**Frames**	**FPS**	**Hours**
CMU [3]	2005	101	6	23	3 893 379	120	9.0
HDM05 [51]	2007	5	5	65	606 191	120	1.4
Human3.6M [91, 92]	2014	7	-	15	2 110 396	200	2.9
NTU RGB+D [52]	2016	40	-	60	4 780 949	30	44.3
LaFAN1[4]	2020	5	15	-	496 672	30	4.6

MOCAP Database HDM05

The HDM05 dataset is a well structured, high quality MOCAP dataset that contains long takes or labelled cuts taken from these takes. In Chapter 3, we use the labelled clips in C3D format to perform supervised training and classification. C3D files contain 3D point clouds of recorded marker positions. They correspond to the raw data that is produced in a MOCAP session. We used a subset of 23 markers to feed the neural networks. We uniformly subsample

[3] http://mocap.cs.cmu.edu/

[4] https://github.com/ubisoftinc/Ubisoft-LaForge-Animation-Dataset

frames from this dataset to get a frame rate of 30 Frames Per Second (FPS). Unfortunately, a number of works on this dataset use an unrealistic partitioning of the data by randomly sampling a test set, which leads to excellent results. We address this in Chapter 3 and show that isolating an actor to use it as a test set leads to lower, more plausible performances.

CMU Graphics Lab Motion Capture Database

The CMU database is a larger mid-quality MOCAP collection. We use it in Chapter 3 without labels for our semi-supervised experiments. We also use the C3D format, and select 23 markers that are common to the HDM05 dataset in order to have a unified data format when mixing both sets. We also subsample the data to reach 30 FPS.

Human 3.6M

Human 3.6M was designed for a public challenge and thus retains some sequences private for testing. We use complete skeletal information (positions and rotations) and subsample the data to 25 FPS. Although this dataset has been widely adopted in the Computer Vision and Pattern Recognition (CVPR) conference community, we explain in Chapter 5 why we think it is ill-suited for motion prediction and completion tasks, motivating our publication of LaFAN1.

NTU RGB+D

The NTU RGB+D dataset is a very large, yet low-quality collection of motion sequences captured through 3 Microsoft Kinect V2 sensors at different angles (i.e. each motion has 3 recordings). We use the skeletal data inferred from the depth sensor. This corresponds to the 3D positions of 25 joints.

Ubisoft LaForge Animation Dataset LaFAN1

Our proposed LaFAN1 dataset is a very-high quality MOCAP database recorded in Ubisoft Montreal's motion capture studio. It contains long uninterrupted sequences of motion that can be broadly categorized, but it has no labels and is not meant for supervised-learning. It is released as a collection of BVH files[5], fully defining skeletal motion.

[5]We also recently released the C3D files.

2.2.4 Motion Classification

Human action recognition with computer vision has been for long a challenging task, especially before depth sensors and marker-based motion capture became accessible. Indeed, action recognition from 2D video has its own set of challenges due to the low signal-to-noise ratio in the inputs. Images must usually be converted into an intermediate representation such as silhouette frames [93, 94] before or instead of another custom feature space [95, 96] to increase the signal-to-noise ratio and be amenable to classification. Machine learning and especially deep learning approaches aim at learning such higher-level representations and features from data. Important work using Hidden Markov Models (HMM) [97] or Support Vector Machines (SVM) [98, 99] still used hand-crafted representations in order to leverage modest datasets, while much larger recent datasets [100–102] coupled with modern deep learning architectures [103–106] enable and further motivate the use of minimal pre-processing.

Depth sensors, which provide per-pixel depth information over a field of view offer a natural way of filtering background information from a subject to facilitate segmentation of human shapes. When combined with an RGB camera, the RGB+D stream offers a useful extra depth dimension per image fragment. Open datasets with RGB+D videos such as MSR Daily Activity3D [107] and MSR Action3D [108] sparked important progress in action recognition techniques [109–111].

An important feature transformation from RGB-D streams was proposed by Shotton *et al.* [112] who were able to detect 3D joint positions in real-time from the depth sensor alone. Such 3D locations (also contained in the MSR Daily Activity3D dataset) have been used as features for activity recognition. MOCAP datasets such as CMU or HDM05 can offer complete skeletal information and thus also contain 3D positions of joints. Many different methods using hand-crafted features from 3D positions have been proposed using classical machine learning approaches such as HMMs [113], Nearest Neighbor variants [114, 115] or SVMs [116–118]. Deep learning methods can also benefit from custom representations of data. Ijjina *et al.* [119] define distance-based feature of joints as inputs to a multi-layer perceptron, Zhang *et al.* [120] define geometric features as inputs to a LSTM, while Ke *et al.* [121] engineer features from the cylindrical coordinates of body joints and feed them to a classification CNN with convolution layers pretrained on an image dataset. Once again, when leveraging deep models and enough data, preprocessing of the datasets becomes minimal as learning the right features can lead to improved performance. The work then focuses on the architectures, losses, regularizers, and other training strategies. Cho & Chen [122] use MLPs with minimal feature extraction and with an auxiliary unsupervised loss for classification on HDM05. Our work in Chapter 3 is thus closely related, while we study similar approaches

with a recurrent encoder-decoder architecture. Du *et al.* [123] propose a hierarchical RNN where skeleton segments are treated independently in lower layers and iteratively merged after each further layer, while Zhu *et al.* propose regularization strategies for LSTM-based action recognition models.

Joint positions are a convenient representation of human poses as they are much more compact than images with far less irrelevant information. Positions however do not contain orientation information, which in some specific cases can hold crucial information on the motion (think pressing an elevator button vs. turning a door knob). Vemulapalli *et al.* [124,125] use rotational information to express motion and Lie group theory to perform classification in the Lie algebra with SVMs. Ofli *et al.* [126] automatically define sequences of most informative joints based on angular data as inputs to their classifiers.

2.2.5 Motion Generation

We review here different families of methods for motion generation even though boundaries between those families may be somewhat blurry. Note also that within the following families, we may further differentiate between online control and motion prediction. We consider online control problems where a temporally-dense signal is given during the generation, usually aimed at real-time applications. In motion prediction, only the start conditions are known and a segment of motion must be generated without additional inputs. The main focus of Chapters 4 and 5 is on transition generation, which can be seen as motion prediction conditioned on extra target information, without temporally-dense signals.

Graph Based Methods

Graph based methods for motion generation have been a popular approach to motion control. They are easily interpretable and based on MOCAP data, usually yielding good visual quality, yet they sometimes offer limited expressiveness or responsiveness. Nodes may represent motion segments or poses while edges may represent inter-sequence transition points, but depending on the method this can be inverted. Random walks in the graph then generates random motion, while searches performed with respect to given user constraint allow for controlled generation. Kovar *et al.* [81] coined the term *Motion Graphs*, in which they automatically detect potential transition points between motion segments to augment the data with generated transitions through blending. This is to cover a wider range of motion and augment the responsiveness of the model. In order to accelerate the search through the graph, two-layers hierarchical Motion Graphs have been proposed [78, 127, 128] where the higher level of the graph is constructed through clustering of similar frames or segments.

Markov processes are often well-suited to frame motion dynamics at a certain level of the graph [78,128]. A higher level of abstraction was added by Beaudoin *et al.* [129] where nodes of the graph correspond to motion *motifs* defined as common sequences of clusters of similar poses. Safonova *et al.* [130] presented interpolated motion graphs, where generated motions consist of the mix of two independent paths in the graph for greater expressiveness. They use a fast approximation of the A* search algorithm in an augmented graph for generating motion with respect to user constraints.

For intuitive real-time control, Chai *et al.* [131] proposed to formulate a query based on low-dimensional marker inputs and past full-poses to find nearest candidates in a pre-computed neighbor graph. They then obtain a local linear model of motion through Principal Component Analysis (PCA) that they tune to minimize an error function based on constraints. Tautges *et al.* [132] proposed a similar method for accelerometer data inputs and perform lazy neighbor graph search to find the best sequences that explain past inputs, and optimize an energy function to select from those the next best pose given current inputs. Real-time control through a gamepad controller can be a harder challenge since inputs are even less correlated with human motion. Motion matching [133] tackles this with a looser motion graph definition where transitions between MOCAP sequences are in theory possible on every frame, similarly to Motion Fields [134]. Each frame has its own pose-feature vector based on foot placement and future trajectory. The system is based on an efficient search through the collection pose-features to minimize a cost function. If the best pose-feature vector found is a better candidate than the one corresponding to the next frame in the current animation, a switch to the corresponding animation of the corresponding frame is performed, with blending. This is an important algorithm that has been deployed in recent AAA games [135–137]. However, like most graph-based approaches, one significant drawback is the limited scalability due to memory requirements. Indeed, graph-based approaches rely on having the animation data in memory at runtime making them ill-suited for very large datasets.

Classical Machine Learning

Even though some works reviewed above have used some machine learning techniques in their pipelines such as PCA, we discuss here techniques that diverge from graph-based frameworks and that rely more heavily on ML techniques and probabilistic modeling. For example, using an energy-based model based on bio-mechanical theory, Liu *et al.* [138] learn style parameters of a subject from a single motion sequence through nonlinear inverse optimization. They find parameters that approximately minimize the required energy to produce the example motion given foot contact constraints. Style can then be applied on novel constraints. Conditional

restricted Boltzman machines with binary latent variables have been used by Taylor *et al.* [139] as auto-regressive generative models for motion. These models have the advantages of having low memory and compute requirements, but lack the capacity and expressiveness of the more general RNNs.

The Maximum A Posteriori (MAP) framework is an interesting framework for animation authoring with respect to user constraints. This formulation often leads to finding the best motion that fits constraints, regularized by a motion prior obtained from a MOCAP dataset. Chai *et al.* [140] define a m-order linear dynamical system on principal components of MOCAP poses and local dynamics as a motion prior. They then define losses according to motion priors and constraints to optimize within the MAP framework using sequential quadratic programming. Min *et al.* [141] model precise movements through low dimensional deformable models parameterized by eigen-vectors obtained through functional PCA. They model a motion prior by fitting a Gaussian Mixture Model (GMM) to the eigen-weights associated to each motion and use the MAP framework to generate constrained movements.

Gaussian Process Latent Variable Models (GPLVMs) [142] have also been used successfully to model human motion. Gaussian Processes systems have the advantages of modeling well uncertainty while GPLVMs also define a latent manifold that is often low-dimensional, yielding faster inference and potential visualizations of such a manifold. Grochow *et al.* [143] have used such a system to solve full-body, stylistic IK. Framing the problem as generic IK solving allows the authors to tackle multiple applications in addition to single-frame IK, such as full-body trajectories from key-points trajectories or 3D reconstruction from sparse markers. They implicitly model motion dynamics by including velocities in their observed feature-vectors, while Wang *et al.* [144] explicitly learn a latent non-linear dynamics model through factorisation of the latent probability distribution through time. This yiels smoother latent trajectories of the training sequences. Their model performs on a per-sequence basis, while Ye *et al.* [145] propose a dynamical model that operates on a per-frame basis and interleave model inference and physics simulation in order to allow a character to respond to physical perturbations. Levine *et al.* [146] also model motion with a GPLVM that they regularize with a latent connectivity prior so that the training sequences are close to each other in latent space.

Combining motion graphs and probabilistic frameworks, Min *et al.* [147] presented Motion Graphs++ with nodes corresponding to morphable motion functions that are parametric models obtained through functional PCA of action segments. They learn priors over the function parameters with a Gaussian Mixture Model and transition probabilities with a Gaussian Process. The Maximum A Posteriori framework is then used to optimize a path

through the graph and function parameters that respect user constraints.

These methods are in general more powerful than purely graph-based methods in that they usually learn some sort of smooth manifold from which novel motions can emerge. However, even though they can produce good results from less data, they usually can't handle big heterogeneous datasets as the runtime computations scale with the data. This limits their expressiveness, as a single model handles a limited range of motion.

Physically-based Motion

Motion generation can also be tackled inside physically simulated environments, establishing a link with robotics. Even though we remain in the kinematics realm in the following chapters, animating and controlling physically-simulated humanoid characters is a related task from which many approaches can give relevant insights. Witkin & Kass [75] use physics to enforce the realism of generated motions. However, operating with physical constraints often leads to additional challenges, such as keeping balance [148]. Moreover, getting complex humanoid characters to produce human-like movements is especially challenging. Many techniques use Proportional-Derivative (PD) controllers - or *servos* - as low-level controllers in order to track desired angles for a physically-based actuated character. These simple controls used to track a desired angle θ_{target} for a Degree of Freedom (DoF) angle θ are summarized by:

$$\tau = k_p(\theta_{target} - \theta) - k_d(\dot{\theta}_{target} - \dot{\theta}) \tag{2.16}$$

where τ is the resulting torque applied on the joint motor for that DoF and the dotted values are time-derivatives of angles, often computed with finite differences. Parameters k_p and k_d are the proportional and derivative gains respectively, which are the parameters to tune or optimize, usually on a per-task basis. Hodgins *et al.* [77] define control laws based on motion theory and observations of athletes to provide desired angle trajectories for PD-controllers applied on their rigid-body models. Zordan *et al.* [79] use PD-controls to keep balance and track MOCAP clips simultaneously for a boxing character, in which the target MOCAP clips are modified with IK for specific punches and PD gains are modified on contacts. Physical simulation was further investigated by Zordan *et al.* [149] as a way to augment quality of transitions following impacts, where a forward simulation on a physical model drives the search for the next motion clip to play after an impact and the ragdoll motion is improved by using PD-controllers to track the desired angles of the next clip. Finite State Machines (FSM) with per-state, balance-aware PD-controls were used by Yin *et al.* [80] to provide real-time controllers for biped locomotion on physically-simulated bipeds. Similarly, Sok *et al.* [150] use PD-servos in finite states to interactively control a 2D biped.

They also optimize and augment the input data to counter character instability. Liu *et al.* [151] present an offline sampling method around MOCAP target states to find the right offset that will lead PD-controllers to follow the right trajectory. In later work [152], the distribution of samples is adapted iteratively, coupled with a sliding-window approach that allows the PD-controls to track longer motion sequences. Other approaches based on better approximations of musculo-skeletal interactions in humans [153–155] or other creatures [156] also show interesting results. A recurring challenge of these approaches is to have versatile controllers that can tackle various actions while operating in the physical simulation, since physics-related challenges such as balance add difficulty to any simple task.

Reinforcement Learning

Generating motion can also be framed as a Markov Decision Process (MDP) where an agent in a given state must take an action that is carried out in an environment. This brings the agent in a new state and yields a reward. The goal of Reinforcement Learning (RL) is to learn a policy that will maximize the sum of future rewards - called *return* - by choosing the best actions in any state. The expected return in a given state is often approximated by a value function, which is often used to learn or drive a policy. We refer the reader to Sutton & Barto [2] for a complete introduction.

When operating in the kinematic domain, actions can directly correspond to poses or motion clips. Lee *et al.* [157] use a collapsed motion graph and discretized target states to pre-compute a policy that will choose the right animation clip (edge) at each transition point (node) in the graph at runtime. Treuille *et al.* [158] approximate the optimal value function through basis functions that are linearly combined, lowering the dimensionality of the problem. They further accelerate the optimization by tabulating the value function over *switchable* and *separable* dimensions. The control policy then chooses motion clips that are blended by a motion model to reach target states while maximising the return based on the learned value function. Actions can also correspond to character poses instead of clips as done by Lee *et al.* [134]. They discretize their value function over the continuous state parameters and over time for tractability and compression and interpolate those values accordingly at runtime. Operating in the per-state paradigm instead of on a per-clip basis leads to quicker response time. Using their learnt GPLVM, Levine *et al.* learn a near-optimal policy that navigates the latent space for desired goals using dynamic programming.

In the physical domain, Coros *et al.* [159] define their actions as selecting pre-designed PD controllers from Yin *et al.* [80] that output joint torques and use fitted value iteration to learn the value function. Tackling more varied actions, Liu *et al.* [160] use the SAMCON system

of [151, 152] to provide control fragments as a dataset of physically realised motions in order to learn per-fragment linear feedback control policies that operate at a lower frequency than the physical simulation.

Most of these RL approaches depend on some sort of discretization, either on the value function, state space, or task in order to simplify computations or to have better performing, specialized controllers to be combined through a higher level process. This will often lead to sub-optimal performance due to interpolation and blending of different controllers or outputs. While Peng *et al.* [161] use K-nearest-neighbors with a Gaussian kernel as an approximation of the continuous value and policy functions, neural networks have become recently the most popular choice for such approximations as we will see below.

Deep Learning

In recent years, deep learning has surpassed many other ML techniques in various domains, and motion generation is no exception. Learning a motion manifold through a deep neural network was proposed by Holden *et al.* [162], in which a CNN auto-encoder is used to encode clips of motion into a motion latent space. This can be used to perform multiple types of denoising robustly. The same manifold is then used in [163] to decode motions encoded by a feed-forward network only through higher-level parameters such as a displacement curve or end-effector constraints. Unconstrained motion prediction with RNNs is tackled by Fragkiadaki *et al.* [33] with recurrent layers surrounded by feed-forward encoder and decoder. This work fostered an significant string of approaches tackling the same task with RNNs. Notably, Jain *et al.* [87] apply RNNs on spatio-temporal graphs of human skeletons or activities, and Martinez *et al.* [32] propose residual RNNs to fix the common problem of first-frame jump in the prediction. Several others [53–55, 85, 89, 164, 165] investigate different recurrent architectures and loss functions to tackle similar prediction benchmarks, while Li *et al.* [65] and Ghosh *et al.* [88] focus on generating long or unbounded predictions. Bütepage *et al.* [166] investigate and compare different feed-forward auto-encoder architectures for classification and prediction. Zhang *et al.* [167] tackle the task of keyframe animation with RNNs - similarly to our focus in Chapters 4 and 5 - in a simpler 2D setting.

Online control from a gamepad or other real-time user inputs through deep neural networks is also an active area of research. With such approaches, greater care must be taken with respect to the computation requirements of the resulting system. Also, by definition, the task consists of unbounded generation, which is a considerable challenge that can be mitigated by the temporally dense user inputs that help disambiguate what pose(s) to generate next. However, those user inputs might not be sufficient and further techniques must be applied. Holden *et*

al. [68] use phase information to blend weights of simple feed-forward neural networks in order to perform such disambiguation. Zhang & Starke *et al.* [74] present a more general approach in which expert models are blended, with blending weights output by a gating network instead of relying on phase. In later work [82], they augment the input state with phase information, goal inputs and volumetric scene representations to allow the model to generate scene interactions. Automatically extracted local-motion phases for each end-effector bones [82] can also disambiguate complex movements and interactions. Disambiguation can also be addressed, at least partially through recurrent layers, since their memory state can encode past motion. Lee *et al.* [168] train RNNs to handle various spatio-temporal constraints to generate activity motions online. Recently, Holden *et al.* [169] replaced the database lookups and pose extraction of Motion Matching [135,170] by neural networks to significantly reduce memory requirements.

Deep Reinforcement Learning

Using neural networks as powerful function approximators for value functions, state-value functions or policies have brought significant improvements to RL approaches. Indeed, deep reinforcement learning removes the need for discretization commonly used in many of the RL approaches listed before. Peng *et al.* [171] tackle 2D biped locomotion and train a deep neural network with a convolutional component for terrain representation learning. The network outputs multiple continuous actions along their estimated Q-values resulting in different, specialized action-critic experts. In DeepLoco [172] two different continuous policies are trained as high-level and low-level controllers and operate on different timescales. The slower high-level controller outputs short-term footstep plans as goals for the fast low-level controller. Similarly to [171] a CNN component is present in the low-level controller for terrain conditioning, but locomotion is performed in 3D space. Liu *et al.* [173] use deep Q-learning [174] to learn a scheduler of short control fragments for real-time control. Actions correspond to control fragment indices and the reward penalizes playing fragments out-of-sequence. Similarly to Sok *et al.* [150], Bergamin *et al.* [84] rely on modifying reference motions for robust tracking, while they operate in 3D. The reference motions for learning in their case comes from motion matching that adapts well to many control scenarios, and they use Proximal Policy Optimization (PPO) to learn a feedback policy that modifies the reference motion to be tracked by PD-servos. Ling *et al.* [83] also use PPO to learn controllers, but in contrast to [84], they operate in kinematic space, and define their action space as the latent space of a learned motion VAE. These works try to bridge the gap between realism and control which is a hard challenge. Indeed, given the current framing of the control task in video games, players often expect predictable and immediate responses which doesn't

necessarily correspond to reality; humans have inertia and muscle activations are noisy, as are our perception and our environment.

2.3 Modulating Neural Networks on Conditioning Information

In Chapter 5, we modulate the inputs of an RNN through additive latent modifiers, impacting the resulting animation. This allowed us to force the network to consider the conditioning information. It proved more effective than adding extra conditioning information in the inputs, as this alternative often led to the model to ignore such information. This trick is related to a recent trend in deep learning where neural architectures perform conditioning with simple - often affine - transformations of the hidden states instead of augmenting the input space with conditioning dimensions. One early example of this was presented by Dumoulin *et al.* [175] in which they tackle the task of style transfer on images with a model capable of handling multiple styles through *conditional instance normalization*. This normalization, or *modulation* scheme is often performed after a batch normalization [176] and can then be seen as a conditional *de-normalization* or *de-modulation*. For example, FiLM networks for visual reasoning [177] output affine transformation parameters based on an LSTM state to modulate the latent representations of a CNN. Similarly, Adaptive Instance Normalization (AdaIN) was used for arbitrary style transfer for images [178] and recently for animations [179]. One could consider StyleGAN [50, 180] as the extreme case of conditioning through modulation, where the input to a generator is a constant learned vector and outputs are shaped by AdaIN layers throughout the generator with parameters coming from a learned mapping from a noise vector. In our case, in Chapter 5 we simply use shifting (as opposed to shifting and scaling) of the latent representations with temporal information and target noise.

CHAPTER 3 RECURRENT SEMI-SUPERVISED CLASSIFICATION AND CONSTRAINED ADVERSARIAL GENERATION WITH MOTION CAPTURE DATA

Abstract

We explore recurrent encoder multi-decoder neural network architectures for semi-supervised sequence classification and reconstruction. We find that the use of multiple reconstruction modules helps models generalize in a classification task when only a small amount of labeled data is available, which is often the case in practice. Such models provide useful high-level representations of motions allowing clustering, searching and faster labeling of new sequences. We also propose a new, realistic partitioning of a well-known, high quality motion-capture dataset for better evaluations. We further explore a novel formulation for future-predicting decoders based on conditional recurrent generative adversarial networks, for which we propose both soft and hard constraints for transition generation derived from desired physical properties of synthesized future movements and desired animation goals. We find that using such constraints allow to stabilize the training of recurrent adversarial architectures for animation generation.

3.1 Introduction

It is often the case that for a given task only a small amount of labeled data is available compared to a much larger amount of unlabeled data. In these cases, semi-supervised learning may be preferred to supervised learning as it uses all the available data for training, and has good regularization and optimization properties [181, 182]. A common technique for semi-supervised learning is to perform training in two phases: unsupervised pre-training, followed by supervised fine tuning [181–184]. The unsupervised pre-training task often consists of training a variant of an auto-encoder (e.g. a denoising auto-encoder) to reconstruct the data. This helps the network bring its initial parameters into a good region of the high-dimensional parameter space before starting to train the model on the supervised task of interest.

Advances in Recurrent Encoder-Decoder networks have afforded models the ability to perform both supervised learning and unsupervised learning. These architectures are often based on the capacity that recurrent neural networks (RNNs) have to model temporal dependencies in sequential inputs. When handling a sequence, the last hidden state of an RNN can summarize information about the whole sequence, allowing the model to encode input sequences of

variable lengths in fixed length vector representations. The separation between the encoder and the decoder networks naturally allows one to easily add, modify or re-purpose decoders for desired tasks. Using multiple decoders forces the encoder to learn rich, multipurpose representations. Additionally, this also allows semi-supervised training in a single phase.

In this work, we jointly train a classifier model with optional frame-reconstruction, frame-classification, and sequence reconstruction decoders which all affect the sequence representation used by the upper, classification-only layers. Our empirical study shows that adding the unsupervised decoders have a regularizing effect on the supervised sequence classification task when few labeled sequences are available. We demonstrate this improvement on HDM05 [51], a well known action recognition dataset. We also explore the limits of this method using the more recent NTU-RGB+D dataset [52]. Finally, we perform separate experiments in which a new constrained recurrent adversarial decoder learns to generate future frames conditioned on a similarly encoded past-context representation. We use Long-Short-Term-Memory (LSTM) models to encode and decode sequences, and a simple multilayer perceptron (MLP) to classify them. Our adversarial discriminator is a bi-directional LSTM (BDLSTM), and outputs predictions at each timestep.

Our main contributions are the following: We introduce a novel Recurrent Encoder, Multi-Decoder architecture which allows for semi-supervised learning with sequences. We define and execute a set of experiments using more realistic and representative test set partitioning of a widely used public MOCAP dataset, thereby facilitating more informative future evaluations. We show that this updated classification task is still challenging when having an appropriate test set. We show improvements over our implementation of previous state-of-the-art techniques for action recognition on such well defined experiments. We also present a novel conditional Recurrent Generative Adversarial Network (ReGAN) architecture for predicting future continuous trajectories which integrate multiple physics based constraints as well as desired animation properties. We show that using such data-driven constraints prevents the adversarial learning of the recurrent generator from diverging, greatly improving the generated transitions.

3.2 Related Work

3.2.1 MOCAP datasets

One challenge with the application of deep learning to MOCAP data is the lack of strongly labeled, quality data. For this work, we used two high-quality publicly available MOCAP datasets and a bigger, lower-quality Kinect dataset. The first MOCAP dataset is the HDM05

dataset [51]. It contains 2329 labeled cuts that are very well suited for action recognition. We use the same 65 classes defined by Cho & Chen [122]. The second dataset is the CMU Graphics Lab Motion Capture Database[1]. This is a significantly bigger MOCAP dataset in terms of number of frames. It contains 2148 weakly labeled or unlabeled sequences. This dataset can hardly be used for supervised learning as the labeling of sequences, if any, was only made to give high level indications of the actions, and does not seem to have followed any stable conventions throughout the dataset. The work by Zhu *et al.* [185], Ijjina *et al.* [119], and Barnachon *et al.* [115] all use different custom class definitions to obtain quantitative results on CMU for classification. In the present work, we use this dataset for unsupervised learning only. The Kinect dataset is the NTU RGB+D dataset [52] which is to our knowledge the biggest motion dataset containing skeletal motion data. It contains 60 actions performed by 40 different actors, recorded with a Kinect 2 sensor. It consists of more than 56000 labeled sequences that either contain one or two subjects. Despite the fact that this dataset is approximately 24 times bigger than HDM05 and has a higher actor count, its lower quality and its well defined, realistic partitioning make action recognition in this context a challenging task. Its two evaluation schemes are based on either held-out actors or a held-out view angle. We wish to provide here a similarly well defined evaluation case for HDM05.

3.2.2 Recurrent-Encoder-Decoders

RNNs have proven over the years to be very powerful models for sequential data, such as speech [30, 186, 187], handwriting [26], text [24, 188], or as in our case, MOCAP [32, 33, 65, 123, 185, 189, 190]. We use LSTMs [25] without in-cell connections (as suggested by Breuel *et al.* [191]) in the models we explore here. A major advantage and key attribute of RNNs based on Recurrent Encoder-Decoders is their ability to transform variable-length sequences into a fixed-size vector in the encoder, then use one or more decoders to decode this vector for different purposes. Using an RNN as an encoder allows one to obtain this representation of the whole input sequence. Cho *et al.* [29] as well as Sutskever *et al.* [28] have used this approach for supervised sequence-to-sequence translation, with some differences in the choice of hidden units and in the use of an additional summary vector (and set of weights) in the case of Cho *et al.* [29]. Both approaches need a symbol of end-of-sequence to allow input and target sequences to have different lengths. They are trained to maximize the conditional probability of the target sequence given the input sequence. Our approach is more closely related to the one used by Srivastava *et al.* [57] in which they perform unsupervised learning, by either reconstructing the sequence, predicting the next frames, or both. In our work, an

[1] http://mocap.cs.cmu.edu/

additional decoder is used for classification of whole sequences, and the future generator may use an adversarial loss to improve generated sequences. Mahasseni *et al.* [192] also make use of a encoder-decoder LSTM to learn a skeletal motion manifold on small datasets. They then use this manifold to regularize another LSTM performing action recognition on videos (from pixels) by pushing its produced representations to be close to the learned manifold. We focus on skeletal data and combine classification and regularization with a motion manifold in a single phase.

3.2.3 Action recognition

Much of the prior work on MOCAP analysis has been based on hand-crafted features. For example, Chaudhry *et al.* [117] created bio-inspired features based on the findings of Hung *et al.* [193] on the neural encoding of shapes and, using Support Vector Machines (SVMs), have obtained good results on classification of 11 actions from the HDM05. Ijjina *et al.* [119] use some joints distance metrics based on domain knowledge to create features that are then used as inputs to a neural classifier (pre-trained as a stacked auto-encoder). They reach good accuracy for 3 custom classes in the CMU dataset. Using this prior domain knowledge helps in particular when the dataset is somewhat specialized and may contain actions of a certain type. However, if the goal is to have a generic action classifier that handles at least as many actions as found in HDM05, it might be more appropriate to learn those features with a more complex architecture. Barnachon *et al.* [115] use a learned vocabulary of key poses (from K-means) and use distances between histograms of sub-actions in order to classify ongoing actions. They present good accuracy (96.67%) on a custom subset of 33 actions from HDM05 (where training samples are taken at random). In our case, we wish to perform classification on the 65 HDM05 actions with a realistic test set.

End-to-end neural approaches have also been tried on HDM05 and CMU in which cases discriminating features are learned throughout the training of a neural network. Cho & Chen [122] have obtained good movement classification rates on simple sequences (cuts) on the HDM05 dataset using a Multi-Layer Perceptron (MLP) + Auto-Encoder (AE) hybrid. Chen *et al.* [194] tested multiple types of features, using extreme machine learning to classify, again, HDM05 cuts. Results were good in both cases, with accuracies of over 95% and 92% with 65 and 40 action classes respectively. Their models were trained at the frame level, and sequence classification was done by majority voting. Other work by Du *et al.* [123] treated the simple sequences' classification problem with the same action classes as Cho & Chen [122] with a hierarchical network handling in its first layer parts of the body separately (i.e. torso, arms and legs), and concatenating some of these parts in each layer until the whole body

is treated in the last hidden layer. They worked with RNNs to use context information, instead of concatenating features of some previous frames at each timestep. This led to better results, and their classification accuracy on simple sequences reached 96.92%. Finally, Zhu *et al.* [185] have a similar, but less constrained recurrent architecture that is regularized by a weight penalty based on the $l_{2,1}$ norm [195], which encourages parts of the network to focus one the most meaningful joints' or features' interactions. They report 97.25% accuracy on HDM05 for classification of simple sequences, with 65 classes.

Other relevant advances for skeletal motion recognition are applied on different datasets, such as NTU RGB+D, and once again focus on defining new motion data representations [120, 121, 125, 196] in order to inject domain knowledge directly into the inputs. Others propose instead new architectural variants to the neural networks for motion recognition [33,87,189,190,197] that sometimes induces prior knowledge in the architecture instead. Our own approach could be considered as an architectural modification that aims at reducing the need for domain knowledge by learning better representations for generalization using several decoders. It is in that sense more generic, and could therefore be combined easily with the above approaches or applied to different domains.

3.2.4 Generative Adversarial Networks

GANs [43] can be powerful tools to map a random noise distribution to a real data distribution and therefore to generate realistic samples. They are composed of a Generator (G) and a discriminator (D) that can be both deep neural networks. The goal of D is to tell if a sample comes from the real distribution or if it was generated by G (i.e. it is fake). The generator G learns from the likelihood signal provided by D in order to produce samples closer to real samples. While impressive work has been done with GANs or some of their variants on image generation [198–200], results on sequential data remains more limited. Ghosh *et al.* [201] make use of recurrent networks to generate the next plausible image as an answer to a sequence query. In that case, the answer should match the only ground truth answer. Mahasseni *et al.* [202] use an LSTM discriminator to classify reconstructed videos from generated summaries. In our case, we aim at producing a realistic series of positions (which might differ from the true trajectories) that lead to a target pose, conditioned on the compressed representation of the past context, a noise vector, and the target pose itself. During training, our generator and discriminator are not given ground truth frames during their generations/predictions, but always have information about the target pose. For text generation, Yu *et al.* [203] use recurrent networks with a policy gradient method to handle discrete outputs. We are interested here in plausible continuous trajectories and thus work

with continuous, differentiable, recurrent GANs.

3.2.5 Transition Generation

Approaches have been proposed for motion prediction with well-designed recurrent neural networks [32, 33, 65, 88] but they don't tackle the transition generation problem where the character needs to reach a desired target. Our method also uses an LSTM to generate frames, with added conditioning weights and computations in order to use the target information and the noise vector (in case of adversarial training). Therefore, our proposed constrained adversarial training stabilization procedure could naturally be applied with those motion prediction methods. Probabilistic models relying on more classical machine learning techniques such as [204] and [144] have been applied to motion gap-filling, but show limited scalability and are applied to individual types of actions. We apply here a more generic and scalable deep-learning approach based on LSTMs and GANs with additional stabilizing losses, and which has a constant runtime independent from the number of training samples.

3.3 Defining a good test set for HDM05

Most previous approaches for classification on HDM05 achieve good classification results when randomly separating the sequences into training, validation and test sets. However, this kind of partitioning is not a fair estimation of the generalization performance of the model, as the network may overfit the action styles of particular actors and perform poorly with new subjects. A more realistic partitioning of HDM05 would therefore be one based on performers, where action recognition accuracy on new subjects can be assessed.

Table 3.1 shows the results of our own implementation of previous state-of-the-art methods [122, 123, 185] on the HDM05 dataset using our controlled experimental setup. It shows how using held out actors as a test set can hurt the accuracy, and illustrates more clearly that despite the good results of previous methods, action classification on this dataset can still be a challenging task. In this setting, we use actors with initials 'tr' and 'dg' as test subjects, and a random 5% of the training data as a validation set for early stopping and hyper-parameter searches. Since using two out of five actors from HDM05 for testing represents approximately 40% of the sequences in the dataset, we tested again the method from Cho & Chen [122] with a balanced, shuffled partition having the same proportions of sequences in each sets to see if this was the only factor influencing the declining results. Finally, we applied our own pre-processing (PP) of the data with these techniques with our newly defined actor-based partitions to make further comparisons fair. Our preprocessing of the data is

Table 3.1]

Accuracies (Acc.) with different test sets, using techniques from Cho & Chen [122], Du *et al.* [123] and Zhu *et al.* [185]. and ours.

Technique	**Test set**	**Acc.(%)**
Du et al.	Random 10%, balanced	92.98
Zhu et al.	Random 10%, balanced	94.53
Cho & Chen	Random 10%, balanced	95.61
SC (Ours)	Random 10%, balanced	96.92
Cho & Chen	Random 40%, balanced	94.13
Cho & Chen	Actors [tr , dg]	64.36
Du et al.	Actors [tr , dg], PP	70.63
Cho & Chen	Actors [tr , dg], PP	**81.64**
Zhu et al.	Actors [tr , dg], PP	**81.64**

explained in Section 3.6.1 and its effects can be seen in in Figure 3.1. As we can see, results using our realistic actors-based partitioning of HDM05 are significantly lower, but our own pre-processing method of the data has a considerable positive effect. Since the techniques of Cho & Chen [122] and Zhu *et al.* [185] yielded the best results with our actor-based partitions and with our pre-processing method, the baseline test accuracy in the rest of this paper will be of 81.64% that was reached with those methods. Finally, we also include results from our sequence-classification architecture (SC), which doesn't use any reconstruction with a random 10% balanced test set and using the same preprocessing as Cho & Chen [122].

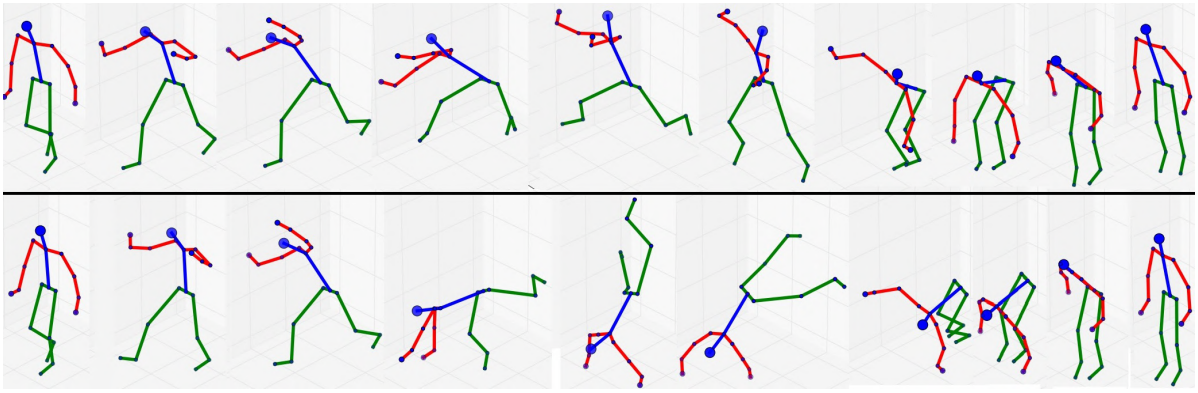
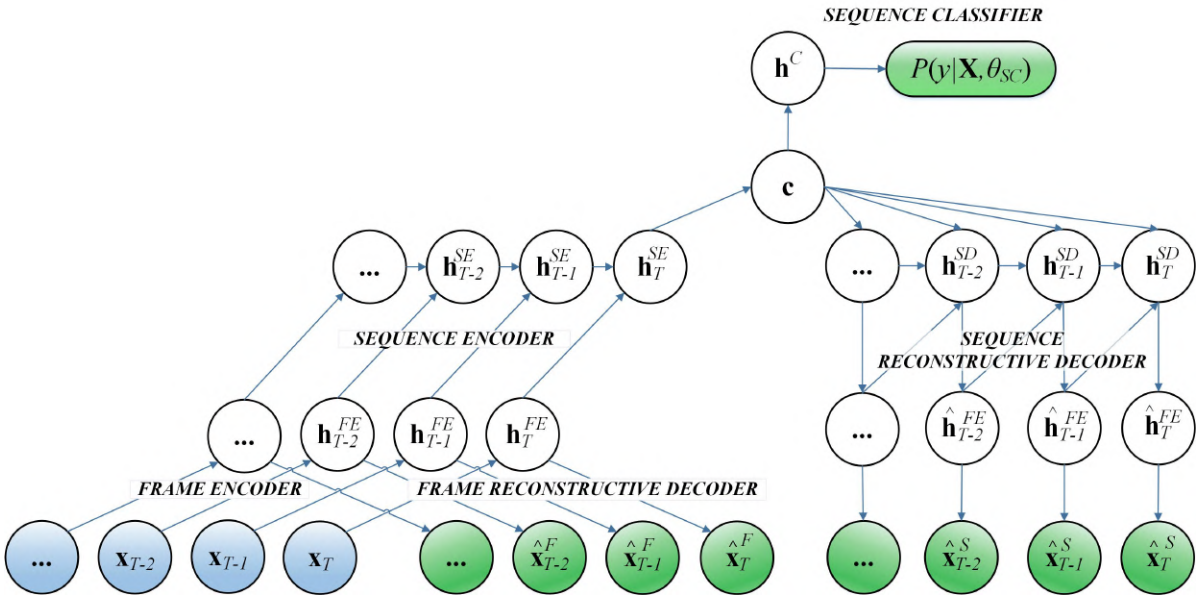


Figure 3.1 **Visual comparison of pre-processing methods** for a *cartWheel* movement from HDM05. UP: Same as Cho & Chen [122]. DOWN: our method, that allows for hips not to be parallel to the ground.

3.4 Our Semi-supervised Classification Models

Figure 3.2 shows an overview of the Frame Reconstructive-Sequence Reconstructive Classifier (FR-SRC) variant of the proposed architecture. The model is composed of 5 main components: a per-frame encoder, a per-frame reconstructive decoder, a sequence encoder, a sequence reconstructive decoder, and a sequence classifier. Each decoder along with the classifier produces an output used to calculate a cost. Individual costs have a more direct impact on different modules of the network, and their combination allow to produce evermore meaningful features throughout the model. Note that two additional modules, the future generator and the per-frame classifier, are used in some experiments but are not depctied in Figure 3.2 to avoid cluttering. The future generator is shown in Section 3.5. The per-frame classifier tries to classify the action based on single frames and takes the per-frame encoding to produce probabilities of actions.



(1).jpg (1).jpg (1).jpg (1).jpg

Figure 3.2 **The FR-SRC variant of the architecture studied.** This network produces 3 types of outputs (in green) with respect to a sequence $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_T]$ and its parameters θ. The set θ_{SC} includes all the weights and biases used to compute class probabilities. The hidden states of the frame encoder, sequence encoder and sequence reconstructive decoder are denoted here by $\mathbf{h}^{FE}$, $\mathbf{h}^{SE}$ and $\mathbf{h}^{SD}$ respectively. The sequence representation $\mathbf{c}$ is created from the hidden state of the sequence encoder at time T, and $\mathbf{h}^c$ represents the sequence classifier's fully connected layers (the softmax activation is not explicitly shown here).

3.4.1 Frame reconstruction

The frame auto-encoder's role is to learn robust per-frame features in an unsupervised manner by trying to reconstruct the clean version ($\mathbf{x}_t$) of a corrupted frame ($\tilde{\mathbf{x}}_t$) at time t. The reconstructive error ($l_{FR,t}$) we use is the well known mean squared error and we apply it for each frame, before calculating its average over the frames to get l_{FR}, where:

$$\begin{aligned} \mathbf{h}_t^{FE} &= u(\tilde{\mathbf{x}}_t) \\ \hat{\mathbf{x}}_t^{F} &= g(\mathbf{h}_t^{FE}) \\ l_{FR,t} &= \frac{1}{2}||\hat{\mathbf{x}}_t^{F} - \mathbf{x}_t||^2 \\ l_{FR} &= \frac{1}{T}\sum_{t=1}^{T} l_{FR,t} \end{aligned}$$

Here, $u()$ is the encoding function learned by the bottom feed-forward layers of the per-frame auto-encoder, while $g()$ is the decoding function learned by its upper layers. In further equations, $\mathbf{H}^{FE}$ will stand for the sequence of features $[\mathbf{h}_1^{FE}, ..., \mathbf{h}_T^{FE}]$ and we will dismiss the corruption sign over $\tilde{\mathbf{x}}$ as we will show equations for a test setting, where the frames are not corrupted. All further symbols W and $\mathbf{b}$ without subscript or superscript will represent the weight matrices and bias vectors for the current layer in order to lighten the notation.

3.4.2 Frame classification

The per-frame classifier uses $\mathbf{h}_t^{FE}$ as an input to yield activations $\mathbf{a}_t = \mathrm{W}(\mathbf{h}_t^{FE}) + \mathbf{b}$ on movement classes for every frame. These activations are then summed over all frames into $\mathbf{a}_f = \sum_{t=1}^{T} \mathbf{a}_t$ and a softmax operation is applied on the result, yielding class probabilities $P(y_k)$ given all the frames $\mathbf{x}_t$ of the whole sequence $\mathbf{X}$, and the parameters of the frame encoder θ_{FE}: $P(y_k|\mathbf{X}, \theta_{FE}) = \mathbf{s}_{f,k} = e^{\mathbf{a}_{f,k}}(\sum_{i=1}^{K} e^{\mathbf{a}_{f,i}})^{-1}$. This is similar to the operation used by Du et al. [123] to classify sequences based on a sequence of activations but differs in the fact that we do not use outputs from recurrent layers here. We use the negative log-likelihood of the correct class as our frame-classification loss:

$$l_{FC} = -log(P(Y = y_k|\mathbf{X}, \theta_{FE}))$$

The combination of the frame auto-encoder and the frame classifier gives something very similar to Cho & Chen's [122] approach, except that each frame's input does not contain information about a previous frame. When per-frame reconstruction is not used, the model still encodes frames with $u()$ before outputting probabilities with a softmax.

3.4.3 Sequence encoding

The LSTM encoder's purpose is to encode the whole sequence of learned features into a fixed-length summary vector $\mathbf{c}$ that models temporal dependencies, and which can be used for both supervised and unsupervised tasks,

$$\mathbf{c} = c(\mathbf{X}) = \tanh(\mathrm{W}\mathbf{h}_T^{SE} + \mathbf{b}),$$

where, $c()$ is a fully connected layer parametrized by the weight matrix $\mathbf{W}$. It uses the last hidden state of the LSTM encoder $\mathbf{h}_T^{SE}$ as its input. The encoder itself takes $\mathbf{H}^{FE}$ as an input sequence.

3.4.4 Sequence reconstruction

If the sequence reconstructive decoder is present, it learns to reconstruct the sequence $\mathbf{X}$ that was fed to the LSTM encoder. At each timestep, the LSTM decoder uses its previous output along its previous hidden state in order to predict the next frame in an auto-regressive fashion. With the outputted $\hat{\mathbf{X}} = [\hat{\mathbf{x}}_1^S, ..., \hat{\mathbf{x}}_T^S]$ from the decoder, and the frame decoding function $g()$, we can calculate our sequence reconstruction error (l_{SR}) as a frame-wise MSE loss with the original input sequence:

$$\begin{aligned}
\mathbf{h}_t^{SD} &= \tanh(\mathrm{W}\hat{\mathbf{h}}_{t-1}^{FE} + \mathrm{U}\mathbf{h}_{t-1}^{SD} + \mathbf{C}\mathbf{c} + \mathbf{b}) \\
\hat{\mathbf{h}}_t^{FE} &= \tanh(\mathrm{W}\mathbf{h}_t^{SD} + \mathbf{b}) \\
\hat{\mathbf{x}}_t &= g(\hat{\mathbf{h}}_t^{FE}) \\
l_{SR,t} &= \frac{1}{2}||\hat{\mathbf{x}}_t^S - \mathbf{x}_t||^2 \\
l_{SR} &= \frac{1}{T}\sum_{t=1}^{T} l_{SR,t}
\end{aligned}$$

Here, W, U and $\mathbf{C}$ are input, recurrent and conditioning weight matrices of the LSTM layer.

3.4.5 Sequence classification

The sequence classifier is a MLP that outputs class probabilities based on the summary vector. This is the main task of interest, the sequence classifier is therefore used in all of our our experiments. We again use the negative log-likelihood as the sequence classification

error (l_{SC}):

$$\mathbf{h}^C = \mathrm{W}\mathbf{c} + \mathbf{b},$$
$$\mathbf{a}_{seq} = \mathrm{W}\mathbf{h}^C + \mathbf{b}$$
$$P(y_k|\mathbf{X}, \theta_{SC}) = \mathbf{s}_{seq,k} = \frac{e^{\mathbf{a}_{seq,k}}}{\sum_{i=1}^{K} e^{\mathbf{a}_{seq,i}}}$$
$$l_{SC} = -log(P(Y = y_k|\mathbf{X}, \theta_{SC}))$$

3.4.6 Total loss

Using a reconstruction weight ω, we can define different models with different loss functions, enabling some or all of the modules of the architecture. For example, the complete FRC-SRC loss is defined as :

$$\ell_{frc-src} = l_{SC} + l_{FC} + \omega \cdot \frac{l_{FR} + l_{SR}}{2}$$

In the generic case, the total loss ℓ can be defined as follow:

$$\ell = l_{SC} + i(FC) \cdot l_{FC} + \omega \cdot \frac{i(FR) \cdot l_{FR} + i(SR) \cdot l_{SR}}{i(FR) + i(SR)} \tag{3.1}$$

where the indicator function $i(m)$ is simply equal to 1 when the module is present and 0 when it is not. Removing all optional decoders will result in a Sequence Classifier only (SC) network. Adding sequence reconstruction to this model will yield a Sequence Reconstructive Classifier (SRC). Adding instead frame reconstruction to the SC will give a Frame Reconstructive-Sequence Classifier (FR-SC), while adding frame reconstruction to the SRC will yield a Frame Reconstructive SRC (FR-SRC). Finally, adding the last module will result in a Frame Reconstructive Classifier-SRC (FRC-SRC).

3.5 Our Constrained Conditional Generation Model

As mentioned above, we have also developed a novel constrained conditional recurrent generative adversarial model aimed at creating high quality conditional transition animations. We explored in this context how one could stabilize the training of GANs combined with RNNs, which are both known to be hard to train, by using using physics-based soft constraints forcing the generated clips to respect certain statistics and actual physical constraints of the data. As a tool for generating transitions could be beneficial to animators when desired segments are missing, using GANs for such a task would naturally allow sampling capabilities for such

a tool. This motivates our exploration with GANs and why stabilizing their learning could be beneficial. Figure 3.3 shows a summary of the generator model. The *past context encoder* has the same structure as the sequence encoder used for classification described above. We describe the other components below.

(1).png (1).png (1).png (1).png

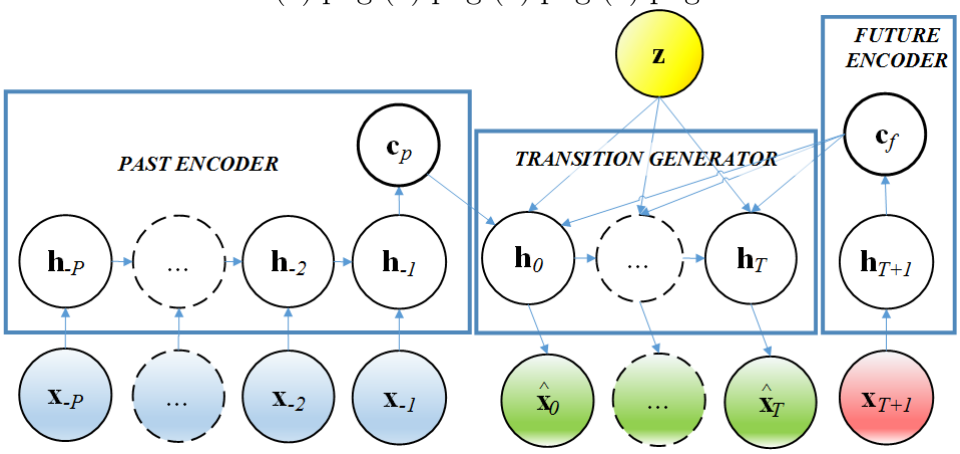


Figure 3.3 **Overview of our generative architecture.** Stacked LSTM layers are not shown. Different time indices are used for this problem.

3.5.1 Future key-pose encoder

Since the future key-pose is a single vector (single frame), we use here a stack of fully connected layers with no recurrence with the same number of neurons as the past encoder. Similarly to past encoding, we create a future context representation $\mathbf{c}_f$ that will be used by the generator. With a single layer, $\mathbf{c}_f = \tanh(\mathrm{W}\mathbf{x}_{T+1} + b)$, where $\mathbf{x}_{T+1}$ is the future key-pose frame at time $t = T + 1$.

3.5.2 Transition generator

Our transition generator is a stack of LSTM layers with additional conditioning connections $\mathbf{C}^{\{i,o,f,c\}}$ that process the conditioning vector $\mathbf{r}$, which is a concatenation of the sampled noise vector $\mathbf{z}$ and the context vector $\mathbf{c}_f$ at each timestep, allowing them to use the target information while generating the transition. The gate and cell values for a single-layer transition generator are computed as follow:

$$
\begin{aligned}
\mathbf{i}_t &= \text{sigmoid}(\text{W}^{(i)}\mathbf{x}_{t-1} + \text{U}^{(i)}\mathbf{h}_{t-1} + \mathbf{C}^{(i)}\mathbf{r} + \mathbf{b}^{(i)}) \\
\mathbf{o}_t &= \text{sigmoid}(\text{W}^{(o)}\mathbf{x}_{t-1} + \text{U}^{(o)}\mathbf{h}_{t-1} + \mathbf{C}^{(o)}\mathbf{r} + \mathbf{b}^{(o)}) \\
\mathbf{f}_t &= \text{sigmoid}(\text{W}^{(f)}\mathbf{x}_{t-1} + \text{U}^{(f)}\mathbf{h}_{t-1} + \mathbf{C}^{(f)}\mathbf{r} + \mathbf{b}^{(f)}) \\
\hat{\mathbf{c}}_t &= \text{W}^{(c)}\mathbf{x}_{t-1} + \text{W}^{(c)}\mathbf{h}_{t-1} + \mathbf{C}^{(c)}\mathbf{r} + \mathbf{b}^{(c)} \\
\mathbf{c}_t &= \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tanh(\hat{\mathbf{c}}_t) \\
\mathbf{h}_t &= \mathbf{o}_{t+1} \odot \tanh(\mathbf{c}_t)
\end{aligned}
$$

where the different $\text{W}^{\{i,o,f,c\}}$ and $\text{U}^{\{i,o,f,c\}}$ matrices are the usual input and recurrent connection weights for the different gates and the cell computation, and $\mathbf{b}^{\{i,o,f,c\}}$ are the bias vectors. The frame output $\mathbf{x}_t$ can then be computed from $\mathbf{h}_t$ with the same frame decoding function $\mathbf{g}()$ as the one used for sequence reconstruction.

3.5.3 Discriminator

The discriminator consists of a stack of BDLSTM layers that take as input a minibatch of sequences with real and fake transitions and tries to yield higher probabilities for real transitions. The output layer performs a per-frame feed-forward activation $\mathbf{a}_t = \text{W}(\mathbf{h}_t^D) + \mathbf{b}$. These activations, for all transition frames, are then summed into a tensor on which the sigmoid classification is done, similarly to what the per-frame classifier does.

3.5.4 Reconstruction objective

A common way to train a generative architecture is to use a reconstruction loss on the outputted sequences. We can obtain our reconstruction loss with

$$
l_{rec} = \frac{1}{T+1} \sum_{t=0}^{T} \frac{1}{2} ||\hat{\mathbf{x}}_t - \mathbf{x}_t||^2
$$

where $T+1$ is the number of frames in the transition.

3.5.5 Adversarial objective

For the adversarial loss, both our generator G and our discriminator D are needed. Regular adversarial networks [43] are designed to be trained by playing a minimax game with

$$
\min_G \max_D V(D,G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))].
$$

In our case, however, incoming data to the discriminator is formatted as a sequence $\mathbf{X}$ of frames containing the past frames $\mathbf{X}_{past}$, transition frames $\mathbf{X}_{trans}$ and future frame $\mathbf{x}_{target}$. Our generator is not only conditioned on the noise vector $\mathbf{z}$, but also the past context $\mathbf{c}_p$ and future context $\mathbf{c}_f$ provided by the past and future encoders. We therefore have the following objective:

$$\begin{aligned}\min_G \max_D V(D,G) =& E_{\mathbf{X}\sim p_{data}(\mathbf{X})}[\log D(\mathbf{X}_{trans}|\mathbf{X}_{past}, \mathbf{x}_{target})] \\ &+ E_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z}, \mathbf{c}_p, \mathbf{c}_f)))]\end{aligned} \tag{3.2}$$

3.5.6 Bone length consistency constraint

Similarly to [163], we apply a bone length consistency constraint in order to preserve rigid bone lengths throughout frames of the generated sequences. In our case, we base our prior knowledge of the bone lengths variations on statistics gathered on the training set. We therefore calculate a vector $\mathbf{b}^{(m)}$ of mean bone lengths differences between consecutive frames, as well as the vector $\mathbf{b}^{(v)}$ of variances for all bones differences. This way, we formulate a Gaussian prior (which we expect to be very narrow) on every bone length variations between two frames, which we fit during training using the Log-Likelihood (**LLB**) of every bone differences:

$$\mathbf{LLB} = -\frac{1}{2}\log(2\pi\mathbf{b}^{(v)}) - \frac{(\mathbf{B} - \mathbf{b}^{(m)})^2}{2\mathbf{b}^{(v)}} \tag{3.3}$$

where $\mathbf{B}$ is the matrix of bone length differences on every frame for every bone in the generated sequence, to which the target key-pose is appended. We can therefore retrieve our bone length consistency loss (L_{bone}) by averaging all the negative **LLB** :

$$l_{bone} = \frac{1}{N}\frac{1}{(T+2)}\sum_{n=0}^{N-1}\sum_{t=0}^{T+1} -\mathbf{LLB}_{n,t} \tag{3.4}$$

where N is the number of bones.

3.5.7 Joint velocity constraint

We also apply a joint velocity constraint based on a mixture of Gaussian priors retrieved from the training set. We perform an EM algorithm to fit two velocity Gaussians for all input dimensions d, based on velocities at every frame in the training set. Since bone velocities are very close to 0 on most frames, our mixtures often contain this spike (with a very small variance) and a broader distribution (higher variance). With these mixtures for every joint,

we can add a negative log likelihood loss on velocities to constrain bones to have normal velocities, reducing gaps between consecutive frames in the generated transitions. For a given vector $\mathbf{v}^{(m)}$ of mean velocities per bones and another vector $\mathbf{v}^{(v)}$ of variances per bones, we get the log-likelihood **LLV**:

$$\mathbf{LLV} = -\frac{1}{2}\log(2\pi\mathbf{v}^{(v)}) - \frac{(\mathbf{V} - \mathbf{v}^{(m)})^2}{2\mathbf{v}^{(v)}} \tag{3.5}$$

where $\mathbf{V}$ is the matrix of velocities of the generated transition to which the target key-pose was appended, for every dimension at every timestep. We can therefore calculate our minimum negative **LLV** for the spike-gaussian and the broader-gaussian velocity statistics and define our loss as:

$$l_{vel} = \frac{1}{D}\frac{1}{(T+2)}\sum_{d=0}^{D-1}\sum_{t=0}^{T+1}\min(-\mathbf{LLV}_{d,t}^{(spike)}, -\mathbf{LLV}_{d,t}^{(broad)}),$$

where D is the number of input dimensions.

3.6 Experimental Setup

3.6.1 Data

The data in these experiments comes from three different datasets. The labeled HDM05 dataset and the unlabeled CMU MOCAP dataset are both recorded at 120 frames per second (fps) and contain more than 30 marker positions. In our case, we sub-sample sequences to 30 frames-per-second and use 23 common markers between the two datasets. We work with the C3D file format, which contains series of X, Y, Z positions for each marker, yielding a frame vector of dimension 69. From the NTU dataset, we retrieve the Kinect's skeletal data for each sequences. We use the same approach as Shahroudy *et al.* [52] to determine the main actor(s) of each sequences. We use the positions of each of the 25 joints for each actor when using this dataset. Since some classes in NTU are two-actor-actions, the standard way of representing the sequences is to concatenate data from the two main actors of each sequences, yielding a data representation with 150 degrees of freedom. In cases where only one actor is present, values for the second actor are kept at 0. We do not sub-sample the NTU sequences as they already have a 30 fps rate.

3.6.2 Preprocessing

Our preprocessing of the data for HDM05 and CMU consists mainly of orienting, centering and scaling the point cloud of every frame given by the files. The orientation process is a basis change of all 3D positions so that the actor's hips are always facing the same horizontal direction, while allowing a changing vertical orientation. We then center the hips of the actor at the origin and scale so every marker is always in the interval $[-1, 1]$. This can help handling different actors of different sizes. In all experiments on these two datasets, we use an additive Gaussian noise with a standard deviation of 0.05 and mean 0 on markers' positions for training. We use minibatches of size 4 when handling HDM05 only data, 8 when adding CMU data and 32 when using the NTU dataset. On the NTU dataset, our only preprocessing consists of standardizing each joint to have zero mean and unit variance across all the dataset as suggested by Lecun *et al.* [205].

3.6.3 Network Specifications

We use a model with a frame encoder that is closely related to the one used by [122], as it has two hidden layers of $[1024, 512]$ units. Two extra layers of $[1024, 69]$ units are used by the reconstruction decoder with tied weights with the encoder. The LSTM encoder, has 3 hidden layers of $[512, 512, 256]$ LSTM memory cells. As the output of a single bi-directional recurrent layer can contain, at each timestep, information for the whole sequence, we use bi-directionality only in the first LSTM layer of the sequence encoder. This means that the second layer of the LSTM encoder has an input of size $2 * 512$ containing past and future information. The $\mathbf{c}$ layer, outputting the summary vector is of size 1024, and the $\mathbf{h}^c$ layer is of size 512. A softmax layer is placed on top of $\mathbf{h}^c$ to output action probabilities. Each layer of the LSTM decoder has a number of units equal to the size of the output of its corresponding layer in the encoder. This leads to $[256, 512, 1024]$ memory cells. All non-linear activations used in the network consist of the tanh() function except for the input, output and forget gates of the memory cells that use sigmoid activations. All reconstructive output layers have linear activations. For feed-forward layers' initializations, their weights are drawn uniformly from $[-\sqrt{1/fanin}, \sqrt{1/fanin}]$, while we use orthonormal initialization for recurrent weight matrices. All biases are initialized at 0, except for LSTM forget gates which are initialized to 1, as proposed by [206] and [207].

3.6.4 Training Procedure

We use early stopping on the validation set with a tolerance of 20 epochs for HDM05 and 10 epochs for NTU. The learning rate is initialized to 0.04, and is halved when the validation accuracy is not improved for a number of epochs equal to the half of the tolerance. We optimize the network parameters with momentum-augmented stochastic gradient descent with a 0.9 momentum value.

Even though our networks can model sequences of arbitrary lengths, empirical analysis showed that using overlapping sliding windows with a voting strategy to classify a single sequence yielded better results than feeding these complete sequences as a whole to the network. Windows correspond to sub-sequences of a given width, with a constant offset, that are fed to the network one at a time. We then combine the outputs of the network for these windows in order to compute the final classification. More specifically, the network's softmax outputs for all windows of the sequence are summed together and the action class is determined to be the one with the highest added probabilities. This allows for high-activation segments of a full sequence to have a bigger weight in the final classification vote. Before conducting experiments over variations of the classification models, we tested the network using the FR-SRC model on HDM05 data in order to explore different values of reconstruction weights and different sliding window's widths (number of frames we feed to the encoder). We had $\omega \in \{0, 1, 5, 10, 20, 50, 100\}$, where $\omega = 0$ means there's no reconstruction, and the window's width w $\in \{20, 30, 90, \infty\}$ where ∞ means taking all frames in the sequence. In all other cases, we used an offset of half the width to slide the window. Based on results on our validation set, we found $\omega = 50$ and w= 30 to be most effective. These two hyper-parameters have been fixed to those values for all further experiments.

The SRC architecture without the classification layers is our Sequence Encoder-Decoder architecture used for generating transitions. It has additional future-conditioning weights in the sequence-decoder to compute each LSTM gate and cell activation. In this case, we modify the targets of the reconstruction to be the future frames of transition. The adversarial learning is standard, where the generator and discriminator alternate updates for optimizing Equation 3.5.5. When using our soft constraints, the generator also minimizes Equation 3.5.6 and Equation 3.5.7 to better shape the generated poses.

3.7 Results

3.7.1 Regularizing Classifiers through Reconstruction

The experiments we conducted here used the small HDM05 dataset only and used our proposed held-out actors as a test set. For these first experiments, we focus on demonstrating the regularizing effects of adding different types of reconstructive modules and losses to the network's composite error function. Table 3.2 shows these effects. Each result is the average classification accuracy for the test set of three different runs with the same architecture.

Table 3.2 Regularization effects of different models on accuracieswith HDM05 data only.

Model	**Train(%)**	**Test(%)**
SC	99.61	84.08
SRC	99.62	84.42
FR-SC	99.86	**86.14**
FR-SRC	98.83	85.71
FRC-SRC	98.90	85.89

First, note that our sequence-classifier-only model already outperforms the implemented baselines of 81.64%. More interestingly, we see that all tested variants with optional modules performed better than our sequence-classifier-only. Specifically, adding only the recurrent reconstruction decoder (SRC) improved only marginally the average performance on the test set, while the biggest improvement came from having per-frame decoders (FR-SC) denoising and improving frame representations for the sequence encoder. It seems however, that in this low-data regime, combining the multiple optional decoders (FR-SRC, FRC-SRC) also help generalization compared to the sequence-classification-only model, but to a lesser extent.

3.7.2 Adding Unlabeled CMU Data to Improve HDM05 Classifications

The following experiments, summarized in Table 3.3, compare our results for the movement classification task using HDM05 with and without using additional unlabeled sequences from the CMU dataset. We compare our results with our implementation of the baseline techniques from Cho & Chen [122] and Zhu *et al.* [185] on the same test set. When adding CMU, we performed the same preprocessing as with HDM05 and, during training, augmented each HDM05 minibatch with an equal number of randomly picked CMU sequences for which no classification cost was computed. As with only labeled sequences, the reconstruction losses were averaged over all sequences in the minibatch. Since SC does not use any

Table 3.3 Test accuracy of different models.

Model	**Dataset**	**Test Accuracy (%)**
Baseline	HDM05	81.64
SC	HDM05	84.08
SRC	HDM05	84.42
SRC	HDM05+CMU	84.20
FR-SC	HDM05	86.14
FR-SC	HDM05+CMU	85.71
FR-SRC	HDM05	85.71
FR-SRC	HDM05+CMU	**87.40**
FRC-SRC	HDM05	85.89
FRC-SRC	HDM05+CMU	86.61

reconstruction, no experiment was done with that model with unlabeled data. The results for HDM05-only experiments from last section are repeated here for easier comparisons. All additional experiments are also averages from three runs in the same setting. In this larger data regime, the use of multiple reconstruction decoders with FR-SRC showed to be the most beneficial addition to the system for generalizing on new subjects. However, when only adding the frame-decoder (FR-SC) or the sequence decoder (SRC), it seems the addition of unlabeled sequences from a different distribution of movements did not help as much as with HDM05 data only. Interestingly, all tested versions of the proposed system showed a similar trend of having CMU data boost performance only when having at least the two reconstructive decoders. We hypothesize that when trying to model the data coming from the different distribution of the CMU classes, too much capacity may be spent on trying to solve the harder job of reconstructing CMU poses when all the weighting of the reconstruction is focused on a single reconstruction task (frame or sequence). However, when combining the reconstruction losses by averaging them (see Equation 3.4.6), it seems that reduced focus on single objectives helps the network improve the representations by having more generalizable reconstructive features, without wasting its capacity on a single, harder reconstruction task. As specified in Section 3.6.4, the search over different ω values was done only with the FR-SRC model, which could explain why this architectural variant performs best with this weight value when adding unlabeled data. Also, the frame-based classification module might not be as useful as other modules, which makes sense intuitively. Estimating probabilities of an action based on a single frame without context might be a task overly complex - or simply impossible - for the network. Therefore, the per-frame encoding layers might try to reduce the very high loss on frame-based classification by (often unsuccessfully) producing

discriminative features at the expense of higher other losses, resulting in less useful features to sent the LSTM encoder.

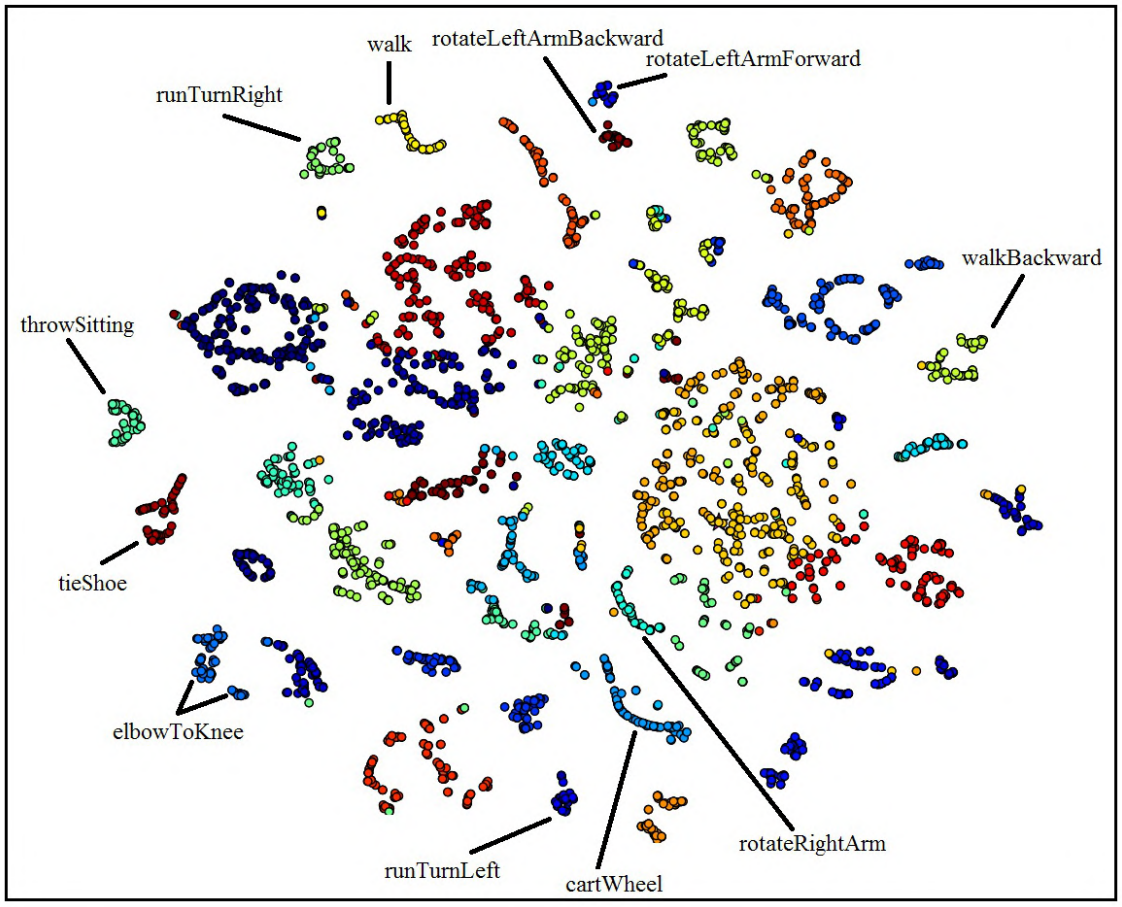
3.7.3 Clustering HDM05

Using the FR-SRC network that yielded the best results on HDM05 classification, we performed clustering on the summary vectors it produced for the test set, unseen during training. We used a Gaussian Mixture Model (GMM) initialized with the K-means++ algorithm, where K was found by using 10% of the set as a validation set to find the best likelihood. This system found 30 clusters that we can visualize in Figure 3.4. Note that feature vectors have 1024 dimensions and clusters were found in that space, while we used the t-SNE algorithm [208] to create a 2D visualization. Some clusters were annotated after manual inspection to give an idea of what movements the network clustered. We can see that such a trained network could help accelerate labeling MOCAP sequences of movements since sequences in the most well defined clusters could be labeled in batch. Manual annotation seems to suggest that HDM05 actions have a considerable impact of the clustered actions, since almost all clusters could be associated with one or two HDM05 labels.

3.7.4 Exploring the Higher Data Regime

To study the effect of our approach in a higher data regime, we evaluate our models on NTU RGB+D [52], a more recent and much larger dataset. It contains 56,880 action sequences (compared to 2329 for HDM05), recorded on 40 different actors and from 3 different camera views. Our results on this dataset are presented in table 3.4.

Our models, trained with the same hyper-parameters as with the other datasets, achieve similar results (slightly better on cross-subject evaluation and slightly worse on cross-views) to those of the Deep LSTM [52] originally proposed as a baseline for NTU. Table 3.4 shows that even though adding frame reconstruction seem to improve our results on cross-subject evaluation, it does the opposite on cross-views, supporting the idea that its effect is mitigated in higher data regimes. Also, it hints at the fact that extra regularization strategies, such as ours, can help to generalize to new actors, but are not required when the test-set is composed of almost identical sequences to the ones found in the training set, as it is the case with the preprocessed NTU sequences in the cross-views evaluation. In other words, additional unsupervised objectives clearly show to be beneficial on HDM05 and CMU datasets, as supported by Table 3.3, but do not provide the same benefit on much larger datasets. We hypothesize that such behavior could emerge from the fact that our multi-decoder approach effectively improves generalization on smaller datasets by acting as a regularization strategy,



(1).jpg (1).jpg (1).jpg (1).jpg

Figure 3.4 **Clustering motion with the FR-SRC network.** 2D visualization of clusters found by the FR-SRC network with some hand-made annotations after inspection of subsequences inside clusters.

as supported by Table 3.2, and that even though action classification on NTU is still a very challenging task, the large number of sequences and the impressive variety of actors included in the training set reduce the need for that kind of regularizer.

3.7.5 Constrained Adversarial Generation

We apply our proposed conditional future generator on the combination of the HDM05 and CMU datasets. In this setup, we use 15 seed frames to produce the past context vector $\mathbf{c}_p$ and generate 30 transition frames that lead to a target frame encoded in $\mathbf{c}_f$. Figure 3.5 shows a sample output with different losses. The improvement due to the addition of physics-based soft constraints is easily noticeable as without them the generator never learns to produce smooth transitions and often exhibits physically implausible artifacts, such

Table 3.4 Performance in terms of accuracy on the NTU RGB+D dataset.

Model	**CS**	**CV**
Skeletal Quads [118]	38.6	41.4
Lie Group [124]	50.1	52.8
FTP Dynamic Skeletons [209]	60.2	65.2
HBDRNN [123]	59.1	64.0
Deep RNN [52]	56.3	64.1
Deep LSTM [52]	60.7	67.3
Part-aware LSTM [52]	62.9	70.3
ST-LSTM [189]	69.2	77.7
STA-LSTM [197]	73.4	81.2
CNN-MTLN [121]	**79.6**	**84.8**
SC	62.5	65.88
FR-SC	63.4	65.60

as stretched bones or jumps between frames. Since differences between samples from the reconstructive and constrained adversarial generators (for example, small reductions of foot sliding) can be difficult to identify on still frames, the reader is encouraged to watch the videos in the supplementary materials. The effects of adding our data-driven soft constraints during training are also depicted in Figure 3.6, where MSE curves are shown for an adversarial training procedure. Even though we do not directly optimize to minimize the MSE, Figures 3.5 and 3.6 together show that over a certain MSE threshold, the generated transitions are neither realistic nor smooth. We further quantify the effects of both our proposed constraints (l_{bone} and l_{vel}) in an ablation study reported in Table 3.5. In summary, the main contribution

Table 3.5 Ablation study for the soft-constraints with adversarial training.

Model	**MSE**
ReGAN	0.355
ReGAN + l_{bone}	0.163
ReGAN + l_{vel}	0.137
ReGAN + l_{vel} + l_{bone}	**0.116**

of these exploratory experiments is the significant improvement of stability for recurrent adversarial learning brought by the soft constraints based on training-data statistics that do not impose a deterministic and unique path given a past and a future context. This can be used in other settings as it is a good way to enforce realism without diminishing the ability of GANs to produce realistic samples untied to a single path. It also significantly makes the training of the combination of LSTM and GANs much more stable.

Figure 3.5 **Comparison of different generated *Walk-Turn* transitions** with the ground truth. Differences between the generated transitions are easier to observe in the supplementary video.

3.8 Conclusion

Recurrent Encoder-Decoder architectures with multiple decoders provide an attractive framework for semi-supervised, multi-purpose representation learning. Our experiments show that the explored architectures outperform our implementations of the state-of-the-art for HDM05 movement classification methods with a realistic actor-based partition of data. We hope this evaluation setup can serve as a benchmark partition for further HDM05 experiments, which is still a challenging dataset due to its high number of action classes and low actor count. Our results also indicate that the inclusion of reconstructive decoders can have a regularizing effect on learning and allow the use of unlabeled data in order to improve generalization. Additionally, we have seen that such networks are well suited for clustering as learned representations compress both reconstructive and discriminative information about sequences. Clusters therefore tend to correspond to actions that resemble labels and could therefore accelerate further labeling.

Our experiments on NTU RGB+D dataset, which contains both many more sequences and more actors, show the limited beneficial aspects of our unsupervised-decoders on larger datasets. This indicates that our approach is most useful when working with a limited amount of labeled data, which is still quite common for real-life applications.

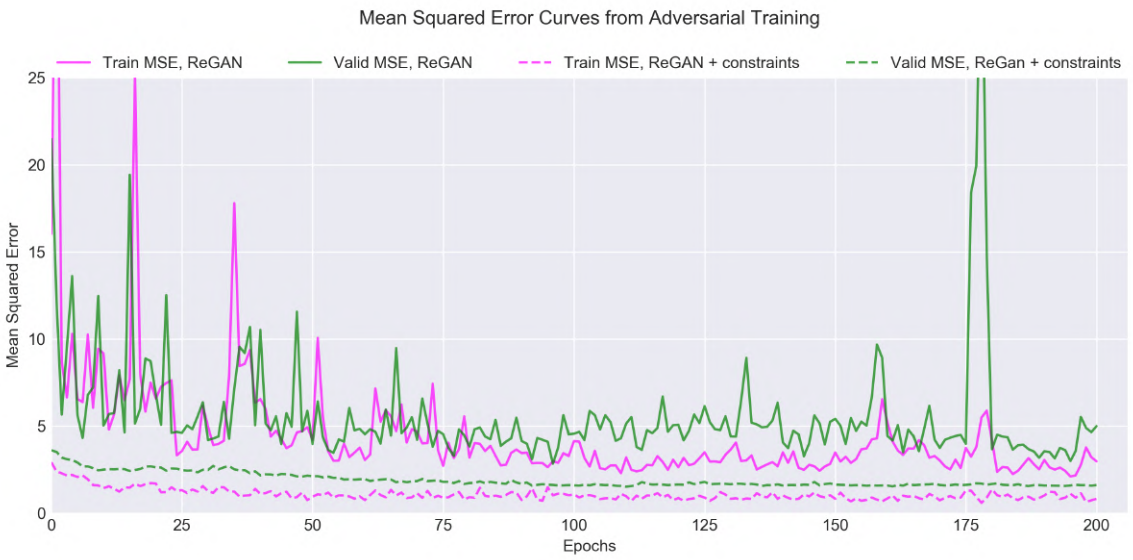


Figure 3.6 **Mean squared error (MSE) curves comparison from adversarial training**, with and without constraints. Our added soft constraints stabilize and improve performance throughout training.

Finally we have also explored a novel constrained recurrent adversarial animation transition generator which can produce plausible continuous skeletal trajectories. Both LSTMs and GANs can be hard to train, but we observed clear benefits from the addition of soft constraints with adversarial training and believe this points to promising directions for realistic animation synthesis and continuous, variable length trajectory generation. The idea of defining data-driven soft constraint could be applied to other temporal domains where GANs are especially hard to train.

Acknowledgements

We thank Ubisoft and the Natural Sciences and Engineering Research Council of Canada for support under the Collaborative Research and Development program and the Discovery Grant program. We also thank the Mitacs Accelerate (IT08585) program for support. Finally, we want to thank the authors of the Theano framework [210].

CHAPTER 4 RECURRENT TRANSITION NETWORKS FOR CHARACTER LOCOMOTION

Abstract

We present a novel approach, based on deep recurrent neural networks, to automatically generate transition animations given a past context of a few frames, a target character state and optionally local terrain information. The proposed Recurrent Transition Network (RTN) is trained without any gait, phase, contact or action labels. Our system produces realistic and fluid transitions that rival the quality of Motion Capture-based animations, even without any inverse-kinematics post-process. Our system could accelerate the creation of transition variations for large coverage or even replace transition nodes in a game's animation graph. The RTN also shows impressive results on a temporal super-resolution task.

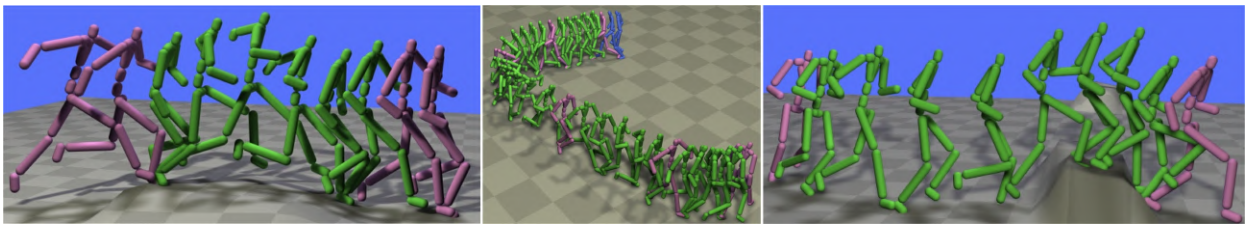


Figure 4.1 **Transition generation results.** *Left* : A 1-second transition animation (green) generated with our proposed system, from past and future contexts (magenta). *Center* : Super-resolution results (green) from a input motion (blue) saved along one target state (magenta) per second. *Right* : a 2-second transition with obstacle awareness. Results are best seen in the accompanying video[2].

4.1 Introduction

Large games often require large animation graphs with a huge amount of unique animations in order to provide realistic and responsive movements. As the number of states grows in the graph, the number of required transitions can grow exponentially, while each of these transitions may require several variations to allow different conditions to be handled. We explore in this research paper a data-driven approach based on deep Recurrent Neural Networks (RNN) to automatically generate transition clips from any character motion in order to reach a desired target, defined as the future desired state of the character. Such an approach

[2]http://y2u.be/lXd_7X-DkTA

aims at leveraging high quality data from Motion Capture (MOCAP) based animations, in order to greatly simplify the task of transition generation for games. We present the Recurrent Transition Network (RTN), designed for transition generation by conditioning on a *past context* of input frames and a target-relative representation called *future context* that is evolving through time. During training, we pick such past and future contexts directly from the data, without any labeling of gait, phase or other information. We further explore additional constraint-conditioning by including local terrain information to the inputs of the network.

4.2 Related Work

4.2.1 Motion Control

Our task is related to motion control, where goals can be represented as key-frames, or as higher-level constraints, such as trajectories or footstep plans. Important work on control and transitions was based on motion graphs [78, 81, 127, 129, 130] but these approaches show limited scalability and generality, as they require the data to be in memory and are limited to naive interpolation of motions from the dataset. Maximum A Posteriori (MAP) frameworks [140, 141], Gaussian Processes (GP) [147] or Gaussian Process Latent Variable Models [142–146] address the limitations of graph-based methods, but have computation times that scale with the data. Many of these are thus applied to single, specific actions. Neural networks have shown impressive results on generic motion synthesis [163] and online control [68, 74] but require explicit disambiguation strategies sometimes based on data annotation. Reinforcement Learning (RL) based methods using clips [157, 158] or states [134] as actions have been proposed, but share limitations of graph-based methods, while Levine *et al.* [146], operating in the the low-dimensional latent space learned from their GPLVM share limitations of other GP-based methods. Physically-based deep RL techniques [172, 173, 211–213] model well interactions with the world, but are limited to specific skills as learning global policies is hard.

4.2.2 Motion Prediction with RNNs

Recently, many RNN-based approaches have been proposed for motion prediction from a past-context of several frames. Fragkiadaki *et al.* [33], Jain *et al.* [87], Martinez *et al.* [32], Li *et al.* [65] and Ghosh *et al.* [88] all build on LSTM networks for successful motion generation. In this work, we augment the Encoder-Recurrent-Decoder (ERD) networks of Fragkiadaki *et al.* [33] combined with residual connections [32] with future and terrain awareness in order

to produce realistic-looking transitions.

4.3 Data formatting

4.3.1 Dataset

The data for this experiment was captured in a MOCAP studio using a Vicon system. It contains a lot of unstructured motion data summarized in Table 4.1, all re-targeted to a common skeleton from which we use $K = 22$ bones, and downsampled at a rate of 30 fps. The Others category contains diverse miscellaneous motions, including dance, sports and

Table 4.1 Dataset motion categories

Motion	Frames (at 30 fps)	Minutes
Flat Locomotion	240 776	133
Terrain Locomotion	113 020	63
Dance	38 916	22
Others	115 716	64
Total	508 428	282

fighting motions. In all of our experiments except for those detailed in Section 4.6.4, we used only the terrain locomotion data, corresponding to approximately one hour of MOCAP. We split the dataset into N overlapping sub-sequences $\mathbf{Y}_n$, $n \in \{0, ..., N-1\}$ that have a length L determined by our desired transition length. As we use a past context of 10 frames in all of our experiments, and 2 frames are needed to produce our target state, we have a necessary $L \geq 10 + P + 2$, where P is our desired transition length. We usually add 8 extra frames in the future context for visualization purposes only. Unless specified otherwise, we use $P = 30$ in our experiments.

4.3.2 Input sequences

Similarly to [163], we work with positional information. The raw data consists of sequences $\mathbf{Y}_n = \{\mathbf{y}_0, ..., \mathbf{y}_{L-1}\}_n$ of vectors $\mathbf{y}_t$ of global 3D positions. The dimensionality of all positional or velocity vectors is of $D = 3 \times K$ where $K = 22$ is the number of bones used. Our preprocessed vectors $\mathbf{x}_t$ extracted from $\mathbf{y}_t$ consist of a concatenation of the global root velocity with the root-relative positions of all other joints. These vectors are standardized with the mean and standard deviation of each dimensions taken from the training set.

4.3.3 Future context

The future context is used as conditioning information at every timestep and consists of the concatenation of two different vectors. The first one is the target vector $\mathbf{t} \in \mathbb{R}^{2*D}$ which is the preprocessed target pose concatenated with the normalized velocity of all joints on that frame. It is constant throughout the transition. The second vector is the global offset vector $\mathbf{o}_t \in \mathbb{R}^D$ which is composed of the euclidean distances of each joints from the target pose in global space.These vectors are also standardized with training set statistics.

4.3.4 Terrain

In most of our experiments, we also make use of local terrain information as an additional guiding signal. Similarly to [172, 213], we use a local heightmap relative to the (x, z) root position. We augment each motion $\mathbf{Y}_n$ of the dataset with 5 plausible terrains $\mathbf{H}_n^l, l \in \{0, ..., 4\}$ that we pick at random during training. We follow the terrain fitting and editing procedure proposed by Holden *et al.* [68] with the same heightmap dataset and fit those terrains. We further process the fitted terrains around the contact points with a normalized 2D 33×33 isotropic Gaussian filter $\mathbf{F}$ spanning 1.3×1.3 meters with a standard deviation of 20 cm in order to smooth out artifacts from the editing function. We then sample at each timestep a 13×13 grid spanning 2.06×2.06 meters centered on the character. We then convert this local heightmap into a grid of y-offsets from the root joint. We use the standardized, 169-dimensional flattened vector $\mathbf{p}_t$ of this grid as our representation for the local patch of terrain.

4.3.5 Random orientation

During training, we randomly rotate each motion sequence $\mathbf{Y}_n$ as well as the terrain around a unit *up*-vector emerging from the center of the terrain with an angle drawn uniformly in $[-\pi, \pi]$ to induce rotation invariance in the system.

4.4 Recurrent Transition Network

4.4.1 System overview

On a given timestep, our system takes as inputs a preprocessed positional vector $\mathbf{x}_t$ retrieved from the corresponding global positional vector $\mathbf{y}_t$ and a local terrain patch representation $\mathbf{p}_t$ relative to $\mathbf{y}_t$, retrieved from a heightmap $\mathbf{H}$. It uses as conditioning information a normalized target vector $\mathbf{t}$ computed from the global positions on the target frame $\mathbf{y}_T$ and the next one

$\mathbf{y}_{T+1}$, and a global-offset vector $\mathbf{o}_t$ retrieved from $\mathbf{y}_t$ and $\mathbf{y}_T$. The RTN is composed of several, specialized sub-networks that we describe here.

4.4.2 Frame encoder

At each timestep, the preprocessed current character configuration $\mathbf{x}_t$ and local patch $\mathbf{p}_t$, are first transformed into a new hidden representation $\mathbf{h}_t^{\mathcal{E}}$ by the frame encoder $\mathcal{E}()$, which consists of a Multi-Layer Perceptron (MLP) with two hidden layers of 512 units:

$$\mathbf{h}_t^{\mathcal{E}} = \mathcal{E}(\mathbf{x}_t) = \phi\Big(\mathrm{W}_{\mathcal{E}}^{(2)}\phi(\mathrm{W}_{\mathcal{E}}^{(1)}\begin{bmatrix}\mathbf{x}_t\\ \mathbf{p}_t\end{bmatrix} + \mathbf{b}_{\mathcal{E}}^{(1)}) + \mathbf{b}_{\mathcal{E}}^{(2)}\Big) \tag{4.1}$$

where $\mathrm{W}_{\mathcal{E}}^{(l)}$ and $\mathbf{b}_{\mathcal{E}}^{(l)}$ are the weight matrices and bias vectors of the l^{th} layer, and ϕ is the Leaky Rectified Linear Unit (LReLU) activation function.

4.4.3 Future context encoders

The future, normalized target $\mathbf{t}$ encoder $\mathcal{F}()$ and the global offset $\mathbf{o}_t$ encoder $\mathcal{O}()$ are both MLPs with two 128-unit layers:

$$\mathbf{h}^{\mathcal{F}} = \mathcal{F}(\mathbf{t}) = \phi\Big(\mathrm{W}_{\mathcal{F}}^{(2)}\phi(\mathrm{W}_{\mathcal{F}}^{(1)}\mathbf{t} + \mathbf{b}_{\mathcal{F}}^{(1)}) + \mathbf{b}_{\mathcal{F}}^{(2)}\Big) \tag{4.2}$$

$$\mathbf{h}_t^{\mathcal{O}} = \mathcal{O}(\mathbf{o}_t) = \phi\Big(\mathrm{W}_{\mathcal{O}}^{(2)}\phi(\mathrm{W}_{\mathcal{O}}^{(1)}\mathbf{o}_t + \mathbf{b}_{\mathcal{O}}^{(1)}) + \mathbf{b}_{\mathcal{O}}^{(2)}\Big) \tag{4.3}$$

The future target $\mathbf{t}$ is encoded once and is constant for every timestep of a given sequence.

4.4.4 Recurrent generator

The recurrent generator $\mathcal{R}$ is responsible for the temporal dynamics modeling and the conditioning on the future context. It is a single LSTM 512-units layer that uses the concatenation $\mathbf{f}_t$ of $\mathbf{h}^{\mathcal{F}}$ and $\mathbf{h}_t^{\mathcal{O}}$ as conditioning information by using this data for the computation of the

gates and cell values:

$$\mathbf{i}_t = \alpha(\mathrm{W}^{(i)}\mathbf{h}_t^{\mathcal{E}} + \mathrm{U}^{(i)}\mathbf{h}_{t-1}^{\mathcal{R}} + \mathbf{C}^{(i)}\mathbf{f}_t + \mathbf{b}^{(i)}) \tag{4.4}$$

$$\mathbf{o}_t = \alpha(\mathrm{W}^{(o)}\mathbf{h}_t^{\mathcal{E}} + \mathrm{U}^{(o)}\mathbf{h}_{t-1}^{\mathcal{R}} + \mathbf{C}^{(o)}\mathbf{f}_t + \mathbf{b}^{(o)}) \tag{4.5}$$

$$\mathbf{f}_t = \alpha(\mathrm{W}^{(f)}\mathbf{h}_t^{\mathcal{E}} + \mathrm{U}^{(f)}\mathbf{h}_{t-1}^{\mathcal{R}} + \mathbf{C}^{(f)}\mathbf{f}_t + \mathbf{b}^{(f)}) \tag{4.6}$$

$$\hat{\mathbf{c}}_t = \mathrm{W}^{(c)}\mathbf{h}_t^{\mathcal{E}} + \mathrm{W}^{(c)}\mathbf{h}_{t-1}^{\mathcal{R}} + \mathbf{C}^{(c)}\mathbf{f}_t + \mathbf{b}^{(c)} \tag{4.7}$$

$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tau(\hat{\mathbf{c}}_t) \tag{4.8}$$

$$\mathbf{h}_t^{\mathcal{R}} = \mathcal{R}(\mathbf{h}_t^{\mathcal{E}}, \mathbf{h}_{t-1}^{\mathcal{R}}, \mathbf{c}_t, \mathbf{h}_t^{\mathcal{O}}, \mathbf{h}^{\mathcal{F}}) = \mathbf{o}_t \odot \tau(\mathbf{c}_t) \tag{4.9}$$

where $\mathrm{W}^{(\{i,o,f,c\})}$, $\mathrm{U}^{(\{i,o,f,c\})}$, $\mathbf{C}^{(\{i,o,f,c\})}$ are feed-forward, recurrent, and conditioning weight matrices respectively, $\mathbf{b}^{(\{i,o,f,c\})}$ are bias vectors, and $\odot$ is an element-wise multiplication. The α and τ functions are the commonly used *sigmoid* and *tanh* non-linearities of LSTM networks.

4.4.5 Frame decoder

Each of the generated output $\mathbf{h}_t^{\mathcal{R}}$ of the LSTM generator is passed to the frame decoder $\mathcal{D}$, which is another MLP which has three layers of 256, 128 and D units respectively:

$$\mathbf{h}_t^{\mathcal{D}} = \mathcal{D}(\mathbf{h}_t^{\mathcal{R}}) = \mathrm{W}_{\mathcal{D}}^{(3)}\phi\Big(\mathrm{W}_{\mathcal{D}}^{(2)}\phi(\mathrm{W}_{\mathcal{D}}^{(1)}\mathbf{h}_t^{\mathcal{R}} + \mathbf{b}_{\mathcal{D}}^{(1)}) + \mathbf{b}_{\mathcal{D}}^{(2)}\Big) + \mathbf{b}_{\mathcal{D}}^{(3)} \tag{4.10}$$

We use a Res-Net LSTM [32], which outputs an offset from the current frame $\mathbf{x}_t$, to reduce the gap between the input seed frames and the beginning of the transition. The final prediction $\hat{\mathbf{x}}_{t+1}$ is therefore obtained with:

$$\hat{\mathbf{x}}_{t+1} = \mathbf{x}_t + \mathbf{h}_t^{\mathcal{D}} \tag{4.11}$$

From $\hat{\mathbf{x}}_{t+1}$, we can retrieve $\hat{\mathbf{y}}_{t+1}$ by applying the inverse of our preprocessing function, which consists of de-normalizing the vector, retrieving the global root positions by cumulatively adding the velocities to the stored first global frame of the sequence, and then adding those root positions to the other joints.

4.4.6 Hidden state initializer

In an effort to go beyond normal LSTM hidden state initializing strategies that provide the same initial hidden state for all sequences, we use an additional MLP that learns to predicts

$\mathbf{h}_{-1}$ given the first frame of the input sequence $\mathbf{x}_0$:

$$\mathbf{h}_{-1} = \mathcal{H}(\mathbf{x}_0) = \mathrm{W}_{\mathcal{H}}^{(2)}\phi(\mathrm{W}_{\mathcal{H}}^{(1)}\mathbf{x}_0 + \mathbf{b}_{\mathcal{H}}^{(1)}) + \mathbf{b}_{\mathcal{H}}^{(2)} \tag{4.12}$$

This method allows for the recurrent hidden states to be initialized differently for any given input into a good region of the hidden space and doesn't require a change to the loss function.

4.4.7 Postprocessing

The only postprocessing we perform is a correction of the gap between the last generated frame and the target position. We refer to this naive postprocessing as *target blend*, which is a linear blending of the offset from the generated target pose from the true target into the generated transition.

4.5 Training

The RTN is trained with the Mean Squared Error (MSE) signal computed on the generated transition and generated target position. We use the AMSGrad [214] optimizer with a learning rate of 0.0005 and minibatches of 32. We use probabilistic teacher forcing [215] with with a fixed probability $p = 0.2$ of feeding the network the true last pose instead of its own prediction during training. For the terrain locomotion dataset, we have 4 611 training and 1 008 validation samples, while we have 20 789 and 4 468 training and validation samples in the whole dataset. Validation samples are come from extracting all sequences from single actor from the dataset. We run the training for 200 epochs in all experiments and show results on the best performing network iteration on the validation set.

4.6 Results

4.6.1 Transition Reconstruction

We first train our network to generate transitions of 30 frames on the rough terrain locomotion dataset. Figure 1 (left) shows a transition during a forward leap. The RTN does not suffer from diverging trajectories or collapsing to the average pose. In most cases, it is hard for external viewers to determine if a given transition was generated or coming from the data. The system successfully models uneven ground locomotion such as walks, runs, jumps, turning motions, changes in velocities and stances, all with a wide range of gaits. The network uses under 15MB of memory. In Table 4.2, we compare our model with ERD

networks and Res-Net LSTMs that we augment with our future-awareness strategy (F-ERD, F-ResLSTM) as they are the most related and competitive baselines to our approach. We also compare to a naive quaternion interpolation strategy (INT). We further provide a measure of Average Centimeter Offset (ACO), consisting of the absolute centimeter offset from the ground truth of the generated transition, averaged over all degrees of freedom and over time.

Table 4.2 Comparison of methods on the MSE on the validation set.

Architecture	MSE	ACO
INT	0.210	6.726
F-ResLSTM	0.144	7.709
F-ERD	0.092	4.770
RTN	**0.087**	**4.751**

4.6.2 Impact of terrain awareness

Interestingly, our experiments show that adding terrain conditioning is not necessary on one-second transitions, as within that length, terrain information seems to be well inferred by the past context and targets. On longer transitions, however, some obstacles cannot be inferred by the input contexts, such as in Figure 4 (right) and adding terrain-awareness allows the character to modify its behavior with respect to obstacles.

4.6.3 Ablation study

In order to assess the benefits of the different modifications we bring to the ERD network, we performed an ablation study summarized in Table 4.3. The hcommon and h0 modifications correspond to using a common, learned initial hidden state and a null initial state for the LSTM layer respectively. The resnet and future modifications indicates that the RTN is not using a ResNet formulation or future information respectively. The ptf modifications relate to the probability of the teacher forcing sampling, where the number corresponds to the probability of using the previous true frame, while the Δ symbol corresponds to a scheduled sampling strategy, where p linearly decreases from 1 to 0 during training.

4.6.4 Temporal Super-Resolution

We use our system to decompress animation saved with only 10 frames of context and a single target state (pose + velocity) per second. The RTN then performs temporal super-resolution

Table 4.3 Ablation study on the RTN network.

Variant	MSE
RTN\future	0.298
RTN\ptf=Δ	0.151
RTN\ptf=1.0	0.147
RTN\ptf=0.0	0.109
RTN\resnet	0.114
RTN\h0	0.095
RTN\hcommon	0.092
RTN	**0.087**

of the animation by predicting transitions in series and taking its last 10 generated frames (with target blend) as past context to reach the next target. The network then performs very high quality lossy temporal decompression, by generating fluid and plausible sequences. We show such a decompressed sequence in Figure 1 (center) and in the accompanying video.

4.7 Limitations and Future Work

Our system, like many deep learning methods, is heavily dependent on the quality and amount of training data and does not model uncertainty well. A limitation specific to our system emerges from the standardization of the global offset vector $\mathbf{o}_t$ that effectively limits the ability of the network to generate transitions of lengths significantly longer than those on which the statistics were computed. Conditioning techniques could be explored in order to remove this upper bound. Bi-directional synthesis methods should also be considered in future research in order to remove the need for the target blending.

CHAPTER 5 ROBUST MOTION IN-BETWEENING

Abstract

In this work we present a novel, robust transition generation technique that can serve as a new tool for 3D animators, based on adversarial recurrent neural networks. The system synthesises high-quality motions that use temporally-sparse keyframes as animation constraints. This is reminiscent of the job of in-betweening in traditional animation pipelines, in which an animator draws motion frames between provided keyframes. We first show that a state-of-the-art motion prediction model cannot be easily converted into a robust transition generator when only adding conditioning information about future keyframes. To solve this problem, we then propose two novel additive embedding modifiers that are applied at each timestep to latent representations encoded inside the network's architecture. One modifier is a time-to-arrival embedding that allows variations of the transition length with a single model. The other is a scheduled target noise vector that allows the system to be robust to target distortions and to sample different transitions given fixed keyframes. To qualitatively evaluate our method, we present a custom MotionBuilder plugin that uses our trained model to perform in-betweening in production scenarios. To quantitatively evaluate performance on transitions and generalizations to longer time horizons, we present well-defined in-betweening benchmarks on a subset of the widely used Human3.6M dataset and on LaFAN1, a novel high quality motion capture dataset that is more appropriate for transition generation. We are releasing this new dataset along with this work, with accompanying code for reproducing our baseline results.

5.1 Introduction

Human motion is inherently complex and stochastic for long-term horizons. This is why Motion Capture (MOCAP) technologies still often surpass generative modeling or traditional animation techniques for 3D characters with many degrees of freedom. However, in modern video games, the number of motion clips needed to properly animate a complex character with rich behaviors is often very large and manually authoring animation sequences with keyframes or using a MOCAP pipeline are highly time-consuming processes. Some methods to improve curve fitting between keyframes [216] or to accelerate the MOCAP workflow [72] have been proposed to improve these processes. On another front, many auto-regressive deep

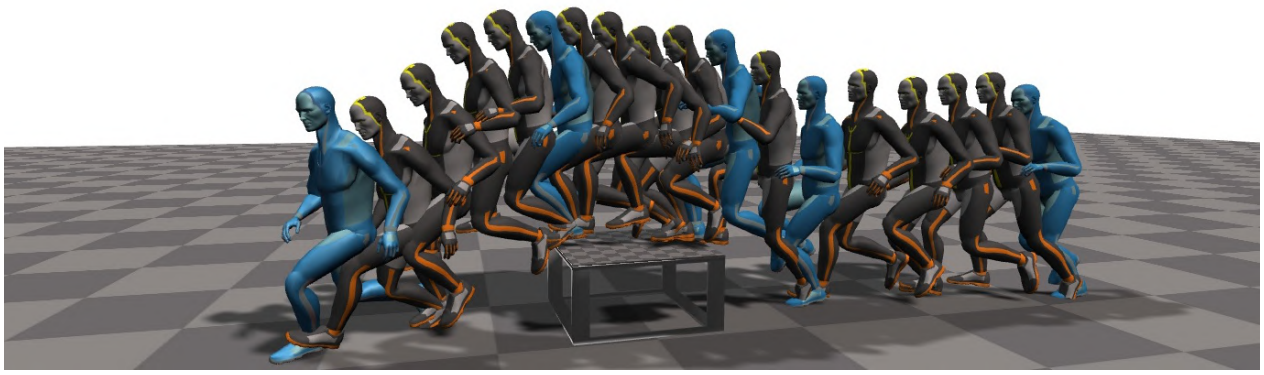


Figure 5.1 **Transitions automatically generated by our system**. Target keyframes are blue. For clarity, only one in four generated frames is shown. Our tool allows for generating transitions of variable lengths and for sampling different variations of motion given fixed keyframes.

learning methods that leverage high quality MOCAP for motion prediction have recently been proposed [32, 33, 53–55, 85, 87]. Inspired by these achievements, we build in this work a transition generation tool that leverages the power of Recurrent Neural Networks (RNN) as powerful motion predictors to go beyond keyframe interpolation techniques, which have limited expressiveness and applicability.

We start by building a state-of-the-art motion predictor based on several recent advances on modeling human motion with RNNs [33,53,55]. Using a recently proposed target-conditioning strategy [217], we convert this unconstrained predictor into a transition generator, and expose the limitations of such a conditioning strategy. These limitations include poor handling of transitions of different lengths for a single model, and the inherent determinism of the architectures. The goal of this work is to tackle such problems in order to present a new architecture that is usable in a production environment.

To do so, we propose two different additive modifiers applied to some of the latent representations encoded by the network. The first one is a *time-to-arrival embedding* applied on the hidden representation of all inputs. This temporal embedding is similar to the positional encoding used in transformer networks [35] in natural language modeling, but serves here a different role. In our case, these embeddings evolve backwards in time from the target frame in order to allow the recurrent layer to have a continuous, dense representation of the number of timesteps remaining before the target keyframe must be reached. This proves to be essential to remove artifacts such as gaps or stalling at the end of transitions. The second embedding modifier is an additive *scheduled target noise* vector that forces the recurrent

layer to receive distorted target embeddings at the beginning of long transitions. The scheduled scaling reduces the norm of the noise during the synthesis in order to reach the correct keyframe. This forces the generator to be robust to noisy target embeddings. We show that it can also be used to enforce stochasticity in the generated transitions more efficiently than another noise-based method. We then further increase the quality of the generated transitions by operating in the Generative Adversarial Network (GAN) framework with two simple discriminators applied on different timescales.

This results in a temporally-aware, stochastic, adversarial architecture able to generate missing motions of variable length between sparse keyframes of animation. The network takes 10 frames of past context and a single target keyframe as inputs and produces a smooth motion that leads to the target, on time. It allows for cyclic and acyclic motions alike and can therefore help generate high-quality animations from sparser keyframes than what is usually allowed by curve-fitting techniques. Our model can fill gaps of an arbitrary number of frames under a soft upper-bound and we show that the particular form of temporal awareness we use is key to achieve this without needing any smoothing post-process. The resulting system allows us to perform robust, automatic in-betweening, or can be used to stitch different pieces of existing motions when blending is impossible or yields poor quality motion.

Our system is tested in production scenarios by integrating a trained network in a custom plugin for Autodesk's MotionBuilder, a popular animation software, where it is used to greatly accelerate prototyping and authoring new animations. In order to also quantitatively assess the performance of different methods on the transition generation task, we present the LaFAN1 dataset, a novel collection of high quality MOCAP sequences that is well-suited for transition generation. We define in-betweening benchmarks on this new dataset as well as on a subset of Human3.6M, commonly used in the motion prediction literature. Our procedure stays close to the common evaluation scheme used in many prediction papers and defined by Jain *et al.* [87], but differs on some important aspects. First, we provide error metrics that take into consideration the global root transformation of the skeleton, which provides a better assessment of the absolute motion of the character in the world. This is mandatory in order to produce and evaluate valid transitions. Second, we train and evaluate the models in an action-agnostic fashion and report average errors on a large evaluation set, as opposed to the commonly used 8 sequences per action. We further report generalization results for transitions that are longer than those seen during training. Finally, we also report the Normalized Power Spectrum Similarity (NPSS) measure for all evaluations, as suggested by Gopalakrishnan *et al.* [54] which reportedly correlates better with human perception of quality.

Our main contributions can thus be summarized as follow:

- Latent additive modifiers to convert state-of-the-art motion predictors into robust transition generators:
  - A *time-to-arrival embedding* allowing robustness to varying transition lengths,
  - A *scheduled target-noise* vector allowing variations in generated transitions,
- New in-betweening benchmarks that take into account global displacements and generalization to longer sequences,
- LaFAN1, a novel high quality motion dataset well-suited for motion prediction that we make publicly available with accompanying code for reproducing our baseline results[1].

5.2 Related Work

5.2.1 Motion Control

We refer to motion control here as scenarios in which temporally-dense external signals, usually user-defined, are used to drive the generation of an animation. Even if the main application of the present work is not focused on online control, many works on motion control stay relevant to this research. Motion graphs [78, 81, 127, 129] allow one to produce motions by traversing nodes and edges that map to character states or motions segments from a dataset. Safonova and Hodgins [130] combine an interpolated motion graph to an anytime A^* search algorithm in order produce transitions that respect some constraints. Motion matching [133] is another search driven motion control technique, where the current character pose and trajectory are matched to segments of animation in a large dataset. Chai & Hodgins, and Tautges *et al.* [131,132] rely on learning local PCA models on pose candidates from a motion dataset given low-dimensional control signals and previously synthesized poses in order to generate the next motion frame. All these techniques require a motion database to be loaded in memory or in the latter cases to perform searches and learning at run-time, limiting their scalability compared to generative models.

Many machine learning techniques can mitigate these requirements. Important work has used the *Maximum A Posteriori* (MAP) framework where a motion prior is used to regularize constraint(s)-related objectives to generate motion. [140] use a statistical dynamics model as a motion prior and user constraints, such as keyframes, to generate motion. Min *et al.* [141] use deformable motion models and optimize the deformable parameters at run-time given the

[1] https://github.com/ubisoftinc/Ubisoft-LaForge-Animation-Dataset

MAP framework. Other statistical models, such as Gaussian Processes [147] and Gaussian Process Latent Variable Models [143–146] have been applied to the constrained motion control task, but are often limited by heavy run-time computations and memory requirements that still scale with the size of the motion database. As a result, these are often applied to separate types of motions and combined together with some post-process, limiting the expressiveness of the systems.

Deep neural networks can circumvent these limitations by allowing huge, heterogeneous datasets to be used for training, while having a fixed computation budget at run-time. Holden *et al.* [162, 163] use feed-forward convolutional neural networks to build a constrained animation synthesis framework that uses root trajectory or end-effectors' positions as control signals. Online control from a gamepad has also been tackled with phase-aware [68], mode-aware [74] and action-aware [82] neural networks that can automatically choose a mixture of network weights at run-time to disambiguate possible motions. Recurrent Neural Networks (RNNs) on the other hand keep an internal memory state at each timestep that allows them to perform naturally such disambiguation, and are very well suited for modeling time series. Lee *et al.* [168] train an RNN for interactive control using multiple control signals. These approaches [68, 74, 163, 168] rely on spatially or temporally dense signals to constrain the motion and thus reduce ambiguity. In our system, a character might have to precisely reach a temporally distant keyframe without any dense spatial or temporal information provided by the user during the transition. The spatial ambiguity is mostly alleviated by the RNN's memory and the target-conditioning, while the timing ambiguity is resolved in our case by time-to-arrival embeddings added to the RNN inputs. Remaining ambiguity can be alleviated with generative adversarial training [43], in which the motion generator learns to fool an additional discriminator network that tries to differentiate generated sequences from real sequences. Barsoum *et al.* [85] and Gui *et al.* [164] both design new loss functions for human motion prediction, while also using adversarial losses using different types of discriminators. These losses help reduce artifacts that may be produced by generators that average different modes of the plausible motions' distribution.

Motion control has also been addressed with Reinforcement Learning (RL) approaches, in which the problem is framed as a Markov Decision Process where *actions* can correspond to actual motion clips [157,158] or character states [134], but again requiring the motion dataset to be loaded in memory at run-time. Physically-based control gets rid of this limitation by having the output of the system operate on a physically-driven character. Coros *et al.* [159] employ fitted value iteration with actions corresponding to optimized Proportional-Derivative (PD) controllers proposed by Yin *et al.* [80]. These RL methods operate on value functions that have discrete domains, which do not represent the continuous nature of motion and

impose run-time estimations through interpolation.

Deep RL methods, which use neural networks as powerful continuous function approximators have recently started being used to address these limitations. Peng *et al.* [172] apply a hierarchical actor-critic algorithm that outputs desired joint angles for PD-controllers. Their approach is applied on a simplified skeleton and does not express human-like quality of movement despite their style constraints. Imitation-learning based RL approaches [211, 212] try to address this with adversarial learning, while others tackle the problem by penalizing distance of a generated state from a reference state [84, 213]. Actions as animation clips, or control fragments [173] can also be used in a deep-RL framework with Q-learning to drive physically-based characters. These methods show impressive results for characters having physical interactions with the world, while still being limited to specific skills or short cyclic motions. We operate in our case in the kinematics domain and train on significantly more heterogeneous motions.

5.2.2 Motion Prediction

We limit here the definition of motion prediction to generating unconstrained motion continuation given single or multiple frames of animation as context. This task implies learning a powerful motion dynamics model which is useful for transition generation. Neural networks have shown over the years to excel in such representation learning. Early work from Taylor *et al.* [139] using Conditional Restricted Boltzmann Machines showed promising results on motion generation by sampling at each timestep the next frame of motion conditioned on the current hidden state and n previous frames. More recently, many RNN-based approaches have been proposed for motion prediction from a past-context of several frames, motivated by the representational power of RNNs for temporal dynamics. Fragkiadki *et al.* [33] propose to separate spatial encoding and decoding from the temporal dependencies modeling with the Encoder-Recurrent-Decoder (ERD) networks, while Jain *et al.* [87] apply structural RNNs to model human motion sequences represented as spatio-temporal graphs. Other recent approaches [32, 53–55, 89, 165] investigate new architectures and loss functions to further improve short-term and long-term prediction of human motion. Others [65, 88] investigate ways to prevent divergence or collapsing to the average pose for long-term predictions with RNNs. In this work, we start by building a powerful motion predictor based on the state-of-the-art recurrent architecture for long-term prediction proposed by Chiu *et al.* [55]. We combine this architecture with the feed-forward encoders of Harvey *et al.* [217] applied to different parts of the input to allow our embedding modifiers to be applied on distinct parts of the inputs. In our case, we operate on joint-local quaternions for all bones, except for the

root, for which we use quaternions and translations local to the last seed frame.

5.2.3 Transition generation

We define transition generation as a type of control with temporally sparse spatial constraints, i.e. where large gaps of motion must be filled without explicit conditioning during the missing frames such as trajectory or contact information. This is related to keyframe or motion interpolation (e.g. [216]), but our work extends interpolation in that the system allows for generating whole cycles of motion, which cannot be done by most key-based interpolation techniques, such a spline fitting. Pioneering approaches [75, 218] on transition generation and interpolation used spacetime constraints and inverse kinematics to produce physically-plausible motion between keyframes. Work with probabilistic models of human motion have also been used for filling gaps of animation. These include the MAP optimizers of Chai *et al.* [140] and Min *et al.* [141], the Gaussian process dynamical models from Wang *et al.* [144] and Markov models with dynamic auto-regressive forests from Lehrmann *et al.* [204]. All of these present specific models for given action and actors. This can make combinations of actions look scripted and sequential. The scalability and expressiveness of deep neural networks has been applied to keyframe animation by Zhang *et al.* [167], who use an RNN conditioned on key-frames to produce jumping motions for a simple 2D model. Harvey *et al.* [217] present Recurrent Transition Networks (RTN) that operate on a more complex human character, but work on fixed-lengths transitions with positional data only and are deterministic. We use the core architecture of the RTN as we make use of the separately encoded inputs to apply our latent modifiers. Hernandez *et al.* [49] recently applied convolutional adversarial networks to pose the problem of prediction or transition generation as an in-painting one, given the success of convolutional generative adversarial networks on such tasks. They also propose frequency-based losses to assess motion quality, but do not provide a detailed evaluation for the task of in-betweening.

5.3 Methods

5.3.1 Data formatting

We use a humanoid skeleton that has $j = 28$ joints when using the Human3.6M dataset and $j = 22$ in the case of the LaFAN1 dataset. We use a local quaternion vector $\mathbf{q}_t$ of $j * 4$ dimensions as our main data representation along with a 3-dimensional global root velocity vector $\dot{\mathbf{r}}_t$ at each timestep t. We also extract from the data, based on toes and feet velocities, contact information as a binary vector $\mathbf{c}_t$ of 4 dimensions that we use when

working with the LaFAN1 dataset. The offset vectors $\mathbf{o}_t^r$ and $\mathbf{o}_t^q$ contain respectively the global root position's offset and local-quaternions' offsets from the target keyframe at time t. Even though the quaternion offset could be expressed as a valid, normalized quaternion, we found that using simpler element-wise linear differences simplifies learning and yields better performance. When computing our positional loss, we reformat the predicted state into a global positions vector $\mathbf{p}_{t+1}$ using $\mathbf{q}_{t+1}$, $\mathbf{r}_{t+1}$ and the stored, constant local bone translations $\mathbf{b}$ by performing Forward Kinematics (FK). The resulting vector $\mathbf{p}_{t+1}$ has $j * 3$ dimensions. We also retrieve through FK the global quaternions vector $\mathbf{g}_{t+1}$, which we use for quantitatively evaluating transitions.

The discriminator use as input sequences of 3-dimensional vectors of global root velocities $\dot{\mathbf{r}}$, concatenated with $\mathbf{x}$ and $\dot{\mathbf{x}}$, the root-relative positions and velocities of all other bones respectively. The vectors $\mathbf{x}$ and $\dot{\mathbf{x}}$ both have $(j-1) * 3$ dimensions.

To simplify the learning process, we rotate each input sequence seen by the network around the Y axis (up) so that the root of the skeleton points towards the X^+ axis on the last frame of past context. Each transition thus starts with the same global horizontal facing. We refer to rotations and positions relative to this frame as *global* in the rest of this work. When using the network inside a content creation software, we store the applied rotation in order to rotate back the generated motion to fit the context. Note however that this has no effect on the public dataset Human3.6M since root transformations are set to the identity on the first frame of any sequences, regardless of the actual world orientation. We also augment the data by mirroring the sequences over the X^+ axes with a probability of 0.5 during training.

5.3.2 Transition Generator

Figure 5.2 presents a visual depiction of our recurrent generator for a single timestep. It uses the same input separation used by the RTN network [217], but operates on angular data and uses FK in order to retrieve global positions [53]. It is also augmented with our latent space modifiers $\mathbf{z}_{tta}$ and $\mathbf{z}_{target}$. Finally it also uses different losses, such as an adversarial loss for improved realism of the generated motions.

As seen in Figure 5.2, the generator has three different encoders that take the different data vectors described above as inputs; the character state encoder, the offset encoder, and the target encoder. The encoders are all fully-connected Feed-Forward Networks (FFN) with a hidden layer of 512 units and an output layer of 256 units. All layers use the Piecewise Linear Activation function (PLU) [219], which performed slightly better than Rectified Linear Units (ReLU) in our experiments. The time-to-arrival embedding $\mathbf{z}_{tta}$ has 256 dimensions and is added to the latent input representations. Offset and target embeddings $\mathbf{h}_t^{offset}$ and $\mathbf{h}_t^{target}$ are

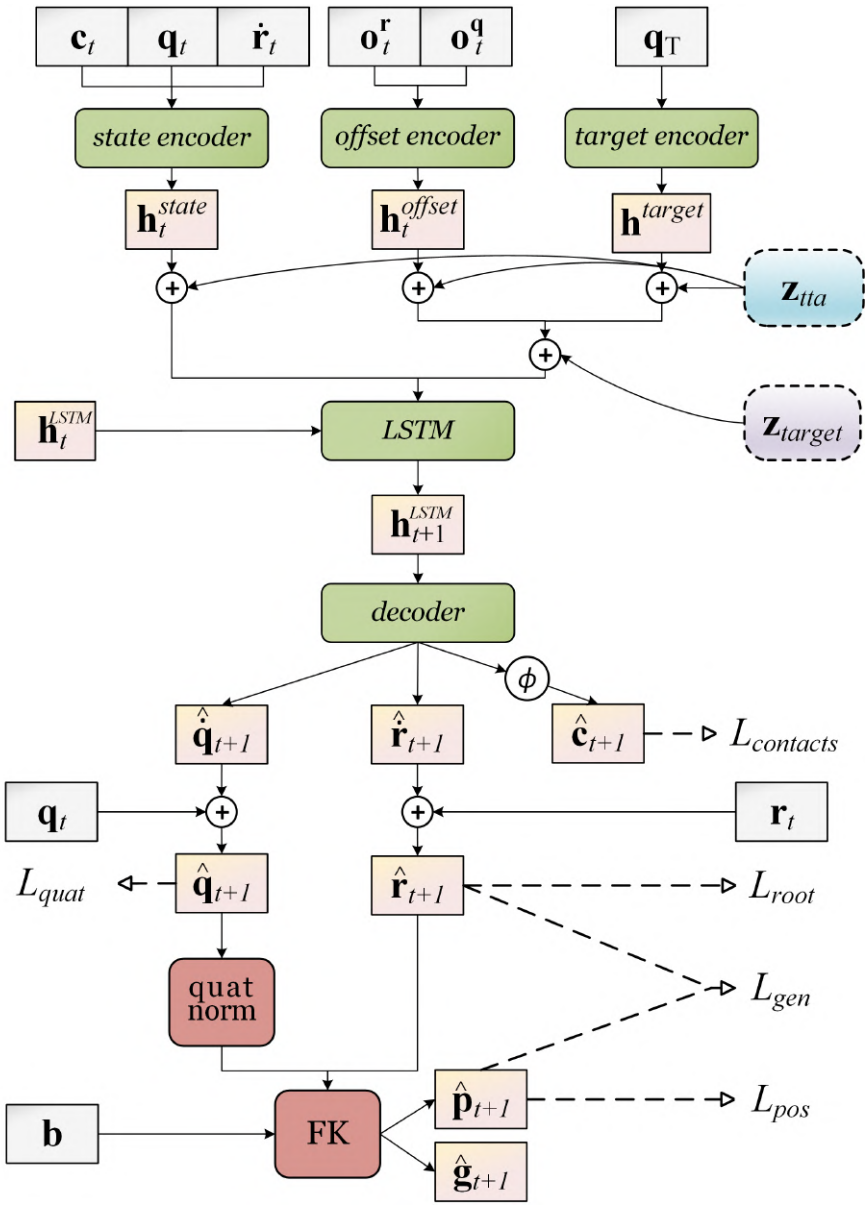


Figure 5.2 **Overview of the TG$_{complete}$ architecture for in-betweening.** Computations for a single timestep are shown. Visual concatenation of input boxes or arrows represents vector concatenation. Green boxes are the jointly trained neural networks. Dashed boxes represent our two proposed embedding modifiers. The "quat norm" and "FK" red boxes represent the quaternion normalization and Forward Kinematics operations respectively. The $\oplus$ sign represents element-wise addition and ϕ is the sigmoid non-linearity. Outputs are linked to associated losses with dashed lines.

then concatenated and added to the 512-dimensional target-noise vector $\mathbf{z}_{target}$. Next, the three augmented embeddings are concatenated and fed as input to a recurrent Long-Short-Term-Memory (LSTM) layer. The embedding from the recurrent layer, $\mathbf{h}_t^{LSTM}$ is then fed to the decoder, another FFN with two PLU hidden layers of 512 and 256 units respectively and a linear output layer. The resulting output is separated into local-quaternion and root velocities $\hat{\dot{\mathbf{q}}}_{t+1}$ and $\hat{\dot{\mathbf{r}}}_{t+1}$ to retrieve the next character state. When working with the LaFAN1 dataset, the decoder has four extra output dimensions that go through a sigmoid non-linearity ϕ to retrieve contact predictions $\hat{\mathbf{c}}_{t+1}$. The estimated quaternions $\hat{\mathbf{q}}_{t+1}$ are normalized as valid unit quaternions and used along with the new root position $\hat{\mathbf{r}}_{t+1}$ and the constant bone offsets $\mathbf{b}$ to perform FK and retrieve the new global positions $\hat{\mathbf{p}}_{t+1}$.

5.3.3 Time-to-arrival embeddings

We present here our method to allow robustness to variable lengths of in-betweening. In order to achieve this, simply adding conditioning information about the target keyframe is insufficient since the recurrent layer must be aware of the number of frames left until the target must be reached. This is essential to produce a smooth transition without teleportation or stalling. Transformer networks [35] are attention-based models that are increasingly used in natural language processing due to their state-of-the-art modeling capacity. They are sequence-to-sequence models that do not use recurrent layers but require positional encodings that modify a word embedding to represent its location in a sentence. Our problem is also a sequence-to-sequence task where we translate a sequence of seed frames to a transition sequence, with additional conditioning on the target keyframe. Although our generator does use a recurrent layer, it needs time-to-arrival awareness in order to gracefully handle transitions of variable lengths. To this end, we use the mathematical formulation of positional encodings, that we base in our case on the time-to-arrival to the target:

$$\mathbf{z}_{tta,2i} = sin\left(\frac{tta}{basis^{2i/d}}\right) \tag{5.1}$$

$$\mathbf{z}_{tta,2i+1} = cos\left(\frac{tta}{basis^{2i/d}}\right) \tag{5.2}$$

where *tta* is the number of timesteps until arrival and the second subscript of the vector $\mathbf{z}_{tta,\_}$ represents the dimension index. The value d is the dimensionality of the input embeddings, and $i \in [0, ..., d/2]$. The *basis* component influences the rate of change in frequencies along the embedding dimensions. It is set to 10 000 as in most transformer implementations.

Time-to-arrival embeddings thus provide continuous codes that will shift input representations in the latent space smoothly and uniquely for each transition step due to the phase and

frequency shifts of the sinusoidal waves on each dimension. Such embedding is thus bounded, smooth and dense, three characteristics beneficial for learning. Its additive nature makes it harder for a neural network to ignore, as can be the case with concatenation methods. This follows the successful trend in computer vision [175,177] of conditioning through transformations of the latent space instead of conditioning with input concatenation. In these cases, the conditioning signals are significantly more complex and the affine transformations need to be learned, whereas Vaswani *et al.* [35] report similar performance when using this sine-based formulation as when using learned embeddings.

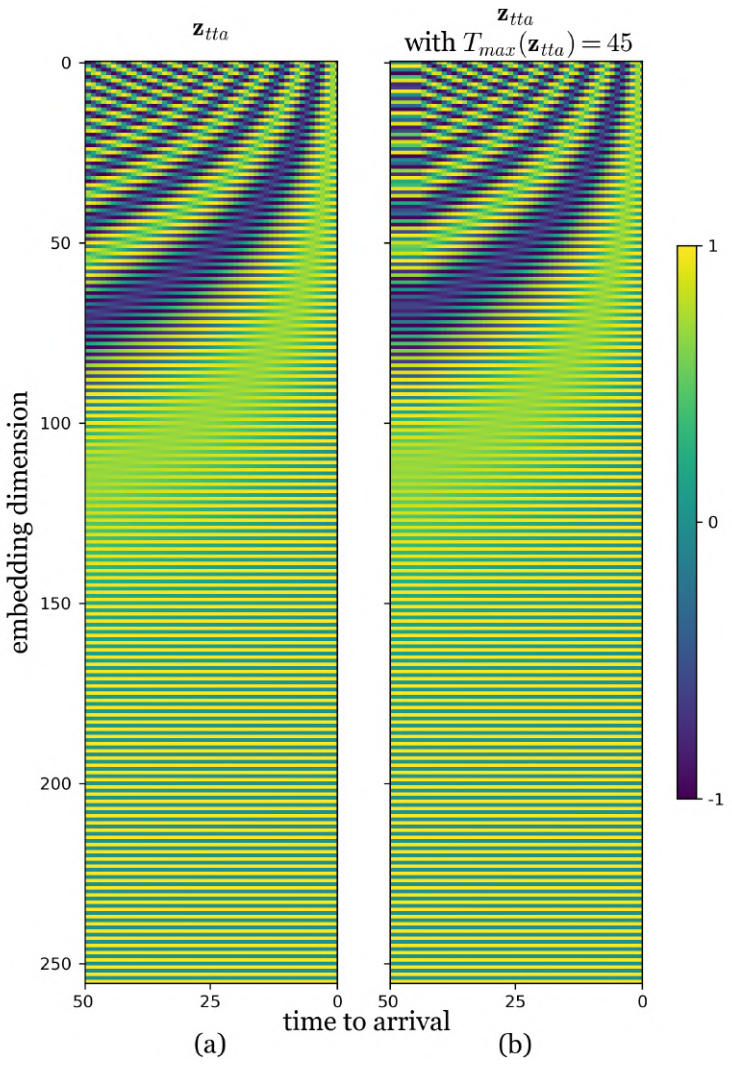


Figure 5.3 **Visual depiction of time-to-arrival embeddings.** Sub-figure (b) shows the effect of using $T_{max}(\mathbf{z}_{tta})$, which in practice improves performances when generalizing to longer transitions as it prevents initializing the LSTM hidden state with novel embeddings.

It is said that positional encodings can generalize to longer sequences in the natural language

domain. However, since $\mathbf{z}_{tta}$ evolves backwards in time to retrieve a time-to-arrival representation, generalizing to longer sequences becomes a more difficult challenge. Indeed, in the cases of Transformers (without temporal reversal), the first embeddings of the sequence are always the same and smoothly evolve towards new ones when generalizing to longer sequences. In our case, longer sequences change drastically the initial embedding seen and may thus generate unstable hidden states inside the recurrent layer before the transition begins. This can hurt performance on the first frames of transitions when extending the time-horizon after training. To alleviate this problem, we define a maximum duration in which we allow $\mathbf{z}_{tta}$ to vary, and fix it past this maximum duration. Precisely, the maximum duration $T_{max}(\mathbf{z}_{tta})$ is set to $T_{max}(trans) + T_{past} - 5$, where $T_{max}(trans)$ is the maximum transition length seen during training and T_{past} is the number of seed frames given before the transition. This means that when dealing with transitions of length $T_{max}(trans)$, the model sees a constant $\mathbf{z}_{tta}$ for 5 frames before it starts to vary. This allows the network to handle a constant $\mathbf{z}_{tta}$ and to keep the benefits of this augmentation even when generalizing to longer transitions. Visual representations of $\mathbf{z}_{tta}$ and the effects $T_{max}(\mathbf{z}_{tta})$ are shown in Figure 5.3.

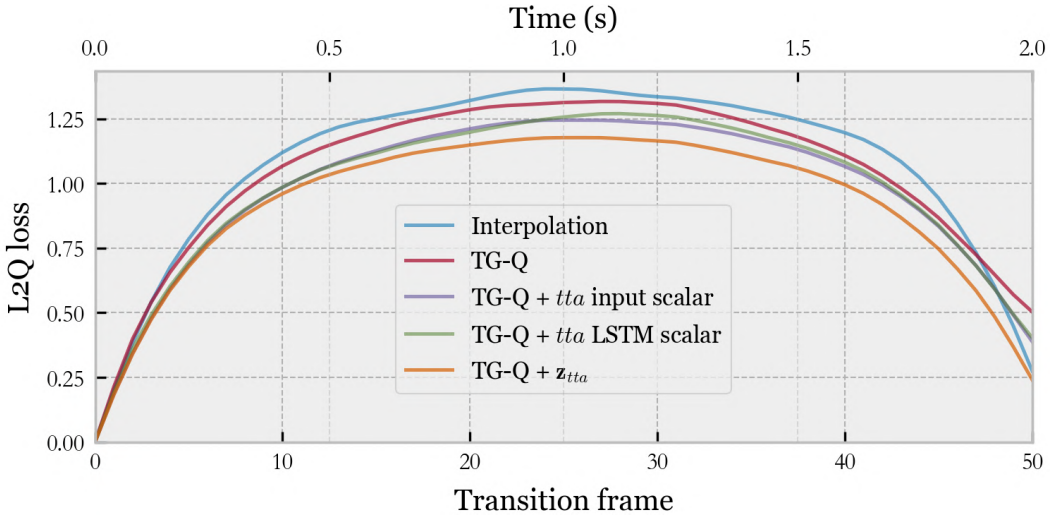


Figure 5.4 **Reducing the L2Q loss with $\mathbf{z}_{tta}$.** We compare simple interpolation with our temporally unaware model (TG-Q) on the walking subset of Human 3.6M. We further test two strategies based on adding a single *tta* dimension either to the character state (TG-Q + *tta* input scalar) or the LSTM inputs (TG-Q + *tta* LSTM scalar). Finally, our use of time-to-arrival embeddings (TG-Q + $\mathbf{z}_{tta}$) yields the best results, mostly noticeable at the end of transitions, where the generated motion is smoother than interpolation.

We explored simpler approaches to induce temporal awareness, such as concatenating a time-to-arrival dimension either to the inputs of the state encoder, or to the LSTM layer's inputs. This *tta* dimension is a single scalar increasing from 0 to 1 during the transition. Its period

of increase is set to $T_{max}(\mathbf{z}_{tta})$. Results comparing these methods with a temporally unaware network, and our use of $\mathbf{z}_{tta}$ can be visualized in Figure 5.4.

5.3.4 Scheduled target noise

Another contribution of this work is to improve robustness to keyframe modifications and to enforce diversity in the generated transitions given a fixed context. To do so we propose a scheduled target-distortion strategy. We first concatenate the encoded embeddings $\mathbf{h}_t^{offset}$ and $\mathbf{h}_t^{target}$ of the current offset vector and the target keyframe respectively. We then add to the resulting vector the target noise vector $\mathbf{z}_{target}$, sampled once per sequence from a spherical, zero-centered Gaussian distribution $\mathcal{N}(0, I * \sigma_{target})$. The standard deviation σ_{target} is an hyper-parameter controlling the level of accepted distortion. In order to produce smooth transitions to the target, we then define a target noise multiplier λ_{target}, responsible for scaling down $\mathbf{z}_{target}$ as the number of remaining timesteps goes down. We define a period of noise-free generation (5 frames) where $\lambda_{target} = 0$ and a period of linear decrease of the target-noise (25 frames) to produce our noise-scale schedule. Beyond 30 frames before the target, the target-noise is therefore constant and $\lambda_{target} = 1$.

$$\lambda_{target} = \begin{cases} 1 & \text{if } tta \geq 30 \\ \frac{tta-5}{25} & \text{if } 5 \leq tta < 30 \\ 0 & \text{if } tta < 5 \end{cases} \tag{5.3}$$

Since this modifier is additive, it also corrupts time-to-arrival information, effectively distorting the timing information. This allows to modify the pace of the generated motion. Our target noise schedule is intuitively similar to an agent receiving a distorted view of its long-term target, with this goal becoming clearer as the agent advances towards it. This additive embedding modifier outperformed in our experiments another common approach in terms of diversity of the transitions while keeping the motion plausible. Indeed a widespread approach to conditional GANs is to use a noise vector $\mathbf{z}_{concat}$ as additional input to the conditioned generator in order to enable stochasticity and potentially disambiguate the possible outcomes from the condition (e.g. avoid mode collapsing). However in highly constrained cases like ours, the condition is often informative enough to obtain good performance, especially at the beginning of the training. This leads to the generator learning to ignore the additional noise, as observed in our tests (see Figure 5.5). We thus force the transition generator to be stochastic by using $\mathbf{z}_{target}$ to distort its view of the target and current offsets.

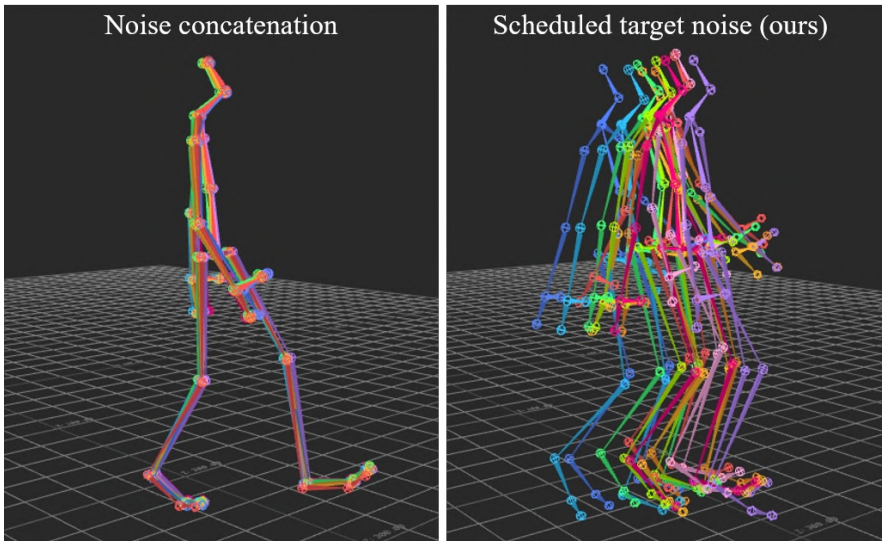


Figure 5.5 **Increasing variability with $\mathbf{z}_{target}$.** We compare $\mathbf{z}_{concat}$ (left) against $\mathbf{z}_{target}$ (right) midway in a 100-frames transition re-sampled 10 times. The generator successfully learns to ignore $\mathbf{z}_{concat}$ while $\mathbf{z}_{target}$ is imposed and leads to noticeable variations with controllable scale.

5.3.5 Motion Discriminators

A common problem with reconstruction-based losses and RNNs is the *blurriness* of the results, which is translated into collapse to the average motion and foot slides when predicting motion. The target keyframe conditioning can slightly alleviate these problems, but additional improvement comes from our use of adversarial losses, given by two discriminator networks. We use two variants of a relatively simple feed-forward architecture for our discriminators, or *critics*, $\mathcal{C}_1$ and $\mathcal{C}_2$. Each discriminator has 3 fully-connected layers, with the last one being a 1D linear output layer. $\mathcal{C}_1$ is a *long-term* critic that looks at sliding windows of 10 consecutive frames of motion and $\mathcal{C}_2$ is the *short-term* critic and looks at windows of instant motion over 2 frames. Both critics have 512 and 256 units in their first and second hidden layers respectively. The hidden layers use ReLU activations. We average the discriminator scores over time in order to produce a single scalar loss. A visual summary of our sliding critics is presented in Appendix A.

5.3.6 Losses

In order to make the training stable and to obtain the most realistic results, we use multiple loss functions as complementary soft constraints that the neural network learns to respect.

Reconstruction Losses

All of our reconstruction losses for a predicted sequence $\hat{X}$ given its ground-truth X are computed with the L1 norm:

$$L_{quat} = \frac{1}{T}\sum_{t=0}^{T-1} \|\hat{\mathbf{q}}_t - \mathbf{q}_t\|_1 \tag{5.4}$$

$$L_{root} = \frac{1}{T}\sum_{t=0}^{T-1} \|\hat{\mathbf{r}}_t - \mathbf{r}_t\|_1 \tag{5.5}$$

$$L_{pos} = \frac{1}{T}\sum_{t=0}^{T-1} \|\hat{\mathbf{p}}_t - \mathbf{p}_t\|_1 \tag{5.6}$$

$$L_{contacts} = \frac{1}{T}\sum_{t=0}^{T-1} \|\hat{\mathbf{c}}_t - \mathbf{c}_t\|_1 \tag{5.7}$$

where T is the sequence length. The two main losses that we use are the local-quaternion loss L_{quat} and the root-position loss L_{root}. The former is computed over all joints' local rotations, including the root-node, which in this case also determines global orientation. The latter is responsible for the learning of the global root displacement. As an additional reconstruction loss, we use a positional-loss L_{pos} that is computed on the global position of each joints retrieved through FK. In theory, the use of L_{pos} isn't necessary to achieve a perfect reconstruction of the character state when using L_{quat} and L_{root}, but as noted by Pavllo *et al.* [53], using global positions helps to implicitly weight the orientation of the bone's hierarchy for better results. As we will show in Section 5.4.2, adding this loss indeed improves results on both quaternion and translation reconstructions. Finally, in order to allow for runtime Inverse-Kinematics correction (IK) of the legs inside an animation software, we also use a contact prediction loss $L_{contacts}$, between predicted contacts $\hat{\mathbf{c}}_t$ and true contacts $\mathbf{c}_t$. We use the contact predictions at runtime to indicate when to perform IK on each leg. This loss is used only for models trained on the LaFAN1 dataset and that are deployed in our MotionBuilder plugin.

Adversarial Losses

We use the Least Square GAN (LSGAN) formulation [71]. As our discriminators operate on sliding windows of motion, we average their losses over time. Our LSGAN losses are defined

as follows:

$$L_{gen} = \frac{1}{2}\mathbb{E}_{\mathrm{X}_p,\mathrm{X}_f\sim p_{Data}}[(D(\mathrm{X}_p, G(X_p, \mathrm{X}_f), \mathrm{X}_f) - 1)^2], \tag{5.8}$$

$$L_{disc} = \frac{1}{2}\mathbb{E}_{\mathrm{X}_p,\mathrm{X}_{trans},\mathrm{X}_f\sim p_{Data}}[(D(\mathrm{X}_p, \mathrm{X}_{trans}, \mathrm{X}_f) - 1)^2] + \frac{1}{2}\mathbb{E}_{\mathrm{X}_p,\mathrm{X}_f\sim p_{Data}}[(D(\mathrm{X}_p, G(X_p, \mathrm{X}_f), \mathrm{X}_f))^2], \tag{5.9}$$

where X_p, X_f, and X_{trans} represent the past context, target state, and transition respectively, in the discriminator input format described in Section 5.3.1. G is the transition generator network. Both discriminators use the same loss, with different input sequence lengths.

5.3.7 Training

Progressive growing of transitions

In order to accelerate training, we adopt a curriculum learning strategy with respect to the transition lengths. Each training starts at the first epoch with $P_{min} = \tilde{P}_{max} = 5$, where P_{min} and $\tilde{P}_{max}$ are the minimal and current maximal transition lengths. During training, we increase $\tilde{P}_{max}$ until it reaches the true maximum transition length P_{max}. The increase rate is set by number of epochs n_{ep-max} by which we wish to have reached $\tilde{P}_{max} = P_{max}$. For each minibatch, we sample uniformly the current transition length between P_{min} and $\tilde{P}_{max}$, making the network train with variable length transitions, while beginning the training with simple tasks only. In our experiments, this leads to similar results as using any teacher forcing strategy, while accelerating the beginning of training due to the shorter batches. Empirically, it also outperformed gradient clipping. At evaluation time, the transition length is fixed to the desired length.

Sliding critics

In practice, our discriminators are implemented as 1D temporal convolutions, with strides of 1, without padding, and with receptive fields of 1 in the last 2 layers, yielding parallel feed-forward networks for each motion window in the sequence.

Hyperparameters

In all of our experiments, we use minibatches of 32 sequences of variable lengths as explained above. We use the AMSgrad optimizer [220] with a learning rate of 0.001 and adjusted parameters ($\beta_1 = 0.5, \beta_2 = 0.9$) for increased stability. We scale all of our losses to be

approximately equal on the LaFAN1 dataset for an untrained network before tuning them with custom weights. In all of our experiments, these relative weights (when applicable) are of 1.0 for L_{quat} and L_{root}, 0.5 for L_{pos}, and 0.1 for L_{gen} and $L_{contacts}$. The target noise's standard deviation σ_{target} is 0.5. In experiments on Human3.6M, we set n_{ep-max} to 5 while it is set to 3 on the larger LaFAN1 dataset.

5.4 Experiments and Results

5.4.1 Motion prediction

Table 5.1 **Unconstrained motion prediction results on Human 3.6M**. The VGRU-d/rl models are from [54]. The TP-RNN is from [55] and has to our knowledge the best published results on motion prediction for this benchmark. Our model, ERD-QV is competitive with the state-of-the-art on angular errors and improves performance with respect to the recently proposed NPSS metric on all actions.

	Walking							**Eating**						
	MAE						NPSS	MAE						NPSS
milliseconds	80	160	320	500	560	1000	0-1000	80	160	320	500	560	1000	0-1000
Zero-Vel	0.39	0.68	0.99	1.15	1.35	1.32	0.1418	0.27	0.48	0.73	0.86	1.04	1.38	0.0843
VGRU-rl	0.34	0.47	0.64	0.72	-	-	-	0.27	0.40	0.64	0.79	-	-	-
VGRU-d	-	-	-	-	-	-	0.1170	-	-	-	-	-	-	0.1210
TP-RNN	0.25	0.41	0.58	-	0.74	**0.77**	-	0.20	**0.33**	**0.53**	-	0.84	**1.14**	-
ERD-QV	**0.20**	**0.34**	**0.56**	**0.64**	**0.72**	0.79	**0.0767**	**0.18**	**0.33**	**0.53**	**0.63**	**0.78**	1.17	**0.0763**

	Smoking							**Discussion**						
	MAE						NPSS	MAE						NPSS
milliseconds	80	160	320	500	560	1000	0-1000	80	160	320	500	560	1000	0-1000
Zero-Vel	0.26	0.48	0.97	0.95	1.02	1.69	0.0638	0.31	0.67	0.94	1.04	1.41	1.96	**0.0638**
VGRU-rl	0.36	0.61	**0.85**	**0.92**	-	-	-	0.46	0.82	0.95	1.21	-	-	-
VGRU-d	-	-	-	-	-	-	0.0840	-	-	-	-	-	-	0.1940
TP-RNN	0.26	0.48	0.88	-	**0.98**	1.66	-	0.26	**0.48**	0.88	-	**0.98**	**1.66**	-
ERD-QV	**0.23**	**0.47**	0.96	0.99	**0.98**	**1.58**	**0.0537**	**0.23**	0.59	**0.86**	**0.93**	1.30	1.75	0.1201

Based on recent advances in motion prediction, we first build a motion prediction network that yields state-of-the-art results. We evaluate our model on the popular motion prediction benchmark that uses the Human 3.6M dataset. We follow the evaluation protocol defined by Jain *et al.* [87] that we base on the code from Martinez *et al.* [32]. We train the networks for

40 500 iterations before evaluation. We use the core architecture of Harvey *et al.* [217] since their separate encoders allow us to apply our embedding modifiers. In the case of unconstrained prediction however, this is more similar to the Encoder-Recurrent-Decoder (ERD) networks from Fragkiadaki *et al.* [33]. We also apply the velocity-based input representation of Chiu *et al.* [55] which seems to be a key component to improve performance. This is empirically shown in our experiments for motion prediction, but as we will see in Section 5.4.2, it doesn't hold for transition generation, where the character state as an input is more informative than velocities to produce correct transitions, evaluated on global angles and positions. Another difference lies in our data representation, which is based on quaternions instead of exponential maps. We call our architecture for motion prediction ERD-Quaternion Velocity network (ERD-QV). This model is therefore similar to the one depicted in Figure 5.2, with quaternions velocities $\dot{\mathbf{q}}_t$ as only inputs of the state encoder instead of $\mathbf{q}_t$ and $\dot{\mathbf{r}}_t$, and without the two other encoders and their inputs. No embedding modifier and no FK are used in this case, and the only loss used is the L1 norm on joint-local quaternions. In this evaluation, the root transform is ignored, to be consistent with previous works.

In Table 5.1, we compare this model with the TP-RNN which obtains to our knowledge state-of-the-art results for Euler angle differences. We also compare with two variants of the VGRU architecture proposed by Gopalakrishnan *et al.* [54], who propose a novel Normalized Power Spectrum Similarity (NPSS) metric for motion prediction that is more correlated to human assessment of quality for motion. Note that in most cases, we improve upon the TP-RNN for angular errors and perform better than the VGRU-d proposed by Gopalakrishnan *et al.* [54] on their proposed metric. This allows us to confirm the performance of our chosen architecture as the basis of our transition generation model.

5.4.2 Walking in-betweens on Human 3.6M

Given our highly performing prediction architecture, we now build upon it to produce a transition generator (TG). We start off by first adding conditioning information about the future target and current offset to the target, and then sequentially add our proposed contributions to show their quantitative benefits on a novel transition benchmark. Even though the Human 3.6M dataset is one of the most used in motion prediction research, most of the actions it contains are ill-suited for long-term prediction or transitions (e.g. *smoking*, *discussion*, *phoning*, ...) as they consists of sporadic, random short movements that are impossible to predict beyond some short time horizons. We thus choose to use a subset of the Human 3.6M dataset consisting only of the three walk-related actions (*walking, walkingdog, walkingtogether*) as they are more interesting to test for transitions over 0.5 seconds long.

Like previous studies, we work with a 25Hz sampling rate and thus subsample the original 50Hz data. The walking data subset has 55 710 frames in total and we keep Subject 5 as the test subject. The test set is composed of windows of motion regularly sampled every 10 frames in the sequences of Subject 5. Our test set thus contains 1419 windows with lengths that depend on the evaluation length. In order to evaluate robustness to variable lengths of transitions, we train the models on transitions of random lengths ranging from 0.2 to 2 seconds (5 to 50 frames) and evaluate on lengths going up to 4 seconds (100 frames) to also test generalization to longer time horizons. We train the models for 55 000 iterations, and report average L2 distances of global quaternions (L2Q) and global positions (L2P):

$$L2Q = \frac{1}{|\mathcal{D}|}\frac{1}{T}\sum_{s\in\mathcal{D}}\sum_{t=0}^{T-1}\|\hat{\mathbf{g}}_t^s - \mathbf{g}_t^s\|_2 \tag{5.10}$$

$$L2P = \frac{1}{|\mathcal{D}|}\frac{1}{T}\sum_{s\in\mathcal{D}}\sum_{t=0}^{T-1}\|\hat{\mathbf{p}}_t^s - \mathbf{p}_t^s\|_2 \tag{5.11}$$

where s is a transition sequence of the test set $\mathcal{D}$, and T is the transition length. Note that we compute L2P on normalized global positions using statistics from the training set. Precisely, we extract the global positions' statistics on windows of 70 frames[2] offset by 10 frames in which the motion has been centered around the origin on the horizontal plane. We center the motion by subtracting the mean of the root's XZ positions on all joints' XZ positions. We report the L2P metric as it is arguably a better metric than any angular loss for assessing visual quality of transitions with global displacements. However, it is not complete in that bone orientations might be wrong even with the right positions. We also report NPSS scores, which are based on angular frequency comparisons with the ground truth. Our results are shown in Table 5.2. Our first baseline consists of a naive interpolation strategy in which we linearly interpolate the root position and spherically interpolate the quaternions between the keyframes defining the transition. On the very short-term, this is an efficient strategy as motion becomes almost linear in sufficiently small timescales. We then compare transition generators that receive quaternion velocities as input (TG-QV) with one receiving normal quaternions (TG-Q) as depicted in Figure 5.2. Both approaches have similar performance on short transitions, while TG-QV shows worst results on longer transitions. This can be expected with such a model that isn't given a clear representation of the character state at each frame. We thus choose TG-Q as our main baseline onto which we sequentially add our proposed modifications. We first add the global positional loss ($+L_{pos}$) as an additional

[2]We train with transitions of maximum lengths of 50 frames, plus 10 seed frames and 10 frames of future context to visually assess the motion continuation, from which the first frame is the target keyframe. This yields windows of 70 frames in total.

Table 5.2 **Transition generation benchmark on Human 3.6M**. Models were trained with transition lengths of maximum 50 frames, but are evaluated beyond this horizon, up to 100 frames (4 seconds).

	L2Q						
Length (frames)	5	10	25	50	75	100	AVG
Interpolation	**0.22**	0.43	0.84	1.09	1.48	2.03	1.02
TG-QV	0.36	0.51	0.76	1.08	1.54	1.97	1.04
TG-Q	0.33	0.48	0.76	1.05	1.40	1.79	0.97
$+L_{pos}$	0.32	0.45	0.74	1.04	1.40	1.80	0.97
$+\mathbf{z}_{tta}$	0.26	0.40	0.70	0.96	1.30	1.67	0.88
$+\mathbf{z}_{target}$	0.26	0.40	**0.68**	0.94	1.22	1.56	0.84
$+L_{gen}$ ($\text{TG}_{complete}$)	0.24	**0.38**	**0.68**	**0.93**	**1.20**	**1.49**	**0.82**
	L2P						
Interpolation	0.32	0.69	1.62	2.70	4.34	6.18	2.64
TG-QV	0.48	0.74	1.22	2.44	4.71	7.11	2.78
TG-Q	0.50	0.75	1.29	2.27	4.07	6.28	2.53
$+ L_{pos}$	0.44	0.67	1.22	2.27	4.14	6.39	2.52
$+ \mathbf{z}_{tta}$	0.34	0.54	1.04	1.82	3.25	5.27	2.04
$+ \mathbf{z}_{target}$	0.32	0.51	**0.97**	**1.67**	2.85	4.67	1.83
$+ L_{gen}$ ($\text{TG}_{complete}$)	**0.28**	**0.48**	**0.97**	1.68	**2.71**	**4.37**	**1.75**
	NPSS						
Interpolation	**0.0020**	0.0110	0.0948	0.3555	0.7790	1.5170	0.4599
TG-QV	0.0033	0.0124	0.0804	0.3118	0.8535	2.0822	0.5573
TG-Q	0.0031	0.0121	0.0842	0.3188	0.8128	1.8151	0.5077
$+ L_{pos}$	0.0031	0.0117	0.0811	0.3105	0.7988	1.9658	0.5285
$+ \mathbf{z}_{tta}$	0.0026	0.0107	0.0773	0.2933	0.7403	1.6459	0.4617
$+ \mathbf{z}_{target}$	0.0026	0.0107	**0.0765**	0.2974	0.7531	1.6270	0.4612
$+ L_{gen}$ ($\text{TG}_{complete}$)	0.0024	**0.0102**	**0.0765**	**0.2873**	**0.6572**	**1.3364**	**0.3950**

training signal, which improves performance on most metrics and lengths. We then add the unconstrained time-to-arrival embedding modifier ($+\mathbf{z}_{tta}$) and observe our most significant improvement. These effects on 50-frames translations are summarized in Figure 5.4. Next, we evaluate the effects of our scheduled target embedding modifier $\mathbf{z}_{target}$. Note that it is turned off for quantitative evaluation. The effects are minor for transitions of 5 and 10 frames, but $\mathbf{z}_{target}$ is shown to generally improve performances for longer transitions. We argue that these improvements come from the fact that this target noise probably helps generalizing to new sequences as it improves the model's robustness to new or noisy conditioning information. Finally, we obtain our complete model ($\text{TG}_{complete}$) by adding our adversarial loss L_{gen}, which interestingly not only improves the visual results of the generated motions, but also most of the quantitative scores.

Qualitatively, enabling the target noise allows the model to produce variations of the same transitions, and it is trivial to control the level of variation by controlling σ_{target}. We compare our approach to a simpler variant that also aims at inducing stochasticity in the generated transition. In this variant, we aim at potentially disambiguating the missing target information such as velocities by concatenating a random noise vector $\mathbf{z}_{concat}$ to the target keyframe input $\mathbf{q}_T$. This is similar to a strategy used in conditional GANs to avoid mode collapse given the condition. Figure 5.5 and the accompanying video show typical results obtained with our technique against this more classical technique.

5.4.3 Scaling up with the LaFAN1 dataset

Given our model selection based on the Human 3.6M walking benchmark discussed above, we further test our complete model on a novel, high quality motion dataset containing a wide range of actions, often with significant global displacements interesting for in-betweening compared to the Human3.6M dataset. This dataset contains 496 672 motion frames sampled at 30Hz and captured in a production-grade MOCAP studio. It contains actions performed by 5 subjects, with Subject 5 used as the test set. Similarly to the procedure used for the Human3.6M walking subset, our test set is made of regularly-sampled motion windows. Given the larger size of this dataset we sample our test windows from Subject 5 at every 40 frames, and thus retrieve 2232 windows for evaluation. The training statistics for normalization are computed on windows of 50 frames offset by 20 frames. Once again our starting baseline is a normal interpolation. We make public this new dataset along with accompanying code that allows one to extract the same training set and statistics as in this work, to extract the same test set, and to evaluate naive baselines (zero-velocity and interpolation) on this test set for our in-betweening benchmark. We hope this will facilitate future research and comparisons on the task of transition generation. We train our models on this dataset for 350 000 iterations on Subjects 1 to 4. We then go on to compare a reconstruction-based, future-conditioned Transition Generator (TG_{rec}) using L_{quat}, L_{root}, L_{pos} and $L_{contacts}$ with our augmented adversarial Transition Generator ($\text{TG}_{complete}$) that adds our proposed embedding modifiers $\mathbf{z}_{tta}$, $\mathbf{z}_{tta}$ and our adversarial loss L_{gen}. Results are presented in Table 5.3. Our contributions improve performance on all quantitative measurements. On this larger dataset with more complex movements, our proposed in-betweeners surpass interpolation even on the very short transitions, as opposed to what was observed on the Human3.6M walking subset. This motivates the use of our system even on short time-horizons.

Table 5.3 **Improving in-betweening on the LaFAN1 dataset**. Models were trained with transition lengths of maximum 30 frames (1 second), and are evaluated on 5, 15, 30, and 45 frames.

	L2Q			
Length (frames)	5	15	30	45
Interpolation	0.22	0.62	0.98	1.25
TG_{rec}	0.21	0.48	0.83	1.20
$TG_{complete}$	**0.17**	**0.42**	**0.69**	**0.94**
	L2P			
Interpolation	0.37	1.25	2.32	3.45
TG_{rec}	0.32	0.85	1.82	3.89
$TG_{complete}$	**0.23**	**0.65**	**1.28**	**2.24**
	NPSS			
Interpolation	0.0023	0.0391	0.2013	0.4493
TG_{rec}	0.0025	0.0304	0.1608	0.4547
$TG_{complete}$	**0.0020**	**0.0258**	**0.1328**	**0.3311**

5.4.4 Practical use inside an animation software

In order to also qualitatively test our models, we deploy networks trained on LaFAN1 in a custom plugin inside Autodesk's MotionBuilder, a widely used animation authoring and editing software. This enables the use of our model on user-defined keyframes or the generation of transitions between existing clips of animation. Figure 5.6 shows an example scene with an incomplete sequence alongside our user interface for the plugin. The *Source Character* is the one from which keyframes are extracted while the generated frames are applied onto the *Target Character*'s skeleton. In this setup it is trivial to re-sample different transitions while controlling the level of target noise through the *Variation* parameter. A variation of 0 makes the model deterministic. Changing the temporal or spatial location of the target keyframes and producing new animations is also trivial. Such examples of variations can be seen in Figure 5.7. The user can decide to apply IK guided by the network's contact predictions through the *Enable IK* checkbox. An example of the workflow and rendered results can be seen in the accompanying video.

The plugin with the loaded neural network takes 170MB of memory. Table 5.4 shows a summary of average speed performances of different in-betweening cases. This shows that an animator can use our tool to generate transition candidates almost for free when compared to manually authoring such transitions or finding similar motions in a motion database.

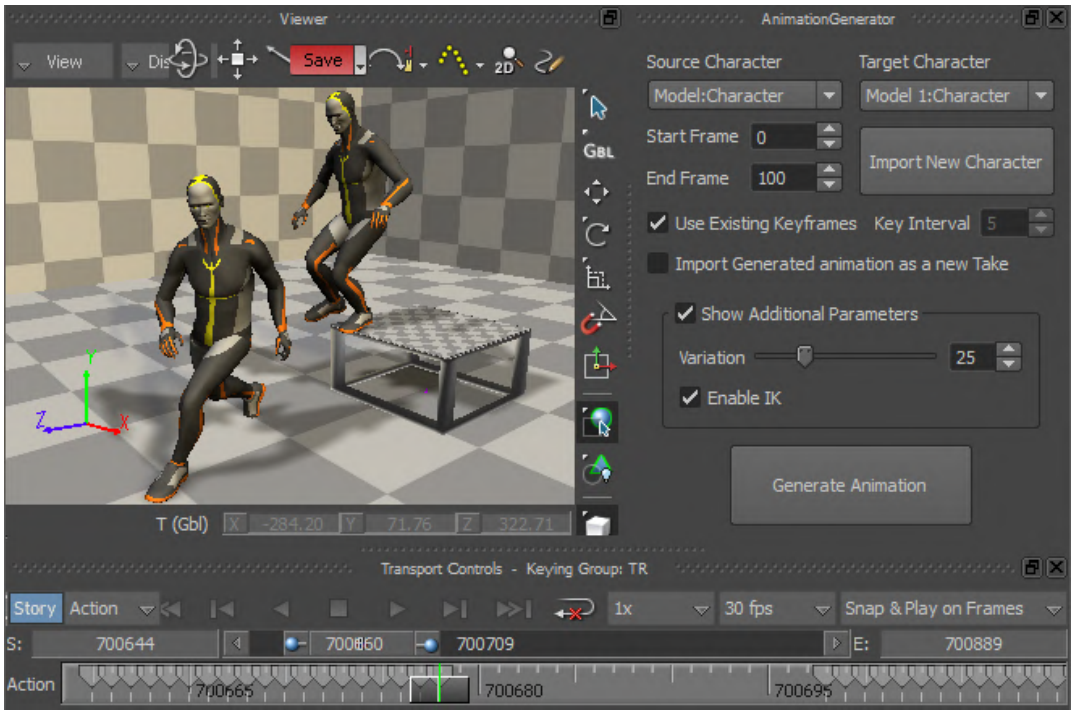


Figure 5.6 **Generating animations inside MotionBuilder.** On the left is a scene where the last seed frame and target keyframe are visible. On the right is our user interface for the plugin that allows, among other things, to specify the level of scheduled target noise for the next generation through the *variation* parameter, and to use the network's contact predictions to apply IK. On the bottom is the timeline where the gap of missing motion is visible.

5.5 Discussion

5.5.1 Additive modifiers

We found our time-to-arrival and scheduled target noise additive modifiers to be very effective for robustness to time variations and for enabling sampling capabilities. We explored relatively simpler concatenation-based methods that showed worse performances. We hypothesize that concatenating time-to-arrival or noise dimensions is often less efficient because the neural network can learn to ignore those extra dimensions which are not crucial in the beginning of the training. Additive embedding modifiers however impose a shift in latent space and thus are harder to bypass.

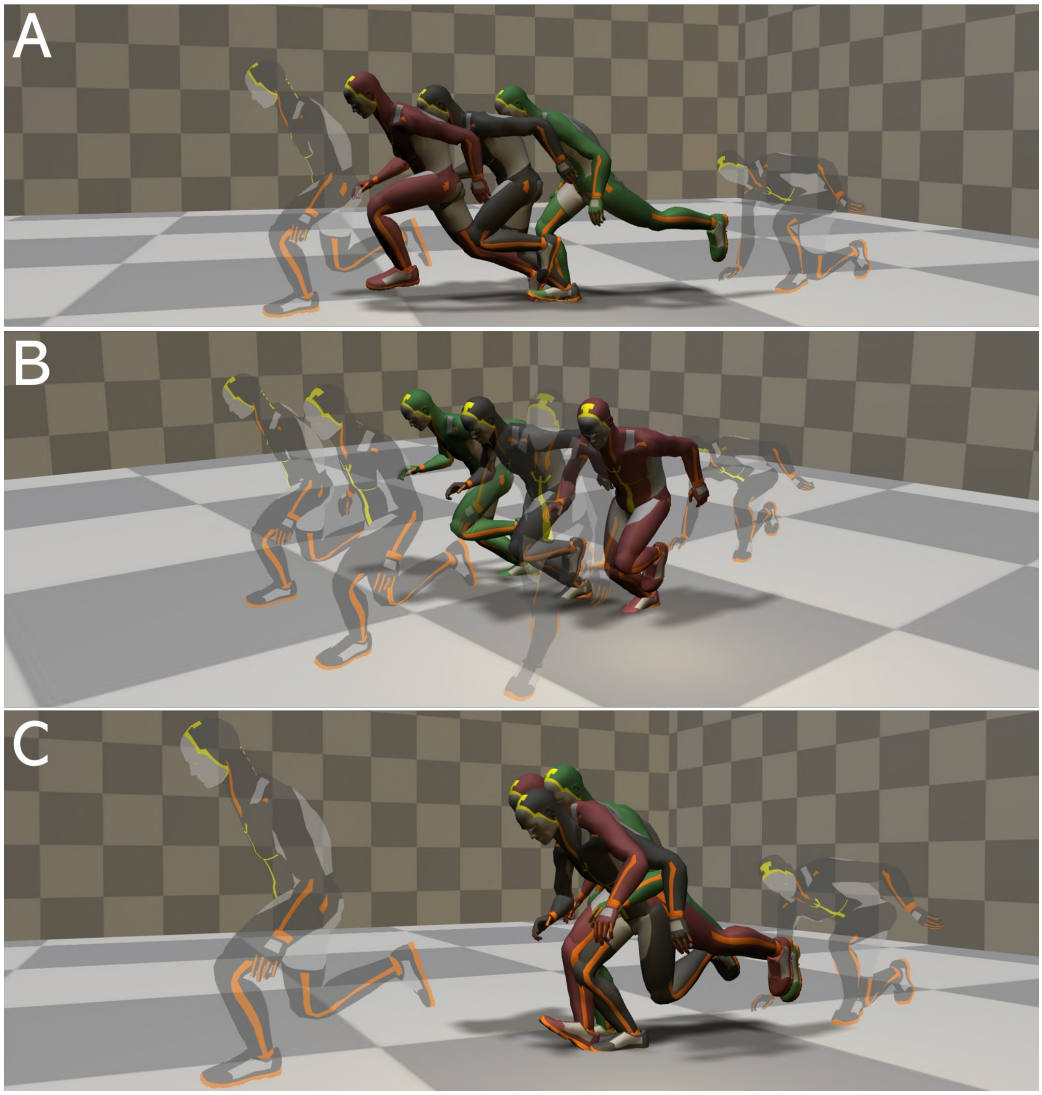


Figure 5.7 **Three types of variations of a *crouch-to-run* transition.** A single frame per generated transition is shown, taken at the same timestep. Semi-transparent poses are the start and end keyframes. **A**: Temporal variations are obtained by changing the temporal location of the second keyframe. This relies on our time-to-arrival embeddings ($\mathbf{z}_{tta}$). **B**: Spatial variations can be obtained by simply moving the target keyframe in space. **C**: Motion variation are obtained by re-sampling the same transition with our scheduled target noise ($\mathbf{z}_{target}$) enabled. These results can also be seen in the accompanying video.

Table 5.4 **Speed performance summary of our MotionBuilder plugin**. The model inference also includes the IK postprocess. The last column indicates the time taken to produce a string of 10 transitions of 30 frames. Everything is run on a Intel Xeon CPU E5-1650 @ 3.20GHz.

Transition time (s)	0.50	1.00	2.00	10 x 1.00
Keyframe extraction (s)	0.01	0.01	0.01	0.01
Model inference (s)	0.30	0.31	0.31	0.40
Applying keyframes (s)	0.72	1.05	1.65	6.79
Total (s)	1.03	1.37	1.97	7.20

5.5.2 Ablated datasets

In order to gain some insights on the importance of the training set content, we trained two additional models with ablated versions of the LaFAN1 dataset. For the first one, we removed all dance training sequences (approximately 10% of the data). For the second one, we kept only those sequences, yielding a much smaller dataset (21.5 minutes). Results showed that keeping only the dance sequences yielded similar results as the bigger ablated dataset, but that the full dataset is necessary to generate transitions that stay in a dancing style. This indicates that large amounts of generic data can be as useful as much fewer specialized sequences for a task, but that combining both is key. An example is shown in the accompanying video.

5.5.3 Dropping the Triangular-Prism

When building our motion predictor ERD-QV, we based our input represention on velocities, as suggested with the TP-RNN architecture proposed by [55]. However, we did not witness any gains when using their proposed Triangular-Prism RNN (TP-RNN) architecture. Although unclear why, it might be due to the added depth of the network by adding in our case a feed-forward encoder, making the triangular prism architecure unnecessary.

5.5.4 Incompleteness of L_{pos}

Although we use FK for our positional loss L_{pos} as suggested by Pavllo *et al.* [53], this loss isn't sufficient to produce fully defined character configurations. Indeed using only this loss may lead to the correct positions of the joints but offers no guarantee for the bone orientations, and led in our experiments to noticeable artifacts especially at the ends of the kinematic chains.

5.5.5 Recurrent cell types

Some recent works on motion prediction prefer Gated Recurrent Units (GRU) over LSTMs for their lower parameter count, but our empirical performance comparisons favored LSTMs over GRUs.

5.6 Limitations and future work

A more informative way of representing the current offset to the target $\mathbf{o}_t$ would be to include positional-offsets in the representation. For this to be informative however, it would need to rely on character-local or global positions, which require FK. Although it is possible to perform FK inside the network at every step of generation, the backward pass during training becomes prohibitively slow justifying our use of root offset and rotational offsets only.

As with many data-driven approaches, our method struggles to generate transitions for which conditions are unrealistic, or outside the range covered by the training set.

Our scheduled target noise allows us to modify to some extent the manner in which a character reaches its target, reminiscent of changing the style of the motion, but doesn't allow yet to have control over those variations. Style control given a fixed context would be very interesting but is out of scope of this work.

5.7 Conclusion

In this work we first showed that state-of-the-art motion predictors cannot be converted into robust transition generators by simply adding conditioning information about the target keyframe. We proposed a time-to-arrival embedding modifier to allow robustness to transition lengths, and a scheduled target noise modifier to allow robustness to target keyframe variations and to enable sampling capabilities in the system. We showed how such a system allows animators to quickly generate quality motion between sparse keyframes inside an animation software. We also presented LaFAN1, a new high quality dataset well suited for transition generation benchmarking.

Acknowledgments

We thank Ubisoft Montreal, the Natural Sciences and Engineering Research Council of Canada and Mitacs for their support. We also thank Daniel Holden, Julien Roy, Paul Barde, Marc-André Carbonneau and Olivier Pomarez for their support and valuable feedback.

CHAPTER 6 GENERAL DISCUSSION

In the articles presented here, we aimed at leveraging the very high quality of MOCAP data as well as its recent availability in significant quantities in order to apply data-driven approaches to different motion modeling tasks. Our main motivations stem from the fact that despite its quality, motion capture is very expensive and time-consuming to produce and process. Therefore, our objectives were twofold. First, we investigated novel ways to facilitate reuse of MOCAP data through automatic motion classification. Second, we tried to bring MOCAP quality into keyframe animation while reducing animators' workload with automatic motion in-filling. Both these tasks aimed at reducing the reliance on novel MOCAP sequences in novel productions. We used a core architecture based on RNNs, and more precisely LSTMs to achieve our goals, and showed that such architectures, with the right data, training strategies and constraints could improve over previous approaches and be useful in production scenarios. We discuss here challenges and observations shared across these studies as well as interesting future directions. We also touch on the first tangible impacts of our work so far.

6.1 The Challenges of Naming Conventions for Action Recognition

In Chapter 3, our end goal was to foster reuse of motion sequences by automatically provide semantic keywords to sequences or sub-sequences of motion. Some datasets, that we leveraged, offer well separated action classes that are great for assessing model accuracy. However, in a real production environment such as at Ubisoft, to deploy our model and for it to be useful for production needs, a choice on actions of interest must be done. The task of designing a good taxonomy of movements for games is then critical, as it will define future available keywords to search by animators. Many discussions with professional animators were then needed and are still ongoing to determine the right representative labels to build a useful and complete taxonomy. We also quickly realized that imposing a single-class labeling system led to a combinatory explosion in the number labels needed as many descriptors could be used simultaneously at any moment (think of *crouching* while *shooting* while *walking*). Therefore, a better system can be built using a multi-class binary labeling approach allowing for multiple labels to be active at once, trained with binary cross-entropy.

6.2 RNNs and Sequential Computations

All of our techniques are based on RNNs and their capacity to model temporal dependencies in time series. Indeed, RNNs, and especially LSTMs are very powerful architectures that were successfully applied to a wide range of applications with serial data. Animation data, consisting of regularly sampled vectors of modest dimensions, is well suited for RNNs.

However, one fundamental limitation of RNNs is the sequential nature of their computations, making their training slower than other architectures. Indeed, in order to learn temporal dependencies, RNNs must process whole sequences of data and learn good recurrent weights. This means that for any sequence, all steps must be seen in series and cannot be processed in parallel. The backpropagation of error must also be done through the whole sequence, making each weight update relatively slow. This is not a surprise, as it is in the architecture's design, but can be somewhat limiting, since in our case it prevented us from using all of the GPUs' parallelism capacity. When using multiple recurrent layers, one can alleviate this problem by splitting those layers across GPUs and offsetting the computations by one timestep on each of those [28]. Yet, simply switching to a different feed-forward architecture, such as a WaveNet [67] or other convolutional architecture can be tempting as it allows for faster updates and potentially faster training. This can thus lead to faster design iterations, which are often critical in practice for deep learning. These architectures however are often trained with an implicit teacher forcing strategy as during training the model is never exposed to its own predictions. This is a condition to avoid sequential processing of sequences. In such cases, a high predictive accuracy is necessary in order to stay close to the training data and prevent divergence at runtime. Maybe this is why Transformer networks [35] have become more popular than recurrent architectures when the amount of available data passes a certain threshold; they allowed for faster training with great test time performance despite being trained only with teacher forcing. Another interesting framework in which the task of keyframe interpolation could be addressed is the denoising diffusion probabilistic models framework, in which image generation [61] and sound wave generation [221] have been tackled. This framework allows for generating the whole sequence of data at once in parallel, while denoising iterations are done on the previous approximation of the whole signal. This can lead to far less iterations per sequence and allows the network to model both forward and backward dependencies, making it a promising architecture for the task of conditional motion generation.

6.3 From Positions to Rotations

In Chapter 3 and Chapter 4, our input data representation and model outputs were based on positions only. We call this representation incomplete in that it is not sufficient to animate a skinned character as there is missing information about bone orientations in position vectors. Therefore, the first step we took to make the RTN usable in practice was to modify the output data representation to include the needed rotational information. We found it surprisingly hard to reach visually similar results through rotational outputs, and reaching satisfactory results took considerable time and effort. Indeed, it seems that in addition to containing more information, the highly non-linear dynamics of rotations are significantly harder to model than positions dynamics, which are linear or close to linear on wider timescales. This educated us on such challenges, leading us to be more critical to such aspects in the existing literature, and sparked interest in finding the right rotational representaion for deep learning on motion.

6.4 The *Chicken or the Egg* Situation of Data-Driven Motion Generation

During our work on transition generation for production use, we witnessed the paradox of the task at hand, where the data availability is inversely proportional to its importance. Most of the motion datasets available, including the high quality LaFAN1 database that we released, are dominated by simple locomotion. Even in cases of critical movements, there is usually basic locomotion during the initial placement of the actor and at the end of the sequence. This leads to the model performing very well on simple locomotion tasks, such as walking and running, but struggling more on rare movements, such as a rolling while shooting a gun. Unfortunately, the rare actions are often the hardest to animate - because animators lack the experience of animating such movements - and therefore the most useful to generate. As a result, the data-space for which the network performs best is the region of least interest, and vice-versa. In order to perform better in these critical cases, more similar data would be needed for training, therefore making it less necessary to generate, leading to a *chicken or the egg* type of situation. In the absence of such data however, other approaches might be investigated. First, a simple, but tedious approach would be to try to balance the training data, potentially without labels. This requires a lot of manual work, but could potentially be accelerated with clustering techniques, especially if at least some labels are available for semi-supervised clustering, as suggested in Chapter 3. Second, training-time techniques of sample selection could be investigated, in which data points for the next minibatch would be sampled with probabilities inversely proportional to some score function that evolves through

time.

6.5 Neural Networks and the Problem of Style Control

In tasks related to artistic content creation, especially in production settings with many constraints, it is preferable, although a challenge, that ML systems offer various levels of control. In Chapter 5, our system allows for re-sampling transitions for a given context, with a stochasticity control knob. Although this is useful to find subtle variations for any in-between, animators cannot specify any control parameter to guide such variations. This is a limitation of the approach, as we realized that in animation for high-end games, even when generated transitions are good, they might not correspond to the style of movement that the animator had in mind. Without finer control than with additional keyframes, artistic control is therefore too limited. Differentiating style from content is an interesting question that has been studied with considerable success in the image domain with the formulation of style transfer [175, 222]. While similar algorithms have been applied to animation [163, 179], they require a style exemplar every time the model generates an animation, which can be tedious or impossible for animators. Another, simpler approach could be to augment the inputs with style-label information to be used in a supervised manner. This could also be done using labels mapped to learned centroids of style representations (as in [179]) to modify a somewhat neutral motion given by a generic transition generator. However style specification may still not be enough as a control parameter for animators. Handling incomplete keyframes could allow the model to use any partial information such as feet or hands positions that do not require additional keyframes from animators. This of course would have other implications on the workflow and front-end of the plugin. Finally, the ultimate goal would be to learn a disentangled latent subspace of motion, in which few dimensions correspond to most semantic aspects of motions and could be controlled by the user. This is still an open problem and active research area in deep learning.

6.6 Impact of the Work on Semi-Supervised Classification

Our first article, published in 2018 showed how recurrent encoder-decoder networks could be trained in a semi-supervised manner on animation data. We used this paradigm to improve action recognition performance on a relatively small labeled dataset. This RNN-based approach has so far partially inspired further work on RNN-based action recognition [189], and more generally deep modeling of motion [223, 224].

Similarly at Ubisoft La Forge, results of our approach have fostered an effort on semi-

supervised action recognition for data reuse that is still active to this day. However, to better handle semi-supervised learning on long sequences of multiple actions, the method was first adapted by handling sliding windows of fixed context to provide labels and reconstructions on such windows. The core architecture was later switched to a semi-supervised CNN in order to optimize parallelism and iteration time. Our clustering results (Figure 6.1, LEFT) inspired a prototype for data exploration, label visualization and bulk label editing (Figure 6.1, RIGHT).

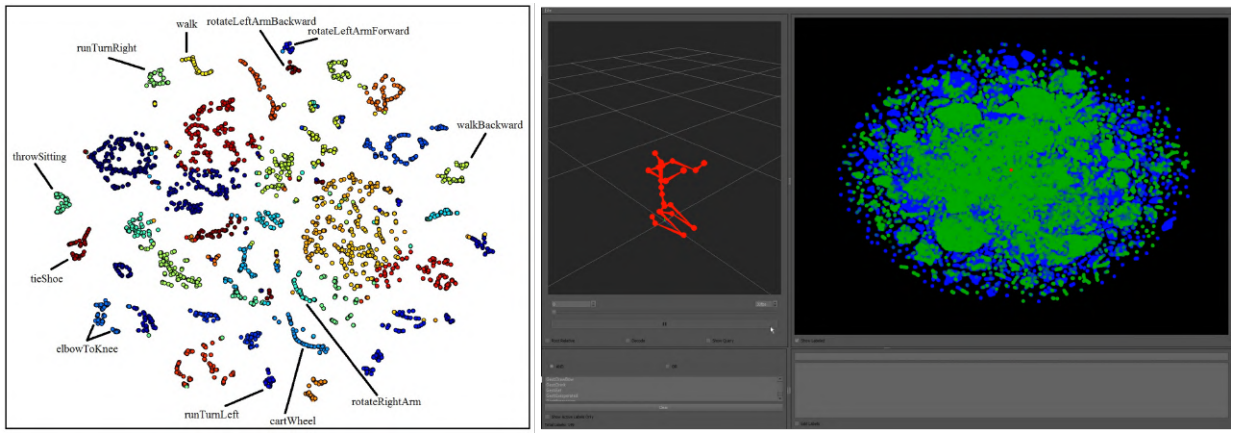


Figure 6.1 **Comparison of cluster visualizations.** LEFT: Our clustering results, taken from Chapter 3. RIGHT: Prototype interface and clustering results of the semi-supervised approach at Ubisoft La Forge, scaled to over 500 000 samples of one-second clips. Green points are labeled, blue points are unlabeled. This prototype offers real-time exploration capabilities through zooming and translating in the 2D data space.

So even though the final architecture used at Ubisoft La Forge differs from our RNN-based model, we are glad that it could spark interest in modeling motion with deep neural networks as well as automatic classification and clustering through semi-supervised learning.

6.7 Impact of the Work on Transition Generation.

Our work on transition generation, mostly from Chapter 4 and Chapter 5 showed how highly performing, deep recurrent motion predictors can be re-purposed for key-frame interpolation. Our proposed approaches allow animators to leverage MOCAP quality data with reduced efforts by specifying sparse keyframes. This work is quite recent, especially our full article on the subject (Chapter 5, published in August 2020) and follow-up work is still scarce, although the RTN was cited in other work on motion interpolation [216, 225], physically-

simulated character control [226], kinematic control [169, 227] and others. We also hope that the public release of our high quality motion dataset, with its related benchmark will facilitate future work and comparisons on the task of transition generation.

Our latest work on motion in-betweening, enabled in practical uses through our Motion Builder plugin, is now an available plugin to all Ubisoft studios. Although the results can be limited in terms of range (see Section 6.4) and control (Section 6.5), our plugin can be used to quickly produce prototype animations, link existing clips, implicitly assess the plausibility of keyframes or quickly produce placeholder animations in early stages of production. Critically, it allows non-technical animators to provide feedback on the results, suggest features, and indentify data region deficiencies in the training set. We are glad to have established this feedback loop that can open the door to user-oriented future iterations. This will be necessary in order to push the quality of such a system in order to produce production-ready animations for a considerable range of motion.

Finally, we are also glad that our work was chosen to be featured on the popular YouTube channel *Two Minute Papers*[1], which highlights interesting work in the fields of computer graphics, physics simulation, computer vision and others. This shall further promote our work and increase its impact.

[1]https://www.youtube.com/watch?v=mb6WJ34xQXg&ab_channel=TwoMinutePapers

REFERENCES

[1] T. B. Brown *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[3] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[4] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[5] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[7] K. Steinbuch and U. A. Piske, "Learning matrices and their applications," *IEEE Transactions on Electronic Computers*, no. 6, pp. 846–862, 1963.

[8] G. Ostrovskii, Y. M. Volin, and W. Borisov, "Uber die berechnung von ableitungen," *Wissenschaftliche Zeitschrift der Technischen Hochschule f ur Chemie, Leuna-Merseburg*, vol. 13, no. 4, pp. 382–384, 1971.

[9] S. Linnainmaa, "Taylor expansion of the accumulated rounding error," *BIT Numerical Mathematics*, vol. 16, no. 2, pp. 146–160, 1976.

[10] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Cited on*, vol. 14, no. 8, 2012.

[11] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[12] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, pp. 265–283.

[13] A. Paszke *et al.*, "Automatic differentiation in pytorch," 2017.

[14] J. Deng *et al.*, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[16] K. Fukushima, S. Miyake, and T. Ito, "Neocognitron: A neural network model for a mechanism of visual pattern recognition," *IEEE transactions on systems, man, and cybernetics*, no. 5, pp. 826–834, 1983.

[17] Y. LeCun *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[18] S. Hooker, "The hardware lottery," *arXiv preprint arXiv:2009.06489*, 2020.

[19] G. Litjens *et al.*, "A survey on deep learning in medical image analysis," *Medical image analysis*, vol. 42, pp. 60–88, 2017.

[20] C. Badue *et al.*, "Self-driving cars: A survey," *Expert Systems with Applications*, p. 113816, 2020.

[21] V. Vavilala and M. Meyer, "Deep learned super resolution for feature film production," in *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks*, 2020, pp. 1–2.

[22] M. Seymour, "Ink lines and machine learning," 2019. [Online]. Available: https://www.fxguide.com/fxfeatured/ink-lines-and-machine-learning/

[23] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[24] A. Graves, "Supervised sequence labelling," in *Supervised sequence labelling with recurrent neural networks*. Springer, 2012, pp. 5–13.

[25] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[26] A. Graves *et al.*, "Unconstrained on-line handwriting recognition with recurrent neural networks," in *Advances in Neural Information Processing Systems*, 2008, pp. 577–584.

[27] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[28] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[29] K. Cho *et al.*, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[30] A. Graves, N. Jaitly, and A.-r. Mohamed, "Hybrid speech recognition with deep bidirectional lstm," in *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*. IEEE, 2013, pp. 273–278.

[31] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 1764–1772.

[32] J. Martinez, M. J. Black, and J. Romero, "On human motion prediction using recurrent neural networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 4674–4683.

[33] K. Fragkiadaki *et al.*, "Recurrent network models for human dynamics," in *Computer Vision (ICCV), 2015 IEEE International Conference on*. IEEE, 2015, pp. 4346–4354.

[34] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[35] A. Vaswani *et al.*, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[36] A. Radford *et al.*, "Improving language understanding with unsupervised learning," *Technical report, OpenAI*, 2018.

[37] J. Devlin *et al.*, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[38] W. L. Hamilton, "Graph representation learning," *Synthesis Lectures on Artifical Intelligence and Machine Learning*, vol. 14, no. 3, pp. 1–159, 2020.

[39] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[40] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[41] M. Zitnik, M. Agrawal, and J. Leskovec, "Modeling polypharmacy side effects with graph convolutional networks," *Bioinformatics*, vol. 34, no. 13, pp. i457–i466, 2018.

[42] A. Bordes *et al.*, "Translating embeddings for modeling multi-relational data," in *Advances in neural information processing systems*, 2013, pp. 2787–2795.

[43] I. Goodfellow *et al.*, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2014, pp. 2672–2680.

[44] C. Ledig *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4681–4690.

[45] S. Iizuka, E. Simo-Serra, and H. Ishikawa, "Globally and locally consistent image completion," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 107, 2017.

[46] J. Engel *et al.*, "Gansynth: Adversarial neural audio synthesis," 2019. [Online]. Available: https://openreview.net/pdf?id=H1xQVn09FX

[47] J.-Y. Zhu *et al.*, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[48] H. Zhang *et al.*, "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5907–5915.

[49] A. Hernandez, J. Gall, and F. Moreno-Noguer, "Human motion prediction via spatio-temporal inpainting," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 7134–7143.

[50] T. Karras *et al.*, "Analyzing and improving the image quality of stylegan," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8110–8119.

[51] M. Müller *et al.*, "Documentation mocap database hdm05," 2007.

[52] A. Shahroudy *et al.*, "Ntu rgb+d: A large scale dataset for 3d human activity analysis," in *IEEE Conference on Computer Vision and Pattern Recognition*, June 2016.

[53] D. Pavllo *et al.*, "Modeling human motion with quaternion-based neural networks," *International Journal of Computer Vision*, pp. 1–18, 2019.

[54] A. Gopalakrishnan *et al.*, "A neural temporal model for human motion prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 116–12 125.

[55] H.-k. Chiu *et al.*, "Action-agnostic human pose forecasting," in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2019, pp. 1423–1432.

[56] C. Dos Santos and M. Gatti, "Deep convolutional neural networks for sentiment analysis of short texts," in *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, 2014, pp. 69–78.

[57] N. Srivastava, E. Mansimov, and R. Salakhutdinov, "Unsupervised learning of video representations using lstms," *arXiv preprint arXiv:1502.04681*, 2015.

[58] I. Goodfellow. (2017, June) Generative models 1. Deep Learning Summer School 2017. [Online]. Available: https://www.vicon.com/products/software/

[59] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[60] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows," *arXiv preprint arXiv:1505.05770*, 2015.

[61] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *arXiv preprint arXiv:2006.11239*, 2020.

[62] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," *arXiv preprint arXiv:1601.06759*, 2016.

[63] S. Bengio *et al.*, "Scheduled sampling for sequence prediction with recurrent neural networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 1171–1179.

[64] A. M. Lamb *et al.*, "Professor forcing: A new algorithm for training recurrent networks," in *Advances in neural information processing systems*, 2016, pp. 4601–4609.

[65] Z. Li *et al.*, "Auto-conditioned lstm network for extended complex human motion synthesis," *arXiv preprint arXiv:1707.05363*, 2017.

[66] A. Van den Oord *et al.*, "Conditional image generation with pixelcnn decoders," in *Advances in neural information processing systems*, 2016, pp. 4790–4798.

[67] A. v. d. Oord *et al.*, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[68] D. Holden, T. Komura, and J. Saito, "Phase-functioned neural networks for character control," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 42, 2017.

[69] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[70] I. Gulrajani *et al.*, "Improved training of wasserstein gans," in *Advances in neural information processing systems*, 2017, pp. 5767–5777.

[71] X. Mao *et al.*, "Least squares generative adversarial networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2794–2802.

[72] D. Holden, "Robust solving of optical motion capture data by denoising," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, pp. 1–12, 2018.

[73] Y. Zhou *et al.*, "On the continuity of rotation representations in neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5745–5753.

[74] H. Zhang *et al.*, "Mode-adaptative neural networks for quadruped motion control," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, 2018.

[75] A. Witkin and M. Kass, "Spacetime constraints," in *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '88. New York, NY, USA: ACM, 1988, pp. 159–168. [Online]. Available: http://doi.acm.org/10.1145/54852.378507

[76] F. Thomas, O. Johnston, and F. Thomas, *The illusion of life: Disney animation*. Hyperion New York, 1995.

[77] J. K. Hodgins *et al.*, "Animating human athletics," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, 1995, pp. 71–78.

[78] J. Lee *et al.*, "Interactive control of avatars animated with human motion data," in *ACM Transactions on Graphics (ToG)*, vol. 21, no. 3. ACM, 2002, pp. 491–500.

[79] V. B. Zordan and J. K. Hodgins, "Motion capture-driven simulations that hit and react," in *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2002, pp. 89–96.

[80] K. Yin, K. Loken, and M. van de Panne, "Simbicon: Simple biped locomotion control," in *ACM Transactions on Graphics (TOG)*, vol. 26. ACM, 2007, p. 105.

[81] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," in *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, 2002, pp. 473–482.

[82] S. Starke *et al.*, "Neural state machine for character-scene interactions," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 6, pp. 1–14, 2019.

[83] H. Y. Ling *et al.*, "Character controllers using motion vaes," vol. 39, no. 4, 2020.

[84] K. Bergamin *et al.*, "Drecon: data-driven responsive control of physics-based characters," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 6, pp. 1–11, 2019.

[85] E. Barsoum, J. Kender, and Z. Liu, "Hp-gan: Probabilistic 3d human motion prediction via gan," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 1418–1427.

[86] E. Corona *et al.*, "Context-aware human motion prediction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 6992–7001.

[87] A. Jain *et al.*, "Structural-rnn: Deep learning on spatio-temporal graphs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5308–5317.

[88] P. Ghosh *et al.*, "Learning human motion models for long-term predictions," *arXiv preprint arXiv:1704.02827*, 2017.

[89] Z. Liu *et al.*, "Towards natural and accurate future motion prediction of humans and animals," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[90] T. Salimans *et al.*, "Improved techniques for training gans," in *Advances in neural information processing systems*, 2016, pp. 2234–2242.

[91] C. Ionescu *et al.*, "Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 7, pp. 1325–1339, jul 2014.

[92] C. S. Catalin Ionescu, Fuxin Li, "Latent structured models for human pose estimation," in *International Conference on Computer Vision*, 2011.

[93] W. Li, Z. Zhang, and Z. Liu, "Expandable data-driven graphical modeling of human actions based on salient postures," *IEEE transactions on Circuits and Systems for Video Technology*, vol. 18, no. 11, pp. 1499–1510, 2008.

[94] F. Lv and R. Nevatia, "Single view human action recognition using key pose matching and viterbi path searching," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2007, pp. 1–8.

[95] J. C. Niebles, H. Wang, and L. Fei-Fei, "Unsupervised learning of human action categories using spatial-temporal words," *International journal of computer vision*, vol. 79, no. 3, pp. 299–318, 2008.

[96] A. F. Bobick and J. W. Davis, "The recognition of human movement using temporal templates," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 23, no. 3, pp. 257–267, 2001.

[97] J. Yamato, J. Ohya, and K. Ishii, "Recognizing human action in time-sequential images using hidden markov model." in *CVPR*, vol. 92, 1992, pp. 379–385.

[98] C. Schuldt, I. Laptev, and B. Caputo, "Recognizing human actions: a local svm approach," in *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, vol. 3. IEEE, 2004, pp. 32–36.

[99] I. Laptev *et al.*, "Learning realistic human actions from movies," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2008, pp. 1–8.

[100] H. Kuehne *et al.*, "Hmdb: a large video database for human motion recognition," in *2011 International Conference on Computer Vision*. IEEE, 2011, pp. 2556–2563.

[101] K. Soomro, A. R. Zamir, and M. Shah, "Ucf101: A dataset of 101 human actions classes from videos in the wild," *arXiv preprint arXiv:1212.0402*, 2012.

[102] S. Abu-El-Haija *et al.*, "Youtube-8m: A large-scale video classification benchmark," *arXiv preprint arXiv:1609.08675*, 2016.

[103] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Advances in neural information processing systems*, 2014, pp. 568–576.

[104] J. Donahue *et al.*, "Long-term recurrent convolutional networks for visual recognition and description," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 2625–2634.

[105] D. Tran *et al.*, "Learning spatiotemporal features with 3d convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 4489–4497.

[106] L. Yao *et al.*, "Describing videos by exploiting temporal structure," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 4507–4515.

[107] J. Wang *et al.*, "Mining actionlet ensemble for action recognition with depth cameras," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 1290–1297.

[108] W. Li, Z. Zhang, and Z. Liu, "Action recognition based on a bag of 3d points," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Workshops*. IEEE, 2010, pp. 9–14.

[109] J. Wang *et al.*, "Robust 3d action recognition with random occupancy patterns," in *European Conference on Computer Vision*. Springer, 2012, pp. 872–885.

[110] X. Yang, C. Zhang, and Y. Tian, "Recognizing actions using depth motion maps-based histograms of oriented gradients," in *Proceedings of the 20th ACM international conference on Multimedia*, 2012, pp. 1057–1060.

[111] O. Oreifej and Z. Liu, "Hon4d: Histogram of oriented 4d normals for activity recognition from depth sequences," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2013, pp. 716–723.

[112] J. Shotton *et al.*, "Real-time human pose recognition in parts from single depth images," in *CVPR 2011*. Ieee, 2011, pp. 1297–1304.

[113] L. Xia, C.-C. Chen, and J. K. Aggarwal, "View invariant human action recognition using histograms of 3d joints," in *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. IEEE, 2012, pp. 20–27.

[114] X. Yang and Y. L. Tian, "Eigenjoints-based action recognition using naive-bayes-nearest-neighbor," in *2012 IEEE computer society conference on computer vision and pattern recognition workshops*. IEEE, 2012, pp. 14–19.

[115] M. Barnachon *et al.*, "Ongoing human action recognition with motion capture," *Pattern Recognition*, vol. 47, no. 1, pp. 238–247, 2014.

[116] M. E. Hussein *et al.*, "Human action recognition using a temporal hierarchy of covariance descriptors on 3d joint locations," in *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[117] R. Chaudhry *et al.*, "Bio-inspired dynamic 3d discriminative skeletal features for human action recognition," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2013 IEEE Conference on*. IEEE, 2013, pp. 471–478.

[118] G. Evangelidis, G. Singh, and R. Horaud, "Skeletal quads: Human action recognition using joint quadruples," in *2014 22nd International Conference on Pattern Recognition*. IEEE, 2014, pp. 4513–4518.

[119] E. P. Ijjina *et al.*, "Classification of human actions using pose-based features and stacked auto encoder," *Pattern Recognition Letters*, 2016.

[120] S. Zhang, X. Liu, and J. Xiao, "On geometric features for skeleton-based action recognition using multilayer lstm networks," in *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*. IEEE, 2017, pp. 148–157.

[121] Q. Ke *et al.*, "A new representation of skeleton sequences for 3d action recognition," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 4570–4579.

[122] K. Cho and X. Chen, "Classifying and visualizing motion capture sequences using deep neural networks," in *Computer Vision Theory and Applications (VISAPP), 2014 International Conference on*, vol. 2. IEEE, 2014, pp. 122–130.

[123] Y. Du, W. Wang, and L. Wang, "Hierarchical recurrent neural network for skeleton based action recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1110–1118.

[124] R. Vemulapalli, F. Arrate, and R. Chellappa, "Human action recognition by representing 3d skeletons as points in a lie group," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 588–595.

[125] R. Vemulapalli and R. Chellapa, "Rolling rotations for recognizing human actions from 3d skeletal data," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4471–4479.

[126] F. Ofli *et al.*, "Sequence of the most informative joints (smij): A new representation for human skeletal action recognition," *Journal of Visual Communication and Image Representation*, vol. 25, no. 1, pp. 24–38, 2014.

[127] O. Arikan and D. A. Forsyth, "Interactive motion generation from examples," in *ACM Transactions on Graphics (TOG)*, vol. 21, no. 3. ACM, 2002, pp. 483–490.

[128] Y. Li, T. Wang, and H.-Y. Shum, "Motion texture: a two-level statistical model for character motion synthesis," in *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, 2002, pp. 465–472.

[129] P. Beaudoin *et al.*, "Motion-motif graphs," in *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Eurographics Association, 2008, pp. 117–126.

[130] A. Safonova and J. K. Hodgins, "Construction and optimal search of interpolated motion graphs," in *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3. ACM, 2007, p. 106.

[131] J. Chai and J. K. Hodgins, "Performance animation from low-dimensional control signals," in *ACM Transactions on Graphics (ToG)*, vol. 24, no. 3. ACM, 2005, pp. 686–696.

[132] J. Tautges *et al.*, "Motion reconstruction using sparse accelerometer data," *ACM Transactions on Graphics (ToG)*, vol. 30, no. 3, p. 18, 2011.

[133] M. Büttner and S. Clavet, "Motion matching - the road to next gen animation," in *Proc. of Nucl.ai 2015*, 2015. [Online]. Available: https://www.youtube.com/watch?v=z_wpgHFSWss&t=658s

[134] Y. Lee *et al.*, "Motion fields for interactive character locomotion," in *ACM Transactions on Graphics (TOG)*, vol. 29. ACM, 2010, p. 138.

[135] S. Clavet, "Motion matching and the road to next-gen animation," in *Proc. of GDC 2016*, 2016.

[136] F. Zinno, "Ml tutorial day: From motion matching to motion synthesis, and all the hurdles in between," in *Proc. of GDC 2019*, 2019.

[137] M. Büttner, "Machine learning for motion synthesis and character control in games," in *Proc. of i3D 2019*, 2019.

[138] C. K. Liu, A. Hertzmann, and Z. Popović, "Learning physics-based motion style with nonlinear inverse optimization," *ACM Transactions on Graphics (TOG)*, vol. 24, no. 3, pp. 1071–1081, 2005.

[139] G. W. Taylor, G. E. Hinton, and S. T. Roweis, "Modeling human motion using binary latent variables," in *Advances in neural information processing systems*, 2007, pp. 1345–1352.

[140] J. Chai and J. K. Hodgins, "Constraint-based motion optimization using a statistical dynamic model," *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3, p. 8, 2007.

[141] J. Min, Y.-L. Chen, and J. Chai, "Interactive generation of human animation with deformable motion models," *ACM Transactions on Graphics (TOG)*, vol. 29, no. 1, p. 9, 2009.

[142] N. D. Lawrence, "Gaussian process latent variable models for visualisation of high dimensional data," in *Advances in neural information processing systems*, 2004, pp. 329–336.

[143] K. Grochow *et al.*, "Style-based inverse kinematics," in *ACM transactions on graphics (TOG)*, vol. 23, no. 3. ACM, 2004, pp. 522–531.

[144] J. M. Wang, D. J. Fleet, and A. Hertzmann, "Gaussian process dynamical models for human motion," *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 2, pp. 283–298, 2008.

[145] Y. Ye and C. K. Liu, "Synthesis of responsive motion using a dynamic model," in *Computer Graphics Forum*, vol. 29, no. 2. Wiley Online Library, 2010, pp. 555–562.

[146] S. Levine *et al.*, "Continuous character control with low-dimensional embeddings," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, p. 28, 2012.

[147] J. Min and J. Chai, "Motion graphs++: a compact generative model for semantic motion analysis and synthesis," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 6, p. 153, 2012.

[148] Y.-C. Pai and J. Patton, "Center of mass velocity-position predictions for balance control," *Journal of biomechanics*, vol. 30, no. 4, pp. 347–354, 1997.

[149] V. B. Zordan *et al.*, "Dynamic response for motion capture animation," *ACM Transactions on Graphics (TOG)*, vol. 24, no. 3, pp. 697–701, 2005.

[150] K. W. Sok, M. Kim, and J. Lee, "Simulating biped behaviors from human motion data," in *ACM SIGGRAPH 2007 papers*, 2007, pp. 107–es.

[151] L. Liu *et al.*, "Sampling-based contact-rich motion control," in *ACM SIGGRAPH 2010 papers*, 2010, pp. 1–10.

[152] L. Liu, K. Yin, and B. Guo, "Improving sampling-based motion control," in *Computer Graphics Forum*, vol. 34, no. 2. Wiley Online Library, 2015, pp. 415–423.

[153] J. M. Wang *et al.*, "Optimizing locomotion controllers using biologically-based actuators and objectives," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, pp. 1–11, 2012.

[154] Y. Lee *et al.*, "Locomotion control for many-muscle humanoids," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 6, pp. 1–11, 2014.

[155] S. Lee *et al.*, "Scalable muscle-actuated human simulation and control," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–13, 2019.

[156] T. Geijtenbeek, M. Van De Panne, and A. F. Van Der Stappen, "Flexible muscle-based locomotion for bipedal creatures," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 6, pp. 1–11, 2013.

[157] J. Lee and K. H. Lee, "Precomputing avatar behavior from human motion data," *Graphical Models*, vol. 68, no. 2, pp. 158–174, 2006.

[158] A. Treuille, Y. Lee, and Z. Popović, "Near-optimal character animation with continuous control," *ACM Transactions on Graphics (tog)*, vol. 26, no. 3, p. 7, 2007.

[159] S. Coros, P. Beaudoin, and M. van de Panne, "Robust task-based control policies for physics-based characters," in *ACM Transactions on Graphics (TOG)*, vol. 28. ACM, 2009, p. 170.

[160] L. Liu, M. V. D. Panne, and K. Yin, "Guided learning of control graphs for physics-based characters," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 3, pp. 1–14, 2016.

[161] X. B. Peng, G. Berseth, and M. Van de Panne, "Dynamic terrain traversal skills using reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, pp. 1–11, 2015.

[162] D. Holden *et al.*, "Learning motion manifolds with convolutional autoencoders," in *SIGGRAPH Asia 2015 Technical Briefs*. ACM, 2015, p. 18.

[163] D. Holden, J. Saito, and T. Komura, "A deep learning framework for character motion synthesis and editing," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 138, 2016.

[164] L.-Y. Gui *et al.*, "Adversarial geometry-aware human motion prediction," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 786–803.

[165] Y. Tang *et al.*, "Long-term human motion prediction by modeling motion context and enhancing motion dynamic," *arXiv preprint arXiv:1805.02513*, 2018.

[166] J. Butepage *et al.*, "Deep representation learning for human motion prediction and classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 6158–6166.

[167] X. Zhang and M. van de Panne, "Data-driven autocompletion for keyframe animation," in *Proceedings of the 11th Annual International Conference on Motion, Interaction, and Games*. ACM, 2018, p. 10.

[168] K. Lee, S. Lee, and J. Lee, "Interactive character animation by learning multi-objective control," in *SIGGRAPH Asia 2018 Technical Papers*. ACM, 2018, p. 180.

[169] D. Holden *et al.*, "Learned motion matching," vol. 39, no. 4, 2020.

[170] K. Zadziuk, "Motion matching, the future of games animation... today," in *Proc. of GDC 2016*, 2016.

[171] X. B. Peng, G. Berseth, and M. Van de Panne, "Terrain-adaptive locomotion skills using deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, pp. 1–12, 2016.

[172] X. B. Peng *et al.*, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 41, 2017.

[173] L. Liu and J. Hodgins, "Learning to schedule control fragments for physics-based characters using deep q-learning," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 3, p. 29, 2017.

[174] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[175] V. Dumoulin, J. Shlens, and M. Kudlur, "A learned representation for artistic style," *ICLR*, 2017. [Online]. Available: https://arxiv.org/abs/1610.07629

[176] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[177] E. Perez *et al.*, "Film: Visual reasoning with a general conditioning layer," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[178] X. Huang and S. Belongie, "Arbitrary style transfer in real-time with adaptive instance normalization," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1501–1510.

[179] K. Aberman *et al.*, "Unpaired motion style transfer from video to animation," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 4, p. 64, 2020.

[180] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019, pp. 4401–4410.

[181] D. Erhan *et al.*, "Why does unsupervised pre-training help deep learning?" *The Journal of Machine Learning Research*, vol. 11, pp. 625–660, 2010.

[182] Y. Bengio *et al.*, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems*, vol. 19, p. 153, 2007.

[183] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[184] D. Yu, L. Deng, and G. Dahl, "Roles of pre-training and fine-tuning in context-dependent dbn-hmms for real-world speech recognition," in *Proc. NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

[185] W. Zhu *et al.*, "Co-occurrence feature learning for skeleton based action recognition using regularized deep lstm networks," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[186] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 6645–6649.

[187] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," *arXiv preprint arXiv:1402.1128*, 2014.

[188] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 1017–1024.

[189] J. Liu *et al.*, "Skeleton-based action recognition using spatio-temporal lstm network with trust gates," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[190] ——, "Global context-aware attention lstm networks for 3d action recognition," in *CVPR*, 2017.

[191] T. M. Breuel, "Benchmarking of lstm networks," *arXiv preprint arXiv:1508.02774*, 2015.

[192] B. Mahasseni and S. Todorovic, "Regularizing long short term memory with 3d human-skeleton sequences for action recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3054–3062.

[193] C.-C. Hung, E. T. Carlson, and C. E. Connor, "Medial axis shape coding in macaque inferotemporal cortex," *Neuron*, vol. 74, no. 6, pp. 1099–1113, 2012.

[194] X. Chen and M. Koskela, "Classification of rgb-d and motion capture sequences using extreme learning machine," in *Image Analysis*. Springer, 2013, pp. 640–651.

[195] S. F. Cotter *et al.*, "Sparse solutions to linear inverse problems with multiple measurement vectors," *IEEE Transactions on Signal Processing*, vol. 53, no. 7, pp. 2477–2488, 2005.

[196] Q. Ke *et al.*, "Skeletonnet: Mining deep part features for 3-d action recognition," *IEEE signal processing letters*, vol. 24, no. 6, pp. 731–735, 2017.

[197] S. Song *et al.*, "An end-to-end spatio-temporal attention model for human action recognition from skeleton data." in *AAAI*, vol. 1, no. 2, 2017, p. 7.

[198] S. Reed *et al.*, "Generative adversarial text to image synthesis," *arXiv preprint arXiv:1605.05396*, 2016.

[199] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[200] X. Huang *et al.*, "Stacked generative adversarial networks," *arXiv preprint arXiv:1612.04357*, 2016.

[201] A. Ghosh *et al.*, "Contextual rnn-gans for abstract reasoning diagram generation," *arXiv preprint arXiv:1609.09444*, 2016.

[202] B. Mahasseni, M. Lam, and S. Todorovic, "Unsupervised video summarization with adversarial lstm networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[203] L. Yu *et al.*, "Seqgan: sequence generative adversarial nets with policy gradient," *arXiv preprint arXiv:1609.05473*, 2016.

[204] A. M. Lehrmann, P. V. Gehler, and S. Nowozin, "Efficient nonlinear markov models for human motion," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1314–1321.

[205] Y. LeCun *et al.*, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 1998, pp. 9–50.

[206] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[207] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015, pp. 2342–2350.

[208] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[209] J.-F. Hu *et al.*, "Jointly learning heterogeneous features for rgb-d activity recognition," in *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. IEEE, 2015, pp. 5344–5352.

[210] T. T. D. Team *et al.*, "Theano: A python framework for fast computation of mathematical expressions," *arXiv preprint arXiv:1605.02688*, 2016.

[211] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 4565–4573.

[212] N. Baram, O. Anschel, and S. Mannor, "Model-based adversarial imitation learning," *arXiv preprint arXiv:1612.02179*, 2016.

[213] X. B. Peng *et al.*, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills," *ACM Transactions on Graphics (Proc. SIGGRAPH 2018 - to appear)*, vol. 37, no. 4, 2018.

[214] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," in *International Conference on Learning Representations*, 2018.

[215] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.

[216] L. Ciccone, C. Öztireli, and R. W. Sumner, "Tangent-space optimization for interactive animation control," *ACM Trans. Graph.*, vol. 38, no. 4, pp. 101:1–101:10, Jul. 2019. [Online]. Available: http://doi.acm.org/10.1145/3306346.3322938

[217] F. G. Harvey and C. Pal, "Recurrent transition networks for character locomotion," in *SIGGRAPH Asia 2018 Technical Briefs*. ACM, 2018, p. 4.

[218] M. Cohen *et al.*, "Efficient generation of motion transitions using spacetime constraints," in *SIGGRAPH 96*. Association for Computing Machinery, Inc., January 1996. [Online]. Available: https://www.microsoft.com/en-us/research/publication/efficient-generation-of-motion-transitions-using-spacetime-constraints/

[219] A. Nicolae, "Plu: The piecewise linear unit activation function," *arXiv preprint arXiv:1809.09534*, 2018.

[220] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=ryQu7f-RZ

[221] N. Chen *et al.*, "Wavegrad: Estimating gradients for waveform generation," *arXiv preprint arXiv:2009.00713*, 2020.

[222] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *arXiv preprint arXiv:1508.06576*, 2015.

[223] D. Holden *et al.*, "Fast neural style transfer for motion data," *IEEE computer graphics and applications*, vol. 37, no. 4, pp. 42–49, 2017.

[224] K. Xu *et al.*, "Greedy layer-wise training of long short term memory networks," in *2018 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*. IEEE, 2018, pp. 1–6.

[225] Y. Zhou *et al.*, "Generative tweening: Long-term inbetweening of 3d human motions," *arXiv preprint arXiv:2005.08891*, 2020.

[226] S. Park *et al.*, "Learning predict-and-simulate policies from unorganized human motion data," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 6, pp. 1–11, 2019.

[227] S. Starke *et al.*, "Local motion phases for learning multi-contact character movements," vol. 39, no. 4, 2020.

APPENDIX A SLIDING CRITICS

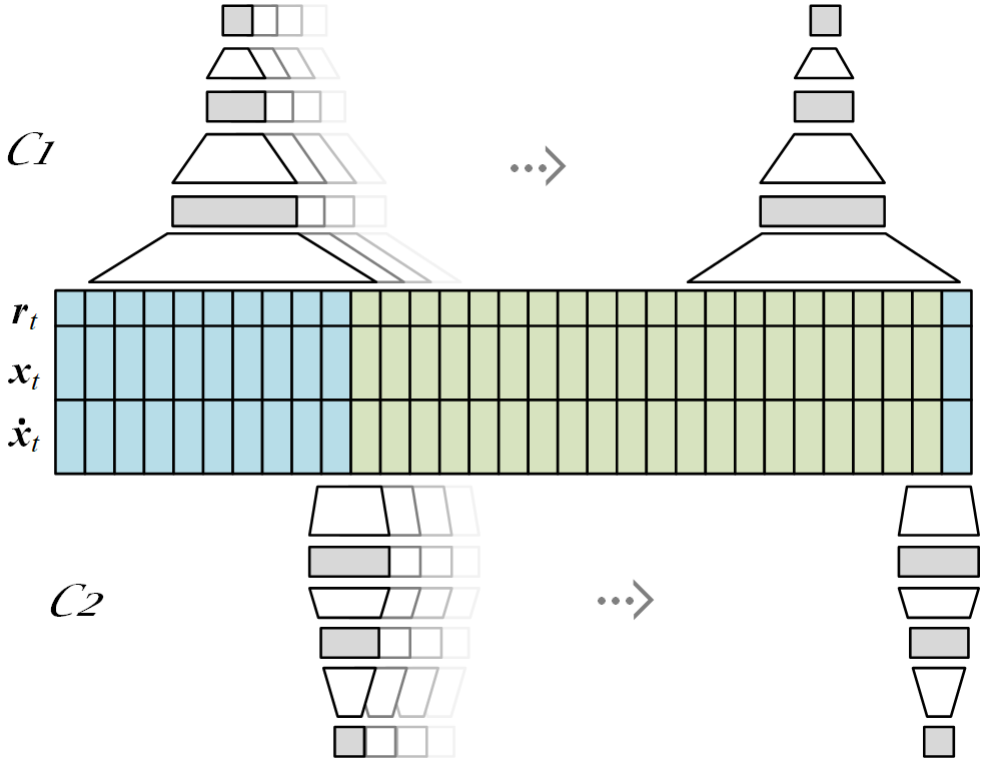


Figure A.1 **Visual summary of the two timescales critics.** Blue frames are the given contexts and green frames correspond to the transition. First and last critic positions are shown without transparency. At the beginning and end of transitions, the critics are conditional in that they include ground-truth context in their input sequences. Scalar scores at each timestep are averaged to get the final score.