Recurrent Semi-supervised Classification and Constrained Adversarial Generation with Motion Capture Data

Félix G. Harvey[a,b,*], Julien Roy[a,b], Christopher Pal[a,b]

[a] *Polytechnique Montréal,*
2900 Édouard-Montpetit blvd,
Montreal, H3T 1J4, Canada
[b] *The Montreal Institute for Learning Algorithms,*
2920 Chemin de la Tour,
Montreal, H3T 1J4, Canada

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. Our classification experiments are conducted using three well known skeletal motion datasets. We also explore a novel formulation for future predicting decoders based on conditional recurrent generative adversarial networks. We further 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 training for recurrent adversarial architectures for animation generation.

Keywords: Action Recognition, Motion Capture, Semi-supervised Learning, Recurrent Neural Networks, Generative Adversarial Networks, Transition Generation

*felix.gingras-harvey@polymtl.ca

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 [10, 2]. A common technique for semi-supervised learning is to perform training in two phases: unsupervised pre-training, followed by supervised fine tuning [21, 2, 10, 49]. 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 commencing 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 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. We demonstrate this improvement on HDM05 [37], a well known action recognition dataset. We also explore the limits of this method using the more recent NTU-RGB+D dataset [41]. 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 timesteps.

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 from a the same generator network trained without those constraints.

2. Related Work

2.1. Recurrent-Encoder-Decoders

RNNs have proven over the years to be very powerful models for sequential data, such as speech [20, 40, 18], handwriting [19], text [44, 17], or as in our case, MOCAP [9, 52]. We use LSTMs [22] without in-cell connections (as suggested by Breuel [3]) 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. [7] as well as Sutskever et al. [45] 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. [7]. 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. [43] in which they perform unsupervised learning, by either reconstructing the sequence, predicting the next frames, or both. In our work, an additional

decoder is used for classification of whole sequences, and the future generator uses an adversarial loss to improve generated sequences.

2.2. Generative Adversarial Networks

GANs [16] 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 [39, 38, 25], results on sequential data remains more limited. Ghosh et al. [14] 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. 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, the 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. [50] 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.

2.3. 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 [37]. It contains 2329 labeled cuts that are very well suited for action recognition. We use the same 65 classes defined by Cho and Chen [6]. 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

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

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. [52], Ijjina et al. [27], and Barnachon et al. [1] 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 [41] 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 contains over 56000 labeled sequences that may contain more than one subject. 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.

2.4. Action recognition

Much of the prior work on MOCAP analysis has been based on hand-crafted featrues. For example, Chaudhry et al. [4] created bio-inspired features based on the findings of Hung et al. [26] 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. [27] 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. take similar feature-based approaches. 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. [1] use a learnt 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.

End-to-end neural approaches have also been tried on HDM05 and CMU in which cases discriminating features are learnt throughout the training of

a neural network. Cho and Chen [6] have obtained good movement classification rates on simple sequences (cuts) on the HDM05 dataset using a Multi-Layer Perceptron (MLP)+Auto-encoder hybrid. Chen and Koskela [5] tested multiple types of features, using a fast technique they call 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. [9] treated the simple sequences' classification problem with the same action classes as Cho and Chen [6] 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. [52] have a similar, but less constrained recurrent architecture that is regularized by a weight penalty based on the $l_{2,1}$ norm (Cotter et al. [8]), which encourages parts of the network to focus 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 [30, 31, 48, 51] in order to inject domain knowledge directly into the inputs. Others propose instead new architectural variants to the neural networks for motion recognition [34, 42, 28, 12] 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 easily be combined with the above approaches or applied to different domains.

2.5. Transition Generation

Approaches have been proposed for motion prediction with well-designed recurrent neural networks [12, 36, 15]. While our method uses the same proposed LSTM decoder used for our reconstruction objective, these architectures could be applied with our proposed constrained adversarial learning strategy that stabilizes training. As we wish to generate transitions and not

only predicting the next frames, we therefore need to add conditioning on future context to allow reaching a desired positions. Our work in that manner is related to Lehrmann et al. [33] for which we add the difficulty of allowing variable-length transitions.

2.6. 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 1: Accuracies (Acc.) with different test sets, using techniques from Cho and Chen [6], Du et al. [9] and Zhu et al. [52]. 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**

Table 1 shows the results of our own implementation of previous state-of-the-art methods [6, 9, 52] 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, this 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 and Chen [6] 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 explained in Section 4.1 and its effects can be seen in in Figure 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 and Chen [6] and Zhu et al. [52] yielded the best results with our actor-based partitions and with our pre-processing method, the baseline test accuracy for the rest of this work will score 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 and Chen [6].

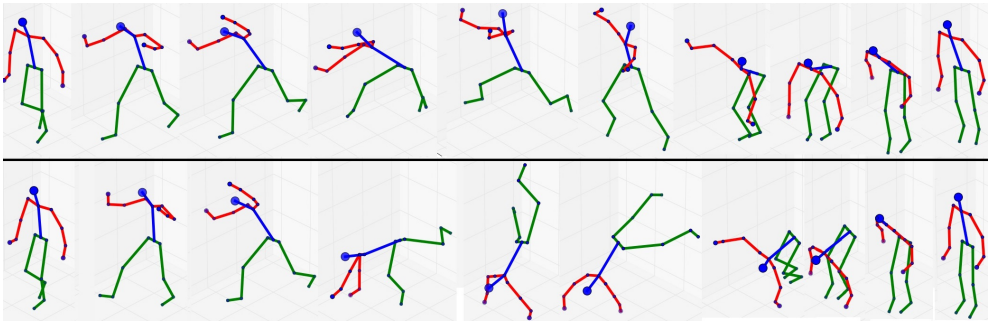

Figure 1: Visual comparison of pre-processing methods for a *cartWheel* movement from HDM05. UP: Same as Cho and Chen [6]. DOWN: our method, that allows for hips not to be parallel to the ground.

3. Our Models

3.1. Multi-Decoder Models

Figure 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

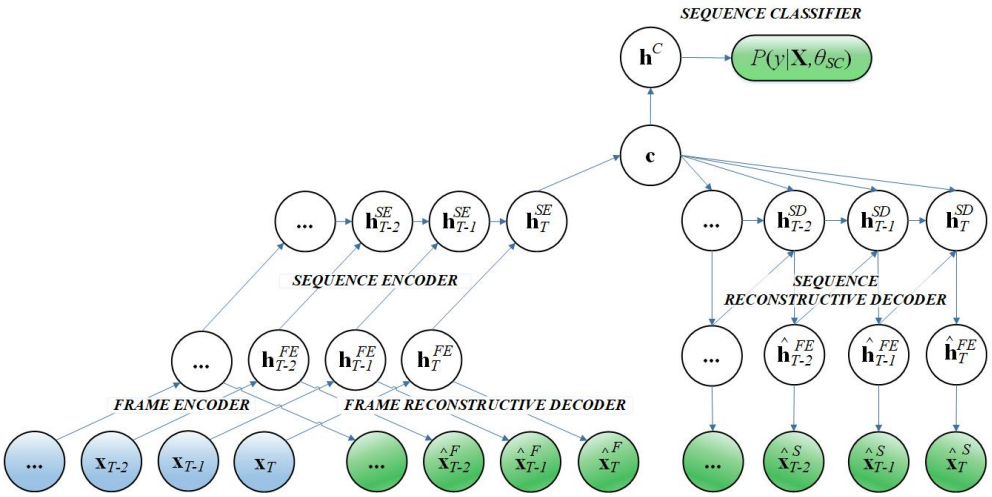

Figure 2: The FR-SRC variant of the architecture studied. This network produces 3 types of outputs (in green) with respect to a sequence $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_T]$ and its parameters θ. The set θ_{SC} includes all the weights and biases used to compute class probabilities. The hidden 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 with 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).

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. Each of these components are added to produce evermore meaningful features as we go up the layers by having multiple costs influencing more directly different modules. Here neither the future generator or the per-frame classifier are shown. The latter tries to classify the action based on single frames and takes the per-frame encodings to produce probabilities of actions.

Frame reconstruction: The frame auto-encoder's role is to learn robust per-frame features in an unsupervised manner by reconstructing the clean version ($\mathbf{x}_t$) of a corrupted frame ($\tilde{\mathbf{x}}_t$) at time t. The reconstructive error ($l_{FRE,t}$) 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 learnt by the bottom feed-forward layers of the per-frame auto-encoder, while $g()$ is the decoding function of the module learnt 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 $\mathbf{W}$ 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.

Frame classification: The per-frame classifier uses $\mathbf{h}_t^{FE}$ as an input to yield activations $\mathbf{a}_t = \mathbf{W}(\mathbf{h}_t^{FE}) + \mathbf{b}$ 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. [9] to classify sequences based on a sequence of activations but differs in the fact that we do not use outputs from recurrent layers.

We use the negative log-likelihood of the correct class as our frame-based 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 [6] 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 $z()$ before outputting probabilities with a softmax.

Sequence encoding: The LSTM encoder's purpose is to encode the whole sequence of learnt features into a fixed length summary vector $\mathbf{c}$ that models temporal dependencies, and which can be used for supervised or unsupervised tasks, $\mathbf{c}(\mathbf{X}) = \tanh(\mathbf{W}_{sc}\mathbf{h}_T^{SE} + \mathbf{b})$, where, $\mathbf{c}(\mathbf{X})$ is the output of

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

Sequence reconstruction: If the sequence reconstructive decoder is present, it learns to reconstruct the sequence $\mathbf{X}$ that was fed to the LSTM encoder. As explained by Srivastava et al. [43], the LSTM decoder can use its own previous prediction at each timestep to predict the current output, making it a conditional decoder. This is what we use in this work. With the outputted $\hat{\mathbf{X}} = [\hat{\mathbf{x}}_1^S, ..., \hat{\mathbf{x}}_T^S]$ from the decoder, and the frame decoding function $\mathbf{g}()$, we can calculate our feature sequence reconstruction error (l_{SR}):

$$\mathbf{h}_t^D = \tanh(\mathbf{W}_{ih}\hat{\mathbf{h}}_{t-1}^{FE} + \mathbf{W}_{hh}\mathbf{h}_{t-1}^D + \mathbf{W}_{ch}\mathbf{c}(\mathbf{X}) + \mathbf{b})$$
$$\hat{\mathbf{h}}_t^{FE} = \tanh(\mathbf{W}\mathbf{h}_t^D + \mathbf{b})$$
$$\hat{\mathbf{x}}_t = g(\hat{\mathbf{h}}_t^{FE})$$
$$l_{SR,t} = \frac{1}{2}||\hat{\mathbf{x}}_t^S - \mathbf{x}_t||^2$$
$$l_{SR} = \frac{1}{T}\sum_{t=1}^{T} l_{SR,t}$$

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}):

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

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:

$$\ell = l_{SC} + i(FC) \cdot l_{FC} + \omega \cdot \frac{i(FR) \cdot l_{FR} + i(SR) \cdot l_{SR}}{i(FR) + i(SR)} \tag{1}$$

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).

3.2. Constrained Conditional Generation

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 adversarial learning procedure for RNNs using physics-based soft constraints by 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 could naturally allow sampling capabilities to such a tool since the generator could potentially add variety to the transitions based on the noise sampling. This motivates our exploration with GANs and why stabilizing their learning could be beneficial. Figure 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.

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, $\mathbf{c}_f = \sigma(\mathbf{W}\mathbf{x}_{T+1} + b)$, where $\mathbf{x}_{T+1}$ is the future key-pose frame at time $t = T + 1$.

Transition generator: Our transition generator is a stack of LSTM layers with additional conditioning connections that transform and transmit $\mathbf{z}$ and $\mathbf{c}_f$ information at each timestep, so it always has access to it's future

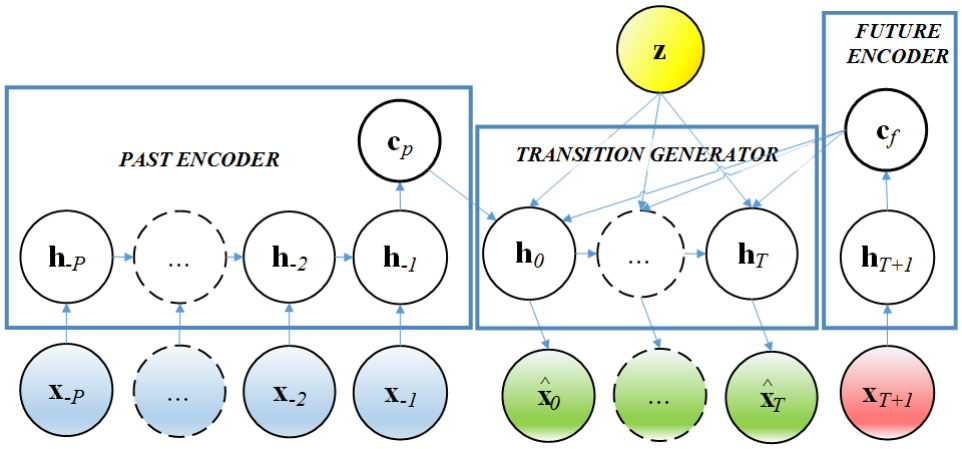

Figure 3: Overview of our generative architecture where stacked LSTM layers are not shown. Different time indices are used for this problem.

target. When generating words, it is common to use an end-of-sentence keyword in order to have variable length generations. In our case, we use a stop criterion that is based on the distance of a generated pose $\hat{\mathbf{x}}_t$ to the future target $\hat{\mathbf{x}}_{T+1}$. We can therefore stop according to the condition $||\hat{\mathbf{x}}_t - \hat{\mathbf{x}}_{T+1}||^2 < \lambda$ where λ is a distance threshold.

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 $\mathbf{a}_t = \mathbf{W}(\mathbf{h}_t^D) + \mathbf{b}$. 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.

Reconstruction objective: A common way to train a generative architecture is to use a reconstruction loss on the outputted sequences. We can obtain a per-frame reconstruction loss $l_{rec,t} = \frac{1}{2}||\hat{\mathbf{x}}_t - \mathbf{x}_t||^2$ and average it to get our full reconstruction loss $l_{rec} = \frac{1}{T+1}\sum_{t=0}^{T} l_{rec,t}$, where $T+1$ is the number of frames in the transition.

Our adversarial objective: For the adversarial loss, both our generator G and our discriminator D are needed. Regular adversarial networks [16] are designed to be trained by playing the 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 but the past and future encoders. We therefore have the following objective:

$$\begin{aligned}\min_G \max_D V(D,G) =& E_{\mathbf{X}\sim p_{data}(\mathbf{X})}[\log D(\mathbf{X}_{trans}|\mathbf{X}_{past}, \mathbf{x}_{target})] \\ &+ E_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z}, \mathbf{c}_p, \mathbf{c}_f)))]\end{aligned} \tag{2}$$

Bone length consistency : Similarly to [23], 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:

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

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** :

$$l_{bone} = \frac{1}{N}\frac{1}{(T+2)}\sum_{n=0}^{N-1}\sum_{t=0}^{T+1} -\mathbf{LLB}_{n,t} \tag{3}$$

where N is the number of bones.

Joint velocity constraints: 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**:

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

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.

4. Experiments

4.1. Experimental Setup

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 markers' 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. [41] 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.

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, and minibatches of size 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. [32].

We use a model with a frame encoder that is closely related to the one used by Cho and Chen [6], 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 probabilities. Each layer of the LSTM decoder has a number of units equal to 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 Gers et al. [13] and Jozefowicz et al. [29]. 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 once 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 before retrieving the position of the maximum value (argmax) for classification. 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 $\omega = 50.0$ 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 3.2. When using our soft constraints, the generator also minimizes Equation 3.2 and Equation 3.2 to better shape the generated poses.

4.2. Regularization by Reconstruction

The experiments we conducted here used the small HDM05 dataset only and used our proposed held-out actors as a test set. Examining these results, we are able to see the regularization effects of adding different types of reconstructive modules and losses to the network's composite error function.

Table 2 shows these effects. Each result is the average classification accuracy for the test set of three different runs with the same architecture.

Table 2: Regularization effects of different models on accuracies with 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, we were able to see that all tested variants with optional modules performed better than our sequence classifier only architecture, which is better than our implemented baselines. 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.

4.3. Adding CMU to HDM05

The following experiments, summarized in Table 3, compares our results for the movement classification task using on 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 and Chen [6] and Zhu et al. [52] 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 were computed. As with only labeled sequences, the reconstruction losses were averaged over all sequences in the minibatch. Since SC does not use any reconstruction, no experiment was done on 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 bigger data-regime, the use of multiple reconstruction decoders

Table 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

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, capacity may be *wasted* 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/sequence). However, when combining the reconstruction losses by averaging them (see Equation 3.1), it seems that reduced focus on a single objective helps the network improve the representations by having more *generalizable* reconstructive features, without *wasting* its capacity on a single, harder reconstruction task. This would hint at the fact that an hyper-parameter search for the reconstruction weight ω for each setting, for different models with and without CMU, could have been beneficial for CMU improving performances everywhere. As specified in Section 4.1, the search over different ω values was done only with the FR-SRC model using only HDM05. 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

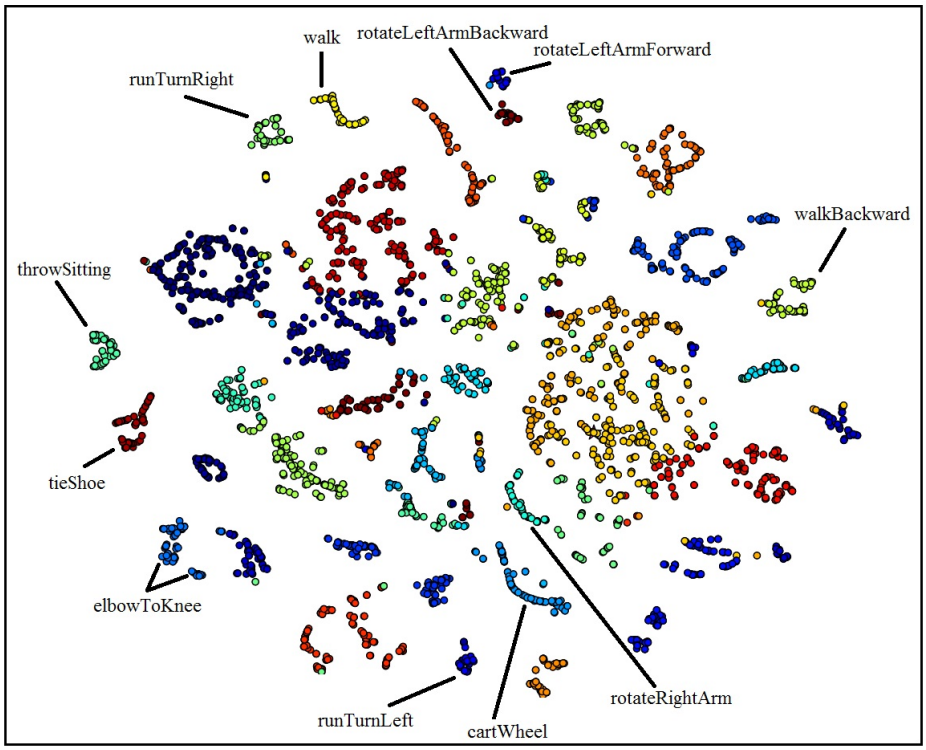

Figure 4: 2D visualization of clusters found by the FR-SRC network with some hand-made annotations after inspection of sub-sequences inside clusters.

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 send to the LSTM encoder.

4.4. Clustering HDM05

Using the FR-SRC network that yielded the best results on HDM05 classification, we produced and 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 4. Note that feature vectors have 1024 dimensions and clusters were found in that space,

while we used the t-SNE algorithm [35] 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.

4.5. Experiments in higher data regime

To study the effect of our approach in a higher data regime, we evaluate our models on NTU RGB+D [41], 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 4.

Table 4: Performance in terms of accuracy on the NTU RGB+D dataset

Model	**CS**	**CV**
Squeletal Quads [11]	38.6	41.4
Lie Group [47]	50.1	52.8
FTP Dynamic Skeletons [24]	60.2	65.2
HBDRNN [9]	59.1	64.0
Deep RNN [41]	56.3	64.1
Deep LSTM [41]	60.7	67.3
Part-aware LSTM [41]	62.9	70.3
ST-LSTM [34]	69.2	77.7
STA-LSTM [42]	73.4	81.2
CNN-MTLN [31]	**79.6**	**84.8**
SC	62.5	65.88
FR-SC	63.4	65.60

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 [41] originally proposed as a baseline for NTU. Table 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. In other words, additional unsupervised objectives clearly show to be beneficial on HDM05 and CMU datasets, as supported by Table 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-decoders approach effectively improves generalization on smaller datasets by acting as a regularization strategy, as supported by Table 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.

4.6. Constrained Adversarial Generation

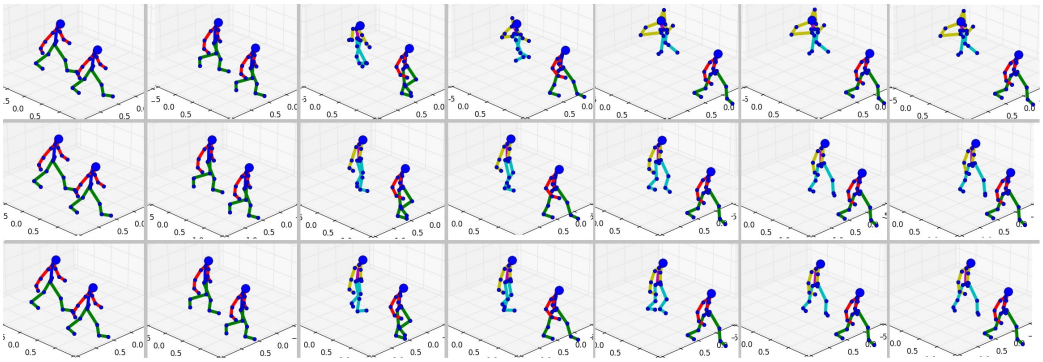

Figure 5: Frames of a sample produced by (TOP) an unconstrained ReGAN, (MIDDLE) a constrained reconstructive generator, and (BOTTOM) a constrained ReGAN.

We apply our proposed conditional future generator on the same datasets. In this setup, we specify a maximum number of transition frames (45) in order to make predictions stop even if the target is not reached. The ground truths contain transitions of 15 frames for the given pairs of past context and future targets. Figure 5 shows a sample output with different losses. The improvement of adding physics-based soft constraints is easily noticeable as without them, the generator never learns to produce stable transitions. As differences between samples from reconstructive and constrained adversarial generators, such as small reductions of foot sliding, are considerably harder to see on still frames, the reader is encouraged to see videos in the supplementary materials. A common limitation of both the reconstructive and constrained generators is the difficulty to produce movements at plausible

speeds. In both experiments, most samples show transitions taking either the minimum or the maximum number of frames allowed. This hints at the fact that the produced representations of the encoder might not contain enough information on velocity or acceleration of movements. The choice of the stop criterion could also be improved. A better design choice based on a distance of each joint to the target pose that depends on the velocity of the said joint could lead better movement continuity when reaching the target. The main contribution of these exploratory experiments are therefore the significant stability improvement of recurrent adversarial learning brought by using soft constraints based on train-data statistics. 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.

5. 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 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.

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 our colleague David Kanaa for help and discussions in times of need. We thank Ubisoft and the Natural Sciences and Engineering Research Council of Canada for support under the Collaborative Research and Development program. Finally we want to thank the authors of the Theano framework [46].

References

[1] Barnachon, M., Bouakaz, S., Boufama, B., Guillou, E., 2014. Ongoing human action recognition with motion capture. Pattern Recognition 47 (1), 238–247.

[2] Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., et al., 2007. Greedy layer-wise training of deep networks. Advances in neural information processing systems 19, 153.

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

[4] Chaudhry, R., Ofli, F., Kurillo, G., Bajcsy, R., Vidal, R., 2013. Bio-inspired dynamic 3d discriminative skeletal features for human action recognition. In: Computer Vision and Pattern Recognition Workshops (CVPRW), 2013 IEEE Conference on. IEEE, pp. 471–478.

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

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

[7] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y., 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.

[8] Cotter, S. F., Rao, B. D., Engan, K., Kreutz-Delgado, K., 2005. Sparse solutions to linear inverse problems with multiple measurement vectors. IEEE Transactions on Signal Processing 53 (7), 2477–2488.

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

[10] Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., Bengio, S., 2010. Why does unsupervised pre-training help deep learning? The Journal of Machine Learning Research 11, 625–660.

[11] Evangelidis, G., Singh, G., Horaud, R., 2014. Skeletal quads: Human action recognition using joint quadruples. In: Pattern Recognition (ICPR), 2014 22nd International Conference on. IEEE, pp. 4513–4518.

[12] Fragkiadaki, K., Levine, S., Felsen, P., Malik, J., 2015. Recurrent network models for human dynamics. In: Computer Vision (ICCV), 2015 IEEE International Conference on. IEEE, pp. 4346–4354.

[13] Gers, F. A., Schmidhuber, J., Cummins, F., 2000. Learning to forget: Continual prediction with lstm. Neural computation 12 (10), 2451–2471.

[14] Ghosh, A., Kulharia, V., Mukerjee, A., Namboodiri, V., Bansal, M., 2016. Contextual rnn-gans for abstract reasoning diagram generation. arXiv preprint arXiv:1609.09444.

[15] Ghosh, P., Song, J., Aksan, E., Hilliges, O., 2017. Learning human motion models for long-term predictions. arXiv preprint arXiv:1704.02827.

[16] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative adversarial nets. In: Advances in Neural Information Processing Systems. pp. 2672–2680.

[17] Graves, A., 2012. Supervised sequence labelling with recurrent neural networks. Vol. 385. Springer.

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

[19] Graves, A., Liwicki, M., Bunke, H., Schmidhuber, J., Fernández, S., 2008. Unconstrained on-line handwriting recognition with recurrent neural networks. In: Advances in Neural Information Processing Systems. pp. 577–584.

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

[21] Hinton, G. E., Osindero, S., Teh, Y.-W., 2006. A fast learning algorithm for deep belief nets. Neural computation 18 (7), 1527–1554.

[22] Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural computation 9 (8), 1735–1780.

[23] Holden, D., Saito, J., Planet, M. A., Komura, T., 2015. A deep learning framework for character motion synthesis and editing. IEEE Transactions on Visualization and Computer Graphics 21, 1.

[24] Hu, J.-F., Zheng, W.-S., Lai, J., Zhang, J., 2015. Jointly learning heterogeneous features for rgb-d activity recognition. In: Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on. IEEE, pp. 5344–5352.

[25] Huang, X., Li, Y., Poursaeed, O., Hopcroft, J., Belongie, S., 2016. Stacked generative adversarial networks. arXiv preprint arXiv:1612.04357.

[26] Hung, C.-C., Carlson, E. T., Connor, C. E., 2012. Medial axis shape coding in macaque inferotemporal cortex. Neuron 74 (6), 1099–1113.

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

[28] Jain, A., Zamir, A. R., Savarese, S., Saxena, A., 2016. Structural-rnn: Deep learning on spatio-temporal graphs. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5308–5317.

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

[30] Ke, Q., An, S., Bennamoun, M., Sohel, F., Boussaid, F., 2017. Skeletonnet: Mining deep part features for 3-d action recognition. IEEE signal processing letters 24 (6), 731–735.

[31] Ke, Q., Bennamoun, M., An, S., Sohel, F., Boussaid, F., 2017. A new representation of skeleton sequences for 3d action recognition. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, pp. 4570–4579.

[32] LeCun, Y., Bottou, L., Orr, G. B., Müller, K.-R., 1998. Efficient backprop. In: Neural networks: Tricks of the trade. Springer, pp. 9–50.

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

[34] Liu, J., Shahroudy, A., Xu, D., Wang, G., 2016. Spatio-temporal lstm with trust gates for 3d human action recognition. In: European Conference on Computer Vision. Springer, pp. 816–833.

[35] Maaten, L. v. d., Hinton, G., 2008. Visualizing data using t-sne. Journal of Machine Learning Research 9 (Nov), 2579–2605.

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

[37] Müller, M., Röder, T., Clausen, M., Eberhardt, B., Krüger, B., Weber, A., 2007. Documentation mocap database hdm05.

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

[39] Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., Lee, H., 2016. Generative adversarial text to image synthesis. arXiv preprint arXiv:1605.05396.

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

[41] Shahroudy, A., Liu, J., Ng, T.-T., Wang, G., 2016. Ntu rgb+ d: A large scale dataset for 3d human activity analysis. arXiv preprint arXiv:1604.02808.

[42] Song, S., Lan, C., Xing, J., Zeng, W., Liu, J., 2017. An end-to-end spatio-temporal attention model for human action recognition from skeleton data. In: AAAI. Vol. 1. p. 7.

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

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

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

[46] Team, T. T. D., Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., Ballas, N., Bastien, F., Bayer, J., Belikov, A., et al., 2016. Theano: A python framework for fast computation of mathematical expressions. arXiv preprint arXiv:1605.02688.

[47] Vemulapalli, R., Arrate, F., Chellappa, R., 2014. 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. pp. 588–595.

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

[49] Yu, D., Deng, L., Dahl, G., 2010. 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.

[50] Yu, L., Zhang, W., Wang, J., Yu, Y., 2016. Seqgan: sequence generative adversarial nets with policy gradient. arXiv preprint arXiv:1609.05473.

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

[52] Zhu, W., Lan, C., Xing, J., Zeng, W., Li, Y., Shen, L., Xie, X., 2016. Co-occurrence feature learning for skeleton based action recognition using regularized deep lstm networks. In: Thirtieth AAAI Conference on Artificial Intelligence.