RECURRENT ENCODER MULTI-DECODER NETWORKS, SEMI-SUPERVISED CLASSIFICATION AND CONSTRAINED ADVERSARIAL GENERATION

Félix G. Harvey & Christopher Pal
Polytechnique Montréal, Montreal, Canada
The Montreal Institute for Machine Learning Algorithms (MILA), Montreal, Canada
{felix.gingras-harvey,chris.pal}@polymtl.ca

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. Our experiments are conducted on two well known Motion Capture data sets. We also explore a novel formulation for future predicting decoders based on conditional recurrent generative adversarial networks. Further our networks have both soft and hard constraints derived from desired physical properties of synthesized future movements and desired animation goals. We find that networks with these properties reduce common artifacts in generated sequences compared to using simpler architectures.

1 INTRODUCTION

It is often the case that for a given task 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 (Erhan et al., 2010; Bengio et al., 2007). A common technique for semi-supervised learning is to perform training in two phases: unsupervised pre-training, followed by supervised fine tuning (Hinton et al., 2006; Bengio et al., 2007; Erhan et al., 2010; Yu et al., 2010). The unsupervised pre-learning 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 highly dimensional parameter space for the non-convex optimization which is the supervised task.

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 capacity that recurrent neural networks (RNNs) have to model temporal dependencies and that when handling a sequence, the last hidden state of an RNN can contain information about the whole sequence. Therefore, this representation has a fixed length, even if sequences do not. The separation between the encoder and the decoder network(s) allows one to easily add, modify or re-purpose decoders for desired tasks. Using multiple decoders forces the encoder to learn rich, multipurpose representations. This can also allow semi-supervised training in a single phase. In our case, we jointly train a classifier decoder and a reconstruction decoder which all use the representation provided by the encoder. Our empirical study shows that having these unsupervised decoders helps regularization on the sequences classification accuracy. We also perform separate experiments in which an new constrained recurrent adversarial decoder learns to generate future frames with the same encoder architecture. We use Long-Short-Term-Memory (LSTM) models to encode and decode sequences, and a multilayer perceptron (MLP) to classify them. Our adversarial discriminator is a bi-directional LSTM (BDLSTM), and outputs predictions at each timesteps. We also studied the effect of adding a per-frame decoder that uses frame representations provided by the frame encoder. We show that our architecture improves movement recognition accuracy on a newly defined realistic partitioning of a popular public Motion Capture (MOCAP) dataset.
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 validation and test set partitioning of a widely used public MOCAP dataset, thereby facilitating more informative future evaluations. We show improvements over our implementation of current state-of-the-art techniques for action recognition on such well defined experiments. We 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 find that this novel type of network formulation can dramatically improve the quality of synthesized motion sequences with Generative Adversarial Networks (GANs) while also allowing animators to impose constraints.

2 RELATED WORK

**Recurrent-Encoder-Decoders:** RNNs have proven over the years to be very powerful models for sequential data, such as speech (Graves et al., 2013b; Sak et al., 2014; Graves et al., 2013a), handwriting (Graves et al., 2008), text (Sutskever et al., 2011; Graves, 2012), or as in our case, MOCAP (Du et al., 2015; Zhu et al., 2016). We use LSTMs (Hochreiter & Schmidhuber, 1997) without in-cell connections (as suggested by Breuel (2015)) 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. (2014) as well as Sutskever et al. (2014) 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. (2014). 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. (2015) 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.

**Generative Adversarial Networks:** GANs (Goodfellow et al., 2014) 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 (Reed et al., 2016; Radford et al., 2015; Huang et al., 2016), results on sequential data remains more limited. Ghosh et al. (2016) 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. (2016) 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.

**MOCAP datasets:** One challenge with the application of deep learning on MOCAP data is the lack of strongly labeled data. For this work, we used the two biggest publicly available MOCAP datasets that we know of. The first is the HDM05 public dataset (Müller et al., 2007). It contains 2329 labeled cuts that are very well suited for action recognition. We use the same 65 classes defined by Cho & Chen (2013). The second dataset is the CMU Graphics Lab Motion Capture Database. This is to our knowledge the biggest public 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.

1 http://mocap.cs.cmu.edu/
and does not seem to have followed any stable conventions throughout the dataset. The work by [Zhu et al. (2016), Ijjina et al. (2016), and Barnachon et al. (2014)] 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.

**Action recognition:** Much of the prior work on MOCAP analysis has been based on hand-crafted features. For example, [Chaudhry et al. (2013)] created bio-inspired features based on the findings of [Hung et al. (2012)] 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. (2016)] 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. (2014)] 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 & Chen (2013)] have obtained good movement classification rates on simple sequences (cuts) on the HDM05 dataset using a Multi-Layer Perceptron (MLP)+Auto-encoder hybrid. [Chen & Koskela (2013)] 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. (2015)] treated the simple sequences’ classification problem with the same action classes as [Cho & Chen (2013)] 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. (2016)] 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. (2005)), which encourages parts 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.

**Defining a good test set:** Table 1 shows the results of our own implementation of previous state-of-the-art methods [Cho & Chen (2013), Du et al. (2015), Zhu et al. (2016)] on the HDM05 dataset using our controlled experimental setup. It shows how using partitions based on actors can hurt the accuracy on the test set. In this setting, we use actors with initials ‘tr’ and ‘dg’ as test subjects, and the actor ‘bk’ as a validation subject. In this scenario, we always train the network once on the three partitions (with early stopping w.r.t subject ‘bk’), then start a new training with the train and validation sets combined. In this second phase, early stopping is done when accuracy on the previous training set reaches the same level as in the first phase. This is done in order to maximize the amount of training data in a principled way, while preventing over-fitting. Moreover, since using three out of five actors from HDM05 for validation and testing leaves about 40% of the sequences

| TECHNIQUE | TEST SET | ACC. (%) |
|-----------|----------|----------|
| CHO & CHEN | RANDOM 10%, BALANCED | 95.61 |
| CHO & CHEN | RANDOM 40%, BALANCED | 94.13 |
| CHO & CHEN | ACTORS [‘TR’, ‘DG’] | 64.36 |
| CHO & CHEN | ACTORS [‘TR’, ‘DG’] - (PP) | 81.64 |
| DU ET AL. | RANDOM 10%, BALANCED | 92.98 |
| DU ET AL. | ACTORS [‘TR’, ‘DG’] - (PP) | 70.63 |
| ZHU ET AL. | RANDOM 10%, BALANCED | 94.53 |
| ZHU ET AL. | ACTORS [‘TR’, ‘DG’] - (PP) | 81.64 |

This paper is under consideration at Image and Vision Computing Journal
in the training set, we tested again the method from Cho & Chen (2013) with a balanced, shuffled partition having the same proportions (40%, 20%, 40%) for each set 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 fairer comparisons later. Our preprocessing of the data is explained in Section 4 and its effects can be seen in Figure 1. As we can see, results using a realistic partition of HDM05 are significantly lower, but our own pre-processing method of the data helps significantly. Since the techniques of Cho & Chen (2013) and Zhu et al. (2016) 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 be 81.64% that was reached by those methods.

3 OUR MODELS

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 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 ever-more 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.

The frame auto-encoder’s role is to learn robust per-frame features in an unsupervised manner by reconstructing the clean version \( x_t \) of a corrupted frame \( \tilde{x}_t \) at time \( t \) [Vincent et al., 2008].
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: $h_{t}^{FE} = z(\hat{x}_t), \hat{x}_t = g(h_{t}^{FE})$ and

$$l_{FR,t} = 1/2||\hat{x}_t - x_t||^2, \quad l_{FR} = 1/T \sum_{t=1}^{T} l_{FR,t}$$

Here, $z(\cdot)$ is the encoding function learnt by the bottom feed-forward layers of the per-frame autoencoder, while $g(\cdot)$ is the decoding function of the module learnt by its upper layers. In further equations, $H^{FE}$ will stand for the sequence of features $[h_{t}^{FE}, \ldots, h_{T}^{FE}]$ and we will dismiss the corruption sign over $x$ ($\hat{x}$) as we will show equations for a test setting, where the frames are not corrupted.

The per-frame classifier uses $h_{t}^{FE}$ as an input to yield activations $a_t = W(h_{t}^{FE}) + b$ on movement classes for every frame. These activations are then summed over all frames into $a_f = \sum_{t=1}^{T} a_t$ and a softmax operation is applied on the result, yielding class probabilities $P(y_k)$ given all the frames $x_t$ of the whole sequence $X$, and the parameters of the frame encoder $\theta_{FE}$:

$$P(y_k|X, \theta_{FE}) = s_{f,k} = e^{a_{f,k}} / (\sum_{i=1}^{K} e^{a_{f,i}})$$

This is similar to the operation used by Du et al. [Du et al., 2015] to classify sequences based on a sequence of activations but differs in the fact that we do not use outputs from recurrent layers.

We then use the negative log-likelihood of the correct class as our frame-based classification error ($l_{FC}$):

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

The combination of the frame auto-encoder and the frame classifier gives something very similar to Cho & Chen’s [Cho & Chen, 2013] 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 $z(\cdot)$ before outputting probabilities with a softmax.

The LSTM encoder’s purpose is to encode the whole sequence of learnt features into a fixed length summary vector $c$ that models temporal dependencies, and which can be used for supervised or unsupervised tasks.

$$c(X) = \tanh(W_{ce}h_{T}^{SE} + b)$$

Here, $c(X)$ is the output of a fully connected layer that has the weight matrix $W_{se}$. It uses the last hidden state of the LSTM encoder $h_{T}^{SE}$ as an input. The encoder itself takes $H^{FE}$ as an input sequence. If the sequence reconstructive decoder is present, it learns to reconstruct the sequence $X$ that was fed to the LSTM encoder. As explained by Srivastava et al. [Srivastava et al., 2015], 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 $\hat{X} = [\hat{x}_1, \ldots, \hat{x}_T]$ from the decoder, and the frame decoding function $g(\cdot)$, we can calculate our feature sequence reconstruction error ($l_{SR}$):

$$h_t^{FE} = \tanh(W_{hh}h_{t-1}^{FE} + W_{hh}h_{t-1}^{D} + W_{ch}c(X) + b)$$

$$h_t^{D} = \tanh(W_{hd}h_{t-1}^{D} + W_{hd}h_{t-1}^{D} + W_{ch}c(X) + b)$$

$$\hat{x}_t = g(h_{t}^{FE})$$

$$l_{SR,t} = 1/2||\hat{x}_t - x_t||^2, \quad l_{SR} = 1/T \sum_{t=1}^{T} l_{SR,t}$$

The sequence classifier is an MLP that outputs class probabilities based on the summary vector. This is the main task of interest, and the sequence classifier is therefore always used in our experiments. We again use the negative log-likelihood as the sequence classification error ($l_{SC}$):

$$h_{c} = W_{c}c(\hat{X}) + b, \quad a_{seq} = Wh_{c} + b$$

$$P(y_k|X, \theta_{SC}) = s_{seq,k} = e^{a_{seq,k}} / (\sum_{i=1}^{K} e^{a_{seq,i}})$$

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

5
Using a sequence classification ratio $r$, we can define different models with different loss functions, enabling some or all of the modules of the architecture. To emphasize on the task of interest, we always put the sequence classification error from the summary vector ($l_{SC}$) against the mean of the other used errors. Given $i_m \in \{0, 1\}^3$ as an indicator of the presence of a module, and $l_m \in \{l_{SR}, l_{FR}, l_{FC}\}$ we formulate our total loss as

$$\ell = r \cdot l_{SC} + (1 - r) \cdot \sum \frac{i_m \cdot l_m}{\sum i_m}$$

Setting $r$ to 1 will result in a Sequence Classifier (SC) network only. 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, using all modules will give a Frame Reconstructive Classifier-SRC (FRC-SRC).

**Constrained Conditional Generation:** As mentioned above, we have also developed a novel constrained conditional recurrent generative adversarial model aimed at creating high quality conditional future continuations for animation sequences to be generated. Figure 3 shows a summary of the generator model. The past context encoder has the same structure as the encoder described above. We describe the other components below.

![Diagram](image)

**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 $c_f$ that will be used by the generator. With a single layer, $c_f = \sigma(Wx + b)$, where $x_{T+1}$ 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 $z$ and $c_f$ information at each timestep, so it always has access to it’s future 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 $\hat{x}_t$ to the future target $\hat{x}_{T+1}$. We can therefore stop according to the condition $||\hat{x}_t - \hat{x}_{T+1}||^2 < \lambda$ where $\lambda$ 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 $a_t = W(h_{T+1}) + b$. 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 $l_{rec,t} = \frac{1}{2}(\hat{x}_t - x_t)^2$ 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 [Goodfellow et al.] (2014) 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 \( X \) of frames containing the past frames \( X_{\text{past}} \), transition frames \( X_{\text{trans}} \) and future frame \( X_{\text{target}} \). Our generator is not only conditioned on the noise vector \( z \), but also on the past context \( c_p \) and future context \( c_f \) provided but the past and future encoders. We therefore have the following objective:
\[
\min_G \max_D V(D, G) = \]
\[
E_{X \sim p_{data}(X)}[\log D(X_{\text{trans}} | X_{\text{past}}, X_{\text{target}})] + E_{z \sim p_z(z)}[\log(1 - D(G(z, c_p, c_f)))]
\]

**Bone length consistency**: Similarly to [Holden et al. (2015)](Holden2015), 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 \( b^{(m)} \) of mean bone lengths differences between consecutive frames, as well as the vector \( b^{(v)} \) 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:
\[
\text{LLB} = \frac{\log(2\pi b^{(v)})}{2} - \frac{(B - b^{(m)})^2}{2b^{(m)}}
\]
where \( B \) 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 \( L_{\text{bone}} \) by averaging all the negative LLB:
\[
l_{\text{bone}} = \frac{1}{N} \frac{1}{(T + 2)} \sum_{n=0}^{N-1} \sum_{t=0}^{T+1} -\text{LLB}_{n,t}
\]
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 \( \psi^{(m)} \) of mean velocities per bones and another vector \( \psi^{(v)} \) of variances per bones, we get the log-likelihood LLV:
\[
\text{LLV} = \frac{\log(2\pi \psi^{(v)})}{2} - \frac{(V - \psi^{(m)})^2}{2\psi^{(m)}},
\]
where \( V \) 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:
\[
l_{\text{vel}} = \frac{1}{D} \frac{1}{(T + 2)} \sum_{d=0}^{D-1} \sum_{t=0}^{T+1} \min(-\text{LLV}_{d,t}^{(\text{spike})}, -\text{LLV}_{d,t}^{(\text{broad})}),
\]
where \( D \) is the number of input dimensions.

4 Experiments

The data in these experiments come from the open HDM05 and CMU MOCAP datasets. They both are recorded at 120 frames per second (fps) and contain more than 30 markers’ positions. In our case, we use 23 common markers between the two datasets. We work with the C3D file format, which contains series of positions for each marker. Our preprocessing of the data 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 with different sizes. Since we use 23 markers, each frame vector has a dimension of 69. We sub-sample sequences to 30 frames-per-second. When using sliding windows (as explained below), we use a simple majority voting strategy on the windows to select the movement class. In all experiments, 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 8 when handling HDM05 only data, and minibatches of size 32 when using CMU and HDM05.

We use a model with a frame encoder that is closely related to the one used by Cho & Chen (2013), 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 $2 \times 512$ containing past and future information. The $c$ layer, outputting the summary vector is of size 1024, and the $h^r$ layer is of size 512. A softmax layer is placed on top of $h^r$ 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.

For feed-forward layers’ initializations, their weight are drawn uniformly from $[-\sqrt{1/\text{fanin}}, \sqrt{1/\text{fanin}}]$, 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. (2000) and Jozefowicz et al. (2015). The learning rate is initialized to 0.04, and is halved when the validation accuracy is not improved in three consecutive epochs, until it reaches below 0.0001. We use early stopping with a tolerance of 25 epochs. We use a 0.9 momentum value.

Before conducting experiments over variations of our model architecture on the HDM05 and CMU datasets, we tested the network using the FR-SC model in order to explore different values of classification cost ratios and different sliding window’s widths (number of frames we feed to the encoder). We had $r \in \{0.5, 0.75, 1.0\}$, where $r = 1.0$ means there’s no reconstruction, and the width $w \in \{20, 30, 90, \infty\}$ where $\infty$ meant 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 had $r = 0.5$ and $w = 30$ as best hyper-parameters, even though windows widths had limited impact on results.

**Table 2: Regularization effects of different models on accuracies with HDM05 data only.**

| MODEL     | TRAIN (%) | VALID (%) | TEST (%) |
|-----------|-----------|-----------|----------|
| SC        | 99.00     | 67.26     | 78.18    |
| FR-SC     | 98.77     | 77.71     | 82.51    |
| FR-SRC    | 97.43     | 71.20     | 79.80    |
| FRC-SRC   | 98.77     | 70.22     | 78.73    |

**Regularization by Reconstruction:** The experiments we conducted here used the three separate sets from HDM05 (first phase of training described in Section 2). 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. As we can see, adding the frame reconstructive module (FR-SC) helps a lot to reduce over-fitting. Figure 4 clearly shows this effect during training. Adding the sequence reconstruction loss (FR-SRC) and then the frame-based classification (FRC-SRC) also have a beneficial impact on over-fitting compared to the SC model, but to a lesser extent. This, however might be due to the fact that these two bigger networks (FRC-SRC, FR-SRC) have a lot more capacity due to their LSTM decoder and may therefore tend to over-fit in this limited data setting (we use here only 40% of HDM05 for training). In order to validate this hypothesis, further tests with more training data were conducted on these networks as well as on the FR-SC.
Adding CMU:F The final experiments here were performed with the two training phases described in Section 2 in order to generalize better on the test set. Importantly, as they also use the validation set for training and perform early stopping using the original train set, they therefore involve a 50% larger training set compared to the experiments of Table 2. Table 3 shows our results for the movement classification task using our more representative test set for HDM05. We compare our results with our implementation of the baseline techniques from Cho & Chen (2013) and Zhu et al. (2016) on the same test set. Since the CMU data, in terms of number of frames, outnumbers HDM05 by a significant factor, we divide errors on reconstruction of unlabeled data by this factor in order to keep the weight updates for labeled and unlabeled data balanced. Experiments performed with the combined datasets (HDM05+CMU) used pre-trained networks to accelerate training, i.e. we used the networks already trained on HDM05. On HDM05 only, the best model was the FRC-SRC, which used all 4 components for the loss, supporting our main hypotheses that we can obtain higher quality representations of the data when using specialized modules with associated costs in the architecture and that the added network capacity is useful with larger training sets. The overall trend of these results suggests that (1) adding different reconstruction modules significantly reduces over-fitting and that (2) adding a large amount of unlabeled data to continue training might not improve performance (FRC-SRC, FR-SC) but may help a network leaving a bad local minima obtained in a limited data setting (FR-SRC). 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 too hard 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.

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) initialised with K-means++, 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 6. Note that feature vectors have 1024 dimensions and clusters were found in that space, while we used the t-SNE algorithm (Maaten & Hinton, 2008) to create a 2D visualization. Some clusters were annotated after manual inspection to give an idea of what movements the network clustered.
Constrained Adversarial Generation: 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 is also naive. 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.
5 Conclusion

Recurrent Encoder-Decoder architectures with multiple decoders provide an attractive framework for semi-supervised, multipurpose 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 partition of data. We hope this evaluation setup can serve as a benchmark partition for further HDM05 experiments. Our results also indicate that the inclusion of reconstructive decoders can have a regularizing effect on learning. Additionally, we have seen that such networks are well suited for clustering as learnt representations compress reconstructive and discriminative information about sequences. Clusters therefore tend to correspond to actions that resemble previous labels. 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 animation super-resolution.

References

Barnachon, Mathieu, Bouakaz, Saïda, Boubama, Boubaker, and Guillou, Erwan. Ongoing human action recognition with motion capture. *Pattern Recognition*, 47(1):238–247, 2014.

Bengio, Yoshua, Lamblin, Pascal, Popovici, Dan, Larochelle, Hugo, et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.

Breuel, Thomas M. Benchmarking of lstm networks. *arXiv preprint arXiv:1508.02774*, 2015.

Chaudhry, Rizwan, Ofli, Ferda, Kurillo, Gregorij, Bajcsy, Ruzena, and Vidal, René. Bio-inspired dynamic 3d discriminative skeletal features for human action recognition. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2013 IEEE Conference on*, pp. 471–478. IEEE, 2013.

Chen, Xi and Koskela, Markus. Classification of rgb-d and motion capture sequences using extreme learning machine. In *Image Analysis*, pp. 640–651. Springer, 2013.

Cho, Kyunghyun and Chen, Xi. Classifying and visualizing motion capture sequences using deep neural networks. *arXiv preprint arXiv:1306.3874*, 2013.

Cho, Kyunghyun, Van Merriënboer, Bart, Gulcehre, Caglar, Bahdanau, Dzmitry, Bougares, Fethi, Schwenk, Holger, and Bengio, Yoshua. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Cotter, Shane F, Rao, Bhaskar D, Engan, Kjersti, and Kreutz-Delgado, Kenneth. Sparse solutions to linear inverse problems with multiple measurement vectors. *IEEE Transactions on Signal Processing*, 53(7):2477–2488, 2005.

Du, Yong, Wang, Wei, and Wang, Liang. Hierarchical recurrent neural network for skeleton based action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1110–1118, 2015.

Erhan, Dumitru, Bengio, Yoshua, Courville, Aaron, Manzagol, Pierre-Antoine, Vincent, Pascal, and Bengio, Samy. Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research*, 11:625–660, 2010.

Gers, Felix A, Schmidhuber, Jürgen, and Cummins, Fred. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

Ghosh, Arnab, Kulharia, Viveka, Mukherjee, Amitabha, Namboodiri, Vinay, and Bansal, Mohit. Contextual rnn-gans for abstract reasoning diagram generation. *arXiv preprint arXiv:1609.09444*, 2016.

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014.
Graves, Alan, Jaitly, Navdeep, and Mohamed, Abdel-rahman. Hybrid speech recognition with deep bidirectional lstm. In Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on, pp. 273–278. IEEE, 2013a.

Graves, Alan, Mohamed, Abdel-rahman, and Hinton, Geoffrey. Speech recognition with deep recurrent neural networks. In Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on, pp. 6645–6649. IEEE, 2013b.

Graves, Alex. Supervised sequence labelling with recurrent neural networks, volume 385. Springer, 2012.

Graves, Alex, Liwicki, Marcus, Bunke, Horst, Schmidhuber, Jürgen, and Fernández, Santiago. Unconstrained on-line handwriting recognition with recurrent neural networks. In Advances in Neural Information Processing Systems, pp. 577–584, 2008.

Hinton, Geoffrey E, Osindero, Simon, and Teh, Yee-Whye. A fast learning algorithm for deep belief nets. Neural computation, 18(7):1527–1554, 2006.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.

Holden, Daniel, Saito, Jun, Planet, Marza Animation, and Komura, Taku. A deep learning framework for character motion synthesis and editing. IEEE Transactions on Visualization and Computer Graphics, 21:1, 2015.

Huang, Xun, Li, Yixuan, Poursaeed, Omid, Hopcroft, John, and Belongie, Serge. Stacked generative adversarial networks. arXiv preprint arXiv:1612.04357, 2016.

Hung, Chia-Chun, Carlson, Eric T, and Connor, Charles E. Medial axis shape coding in macaque inferotemporal cortex. Neuron, 74(6):1099–1113, 2012.

Ijjina, Earnest Paul et al. Classification of human actions using pose-based features and stacked auto encoder. Pattern Recognition Letters, 2016.

Jozefowicz, Rafal, Zaremba, Wojciech, and Sutskever, Ilya. An empirical exploration of recurrent network architectures. In Proceedings of the 32nd International Conference on Machine Learning (ICML-15), pp. 2342–2350, 2015.

Maaten, Laurens van der and Hinton, Geoffrey. Visualizing data using t-sne. Journal of Machine Learning Research, 9(Nov):2579–2605, 2008.

Müller, Meinard, Röder, Tido, Clausen, Michael, Eberhardt, Bernhard, Krüger, Björn, and Weber, Andreas. Documentation mocap database hdm05, 2007.

Radford, Alec, Metz, Luke, and Chintala, Soumith. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434, 2015.

Reed, Scott, Akata, Zeynep, Yan, Xinchen, Logeswaran, Lajanugen, Schiele, Bernt, and Lee, Honglak. Generative adversarial text to image synthesis. arXiv preprint arXiv:1605.05396, 2016.

Sak, Haşim, Senior, Andrew, and Beaufays, Françoise. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. arXiv preprint arXiv:1402.7128, 2014.

Srivastava, Nitish, Mansimov, Elman, and Salakhutdinov, Ruslan. Unsupervised learning of video representations using lstms. arXiv preprint arXiv:1502.04687, 2015.

Sutskever, Ilya, Martens, James, and Hinton, Geoffrey E. Generating text with recurrent neural networks. In Proceedings of the 28th International Conference on Machine Learning (ICML-11), pp. 1017–1024, 2011.

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc VV. Sequence to sequence learning with neural networks. In Advances in neural information processing systems, pp. 3104–3112, 2014.
Vincent, Pascal, Larochelle, Hugo, Bengio, Yoshua, and Manzagol, Pierre-Antoine. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103. ACM, 2008.

Yu, Dong, Deng, Li, and Dahl, G. 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.

Yu, Lantao, Zhang, Weinan, Wang, Jun, and Yu, Yong. Seqgan: sequence generative adversarial nets with policy gradient. *arXiv preprint arXiv:1609.05475*, 2016.

Zhu, Wentao, Lan, Cuiling, Xing, Junliang, Zeng, Wenjun, Li, Yanghao, Shen, Li, and Xie, Xiaohui. Co-occurrence feature learning for skeleton based action recognition using regularized deep lstm networks. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.