Classifying and Visualizing Motion Capture Sequences using Deep Neural Networks

Kyunghyun Cho and Xi Chen

Department of Information and Computer Science, Aalto University School of Science, Espoo, Finland

Keywords: Gesture Recognition, Motion Capture, Deep Neural Network.

Abstract: The gesture recognition using motion capture data and depth sensors has recently drawn more attention in vision recognition. Currently most systems only classify dataset with a couple of dozens different actions. Moreover, feature extraction from the data is often computational complex. In this paper, we propose a novel system to recognize the actions from skeleton data with simple, but effective, features using deep neural networks. Features are extracted for each frame based on the relative positions of joints (PO), temporal differences (TD), and normalized trajectories of motion (NT). Given these features a hybrid multi-layer perceptron is trained, which simultaneously classifies and reconstructs input data. We use deep autoencoder to visualize learnt features. The experiments show that deep neural networks can capture more discriminative information than, for instance, principal component analysis can. We test our system on a public database with 65 classes and more than 2,000 motion sequences. We obtain an accuracy above 95% which is, to our knowledge, the state of the art result for such a large dataset.

1 INTRODUCTION

Gesture recognition has been a hot and challenging research topic for several decades. There are two main kinds of source data: video and motion capture data (Mocap). Mocap records the human actions based on the human skeleton information. Its classification is very important in computer animation, sports science, human–computer interaction (HCI) and filmmaking.

Recently the low cost and high mobility of RGB-D sensors, such as Kinect, have become widely adopted by the game industry as well as in HCI. Especially in computer vision, the gesture recognition using data from the RGB-D sensors is gaining more and more attention. However, the computational difficulty in directly processing 3–D cloud data from depth information often leads to utilizing the human skeleton extracted from the depth information (Shotton et al., 2011) instead.

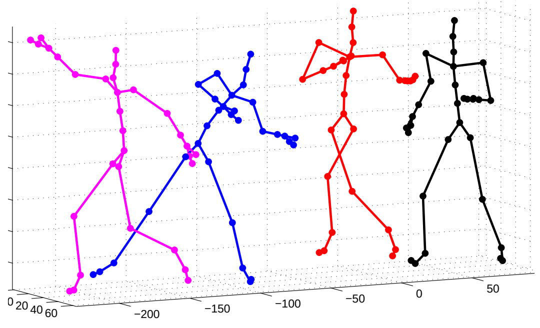
However, conventional recognition systems are mostly applied on a small dataset with a couple of dozens different actions, which is often the limitation imposed by the design of a system. Conventional designs may be classified into two categories: a whole motion is represented by one feature matrix or vector (Raptis et al., 2011), and classified by a classifier as a whole (Müller and Roder, 2006); or a library of key features (Wang et al., 2012) is learned from the whole dataset, and then each motion is represented as a bag or histogram of words (Raptis et al., 2008) or a path in a graph (Barnachon et al., 2013).

In the first type of system, principal component analysis (PCA) is often used to form equal-size feature matrices or vectors from variable-length motion sequences (Zhao et al., 2013; Vieira et al., 2012). However, due to a large number of inter- and intra-class variations among motions, a single feature matrix or vector is likely not enough to capture important discriminative information. This makes these systems inadequate for a large dataset.

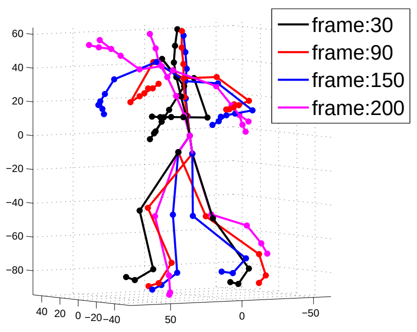
The second type of system decomposes a motion with a manual setup sliding window or key features (Raptis et al., 2008) and builds a codebook by clustering (Chung and Yang, 2013). These approaches also suffer from a large number of action classes due to a potentially excessive size of codebook, in the case of using a classifier such as support vector machines (Ofli et al., 2013) as well as an overly complicated structure, if one tries to build a motion graph.

In this paper, we recognize actions from skeleton data with two major contributions: (1) we propose to build the recognition system based on joint distribution model of the per-frame feature set con-

Cho K. and Chen X..
Classifying and Visualizing Motion Capture Sequences using Deep Neural Networks.
DOI: 10.5220/0004718301220130
In *Proceedings of the 9th International Conference on Computer Vision Theory and Applications* (VISAPP-2014), pages 122-130
ISBN: 978-989-758-004-8



(a) Original



(b) Root

Figure 1: A *throwFar* action in (a) original and (b) root coordinates.

taining information of the relative positions of joints, their temporal difference and the normalized trajectory of the motion; (2) we propose a novel variant of a multi-layer perceptron, called a hybrid MLP, that simultaneously classifies and reconstructs the features, which outperforms a regular MLP, extreme learning machines (ELM) as well as SVM; meanwhile a deep autoencoder is trained to visualize the features in 2-dimensional space, comparing with PCA using the two leading principal components, we clearly see that autoencoder can extract more distinctive information from features. We test our system on a publicly available database containing 65 action classes and more than 2,000 motions with above 95% accuracy.

2 FEATURE EXTRACTION

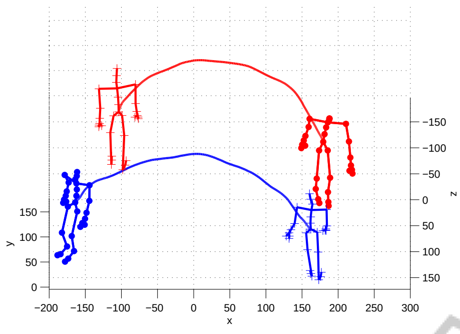
In this section we describe how to extract the proposed features from each frame. Fig. 1 (a) shows some frames from a motion sequence *throwfar*. Since the original coordinates are dependent on a performer and the space to which the performer belongs, those coordinates are not directly comparable between different performers even if they all perform the same action. Hence, we normalize the orientation such that each and every skeleton has its root at the origin $(0,0,0)$ with the same orientation matrix of identity. For example, Fig. 1 (b) shows the orientation-normalized versions of the skeleton in Fig. 1 (a). We further normalize the length of all connected joints to be 1 to make them independent of a performer. The concatenation of 3D coordinates of joints forms the feature (PO), which describes relative relationships among the joints.

Some actions are similar to each other in a frame-level. For instance, the actions of standing up and sitting down are just reverse in time but with almost identical frames, which results in almost identical PO features for corresponding frames in those actions. Hence, we compute the temporal differences (TD) between pairs of PO feature by

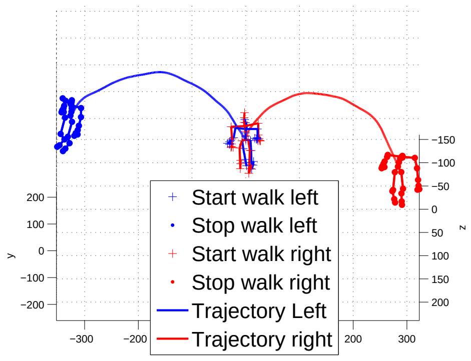
$$\mathbf{f}_{\mathrm{TD}}^{i} = \begin{cases} \mathbf{f}_{\mathrm{PO}}^{i} & 1 \le i < m \\ \dfrac{(\mathbf{f}_{\mathrm{PO}}^{i} - \mathbf{f}_{\mathrm{PO}}^{i-m+1})}{||\mathbf{f}_{\mathrm{PO}}^{i} - \mathbf{f}_{\mathrm{PO}}^{i-m+1}||} & m \le i \le N \end{cases}, \quad (1)$$

where $\mathbf{f}_{\mathrm{PO}}^{i}$ and m are the PO feature vector at the i-th frame and the temporal offset ($1 < m < N$), respectively. TD feature preserves the temporal relationship of the same joint. Normalized trajectory (NT) extracts the absolute trajectory of the motion. Fig.2(a) shows two motions: *walk in a left circle* and *walk in a right circle*. However, in this figure the trajectories of left circle and right circle are not distinguishable. In order to incorporate the trajectory information, we set the same orientation and starting position for the root in the first frame and use a relative position of the root of all the rest frames in the motion sequences from the initial frame, normalized into $[-1,1]$ in each dimension. See Fig. 2 (b) for the effect of this transformation. The final feature for each frame is a concatenation of three features. The dimension of the feature is $3\times n\times 2+3$, where n is the number of joints in use in PO.

For skeleton extracted from RGB-D sensor, often the rotation matrix and translation vector related with the joints are not available. In this case any skeleton can be selected as a stardard template frame, the rotation matrix between the other skeletons and the template can be calculated as in (Chen and Koskela, 2013). In the similar way the features from skeleton data with only 3D joint coordinates can be extracted.



(a) Original



(b) Transformed

Figure 2: Trajectories of two different walks in (a) original and (b) transformed coordinates.

3 DEEP NEURAL NETWORKS: MULTI-LAYER PERCEPTRONS

A multi-layer perceptron (MLP) is a type of deep neural networks that is able to perform classification (see, e.g., (Haykin, 2009)). An MLP can approximate any smooth, nonlinear mapping from a high dimensional sample to a class through multiple layers of hidden neurons.

The output or prediction of an MLP having L hidden layers and q output neurons given a sample $\mathbf{x}$ is typically computed by

$$u(\mathbf{x} \mid \theta) = \sigma\left(\mathbf{U}^\top \phi\left(\mathbf{W}_{[L]}^\top \phi\left(\mathbf{W}_{[L-1]}^\top \cdots \phi\left(\mathbf{W}_{[1]}^\top \mathbf{x}\right) \cdots\right)\right)\right), \tag{2}$$

where σ and ϕ are component-wise nonlinear functions, and $\theta = \left\{\mathbf{U}, \mathbf{W}_{[1]}, \ldots, \mathbf{W}_{[L]}\right\}$ is a set of parameters. We have omitted a bias without loss of generality. A logistic sigmoid function is usually used for the last nonlinear function σ. Each output neuron corresponds to a single class.

Given a training set $\left\{\left(\mathbf{x}^{(n)}, \mathbf{y}^{(n)}\right)\right\}_{n=1}^N$, an MLP is trained to approximate the posterior probability $p(y_j = 1 \mid \mathbf{x})$ of each output class y_j given a sample $\mathbf{x}$ by maximizing the log-likelihood

$$\mathcal{L}_{\text{sup}}(\theta) = \sum_{n=1}^N \sum_{j=1}^q \left(y_j^{(n)} \log u_j\left(\mathbf{x}^{(n)}\right) + \left(1 - y_j^{(n)}\right) \log\left(1 - u_j\left(\mathbf{x}^{(n)}\right)\right)\right), \tag{3}$$

where a subscript j indicates the j-th component. We omitted θ to make the above equation uncluttered. Training can be efficiently done by backpropagation (Rumelhart et al., 1986).

3.1 Hybrid Multi-layer Perceptron

It has been noticed by many that it is not trivial to train deep neural networks to have a good generalization performance (see, e.g., (Bengio and LeCun, 2007) and references therein), especially when there are many hidden layers between input and output layers. One of promising hypotheses explaining this difficulty is that backpropagation applied on a deep MLP tends to utilize only a few top layers (Bengio et al., 2007). A method of layer-wise pretraining has been proposed to overcome this problem by initializing the weights in lower layers with unsupervised learning (Hinton and Salakhutdinov, 2006).

Here, we propose another strategy that forces backpropagation algorithm to utilize lower layers. The strategy involves training an MLP to classify and reconstruct simultaneously by training a deep autoencoder with the same set of parameters, except for the weights between the penultimate and output layers to reconstruct an input sample as well as possible.

A deep autoencoder is a symmetric feedforward neural network consisting of an encoder

$$\mathbf{h} = f(\mathbf{x}) = f_{[L-1]} \circ f_{[L-2]} \circ \cdots \circ f_{[1]}(\mathbf{x})$$

and a decoder

$$\tilde{\mathbf{x}} = g(\mathbf{x}) = g_{[2]} \circ \cdots \circ g_{[L-1]}(\mathbf{h}),$$

where

$$f_{[l]}\left(\mathbf{s}_{[l-1]}\right) = \phi\left(\mathbf{W}_{[l]}^\top \mathbf{s}_{[l-1]}\right),$$
$$g_{[l]}\left(\mathbf{s}_{[l+1]}\right) = \varphi\left(\mathbf{W}_{[l]} \mathbf{s}_{[l+1]}\right).$$

ϕ and φ are component-wise nonlinear functions.

The parameters of a deep autoencoder is estimated by maximizing the negative squared difference which is defined to be

$$\mathcal{L}_{\text{unsup}}(\theta) = -\frac{1}{2} \sum_{n=1}^N \left\| \mathbf{x}^{(n)} - \tilde{\mathbf{x}}^{(n)} \right\|_2^2. \tag{4}$$

Our proposed strategy combines these two networks while sharing a single set of parameters θ by optimizing a weighted sum of Eq. (3) and Eq. (4):

$$\mathcal{L}(\theta) = (1-\lambda)\mathcal{L}_{\text{sup}}(\theta) + \lambda\mathcal{L}_{\text{unsup}}(\theta), \qquad (5)$$

where $\lambda \in [0,1]$ is a hyperparameter. When λ is 0, the trained model will be purely an MLP, while it will be an autoencoder if $\lambda = 1$. We call an MLP that was trained with this strategy with non-zero λ a *hybrid* MLP[1].

There are two advantages in the proposed strategy. First, the weights in lower layers naturally have to be utilized, since those weights must be adapted to reconstruct an input sample well. This may further help achieving a better classification accuracy similarly to the way unsupervised layer-wise pretraining which also optimized the reconstruction error , in the case of using autoencoders, helps obtaining a better classification accuracy on novel samples. Secondly, in this framework, it is trivial to use vast amount of unlabeled samples in addition to labeled samples. If stochastic backpropagation is used, one can compute the gradients of $\mathcal{L}$ by combining the gradients of $\mathcal{L}_{\text{sup}}$ and $\mathcal{L}_{\text{unsup}}$ separately using separate sets of labeled and unlabeled samples.

4 CLASSIFYING AN ACTION SEQUENCE

An action sequence is composed of an certain amount of frames. We use a multi-layer perceptron to model a posterior distribution of classes given each frame. Let us define $s_c \in \{0,1\}$ be a binary indicator variable. If s_c is one, the sequence belongs to the action c, and otherwise, belongs to another action. Since each sequence consists of $N \geq 1$ frames, let us further define $f_{i,c} \in \{0,1\}$ as a binary variable indicating whether the i-th frame belongs to the action c.

When a given sequence $\mathbf{s} = (\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_N)$ is of an action c, every frame $\mathbf{f}_i$ in the sequence is also of an action c. In other words, if $s_c = 1$, $f_{i,c} = 1$ for all i. So, we may check the joint probability of all frames in the sequence to determine the action of the sequence:

$$p(s_c = 1 \mid \mathbf{s}) = \qquad (6)$$
$$p(f_{1,c} = 1, f_{2,c} = 1, \ldots, f_{N,c} = 1 \mid \mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_N).$$

In this paper, we assume temporal independence among the frames in a single sequence, which means that the class of each frame depends *only* on the features of the frame. Then, Eq. (6) can be simplified into

$$p(s_c = 1 \mid \mathbf{s}) = \prod_{i=1}^{N} p(f_{i,c} = 1 \mid \mathbf{f}_i). \qquad (7)$$

With this assumption, the problem of gesture recognition is reduced to first train a classifier to perform frame-level classification and then to combine the output of the classifier according to Eq. (7). A multi-layer perceptron which approximates the posterior probability distribution over classes by Eq. (2) is naturally suited to this approach.

5 EXPERIMENTS

In the experiment we tried to evaluate the performance of our proposed recognition system through a public dataset. We assessed the performance of deep neural networks including regular and hybrid MLP by comparing them against extreme learning machines (ELM) and support vector machines (SVM). The effectiveness of the feature set was evaluated by the classification accuracy and the visualization in 2D space by deep autoencoders.

5.1 Dataset

The Motion Capture Database HDM05 (Müller et al., 2007) is a well organized large MOCAP dataset. It provides a set of short cut MOCAP clips, and each clip contains one integral motion. In the original dataset, there are 130 gesture groups. However, there are some gestures that essentially belong to a single class. For instance, *walk 2 steps* and *walk 4 steps* belong to a single action *walk*. Hence, we combined some of the classes based on the following rules:

1. Motions repeating with different times are combined into one action.
2. Motions with the only difference of the starting limb are combined into one action.

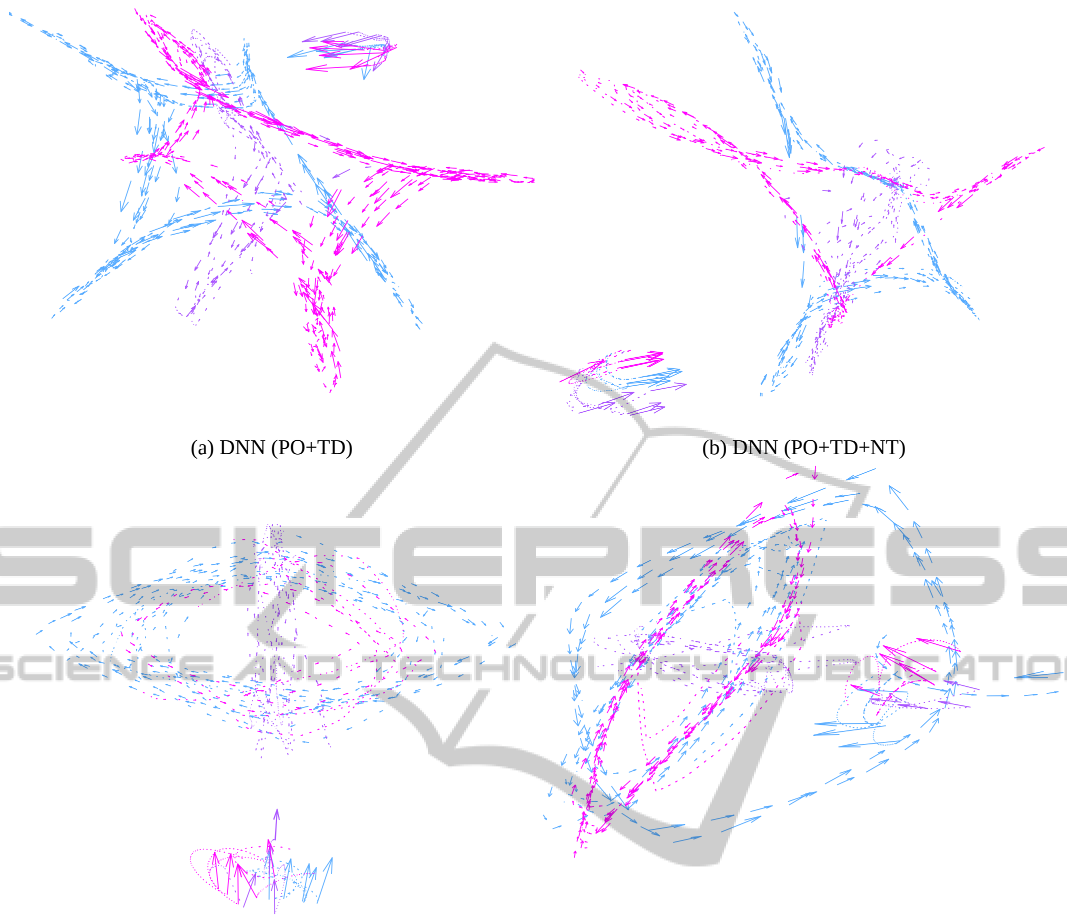
After the reorganization the whole dataset consisting of 2,337 motion sequences and 613,377 frames is divided into 65 actions[2].

5.2 Settings

We used 10-fold cross validation to assess the performance of a classifier. The data was randomly split into 10 balanced partitions of sequences. PO feature was formed by 5 joints:head, hands and feet. The parameter m in TD was set as 0.3 second interval between

[1]A similar approach was proposed in (Larochelle and Bengio, 2008) in the case of restricted Boltzmann machines.

[2]See the appendix for the complete list of 65 actions



(a) DNN (PO+TD)

(b) DNN (PO+TD+NT)

(c) PCA (PO+TD)

(d) PCA (PO+TD+NT)

Figure 3: Visualization of actions *rotateArmsRBackward* (blue), *rotateArmsBothBackward* (purple) and *rotateArmsLBackward* (red). Each arrow denotes the direction and magnitude of change in the latent space. Five randomly selected sequences per action are shown.

frames. The total dimension of the feature vector is 33. To test the distinctiveness of the features, we reported the classification accuracy for each frame, and evaluated the system performance by the accuracy of each sequence. The standard deviations were also calculated for 10-fold cross validation.

We trained deep neural networks having two hidden layers of sizes 1000 and 500 with rectified linear units[3]. A learning rate was selected automatically by a recently proposed ADADELTA (Zeiler, 2012). Usually the optimal λ can be selected by the validation set and on a grid search. To illustrate the influences of λ in hybrid MLP, we selected four different values for λ: 0, 0.1, 0.5 and 0.9. The parameters were simply initialized randomly, and no pretraining strategy was used[4].

When a tested classifier outputs the posterior probability of a class given a frame, we chose the class of a sequence by

$$\arg\max_{c} \sum_{i=1}^{N} \log p(f_{i,c} = 1 \mid \mathbf{f}_i)$$

based on Eq. (7). If a classifier does not return a probability but only the chosen class, we used a simple majority voting.

As a comparison, we tried an extreme learning machine (ELM) (Huang et al., 2006) and SVM in the same system. We used 2,000 hidden neurons for ELM. For SVM we used a radial-basis function kernel, and the hyperparameters C and γ were found through a grid-search and cross-validation.

[3]The activation of a rectified linear unit is $\max(0,\alpha)$, where α is the input to the unit.

[4]We used a publicly available MATLAB toolbox *deepmat* for training and evaluating deep neural networks: https://github.com/kyunghyuncho/deepmat

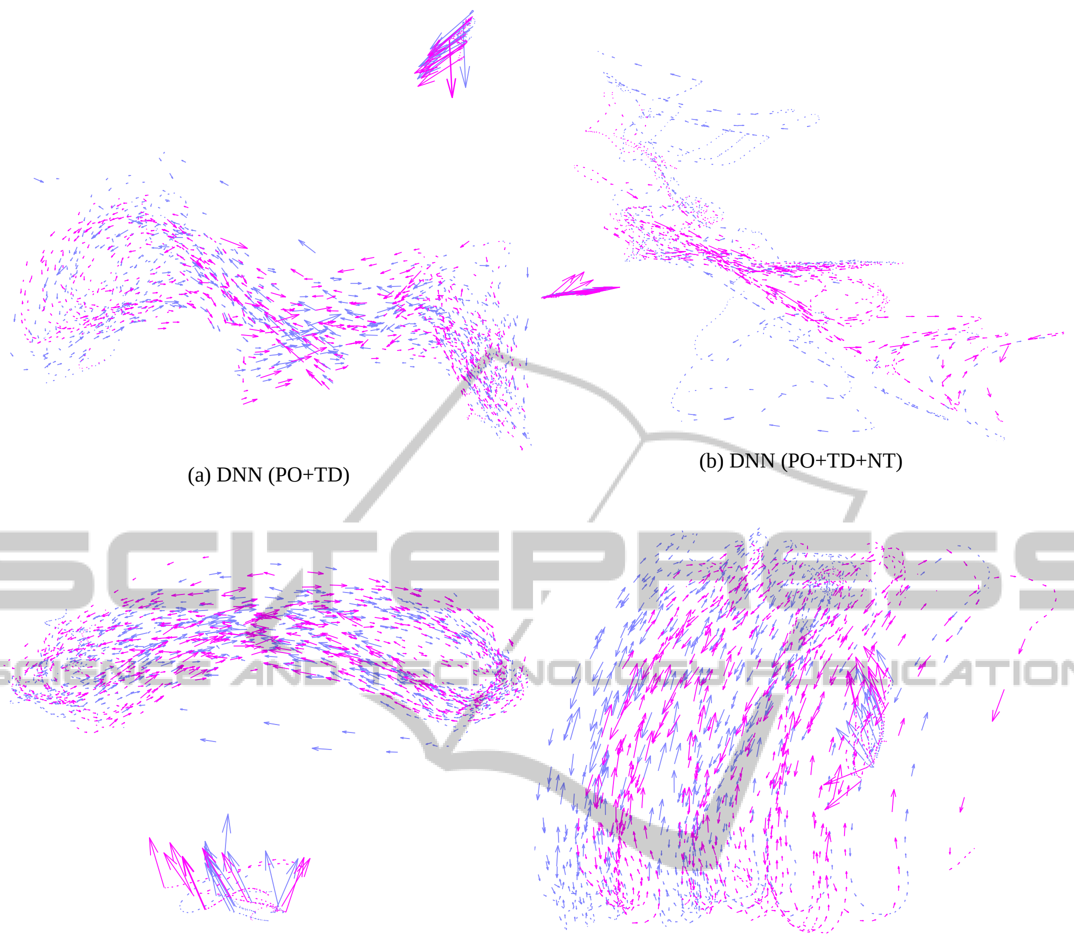


Figure 4: Visualization of actions *jogLeftCircle* (blue) and *jogRightCircle* (purple). Each arrow denotes the direction and magnitude of change in the latent space. Ten randomly chosen sequences per action were visualized.

5.3 Qualitative Analysis: Visualization

In order to have a better understanding of what a deep neural network learns from the features, we visualized the features using a deep autoencoder with two linear neurons in the middle layer (Hinton and Salakhutdinov, 2006). The deep autoencoder had three hidden layers of size 1000, 500 and 100 between the input and middle layers. It should be noted that no label information was used to train these deep autoencoders. In the experiment, we trained two deep autoencoders using with and without the normalized trajectories (NT) to see what the relative feature (PO+TD) provides to the system and the impact of the absolute feature. Since in previous works PCA has been often used for dimensionality reduction of motion features, we also tried to visualize features using the two leading principal components.

In Fig. 3, we visualized three distinct, but very similar, actions; *rotateArmsRBackward*, *rotateArmsBothBackward* and *rotateArmsLBackward*. These actions in the figure were clearly distinguishable when the deep autoencoder was used. However, *rotateArmsRBackward* and *rotateArmsLBackward* were not distinguishable at all when only the PO and TD features were used by PCA (see Fig. 3 (c)). Even when all three features (PO+TD+NT) were used, the visualization by PCA did not help distinguishing these actions clearly.

In Fig. 4, two actions, *jogLeftCircle* and *jogRightCircle*, were visualized. When only PO and TD features were used, neither the deep autoencoder nor PCA was able to capture differences between those actions. However, the deep autoencoder was able to distinguish those actions clearly when all three proposed features were used (see Fig. 4 (b)).

The former visualization shows that a deep neural network with multiple nonlinear hidden layers can

Table 1: Classification accuracies. Standard deviations are shown inside brackets. The highest accuracy in each row is marked bold.

Feature Set	ELM	SVM	MLP	Hybrid MLP $\lambda = 0.1$	0.5	0.9
PO+TD	70.40% (1.32)	83.82% (0.79)	84.35% (0.91)	84.39% (0.87)	**84.57%** (1.56)	84.23% (1.27)
PO+TD+NT	74.28% (1.56)	87.06% (0.82)	87.42% (1.43)	**87.96%** (1.38)	87.34% (0.66)	87.28% (1.38)

Feature Set	ELM	SVM	MLP	Hybrid MLP $\lambda = 0.1$	0.5	0.9
PO+TD	91.57% (0.88)	94.95% (0.82)	95.20% (1.38)	95.46% (0.99)	**95.59%** (0.76)	95.55% (1.14)
PO+TD+NT	92.76% (1.53)	95.12% (0.58)	94.86% (0.99)	**95.21%** (0.86)	94.82% (1.17)	95.04% (0.86)

learn more discriminative structure of data. Furthermore, according to the latter visualization, we can see that the normalized trajectories help distinguish locomotions with different traces, however, with only a powerful model as a deep neural network. Through the experiment we could see that deep neural networks are able to learn highly discriminative information from our features of motion.

5.4 Quantitative Analysis: Recognition

In Tab. 1, the frame-level accuracies obtained by various classifiers with two different sets of features can be found. We can see that the NT feature clearly increases the classification accuracy around $3-4\%$ for all the classifiers. Comparing the different classifiers, we can see that the MLPs were able to obtain significantly higher accuracies than the ELM and perform slightly better than SVM. Furthermore, although it is not clearly significant statistically, we can see that a hybrid MLP often outperforms the regular MLP with a right choice of λ.

A similar trend of the MLPs outperforming the other classifiers could be observed also in sequence-level performance shown in Tab. 1. Again in the sequence-level classification, we observed that the hybrid MLP with a right choice of λ marginally outperformed the regular MLP, and it also outperformed SVM and ELM. For both frame-level and sequence-level accuracy, the highest accuracy for PO+TD features is from hybrid MLP with $\lambda = 0.5$ and for the whole feature set with $\lambda = 0.1$.

However, the classification accuracies obtained using the two sets of features (PO+TD vs PO+TD+NT) are very close to each other. Compared to the $3-4\%$ differences in the classification for each frame, the differences between the performance obtained using the two sets are within the standard deviations. Even though NT feature increases the frame accuracy significantly it did not have the same effect on the sequence level. One potential reason is that once a certain level of frame level recognition is achieved, the sequence level performance using our posterior probability model saturates.

6 CONCLUSIONS

In this paper, we proposed a gesture recognition system using multi-layer perceptrons for recognizing motion sequences with novel features based on relative joint positions (PO), temporal differences (TD) and normalized trajectories (NT).

The experiments with a large motion capture dataset (HDM05) revealed that (hybrid) multi-layer perceptrons could achieve higher recognition rate than there is the other classifiers could, for 65 classes with an accuracy of above 95%. Furthermore, the visualization of feature set of the motion sequences by deep autoencoders showed the effectiveness of the proposed feature sets and enabled us to study what deep neural networks learned. Interestingly, a powerful model like a deep neural network combined with an informative feature set was able to capture the discriminative structure of motion sequences, which was confirmed by both the recognition and visualization experiments. This suggests that a deep neural network is able to extract highly discriminative features from motion data.

One limitation of our approach is that temporal independence was assumed when combining the per-frame posterior probabilities in a sequence. In future it will be interesting to investigate possibilities of modeling temporal dependence.

ACKNOWLEDGEMENTS

This work was funded by Aalto MIDE programme (project UI-ART), Multimodally grounded language technology (254104) and Finnish Center of Excellence in Computational Inference Research COIN (251170) of the Academy of Finland.

REFERENCES

Barnachon, M., Bouakaz, S., Boufama, B., and Guillou, E. (2013). A real-time system for motion retrieval and interpretation. *Pattern Recognition Letters*.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In Schölkopf, B., Platt, J., and Hoffman, T., editors, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press, Cambridge, MA.

Bengio, Y. and LeCun, Y. (2007). Scaling learning algorithms towards AI. In Bottou, L., Chapelle, O., DeCoste, D., and Weston, J., editors, *Large Scale Kernel Machines*. MIT Press.

Chen, X. and Koskela, M. (2013). Classification of rgb-d and motion capture sequences using extreme learning machine. In *Proceedings of 18th Scandinavian Conference on Image Analysis*.

Chung, H. and Yang, H.-D. (2013). Conditional random field-based gesture recognition with depth information. *Optical Engineering*, 52(1):017201–017201.

Haykin, S. (2009). *Neural Networks and Learning Machines*. Pearson Education, 3rd edition.

Hinton, G. and Salakhutdinov, R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

Huang, G.-B., Zhu, Q.-Y., and Siew, C.-K. (2006). Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1-3):489–501.

Larochelle, H. and Bengio, Y. (2008). Classification using discriminative restricted Boltzmann machines. In *Proceedings of the 25th international conference on Machine learning (ICML 2008)*, pages 536–543, New York, NY, USA. ACM.

Müller, M. and Roder, T. (2006). Motion templates for automatic classification and retrieval of motion capture data. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Computer animation*, volume 2, pages 137–146, Vienna, Austria.

Müller, M., Röder, T., Clausen, M., Eberhardt, B., Krüger, B., and Weber, A. (2007). Documentation mocap database HDM05. Technical Report CG-2007-2, U. Bonn.

Ofli, F., Chaudhry, R., Kurillo, G., Vidal, R., and Bajcsy, R. (2013). Berkeley mhad: A comprehensive multimodal human action database. In *Applications of Computer Vision (WACV), 2013 IEEE Workshop on*, pages 53–60.

Raptis, M., Kirovski, D., and Hoppe, H. (2011). Real-time classification of dance gestures from skeleton animation. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 147–156. ACM.

Raptis, M., Wnuk, K., Soatto, S., et al. (2008). Flexible dictionaries for action classification. In *Proc. MLVMA'08*.

Rumelhart, D. E., Hinton, G., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(Oct):533–536.

Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., and Blake, A. (2011). Real-time human pose recognition in parts from single depth images. In *Proc. Computer Vision and Pattern Recognition*.

Vieira, A., Lewiner, T., Schwartz, W., and Campos, M. (2012). Distance matrices as invariant features for classifying MoCap data. In *21st International Conference on Pattern Recognition (ICPR)*, Tsukuba, Japan.

Wang, J., Liu, Z., Wu, Y., and Yuan, J. (2012). Mining actionlet ensemble for action recognition with depth cameras. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1290–1297. IEEE.

Zeiler, M. D. (2012). ADADELTA: An adaptive learning rate method. *arXiv:*`1212.5701 [cs.LG]`.

Zhao, X., Li, X., Pang, C., and Wang, S. (2013). Human action recognition based on semi-supervised discriminant analysis with global constraint. *Neurocomputing*, 105(0):45 – 50.

APPENDIX: 65 Actions in HDM05 Dataset

1. cartwheelLHandStart1Reps
   cartwheelLHandStart2Reps
   cartwheelRHandStart1Reps
2. clap1Reps
   clap5Reps
3. clapAboveHead1Reps
   clapAboveHead5Reps
4. depositFloorR
5. depositHighR
6. depositLowR
7. depositMiddleR
8. elbowToKnee1RepsLelbowStart
   elbowToKnee1RepsRelbowStart
   elbowToKnee3RepsLelbowStart
   elbowToKnee3RepsRelbowStart
9. grabFloorR
10. grabHighR
11. grabLowR
12. grabMiddleR
13. hitRHandHead
14. hopBothLegs1hops
    hopBothLegs2hops
    hopBothLegs3hops
15. hopLLeg1hops
    hopLLeg2hops
    hopLLeg3hops
16. hopRLeg1hops
    hopRLeg2hops
    hopRLeg3hops
17. jogLeftCircle4StepsRstart
    jogLeftCircle6StepsRstart

18. jogOnPlaceStartAir2StepsLStart
    jogOnPlaceStartAir2StepsRStart
    jogOnPlaceStartAir4StepsLStart
    jogOnPlaceStartFloor2StepsRStart
    jogOnPlaceStartFloor4StepsRStart
19. jogRightCircle4StepsLstart
    jogRightCircle4StepsRstart
    jogRightCircle6StepsLstart
    jogRightCircle6StepsRstart
20. jumpDown
21. jumpingJack1Reps
    jumpingJack3Reps
22. kickLFront1Reps
    kickLFront2Reps
23. kickLSide1Reps
    kickLSide2Reps
24. kickRFront1Reps
    kickRFront2Reps
25. kickRSide1Reps
    kickRSide2Reps
26. lieDownFloor
27. punchLFront1Reps
    punchLFront2Reps
28. punchLSide1Reps
    punchLSide2Reps
29. punchRFront1Reps
    punchRFront2Reps
30. punchRSide1Reps
    punchRSide2Reps
31. rotateArmsBothBackward1Reps
    rotateArmsBothBackward3Reps
32. rotateArmsBothForward1Reps
    rotateArmsBothForward3Reps
33. rotateArmsLBackward1Reps
    rotateArmsLBackward3Reps
34. rotateArmsLForward1Reps
    rotateArmsLForward3Reps
35. rotateArmsRBackward1Reps
    rotateArmsRBackward3Reps
36. rotateArmsRForward1Reps
    rotateArmsRForward3Reps
37. runOnPlaceStartAir2StepsLStart
    runOnPlaceStartAir2StepsRStart
    runOnPlaceStartAir4StepsLStart
    runOnPlaceStartFloor2StepsRStart
    runOnPlaceStartFloor4StepsRStart
38. shuffle2StepsLStart
    shuffle2StepsRStart
    shuffle4StepsLStart
    shuffle4StepsRStart
39. sitDownChair
40. sitDownFloor
41. sitDownKneelTieShoes
42. sitDownTable
43. skier1RepsLstart
    skier3RepsLstart
44. sneak2StepsLStart
    sneak2StepsRStart
    sneak4StepsLStart
    sneak4StepsRStart
45. squat1Reps
    squat3Reps
46. staircaseDown3Rstart
47. staircaseUp3Rstart
48. standUpKneelToStand
49. standUpLieFloor
50. standUpSitChair
51. standUpSitFloor
52. standUpSitTable
53. throwBasketball
54. throwFarR
55. throwSittingHighR
    throwSittingLowR
56. throwStandingHighR
    throwStandingLowR
57. turnLeft
58. turnRight
59. walk2StepsLstart
    walk2StepsRstart
    walk4StepsLstart
    walk4StepsRstart
60. walkBackwards2StepsRstart
    walkBackwards4StepsRstart
61. walkLeft2Steps
    walkLeft3Steps
62. walkLeftCircle4StepsLstart
    walkLeftCircle4StepsRstart
    walkLeftCircle6StepsLstart
    walkLeftCircle6StepsRstart
63. walkOnPlace2StepsLStart
    walkOnPlace2StepsRStart
    walkOnPlace4StepsLStart
    walkOnPlace4StepsRStart
64. walkRightCircle4StepsLstart
    walkRightCircle4StepsRstart
    walkRightCircle6StepsLstart
    walkRightCircle6StepsRstart
65. walkRightCrossFront2Steps
    walkRightCrossFront3Steps