Classifying movements using efficient kinematic codes

Leif Johnson (leif@cs.utexas.edu) and **Dana Ballard (dana@cs.utexas.edu)**
Department of Computer Science
The University of Texas at Austin

Abstract

Efficient codes have been shown to perform well in image and audio classification tasks, but the impact of sparsity—and indeed the entire notion of efficient coding—has not yet been well explored in the context of human movements. This paper tests several coding approaches on a movement classification task and finds that efficient codes for kinematic (joint angle) data perform well for classifying many different types of movements. In particular, the best classification method relied on a sparse coding algorithm combined with a codebook that was tuned to kinematic movement data. The other approaches tested here—sparse coding with a random codebook, and "dense" coding using PCA—provide interesting baseline results and allow us to investigate why sparse codes appear to work well.

Introduction

When modeling sensory data like images and sound, efficient codes were proposed (Barlow, 1961) as a mechanism for reducing statistical redundancy in natural inputs, thus providing a neural substrate with an effective use of limited metabolic resources. Indeed, in the past decades, sparse codes have been shown to yield representations of natural sensory data that are similar to receptive fields in living animals (Olshausen & Field, 1996; Smith & Lewicki, 2006), interpretable by humans (Tibshirani, 1996), and effective for computational classification tasks (Lee, Battle, Raina, & Ng, 2007; Glorot, Bordes, & Bengio, 2011; Le, Karpenko, Ngiam, & Ng, 2011; Coates & Ng, 2011). However, in computer science and machine learning, sparsity has not yet been applied widely outside visual and auditory domains; partly this seems to be due to the ease with which photos and sounds can be interpreted by human researchers, and partly this might be due to the large amount of such data available online.

At the same time, sparsity seems ideal for coding movement information because, like sensory data sampled from the natural world, human movements appear to lie along a low-dimensional manifold embedded within the space of all possible movements (Scholz & Schöner, 1999; Latash, Scholz, & Schöner, 2002). Recent ideas in coding (Olshausen & Field, 2004) and feature learning (Bengio, 2013) suggest that sparse codes are effective for representing data along low-dimensional manifolds because the basis vectors that are used to represent a particular data element can be spread out along the manifold, with only a few basis elements representing any particular location in space.

This paper explores the use of efficient codes for classifying kinematic data derived from human movements. We first describe the data source and our computational model for movement classification, and then briefly present the coding approaches that we evaluated for the classification task. The paper concludes by discussing the results of our experiments and comparing them with similar, existing research.

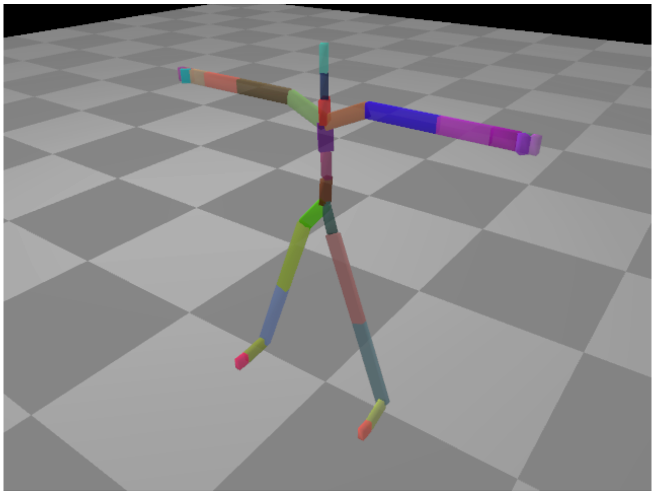

Figure 1: The articulated skeleton in the CMU mocap database consists of 30 rigid "bone" segments joined together with a total of 59 angular degrees of freedom. The joint angles in each frame are computed by the motion capture system, which combines the observed marker positions with a fitted skeleton to compute an angular kinematic representation of the pose.

Data Processing

We used motion-capture data available online through the CMU Mocap Database [1]; the database contains motion-capture recordings from more than 100 subjects performing a variety of actions, ranging from simple walking to complex acrobatic stunts and even common household activities like washing up. The database is not uniformly covered, however: some subjects only performed one type of action, while others performed several; likewise, some actions were only performed once, while others were repeated multiple times. In addition, some motion-capture recordings are quite long (tens of seconds), while many are very brief (just two or three seconds).

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

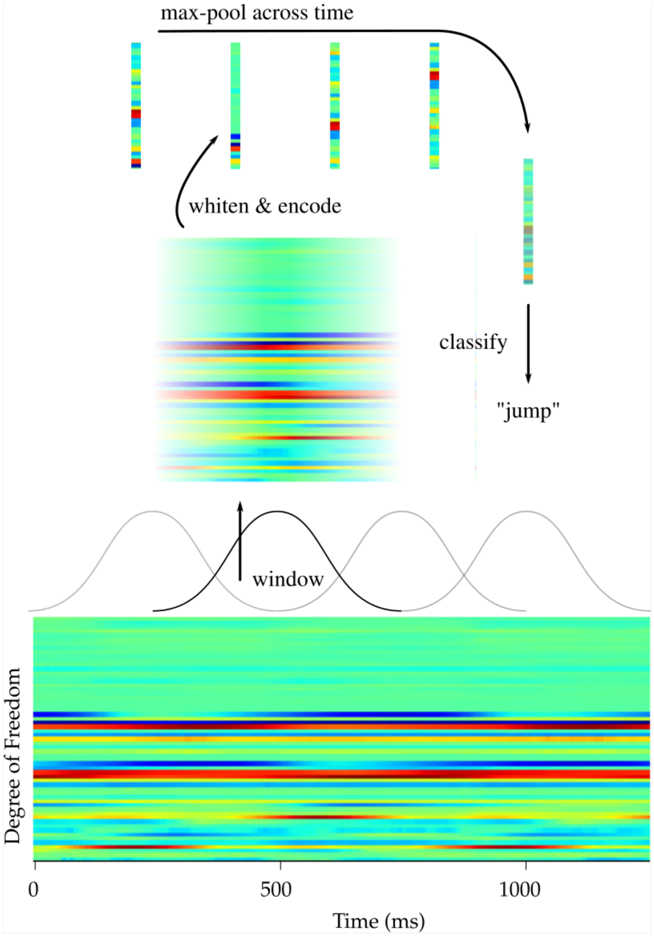

Figure 2: The data processing pipeline consists of five stages: windowing, whitening, encoding, pooling, and classification. Each stage reduces the size of the data; overall, we wish to preserve as much information as possible at each of these steps, in order to maximize the classification performance at the end.

In addition to the complexity of the mocap data itself, the labels in the database are somewhat free-form; for example, movement labels containing the word "jump" are quite varied, including "180 degree jump", "2 foot jump," and "jump up to grab, reach for, tiptoe." This labeling variability actually highlights a problem with "classifying" movement in general, particularly long recordings of movement—what is the correct way to describe a complex motion? Rather than attempt to provide an answer here, we follow an approach that other researchers have used when working with this dataset, namely to restrict analysis to a limited subset of the available movement data (Taylor, Hinton, & Roweis, 2007). Instead of focusing on types of walks, however, we simplified the labeling for our task by focusing solely on recordings of people mimicking different animals; the recordings of such mimicking movements were all labeled "some-animal-name (human subject)," all lasted approximately the same amount of time, and seemed to be consistently labeled. We further reduced the scope of the data to labels for which at least two humans performed a given imitation, yielding a dataset of 102 recordings spanning 20 different types of movement.

Having isolated a dataset of movements that looked reasonably challenging for classification, we designed a data processing pipeline that consisted of five stages: windowing, whitening, encoding, max-pooling, and classification (see Figure 2). We describe the computational encoding and classification stages in the next section, but first we briefly describe the three stages common across the experiments. These stages (windowing, whitening, and max-pooling) were performed in the same way for all movements and all coding strategies.

Windowing

Each movement in the database can be represented as a matrix

$$\mathbf{A}_i = [\mathbf{a}_1 \ldots \mathbf{a}_{T_i}]$$

whose columns $\mathbf{a}_j = [a_{j1} \ldots a_{jD}]^\top$ represent the angles of each of the $D = 59$ degrees of freedom of the articulated human model (see Figure 1) at each frame $0 \leq j \leq T_i$.

Because each recording might be a different duration T_i, we needed a simple way to normalize the length of time across movement examples, while preserving the ability to evaluate different encoding methods. We accomplished this normalization by windowing the movement data, and then "pooling" across the windows. For the experiments reported here, we selected a window length of $L = 60$ frames (500 ms) and applied a hanning envelope along the time dimension of each window so that movements could be decomposed into overlapping windows without introducing significant ringing from the windowing process.

Whitening

Because kinematic representations are highly redundant, especially over short time spans, we whitened all windowed segments of movement information, so that global, lower–order correlations among joint angles within each window would be removed from the data before attempting to encode anything. For sparse codes, whitening is furthermore thought to improve the encoding process by ensuring that variables are approximately the same scale.

To compute the whitening transform, we extracted and mean-centered 100000 randomly selected windows of movement information from the CMU mocap database, and used the covariance Σ of these windows to compute a standard PCA whitening transform. Briefly, because the covariance is positive semi-definite,

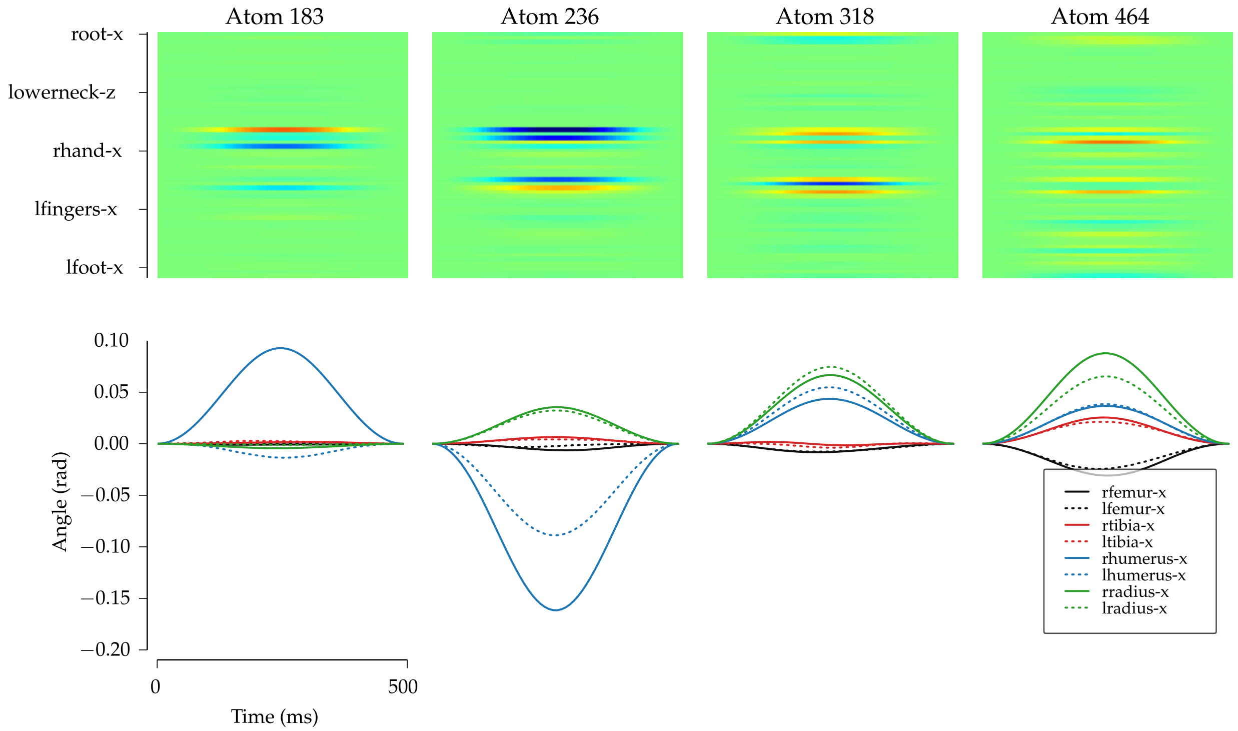

Figure 3: Learned codebook elements. The top row of plots shows a spectrogram of joint angles for all degrees of freedom over one time window; the bottom row of plots shows several specific channels from the corresponding codebook atom to demonstrate patterns across multiple degrees of freedom.

we compute the real-valued eigendecomposition $\mathbf{E}\Lambda = \Sigma\Lambda$, where the columns of $\mathbf{E}$ are eigenvectors and Λ is a diagonal matrix with eigenvalues corresponding to each eigenvector, arranged in decreasing order of magnitude. Given this decomposition, the whitening transform $\mathbf{W} = \mathbf{S}_K\mathbf{E}_K$ is the product of the first K eigenvectors (an orthonormal rotation matrix) and a diagonal matrix $\mathbf{S}_K$ whose elements are the inverse square roots of the first K eigenvalues, $s_{ii} = \frac{1}{\sqrt{\lambda_i}}$ (a scaling matrix).

We selected K for our data such that 99% of the variance in the windowed angle data was preserved. This process confirmed that the joint angle windows were highly redundant. For example, for a window length of 60 samples, the raw data length of 3540 variables compressed down to the first 34 principal components (a compression of more than 99%); the first principal component in the joint angle windows typically explained more than 50% of the variance in the data!

Max-Pooling

When classifying a segment of movement information, we extracted contiguous windows from the movement, whitened them, and encoded them using one of the coding techniques described below. However, due again to the variability in lengths of the different movement recordings, this process still resulted in a variable number of encoded windows per movement. To remove time from these movement representations, we adopted a "max-pooling" technique that is common in neural network models (Lee, Grosse, Ranganath, & Ng, 2009). Formally, if $\mathbf{z}_j = [z_{j1} \dots z_{jK}]^\top$ represents the encoded form of window j, then the max-pooled representation

$$\mathbf{z} = \left[\max_j(z_{j1}) \dots \max_j(z_{jK})\right]^\top$$

consists of the maximum values taken by each variable across time. By taking the maximum of the encodings in this way, the final representation that gets passed to the classifier (a) is a standard size, and (b) contains any feature that was present in any of its constituent windows—that is, at any point in time during the movement.

Computational Models

Because we wished to evaluate the effect of encoding when trying to classify movements, we used the "raw" whitened movement windows as an encoding baseline. However, because the windows were already whitened using PCA, which encodes data points according to their projections onto the principal components of the data, we could test the effect of reducing the dimensionality of the movement data by simply varying K.

Sparse Coding

PCA is widely used for data preprocessing, but using it as an encoding technique often results in "dense" representations of data that are difficult for humans to interpret. In contrast, sparse coding explicitly seeks a representation of data $\mathbf{Z}$ that uses as few elements of a given codebook $\mathbf{D}$ as possible to reconstruct the data $\mathbf{X}$. This goal can be formulated as an optimization problem:

$$\mathbf{Z} = \arg\min_{\mathbf{A}} ||\mathbf{DA} - \mathbf{X}||_2^2 + \lambda ||\mathbf{A}||_1$$

We used the "lasso" (Tibshirani, 1996) implementation of sparse coding provided by the `scikit-learn` Python package (Pedregosa et al., 2011) to perform the encodings. For the experiments reported here, we followed best practices from the literature (Mairal, Bach, Ponce, & Sapiro, 2009) and set $\lambda = \frac{1}{\sqrt{n}}$, where n is the number of variables in the whitened, windowed movement data.

Codebook Learning

Some existing evidence (Coates & Ng, 2011) suggests that the sparse coding algorithm itself is critical in the performance of sparse-coded classification systems, and that codebooks constructed from even random vectors can yield classifiers with good, although not necessarily state-of-the-art, performance. We tested this phenomenon with our datasets by constructing random codebooks and using them in conjunction with the sparse coding algorithm above.

Even though random codebooks might be effective for some tasks, all available evidence also indicates that the best performance is obtained by using a codebook specifically tuned to the problem at hand. Several algorithms have been proposed to learn the optimal codebook for sparse coding using just a dataset $\mathbf{X}$ (Smith & Lewicki, 2006; Mairal et al., 2009); for this work we adopted a formulation of the problem that factors the codebook $\mathbf{D}$ into the product of a matrix $\mathbf{W}$ with itself (Le et al., 2011):

$$\mathbf{D} = \arg\min_{\mathbf{W}} ||\mathbf{W}\mathbf{W}^\top\mathbf{X} - \mathbf{X}||_2^2 + \lambda ||\mathbf{W}^\top\mathbf{X}||_1$$

This formulation is particularly easy to optimize because of the linear coding and decoding process, and in our experiments it seems relatively robust to the choice of sparsity regularizer λ. We used the `theanets` Python package [2] for defining and optimizing the appropriate losses.

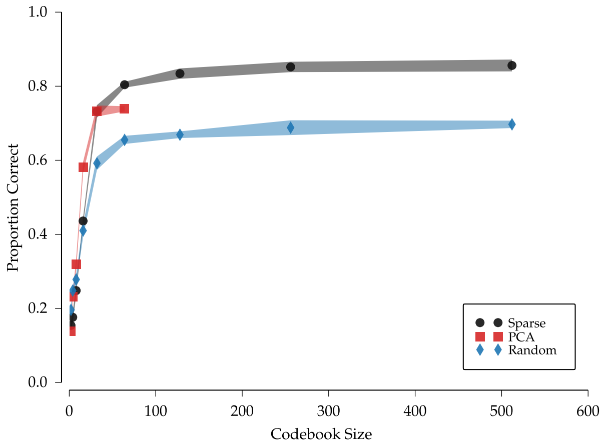

Figure 4: Classifier accuracy versus feature set size; in general, more features improve classifier accuracy. PCA is limited in its feature set size due to orthonormality constraints. Random codebooks (blue diamonds) are not well tuned to the dataset. Overcomplete codebooks with sparse–coded features are able to improve significantly on both baselines. Shaded regions show ±1 standard error using a 9-fold cross validation.

Classification

In preliminary experiments, we evaluated three separate classifiers—support vector machines (Boser, Guyon, & Vapnik, 1992), logistic regression, and random forests (Breiman, 2001)—for mapping from encoded movement data to movement labels. However, the three classifiers performed so similarly that we decided to focus on the logistic regression classifier, since its features are extremely easy to analyze after training. Briefly, a logistic regression uses a dataset $\mathbf{X}$ with labels $\mathbf{Y}$ to optimize parameters Θ so that the probability of the dataset is maximized, under the model $\log p(y_l|\mathbf{x}) \propto \theta_l \mathbf{x}$. We used the classifier implementation provided by `scikit-learn` for our experiments. Once the model is trained, the features with the largest weights θ_{lj} are the most indicative of class l.

Results

We first examined the effects of varying the coding algorithm and codebook size on overall classification performance; the results of these experiments are shown in Figure 4. Some features of these results were expected; in particular, the fact that PCA only discards information from the data results in a significant drop in performance as K shrinks, in addition to putting an upper bound on performance when $K = n$. Reducing K also hurt the other two coding strategies—sparse coding with a random codebook, and sparse coding with a learned codebook—at about the same rate. Similarly, in

[2] http://github.com/lmjohns3/theano-nets

		Random/512			PCA/32			Sparse/512		
Movement	Samples	P	R	F1	P	R	F1	P	R	F1
bear	6	0.846	0.579	0.688	0.647	0.579	0.611	0.857	0.632	**0.727**
cat	5	0.546	0.400	0.462	0.667	0.400	0.500	1	0.933	**0.966**
chicken	10	0.844	1	0.916	0.857	0.790	0.822	1	0.921	**0.959**
dinosaur	2	1	0.600	0.750	1	0.400	0.571	1	0.800	**0.889**
dog	5	0.875	0.700	**0.778**	0.583	0.700	0.636	0.583	0.700	0.636
dragon	3	1	0.300	0.462	0.889	0.800	0.842	0.909	1	**0.952**
elephant	5	1	0.625	0.769	0.900	0.563	0.692	0.889	1	**0.941**
ghost	2	1	0.333	0.500	0.857	1	**0.923**	0.500	0.500	0.500
hummingbird	3	1	1	**1**	1	1	**1**	1	1	**1**
insect	2	1	0.167	0.286	0.833	0.833	0.833	1	0.833	**0.909**
monkey	11	0.585	0.939	0.721	0.649	0.727	0.686	0.838	0.939	**0.886**
mouse	5	1	0.368	0.539	0.700	0.368	0.483	0.938	0.790	**0.857**
penguin	3	1	0.909	0.952	1	1	**1**	1	1	**1**
prairie dog	9	0.521	0.962	0.676	0.544	0.962	0.694	0.867	1	**0.929**
pterosaur	4	0.636	0.875	0.737	0.857	0.750	0.800	0.889	1	**0.941**
roadrunner	2	1	0.800	0.889	1	1	**1**	1	1	**1**
snake	5	0.667	1	0.800	0.600	0.750	0.667	0.833	0.833	**0.833**
squirrel	2	1	1	**1**	0.667	1	0.800	1	0.833	0.909
t-rex	3	1	0.250	0.400	1	0.500	0.667	1	1	**1**
whale	4	1	0.818	0.900	0.846	1	0.917	0.917	1	**0.957**
Average				0.661			0.725			**0.848**

Table 1: Summary of model performance in each of the 20 movement categories. The best F1 score for each category is highlighted in bold text. Average values are weighted by the size of each movement category.

agreement with existing research (Johnson, Cooper, & Ballard, 2013), the sparse coding algorithm seemed to benefit from being able to use an overcomplete, learned codebook. However, an unexpected finding from this experiment was that the sparse coding algorithm performed drastically worse when paired with a random codebook, even when the codebook was allowed to be more than 10× overcomplete.

The full results from the best sample of each coding strategy are presented in Table 1. These results provide some insight into the performance of each coder with respect to specific types of movement data. "Hummingbird" was the easiest category to predict for all models. "Dog" and "ghost" were the most difficult for the sparse coded features, but PCA features performed well on "ghost," while sparse coding with a random codebook performed well on "dog."

Random features tended to yield classification performance with high precision but low recall. This result is somewhat expected, since in high–dimensional spaces, and particularly when using a sparse coding algorithm, features from random codebooks will tend to be orthogonal to the data, thus only "firing" when a particular whitened movement window happens to fall nearby. This behavior provides high specificity for certain types of movements, but somewhat erroneously does not respond to most data. In contrast, the sparse coding algorithm had very high recall when paired with a learned codebook. This similarly provides support for the notion that the codebook learning process was able to identify the relevant manifolds in movement space; not only do the features in the tuned codebook fire often when presented with real movement data, an inspection of the most indicative features of each class revealed that the sparse features with a tuned codebook were able to partition the types of movement into distinct subspaces.

Unfortunately, there is no standard train/test dataset for motion recognition using the CMU database. However, the most recent research with comparable results (Parameswaran & Chellappa, 2006; Han, Wu, Liang, Hou, & Jia, 2010; Junejo, Dexter, Laptev, & Pérez, 2011; Shotton et al., 2011) indicates that, qualitatively, the performance of the encodings and classifier described here is competitive with other approaches. Since our work relies solely on kinematic (i.e., joint angle) representations of movement, comparing with human performance on the same classification task would be difficult.

While the results presented here hold promise that sparse codes might indeed be effective tools when dealing with kinematic action representations, there remains much work to be done in this area. In particular, sparse codes might effectively be combined with data from multiple modalities (e.g., depth-sensor readings) for action recognition. In addition, this work has touched only on kinematic representations of movement. Movements are generated by applying forces to a skeleton using muscles, but we know little about how this process is implemented. The question of whether efficient coding applies equally to dynamics is quite interesting and could reveal important new insights into how humans and animals move in their environments.

References

Barlow, H. (1961). Possible principles underlying the transformation of sensory messages. *Sensory Communication*, 217–234.

Bengio, Y. (2013). Deep learning of representations: Looking forward. *arXiv preprint arXiv:1305.0445*.

Boser, B., Guyon, I., & Vapnik, V. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on computational learning theory* (pp. 144–152).

Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5–32.

Coates, A., & Ng, A. (2011). The importance of encoding versus training with sparse coding and vector quantization. In *Proc. 28th Intl. Conf. on Machine Learning* (Vol. 8, p. 10).

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proc. 14th Intl. Conf. on Artificial Intelligence and Statistics.*

Han, L., Wu, X., Liang, W., Hou, G., & Jia, Y. (2010). Discriminative human action recognition in the learned hierarchical manifold space. *Image and Vision Computing*, *28*(5), 836–849.

Johnson, L., Cooper, J., & Ballard, D. (2013). Unified loss functions for multi-modal pose regression. In *Proc. Intl. Joint Conf. on Neural Networks.*

Junejo, I., Dexter, E., Laptev, I., & Pérez, P. (2011). View-independent action recognition from temporal self-similarities. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, *33*(1), 172–185.

Latash, M., Scholz, J., & Schöner, G. (2002). Motor control strategies revealed in the structure of motor variability. *Exercise and Sport Sciences Reviews*, *30*(1), 26–31.

Le, Q., Karpenko, A., Ngiam, J., & Ng, A. (2011). ICA with reconstruction cost for efficient overcomplete feature learning. *Advances in Neural Information Processing Systems*.

Lee, H., Battle, A., Raina, R., & Ng, A. (2007). Efficient sparse coding algorithms. *Advances in Neural Information Processing Systems*, *19*, 801.

Lee, H., Grosse, R., Ranganath, R., & Ng, A. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proc. 26th intl. conf. on machine learning* (pp. 609–616).

Mairal, J., Bach, F., Ponce, J., & Sapiro, G. (2009). Online dictionary learning for sparse coding. In *Proc. 26th Intl. Conf. on Machine Learning* (pp. 689–696).

Olshausen, B., & Field, D. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, *381*(6583), 607–609.

Olshausen, B., & Field, D. (2004). Sparse coding of sensory inputs. *Current Opinion in Neurobiology*, *14*(4), 481–487.

Parameswaran, V., & Chellappa, R. (2006). View invariance for human action recognition. *International Journal of Computer Vision*, *66*(1), 83–101.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, É. (2011). Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, *12*, 2825–2830.

Scholz, J., & Schöner, G. (1999). The uncontrolled manifold concept: identifying control variables for a functional task. *Experimental Brain Research*, *126*(3), 289–306.

Shotton, J., Sharp, T., Kipman, A., Fitzgibbon, A., Finocchio, M., Blake, A., … Moore, R. (2011). Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, *56*(1), 116–124.

Smith, E., & Lewicki, M. (2006). Efficient auditory coding. *Nature*, *439*(7079), 978–982.

Taylor, G., Hinton, G., & Roweis, S. (2007). Modeling human motion using binary latent variables. *Advances in Neural Information Processing Systems*, *19*, 1345.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 267–288.