Unified losses for multi-modal pose coding and regression

Leif Johnson and Joseph Cooper and Dana Ballard

Abstract— **Sparsity and redundancy reduction have been evaluated in machine learning primarily on classification tasks using datasets of natural images and sounds. In comparison, relatively little work has investigated the use of sparse codes for representing human movements and poses, or for using these codes in regression tasks. This paper defines a basic coding and regression architecture for evaluating the impact of sparsity when coding human pose information, and tests the performance of several coding methods within this framework for the task of mapping from a kinematic (joint angle) modality to a dynamic (joint torque) one. In addition, we evaluate the performance of unified loss functions defined on the same class of models. We show that, while sparse codes are useful for effective mappings between modalities, their primary benefit for this task seems to be in allowing for overcomplete codebooks. In addition, we make use of the proposed architecture to examine in detail the sources of error for each stage in the model under various coding strategies. Furthermore, we show that using a unified loss function that passes gradient information between stages of the coding and regression architecture provides substantial reductions in overall error.**

I. Overview

Sparsity, or redundancy reduction (Barlow, 1961), appears to play an important role when coding sensory data, because the space of all possible inputs (e.g., all possible 1000×1000 images) is not uniformly covered by the samples (e.g., photos or retinal inputs) that we tend to encounter in the natural world. Sparse codes are thought to be effective for representing sensory data that lie along low-dimensional manifolds because the basis vectors in the code can be used efficiently (i.e., using just a few nonzero coefficients) to indicate, for a particular data point, its location along the manifold rather than its coordinates in the higher-dimensional space.

In the machine learning literature, much of the recent work on these codes has focused on sensory data like images or sound, using classification tasks for evaluation (e.g., Ranzato, Boureau, & LeCun, 2007; Dahl, Ranzato, Mohamed, & Hinton, 2010). Concurrent research in motor control, however, has suggested that human movements might also lie along a relatively low-dimensional manifold embedded in the space of all possible movements (Scholz & Schöner, 1999; Latash, Scholz, & Schöner, 2002). Sparse codes might be useful, then, for representing information about movement and pose in humans, and such codes might be useful for regression tasks as well. To our knowledge, however, such codes have not been evaluated extensively on natural movement or pose data.

This paper proposes a basic architecture for testing the effectiveness of a broad class of coding techniques when mapping from kinematic (joint angle) to dynamic (torque) data in human poses. Because it is computationally straightforward, the model allows us to compare and evaluate several possible approaches to this coding and regression task. We show that, for the class of techniques captured by our model, sparsity is useful for representing and manipulating pose data, but only inasmuch as sparse codes tend to encourage overcomplete codebooks. Even though the absolute decoding error associated with sparse codes can be larger than the corresponding absolute error for dense codes, the information captured by each coefficient in a sparse code is larger than for dense codes. In this way, sparse codes appear to facilitate the task of mapping from one information modality to another. Finally, we combine the coding, regression, and decoding stages of our architecture into a single neural network–style loss function and show that substantially lower prediction errors can be achieved using an integrated system.

II. Problem setting

Humans are extremely competent at controlling their bodies, whether to mimic a conspecific's actions or to attain cognitive goals. However, studying human movements is difficult: the parameters describing movements are high-dimensional and time-varying, and, in addition, most of the quantities that are relevant for describing the control of movement are invisible to an outside observer. Even though we cannot measure all of the joint torques or angles during a complex, multijoint human movement, we can use motion capture (Figure 2) to measure the external aspects of movement, and convert these observations into angles and forces that would have been required for a simplified model of the human skeleton to effect the same movements (Cooper & Ballard, 2012). These computed angles and forces, while still a coarse proxy for the information that might be used by the central nervous system, constitute the data for this paper.

Theoretically, one could transform information from one modality into another by amassing a large quantity of corresponding data from these two modalities and computing regression coefficients directly. However, this is inefficient for at least two reasons. First, computing a regression between two datasets becomes increasingly problematic as the dimensionality of the data increases; this difficulty is compounded when there is noise in the data. Second, if the manifold hypothesis is accurate, then each modality of the raw data will have statistical redundancies that would need to be captured by the regression process, in addition to mapping between domains. Rather than working in the space of raw measurements, then, we hypothesize that manipulating or combining movement information is more efficient in a space defined by codes that somehow represent the raw signals (Srivastava & Salakhutdinov, 2012). The questions addressed by this paper are, which types of codes are most efficient

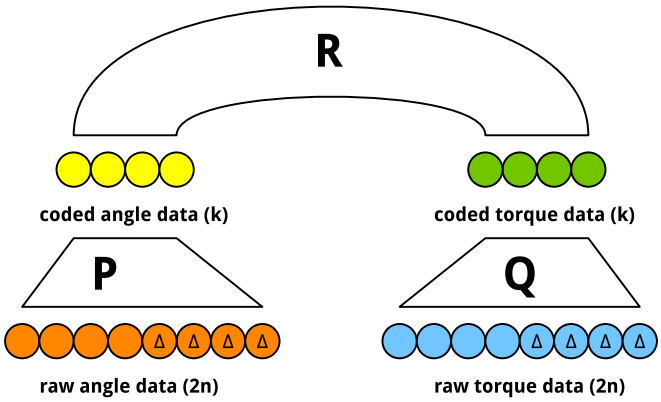

Fig. 1

INFORMATION PROCESSING ARCHITECTURE FOR MULTI-MODAL CODING AND REGRESSION. INFORMATION FROM A FRAME OF ONE MODALITY OF POSE DATA, SUCH AS ANGLES (ORANGE), IS TO BE MAPPED ONTO INFORMATION FROM ANOTHER MODALITY, SUCH AS TORQUES (BLUE). THIS MAPPING IS ACCOMPLISHED BY CODING A FRAME OF ANGLE DATA, AUGMENTED WITH ITS DERIVATIVE Δ, USING PARAMETERS $\mathbf{P}$; LIKEWISE, TORQUES AUGMENTED WITH DERIVATIVES Δ ARE ENCODED USING PARAMETERS $\mathbf{Q}$. FINALLY, A PARAMETRIC REGRESSION $\mathbf{R}$ IS COMPUTED BETWEEN THE CODES (YELLOW AND GREEN).

for processing information about movement, and which loss functions generate the best codes for computing regression between modalities?

III. POSE CODING AND REGRESSION

We assume that we have a set of data that represents kinematic and dynamic views of human motion, modeled using an articulated body with n degrees of freedom, and measured over a consecutive sequence of m discrete time steps. Formally, we represent a sequence of raw joint angles as a matrix $\mathbf{B} \in \mathbb{R}^{n \times m}$, where each column $\mathbf{b}^{(t)}$ represents a single frame of angle data. Similarly, we represent a sequence of raw joint torques as a matrix $\mathbf{U} \in \mathbb{R}^{n \times m}$ whose columns $\mathbf{u}^{(t)}$ each contain a frame of torque data. We define these matrices as complementary views of a single motion trajectory, so that for any frame t, the joint angles $\mathbf{b}^{(t)}$ correspond to the torques $\mathbf{u}^{(t)}$.

As mentioned above, movement is complex to model because it is high-dimensional (n is often large) and varies over time (m is often large). Rather than attempt to tackle both of these challenges at once, we simplify the modeling task here by considering the task of mapping between these two modalities for single poses (frames). Such a simplification makes the modeling task obviously difficult, since a single frame of kinematic pose data, for instance, does not indicate the direction in which the joint angles will be changing in subsequent frames. To address this issue, we make use of a common technique from speech recognition (Picone, 1993) and augment each of the raw data frames in our system with its first derivative. This provides information about the rates at which the angles and torques are changing, which could be useful when trying to compute torques on the basis of angles. The augmented data matrices $\mathbf{A}, \mathbf{V} \in \mathbb{R}^{2n \times m}$ are then defined as

$$\mathbf{A} = \begin{bmatrix} \mathbf{B} \\ \Delta\mathbf{B} \end{bmatrix} \quad \text{and} \quad \mathbf{V} = \begin{bmatrix} \mathbf{U} \\ \Delta\mathbf{U} \end{bmatrix},$$

where $\Delta\mathbf{X}^{(t)} = (\mathbf{x}^{(t+1)} - \mathbf{x}^{(t-1)})/2$ represents a secant approximation to the derivative at each frame.

Having created a set of kinematic and matched dynamic data describing sequences of human poses, we propose an information processing architecture for computing regressions from angles to torques. In this framework (Figure 1), a single frame of n input angles, augmented with its derivative, is encoded first into k angle-code coefficients using a coding model characterized by parameters $\mathbf{P} \in \mathbb{R}^{k \times 2n}$. Then a regression model characterized by parameters $\mathbf{R} \in \mathbb{R}^{k \times k}$ transforms the k angle-code coefficients into a k torque-code coefficients. Finally, this torque encoding is converted back into a frame of n torques, augmented with its first derivative, by inverting the torque coding model characterized by parameters $\mathbf{Q} \in \mathbb{R}^{2n \times k}$. If the manifold hypothesis holds for human pose data, then code parameters $\mathbf{P}$ and $\mathbf{Q}$ can be learned independently, because these parameters will describe the structure of the manifold for each modality of pose data; codes for each manifold should then be useful for a wide variety of other information processing tasks (Vincent, Larochelle, Lajoie, Bengio, & Manzagol, 2010). Regression parameters $\mathbf{R}$ can then be learned using the encoded data from each modality.

Given our parametric framework for coding and regression, the coding approaches considered here all assume a finite codebook $\mathbf{D} \in \mathbb{R}^{2n \times k}$ whose columns $\mathbf{d}_i$ each represent a basis vector that is used in some way to encode data. This paper does not focus specifically on learning the codebook, though it does describe a few approaches to codebook learning below.

Separating the model into isolated coding and regression stages brings three advantages to the problem at hand. First, it allows us to manipulate the number of parameters in the model in a controlled way. The multistage model contains $k^2 + 4kn$ parameters, while direct regression requires $4n^2$ parameters. When $k < 2n$, the multistage model has fewer parameters than direct regression, but when k exceeds the dimensionality of the data, the multistage model has more parameters. Models with more parameters tend to be more accurate, but they might overfit the data and capture more noise than desired. Second, separate modules for coding in each modality, and regression between codes, allows for in-depth analysis of the performance of each module: codes for one modality that provide for low decoding error could also be ones that do not permit easy regression, for example. Finally, defining distinct coding modules permit an analysis of the degree to which coding, in isolation, provides an efficient representation of the data.

A. Coding algorithms

Mathematically, this paper treats coding as a general term for transforming a vector of raw data $\mathbf{x} \in \mathbb{R}^{2n}$ into another vector of coefficients $\mathbf{z} \in \mathbb{R}^k$, such that $\mathbf{z}$ contains sufficient information to recover $\mathbf{x}$ with some tolerated level of error. More formally, coding is often defined in terms of minimizing a cost function

$$\|g(\mathbf{D}, \mathbf{z}) - \mathbf{x}\|_2^2 + \lambda R(\mathbf{z})$$

where $g(\mathbf{D}, \mathbf{z})$ refers to a decoding operation that converts coefficients $\mathbf{z}$ into an estimate of the raw data $\hat{\mathbf{x}}$, and R is a regularizer that can be chosen to prevent overfitting, promote sparsity in the code, etc. For this paper, we evaluate several approaches to coding, each described briefly below.

1) PCA: Principal component analysis (PCA) is widely used as a data preprocessing technique, but here we use PCA to refer to an encoding that simply computes the inner product of a data point $\mathbf{x}$ with each of the codebook vectors $\mathbf{d}_i$: that is, $\mathbf{z} = \mathbf{D}^T\mathbf{x}$.

The PCA codebook consists of the eigenvectors of the covariance matrix of the data, sorted in decreasing order of magnitude of the corresponding eigenvalue. By retaining only the first k eigenvectors in the codebook, PCA retains the maximally varying components of the data first, and progressively refines the error by retaining smaller components.

Used in this way, PCA implicitly models the underlying data as a multivariate normal distribution. Because the codebook for PCA is composed of eigenvectors (which are orthogonal to each other), there can be at most as many codebook vectors as dimensions in the input data.

2) K-means: K-means (MacQueen et al., 1967) can be seen as an extremely sparse coding technique that represents a data point $\mathbf{x}$ using only the closest basis vector in the codebook: for this approach, $\mathbf{z} = [\xi_1 \dots \xi_k]^T$ such that

$$\xi_i = \begin{cases} 1 & \text{if } \|\mathbf{x} - \mathbf{d}_i\|_2 < \|\mathbf{x} - \mathbf{d}_j\|_2 \text{ for } j \neq i \\ 0 & \text{otherwise.} \end{cases}$$

The codebook corresponding to this coding approach is learned from the data by setting the $\mathbf{d}_i$ to random elements from the training data, and then iteratively adjusting the columns of $\mathbf{D}$ so that the sum of the Euclidean distances from each data point to its closest codebook vector is minimized.

3) Sparse coding: Sparse coding (Tibshirani, 1996), also called lasso regression, computes the coefficients $\mathbf{z}$ for data point $\mathbf{x}$ by minimizing a least-squares cost function with a regularization penalty on the magnitude of the code coefficients:

$$\mathbf{z} = \arg\min_{\zeta} \|\mathbf{D}\zeta - \mathbf{x}\|_2^2 + \lambda \|\zeta\|_1 .$$

λ is a parameter that controls the tradeoff between accurate representation and sparsity; following results from Mairal, Bach, Ponce, and Sapiro (2009), we set $\lambda \propto 1/\sqrt{n}$.

Coates and Ng (2011) reported that sparse coding worked well across many types of codebooks for their tasks (image classification). For this coding algorithm, then, we tested several different codebooks: random, sampled, and learned. The random codebook consisted of IID vectors drawn from the standard normal distribution, normalized to unit length. The sampled codebook contained samples drawn uniformly from the training data, also normalized to unit length. The learned codebook used a fast, online algorithm (Mairal et al., 2009) to tune the codebook to the data. Briefly, the general algorithm is a variation of coordinate descent, where sparse encoding computations are alternated with codebook updates. The codebook learning process attempts to minimize the lasso cost function above, both with respect to the codes $\mathbf{z}$ and also with respect to the codebook $\mathbf{D}$.

B. Regression

Once codes have been computed for the source and target datasets, the next task is to compute a regression matrix $\mathbf{R}$ that will convert coefficients from one modality into coefficients from another. We used ridge regression (Hoerl & Kennard, 1970) to compute the best parameters for inferring coefficients across coded modalities. We can express the regression task between codes $\mathbf{z}_\alpha$ and $\mathbf{z}_\beta$ as optimizing the cost function

$$\|\mathbf{R}\mathbf{z}_\alpha - \mathbf{z}_\beta\|_2^2 + \lambda\|\mathbf{R}\|_F^2$$

where λ captures the degree to which the modeler is willing to tolerate large values in $\mathbf{R}$ when explaining the observed data. Essentially, ridge regression is the same as linear regression, but it adds a penalty on large values of the coefficients that are used to describe the data. In our experiments, the value of λ was set empirically by cross-validation on the training set.

IV. Unified loss function

The framework presented above consists of a set of parameters for each of the three tasks in mapping from one modality of data to another: $\mathbf{P}$ and $\mathbf{Q}$ contain the parameters for encoding the source and target data vectors, respectively, and $\mathbf{R}$ contains the parameters for mapping from the source code to the target code.

In the architecture described so far, these parameters are trained separately and only combined at test time to evaluate their efficacy on the entire data mapping pipeline. While advantageous for the reasons discussed above—separation of tasks for straightforward analysis, representation of data using hypothesized underlying manifolds—this isolated coding strategy presents a fundamental learning difficulty, namely, that errors introduced into any of the stages of the model cannot be accounted for by other stages of the model. This is particularly problematic when a specific task, such as the multimodal regression task considered in this paper, might benefit from a slightly different coded representation than the manifold (or other sparse code) representation. Consider, for example, a code for joint angles that maps each joint–angle vector onto a coded representation that is optimal for reconstruction.[1] This coding approach, while powerful for many tasks, probably captures information about the

[1] As shown above, reconstruction is often the criterion for obtaining coefficients under such coding regimes.

"principal axes" of the manifold on which the data lies. However, these principal directions might not be relevant or important for a specific task like computing a regression to another coded information modality.

We can remedy this separation of errors by taking advantage of the large body of knowledge from the neural networks community. We define a unified loss function that retains the general architecture presented earlier—and in particular has the same parameterization—but allows errors in each stage to flow across parameter boundaries:

$$\mathcal{L} = \|\mathbf{Q}g\left(\mathbf{R}g\left(\mathbf{P}\mathbf{x}\right)\right) - \mathbf{y}\|_2^2$$

where $\mathbf{x}$ and $\mathbf{y}$ are input and target vectors, respectively, and $g(\cdot)$ is a nonlinear activation function that prevents the model from collapsing to a single linear operation. This unified loss function can be used to train all parameters in the network simultaneously, which permits errors introduced by one parameter to be compensated for by other parameters.

A. Nonlinearities

The only issue introduced by the unified loss is which nonlinearity to choose. In this paper we evaluate two methods, one traditional and one recently popularized by work in the feature learning community.

1) Logistic: Traditionally, g is set to be the logistic function $g(z) = 1/\left(1 + e^{-z}\right)$, because this function is bounded and has an easily computed, continuous derivative. However, logistic activations suffer from the problem of vanishing gradients (Bengio, Simard, & Frasconi, 1994), in which networks with many layers—even, in practice, more than two layers—erroneously appear to converge during training because first-order gradient computations reveal negligible slope, while in fact the parameter settings might be located on a broad saddle point of the loss. This problem has been addressed recently by Martens (2010) and Martens and Sutskever (2011), who developed a practical, second-order Hessian-free training method for neural network models. This learning method is capable of dramatically improving parameter settings for neural network models that appear to have converged using first-order gradient descent, so we use it in our experiments to train unified networks with logistic hidden unit activations.

2) Rectified linear: A parallel line of work in the deep learning community (Glorot, Bordes, & Bengio, 2011) has recently emphasized the performance advantages of a different kind of nonlinearity, the relu or rectified linear unit $g(z) = \max(0, z)$. This activation function addresses the vanishing gradient problem by having a constant derivative for $z > 0$. As a side benefit, networks of rectified linear units tend to be naturally sparse, because $g(z) = 0$ (a true zero, not just a small nonzero value) for $z \leq 0$. In practice, rectified linear networks can be trained using simple first-order learning techniques, and they tend to perform equivalently to large networks of logistic sigmoid units that require significantly more effort to train.

Using a rectified activation function, the unified loss becomes surprisingly simple:

$$\mathcal{L} = \|\mathbf{Q}\left[\mathbf{R}\left[\mathbf{P}\mathbf{x}\right]_+\right]_+ - \mathbf{y}\|_2^2$$

where $[\cdot]_+$ denotes the rectification nonlinearity. For some input $\mathbf{x}$, some of the units in the first hidden layer of the network will have negative pre-activations and will be set to zero by the nonlinearity, while the remainder will simply pass their activations along to the next layer of the network. This process repeats for all hidden layers, effectively using the nonlinearity as a switch to select a subset of network nodes to encode, regress, and decode each input. Using $\mathbf{z}_+$ to denote only the nonzero elements of $\mathbf{z}$, the network output for $\mathbf{x}$ becomes a linear operation $\hat{\mathbf{y}} = \mathbf{Q}_+\mathbf{R}_+\mathbf{P}_+\mathbf{x}$. This switching behavior retains the speed and simplicity advantages of linear codes, while simultaneously packing exponentially many such linear codes into one model (Nair & Hinton, 2010).

B. Training

As mentioned above, parameters in neural networks are commonly optimized using one of two learning algorithms. For this work, all network parameters were first adjusted using first-order stochastic gradient descent (SGD), which computes parameter updates as the mean of the gradient over a mini-batch of r training examples

$$\Delta\theta_b = \frac{1}{r}\sum_{i=1}^{r}\frac{\partial\mathcal{L}}{\partial\theta}\left(\mathbf{x}^{(br+i)}\right)$$

and then updates parameters using a small learning rate α: $\theta_{b+1} = \theta_b + \alpha\Delta\theta_b$. Usually, α is reduced over time; for our experiments, we started with $\alpha = 0.1$ and reduced this value by 1% after every mini-batch.

For rectified linear networks, training stopped when this process converged to a region of zero gradient, because these networks' second derivatives are zero. For the logistic networks, however, second-order information can be used to compute further parameter updates once an apparent minimum has been reached. After convergence of first-order SGD, then, we further updated parameters in these networks using a Hessian-free algorithm (Martens, 2010; Martens & Sutskever, 2011). Briefly, this algorithm uses conjugate gradient (CG) to compute a parameter update $\mathbf{q}_b = \arg\min_{\mathbf{p}} \mathbf{H}\mathbf{p} + \lambda\mathbf{p} - \Delta\theta_b$, where $\mathbf{H}\mathbf{p}$ is an implicit computation of the full Hessian in a specific direction $\mathbf{p}$. Parameters are then updated without a learning rate: $\theta_{b+1} = \theta_b + \mathbf{q}$. This training regimen is able to reduce error in "converged" sigmoid networks by a large amount (Figure 5).

V. Experiments

To measure and collect human movement data, we used a 16–camera Phasespace [2] motion capture system in conjunction with a standard treadmill (Figure 2). Human subjects in the motion tracking area wore a full-body suit equipped

[2] phasespace.com/impulse_motion_capture.html

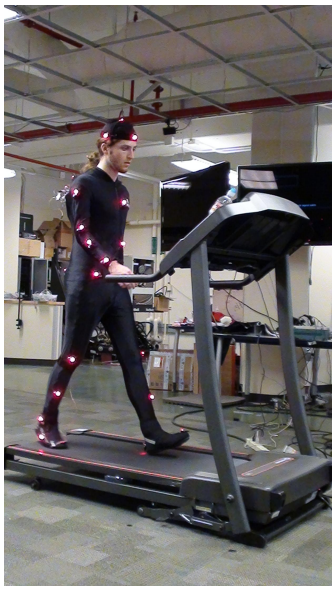

Fig. 2

THE MOTION CAPTURE ENVIRONMENT CONSISTS OF A FULL-BODY MOTION CAPTURE SUIT (BLACK, WITH RED LEDS), AND A TREADMILL CENTERED IN THE MOTION CAPTURE SPACE.

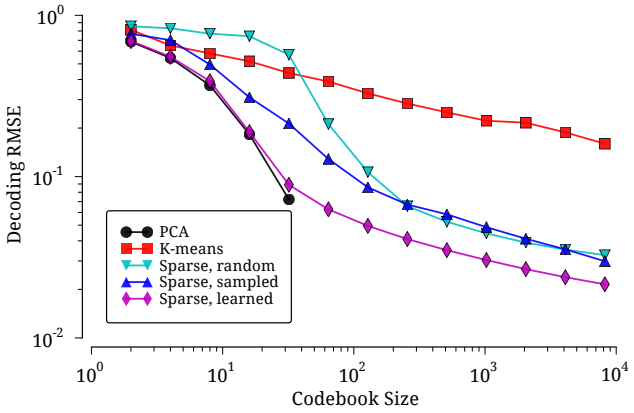

Fig. 3

MEAN RMS DECODING ERROR FOR JOINT TORQUES, MEASURED WITH RESPECT TO THE SIZE OF THE CODEBOOK. LARGER CODEBOOKS RESULT IN CODES THAT CAPTURE MORE OF THE VARIANCE IN THE DATA, EVEN WHEN THE CODEBOOK IS CREATED USING IID STANDARD NORMAL SAMPLES. A LOG SCALE HAS BEEN USED ON BOTH AXES TO REVEAL TRENDS MORE CLEARLY.

with active-pulse LED motion tracking markers and were recorded as they walked and ran on the treadmill at a variety of speeds.

For the results reported here, we recorded the positions of $L = 48$ markers from one subject as he walked at speeds ranging from 0.22 to 2.68 m/s. The recording lasted twenty minutes. The Phasespace system produces frames of motion capture data at a rate of 120Hz, so this recording resulted in more than 120,000 frames of raw motion-capture data. These frames were processed using the articulated forward model proposed by Cooper and Ballard (2012), resulting in three sequences of measurements for the observed motion: the sequence of interpolated marker positions $\mathbf{X} = \left[\mathbf{x}^{(1)} \dots \mathbf{x}^{(N)}\right]$ representing the positions of the segments of the articulated model over time; the sequence of observed angles $\mathbf{A} = \left[\mathbf{a}^{(1)} \dots \mathbf{a}^{(N)}\right]$ for each of the 54 degrees of freedom in the model; and the corresponding torques $\mathbf{V} = \left[\mathbf{v}^{(1)} \dots \mathbf{v}^{(N)}\right]$ that were necessary to cause those angles to move through the observed dynamic trajectory of the model.

A. Preprocessing

For this paper, we were concerned with mapping angles to torques, so we discarded the marker data $\mathbf{X}$. To obtain datasets for training and testing the coding and regression models, we needed to perform some preprocessing to obtain matched sets of frames that would permit a fair comparison.

First, the sequences obtained from the model were smoothed by convolving each channel in each modality with a 5-sample (42 millisecond) rectangular window over time. After smoothing, each channel of the data was normalized by subtracting out the mean value and dividing by the standard deviation. These steps ensured that the data did not contain residual noise due to marker dropouts, and also that the data values were all approximately the same scale.

Each frame of data was then augmented with an approximation of its first derivative by calculating the secant approximation of these quantities using the neighboring two frames. [3]

Next, the smoothed, normalized, derivative-augmented frames were segmented into three distinct regions, each containing 24000 frames (200 seconds) of data: the first (segment A) consisted of slow walks, the second (segment B) consisted of fast walks, and the third (segment C) consisted of running movements. To evaluate the coding and regression models, each segment was further partitioned into disjoint training, validation, and test sets such that 10% of frames from each segment were used for validation, 10% were used only for testing, and the remainder were available for training.

B. Coding efficiency

We first analyzed the performance of the different coding techniques discussed above when reconstructing the raw torque data using the torque codes. Formally, after training the dictionaries as needed, we computed $\mathbf{z}$ for each frame of augmented torque data $\mathbf{v}$ in the test set, and then computed the decoding operation to obtain an estimate $\hat{\mathbf{v}}$. The decoding error $e_{\mathbf{v}}$ was then defined as the RMS value of the residual:

$$e_{\mathbf{v}} = \sqrt{\frac{1}{n}\|\hat{\mathbf{v}} - \mathbf{v}\|_2^2}.$$

[3]The first frame was dropped from each dataset to match the number of frames of data with the number of frames of derivative.

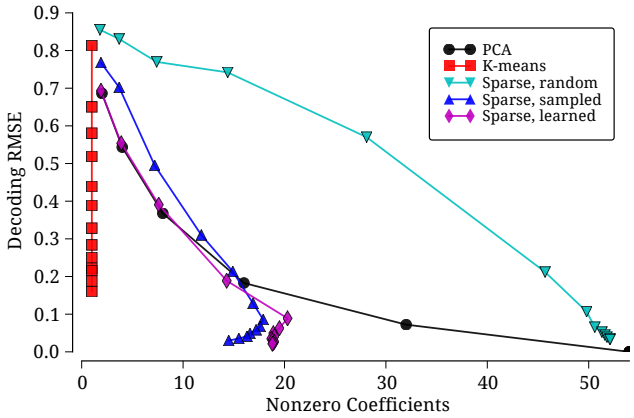

Fig. 4

MEAN RMS DECODING ERROR FOR JOINT TORQUES UNDER ISOLATED CODING STRATEGIES, MEASURED PER NONZERO COEFFICIENT IN THE ENCODING. SPARSE CODES LIKE LASSO REGRESSION WERE MORE EFFECTIVE, PER COEFFICIENT, THAN DENSE CODES LIKE PCA, BUT ONLY WHEN THE CODEBOOK WAS TUNED TO THE DATASET.

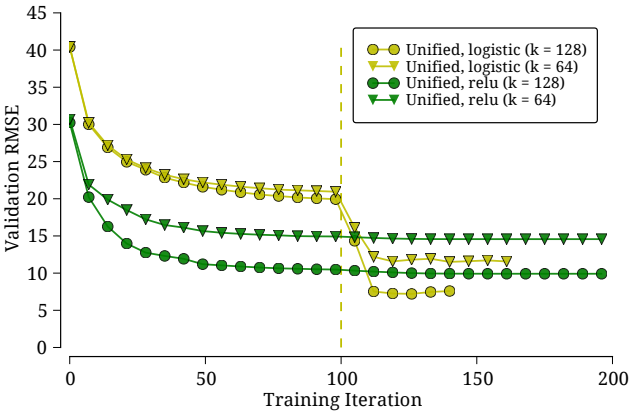

Fig. 5

LEARNING CURVES FOR TWO EXAMPLE NEURAL NETWORKS USING THE UNIFIED LOSS. ALL NETWORKS WERE TRAINED FIRST USING FIRST-ORDER STOCHASTIC GRADIENT DESCENT, WHILE THE LOGISTIC NETWORKS WERE ADDITIONALLY TRAINED WITH A HESSIAN-FREE SECOND-ORDER METHOD AFTER ITERATION 100.

Figure 3 shows the mean RMSE for each coding approach, measured with various sizes of codebooks, and applied solely to the torque data. (Results for the angle data were similar.) Unsurprisingly, larger codebooks were able to capture more of the variance in the data than smaller codebooks, regardless of the coding method. Perhaps more interesting, however, was the finding shown in Figure 4: when measured by the number of nonzero coefficients used in the code, sparse codes produced more accurate reconstructions than dense codes. This was somewhat vacuously true of K-means, since it only uses 1 coefficient for each $\mathbf{z}$; in comparison, however, this was not true for sparse coding combined with the random codebook.

C. Unified loss performance

The two nonlinearities that we evaluated in the unified loss showed different behavior during both training and testing. The rectified linear activation was much faster to train because it achieved low error using a simple first-order gradient method, while the logistic activation function required enormous amounts of time to compute the second-order Hessian-free learning method, which was started after the first-order method appeared to plateau. The logistic network training process required so much time to train, in fact, that it did not complete for $k = 2048$.

Sample learning curves for these networks are shown in Figure 5; these curves reveal two trends that occurred at nearly all network sizes during learning. First, the first-order stochastic gradient training method appears to plateau for logistic networks at a higher error rate than for rectified linear networks of the same topology. Second, applying a second-order Hessian-free learning method to these trained networks resulted in large performance improvements for logistic networks, but no improvement for rectified linear ones.

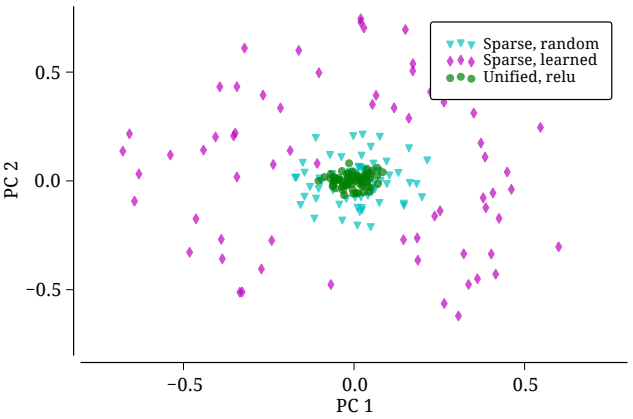

Fig. 6

LEARNED ANGLE CODEBOOKS, PROJECTED ONTO THE FIRST TWO PRINCIPLE COMPONENTS OF THE ANGLE DATA. VALUES NEAR THE ORIGIN INDICATE CODEBOOK VECTORS THAT DO NOT CORRELATE STRONGLY WITH EITHER OF THE FIRST TWO PRINCIPAL COMPONENTS OF THE ANGLE DATA.

D. Feature correlations

Because the processing architecture described here shares a common parametric formulation, we can analyze the parameter groups $\mathbf{P}$, $\mathbf{Q}$ and $\mathbf{R}$ across learning algorithms. For instance, across models we can treat $\mathbf{P}$ as being responsible for "encoding" the input angle vector, and examine the elements of this matrix in that light, regardless of whether the matrix in question was learned using the unified or the isolated losses. To get some idea of how the features in each

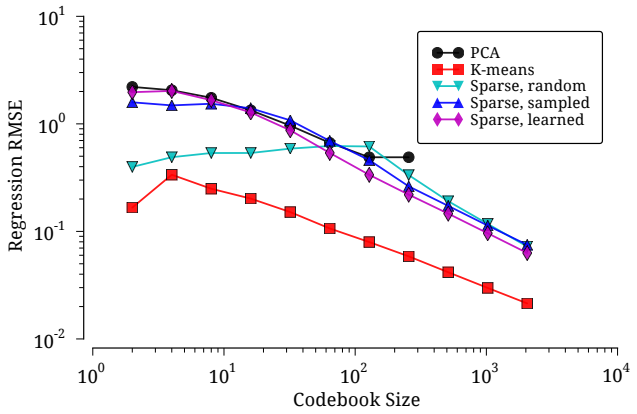

Fig. 7

RMS REGRESSION ERROR FOR ISOLATED CODING STRATEGIES, MEASURED WITH RESPECT TO ENCODED TORQUE VALUES. A LOG SCALE HAS BEEN USED ON BOTH AXES TO REVEAL PATTERNS MORE CLEARLY.

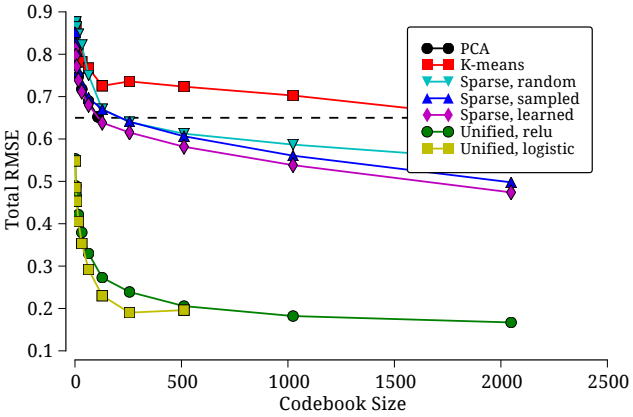

Fig. 8

MEAN OVERALL RMS ERROR, MEASURED WITH RESPECT TO RAW TORQUE VALUES FOR EACH FRAME. THE DASHED BLACK LINE SHOWS THE BASELINE RMS ERROR, COMPUTED USING RIDGE REGRESSION DIRECTLY BETWEEN RAW ANGLES AND RAW TORQUES. LOGISTIC NEURAL NETWORKS FOR $k > 1000$ FAILED TO CONVERGE DURING TRAINING.

algorithm self-organize, we mapped each of our learned angle codebooks onto the first two principal components of the angle data (Figure 6). Codebooks differed noticeably across algorithms; for instance, random features tended not to be strongly correlated with any principal component of the data, since random vectors in high-dimensional spaces tend to be orthogonal to each other. In comparison, a learned dictionary like the one obtained from sparse coding tended to model more strongly the space defined by the principal components. Finally, the differences in learned features between the two neural network models was striking: networks of rectified linear units have the same correlation with the principal components as a random set of vectors, while networks with logistic activations tended to have features that were nearly always orthogonal to the principal components.

E. Predicting torques from angles

In addition to comparing the effectiveness of different coding schemes for torque data, we also used our framework to compare the encoding methods in a larger context, namely predicting torque values on the basis of angle values. From one perspective, this task could be seen as a coarse approximation for a control task: given a target kinematic pose, what are the torques that would be associated with that pose?

Because the analysis framework proposed in this paper breaks this task into three separate stages—encoding, regression, and decoding—we could analyze, for the isolated coding strategies, the regression component of the task separately from the other components. In general, we found that RMS error for the regression task alone (Figure 7) followed the same pattern as errors for the encoding and decoding components: larger codebooks tended to yield lower errors. However, sparsity played a critical role in the regression task between two codes, since K-means yielded the lowest regression errors, while PCA yielded relatively large errors. Intuitively, sparse codes should be easier to perform regression between than dense codes, since there are more zeros in sparse source and target vectors.

Bringing together all stages of the framework, we compared the overall torque regression error for the entire coding and regression pipeline, as measured by comparing the outputs from our processing model with the true torques measured during the experiment (Figure 8). As a baseline, we computed a direct regression from angle to torque data: this resulted in an RMS error of 0.65 on the test set. PCA performed below baseline for undercomplete codebooks, and at baseline for complete codebooks, which is unsurprising since PCA simply rescales the original data. However, some of the sparse coding approaches were able to outperform PCA by a large margin (up to 30% reduction in error). In particular, using lasso coding combined with a large, learned dictionary produced lower RMS errors than any of the other isolated coding approaches examined here.

Finally, we compared the performance of models trained using a unified loss with the isolated coding strategies. The unified loss function dramatically outperformed even the best sparse coding strategy, suggesting that coding alone (or, more specifically, coding based on optimizing a reconstruction loss) is insufficient or inappropriate for the specific regression task that we used in this paper. This is intuitively clear from the top group of results in Figure 8: while large codebooks and sparse codes are able to outperform the baseline approach, the absolute RMS error for isolated approaches remains relatively high. In contrast, a unified loss that can correct for errors across separate coding, regression, and decoding stages is better able to model the task at hand. Interestingly, one exception to this claim seems to be the performance of large networks of sigmoidal units—such

networks on our task actually underperformed other models (in addition to being significantly slower to train), suggesting that the loss landscape for these models might have many poor local optima that are difficult to avoid even with a second-order learning approach.

VI. Conclusion

This paper presented an efficient coding and regression model for human pose information and used this model to examine the performance of several coding algorithms on human pose information. The model allowed us to examine separately the errors in coding information about poses and in computing regressions from one modality to another. We learned that even though some approaches produce extremely low coding and decoding errors, and other approaches were conducive to learning regressions between codes, in order to perform well on the task of predicting information across information modalities, a coding approach must have extremely low error on both tasks.

In addition, combining all data processing parameters into a single, unified loss function significantly increases the performance of the processing pipeline, by both incorporating a nonlinear data transformation (even if this transformation is only a "switch"), and by allowing error information to propagate throughout the pipeline. This discrepancy in performance is interesting in light of recent results from the feature learning community (e.g., Hinton & Salakhutdinov, 2006), in which features that are learned during a purely unsupervised pre-training phase tend to be effective for many tasks. Our results suggest that more information is needed about what task is to be performed, and how the features are to be used.

In several ways this paper is just a first look at this sort of modeling on human pose information. In particular, we limited our examination of human pose information to snapshots of single moments in time. Movement, however, is fundamentally dynamic, so we plan to expand the techniques presented here to temporal sequences of poses, by learning codes for entire movements.

VII. Acknowledgements

The authors would like to thank the anonymous reviewers for several insightful comments and suggestions that greatly improved this paper.

References

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

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166.

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).

Cooper, J., & Ballard, D. (2012). Realtime, physics-based marker following. In *Proc. Motion in Games* (pp. 350–361). Springer.

Dahl, G., Ranzato, M., Mohamed, A., & Hinton, G. (2010). Phone recognition with the mean-covariance restricted boltzmann machine. *Advances in neural information processing systems*, *23*, 469–477.

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

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

Hoerl, A., & Kennard, R. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, *12*(1), 55–67.

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.

MacQueen, J., et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkeley Symposium on Mathematical Statistics and Probability* (Vol. 1, p. 14).

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).

Martens, J. (2010). Deep learning via hessian-free optimization. In *Proc. 27th Intl. Conf. on Machine Learning* (Vol. 951, p. 2010).

Martens, J., & Sutskever, I. (2011). Learning recurrent neural networks with hessian-free optimization. In *Proc. 28th Int. Conf. on Machine Learning.*

Nair, V., & Hinton, G. (2010). Rectified linear units improve restricted boltzmann machines. In *Proc. 27th Intl. Conf. on Machine Learning* (pp. 807–814).

Picone, J. (1993). Signal modeling techniques in speech recognition. *Proc. IEEE*, *81*(9), 1215–1247.

Ranzato, M., Boureau, Y., & LeCun, Y. (2007). Sparse feature learning for deep belief networks. *Advances in neural information processing systems*, *20*, 1185–1192.

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

Srivastava, N., & Salakhutdinov, R. (2012). Multimodal learning with deep boltzmann machines. In *Advances in neural information processing systems 25* (pp. 2231–2239).

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

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, *11*, 3371–3408.