Survey of Multiview Representation Learning Techniques

Corbin Rosset
Dept. of Computer Science
Johns Hopkins University
Baltimore, MD 21218
crosset2@jhu.edu

Neil Mallinar
Dept. of Computer Science
Dept. of Mathematics
Johns Hopkins University
Baltimore, MD 21218
nmallin1@jhu.edu

Akshay Srivatsan
Dept. of Computer Science
Dept. of Applied Math and Stats
Johns Hopkins University
Baltimore, MD 21218
asrivat1@jhu.edu

Abstract

This study surveys state of the art methods for scalable multi-view representation learning. We compared kernel and deep neural network techniques for canonical correlation analysis (CCA) to learn representations for each view that jointly maximize total correlation, as well as split autoencoders that attempt to learn a shared representation of the views. We compared these various methods' performance on the Wisconson X-Ray Micro Beam dataset in terms of classification accuracy and also quality of clusters in the new representation. All of our deep learning was implemented in Google's newly released TensorFlow library, which obviated the need for explicit gradient computation.

1 Introduction

It is common in modern data sets to have multiple views of data collected of a phenomenon, for instance, a set of images and their captions in text, or audio and video data of the same event. If there exist labels, the views are conditionally uncorrelated on them, and it is typically assumed that noise sources between views are uncorrelated so that the representations are discriminating of the underlying semantic content. To distinguish it from multi-modal learning, multi-view learning trains a model or classifier for each view, the application of which depends on what data is available at test time. Typically it is desirable to find representations for each view that are predictive of - and predicted by - the other views so that if one view is not available at test time, it can serve to denoise the other views, or serve as a soft supervisor providing pseudo-labels. The benefits of training on multiple views include reduced sample complexity for prediction scenarios [1], relaxed separation conditions for clustering [2], among others [3][4][5].

Canonical Correlation Analysis (CCA), developed by Hotelling in 1936, is a widely used procedure motivated by the belief that multiple sources of information might ease learnability of useful, low-dimensional representations of data. Specifically, CCA learns linear projections of vectors from two views that are maximally correlated [6]. While CCA is affine invariant, the representations it learns are not sufficient for data that lies on a manifold. We apply a scalable[1] extension of kernel CCA (KCCA) with Gaussian and polynomial kernels. However, kernel methods scale poorly in both feature dimension (a dot product or norm is required) and number of samples (Gram matrix is polynomial in size). To remedy this, we apply deep methods, which enjoy learning a parametric form of a nonlinear target transformation. Those methods described in Arora *et al* are split autoencoders

[1]Although the dimension of the XRMB data is on the order of 10^2, the number of samples ($\approx$ 50,000) would force the Gram matrix to exceed memory capacity

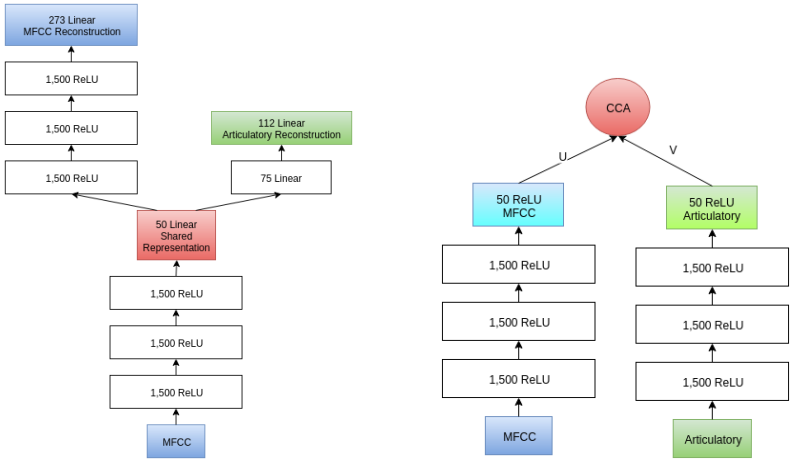


Figure 1: SplitAE and DCCA network architecture diagrams.

(SplitAE)[2], deep CCA (DCCA), and deep CCA autoencoders (DCCAE) to the multiview data in XRMB for the purpose of speech recognition [7].

We investigate the enhancement of phonetic recognition by learning a transformation of acoustic feature vectors (the primary view) informed by articular data (a second view only available at training time). This experimental setting is motivated by previous work, where it was found that articulatory data is a qualified view of speech content that is usually only available in training settings [8].

2 Related Work

2.1 Canonical Correlation Analysis

CCA can be interpreted as an extension of principle component analysis to multiple data sets with the added constraint that the principle components learned in each subspace are maximally correlated. Concretely, given n observations (x_i, y_i), $x_i \in \mathbb{R}^{d_x}$ and $y_i \in \mathbb{R}^{d_y}$ comprising two views of data $\mathcal{X}$, $\mathcal{Y}$ described by an unknown joint distribution $\mathcal{D}$, find k pairs of vectors (u_j, v_j) of the same dimensions to

$$\max \mathit{correlation}(u_i^\top x, v_i^\top y) \tag{1}$$

subject to the constraint that (u_j, v_j) is uncorrelated to all other (u_r, v_r), $j \neq r$, $1 \leq j \leq k$. After finding the first pair (u_1, v_1), subsequent pairs are found via deflation of the views subject to the uncorrelation constraint above. Expanding the definition of correlation for the vectors u_i and v_i:

$$\max_{u_i \in \mathbb{R}^{d_x}, v_i \in \mathbb{R}^{d_y}} \frac{\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[u_i^\top x y^\top v_i\right]}{\sqrt{\mathbb{E}_x[u_i^\top x x^\top u_i]\mathbb{E}_y[v_i^\top y y^\top v_i]}} \tag{2}$$

Note that scaling of u_i or v_i does not change the objective, therefore the variance of u and v can be normalized to one. Expressed equivalently for all u and v to be learned,

$$\begin{aligned} \max_{U \in \mathbb{R}^{d_x \times k}, V \in \mathbb{R}^{d_y \times k}} \quad & \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\text{trace}\left(U^\top x y^\top V\right)\right] \\ \text{subject to} \quad & \mathbb{E}[U^\top x x^\top U] = I, \mathbb{E}[V^\top y y^\top V] = I \end{aligned} \tag{3}$$

[2]As described later, SplitAE reconstructs each view from a shared representation, which is not correlation analysis

Solving for u and v using the Lagrange multiplier method, linear CCA takes the form of a generalized eigenvalue problem, for which the solution is a product of covariance[3] and cross-covariance matrices [9]: U is the top k left eigenvectors (sorted by decreasing eigenvalue) of $C_{xx}^{-1}C_{xy}C_{yy}^{-1}C_{yx}$, which when regularized becomes $(C_{xx}+r_xI)^{-1/2}C_{xy}(C_{yy}+r_yI)^{-1/2}C_{yx}(C_{xx}+r_xI)^{-1/2}$. The i'th column of V is then chosen as $\frac{C_{yy}^{-1}C_{yx}u_i}{\sqrt{\lambda_i}}$, which is regularized similarly.

A second interpretation of CCA is that optimal U and V projection matrices minimize the squared error of reconstructing x (respectively, y) given y (resp. x). Hence, the optimization problems given in Equations 1, 2, 3, and 4 are all equivalent (for i = 1...k, where applicable).

$$\begin{aligned} \min_{U\in\mathbb{R}^{d_x\times k},V\in\mathbb{R}^{d_y\times k}} \quad & \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\|U^\top x - V^\top y\|_2^2\right] \\ \text{subject to} \quad & \mathbb{E}[U^\top C_{xx}U] = I_k, \mathbb{E}[V^\top C_{yy}V] = I_k \end{aligned} \tag{4}$$

There are a number of settings to which CCA, and all correlation analysis variants, can be applied. The first is when all views are available at test and train time. The second is when only a non-empty subset of the views is available at test time. Either of these settings can be enhanced by labels.

2.2 Kernel CCA

KCCA by Akaho *et al* generalizes the transformations learned by CCA to nonlinear functions living in a reproducing kernel hilbert space (RKHS), making it suitable for manifold learning. Given $\mathcal{F}$ and $\mathcal{G}$ are function classes living in reproducing kernel hilbert spaces reserved for $\mathcal{X}$ and $\mathcal{Y}$ respectively, the goal is to find two sets of k functions, $\{f_1,...,f_k\}:\mathbb{R}^{d_x}\to\mathbb{R}$, $\{g_1,...,g_k\}:\mathbb{R}^{d_y}\to\mathbb{R}$ such that for any i, f_i and g_i minimize the squared error of reconstructing each other in the RKHS. That is, given $x\in\mathcal{X}$ and $y\in\mathcal{Y}$, f and g are maximally predictive of, and predictable by, the other,

$$\begin{aligned} \min_{U\in\mathbb{R}^{d_x\times k},V\in\mathbb{R}^{d_y\times k}} \quad & \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\|f_i(x) - g_i(y)\|_2^2\right] \\ \text{subject to} \quad & \mathbb{E}[f_i(x)f_j(x)] = \delta_{ij}, \\ & \mathbb{E}[g_i(y)g_j(y)] = \delta ij \end{aligned} \tag{5}$$

for δ_{ij} an indicator random variable that assumes one if $i=j$ and zero otherwise. We will also introduce equivalent interpretations similar to the above: maximize the correlation of $f(x)$ and $g(x)$ or maximize the covariance $\mathbb{E}_{(x,y)\sim\mathcal{D}}[f_i(x)g_i(y)]$ with the same constraints as Equation 5. As before, every pair of functions (f_j,g_j) is uncorrelated with all other pairs of functions for all k.

For n data vectors from each of $\mathcal{X}$ and $\mathcal{Y}$, the Gram matrices K_x and K_y from $\mathbb{R}^{n\times n}$ can be constructed[4] given a positive definite kernel function $\kappa(\cdot,\cdot):\mathcal{X}\times\mathcal{Y}\to\mathbb{R}$ which is by construction equal to the dot product of its arguments in the feature space[5], $k(u,v)=\Phi(u)^\top\Phi(v)$ for a nonlinear feature map $\Phi:\mathbb{R}^d\to\mathcal{H}$. It is a property of an RKHS that each of its constituent functions f and g from each view can be represented as a linear combination of the n observed feature vectors, $f_i(x)=\sum_{i=1}^n\alpha_j^{(i)}\kappa(x,x_j)=\alpha K_x(x)$. Similarly, $g_i(y)=\beta K_y(y)$. Mirroring arguments from CCA[6], minimizing the objective in Equation 5 is equivalent to maximizing trace $\left(\alpha^\top K_xK_y\beta\right)$ where $\mathbb{E}\left[f_i(x)f_j(x)\right]=\frac{1}{n}\alpha_i^\top K_x^2\alpha_j$. Absorbing the $\frac{1}{n}$ into α, Equation 5 can be interpreted as

$$\begin{aligned} \max_{\alpha\in\mathbb{R}^{n\times k},\beta\in\mathbb{R}^{n\times k}} \quad & \text{trace } \alpha^\top K_xK_y\beta \\ \text{subject to} \quad & \alpha^\top K_x^2\alpha = I_k, \\ & \beta^\top K_y^2\beta = I_k \end{aligned} \tag{6}$$

[3]For numerical stability, a scaled identity matrix is added to a covariance matrix before it is inverted

[4]e.g. $[K_x]_{ij}=\kappa(x_i,x_j)$. Note also that the matrices are assumed to be centered

[5]known as the kernel trick

[6]An unconstrained KCCA objective can be written by moving the square root of the product of the constraints in Equation 6 to the denominator and removing the trace operator as in Equation 2.

As before, α can be solved as a generalized eigenvalue problem, whose regularized solution is $\alpha = K_x^{-1} K_y K_y^{-1} K_x$ and the ith β vector is $\beta_{(i)} = \frac{1}{\lambda_i}(K_y + r_y I)^{-1} K_x \alpha$. This solution takes $O(n^3)$ time and $O(n^2)$ space. When $k << n$, it is wasteful to compute all n columns of α and β. Scalable implementations of KCCA employ rank-m approximations of the eigendecompositions that give the solutions to α and β.

The primary drawback of KCCA is its nonparametric form, which requires α to be stored for use on all unseen examples (test sets). Also, computation of dot products for the kernel functions cannot be avoided; $O(n^2)$ of these dot products are needed at training time. The binding of each instance of KCCA to a specific kernel also limits the function class of our solution. In high dimensional settings, KCCA is also susceptible to overfitting, that is, learning spurious correlations. A remedy is to regularize with a scaled identity matrix those Gram matrices to be inverted as in Equation 7. The regularization parameters also need to be tuned.

$$\max_{\alpha \in \mathbb{R}^{n \times k}, \beta \in \mathbb{R}^{n \times k}} \frac{\alpha^\top K_x K_y \beta}{\sqrt{(\alpha^\top K_x^2 \alpha + r_x \alpha^\top K_x \alpha)((\beta^\top K_y^2 \beta + r_y \beta^\top K_y \beta))}} \tag{7}$$

The regularized solution for α is the top eigenvectors sorted by decreasing eigenvalue of the matrix $(k_x + r_x I)^{-1} K_y (K_y + r_y I)^{-1} K_x$ and the ith β vector is as before.

2.3 Split AutoEncoders

The work of Ngiam et al on multimodal autoencoders introduced a deep network architecture for learning a single representation from one or more views of data [10]. This was later followed up by Wang et al with applications to the Wisconsin X-Ray dataset, as well as a constructed multi-view version of the MNIST data set [7]. The goal of deep autoencoder architectures in multiview learning is to find some shared representation that minimizes the reconstruction error for all views. These deep autoencoders can take as input one or many views at the same time, depending on which are available for testing. For the sake of this paper, we will restrict ourselves to two views.

There are two architectures given by Ngiam et al that find a shared representation of two views. [10] The dataset used to train these have two views available at train time and one view available at test time, as is the case with the XRMB dataset that we are using in this paper.

$$\min_{\mathbf{W}_f, \mathbf{W}_g, \mathbf{W}_p, \mathbf{W}_q} \frac{1}{2} \sum_{i=1}^{N} (\|\mathbf{x}_i - \mathbf{p}(\mathbf{f}(\mathbf{x}_i))\|^2 + \|\mathbf{y}_i - \mathbf{q}(\mathbf{g}(\mathbf{y}_i))\|^2) \tag{8}$$

The first architecture trains using information from both views, minimizing Equation 8 which is the sum of the L_2 Norm of the reconstruction for both views. Both inputs are fed into the autoencoder separately and go through multiple hidden layers before being combined into a shared hidden layer representation of the two views. The decoders are then symmetrical to the encoders. At test time, all of the weights from the decoder of the view not available at test time are ignored, and the shared hidden layer representation calculated from the single view available is used.

$$\min_{\mathbf{W}_f, \mathbf{W}_p, \mathbf{W}_q} \frac{1}{2} \sum_{i=1}^{N} (\|\mathbf{x}_i - \mathbf{p}(\mathbf{f}(\mathbf{x}_i))\|^2 + \|\mathbf{y}_i - \mathbf{q}(\mathbf{f}(\mathbf{x}_i))\|^2) \tag{9}$$

The second architecture, used by Wang et al, has a single encoder that takes as input the view available at train time. It then attempts to learn a shared representation and sets of weights that can reconstruct both views. The decoder for the view available at test time is symmetric to the encoder. The decoder for the view that's only available at train time can be a multilayer decoder or a single layer decoder that is experimentally tuned for number of layers and nodes.

2.4 Deep CCA

Deep CCA, introduced by Andrew et al [2013], is a parametric technique to simultaneously learn nonlinear mappings for each view which are maximally correlated. It is similar to KCCA in that

both are computing nonlinear mappings to maximize canonical correlation between views. However, KCCA has a significant cost in that KCCA requires large kernel matrices which may not often be practical. DCCA on the other hand computes the nonlinear mappings using deep neural networks, which are capable of representing nonlinear, high-level abstractions on top of the input data. The use of neural networks makes DCCA much more scalable than standard KCCA, as the size of the network is not tied to the size of the dataset.

Given views X and Y, DCCA learns representations F and G such that F=$f(X)$ and G=$g(Y)$ where f and g are the transformations computed by two deep neural networks, which are described by network weights $\mathbf{w_f}$ and $\mathbf{w_g}$ respectively. DCCA trains f and g according to the following objective, which maximizes the canonical correlation at the output layer between the two views.

$$\begin{aligned} \max_{U,V,\mathbf{w_f},\mathbf{w_g}} \quad & \frac{1}{N}\mathbf{tr}(U^T F G^T V) \\ \text{s.t.} \quad & U^T(\frac{FF^T}{N} + r_x I)U = V^T(\frac{GG^T}{N} + r_y I)V = I \end{aligned} \tag{10}$$

3 Experimental Methods

The Wisconsin XRMB corpus [11] contains two views of data recorded simultaneously of multiple speakers pronouncing certain English words; the feature vectors of the first view are 273 dimensions long, while those of the second view are 112 dimensions. The first view consists of acoustic data, primarily MFCC features that were manually computed, and the second view consists of data collected from sensors placed on the speakers' tongue and face. All views were mean centered and normalized to unit variance.

For the purpose of this survey, we used a subset of the data and did all of our parameter tuning and testing on the data for one specific speaker (JW11). In all experiments, the training data from view one was projected onto the top k principle components learned by CCA $L = U_k' * Xtrain$ (or encoded into the k-dimensional shared representation in the case of SplitAE). These learned representations were then stacked onto the baseline acoustic features (computed as the middle 39 dimensions of the 273 dimensional acoustic vector).

Lastly, we evaluated our stacked feature vectors using the MATLAB built-in KNN classifier. As a baseline, K-NN was trained on the the un-projected view one data and tested on the respective test set.

3.1 Kernel CCA and Scalable KCCA

A gaussian kernel with variance for the first view being 1200 and that for the second view being 4800 was found to be the best that we tried. The data was randomly partitioned into a training set of 25,000 examples, a developmental set of 10,000, and a test set of 15,000. We implemented a scalable version of KCCA that does not require storing an entire Gram matrix at any point in time; only the alpha matrix, which is $n \times k$ for n training examples. It was extremely time consuming to tune KCCA, and it is likely that given the time constraints, our parameters are somewhat suboptimal.

3.2 Split AutoEncoders

We implemented the shared representation autoencoder using the second architecture described in Section 2.3. This is the design in which one view is used as input into the autoencoder, and a shared representation that can best reconstruct both views is found.

The encoder of the first view consists of three hidden layers of rectified linear units and a linear output layer (the shared representation hidden layer). The rectified linear hidden layers each have 1500 nodes, while the output dimensionality is 50. In general the larger the width of the hidden layers, the better the autoencoders performed. The dimensionality was tuned over $\{30, 50, 70\}$ while the hidden layer width was tuned over many values between 1 and 1500.

The decoder of the first view is symmetric to its encoder. The decoder for the second view consists of two linear layers with dimensionality 50 and 75, respectively. The number of hidden layers was

tuned between 1 and 3 and the width of each layer over values between 1 and 112, the latter being the dimensionality of the second view. We used the RMSProp Algorithm to train SplitAE with a learning rate of 0.0001 and a decay of 0.99.

After training the autoencoders in an unsupervised fashion, we fine-tuned the encoder weights using a single-layer linear classifier at the shared representation layer. The labels were one-hot encoded and the linear classifier was trained using the Adam Method of Stochastic Approximation, as proposed by Kingma $\&$ Ba, with a learning rate of 0.0001 [13]. Errors from the linear classifier were backpropagated across the entire encoder to update weights. This made the encoder more specific to the task of phone classification. Whereas in a completely unsupervised setting, we would have fixed the weights with no fine-tuning and sacrificed classification error for generalization.

All training was done in mini-batches of 100 instances each, while tuning accuracy was computed over the entire tuning dataset (full-batch).

3.3 Deep CCA

We implemented DCCA as described above, where given two views, we learn a high-level representation for each view using a neural network, such that canonical correlation between the representations of the two views is maximized.

Both networks consisted of three hidden layers, each containing 1500 ReLU nodes. The final output layer was of dimensionality 50. Canonical correlation between the output layers was maximized specifically for a subspace of dimensionality 40.

Training DCCA is challenging as unlike traditional deep learning objectives, this one depends on the covariance of the output layer. To efficiently train, we used the stochastic approach described by Wang et al [2015]. As our cost function is defined in terms of the CCA of the output layers, we approximated CCA using minibatches of size 20, and learned U and V for that batch. Given U and V we then optimized the cost function for that batch using the Adam optimization algorithm with a learning rate of 0.000001.

During the training process we used a dropout rate of 0.9 for the input layer and 0.8 for all hidden layers, but set these values each to 1.0 for evaluation on test data.

Having learned the network weights, we then fine-tuned the network by adding a softmax classifying layer on top of the output of the MFCC layer, concatenated with the 39 dimensional baseline acoustic data. The fine-tuning was optimized also using Adam, this time with a learning rate of 0.0001.

4 Results and Discussion

Method	**% Test Accuracy**
Baseline K-NN	79.70
CCA	83.20
KCCA	77.60
SplitAE	84.58
DCCA	84.94

Table 1: Accuracy achieved with best parameters for each correlation analysis method implemented. For CCA and KCCA, the best k was found to 60, and the number of neighbors for K-NN was 4. The size of the output layer of splitAE and DCCA was 50. The authors believe that KCCA requires more tuning.

The performance of the baseline K-NN classifier is admirable, however the techniques implemented here are able to uncover more semantic content. Although only one speaker was used in these experiments, it is clear that deep methods outperform kernel and linear CCA techniques in discerning

phonemes from speech. The choice of kernel constrains the class of functions which can be learned by KCCA, and the parameters associated with the kernel can be difficult to optimize given the time complexity of running a single batch. It is the opinion of these authors, supported by recent work by Arora et al, that correlation based learners should outperform autoencoders because their explicit objective is to maximize predictability of each view from the other, rather than maximizing reconstruction of the inputs. However, it is difficult reduce these claims to practice based on the ease at which highly accurate neural nets can be learned.

While the autoencoder may not have found a shared representation that displayed any correlation amongst the views, it still managed to cluster the data fairly well into the various phoneme groups. We ran the test data through the encoder and saw that it still maintained most of the same clusters, with the exception of the very center in which small clusters tended to become closer to noise than well clustered phonemes. Overall, the autoencoder performed very well when fine-tuned to the task of classification. A future direction for work, as explored by Arora et al, would be to impose correlation constraints onto the autoencoders. In their paper, they propose correlated autoencoders as one such example of this correlation component added to the autoencoder objective.

DCCA was the best performing method, especially when one considers the memory and runtime costs in addition to accuracy. It is quite telling that it was able to get better accuracy than KCCA with roughly one third the runtime, and significantly lower memory usage. We were able to achieve good separation between clusters of the learned representations for the training data in the first view. However, we found that for the test data, the clusters were much less cleanly separated, which indicates that the model was overfitting. This disparity is especially apparent when compared to SplitAE's t-SNE plots. One expalanation for this is that SplitAE was trained with a dropout of 0.8 for the input and 0.5 for the hidden layers, while DCCA was trained with 0.9 for input and 0.8 for hidden. A higher retention rate means that the model is less sparse, and given the high dimensionality of the hidden layers, more likely to overfit as a result. It's quite likely based on these results that with more tuning, we might be able to prevent this overfitting and improve DCCA's overall performance.

DCCA is also a fairly simple model, so there are many improvements one could make to further improve accuracy. For example, we are currently initializing the weights randomly, but it might be useful to pretrain the networks as autoencoders. One could take this idea even further to DCCAE, which essentially consists of an autoencoder on each view, with canonical correlation of the bottlenecks maximized.

Finally, it's worth noting that traditionally DCCA is very hard to train. The derivation of the gradient is difficult and fairly complicated. However, we were able to make use of Google's recently released TensorFlow to make this much easier. We can define U and V as placeholders, and define our cost function in terms of those, filling in U and V based on the batch CCA. TensorFlow can then perform gradient steps on the network given just the cost function defined in terms of as of yet undefined U and V. This means that we can avoid having to find the gradient ourselves which makes the process much easier from an implementation standpoint.

5 References

[1] Kakade, Sham M., and Dean P. Foster. "Multi-view regression via canonical correlation analysis." Learning Theory. Springer Berlin Heidelberg, 2007. 82-96.

[2] Chaudhuri, Kamalika, et al. "Multi-view clustering via canonical correlation analysis." Proceedings of the 26th annual international conference on machine learning. ACM, 2009.

[3] Hardoon, David R., et al. "Unsupervised analysis of fMRI data using kernel canonical correlation." NeuroImage 37.4 (2007): 1250-1259.

[4] Vinokourov, Alexei, Nello Cristianini, and John S. Shawe-Taylor. "Inferring a semantic representation of text via cross-language correlation analysis." Advances in neural information processing systems. 2002.

[5] Dhillon, Paramveer, Dean P. Foster, and Lyle H. Ungar. "Multi-view learning of word embeddings via cca." Advances in Neural Information Processing Systems. 2011.

[6] Hotelling, Harold. "Relations between two sets of variates." Biometrika (1936): 321-377.

[7] Weiran Wang, Raman Arora, Karen Livescu and Jeff Bilmes. On Deep Multi-View Representation Learning. In Proceedings of the 32nd International Conference on Machine Learning (ICML), 2015

[8] Raman Arora and Karen Livescu. Kernel CCA for multi-view acoustic feature learning using articulatory measurements. In Proceedings of the Machine Learning Symposium on Language and Speech Processing (MLSLP), 2012

[9] Rastogi, Pushpendre, Benjamin Van Durme, and Raman Arora. "Multiview LSA: Representation Learning via Generalized CCA."

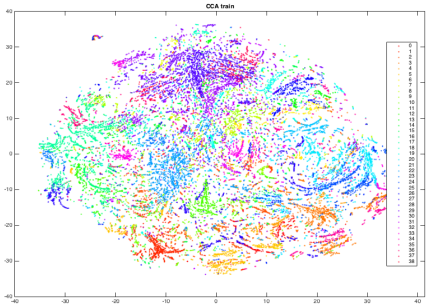
[10] Ngiam, Jiquan, et al. "Multimodal deep learning." Proceedings of the 28th international conference on machine learning (ICML-11). 2011.

[11] Westbury, John R. X-ray Microbeam Speech Production Database User's Handbook Version 1.0 1994 http://www.haskins.yale.edu/staff/gafos_downloads/ubdbman

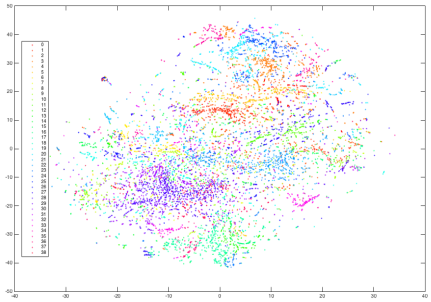
[12] Galen Andrew, Raman Arora, Jeff Bilmes and Karen Livescu. Deep Canonical Correlation Analysis. In Proceedings of the 30th International Conference on Machine Learning (ICML), 2013

[13] Diederik Kingma, Jimmy Ba. Adam: A Method for Stochastic Optimization. In 3rd International Conference for Learning Representations, San Diego, 2015

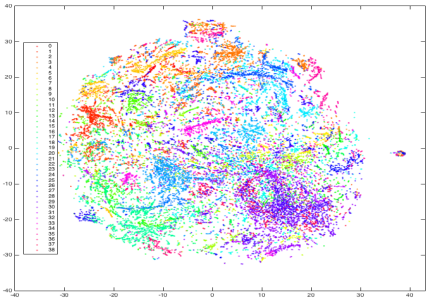
[14] Akaho, Shotaro. "A kernel method for canonical correlation analysis." arXiv preprint cs/0609071 (2001).



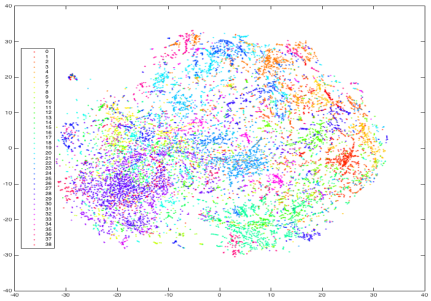
(a) CCA Train



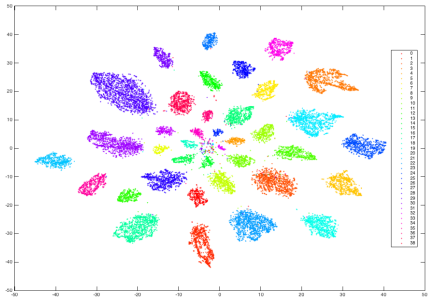
(b) CCA Test



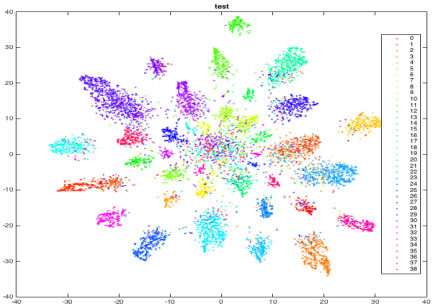
(c) KCCA Train



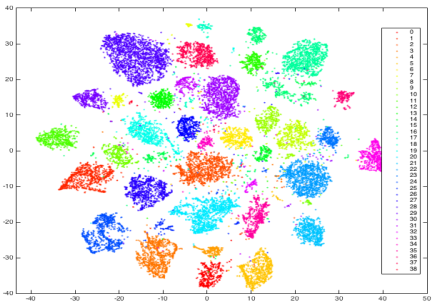
(d) KCCA Test



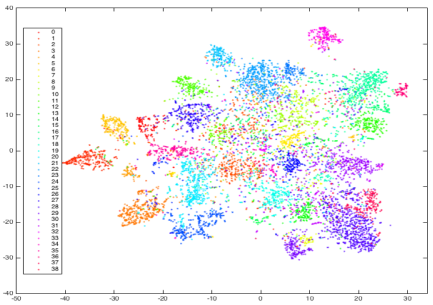
(e) SplitAE Train



(f) SplitAE Test



(g) DCCA Train



(h) DCCA Test

Figure 2: Comparison of t-SNE (perplexity 100) clustering on learned representations for CCA, KCCA, SplitAE, and DCCA