# Clustering-Oriented Representation Learning in Neural Networks

Kian Kenyon-Dean

December 20, 2017

COMP 652 Final Project

*Profs. Guillaume Rabusseau & Riashat Islam*

McGill University

*The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise.* E. W. Dijkstra

**Abstract**

The power of neural networks rests in their ability to learn non-linear transformations of data, allowing them to, hopefully, disentangle the explanatory factors of variation within the data. These nonlinear transformations create new vector representations, the hidden or latent representations. Such representations are new expressions of the input, situated on a highly non-linear manifold that is learned by the neural network. Typically, neural network architectures use hidden layers as a "black box" to be used solely as a pipeline for obtaining the precious final output prediction. In this paper, we beg the question: why limit ourselves to solely predicting and training with discrete output? If we seek to fully take advantage of the expressivity that can be invoked by latent representations, should we not instead act directly on them during the learning process?

We propose a neural network built for supervised classification, with clear directions for extension to semi-supervised settings. The fundamental characteristic of this neural network is that it works solely and directly at the level of representation; there is no output layer. We train our network to build representations of data by using a loss function oriented toward clustering, toward disentangling samples corresponding to different object classes by pushing them onto separate submanifolds in the "latent" space. On a difficult 10-class classification problem, our clustering-oriented network performs better than a neural network trained with categorical cross entropy.

# 1 Introduction

The art of deep learning is, fundamentally, a question of representation learning [9], of learning how to build a representation of data that disentangles the factors of variation within it. At each hidden layer, a neural network creates a new representation of the data, becoming more and more abstract as depth of the layers increases. The power of neural networks is thus found in their ability to learn how to perform abstraction, to represent the data in useful ways – better, to extract features from the data – in ways that humans could not replicate with hand made rules.

In most neural networks, the final output (say, a prediction of the class or label of a sample) is simply the result of using a linear model (such as logistic regression) to draw a hyperplane through the data as it is expressed on the last hidden layer [13]. Crucially, however, the representations in the last hidden layer lie in a very unique space, a space that is learned from experience, a highly non-linear manifold (often very crumpled and folded all over itself) upon which a linear separation can be performed; yet, expressed in its original form, the data was (likely) not linearly separable at all.

Standard methods for training act on the final output of neural models, often forcing the output to be as close to discrete as possible. For example, it is almost universal to use categorical-cross entropy (CCE) loss for classification problems. Training with CCE causes neural networks to learn how to express the data onto such a manifold such that it will be linearly separable. However, there is no guarantee that these final representations will be as "expressive" as they could be, other than for the sole purpose of linear separation for the specific task. This lack of a guarantee is because there are essentially an infinite number of ways that the neural network could represent the data such that it becomes linearly separable; in the worst case by memorizing it. This capacity of neural networks explains why they can perfectly fit to completely random, meaningless data [18]; however, it should be noted that in [4] the authors show that neural networks will seek to learn patterns before memorizing the data, but this does not necessarily guarantee powerful expressivity.

Of course, we must ask, what do we mean by the "expressivity" of a model? In [10] we observe a powerful critique of what the author describes as the "Mythos of Interpretability". The author critically engages with the phenomenon in the deep learning community of the commonly used buzzword, "interpretable", which, while on the surface is important, has no scientifically agreed-upon definition and lacks a rigorous understanding (although [12] offer insights into formally "interpreting" deep models).

In the present work, we argue that an "expressive" model is one in which a categorical variable, an object class, can be represented globally as a combination of all specific instances of that class. The global representation of a class, and the specific samples of the class, should be geometrically linked with each other; samples should be *similar* to their class's global representation, while they should be *dissimilar* to global representation of other classes. Geometrically, this is expressed as a prior that

makes a representation "good" [5], because it means that our representations should reflect the quality of "natural clustering" in the learned latent space. In other words, "different values of categorical variables such as object classes [should be] associated with separate manifolds" [5]. A related expression of this quality is that the "manifold hypothesis for classification" [16] should hold in the latent space.

In this paper, we propose a model design that builds these desirably expressive qualities in our representations: the *Clustering-Oriented REpresentation Learning neural Network*, **COREL-Net**. Our design uses simple methods, yet obtains test-set accuracy that surpasses a neural network trained with categorical cross entropy. To build global representations of object classes, we compute the latent categorical centroid (e.g., mean vector) over all the latent representations of all training set samples belonging to the class (Section 2.1). To separate the object classes (and thus, the samples) onto distinct manifolds, we train the network with loss functions that act on the cosine distance (Section 2.2) between samples and centroids *within the latent space* (Section 2.4). Therefore, as our loss does not work at the level of any final output prediction, but in the domain of the latent space and the representations built there, *our model does not require an output layer* (Section 2; Figure 1). Lastly, for inference, we predict that the class of a test-set sample is simply the class of the latent centroid that the sample's representation is most similar to (Section 2.5). The methods presented here offer much room for further exploration, and our high-quality results (Section 4) motivate such exploration. Particularly, there are clear extensions to the domain of semi-supervised learning (Section 5), and also more exploration can be done with regards to building the global representations and measuring similarity.

# 2 Clustering-Oriented Representation Learning

Our objective is to impose the quality of "natural clustering" [5] onto the representations expressed in the learned latent space. In other words, we seek to separate the representations of different classes onto different manifolds in the latent space. Intuitively, if this property holds, then new samples (if sampled from the same distribution as the training set samples) will be very easy to cluster into their correct classes since the neural network will project them into a latent space designed precisely for this purpose. We call the neural network designed for this purpose the COREL-Net, and it is distinguished from a standard feed-forward neural network for classification by the following characteristics:

1. There is no output layer; instead, loss is backpropagated from the final "hidden layer" (although this layer is no longer so "hidden"), the *representation layer*;
2. The model consistently carries and updates *latent categorical centroids* with representations of samples from the training set, in order to represent the classes in the data (Section 2.1);
3. Loss is not computed with categorical cross entropy over predictions, rather, the loss is computed by maximizing the (dis-)similarity (Section 2.2) between the latent representations of samples and the current latent categorical centroids of each class (Section 2.4).
4. Inference is performed on a test-set sample by feeding it forward through the model, then predicting that the sample's class is the class of the centroid its representation is most similar to (Section 2.5).

The following notation will hold for the remainder of this section. The vector of an arbitrary input sample is denoted as $\mathbf{x}$, and the matrix of input samples is denoted $\mathbf{X}$, with $\mathbf{x}_i$ corresponding to the $i$th row of $\mathbf{X}$; $\mathbf{H}$ denotes the matrix of input samples after being fed-forward through our neural network to the final *representation layer* (with dimensionality $h$) (Figure 1), and $\mathbf{h}_i$ similarly corresponds to the $i$th row of $\mathbf{H}$, our neural representation of sample $\mathbf{x}_i$. The italicized function $H_i(\mathbf{A})$ is corresponds to a standard neural nonlinear transformation at layer $i$, parametrized by a learnable weight matrix $\mathbf{W}^{(i)}$ and bias vector $\mathbf{b}^{(i)}$, in addition to an element-wise nonlinear activation function $a(\cdot)$, thus giving us $H_i(\mathbf{A}) = a(\mathbf{A}\mathbf{W}^{(i)} + \mathbf{b}^{(i)})$. In Figure 1, we visualize the COREL-Net, and the specific parameters displayed are those obtained during during validation testing for model optimization (Section 3.2).

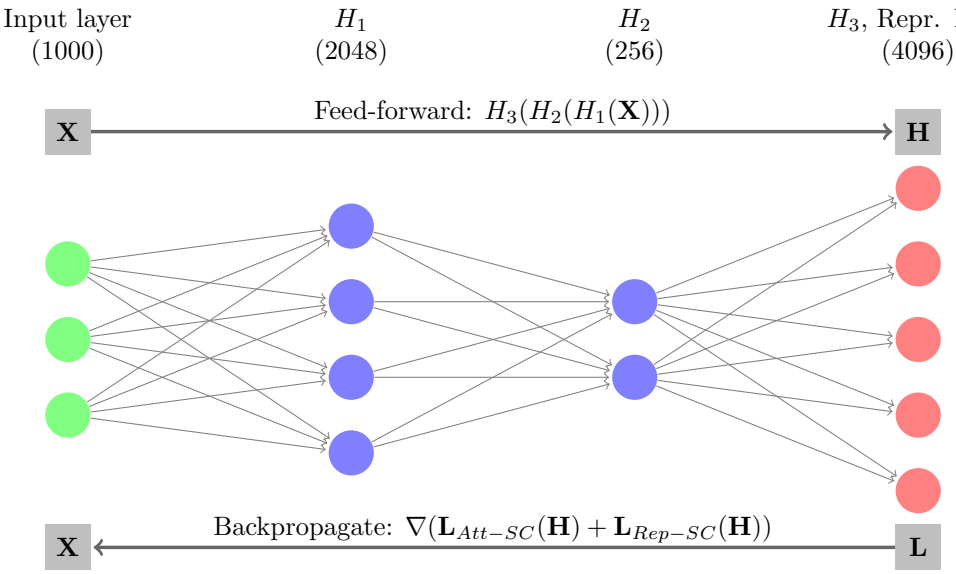



Figure 1: Simplified visualization of our COREL-Net, with parameters determined through validation testing (Section 3.2). Numbers in parentheses denote the number of neurons in the layer.

## 2.1 Latent Categorical Centroids

Our model works with global representations of object classes for the purposes of training and inference. Thus, for each class $k \in \{1, \dots K\}$, we build a representation of the class. A philosophical perspective might help us interpret this: we are building "Platonic Forms" of object classes, building general Forms and comparing them to their specific instances as they appear in the world [14]. For our purposes, we build *latent categorical centroids* to create such Forms. Let the centroid of a class be denoted $\boldsymbol{\mu}_k$, where the model's representation of a sample $\mathbf{x}_i$ is denoted $\mathbf{h}_i$; additionally, let $C_k$ denote the set of indices $i \in C_k$ such that sample $i$ belongs to class $k$. We thus define and build our centroids with Equation 1:

$$\boldsymbol{\mu}_k = \frac{1}{|C_k|} \sum_{i \in C_k} \mathbf{h}_i \tag{1}$$

The latent categorical centroid, $\boldsymbol{\mu}_k$, is thus the mean of the model's representations of all samples belonging to class $k$. Note that, at each step of training, the changing model parameters will change the centroids and representations of samples. Building these centroids is where supervision occurs in our COREL-Net, and it occurs again in computing the loss (Section 2.4). In Section 5, we offer perspectives for how this design may be expandable to a semi-supervised setting, based on intuitions garnered from the K-means clustering algorithm and expectation-maximization. Future work may also involve using density-based weighted averaging to build the centroid, rather than a flat mean; one might motivate this by arguing that it is not desirable to weight noisy samples equally with less noisy ones.

## 2.2 Measuring (Dis-)Similarity

We induce this quality of a clustering-oriented latent space by using the loss functions described in Section 2.4. Each loss is dependent on some predefined distance function (or, rather, measure of dissimilarity) $d(u, v)$ between two vectors $u, v \in \mathbb{R}^h$. In our implementation, we use cosine-distance, setting $d(u, v) = cosd(u, v)$, as shown below in Equation 2.

$$cosd(u, v) = \frac{1}{2}\left(1 - \frac{u \cdot v}{||u||_2 ||v||_2}\right) \tag{2}$$

Cosine distance has several important qualities to consider:

1. It is constrained to be between 0 and 1 for any pair of vectors; e.g., $\forall u, v \in \mathbb{R}^h : cosd(u, v) \in [0, 1]$.
2. It is magnitude invariant since the vectors are normalized.
3. By virtue of being magnitude invariant, it is geometrically understood as measuring the squared euclidean distance between two vectors when projected onto the surface of the unit $h$-hypersphere[1]; see appendix for proof. Concurrently, it is understood as a measurement of the angle between the two vectors.

While one might be concerned that the "expressivity" of the loss will be limited by virtue of the fact that this distance function is magnitude-invariant, our results (Section 4) suggest that this is not a problem. It is well-known that euclidean distance is problematic and not very meaningful in a high-dimensional spaces because it becomes very sensitive to small perturbations in the space [1], and also has no maximum bound. Thus, if we were to maximize the distance between vectors, using euclidean distance would cause the vectors to "explode" away from each other to infinity, while cosine distance would simply cause them to orient away from each other to point in opposite directions. Indeed, in such high dimensional spaces we are inclined to hypothesize that cosine-distance is more expressive than euclidean distance, although future study would have to be pursued to verify this hypothesis.

### 2.3 Manifold Motivations

The intuitions of our model design are largely inspired by recent work and theoretical expositions of manifold learning and the relationships between manifolds and neural networks [2, 16, 5, 13, 6]. The two primary intuitions we are inspired by – which we, in fact, seek to impose into the latent representation space – are understood as "generic priors" for machine learning:

- **The manifold hypothesis.** Data presented in high dimensional spaces has high *probability concentration* in the vicinity of non-linear sub-manifolds of lower dimensionality [16, 5, 6].
- **The manifold hypothesis for classification.** Points of different classes have concentrate along different sub-manifolds, separated by low density regions of the input space [16].

These hypotheses are not particularly controversial, and a simple example leads one to immediately accept their validity. Consider the following example (inspired by [6]): in the space of all possible 32-by-32 images of, say, cats, what is the probability that an image generated at random will be a cat? Essentially, zero. This is because, on the one hand, images belong to a very high dimensional space (e.g., 256 possible values of $32 \times 32 \times 3 = 3072$ pixels equals $3072^{256}$ possible configurations!), but, on the other hand, the space of possible *relevant* input configurations (of pixels that make realistic cat images) is much much smaller than the total space of all input configurations. If we could leverage the manifold hypothesis and find this region of high probability mass (of cat images) within this space, then we could move along this sub-manifold such that small perturbations or interpolations in this space would correspond with probable configurations (of artificial, but convincing, cat images).

While using the insights and intuitions of information geometry [2, 3] is beyond the scope of this work, future work would certainly involve harnessing the vast literature in information geometry to better inform model design decisions. [16] use explicit intuitions from manifold geometry to inform the design decisions of their Manifold Tangent Classifier, but, in the present work, we use comparatively simple geometric intuitions in our model design, yet still obtain high-quality results. The relationship between our design decisions, particularly with respect to using centroids and cosine-distance, with the specific qualities of the created sub-manifolds (such as their curvature and the results of interpolating along them), involves future work, and may be better interpreted when applying the methods presented here to real-world data and analyzing the results (rather than synthetic data; Section 3.1).

[1] E.g., a spherical surface in $h$-dimensional space defined by $||v||_2 = 1$, $\forall v \in \mathbb{R}^h$.

## 2.4 Loss Functions

Here we present the foundation of Clustering-Oriented Representation Learning, of our COREL-Net. These loss functions seek to perform two simultaneous tasks, attraction and repulsion. On the one hand, we want to minimize the distance between representations of samples and the centroids of their class (Equation 3), thus bringing all samples belonging to a single class into a dense probability mass, essentially imposing the manifold hypothesis (Section 2.3) on this distribution. The manifold hypothesis becomes doubly imposed (in addition to the manifold hypothesis for classification; Section 2.3) when considering the repulsive loss function (Equation 4), which attempts to maximize the distance between representations of samples and the centroids of each other class, therefore seeking to concentrate the probability mass of distinct classes onto distinct sub-manifolds, thus separating such sub-manifolds by regions of low density, of low probability mass concentration. Of course, it should be noted that these manifolds are fundamentally characterized by cosine-distance, and thus by dense regions on the surface of the unit-hypersphere in the representation space; future work would involve exploring other ways to characterize these manifolds, likely by using different distance functions.

Note that $K$ is the total number of classes; $n$ is the number of samples in the batch; $C_k$ is the set of all indices $i$ corresponding to samples in the training set that belong to class $k$; $\boldsymbol{\mu}_k$ is the latent categorical centroid for class $k$ (Section 2.1); $d(u, v)$ is our cosine-distance function (Section 2.2); and, $\mathbf{h}_i$ is the representation of sample $\mathbf{x}_i$ after it is fed forward to the last layer of our COREL-Net (Figure 1). Each of these loss functions act on the set of training set representations of a batch, $\mathbf{H}$, the result of an input matrix $\mathbf{X}$ being fed-forward through the network. See appendix for their practical implementation.

**Sample-to-Centroid Attraction.** Here we seek to attract representations to be closer to their class's latent categorical centroid (Section 2.1). This loss computes the average distance between samples and their class's centroids, and seeks to minimize these distances. Note that there are exactly $n$ distances computed in this equation, and we thus take the average over all such distances. $\lambda_1$ is a hyperparameter that determines the influence of this loss function on the total gradient.

$$\mathbf{L}_{Att-SC} = \frac{\lambda_1}{n} \sum_{k=1}^{K} \sum_{i \in C_k} d(\boldsymbol{\mu}_k, \mathbf{h}_i) \tag{3}$$

**Sample-to-Centroid Repulsion.** This loss function seeks to repulse (or, push away) representations of samples from the centroids of classes that they do not belong to by computing the average distance between samples and the centroids of other classes. This function is negated because we seek to maximize this quality, as we want to separate samples from other classes as much as possible in order to associate classes with distinct sub-manifolds. Note that there are exactly $n(k-1)$ distances computed by this equation since there are exactly $k-1$ classes that a sample does not belong to. $\lambda_2$ determines the influence of this loss function on the total gradient.

$$\mathbf{L}_{Rep-SC} = -\frac{\lambda_2}{n(K-1)} \sum_{k=1}^{K} \sum_{j \notin C_k} d(\boldsymbol{\mu}_k, \mathbf{h}_j) \tag{4}$$

**Centroid-to-Centroid Repulsion.** In practice, this final loss function proved to be a hindrance and unnecessary during training (Section 3.2), but we still present it for the purpose of exposition. It's intention is to maximize the distance between the latent categorical centroids; however, it is not surprising that using $\mathbf{L}_{Rep-SC}$ (Equation 4) already imposes this quality, as can be seen in the loss progression presented in Figure 2. Nonetheless, the intuition of this loss was to more explicitly separate the centroids onto distinct sub-manifolds to further impose the manifold hypothesis for classification. $\lambda_3$ determines the influence of this loss function on the total gradient.

$$\mathbf{L}_{Rep-CC} = -\frac{\lambda_3}{K(K-1)} \sum_{k=1}^{K} \sum_{k'=1}^{K} d(\boldsymbol{\mu}_k, \boldsymbol{\mu}_{k'}) \qquad [k \neq k'] \tag{5}$$

### 2.5 Inference using Clustering-Oriented Representations

Since our model does not have an output layer, it is necessary to perform some kind of action on the representation of a test set sample in order to predict its class. Fortunately, our model possesses the latent categorical centroids of each class $k$, $\boldsymbol{\mu}_k$ (Section 2.1). We can thus use a very simple method for prediction, motivated by the Nearest Centroid Classifier [7, 17], which highly resembles the K-means algorithm during the cluster/class prediction stage. Namely, using the our model's representation, $\mathbf{h}$, of the sample, $\mathbf{x}$, we predict that the class of the sample is the class corresponding to the latent categorical centroid that $\mathbf{h}$ is most similar to, as measured by cosine-similarity:

$$class(\mathbf{x}) = \underset{k}{\text{argmax}} \left( \frac{\mathbf{h} \cdot \boldsymbol{\mu}_k}{||\mathbf{h}||_2 ||\boldsymbol{\mu}_k||_2} \right) = \underset{k}{\text{argmax}} \left( \mathbf{h} \cdot \frac{\boldsymbol{\mu}_k}{||\boldsymbol{\mu}_k||_2} \right) \tag{6}$$

In practice, we can save computation time by not computing the norm of the representation ($||\mathbf{h}||_2$) since it is an unchanging constant over the full *argmax* computation (but, if the specific similarities need to be interpreted, the norm would need to be included). So, for the purpose of more rapid inference (particularly when predicting on the validation set during training), we do not include the term.

### 2.6 Training Process

During training, we ran our models for 100 epochs. Each epoch was characterized by a set of batches passed through the model, although, in practice, we used the entire training set as the batchsize (3400) for our COREL-Net. In practice, this is problematic for large datasets, but we merely sought to avoid the problem of building centroids from small batches, which would likely result in unstable centroids. In future work, we will compute the centroids of a batch using a running average across the epoch, which will help to avoid the problem of centroid instability. At the end of each epoch, we predict the classes of the validation set samples (Section 2.5), using the full training set to build the centroids, obtaining an accuracy computation for validation set performance. We selected the model to be used for final testing as the one corresponding to the epoch which achieved the optimal validation set performance.

## 3 Experimental Design

To ascertain our model performance in the present work, we isolate model analysis to synthetic data in order to avoid any possibility of artifacts in the dataset misinforming our intuitions of our model. The most substantial comparison of our highly-optimized COREL-Net is with a highly-optimized CCE-Net, a feed-forward neural network built for classification that uses categorical cross entropy loss. We present the loss function, $\mathbf{L}_{CCE}$, in Equation 7 below; note that $y_{i,k} = 1$ if class of sample $i$ is $k$, else it is 0; $\hat{y}_{i,k}$ is our model's prediction of the sample's class within the output layer.

$$\mathbf{L}_{CCE} = \frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} y_{i,k} \log(\hat{y}_{i,k}) \tag{7}$$

As described in Section 3.2, we also compare with several shallow models. In the final analysis, our COREL-Net performs better than every other model, with a test-set F1-accuracy of 0.849 (Table 1).

### 3.1 Dataset

In order to isolate the model analysis from possible artifacts arising from particular datasets, we opted to use synthetic data[2], although future work naturally involves applying our model to real-world datasets. We designed our synthetic dataset similar to what might be encountered in a natural language processing problem – a large number of features (most of which are uninformative), relatively low number of

[2]Made with scikit-learn's function "make_classification": `http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html`.

| **Model** | Training F1 | Validation F1 | Test F1 | *Parameters* |
|---|---|---|---|---|
| Logistic Regression | 0.707 | 0.233 | 0.282 | *N/A* |
| Logistic Regression-CV | 0.471 | 0.401 | 0.388 | *N/A* |
| Linear SVM | 1.000 | 0.357 | 0.352 | $C = 1.0$ |
| RBF-SVM | 1.000 | 0.630 | 0.590 | $C = 2.0$ |
| CCE-Net | 1.000 | 0.830 | 0.831 | *See Table 2* |
| COREL-Net | 1.000 | **0.870** | **0.849** | *See Table 2* |

Table 1: F1-Accuracy results obtained with optimal model parameters, as determined with hyperparameter tuning to the validation set, over the training, validation, and test sets.

| **Model** | $H_1$ | $H_2$ | $H_3$ | Nonlinearity | $\alpha$ | Batchsize | $\lambda_1$ | $\lambda_2$ | $\lambda_3$ |
|---|---|---|---|---|---|---|---|---|---|
| CCE-Net | 2,048 | 512 | 4,096 | $LeakyReLU_{0.1}$ | 0.0005 | 100 | *N/A* | *N/A* | *N/A* |
| COREL-Net | 2,048 | 256 | 4,096 | $LeakyReLU_{0.1}$ | 0.00065 | 3,400 | 1 | 1 | 0 |

Table 2: Optimal model parameters for the neural networks, determined by extensive validation set testing. $H_i$ is the dimensionality (number of neurons) in hidden layer $i$; $\alpha$ is the learning rate. See Section 2.4 for explanation of the $\lambda$-parameters.

samples, and a significant number of classes. This dataset was designed to be very difficult and noisy, as we can observe from the low accuracy obtained with linear models (Table 1). Indeed, in Figure 5 (Appendix) we observe that the data is quite indiscernible for humans, looking almost entirely like noise. It is characterized as follows:

- 5,000 samples (3,400 for training, 600 for validation, 1,000 for final testing);
- 10 balanced, unique classes (or, categorical variables);
- 1,000 features per sample;
- 50 purely "informative" features per sample and two clusters per class, meaning that each class is composed of two distinct normally distributed clusters (distributed with a high standard deviation of 20) located around the vertices of a 50-dimensional hypercube with small side length (e.g., the class separation value is equal to 1); the rest of the features are either noise or linear combinations of these informative features.

## 3.2 Hyperparameter Selection

To offer a robust analysis of our COREL-Net we compare to linear and nonlinear models. For linear models, we present results obtained with logistic regression, logistic regression with internal cross-validation parameter tuning[3], and a support vector machine with a linear kernel (LSVM). For non-linear models, we compare with a support vector machine with RBF kernel (RBF-SVM) and a feed forward neural network trained with categorical cross entropy (CCE-Net).

**Support vector machines.** SVMs require standardizing the data (e.g., subtracting by the mean of the training set, and dividing by its standard deviation), and then tuning the hyperparameter $C$ – the error penalty during model fitting – on the validation set. We tested 50 different values of $C$ between 0.05 and 5, finding that the optimal LSVM had $C = 1$, and the optimal RBF-SVM had $C = 2$; we note clear overfitting of the models, but that the non-linear RBF-SVM generalizes better than the LSVM.

[3]See scikit-learn's *LogisticRegressionCV* class: `http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegressionCV.html`.

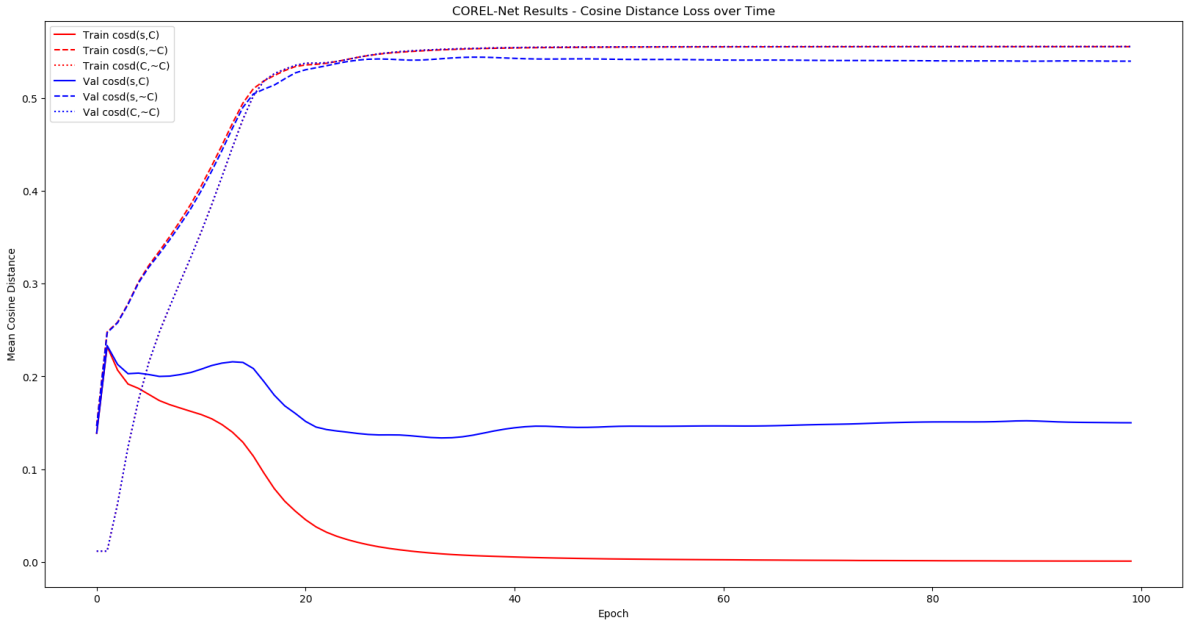



Figure 2: Cosine distance visualization computed at each epoch during training, representing the COREL loss functions (see Section 2.4). Note that "cosd(s, C)" indicates the mean cosine distance between a sample's representation and its corresponding latent categorical centroid; "cosd(s, $\sim$ C)" indicates the same mean cosine distance but between a sample and the centroids of each other class; "cosd(C,$\sim$ C)" indicates the mean cosine distance between a centroid and each other centroid.

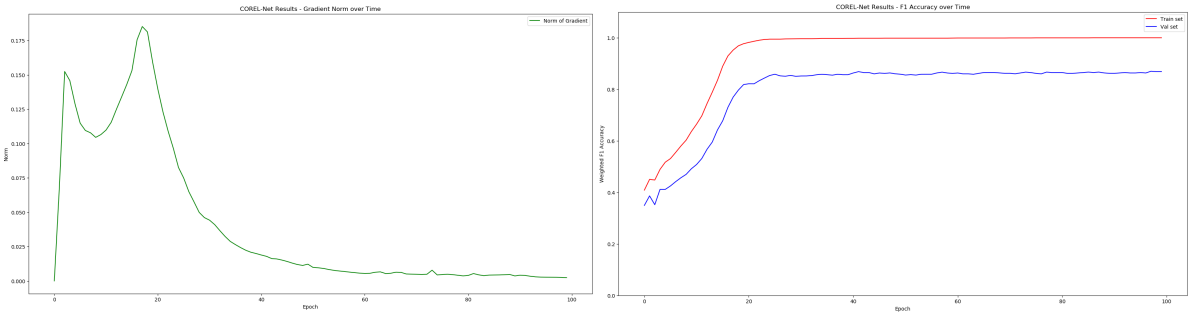
**Neural networks.** Optimizing the neural networks required a significant amount of parameter tuning, where we ultimately experimented with several thousand model variants. For the COREL-Net, the first insight we gained is that the centroid-to-centroid repulsive loss function, $\mathbf{L}_{Rep-CC}$ (Equation 5) is a hindrance on the model and results in worse accuracy. This is not surprising since $\mathbf{L}_{Rep-SC}$ implicitly causes the centroids to be dissimilar, and, as can be seen from the results of our best model variant (Figure 2), where we observe that training with only $\mathbf{L}_{Att-SC}$ and $\mathbf{L}_{Rep-SC}$ results in $\mathbf{L}_{Rep-CC}$ being optimized anyway. Additionally, setting $\lambda_1 = \lambda_2 = 1$ was better than scaling one or the other, suggesting that both losses are equally important.

Secondly, we found that the activation function was highly important for obtaining high accuracy. Models trained with smooth activations (e.g., Tanh, sigmoid, ELU) performed quite poorly, while the piecewise-linear (or, "jagged") activations of ReLU and LeakyReLU were much better (see Appendix for their equations and visualizations, Figure 6). It is not surprising that LeakyReLU was better than ReLU, since ReLU forces all the representations to be completely non-negative, which is disadvantageous for cosine distance since the maximum cosine distance between two non-negative vectors is 0.5, not 1.

Finally, we found that tuning the dimensionality and number of hidden layers was decisive. Models with one or two hidden layers were worse than models with three layers. Additionally, we found that results improved when using the second hidden layer as a "bottleneck" before the representation layer. For example, hidden layer structures such as $[2048 - 256 - 1024]$[4] were better (and have much fewer parameters) than ones like $[2048 - 2048 - 2048]$. Also, larger dimensionalities proved to be better; for example, a model with hidden layer structure $[2048 - 2048 - 2048]$ had test set F1-Accuracy of 0.82, compared to a model with structure $[128 - 128 - 128]$ which got an accuracy of 0.62.

To summarize, we ran several thousand experiments with a selective gridsearch over the many different parameter settings described above, and optimized the COREL-Net and CCE-Net on the validation set until performance no longer seemed to improve. Both used Adam [8] for optimization.

[4]E.g., 2048 neurons in hidden layer 1, 256 in hidden layer 2, 1024 in hidden layer 3.




(a) Norm of the gradient over time.

(b) Train and validation F1-accuracy over time.

Figure 3: Visualizations of COREL-Net training process; gradient norm and F1-accuracy over time.

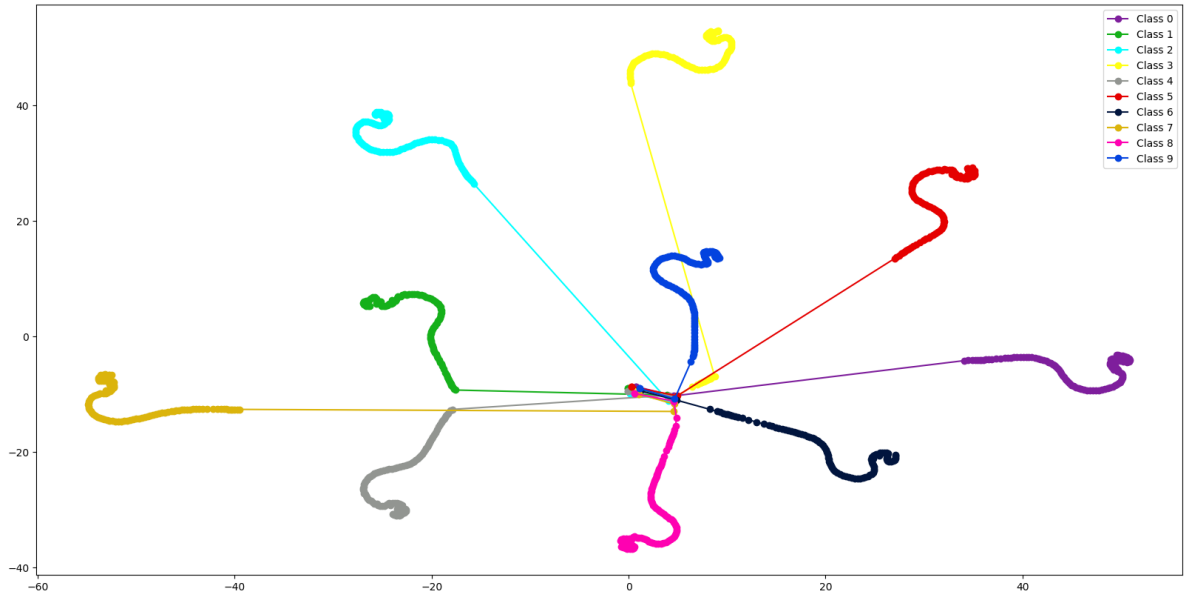



Figure 4: T-SNE visualization of the position of centroids in the latent space during training epoch.

# 4 Results Analysis

We track model progress in several different ways during each epoch of the training process. In Figure 2, we present the cosine distances over time between representations and the latent categorical centroids. In Figure 4, we present the position of the centroids within the latent space over time[5], projected into two dimensions using T-SNE [11].

As can be seen from Figure 2, the model successfully attracts representations to their centroids, and away from the centroids of other classes. This phenomenon occurs for both the training set representations and the validation set representations, indicating that the model is actually learning the general distribution of the classes *without categorical cross-entropy*. Additionally, we observe that the distance between centroids is completely optimized for both the training and validation sets, indicating that we do not need $\mathbf{L}_{Rep-CC}$ to separate the centroids onto distinct sub-manifolds, that $\mathbf{L}_{Att-SC}$ and $\mathbf{L}_{Rep-SC}$ do so on their own. Observing the centroid positions over time, during training (Figure 4) doubly confirms that the classes are moving onto their own distinct sub-manifolds in the latent space.

[5]See Figure 8 (Appendix) for a less-accurate visualization, which uses linear PCA; see Figure 7 (Appendix) for the velocity of the centroids, how much they change each epoch.

It should be noted that there is clear overfitting to the training set, and we would like the distance between validation set representations and their class centroids to be closer than it is currently. It also may not be desirable for the training set representations to be so similar to their centroids; in the future, we recommend experimenting with methods that ensure probability concentration is more spread out along the sub-manifold of the class so that the variance of specific samples is not lost. While we did not experiment with any regularization techniques in this work, future work would involve testing methods such as dropout and weight decay. Despite these clear areas for improvement, it is impressive that our model obtains test-set accuracy that surpasses every other model, including the CCE-Net (Table 1).

A particularly fascinating commonality between each expression of the training process is that, around epochs 15-20, the model undergoes, to speak in the language of catastrophe theory, a *catastrophic change* [15]. This was not an artifact of this particular training iteration; variants of the COREL-Net with different hyperparameter configurations all had to experience this catastrophic change *in order to converge*; models with poor hyperparameter configurations could not escape the catastrophe.

The change in the norm of the gradient during those epochs (Figure 3a) most powerfully reflects this catastrophic change, as we see a dramatic increase in the gradient norm before being able to reach convergence. This is, to a lesser extent, reflected in the visualizations of the COREL distance changes (Figure 2), where, while the repulsive distance ($\mathbf{L}_{Rep-SC}$) is consistently improved (e.g., maximized), the attractive distance ($\mathbf{L}_{Att-SC}$) does not improve (e.g., get minimized) at nearly the same rate; it takes the catastrophic shift to allow the attractive loss to converge. This shift likely is the result of a need for a significant reorientation of the manifold structure in the COREL-Net. Indeed, by observing the movement of the centroids within the latent space over time (Figure 4), we see that it is between epochs 15-20 where each of the centroids begins to start "turning", making the fish-hook shape; similarly, it is also around these epochs where the centroids "turn around" in the PCA visualization (Appendix, Figure 8); again, this is reflected in the centroid velocity at each epoch where the centroids accelerate during these catastrophic epochs (Appendix, Figure 7). These results indicate that the manifold structure does indeed undergo a catastrophic change, but further exploration into catastrophe theory, combined with an interpretation of the process of training a neural network as a dynamical system, would be necessary to have a more complete understanding of this behavior.

# 5 Perspectives

We have presented the COREL-Net, a neural network that does not require an output layer to perform classification. The COREL-Net relies on latent categorical centroids to model the general forms of different classes in the data. There is an immediate application of this technology to semi-supervised learning: first, we would fit the model to a small amount of data, getting the class centroids; then, we feed unlabeled samples through model, find the centroid their representation is most similar to, assign that as its label; using these labels, train for one epoch; then, repeat until convergence. This would be similar to a hard-EM (expectation maximization) approach to learning; a soft-EM variant could also be used, where we use the centroid similarity measures as weights for class assignment during training, rather than the argmax. Additionally, this need not be a parametric model; if the representations of a certain set of samples seem to be too different from the current centroids, we could make a new centroid using that set. A related consideration for future work is that we may not necessarily want to discriminate between all object classes equally; for example, we would probably not want to impose into our latent space that the *Dog* object class should be as different from the *Cat* class as it is from the *Skyscraper* class. However, it may be the case that our latent space would implicitly reflect this quality as a byproduct of the learning process, although further analysis into the latent categorical centroids created on real-world datasets would be required to verify this hypothesis.

In any machine learning setting where there are discrete classes, our model could be used. One large advantage of our model over a neural network trained with categorical-cross entropy is that our model creates clear, expressive representations of the global classes in our data by using latent categorical centroids. This is a desirable quality for applications where it would be useful to understand the general character of an object class. For example, by training a decoder network on top of our

representations, we could decode our centroids to get general presentations of object classes, such as, say, the general, Platonic form of cat images. Another application of this technology would be to have general representations of market-actor classes in an online marketplace; e.g., we may want to characterize exactly where a user falls within the space of general object classes in order to make better personalized recommendations, i.e., if the user's behavior as fed into a COREL-Net leads to a representation that is more similar to the general form of the *PC-gaming Enthusiast* user-class than to, say, that of the *Cookbook-Hoarding* user-class, we could make a more informed decision of the best recommendation to display to the user.

# References


[1] Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim. "On the surprising behavior of distance metrics in high dimensional spaces". In: *ICDT*. Vol. 1. Springer. 2001, pp. 420–434.

[2] Shun-Ichi Amari. "Information geometry of the EM and em algorithms for neural networks". In: *Neural networks* 8.9 (1995), pp. 1379–1408.

[3] Shun-ichi Amari and Hiroshi Nagaoka. *Methods of information geometry*. Vol. 191. American Mathematical Soc., 2007.

[4] Devansh Arpit et al. "A closer look at memorization in deep networks". In: *arXiv preprint arXiv:1706.05394* (2017).

[5] Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation learning: A review and new perspectives". In: *IEEE* 35.8 (2013), pp. 1798–1828.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *The Manifold Perspective on Representation Learning*. 2015. URL: `http://www.deeplearningbook.org/version-2015-11-24/contents/manifolds.html` (visited on 11/02/2017).

[7] Eui-Hong Sam Han and George Karypis. "Centroid-based document classification: Analysis and experimental results". In: *European conference on principles of data mining and knowledge discovery*. Springer. 2000, pp. 424–431.

[8] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[9] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[10] Zachary C Lipton. "The mythos of model interpretability". In: *arXiv preprint arXiv:1606.03490* (2016).

[11] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of Machine Learning Research* 9.Nov (2008), pp. 2579–2605.

[12] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. "Methods for interpreting and understanding deep neural networks". In: *Digital Signal Processing* (2017).

[13] Christopher Olah. *Neural Networks, Manifolds, and Topology*. 2014. URL: `http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/` (visited on 12/18/2017).

[14] Plato and George Maximilian Anthony Grube. *Plato's Republic*. JSTOR, 1974.

[15] Tim Poston and Ian Stewart. *Catastrophe theory and its applications*. Courier Corporation, 2014.

[16] Salah Rifai et al. "The manifold tangent classifier". In: *Advances in Neural Information Processing Systems*. 2011, pp. 2294–2302.

[17] Robert Tibshirani et al. "Diagnosis of multiple cancer types by shrunken centroids of gene expression". In: *Proceedings of the National Academy of Sciences* 99.10 (2002), pp. 6567–6572.

[18] Chiyuan Zhang et al. "Understanding deep learning requires rethinking generalization". In: *arXiv preprint arXiv:1611.03530* (2016).

# Appendix

**Proof of relationship between cosine-distance and euclidean distance.** The cosine-distance between to vectors $u, v \in \mathbb{R}^n$, is equivalent to the squared euclidean distance between the normalized vectors $\frac{1}{||u||_2}u$, $\frac{1}{||v||_2}v$.

- Squared euclidean distance between $u$ and $v$: $||u-v||_2^2 = (u-v)^T(u-v) = ||u||_2^2 + ||v||_2^2 - 2u \cdot v$
- If $u$ and $v$ are normalized then we have: $||\frac{1}{||u||_2}u - \frac{1}{||v||_2}v||_2^2 = 1 + 1 - 2\frac{1}{||u||_2}u \cdot \frac{1}{||v||_2}v$
- Thus giving us: $2(1 - \frac{u \cdot v}{||u||_2||v||_2})$, exactly the equation for the cosine distance between $u$ and $v$.

Therefore, the cosine distance between any two vectors is equal to the squared euclidean distance between those two vectors when normalized, or, equivalently, when projected onto the unit hypersphere.

**Matrix-based implementation of COREL-Net loss functions.** In Section 2.4, Equations 3, 4, 5 are all expressed in their most intuitive form for the purpose of exposition. However, in the actual implementation of these functions, it would be extremely inefficient to express the loss function of a neural network using a summation loop. Indeed, it is desirable to express the functions entirely in terms of matrix computations, particularly when using symbolic programming for neural network design. Here we present the matrix-computation-based derivation of the loss functions, where the computations for Equations 3 and 4 are computed simultaneously with the same matrices, the only difference is the final masking. We do not present the derivation of $\mathbf{L}_{Rep-CC}$ (Equation 5) since it was not necessary in the final implementation.

Let $\mathbf{H} \in \mathbb{R}^{n \times h}$ be the result of feeding-forward a matrix of input samples to the representation layer of the COREL-Net (Figure 1); let $\mathbf{M} \in \mathbb{R}^{K \times h}$ be the matrix of centroids such that row $k$ corresponds to the latent categorical centroid $\boldsymbol{\mu}_k$ (Section 2.1). Recalling the definition of cosine-distance in Equation 2, we construct the following matrices:

1. We set $\mathbf{D}$ to be the matrix of dot products between samples and centroids: $\mathbf{D} = \mathbf{H}\mathbf{M}^T$; note that $\mathbf{D} \in \mathbb{R}^{n \times K}$, and $\mathbf{D}_{i,k} = \mathbf{h}_i \cdot \boldsymbol{\mu}_k$.
2. Let $\boldsymbol{l}^{(h)} \in \mathbb{R}^{n \times 1}$ denote the vector of euclidean norms (as denominators) of each row (i.e., each latent representation of a sample) in $\mathbf{H}$; e.g., $\boldsymbol{l}_i^{(h)} = \frac{1}{||\mathbf{h}_i||_2}$. Similarly, let $\boldsymbol{l}^{(\mu)} \in \mathbb{R}^{K \times 1}$ denote the vector of euclidean norms of each row (i.e., each centroid) in $\mathbf{M}$; e.g., $\boldsymbol{l}_k^{(\mu)} = \frac{1}{||\boldsymbol{\mu}_k||_2}$.
3. We now set $\mathbf{N}$ to be the matrix of the norms multiplied together of the samples and centroids: $\mathbf{N} = \boldsymbol{l}^{(h)}(\boldsymbol{l}^{(\mu)})^T$; note that $\mathbf{N} \in \mathbb{R}^{n \times K}$, and $\mathbf{N}_{i,k} = \frac{1}{||\mathbf{h}_i||_2||\boldsymbol{\mu}_k||_2}$.
4. We can now define $\mathbf{S}$ to be the matrix of cosine-similarities, where $\mathbf{S} = \mathbf{D} \odot \mathbf{N}$, the element-wise product of $\mathbf{D}$ and $\mathbf{N}$.
5. We can now construct our matrix of cosine-distances[6] $\mathbf{C} = \frac{1}{2}(\mathbf{1} - \mathbf{S})$.
6. Now we construct a mask matrix $\mathbf{T} \in \mathbb{R}^{n \times K}$ such that $\mathbf{T}_{i,k} = \frac{\lambda_1}{n}$ if $i \in C_k$ else $\mathbf{T}_{i,k} = -\frac{\lambda_2}{n(k-1)}$.
7. Taking the sum of all the elements in the following component-wise product, $\text{sum}(\mathbf{C} \odot \mathbf{T})$, gives us exactly the sum of our two sample-to-centroid loss functions, $\mathbf{L}_{Att-SC} + \mathbf{L}_{Rep-SC}$. □

**Equations of standard activation functions in neural networks.** The following nonlinear activation functions (Equations 8-12) are commonly experimented with when designing neural models, and during hyperparameter tuning we experimented with each one for both the COREL-Net and the CCE-Net. We also tested different values of $\alpha$ for *LeakyReLU*, and $\alpha = 0.1$ turned out to be the best for both. See Figure 6 for their visualizations.

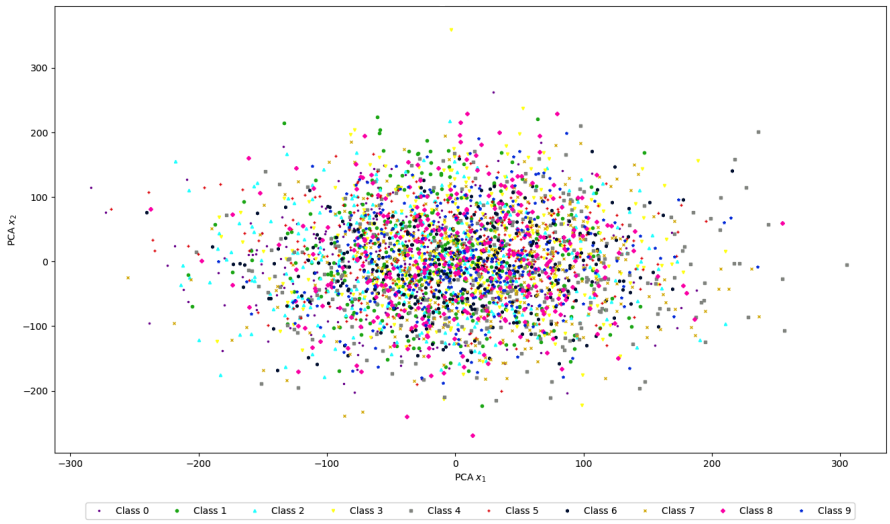



Figure 5: Visualization of our training set when compressed to two dimensions with linear PCA.

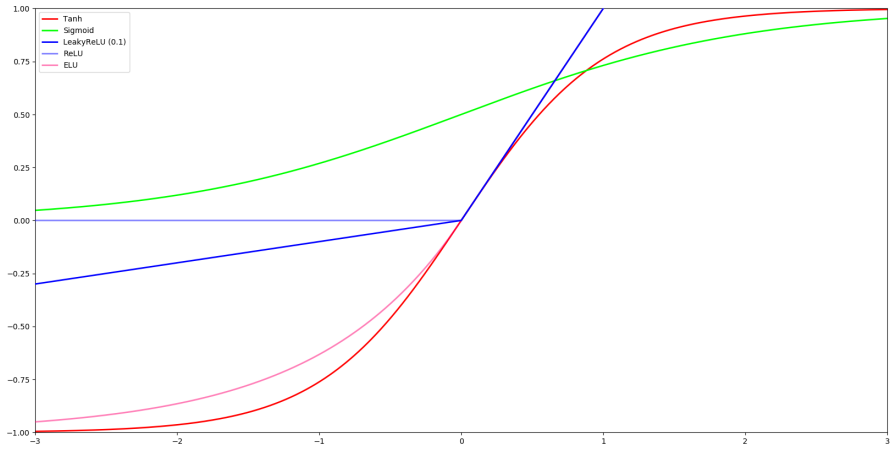



Figure 6: Plot of five different commonly used activation functions in neural networks.

$$sigmoid(x) = \frac{1}{1+e^{-x}} \qquad \in [0,1]\ \forall x \in \mathbb{R} \tag{8}$$

$$Tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}} \qquad \in [-1,1]\ \forall x \in \mathbb{R} \tag{9}$$

$$ReLU(x) = max(0,x) \qquad \in [0,\inf]\ \forall x \in \mathbb{R} \tag{10}$$

$$LeakyReLU_\alpha(x) = max(0,x) + \alpha \cdot min(0,x) \qquad \in [-\inf,\inf]\ \forall x \in \mathbb{R} \tag{11}$$

$$ELU(x) = max(0,x) + min(0,e^x - 1) \qquad \in [-1,\inf]\ \forall x \in \mathbb{R} \tag{12}$$

[6]Note, it is not actually necessary to include the matrix of ones $\mathbf{1}$ in this equation since it is a constant, which during backpropagation would have a gradient of zeroes associated with it; nonetheless, we include it in our implementation for easy interpretability of our loss functions.

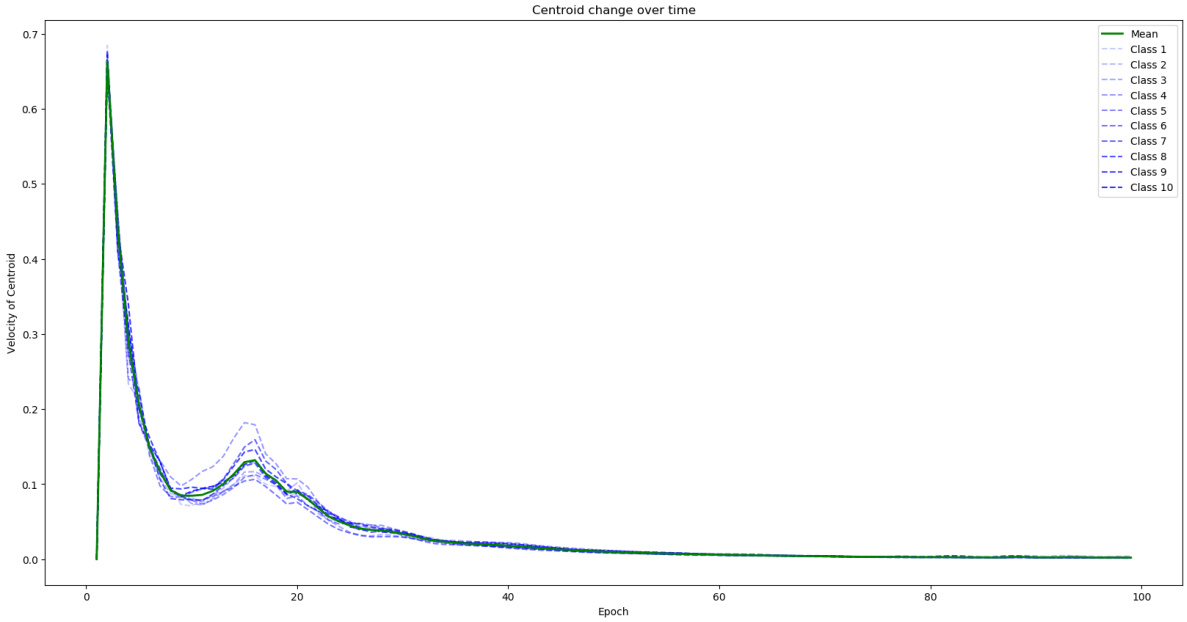



Figure 7: Centroid velocity over time.

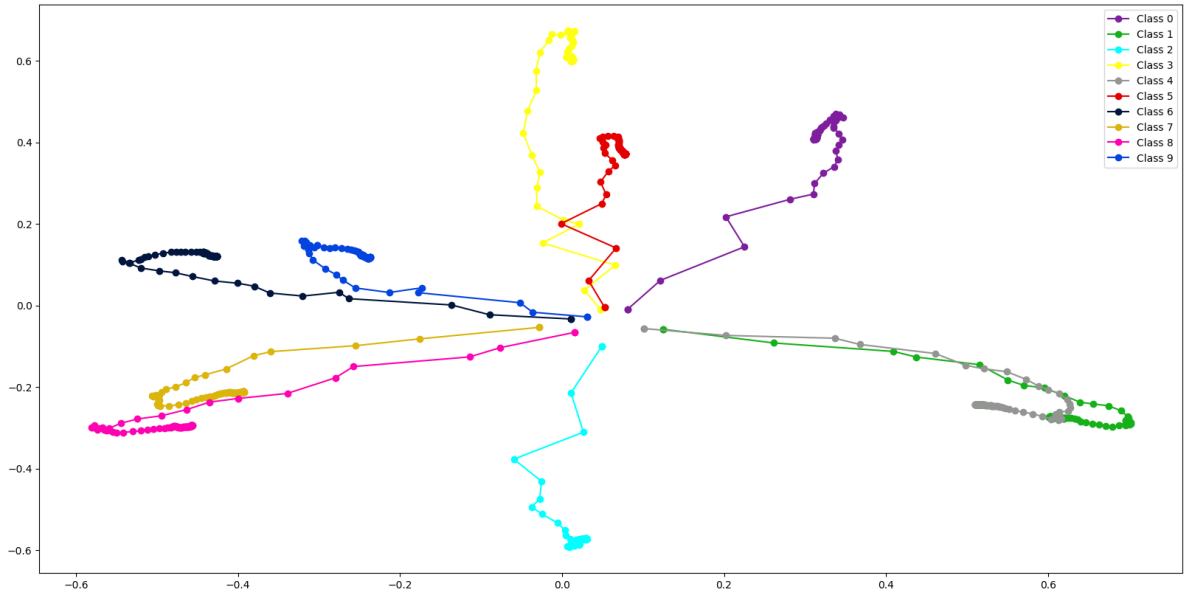



Figure 8: Centroid position over time, projected into two dimensions using linear PCA.