Deep Hidden Layer Learning

Robert Timpe
rtimpe@cs.ucsd.edu

Kristjan Jonsson
kjonsson@eng.ucsd.edu

Abstract

Deep neural networks have achieved state of the art performance on a number of tasks, most notably object detection [6]. However, they are still difficult to train and design. Backpropagation suffers from the vanishing gradient problem [3] [4], making the training of deep networks slow and difficult. Additionally, the design of these networks is largely arbitrary. We explore a novel way to deal with this problem by generating targets for the hidden layers by interpolating between training example and label similarities. Using this method we can reduce the training problem to training a series of single layer networks. As part of this process, we also try to determine the effective dimension of each set of hidden targets. This information can be used to determine the correct number of hidden units at each layer, as well as the correct number of hidden layers.

1. Introduction

Multilayer perceptrons are usually trained using the backpropagation algorithm [7]. In this approach, targets are known only at the output layer, and error signals must be propagated backwards through the network to the hidden units. Backpropagation often suffers from the vanishing gradient problem [3] [4]. That is, the size of the error tends to decrease as it is propagated through each layer. By the time it reaches the earlier layers in the network, the error signal may be extremely small. As a result, the weight changes at each iteration will be very small, and it will take a long time for gradient descent to converge.

In addition to the vanishing gradient problem, it is also difficult to design the architecture of a multilayer perceptron. There is no obvious method for selecting either the number of hidden layers or the number of hidden units at each hidden layer.

We attempt to solve the vanishing gradient problem. In doing so, we hope to gain some insight into the correct design of the network.

2. Generating hidden targets

2.1. Problem statement

We consider the binary classification problem with training examples $x_1, ..., x_N$ drawn from $\mathbb{R}^d$ and binary labels $y_1, ..., y_N \in \{-1, 1\}$. We will attempt to interpolate between the inputs x_i and outputs y_i to derive targets h_i suitable for each hidden layer. Once these targets are known, the problem of training the entire network reduces to the problem of training a series of single layer networks.

2.2. Algorithm

The most naive approach one could come up with for getting hidden targets, h_i, is interpolating between an input example, x_i and its label y_i, i.e.

$$h_i(\alpha) = (1-\alpha)x_i + \alpha y_i, \quad \alpha \in [0, 1].$$

However this is not well defined since the labels and input examples live in different vector spaces. To get around this we instead look at similarities between two input examples, x_i and x_j, and their respective labels and interpolate, i.e.

$$\text{sim}(h_i, h_j)(\alpha) = (1-\alpha)\cdot\text{sim}(x_i, x_j) + \alpha\cdot\text{sim}(y_i, y_j) \quad (1)$$

An appealing similarity metric for vectors is the cosine similarity

$$\text{sim}(a, b) = \frac{a^T b}{|a||b|}$$

or simply dot product which can be thought of as an unnormalized cosine similarity measure i.e.

$$\text{sim}(a, b) = a^T b.$$

Plugging into equation 1 yields

$$h_i^T h_j(\alpha) = (1-\alpha)x_i^T x_j + \alpha y_i^T y_j. \quad (2)$$

Since $y_i \in \{+1, -1\}$ we have $y_i^T y_j = +1$ if examples i and j are from the same class, otherwise $y_i^T y_j = -1$. Equation 2 can thus be interpreted in the following way: We want to find representations $h_i(\alpha)$ for example i such that as α increases $h_i(\alpha)$ becomes increasingly similar to

example $h_j(\alpha)$ if j is of the same class (closer to $+1$) and dissimilar for j from the other class (closer to -1).

Vectorizing equation 2 gives

$$H_\alpha H_\alpha^T = (1-\alpha)XX^T + \alpha yy^T, \tag{3}$$

where

$$X = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_N^T \end{pmatrix}, \qquad y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

and

$$H_\alpha = \begin{pmatrix} h_1^T(\alpha) \\ h_2^T(\alpha) \\ \vdots \\ h_N^T(\alpha) \end{pmatrix}.$$

We can easily calculate the right hand side of equation 3 from the dataset, i.e.

$$G_\alpha = (1-\alpha)XX^T + \alpha yy^T.$$

What remains is to extract H_α such that $H_\alpha H_\alpha^T = G_\alpha$. Since G_α is PSD we can use eigendecomposition to get

$$G_\alpha = U\Lambda U^T,$$

where $U \in \mathbb{R}^{N\times k}$ and $k \leq d+1$ is the rank of G_α and $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_k)$. Setting

$$H_\alpha = U\Lambda^{\frac{1}{2}}$$

gives the desired property. To summarize, extracting hidden targets given $\alpha \in [0,1]$ is done in three steps:

1. Compute $G_\alpha = (1-\alpha)XX^T + \alpha yy^T$
2. Extract U and Λ using eigenvalue decomposition, i.e. $G_\alpha = U\Lambda U^T$
3. Set $H_\alpha = U\Lambda^{\frac{1}{2}}$

The i-th row of H_α is then the hidden target $h_i(\alpha)$ for example x_i.

2.3. Finding the correct range of α

A key problem we have to address is where to place α such that a single layer can learn the mapping between the two adjacent hidden targets. For very small values of α, the hidden targets will be nearly identical to the training examples. For very large values of α, the hidden targets will be nearly identical to the class labels. In order for the neural network to learn a useful representation, it is necessary to find values of α for which the hidden targets will be in some intermediate stage - transitioning from the training examples to their labels. It will also be necessary to space the values of α correctly. If the step size is too small, we will have unnecessary hidden layers. If it is too large, then the transformation may be too complicated for a single hidden layer to learn. We call this range of values of α the Critical Zone.

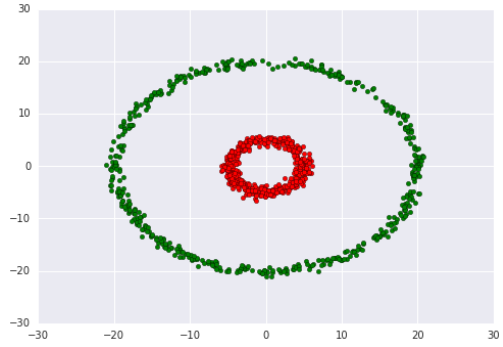


Figure 1: A sample dataset that is not linearly separable

To illustrate the problem, consider the two dimensional dataset shown in Figure 1. This is a simple dataset that is not linearly separable (a linear decision function can only achieve about 50 % accuracy). For most values of α not near 0 or 1, the hidden targets will have 3 dimensions (because XX^T has rank 2 and yy^T has rank 1). Figure 2 shows these three dimensions when $\alpha = .05$. The first two dimensions are largely the same, while the third dimension is essentially just the class label. In other words, at this value of α, we are adding the class label without significantly changing the original representation. Figure 3 shows the same information for the case when $\alpha = .95$. While the scale of the features has changed, not much has changed in terms of separability.

However, this begins to change at around $\alpha = .99$ (shown in Figure 5). At this value of α, we see that the first two dimensions are becoming more easily separable, while the third dimension is becoming less so. By the time $\alpha = .992$ (Figure 6), the inner circle has moved outside of the outer circle, and the data is easily linearly separable. This process can be summarized as follows. For values of α up to about .99, the third dimension of the hidden targets is the label, and the first two dimensions are noise. As α increases to about .992, a transformation happens, and the first dimension becomes the most discriminant, while the other two become noisy. Finally, when $\alpha = 1$, there is only 1 dimension left, which is the class label. Scaling the data to be in the same range as the labels changes the location of this transformation within the range of values of α, but does not change the transformation itself.

The region of values of α in which this transformation takes place is what we call the Critical Zone. Within this range of values of α, the hidden targets do indeed transition

between different representations. Unfortunately, they do not do so in a way that is easily learnable by a series of single layer perceptrons. For example, Figure 7 shows the first two dimensions at $\alpha = .9905$. At this point in the process, many of the datapoints from the two classes are overlapping. So if one layer of a neural net learned this representation, the next layer would have trouble distinguishing between the two classes (many of the points are overlapped in the 3rd dimension as well). In other words, we have found the range of α values where an interesting transformation takes place, but it does not result in intermediate representations that are useful for training a neural network.

Figure 4 helps to illustrate the difficulty in correctly identifying the Critical Zone. It shows the error rate of logistic regression trained on the hidden targets $h_i(\alpha)$ for $\alpha = \{0, .05, .1, ..., 1\}$. At $\alpha = 0$, the dataset is unchanged (up to a possible rotation) and we get about % 50 accuracy. However, by the time $\alpha = .05$ the data is already linearly separable. This is a result of the effect seen in Figure 2, where the class label has effectively been added as a feature.

2.4. Effective Dimension

In addition to looking for the Critical Zone of α values, we also were interested in the effective dimension of the dataset. We defined the effective dimension of the hidden targets at a particular value of α as the number of dimensions needed to explain 95 % of the variance. Recall that the matrix H_α is of rank at most $d + 1$. However, many of its eigenvectors have very small eigenvalues - corresponding to features of little significance. By looking at the effective dimension, we hope to eliminate these unnecessary features and any noise they may introduce.

While this does run the risk of eliminating potentially discriminative features, this may actually be an advantage. In the previous section we saw that, outside of the Critical Zone, the "label" feature was always the one with the corresponding smallest eigenvalue. Eliminating it from the hidden targets would be beneficial, since by including it we are effectively forcing each layer of the network to perform the full task of classification.

Additionally, by reducing the dimensionality of the hidden targets, we reduce the number of hidden units in each layer, therefore reducing the complexity of the network (hopefully without also reducing its predictive power). At $\alpha = 0$ the effective dimension is about 140 (the approximate number of pixels that vary between training examples), and it decreases steadily to 1 at $\alpha = 1$. This makes sense - as the contribution of the original dataset (which has high rank) decreases and the contribution of the labels matrix (which has rank 1) increases, the effective dimension gets smaller. This shows the influence of the labels on the original dataset.

3. Results on MNIST

3.1. Choosing α

In addition to the experiments described above, we also attempted to apply this algorithm to the MNIST dataset. Specifically, we focused on classifying the 3's and 8's. Figure 9 shows the error rate of logistic regression on classifying 3's and 8's for different hidden targets as a function of α. As was the case in the circles dataset, the error rate goes to 0 very quickly. This is again due to the fact that the label is being added as a new dimension. At around $\alpha = 1e - 9$, the logistic regression is able to pick up on this feature, and the data becomes linearly separable.

This behavior makes it very difficult to properly identify the Critical Zone. We tried several methods of attempting to eliminate this "label" dimension, with limited success. One approach we tried was to compute the correlation of each feature with the labels, and then remove the feature that was most correlated. We hoped that this would identify the label feature and remove it, thus allowing a neural net to learn a useful representation. Figure 10 shows the result of running logistic regression on this data. Initially, the error rate is higher than the baseline, and around $\alpha = .15$ it suddenly drops to 0. The fact that the error rate goes from nearly 40 % to 0 % over such a small range of values of α indicates that it would likely be difficult to train a neural network using this data. Ideally, we would like to see a smoother, slower transition to 0 % training error. This would indicate the sort of gradual transition that a neural network could reasonably be expected to learn.

3.2. Training a Neural Net

Originally we had planned to train a single layer between the hidden targets composed of a single affine layer and a non-linearity, i.e.

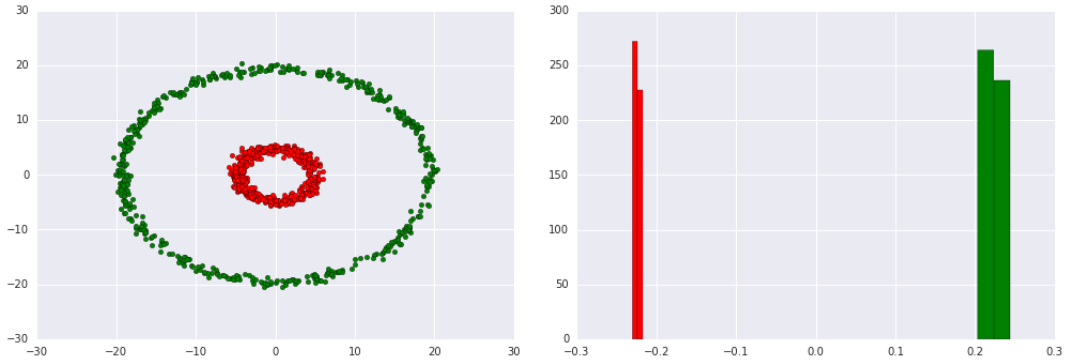
$$y = f(Wx + b),$$

where x is the input, W and b the affine parameters and f is a element wise non-linearity such as ReLU, sigmoid or tanh. This does not work since our hidden targets don't have a deterministic range while ReLU only outputs positive numbers and sigmoid and tanh output values between 0 and 1 and -1 and 1 respectively. We thus augmented our single layer with a data a second affine layer,

$$y = W_2 f(W_1 x + b_1) + b_2,$$

where W_2 and b_2 are the parameters for the second affine layer. We used ReLU for our nonlinearity. Learning the hidden targets is a regression task so we used mean-squared loss for learning the hidden targets but log-loss at the output layer where we have binary label outputs.

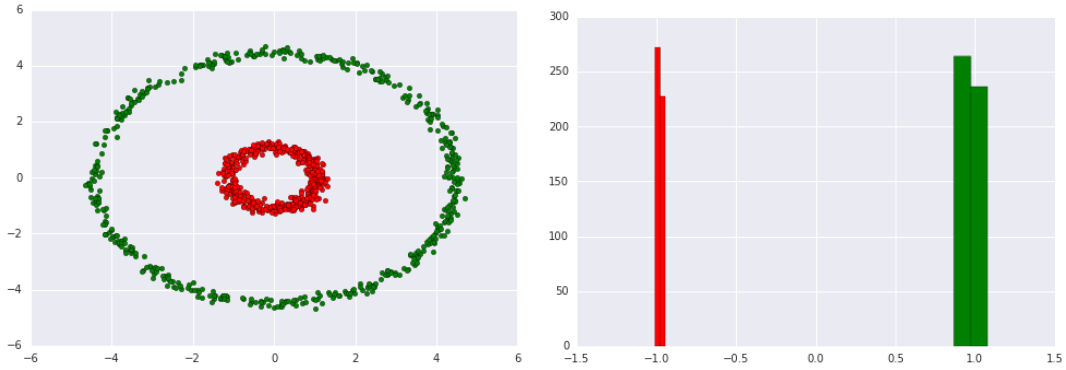
We sampled 5 α evenly on a logarithmic scale from $3 \cdot 10{-}10$ to 10^{-7}. This is the range we estimated as the



(a) Plot of first two dimensions

(b) Histogram of values of third dimension

Figure 2: Visualization of the 3 dimensions of the hidden targets when $\alpha = .05$. Note how the first 2 dimensions are almost identical to the original data and that the third is just a a class label scaled.



(a) Plot of first two dimensions

(b) Histogram of values of third dimension

Figure 3: Visualization of the 3 dimensions of the hidden targets when $\alpha = .95$

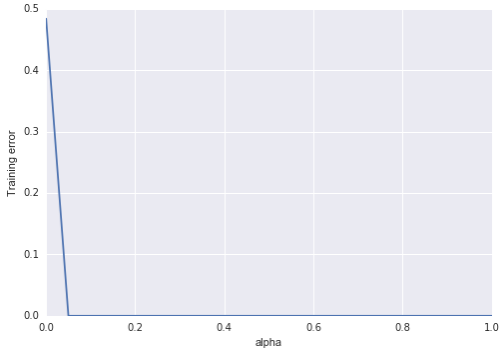


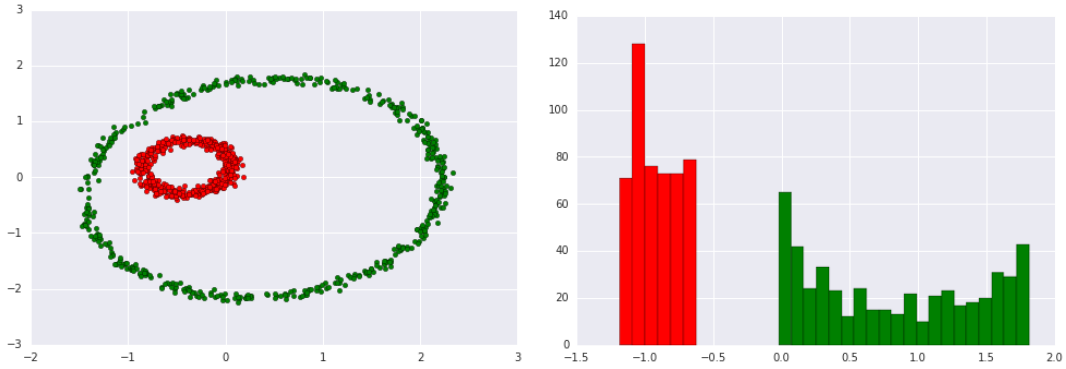
Figure 4: Training error of logistic regression for values of α from 0 to 1 on circle data

Critical Zone based on training (see figure 11). This made for 5 hidden layers for each hidden target matrix and additionally a final affine layer with log-loss.

The training set of 3s and 8s in MNIST make up about 12000 examples which we split into 90-10 train-validation sets. We trained each layer using Adam, a first-order gradient-based optimization based on adaptive estimates of lower-order moments [5]. We trained the network without weight decay.

The layers were trained one-by-one starting with the 1st layer (closest to the input). After training a layer we pass the input data through all layers we've trained so far to use as inputs for the next layer. Another way would be to train them all in parallel using the previous hidden target directly as the input. We chose the former because an earlier layer might not be able to learn a perfect mapping for its input and outputs. By passing the training data trough the previously trained layers the newer layer could possibly learn to correct it.

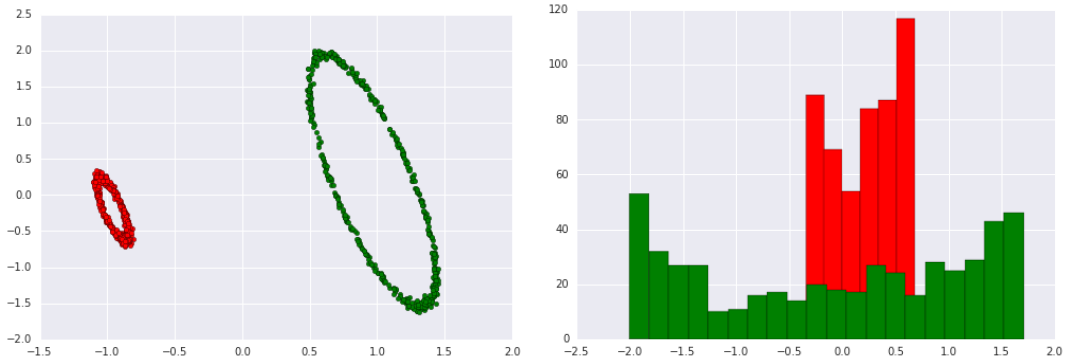
Our neural network layers were not able to learn the hidden target mappings after the first 2 hidden layers. Both the training and validation error went up significantly. Figure 12 shows the difference in loss between the first and last layers trained on their respective hidden targets. There is significant over-fitting which we could have addressed with



(a) Plot of first two dimensions

(b) Histogram of values of third dimension

Figure 5: Visualization of the 3 dimensions of the hidden targets when $\alpha = .99$



(a) Plot of first two dimensions

(b) Histogram of values of third dimension

Figure 6: Visualization of the 3 dimensions of the hidden targets when $\alpha = .992$

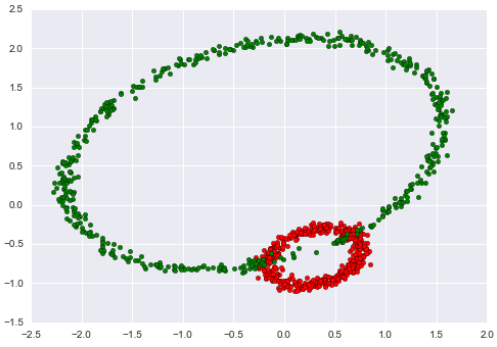


Figure 7: First two dimensions when $\alpha = .9905$

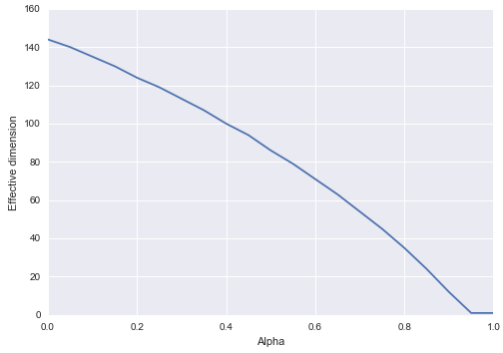


Figure 8: Effective dimension of hidden targets for MNIST dataset as a function of α

weight decay but the high training error suggest a lack of representative power of the layers.

The final accuracy of the model was 85%. This might seem good at first but comparing it to logistic regression reveals it is not. Logistic regression achieves 95% on the raw inputs. Considering that our Neural Net is equivalent to a 5 layers deep network with a logistic regression at the head we see that the representation we learned with the hidden targets decreased performance by 10%.

At this point we felt like something fundamental was going wrong with our approach. Instead of trying to improve

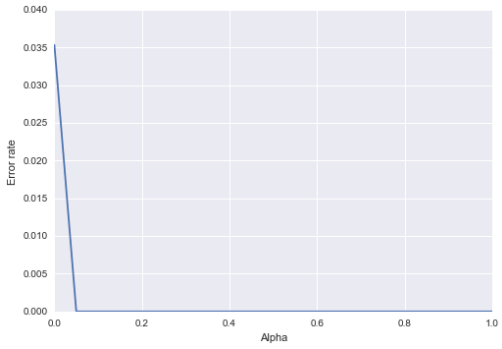


Figure 9: Training error of logistic regression for values of α from 0 to 1 on MNIST data

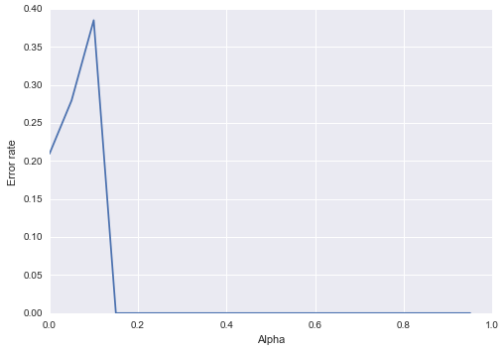


Figure 10: Training error of logistic regression for values of α from 0 to 1 on MNIST data with the feature that is most correlated with the labels removed

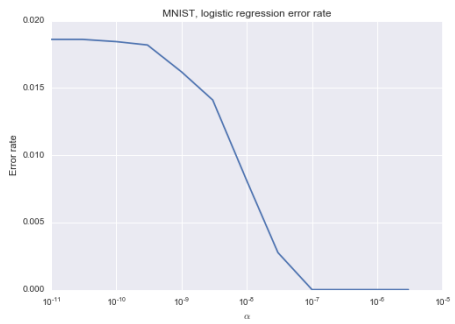


Figure 11: Training error of logistic regression on hidden targets H_α. Notice that the hidden targets become linearly separable at $\alpha = 10^{-7}$.

the architecture of our model by adding weight decay for example we abandoned MNIST in favor of the toy dataset described in previous section in order to understand our approach better.

For this part of the project we implemented the Neural Network ourselves along with the Adam update. The code can be found on Github.

4. Future work

4.1. Interpolation

We've only tried interpolating between similarities of training samples and labels with a weighted arithmetic mean. It is possible that this type of interpolation does not result in representations that are meaningful or helpful for the learning task. We considered taking a geometric or harmonic mean but those are only well defined for positive symmetric matrices. Bonnabel et al [1] introduce a new mean on positive semi-definite matrices of fixed rank that has similar properties as the geometric mean.

4.2. Residual networks

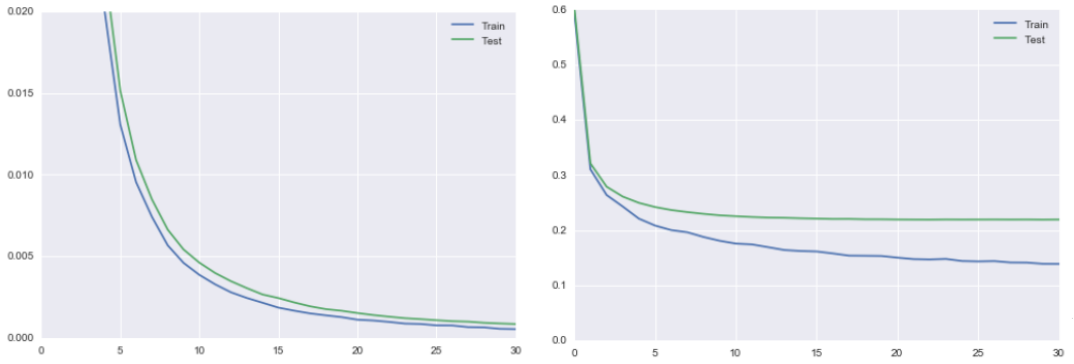
We experienced difficulty in learning a mapping between hidden vectors in a single layer neural network. This was disappointing especially since the mappings only needed to learn a small displacements between the hidden targets. One solution would be to increase the learning capacity between hidden targets by increasing the number of layers and hidden units in-between. Another possibility would be to incorporate a residual network architecture [2]. Kaiming He et al. introduced a new deep architecture that won the ImageNet classification competition in 2015. A residual block consists of a non-linear transformation of the input x of arbitrary architecture that gets added to the raw input x to form the output, i.e.

$$y = x + f(x),$$

where $f(x)$ is an arbitrary neural net module. This seems appropriate for our application since we want to learn small nonlinear displacements between hidden targets.

5. Conclusion

We set out to explore a novel algorithm for determining hidden layer targets for use in training a deep neural network. Unfortunately, producing good hidden targets turned out to be more difficult than expected. The linear interpolation between inputs and their labels tends to simply add the label as a feature, without really creating good intermediate representations. This can be seen in our plots of training error for logistic regression as a function of α, where it is evident that the data becomes linearly separable over a very small range of values of α (corresponding to the addition of the "label feature"). Attempts to fix this problem by discarding the "label feature" were not successful. While we still believe that this sort of algorithm has promise, it seems likely that this particular approach is not capable of producing good hidden targets.



(a) Mean squared loss of the 1st layer of the network. (b) Mean squared loss of the 5th layer of the network.

Figure 12: Difference between the loss at layer 1 (left) and layer 5 (right) for training on hidden targets. Not only do we have over-fitting at the 5th layer but the training loss is very high.

6. Acknowledgements

We would like to thank Professor Lawrence Saul for his guidance on this project.

References

[1] S. Bonnabel and R. Sepulchre. Riemannian metric and geometric mean for positive semidefinite matrices of fixed rank. *SIAM Journal on Matrix Analysis and Applications*, 31(3):1055–1070, 2009.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[3] S. Hochreiter. Untersuchungen zu dynamischen neuronalen netzen, 1991. Diploma thesis, Institut f. Informatik, Technische Univ. Munich.

[4] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[5] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.