10807 Topics in Deep Learning HW1

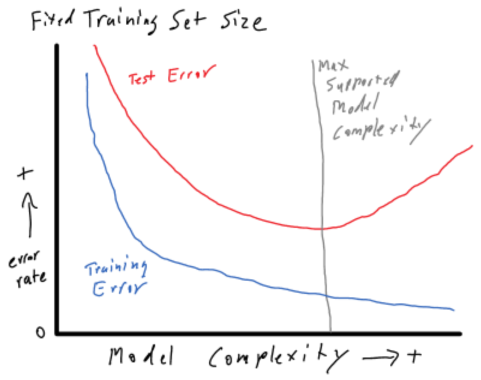
Erik Sjoberg
esjoberg
School of Computer Science
The Robotics Institute
`esjoberg@cmu.edu`

Abstract

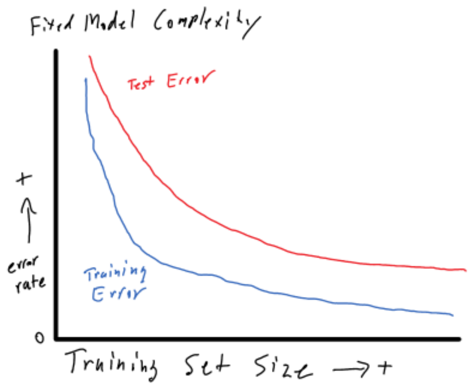
Simple implementation of backpropagation in a fully connected neural network.

1 Overfitting and Model Complexity

1.1 Typical Training Behavior: Error Rate vs Model Complexity



1.2 Typical Training Behavior: Error Rate vs Training Set Size



2 Expected Loss for Regression

2.1 a) Case q = 1

Starting with

$$E\left[L_q\right] = \int\int |y(x) - t|^q p(x,t) dt dx \tag{1}$$

using the independence of our choice of y(x) every point x, we can simplify this for the purposes of minimization to the following:

$$E\left[L_q\right] = \int |y(x) - t|^q p(t|x) dt \tag{2}$$

For the case q=1, the both sides of the integral around y(x) have linearly increasing weight, and the minimum is reached when this linear valley is centered on the average of the distribution of density.

With Q=1, taking the directional derivative of the above with respect to y(x) and setting it equal to zero gives

$$\int_{-\infty}^{y(x)} p(t|x) dt - \int_{y(x)}^{\infty} p(t|x) dt = 0 \tag{3}$$

Which is the same as saying that the expectation corresponds to the conditional median.

2.2 b) Case q -> 0

In the case of the limit where q approaches 0, the distribution becomes equal to 1 everywhere except for an infinitely narrow valley right above $y(x) = t$.

Therefore, the expected loss will be minimized when this valley is directly above the point of the function with the highest probability density. This corresponds to the conditional mode.

3 Binary Classification Error Function

We know that for general binary classification, the cross-entropy loss function takes the form $l = -\sum c_i \log(p_i)$ where c_i is the class indicator. This corresponds to the negative log likelihood.

If there is a probability ϵ that a class label is wrong, we can say that

$$p_i = \epsilon/N + (1-\epsilon)y_i$$

where N is the total number of classes, 2 in this case.

The error function then becomes:

$$l = -\sum c_i \log(\epsilon/2 + (1-\epsilon)y_i)$$

4 Generalized Gaussian

4.1 Show Distribution is Normalized

Starting with the generalized Gaussian below,

$$P(x) = \frac{q}{2(2\sigma^2)^{1/q}\Gamma(1/q)} exp(-\frac{|x|^q}{2\sigma^2})$$

first see that when q = 2, it can be shown that $\Gamma(1/2) = \sqrt{\pi}$ and $(2\sigma^2)^{1/2} = \sigma\sqrt{2}$. Canceling the 2 from the top and bottom of the left side, brings us back to the familiar left-hand form of the gaussian:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

This generalized distribution can be shown to be properly normalized (adds up to one) with the knowledge that the following equation holds in the eve more general case where x is not restricted to be positive.

$$\int \exp\left(-\frac{x^b}{2a}\right) dx = -\frac{2^{\frac{1}{b}}\, x \left(\frac{x^b}{a}\right)^{-1/b} \Gamma\left(\frac{1}{b}, \frac{x^b}{2a}\right)}{b} + \text{constant} \tag{4}$$

When x is positive, as in our example above, the second gamma term is redundant and this factor perfectly cancels out the remaining terms of our original generalized Gaussian.

4.2 Derive Log-likelihood Function

The derivation of the log-likelihood function can proceed as follows

$$L(q, \sigma, x_1, ..., x_n) = \prod_{j=1}^{n} \frac{q}{2(2\sigma^2)^{1/q}\Gamma(1/q)} exp(-\frac{|x_j|^q}{2\sigma^2})$$

$$L(q, \sigma, x_1, ..., x_n) = \frac{q}{2(2\sigma^2)^{1/q}\Gamma(1/q)} exp(-1/2\sigma^2 \sum_{j=1}^{n} |x_j|^q)$$

$$log(L(q, \sigma, x_1, ..., x_n)) = ln(\frac{q}{2(2\sigma^2)^{1/q}\Gamma(1/q)}) + ln(exp(-1/2\sigma^2 \sum_{j=1}^{n} |x_j|^q))$$

$$log(L(q, \sigma, x_1, ..., x_n)) = ln(\frac{q}{2(2\sigma^2)^{1/q}\Gamma(1/q)}) - \frac{1}{2\sigma^2} \sum_{j=1}^{n} |x_j|^q$$

5 Implementation of Backpropagation

5.1 a) Basic Generalization

The red lines in Figure 1 shows test-set performance across 200 epochs, while the blue lines show the train-set performance. The network was trained 5 times with different seeds each time, with a learning rate of 0.1.

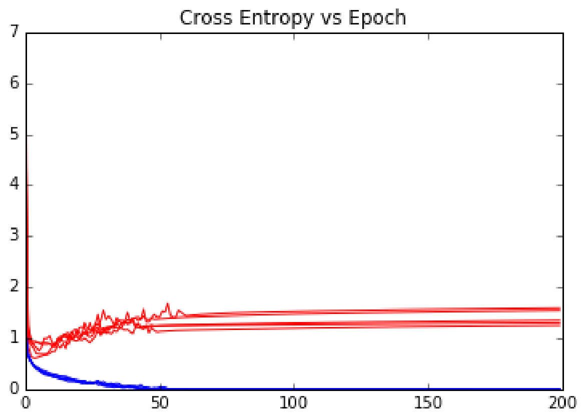


Figure 1: Training cross-entropy error (blue) and test cross-entropy error (red) for 5 seeds.

It is clear that the performance is significantly different on the test set versus the training set. With this learning rate, the network achieves it's best test-set performance after only 10 epochs, after which the cross-entropy of the test-set starts to rise. In contrast, the train-set cross-entropy performance continues to decrease for up to around 60 epochs before leveling out. The validation curve clearly shows a different behavior.

5.2 b) Classification Error

Figure 2 shows the test and train set performance as measured by the classification error across 200 epochs. The behavior of the classification training error is clearly different from that of the cross entropy test error: the classification error never actually increases as training progresses, in contrast with the cross entropy error which did rise (as expected) as training progressed.

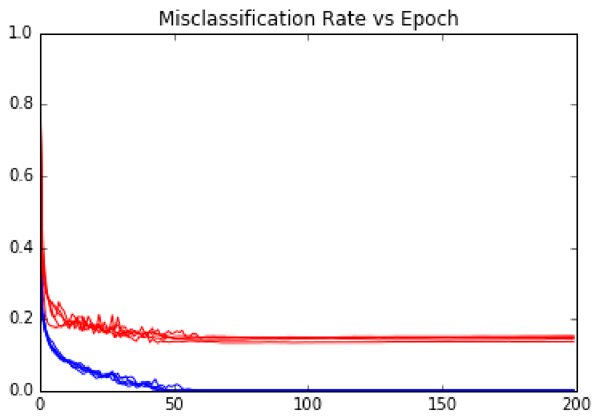


Figure 2: Training classification error (blue) and test classification error (red) for 5 seeds.

This suggests that cross-entropy may do a better job than classification error of indicating when over-fitting is taking place.

5.3 c) Visualizing Parameters

Figure 3 shows a good amount of structure in the learned parameter W from the bottom level of the network. Curves, edges, and lines reminiscent of numbers and parts of numbers can be seen.

This visible structure is suggestive that the backpropagation is working, and the network is learning generalized structure from the data.

5.4 d) Learning Rate

For my implementation, I found that the performance with lower learning rates is significantly better than higher learning rates as can be see in Figure 4. The convergence of test and training performance is far more similar with a rate of 0.01 as compared to 0.1 or 0.2. With a learning rate of 0.5, no learning appears to take place whatsoever.

Figure 5 also shows interesting behavior, with the network training significantly better in the absence of momentum. With large momentum, even the train error (blue) fails to converge, showing a U shape. Absolute values of both error functions are also better with minimal momentum. This may be due to the usage of mini-batch size one (gradient updates occur after every training sample).

To choose the best parameters, it is clearly necessary to test them empirically, perhaps performing a grid-search over promising candidate parameter sets. In this case, I would lean towards low or no momentum, as well as a low learning rate to encourage stability when training with mini-batch size equal to 1.

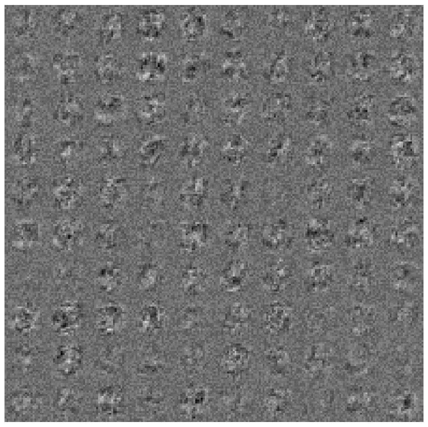


Figure 3: Visualization of 10-by-10 grid of learned parameters from the bottom layer of the network.

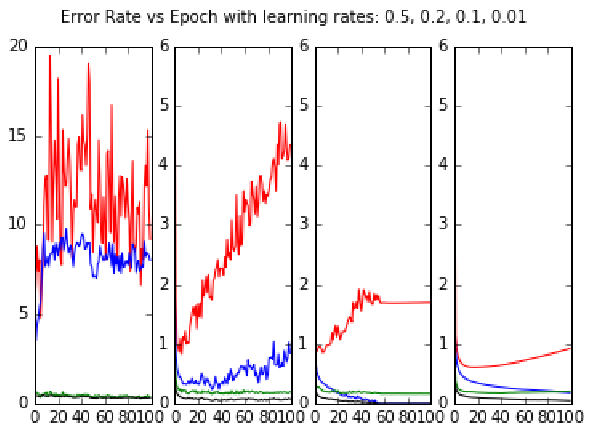


Figure 4: Visualization of error rates with various learning rates. Red and blue are cross-entropy error for test and train, while green and black are categorization error for test and train respectively.

5.5 e) Number of Hidden Units

Varying the number of hidden units, as seen in Figure 6, has an interesting effect on the error rates as training progresses. In all cases, higher hidden layer counts resulted in progressively decreased train-set error, as well as faster convergence to those lower values.

On the other hand, overfitting was a problem for networks with more hidden units as would be expected. The test-set errors (red line) can be seen to grow significantly more quickly with the networks trained with 200 and 500 hidden units.

It looks like the current setting of 100 hidden units was a sweet spot, as this setting achieved the lowest classification error among all three (0.165 vs as high as 0.211) These networks were trained with rate=0.01 and momentum=0.5, which in contrast to the default setting of hidden layers appears to be a sub-optimal setting for this network.

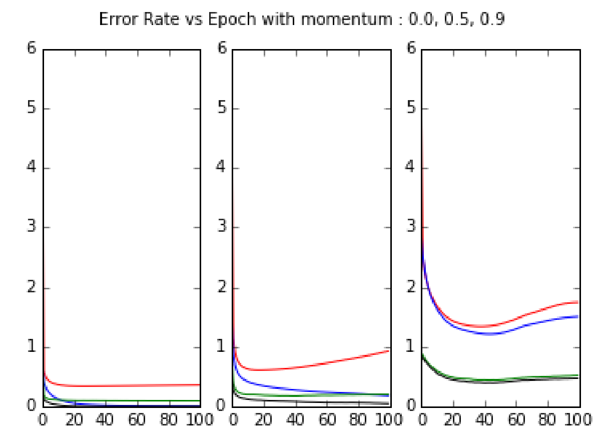


Figure 5: Visualization of error rates with various momentum values. Red and blue are cross-entropy error for test and train, while green and black are categorization error for test and train respectively.

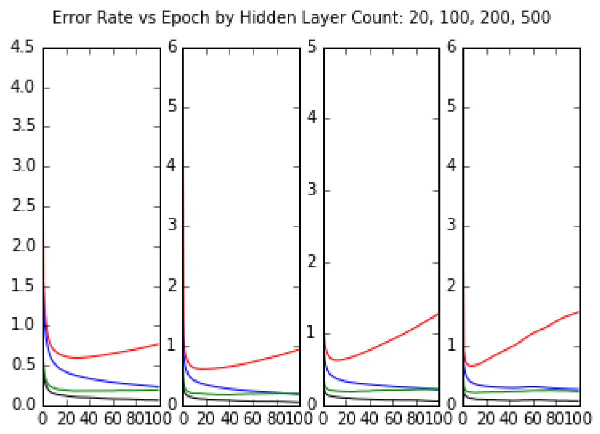


Figure 6: Visualization of error rates with various numbers of hidden layers. Red and blue are cross-entropy error for test and train, while green and black are categorization error for test and train respectively.

5.6 f) Dropout

Dropout did help with generalization issues for long-term training, but I found that early stopping was sufficient, and did not use Dropout in my best model.

5.7 g) Best-Performing Single Layer

Learning Rates

A learning rate of 0.05 seemed to be the optimal setting for my network. Higher rates tend to result in a higher minimum error as well as significant over-fitting, while lower values never seemed to find solutions which were as effective.

Momentum

Momentum did not seem to be helpful in my network, and all of my best results used 0 momentum. Larger values of momentum ended up making the training more prone to over-fitting, especially after epoch 50 or so.

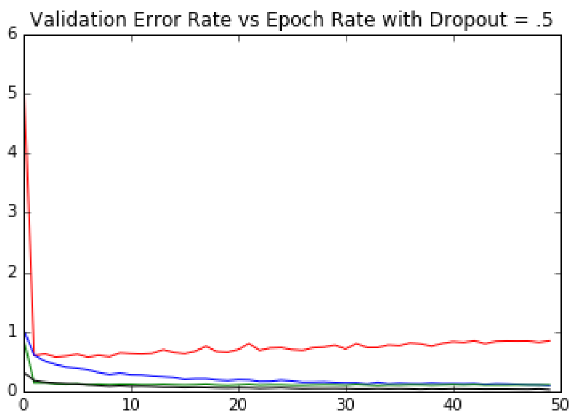


Figure 7: Visualization of error rates during training with dropout. Red and blue are cross-entropy error for test and train, while green and black are categorization error for test and train respectively.

Number of Hidden Units

The default value of 100 seemed to be a reasonable number of hidden units for this task. Lower values caused significantly higher minimum error rates (under-fitting), while higher values of 200 or 500 ended up causing the test-set performance to diverge from the train-set performance due to over-fitting.

Number of Epochs (early stopping)

Early stopping at around 30 epochs was critical for good performance of the vast majority of hyperparameter values. While low learning rates such as 0.01 sometimes did not require the early stopping, I imagine it will be an important facet of any real-world system.

Dropout values

Dropout had an interesting effect on the training error. As can be seen in Figure 9, the training error very quickly reaches a low value, then oscilates up and down instead of increasing as is usual. This appears to be thanks to the dropout-induced regularization, which is allowing for better generalization by keeping the network from over-optimizing too much.

When tested with various numbers of numbers of hidden layers, it seemed to continue to do a good job of keeping the generalization error low even for 200 hidden layers, in contrast to what happened without dropout.

L_2 Regularization

L2 regularization did not seem to have a significant effect on my results for the single-layer network with it's best values. It seems that lower learning rates and early-stopping were sufficient regularization.

Best Result

After exploring a wide range of parameter values over the previous sections, it became clear that a learning rate of 0.05, momentum of 0, a layer size of 100, and early-stopping at 30 epochs is the best performing combination for this code, achieving a minimum validation error of 7.9/

Results for Best 1-layer Performance:
Class Error (Validate): 0.079
Class Error (Test): 0.083
Class Error (Train): 0.0
CrossEnt (Validate): 0.305

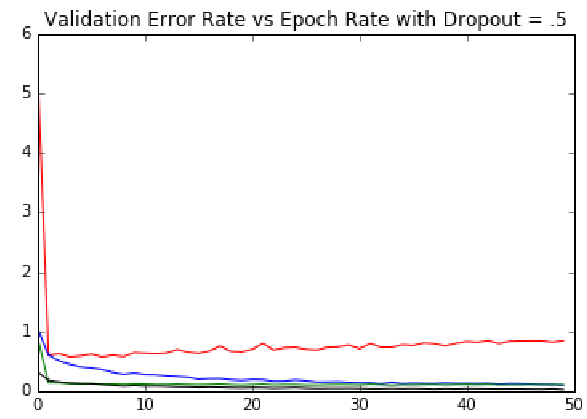


Figure 8: Visualization of error rates of standard single-layer a network trained with dropout. Red and blue are cross-entropy error for test and train, while green and black are categorization error for test and train respectively.

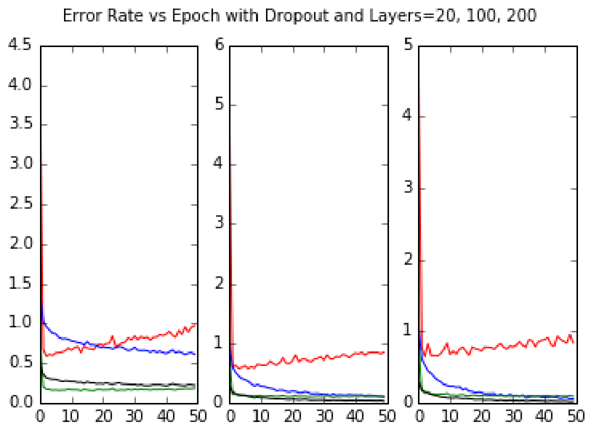


Figure 9: Visualization of error rates of standard single-layer a network trained with dropout and three different numbers of hidden layers. Red and blue are cross-entropy error for test and train, while green and black are categorization error for test and train respectively.

CrossEnt (Test): 0.330
CrossEnt (Train): 0.003

Figure 10 shows the validation error during training (red line), while Figure 11 shows a visualization of the top-level features, showing a nice level of structure in the distribution of weights corresponding to edges, lines, and circular shapes of the various digits.

5.8 h) Extension to Multiple Layers

When training a network with multiple hidden layers (two in this case), it was interesting to note that less over-fitting was observed on the test set during training out to 200 epochs for most standard settings. This was a bit unexpected, since the network with more layers has more capacity to fit a model. It may be that this is reflecting the.

To evaluate the performance of the network, a 9-cell grid-search over 3 values each of momentum and learning rate was carried out, as can be seen in Figure 12, 13 and 14. Momentum varied between

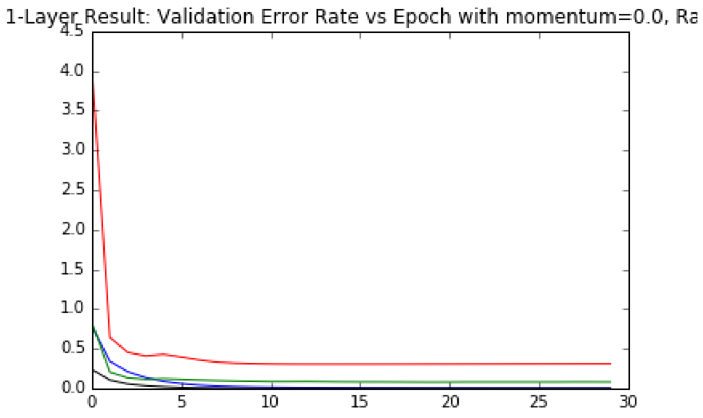


Figure 10: Visualization of VALIDATION error rates of the single-layer network with the best performance. Red and blue are cross-entropy error for validation and train, while green and black are categorization error for validation and train respectively.

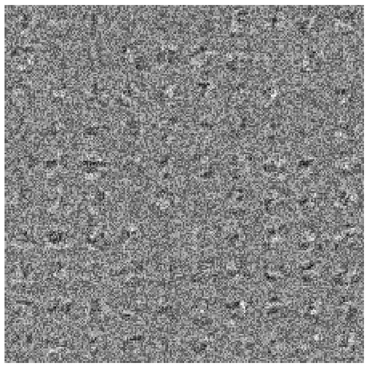


Figure 11: Best 1-layer features: visualization of 100 28x28 element top-level weights arranged in a grid.

0.0, 0.2, and 0.5, while learning rate varied between 0.1, 0.05, and 0.01. During this training dropout was 0, and hidden-layer count was 100.

As can be seen in the figures, higher learning rates (0.1+) and momentum (0.5) was catastrophic for the performance of the training. It appears to be significantly more difficult to train this 2-layer network as compared to the 1-layer counterpart.

Figure 15 shows the validation-set error of the best 2-layer performer was the test with momentum at 0 and learning rate at 0.01, achieving a classification error of 7.6% at around epoch 10. It turned out that the 2-layer network network performed slightly better than the one-layer network, likely due to the increase capacity of the model. I would have expected even better performance, but it may be that backpropagation through two layers of sigmoids is difficult due to vanishing gradients..

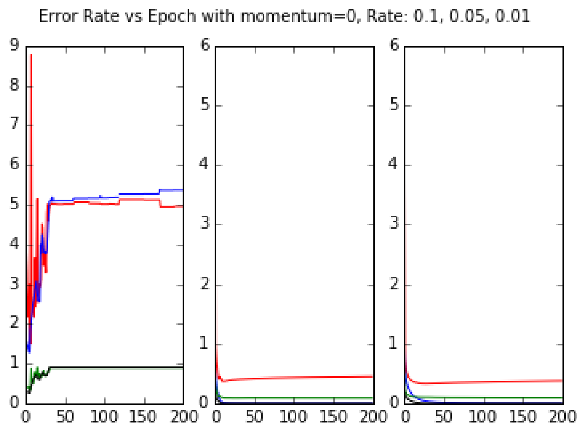


Figure 12: Visualization of error rates with various momentum values. Red and blue are cross-entropy error for test and train, while green and black are categorization error for test and train respectively.

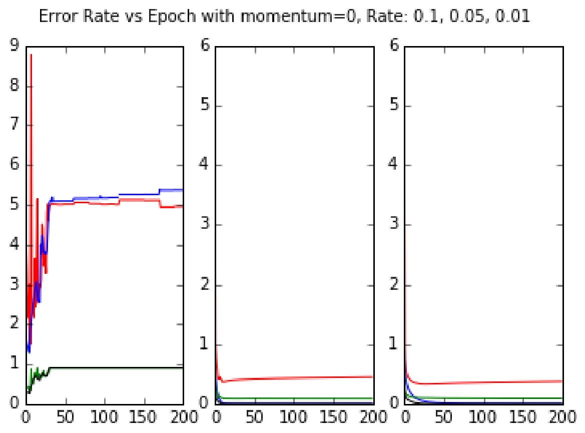


Figure 13: Visualization of error rates with various momentum values. Red and blue are cross-entropy error for test and train, while green and black are categorization error for test and train respectively.

Results for Best 2-layer Performance:
Class Error (Validate): 0.076
Class Error (Test): 0.089
Class Error (Train): 0.0
CrossEnt (Validate): 0.293
CrossEnt (Test): 0.370
CrossEnt (Train): 0.002

The feature map from this 2-layer model, shown in Figure 16, is suggestive of why the performance was better than the 1-layer network. For this 2-layer network, the top-level weight parameters were slightly more structured, showing more distinct bright and dark areas corresponding with edges and lines. Better features would naturally contribute to an advantage for the 2-layer model.

In the end, in my testing the 1-layer network DID outperform the 2-layer network.

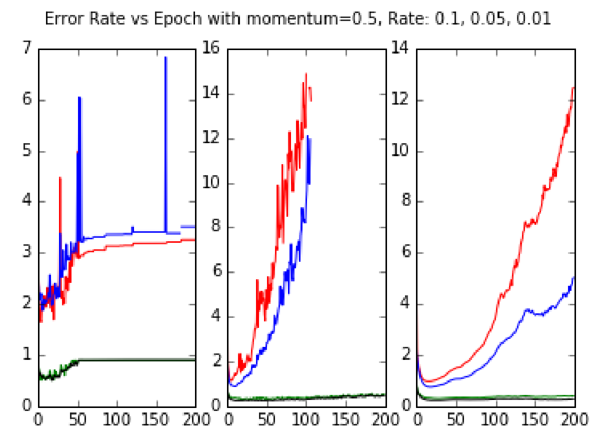


Figure 14: Visualization of error rates with various momentum values. Red and blue are cross-entropy error for test and train, while green and black are categorization error for test and train respectively.

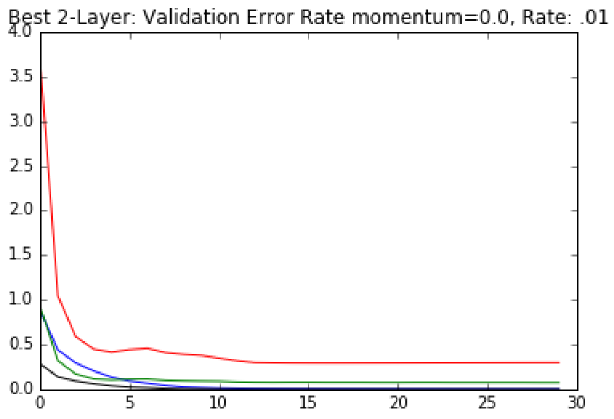


Figure 15: Visualization of VALIDATION error rates for the best two-layer network. Red and blue are cross-entropy error for test and validation, while green and black are categorization error for test and validation respectively.

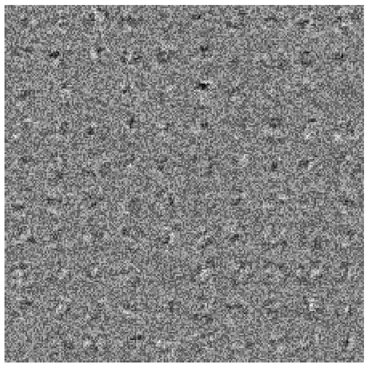


Figure 16: Best 2-layer features: visualization of 100 28x28 element top-level weights arranged in a grid.