10807 Topics in Deep Learning HW1

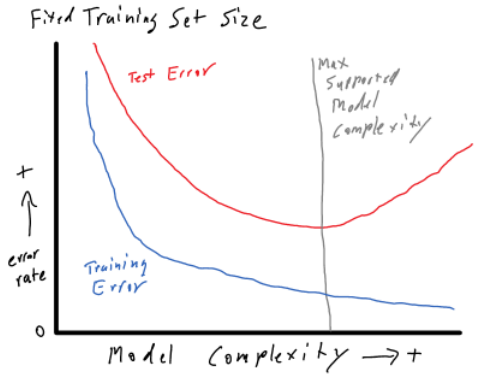
Erik Sjoberg
esjoberg
School of Computer Science
The Robotics Institute
esjoberg@cmu.edu

Abstract

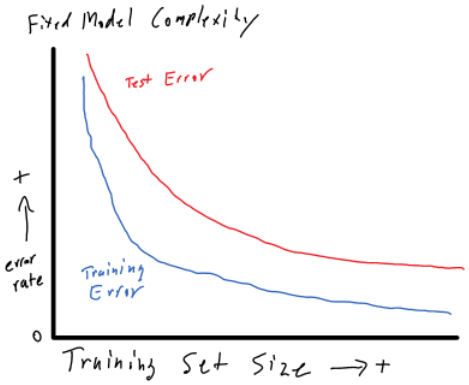
Simple implementation of backpropagation in a fully connected neural network.

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



10807 Class Project (2016).

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.

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

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.

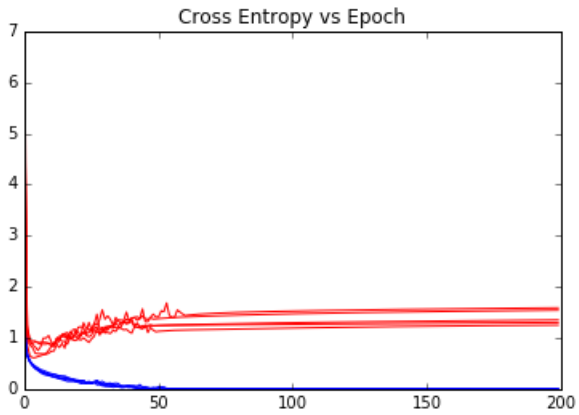


Figure 1: Training cross-entropy error (blue) and test cross-entropy error (red) for 5 seeds.

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.

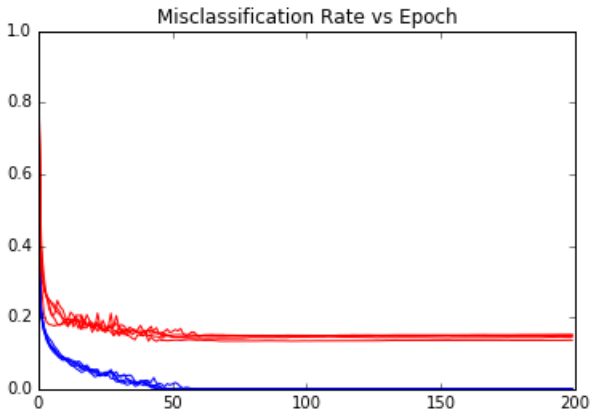


Figure 2: Training classification error (blue) and test classification error (red) for 5 seeds.

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.

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.

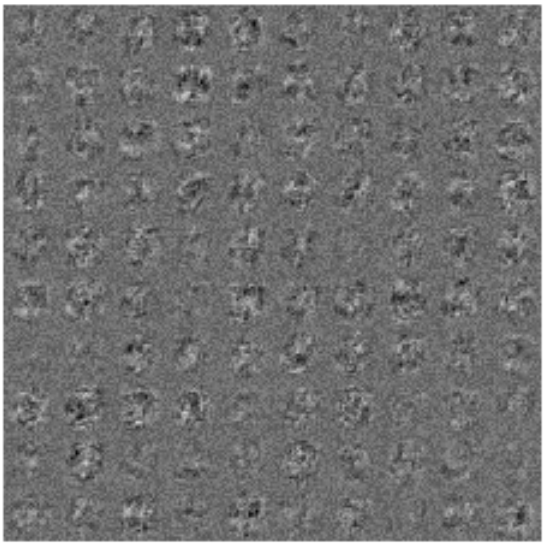


Figure 3: Visualization of 10-by-10 grid of learned parameters from the bottom layer of the network.

5.5 e) Number of Hidden Units

5.6 f) Dropout

5.7 g) Best-Performing Single Layer

5.8 h) Extension to Multiple Layers

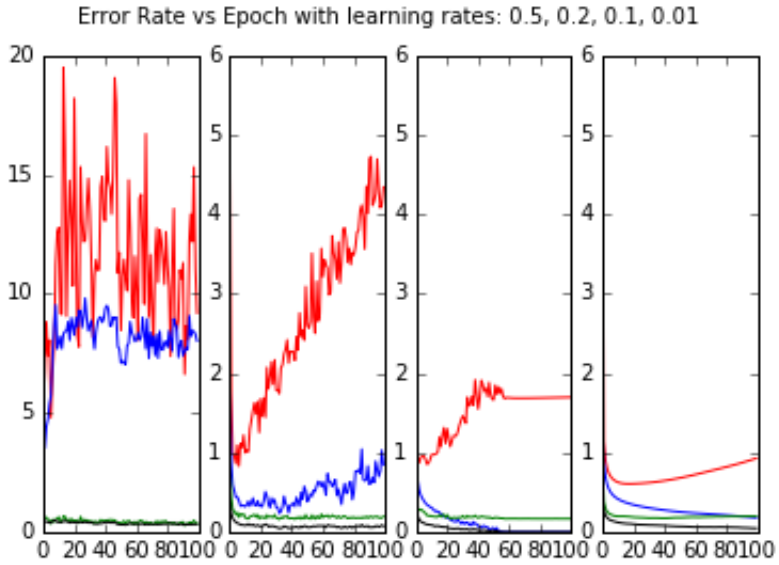


Figure 4: Visualization of error rates with various learning rates. Red and blue are cross-entropy error for test and train, while green and black are categorization error for test and train respectively.

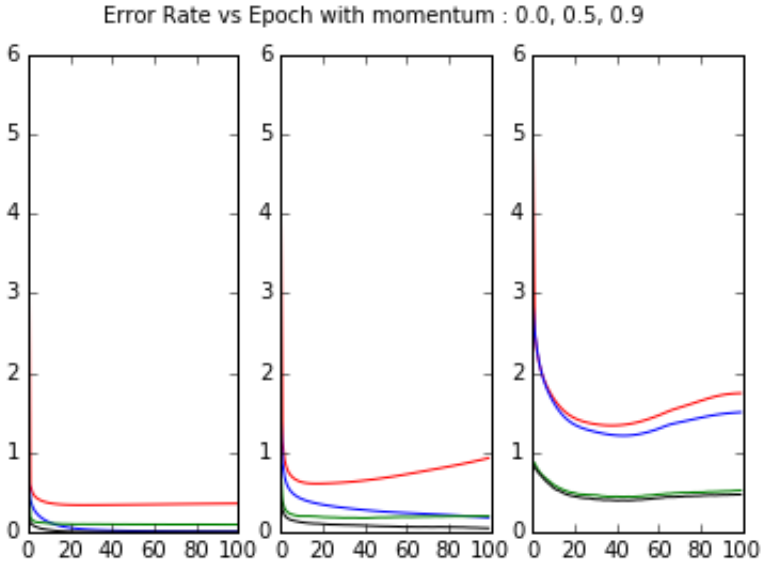


Figure 5: Visualization of error rates with various momentum values. Red and blue are cross-entropy error for test and train, while green and black are categorization error for test and train respectively.