HW3: Neural Networks
Advanced Machine Learning

Francisco Calderon

2/15/2017

1.0 Back Propagation for Classification

The derivatives for w are derived in the following way:

$$\hat{y}(x,w) = \sigma\left(\sum_{j=1}^{M} w_j^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x^i + w_{jo}^{(1)}\right) + w_0^{(2)}\right)$$

$$Err(w) = -\sum_{n=1}^{N} \{ y_n \log \hat{y}(x_n,w) + (1-y_n)\log(1-\hat{y}(x_n\ \ ))\}$$

$$a_j^{(1)} = \sum_{i=1}^{D} w_{ji}^{(1)} x^i + w_{jo}^{(1)}$$

$$a_n^{(2)} = \sum_{j=1}^{M} w_j^{(2)} h(a_j^{(1)}) + w_0^{(2)}$$

$$\frac{\partial Err}{\partial w_{ji}^{(1)}} = \left(\frac{-y_n}{\hat{y}(x_n,w)} + \frac{(1-y_n)}{1-\hat{y}(x_n,w)}\right)\sigma'(a^{(2)}) w_j^{(2)} h'(a_j^{(1)}) x_n^i$$

$$\frac{\partial Err}{\partial w_{jo}^{(1)}} = \left(\frac{-y_n}{\hat{y}(x_n,w)} + \frac{(1-y_n)}{1-\hat{y}(x_n,w)}\right)\sigma'(a^{(2)}) w_j^{(2)} h'(a_j^{(1)})$$

$$\frac{\partial Err}{\partial w_j^{(2)}} = \left(\frac{-y_n}{\hat{y}(x_n,w)} + \frac{(1-y_n)}{1-\hat{y}(x_n,w)}\right)\sigma'(a_n^{(2)}) h(a_j^{(1)})$$

$$\frac{\partial Err}{\partial w_0^{(2)}} = \left(\frac{-y_n}{\hat{y}(x_n,w)} + \frac{(1-y_n)}{1-\hat{y}(x_n,w)}\right)\sigma'(a_n^{(2)})$$

where $\sigma(a) = \frac{1}{1+\exp(-a)}$

and $h(a) = \max(0, a)$

The back propagation equations then become the following:

$$ {}^{t+1}w_{ji}^{(1)} = {}^{t}w_{ji}^{(1)} - \eta \sum_{i=1}^{N} \left(\frac{-y_n}{\hat{y}(x_n,w)} + \frac{(1-y_n)}{1-\hat{y}(x_n,w)}\right) \sigma'\left(a_n^{(2)}\right) w_{jn}^{(2)} h'\left(a_{jn}^{(1)}\right) x_n^i $$

$$ {}^{t+1}w_{j0}^{(1)} = {}^{t}w_{j0}^{(1)} - \eta \sum_{i=1}^{N} \left(\frac{-y_n}{\hat{y}(x_n,w)} + \frac{(1-y_n)}{1-\hat{y}(x_n,w)}\right) \sigma'\left(a_n^{(2)}\right) w_{jn}^{(2)} h'\left(a_j^{(1)}\right) $$

$$ {}^{t+1}w_{j}^{(2)} = {}^{t}w_{j}^{(2)} - \eta \sum_{i=1}^{N} \left(\frac{-y_n}{\hat{y}(x_n,w)} + \frac{(1-y_n)}{1-\hat{y}(x_n,w)}\right) \sigma'\left(a_n^{(2)}\right) h\left(a_{jn}^{(1)}\right) $$

$$ {}^{t+1}w_{0}^{(2)} = {}^{t}w_{0}^{(2)} - \eta \sum_{i=1}^{N} \left(\frac{-y_n}{\hat{y}(x_n,w)} + \frac{(1-y_n)}{1-\hat{y}(x_n,w)}\right) \sigma'\left(a_n^{(2)}\right) $$

with learning rate η

2.0 Neural Networks for Handwritten Digit Recognition

2.1 Different Learning Rates

Learning Rate	Validation Accuracy
1.0	0.0892
0.1	0.1010
0.01	0.9731
0.001	0.9782
0.0001	0.9567
0.00001	0.8982

Table 1: Validation accuracy as learning rate decreases

As show in Table 1, the potential optimal learning rate is between the values 0.001 and 0.01. In order to interpolate the optimal learning rate, we will search through values in between 0.001 and 0.01 with an interval of 0.001.

One can see from Table 2 the highest validation accuracy corresponds to a learning rate of 0.003.

2.2 Different Layer Sizes

After training increasing amount of layer sizes, a layer size of 2000 had the highest validation accuracy. As show in Table 3, the validation accuracy for layer size of 2000 is 0.9766. One of the signs of over-fitting is an increase in training accuracy and a drop in validation/test accuracy. While the validation accuracy does not drop, there is higher gap between the train accuracy and validation accuracy for layer size 100. Therefore layer size 100 is over-fit.

Learning Rate	Validation Accuracy
0.002	0.9790
0.003	0.9814
0.004	0.9790
0.005	0.9782
0.006	0.9708
0.007	0.9758
0.008	0.9771
0.009	0.9747

Table 2: Interpolating the Optimal Learning Rate

Layer Size	Validation Accuracy	Train Accuracy	Accuracy Difference
10	0.9097	0.9223	0.0126
50	0.9634	0.9822	0.0188
100	0.9673	0.9886	0.0213
300	0.9701	0.9884	0.0183
1000	0.9745	0.9890	0.0145
2000	0.9766	0.9887	0.0121

Table 3: Validation accuracy as Layer Size Increases

2.3 Neural Network with L2 Regularization

IsRegularized	Learning Rate	Validation Accuracy	Training Accuracy	N-Epochs
No	0.01	0.9757	0.9881	10
No	0.001	0.9806	0.9954	10
Yes	0.001	0.9742	0.9804	30

Table 4: Regularization vs Non Regularized Networks

From the figures shown in Table 4, one can see the validation accuracy increases a bit by decreasing the learning rate. Additionally, the gap between validation accuracy and training accuracy widens when we drop the learning rate with no regularization. This a sign of an over-fitted model. When we employ the L2 regularization, the training error increases as well as the validation error, but the regularized model is not over fit.

2.4 Neural Network with Dropout

The most optimal dropout weight according to the results shown in Table 5 is a weight of 0.3 that yields a validation accuracy of 0.9841. This can be explained by the results that show models with dropout weights achieved the lowest validation errors. A similar model with no dropout weight and learning rate of .001 achieved an error of 1.94%, where as most models with a dropout weight from 0.1 to 0.7 had errors that were much lower than that. Drop out was a more effective change in this case than L2 regularization was. A model with L2 regularization and 0.001 learning rate achieved an error rate of 2.58%, while most models with the dropout technique, with the exception of models with 0.8 and 0.9 dropout weights, generated error rates that were much smaller than that.

Dropout Weight	Validation Accuracy	Training Accuracy
0.1	0.9826	0.9984
0.2	0.9840	0.9978
0.3	0.9841	0.9963
0.4	0.9827	0.9938
0.5	0.9835	0.9901
0.6	0.9827	0.9844
0.7	0.9813	0.9749
0.8	0.9563	0.9769
0.9	0.8978	0.9653
1.0	1.0	0.9818

Table 5: Validation Accuracy as Dropout Weight Increases

2.5 Neural Network with 3 Layers

First Hidden Layer	Second Hidden Layer	Validation Accuracy
300	100	0.9793
500	150	0.9777
500	300	0.9816

Table 6: Regularization vs Non Regularized Networks

Some of the best performing 3 layer size combinations as displayed in `http://yann.lecun.com/exdb/mnist/` are included in Table 6. The validation accuracy for each are pretty high, with 500+300 layer size combination having the best validation accuracy. With additional tuning, the performance of these models can easily be improved.

2.5.1 Code Used to Add Extra Layer

```
def get_3layer_model(lr=0.001, M=300, K=100):
    model = Sequential()
    model.add(Dense(M, input_dim=num_pixels, init='normal', activation='relu'))
    model.add(Dense(K, init='normal', activation='relu'))
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    model.optimizer.lr.set_value(lr)
    return model

# NN 300 + 100
print "\n−−−−−−−−−−−− NN Model with 3 layer 300 + 100 −−−−−−−−−−−−\n"
model = get_3layer_model(lr=0.003, M=300, K=100)
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), nb_epoch=10,
    batch_size=200, verbose=2)
scores = model.evaluate(X_test, Y_test, verbose=0)
print("Baseline Error: %.2f%%" % (100-scores[1]*100))

# NN 500 + 150
print "\n−−−−−−−−−−−− NN Model with 3 layer 500 + 150 −−−−−−−−−−−−\n"
model = get_3layer_model(lr=0.003, M=500, K=150)
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), nb_epoch=10,
    batch_size=200, verbose=2)
scores = model.evaluate(X_test, Y_test, verbose=0)
print("Baseline Error: %.2f%%" % (100-scores[1]*100))

# NN 500 + 300
print "\n−−−−−−−−−−−− NN Model with 3 layer 500 + 300 −−−−−−−−−−−−\n"
model = get_3layer_model(lr=0.003, M=500, K=300)
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), nb_epoch=10,
    batch_size=200, verbose=2)
scores = model.evaluate(X_test, Y_test, verbose=0)
print("Baseline Error: %.2f%%" % (100-scores[1]*100))
```