Deep Learning and Boosting

Anjali K Prasad

November 2016

1 Boosting

a)

$$L(y_i, \hat{y_i}) = (y_i - \hat{y_i})^2$$
$$\frac{\partial L(y_i, \hat{y_i})}{\partial \hat{y_i}} = -2(y_i - \hat{y_i})$$

b) To show that the optimal value of the step size gamma can be computed in the closed form in this step, thus the selection rule for h* can be derived independent of gamma.

$$h^* = argmin_{h \epsilon H}[min_{\gamma \epsilon R} \sum_{i=1}^{n} (-g_i - \gamma h(x_i))^2]$$

Differentiating h* with respect to gamma

$$\frac{\partial h^*}{\partial \gamma} = \sum_{i=1}^{n} 2(-g_i - \gamma h(x_i))h(x_i) = 0$$

$$\sum_{i=1}^{n} 2(-g_i h(x_i) - \gamma (h(x_i))^2) = 0$$

$$\sum_{i=1}^{n} \gamma (h(x_i))^2) = -\sum_{i=1}^{n} g_i h(x_i)$$

$$\gamma = -\frac{\sum_{i=1}^{n} g_i}{\sum_{i=1}^{n} h(x_i)}$$

replacing gamma in original equation

$$h^* = argmin_{h \epsilon H}[min_{\gamma \epsilon R} \sum_{i=1}^{n} (-g_i + \frac{\sum_{i=1}^{n} g_i}{\sum_{i=1}^{n} h(x_i)} h(x_i))^2]$$

c) To select the step size that minimizes the loss:

$$\alpha^* = argmin_{\alpha \epsilon R} \sum_{i=1}^{n} L(y_i, \hat{y_i} + \alpha h^*(x_i))$$

$$\alpha^* = argmin_{\alpha \epsilon R} \sum_{i=1}^{n} (y_i - (\hat{y_i} + \alpha h^*(x_i)))^2$$

Differentiating wrt alpha

$$\frac{\partial \alpha^*}{\partial \alpha} = \sum_{i=1}^{n} 2(y_i - \hat{y_i} - \alpha h^*(x_i))h^*(x_i) = 0$$

$$= \sum_{i=1}^{n} (y_i h^*(x_i) - \hat{y_i} h^*(x_i) - \alpha (h^*(x_i))^2) = 0$$

$$= \sum_{i=1}^{n} \alpha(h^*(x_i))^2) = \sum_{i=1}^{n} (y_i h^*(x_i) - \hat{y_i} h^*(x_i))$$

$$\alpha = \frac{\sum_{i=1}^{n}(y_i - \hat{y_i})}{\sum_{i=1}^{n}(h^*(x_i))}$$

putting pack the value of alpha in the original equation:

$$\alpha^* = argmin_{\alpha \epsilon R} \sum_{i=1}^{n} (y_i - (\hat{y_i} + \frac{\sum_{i=1}^{n}(y_i - \hat{y_i})}{\sum_{i=1}^{n} h^*(x_i))} h^*(x_i)))^2$$

$$\alpha^* = argmin_{\alpha \epsilon R} \sum_{i=1}^{n} (y_i - \hat{y_i} - \frac{\sum_{i=1}^{n}(y_i - \hat{y_i})}{\sum_{i=1}^{n}(h^*(x_i))} h^*(x_i))^2$$

Finally, we perform the following updating step:

$$\hat{y_i} < -\hat{y_i} + \alpha^* h^*(x_i)$$

where alpha* is equal to the value derived by us.

2 Neural Networks

a) We have to show that a neural network with a single logistic output and with linear activation functions in the hidden layers (possibly with multiple hidden layers) is equivalent to the logistic regression. Let there be n layers with layer 1 being the input layer and layer n being the output layer. All the layers in between are hidden layers. All these hidden layers have linear activation. The output layer has sigmoid activation.

For all layers except the input and the output layer, the activation function is linear. Therefore for all layers l between 2 and n-1 we have:

$$a^l = z^l = w^l a^{l-1} + b^l$$

For layer l = n we have :

$$a^n = \sigma(z^n) = \sigma(w^n a^{n-1} + b^n) = \frac{1}{1 + e^{-(w^n a^{n-1} + b^n)}} = \frac{1}{1 + e^{-(w^n (w^{n-1} a^{n-2} + b^{n-1}) + b^n)}}$$

expanding the above equation Here we can clearly see that if we substitute :

$$W = w^n w^{n-1} .... w^3 w^2$$

$$B = b^n + w^n b^{n-1} + ... + w^n w^{n-1} ... w^3 b^2$$

So here we can see :

$$a^n = \frac{1}{1 + e^{-(Wx+B)}}$$

this is same as logistic regression where Wx+B is linear function

b) We are given cost function to be

$$C = L(y, \hat{y}) = \frac{1}{2}((y_1 - \hat{y_1})^2 + (y_2 - \hat{y_2})^2)$$

we know that

$$\hat{y_1} = \sum_{k=1}^{4} v_{1k}^3 z_k^2$$

$$\hat{y_2} = \sum_{k=1}^{4} v_{2k}^3 z_k^2$$

therefore:

$$\hat{y_j} = \sum_{j=1}^{2} \sum_{k=1}^{4} v_{jk}^3 z_k^2$$

so our cost function becomes:

$$C = L(y, \hat{y}) = \frac{1}{2}\sum_{j=1}^{2}(y_j^3 - \hat{y_j^3})^2$$

$$C = L(y, \hat{y}) = \frac{1}{2}\sum_{j=1}^{2}(y_j^3 - \sum_{k=1}^{4} v_{jk}^2 z_k^2)^2$$

partially differentiating the above equation wrt v

$$\frac{\partial C}{\partial v_{jk}^3} = -\sum_{j=1}^{2}(y_j^3 - \sum_{k=1}^{4} v_{jk}^2 z_k^2)\sum_{k=1}^{4} z_k^2$$

let us assume the following:

$$s_j^3 = \sum_{k=1}^{4} v_{jk}^3 z_k^2$$

$$s_k^2 = \sum_{i=1}^{3} w_{ki} x_i$$

Therefore we can derivate:

$$y_j^3 = s_j^3$$

$$z_k^2 = tanh(s_k^2)$$

Let the delta be the error function as given below:

$$\delta_j^3 = \frac{\partial C}{\partial s_j^3}$$

$$\delta_k^2 = \frac{\partial C}{\partial s_k^2}$$

We know that the derivative of cost function C wrt v can also be taken in the following way using chain rule:

$$\frac{\partial C}{\partial v_{jk}^3} = \frac{\partial C}{\partial s_j^3}\frac{\partial s_j^3}{\partial v_{jk}^3} = \delta_j^3 z_k^2$$

comparing this equation with previous equation we get :

$$\delta_j^3 = -\sum_{j=1}^{2}(y_j^3 - \sum_{k=1}^{4} v_{jk}^2 z_k^2)$$

Now lets move on to w.

$$\delta_k^2 = \frac{\partial C}{\partial s_j^3}\frac{\partial s_j^3}{\partial z_k^2}\frac{\partial z_k^2}{\partial s_k^2} = \delta_j^3 v_{jk}^3 \sum_{k=1}^{4} tanh'(w_{ki}^2 x_i) = (1 - (x_i^1)^2)\sum_{k=1}^{4} v_{jk}^3 \delta_j^3$$

$$\frac{\partial C}{\partial w_{ki}^2} = \frac{\partial C}{\partial s_j^3}\frac{\partial s_j^3}{\partial z_k^2}\frac{\partial z_k^2}{\partial s_k^2}\frac{\partial s_k^2}{\partial w_{ki}^2} = \delta_j^3 v_{jk}^3 \sum_{k=1}^{4} tanh'(w_{ki}^2 x_i) x_i^1$$

If we replace the walue of delta and derivative of tanh we will get the following result:

$$-\sum_{j=1}^{2}(y_j^3 - \sum_{k=1}^{4} v_{jk}^2 (tanh(\sum_{i=1}^{3} w_{ki}^2 x_i^1)))v_{jk}^2(1 - tanh^2(\sum_{i=1}^{3} w_{ki}^2 x_i^1))\sum_{i=1}^{3} x_i^1$$

This can also be written as :

$$\frac{\partial C}{\partial w_{ki}^2} = (1 - (x_i^1)^2)x_i^1 \sum_{k=1}^{4} v_{jk}^3 \delta_j^3$$

Update rule for v:

$$v_{jk}^3 \leftarrow v_{jk}^3 - \eta z_k^2 \delta_j^3$$

Update rule for w:

$$w_{ki}^2 \leftarrow w_{ki}^2 - \eta x_i^1 \delta_k^2$$

I have already defined the values for deltas in the solution

3 Programming: Deep Learning

d) Linear Activation

```
Score for architecture = [50, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = linear: 0.827624651274
Score for architecture = [50, 50, 2], lambda = 0.0, decay = 0.0, momentum = 0.0,
    actfn = linear: 0.84011840623
Score for architecture = [50, 50, 50, 2], lambda = 0.0, decay = 0.0, momentum = 0.0,
    actfn = linear: 0.842578701193
Score for architecture = [50, 50, 50, 50, 2], lambda = 0.0, decay = 0.0, momentum = 0.0,
    actfn = linear: 0.846000073794
Best Config: architecture = [50, 50, 50, 50, 2], lambda = 0.0, decay = 0.0, momentum = 0.0,
    actfn = linear, best_acc=0.846000073794
```

Trend: A Linear model added to linear model gives linear model intuitively. In the hidden layers we have linear activation the combination of which is also linear. The output layer has softmax activation. So the non linearity is introduced only in the last layer. So increasing the number of layer will not drastically increase the fitting and concurrently overfitting.Therefore the model fits better with more number of neurons in this case.We can see that with increasing number of layers, our test accuracy is increasing.

```
Score for architecture = [50, 50, 2], lambda = 0.0, decay = 0.0, momentum = 0.0,
    actfn = linear: 0.839464888376
Score for architecture = [50, 500, 2], lambda = 0.0, decay = 0.0, momentum = 0.0,
    actfn = linear: 0.840233727115
Score for architecture = [50, 500, 300, 2], lambda = 0.0, decay = 0.0, momentum = 0.0,
    actfn = linear: 0.844193285229
Score for architecture = [50, 800, 500, 300, 2], lambda = 0.0, decay = 0.0, momentum = 0.0,
    actfn = linear: 0.847845306472
Score for architecture = [50, 800, 800, 500, 300, 2], lambda = 0.0, decay = 0.0, momentum = 0.0,
    actfn = linear: 0.849037023588
Best Config: architecture = [50, 800, 800, 500, 300, 2], lambda = 0.0, decay = 0.0, momentum = 0.0,
    actfn = linear, best_acc = 0.849037023588
Time taken for linear activation: 231.666823864
```

Trend: Multiple hidden layers made of linear activation is equivalent to one hidden layer with linear activation. This is because intuitively when we add 2 lines, we get another line. So if there are say 2 hidden layers, we can tweak the weights and make the same architecture with two hidden layers into one hidden layer. In this case the accuracy is increasing with the increase number of layers. But we have to note that by tweaking the weights we can actually achieve the same accuracy as in the best model architecture = [50, 800, 800, 500, 300, 2] with just 1 hidden layer.A Linear model added to linear model gives linear model intuitively. In the hidden layers we have linear activation the combination of which is also linear. The output layer has softmax activation. So the non linearity is introduced only in the last layer. So increasing the number of layer will not drastically increase the fitting and concurrently overfitting.Therefore the model fits better with more number of neurons in this case.We can see that with increasing number of layers, our test accuracy is increasing.

$$o = f(W\bar{x})$$

$$o_1 = W_1\bar{x}$$

$$o_2 = W_2 o_1 = W_2 W_1 \bar{x}$$

e) Sigmoid activation

```
Score for architecture = [50, 50, 2], lambda = 0.0, decay = 0.0, momentum = 0.0,
    actfn = sigmoid: 0.739976159265
Score for architecture = [50, 500, 2], lambda = 0.0, decay = 0.0, momentum = 0.0,
    actfn = sigmoid: 0.767347094244
Score for architecture = [50, 500, 300, 2], lambda = 0.0, decay = 0.0, momentum = 0.0,
    actfn = sigmoid: 0.7205627933
Score for architecture = [50, 800, 500, 300, 2], lambda = 0.0, decay = 0.0, momentum = 0.0,
    actfn = sigmoid: 0.7205627933
Score for architecture = [50, 800, 800, 500, 300, 2], lambda = 0.0, decay = 0.0, momentum = 0.0,
    actfn = sigmoid: 0.7205627933
Best Config: architecture = [50, 500, 2], lambda = 0.0, decay = 0.0, momentum = 0.0,
```

```
    actfn = sigmoid, best_acc = 0.767347094244
Time taken for Sigmoid activation: 815.819905996
```

Trend: The best result for sigmoid activation is given by architecture = [50, 500, 2]. The 1st architecture = [50,50,2] has very few neurons as compared to the data we have in hand. Too few nodes will lead to high error for the system as the predictive factors is too complex for a small number of nodes to capture. The second architecture = [50, 500, 2] has sufficient number of nodes in its hidden layer to properly fit the data and get best accuracy. Further increasing the hidden layers in later architectures leads to overfitting of the training set which in turn reduces the accuracy of the testing set. Too many hidden layers will overfit the training data and not generalize well. This is the reason 3rd ,4th and 5th architecture have lower accuracy than 2nd architecture.

Different from linear activations Sigmoid activation is different from linear activation. We can construct entirely different (nonlinear) models in by introducing nonlinearity which extends the kinds of functions that we can represent with our neural network. A straight line can rarely overfit real data in the hidden layers. Therefore we have not reached the overfitting trend in linear activation which is visible in sigmoid activation.

Time:

The time take by sigmoid is 815.819905996 which is much greater than the time taken by linear activation 231.666823864. This is because sigmoid calculation requires computation of exponential function which is a heavy computation :

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Whereas linear functions are just linear combination of X and Weights which is very cheap to calculate wrt sigmoid function.

$$y = w^T X + b$$

f) ReLu activation

```
Score for architecture = [50, 50, 2], lambda = 0.0, decay = 0.0, momentum = 0.0,
    actfn = relu: 0.79886979465
Score for architecture = [50, 500, 2], lambda = 0.0, decay = 0.0, momentum = 0.0,
    actfn = relu: 0.820397497014
Score for architecture = [50, 500, 300, 2], lambda = 0.0, decay = 0.0, momentum = 0.0,
    actfn = relu: 0.815054010381
Score for architecture = [50, 800, 500, 300, 2], lambda = 0.0, decay = 0.0, momentum = 0.0,
    actfn = relu: 0.810902240864
Score for architecture = [50, 800, 800, 500, 300, 2], lambda = 0.0, decay = 0.0, momentum = 0.0,
    actfn = relu: 0.80498211773

Best Config: architecture = [50, 500, 2], lambda = 0.0, decay = 0.0, momentum = 0.0,
    actfn = relu, best_acc = 0.820397497014
Time taken for Relu activation: 391.832270145
```

Trend: The best result for ReLu activation is given by architecture = [50, 500, 2]. The 1st architecture = [50,50,2] has very few neurons as compared to the data we have in hand. Too few nodes will lead to high error for the system as the predictive factors is too complex for a small number of nodes to capture. The second architecture = [50, 500, 2] has sufficient number of nodes in its hidden layer to properly fit the data and get best accuracy. Further increasing the hidden layers in later architectures leads to overfitting of the training set which in turn reduces the accuracy of the testing set. Too many hidden layers will overfit the training data and not generalize well. This is the reason 3rd ,4th and 5th architecture have lower accuracy than 2nd architecture.

Different from linear activations relu activation is different from linear activation. We can construct entirely different (nonlinear) models in by introducing nonlinearity which extends the kinds of functions that we can represent with our neural network. A straight line can rarely overfit real data in the hidden layers. Therefore we have not reached the overfitting trend in linear activation which is visible in relu activation.

Time:

The time taken for relu is 391.832270145 which is a bit higher than the time taken by linear activation 231.666823864. This is because computation of max function takes more time on the machine than any combination of linear function.

The time taken for relu is 391.832270145 which is much lower than the time taken by sigmoid activation 815.819905996 this is because ReLu has faster training as compared to sigmoid, since no need to compute exponential.

g) L2-Regularization

```
Score for architecture = [50, 800, 500, 300, 2], lambda = 1e-07, decay = 0.0, momentum = 0.0,
    actfn = relu: 0.8135932028
Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-07, decay = 0.0, momentum = 0.0,
    actfn = relu: 0.808826357251
Score for architecture = [50, 800, 500, 300, 2], lambda = 1e-06, decay = 0.0, momentum = 0.0,
    actfn = relu: 0.801752972572
Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-06, decay = 0.0, momentum = 0.0,
    actfn = relu: 0.807557758782
Score for architecture = [50, 800, 500, 300, 2], lambda = 1e-05, decay = 0.0, momentum = 0.0,
    actfn = relu: 0.813055010413
Best Config: architecture = [50, 800, 500, 300, 2], lambda = 1e-07, decay = 0.0, momentum = 0.0,
    actfn = relu, best_acc = 0.8135932028
Best Regularization coefficient is:  1e-07
```

Trend: We can see that as the value of lambda increases, the accuracy decreases a bit then increases a bit. We can see that when we are increasing the lambda, we make the model more general. Making a model more general may lead to underfitting. So when we increase the regulatization term after a certain best fit lambda will start making the model too simple for it to give good accuracies. We can see that regularization has increased the accuracy on testing set with respect to the model that did not use regularization in the relu activation part.

h)Early Stopping and L2-regularization

```
Score for architecture = [50, 800, 500, 300, 2], lambda = 1e-07, decay = 0.0, momentum = 0.0,
    actfn = relu: 0.806519818121
Epoch 00008: early stopping
Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-07, decay = 0.0, momentum = 0.0,
    actfn = relu: 0.757159880709
Score for architecture = [50, 800, 500, 300, 2], lambda = 1e-06, decay = 0.0, momentum = 0.0,
    actfn = relu: 0.791104445219
Epoch 00007: early stopping
Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-06, decay = 0.0, momentum = 0.0,
    actfn = relu: 0.7205627933
Epoch 00008: early stopping
Score for architecture = [50, 800, 500, 300, 2], lambda = 1e-05, decay = 0.0, momentum = 0.0,
    actfn = relu: 0.768615696803
Best Config: architecture = [50, 800, 500, 300, 2], lambda = 1e-07, decay = 0.0, momentum = 0.0, \       actfn
Best Regularization coefficient with early stopping is:  1e-07
```

Trend: Early stopping does not help much in this case. Yes the best value of the regularization hyperparameter this time is same as the the one with only L2 regularization. Early stopping is decreasing the accuracy on testing set. Our model have the capability to go forward and produce high accuracy on testing set maybe because of regularization term but early stopping is leading to lower accuracies.

i) SGD with weight decay

```
Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-07, decay = 1e-05, momentum = 0.0,
    actfn = relu: 0.754276711459
Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-07, decay = 5e-05, momentum = 0.0,
    actfn = relu: 0.720409026011
Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-07, decay = 0.0001, momentum = 0.0,
    actfn = relu: 0.421212469396
Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-07, decay = 0.0003, momentum = 0.0,
    actfn = relu: 0.757044550338
Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-07, decay = 0.0007, momentum = 0.0,
    actfn = relu: 0.638411565044
Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-07, decay = 0.001, momentum = 0.0,
    actfn = relu: 0.711298197743
Best Config: architecture = [50, 800, 500, 300, 2], lambda = 5e-07, decay = 0.0003, momentum = 0.0,
    actfn = relu, best_acc = 0.757044550338
Best weight decay is:  0.0003
```

j) Momentum

```
Score for architecture = [50, 800, 500, 300, 2], lambda = 0.0, decay = 0.0003, momentum = 0.99,
    actfn = relu: 0.837158344663
Score for architecture = [50, 800, 500, 300, 2], lambda = 0.0, decay = 0.0003, momentum = 0.98,
    actfn = relu: 0.807980625702
Score for architecture = [50, 800, 500, 300, 2], lambda = 0.0, decay = 0.0003, momentum = 0.95,
    actfn = relu: 0.771152886867
Score for architecture = [50, 800, 500, 300, 2], lambda = 0.0, decay = 0.0003, momentum = 0.9,
    actfn = relu: 0.772536810803
Score for architecture = [50, 800, 500, 300, 2], lambda = 0.0, decay = 0.0003, momentum = 0.85,
    actfn = relu: 0.737208322357
Best Config: architecture = [50, 800, 500, 300, 2], lambda = 0.0, decay = 0.0003, momentum = 0.99,
    actfn = relu, best_acc = 0.837158344663
Best momentum is:  0.99
```

k) Combining the above

```
Score for architecture = [50, 800, 500, 300, 2], lambda = 1e-07, decay = 0.0003, momentum = 0.99,
    actfn = relu: 0.849113905434
Best Config: architecture = [50, 800, 500, 300, 2], lambda = 1e-07, decay = 0.0003, momentum = 0.99,
    actfn = relu, best_acc = 0.849113905434
```

trend:Accuracy is observed to be the best by far in this case. This is because we are combining all the best parameters(regularization coefficient, decay and momentum coefficient) values which we found in previous runs.This is by far the best accuracy.

l) Grid search with cross-validation

```
Best Config: architecture = [50, 800, 800, 500, 300, 2], lambda = 1e-07, decay = 1e-05, momentum = 0.99,
    actfn = relu, best_acc = 0.874524279976
```