1. **Boosting**

a. **Gradient Calculation**

We have the least square loss function $L(y_i, \widehat{y_i})$ given by

$$L(y_i, \widehat{y_i}) = (y_i - \widehat{y_i})^2$$

We have to calculate the gradient of the loss function w.r.t the current prediction , $\widehat{y_i}$ as

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

or

$$\boxed{g_i = -2(y_i - \widehat{y_i})}$$

b. **Weak Learner Selection**

Now, we choose the next learner to be the one that can best predict the gradients as,

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

We can show that the optimal value of the step size γ can be computed in the closed form in this step. Let us consider,

$$\gamma^* = argmin_{\gamma \epsilon \mathbb{R}} \left(\sum_{i=1}^{n} \left(-g_i - \gamma h(x_i)\right)^2 \right)$$

Substituting for g_i from above, we get

$$\gamma^* = argmin_{\gamma \epsilon \mathbb{R}} \left(\sum_{i=1}^{n} \left(2(y_i - \widehat{y_i}) - \gamma h(x_i)\right)^2 \right)$$

To get the optimal value of γ we can differentiate w.r.t γ and equate it to zero. Which will be,

$$\sum_{i=1}^{n} -2h(x_i)[2(y_i - \widehat{y_i}) - \gamma h(x_i)] = 0$$

$$\sum_{i=1}^{n} -4h(x_i)(y_i - \widehat{y_i}) + \sum_{i=1}^{n} 2\gamma h(x_i)^2 = 0$$

Or

$$\sum_{i=1}^{n} 2\gamma h(x_i)^2 = \sum_{i=1}^{n} 4h(x_i)(y_i - \widehat{y_i})$$

$$\boxed{\gamma = \frac{2\sum_{i=1}^{n} h(x_i)(y_i - \widehat{y_i})}{\sum_{i=1}^{n} h(x_i)^2}}$$

c. Step size selection

We then select the step size α^* that minimizes the loss using

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

Now, applying the loss function as $L(y_i, \widehat{y_i}) = (y_i - \widehat{y_i})^2$, we get

$$\alpha^* = argmin_{\alpha \epsilon \mathbb{R}} \sum_{i=1}^{n} \left(y_i - \left(\widehat{y_i} + \alpha\, h^*(x_i)\right)\right)^2$$

Differentiating w.r.t α and equating it to zero,

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

Or,

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

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

Therefore, we can calculate α^* in terms of $y_i\ , \widehat{y_i}\ , and\ h^*$ as

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

2. Neural Networks

a.

For a neural network with a single logistic output and with linear activation functions in the hidden layers, the prediction function can be written as a function of the weights w of the second layer and weights v of the first layer as follows:

$$y = \sigma\left(\sum_k w_k \left(\sum_i v_{ki}\, x_i\right)\right)$$

Now applying the sigmoid function as $\sigma(a) = \frac{1}{1+e^{-a}}$

$$y = \frac{1}{1 + e^{-\sum_k w_k (\sum_i v_{ki} x_i)}}$$

$$y = \frac{1}{1 + e^{-\sum_k \sum_i w_k v_{ki} x_i}}$$

$$y = \frac{1}{1 + e^{-\sum_k \sum_i w_k v_{ki} x_i}}$$

Or,

$$\boxed{y = \frac{1}{1 + e^{-\sum_i w_{i'}\, x_i}}}$$

Where $w_i' = \sum_k w_k v_{ki}$

From the above equation we can see that it's similar to logistic regression with different weights combination. Even with multiple hidden layers, the equation would be in similar format.

b.

We are given the following Neural network,

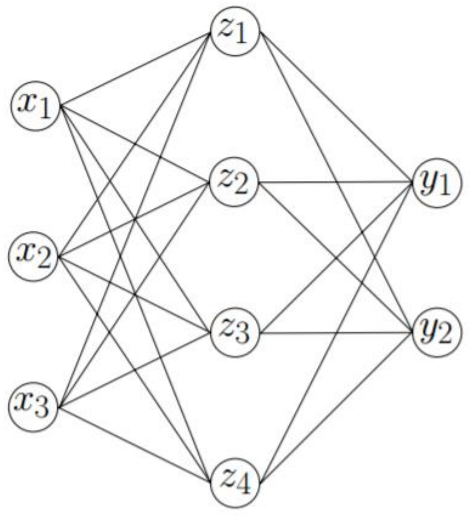


Figure 1: A neural network with one hidden layer

Where each hidden layer is defined as

$$z_k = \tanh(\sum_{i=1}^{3} w_{ki}x_i) \ \ for\ k = 1, 2, ..4$$

And the outputs are defined as

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

We are given the square loss function

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

Which can be represented as

$$L(y, \hat{y}) = \frac{1}{2}\sum_{j=1}^{2}\left(y_j - \widehat{y_J}\right)^2$$

Where, y_1 and $\widehat{y_1}$ represent the true outputs and our estimations, respectively.

Now, taking gradient with respect to v_{jk}:

$$\frac{\partial L}{\partial v_{jk}} = \frac{\partial L}{\partial b_j}\frac{\partial b_j}{\partial v_{jk}} \quad where, \quad b_j = \widehat{y_J} \ = \sum_{k} v_{jk}\, z_k$$

$$\frac{\partial L}{\partial v_{jk}} = \ -\left(y_j - \widehat{y_J}\right)z_k$$

And taking gradient with respect to w_{ki}:

$$\frac{\partial L}{\partial w_{ki}} = \frac{\partial L}{\partial a_k}\frac{\partial a_k}{\partial w_{ki}} \quad where, \quad a_k = \sum_{i} w_{ki}\, x_i$$

$$\frac{\partial L}{\partial w_{ki}} = \ \frac{\partial L}{\partial a_k}x_i$$

$$\frac{\partial L}{\partial w_{ki}} = \sum_{j}\frac{\partial L}{\partial b_j}\frac{\partial b_j}{\partial a_k}x_i \ = \ \sum_{j}\frac{\partial L}{\partial b_j}\frac{\partial b_j}{\partial z_k}\frac{\partial z_k}{\partial a_k}x_i$$

$$\frac{\partial L}{\partial w_{ki}} = \sum_{j} -\left(y_j - \widehat{y_J}\right) v_{jk}\,\frac{\partial(\tanh(a_k))}{\partial a_k}x_i$$

Therefore, finally, we get,

$$\boxed{\frac{\partial L}{\partial w_{ki}} = -\sum_{j} (y_j - \hat{y}_j) v_{jk} \, (1 - \tanh^2(\sum_{i} w_{ki} \, x_i)) x_i}$$

3. Programming

d. Linear activations

Linear Activations

Architecture 1

Score for architecture = [50, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = linear: 0.824626148377

Score for architecture = [50, 50, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = linear: 0.840733478172

Score for architecture = [50, 50, 50, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = linear: 0.843808865206

Score for architecture = [50, 50, 50, 50, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = linear: 0.845154342737

Best Config: architecture = [50, 50, 50, 50, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = linear, best_acc = 0.845154342737

Training time = 27.0568940639 s

Architecture 2

Score for architecture = [50, 50, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = linear: 0.838926689607

Score for architecture = [50, 500, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = linear: 0.841463881963

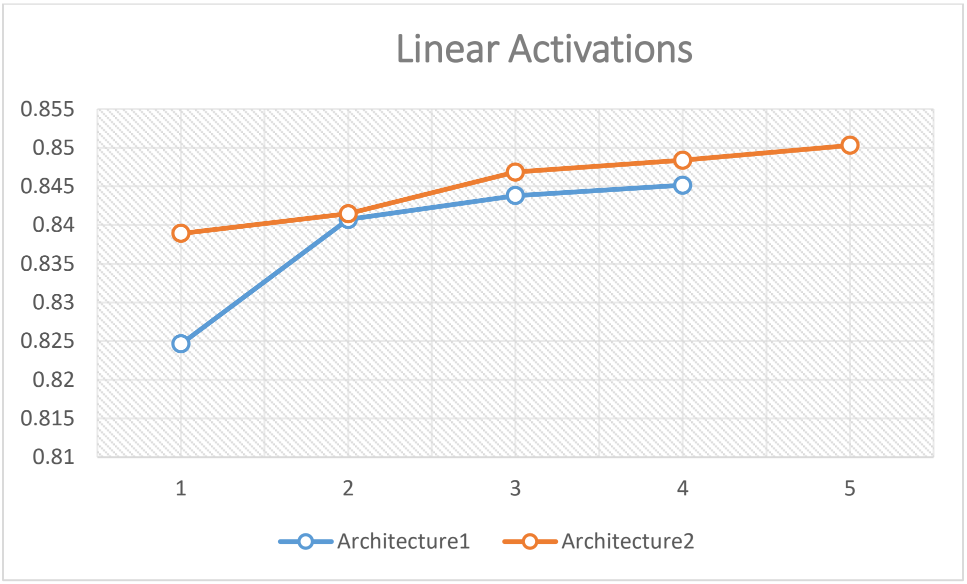
Score for architecture = [50, 500, 300, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = linear: 0.846845812709

Score for architecture = [50, 800, 500, 300, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = linear: 0.848383499352

Score for architecture = [50, 800, 800, 500, 300, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = linear: 0.85030561846

Best Config: architecture = [50, 800, 800, 500, 300, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = linear, best_acc = 0.85030561846

Training time = 253.143762827 s



Trends:

As can be seen in the graph, the accuracies increase with more complex architectures with linear activations. This is true for both Architecture 1 and Architecture 2.

It can be seen even between Architectures 1 and 2 that Architecture 2 (with higher complexity) performs better.

Though, more complex architectures also take a lot more time to train.

e. Sigmoid Activations

Sigmoid Activations

Score for architecture = [50, 50, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = sigmoid: 0.733594742122

Score for architecture = [50, 500, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = sigmoid: 0.761388537147

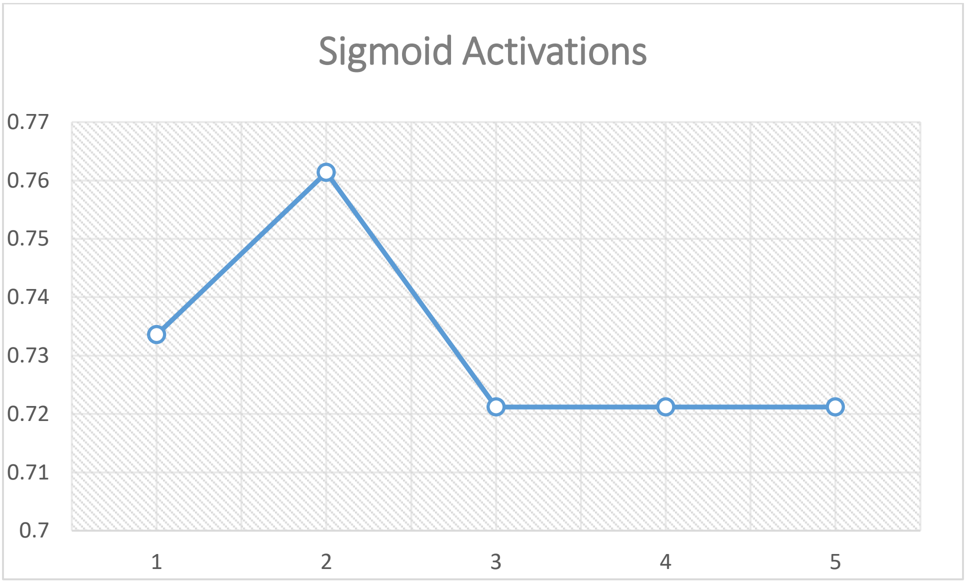
Score for architecture = [50, 500, 300, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = sigmoid: 0.721216312633

Score for architecture = [50, 800, 500, 300, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = sigmoid: 0.721216312633

Score for architecture = [50, 800, 800, 500, 300, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = sigmoid: 0.721216312633

Best Config: architecture = [50, 500, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = sigmoid, best_acc = 0.761388537147

Training time = 842.221850157 s



Trends:

Sigmoid activation does better with simpler architectures. As you can see from the graph, the accuracy increases initially and then starts to drop as the architectures get more complex. This is very useful for gradient based methods (like descent) as sigmoid is easily differentiable and has some good attributes in this regard. This may in a way explain the trend as we know that we observe similar trends in logistic regression and gradient methods.

Sigmoid also introduces some issues related to approximation and since it varies between 0 and 1 converges slower than functions varying between -1 and 1 like say tanH. And is generally not very effective especially for back propagation (its gradient is not monotonic) and can also suffer from gradient issues like vanishing and exploding gradients. And the issues get more prominent as the networks become deeper (as can also be seen in our trend)

Sigmoid takes more time since it takes longer time to model the more complex sigmoid function. In addition to the convergence issues that may be faced.

f. ReLu Activation

ReLu Activations

Score for architecture = [50, 50, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = relu: 0.825510319457

Score for architecture = [50, 500, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = relu: 0.819244214847

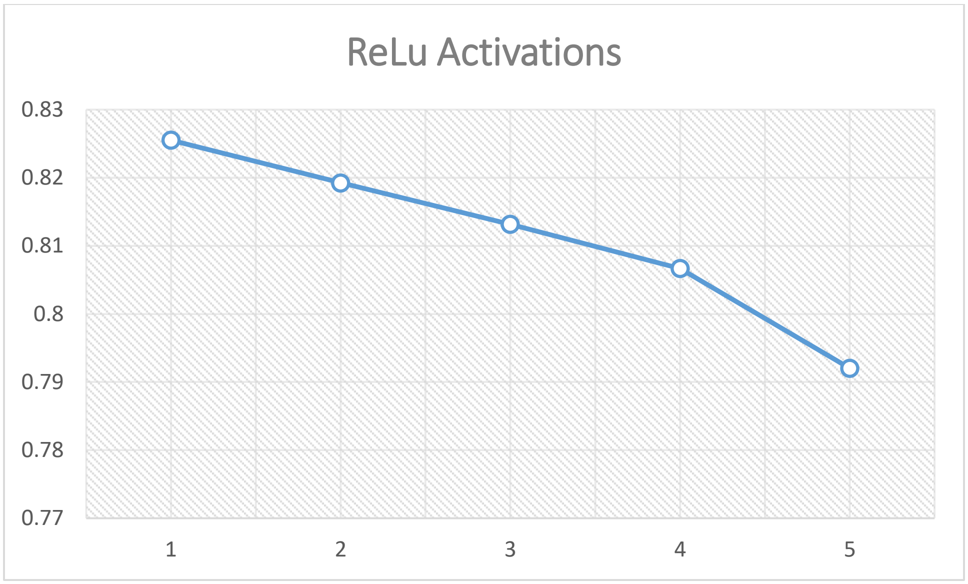
Score for architecture = [50, 500, 300, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = relu: 0.813170340955

Score for architecture = [50, 800, 500, 300, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = relu: 0.806673585903

Score for architecture = [50, 800, 800, 500, 300, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = relu: 0.792027065981

Best Config: architecture = [50, 50, 2], lambda = 0.0, decay = 0.0, momentum = 0.0, actfn = relu, best_acc = 0.825510319457

Training time = 399.599398851 s



Trend:

In a randomly initialized network, only about 50% of hidden units are activated (having a non-zero output). And ReLu considers the sparsity of data and generally performs better.

ReLu's biggest advantage is that it doesn't change the gradient as it passes back during back-propagation, because the derivative of the function is 1 for x > 0. This is especially important for deep networks, which would otherwise suffer from the vanishing gradient problem. ReLu is a little strange in that it's not differentiable at 0, and does not have upper saturation, so some extra care must be taken when using it else it may blow up the activation.

Also, ReLu is different from linear activation in the sense that we want to separate multiple successive linear transformation by nonlinearity, otherwise they will collapse to a single linear transformation.

ReLu runs faster than sigmoid (as we discussed earlier, it is better in backpropagation owing from a more uniform gradient). Linear is faster than ReLu.

g. L2 regularization

ReLu Activations with L2 Regularization

Score for architecture = [50, 800, 500, 300, 2], lambda = 1e-07, decay = 0.0, momentum = 0.0, actfn = relu: 0.800830362454

Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-07, decay = 0.0, momentum = 0.0, actfn = relu: 0.797985624063

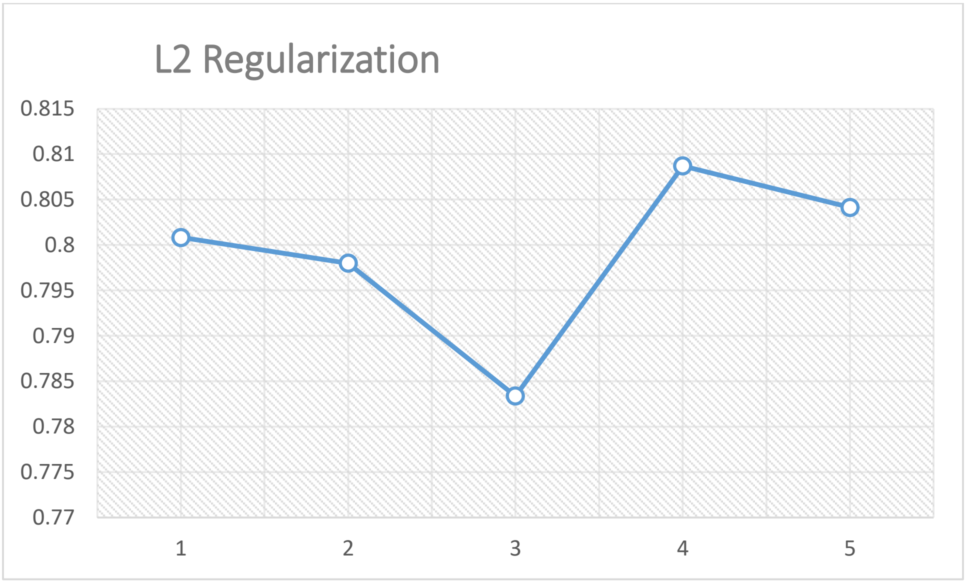
Score for architecture = [50, 800, 500, 300, 2], lambda = 1e-06, decay = 0.0, momentum = 0.0, actfn = relu: 0.78337754138

Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-06, decay = 0.0, momentum = 0.0, actfn = relu: 0.808711027694

Score for architecture = [50, 800, 500, 300, 2], lambda = 1e-05, decay = 0.0, momentum = 0.0, actfn = relu: 0.804136395839

Best Config: architecture = [50, 800, 500, 300, 2], lambda = 5e-06, decay = 0.0, momentum = 0.0, actfn = relu, best_acc = 0.808711027694

Training time = 630.186666965 s



Trend:

Regularization prevents over fitting. Its effects are more prominent as the network gets more complicated and the accuracies get better. However, eventually an increase in bias tends to lower accuracies.

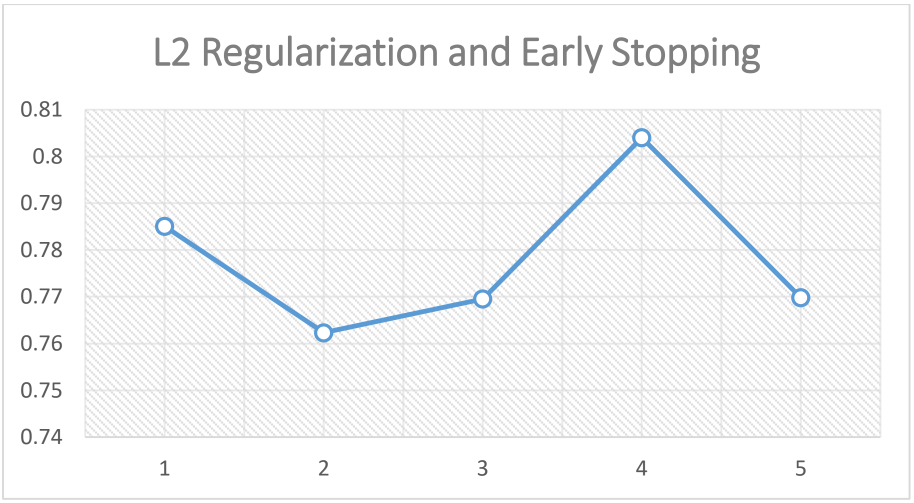
h. Early stopping and L2 regularization

ReLu Activations with L2 Regularization and Early Stopping

```
-----------------------------------------------------------

Score for architecture = [50, 800, 500, 300, 2], lambda = 1e-07, decay = 0.0, momentum = 0.0, actfn = relu: 0.785069004477

Epoch 00007: early stopping

Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-07, decay = 0.0, momentum = 0.0, actfn = relu: 0.762311152341

Epoch 00008: early stopping

Score for architecture = [50, 800, 500, 300, 2], lambda = 1e-06, decay = 0.0, momentum = 0.0, actfn = relu: 0.769538304309

Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-06, decay = 0.0, momentum = 0.0, actfn = relu: 0.804021070372

Epoch 00008: early stopping

Score for architecture = [50, 800, 500, 300, 2], lambda = 1e-05, decay = 0.0, momentum = 0.0, actfn = relu: 0.769807401649
```

Best Config: architecture = [50, 800, 500, 300, 2], lambda = 5e-06, decay = 0.0, momentum = 0.0, actfn = relu, best_acc = 0.804021070372

Training time = 339.632090092 s



Trend:

While it functions very similar to the L2 regularization, the key difference being that early stopping is faster as the convergence is faster. But the possible trade off being that it leads to poorer accuracies. Since we don't have a complete view and stop when the errors start increasing, this may well be a local trend as opposed to a global trend. It tends to only get worse when the networks become more complex. The effects are less prominent when training less complex networks.

i. SGD with weight decay

SGD with weight decay

Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-07, decay = 1e-05, momentum = 0.0, actfn = relu: 0.754507359289

Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-07, decay = 5e-05, momentum = 0.0, actfn = relu: 0.721216312633

Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-07, decay = 0.0001, momentum = 0.0, actfn = relu: 0.721216312633

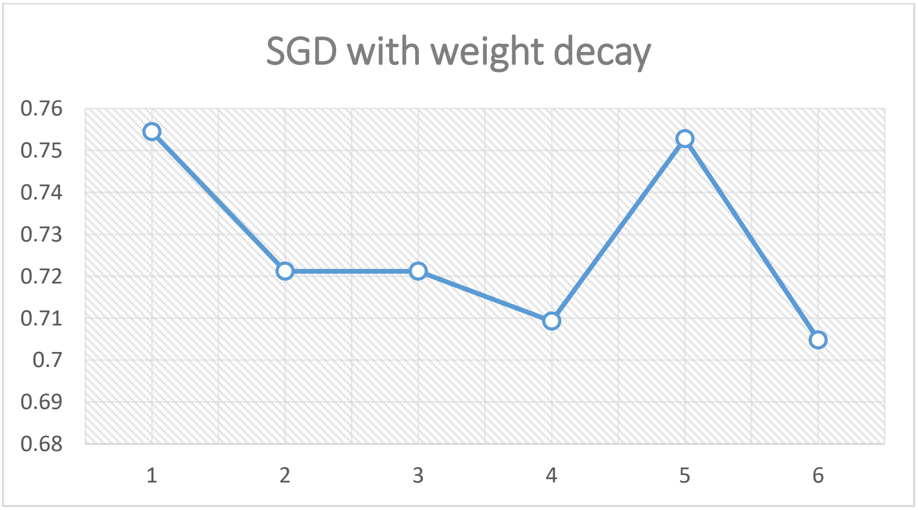
Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-07, decay = 0.0003, momentum = 0.0, actfn = relu: 0.709299195976

Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-07, decay = 0.0007, momentum = 0.0, actfn = relu: 0.752815895698

Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-07, decay = 0.001, momentum = 0.0, actfn = relu: 0.704839887297

Best Config: architecture = [50, 800, 500, 300, 2], lambda = 5e-07, decay = 1e-05, momentum = 0.0, actfn = relu, best_acc = 0.754507359289

Training time = 2451.47704887 s



j. Momentum

Momentum

Score for architecture = [50, 800, 500, 300, 2], lambda = 0.0, decay = 1e-05, momentum = 0.99, actfn = relu: 0.858416945598

Score for architecture = [50, 800, 500, 300, 2], lambda = 0.0, decay = 1e-05, momentum = 0.98, actfn = relu: 0.822511825233

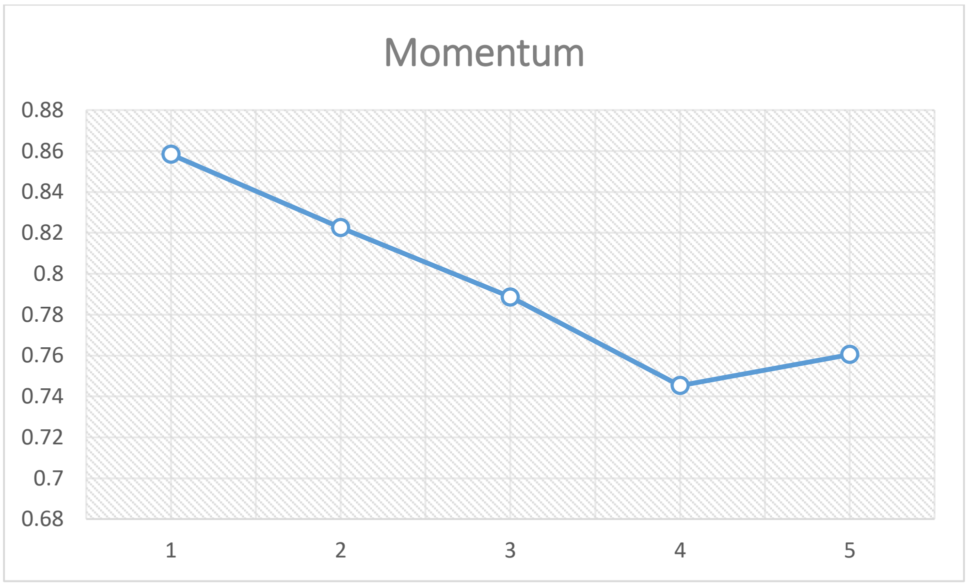
Score for architecture = [50, 800, 500, 300, 2], lambda = 0.0, decay = 1e-05, momentum = 0.95, actfn = relu: 0.788644137494

Score for architecture = [50, 800, 500, 300, 2], lambda = 0.0, decay = 1e-05, momentum = 0.9, actfn = relu: 0.745319651466

Score for architecture = [50, 800, 500, 300, 2], lambda = 0.0, decay = 1e-05, momentum = 0.85, actfn = relu: 0.76054280789

Best Config: architecture = [50, 800, 500, 300, 2], lambda = 0.0, decay = 1e-05, momentum = 0.99, actfn = relu, best_acc = 0.858416945598

Training time = 970.961313009 s



k. Combining the above

Combining the above

Epoch 00008: early stopping

Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-06, decay = 1e-05, momentum = 0.99, actfn = relu: 0.772844344569

Best Config: architecture = [50, 800, 500, 300, 2], lambda = 5e-06, decay = 1e-05, momentum = 0.99, actfn = relu, best_acc = 0.772844344569

Training time = 35.4904181957 s

Trend:

Combining the best parameters doesn't necessarily guarantee the best accuracy as can be seen from the above results. Also, the parameters are tuned assuming independence which may not necessarily be true. The best value of each parameter doesn't necessarily translate to the best model. Also since the data is picked randomly, it may lead to some uncertainty in predictions.

I. **Grid search with cross validation**

Grid search with cross validation

Score for architecture = [50, 50, 2], lambda = 1e-07, decay = 1e-05, momentum = 0.99, actfn = relu: 0.846768919899

Score for architecture = [50, 50, 2], lambda = 1e-07, decay = 5e-05, momentum = 0.99, actfn = relu: 0.835389998413

Score for architecture = [50, 50, 2], lambda = 1e-07, decay = 0.0001, momentum = 0.99, actfn = relu: 0.842078957995

Score for architecture = [50, 50, 2], lambda = 5e-07, decay = 1e-05, momentum = 0.99, actfn = relu: 0.853035021726

Score for architecture = [50, 50, 2], lambda = 5e-07, decay = 5e-05, momentum = 0.99, actfn = relu: 0.847422440045

Score for architecture = [50, 50, 2], lambda = 5e-07, decay = 0.0001, momentum = 0.99, actfn = relu: 0.839311110122

Score for architecture = [50, 50, 2], lambda = 1e-06, decay = 1e-05, momentum = 0.99, actfn = relu: 0.845577209657

Score for architecture = [50, 50, 2], lambda = 1e-06, decay = 5e-05, momentum = 0.99, actfn = relu: 0.838234725348

Score for architecture = [50, 50, 2], lambda = 1e-06, decay = 0.0001, momentum = 0.99, actfn = relu: 0.84519278456

Epoch 00014: early stopping

```
Score for architecture = [50, 50, 2], lambda = 5e-06, decay = 1e-05, momentum = 0.99, actfn = relu:
0.816630144414

Score for architecture = [50, 50, 2], lambda = 5e-06, decay = 5e-05, momentum = 0.99, actfn = relu:
0.842309606638

Score for architecture = [50, 50, 2], lambda = 5e-06, decay = 0.0001, momentum = 0.99, actfn = relu:
0.842655592204

Score for architecture = [50, 50, 2], lambda = 1e-05, decay = 1e-05, momentum = 0.99, actfn = relu:
0.850536268901

Score for architecture = [50, 50, 2], lambda = 1e-05, decay = 5e-05, momentum = 0.99, actfn = relu:
0.844769915348

Score for architecture = [50, 50, 2], lambda = 1e-05, decay = 0.0001, momentum = 0.99, actfn = relu:
0.842194285753

Score for architecture = [50, 500, 2], lambda = 1e-07, decay = 1e-05, momentum = 0.99, actfn = relu:
0.853073461257

Epoch 00011: early stopping

Score for architecture = [50, 500, 2], lambda = 1e-07, decay = 5e-05, momentum = 0.99, actfn = relu:
0.812324607115

Score for architecture = [50, 500, 2], lambda = 1e-07, decay = 0.0001, momentum = 0.99, actfn = relu:
0.84492368722

Score for architecture = [50, 500, 2], lambda = 5e-07, decay = 1e-05, momentum = 0.99, actfn = relu:
0.851574218727

Epoch 00010: early stopping

Score for architecture = [50, 500, 2], lambda = 5e-07, decay = 5e-05, momentum = 0.99, actfn = relu:
0.815438441047

Score for architecture = [50, 500, 2], lambda = 5e-07, decay = 0.0001, momentum = 0.99, actfn = relu:
0.845500323721

Score for architecture = [50, 500, 2], lambda = 1e-06, decay = 1e-05, momentum = 0.99, actfn = relu:
0.851036024049

Score for architecture = [50, 500, 2], lambda = 1e-06, decay = 5e-05, momentum = 0.99, actfn = relu:
0.849075468194

Score for architecture = [50, 500, 2], lambda = 1e-06, decay = 0.0001, momentum = 0.99, actfn = relu:
0.846730482659

Score for architecture = [50, 500, 2], lambda = 5e-06, decay = 1e-05, momentum = 0.99, actfn = relu:
0.850536271193
```

```
Score for architecture = [50, 500, 2], lambda = 5e-06, decay = 5e-05, momentum = 0.99, actfn = relu:
0.849652090947

Score for architecture = [50, 500, 2], lambda = 5e-06, decay = 0.0001, momentum = 0.99, actfn = relu:
0.846230729803

Score for architecture = [50, 500, 2], lambda = 1e-05, decay = 1e-05, momentum = 0.99, actfn = relu:
0.853496325885

Score for architecture = [50, 500, 2], lambda = 1e-05, decay = 5e-05, momentum = 0.99, actfn = relu:
0.849921195161

Score for architecture = [50, 500, 2], lambda = 1e-05, decay = 0.0001, momentum = 0.99, actfn = relu:
0.848460385288

Score for architecture = [50, 500, 300, 2], lambda = 1e-07, decay = 1e-05, momentum = 0.99, actfn =
relu: 0.861799869995

Epoch 00009: early stopping

Score for architecture = [50, 500, 300, 2], lambda = 1e-07, decay = 5e-05, momentum = 0.99, actfn =
relu: 0.799215778418

Epoch 00008: early stopping

Score for architecture = [50, 500, 300, 2], lambda = 1e-07, decay = 0.0001, momentum = 0.99, actfn =
relu: 0.788221275156

Score for architecture = [50, 500, 300, 2], lambda = 5e-07, decay = 1e-05, momentum = 0.99, actfn =
relu: 0.861069463913

Score for architecture = [50, 500, 300, 2], lambda = 5e-07, decay = 5e-05, momentum = 0.99, actfn =
relu: 0.859454891334

Score for architecture = [50, 500, 300, 2], lambda = 5e-07, decay = 0.0001, momentum = 0.99, actfn =
relu: 0.853304119065

Score for architecture = [50, 500, 300, 2], lambda = 1e-06, decay = 1e-05, momentum = 0.99, actfn =
relu: 0.86145388672

Score for architecture = [50, 500, 300, 2], lambda = 1e-06, decay = 5e-05, momentum = 0.99, actfn =
relu: 0.85714834942

Score for architecture = [50, 500, 300, 2], lambda = 1e-06, decay = 0.0001, momentum = 0.99, actfn =
relu: 0.856071962354

Score for architecture = [50, 500, 300, 2], lambda = 5e-06, decay = 1e-05, momentum = 0.99, actfn =
relu: 0.862030523221

Epoch 00008: early stopping
```

```
Score for architecture = [50, 500, 300, 2], lambda = 5e-06, decay = 5e-05, momentum = 0.99, actfn =
relu: 0.797447429384

Score for architecture = [50, 500, 300, 2], lambda = 5e-06, decay = 0.0001, momentum = 0.99, actfn =
relu: 0.85499557758

Score for architecture = [50, 500, 300, 2], lambda = 1e-05, decay = 1e-05, momentum = 0.99, actfn =
relu: 0.860723480638

Score for architecture = [50, 500, 300, 2], lambda = 1e-05, decay = 5e-05, momentum = 0.99, actfn =
relu: 0.854764919772

Score for architecture = [50, 500, 300, 2], lambda = 1e-05, decay = 0.0001, momentum = 0.99, actfn =
relu: 0.85180486737

Score for architecture = [50, 800, 500, 300, 2], lambda = 1e-07, decay = 1e-05, momentum = 0.99, actfn
= relu: 0.869103910191

Score for architecture = [50, 800, 500, 300, 2], lambda = 1e-07, decay = 5e-05, momentum = 0.99, actfn
= relu: 0.864990577913

Score for architecture = [50, 800, 500, 300, 2], lambda = 1e-07, decay = 0.0001, momentum = 0.99, actfn
= relu: 0.858609152419

Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-07, decay = 1e-05, momentum = 0.99, actfn
= relu: 0.871102907868

Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-07, decay = 5e-05, momentum = 0.99, actfn
= relu: 0.865951644095

Epoch 00008: early stopping

Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-07, decay = 0.0001, momentum = 0.99, actfn
= relu: 0.771921738541

Score for architecture = [50, 800, 500, 300, 2], lambda = 1e-06, decay = 1e-05, momentum = 0.99, actfn
= relu: 0.870872254642

Score for architecture = [50, 800, 500, 300, 2], lambda = 1e-06, decay = 5e-05, momentum = 0.99, actfn
= relu: 0.862030518638

Epoch 00008: early stopping

Score for architecture = [50, 800, 500, 300, 2], lambda = 1e-06, decay = 0.0001, momentum = 0.99, actfn
= relu: 0.764348598714

Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-06, decay = 1e-05, momentum = 0.99, actfn
= relu: 0.872640603676

Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-06, decay = 5e-05, momentum = 0.99, actfn
= relu: 0.867681549015
```

```
Score for architecture = [50, 800, 500, 300, 2], lambda = 5e-06, decay = 0.0001, momentum = 0.99, actfn
= relu: 0.862799364251

Score for architecture = [50, 800, 500, 300, 2], lambda = 1e-05, decay = 1e-05, momentum = 0.99, actfn
= relu: 0.871295123854

Score for architecture = [50, 800, 500, 300, 2], lambda = 1e-05, decay = 5e-05, momentum = 0.99, actfn
= relu: 0.861453882137

Score for architecture = [50, 800, 500, 300, 2], lambda = 1e-05, decay = 0.0001, momentum = 0.99, actfn
= relu: 0.862030518638

Score for architecture = [50, 800, 800, 500, 300, 2], lambda = 1e-07, decay = 1e-05, momentum = 0.99,
actfn = relu: 0.875792876646

Epoch 00008: early stopping

Score for architecture = [50, 800, 800, 500, 300, 2], lambda = 1e-07, decay = 5e-05, momentum = 0.99,
actfn = relu: 0.733133428797

Epoch 00007: early stopping

Score for architecture = [50, 800, 800, 500, 300, 2], lambda = 1e-07, decay = 0.0001, momentum = 0.99,
actfn = relu: 0.73251836193

Epoch 00008: early stopping

Score for architecture = [50, 800, 800, 500, 300, 2], lambda = 5e-07, decay = 1e-05, momentum = 0.99,
actfn = relu: 0.743282207384

Epoch 00008: early stopping

Score for architecture = [50, 800, 800, 500, 300, 2], lambda = 5e-07, decay = 5e-05, momentum = 0.99,
actfn = relu: 0.753853846017

Score for architecture = [50, 800, 800, 500, 300, 2], lambda = 5e-07, decay = 0.0001, momentum = 0.99,
actfn = relu: 0.865298123949

Epoch 00007: early stopping

Score for architecture = [50, 800, 800, 500, 300, 2], lambda = 1e-06, decay = 1e-05, momentum = 0.99,
actfn = relu: 0.733325644783

Score for architecture = [50, 800, 800, 500, 300, 2], lambda = 1e-06, decay = 5e-05, momentum = 0.99,
actfn = relu: 0.870257185484

Score for architecture = [50, 800, 800, 500, 300, 2], lambda = 1e-06, decay = 0.0001, momentum = 0.99,
actfn = relu: 0.868834815143

Epoch 00007: early stopping

Score for architecture = [50, 800, 800, 500, 300, 2], lambda = 5e-06, decay = 1e-05, momentum = 0.99,
actfn = relu: 0.751777964695
```

Score for architecture = [50, 800, 800, 500, 300, 2], lambda = 5e-06, decay = 5e-05, momentum = 0.99, actfn = relu: 0.870641601417

Epoch 00007: early stopping

Score for architecture = [50, 800, 800, 500, 300, 2], lambda = 5e-06, decay = 0.0001, momentum = 0.99, actfn = relu: 0.737861841689

Score for architecture = [50, 800, 800, 500, 300, 2], lambda = 1e-05, decay = 1e-05, momentum = 0.99, actfn = relu: 0.873332563353

Epoch 00007: early stopping

Score for architecture = [50, 800, 800, 500, 300, 2], lambda = 1e-05, decay = 5e-05, momentum = 0.99, actfn = relu: 0.729366086669

Epoch 00008: early stopping

Score for architecture = [50, 800, 800, 500, 300, 2], lambda = 1e-05, decay = 0.0001, momentum = 0.99, actfn = relu: 0.736862338268

Best Config: architecture = [50, 800, 800, 500, 300, 2], lambda = 1e-07, decay = 1e-05, momentum = 0.99, actfn = relu, best_acc = 0.875792876646

Training time = 13234.1427898 s

This is an exhaustive approach. And like other similar (brute force) search, tends to give good accuracy. And the obvious downside being the time taken to train.

Collaboration:

Collaborated on thoughts and ideas with **Adarsha Desai** and **Mahesh Pottippala Subrahmanya**