MP1 - CS291K (SHARATH RAO)(9972332)

Code Snippets

Zero centering images

```
mean_image = np.mean(X_train, axis=0)
X_train -= mean_image
X_val -= mean_image
X_test -= mean_image
```

This prevented high exponential values and also increased accuracy in general. The below is the code for Forward and Back Propagation.

Forward Propagation

```
Z1 = X.dot(W1) + b1
A1 = self.Relu(Z1)
Z2 = A1.dot(W2) + b2
```

Loss

```
softmax_numerator = np.exp(Z2)
softmax_denominator = np.sum(softmax_numerator, axis=1)
data_loss= np.sum(-Z2[range(N), y] + np.log(softmax_denominator)) / N
regularization_loss = self.calculate_L2_regularization(W1, W2, reg)
loss = data_loss + regularization_loss
```

Back propagation

```
d_A2=1.0
d_Z2=(softmax_numerator.T/softmax_denominator).T
ground_truth=self.get_ground_truth(N,d_Z2,y)
d_Z2=((d_Z2-ground_truth)/float(N))*d_A2

d_A1=d_Z2.dot(W2.T)
d_Z1=d_A1*self.derivative_relu(Z1)

dW1=X.T.dot(d_Z1)
dW2=A1.T.dot(d_Z2)
db1=np.sum(d_Z1,axis=0)
db2=np.sum(d_Z2,axis=0)

dW1+=reg*W1
dW2+=reg*W2
```

Training Neural Network

First step was to make sure the network is learning correctly. Therefore, I set the parameters learning rate =0.01, reg = 0.5, hidden neurons = 5000, batch size =1000, iterations =100 to these values and checked the correctness of the network. I chose these values based on

previous experience with neural networks. The weights were initialized using the standard random process available in python using the below. The bias values were set to 0.0.

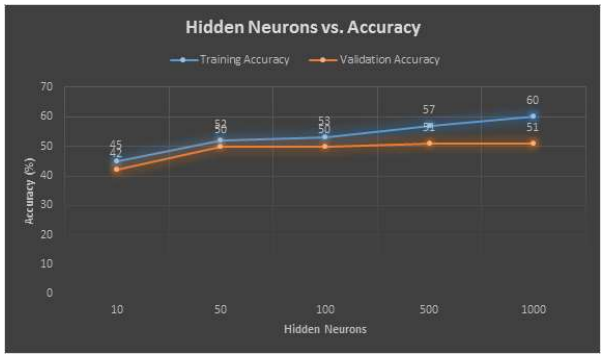
```
self.params['W1'] = std * np.random.randn(input_size, hidden_size)
self.params['b1'] = np.zeros(hidden_size)
```

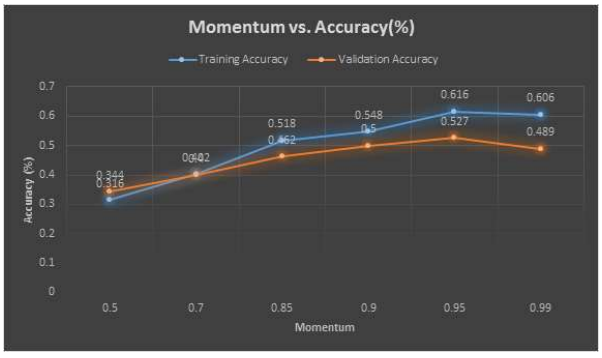
It turns out, the loss was fluctuating very badly and at one point, it gave a negative result. The problem was that the image values were very high and not normalized initially. After fixing that the neural network was ready for tuning.

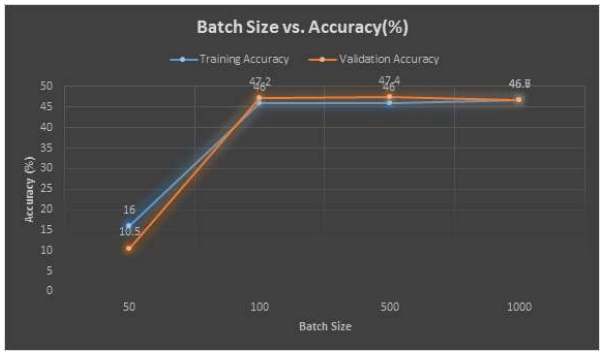
Hyperparameters

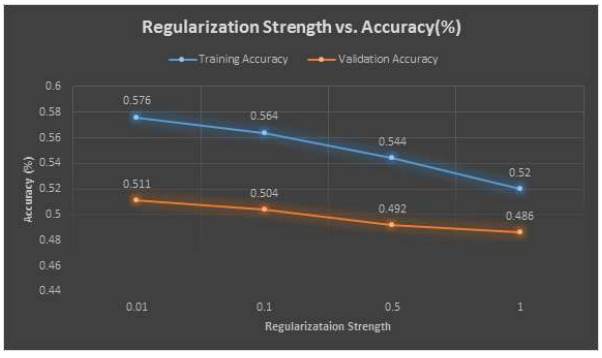
I wrote a file hyperparameter.py, which takes a list of parameter values and runs the neural network. Both the validation accuracy and training accuracy was captured. The below tables show the accuracy values for six parameters (see table after that)

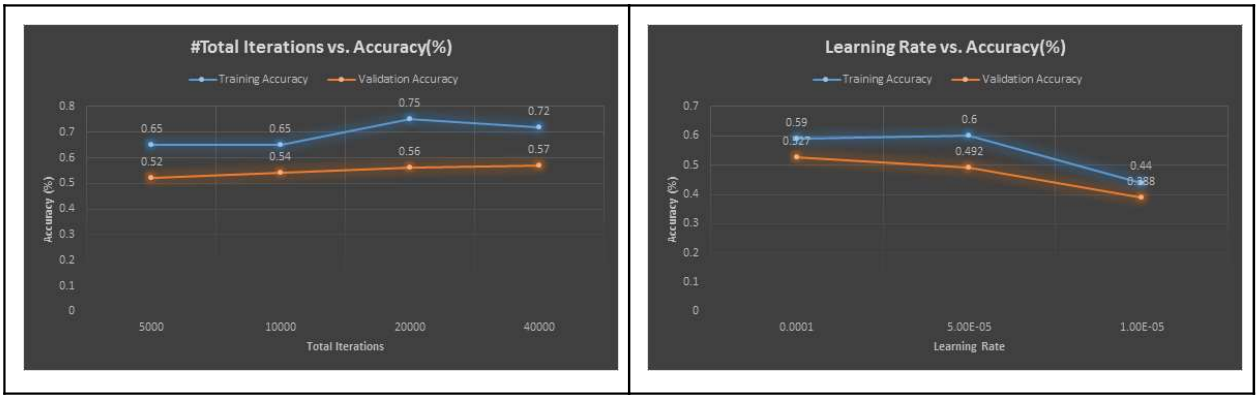
Default parameters- Learning rate =0.0001, Reg =1.0, Momentum-0.9, Batch size = 100, Hidden Neurons = 500, Iterations = 5000









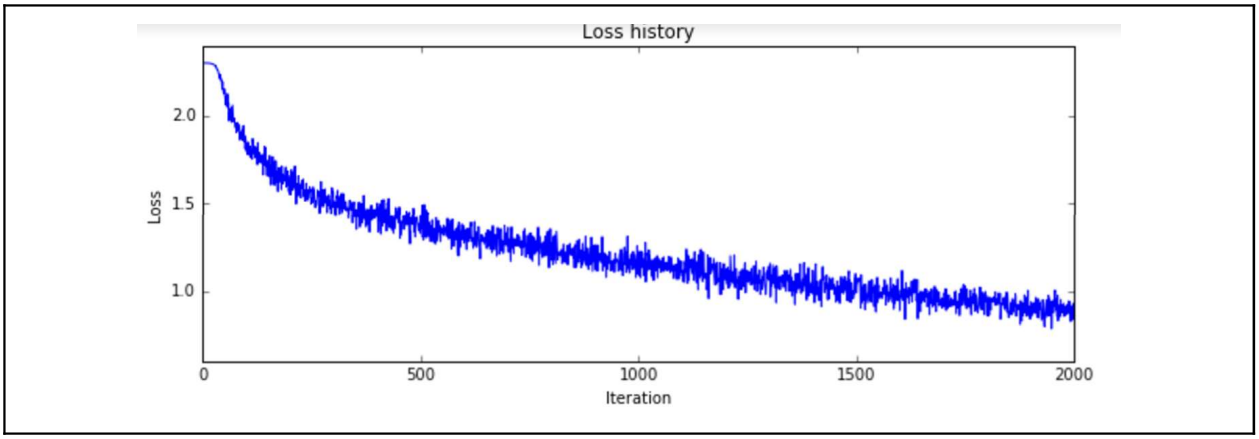


Increasing hidden units helped up to 500 neurons and the val acc converged after -Fig1. Increasing Momentum increased val acc up until momentum became 0.95.For 0.99 the acc fell(Fig 2). Batch size increase up to 500 increases accuracy and converges after (Fig3). Increasing the regularization decreased the training and validation accuracy of the network. My understanding was that there was less overfitting for the iterations I ran. Therefore reg=0.01 was enough(Fig 4). Increasing the iterations up to 20K increases the val acc. Converges after that(Fig5). Decreasing learning rate decreased accuracy too (as the iterations was constant). As there was no jump at 0.0001, I finalized this value.

Best Parameters

Parameter	Value
Momentum	0.95
Learning Rate	0.0001
Batch Size	500
#Iterations	20000
Regularization Strength	0.01
#Hidden Neurons	500

Loss function versus training steps



Final Results

Using the best parameters the following are the results

Training Time (sec)	4627.43s
Training Accuracy	74.4%
Validation Accuracy	53.5%
Test Accuracy (Top 1)	54.4%
Test Accuracy (Top 2)	73.66%
Test Accuracy (Top 3)	83.16%

Extra Credits

Momentum

```
self.old_weights['W1'] = self.momentum * self.old_weights['W1'] -
learning_rate * grads['W1']
W1+= self.old_weights['W1']
```

Momentum Impact (with best parameters)

	Without Momentum	**With Momentum**
Validation Accuracy	20.0%	**40.0%**
Training Accuracy	26.0%	**50.4%**
Training Time	170.11s	**173.88s**

Dropout

Dropout is not needed here. In fact, the results show it is counter productive for this use case.

Dropout Impact (with best parameters)

	Without Dropout	**With** Dropout
Validation Accuracy	**50.0%**	40.0%
Training Accuracy	**69.6%**	52.2%
Training Time	**896.69s**	797.38s

Different Activations

	Leaky Relu	**Relu**	Tanh
Validation Accuracy	30.0%	**40.0%**	40.0%
Training Accuracy	43.0%	**56.4%**	44.8%
Training Time	615.92s	**170.95s**	212.99s

Did not use sigmoid as the training time would increase and also it is proven to be not helpful.

Different Weight Initializations

- Dividing the random weights by square root of their sizes.

	Before	After
Validation Accuracy	47.9%	**49.8%**
Training Accuracy	51.2%	**50.8%**
Training Time	149.2s	**136.32s**

Challenges

- I had to deal with large values and fluctuating loss values initially. Data normalization helped mitigate this problem.
- Understanding Back propagation took time.
- Finding the first set of parameter values was difficult

Possible Improvements

- More hyper params with Dropout
- Data Augmentation techniques
- PCA and Whitening