private note @145 2 views

HW6_2017310936_Md_Shirajum_Munir

I) **Theory:** Derive the following parts:

1. The backpropagation algorithm

(I) (1) Back propagation:

Let, z_j: input to node j for layer l

g_j : activation function for node j in layer l

$a_j = g_j(z_j)$: output / activation of node j in layer l.

b_j : bias for unit j in layer l

w_{ij} : weights connecting node i in layer $(l-1)$ to node j in layer l.

t_k : target value for node k in the output layer.

Gradients for output layer weights:

Let output layer connection weight, w_{jk}

$$\frac{\partial E}{\partial w_{jk}} = \frac{1}{2}\sum_{k\in K}(a_k - t_k)^2$$

$$= (a_k - t_k)\frac{\partial}{\partial w_{jk}}(a_k - t_k)$$

$$= (a_k - t_k)\frac{\partial}{\partial w_{jk}} g_k(z_k)$$

$$= (a_k - t_k)\, g_k'(z_k)\frac{\partial}{\partial w_{jk}} z_k,$$

when again ; $z_k = b_j + \sum_j g_j(z_j) w_{jk}$ and

- thus $\frac{\partial z_k}{\partial w_{jk}} = g_j(z_j) = a_j$

$$\frac{\partial E}{\partial w_{jk}} = (a_k - t_k)\, g_k'(z_k)\, a_j$$

The gradient of the error function with respect to the output layer weights is a product of three terms. The first term is the difference between the network output and the target value t_k. The second term is the derivative of output layer activation function. And the third term is the activation output of node j in the hidden layer.

Let define δ_k to be the all terms that involve index k:

$$\delta_k = (a_k - t_k)\, g_k'(z_k)$$

$$\frac{\partial E}{\partial w_{jk}} = \delta_k a_j$$

Now the update is, $w_{jk} \leftarrow w_{jk} - \eta \frac{\partial E}{\partial w_{jk}}$

η is the learning rate.

Output layer biases, b_k.

$$\frac{\partial E}{\partial b_k} = (a_k - t_k)\, g_k'(z_k)$$
$$= \delta_k$$

I (1) Back Propagation

Gradients for hidden layer:

$$\frac{\partial E}{\partial w_{ij}} = \frac{1}{2}\sum_{k\in K}(a_k - t_k)^2$$

$$= \sum_{k\in K}(a_k - t_k)\frac{\partial}{\partial w_{ij}}a_k$$

$$= \sum_{k\in K}(a_k - t_k)\frac{\partial}{\partial w_{ij}}g_k(z_k)$$

$$= \sum_{k\in K}(a_k - t_k)\, g'_k(z_k)\frac{\partial}{\partial w_{ij}}z_k \quad \cdots (1)$$

$$z_k = b_k + \sum_j a_j w_{jk}$$

$$= b_k + \sum_j g_j(z_j)\, w_{jk}$$

$$= b_k + \sum_j g_j\left(b_j + \sum_i z_i w_{ij}\right) w_{jk} \quad \cdots (11)$$

Now,

$$\frac{\partial z_k}{\partial w_{ij}} = \frac{\partial z_k}{\partial a_j}\cdot\frac{\partial a_j}{\partial w_{ij}}$$

$$= \frac{\partial}{\partial a_j} a_j\, w_{jk}\frac{\partial a_j}{\partial w_{ij}}$$

$$= w_{jk}\frac{\partial a_j}{\partial w_{ij}}$$

$$= w_{jk}\, g'_j(z_j)\frac{\partial z_j}{\partial w_{ij}}$$

$$= w_{jk}\, g'_j(z_j)\frac{\partial}{\partial w_{ij}}\left(b_i + \sum_i a_i w_{ij}\right)$$

$$= w_{jk}\, g_j(z_j)\, a_i \quad \cdots (111)$$

Now plugging eq.(iii) into z_k in eqn(i).

$$\frac{\partial E}{\partial w_{ij}} = \sum_{k \in K} (a_k - t_k)\, g_k'(z_k)\, w_{jk}\, g_j'(z_j)\, a_i$$

$$= g_j'(z_j)\, a_i \sum_{k \in K} (a_k - t_k)\, g_k'(z_k)\, w_{jk}$$

$$= a_i\, g_j'(z_j) \sum_{k \in K} \delta_k w_{jk} \quad \cdots (iv)$$

Now from eq.(iv)

$$\boxed{\begin{aligned} \frac{\partial E}{\partial w_{ij}} &= a_i\, g_j'(z_j) \sum_{k \in K} \delta_k w_{jk} \\ &= \delta_j\, a_i \\ \text{where } \delta_j &= g_j'(z_j) \sum_{k \in K} \delta_k w_{jk} \end{aligned}}$$

Output Layer biases:

$$\frac{\partial z_k}{\partial b_i} = w_{jk}\, g_j'(z_j) \frac{\partial z_j}{\partial b_i}$$

$$= w_{jk}\, g_j'(z_j) \frac{\partial}{\partial b_i}\left(b_i + \sum_i a_i w_{ij}\right)$$

$$\boxed{\begin{aligned} \frac{\partial z_k}{\partial b_i} &= w_{jk}\, g_j'(z_j) \\ \text{where, } \frac{\partial E}{\partial b_i} &= g_j'(z_j) \sum_{k \in K} \delta_k w_{jk} \\ &= \delta_j \end{aligned}}$$

(1)(2) The derivative of sigmoid function:

Sigmoid function:

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

$$g'(z) = \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right)$$

$$= \frac{e^{-z}}{(1+e^{-z})^2}$$

$$= \frac{1+e^{-z}-1}{(1+e^{-z})^2}$$

$$= \frac{1+e^{-z}}{(1+e^{-z})^2} - \left(\frac{1}{1+e^{-z}}\right)^2$$

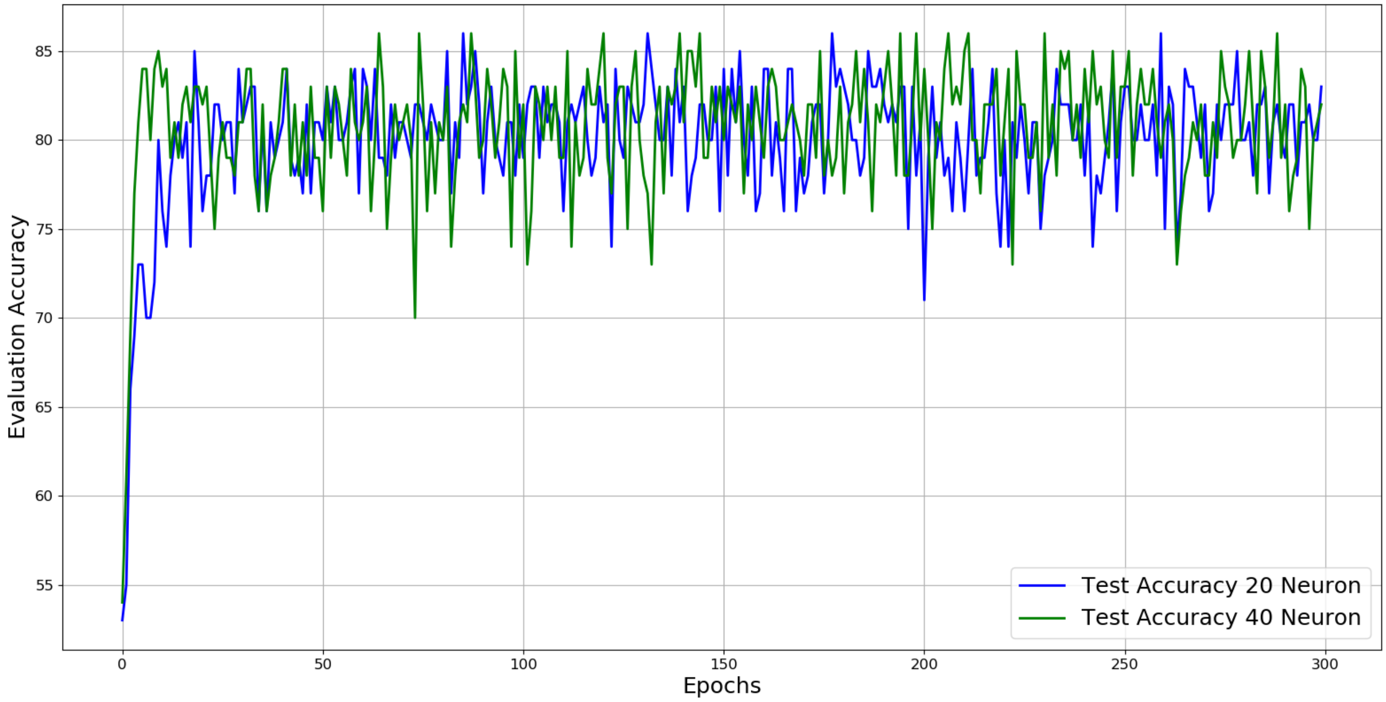
$$= \left(\frac{1}{1+e^{-z}}\right) - \left(\frac{1}{1+e^{-z}}\right)^2$$

$$= g(z) - (g(z))^2$$

$$\boxed{g'(z) = g(z)\left(1 - g(z)\right)}$$

II) Coding: Following the content and codes of this chapter, the dataset is of course MNIST

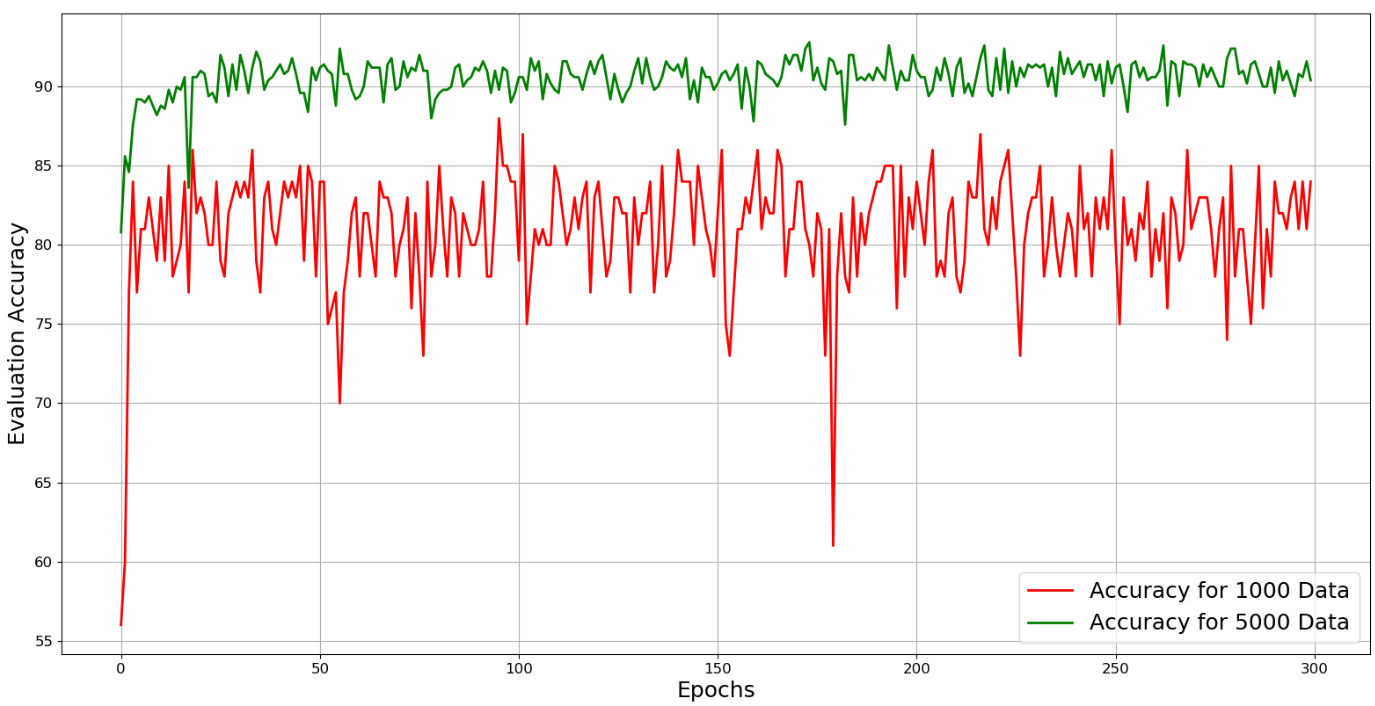
1. Compare the accuracy results between 20 and 40 hidden neuron network, using the same 1,000 training images, cross-entropy cost function, learning rate of η=0.5, mini-batch size of 10, and 300 epochs.




```
import mnist_loader
import matplotlib.pyplot as plt

training_data, validation_data, test_data = mnist_loader.load_data_wrapper()
import network2
# For example, if the list was [2, 3, 1] then it would be a three-layer network, with the first layer
# containing 2 neurons, the second layer 3 neurons, and the third layer 1 neuron.

Nuron = 20
net = network2.Network([784, Nuron, 10], cost=network2.CrossEntropyCost)
net.large_weight_initializer()
# net.SGD(training_data, 30, 10, 0.5, evaluation_data=test_data, monitor_evaluation_accuracy=True)

trainingSample1 = 1000
testSample1 = 100
# epochs  30 mini_batch_size  10 eta 1.0
epochs = 300
mini_batch = 10
eta = 0.5
evaluation_cost_20, evaluation_accuracy_20, training_cost_20, training_accuracy_20 = net.SGD(training_data[:trainingSample1], epochs, mini_bat
ch, eta, lmbda = 5.0, evaluation_data=validation_data[:testSample1],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_eval
uation_cost=True,monitor_training_accuracy= True)
print evaluation_cost_20
print evaluation_accuracy_20
print training_cost_20
print training_accuracy_20

# This is for 40
Nuron = 40
net = network2.Network([784, Nuron, 10], cost=network2.CrossEntropyCost)
net.large_weight_initializer()
trainingSample2 = 1000
testSample2 = 100
# epochs  30 mini_batch_size  10 eta 1.0
epochs = 300
mini_batch = 10
eta = 0.5
evaluation_cost_40, evaluation_accuracy_40, training_cost_40, training_accuracy_40 = net.SGD(training_data[:trainingSample2], epochs, mini_bat
ch, eta, lmbda = 5.0, evaluation_data=validation_data[:testSample2],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_eval
uation_cost=True,monitor_training_accuracy= True)
# net.save( "filename.json")
print evaluation_cost_40
print evaluation_accuracy_40
print training_cost_40
print training_accuracy_40

import numpy as np
npaTestAccuracy_20 = np.asarray(evaluation_accuracy_20, dtype=np.float32)
print "Test npa : ", (npaTestAccuracy_20/testSample1)*100
npaTestAccuracy_40 = np.asarray(evaluation_accuracy_40, dtype=np.float32)
print "Test npa : ", (npaTestAccuracy_40/testSample2)*100

t = np.arange(0, 110, 10)
# plt.rc('font',family='Comic Sans MS')
fig, ax = plt.subplots()
```

```
# ax.plot( (npaTestAccuracy_20/testSample1)*100, color='b', marker='^', ls='--', lw=2.0, label='Test Accuracy 20 Neuron')
# ax.plot((npaTestAccuracy_40/testSample2)*100, color='g', marker='d', ls='-.', lw=2.0, label='Test Accuracy 40 Neuron')
ax.plot( (npaTestAccuracy_20/testSample1)*100, color='b',lw=2.0, label='Test Accuracy 20 Neuron')
ax.plot((npaTestAccuracy_40/testSample2)*100, color='g',  lw=2.0, label='Test Accuracy 40 Neuron')
plt.ylabel('Evaluation Accuracy', fontsize = 18)
plt.xlabel('Epochs', fontsize = 18)
plt.legend(loc='best',fontsize = 18)
ax.grid(True)
ticklines = ax.get_xticklines() + ax.get_yticklines()
gridlines = ax.get_xgridlines()
ticklabels = ax.get_xticklabels() + ax.get_yticklabels()
for line in ticklines:
    line.set_linewidth(3)

for line in gridlines:
    line.set_linestyle('-')

for line in gridlines:
    line.set_linestyle('-')

for label in ticklabels:
    label.set_color('black')
    label.set_fontsize('large')
plt.show()


Output:
[54, 61, 69, 77, 81, 84, 84, 80, 84, 85, 83, 84, 79, 81, 79, 82, 83, 81, 83, 83, 82, 83, 79, 75, 79, 81, 79, 79, 78, 81, 81, 84, 84, 78, 76, 8
2, 76, 78, 79, 81, 84, 84, 78, 82, 78, 81, 78, 83, 79, 79, 76, 83, 79, 83, 82, 80, 78, 84, 81, 80, 81, 83, 76, 80, 86, 83, 75, 79, 82, 80, 81,
82, 80, 70, 86, 82, 76, 81, 77, 81, 80, 83, 74, 78, 81, 82, 81, 86, 83, 79, 80, 84, 82, 79, 81, 84, 83, 74, 85, 79, 82, 73, 76, 83, 82, 80, 83
, 80, 83, 79, 79, 85, 74, 81, 78, 79, 84, 82, 82, 84, 86, 79, 77, 82, 83, 83, 75, 83, 85, 80, 78, 77, 73, 81, 83, 77, 83, 82, 83, 86, 81, 85,
85, 83, 86, 79, 79, 83, 81, 83, 80, 83, 82, 81, 83, 77, 82, 80, 83, 81, 79, 83, 84, 83, 80, 80, 81, 82, 81, 80, 78, 82, 82, 79, 85, 78, 80, 78
, 79, 83, 77, 81, 82, 85, 81, 84, 81, 76, 82, 81, 83, 85, 82, 78, 86, 78, 78, 81, 86, 80, 84, 80, 75, 81, 80, 84, 86, 82, 83, 82, 85, 86, 80,
80, 77, 82, 82, 82, 84, 78, 81, 84, 73, 85, 82, 82, 79, 79, 81, 76, 86, 79, 82, 78, 85, 84, 85, 80, 82, 79, 84, 81, 85, 82, 83, 80, 79, 85, 79
, 83, 83, 85, 78, 82, 84, 82, 82, 84, 81, 79, 81, 82, 80, 73, 76, 78, 79, 81, 80, 82, 78, 78, 81, 79, 85, 83, 82, 79, 80, 80, 82, 85, 81, 77,
85, 83, 79, 81, 86, 79, 82, 76, 78, 79, 84, 83, 75, 80, 81, 82]
```

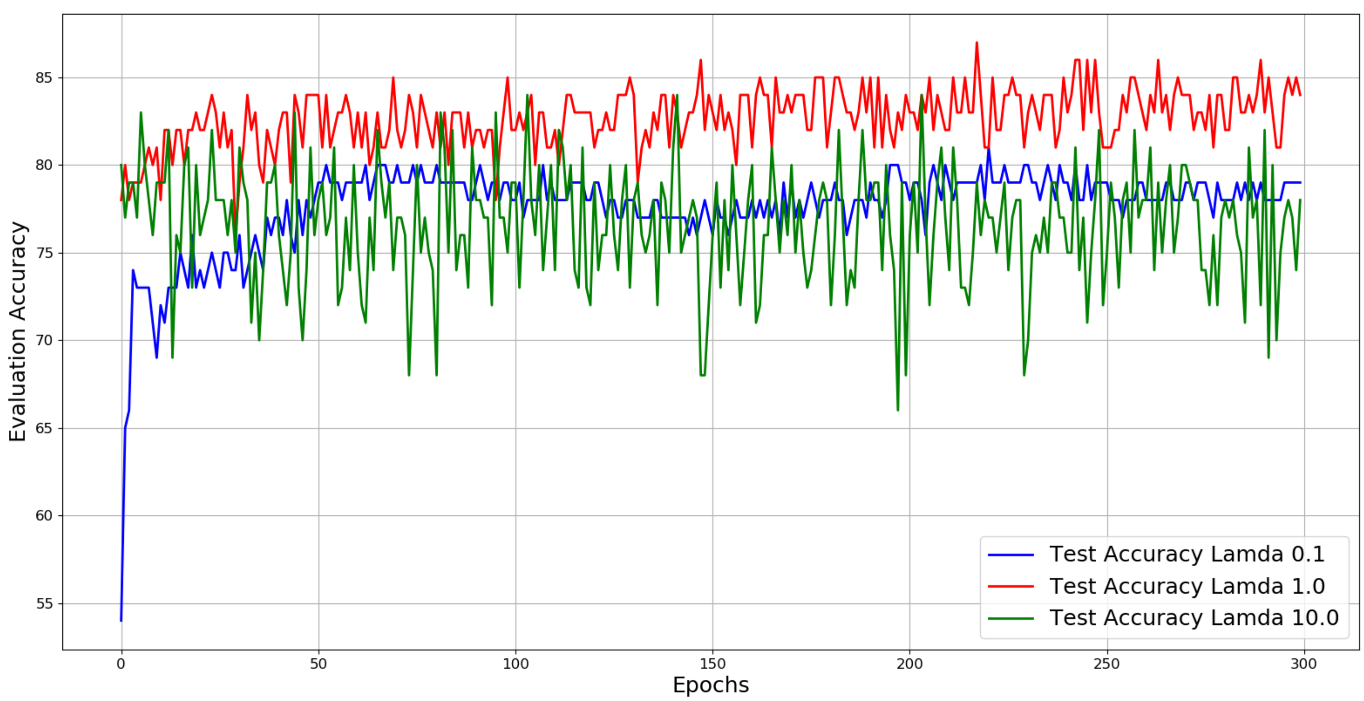
2. With a 40 hidden neuron network, compare the accuracy results between 1,000 and 5,000 training images, using cross-entropy cost function, learning rate of η=0.5, mini-batch size of 10, and 300 epochs.




```
import mnist_loader
import matplotlib.pyplot as plt

training_data, validation_data, test_data = mnist_loader.load_data_wrapper()
import network2
<strong># This is for 1000</strong>
Nuron = 40
net = network2.Network([784, Nuron, 10], cost=network2.CrossEntropyCost)
net.large_weight_initializer()
trainingSample1 = 1000
testSample1 = 100
epochs = 300
mini_batch = 10
```

```
eta = 0.5
evaluation_cost_20, evaluation_accuracy_20, training_cost_20, training_accuracy_20 = net.SGD(training_data[:trainingSample1], epochs, mini_batch, eta, lmbda = 5.0, evaluation_data=validation_data[:testSample1],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_evaluation_cost=True,monitor_training_accuracy= True)

<strong># This is for 5000</strong>
Nuron = 40
net = network2.Network([784, Nuron, 10], cost=network2.CrossEntropyCost)
net.large_weight_initializer()
trainingSample2 = 5000
testSample2 = 500
epochs = 300
mini_batch = 10
eta = 0.5
evaluation_cost_40, evaluation_accuracy_40, training_cost_40, training_accuracy_40 = net.SGD(training_data[:trainingSample2], epochs, mini_batch, eta, lmbda = 5.0, evaluation_data=validation_data[:testSample2],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_evaluation_cost=True,monitor_training_accuracy= True)

import numpy as np
npaTestAccuracy_20 = np.asarray(evaluation_accuracy_20, dtype=np.float32)
npaTestAccuracy_40 = np.asarray(evaluation_accuracy_40, dtype=np.float32)

t = np.arange(0, 110, 10)
fig, ax = plt.subplots()
ax.plot( (npaTestAccuracy_20/testSample1)*100, color='r',  lw=2.0, label='Accuracy for 1000 Data')
ax.plot((npaTestAccuracy_40/testSample2)*100, color='g', lw=2.0, label='Accuracy for 5000 Data')
plt.ylabel('Evaluation Accuracy', fontsize = 18)
plt.xlabel('Epochs', fontsize = 18)
plt.legend(loc='best',fontsize = 18)
ax.grid(True)
ticklines = ax.get_xticklines() + ax.get_yticklines()
gridlines = ax.get_xgridlines()
ticklabels = ax.get_xticklabels() + ax.get_yticklabels()

for line in ticklines:
    line.set_linewidth(3)

for line in gridlines:
    line.set_linestyle('-')

for line in gridlines:
    line.set_linestyle('-')

for label in ticklabels:
    label.set_color('black')
    label.set_fontsize('large')

plt.show()
```

3. With a 40 hidden neuron network, compare the accuracy results between with and without regularization, using the same 1,000 training images, cross-entropy cost function, learning rate of η=0.5, mini-batch size of 10, and 300 epochs, λ is chosen in {0.1, 1, 10}



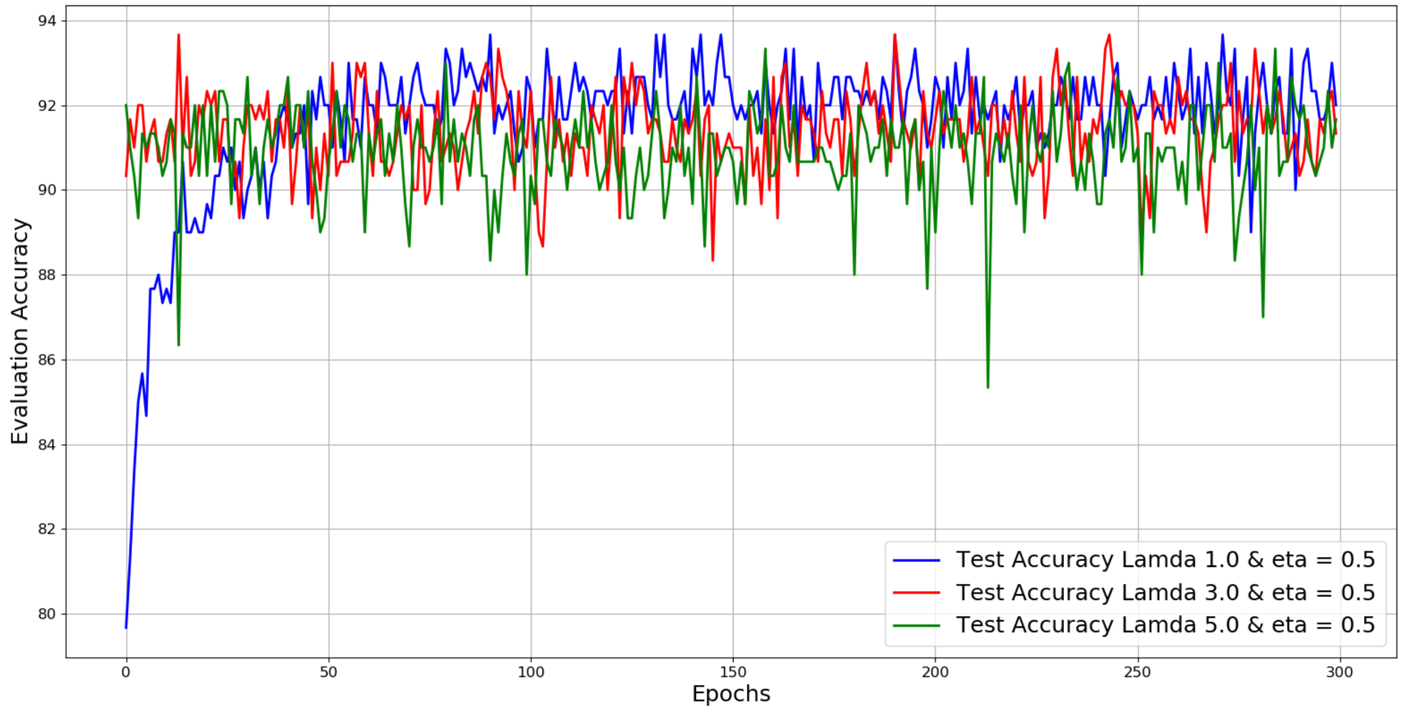

```
import mnist_loader
import matplotlib.pyplot as plt
```

```
training_data, validation_data, test_data = mnist_loader.load_data_wrapper()
import network2

Nuron = 40
net = network2.Network([784, Nuron, 10], cost=network2.CrossEntropyCost)
net.large_weight_initializer()
trainingSample2 = 1000
testSample2 = 100
# epochs  30 mini_batch_size  10 eta 1.0
epochs = 300
mini_batch = 10
eta = 0.5
# This is for lamda 0.1
evaluation_cost_01, evaluation_accuracy_01, training_cost_01, training_accuracy_01 = net.SGD(training_data[:trainingSample2], epochs, mini_batch, eta, lmbda = 0.1, evaluation_data=validation_data[:testSample2],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_evaluation_cost=True,monitor_training_accuracy= True)
# This is for lamda 1.0
evaluation_cost_1, evaluation_accuracy_1, training_cost_1, training_accuracy_1 = net.SGD(training_data[:trainingSample2], epochs, mini_batch,
 eta, lmbda = 1.0, evaluation_data=validation_data[:testSample2],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_evaluation_cost=True,monitor_training_accuracy= True)
# This is for lamda 10.0
evaluation_cost_10, evaluation_accuracy_10, training_cost_10, training_accuracy_10 = net.SGD(training_data[:trainingSample2], epochs, mini_batch, eta, lmbda = 10.0, evaluation_data=validation_data[:testSample2],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_evaluation_cost=True,monitor_training_accuracy= True)
import numpy as np
npaTestAccuracy_01 = np.asarray(evaluation_accuracy_01, dtype=np.float32)
npaTestAccuracy_1 = np.asarray(evaluation_accuracy_1, dtype=np.float32)
npaTestAccuracy_10 = np.asarray(evaluation_accuracy_10, dtype=np.float32)

t = np.arange(0, 110, 10)
fig, ax = plt.subplots()
ax.plot( (npaTestAccuracy_01/testSample2)*100, color='b',  lw=2.0, label='Test Accuracy Lamda 0.1')
ax.plot( (npaTestAccuracy_1/testSample2)*100, color='r', lw=2.0, label='Test Accuracy Lamda 1.0')
ax.plot((npaTestAccuracy_10/testSample2)*100, color='g',  lw=2.0, label='Test Accuracy Lamda 10.0')
plt.ylabel('Evaluation Accuracy', fontsize = 18)
plt.xlabel('Epochs', fontsize = 18)
plt.legend(loc='best',fontsize = 18)
ax.grid(True)
ticklines = ax.get_xticklines() + ax.get_yticklines()
gridlines = ax.get_xgridlines()
ticklabels = ax.get_xticklabels() + ax.get_yticklabels()
for line in ticklines:
    line.set_linewidth(3)

for line in gridlines:
    line.set_linestyle('-')

for line in gridlines:
    line.set_linestyle('-')

for label in ticklabels:
    label.set_color('black')
    label.set_fontsize('large')

plt.show()
```

4. Using heuristic approach, find the best hyper-parameters for learning a 40 hidden neuron network using the same 3,000 training images, cross-entropy cost function, mini-batch size of 10, and 300 epochs. Show the accuracy of learning epochs with these chosen parameters.

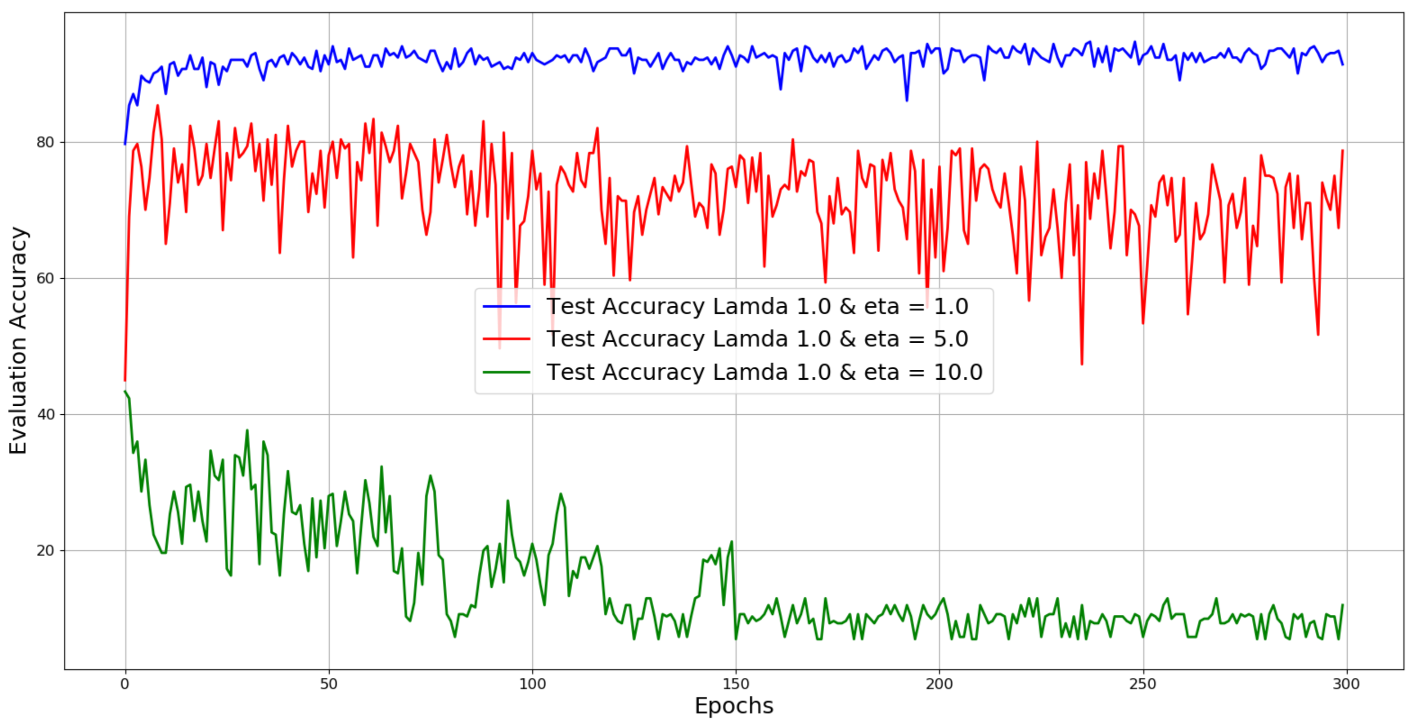
a. Lamda 1.0 & eta = 0.5 vs. Lamda 3.0 & eta = 0.5 vs. Lamda 5.0 & eta = 0.5

We get best performace for **Lamda 1.0 & eta = 0.5**



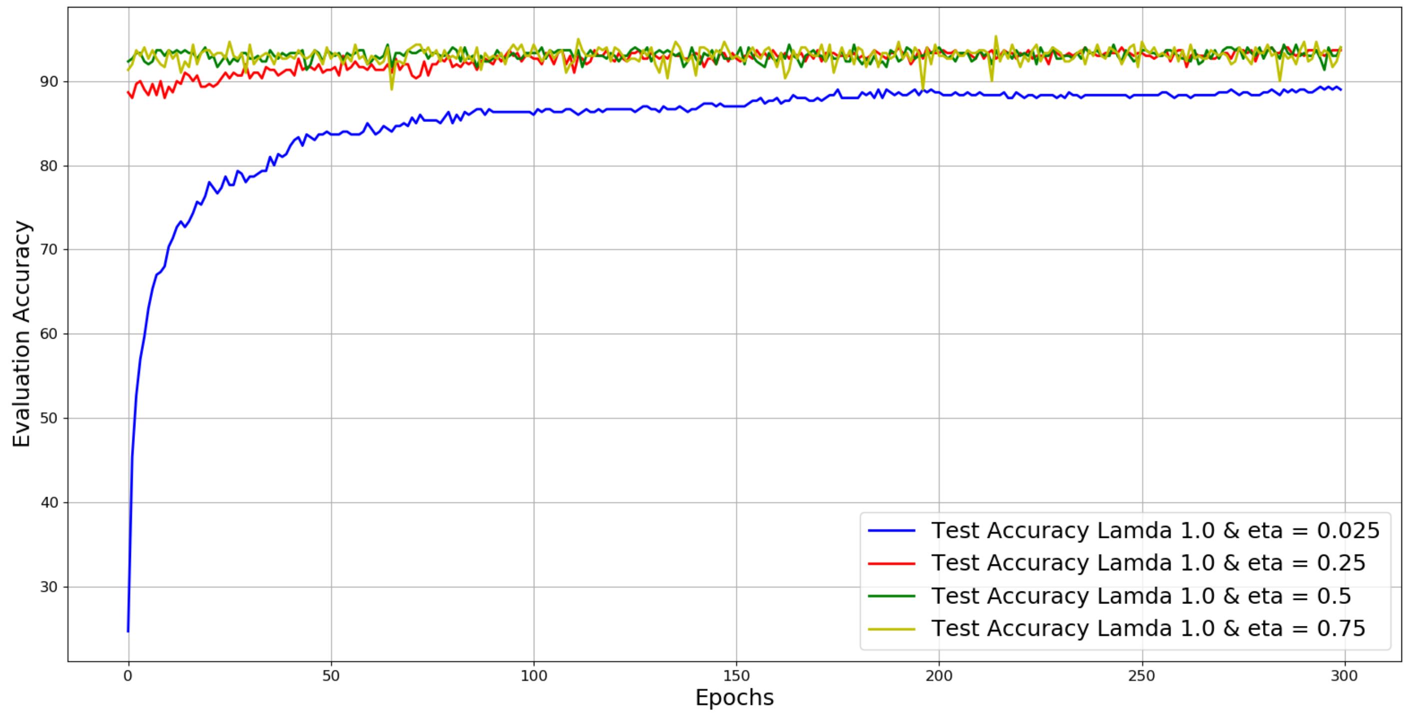
b. Lamda 1.0 & eta = 1.0 vs. Lamda 1.0 & eta = 5.0 vs. Lamda1.0 & eta = 10.0

Based on the previous result now we are tuning the eta value. We get best performance for **Lamda 1.0 & eta = 1.0** but this result is poor compared to **Lamda 1.0 & eta = 0.5** in previous figure.



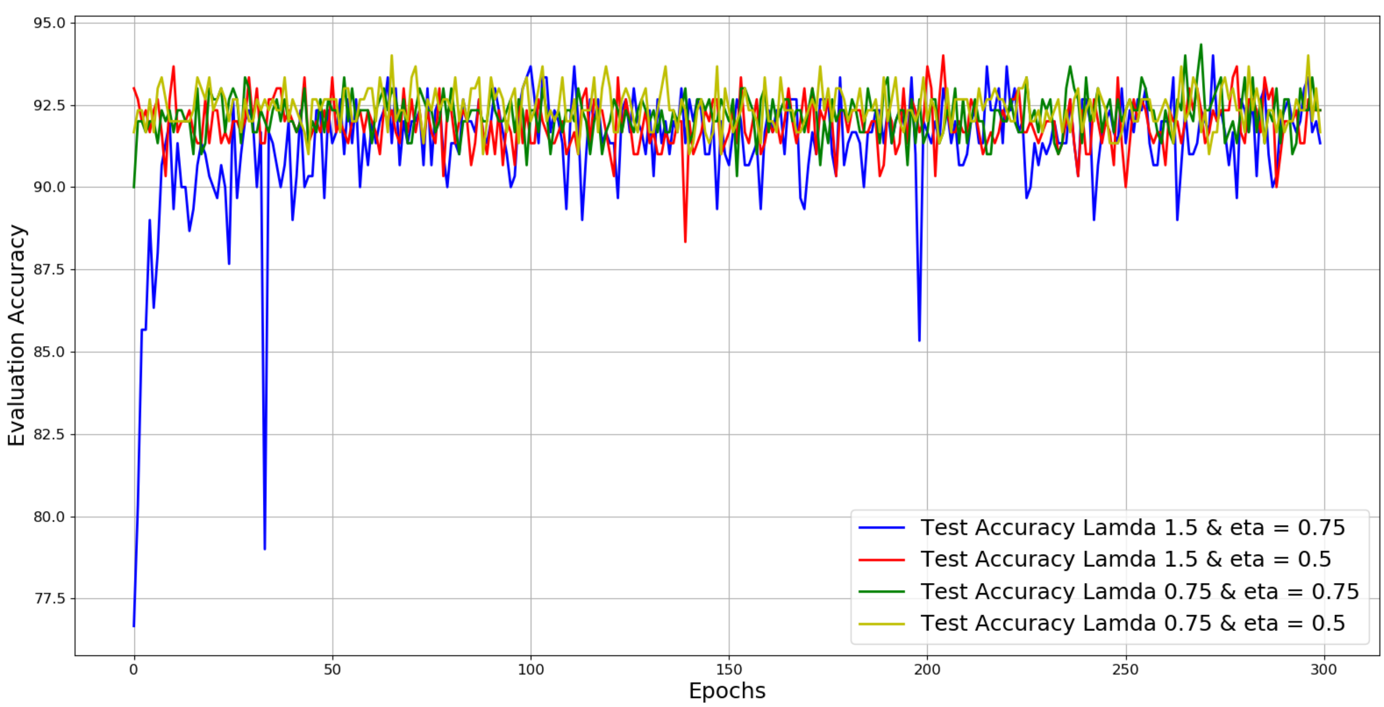
c. Lamda 1.0 & eta = 0.025 vs. Lamda 1.0 & eta = 0.25 vs. Lamda1.0 & eta = 0.5 vs. Lamda1.0 & eta = 0.75

Based on the previous result (b) now we are tuning the eta value again. We get almost similar performance for **Lamda 1.0 & eta = 0.5** and **Lamda 1.0 & eta = 0.75** in figure c.



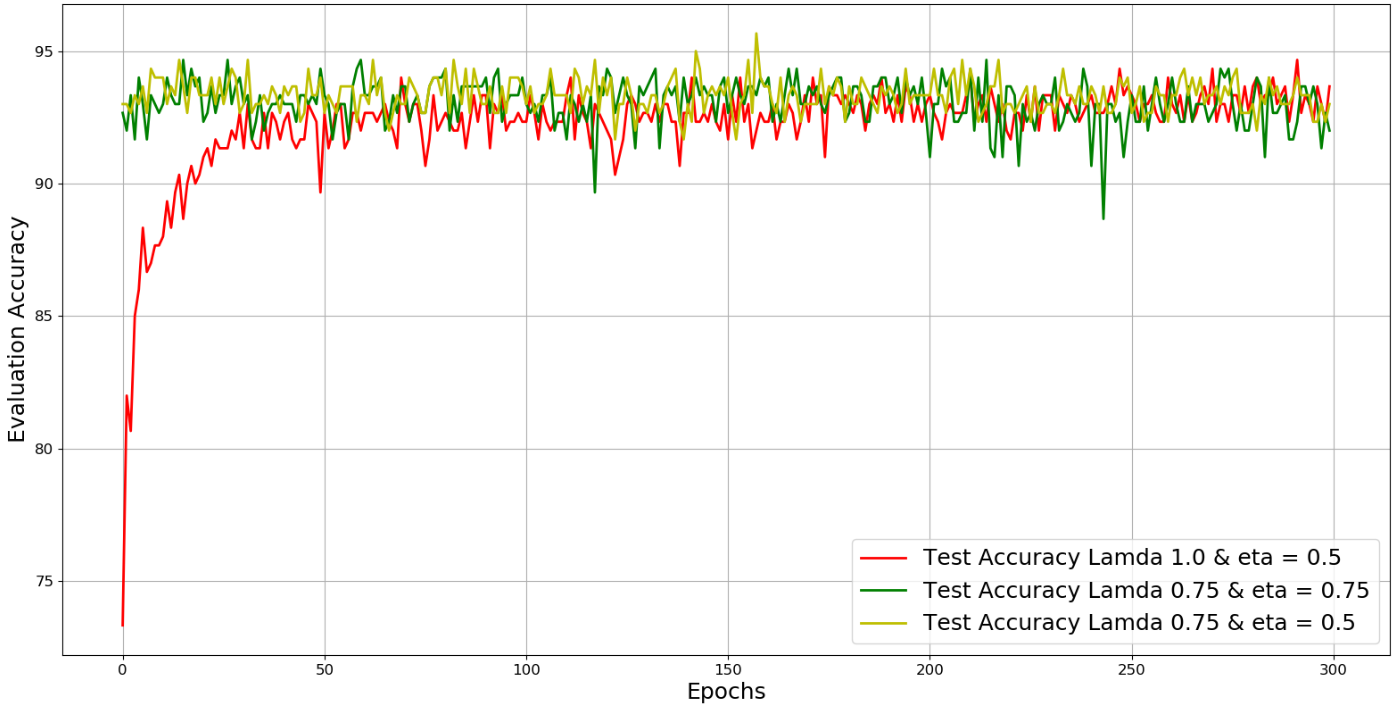
d. Lamda 1.5 & eta = 0.75 vs. Lamda 1.5 & eta = 0.5 vs. Lamda 0.75 & eta = 0.75 vs. Lamda 0.75 & eta = 0.5

Based on the previous result (c) now we are tuning the eta and lamda together. We get almost best performance for **Lamda 0.75 & eta = 0.5** d.



e. Lamda 1.0 & eta = 0.5 vs. Lamda 0.75 & eta = 0.75 vs. Lamda 0.75 & eta = 0.5

Based on the previous result (a, b, d, and d) now we are making decision about the eta and lamda . We get best performance for **Lamda 0.75 & eta = 0.5** in fig e.




```python
import mnist_loader
import matplotlib.pyplot as plt

training_data, validation_data, test_data = mnist_loader.load_data_wrapper()
import network2
Nuron = 40
net = network2.Network([784, Nuron, 10], cost=network2.CrossEntropyCost)
net.large_weight_initializer()
trainingSample2 = 3000
testSample2 = 300
epochs = 300
mini_batch = 10
eta = 0.5

#Lamda 1.0 & eta = 0.5 vs. Lamda 3.0 & eta = 0.5 vs. Lamda 5.0 & eta = 0.5
evaluation_cost_01, evaluation_accuracy_01, training_cost_01, training_accuracy_01 = net.SGD(training_data[:trainingSample2], epochs, mini_bat
ch, eta, lmbda = 1.0, evaluation_data=validation_data[:testSample2],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_eval
uation_cost=True,monitor_training_accuracy= True)
evaluation_cost_1, evaluation_accuracy_1, training_cost_1, training_accuracy_1 = net.SGD(training_data[:trainingSample2], epochs, mini_batch,
 eta, lmbda = 3.0, evaluation_data=validation_data[:testSample2],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_evaluat
ion_cost=True,monitor_training_accuracy= True)

evaluation_cost_10, evaluation_accuracy_10, training_cost_10, training_accuracy_10 = net.SGD(training_data[:trainingSample2], epochs, mini_bat
ch, eta, lmbda = 5.0, evaluation_data=validation_data[:testSample2],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_eval
uation_cost=True,monitor_training_accuracy= True)
import numpy as np
npaTestAccuracy_01 = np.asarray(evaluation_accuracy_01, dtype=np.float32)
npaTestAccuracy_1 = np.asarray(evaluation_accuracy_1, dtype=np.float32)
npaTestAccuracy_10 = np.asarray(evaluation_accuracy_10, dtype=np.float32)
t = np.arange(0, 110, 10)
fig, ax = plt.subplots()
ax.plot( (npaTestAccuracy_01/testSample2)*100, color='b',  lw=2.0, label='Test Accuracy Lamda 1.0 & eta = 0.5')
ax.plot( (npaTestAccuracy_1/testSample2)*100, color='r', lw=2.0, label='Test Accuracy Lamda 3.0 & eta = 0.5')
ax.plot((npaTestAccuracy_10/testSample2)*100, color='g',  lw=2.0, label='Test Accuracy Lamda 5.0 & eta = 0.5')
plt.ylabel('Evaluation Accuracy', fontsize = 18)
plt.xlabel('Epochs', fontsize = 18)
plt.legend(loc='best',fontsize = 18)
plt.show()

#Lamda 1.0 & eta = 1.0 vs. Lamda 1.0 & eta = 5.0 vs. Lamda1.0 & eta = 10.0
evaluation_cost_01, evaluation_accuracy_01, training_cost_01, training_accuracy_01 = net.SGD(training_data[:trainingSample2], epochs, mini_bat
ch, 1.0, lmbda = 1.0, evaluation_data=validation_data[:testSample2],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_eval
uation_cost=True,monitor_training_accuracy= True)
evaluation_cost_1, evaluation_accuracy_1, training_cost_1, training_accuracy_1 = net.SGD(training_data[:trainingSample2], epochs, mini_batch,
5.0, lmbda = 1.0, evaluation_data=validation_data[:testSample2],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_evaluati
on_cost=True,monitor_training_accuracy= True)
evaluation_cost_10, evaluation_accuracy_10, training_cost_10, training_accuracy_10 = net.SGD(training_data[:trainingSample2], epochs, mini_bat
ch, 10.0, lmbda = 1.0, evaluation_data=validation_data[:testSample2],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_eva
luation_cost=True,monitor_training_accuracy= True)
import numpy as np
npaTestAccuracy_01 = np.asarray(evaluation_accuracy_01, dtype=np.float32)
npaTestAccuracy_1 = np.asarray(evaluation_accuracy_1, dtype=np.float32)
npaTestAccuracy_10 = np.asarray(evaluation_accuracy_10, dtype=np.float32)
t = np.arange(0, 110, 10)
fig, ax = plt.subplots()
```

```
ax.plot( (npaTestAccuracy_01/testSample2)*100, color='b',  lw=2.0, label='Test Accuracy Lamda 1.0 & eta = 1.0')
ax.plot( (npaTestAccuracy_1/testSample2)*100, color='r', lw=2.0, label='Test Accuracy Lamda 1.0 & eta = 5.0')
ax.plot((npaTestAccuracy_10/testSample2)*100, color='g',  lw=2.0, label='Test Accuracy Lamda 1.0 & eta = 10.0')
plt.ylabel('Evaluation Accuracy', fontsize = 18)
plt.xlabel('Epochs', fontsize = 18)
plt.legend(loc='best',fontsize = 18)
ax.grid(True)
ticklines = ax.get_xticklines() + ax.get_yticklines()
gridlines = ax.get_xgridlines()
ticklabels = ax.get_xticklabels() + ax.get_yticklabels()
plt.show()


#Lamda 1.0 & eta = 0.025 vs. Lamda 1.0 & eta = 0.25  vs. Lamda1.0 & eta = 0.5  vs. Lamda1.0 & eta = 0.75
evaluation_cost_01, evaluation_accuracy_01, training_cost_01, training_accuracy_01 = net.SGD(training_data[:trainingSample2], epochs, mini_bat
ch, 0.025, lmbda = 1.0, evaluation_data=validation_data[:testSample2],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_ev
aluation_cost=True,monitor_training_accuracy= True)
evaluation_cost_1, evaluation_accuracy_1, training_cost_1, training_accuracy_1 = net.SGD(training_data[:trainingSample2], epochs, mini_batch,
0.25, lmbda = 1.0, evaluation_data=validation_data[:testSample2],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_evaluat
ion_cost=True,monitor_training_accuracy= True)
evaluation_cost_10, evaluation_accuracy_10, training_cost_10, training_accuracy_10 = net.SGD(training_data[:trainingSample2], epochs, mini_bat
ch, 0.5, lmbda = 1.0, evaluation_data=validation_data[:testSample2],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_eval
uation_cost=True,monitor_training_accuracy= True)
evaluation_cost_11, evaluation_accuracy_11, training_cost_11, training_accuracy_11 = net.SGD(training_data[:trainingSample2], epochs, mini_bat
ch, 0.75, lmbda = 1.0, evaluation_data=validation_data[:testSample2],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_eva
luation_cost=True,monitor_training_accuracy= True)
import numpy as np
npaTestAccuracy_01 = np.asarray(evaluation_accuracy_01, dtype=np.float32)
npaTestAccuracy_1 = np.asarray(evaluation_accuracy_1, dtype=np.float32)
npaTestAccuracy_10 = np.asarray(evaluation_accuracy_10, dtype=np.float32)
paTestAccuracy_11 = np.asarray(evaluation_accuracy_11, dtype=np.float32)
t = np.arange(0, 110, 10)
fig, ax = plt.subplots()
ax.plot( (npaTestAccuracy_01/testSample2)*100, color='b',  lw=2.0, label='Test Accuracy Lamda 1.0 & eta = 0.025')
ax.plot( (npaTestAccuracy_1/testSample2)*100, color='r', lw=2.0, label='Test Accuracy Lamda 1.0 & eta = 0.25')
ax.plot((npaTestAccuracy_10/testSample2)*100, color='g',  lw=2.0, label='Test Accuracy Lamda 1.0 & eta = 0.5')
ax.plot((npaTestAccuracy_11/testSample2)*100, color='y',  lw=2.0, label='Test Accuracy Lamda 1.0 & eta = 0.75')
plt.ylabel('Evaluation Accuracy', fontsize = 18)
plt.xlabel('Epochs', fontsize = 18)
plt.legend(loc='best',fontsize = 18)
ax.grid(True)
ticklines = ax.get_xticklines() + ax.get_yticklines()
gridlines = ax.get_xgridlines()
ticklabels = ax.get_xticklabels() + ax.get_yticklabels()
plt.show()


#Lamda 1.5 & eta = 0.75 vs. Lamda 1.5 & eta = 0.5  vs. Lamda 0.75 & eta = 0.75  vs. Lamda 0.75 & eta = 0.5
evaluation_cost_01, evaluation_accuracy_01, training_cost_01, training_accuracy_01 = net.SGD(training_data[:trainingSample2], epochs, mini_bat
ch, 0.75, lmbda = 1.5, evaluation_data=validation_data[:testSample2],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_eva
luation_cost=True,monitor_training_accuracy= True)
evaluation_cost_1, evaluation_accuracy_1, training_cost_1, training_accuracy_1 = net.SGD(training_data[:trainingSample2], epochs, mini_batch,
0.5, lmbda = 1.5, evaluation_data=validation_data[:testSample2],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_evaluati
on_cost=True,monitor_training_accuracy= True)
evaluation_cost_10, evaluation_accuracy_10, training_cost_10, training_accuracy_10 = net.SGD(training_data[:trainingSample2], epochs, mini_bat
ch, 0.75, lmbda = 0.75, evaluation_data=validation_data[:testSample2],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_ev
aluation_cost=True,monitor_training_accuracy= True)
evaluation_cost_11, evaluation_accuracy_11, training_cost_11, training_accuracy_11 = net.SGD(training_data[:trainingSample2], epochs, mini_bat
ch, 0.5, lmbda = 0.75, evaluation_data=validation_data[:testSample2],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_eva
luation_cost=True,monitor_training_accuracy= True)
import numpy as np
npaTestAccuracy_01 = np.asarray(evaluation_accuracy_01, dtype=np.float32)
npaTestAccuracy_1 = np.asarray(evaluation_accuracy_1, dtype=np.float32)
npaTestAccuracy_10 = np.asarray(evaluation_accuracy_10, dtype=np.float32)
paTestAccuracy_11 = np.asarray(evaluation_accuracy_11, dtype=np.float32)
t = np.arange(0, 110, 10)
fig, ax = plt.subplots()
ax.plot( (npaTestAccuracy_01/testSample2)*100, color='b',  lw=2.0, label='Test Accuracy Lamda 1.5 & eta = 0.75')
ax.plot( (npaTestAccuracy_1/testSample2)*100, color='r', lw=2.0, label='Test Accuracy Lamda 1.5 & eta = 0.5')
ax.plot((npaTestAccuracy_10/testSample2)*100, color='g',  lw=2.0, label='Test Accuracy Lamda 0.75 & eta = 0.75')
# ax.plot((npaTestAccuracy_11/testSample2)*100, color='y',  lw=2.0, label='Test Accuracy Lamda 0.75 & eta = 0.5')
plt.ylabel('Evaluation Accuracy', fontsize = 18)
plt.xlabel('Epochs', fontsize = 18)
plt.legend(loc='best',fontsize = 18)
ax.grid(True)
ticklines = ax.get_xticklines() + ax.get_yticklines()
gridlines = ax.get_xgridlines()
ticklabels = ax.get_xticklabels() + ax.get_yticklabels()
plt.show()


#Lamda 1.0 & eta = 0.5 vs. Lamda 0.75 & eta = 0.75  vs. Lamda 0.75 & eta = 0.5
evaluation_cost_1, evaluation_accuracy_1, training_cost_1, training_accuracy_1 = net.SGD(training_data[:trainingSample2], epochs, mini_batch,
0.5, lmbda = 1.0, evaluation_data=validation_data[:testSample2],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_evaluati
on_cost=True,monitor_training_accuracy= True)
evaluation_cost_10, evaluation_accuracy_10, training_cost_10, training_accuracy_10 = net.SGD(training_data[:trainingSample2], epochs, mini_bat
ch, 0.75, lmbda = 0.75, evaluation_data=validation_data[:testSample2],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_ev
aluation_cost=True,monitor_training_accuracy= True)
evaluation_cost_11, evaluation_accuracy_11, training_cost_11, training_accuracy_11 = net.SGD(training_data[:trainingSample2], epochs, mini_bat
ch, 0.5, lmbda = 0.75, evaluation_data=validation_data[:testSample2],  monitor_evaluation_accuracy=True,monitor_training_cost=True,monitor_eva
luation_cost=True,monitor_training_accuracy= True)
```

```
import numpy as np
npaTestAccuracy_1 = np.asarray(evaluation_accuracy_1, dtype=np.float32)
npaTestAccuracy_10 = np.asarray(evaluation_accuracy_10, dtype=np.float32)npaTestAccuracy_11 = np.asarray(evaluation_accuracy_11, dtype=np.floa
t32)
t = np.arange(0, 110, 10)
fig, ax = plt.subplots()
ax.plot( (npaTestAccuracy_1/testSample2)*100, color='r', lw=2.0, label='Test Accuracy Lamda 1.0 & eta = 0.5')
ax.plot((npaTestAccuracy_10/testSample2)*100, color='g',  lw=2.0, label='Test Accuracy Lamda 0.75 & eta = 0.75')
ax.plot((npaTestAccuracy_11/testSample2)*100, color='y',  lw=2.0, label='Test Accuracy Lamda 0.75 & eta = 0.5')
plt.ylabel('Evaluation Accuracy', fontsize = 18)
plt.xlabel('Epochs', fontsize = 18)
plt.legend(loc='best',fontsize = 18)
ax.grid(True)
ticklines = ax.get_xticklines() + ax.get_yticklines()
gridlines = ax.get_xgridlines()
ticklabels = ax.get_xticklabels() + ax.get_yticklabels()
plt.show()
```

hw6

~ An instructor (Nguyen H. Tran) thinks this is a good note ~

This private post is only visible to Instructors and MD SHIRAJUM MUNIR

Updated 2 years ago by MD SHIRAJUM MUNIR

followup discussions *for lingering questions and comments*