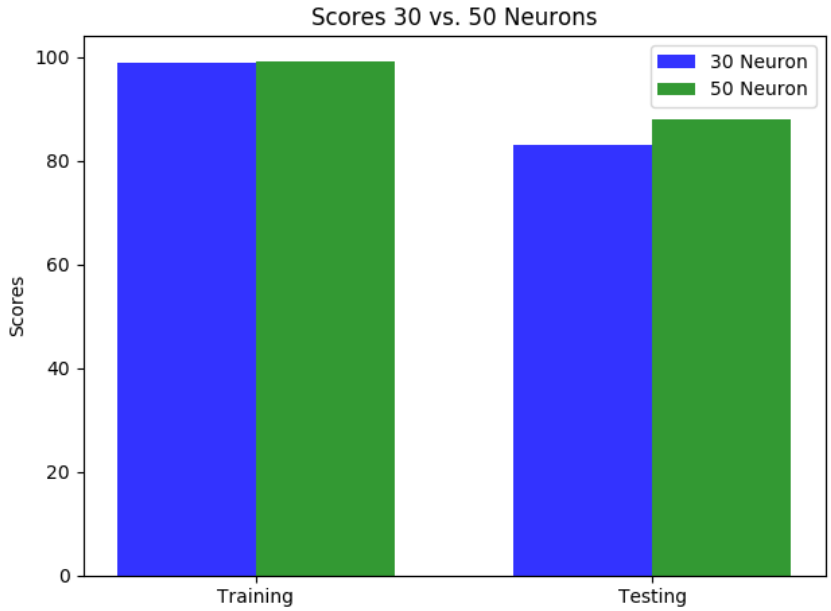
! This class has been made inactive. No posts will be allowed until an instructor reactivates the class.

private note @162 2 views

HW7_2017310936_Md_Shirajum_Munir

1. Compare the accuracy results between 30 and 50 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.

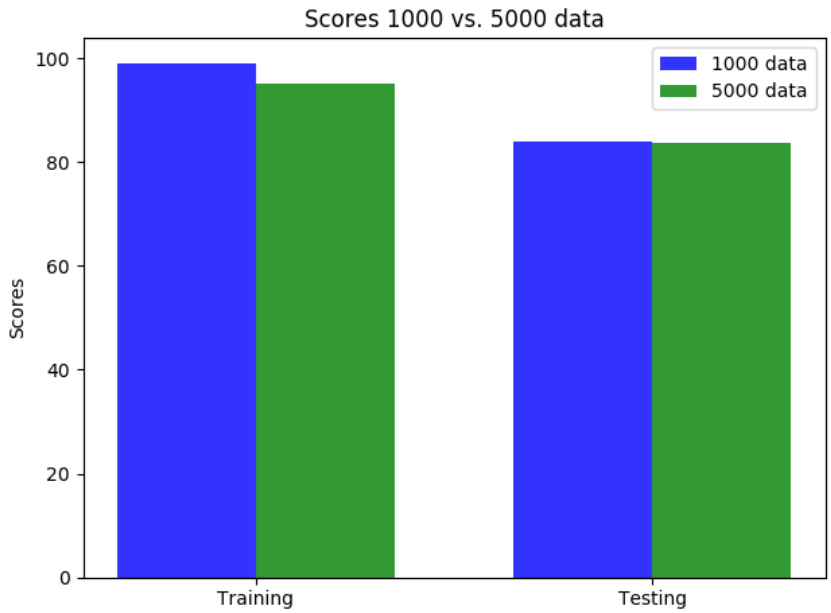



```
import tensorflow as tf
import tensorflow.contrib.keras as keras
import numpy as np

# importing mnist dataset
from tensorflow.examples.tutorials.mnist import input_data

epochsNo = 300
mini_batch_size = 10
eta = 0.5
trainingSample = 1000
testingSample = 100
noOfNeuron = 30

mnist = input_data.read_data_sets("/tmp/data/", one_hot=False)
X_train = mnist.train.images[:trainingSample]
y_train = mnist.train.labels[:trainingSample]
X_test = mnist.test.images[:testingSample]
y_test = mnist.test.labels[:testingSample]

print(len(X_train))
print(len(X_test))

## mean centering and normalization:
mean_vals = np.mean(X_train, axis=0)
std_val = np.std(X_train)
X_train_centered = (X_train - mean_vals) / std_val
X_test_centered = (X_test - mean_vals) / std_val
del X_train, X_test

# makinf keras model
y_train_onehot = keras.utils.to_categorical(y_train)
model = keras.models.Sequential()
model.add(
    keras.layers.Dense(
        units=noOfNeuron,
        input_dim=X_train_centered.shape[1],
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        activation='tanh'))

model.add(
    keras.layers.Dense(
        units=noOfNeuron,
        input_dim=noOfNeuron,
        kernel_initializer='glorot_uniform',
```

```
        bias_initializer='zeros',
        activation='tanh'))

model.add(
    keras.layers.Dense(
        units=y_train_onehot.shape[1],
        input_dim=noOfNeuron,
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        activation='softmax'))

# declare the optimizer and cost function
sgd_optimizer = keras.optimizers.SGD(lr=eta)
model.compile(optimizer=sgd_optimizer, loss='categorical_crossentropy')
history = model.fit(X_train_centered, y_train_onehot,
                    batch_size=mini_batch_size, epochs=epochsNo,
                    verbose=1,
                    validation_split=0.1)

# checking accuracy on training and testing dataset
y_train_pred = model.predict_classes(X_train_centered, verbose=0)

correct_preds = np.sum(y_train == y_train_pred, axis=0)
train_acc = correct_preds / y_train.shape[0]
print('Training accuracy: %.2f%%' % (train_acc * 100))

y_test_pred = model.predict_classes(X_test_centered, verbose=0)
correct_preds = np.sum(y_test == y_test_pred, axis=0)
test_acc = correct_preds / y_test.shape[0]
print('Test accuracy: %.2f%%' % (test_acc * 100))

epochsNo = 300
mini_batch_size = 10
eta = 0.5
trainingSample = 1000
testingSample = 100
noOfNeuron = 50

mnist = input_data.read_data_sets("/tmp/data/", one_hot=False)
X_train = mnist.train.images[:trainingSample]
y_train = mnist.train.labels[:trainingSample]
X_test = mnist.test.images[:testingSample]
y_test = mnist.test.labels[:testingSample]

print(len(X_train))
print(len(X_test))

## mean centering and normalization:
mean_vals = np.mean(X_train, axis=0)
std_val = np.std(X_train)
X_train_centered = (X_train - mean_vals) / std_val
X_test_centered = (X_test - mean_vals) / std_val
del X_train, X_test

# makinf keras model
y_train_onehot = keras.utils.to_categorical(y_train)
model = keras.models.Sequential()
model.add(
    keras.layers.Dense(
        units=noOfNeuron,
        input_dim=X_train_centered.shape[1],
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        activation='tanh'))

model.add(
    keras.layers.Dense(
        units=noOfNeuron,
        input_dim=noOfNeuron,
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        activation='tanh'))

model.add(
    keras.layers.Dense(
        units=y_train_onehot.shape[1],
        input_dim=noOfNeuron,
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        activation='softmax'))

# declare the optimizer and cost function
sgd_optimizer = keras.optimizers.SGD(lr=eta)
model.compile(optimizer=sgd_optimizer, loss='categorical_crossentropy')
history = model.fit(X_train_centered, y_train_onehot,
                    batch_size=mini_batch_size, epochs=epochsNo,
                    verbose=1,
                    validation_split=0.1)

# checking accuracy on training and testing dataset
y_train_pred = model.predict_classes(X_train_centered, verbose=0)
```

```
correct_preds = np.sum(y_train == y_train_pred, axis=0)
train_acc = correct_preds / y_train.shape[0]
print('Training accuracy: %.2f%%' % (train_acc * 100))

y_test_pred = model.predict_classes(X_test_centered, verbose=0)
correct_preds = np.sum(y_test == y_test_pred, axis=0)
test_acc = correct_preds / y_test.shape[0]
print('Test accuracy: %.2f%%' % (test_acc * 100))

import numpy as np
import matplotlib.pyplot as plt

# data to plot
n_groups = 2

neuron_30  = (98.80, 83.00)
neuron_50 = (99.10, 88.00)

# create plot
fig, ax = plt.subplots()
index = np.arange(n_groups)
bar_width = 0.35
opacity = 0.8

rects1 = plt.bar(index, neuron_30, bar_width,
                 alpha=opacity,
                 color='b',
                 label='30 Neuron')

rects2 = plt.bar(index + bar_width, neuron_50, bar_width,
                 alpha=opacity,
                 color='g',
                 label='50 Neuron')

# plt.xlabel('# of Neuron')
plt.ylabel('Scores')
plt.title('Scores 30 vs. 50 Neurons')
plt.xticks((index + bar_width/2.0), ('Training', 'Testing'))
plt.legend()

plt.tight_layout()
plt.show()

Output:
1000
100
Train on 900 samples, validate on 100 samples
Training accuracy: 98.80%
Test accuracy: 83.00%
Training accuracy: 99.10%
Test accuracy: 88.00%
```

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.



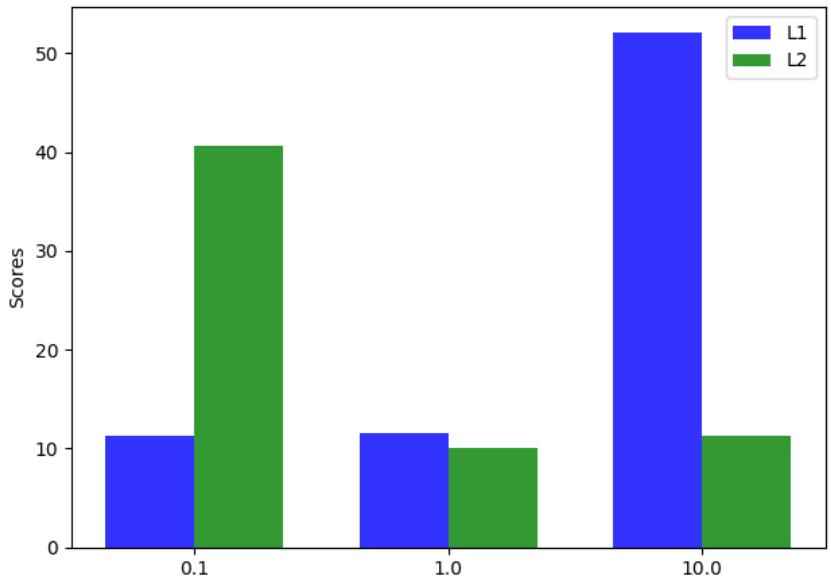
```
import tensorflow as tf
import tensorflow.contrib.keras as keras
import numpy as np

# importing mnist dataset
from tensorflow.examples.tutorials.mnist import input_data

epochsNo = 300
mini_batch_size = 10
eta = 0.5
trainingSample = 1000
testingSample = 100
noOfNeuron = 40

mnist = input_data.read_data_sets("/tmp/data/", one_hot=False)
X_train = mnist.train.images[:trainingSample]
y_train = mnist.train.labels[:trainingSample]
X_test = mnist.test.images[:testingSample]
y_test = mnist.test.labels[:testingSample]

print(len(X_train))
print(len(X_test))

## mean centering and normalization:
mean_vals = np.mean(X_train, axis=0)
std_val = np.std(X_train)
X_train_centered = (X_train - mean_vals) / std_val
X_test_centered = (X_test - mean_vals) / std_val
del X_train, X_test

# makinf keras model
y_train_onehot = keras.utils.to_categorical(y_train)
model = keras.models.Sequential()
model.add(
    keras.layers.Dense(
        units=noOfNeuron,
        input_dim=X_train_centered.shape[1],
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        activation='tanh'))

model.add(
    keras.layers.Dense(
        units=noOfNeuron,
        input_dim=noOfNeuron,
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        activation='tanh'))

model.add(
    keras.layers.Dense(
        units=y_train_onehot.shape[1],
        input_dim=noOfNeuron,
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        activation='softmax'))

# declare the optimizer and cost function
sgd_optimizer = keras.optimizers.SGD(lr=eta)
model.compile(optimizer=sgd_optimizer, loss='categorical_crossentropy')
history = model.fit(X_train_centered, y_train_onehot,
                    batch_size=mini_batch_size, epochs=epochsNo,
                    verbose=1,
                    validation_split=0.1)

# checking accuracy on training and testing dataset
y_train_pred = model.predict_classes(X_train_centered, verbose=0)

correct_preds = np.sum(y_train == y_train_pred, axis=0)
train_acc = correct_preds / y_train.shape[0]
print('Training accuracy: %.2f%%' % (train_acc * 100))

y_test_pred = model.predict_classes(X_test_centered, verbose=0)
correct_preds = np.sum(y_test == y_test_pred, axis=0)
test_acc = correct_preds / y_test.shape[0]
print('Test accuracy: %.2f%%' % (test_acc * 100))



epochsNo = 300
mini_batch_size = 10
eta = 0.5
trainingSample = 5000
testingSample = 500
noOfNeuron = 40

mnist = input_data.read_data_sets("/tmp/data/", one_hot=False)
X_train = mnist.train.images[:trainingSample]
y_train = mnist.train.labels[:trainingSample]
X_test = mnist.test.images[:testingSample]
```

```
y_test = mnist.test.labels[:testingSample]

print(len(X_train))
print(len(X_test))

## mean centering and normalization:
mean_vals = np.mean(X_train, axis=0)
std_val = np.std(X_train)
X_train_centered = (X_train - mean_vals) / std_val
X_test_centered = (X_test - mean_vals) / std_val
del X_train, X_test

# makinf keras model
y_train_onehot = keras.utils.to_categorical(y_train)
model = keras.models.Sequential()
model.add(
    keras.layers.Dense(
        units=noOfNeuron,
        input_dim=X_train_centered.shape[1],
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        activation='tanh'))

model.add(
    keras.layers.Dense(
        units=noOfNeuron,
        input_dim=noOfNeuron,
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        activation='tanh'))

model.add(
    keras.layers.Dense(
        units=y_train_onehot.shape[1],
        input_dim=noOfNeuron,
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        activation='softmax'))

# declare the optimizer and cost function
sgd_optimizer = keras.optimizers.SGD(lr=eta)
model.compile(optimizer=sgd_optimizer, loss='categorical_crossentropy')
history = model.fit(X_train_centered, y_train_onehot,
                    batch_size=mini_batch_size, epochs=epochsNo,
                    verbose=1,
                    validation_split=0.1)

# checking accuracy on training and testing dataset
y_train_pred = model.predict_classes(X_train_centered, verbose=0)

correct_preds = np.sum(y_train == y_train_pred, axis=0)
train_acc = correct_preds / y_train.shape[0]
print('Training accuracy: %.2f%%' % (train_acc * 100))

y_test_pred = model.predict_classes(X_test_centered, verbose=0)
correct_preds = np.sum(y_test == y_test_pred, axis=0)
test_acc = correct_preds / y_test.shape[0]
print('Test accuracy: %.2f%%' % (test_acc * 100))


import numpy as np
import matplotlib.pyplot as plt

# data to plot
n_groups = 2
data1000  = (99.00, 84.00)
data5000 = (95.26, 83.80)

# create plot
fig, ax = plt.subplots()
index = np.arange(n_groups)
bar_width = 0.35
opacity = 0.8

rects1 = plt.bar(index, data1000, bar_width,
                 alpha=opacity,
                 color='b',
                 label='1000 data')

rects2 = plt.bar(index + bar_width, data5000, bar_width,
                 alpha=opacity,
                 color='g',
                 label='5000 data')

# plt.xlabel('# of Neuron')
plt.ylabel('Scores')
plt.title('Scores 1000 vs. 5000 data')
plt.xticks((index + bar_width/2.0), ('Training', 'Testing'))
plt.legend()

plt.tight_layout()
```

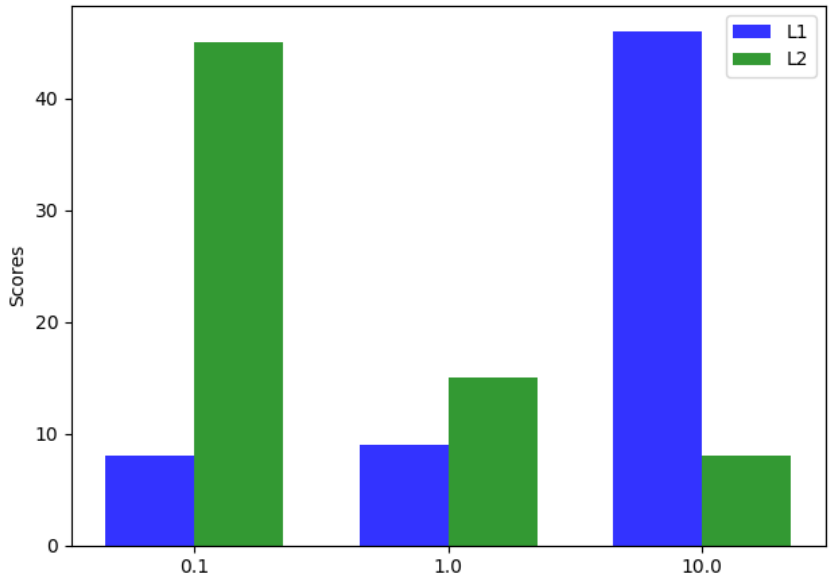
```
plt.show()

Output:
1000
100
Train on 900 samples, validate on 100 samples
Training accuracy: 99.00%
Test accuracy: 84.00%
5000
500
Train on 4500 samples, validate on 500 samples
Training accuracy: 95.26%
Test accuracy: 83.80%
```

3. With a 40 hidden neuron network, compare the accuracy results between L1 and L2 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}

Training Scores for L1 and L2



Testing Scores for L1 and L2




```
import tensorflow as tf
import tensorflow.contrib.keras as keras
import numpy as np

# importing mnist dataset
from tensorflow.examples.tutorials.mnist import input_data
```

```
epochsNo = 300
mini_batch_size = 10
eta = 0.5
trainingSample = 1000
testingSample = 100
noOfNeuron = 40

lmbda = [0.1, 1, 10]

mnist = input_data.read_data_sets("/tmp/data/", one_hot=False)
X_train = mnist.train.images[:trainingSample]
y_train = mnist.train.labels[:trainingSample]
X_test = mnist.test.images[:testingSample]
y_test = mnist.test.labels[:testingSample]

print(len(X_train))
print(len(X_test))

## mean centering and normalization:
mean_vals = np.mean(X_train, axis=0)
std_val = np.std(X_train)
X_train_centered = (X_train - mean_vals) / std_val
X_test_centered = (X_test - mean_vals) / std_val
del X_train, X_test
from tensorflow.contrib.keras import regularizers
# makinf keras model
y_train_onehot = keras.utils.to_categorical(y_train)
model = keras.models.Sequential()
model.add(
    keras.layers.Dense(
        units=noOfNeuron,
        input_dim=X_train_centered.shape[1],
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        kernel_regularizer=regularizers.l1(lmbda[0]),
        activation='tanh'))

model.add(
    keras.layers.Dense(
        units=noOfNeuron,
        input_dim=noOfNeuron,
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        kernel_regularizer=regularizers.l1(lmbda[0]),
        activation='tanh'))

model.add(
    keras.layers.Dense(
        units=y_train_onehot.shape[1],
        input_dim=noOfNeuron,
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        kernel_regularizer=regularizers.l1(lmbda[0]),
        activation='softmax'))

# declare the optimizer and cost function
sgd_optimizer = keras.optimizers.SGD(lr=eta)
model.compile(optimizer=sgd_optimizer, loss='categorical_crossentropy')
history = model.fit(X_train_centered, y_train_onehot,
                    batch_size=mini_batch_size, epochs=epochsNo,
                    verbose=1,
                    validation_split=0.1)

# checking accuracy on training and testing dataset
y_train_pred = model.predict_classes(X_train_centered, verbose=0)

correct_preds = np.sum(y_train == y_train_pred, axis=0)
train_acc = correct_preds / y_train.shape[0]
print('Training accuracy L1 Lamda = 0.1: %.2f%%' % (train_acc * 100))

y_test_pred = model.predict_classes(X_test_centered, verbose=0)
correct_preds = np.sum(y_test == y_test_pred, axis=0)
test_acc = correct_preds / y_test.shape[0]
print('Test accuracy L1 Lamda = 0.1: %.2f%%' % (test_acc * 100))


epochsNo = 300
mini_batch_size = 10
eta = 0.5
trainingSample = 1000
testingSample = 100
noOfNeuron = 40

lmbda = [0.1, 1, 10]

mnist = input_data.read_data_sets("/tmp/data/", one_hot=False)
X_train = mnist.train.images[:trainingSample]
y_train = mnist.train.labels[:trainingSample]
X_test = mnist.test.images[:testingSample]
y_test = mnist.test.labels[:testingSample]
```

```
print(len(X_train))
print(len(X_test))

## mean centering and normalization:
mean_vals = np.mean(X_train, axis=0)
std_val = np.std(X_train)
X_train_centered = (X_train - mean_vals) / std_val
X_test_centered = (X_test - mean_vals) / std_val
del X_train, X_test
from tensorflow.contrib.keras import regularizers
# makinf keras model
y_train_onehot = keras.utils.to_categorical(y_train)
model = keras.models.Sequential()
model.add(
    keras.layers.Dense(
        units=noOfNeuron,
        input_dim=X_train_centered.shape[1],
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        kernel_regularizer=regularizers.l1(lmbda[1]),
        activation='tanh'))

model.add(
    keras.layers.Dense(
        units=noOfNeuron,
        input_dim=noOfNeuron,
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        kernel_regularizer=regularizers.l1(lmbda[1]),
        activation='tanh'))

model.add(
    keras.layers.Dense(
        units=y_train_onehot.shape[1],
        input_dim=noOfNeuron,
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        kernel_regularizer=regularizers.l1(lmbda[1]),
        activation='softmax'))

# declare the optimizer and cost function
sgd_optimizer = keras.optimizers.SGD(lr=eta)
model.compile(optimizer=sgd_optimizer, loss='categorical_crossentropy')
history = model.fit(X_train_centered, y_train_onehot,
                    batch_size=mini_batch_size, epochs=epochsNo,
                    verbose=1,
                    validation_split=0.1)

# checking accuracy on training and testing dataset
y_train_pred = model.predict_classes(X_train_centered, verbose=0)

correct_preds = np.sum(y_train == y_train_pred, axis=0)
train_acc = correct_preds / y_train.shape[0]
print('Training accuracy L1 Lamda = 1: %.2f%%' % (train_acc * 100))

y_test_pred = model.predict_classes(X_test_centered, verbose=0)
correct_preds = np.sum(y_test == y_test_pred, axis=0)
test_acc = correct_preds / y_test.shape[0]
print('Test accuracy L1 Lamda = 1: %.2f%%' % (test_acc * 100))




epochsNo = 300
mini_batch_size = 10
eta = 0.5
trainingSample = 1000
testingSample = 100
noOfNeuron = 40

lmbda = [0.1, 1, 10]

mnist = input_data.read_data_sets("/tmp/data/", one_hot=False)
X_train = mnist.train.images[:trainingSample]
y_train = mnist.train.labels[:trainingSample]
X_test = mnist.test.images[:testingSample]
y_test = mnist.test.labels[:testingSample]

print(len(X_train))
print(len(X_test))

## mean centering and normalization:
mean_vals = np.mean(X_train, axis=0)
std_val = np.std(X_train)
X_train_centered = (X_train - mean_vals) / std_val
X_test_centered = (X_test - mean_vals) / std_val
del X_train, X_test
from tensorflow.contrib.keras import regularizers
# makinf keras model
y_train_onehot = keras.utils.to_categorical(y_train)
```

```
model = keras.models.Sequential()
model.add(
    keras.layers.Dense(
        units=noOfNeuron,
        input_dim=X_train_centered.shape[1],
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        kernel_regularizer=regularizers.l1(lmbda[2]),
        activation='tanh'))

model.add(
    keras.layers.Dense(
        units=noOfNeuron,
        input_dim=noOfNeuron,
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        kernel_regularizer=regularizers.l1(lmbda[2]),
        activation='tanh'))

model.add(
    keras.layers.Dense(
        units=y_train_onehot.shape[1],
        input_dim=noOfNeuron,
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        kernel_regularizer=regularizers.l1(lmbda[2]),
        activation='softmax'))

# declare the optimizer and cost function
sgd_optimizer = keras.optimizers.SGD(lr=eta)
model.compile(optimizer=sgd_optimizer, loss='categorical_crossentropy')
history = model.fit(X_train_centered, y_train_onehot,
                    batch_size=mini_batch_size, epochs=epochsNo,
                    verbose=1,
                    validation_split=0.1)

# checking accuracy on training and testing dataset
y_train_pred = model.predict_classes(X_train_centered, verbose=0)

correct_preds = np.sum(y_train == y_train_pred, axis=0)
train_acc = correct_preds / y_train.shape[0]
print('Training accuracy L1 Lamda = 10: %.2f%%' % (train_acc * 100))

y_test_pred = model.predict_classes(X_test_centered, verbose=0)
correct_preds = np.sum(y_test == y_test_pred, axis=0)
test_acc = correct_preds / y_test.shape[0]
print('Test accuracy L1 Lamda = 10: %.2f%%' % (test_acc * 100))


epochsNo = 300
mini_batch_size = 10
eta = 0.5
trainingSample = 1000
testingSample = 100
noOfNeuron = 40

lmbda = [0.1, 1, 10]

mnist = input_data.read_data_sets("/tmp/data/", one_hot=False)
X_train = mnist.train.images[:trainingSample]
y_train = mnist.train.labels[:trainingSample]
X_test = mnist.test.images[:testingSample]
y_test = mnist.test.labels[:testingSample]

print(len(X_train))
print(len(X_test))

## mean centering and normalization:
mean_vals = np.mean(X_train, axis=0)
std_val = np.std(X_train)
X_train_centered = (X_train - mean_vals) / std_val
X_test_centered = (X_test - mean_vals) / std_val
del X_train, X_test
from tensorflow.contrib.keras import regularizers
# makinf keras model
y_train_onehot = keras.utils.to_categorical(y_train)
model = keras.models.Sequential()
model.add(
    keras.layers.Dense(
        units=noOfNeuron,
        input_dim=X_train_centered.shape[1],
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        kernel_regularizer=regularizers.l2(lmbda[0]),
        activation='tanh'))

model.add(
    keras.layers.Dense(
        units=noOfNeuron,
        input_dim=noOfNeuron,
        kernel_initializer='glorot_uniform',
```

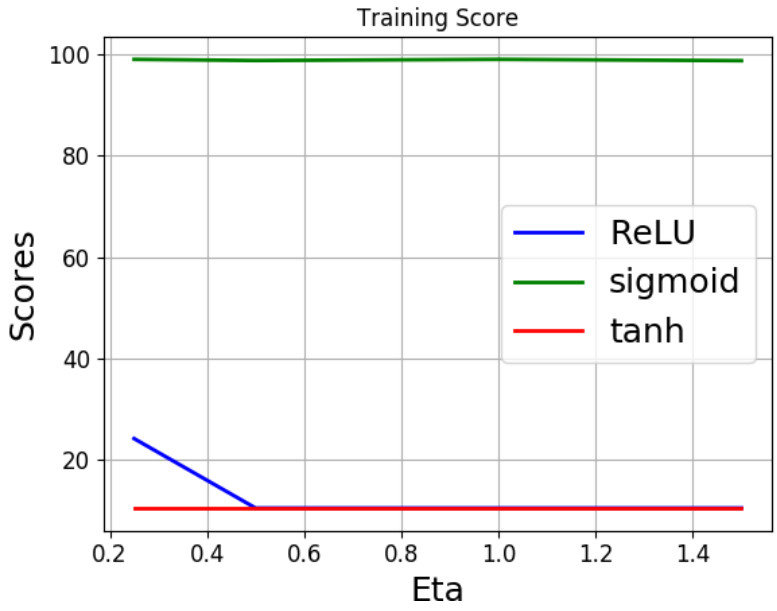
```
        bias_initializer='zeros',
        kernel_regularizer=regularizers.l2(lmbda[0]),
        activation='tanh'))

model.add(
    keras.layers.Dense(
        units=y_train_onehot.shape[1],
        input_dim=noOfNeuron,
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        kernel_regularizer=regularizers.l2(lmbda[0]),
        activation='softmax'))

# declare the optimizer and cost function
sgd_optimizer = keras.optimizers.SGD(lr=eta)
model.compile(optimizer=sgd_optimizer, loss='categorical_crossentropy')
history = model.fit(X_train_centered, y_train_onehot,
                    batch_size=mini_batch_size, epochs=epochsNo,
                    verbose=1,
                    validation_split=0.1)

# checking accuracy on training and testing dataset
y_train_pred = model.predict_classes(X_train_centered, verbose=0)

correct_preds = np.sum(y_train == y_train_pred, axis=0)
train_acc = correct_preds / y_train.shape[0]
print('Training accuracy L2 Lamda = 0.1: %.2f%%' % (train_acc * 100))

y_test_pred = model.predict_classes(X_test_centered, verbose=0)
correct_preds = np.sum(y_test == y_test_pred, axis=0)
test_acc = correct_preds / y_test.shape[0]
print('Test accuracy L2 Lamda = 0.1: %.2f%%' % (test_acc * 100))


epochsNo = 300
mini_batch_size = 10
eta = 0.5
trainingSample = 1000
testingSample = 100
noOfNeuron = 40

lmbda = [0.1, 1, 10]

mnist = input_data.read_data_sets("/tmp/data/", one_hot=False)
X_train = mnist.train.images[:trainingSample]
y_train = mnist.train.labels[:trainingSample]
X_test = mnist.test.images[:testingSample]
y_test = mnist.test.labels[:testingSample]

print(len(X_train))
print(len(X_test))

## mean centering and normalization:
mean_vals = np.mean(X_train, axis=0)
std_val = np.std(X_train)
X_train_centered = (X_train - mean_vals) / std_val
X_test_centered = (X_test - mean_vals) / std_val
del X_train, X_test
from tensorflow.contrib.keras import regularizers
# makinf keras model
y_train_onehot = keras.utils.to_categorical(y_train)
model = keras.models.Sequential()
model.add(
    keras.layers.Dense(
        units=noOfNeuron,
        input_dim=X_train_centered.shape[1],
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        kernel_regularizer=regularizers.l2(lmbda[1]),
        activation='tanh'))

model.add(
    keras.layers.Dense(
        units=noOfNeuron,
        input_dim=noOfNeuron,
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        kernel_regularizer=regularizers.l2(lmbda[1]),
        activation='tanh'))

model.add(
    keras.layers.Dense(
        units=y_train_onehot.shape[1],
        input_dim=noOfNeuron,
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        kernel_regularizer=regularizers.l2(lmbda[1]),
        activation='softmax'))

# declare the optimizer and cost function
sgd_optimizer = keras.optimizers.SGD(lr=eta)
```

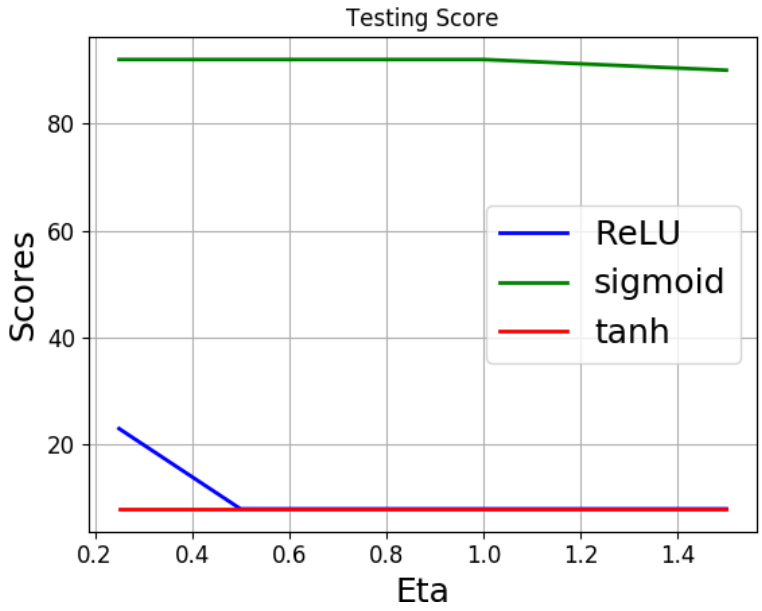
```
model.compile(optimizer=sgd_optimizer, loss='categorical_crossentropy')
history = model.fit(X_train_centered, y_train_onehot,
                    batch_size=mini_batch_size, epochs=epochsNo,
                    verbose=1,
                    validation_split=0.1)

# checking accuracy on training and testing dataset
y_train_pred = model.predict_classes(X_train_centered, verbose=0)

correct_preds = np.sum(y_train == y_train_pred, axis=0)
train_acc = correct_preds / y_train.shape[0]
print('Training accuracy L2 Lamda = 1: %.2f%%' % (train_acc * 100))

y_test_pred = model.predict_classes(X_test_centered, verbose=0)
correct_preds = np.sum(y_test == y_test_pred, axis=0)
test_acc = correct_preds / y_test.shape[0]
print('Test accuracy L2 Lamda = 1: %.2f%%' % (test_acc * 100))




epochsNo = 300
mini_batch_size = 10
eta = 0.5
trainingSample = 1000
testingSample = 100
noOfNeuron = 40

lmbda = [0.1, 1, 10]

mnist = input_data.read_data_sets("/tmp/data/", one_hot=False)
X_train = mnist.train.images[:trainingSample]
y_train = mnist.train.labels[:trainingSample]
X_test = mnist.test.images[:testingSample]
y_test = mnist.test.labels[:testingSample]

print(len(X_train))
print(len(X_test))

## mean centering and normalization:
mean_vals = np.mean(X_train, axis=0)
std_val = np.std(X_train)
X_train_centered = (X_train - mean_vals) / std_val
X_test_centered = (X_test - mean_vals) / std_val
del X_train, X_test
from tensorflow.contrib.keras import regularizers
# makinf keras model
y_train_onehot = keras.utils.to_categorical(y_train)
model = keras.models.Sequential()
model.add(
    keras.layers.Dense(
        units=noOfNeuron,
        input_dim=X_train_centered.shape[1],
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        kernel_regularizer=regularizers.l2(lmbda[2]),
        activation='tanh'))

model.add(
    keras.layers.Dense(
        units=noOfNeuron,
        input_dim=noOfNeuron,
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        kernel_regularizer=regularizers.l2(lmbda[2]),
        activation='tanh'))

model.add(
    keras.layers.Dense(
        units=y_train_onehot.shape[1],
        input_dim=noOfNeuron,
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        kernel_regularizer=regularizers.l2(lmbda[2]),
        activation='softmax'))

# declare the optimizer and cost function
sgd_optimizer = keras.optimizers.SGD(lr=eta)
model.compile(optimizer=sgd_optimizer, loss='categorical_crossentropy')
history = model.fit(X_train_centered, y_train_onehot,
                    batch_size=mini_batch_size, epochs=epochsNo,
                    verbose=1,
                    validation_split=0.1)

# checking accuracy on training and testing dataset
y_train_pred = model.predict_classes(X_train_centered, verbose=0)

correct_preds = np.sum(y_train == y_train_pred, axis=0)
train_acc = correct_preds / y_train.shape[0]
print('Training accuracy L2 Lamda = 10: %.2f%%' % (train_acc * 100))
```

```
y_test_pred = model.predict_classes(X_test_centered, verbose=0)
correct_preds = np.sum(y_test == y_test_pred, axis=0)
test_acc = correct_preds / y_test.shape[0]
print('Test accuracy L2 Lamda = 10: %.2f%%' % (test_acc * 100))

import numpy as np
import matplotlib.pyplot as plt

# data to plot
n_groups = 3

L1 = (11.30, 11.60,52.10)
L2 = (40.60, 10.10, 11.30)

# create plot
fig, ax = plt.subplots()
index = np.arange(n_groups)
bar_width = 0.35
opacity = 0.8

rects1 = plt.bar(index, L1, bar_width,
                 alpha=opacity,
                 color='b',
                 label='L1')

rects2 = plt.bar(index + bar_width, L2, bar_width,
                 alpha=opacity,
                 color='g',
                 label='L2')

# plt.xlabel('# of Neuron')
plt.ylabel('Scores')
plt.title('Scores L1 vs. L2 Training')
plt.xticks((index + bar_width/2.0), ('0.1', '1.0','10.0'))
plt.legend()

plt.tight_layout()
plt.show()




# data to plot
n_groups = 3
#
L1 = (8.00, 9.00,46.00)
L2 = (45.00, 15.00, 8.00)

# create plot
fig, ax = plt.subplots()
index = np.arange(n_groups)
bar_width = 0.35
opacity = 0.8

rects1 = plt.bar(index, L1, bar_width,
                 alpha=opacity,
                 color='b',
                 label='L1')

rects2 = plt.bar(index + bar_width, L2, bar_width,
                 alpha=opacity,
                 color='g',
                 label='L2')

# plt.xlabel('# of Neuron')
plt.ylabel('Scores')
plt.title('Scores L1 vs. L2 Testing')
plt.xticks((index + bar_width/2.0), ('0.1', '1.0','10.0'))
plt.legend()

plt.tight_layout()
plt.show()

Output:
1000
100
Train on 900 samples, validate on 100 samples
Training accuracy L1 Lamda = 0.1: 11.30%
Test accuracy L1 Lamda = 0.1: 8.00%
Training accuracy L1 Lamda = 1: 11.60%
Test accuracy L1 Lamda = 1: 9.00%
Training accuracy L1 Lamda = 10: 52.10%
Test accuracy L1 Lamda = 10: 46.00%
Training accuracy L2 Lamda = 0.1: 40.60%
Test accuracy L2 Lamda = 0.1: 45.00%
Training accuracy L2 Lamda = 1: 10.10%
Test accuracy L2 Lamda = 1: 15.00%
Training accuracy L2 Lamda = 10: 11.30%
Test accuracy L2 Lamda = 10: 8.00%
```

4. Comparing different activation functions: ReLU, sigmoid, tanh, by using 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 of compared methods.

The best result is learning rate 1 for sigmoid activation function.






```
import tensorflow as tf
import tensorflow.contrib.keras as keras
import numpy as np
import matplotlib.pyplot as plt

# importing mnist dataset
from tensorflow.examples.tutorials.mnist import input_data

epochsNo = 300
mini_batch_size = 10
eta = 0.5
trainingSample = 3000
testingSample = 300
noOfNeuron = 40

mnist = input_data.read_data_sets("/tmp/data/", one_hot=False)
X_train = mnist.train.images[:trainingSample]
y_train = mnist.train.labels[:trainingSample]
X_test = mnist.test.images[:testingSample]
y_test = mnist.test.labels[:testingSample]

print(len(X_train))
```

```
print(len(X_test))

## mean centering and normalization:
mean_vals = np.mean(X_train, axis=0)
std_val = np.std(X_train)
X_train_centered = (X_train - mean_vals) / std_val
X_test_centered = (X_test - mean_vals) / std_val
del X_train, X_test

act_fn = ['relu', 'tanh', 'sigmoid']
color = ['b', 'r', 'g']

for ite in range(3):
    y_train_onehot = keras.utils.to_categorical(y_train)
    model = keras.models.Sequential()
    model.add(
        keras.layers.Dense(
            units=40,
            input_dim=X_train_centered.shape[1],
            kernel_initializer='glorot_uniform',
            bias_initializer='zeros',
            activation=act_fn[ite]))
    model.add(
        keras.layers.Dense(
            units=40,
            input_dim=40,
            kernel_initializer='glorot_uniform',
            bias_initializer='zeros',
            activation=act_fn[ite]))
    model.add(
        keras.layers.Dense(
            units=y_train_onehot.shape[1],
            input_dim=40,
            kernel_initializer='glorot_uniform',
            bias_initializer='zeros',
            activation=act_fn[ite]))
    # declare the optimizer and cost function
    sgd_optimizer = keras.optimizers.SGD(lr=eta)
    model.compile(optimizer=sgd_optimizer, loss='categorical_crossentropy')
    history = model.fit(X_train_centered, y_train_onehot,
                        batch_size=mini_batch_size, epochs=epochsNo,
                        verbose=0,
                        validation_split=0.1)

    # checking accuracy on training and testing dataset
    y_train_pred = model.predict_classes(X_train_centered, verbose=0)
    correct_preds = np.sum(y_train == y_train_pred, axis=0)
    train_acc = correct_preds / y_train.shape[0]
    print('Training accuracy for eta = 0.5  %s is: %.2f%%' % (act_fn[ite], train_acc * 100))

    y_test_pred = model.predict_classes(X_test_centered, verbose=0)
    correct_preds = np.sum(y_test == y_test_pred, axis=0)
    test_acc = correct_preds / y_test.shape[0]
    print('Test accuracy for eta = 0.5 %s is: %.2f%%' % (act_fn[ite], test_acc * 100))



epochsNo = 3000
mini_batch_size = 10
eta = 1
trainingSample = 3000
testingSample = 300
noOfNeuron = 40

mnist = input_data.read_data_sets("/tmp/data/", one_hot=False)
X_train = mnist.train.images[:trainingSample]
y_train = mnist.train.labels[:trainingSample]
X_test = mnist.test.images[:testingSample]
y_test = mnist.test.labels[:testingSample]

print(len(X_train))
print(len(X_test))

## mean centering and normalization:
mean_vals = np.mean(X_train, axis=0)
std_val = np.std(X_train)
X_train_centered = (X_train - mean_vals) / std_val
X_test_centered = (X_test - mean_vals) / std_val
del X_train, X_test

act_fn = ['relu', 'tanh', 'sigmoid']
color = ['b', 'r', 'g']

for ite in range(3):
    y_train_onehot = keras.utils.to_categorical(y_train)
    model = keras.models.Sequential()
    model.add(
        keras.layers.Dense(
            units=40,
            input_dim=X_train_centered.shape[1],
            kernel_initializer='glorot_uniform',
```

```
            bias_initializer='zeros',
            activation=act_fn[ite]))
    model.add(
        keras.layers.Dense(
            units=40,
            input_dim=40,
            kernel_initializer='glorot_uniform',
            bias_initializer='zeros',
            activation=act_fn[ite]))
    model.add(
        keras.layers.Dense(
            units=y_train_onehot.shape[1],
            input_dim=40,
            kernel_initializer='glorot_uniform',
            bias_initializer='zeros',
            activation=act_fn[ite]))
    # declare the optimizer and cost function
    sgd_optimizer = keras.optimizers.SGD(lr=eta)
    model.compile(optimizer=sgd_optimizer, loss='categorical_crossentropy')
    history = model.fit(X_train_centered, y_train_onehot,
                        batch_size=mini_batch_size, epochs=epochsNo,
                        verbose=0,
                        validation_split=0.1)

    # checking accuracy on training and testing dataset
    y_train_pred = model.predict_classes(X_train_centered, verbose=0)
    correct_preds = np.sum(y_train == y_train_pred, axis=0)
    train_acc = correct_preds / y_train.shape[0]
    print('Training accuracy for eta = 1  %s is: %.2f%%' % (act_fn[ite], train_acc * 100))

    y_test_pred = model.predict_classes(X_test_centered, verbose=0)
    correct_preds = np.sum(y_test == y_test_pred, axis=0)
    test_acc = correct_preds / y_test.shape[0]
    print('Test accuracy for eta = 1 %s is: %.2f%%' % (act_fn[ite], test_acc * 100))




epochsNo = 300
mini_batch_size = 10
eta = 0.25
trainingSample = 3000
testingSample = 300
noOfNeuron = 40

mnist = input_data.read_data_sets("/tmp/data/", one_hot=False)
X_train = mnist.train.images[:trainingSample]
y_train = mnist.train.labels[:trainingSample]
X_test = mnist.test.images[:testingSample]
y_test = mnist.test.labels[:testingSample]

print(len(X_train))
print(len(X_test))

## mean centering and normalization:
mean_vals = np.mean(X_train, axis=0)
std_val = np.std(X_train)
X_train_centered = (X_train - mean_vals) / std_val
X_test_centered = (X_test - mean_vals) / std_val
del X_train, X_test

act_fn = ['relu', 'tanh', 'sigmoid']
color = ['b', 'r', 'g']

for ite in range(3):
    y_train_onehot = keras.utils.to_categorical(y_train)
    model = keras.models.Sequential()
    model.add(
        keras.layers.Dense(
            units=40,
            input_dim=X_train_centered.shape[1],
            kernel_initializer='glorot_uniform',
            bias_initializer='zeros',
            activation=act_fn[ite]))
    model.add(
        keras.layers.Dense(
            units=40,
            input_dim=40,
            kernel_initializer='glorot_uniform',
            bias_initializer='zeros',
            activation=act_fn[ite]))
    model.add(
        keras.layers.Dense(
            units=y_train_onehot.shape[1],
            input_dim=40,
            kernel_initializer='glorot_uniform',
            bias_initializer='zeros',
            activation=act_fn[ite]))
    # declare the optimizer and cost function
    sgd_optimizer = keras.optimizers.SGD(lr=eta)
    model.compile(optimizer=sgd_optimizer, loss='categorical_crossentropy')
```

```
    history = model.fit(X_train_centered, y_train_onehot,
                        batch_size=mini_batch_size, epochs=epochsNo,
                        verbose=0,
                        validation_split=0.1)

    # checking accuracy on training and testing dataset
    y_train_pred = model.predict_classes(X_train_centered, verbose=0)
    correct_preds = np.sum(y_train == y_train_pred, axis=0)
    train_acc = correct_preds / y_train.shape[0]
    print('Training accuracy for eta = 0.25  %s is: %.2f%%' % (act_fn[ite], train_acc * 100))

    y_test_pred = model.predict_classes(X_test_centered, verbose=0)
    correct_preds = np.sum(y_test == y_test_pred, axis=0)
    test_acc = correct_preds / y_test.shape[0]
    print('Test accuracy for eta = 0.25 %s is: %.2f%%' % (act_fn[ite], test_acc * 100))



epochsNo = 300
mini_batch_size = 10
eta = 1.5
trainingSample = 3000
testingSample = 300
noOfNeuron = 40

mnist = input_data.read_data_sets("/tmp/data/", one_hot=False)
X_train = mnist.train.images[:trainingSample]
y_train = mnist.train.labels[:trainingSample]
X_test = mnist.test.images[:testingSample]
y_test = mnist.test.labels[:testingSample]

print(len(X_train))
print(len(X_test))

## mean centering and normalization:
mean_vals = np.mean(X_train, axis=0)
std_val = np.std(X_train)
X_train_centered = (X_train - mean_vals) / std_val
X_test_centered = (X_test - mean_vals) / std_val
del X_train, X_test

act_fn = ['relu', 'tanh', 'sigmoid']
color = ['b', 'r', 'g']

for ite in range(3):
    y_train_onehot = keras.utils.to_categorical(y_train)
    model = keras.models.Sequential()
    model.add(
        keras.layers.Dense(
            units=40,
            input_dim=X_train_centered.shape[1],
            kernel_initializer='glorot_uniform',
            bias_initializer='zeros',
            activation=act_fn[ite]))
    model.add(
        keras.layers.Dense(
            units=40,
            input_dim=40,
            kernel_initializer='glorot_uniform',
            bias_initializer='zeros',
            activation=act_fn[ite]))
    model.add(
        keras.layers.Dense(
            units=y_train_onehot.shape[1],
            input_dim=40,
            kernel_initializer='glorot_uniform',
            bias_initializer='zeros',
            activation=act_fn[ite]))
    # declare the optimizer and cost function
    sgd_optimizer = keras.optimizers.SGD(lr=eta)
    model.compile(optimizer=sgd_optimizer, loss='categorical_crossentropy')
    history = model.fit(X_train_centered, y_train_onehot,
                        batch_size=mini_batch_size, epochs=epochsNo,
                        verbose=0,
                        validation_split=0.1)

    # checking accuracy on training and testing dataset
    y_train_pred = model.predict_classes(X_train_centered, verbose=0)
    correct_preds = np.sum(y_train == y_train_pred, axis=0)
    train_acc = correct_preds / y_train.shape[0]
    print('Training accuracy for eta = 1.5  %s is: %.2f%%' % (act_fn[ite], train_acc * 100))

    y_test_pred = model.predict_classes(X_test_centered, verbose=0)
    correct_preds = np.sum(y_test == y_test_pred, axis=0)
    test_acc = correct_preds / y_test.shape[0]
    print('Test accuracy for eta = 1.5 %s is: %.2f%%' % (act_fn[ite], test_acc * 100))

import numpy as np
import matplotlib.pyplot as plt

ReLUTraining = [24.23,10.53,10.53,10.53]
```

```
tanhTraining = [10.53,10.53,10.53,10.53]
sigmoidTraining = [99.03,98.80,99.03,98.77]
eta = [0.25,0.5,1,1.5]
# eta = [0.5,1,0.25]

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(eta, ReLUTraining, color='b',lw=2.0, label='ReLU')
ax.plot(eta,sigmoidTraining, color='g',  lw=2.0, label='sigmoid')
ax.plot(eta,tanhTraining, color='r',  lw=2.0, label='tanh')
plt.title("Training Score")
plt.ylabel('Scores', fontsize = 18)
plt.xlabel('Eta', 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()


ReLUTesting = [23.00,8.00,8.00,8.00]
tanhTesting = [8.00,8.00,8.00,8.00]
sigmoidTesting = [92.00,92.00,92.00,90.00]

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( eta, ReLUTesting, color='b',lw=2.0, label='ReLU')
ax.plot(eta, sigmoidTesting, color='g',  lw=2.0, label='sigmoid')
ax.plot(eta, tanhTesting, color='r',  lw=2.0, label='tanh')
plt.title("Testing Score")
plt.ylabel('Scores', fontsize = 18)
plt.xlabel('Eta', 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:
3000
300
Training accuracy for eta = 0.5  relu is: 10.53%
Test accuracy for eta = 0.5 relu is: 8.00%
Training accuracy for eta = 0.5  tanh is: 10.53%
Test accuracy for eta = 0.5 tanh is: 8.00%
Training accuracy for eta = 0.5  sigmoid is: 98.80%
Test accuracy for eta = 0.5 sigmoid is: 92.00%
3000
300
Training accuracy for eta = 1  relu is: 10.53%
Test accuracy for eta = 1 relu is: 8.00%
Training accuracy for eta = 1  tanh is: 10.53%
Test accuracy for eta = 1 tanh is: 8.00%
Training accuracy for eta = 1  sigmoid is: 99.03%
Test accuracy for eta = 1 sigmoid is: 92.00%
3000
300
Training accuracy for eta = 0.25  relu is: 24.23%
Test accuracy for eta = 0.25 relu is: 23.00%
```

```
Training accuracy for eta = 0.25  tanh is: 10.53%
Test accuracy for eta = 0.25 tanh is: 8.00%
Training accuracy for eta = 0.25  sigmoid is: 99.03%
Test accuracy for eta = 0.25 sigmoid is: 92.00%
3000
300
Training accuracy for eta = 1.5  relu is: 10.53%
Test accuracy for eta = 1.5 relu is: 8.00%
Training accuracy for eta = 1.5  tanh is: 10.53%
Test accuracy for eta = 1.5 tanh is: 8.00%
Training accuracy for eta = 1.5  sigmoid is: 98.77%
Test accuracy for eta = 1.5 sigmoid is: 90.00%
```

hw7

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*