```
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
```

```
Using TensorFlow backend.
```

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Test Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

```
# the data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

```
print("Number of training examples :", X_train.shape[0], "and each image is of shape (%d, %d)"%(X
print("Number of testing examples :", X_test.shape[0], "and each image is of shape (%d, %d)"%(X_t
```

```
Number of training examples : 60000 and each image is of shape (28, 28)
Number of testing examples : 10000 and each image is of shape (28, 28)
```

```
# if you observe the input shape its 2 dimensional vector
# for each image we have a (28*28) vector
# we will convert the (28*28) vector into single dimensional vector of 1 * 784

X_train = X_train.reshape(X_train.shape[0], X_train.shape[1]*X_train.shape[2])
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1]*X_test.shape[2])
```

```
# after converting the input images from 3d to 2d vectors

print("Number of training examples :", X_train.shape[0], "and each image is of shape (%d)"%(X_tra
print("Number of testing examples :", X_test.shape[0], "and each image is of shape (%d)"%(X_test.
```

```
Number of training examples : 60000 and each image is of shape (784)
Number of testing examples : 10000 and each image is of shape (784)
```

```
# An example data point
print(X_train[0])
```

```
```

```
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 255
 247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  36  94 154
 170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0   0   0
   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82
  82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 253
 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  35 241
 225 160 108   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
 253 207   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253
 253 201  78   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0  18 171 219 253 253 253 253 195
  80   9   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0 136 253 253 253 212 135 132  16
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

```
# if we observe the above matrix each cell is having a value between 0-255
# before we move to apply machine learning algorithms lets try to normalize the data
# X => (X - Xmin)/(Xmax-Xmin) = X/255

X_train = X_train/255
X_test = X_test/255


# example data point after normlizing
print(X_train[0])
```

```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.01176471 0.07058824 0.07058824 0.07058824
 0.49411765 0.53333333 0.68627451 0.10196078 0.65098039 1.
 0.96862745 0.49803922 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.11764706 0.14117647 0.36862745 0.60392157
 0.66666667 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.88235294 0.6745098  0.99215686 0.94901961 0.76470588 0.25098039
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.19215686
 0.93333333 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.99215686 0.99215686 0.99215686 0.98431373 0.36470588 0.32156863
 0.32156863 0.21960784 0.15294118 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.07058824 0.85882353 0.99215686
 0.99215686 0.99215686 0.99215686 0.99215686 0.77647059 0.71372549
 0.96862745 0.94509804 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.31372549 0.61176471 0.41960784 0.99215686
 0.99215686 0.80392157 0.04313725 0.         0.16862745 0.60392157
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.05490196 0.00392157 0.60392157 0.99215686 0.35294118
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.54509804 0.99215686 0.74509804 0.00784314 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.04313725
 0.74509804 0.99215686 0.2745098  0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
```

```
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.1372549  0.94509804
0.88235294 0.62745098 0.42352941 0.00392157 0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.31764706 0.94117647 0.99215686
0.99215686 0.46666667 0.09803922 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.17647059 0.72941176 0.99215686 0.99215686
0.58823529 0.10588235 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.0627451  0.36470588 0.98823529 0.99215686 0.73333333
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.97647059 0.99215686 0.97647059 0.25098039 0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.18039216 0.50980392 0.71764706 0.99215686
0.99215686 0.81176471 0.00784314 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.15294118 0.58039216
0.89803922 0.99215686 0.99215686 0.99215686 0.98039216 0.71372549
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.09411765 0.44705882 0.86666667 0.99215686 0.99215686 0.99215686
0.99215686 0.78823529 0.30588235 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.09019608 0.25882353 0.83529412 0.99215686
0.99215686 0.99215686 0.99215686 0.77647059 0.31764706 0.00784314
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.07058824 0.67058824
0.85882353 0.99215686 0.99215686 0.99215686 0.99215686 0.76470588
0.31372549 0.03529412 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.21568627 0.6745098  0.88627451 0.99215686 0.99215686 0.99215686
0.99215686 0.95686275 0.52156863 0.04313725 0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.53333333 0.99215686
0.99215686 0.99215686 0.83137255 0.52941176 0.51764706 0.0627451
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
```

```
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
```

```
# here we are having a class number for each image
print("Class label of first image :", y_train[0])

# lets convert this into a 10 dimensional vector
# ex: consider an image is 5 convert it into 5 => [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
# this conversion needed for MLPs

Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)

print("After converting the output into a vector : ",Y_train[0])
```

```
Class label of first image : 5
After converting the output into a vector :  [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

MLP + Dropout + AdamOptimizer

1.Two hidden layers

```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-function-in

from keras.models import Sequential
from keras.layers import Dense, Activation

from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization

model = Sequential()

model.add(Dense(435, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(156, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, s
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(output_dim, activation='softmax'))


model.summary()
```

```
WARNING: Logging before flag parsing goes to stderr.
W0825 20:34:50.848045 140160636204928 deprecation_wrapper.py:119] From /usr/local/lib

W0825 20:34:50.882638 140160636204928 deprecation_wrapper.py:119] From /usr/local/lib

W0825 20:34:50.890069 140160636204928 deprecation_wrapper.py:119] From /usr/local/lib

W0825 20:34:50.998440 140160636204928 deprecation_wrapper.py:119] From /usr/local/lib

W0825 20:34:51.022265 140160636204928 deprecation.py:506] From /usr/local/lib/python3
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob
W0825 20:34:51.133279 140160636204928 deprecation_wrapper.py:119] From /usr/local/lib
```

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 435)	341475
batch_normalization_1 (Batch	(None, 435)	1740
dropout_1 (Dropout)	(None, 435)	0
dense_2 (Dense)	(None, 156)	68016
batch_normalization_2 (Batch	(None, 156)	624
dropout_2 (Dropout)	(None, 156)	0
dense_3 (Dense)	(None, 10)	1570

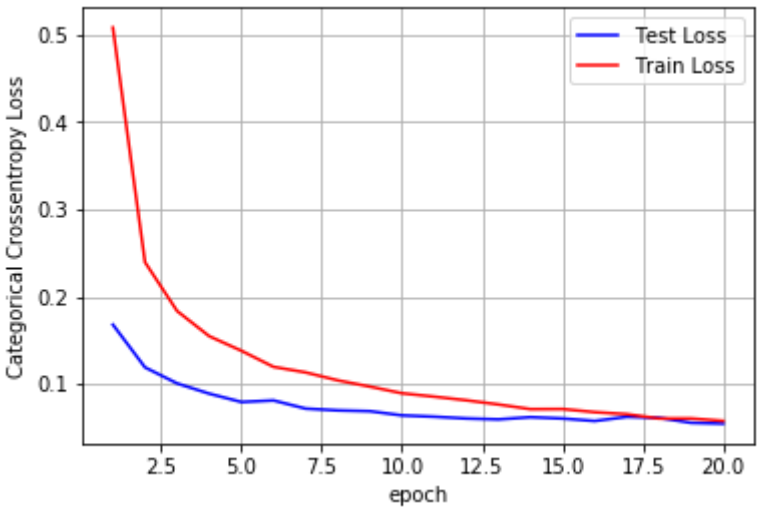
Total params: 413,425

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validati
```

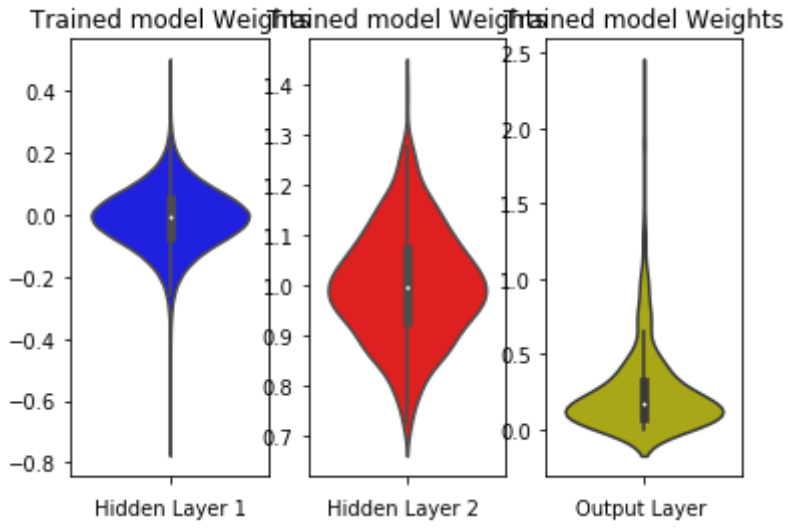
```
W0825 20:34:56.736942 140160636204928 deprecation_wrapper.py:119] From /usr/local/lib

W0825 20:34:56.864316 140160636204928 deprecation.py:323] From /usr/local/lib/python3
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.5083 - acc: 0.
Epoch 2/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.2395 - acc: 0.
Epoch 3/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.1836 - acc: 0.
Epoch 4/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.1547 - acc: 0.
Epoch 5/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.1381 - acc: 0.
Epoch 6/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.1196 - acc: 0.
Epoch 7/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.1133 - acc: 0.
Epoch 8/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.1042 - acc: 0.
Epoch 9/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.0970 - acc: 0.
Epoch 10/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.0894 - acc: 0.
Epoch 11/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.0855 - acc: 0.
Epoch 12/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0813 - acc: 0.
Epoch 13/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0765 - acc: 0.
Epoch 14/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.0712 - acc: 0.
Epoch 15/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.0713 - acc: 0.
Epoch 16/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.0677 - acc: 0.
Epoch 17/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.0654 - acc: 0.
Epoch 18/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.0604 - acc: 0.
Epoch 19/20
```

```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy

vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.054573897721024694
Test accuracy: 0.9833
```




```
w_after = model.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



2.Three hidden layers

```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-function-in

from keras.models import Sequential
from keras.layers import Dense, Activation

from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization

model = Sequential()

model.add(Dense(506, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(290, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, s
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(130, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.225, s
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(output_dim, activation='softmax'))


model.summary()
```

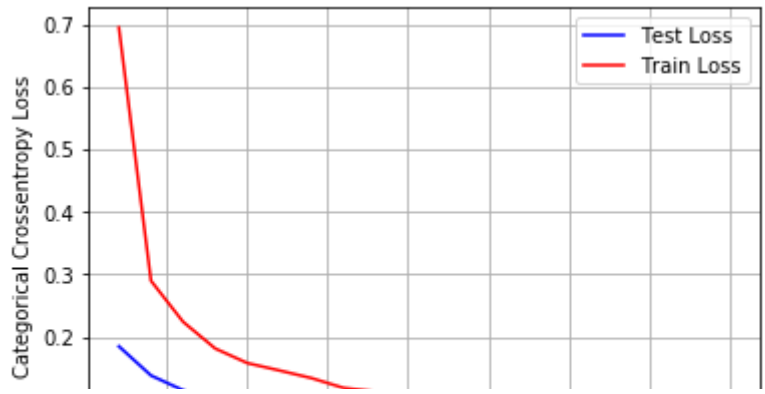
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 506)               397210    
_________________________________________________________________
batch_normalization_3 (Batch (None, 506)               2024      
_________________________________________________________________
dropout_3 (Dropout)          (None, 506)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 290)               147030    
_________________________________________________________________
batch_normalization_4 (Batch (None, 290)               1160      
_________________________________________________________________
dropout_4 (Dropout)          (None, 290)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 130)               37830     
_________________________________________________________________
batch_normalization_5 (Batch (None, 130)               520       
_________________________________________________________________
dropout_5 (Dropout)          (None, 130)               0         
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1310      
=================================================================
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validati
```

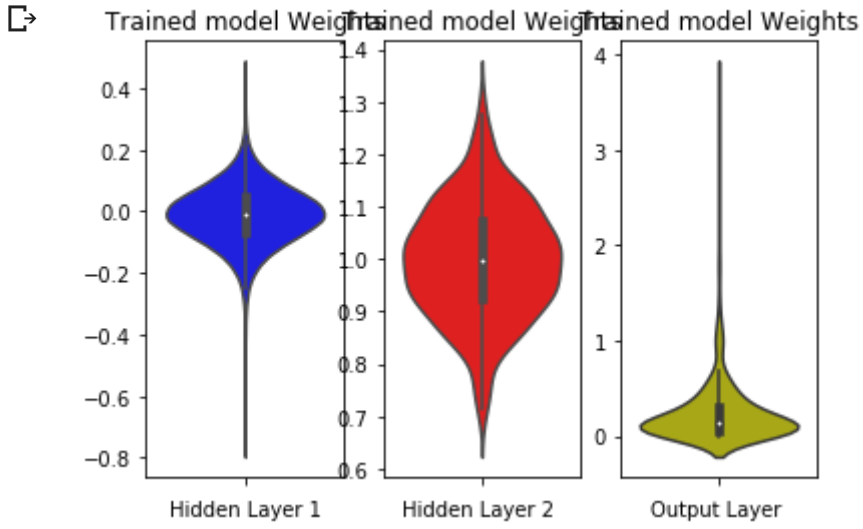
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 187us/step - loss: 0.6955 - acc: 0
Epoch 2/20
60000/60000 [==============================] - 10s 173us/step - loss: 0.2897 - acc: 0
Epoch 3/20
60000/60000 [==============================] - 10s 173us/step - loss: 0.2240 - acc: 0
Epoch 4/20
60000/60000 [==============================] - 10s 173us/step - loss: 0.1812 - acc: 0
Epoch 5/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.1579 - acc: 0
Epoch 6/20
60000/60000 [==============================] - 10s 174us/step - loss: 0.1461 - acc: 0
Epoch 7/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.1338 - acc: 0
Epoch 8/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.1180 - acc: 0
Epoch 9/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.1122 - acc: 0
Epoch 10/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.1032 - acc: 0
Epoch 11/20
60000/60000 [==============================] - 10s 173us/step - loss: 0.0964 - acc: 0
Epoch 12/20
60000/60000 [==============================] - 10s 174us/step - loss: 0.0915 - acc: 0
Epoch 13/20
60000/60000 [==============================] - 10s 173us/step - loss: 0.0845 - acc: 0
Epoch 14/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.0826 - acc: 0
Epoch 15/20
60000/60000 [==============================] - 10s 173us/step - loss: 0.0796 - acc: 0
Epoch 16/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.0776 - acc: 0
Epoch 17/20
60000/60000 [==============================] - 11s 175us/step - loss: 0.0771 - acc: 0
Epoch 18/20
60000/60000 [==============================] - 10s 174us/step - loss: 0.0697 - acc: 0
```

```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy

vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.06090008487093437
Test accuracy: 0.9841
```




```
w_after = model.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



3.Five hidden layers

```
model = Sequential()

model.add(Dense(609, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal
model.add(BatchNormalization())
model.add(Dropout(0.5))
```

```
model.add(Dense(510, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, s
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(390, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.205, s
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(270, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.285, s
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(110, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.325, s
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(output_dim, activation='softmax'))


model.summary()
```

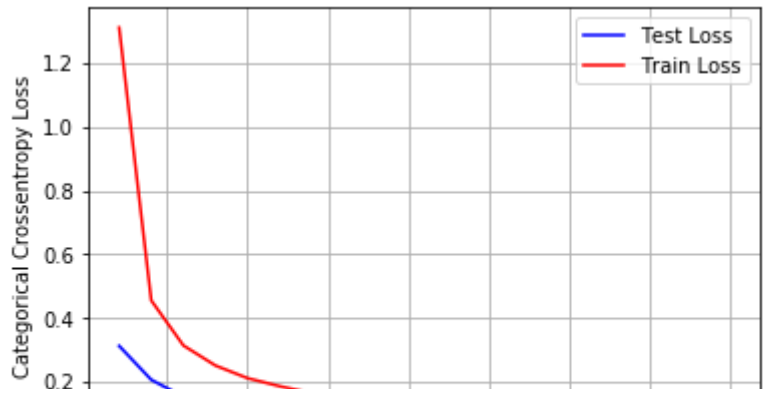
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 609)               478065
_________________________________________________________________
batch_normalization_6 (Batch (None, 609)               2436
_________________________________________________________________
dropout_6 (Dropout)          (None, 609)               0
_________________________________________________________________
dense_9 (Dense)              (None, 510)               311100
_________________________________________________________________
batch_normalization_7 (Batch (None, 510)               2040
_________________________________________________________________
dropout_7 (Dropout)          (None, 510)               0
_________________________________________________________________
dense_10 (Dense)             (None, 390)               199290
_________________________________________________________________
batch_normalization_8 (Batch (None, 390)               1560
_________________________________________________________________
dropout_8 (Dropout)          (None, 390)               0
_________________________________________________________________
dense_11 (Dense)             (None, 270)               105570
_________________________________________________________________
batch_normalization_9 (Batch (None, 270)               1080
_________________________________________________________________
dropout_9 (Dropout)          (None, 270)               0
_________________________________________________________________
dense_12 (Dense)             (None, 110)               29810
_________________________________________________________________
batch_normalization_10 (Batc (None, 110)               440
_________________________________________________________________
dropout_10 (Dropout)         (None, 110)               0
_________________________________________________________________
dense_13 (Dense)             (None, 10)                1110
=================================================================
Total params: 1,132,501
Trainable params: 1,128,723
Non-trainable params: 3,778
_________________________________________________________________
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validati
```

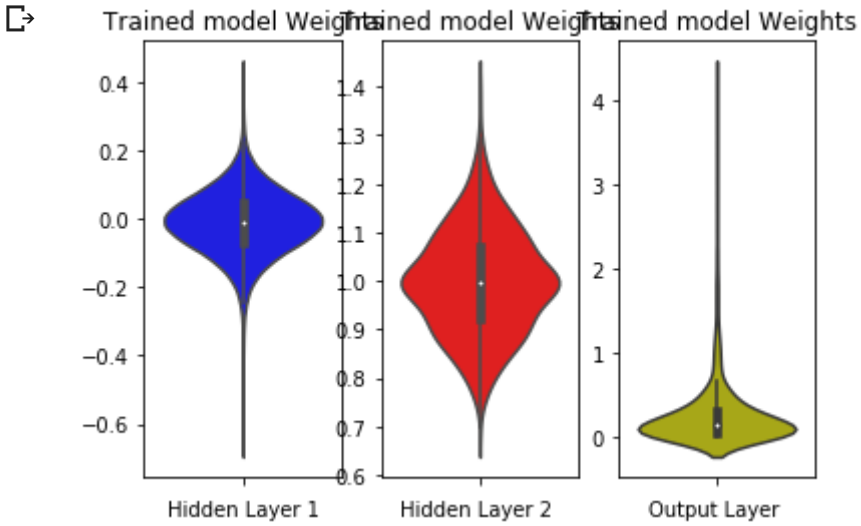
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 21s 358us/step - loss: 1.3110 - acc: 0
Epoch 2/20
60000/60000 [==============================] - 20s 326us/step - loss: 0.4553 - acc: 0
Epoch 3/20
60000/60000 [==============================] - 20s 328us/step - loss: 0.3142 - acc: 0
Epoch 4/20
60000/60000 [==============================] - 20s 327us/step - loss: 0.2511 - acc: 0
Epoch 5/20
60000/60000 [==============================] - 20s 327us/step - loss: 0.2115 - acc: 0
Epoch 6/20
60000/60000 [==============================] - 20s 327us/step - loss: 0.1863 - acc: 0
Epoch 7/20
60000/60000 [==============================] - 20s 327us/step - loss: 0.1643 - acc: 0
Epoch 8/20
60000/60000 [==============================] - 20s 327us/step - loss: 0.1540 - acc: 0
Epoch 9/20
60000/60000 [==============================] - 19s 324us/step - loss: 0.1399 - acc: 0
Epoch 10/20
60000/60000 [==============================] - 20s 328us/step - loss: 0.1322 - acc: 0
Epoch 11/20
60000/60000 [==============================] - 20s 326us/step - loss: 0.1216 - acc: 0
Epoch 12/20
60000/60000 [==============================] - 20s 332us/step - loss: 0.1200 - acc: 0
Epoch 13/20
60000/60000 [==============================] - 20s 329us/step - loss: 0.1092 - acc: 0
Epoch 14/20
60000/60000 [==============================] - 20s 329us/step - loss: 0.1043 - acc: 0
Epoch 15/20
60000/60000 [==============================] - 19s 322us/step - loss: 0.1009 - acc: 0
Epoch 16/20
60000/60000 [==============================] - 19s 325us/step - loss: 0.0983 - acc: 0
Epoch 17/20
60000/60000 [==============================] - 20s 327us/step - loss: 0.0936 - acc: 0
Epoch 18/20
60000/60000 [==============================] - 20s 326us/step - loss: 0.0882 - acc: 0
```

```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy

vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.07164842545758002
Test accuracy: 0.9812
```




```
w_after = model.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



MLP + Sigmoid activation + SGDOptimizer

```
#three hidden layers
model_sigmoid = Sequential()
model_sigmoid.add(Dense(512, activation='sigmoid', input_shape=(input_dim,)))
model_sigmoid.add(Dense(128, activation='sigmoid'))
model_sigmoid.add(Dense(65, activation='sigmoid'))
```

```
model_sigmoid.add(Dense(output_dim, activation='softmax'))

model_sigmoid.summary()
```

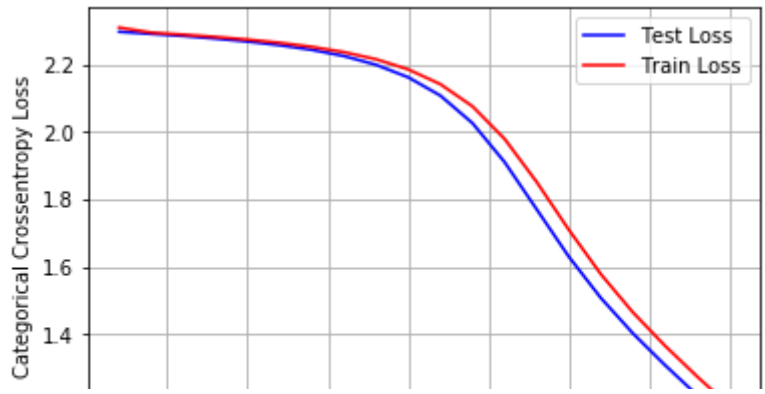
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_14 (Dense)             (None, 512)               401920    
_________________________________________________________________
dense_15 (Dense)             (None, 128)               65664     
_________________________________________________________________
dense_16 (Dense)             (None, 65)                8385      
_________________________________________________________________
dense_17 (Dense)             (None, 10)                660       
=================================================================
Total params: 476,629
Trainable params: 476,629
Non-trainable params: 0
_________________________________________________________________
```

```
model_sigmoid.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_sigmoid.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1,
```

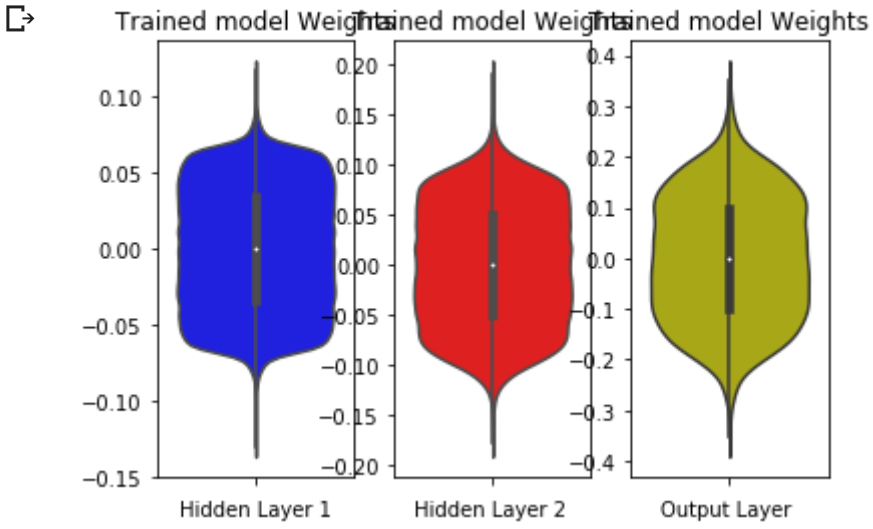
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 92us/step - loss: 2.3097 - acc: 0.1
Epoch 2/20
60000/60000 [==============================] - 5s 83us/step - loss: 2.2949 - acc: 0.1
Epoch 3/20
60000/60000 [==============================] - 5s 83us/step - loss: 2.2887 - acc: 0.1
Epoch 4/20
60000/60000 [==============================] - 5s 81us/step - loss: 2.2820 - acc: 0.1
Epoch 5/20
60000/60000 [==============================] - 5s 82us/step - loss: 2.2742 - acc: 0.1
Epoch 6/20
60000/60000 [==============================] - 5s 83us/step - loss: 2.2646 - acc: 0.1
Epoch 7/20
60000/60000 [==============================] - 5s 80us/step - loss: 2.2525 - acc: 0.2
Epoch 8/20
60000/60000 [==============================] - 5s 85us/step - loss: 2.2368 - acc: 0.2
Epoch 9/20
60000/60000 [==============================] - 5s 85us/step - loss: 2.2157 - acc: 0.3
Epoch 10/20
60000/60000 [==============================] - 5s 79us/step - loss: 2.1858 - acc: 0.4
Epoch 11/20
60000/60000 [==============================] - 5s 82us/step - loss: 2.1419 - acc: 0.4
Epoch 12/20
60000/60000 [==============================] - 5s 84us/step - loss: 2.0762 - acc: 0.5
Epoch 13/20
60000/60000 [==============================] - 5s 86us/step - loss: 1.9797 - acc: 0.5
Epoch 14/20
60000/60000 [==============================] - 5s 85us/step - loss: 1.8517 - acc: 0.5
Epoch 15/20
60000/60000 [==============================] - 5s 83us/step - loss: 1.7101 - acc: 0.5
Epoch 16/20
60000/60000 [==============================] - 5s 84us/step - loss: 1.5782 - acc: 0.5
Epoch 17/20
60000/60000 [==============================] - 5s 83us/step - loss: 1.4647 - acc: 0.6
Epoch 18/20
60000/60000 [==============================] - 5s 81us/step - loss: 1.3658 - acc: 0.6
```

```
score = model_sigmoid.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 1.1352740411758422
Test accuracy: 0.6817
```




```
w_after = model_sigmoid.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



MLP + Sigmoid activation + ADAM

```
#five hidden layers with BN and Dropout
model_sigmoid = Sequential()

model_sigmoid.add(Dense(665, activation='sigmoid', input_shape=(input_dim,)))
model_sigmoid.add(BatchNormalization())
```

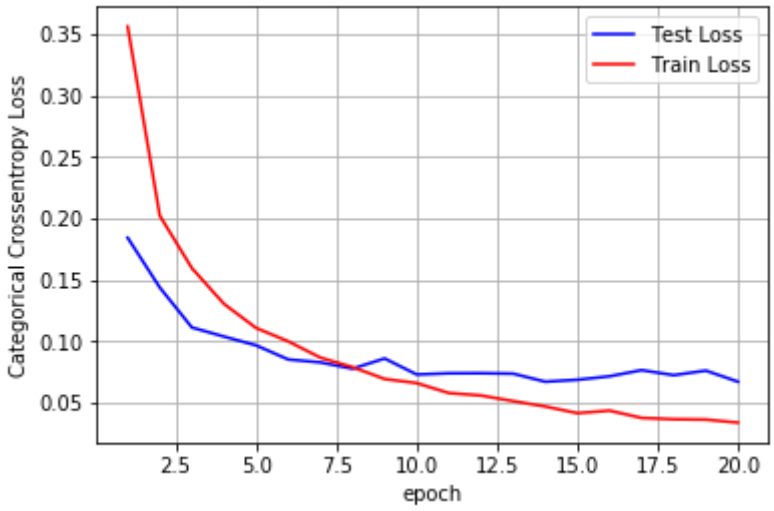
```
model_sigmoid.add(Dropout(0.2))

model_sigmoid.add(Dense(456, activation='sigmoid'))
model_sigmoid.add(BatchNormalization())
model_sigmoid.add(Dropout(0.2))

model_sigmoid.add(Dense(348, activation='sigmoid'))
model_sigmoid.add(BatchNormalization())
model_sigmoid.add(Dropout(0.2))

model_sigmoid.add(Dense(250, activation='sigmoid'))
model_sigmoid.add(BatchNormalization())
model_sigmoid.add(Dropout(0.2))

model_sigmoid.add(Dense(115, activation='sigmoid'))
model_sigmoid.add(BatchNormalization())
model_sigmoid.add(Dropout(0.2))

model_sigmoid.add(Dense(output_dim, activation='softmax'))

model_sigmoid.summary()

model_sigmoid.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_sigmoid.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1,
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_47 (Dense)             (None, 665)               522025    
_________________________________________________________________
batch_normalization_30 (Batc (None, 665)               2660      
_________________________________________________________________
dropout_25 (Dropout)         (None, 665)               0         
_________________________________________________________________
dense_48 (Dense)             (None, 456)               303696    
_________________________________________________________________
batch_normalization_31 (Batc (None, 456)               1824      
_________________________________________________________________
dropout_26 (Dropout)         (None, 456)               0         
_________________________________________________________________
dense_49 (Dense)             (None, 348)               159036    
_________________________________________________________________
batch_normalization_32 (Batc (None, 348)               1392      
_________________________________________________________________
dropout_27 (Dropout)         (None, 348)               0         
_________________________________________________________________
dense_50 (Dense)             (None, 250)               87250     
_________________________________________________________________
batch_normalization_33 (Batc (None, 250)               1000      
_________________________________________________________________
dropout_28 (Dropout)         (None, 250)               0         
_________________________________________________________________
dense_51 (Dense)             (None, 115)               28865     
_________________________________________________________________
batch_normalization_34 (Batc (None, 115)               460       
_________________________________________________________________
dropout_29 (Dropout)         (None, 115)               0         
_________________________________________________________________
dense_52 (Dense)             (None, 10)                1160      
=================================================================
Total params: 1,109,368
Trainable params: 1,105,700
Non-trainable params: 3,668
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 22s 362us/step - loss: 0.3562 - acc: 0
Epoch 2/20
60000/60000 [==============================] - 18s 307us/step - loss: 0.2023 - acc: 0
Epoch 3/20
60000/60000 [==============================] - 19s 309us/step - loss: 0.1596 - acc: 0
Epoch 4/20
60000/60000 [==============================] - 19s 309us/step - loss: 0.1305 - acc: 0
Epoch 5/20
60000/60000 [==============================] - 19s 310us/step - loss: 0.1109 - acc: 0
Epoch 6/20
60000/60000 [==============================] - 19s 313us/step - loss: 0.1000 - acc: 0
Epoch 7/20
60000/60000 [==============================] - 19s 312us/step - loss: 0.0868 - acc: 0
Epoch 8/20
60000/60000 [==============================] - 19s 311us/step - loss: 0.0791 - acc: 0
Epoch 9/20
60000/60000 [==============================] - 19s 311us/step - loss: 0.0694 - acc: 0
Epoch 10/20
60000/60000 [==============================] - 19s 311us/step - loss: 0.0661 - acc: 0
Epoch 11/20
```

```
60000/60000 [==============================] - 19s 312us/step - loss: 0.0580 - acc: 0
Epoch 12/20
60000/60000 [==============================] - 19s 310us/step - loss: 0.0560 - acc: 0
Epoch 13/20
60000/60000 [==============================] - 19s 312us/step - loss: 0.0515 - acc: 0
Epoch 14/20
60000/60000 [==============================] - 19s 310us/step - loss: 0.0471 - acc: 0
Epoch 15/20
60000/60000 [==============================] - 19s 309us/step - loss: 0.0417 - acc: 0
Epoch 16/20
60000/60000 [==============================] - 19s 313us/step - loss: 0.0437 - acc: 0
Epoch 17/20
60000/60000 [==============================] - 19s 308us/step - loss: 0.0378 - acc: 0
Epoch 18/20
60000/60000 [==============================] - 19s 310us/step - loss: 0.0367 - acc: 0
```

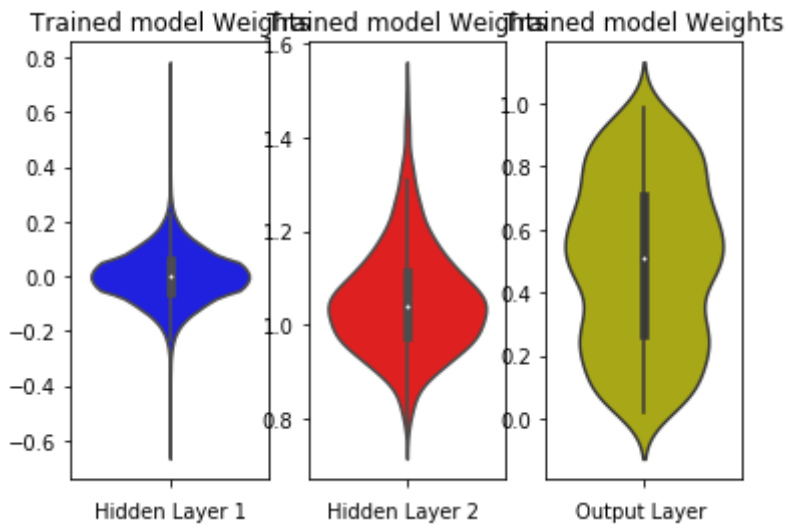
```
score = model_sigmoid.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.0672482188842725
Test accuracy: 0.9833
```




```
w_after = model_sigmoid.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
```

```
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



MLP + ReLU +SGD

```
#three hidden layers with different weight initializations
model_relu = Sequential()
model_relu.add(Dense(548, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomN
model_relu.add(Dense(328, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.1
model_relu.add(Dense(158, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.1
model_relu.add(Dense(output_dim, activation='softmax'))

model_relu.summary()
```

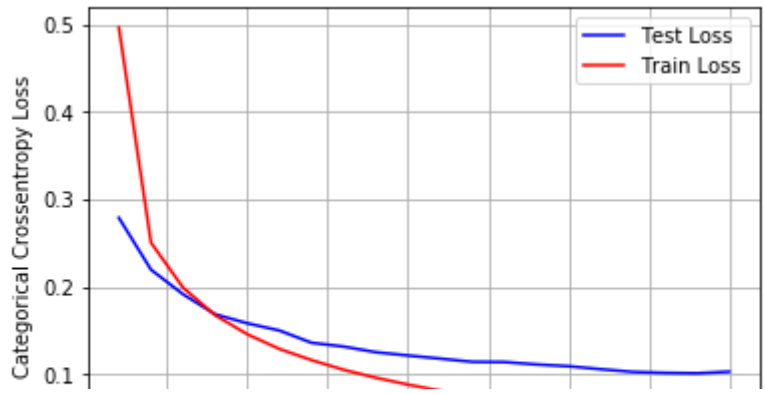
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_25 (Dense)             (None, 548)               430180
_________________________________________________________________
dense_26 (Dense)             (None, 328)               180072
_________________________________________________________________
dense_27 (Dense)             (None, 158)               51982
_________________________________________________________________
dense_28 (Dense)             (None, 10)                1590
=================================================================
Total params: 663,824
Trainable params: 663,824
Non-trainable params: 0
_________________________________________________________________
```

```
model_relu.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, val
```

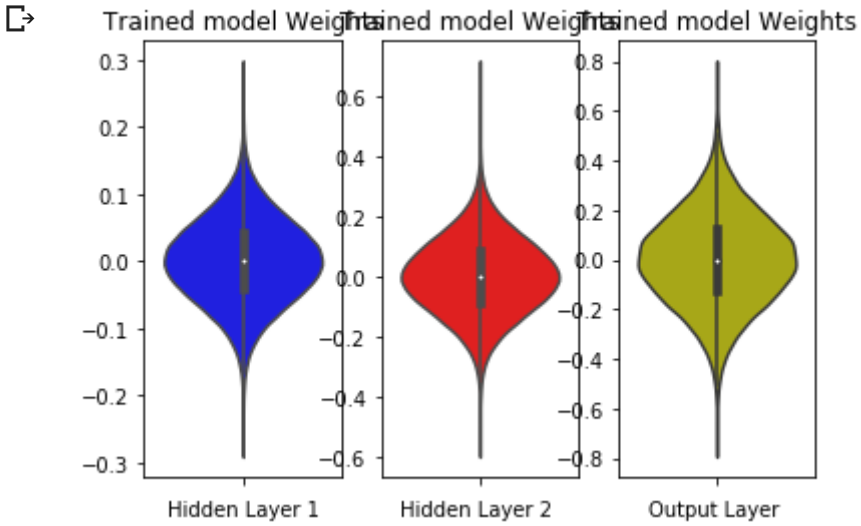
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.4969 - acc: 0.
Epoch 2/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.2506 - acc: 0.
Epoch 3/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.1990 - acc: 0.
Epoch 4/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.1674 - acc: 0.
Epoch 5/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.1456 - acc: 0.
Epoch 6/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.1287 - acc: 0.
Epoch 7/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.1160 - acc: 0.
Epoch 8/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.1049 - acc: 0.
Epoch 9/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0958 - acc: 0.
Epoch 10/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0880 - acc: 0.
Epoch 11/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0809 - acc: 0.
Epoch 12/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0750 - acc: 0.
Epoch 13/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0694 - acc: 0.
Epoch 14/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0648 - acc: 0.
Epoch 15/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.0603 - acc: 0.
Epoch 16/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.0562 - acc: 0.
Epoch 17/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0526 - acc: 0.
Epoch 18/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0493 - acc: 0.
```

```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.10271617978247814
Test accuracy: 0.9711
```




```
w_after = model_relu.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



MLP + Batch-Norm on hidden Layers + AdamOptimizer

```
#Five hidden layers with activation='sigmoid', BN & different weight initializations

model_batch = Sequential()

model_batch.add(Dense(512, activation='sigmoid', input_shape=(input_dim,), kernel_initializer=Ran
```

```
model_batch.add(BatchNormalization())

model_batch.add(Dense(456, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev
model_batch.add(BatchNormalization())

model_batch.add(Dense(368, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev
model_batch.add(BatchNormalization())

model_batch.add(Dense(228, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev
model_batch.add(BatchNormalization())

model_batch.add(Dense(123, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev
model_batch.add(BatchNormalization())

model_batch.add(Dense(output_dim, activation='softmax'))


model_batch.summary()
```

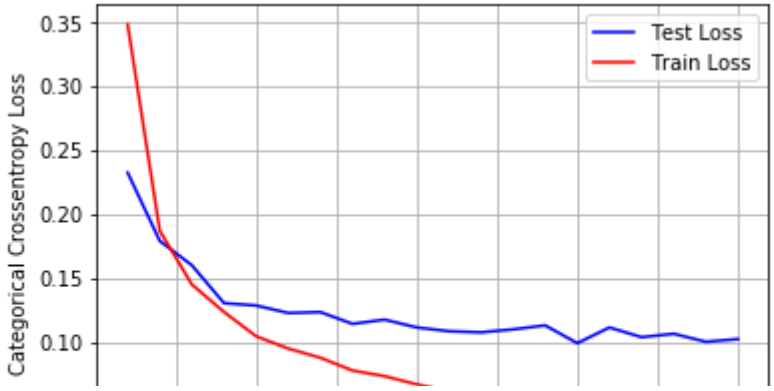
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_33 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_19 (Batc (None, 512)               2048
_________________________________________________________________
dense_34 (Dense)             (None, 456)               233928
_________________________________________________________________
batch_normalization_20 (Batc (None, 456)               1824
_________________________________________________________________
dense_35 (Dense)             (None, 368)               168176
_________________________________________________________________
batch_normalization_21 (Batc (None, 368)               1472
_________________________________________________________________
dense_36 (Dense)             (None, 228)               84132
_________________________________________________________________
batch_normalization_22 (Batc (None, 228)               912
_________________________________________________________________
dense_37 (Dense)             (None, 123)               28167
_________________________________________________________________
batch_normalization_23 (Batc (None, 123)               492
_________________________________________________________________
dense_38 (Dense)             (None, 10)                1240
=================================================================
Total params: 924,311
Trainable params: 920,937
Non-trainable params: 3,374
_________________________________________________________________
```

```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, va
```

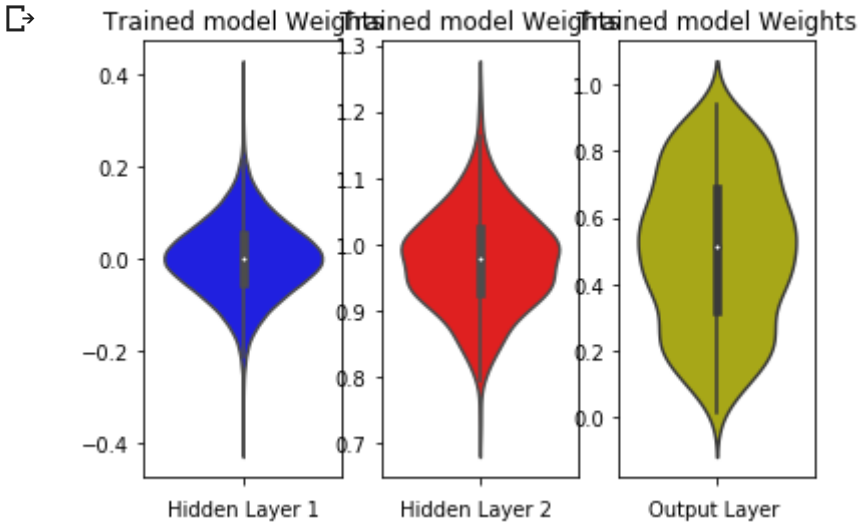
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 18s 294us/step - loss: 0.3484 - acc: 0
Epoch 2/20
60000/60000 [==============================] - 15s 249us/step - loss: 0.1872 - acc: 0
Epoch 3/20
60000/60000 [==============================] - 15s 252us/step - loss: 0.1451 - acc: 0
Epoch 4/20
60000/60000 [==============================] - 15s 252us/step - loss: 0.1237 - acc: 0
Epoch 5/20
60000/60000 [==============================] - 15s 252us/step - loss: 0.1047 - acc: 0
Epoch 6/20
60000/60000 [==============================] - 15s 248us/step - loss: 0.0951 - acc: 0
Epoch 7/20
60000/60000 [==============================] - 15s 253us/step - loss: 0.0879 - acc: 0
Epoch 8/20
60000/60000 [==============================] - 15s 253us/step - loss: 0.0779 - acc: 0
Epoch 9/20
60000/60000 [==============================] - 15s 250us/step - loss: 0.0735 - acc: 0
Epoch 10/20
60000/60000 [==============================] - 15s 250us/step - loss: 0.0671 - acc: 0
Epoch 11/20
60000/60000 [==============================] - 15s 248us/step - loss: 0.0617 - acc: 0
Epoch 12/20
60000/60000 [==============================] - 15s 245us/step - loss: 0.0560 - acc: 0
Epoch 13/20
60000/60000 [==============================] - 15s 248us/step - loss: 0.0557 - acc: 0
Epoch 14/20
60000/60000 [==============================] - 15s 252us/step - loss: 0.0529 - acc: 0
Epoch 15/20
60000/60000 [==============================] - 15s 250us/step - loss: 0.0510 - acc: 0
Epoch 16/20
60000/60000 [==============================] - 15s 249us/step - loss: 0.0467 - acc: 0
Epoch 17/20
60000/60000 [==============================] - 15s 248us/step - loss: 0.0451 - acc: 0
Epoch 18/20
60000/60000 [==============================] - 15s 252us/step - loss: 0.0416 - acc: 0
```

```
score = model_batch.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.10247163554290309
Test accuracy: 0.9707
```




```
w_after = model_batch.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



MLP + Dropout + AdamOptimizer

```
#three hidden layers with different weight initializations
model_drop = Sequential()

model_drop.add(Dense(412, activation='sigmoid', input_shape=(input_dim,), kernel_initializer=Rand
model_drop.add(BatchNormalization())
```

```
model_drop.add(Dropout(0.3))

model_drop.add(Dense(228, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.3))

model_drop.add(Dense(98, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0, stddev=0
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.3))

model_drop.add(Dense(output_dim, activation='softmax'))


model_drop.summary()
```

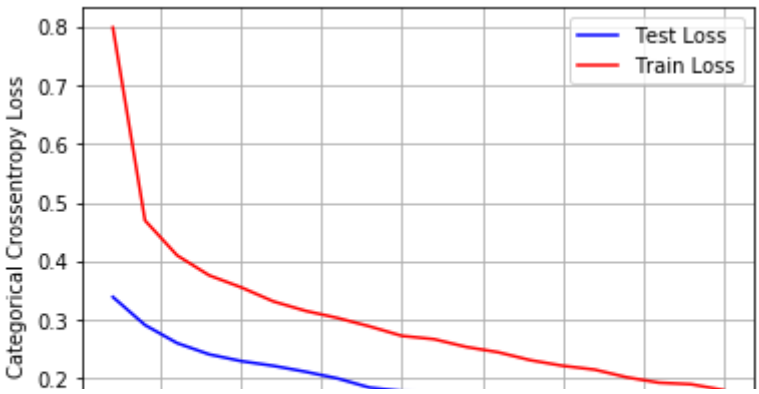
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_43 (Dense)             (None, 412)               323420    
_________________________________________________________________
batch_normalization_27 (Batc (None, 412)               1648      
_________________________________________________________________
dropout_22 (Dropout)         (None, 412)               0         
_________________________________________________________________
dense_44 (Dense)             (None, 228)               94164     
_________________________________________________________________
batch_normalization_28 (Batc (None, 228)               912       
_________________________________________________________________
dropout_23 (Dropout)         (None, 228)               0         
_________________________________________________________________
dense_45 (Dense)             (None, 98)                22442     
_________________________________________________________________
batch_normalization_29 (Batc (None, 98)                392       
_________________________________________________________________
dropout_24 (Dropout)         (None, 98)                0         
_________________________________________________________________
dense_46 (Dense)             (None, 10)                990       
=================================================================
Total params: 443,968
Trainable params: 442,492
Non-trainable params: 1,476
_________________________________________________________________
```

```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, val
```

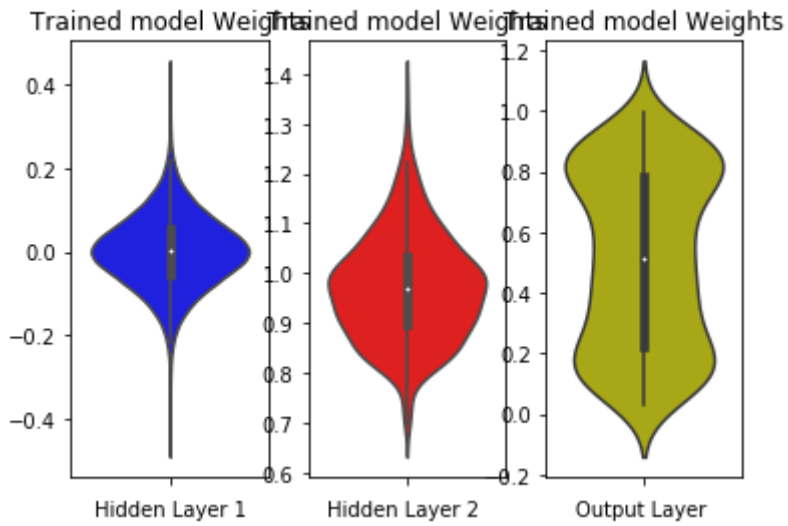
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 178us/step - loss: 0.7992 - acc: 0
Epoch 2/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.4700 - acc: 0.
Epoch 3/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.4104 - acc: 0.
Epoch 4/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.3763 - acc: 0.
Epoch 5/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.3557 - acc: 0.
Epoch 6/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.3318 - acc: 0.
Epoch 7/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.3157 - acc: 0.
Epoch 8/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.3036 - acc: 0.
Epoch 9/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.2891 - acc: 0.
Epoch 10/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.2730 - acc: 0.
Epoch 11/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.2674 - acc: 0.
Epoch 12/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.2542 - acc: 0.
Epoch 13/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.2451 - acc: 0.
Epoch 14/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.2316 - acc: 0.
Epoch 15/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.2220 - acc: 0.
Epoch 16/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.2155 - acc: 0.
Epoch 17/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.2024 - acc: 0.
Epoch 18/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.1932 - acc: 0.
```

```
score = model_drop.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.12330705345384776
Test accuracy: 0.9624
```




```
w_after = model_drop.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[2].flatten().reshape(-1,1)
out_w = w_after[4].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 3, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



Conclusion.

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Activation Function","optimizer","Batch Normalisations","Dropout","No. of hidde
```

```
x.add_row(["relu", "AdamOptimizer", "Yes", 0.5, 2, 0.054, 0.98])
x.add_row(["relu", "AdamOptimizer", "Yes", 0.5, 3, 0.060, 0.98])
x.add_row(["relu", "AdamOptimizer", "Yes", 0.5, 5, 0.071, 0.98])
x.add_row(["relu", "SGDOptimizer", "No", "No dropout", 3, 0.102, 0.97])
x.add_row(["Sigmoid", "SGDOptimizer", "No", "No droput", 3, 1.13, 0.68])
x.add_row(["Sigmoid", "AdamOptimizer", "Yes", 0.2, 5, 0.067, 0.98])
x.add_row(["Sigmoid", "AdamOptimizer", "Yes", "No droput", 5, 0.102, 0.97])
x.add_row(["Sigmoid", "AdamOptimizer", "Yes", 0.3, 3, 0.123, 0.96])
print(x)
```

```
+---------------------+---------------+----------------------+------------+----------
| Activation Function |   optimizer   | Batch Normalisations |  Dropout   | No. of hi
+---------------------+---------------+----------------------+------------+----------
|         relu        | AdamOptimizer |         Yes          |    0.5     |
|         relu        | AdamOptimizer |         Yes          |    0.5     |
|         relu        | AdamOptimizer |         Yes          |    0.5     |
|         relu        |  SGDOptimizer |          No          | No dropout |
|       Sigmoid       |  SGDOptimizer |          No          | No droput  |
|       Sigmoid       | AdamOptimizer |         Yes          |    0.2     |
|       Sigmoid       | AdamOptimizer |         Yes          | No droput  |
|       Sigmoid       | AdamOptimizer |         Yes          |    0.3     |
+---------------------+---------------+----------------------+------------+----------
```

Obervations:

1.We can see with different architecture,activation function & optimizer test score and test acc

2.from above table, we can observe relu activation have minimum test score and maximum tes

3.AdamOptimizer is better than sgdoptimizer

4.Batch Normalisations and Dropout are playing important role in improving test accuracy

5.MLP having sigmoid activation and sgdoptimizer alongwith three hidden layers and no BN & i.e.0.68

6.With keras it become easy to implement deep neural network.While doing this assignment I l very powerful than classical modeling.