Keras -- MLPs on MNIST

```
# if you keras is not using tensorflow as backend set "KERAS_BACKEND=tensorflow" use this command
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
```

```
%matplotlib notebook
%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="Validation 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()
```

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

```
Number of training examples : 60000 and each image is of shape (28, 28)
Number of training 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_train.sh
print("Number of training examples :", X_test.shape[0], "and each image is of shape (%d)"%(X_test.shap
```

```
Number of training examples : 60000 and each image is of shape (784)
Number of training 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
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   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.
```

```
# 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.]
```

Softmax classifier

```
# https://keras.io/getting-started/sequential-model-guide/

# The Sequential model is a linear stack of layers.
# you can create a Sequential model by passing a list of layer instances to the constructor:

# model = Sequential([
#     Dense(32, input_shape=(784,)),
#     Activation('relu'),
#     Dense(10),
#     Activation('softmax'),
# ])

# You can also simply add layers via the .add() method:

# model = Sequential()
# model.add(Dense(32, input_dim=784))
# model.add(Activation('relu'))

###
```

```
# https://keras.io/layers/core/

# keras.layers.Dense(units, activation=None, use_bias=True, kernel_initializer='glorot_uniform',
# bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None,
# kernel_constraint=None, bias_constraint=None)

# Dense implements the operation: output = activation(dot(input, kernel) + bias) where
# activation is the element-wise activation function passed as the activation argument,
# kernel is a weights matrix created by the layer, and
# bias is a bias vector created by the layer (only applicable if use_bias is True).

# output = activation(dot(input, kernel) + bias)  => y = activation(WT. X + b)

####

# https://keras.io/activations/

# Activations can either be used through an Activation layer, or through the activation argument suppo

# from keras.layers import Activation, Dense

# model.add(Dense(64))
# model.add(Activation('tanh'))

# This is equivalent to:
# model.add(Dense(64, activation='tanh'))

# there are many activation functions ar available ex: tanh, relu, softmax


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



# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 30
```

> **MLP + BN + Dropout + AdamOptimizer + ReLu activation**

> **Architecture 1**: 784-366-100-10;
> droput: .5,.5

std = sqrt(2/fan-in) as it is relu

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


model = Sequential()

model.add(Dense(366, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean
model.add(BatchNormalization())
model.add(Dropout(0.5))


model.add(Dense(100, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.14142, seed
model.add(BatchNormalization())
```

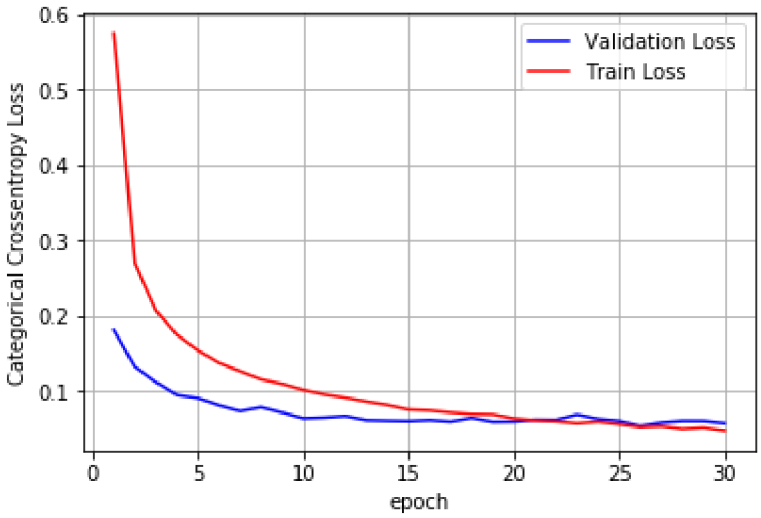
```
model.add(Dropout(0.5))

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

model.summary()

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, validation_da
```

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_44 (Dense)             (None, 366)               287310    
_________________________________________________________________
batch_normalization_31 (Batc (None, 366)               1464      
_________________________________________________________________
dropout_31 (Dropout)         (None, 366)               0         
_________________________________________________________________
dense_45 (Dense)             (None, 100)               36700     
_________________________________________________________________
batch_normalization_32 (Batc (None, 100)               400       
_________________________________________________________________
dropout_32 (Dropout)         (None, 100)               0         
_________________________________________________________________
dense_46 (Dense)             (None, 10)                1010      
=================================================================
Total params: 326,884
Trainable params: 325,952
Non-trainable params: 932
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 9s 158us/step - loss: 0.5750 - acc: 0.
Epoch 2/30
60000/60000 [==============================] - 5s 88us/step - loss: 0.2700 - acc: 0.9
Epoch 3/30
60000/60000 [==============================] - 5s 91us/step - loss: 0.2075 - acc: 0.9
Epoch 4/30
60000/60000 [==============================] - 6s 94us/step - loss: 0.1752 - acc: 0.9
Epoch 5/30
60000/60000 [==============================] - 6s 95us/step - loss: 0.1544 - acc: 0.9
Epoch 6/30
60000/60000 [==============================] - 5s 87us/step - loss: 0.1380 - acc: 0.9
Epoch 7/30
60000/60000 [==============================] - 6s 93us/step - loss: 0.1268 - acc: 0.9
Epoch 8/30
60000/60000 [==============================] - 5s 90us/step - loss: 0.1162 - acc: 0.9
Epoch 9/30
60000/60000 [==============================] - 5s 90us/step - loss: 0.1100 - acc: 0.9
Epoch 10/30
60000/60000 [==============================] - 5s 89us/step - loss: 0.1020 - acc: 0.9
Epoch 11/30
60000/60000 [==============================] - 5s 91us/step - loss: 0.0961 - acc: 0.9
Epoch 12/30
60000/60000 [==============================] - 5s 91us/step - loss: 0.0918 - acc: 0.9
Epoch 13/30
60000/60000 [==============================] - 5s 90us/step - loss: 0.0866 - acc: 0.9
Epoch 14/30
60000/60000 [==============================] - 5s 92us/step - loss: 0.0821 - acc: 0.9
Epoch 15/30
60000/60000 [==============================] - 6s 92us/step - loss: 0.0762 - acc: 0.9
Epoch 16/30
60000/60000 [==============================] - 6s 92us/step - loss: 0.0750 - acc: 0.9
Epoch 17/30
60000/60000 [==============================] - 6s 92us/step - loss: 0.0722 - acc: 0.9
```

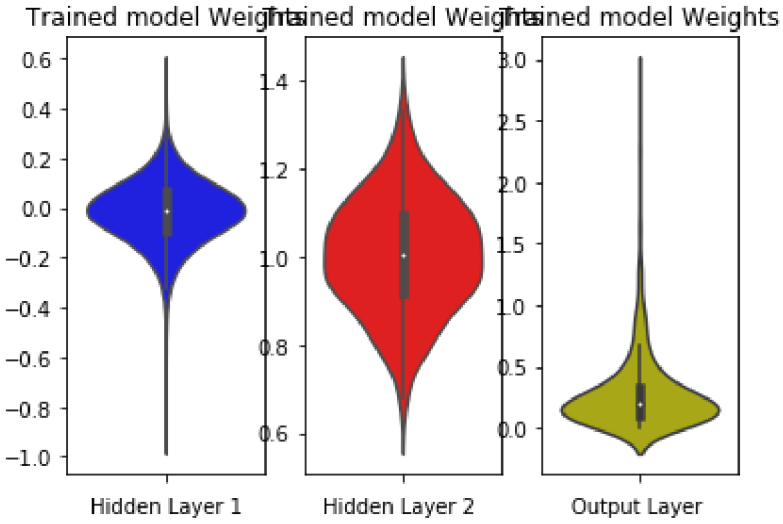
```
Epoch 18/30
60000/60000 [==============================] - 5s 89us/step - loss: 0.0700 - acc: 0.9
Epoch 19/30
60000/60000 [==============================] - 5s 91us/step - loss: 0.0695 - acc: 0.9
Epoch 20/30
60000/60000 [==============================] - 5s 89us/step - loss: 0.0640 - acc: 0.9
Epoch 21/30
60000/60000 [==============================] - 5s 91us/step - loss: 0.0616 - acc: 0.9
Epoch 22/30
60000/60000 [==============================] - 5s 90us/step - loss: 0.0604 - acc: 0.9
Epoch 23/30
60000/60000 [==============================] - 5s 89us/step - loss: 0.0577 - acc: 0.9
Epoch 24/30
60000/60000 [==============================] - 5s 87us/step - loss: 0.0599 - acc: 0.9
Epoch 25/30
60000/60000 [==============================] - 5s 90us/step - loss: 0.0567 - acc: 0.9
Epoch 26/30
60000/60000 [==============================] - 5s 90us/step - loss: 0.0529 - acc: 0.9
Epoch 27/30
60000/60000 [==============================] - 5s 89us/step - loss: 0.0540 - acc: 0.9
Epoch 28/30
60000/60000 [==============================] - 5s 88us/step - loss: 0.0505 - acc: 0.9
Epoch 29/30
60000/60000 [==============================] - 6s 92us/step - loss: 0.0521 - acc: 0.9
Epoch 30/30
60000/60000 [==============================] - 6s 92us/step - loss: 0.0482 - acc: 0.9
```

```
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt

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))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, valida

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of epochs


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

```
Test score: 0.05789125898362254
Test accuracy: 0.9844
```




```
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt

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 + BN + Dropout + AdamOptimizer + ReLu activation**

> **Architecture 2**: 784-456-128-10;
> droput: .5,.5

std = sqrt(2/fan-in) as it is ReLu

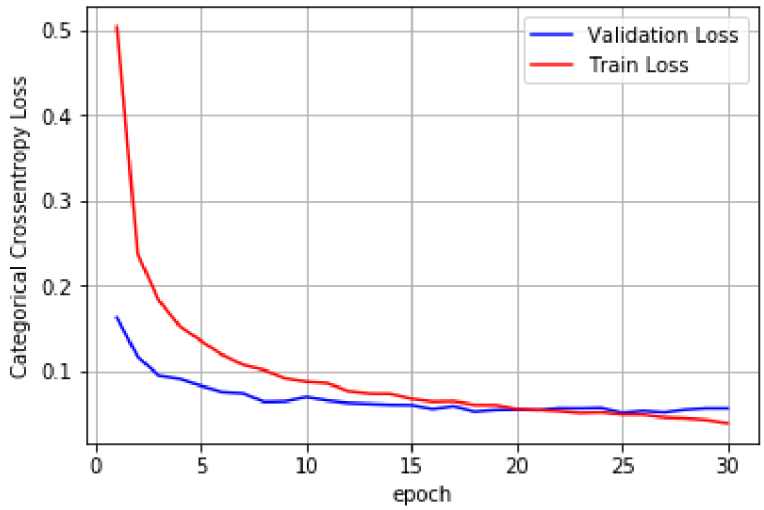
```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout
import warnings
warnings.filterwarnings('ignore')

model2 = Sequential()

model2.add(Dense(456, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mea
model2.add(BatchNormalization())
model2.add(Dropout(0.5))


model2.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=
model2.add(BatchNormalization())
model2.add(Dropout(0.5))


model2.add(Dense(output_dim, activation='softmax'))
model2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history2 = model2.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 10s 165us/step - loss: 0.5033 - acc: 0
Epoch 2/30
60000/60000 [==============================] - 6s 94us/step - loss: 0.2371 - acc: 0.9
Epoch 3/30
60000/60000 [==============================] - 6s 94us/step - loss: 0.1830 - acc: 0.9
Epoch 4/30
60000/60000 [==============================] - 6s 97us/step - loss: 0.1527 - acc: 0.9
Epoch 5/30
60000/60000 [==============================] - 6s 94us/step - loss: 0.1358 - acc: 0.9
Epoch 6/30
60000/60000 [==============================] - 6s 95us/step - loss: 0.1195 - acc: 0.9
Epoch 7/30
60000/60000 [==============================] - 6s 96us/step - loss: 0.1078 - acc: 0.9
Epoch 8/30
60000/60000 [==============================] - 6s 94us/step - loss: 0.1015 - acc: 0.9
Epoch 9/30
60000/60000 [==============================] - 6s 94us/step - loss: 0.0918 - acc: 0.9
Epoch 10/30
60000/60000 [==============================] - 6s 94us/step - loss: 0.0878 - acc: 0.9
Epoch 11/30
60000/60000 [==============================] - 6s 93us/step - loss: 0.0861 - acc: 0.9
Epoch 12/30
60000/60000 [==============================] - 6s 93us/step - loss: 0.0768 - acc: 0.9
Epoch 13/30
60000/60000 [==============================] - 6s 94us/step - loss: 0.0740 - acc: 0.9
Epoch 14/30
60000/60000 [==============================] - 6s 97us/step - loss: 0.0736 - acc: 0.9
Epoch 15/30
60000/60000 [==============================] - 6s 96us/step - loss: 0.0678 - acc: 0.9
Epoch 16/30
60000/60000 [==============================] - 6s 95us/step - loss: 0.0644 - acc: 0.9
Epoch 17/30
60000/60000 [==============================] - 6s 94us/step - loss: 0.0650 - acc: 0.9
Epoch 18/30
60000/60000 [==============================] - 6s 95us/step - loss: 0.0606 - acc: 0.9
Epoch 19/30
60000/60000 [==============================] - 6s 93us/step - loss: 0.0605 - acc: 0.9
Epoch 20/30
60000/60000 [==============================] - 6s 93us/step - loss: 0.0559 - acc: 0.9
Epoch 21/30
60000/60000 [==============================] - 6s 94us/step - loss: 0.0548 - acc: 0.9
Epoch 22/30
60000/60000 [==============================] - 6s 92us/step - loss: 0.0533 - acc: 0.9
Epoch 23/30
60000/60000 [==============================] - 6s 96us/step - loss: 0.0511 - acc: 0.9
Epoch 24/30
60000/60000 [==============================] - 6s 95us/step - loss: 0.0519 - acc: 0.9
Epoch 25/30
60000/60000 [==============================] - 6s 94us/step - loss: 0.0493 - acc: 0.9
Epoch 26/30
60000/60000 [==============================] - 6s 92us/step - loss: 0.0492 - acc: 0.9
Epoch 27/30
60000/60000 [==============================] - 6s 95us/step - loss: 0.0459 - acc: 0.9
Epoch 28/30
60000/60000 [==============================] - 6s 95us/step - loss: 0.0449 - acc: 0.9
```

```
Epoch 29/30
60000/60000 [==============================] - 6s 98us/step - loss: 0.0430 - acc: 0.9
Epoch 30/30
60000/60000 [==============================] - 6s 94us/step - loss: 0.0393 - acc: 0.9
```

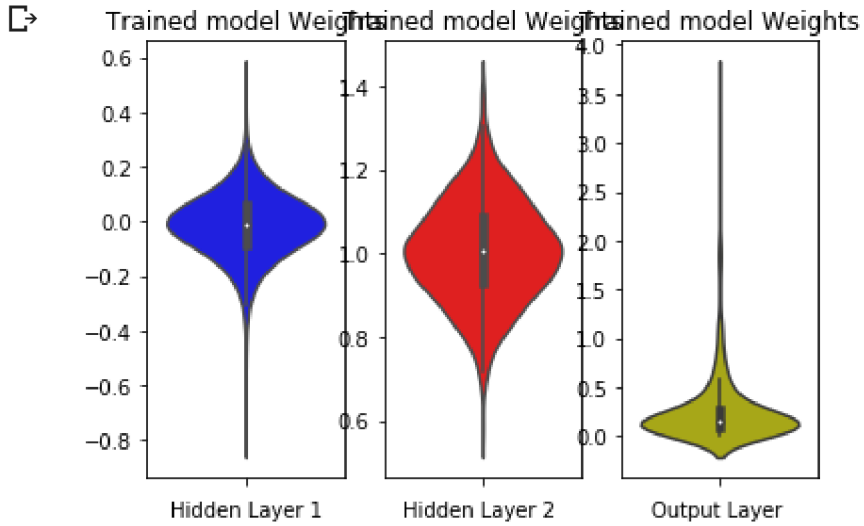
```
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

score2 = model2.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score2[0])
print('Test accuracy:', score2[1])

fig2,ax2 = plt.subplots(1,1)
ax2.set_xlabel('epoch') ; ax2.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x2 = list(range(1,nb_epoch+1))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, valida

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of epochs


vy2 = history2.history['val_loss']
ty2 = history2.history['loss']
plt_dynamic(x2, vy2, ty2, ax2)
```

```
Test score: 0.056721248715037656
Test accuracy: 0.984
```




```
%matplotlib notebook
%matplotlib inline
```

```
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

w_after2 = model2.get_weights()

h1_w = w_after2[0].flatten().reshape(-1,1)
h2_w = w_after2[2].flatten().reshape(-1,1)
out_w = w_after2[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 + BN + Dropout + AdamOptimizer + ReLu activation**

> **Architecture 3**: 784-256-128-10;
> droput: .5,.5

std = sqrt(2/fan-in) as it is ReLu

```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout
import warnings
warnings.filterwarnings('ignore')
```

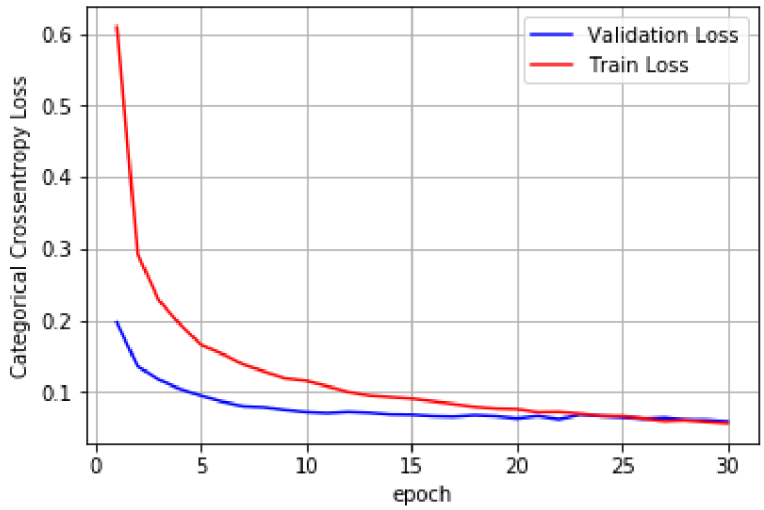
```
model3 = Sequential()

model3.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mea
model3.add(BatchNormalization())
model3.add(Dropout(0.5))


model3.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=
model3.add(BatchNormalization())
model3.add(Dropout(0.5))


model3.add(Dense(output_dim, activation='softmax'))
model3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history3 = model3.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 10s 166us/step - loss: 0.6093 - acc: 0
Epoch 2/30
60000/60000 [==============================] - 5s 89us/step - loss: 0.2919 - acc: 0.9
Epoch 3/30
60000/60000 [==============================] - 5s 90us/step - loss: 0.2280 - acc: 0.9
Epoch 4/30
60000/60000 [==============================] - 5s 86us/step - loss: 0.1943 - acc: 0.9
Epoch 5/30
60000/60000 [==============================] - 5s 89us/step - loss: 0.1661 - acc: 0.9
Epoch 6/30
60000/60000 [==============================] - 5s 89us/step - loss: 0.1532 - acc: 0.9
Epoch 7/30
60000/60000 [==============================] - 5s 88us/step - loss: 0.1386 - acc: 0.9
Epoch 8/30
60000/60000 [==============================] - 5s 89us/step - loss: 0.1285 - acc: 0.9
Epoch 9/30
60000/60000 [==============================] - 5s 87us/step - loss: 0.1190 - acc: 0.9
Epoch 10/30
60000/60000 [==============================] - 5s 88us/step - loss: 0.1156 - acc: 0.9
Epoch 11/30
60000/60000 [==============================] - 5s 88us/step - loss: 0.1078 - acc: 0.9
Epoch 12/30
60000/60000 [==============================] - 5s 88us/step - loss: 0.0999 - acc: 0.9
Epoch 13/30
60000/60000 [==============================] - 5s 88us/step - loss: 0.0949 - acc: 0.9
Epoch 14/30
60000/60000 [==============================] - 5s 87us/step - loss: 0.0927 - acc: 0.9
Epoch 15/30
60000/60000 [==============================] - 5s 88us/step - loss: 0.0906 - acc: 0.9
Epoch 16/30
60000/60000 [==============================] - 5s 89us/step - loss: 0.0868 - acc: 0.9
Epoch 17/30
60000/60000 [==============================] - 5s 90us/step - loss: 0.0830 - acc: 0.9
Epoch 18/30
60000/60000 [==============================] - 5s 87us/step - loss: 0.0791 - acc: 0.9
Epoch 19/30
60000/60000 [==============================] - 5s 89us/step - loss: 0.0768 - acc: 0.9
Epoch 20/30
60000/60000 [==============================] - 5s 88us/step - loss: 0.0761 - acc: 0.9
Epoch 21/30
60000/60000 [==============================] - 5s 89us/step - loss: 0.0716 - acc: 0.9
Epoch 22/30
60000/60000 [==============================] - 5s 86us/step - loss: 0.0721 - acc: 0.9
Epoch 23/30
60000/60000 [==============================] - 5s 87us/step - loss: 0.0696 - acc: 0.9
Epoch 24/30
60000/60000 [==============================] - 5s 87us/step - loss: 0.0663 - acc: 0.9
Epoch 25/30
60000/60000 [==============================] - 5s 89us/step - loss: 0.0662 - acc: 0.9
Epoch 26/30
60000/60000 [==============================] - 5s 88us/step - loss: 0.0630 - acc: 0.9
Epoch 27/30
60000/60000 [==============================] - 5s 88us/step - loss: 0.0594 - acc: 0.9
Epoch 28/30
60000/60000 [==============================] - 5s 88us/step - loss: 0.0605 - acc: 0.9
```

```
Epoch 29/30
60000/60000 [==============================] - 5s 90us/step - loss: 0.0582 - acc: 0.9
Epoch 30/30
60000/60000 [==============================] - 5s 87us/step - loss: 0.0564 - acc: 0.9
```

```
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

score3 = model3.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score3[0])
print('Test accuracy:', score3[1])

fig3,ax3 = plt.subplots(1,1)
ax3.set_xlabel('epoch') ; ax3.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x3 = list(range(1,nb_epoch+1))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, valida

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of epochs


vy3 = history3.history['val_loss']
ty3 = history3.history['loss']
plt_dynamic(x3, vy3, ty3, ax3)
```

```
Test score: 0.05895621230599354
Test accuracy: 0.9823
```



MLP + BN + Dropout + AdamOptimizer + ReLu activation

> **Architecture 4**: 784-256-96-10;
> droput: .5,.5

std = sqrt(2/fan-in) as it is ReLu

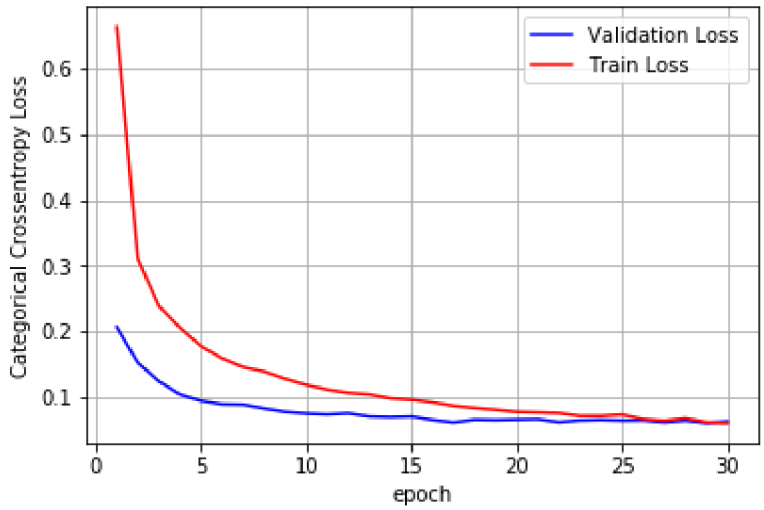
```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout
import warnings
warnings.filterwarnings('ignore')

model4 = Sequential()

model4.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mea
model4.add(BatchNormalization())
model4.add(Dropout(0.5))


model4.add(Dense(96, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.14433, seed
model4.add(BatchNormalization())
model4.add(Dropout(0.5))


model4.add(Dense(output_dim, activation='softmax'))
model4.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history4 = model4.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 10s 168us/step - loss: 0.6637 - acc: 0
Epoch 2/30
60000/60000 [==============================] - 5s 91us/step - loss: 0.3106 - acc: 0.9
Epoch 3/30
60000/60000 [==============================] - 5s 87us/step - loss: 0.2386 - acc: 0.9
Epoch 4/30
60000/60000 [==============================] - 5s 88us/step - loss: 0.2054 - acc: 0.9
Epoch 5/30
60000/60000 [==============================] - 5s 91us/step - loss: 0.1773 - acc: 0.9
Epoch 6/30
60000/60000 [==============================] - 5s 91us/step - loss: 0.1587 - acc: 0.9
Epoch 7/30
60000/60000 [==============================] - 5s 90us/step - loss: 0.1454 - acc: 0.9
Epoch 8/30
60000/60000 [==============================] - 5s 90us/step - loss: 0.1385 - acc: 0.9
Epoch 9/30
60000/60000 [==============================] - 5s 88us/step - loss: 0.1274 - acc: 0.9
Epoch 10/30
60000/60000 [==============================] - 5s 89us/step - loss: 0.1181 - acc: 0.9
Epoch 11/30
60000/60000 [==============================] - 5s 89us/step - loss: 0.1108 - acc: 0.9
Epoch 12/30
60000/60000 [==============================] - 5s 89us/step - loss: 0.1060 - acc: 0.9
Epoch 13/30
60000/60000 [==============================] - 5s 90us/step - loss: 0.1036 - acc: 0.9
Epoch 14/30
60000/60000 [==============================] - 5s 89us/step - loss: 0.0976 - acc: 0.9
Epoch 15/30
60000/60000 [==============================] - 5s 90us/step - loss: 0.0956 - acc: 0.9
Epoch 16/30
60000/60000 [==============================] - 5s 89us/step - loss: 0.0915 - acc: 0.9
Epoch 17/30
60000/60000 [==============================] - 6s 93us/step - loss: 0.0862 - acc: 0.9
Epoch 18/30
60000/60000 [==============================] - 5s 89us/step - loss: 0.0829 - acc: 0.9
Epoch 19/30
60000/60000 [==============================] - 5s 89us/step - loss: 0.0803 - acc: 0.9
Epoch 20/30
60000/60000 [==============================] - 5s 92us/step - loss: 0.0772 - acc: 0.9
Epoch 21/30
60000/60000 [==============================] - 5s 88us/step - loss: 0.0763 - acc: 0.9
Epoch 22/30
60000/60000 [==============================] - 5s 91us/step - loss: 0.0753 - acc: 0.9
Epoch 23/30
60000/60000 [==============================] - 6s 93us/step - loss: 0.0710 - acc: 0.9
Epoch 24/30
60000/60000 [==============================] - 5s 90us/step - loss: 0.0710 - acc: 0.9
Epoch 25/30
60000/60000 [==============================] - 5s 87us/step - loss: 0.0730 - acc: 0.9
Epoch 26/30
60000/60000 [==============================] - 5s 89us/step - loss: 0.0659 - acc: 0.9
Epoch 27/30
60000/60000 [==============================] - 5s 90us/step - loss: 0.0634 - acc: 0.9
Epoch 28/30
60000/60000 [==============================] - 5s 88us/step - loss: 0.0674 - acc: 0.9
```

```
Epoch 29/30
60000/60000 [==============================] - 5s 87us/step - loss: 0.0611 - acc: 0.9
Epoch 30/30
60000/60000 [==============================] - 5s 90us/step - loss: 0.0598 - acc: 0.9
```

```
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

score4 = model4.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score4[0])
print('Test accuracy:', score4[1])

fig4,ax4 = plt.subplots(1,1)
ax4.set_xlabel('epoch') ; ax4.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x4 = list(range(1,nb_epoch+1))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, valida

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of epochs


vy4 = history4.history['val_loss']
ty4 = history4.history['loss']
plt_dynamic(x4, vy4, ty4, ax4)
```

```
Test score: 0.06231452802268032
Test accuracy: 0.9832
```



MLP + BN + Dropout + AdamOptimizer + ReLu activation

> **Part 2 : Here we use 3 Hidden Layers**

> **Architecture 5**: 784-512-392-128-10;
> droput: .5,.5

std = sqrt(2/fan-in) as it is ReLu

```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout
import warnings
warnings.filterwarnings('ignore')

model5 = Sequential()

model5.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mea
model5.add(BatchNormalization())
model5.add(Dropout(0.5))


model5.add(Dense(392, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.07142, see
model5.add(BatchNormalization())
model5.add(Dropout(0.5))

model5.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=
model5.add(BatchNormalization())
model5.add(Dropout(0.5))


model5.add(Dense(output_dim, activation='softmax'))
model5.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model5.summary()


history5 = model5.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_
```

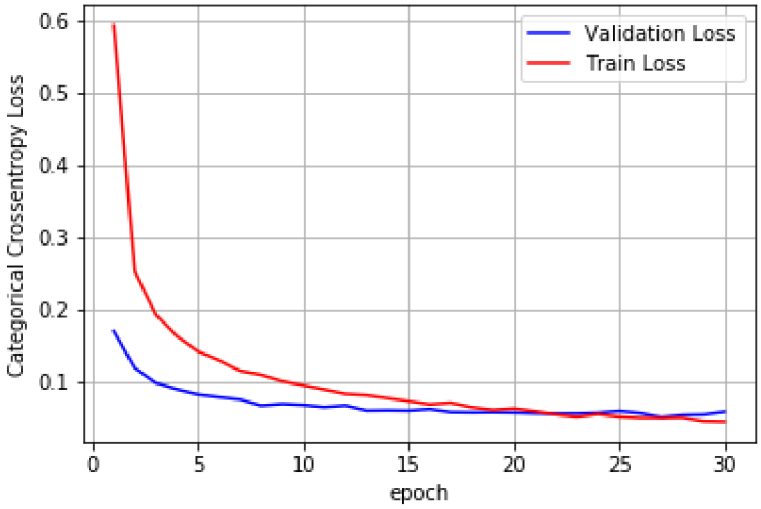
```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_56 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_39 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_39 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_57 (Dense)             (None, 392)               201096    
_________________________________________________________________
batch_normalization_40 (Batc (None, 392)               1568      
_________________________________________________________________
dropout_40 (Dropout)         (None, 392)               0         
_________________________________________________________________
dense_58 (Dense)             (None, 128)               50304     
_________________________________________________________________
batch_normalization_41 (Batc (None, 128)               512       
_________________________________________________________________
dropout_41 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_59 (Dense)             (None, 10)                1290      
=================================================================
Total params: 658,738
Trainable params: 656,674
Non-trainable params: 2,064
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 13s 212us/step - loss: 0.5931 - acc: 0
Epoch 2/30
60000/60000 [==============================] - 7s 123us/step - loss: 0.2522 - acc: 0.
Epoch 3/30
60000/60000 [==============================] - 7s 121us/step - loss: 0.1926 - acc: 0.
Epoch 4/30
60000/60000 [==============================] - 7s 123us/step - loss: 0.1629 - acc: 0.
Epoch 5/30
60000/60000 [==============================] - 7s 123us/step - loss: 0.1417 - acc: 0.
Epoch 6/30
60000/60000 [==============================] - 7s 122us/step - loss: 0.1293 - acc: 0.
Epoch 7/30
60000/60000 [==============================] - 7s 122us/step - loss: 0.1146 - acc: 0.
Epoch 8/30
60000/60000 [==============================] - 7s 122us/step - loss: 0.1092 - acc: 0.
Epoch 9/30
60000/60000 [==============================] - 7s 121us/step - loss: 0.1003 - acc: 0.
Epoch 10/30
60000/60000 [==============================] - 7s 123us/step - loss: 0.0945 - acc: 0.
Epoch 11/30
60000/60000 [==============================] - 7s 120us/step - loss: 0.0888 - acc: 0.
Epoch 12/30
60000/60000 [==============================] - 7s 120us/step - loss: 0.0827 - acc: 0.
Epoch 13/30
60000/60000 [==============================] - 7s 121us/step - loss: 0.0810 - acc: 0.
Epoch 14/30
60000/60000 [==============================] - 7s 121us/step - loss: 0.0770 - acc: 0.
```

```
Epoch 15/30
60000/60000 [==============================] - 7s 121us/step - loss: 0.0727 - acc: 0.
Epoch 16/30
60000/60000 [==============================] - 7s 122us/step - loss: 0.0682 - acc: 0.
Epoch 17/30
60000/60000 [==============================] - 7s 122us/step - loss: 0.0703 - acc: 0.
Epoch 18/30
60000/60000 [==============================] - 7s 123us/step - loss: 0.0641 - acc: 0.
Epoch 19/30
60000/60000 [==============================] - 7s 122us/step - loss: 0.0604 - acc: 0.
Epoch 20/30
60000/60000 [==============================] - 7s 123us/step - loss: 0.0624 - acc: 0.
Epoch 21/30
60000/60000 [==============================] - 7s 123us/step - loss: 0.0586 - acc: 0.
Epoch 22/30
60000/60000 [==============================] - 7s 122us/step - loss: 0.0543 - acc: 0.
Epoch 23/30
60000/60000 [==============================] - 7s 122us/step - loss: 0.0510 - acc: 0.
Epoch 24/30
60000/60000 [==============================] - 7s 123us/step - loss: 0.0549 - acc: 0.
Epoch 25/30
60000/60000 [==============================] - 7s 124us/step - loss: 0.0511 - acc: 0.
Epoch 26/30
60000/60000 [==============================] - 7s 122us/step - loss: 0.0496 - acc: 0.
Epoch 27/30
60000/60000 [==============================] - 7s 121us/step - loss: 0.0495 - acc: 0.
Epoch 28/30
60000/60000 [==============================] - 7s 121us/step - loss: 0.0498 - acc: 0.
Epoch 29/30
60000/60000 [==============================] - 7s 121us/step - loss: 0.0449 - acc: 0.
Epoch 30/30
60000/60000 [==============================] - 7s 123us/step - loss: 0.0442 - acc: 0.
```

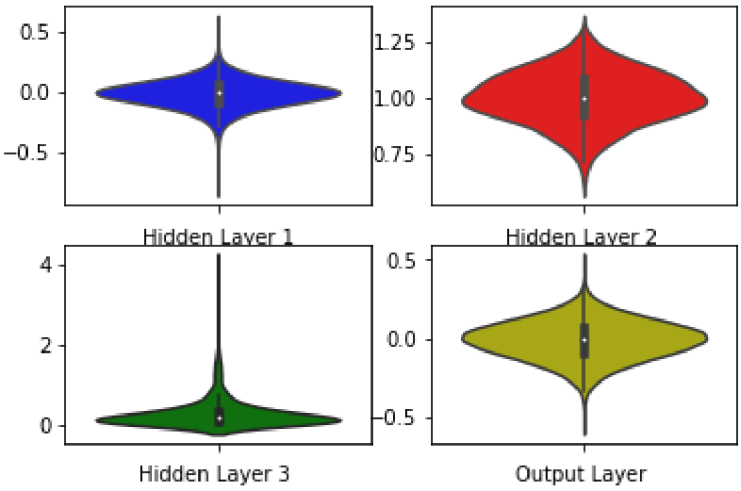
```
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

score5 = model5.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score5[0])
print('Test accuracy:', score5[1])

fig5,ax5 = plt.subplots(1,1)
ax5.set_xlabel('epoch') ; ax5.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x5 = list(range(1,nb_epoch+1))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, valida

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy
```

```
# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of epochs


vy5 = history5.history['val_loss']
ty5 = history5.history['loss']
plt_dynamic(x5, vy5, ty5, ax5)
```

```
Test score: 0.057912785604948294
Test accuracy: 0.9846
```




```
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

w_after5 = model5.get_weights()

h1_w = w_after5[0].flatten().reshape(-1,1)
h2_w = w_after5[2].flatten().reshape(-1,1)
h3_w = w_after5[4].flatten().reshape(-1,1)
out_w = w_after5[6].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(2, 2, 1)
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(2, 2, 2)
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(2, 2, 3)
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(2, 2, 4)
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



> **MLP + BN + Dropout + AdamOptimizer + ReLu activation**

> **Part 2 : Here we use 3 Hidden Layers**

> **Architecture 6**: 784-600-300-100-10;
> droput: .5,.5

std = sqrt(2/fan-in) as it is ReLu

```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout
import warnings
warnings.filterwarnings('ignore')

model6 = Sequential()

model6.add(Dense(600, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mea
model6.add(BatchNormalization())
model6.add(Dropout(0.5))


model6.add(Dense(300, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.08164, see
model6.add(BatchNormalization())
model6.add(Dropout(0.5))

model6.add(Dense(100, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.14142, see
model6.add(BatchNormalization())
model6.add(Dropout(0.5))


model6.add(Dense(output_dim, activation='softmax'))
model6.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model6.summary()


history6 = model6.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_
```

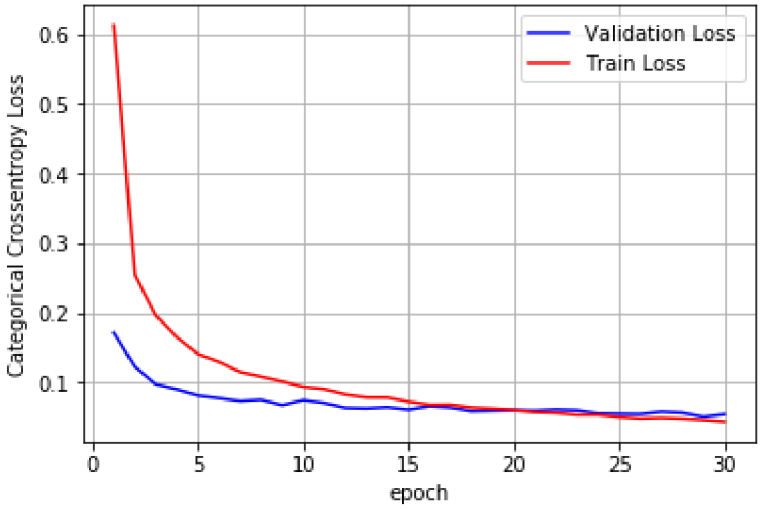
```
Model: "sequential_19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_60 (Dense)             (None, 600)               471000    
_________________________________________________________________
batch_normalization_42 (Batc (None, 600)               2400      
_________________________________________________________________
dropout_42 (Dropout)         (None, 600)               0         
_________________________________________________________________
dense_61 (Dense)             (None, 300)               180300    
_________________________________________________________________
batch_normalization_43 (Batc (None, 300)               1200      
_________________________________________________________________
dropout_43 (Dropout)         (None, 300)               0         
_________________________________________________________________
dense_62 (Dense)             (None, 100)               30100     
_________________________________________________________________
batch_normalization_44 (Batc (None, 100)               400       
_________________________________________________________________
dropout_44 (Dropout)         (None, 100)               0         
_________________________________________________________________
dense_63 (Dense)             (None, 10)                1010      
=================================================================
Total params: 686,410
Trainable params: 684,410
Non-trainable params: 2,000
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 13s 221us/step - loss: 0.6130 - acc: 0
Epoch 2/30
60000/60000 [==============================] - 8s 125us/step - loss: 0.2548 - acc: 0.
Epoch 3/30
60000/60000 [==============================] - 8s 126us/step - loss: 0.1962 - acc: 0.
Epoch 4/30
60000/60000 [==============================] - 7s 124us/step - loss: 0.1651 - acc: 0.
Epoch 5/30
60000/60000 [==============================] - 8s 131us/step - loss: 0.1409 - acc: 0.
Epoch 6/30
60000/60000 [==============================] - 8s 126us/step - loss: 0.1297 - acc: 0.
Epoch 7/30
60000/60000 [==============================] - 7s 124us/step - loss: 0.1151 - acc: 0.
Epoch 8/30
60000/60000 [==============================] - 8s 126us/step - loss: 0.1085 - acc: 0.
Epoch 9/30
60000/60000 [==============================] - 8s 125us/step - loss: 0.1015 - acc: 0.
Epoch 10/30
60000/60000 [==============================] - 8s 126us/step - loss: 0.0937 - acc: 0.
Epoch 11/30
60000/60000 [==============================] - 8s 127us/step - loss: 0.0904 - acc: 0.
Epoch 12/30
60000/60000 [==============================] - 8s 125us/step - loss: 0.0829 - acc: 0.
Epoch 13/30
60000/60000 [==============================] - 7s 125us/step - loss: 0.0788 - acc: 0.
Epoch 14/30
60000/60000 [==============================] - 8s 125us/step - loss: 0.0786 - acc: 0.
```

```
Epoch 15/30
60000/60000 [==============================] - 7s 125us/step - loss: 0.0726 - acc: 0.
Epoch 16/30
60000/60000 [==============================] - 7s 124us/step - loss: 0.0678 - acc: 0.
Epoch 17/30
60000/60000 [==============================] - 8s 125us/step - loss: 0.0680 - acc: 0.
Epoch 18/30
60000/60000 [==============================] - 8s 126us/step - loss: 0.0638 - acc: 0.
Epoch 19/30
60000/60000 [==============================] - 7s 125us/step - loss: 0.0624 - acc: 0.
Epoch 20/30
60000/60000 [==============================] - 8s 125us/step - loss: 0.0603 - acc: 0.
Epoch 21/30
60000/60000 [==============================] - 8s 126us/step - loss: 0.0574 - acc: 0.
Epoch 22/30
60000/60000 [==============================] - 7s 123us/step - loss: 0.0565 - acc: 0.
Epoch 23/30
60000/60000 [==============================] - 7s 125us/step - loss: 0.0537 - acc: 0.
Epoch 24/30
60000/60000 [==============================] - 7s 125us/step - loss: 0.0537 - acc: 0.
Epoch 25/30
60000/60000 [==============================] - 7s 125us/step - loss: 0.0503 - acc: 0.
Epoch 26/30
60000/60000 [==============================] - 7s 124us/step - loss: 0.0482 - acc: 0.
Epoch 27/30
60000/60000 [==============================] - 7s 124us/step - loss: 0.0491 - acc: 0.
Epoch 28/30
60000/60000 [==============================] - 7s 124us/step - loss: 0.0480 - acc: 0.
Epoch 29/30
60000/60000 [==============================] - 8s 127us/step - loss: 0.0461 - acc: 0.
Epoch 30/30
60000/60000 [==============================] - 8s 126us/step - loss: 0.0437 - acc: 0.
```

```
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

score6 = model6.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score6[0])
print('Test accuracy:', score6[1])

fig6,ax6 = plt.subplots(1,1)
ax6.set_xlabel('epoch') ; ax6.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x6 = list(range(1,nb_epoch+1))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, valida

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy
```

```
# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of epochs


vy6 = history6.history['val_loss']
ty6 = history6.history['loss']
plt_dynamic(x6, vy6, ty6, ax6)
```

```
Test score: 0.05445547826328548
Test accuracy: 0.9857
```



> **MLP + BN + Dropout + AdamOptimizer + ReLu activation**

> **Part 2 : Here we use 3 Hidden Layers**

> **Architecture 7**: 784-512-456-128-10;
> droput: .5,.5

std = sqrt(2/fan-in) as it is ReLu

```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout
import warnings
warnings.filterwarnings('ignore')

model7 = Sequential()

model7.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mea
model7.add(BatchNormalization())
model7.add(Dropout(0.5))


model7.add(Dense(456, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.0662, seed
model7.add(BatchNormalization())
model7.add(Dropout(0.5))

model7.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=
model7.add(BatchNormalization())
model7.add(Dropout(0.5))


model7.add(Dense(output_dim, activation='softmax'))
model7.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

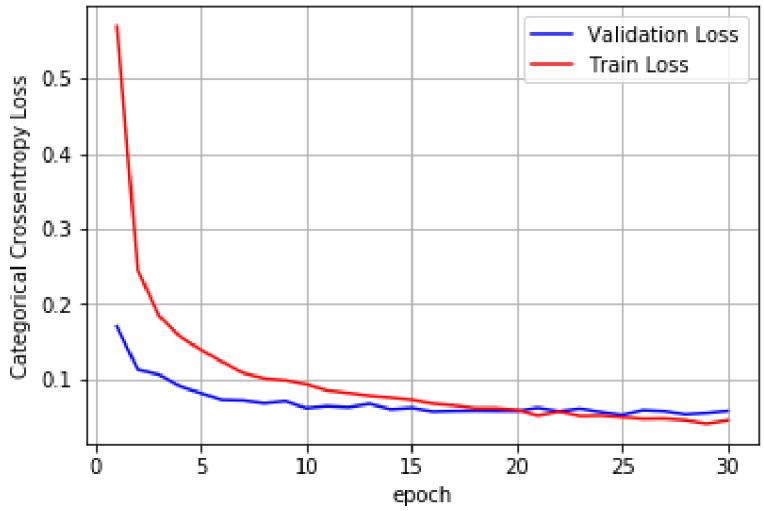
```
model7.summary()

history7 = model7.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_
```

```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_64 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_45 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_45 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_65 (Dense)             (None, 456)               233928    
_________________________________________________________________
batch_normalization_46 (Batc (None, 456)               1824      
_________________________________________________________________
dropout_46 (Dropout)         (None, 456)               0         
_________________________________________________________________
dense_66 (Dense)             (None, 128)               58496     
_________________________________________________________________
batch_normalization_47 (Batc (None, 128)               512       
_________________________________________________________________
dropout_47 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_67 (Dense)             (None, 10)                1290      
=================================================================
Total params: 700,018
Trainable params: 697,826
Non-trainable params: 2,192
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 14s 227us/step - loss: 0.5690 - acc: 0
Epoch 2/30
60000/60000 [==============================] - 7s 123us/step - loss: 0.2451 - acc: 0.
Epoch 3/30
60000/60000 [==============================] - 7s 124us/step - loss: 0.1847 - acc: 0.
Epoch 4/30
60000/60000 [==============================] - 7s 122us/step - loss: 0.1569 - acc: 0.
Epoch 5/30
60000/60000 [==============================] - 7s 124us/step - loss: 0.1392 - acc: 0.
Epoch 6/30
60000/60000 [==============================] - 7s 122us/step - loss: 0.1233 - acc: 0.
Epoch 7/30
60000/60000 [==============================] - 7s 122us/step - loss: 0.1084 - acc: 0.
Epoch 8/30
60000/60000 [==============================] - 7s 123us/step - loss: 0.1011 - acc: 0.
Epoch 9/30
60000/60000 [==============================] - 7s 123us/step - loss: 0.0986 - acc: 0.
Epoch 10/30
60000/60000 [==============================] - 7s 122us/step - loss: 0.0933 - acc: 0.
Epoch 11/30
60000/60000 [==============================] - 7s 122us/step - loss: 0.0850 - acc: 0.
Epoch 12/30
60000/60000 [==============================] - 7s 121us/step - loss: 0.0817 - acc: 0.
Epoch 13/30
60000/60000 [==============================] - 7s 121us/step - loss: 0.0780 - acc: 0.
Epoch 14/30
60000/60000 [==============================] - 7s 123us/step - loss: 0.0756 - acc: 0.
```

```
Epoch 15/30
60000/60000 [==============================] - 7s 125us/step - loss: 0.0727 - acc: 0.
Epoch 16/30
60000/60000 [==============================] - 8s 127us/step - loss: 0.0677 - acc: 0.
Epoch 17/30
60000/60000 [==============================] - 7s 123us/step - loss: 0.0653 - acc: 0.
Epoch 18/30
60000/60000 [==============================] - 7s 123us/step - loss: 0.0615 - acc: 0.
Epoch 19/30
60000/60000 [==============================] - 7s 123us/step - loss: 0.0615 - acc: 0.
Epoch 20/30
60000/60000 [==============================] - 7s 122us/step - loss: 0.0593 - acc: 0.
Epoch 21/30
60000/60000 [==============================] - 7s 124us/step - loss: 0.0517 - acc: 0.
Epoch 22/30
60000/60000 [==============================] - 7s 121us/step - loss: 0.0572 - acc: 0.
Epoch 23/30
60000/60000 [==============================] - 7s 122us/step - loss: 0.0511 - acc: 0.
Epoch 24/30
60000/60000 [==============================] - 7s 122us/step - loss: 0.0517 - acc: 0.
Epoch 25/30
60000/60000 [==============================] - 7s 123us/step - loss: 0.0496 - acc: 0.
Epoch 26/30
60000/60000 [==============================] - 7s 122us/step - loss: 0.0474 - acc: 0.
Epoch 27/30
60000/60000 [==============================] - 7s 123us/step - loss: 0.0478 - acc: 0.
Epoch 28/30
60000/60000 [==============================] - 7s 123us/step - loss: 0.0457 - acc: 0.
Epoch 29/30
60000/60000 [==============================] - 7s 122us/step - loss: 0.0411 - acc: 0.
Epoch 30/30
60000/60000 [==============================] - 7s 122us/step - loss: 0.0454 - acc: 0.
```

```
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

score7 = model7.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score7[0])
print('Test accuracy:', score7[1])

fig7,ax7 = plt.subplots(1,1)
ax7.set_xlabel('epoch') ; ax7.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x7 = list(range(1,nb_epoch+1))

vy7 = history7.history['val_loss']
ty7 = history7.history['loss']
plt_dynamic(x7, vy7, ty7, ax7)
```

```
Test score: 0.05791448878053052
Test accuracy: 0.9839
```



> **MLP + BN + Dropout + AdamOptimizer + ReLu activation**

> **Part 2 : Here we use 3 Hidden Layers**

> **Architecture 8**: 784-456-128-32-10;
> dropout: .5,.5

std = sqrt(2/fan-in) as it is ReLu

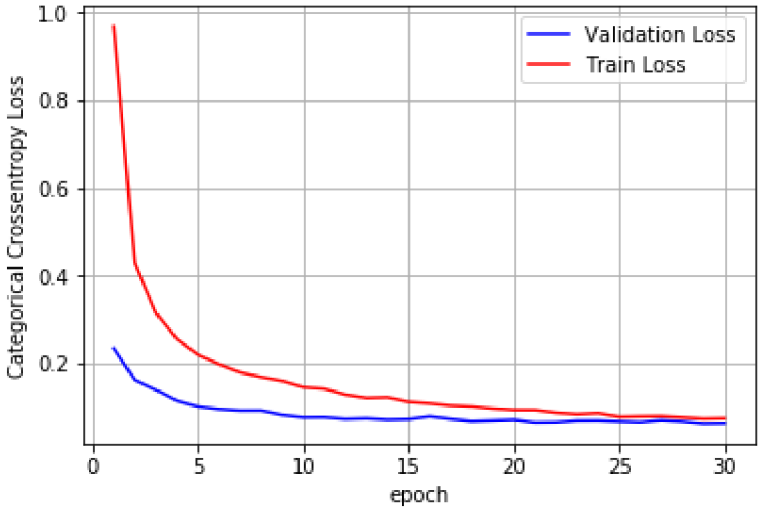
```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout
import warnings
warnings.filterwarnings('ignore')

model8 = Sequential()

model8.add(Dense(456, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mea
model8.add(BatchNormalization())
model8.add(Dropout(0.5))


model8.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=
model8.add(BatchNormalization())
model8.add(Dropout(0.5))

model8.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.25, seed=No
model8.add(BatchNormalization())
model8.add(Dropout(0.5))


model8.add(Dense(output_dim, activation='softmax'))
model8.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model8.summary()


history8 = model8.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_
```

```
Model: "sequential_21"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_68 (Dense)             (None, 456)               357960    
_________________________________________________________________
batch_normalization_48 (Batc (None, 456)               1824      
_________________________________________________________________
dropout_48 (Dropout)         (None, 456)               0         
_________________________________________________________________
dense_69 (Dense)             (None, 128)               58496     
_________________________________________________________________
batch_normalization_49 (Batc (None, 128)               512       
_________________________________________________________________
dropout_49 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_70 (Dense)             (None, 32)                4128      
_________________________________________________________________
batch_normalization_50 (Batc (None, 32)                128       
_________________________________________________________________
dropout_50 (Dropout)         (None, 32)                0         
_________________________________________________________________
dense_71 (Dense)             (None, 10)                330       
=================================================================
Total params: 423,378
Trainable params: 422,146
Non-trainable params: 1,232
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 14s 232us/step - loss: 0.9679 - acc: 0
Epoch 2/30
60000/60000 [==============================] - 7s 120us/step - loss: 0.4291 - acc: 0.
Epoch 3/30
60000/60000 [==============================] - 7s 121us/step - loss: 0.3162 - acc: 0.
Epoch 4/30
60000/60000 [==============================] - 7s 120us/step - loss: 0.2564 - acc: 0.
Epoch 5/30
60000/60000 [==============================] - 7s 118us/step - loss: 0.2209 - acc: 0.
Epoch 6/30
60000/60000 [==============================] - 7s 119us/step - loss: 0.1987 - acc: 0.
Epoch 7/30
60000/60000 [==============================] - 7s 117us/step - loss: 0.1805 - acc: 0.
Epoch 8/30
60000/60000 [==============================] - 7s 121us/step - loss: 0.1696 - acc: 0.
Epoch 9/30
60000/60000 [==============================] - 7s 118us/step - loss: 0.1608 - acc: 0.
Epoch 10/30
60000/60000 [==============================] - 7s 119us/step - loss: 0.1469 - acc: 0.
Epoch 11/30
60000/60000 [==============================] - 7s 121us/step - loss: 0.1432 - acc: 0.
Epoch 12/30
60000/60000 [==============================] - 7s 118us/step - loss: 0.1295 - acc: 0.
Epoch 13/30
60000/60000 [==============================] - 7s 123us/step - loss: 0.1219 - acc: 0.
Epoch 14/30
60000/60000 [==============================] - 7s 120us/step - loss: 0.1233 - acc: 0.
```

```
Epoch 15/30
60000/60000 [==============================] - 7s 119us/step - loss: 0.1133 - acc: 0.
Epoch 16/30
60000/60000 [==============================] - 7s 118us/step - loss: 0.1098 - acc: 0.
Epoch 17/30
60000/60000 [==============================] - 7s 120us/step - loss: 0.1049 - acc: 0.
Epoch 18/30
60000/60000 [==============================] - 7s 119us/step - loss: 0.1025 - acc: 0.
Epoch 19/30
60000/60000 [==============================] - 7s 121us/step - loss: 0.0978 - acc: 0.
Epoch 20/30
60000/60000 [==============================] - 7s 120us/step - loss: 0.0947 - acc: 0.
Epoch 21/30
60000/60000 [==============================] - 7s 120us/step - loss: 0.0945 - acc: 0.
Epoch 22/30
60000/60000 [==============================] - 7s 120us/step - loss: 0.0883 - acc: 0.
Epoch 23/30
60000/60000 [==============================] - 7s 117us/step - loss: 0.0850 - acc: 0.
Epoch 24/30
60000/60000 [==============================] - 7s 119us/step - loss: 0.0873 - acc: 0.
Epoch 25/30
60000/60000 [==============================] - 7s 120us/step - loss: 0.0792 - acc: 0.
Epoch 26/30
60000/60000 [==============================] - 7s 119us/step - loss: 0.0801 - acc: 0.
Epoch 27/30
60000/60000 [==============================] - 7s 124us/step - loss: 0.0806 - acc: 0.
Epoch 28/30
60000/60000 [==============================] - 7s 122us/step - loss: 0.0779 - acc: 0.
Epoch 29/30
60000/60000 [==============================] - 7s 120us/step - loss: 0.0750 - acc: 0.
Epoch 30/30
60000/60000 [==============================] - 7s 121us/step - loss: 0.0759 - acc: 0.
```

```
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

score8 = model8.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score8[0])
print('Test accuracy:', score8[1])

fig8,ax8 = plt.subplots(1,1)
ax8.set_xlabel('epoch') ; ax8.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x8 = list(range(1,nb_epoch+1))

vy8 = history8.history['val_loss']
ty8 = history8.history['loss']
plt_dynamic(x8, vy8, ty8, ax8)
```

```
Test score: 0.06451866653093893
Test accuracy: 0.9845
```



> **MLP + BN + Dropout + AdamOptimizer + ReLu activation**

> **Part 3 : Here we use 5 Hidden Layers**

> **Architecture 9**: 784-600-500-400-200-128-10;
> dropout: .5,.5

std = sqrt(2/fan-in) as it is ReLu

```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout
import warnings
warnings.filterwarnings('ignore')

model9 = Sequential()

model9.add(Dense(600, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mea
model9.add(BatchNormalization())
model9.add(Dropout(0.5))


model9.add(Dense(500, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.06324, see
model9.add(BatchNormalization())
model9.add(Dropout(0.5))

model9.add(Dense(400, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.0707, seed
model9.add(BatchNormalization())
model9.add(Dropout(0.5))

model9.add(Dense(300, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.08164, see
model9.add(BatchNormalization())
model9.add(Dropout(0.5))

model9.add(Dense(200, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.1, seed=No
model9.add(BatchNormalization())
model9.add(Dropout(0.5))

model9.add(Dense(output_dim, activation='softmax'))
model9.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
model9.summary()

history9 = model9.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_
```

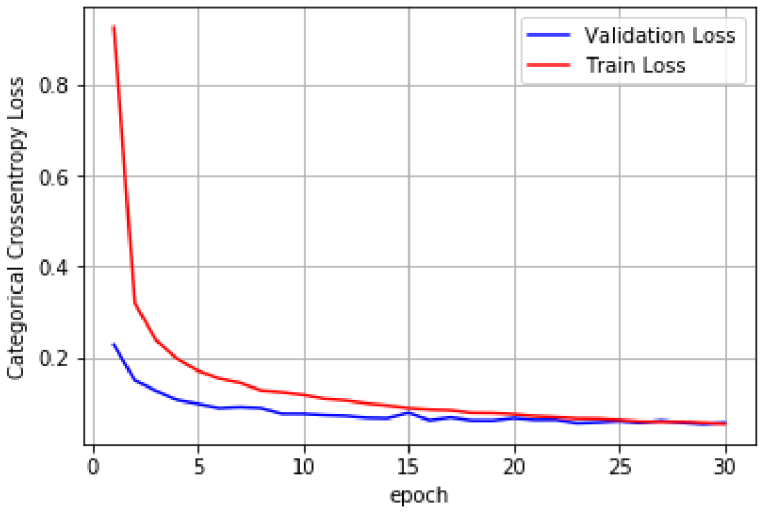
```
Model: "sequential_22"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_72 (Dense)             (None, 600)               471000    
_________________________________________________________________
batch_normalization_51 (Batc (None, 600)               2400      
_________________________________________________________________
dropout_51 (Dropout)         (None, 600)               0         
_________________________________________________________________
dense_73 (Dense)             (None, 500)               300500    
_________________________________________________________________
batch_normalization_52 (Batc (None, 500)               2000      
_________________________________________________________________
dropout_52 (Dropout)         (None, 500)               0         
_________________________________________________________________
dense_74 (Dense)             (None, 400)               200400    
_________________________________________________________________
batch_normalization_53 (Batc (None, 400)               1600      
_________________________________________________________________
dropout_53 (Dropout)         (None, 400)               0         
_________________________________________________________________
dense_75 (Dense)             (None, 300)               120300    
_________________________________________________________________
batch_normalization_54 (Batc (None, 300)               1200      
_________________________________________________________________
dropout_54 (Dropout)         (None, 300)               0         
_________________________________________________________________
dense_76 (Dense)             (None, 200)               60200     
_________________________________________________________________
batch_normalization_55 (Batc (None, 200)               800       
_________________________________________________________________
dropout_55 (Dropout)         (None, 200)               0         
_________________________________________________________________
dense_77 (Dense)             (None, 10)                2010      
=================================================================
Total params: 1,162,410
Trainable params: 1,158,410
Non-trainable params: 4,000
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 19s 312us/step - loss: 0.9268 - acc: 0
Epoch 2/30
60000/60000 [==============================] - 11s 183us/step - loss: 0.3195 - acc: 0
Epoch 3/30
60000/60000 [==============================] - 11s 182us/step - loss: 0.2385 - acc: 0
Epoch 4/30
60000/60000 [==============================] - 11s 183us/step - loss: 0.1976 - acc: 0
Epoch 5/30
60000/60000 [==============================] - 11s 184us/step - loss: 0.1706 - acc: 0
Epoch 6/30
60000/60000 [==============================] - 11s 183us/step - loss: 0.1541 - acc: 0
Epoch 7/30
60000/60000 [==============================] - 11s 184us/step - loss: 0.1441 - acc: 0
Epoch 8/30
60000/60000 [==============================] - 11s 184us/step - loss: 0.1272 - acc: 0
```

```
Epoch 9/30
60000/60000 [==============================] - 11s 182us/step - loss: 0.1234 - acc: 0
Epoch 10/30
60000/60000 [==============================] - 11s 183us/step - loss: 0.1179 - acc: 0
Epoch 11/30
60000/60000 [==============================] - 11s 183us/step - loss: 0.1090 - acc: 0
Epoch 12/30
60000/60000 [==============================] - 11s 182us/step - loss: 0.1054 - acc: 0
Epoch 13/30
60000/60000 [==============================] - 11s 182us/step - loss: 0.0987 - acc: 0
Epoch 14/30
60000/60000 [==============================] - 11s 183us/step - loss: 0.0941 - acc: 0
Epoch 15/30
60000/60000 [==============================] - 11s 184us/step - loss: 0.0886 - acc: 0
Epoch 16/30
60000/60000 [==============================] - 11s 185us/step - loss: 0.0853 - acc: 0
Epoch 17/30
60000/60000 [==============================] - 11s 183us/step - loss: 0.0838 - acc: 0
Epoch 18/30
60000/60000 [==============================] - 11s 182us/step - loss: 0.0780 - acc: 0
Epoch 19/30
60000/60000 [==============================] - 11s 180us/step - loss: 0.0775 - acc: 0
Epoch 20/30
60000/60000 [==============================] - 11s 184us/step - loss: 0.0749 - acc: 0
Epoch 21/30
60000/60000 [==============================] - 11s 185us/step - loss: 0.0706 - acc: 0
Epoch 22/30
60000/60000 [==============================] - 11s 182us/step - loss: 0.0685 - acc: 0
Epoch 23/30
60000/60000 [==============================] - 11s 184us/step - loss: 0.0655 - acc: 0
Epoch 24/30
60000/60000 [==============================] - 11s 183us/step - loss: 0.0651 - acc: 0
Epoch 25/30
60000/60000 [==============================] - 11s 186us/step - loss: 0.0622 - acc: 0
Epoch 26/30
60000/60000 [==============================] - 11s 188us/step - loss: 0.0589 - acc: 0
Epoch 27/30
60000/60000 [==============================] - 11s 184us/step - loss: 0.0581 - acc: 0
Epoch 28/30
60000/60000 [==============================] - 11s 182us/step - loss: 0.0583 - acc: 0
Epoch 29/30
60000/60000 [==============================] - 11s 182us/step - loss: 0.0565 - acc: 0
Epoch 30/30
60000/60000 [==============================] - 11s 184us/step - loss: 0.0537 - acc: 0
```

```
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

score9 = model9.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score9[0])
print('Test accuracy:', score9[1])
```

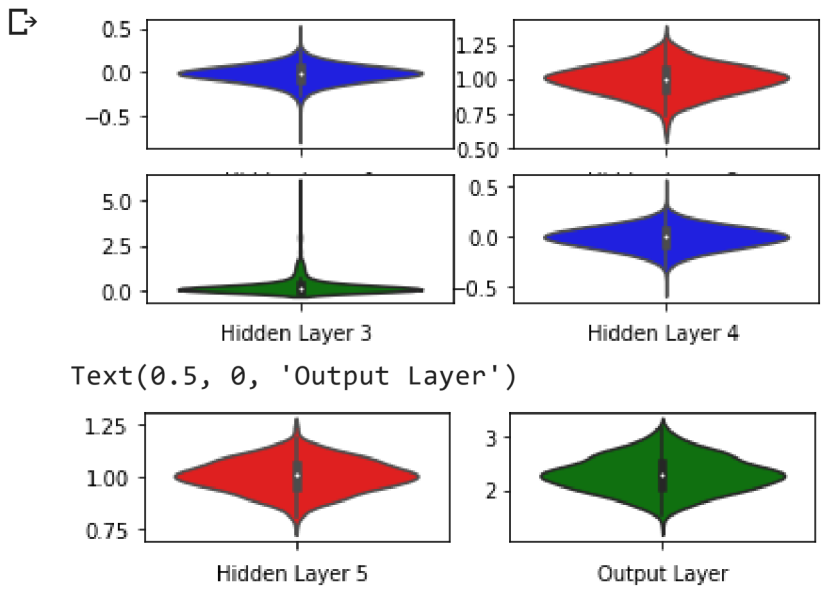
```
fig9,ax9 = plt.subplots(1,1)
ax9.set_xlabel('epoch') ; ax9.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x9 = list(range(1,nb_epoch+1))

vy9 = history9.history['val_loss']
ty9 = history9.history['loss']
plt_dynamic(x9, vy9, ty9, ax9)
```

```
Test score: 0.05667422412817832
Test accuracy: 0.9854
```




```
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

w_after9 = model9.get_weights()

h1_w = w_after9[0].flatten().reshape(-1,1)
h2_w = w_after9[2].flatten().reshape(-1,1)
h3_w = w_after9[4].flatten().reshape(-1,1)
h4_w = w_after9[6].flatten().reshape(-1,1)
h5_w = w_after9[8].flatten().reshape(-1,1)
out_w = w_after9[10].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(3, 2, 1)
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(3, 2, 2)
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(3, 2, 3)
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(3, 2, 4)
ax = sns.violinplot(y=h4_w,color='b')
plt.xlabel('Hidden Layer 4 ')
plt.show()
```

```
plt.subplot(3, 2, 5)
ax = sns.violinplot(y=h5_w, color='r')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(3, 2, 6)
ax = sns.violinplot(y=out_w, color='g')
plt.xlabel('Output Layer')
```




```
Text(0.5, 0, 'Output Layer')
```


> **MLP + BN + Dropout + AdamOptimizer + ReLu activation**

> **Part 3 : Here we use 5 Hidden Layers**

> **Architecture 10**: 784-512-456-392-256-128-10;
> droput: .5,.5

std = sqrt(2/fan-in) as it is ReLu

```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout
import warnings
warnings.filterwarnings('ignore')

model10 = Sequential()

model10.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(me
model10.add(BatchNormalization())
model10.add(Dropout(0.5))


model10.add(Dense(456, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.06622, se
model10.add(BatchNormalization())
model10.add(Dropout(0.5))

model10.add(Dense(392, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.07142, se
model10.add(BatchNormalization())
model10.add(Dropout(0.5))

model10.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.08838, se
model10.add(BatchNormalization())
model10.add(Dropout(0.5))
```

```
model10.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed
model10.add(BatchNormalization())
model10.add(Dropout(0.5))

model10.add(Dense(output_dim, activation='softmax'))
model10.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model10.summary()


history10 = model10.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validatio
```

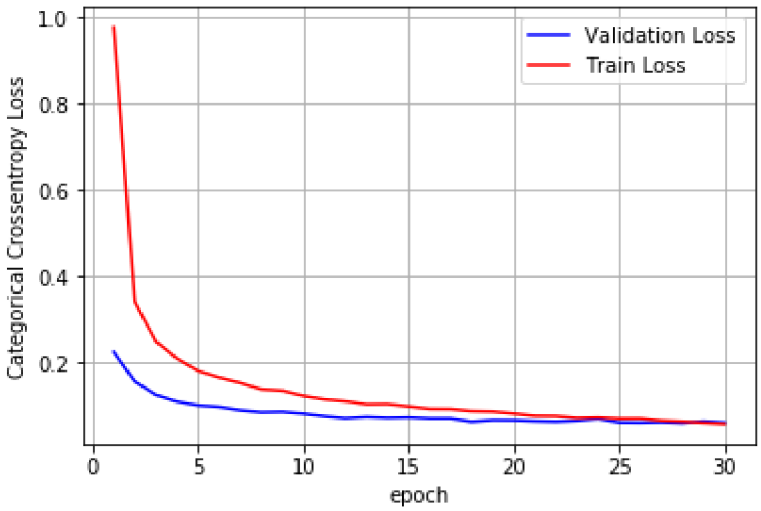
```
Model: "sequential_23"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_78 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_56 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_56 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_79 (Dense)             (None, 456)               233928    
_________________________________________________________________
batch_normalization_57 (Batc (None, 456)               1824      
_________________________________________________________________
dropout_57 (Dropout)         (None, 456)               0         
_________________________________________________________________
dense_80 (Dense)             (None, 392)               179144    
_________________________________________________________________
batch_normalization_58 (Batc (None, 392)               1568      
_________________________________________________________________
dropout_58 (Dropout)         (None, 392)               0         
_________________________________________________________________
dense_81 (Dense)             (None, 256)               100608    
_________________________________________________________________
batch_normalization_59 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_59 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_82 (Dense)             (None, 128)               32896     
_________________________________________________________________
batch_normalization_60 (Batc (None, 128)               512       
_________________________________________________________________
dropout_60 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_83 (Dense)             (None, 10)                1290      
=================================================================
Total params: 956,762
Trainable params: 953,274
Non-trainable params: 3,488
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 19s 314us/step - loss: 0.9767 - acc: 0
Epoch 2/30
60000/60000 [==============================] - 10s 173us/step - loss: 0.3401 - acc: 0
Epoch 3/30
60000/60000 [==============================] - 10s 173us/step - loss: 0.2488 - acc: 0
Epoch 4/30
60000/60000 [==============================] - 10s 175us/step - loss: 0.2093 - acc: 0
Epoch 5/30
60000/60000 [==============================] - 10s 174us/step - loss: 0.1804 - acc: 0
Epoch 6/30
60000/60000 [==============================] - 10s 174us/step - loss: 0.1651 - acc: 0
Epoch 7/30
60000/60000 [==============================] - 11s 177us/step - loss: 0.1526 - acc: 0
Epoch 8/30
60000/60000 [==============================] - 10s 173us/step - loss: 0.1375 - acc: 0
```

```
Epoch 9/30
60000/60000 [==============================] - 10s 174us/step - loss: 0.1343 - acc: 0
Epoch 10/30
60000/60000 [==============================] - 10s 173us/step - loss: 0.1224 - acc: 0
Epoch 11/30
60000/60000 [==============================] - 10s 171us/step - loss: 0.1142 - acc: 0
Epoch 12/30
60000/60000 [==============================] - 10s 171us/step - loss: 0.1098 - acc: 0
Epoch 13/30
60000/60000 [==============================] - 11s 178us/step - loss: 0.1027 - acc: 0
Epoch 14/30
60000/60000 [==============================] - 11s 176us/step - loss: 0.1032 - acc: 0
Epoch 15/30
60000/60000 [==============================] - 11s 176us/step - loss: 0.0977 - acc: 0
Epoch 16/30
60000/60000 [==============================] - 11s 176us/step - loss: 0.0920 - acc: 0
Epoch 17/30
60000/60000 [==============================] - 11s 176us/step - loss: 0.0917 - acc: 0
Epoch 18/30
60000/60000 [==============================] - 11s 176us/step - loss: 0.0869 - acc: 0
Epoch 19/30
60000/60000 [==============================] - 10s 172us/step - loss: 0.0859 - acc: 0
Epoch 20/30
60000/60000 [==============================] - 11s 176us/step - loss: 0.0813 - acc: 0
Epoch 21/30
60000/60000 [==============================] - 11s 175us/step - loss: 0.0759 - acc: 0
Epoch 22/30
60000/60000 [==============================] - 11s 178us/step - loss: 0.0754 - acc: 0
Epoch 23/30
60000/60000 [==============================] - 11s 181us/step - loss: 0.0711 - acc: 0
Epoch 24/30
60000/60000 [==============================] - 11s 179us/step - loss: 0.0722 - acc: 0
Epoch 25/30
60000/60000 [==============================] - 11s 176us/step - loss: 0.0696 - acc: 0
Epoch 26/30
60000/60000 [==============================] - 11s 177us/step - loss: 0.0698 - acc: 0
Epoch 27/30
60000/60000 [==============================] - 11s 177us/step - loss: 0.0648 - acc: 0
Epoch 28/30
60000/60000 [==============================] - 11s 178us/step - loss: 0.0626 - acc: 0
Epoch 29/30
60000/60000 [==============================] - 10s 174us/step - loss: 0.0596 - acc: 0
Epoch 30/30
60000/60000 [==============================] - 11s 175us/step - loss: 0.0571 - acc: 0
```

```
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

score10 = model10.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score10[0])
print('Test accuracy:', score10[1])
```

```
fig10,ax10 = plt.subplots(1,1)
ax10.set_xlabel('epoch') ; ax10.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x10 = list(range(1,nb_epoch+1))

vy10 = history10.history['val_loss']
ty10 = history10.history['loss']
plt_dynamic(x10, vy10, ty10, ax10)
```

```
Test score: 0.06008605453777127
Test accuracy: 0.9857
```



S.No	Architecture	Score	Test Accuracy
1.	784-366-100-10	0.0578912589	0.9844
2.	784-456-128-10	0.0567212487	0.984
3.	784-256-128-10	0.058956212	0.9823
4.	784-256-96-10	0.062314528	0.9832
---	---	---	---
5.	784-366-100-10	0.057912785	0.9846
6.	784-600-300-100-10	0.054455478	0.9857
7.	784-512-456-128-10	0.057914488	0.9839
8.	784-456-128-32-10	0.06451866	0.9845
---	---	---	---
9.	-600-500-400-200-128-	0.05667422	0.9854
10.	-512-456-392-256-128-	0.06008605	0.9857

Model 1 ,2 ,3, 4 --> 2 Hidden Layers

Model 5 ,6 ,7, 8 --> 3 Hidden Layers

Model 9, 10 --> 5 Hidden Layers

-

So in general we do notice as numbers of layer increase Test Accuracy do increase however MNIST dataset is so simp
be commenting on the basis of this. Also we see the dataset where first layer has lesser number of neurons compare
worser. Example whenever our first hidden layer is low like 256 or 366 (check model 3,4 and 5) accuracy is lower. Aga
and gain in accuracy is so nominal we should not be "concluding" right away but in general these two points we do n