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
```

```
Using TensorFlow backend.
```

```
%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()
```

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 2s 0us/step
```

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

```
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_tra
print("Number of training 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 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]
```

```python
# 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.
        0.          0.          0.          0.          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.]
```

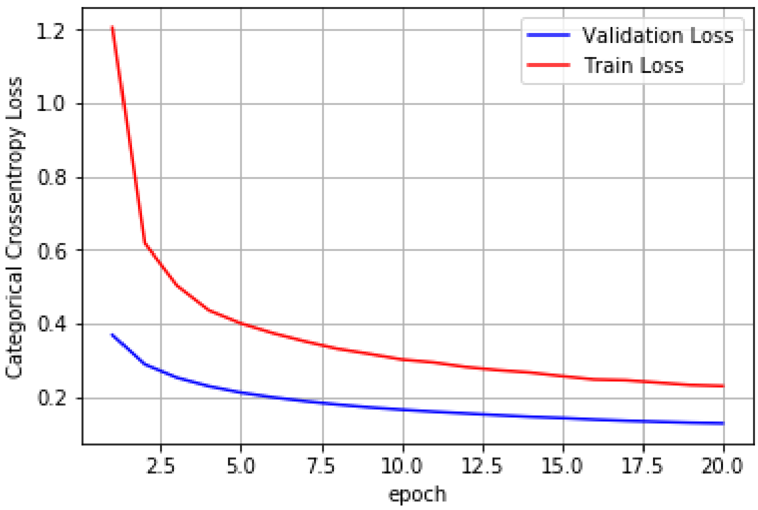
2-Hidden Layers

```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
from keras.initializers import he_normal

model_2 = Sequential()

model_2.add(Dense(464, activation='relu', input_dim=784, kernel_initializer=RandomNormal(mean=0.0
model_2.add(BatchNormalization())
model_2.add(Dropout(0.5))

model_2.add(Dense(184, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.05,
model_2.add(BatchNormalization())
model_2.add(Dropout(0.5))

model_2.add(Dense(10, activation='softmax'))


model_2.summary()

model_2.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_2.fit(X_train, Y_train, batch_size=256, epochs=20, verbose=1, validation_data=(X_


score = model_2.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(1,21))

# 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, v
```

```
# 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)

w_after = model_2.get_weights()
for i in range (0,len(w_after)):
  print(i,w_after[i].shape)
```

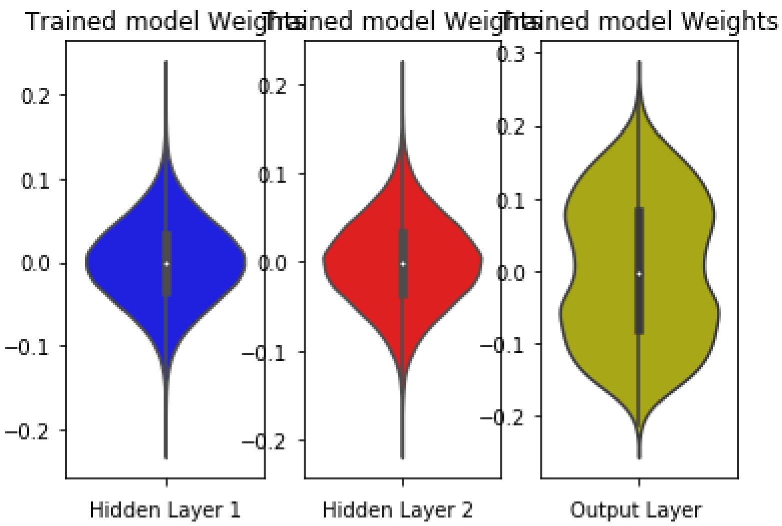
```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 464)               364240    
_________________________________________________________________
batch_normalization_1 (Batch (None, 464)               1856      
_________________________________________________________________
dropout_1 (Dropout)          (None, 464)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 184)               85560     
_________________________________________________________________
batch_normalization_2 (Batch (None, 184)               736       
_________________________________________________________________
dropout_2 (Dropout)          (None, 184)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1850      
=================================================================
Total params: 454,242
Trainable params: 452,946
Non-trainable params: 1,296
_________________________________________________________________
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:79

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/
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 [==============================] - 7s 116us/step - loss: 1.2065 - acc: 0.
Epoch 2/20
60000/60000 [==============================] - 2s 38us/step - loss: 0.6186 - acc: 0.8
Epoch 3/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.5041 - acc: 0.8
Epoch 4/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.4364 - acc: 0.8
Epoch 5/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.4008 - acc: 0.8
Epoch 6/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.3742 - acc: 0.8
Epoch 7/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.3518 - acc: 0.8
Epoch 8/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.3318 - acc: 0.8
Epoch 9/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.3178 - acc: 0.9
Epoch 10/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.3024 - acc: 0.9
```

```
Epoch 11/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.2944 - acc: 0.9
Epoch 12/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.2820 - acc: 0.9
Epoch 13/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.2734 - acc: 0.9
Epoch 14/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.2669 - acc: 0.9
Epoch 15/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.2570 - acc: 0.9
Epoch 16/20
60000/60000 [==============================] - 2s 38us/step - loss: 0.2480 - acc: 0.9
Epoch 17/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.2458 - acc: 0.9
Epoch 18/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.2392 - acc: 0.9
Epoch 19/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.2328 - acc: 0.9
Epoch 20/20
60000/60000 [==============================] - 2s 38us/step - loss: 0.2301 - acc: 0.9
Test score: 0.12829583922997118
Test accuracy: 0.9591
0 (784, 464)
1 (464,)
2 (464,)
3 (464,)
4 (464,)
5 (464,)
6 (464, 184)
7 (184,)
8 (184,)
9 (184,)
10 (184,)
11 (184,)
12 (184, 10)
13 (10,)
```




```
w_after = model_2.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[6].flatten().reshape(-1,1)
out_w = w_after[12].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()
```

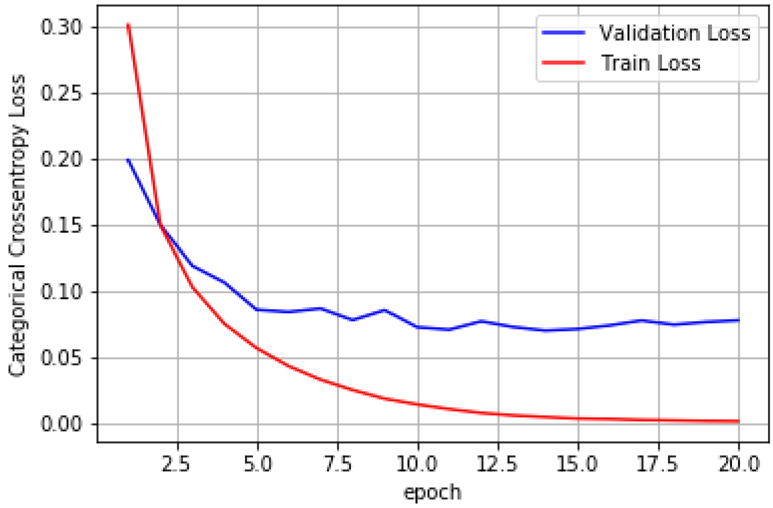



```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
from keras.initializers import glorot_normal

model_2 = Sequential()

model_2.add(Dense(660, activation='sigmoid', input_dim=784, kernel_initializer=glorot_normal(seec
model_2.add(BatchNormalization())


model_2.add(Dense(240, activation='sigmoid', kernel_initializer=glorot_normal(seed=None)))
model_2.add(BatchNormalization())


model_2.add(Dense(10, activation='softmax'))


model_2.summary()

model_2.compile(optimizer='adadelta', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_2.fit(X_train, Y_train, batch_size=256, epochs=20, verbose=1, validation_data=(X_


score = model_2.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(1,21))
```

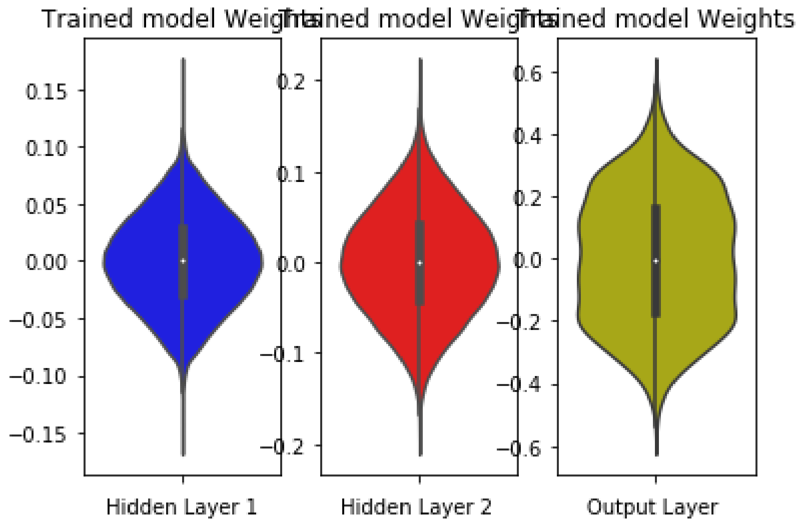
```
# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, v

# 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)

w_after = model_2.get_weights()
for i in range (0,len(w_after)):
  print(i,w_after[i].shape)
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 660)               518100    
_________________________________________________________________
batch_normalization_3 (Batch (None, 660)               2640      
_________________________________________________________________
dense_5 (Dense)              (None, 240)               158640    
_________________________________________________________________
batch_normalization_4 (Batch (None, 240)               960       
_________________________________________________________________
dense_6 (Dense)              (None, 10)                2410      
=================================================================
Total params: 682,750
Trainable params: 680,950
Non-trainable params: 1,800
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.3014 - acc: 0.9
Epoch 2/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.1496 - acc: 0.9
Epoch 3/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.1030 - acc: 0.9
Epoch 4/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0753 - acc: 0.9
Epoch 5/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0570 - acc: 0.9
Epoch 6/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0433 - acc: 0.9
Epoch 7/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0330 - acc: 0.9
Epoch 8/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0252 - acc: 0.9
Epoch 9/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0186 - acc: 0.9
Epoch 10/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0142 - acc: 0.9
Epoch 11/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0107 - acc: 0.9
Epoch 12/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0078 - acc: 0.9
Epoch 13/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0059 - acc: 0.9
Epoch 14/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0047 - acc: 0.9
Epoch 15/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0035 - acc: 0.9
Epoch 16/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0032 - acc: 0.9
Epoch 17/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0025 - acc: 0.9
Epoch 18/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0021 - acc: 1.0
Epoch 19/20
60000/60000 [==============================] - 3s 49us/step - loss: 0.0017 - acc: 1.0
Epoch 20/20
60000/60000 [==============================] - 3s 50us/step - loss: 0.0015 - acc: 1.0
Test score: 0.07796066140190087
Test accuracy: 0.9783
```

```
0 (784, 660)
1 (660,)
2 (660,)
3 (660,)
4 (660,)
5 (660,)
6 (660, 240)
7 (240,)
8 (240,)
9 (240,)
10 (240,)
11 (240,)
12 (240, 10)
13 (10,)
```




```
w_after = model_2.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[6].flatten().reshape(-1,1)
out_w = w_after[12].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()
```

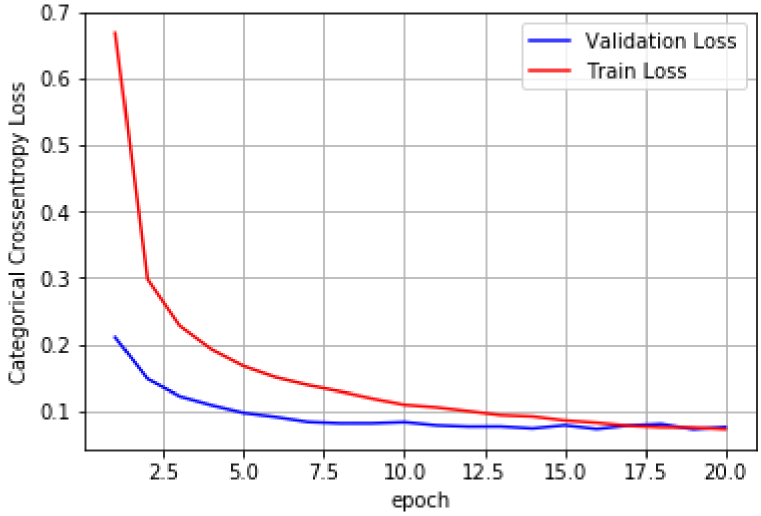



```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
from keras.initializers import he_normal

model_2 = Sequential()

model_2.add(Dense(320, activation='relu', input_dim=784, kernel_initializer=he_normal(seed=None))

model_2.add(Dropout(0.5))

model_2.add(Dense(80, activation='relu', kernel_initializer=he_normal(seed=None)))

model_2.add(Dropout(0.5))

model_2.add(Dense(10, activation='softmax'))


model_2.summary()

model_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_2.fit(X_train, Y_train, batch_size=256, epochs=20, verbose=1, validation_data=(X_


score = model_2.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(1,21))

# 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, v

# 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)
```

```
w_after = model_2.get_weights()
for i in range (0,len(w_after)):
  print(i,w_after[i].shape)
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_7 (Dense)              (None, 320)               251200    
_________________________________________________________________
dropout_3 (Dropout)          (None, 320)               0         
_________________________________________________________________
dense_8 (Dense)              (None, 80)                25680     
_________________________________________________________________
dropout_4 (Dropout)          (None, 80)                0         
_________________________________________________________________
dense_9 (Dense)              (None, 10)                810       
=================================================================
Total params: 277,690
Trainable params: 277,690
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.6689 - acc: 0.7
Epoch 2/20
60000/60000 [==============================] - 2s 27us/step - loss: 0.2981 - acc: 0.9
Epoch 3/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.2279 - acc: 0.9
Epoch 4/20
60000/60000 [==============================] - 2s 27us/step - loss: 0.1921 - acc: 0.9
Epoch 5/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.1671 - acc: 0.9
Epoch 6/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.1503 - acc: 0.9
Epoch 7/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.1387 - acc: 0.9
Epoch 8/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.1291 - acc: 0.9
Epoch 9/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.1186 - acc: 0.9
Epoch 10/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.1091 - acc: 0.9
Epoch 11/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.1053 - acc: 0.9
Epoch 12/20
60000/60000 [==============================] - 2s 27us/step - loss: 0.0997 - acc: 0.9
Epoch 13/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.0937 - acc: 0.9
Epoch 14/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.0917 - acc: 0.9
Epoch 15/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.0857 - acc: 0.9
Epoch 16/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.0823 - acc: 0.9
Epoch 17/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.0776 - acc: 0.9
Epoch 18/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.0754 - acc: 0.9
Epoch 19/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.0750 - acc: 0.9
Epoch 20/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.0725 - acc: 0.9
Test score: 0.07571211329051075
Test accuracy: 0.9794
```

```
0 (784, 320)
1 (320,)
2 (320, 80)
3 (80,)
4 (80, 10)
5 (10,)
```

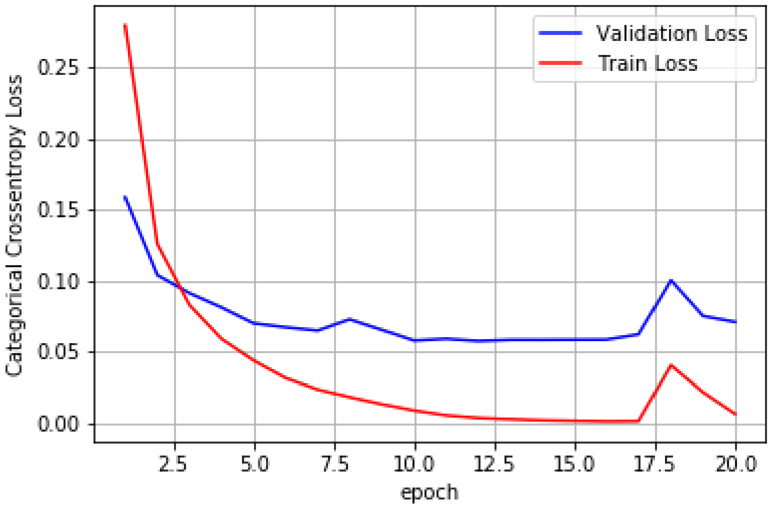



```
w_after = model_2.get_weights()

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()
```

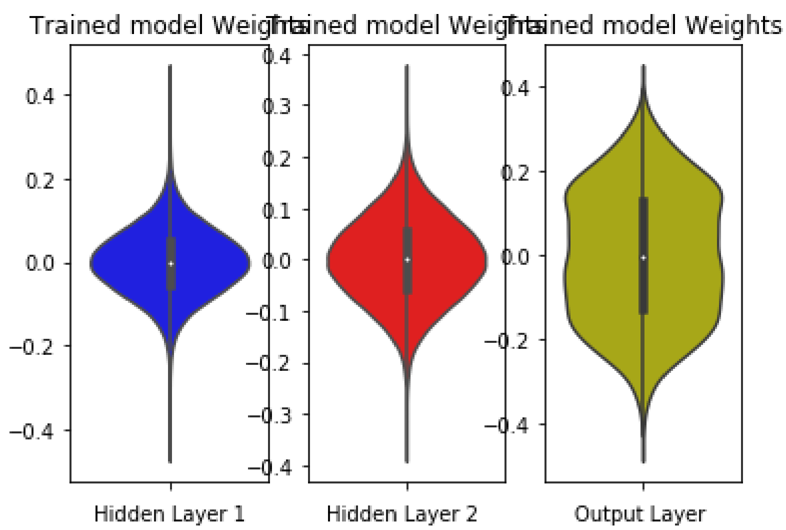
```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
from keras.initializers import he_normal

model_2 = Sequential()

model_2.add(Dense(712, activation='tanh', input_dim=784, kernel_initializer=he_normal( seed=None)


model_2.add(Dense(360, activation='tanh', kernel_initializer=he_normal( seed=None)))



model_2.add(Dense(10, activation='softmax'))


model_2.summary()

model_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_2.fit(X_train, Y_train, batch_size=256, epochs=20, verbose=1, validation_data=(X_


score = model_2.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(1,21))

# 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, v

# 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)

w_after = model_2.get_weights()
for i in range (0,len(w_after)):
  print(i,w_after[i].shape)
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 712)               558920
_________________________________________________________________
dense_11 (Dense)             (None, 360)               256680
_________________________________________________________________
dense_12 (Dense)             (None, 10)                3610
=================================================================
Total params: 819,210
Trainable params: 819,210
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.2793 - acc: 0.9
Epoch 2/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.1256 - acc: 0.9
Epoch 3/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0828 - acc: 0.9
Epoch 4/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0589 - acc: 0.9
Epoch 5/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0438 - acc: 0.9
Epoch 6/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0314 - acc: 0.9
Epoch 7/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0233 - acc: 0.9
Epoch 8/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.0179 - acc: 0.9
Epoch 9/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0129 - acc: 0.9
Epoch 10/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.0085 - acc: 0.9
Epoch 11/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0052 - acc: 0.9
Epoch 12/20
60000/60000 [==============================] - 2s 30us/step - loss: 0.0034 - acc: 0.9
Epoch 13/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0025 - acc: 1.0
Epoch 14/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0017 - acc: 1.0
Epoch 15/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0013 - acc: 1.0
Epoch 16/20
60000/60000 [==============================] - 2s 31us/step - loss: 9.5816e-04 - acc:
Epoch 17/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0011 - acc: 1.0
Epoch 18/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.0404 - acc: 0.9
Epoch 19/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.0213 - acc: 0.9
Epoch 20/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.0062 - acc: 0.9
Test score: 0.07075842707212796
Test accuracy: 0.9806
0 (784, 712)
1 (712,)
2 (712, 360)
3 (360,)
```

```
4 (360, 10)
5 (10,)
```




```
w_after = model_2.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()
```

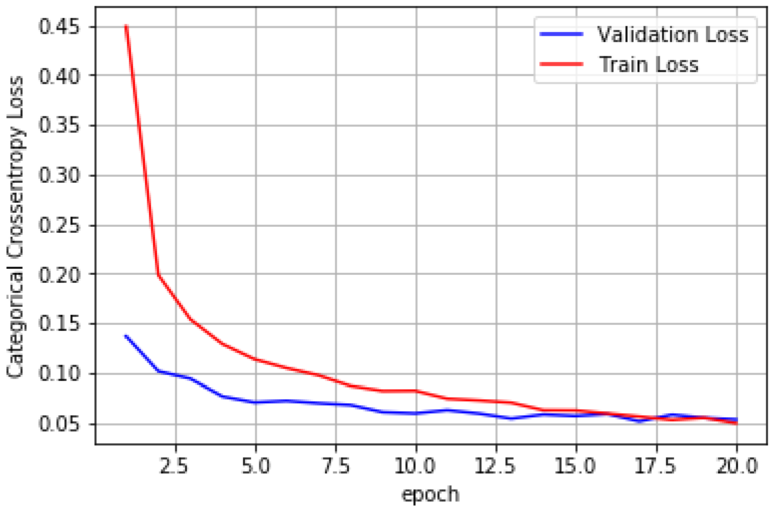


3-Hidden Layers

```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
from keras.initializers import he_normal

model_3 = Sequential()

model_3.add(Dense(720, activation='relu', input_dim=784, kernel_initializer=he_normal(seed=None)))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(540, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(360, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(10, activation='softmax'))


model_3.summary()

model_3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_3.fit(X_train, Y_train, batch_size=128, epochs=20, verbose=1, validation_data=(X_

score = model_3.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(1,21))

# 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, v

# 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)

w_after = model_3.get_weights()
for i in range (0,len(w_after)):
  print(i,w_after[i].shape)
```

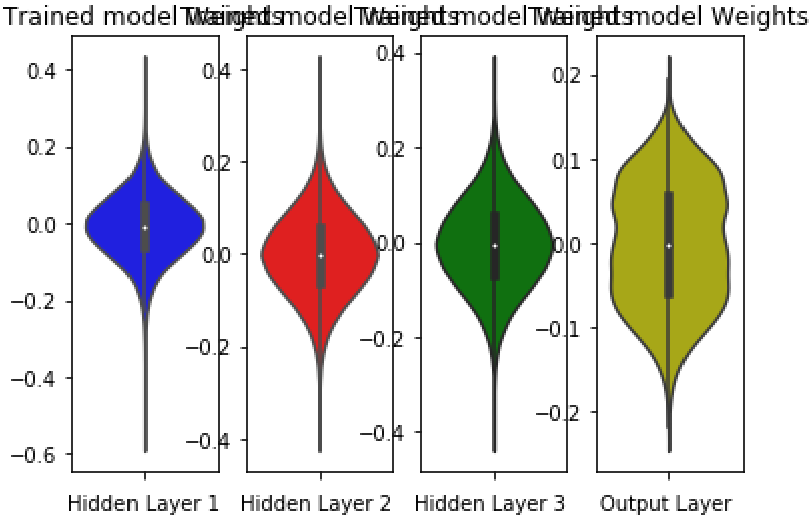
```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_13 (Dense)             (None, 720)               565200    
_________________________________________________________________
batch_normalization_5 (Batch (None, 720)               2880      
_________________________________________________________________
dropout_5 (Dropout)          (None, 720)               0         
_________________________________________________________________
dense_14 (Dense)             (None, 540)               389340    
_________________________________________________________________
batch_normalization_6 (Batch (None, 540)               2160      
_________________________________________________________________
dropout_6 (Dropout)          (None, 540)               0         
_________________________________________________________________
dense_15 (Dense)             (None, 360)               194760    
_________________________________________________________________
batch_normalization_7 (Batch (None, 360)               1440      
_________________________________________________________________
dropout_7 (Dropout)          (None, 360)               0         
_________________________________________________________________
dense_16 (Dense)             (None, 10)                3610      
=================================================================
Total params: 1,159,390
Trainable params: 1,156,150
Non-trainable params: 3,240
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.4488 - acc: 0.
Epoch 2/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.1985 - acc: 0.
Epoch 3/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.1539 - acc: 0.
Epoch 4/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.1290 - acc: 0.
Epoch 5/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.1137 - acc: 0.
Epoch 6/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.1048 - acc: 0.
Epoch 7/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.0975 - acc: 0.
Epoch 8/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.0869 - acc: 0.
Epoch 9/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0819 - acc: 0.
Epoch 10/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0821 - acc: 0.
Epoch 11/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0742 - acc: 0.
Epoch 12/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.0724 - acc: 0.
Epoch 13/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.0702 - acc: 0.
Epoch 14/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0626 - acc: 0.
Epoch 15/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0624 - acc: 0.
Epoch 16/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0593 - acc: 0.
```

```
Epoch 17/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0562 - acc: 0.
Epoch 18/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0528 - acc: 0.
Epoch 19/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.0550 - acc: 0.
Epoch 20/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.0495 - acc: 0.
Test score: 0.05327858295345068
Test accuracy: 0.9846
0 (784, 720)
1 (720,)
2 (720,)
3 (720,)
4 (720,)
5 (720,)
6 (720, 540)
7 (540,)
8 (540,)
9 (540,)
10 (540,)
11 (540,)
12 (540, 360)
13 (360,)
14 (360,)
15 (360,)
16 (360,)
17 (360,)
18 (360, 10)
19 (10,)
```




```
w_after = model_3.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[6].flatten().reshape(-1,1)
h3_w = w_after[12].flatten().reshape(-1,1)
out_w = w_after[18].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
```

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

plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```

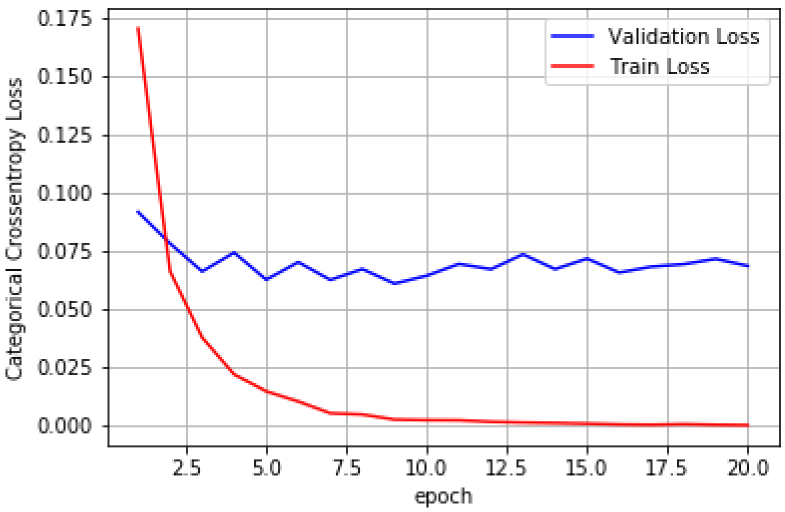



```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
from keras.initializers import he_normal

model_3 = Sequential()

model_3.add(Dense(640, activation='relu', input_dim=784, kernel_initializer=glorot_normal(seed=Nc
model_3.add(BatchNormalization())


model_3.add(Dense(480, activation='relu', kernel_initializer=glorot_normal(seed=None)) )
model_3.add(BatchNormalization())


model_3.add(Dense(160, activation='relu', kernel_initializer=glorot_normal(seed=None)) )
model_3.add(BatchNormalization())


model_3.add(Dense(10, activation='softmax'))


model_3.summary()

model_3.compile(optimizer='adadelta', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_3.fit(X_train, Y_train, batch_size=128, epochs=20, verbose=1, validation_data=(X_

score = model_3.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(1,21))

# 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, v

# 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)

w_after = model_3.get_weights()
for i in range (0,len(w_after)):
  print(i,w_after[i].shape)
```

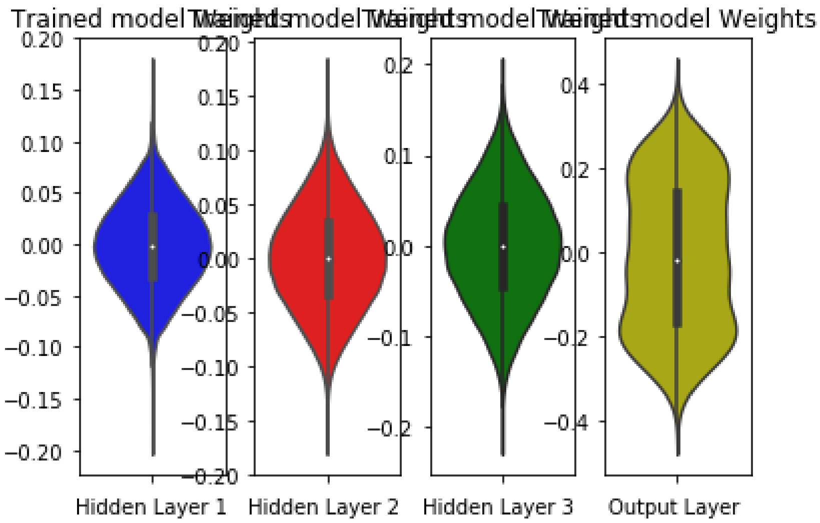
```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_17 (Dense)             (None, 640)               502400
_________________________________________________________________
batch_normalization_8 (Batch (None, 640)               2560
_________________________________________________________________
dense_18 (Dense)             (None, 480)               307680
_________________________________________________________________
batch_normalization_9 (Batch (None, 480)               1920
_________________________________________________________________
dense_19 (Dense)             (None, 160)               76960
_________________________________________________________________
batch_normalization_10 (Batc (None, 160)               640
_________________________________________________________________
dense_20 (Dense)             (None, 10)                1610
=================================================================
Total params: 893,770
Trainable params: 891,210
Non-trainable params: 2,560
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.1703 - acc: 0.
Epoch 2/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0662 - acc: 0.
Epoch 3/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0378 - acc: 0.
Epoch 4/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0218 - acc: 0.
Epoch 5/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0147 - acc: 0.
Epoch 6/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0104 - acc: 0.
Epoch 7/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0053 - acc: 0.
Epoch 8/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0048 - acc: 0.
Epoch 9/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.0026 - acc: 0.
Epoch 10/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0023 - acc: 0.
Epoch 11/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0023 - acc: 0.
Epoch 12/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0016 - acc: 0.
Epoch 13/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0013 - acc: 0.
Epoch 14/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0011 - acc: 0.
Epoch 15/20
60000/60000 [==============================] - 7s 119us/step - loss: 8.2971e-04 - acc
Epoch 16/20
60000/60000 [==============================] - 7s 119us/step - loss: 4.9791e-04 - acc
Epoch 17/20
60000/60000 [==============================] - 7s 121us/step - loss: 3.6205e-04 - acc
Epoch 18/20
60000/60000 [==============================] - 7s 119us/step - loss: 6.0406e-04 - acc
Epoch 19/20
60000/60000 [==============================] - 7s 119us/step - loss: 3.4281e-04 - acc
```

```
Epoch 20/20
60000/60000 [==============================] - 7s 121us/step - loss: 1.7719e-04 - acc
Test score: 0.06856017326894634
Test accuracy: 0.9854
0 (784, 640)
1 (640,)
2 (640,)
3 (640,)
4 (640,)
5 (640,)
6 (640, 480)
7 (480,)
8 (480,)
9 (480,)
10 (480,)
11 (480,)
12 (480, 160)
13 (160,)
14 (160,)
15 (160,)
16 (160,)
17 (160,)
18 (160, 10)
19 (10,)
```




```
w_after = model_3.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[6].flatten().reshape(-1,1)
h3_w = w_after[12].flatten().reshape(-1,1)
out_w = w_after[18].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
```

```
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```

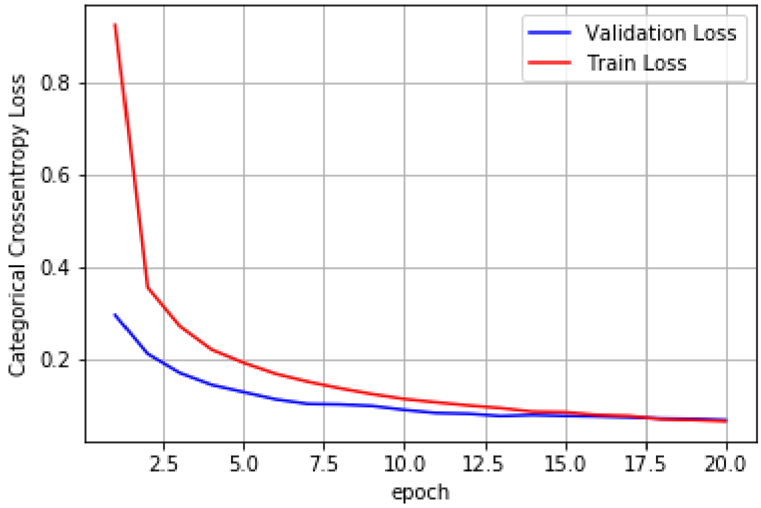



```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
from keras.initializers import he_normal

model_3 = Sequential()

model_3.add(Dense(564, activation='sigmoid', input_dim=784, kernel_initializer=he_normal(seed=Nor

model_3.add(Dropout(0.5))

model_3.add(Dense(324, activation='sigmoid', kernel_initializer=he_normal(seed=None)) )

model_3.add(Dropout(0.5))

model_3.add(Dense(144, activation='sigmoid', kernel_initializer=he_normal(seed=None)) )

model_3.add(Dropout(0.5))

model_3.add(Dense(10, activation='softmax'))


model_3.summary()

model_3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_3.fit(X_train, Y_train, batch_size=128, epochs=20, verbose=1, validation_data=(X_

score = model_3.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(1,21))

# 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, v

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

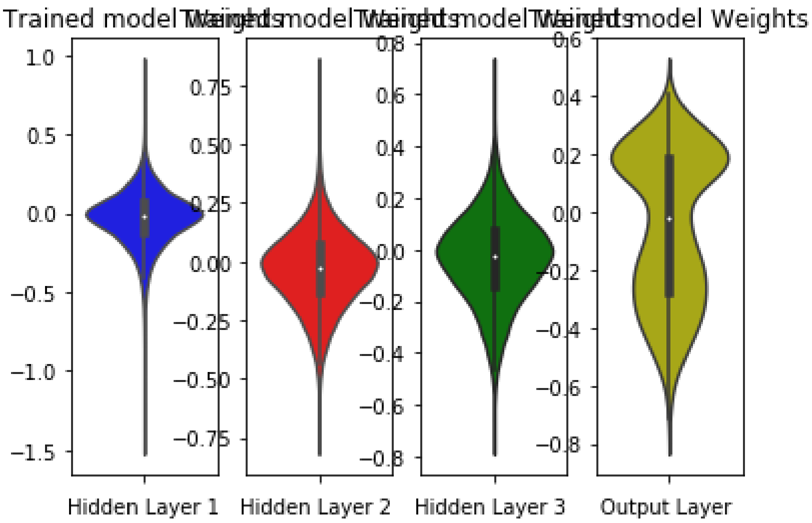
```
# val_acc : validation accuracy

# 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)

w_after = model_3.get_weights()
for i in range (0,len(w_after)):
  print(i,w_after[i].shape)
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_21 (Dense)             (None, 564)               442740    
_________________________________________________________________
dropout_8 (Dropout)          (None, 564)               0         
_________________________________________________________________
dense_22 (Dense)             (None, 324)               183060    
_________________________________________________________________
dropout_9 (Dropout)          (None, 324)               0         
_________________________________________________________________
dense_23 (Dense)             (None, 144)               46800     
_________________________________________________________________
dropout_10 (Dropout)         (None, 144)               0         
_________________________________________________________________
dense_24 (Dense)             (None, 10)                1450      
=================================================================
Total params: 674,050
Trainable params: 674,050
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.9243 - acc: 0.6
Epoch 2/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.3552 - acc: 0.8
Epoch 3/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.2712 - acc: 0.9
Epoch 4/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.2200 - acc: 0.9
Epoch 5/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.1913 - acc: 0.9
Epoch 6/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.1675 - acc: 0.9
Epoch 7/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.1504 - acc: 0.9
Epoch 8/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.1360 - acc: 0.9
Epoch 9/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.1235 - acc: 0.9
Epoch 10/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.1127 - acc: 0.9
Epoch 11/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.1055 - acc: 0.9
Epoch 12/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0988 - acc: 0.9
Epoch 13/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0931 - acc: 0.9
Epoch 14/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0853 - acc: 0.9
Epoch 15/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0843 - acc: 0.9
Epoch 16/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0782 - acc: 0.9
Epoch 17/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0762 - acc: 0.9
Epoch 18/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0688 - acc: 0.9
Epoch 19/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.0672 - acc: 0.9
```

```
Epoch 20/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0645 - acc: 0.9
Test score: 0.06716126223056344
Test accuracy: 0.9802
0 (784, 564)
1 (564,)
2 (564, 324)
3 (324,)
4 (324, 144)
5 (144,)
6 (144, 10)
7 (10,)
```




```
w_after = model_3.get_weights()

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


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```

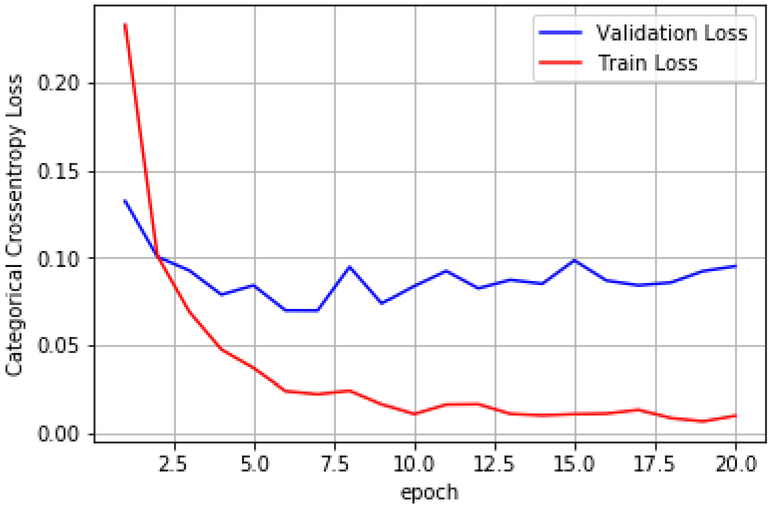



```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
from keras.initializers import he_normal

model_3 = Sequential()

model_3.add(Dense(480, activation='tanh', input_dim=784, kernel_initializer=he_normal(seed=None))


model_3.add(Dense(360, activation='tanh', kernel_initializer=he_normal(seed=None)) )


model_3.add(Dense(240, activation='tanh', kernel_initializer=he_normal(seed=None)) )


model_3.add(Dense(10, activation='softmax'))


model_3.summary()

model_3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_3.fit(X_train, Y_train, batch_size=128, epochs=20, verbose=1, validation_data=(X_

score = model_3.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(1,21))

# 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, v

# 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)
```

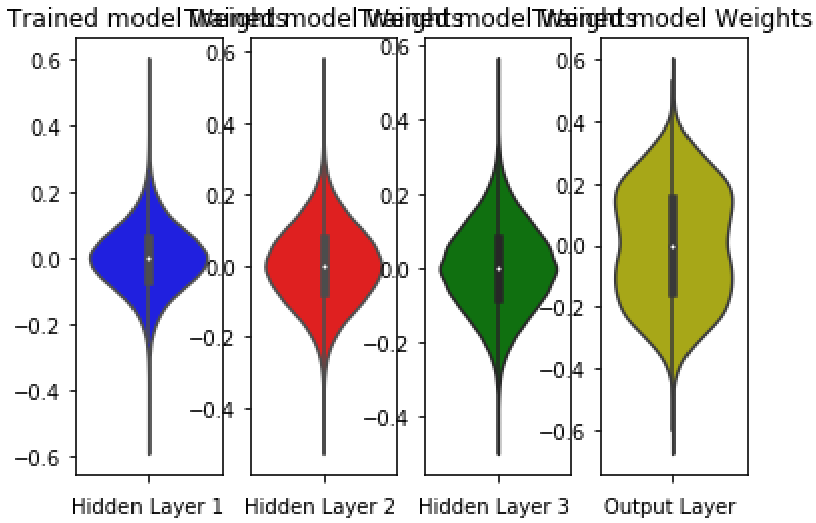
```
w_after = model_3.get_weights()
for i in range (0,len(w_after)):
  print(i,w_after[i].shape)
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_25 (Dense)             (None, 480)               376800
_________________________________________________________________
dense_26 (Dense)             (None, 360)               173160
_________________________________________________________________
dense_27 (Dense)             (None, 240)               86640
_________________________________________________________________
dense_28 (Dense)             (None, 10)                2410
=================================================================
Total params: 639,010
Trainable params: 639,010
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.2327 - acc: 0.9
Epoch 2/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.1010 - acc: 0.9
Epoch 3/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0690 - acc: 0.9
Epoch 4/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0474 - acc: 0.9
Epoch 5/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0369 - acc: 0.9
Epoch 6/20
60000/60000 [==============================] - 4s 61us/step - loss: 0.0239 - acc: 0.9
Epoch 7/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0221 - acc: 0.9
Epoch 8/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0241 - acc: 0.9
Epoch 9/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0163 - acc: 0.9
Epoch 10/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0108 - acc: 0.9
Epoch 11/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0162 - acc: 0.9
Epoch 12/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0164 - acc: 0.9
Epoch 13/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0109 - acc: 0.9
Epoch 14/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0100 - acc: 0.9
Epoch 15/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0108 - acc: 0.9
Epoch 16/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0111 - acc: 0.9
Epoch 17/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0132 - acc: 0.9
Epoch 18/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0085 - acc: 0.9
Epoch 19/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.0066 - acc: 0.9
Epoch 20/20
60000/60000 [==============================] - 4s 59us/step - loss: 0.0098 - acc: 0.9
Test score: 0.09506789306795617
Test accuracy: 0.9756
0 (784, 480)
1 (480,)
```

```
2 (480, 360)
3 (360,)
4 (360, 240)
5 (240,)
6 (240, 10)
7 (10,)
```




```
w_after = model_3.get_weights()

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


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 4, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 4, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 4, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 4, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



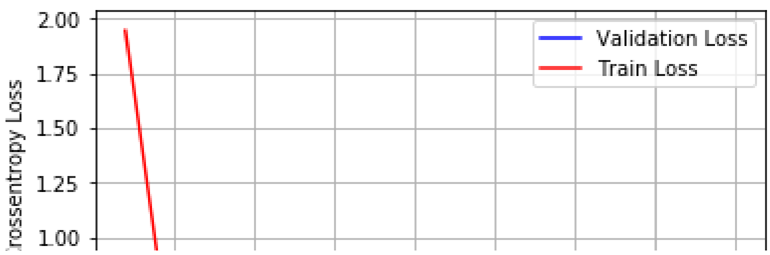
5-Hidden Layers

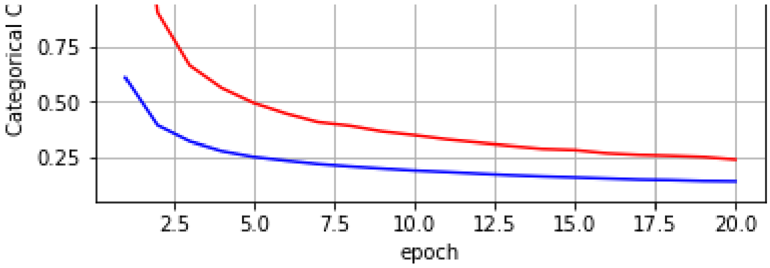
```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
from keras.initializers import he_normal

model_5 = Sequential()

model_5.add(Dense(720, activation='relu', input_dim=784, kernel_initializer=RandomNormal(mean=0.0
model_5.add(BatchNormalization())
model_5.add(Dropout(0.5))

model_5.add(Dense(540, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.05,
model_5.add(BatchNormalization())
model_5.add(Dropout(0.5))

model_5.add(Dense(360, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.05,
model_5.add(BatchNormalization())
model_5.add(Dropout(0.5))

model_5.add(Dense(240, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.05,
model_5.add(BatchNormalization())
model_5.add(Dropout(0.5))

model_5.add(Dense(120, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.05,
model_5.add(BatchNormalization())
model_5.add(Dropout(0.5))

model_5.add(Dense(10, activation='softmax'))


model_5.summary()

model_5.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_5.fit(X_train, Y_train, batch_size=128, epochs=20, verbose=1, validation_data=(X_



score = model_5.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(1,21))

# 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, v

# 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)


w_after = model_5.get_weights()
for i in range (0,len(w_after)):
  print(i,w_after[i].shape)
```

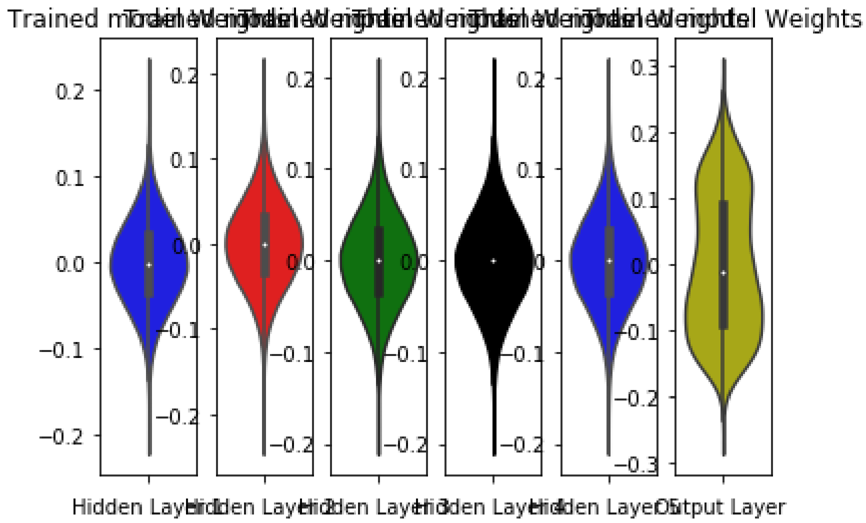
```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_29 (Dense)             (None, 720)               565200    
_________________________________________________________________
batch_normalization_11 (Batc (None, 720)               2880      
_________________________________________________________________
dropout_11 (Dropout)         (None, 720)               0         
_________________________________________________________________
dense_30 (Dense)             (None, 540)               389340    
_________________________________________________________________
batch_normalization_12 (Batc (None, 540)               2160      
_________________________________________________________________
dropout_12 (Dropout)         (None, 540)               0         
_________________________________________________________________
dense_31 (Dense)             (None, 360)               194760    
_________________________________________________________________
batch_normalization_13 (Batc (None, 360)               1440      
_________________________________________________________________
dropout_13 (Dropout)         (None, 360)               0         
_________________________________________________________________
dense_32 (Dense)             (None, 240)               86640     
_________________________________________________________________
batch_normalization_14 (Batc (None, 240)               960       
_________________________________________________________________
dropout_14 (Dropout)         (None, 240)               0         
_________________________________________________________________
dense_33 (Dense)             (None, 120)               28920     
_________________________________________________________________
batch_normalization_15 (Batc (None, 120)               480       
_________________________________________________________________
dropout_15 (Dropout)         (None, 120)               0         
_________________________________________________________________
dense_34 (Dense)             (None, 10)                1210      
=================================================================
Total params: 1,273,990
Trainable params: 1,270,030
Non-trainable params: 3,960
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 178us/step - loss: 1.9445 - acc: 0
Epoch 2/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.9082 - acc: 0.
Epoch 3/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.6674 - acc: 0.
Epoch 4/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.5639 - acc: 0.
Epoch 5/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.4957 - acc: 0.
Epoch 6/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.4476 - acc: 0.
Epoch 7/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.4085 - acc: 0.
Epoch 8/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.3916 - acc: 0.
Epoch 9/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.3660 - acc: 0.
Epoch 10/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.3489 - acc: 0.
```

```
Epoch 11/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.3302 - acc: 0.
Epoch 12/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.3156 - acc: 0.
Epoch 13/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.3006 - acc: 0.
Epoch 14/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.2875 - acc: 0.
Epoch 15/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.2826 - acc: 0.
Epoch 16/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.2680 - acc: 0.
Epoch 17/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.2608 - acc: 0.
Epoch 18/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.2565 - acc: 0.
Epoch 19/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.2513 - acc: 0.
Epoch 20/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.2395 - acc: 0.
Test score: 0.14006898400774226
Test accuracy: 0.9581
0 (784, 720)
1 (720,)
2 (720,)
3 (720,)
4 (720,)
5 (720,)
6 (720, 540)
7 (540,)
8 (540,)
9 (540,)
10 (540,)
11 (540,)
12 (540, 360)
13 (360,)
14 (360,)
15 (360,)
16 (360,)
17 (360,)
18 (360, 240)
19 (240,)
20 (240,)
21 (240,)
22 (240,)
23 (240,)
24 (240, 120)
25 (120,)
26 (120,)
27 (120,)
28 (120,)
29 (120,)
30 (120, 10)
31 (10,)
```






```
w_after = model_5.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[6].flatten().reshape(-1,1)
h3_w = w_after[12].flatten().reshape(-1,1)
h4_w = w_after[18].flatten().reshape(-1,1)
h5_w = w_after[24].flatten().reshape(-1,1)
out_w = w_after[30].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='0')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='b')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```

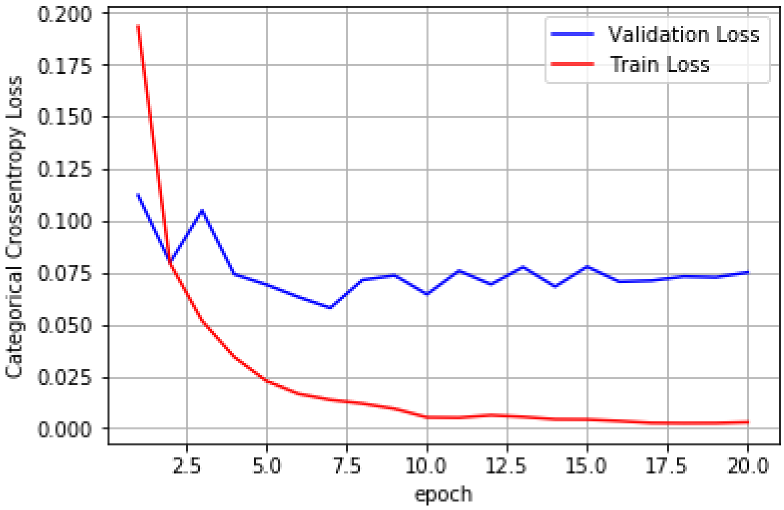


```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
from keras.initializers import he_normal

model_5 = Sequential()

model_5.add(Dense(600, activation='relu', input_dim=784, kernel_initializer=RandomNormal(mean=0.0
model_5.add(BatchNormalization())


model_5.add(Dense(500, activation='relu', kernel_initializer=glorot_normal( seed=None)))
model_5.add(BatchNormalization())


model_5.add(Dense(400, activation='relu', kernel_initializer=glorot_normal( seed=None)))
model_5.add(BatchNormalization())


model_5.add(Dense(300, activation='relu', kernel_initializer=glorot_normal( seed=None)) )
model_5.add(BatchNormalization())


model_5.add(Dense(200, activation='relu', kernel_initializer=glorot_normal( seed=None)))
model_5.add(BatchNormalization())


model_5.add(Dense(10, activation='softmax'))


model_5.summary()

model_5.compile(optimizer='adadelta', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_5.fit(X_train, Y_train, batch_size=128, epochs=20, verbose=1, validation_data=(X_




score = model_5.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(1,21))

# print(history.history.keys())
```

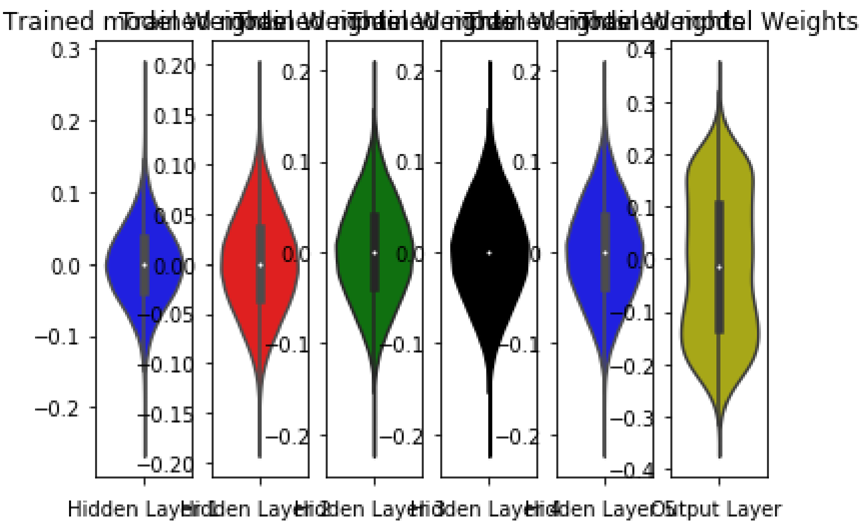
```
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, v

# 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)


w_after = model_5.get_weights()
for i in range (0,len(w_after)):
  print(i,w_after[i].shape)
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_35 (Dense)             (None, 600)               471000    
_________________________________________________________________
batch_normalization_16 (Batc (None, 600)               2400      
_________________________________________________________________
dense_36 (Dense)             (None, 500)               300500    
_________________________________________________________________
batch_normalization_17 (Batc (None, 500)               2000      
_________________________________________________________________
dense_37 (Dense)             (None, 400)               200400    
_________________________________________________________________
batch_normalization_18 (Batc (None, 400)               1600      
_________________________________________________________________
dense_38 (Dense)             (None, 300)               120300    
_________________________________________________________________
batch_normalization_19 (Batc (None, 300)               1200      
_________________________________________________________________
dense_39 (Dense)             (None, 200)               60200     
_________________________________________________________________
batch_normalization_20 (Batc (None, 200)               800       
_________________________________________________________________
dense_40 (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 1/20
60000/60000 [==============================] - 13s 209us/step - loss: 0.1933 - acc: 0
Epoch 2/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.0794 - acc: 0
Epoch 3/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.0517 - acc: 0
Epoch 4/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.0343 - acc: 0
Epoch 5/20
60000/60000 [==============================] - 11s 176us/step - loss: 0.0228 - acc: 0
Epoch 6/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.0166 - acc: 0
Epoch 7/20
60000/60000 [==============================] - 10s 173us/step - loss: 0.0137 - acc: 0
Epoch 8/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.0119 - acc: 0
Epoch 9/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.0094 - acc: 0
Epoch 10/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.0052 - acc: 0
Epoch 11/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.0051 - acc: 0
Epoch 12/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.0063 - acc: 0
Epoch 13/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.0055 - acc: 0
Epoch 14/20
60000/60000 [==============================] - 11s 180us/step - loss: 0.0043 - acc: 0
Epoch 15/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.0042 - acc: 0
```

```
Epoch 16/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.0035 - acc: 0
Epoch 17/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.0026 - acc: 0
Epoch 18/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.0025 - acc: 0
Epoch 19/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.0025 - acc: 0
Epoch 20/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.0029 - acc: 0
Test score: 0.07509704349693738
Test accuracy: 0.9836
0 (784, 600)
1 (600,)
2 (600,)
3 (600,)
4 (600,)
5 (600,)
6 (600, 500)
7 (500,)
8 (500,)
9 (500,)
10 (500,)
11 (500,)
12 (500, 400)
13 (400,)
14 (400,)
15 (400,)
16 (400,)
17 (400,)
18 (400, 300)
19 (300,)
20 (300,)
21 (300,)
22 (300,)
23 (300,)
24 (300, 200)
25 (200,)
26 (200,)
27 (200,)
28 (200,)
29 (200,)
30 (200, 10)
31 (10,)
```



```
w_after = model_5.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)
h2_w = w_after[6].flatten().reshape(-1,1)
h3_w = w_after[12].flatten().reshape(-1,1)
h4_w = w_after[18].flatten().reshape(-1,1)
h5_w = w_after[24].flatten().reshape(-1,1)
out_w = w_after[30].flatten().reshape(-1,1)


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='0')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='b')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```

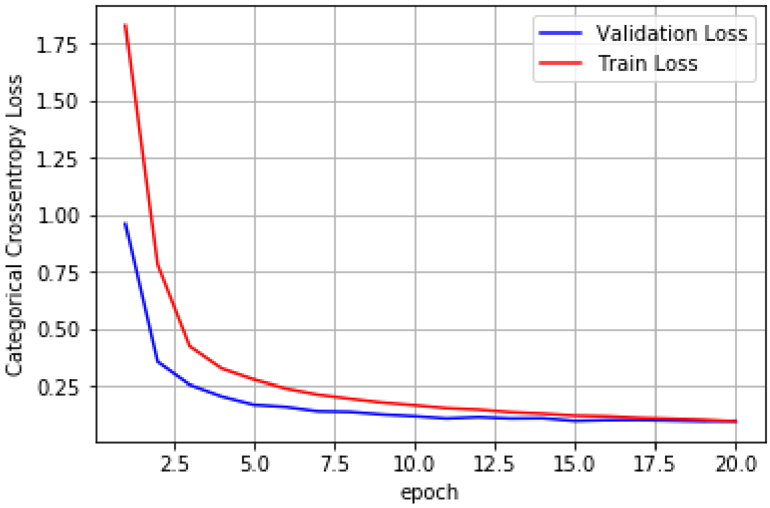



```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
from keras.initializers import he_normal

model_5 = Sequential()
```

```
model_5.add(Dense(500, activation='sigmoid', input_dim=784, kernel_initializer=glorot_normal( see

model_5.add(Dropout(0.5))

model_5.add(Dense(400, activation='sigmoid',kernel_initializer=glorot_normal( seed=None)))

model_5.add(Dropout(0.5))

model_5.add(Dense(300, activation='sigmoid', kernel_initializer=glorot_normal( seed=None)))

model_5.add(Dropout(0.5))

model_5.add(Dense(200, activation='sigmoid', kernel_initializer=glorot_normal( seed=None)))

model_5.add(Dropout(0.5))

model_5.add(Dense(100, activation='sigmoid', kernel_initializer=glorot_normal( seed=None)))

model_5.add(Dropout(0.5))

model_5.add(Dense(10, activation='softmax'))


model_5.summary()

model_5.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_5.fit(X_train, Y_train, batch_size=128, epochs=20, verbose=1, validation_data=(X_



score = model_5.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(1,21))

# 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, v

# 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)


w_after = model_5.get_weights()
for i in range (0,len(w_after)):
  print(i,w_after[i].shape)
```

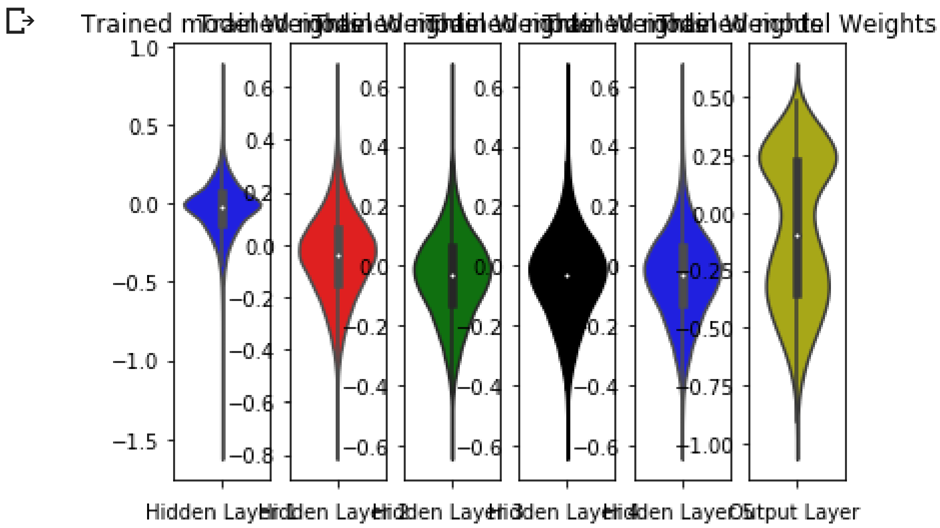
```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_41 (Dense)             (None, 500)               392500
_________________________________________________________________
dropout_16 (Dropout)         (None, 500)               0
_________________________________________________________________
dense_42 (Dense)             (None, 400)               200400
_________________________________________________________________
dropout_17 (Dropout)         (None, 400)               0
_________________________________________________________________
dense_43 (Dense)             (None, 300)               120300
_________________________________________________________________
dropout_18 (Dropout)         (None, 300)               0
_________________________________________________________________
dense_44 (Dense)             (None, 200)               60200
_________________________________________________________________
dropout_19 (Dropout)         (None, 200)               0
_________________________________________________________________
dense_45 (Dense)             (None, 100)               20100
_________________________________________________________________
dropout_20 (Dropout)         (None, 100)               0
_________________________________________________________________
dense_46 (Dense)             (None, 10)                1010
=================================================================
Total params: 794,510
Trainable params: 794,510
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 108us/step - loss: 1.8303 - acc: 0.
Epoch 2/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.7828 - acc: 0.7
Epoch 3/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.4237 - acc: 0.8
Epoch 4/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.3259 - acc: 0.9
Epoch 5/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.2778 - acc: 0.9
Epoch 6/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.2386 - acc: 0.9
Epoch 7/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.2122 - acc: 0.9
Epoch 8/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.1942 - acc: 0.9
Epoch 9/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.1778 - acc: 0.9
Epoch 10/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.1659 - acc: 0.9
Epoch 11/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.1530 - acc: 0.9
Epoch 12/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.1467 - acc: 0.9
Epoch 13/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.1355 - acc: 0.9
Epoch 14/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.1292 - acc: 0.9
Epoch 15/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.1203 - acc: 0.9
```

```
Epoch 16/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.1175 - acc: 0.9
Epoch 17/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.1103 - acc: 0.9
Epoch 18/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.1064 - acc: 0.9
Epoch 19/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.1021 - acc: 0.9
Epoch 20/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0947 - acc: 0.9
Test score: 0.09683186618569307
Test accuracy: 0.9763
0 (784, 500)
1 (500,)
2 (500, 400)
3 (400,)
4 (400, 300)
5 (300,)
6 (300, 200)
7 (200,)
8 (200, 100)
9 (100,)
10 (100, 10)
11 (10,)
```




```
w_after = model_5.get_weights()

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


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
```

```
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='0')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='b')
plt.xlabel('Hidden Layer 5 ')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```

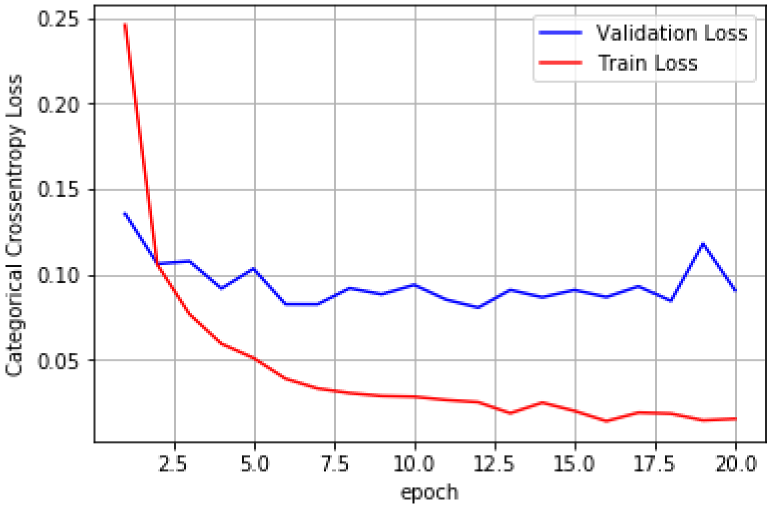



```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
from keras.initializers import he_normal

model_5 = Sequential()

model_5.add(Dense(480, activation='tanh', input_dim=784, kernel_initializer=RandomNormal(mean=0.0


model_5.add(Dense(360, activation='tanh', kernel_initializer=he_normal(seed=None)))


model_5.add(Dense(240, activation='tanh', kernel_initializer=he_normal(seed=None)))


model_5.add(Dense(120, activation='tanh', kernel_initializer=he_normal(seed=None)))


model_5.add(Dense(60, activation='tanh', kernel_initializer=he_normal(seed=None)))


model_5.add(Dense(10, activation='softmax'))


model_5.summary()

model_5.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_5.fit(X_train, Y_train, batch_size=128, epochs=20, verbose=1, validation_data=(X_
```

```
score = model_5.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(1,21))

# 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, v

# 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)


w_after = model_5.get_weights()
for i in range (0,len(w_after)):
  print(i,w_after[i].shape)
```

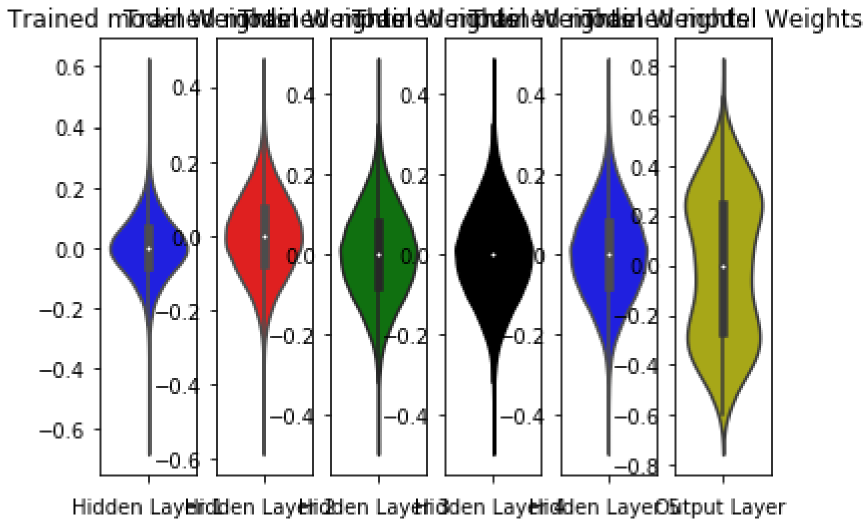
```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_47 (Dense)             (None, 480)               376800
_________________________________________________________________
dense_48 (Dense)             (None, 360)               173160
_________________________________________________________________
dense_49 (Dense)             (None, 240)               86640
_________________________________________________________________
dense_50 (Dense)             (None, 120)               28920
_________________________________________________________________
dense_51 (Dense)             (None, 60)                7260
_________________________________________________________________
dense_52 (Dense)             (None, 10)                610
=================================================================
Total params: 673,390
Trainable params: 673,390
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.2458 - acc: 0.9
Epoch 2/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.1055 - acc: 0.9
Epoch 3/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0766 - acc: 0.9
Epoch 4/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0592 - acc: 0.9
Epoch 5/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0509 - acc: 0.9
Epoch 6/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0388 - acc: 0.9
Epoch 7/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0331 - acc: 0.9
Epoch 8/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0303 - acc: 0.9
Epoch 9/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0286 - acc: 0.9
Epoch 10/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0282 - acc: 0.9
Epoch 11/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0263 - acc: 0.9
Epoch 12/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0250 - acc: 0.9
Epoch 13/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0186 - acc: 0.9
Epoch 14/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0248 - acc: 0.9
Epoch 15/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0199 - acc: 0.9
Epoch 16/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0139 - acc: 0.9
Epoch 17/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0189 - acc: 0.9
Epoch 18/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0183 - acc: 0.9
Epoch 19/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0144 - acc: 0.9
Epoch 20/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0153 - acc: 0.9
```

```
Test score: 0.09052809053522069
Test accuracy: 0.98
0 (784, 480)
1 (480,)
2 (480, 360)
3 (360,)
4 (360, 240)
5 (240,)
6 (240, 120)
7 (120,)
8 (120, 60)
9 (60,)
10 (60, 10)
11 (10,)
```




```
w_after = model_5.get_weights()

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


fig = plt.figure()
plt.title("Weight matrices after model trained")
plt.subplot(1, 6, 1)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.subplot(1, 6, 2)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='g')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='0')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w, color='b')
```

```
plt.xlabel('Hidden Layer 5 ')

plt.subplot(1, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```



Summary

```
#pretty table

from prettytable import PrettyTable
x = PrettyTable()

x.field_names = ["No of hidden layers","Batch_Normalization", "Activation_function", "optimizers"

x.add_row([2,"Y","Relu","sgd","Random Normal","464-184","Y",0.9614])
x.add_row([2,"Y","sigmoid","adadelta","Glorot Normal","660-240","N",0.9807])
x.add_row([2,"N","Relu","adam","He Normal","320-80","Y",0.98])
x.add_row([2,"N","tanh","adam","He Normal","712-360","N",0.9818])

x.add_row([3,"Y","Relu","sgd","Random Normal","720-540-360","Y",0.9842])
x.add_row([3,"Y","Relu","adadelta","Glorot Normal","640-480-160","N",0.9859])
x.add_row([3,"N","sigmoid","adam","He Normal","564-324-144","Y",0.981])
x.add_row([3,"N","tanh","adam","He Normal","480-360-240","N",0.9797])

x.add_row([5,"Y","Relu","sgd","Random Normal","720-540-360-240-120","Y",0.9588])
x.add_row([5,"Y","Relu","adadelta","Glorot Normal","600-500-400-300-200","N",0.9849])
x.add_row([5,"N","sigmoid","adam","Glorot Normal","500-400-300-200-100","Y",0.9783])
x.add_row([5,"N","tanh","adam","He Normal","480-360-240-120-60","N",0.9781])


print(x)
```

No of hidden layers	Batch_Normalization	Activation_function	optimizers	weig
2	Y	Relu	sgd	
2	Y	sigmoid	adadelta	
2	N	Relu	adam	
2	N	tanh	adam	
3	Y	Relu	sgd	
3	Y	Relu	adadelta	
3	N	sigmoid	adam	
3	N	tanh	adam	
5	Y	Relu	sgd	
5	Y	Relu	adadelta	
5	N	sigmoid	adam	
5	N	tanh	adam	

3 hidden layers with relu activation and adadelta optimizers and glorot no yield best accuracy rate of 0.9859