In [0]:

```
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
import warnings
warnings.filterwarnings("ignore")
```

Plotting code snippet below :

In [0]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import time

def plt_dynamic (x , vy , ty , cx , colors = ['b']):
    ax.plot (x , vy , 'b' , label = "Validation Loss")
    ax.plot (x , vy , 'r' , label = "Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

Loading the train and test data

In [3]:

```
#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 [==============================] - 3s 0us/step
```

Printing the training examples

In [4]:
```
print("Number of training examples :", X_train.shape[0], "and each imag
e is of shape (%d, %d)"%(X_train.shape[1], X_train.shape[2]))
print("Number of training examples :", X_test.shape[0], "and each image
 is of shape (%d, %d)"%(X_test.shape[1], X_test.shape[2]))
```

```
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)
```

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

In [6]:
```
print("Number of training examples :", X_train.shape[0], "and each imag
e is of shape (%d)"%(X_train.shape[1]))
print("Number of training examples :", X_test.shape[0], "and each image
 is of shape (%d)"%(X_test.shape[1]))
```

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

In [7]:
```
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 25
5
 247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  36  94 15
4
 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  8
2
  82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 25
3
 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 15
4
   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 24
1
 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 18
7
```

```
   0   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 25
3
 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 25
3
 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 19
5
  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   1
6
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0]
```

0 means white 255 means black Between 0 and 255 means grey

In [0]:
```
X_train = X_train/255
X_test = X_test/255
```

Observation:

Normalization being done here

In [9]:
```
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.        ]
```

In [10]:

```
print("Class label of first image :", y_train[0])


#converting class labels into one hot encoding
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.]
```

In [0]:

```
from keras.models import Sequential
```

```python
from keras.layers import Dense, Activation
```

In [0]:
```python
output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

In [0]:
```python
model = Sequential()



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

MLP + SGD OPTIMIZER + SIGMOID ACTIVATION

In [14]:
```python
model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics
=['accuracy'])



history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_
epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 2s 26us/step - loss: 1.3
017 - acc: 0.6960 - val_loss: 0.8110 - val_acc: 0.8389
Epoch 2/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.7
186 - acc: 0.8418 - val_loss: 0.6070 - val_acc: 0.8639
Epoch 3/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.5
884 - acc: 0.8593 - val_loss: 0.5251 - val_acc: 0.8747
Epoch 4/20
```

```
60000/60000 [==============================] - 1s 19us/step - loss: 0.5
262 - acc: 0.8687 - val_loss: 0.4795 - val_acc: 0.8822
Epoch 5/20
60000/60000 [==============================] - 1s 20us/step - loss: 0.4
883 - acc: 0.8756 - val_loss: 0.4499 - val_acc: 0.8868
Epoch 6/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.4
622 - acc: 0.8808 - val_loss: 0.4282 - val_acc: 0.8903
Epoch 7/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.4
429 - acc: 0.8837 - val_loss: 0.4124 - val_acc: 0.8924
Epoch 8/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.4
279 - acc: 0.8871 - val_loss: 0.3995 - val_acc: 0.8942
Epoch 9/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.4
158 - acc: 0.8895 - val_loss: 0.3893 - val_acc: 0.8970
Epoch 10/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.4
058 - acc: 0.8916 - val_loss: 0.3805 - val_acc: 0.8989
Epoch 11/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.3
973 - acc: 0.8933 - val_loss: 0.3732 - val_acc: 0.9005
Epoch 12/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.3
900 - acc: 0.8948 - val_loss: 0.3670 - val_acc: 0.9016
Epoch 13/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.3
836 - acc: 0.8968 - val_loss: 0.3616 - val_acc: 0.9024
Epoch 14/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.3
780 - acc: 0.8977 - val_loss: 0.3564 - val_acc: 0.9034
Epoch 15/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.3
729 - acc: 0.8986 - val_loss: 0.3518 - val_acc: 0.9044
Epoch 16/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.3
684 - acc: 0.8997 - val_loss: 0.3481 - val_acc: 0.9052
Epoch 17/20
```

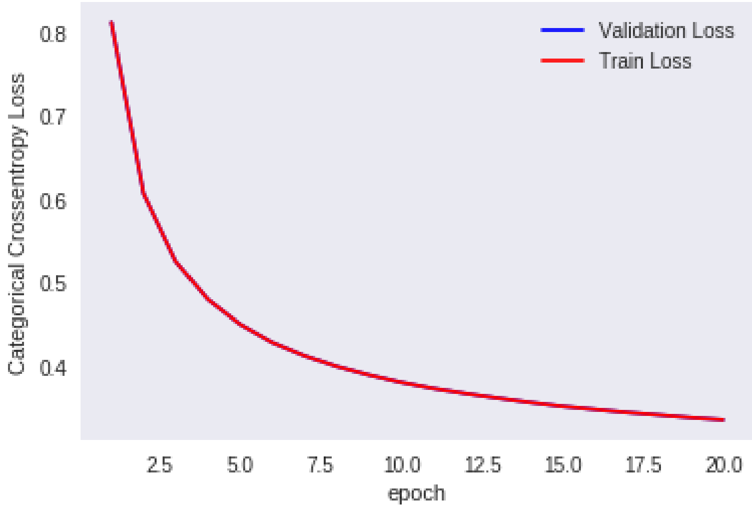
```
60000/60000 [==============================] - 1s 21us/step - loss: 0.3
643 - acc: 0.9004 - val_loss: 0.3445 - val_acc: 0.9066
Epoch 18/20
60000/60000 [==============================] - 1s 20us/step - loss: 0.3
606 - acc: 0.9011 - val_loss: 0.3414 - val_acc: 0.9074
Epoch 19/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.3
571 - acc: 0.9018 - val_loss: 0.3382 - val_acc: 0.9088
Epoch 20/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.3
540 - acc: 0.9023 - val_loss: 0.3355 - val_acc: 0.9089
```

In [15]:

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

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

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

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

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

vy = history.history['val_loss']
```

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

```
Test score: 0.3354729148030281
Test accuracy: 0.9089
```



In [16]:

```
model_sigmoid = Sequential()
model_sigmoid.add(Dense(512, activation='sigmoid', input_shape=(input_d
im,)))
model_sigmoid.add(Dense(128, activation='sigmoid'))
model_sigmoid.add(Dense(output_dim, activation='softmax'))

model_sigmoid.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_3 (Dense)              (None, 128)               65664
_________________________________________________________________
dense_4 (Dense)              (None, 10)                1290
=================================================================
```

```
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
```

In [17]:
```
model_sigmoid.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_sigmoid.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

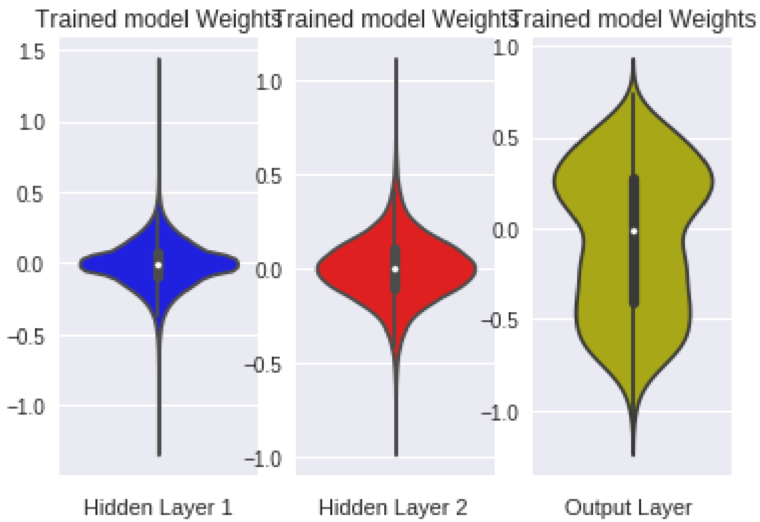
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 77us/step - loss: 2.2696 - acc: 0.2015 - val_loss: 2.2274 - val_acc: 0.3192
Epoch 2/20
60000/60000 [==============================] - 4s 74us/step - loss: 2.1849 - acc: 0.4129 - val_loss: 2.1323 - val_acc: 0.4897
Epoch 3/20
60000/60000 [==============================] - 4s 74us/step - loss: 2.0732 - acc: 0.5437 - val_loss: 1.9958 - val_acc: 0.5616
Epoch 4/20
60000/60000 [==============================] - 4s 73us/step - loss: 1.9136 - acc: 0.6075 - val_loss: 1.8044 - val_acc: 0.6549
Epoch 5/20
60000/60000 [==============================] - 4s 75us/step - loss: 1.7052 - acc: 0.6553 - val_loss: 1.5760 - val_acc: 0.7009
Epoch 6/20
60000/60000 [==============================] - 4s 74us/step - loss: 1.4768 - acc: 0.6995 - val_loss: 1.3489 - val_acc: 0.7233
Epoch 7/20
60000/60000 [==============================] - 4s 74us/step - loss: 1.2680 - acc: 0.7345 - val_loss: 1.1597 - val_acc: 0.7576
Epoch 8/20
60000/60000 [==============================] - 4s 73us/step - loss: 1.1004 - acc: 0.7637 - val_loss: 1.0120 - val_acc: 0.7852
Epoch 9/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.9716 - acc: 0.7850 - val_loss: 0.8998 - val_acc: 0.8035
```

```
Epoch 10/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.8
720 - acc: 0.8021 - val_loss: 0.8133 - val_acc: 0.8171
Epoch 11/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.7
937 - acc: 0.8145 - val_loss: 0.7431 - val_acc: 0.8244
Epoch 12/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.7
306 - acc: 0.8256 - val_loss: 0.6868 - val_acc: 0.8373
Epoch 13/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.6
793 - acc: 0.8341 - val_loss: 0.6406 - val_acc: 0.8432
Epoch 14/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.6
373 - acc: 0.8411 - val_loss: 0.6022 - val_acc: 0.8485
Epoch 15/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.6
022 - acc: 0.8476 - val_loss: 0.5700 - val_acc: 0.8553
Epoch 16/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.5
726 - acc: 0.8531 - val_loss: 0.5430 - val_acc: 0.8614
Epoch 17/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.5
475 - acc: 0.8580 - val_loss: 0.5198 - val_acc: 0.8646
Epoch 18/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.5
258 - acc: 0.8621 - val_loss: 0.4999 - val_acc: 0.8681
Epoch 19/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.5
069 - acc: 0.8665 - val_loss: 0.4821 - val_acc: 0.8723
Epoch 20/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.4
902 - acc: 0.8698 - val_loss: 0.4675 - val_acc: 0.8753
```

In [27]:

```
w_after = model_sigmoid.get_weights()

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

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

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

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



MLP + ADAM OPTIMIZER +SIGMOID

ACTIVATION

In [19]:
```
model_sigmoid = Sequential()
model_sigmoid.add(Dense(512, activation='sigmoid', input_shape=(input_d
im,)))
model_sigmoid.add(Dense(128, activation='sigmoid'))
model_sigmoid.add(Dense(output_dim, activation='softmax'))

model_sigmoid.summary()

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

history = model_sigmoid.fit(X_train, Y_train, batch_size=batch_size, ep
ochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_5 (Dense)              (None, 512)               401920    
_________________________________________________________________
dense_6 (Dense)              (None, 128)               65664     
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1290      
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.5
567 - acc: 0.8541 - val_loss: 0.2564 - val_acc: 0.9262
Epoch 2/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.2
213 - acc: 0.9348 - val_loss: 0.1892 - val_acc: 0.9432
Epoch 3/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.1
625 - acc: 0.9519 - val_loss: 0.1414 - val_acc: 0.9565
```

```
Epoch 4/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.1
246 - acc: 0.9633 - val_loss: 0.1230 - val_acc: 0.9608
Epoch 5/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0
994 - acc: 0.9709 - val_loss: 0.0976 - val_acc: 0.9705
Epoch 6/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
785 - acc: 0.9767 - val_loss: 0.0903 - val_acc: 0.9718
Epoch 7/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
640 - acc: 0.9806 - val_loss: 0.0847 - val_acc: 0.9726
Epoch 8/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0
512 - acc: 0.9848 - val_loss: 0.0774 - val_acc: 0.9751
Epoch 9/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0
412 - acc: 0.9881 - val_loss: 0.0702 - val_acc: 0.9780
Epoch 10/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0
332 - acc: 0.9906 - val_loss: 0.0667 - val_acc: 0.9793
Epoch 11/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
275 - acc: 0.9923 - val_loss: 0.0636 - val_acc: 0.9808
Epoch 12/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0
211 - acc: 0.9945 - val_loss: 0.0628 - val_acc: 0.9815
Epoch 13/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0
174 - acc: 0.9957 - val_loss: 0.0606 - val_acc: 0.9823
Epoch 14/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0
138 - acc: 0.9966 - val_loss: 0.0618 - val_acc: 0.9819
Epoch 15/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0
112 - acc: 0.9974 - val_loss: 0.0612 - val_acc: 0.9827
Epoch 16/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0
088 - acc: 0.9980 - val_loss: 0.0739 - val_acc: 0.9798
```

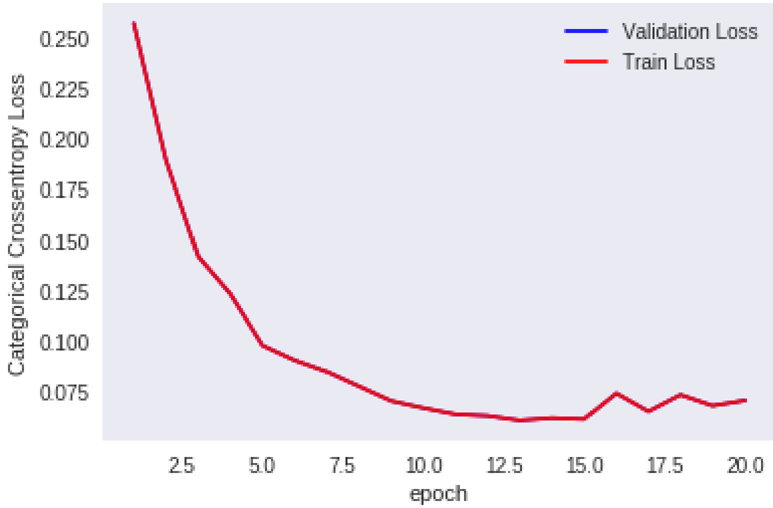
```
Epoch 17/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
074 - acc: 0.9984 - val_loss: 0.0650 - val_acc: 0.9832
Epoch 18/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0
055 - acc: 0.9990 - val_loss: 0.0732 - val_acc: 0.9813
Epoch 19/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
045 - acc: 0.9991 - val_loss: 0.0679 - val_acc: 0.9820
Epoch 20/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0
041 - acc: 0.9990 - val_loss: 0.0704 - val_acc: 0.9814
```

In [20]:

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

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

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

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter vali
dation_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']
```

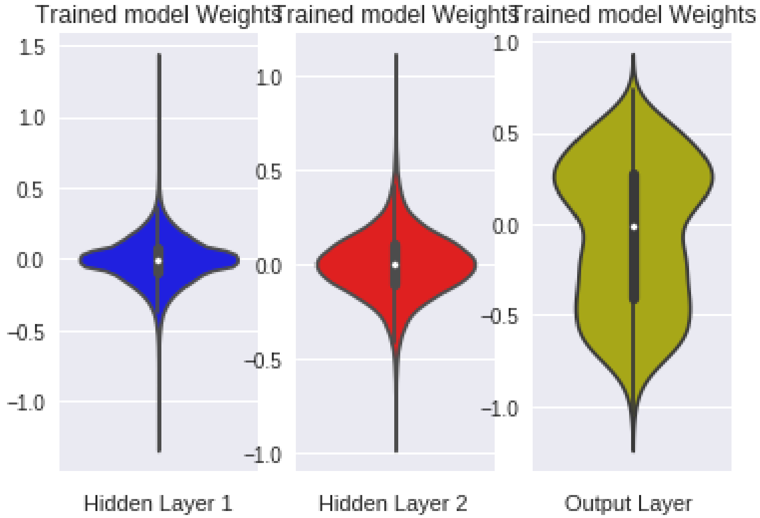
```
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
plt.show()
```

```
Test score: 0.07042204804238318
Test accuracy: 0.9814
```



In [28]:
```
w_after = model_sigmoid.get_weights()

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


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

plt.subplot(1, 3, 2)
plt.title("Trained model Weights")
```

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

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



MLP + RELU ACTIVATION + SGD OPTIMIZER

In [22]:
```
model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(Dense(128, activation='relu', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

model_relu.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_9 (Dense)              (None, 128)               65664
_________________________________________________________________
dense_10 (Dense)             (None, 10)                1290
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
```

In [23]:

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

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.7554 - acc: 0.7889 - val_loss: 0.3819 - val_acc: 0.8929
Epoch 2/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.3461 - acc: 0.9019 - val_loss: 0.2969 - val_acc: 0.9148
Epoch 3/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.2848 - acc: 0.9193 - val_loss: 0.2581 - val_acc: 0.9271
Epoch 4/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.2520 - acc: 0.9285 - val_loss: 0.2377 - val_acc: 0.9338
Epoch 5/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.2292 - acc: 0.9355 - val_loss: 0.2193 - val_acc: 0.9374
Epoch 6/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.2118 - acc: 0.9405 - val_loss: 0.2051 - val_acc: 0.9419
Epoch 7/20
```

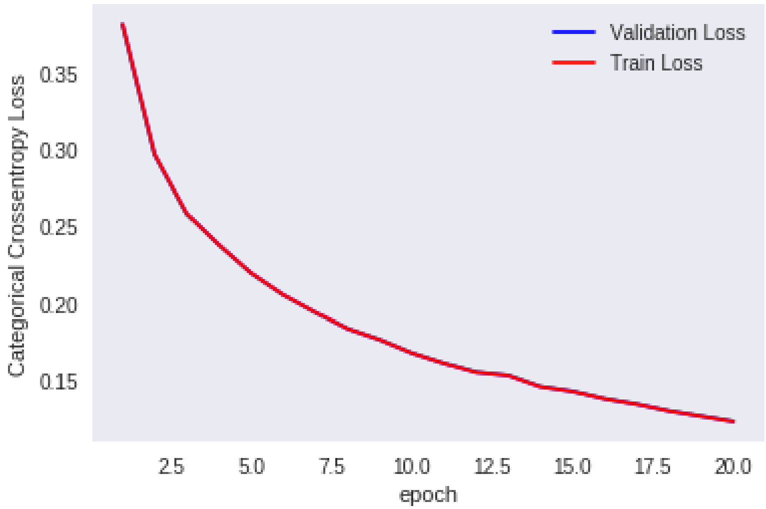
```
60000/60000 [==============================] - 4s 72us/step - loss: 0.1
972 - acc: 0.9449 - val_loss: 0.1937 - val_acc: 0.9462
Epoch 8/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.1
851 - acc: 0.9481 - val_loss: 0.1829 - val_acc: 0.9477
Epoch 9/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.1
746 - acc: 0.9508 - val_loss: 0.1757 - val_acc: 0.9496
Epoch 10/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.1
651 - acc: 0.9540 - val_loss: 0.1670 - val_acc: 0.9519
Epoch 11/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.1
570 - acc: 0.9563 - val_loss: 0.1603 - val_acc: 0.9537
Epoch 12/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.1
493 - acc: 0.9585 - val_loss: 0.1545 - val_acc: 0.9553
Epoch 13/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.1
424 - acc: 0.9603 - val_loss: 0.1524 - val_acc: 0.9555
Epoch 14/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.1
362 - acc: 0.9622 - val_loss: 0.1450 - val_acc: 0.9583
Epoch 15/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.1
305 - acc: 0.9636 - val_loss: 0.1420 - val_acc: 0.9590
Epoch 16/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.1
253 - acc: 0.9650 - val_loss: 0.1370 - val_acc: 0.9607
Epoch 17/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.1
203 - acc: 0.9666 - val_loss: 0.1335 - val_acc: 0.9614
Epoch 18/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.1
160 - acc: 0.9675 - val_loss: 0.1291 - val_acc: 0.9639
Epoch 19/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.1
117 - acc: 0.9687 - val_loss: 0.1257 - val_acc: 0.9636
Epoch 20/20
```

```
60000/60000 [==============================] - 4s 71us/step - loss: 0.1
077 - acc: 0.9699 - val_loss: 0.1222 - val_acc: 0.9652
```

In [24]:

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

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

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

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

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

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

```
Test score: 0.12223365796022118
Test accuracy: 0.9652
```



In [29]:

```
w_after = model_relu.get_weights()

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


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

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

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

```
plt.xlabel('Output Layer ')
plt.show()
```