Keras_Mnist_mayankgupta9968_gmail_com_12

November 6, 2019

0.1 Keras – MLPs on MNIST

[2]:
```
# Silent deprecation warnings
import warnings
warnings.filterwarnings('ignore')




# Silent tensor flow warning related to version 2 https://www.tensorflow.org/
↪guide/migrate (below two lines are not working)
# import tensorflow.compat.v1 as tf
# tf.disable_v2_behavior()

# if your 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

# Explicilty set log level to error i.e. supress info and warnings from TF
import os
os.environ["TF_CPP_MIN_LOG_LEVEL"]="2"
```

```
Using TensorFlow backend.

<IPython.core.display.HTML object>
```

[0]:
```
import matplotlib
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()
```

[0]:
```
# the data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

[5]:
```
# shape of the data
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(60000, 28, 28) (60000,)
(10000, 28, 28) (10000,)
```

[6]:
```
X_train[0]
```

[6]:
```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   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]], dtype=uint8)
```

[7]:
```
print("Number of training examples :", X_train.shape[0], "and each image 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)
```

[8]:
```
# type(X_train)
a = np.array([[1,2,3], [4,5,6], [7,8,9]])
a = a.reshape(9,1)
a
```

[8]:
```
array([[1],
       [2],
       [3],
       [4],
       [5],
       [6],
       [7],
       [8],
       [9]])
```

[0]:
```
# 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])
```

[10]:
```
# after converting the input images from 3d to 2d vectors

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

[11]:
```
# 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]
```

[0]:
```
# if we observe the above matrix each cell is having a value between 0-255
```

```python
# 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
```

[13]:
```python
# example data point after normlizing
X_train[0]
```

[13]:
```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 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.        ])
```

[14]:
```
# 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])
print("After converting the output into a vector : ",Y_train[1])
```

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

Softmax classifier

[0]:
```
# https://keras.io/getting-started/sequential-model-guide/

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

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

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

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

###

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

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

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

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

####

# https://keras.io/activations/

# Activations can either be used through an Activation layer, or through the␣
↪activation argument supported by all forward layers:

# from keras.layers import Activation, Dense

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

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

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


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

[0]:
```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

[17]:
```
# start building a model
model = Sequential()

# The model needs to know what input shape it should expect.
# For this reason, the first layer in a Sequential model
# (and only the first, because following layers can do automatic shape␣
 ↪inference)
# needs to receive information about its input shape.
# you can use input_shape and input_dim to pass the shape of input

# output_dim represent the number of nodes need in that layer
# here we have 10 nodes

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

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:66: The name tf.get_default_graph
is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:541: The name tf.placeholder is
deprecated. Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:4432: The name tf.random_uniform is
deprecated. Please use tf.random.uniform instead.
```

[18]:
```
# Before training a model, you need to configure the learning process, which is␣
↪done via the compile method

# It receives three arguments:
# An optimizer. This could be the string identifier of an existing optimizer ,␣
↪https://keras.io/optimizers/
# A loss function. This is the objective that the model will try to minimize.,␣
↪https://keras.io/losses/
# A list of metrics. For any classification problem you will want to set this␣
↪to metrics=['accuracy'].  https://keras.io/metrics/


# Note: when using the categorical_crossentropy loss, your targets should be in␣
↪categorical format
# (e.g. if you have 10 classes, the target for each sample should be a␣
↪10-dimensional vector that is all-zeros except
# for a 1 at the index corresponding to the class of the sample).

# that is why we converted out labels into vectors

model.compile(optimizer='sgd', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

# Keras models are trained on Numpy arrays of input data and labels.
# For training a model, you will typically use the  fit function

# fit(self, x=None, y=None, batch_size=None, epochs=1, verbose=1,␣
↪callbacks=None, validation_split=0.0,
# validation_data=None, shuffle=True, class_weight=None, sample_weight=None,␣
↪initial_epoch=0, steps_per_epoch=None,
# validation_steps=None)

# fit() function Trains the model for a fixed number of epochs (iterations on a␣
↪dataset).

# it returns A History object. Its History.history attribute is a record of␣
↪training loss values and
# metrics values at successive epochs, as well as validation loss values and␣
↪validation metrics values (if applicable).

# https://github.com/openai/baselines/issues/20

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

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/optimizers.py:793: The name tf.train.Optimizer is deprecated.
```

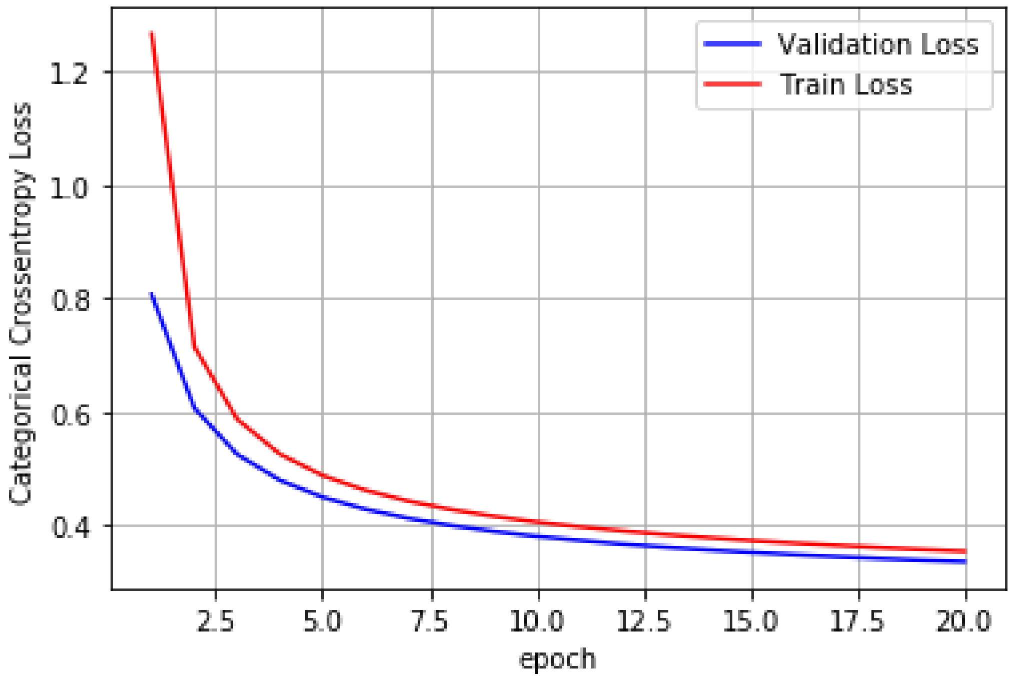
```
Please use tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:3576: The name tf.log is
deprecated. Please use tf.math.log instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow_core/python/ops/math_grad.py:1424: where (from
tensorflow.python.ops.array_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:1033: The name tf.assign_add is
deprecated. Please use tf.compat.v1.assign_add instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:1020: The name tf.assign is
deprecated. Please use tf.compat.v1.assign instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:3005: The name tf.Session is
deprecated. Please use tf.compat.v1.Session instead.

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:190: The name
tf.get_default_session is deprecated. Please use
tf.compat.v1.get_default_session instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:197: The name tf.ConfigProto is
deprecated. Please use tf.compat.v1.ConfigProto instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:207: The name tf.global_variables
is deprecated. Please use tf.compat.v1.global_variables instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:216: The name
tf.is_variable_initialized is deprecated. Please use
tf.compat.v1.is_variable_initialized instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:223: The name
tf.variables_initializer is deprecated. Please use
tf.compat.v1.variables_initializer instead.
```

```
60000/60000 [==============================] - 3s 51us/step - loss: 1.2654 -
acc: 0.7027 - val_loss: 0.8061 - val_acc: 0.8320
Epoch 2/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.7141 -
acc: 0.8396 - val_loss: 0.6060 - val_acc: 0.8604
Epoch 3/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.5865 -
acc: 0.8594 - val_loss: 0.5245 - val_acc: 0.8729
Epoch 4/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.5249 -
acc: 0.8692 - val_loss: 0.4791 - val_acc: 0.8796
Epoch 5/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.4872 -
acc: 0.8761 - val_loss: 0.4492 - val_acc: 0.8832
Epoch 6/20
60000/60000 [==============================] - 2s 31us/step - loss: 0.4615 -
acc: 0.8800 - val_loss: 0.4280 - val_acc: 0.8883
Epoch 7/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.4423 -
acc: 0.8836 - val_loss: 0.4120 - val_acc: 0.8907
Epoch 8/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.4274 -
acc: 0.8867 - val_loss: 0.3993 - val_acc: 0.8950
Epoch 9/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.4155 -
acc: 0.8892 - val_loss: 0.3889 - val_acc: 0.8966
Epoch 10/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.4054 -
acc: 0.8910 - val_loss: 0.3801 - val_acc: 0.8979
Epoch 11/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.3970 -
acc: 0.8928 - val_loss: 0.3731 - val_acc: 0.9000
Epoch 12/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.3898 -
acc: 0.8946 - val_loss: 0.3665 - val_acc: 0.9015
Epoch 13/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.3834 -
acc: 0.8955 - val_loss: 0.3611 - val_acc: 0.9035
Epoch 14/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.3778 -
acc: 0.8967 - val_loss: 0.3563 - val_acc: 0.9049
Epoch 15/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.3728 -
acc: 0.8983 - val_loss: 0.3519 - val_acc: 0.9061
Epoch 16/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.3683 -
acc: 0.8991 - val_loss: 0.3479 - val_acc: 0.9058
```

```
Epoch 17/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.3642 -
acc: 0.9002 - val_loss: 0.3444 - val_acc: 0.9071
Epoch 18/20
60000/60000 [==============================] - 2s 33us/step - loss: 0.3605 -
acc: 0.9010 - val_loss: 0.3411 - val_acc: 0.9075
Epoch 19/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.3570 -
acc: 0.9016 - val_loss: 0.3381 - val_acc: 0.9084
Epoch 20/20
60000/60000 [==============================] - 2s 32us/step - loss: 0.3539 -
acc: 0.9022 - val_loss: 0.3355 - val_acc: 0.9095
```

[19]:
```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

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

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

```
Test score: 0.33551198016405104
Test accuracy: 0.9095
```



MLP + Sigmoid activation + SGDOptimizer

[20]:
```
# Multilayer perceptron

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

model_sigmoid.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
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
_________________________________________________________________
```

[21]:
```
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))
```

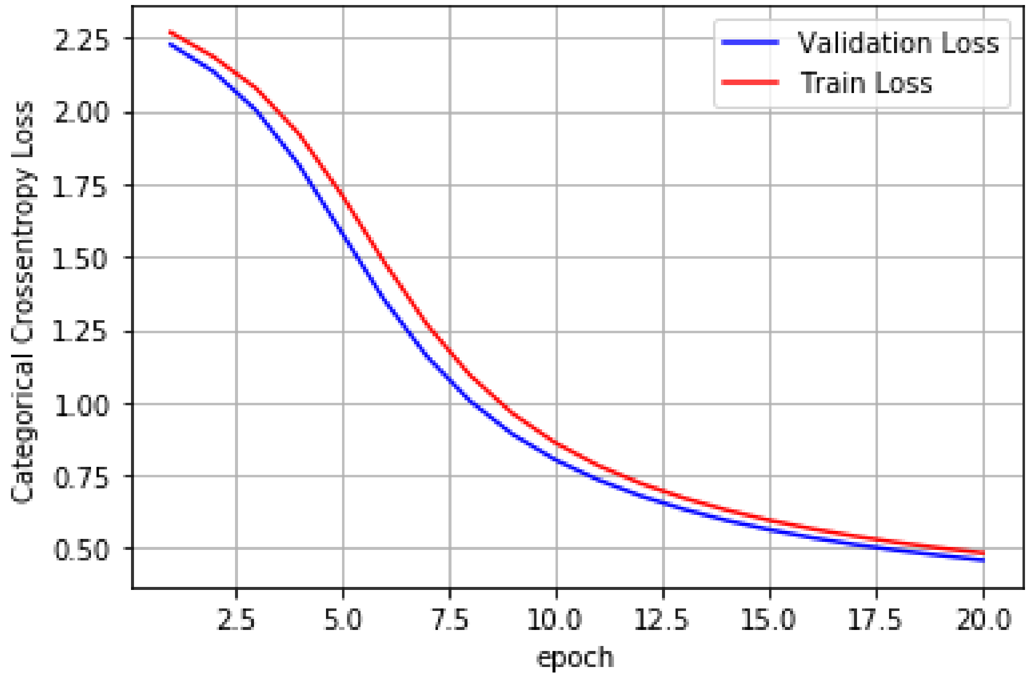
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 2s 39us/step - loss: 2.2693 -
acc: 0.2240 - val_loss: 2.2278 - val_acc: 0.3605
Epoch 2/20
60000/60000 [==============================] - 2s 36us/step - loss: 2.1858 -
acc: 0.4571 - val_loss: 2.1353 - val_acc: 0.4998
Epoch 3/20
60000/60000 [==============================] - 2s 36us/step - loss: 2.0775 -
acc: 0.5888 - val_loss: 2.0020 - val_acc: 0.6262
Epoch 4/20
60000/60000 [==============================] - 2s 35us/step - loss: 1.9217 -
acc: 0.6473 - val_loss: 1.8158 - val_acc: 0.6552
Epoch 5/20
60000/60000 [==============================] - 2s 37us/step - loss: 1.7140 -
acc: 0.6848 - val_loss: 1.5841 - val_acc: 0.7059
Epoch 6/20
60000/60000 [==============================] - 2s 35us/step - loss: 1.4812 -
acc: 0.7159 - val_loss: 1.3517 - val_acc: 0.7471
Epoch 7/20
60000/60000 [==============================] - 2s 35us/step - loss: 1.2667 -
acc: 0.7466 - val_loss: 1.1569 - val_acc: 0.7660
Epoch 8/20
60000/60000 [==============================] - 2s 36us/step - loss: 1.0937 -
acc: 0.7704 - val_loss: 1.0051 - val_acc: 0.7908
Epoch 9/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.9617 -
acc: 0.7871 - val_loss: 0.8902 - val_acc: 0.8042
Epoch 10/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.8611 -
acc: 0.8023 - val_loss: 0.8026 - val_acc: 0.8153
Epoch 11/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.7832 -
acc: 0.8143 - val_loss: 0.7338 - val_acc: 0.8300
Epoch 12/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.7215 -
acc: 0.8242 - val_loss: 0.6784 - val_acc: 0.8408
Epoch 13/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.6715 -
acc: 0.8334 - val_loss: 0.6329 - val_acc: 0.8474
Epoch 14/20
```

```
60000/60000 [==============================] - 2s 34us/step - loss: 0.6303 -
acc: 0.8417 - val_loss: 0.5951 - val_acc: 0.8531
Epoch 15/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.5957 -
acc: 0.8495 - val_loss: 0.5633 - val_acc: 0.8579
Epoch 16/20
60000/60000 [==============================] - 2s 34us/step - loss: 0.5662 -
acc: 0.8551 - val_loss: 0.5358 - val_acc: 0.8639
Epoch 17/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.5411 -
acc: 0.8602 - val_loss: 0.5124 - val_acc: 0.8687
Epoch 18/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.5193 -
acc: 0.8652 - val_loss: 0.4923 - val_acc: 0.8724
Epoch 19/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.5002 -
acc: 0.8693 - val_loss: 0.4749 - val_acc: 0.8763
Epoch 20/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.4836 -
acc: 0.8732 - val_loss: 0.4584 - val_acc: 0.8796
```

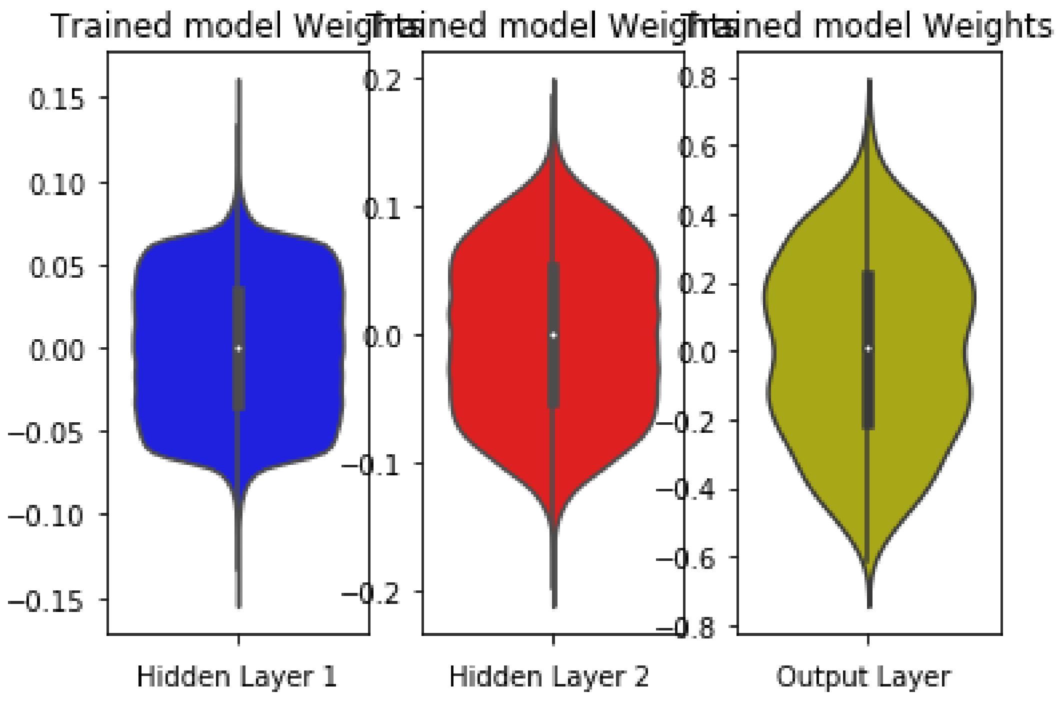
[22]:
```python
score = model_sigmoid.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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,␣
 ↪epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

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

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

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

```
Test score: 0.45842404165267947
Test accuracy: 0.8796
```



[23]:
```
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()
```



[24]:
```
# Multilayer perceptron (Tweak hidden layers)

model_sigmoid = Sequential()
model_sigmoid.add(Dense(256, activation='sigmoid', input_shape=(input_dim,)))
model_sigmoid.add(Dense(256, activation='sigmoid'))
model_sigmoid.add(Dense(output_dim, activation='softmax'))

model_sigmoid.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 256)               200960
_________________________________________________________________
dense_6 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_7 (Dense)              (None, 10)                2570
```

```
=================================================================
Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0
_________________________________________________________________
```

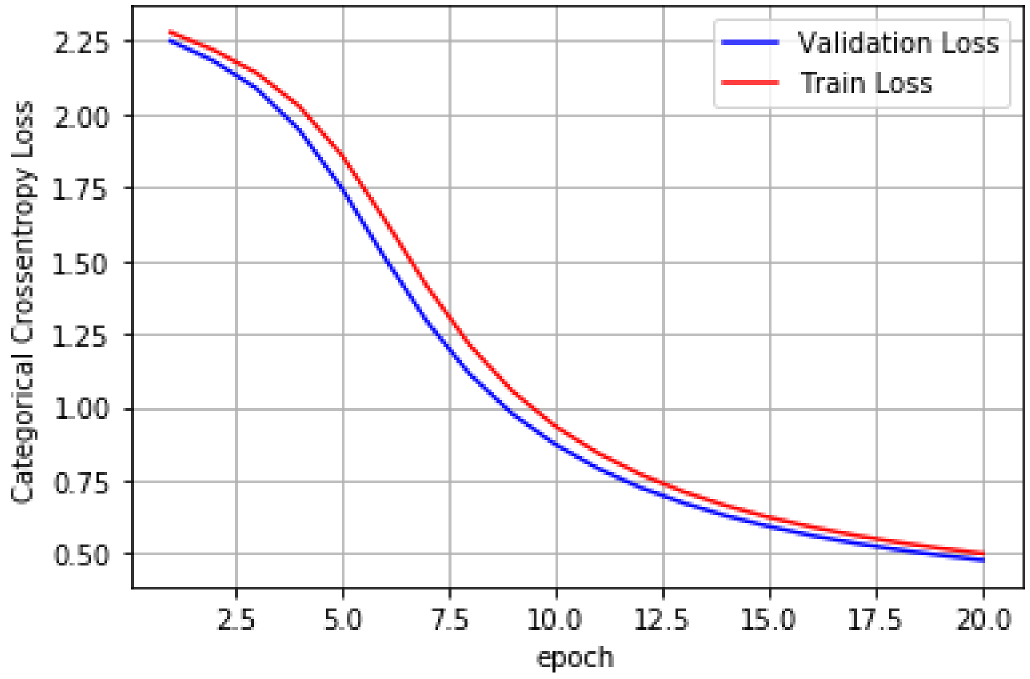
[25]:
```
model_sigmoid.compile(optimizer='sgd', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

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 [==============================] - 2s 39us/step - loss: 2.2777 -
acc: 0.1965 - val_loss: 2.2482 - val_acc: 0.1428
Epoch 2/20
60000/60000 [==============================] - 2s 35us/step - loss: 2.2178 -
acc: 0.3665 - val_loss: 2.1810 - val_acc: 0.3797
Epoch 3/20
60000/60000 [==============================] - 2s 37us/step - loss: 2.1407 -
acc: 0.5109 - val_loss: 2.0880 - val_acc: 0.5026
Epoch 4/20
60000/60000 [==============================] - 2s 35us/step - loss: 2.0277 -
acc: 0.5750 - val_loss: 1.9472 - val_acc: 0.5706
Epoch 5/20
60000/60000 [==============================] - 2s 36us/step - loss: 1.8605 -
acc: 0.6149 - val_loss: 1.7482 - val_acc: 0.6628
Epoch 6/20
60000/60000 [==============================] - 2s 37us/step - loss: 1.6419 -
acc: 0.6559 - val_loss: 1.5121 - val_acc: 0.7099
Epoch 7/20
60000/60000 [==============================] - 2s 37us/step - loss: 1.4114 -
acc: 0.6947 - val_loss: 1.2909 - val_acc: 0.7338
Epoch 8/20
60000/60000 [==============================] - 2s 36us/step - loss: 1.2110 -
acc: 0.7323 - val_loss: 1.1121 - val_acc: 0.7629
Epoch 9/20
60000/60000 [==============================] - 2s 35us/step - loss: 1.0542 -
acc: 0.7619 - val_loss: 0.9763 - val_acc: 0.7620
Epoch 10/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.9347 -
acc: 0.7841 - val_loss: 0.8725 - val_acc: 0.7903
Epoch 11/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.8426 -
acc: 0.8013 - val_loss: 0.7903 - val_acc: 0.8167
Epoch 12/20
```

```
60000/60000 [==============================] - 2s 38us/step - loss: 0.7700 -
acc: 0.8153 - val_loss: 0.7253 - val_acc: 0.8273
Epoch 13/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.7113 -
acc: 0.8269 - val_loss: 0.6728 - val_acc: 0.8339
Epoch 14/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.6635 -
acc: 0.8355 - val_loss: 0.6290 - val_acc: 0.8445
Epoch 15/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.6239 -
acc: 0.8430 - val_loss: 0.5927 - val_acc: 0.8520
Epoch 16/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.5909 -
acc: 0.8494 - val_loss: 0.5618 - val_acc: 0.8581
Epoch 17/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.5628 -
acc: 0.8547 - val_loss: 0.5360 - val_acc: 0.8629
Epoch 18/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.5390 -
acc: 0.8594 - val_loss: 0.5138 - val_acc: 0.8668
Epoch 19/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.5185 -
acc: 0.8641 - val_loss: 0.4946 - val_acc: 0.8692
Epoch 20/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.5006 -
acc: 0.8675 - val_loss: 0.4779 - val_acc: 0.8728
```

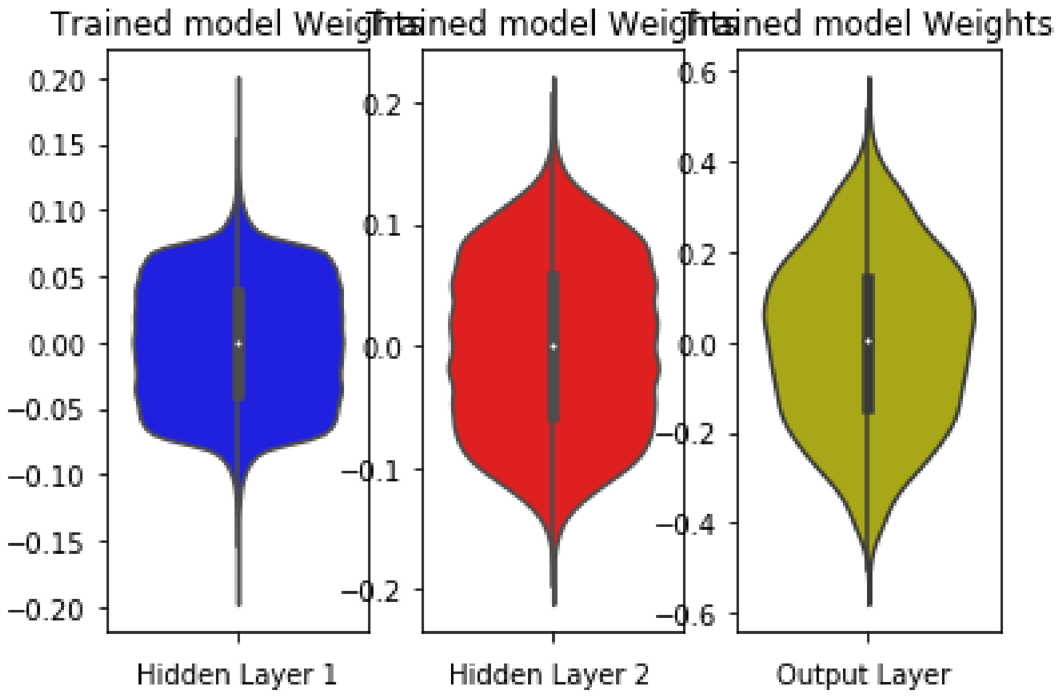
[26]:
```
score = model_sigmoid.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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,␣
 ↪epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter␣
 ↪validation_data
# val_loss : validation loss
# val_acc : validation accuracy
# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to␣
 ↪number of epochs

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

```
Test score: 0.4778948999404907
Test accuracy: 0.8728
```



[0]:
```
# b = np.array([[1,2,3], [1,2,3]])
# b.reshape(1,-1)
# b
```

[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 + Sigmoid activation + ADAM

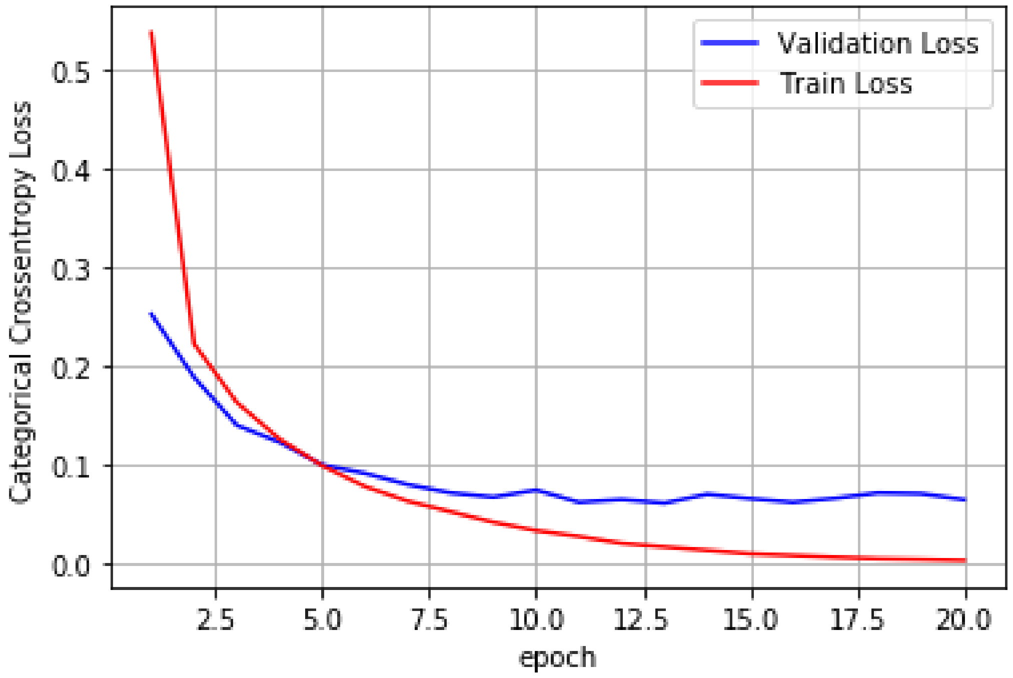
[29]:
```
model_sigmoid = Sequential()
model_sigmoid.add(Dense(512, activation='sigmoid', input_shape=(input_dim,)))
model_sigmoid.add(Dense(128, activation='sigmoid'))
model_sigmoid.add(Dense(output_dim, activation='softmax'))

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

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

```
Model: "sequential_4"
_________________________________________________________________
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
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.5367 -
acc: 0.8587 - val_loss: 0.2525 - val_acc: 0.9246
Epoch 2/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.2225 -
acc: 0.9347 - val_loss: 0.1887 - val_acc: 0.9438
Epoch 3/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.1632 -
acc: 0.9518 - val_loss: 0.1402 - val_acc: 0.9585
Epoch 4/20
60000/60000 [==============================] - 2s 42us/step - loss: 0.1263 -
acc: 0.9627 - val_loss: 0.1230 - val_acc: 0.9627
Epoch 5/20
60000/60000 [==============================] - 2s 42us/step - loss: 0.0993 -
acc: 0.9704 - val_loss: 0.1000 - val_acc: 0.9703
Epoch 6/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0782 -
acc: 0.9771 - val_loss: 0.0916 - val_acc: 0.9719
Epoch 7/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0634 -
acc: 0.9813 - val_loss: 0.0802 - val_acc: 0.9746
Epoch 8/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0528 -
acc: 0.9843 - val_loss: 0.0718 - val_acc: 0.9774
Epoch 9/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0423 -
```

```
acc: 0.9876 - val_loss: 0.0675 - val_acc: 0.9792
Epoch 10/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0339 -
acc: 0.9907 - val_loss: 0.0747 - val_acc: 0.9771
Epoch 11/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0279 -
acc: 0.9923 - val_loss: 0.0624 - val_acc: 0.9801
Epoch 12/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0210 -
acc: 0.9948 - val_loss: 0.0651 - val_acc: 0.9792
Epoch 13/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0174 -
acc: 0.9957 - val_loss: 0.0615 - val_acc: 0.9805
Epoch 14/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0139 -
acc: 0.9968 - val_loss: 0.0705 - val_acc: 0.9796
Epoch 15/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0105 -
acc: 0.9977 - val_loss: 0.0661 - val_acc: 0.9804
Epoch 16/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0087 -
acc: 0.9983 - val_loss: 0.0623 - val_acc: 0.9821
Epoch 17/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0069 -
acc: 0.9988 - val_loss: 0.0664 - val_acc: 0.9804
Epoch 18/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0054 -
acc: 0.9988 - val_loss: 0.0718 - val_acc: 0.9791
Epoch 19/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0048 -
acc: 0.9990 - val_loss: 0.0710 - val_acc: 0.9794
Epoch 20/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0038 -
acc: 0.9992 - val_loss: 0.0650 - val_acc: 0.9826
```

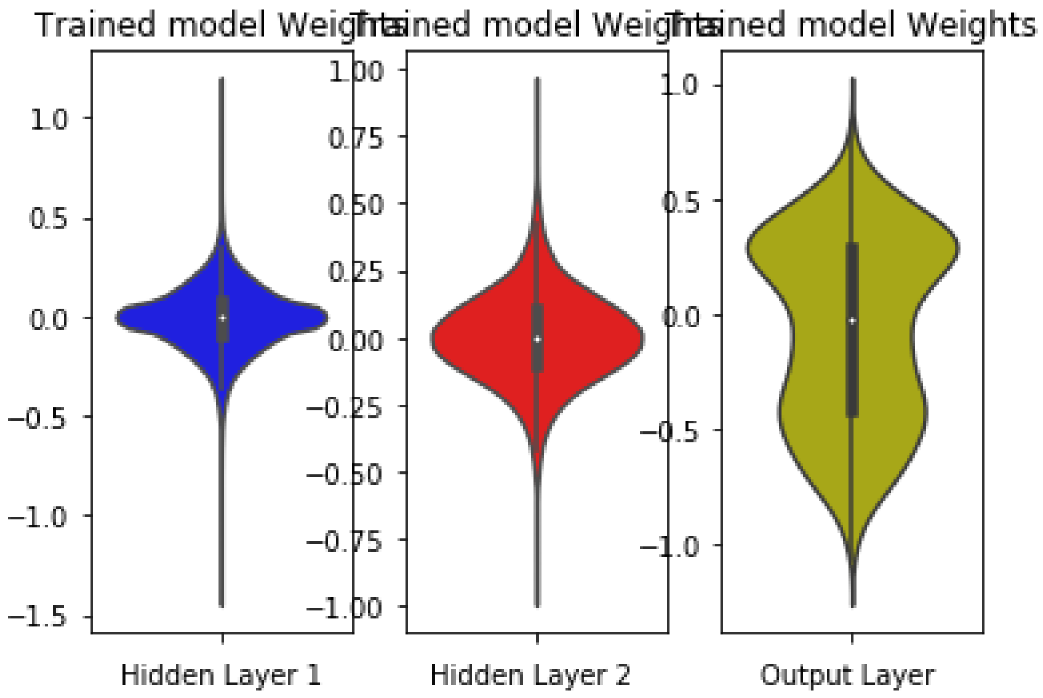
[30]:
```python
score = model_sigmoid.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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,␣
 ↪epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

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

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

```
Test score: 0.06496948067060403
Test accuracy: 0.9826
```



[31]:
```
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()
```



[32]:
```
# Custom with 3 layers
model_sigmoid = Sequential()
model_sigmoid.add(Dense(512, activation='sigmoid', input_shape=(input_dim,)))
model_sigmoid.add(Dense(256, activation='sigmoid'))
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,␣
 ↪epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

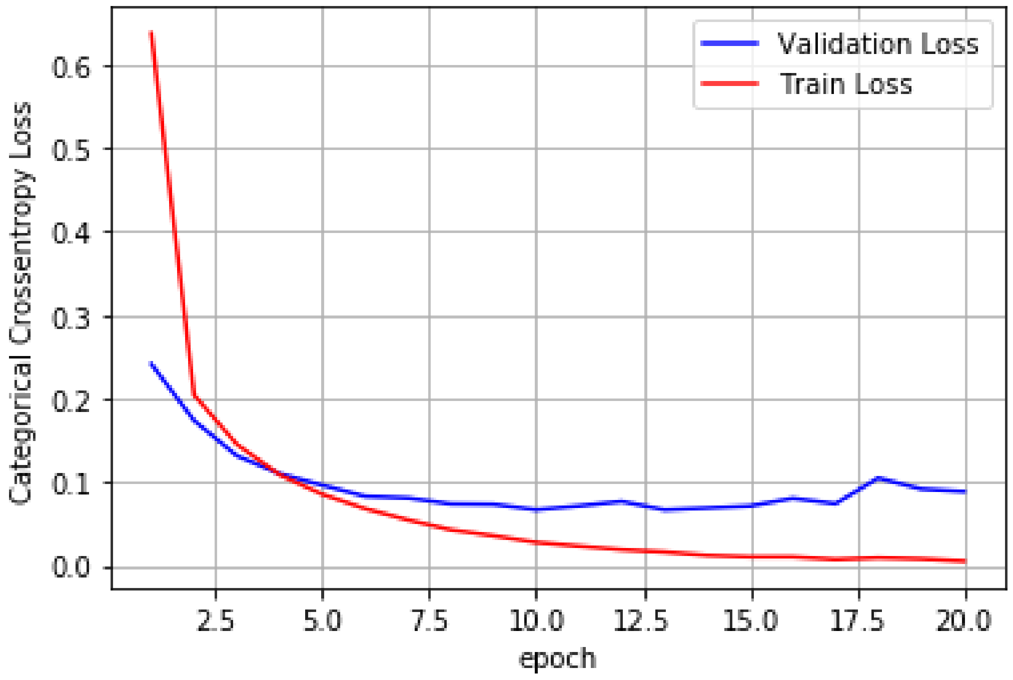
```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_11 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_12 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_13 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_14 (Dense)             (None, 10)                1290
=================================================================
Total params: 567,434
Trainable params: 567,434
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 52us/step - loss: 0.6370 -
acc: 0.8180 - val_loss: 0.2412 - val_acc: 0.9276
Epoch 2/20
60000/60000 [==============================] - 3s 47us/step - loss: 0.2041 -
acc: 0.9399 - val_loss: 0.1734 - val_acc: 0.9467
Epoch 3/20
60000/60000 [==============================] - 3s 45us/step - loss: 0.1450 -
acc: 0.9564 - val_loss: 0.1312 - val_acc: 0.9589
Epoch 4/20
60000/60000 [==============================] - 3s 48us/step - loss: 0.1085 -
acc: 0.9677 - val_loss: 0.1097 - val_acc: 0.9668
Epoch 5/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.0852 -
acc: 0.9746 - val_loss: 0.0962 - val_acc: 0.9717
Epoch 6/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0680 -
acc: 0.9794 - val_loss: 0.0828 - val_acc: 0.9760
Epoch 7/20
60000/60000 [==============================] - 3s 45us/step - loss: 0.0545 -
```

```
acc: 0.9834 - val_loss: 0.0808 - val_acc: 0.9745
Epoch 8/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0427 -
acc: 0.9875 - val_loss: 0.0737 - val_acc: 0.9776
Epoch 9/20
60000/60000 [==============================] - 3s 45us/step - loss: 0.0355 -
acc: 0.9893 - val_loss: 0.0732 - val_acc: 0.9789
Epoch 10/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.0276 -
acc: 0.9918 - val_loss: 0.0667 - val_acc: 0.9807
Epoch 11/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0231 -
acc: 0.9931 - val_loss: 0.0712 - val_acc: 0.9796
Epoch 12/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0188 -
acc: 0.9942 - val_loss: 0.0767 - val_acc: 0.9783
Epoch 13/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0160 -
acc: 0.9954 - val_loss: 0.0666 - val_acc: 0.9811
Epoch 14/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0119 -
acc: 0.9967 - val_loss: 0.0688 - val_acc: 0.9818
Epoch 15/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0102 -
acc: 0.9971 - val_loss: 0.0711 - val_acc: 0.9810
Epoch 16/20
60000/60000 [==============================] - 3s 45us/step - loss: 0.0102 -
acc: 0.9970 - val_loss: 0.0804 - val_acc: 0.9805
Epoch 17/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.0072 -
acc: 0.9980 - val_loss: 0.0739 - val_acc: 0.9816
Epoch 18/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0091 -
acc: 0.9971 - val_loss: 0.1048 - val_acc: 0.9758
Epoch 19/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0079 -
acc: 0.9974 - val_loss: 0.0916 - val_acc: 0.9799
Epoch 20/20
60000/60000 [==============================] - 3s 46us/step - loss: 0.0051 -
acc: 0.9984 - val_loss: 0.0886 - val_acc: 0.9808
```

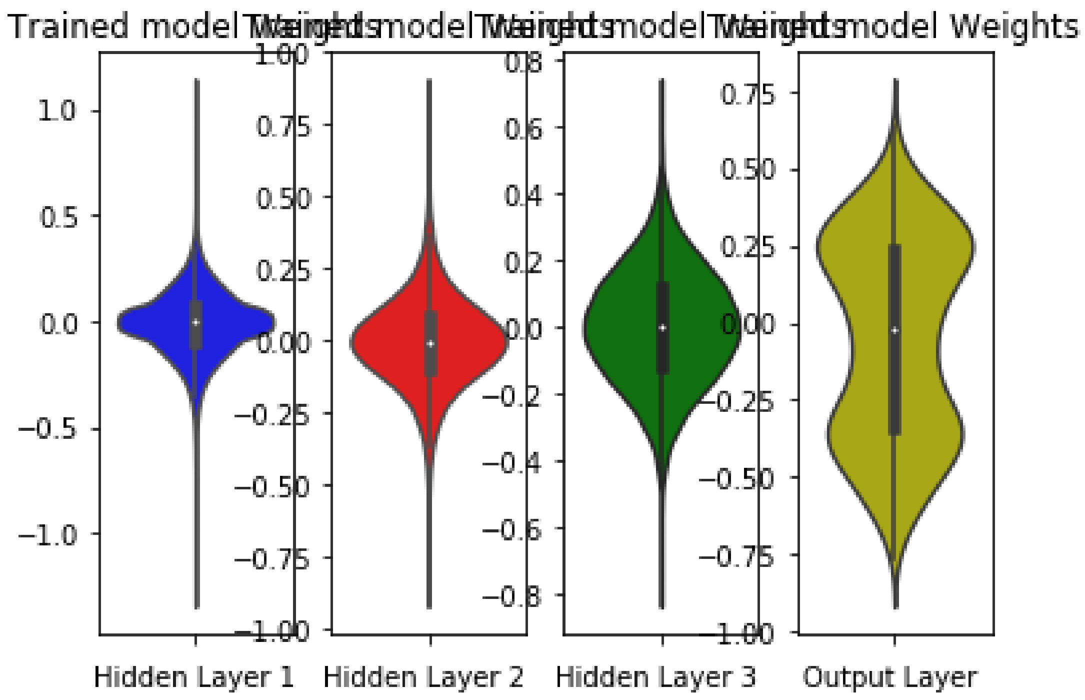
[33]:
```
score = model_sigmoid.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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,␣
 ↪epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

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

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

```
Test score: 0.08855799371209142
Test accuracy: 0.9808
```



[34]:
```
w_after = model_sigmoid.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()
```



MLP + ReLU +SGD

[35]:
```
# Multilayer perceptron

# https://arxiv.org/pdf/1707.09725.pdf#page=95
# for relu layers
# If we sample weights from a normal distribution N(0,) we satisfy this␣
 ↪condition with =(2/(ni).
# h1 =>  =(2/(fan_in) = 0.062  => N(0,) = N(0,0.062)
# h2 =>  =(2/(fan_in) = 0.125  => N(0,) = N(0,0.125)
# out =>  =(2/(fan_in+1) = 0.120  => N(0,) = N(0,0.120)

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=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

model_relu.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:4409: The name tf.random_normal is
deprecated. Please use tf.random.normal instead.

Model: "sequential_6"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_15 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_16 (Dense)             (None, 128)               65664
_________________________________________________________________
dense_17 (Dense)             (None, 10)                1290
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
```

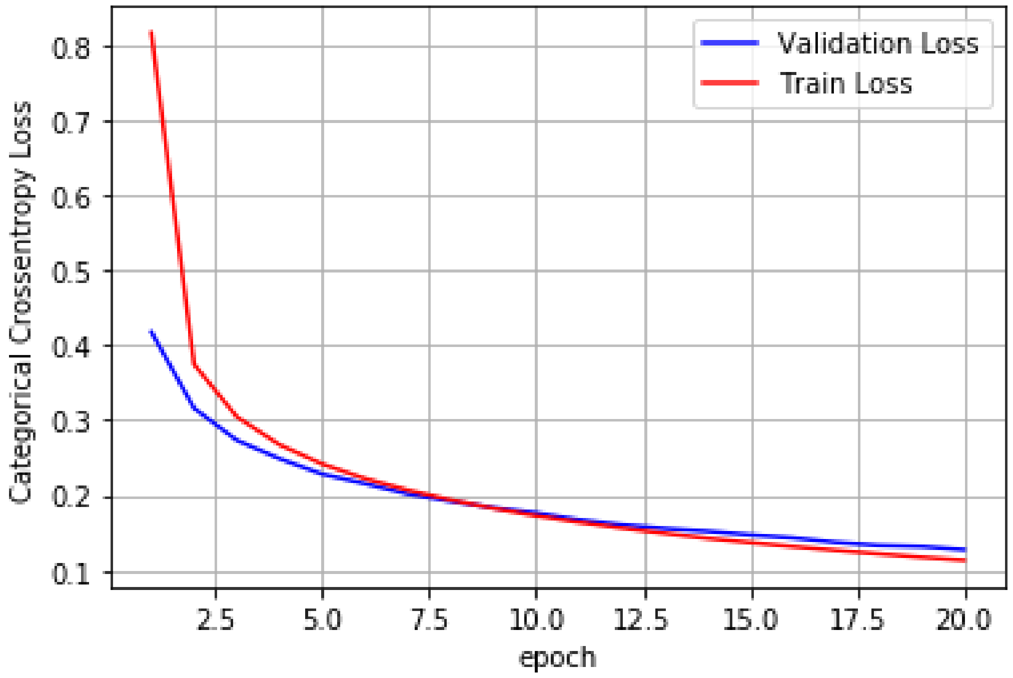
[36]:
```
model_relu.compile(optimizer='sgd', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

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 [==============================] - 3s 44us/step - loss: 0.8161 -
acc: 0.7685 - val_loss: 0.4177 - val_acc: 0.8881
Epoch 2/20
60000/60000 [==============================] - 2s 38us/step - loss: 0.3744 -
acc: 0.8960 - val_loss: 0.3159 - val_acc: 0.9139
Epoch 3/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.3045 -
acc: 0.9143 - val_loss: 0.2732 - val_acc: 0.9242
Epoch 4/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.2672 -
acc: 0.9246 - val_loss: 0.2486 - val_acc: 0.9307
Epoch 5/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.2415 -
acc: 0.9317 - val_loss: 0.2280 - val_acc: 0.9354
Epoch 6/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.2221 -
acc: 0.9370 - val_loss: 0.2154 - val_acc: 0.9372
Epoch 7/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.2067 -
acc: 0.9422 - val_loss: 0.2016 - val_acc: 0.9414
Epoch 8/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.1936 -
acc: 0.9457 - val_loss: 0.1919 - val_acc: 0.9442
Epoch 9/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.1824 -
```

```
acc: 0.9492 - val_loss: 0.1833 - val_acc: 0.9464
Epoch 10/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.1727 -
acc: 0.9517 - val_loss: 0.1764 - val_acc: 0.9487
Epoch 11/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.1641 -
acc: 0.9539 - val_loss: 0.1674 - val_acc: 0.9517
Epoch 12/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.1564 -
acc: 0.9566 - val_loss: 0.1608 - val_acc: 0.9539
Epoch 13/20
60000/60000 [==============================] - 2s 35us/step - loss: 0.1493 -
acc: 0.9587 - val_loss: 0.1558 - val_acc: 0.9540
Epoch 14/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.1427 -
acc: 0.9606 - val_loss: 0.1521 - val_acc: 0.9559
Epoch 15/20
60000/60000 [==============================] - 2s 38us/step - loss: 0.1370 -
acc: 0.9617 - val_loss: 0.1473 - val_acc: 0.9579
Epoch 16/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.1317 -
acc: 0.9638 - val_loss: 0.1433 - val_acc: 0.9586
Epoch 17/20
60000/60000 [==============================] - 2s 36us/step - loss: 0.1267 -
acc: 0.9652 - val_loss: 0.1374 - val_acc: 0.9599
Epoch 18/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.1220 -
acc: 0.9662 - val_loss: 0.1333 - val_acc: 0.9618
Epoch 19/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.1174 -
acc: 0.9677 - val_loss: 0.1319 - val_acc: 0.9612
Epoch 20/20
60000/60000 [==============================] - 2s 37us/step - loss: 0.1134 -
acc: 0.9688 - val_loss: 0.1275 - val_acc: 0.9630
```

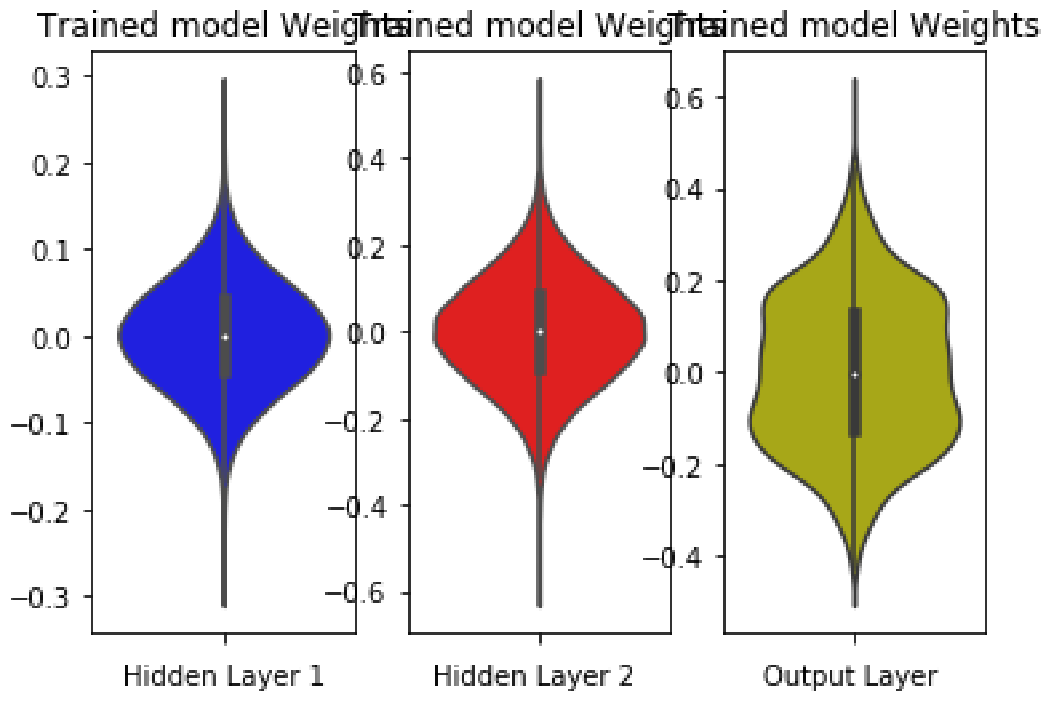
[37]:
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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,␣
 ↪epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

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

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

```
Test score: 0.12753310488164424
Test accuracy: 0.963
```



[38]:
```
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()
```



[39]:
```
# Multilayer perceptron (Custom with 5 layers)

# https://arxiv.org/pdf/1707.09725.pdf#page=95
# for relu layers
# If we sample weights from a normal distribution N(0,) we satisfy this␣
↪condition with =(2/(ni).
# h1 =>  =(2/(fan_in) = 0.062  => N(0,) = N(0,0.062)
# h2 =>  =(2/(fan_in) = 0.125  => N(0,) = N(0,0.125)
# out =>  =(2/(fan_in+1) = 0.120  => N(0,) = N(0,0.120)

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(256, activation='relu',␣
↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(128, activation='relu',␣
↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(64, activation='relu',␣
↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(32, activation='relu',␣
↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

model_relu.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_18 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_19 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_20 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_21 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_22 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_23 (Dense)             (None, 10)                330
=================================================================
Total params: 576,810
Trainable params: 576,810
Non-trainable params: 0
_________________________________________________________________
```

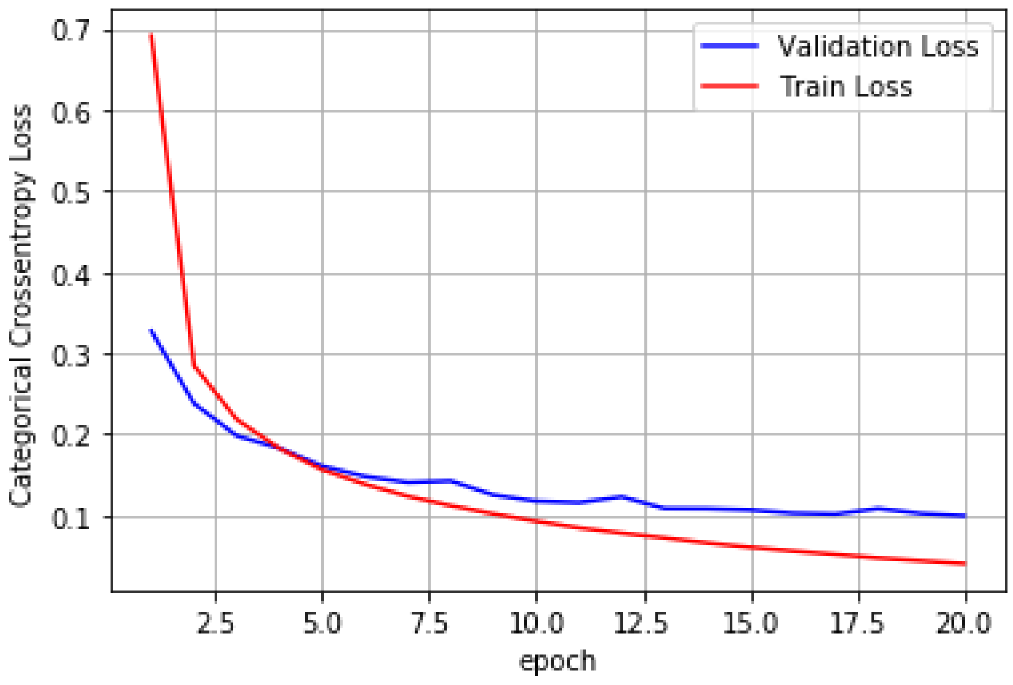
[40]:
```
model_relu.compile(optimizer='sgd', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

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 [==============================] - 3s 49us/step - loss: 0.6912 -
acc: 0.7924 - val_loss: 0.3273 - val_acc: 0.9017
Epoch 2/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.2847 -
acc: 0.9167 - val_loss: 0.2379 - val_acc: 0.9302
Epoch 3/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.2182 -
acc: 0.9358 - val_loss: 0.1981 - val_acc: 0.9429
Epoch 4/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.1823 -
acc: 0.9460 - val_loss: 0.1828 - val_acc: 0.9462
Epoch 5/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.1562 -
acc: 0.9538 - val_loss: 0.1607 - val_acc: 0.9524
Epoch 6/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.1384 -
acc: 0.9592 - val_loss: 0.1479 - val_acc: 0.9561
Epoch 7/20
60000/60000 [==============================] - 2s 42us/step - loss: 0.1236 -
acc: 0.9636 - val_loss: 0.1407 - val_acc: 0.9566
Epoch 8/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.1120 -
acc: 0.9671 - val_loss: 0.1426 - val_acc: 0.9559
Epoch 9/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.1019 -
acc: 0.9705 - val_loss: 0.1253 - val_acc: 0.9624
Epoch 10/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0928 -
acc: 0.9732 - val_loss: 0.1176 - val_acc: 0.9638
Epoch 11/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0847 -
acc: 0.9751 - val_loss: 0.1158 - val_acc: 0.9631
Epoch 12/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0781 -
acc: 0.9771 - val_loss: 0.1230 - val_acc: 0.9624
Epoch 13/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0722 -
acc: 0.9787 - val_loss: 0.1085 - val_acc: 0.9664
Epoch 14/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0660 -
acc: 0.9813 - val_loss: 0.1083 - val_acc: 0.9667
Epoch 15/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0606 -
acc: 0.9824 - val_loss: 0.1068 - val_acc: 0.9672
Epoch 16/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0561 -
```

```
acc: 0.9844 - val_loss: 0.1028 - val_acc: 0.9681
Epoch 17/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0518 -
acc: 0.9853 - val_loss: 0.1016 - val_acc: 0.9690
Epoch 18/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0474 -
acc: 0.9867 - val_loss: 0.1084 - val_acc: 0.9674
Epoch 19/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0440 -
acc: 0.9881 - val_loss: 0.1024 - val_acc: 0.9689
Epoch 20/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0405 -
acc: 0.9892 - val_loss: 0.0995 - val_acc: 0.9702
```

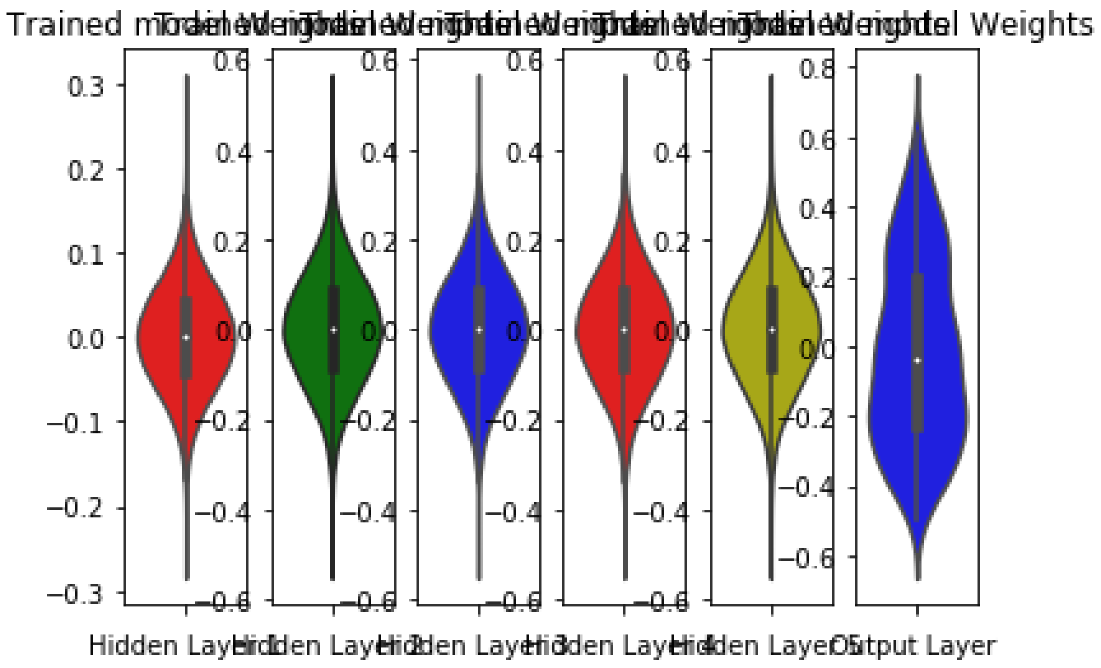
[41]:
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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,␣
 ↪epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

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

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

```
Test score: 0.09954953634198754
Test accuracy: 0.9702
```



[42]:
```
w_after = model_relu.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='r')
plt.xlabel('Hidden Layer 1')

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

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='b')
```

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

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='y')
plt.xlabel('Hidden Layer 5 ')

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



MLP + ReLU + ADAM

[43]:
```
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=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))
```

```
print(model_relu.summary())

model_relu.compile(optimizer='adam', 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))
```

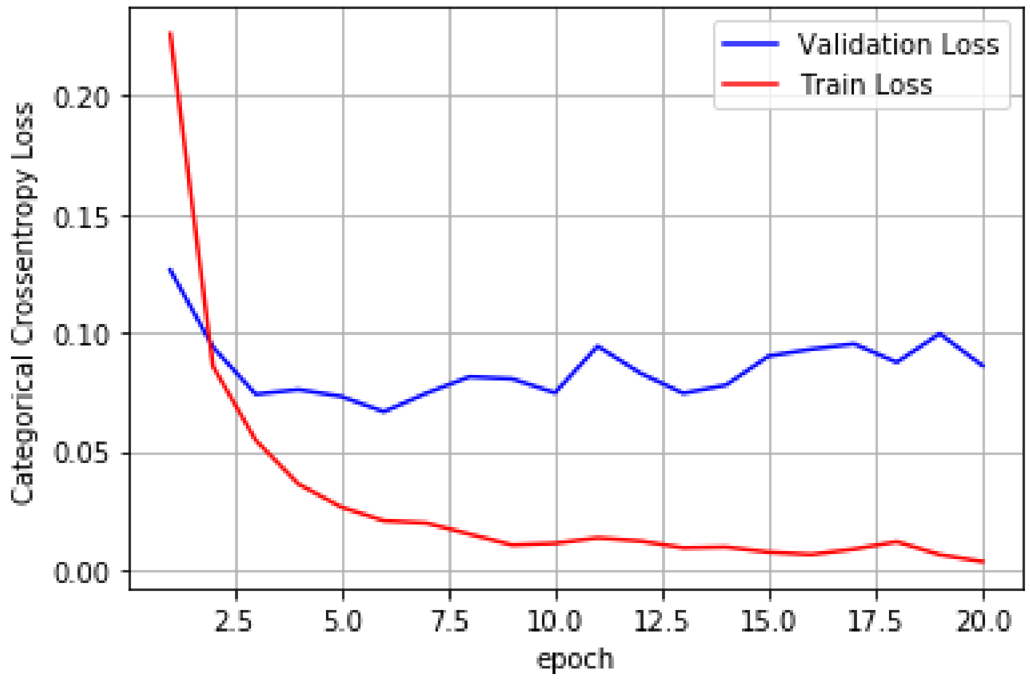
```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_24 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_25 (Dense)             (None, 128)               65664
_________________________________________________________________
dense_26 (Dense)             (None, 10)                1290
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 53us/step - loss: 0.2260 -
acc: 0.9323 - val_loss: 0.1264 - val_acc: 0.9611
Epoch 2/20
60000/60000 [==============================] - 2s 42us/step - loss: 0.0858 -
acc: 0.9737 - val_loss: 0.0940 - val_acc: 0.9693
Epoch 3/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0546 -
acc: 0.9829 - val_loss: 0.0740 - val_acc: 0.9766
Epoch 4/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0362 -
acc: 0.9888 - val_loss: 0.0760 - val_acc: 0.9760
Epoch 5/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0265 -
acc: 0.9918 - val_loss: 0.0731 - val_acc: 0.9780
Epoch 6/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0207 -
acc: 0.9938 - val_loss: 0.0667 - val_acc: 0.9804
Epoch 7/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0197 -
acc: 0.9936 - val_loss: 0.0745 - val_acc: 0.9786
Epoch 8/20
```

```
60000/60000 [==============================] - 3s 42us/step - loss: 0.0151 -
acc: 0.9951 - val_loss: 0.0814 - val_acc: 0.9777
Epoch 9/20
60000/60000 [==============================] - 2s 42us/step - loss: 0.0105 -
acc: 0.9966 - val_loss: 0.0806 - val_acc: 0.9790
Epoch 10/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0112 -
acc: 0.9964 - val_loss: 0.0746 - val_acc: 0.9813
Epoch 11/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0135 -
acc: 0.9956 - val_loss: 0.0943 - val_acc: 0.9787
Epoch 12/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0121 -
acc: 0.9960 - val_loss: 0.0830 - val_acc: 0.9779
Epoch 13/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0092 -
acc: 0.9970 - val_loss: 0.0744 - val_acc: 0.9819
Epoch 14/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0096 -
acc: 0.9968 - val_loss: 0.0779 - val_acc: 0.9816
Epoch 15/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0074 -
acc: 0.9974 - val_loss: 0.0902 - val_acc: 0.9799
Epoch 16/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0064 -
acc: 0.9978 - val_loss: 0.0931 - val_acc: 0.9815
Epoch 17/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0087 -
acc: 0.9970 - val_loss: 0.0953 - val_acc: 0.9808
Epoch 18/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0118 -
acc: 0.9963 - val_loss: 0.0875 - val_acc: 0.9802
Epoch 19/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0064 -
acc: 0.9981 - val_loss: 0.0996 - val_acc: 0.9815
Epoch 20/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0035 -
acc: 0.9990 - val_loss: 0.0861 - val_acc: 0.9838
```

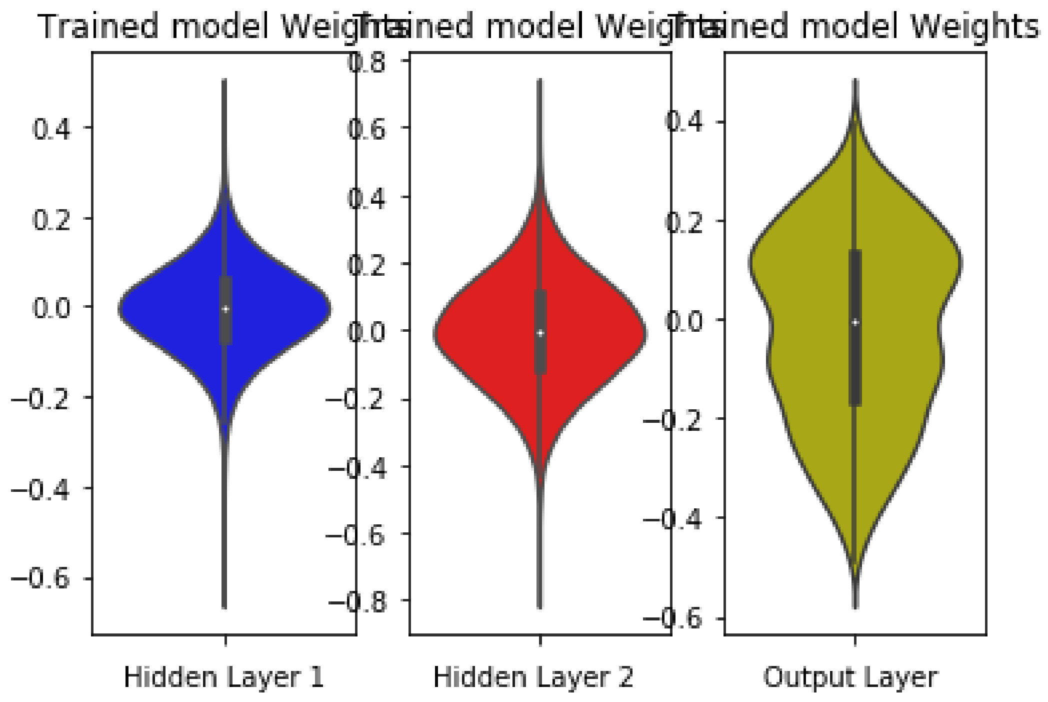
[44]:
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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,␣
 ↪epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

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


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

```
Test score: 0.08614931825476665
Test accuracy: 0.9838
```



[45]:
```
w_after = model_relu.get_weights()

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


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

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

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



MLP + Batch-Norm on hidden Layers + AdamOptimizer </2>

[46]:
```
# Multilayer perceptron

# https://intoli.com/blog/neural-network-initialization/
# If we sample weights from a normal distribution N(0,) we satisfy this␣
↪condition with =(2/(ni+ni+1).
# h1 =>  =(2/(ni+ni+1) = 0.039  => N(0,) = N(0,0.039)
# h2 =>  =(2/(ni+ni+1) = 0.055  => N(0,) = N(0,0.055)
# h1 =>  =(2/(ni+ni+1) = 0.120  => N(0,) = N(0,0.120)

from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_batch.add(Dense(512, activation='sigmoid', input_shape=(input_dim,),␣
↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(128, activation='sigmoid',␣
↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )
model_batch.add(BatchNormalization())

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


model_batch.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:148: The name
tf.placeholder_with_default is deprecated. Please use
tf.compat.v1.placeholder_with_default instead.

Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_27 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048
_________________________________________________________________
dense_28 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_2 (Batch (None, 128)               512
_________________________________________________________________
dense_29 (Dense)             (None, 10)                1290
=================================================================
Total params: 471,434
```

```
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

[47]:
```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

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

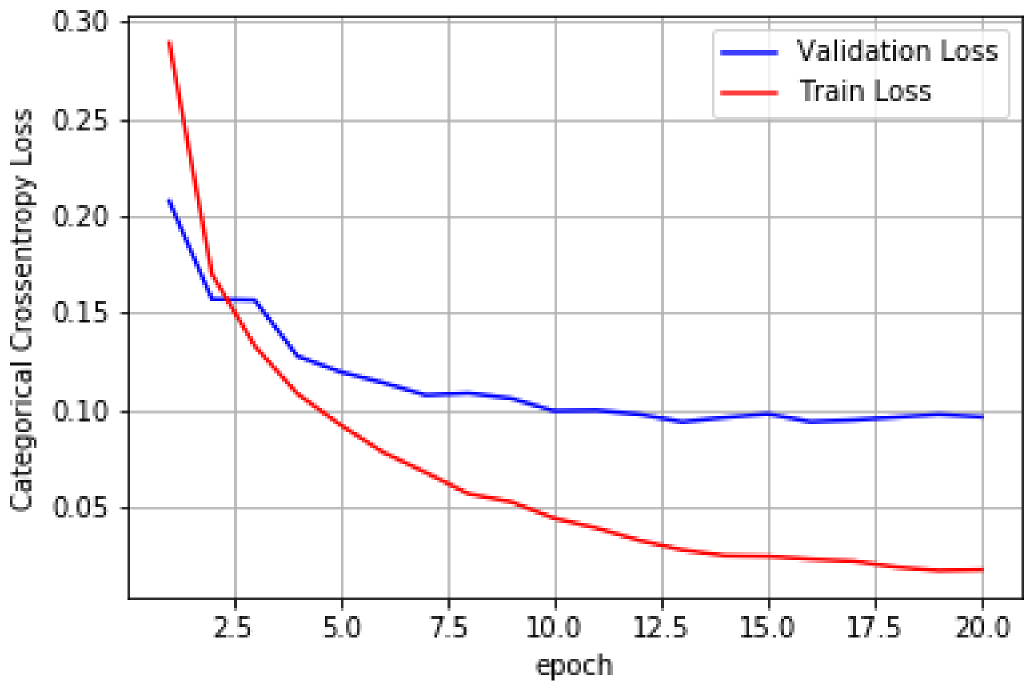
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.2889 -
acc: 0.9148 - val_loss: 0.2075 - val_acc: 0.9382
Epoch 2/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.1700 -
acc: 0.9511 - val_loss: 0.1571 - val_acc: 0.9540
Epoch 3/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.1331 -
acc: 0.9608 - val_loss: 0.1566 - val_acc: 0.9542
Epoch 4/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.1085 -
acc: 0.9673 - val_loss: 0.1279 - val_acc: 0.9623
Epoch 5/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0926 -
acc: 0.9721 - val_loss: 0.1198 - val_acc: 0.9648
Epoch 6/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0785 -
acc: 0.9761 - val_loss: 0.1142 - val_acc: 0.9637
Epoch 7/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0680 -
acc: 0.9784 - val_loss: 0.1078 - val_acc: 0.9681
Epoch 8/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0571 -
acc: 0.9822 - val_loss: 0.1088 - val_acc: 0.9675
Epoch 9/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0530 -
acc: 0.9833 - val_loss: 0.1061 - val_acc: 0.9696
Epoch 10/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0446 -
acc: 0.9862 - val_loss: 0.0997 - val_acc: 0.9708
Epoch 11/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0395 -
acc: 0.9875 - val_loss: 0.1000 - val_acc: 0.9702
Epoch 12/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0331 -
acc: 0.9895 - val_loss: 0.0979 - val_acc: 0.9712
```

```
Epoch 13/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0282 -
acc: 0.9910 - val_loss: 0.0941 - val_acc: 0.9731
Epoch 14/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0253 -
acc: 0.9920 - val_loss: 0.0963 - val_acc: 0.9700
Epoch 15/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0250 -
acc: 0.9917 - val_loss: 0.0981 - val_acc: 0.9726
Epoch 16/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0235 -
acc: 0.9922 - val_loss: 0.0941 - val_acc: 0.9739
Epoch 17/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0225 -
acc: 0.9924 - val_loss: 0.0950 - val_acc: 0.9756
Epoch 18/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0194 -
acc: 0.9940 - val_loss: 0.0964 - val_acc: 0.9741
Epoch 19/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0176 -
acc: 0.9945 - val_loss: 0.0979 - val_acc: 0.9744
Epoch 20/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0180 -
acc: 0.9941 - val_loss: 0.0966 - val_acc: 0.9748
```

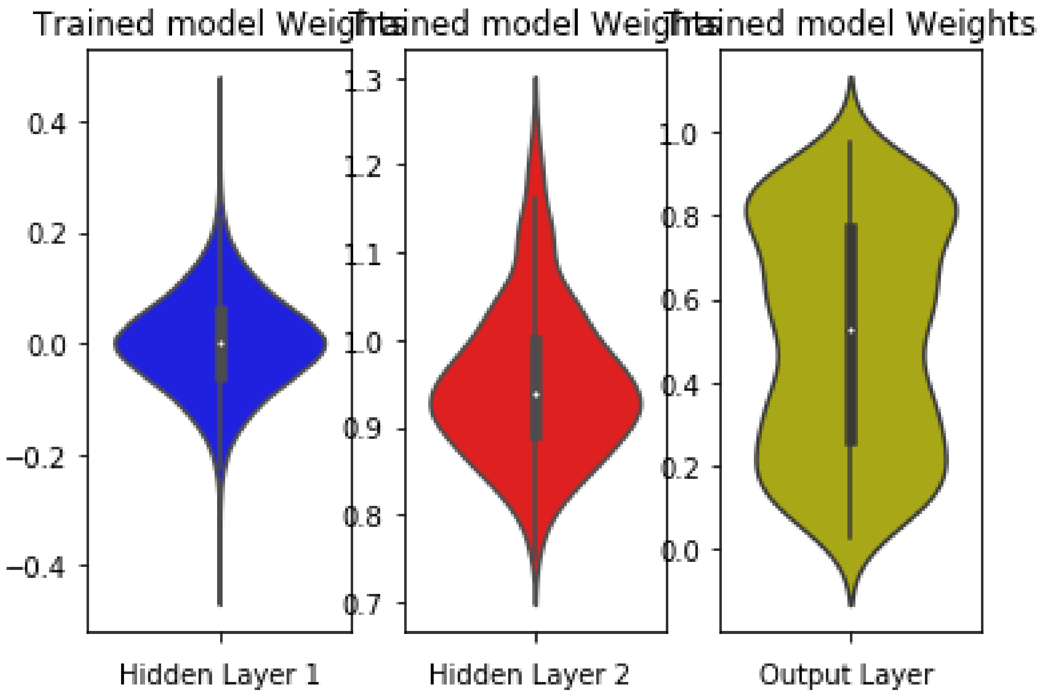
[48]:

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

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

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

# 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, validation_data=(X_test, Y_test))

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

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

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

```
Test score: 0.09660389611198916
Test accuracy: 0.9748
```



[49]:
```
w_after = model_batch.get_weights()

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


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

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

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



5. MLP + Dropout + AdamOptimizer

[50]:
```
# https://stackoverflow.com/questions/34716454/
 ↪where-do-i-call-the-batchnormalization-function-in-keras

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(512, activation='sigmoid', input_shape=(input_dim,),␣
 ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
model_drop.add(Dense(128, activation='sigmoid',␣
 ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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


model_drop.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:3733: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed
in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_30 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_3 (Batch (None, 512)               2048
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_31 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_32 (Dense)             (None, 10)                1290
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

[51]:
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

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

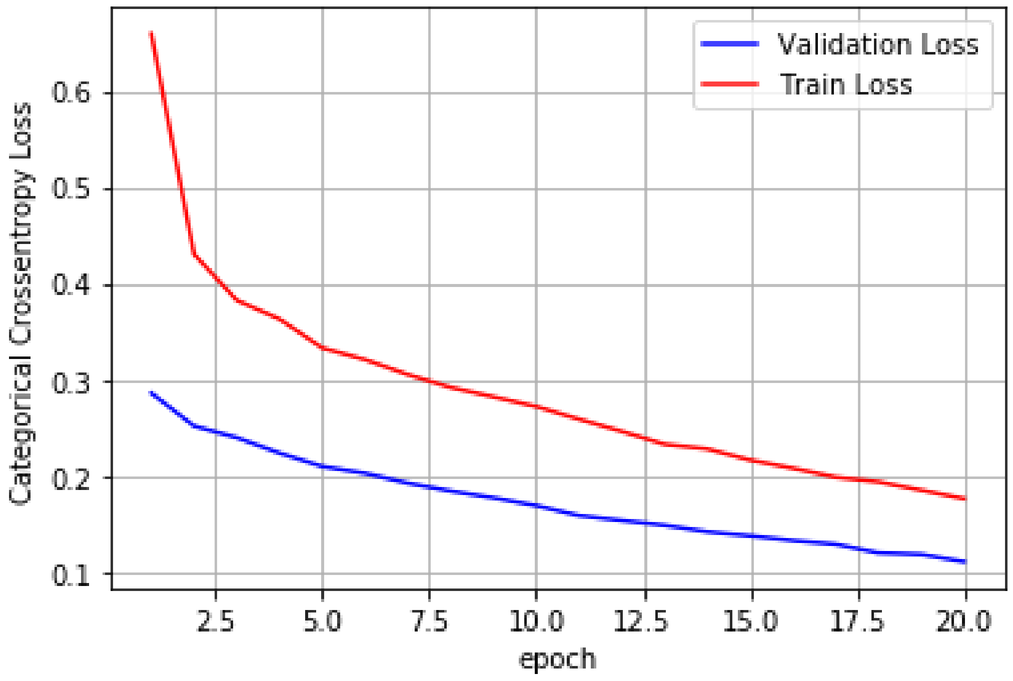
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
```

```
60000/60000 [==============================] - 5s 89us/step - loss: 0.6590 -
acc: 0.7969 - val_loss: 0.2858 - val_acc: 0.9165
Epoch 2/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.4301 -
acc: 0.8697 - val_loss: 0.2516 - val_acc: 0.9276
Epoch 3/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.3822 -
acc: 0.8849 - val_loss: 0.2395 - val_acc: 0.9316
Epoch 4/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.3627 -
acc: 0.8912 - val_loss: 0.2234 - val_acc: 0.9358
Epoch 5/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.3327 -
acc: 0.8998 - val_loss: 0.2095 - val_acc: 0.9373
Epoch 6/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.3207 -
acc: 0.9029 - val_loss: 0.2024 - val_acc: 0.9415
Epoch 7/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.3051 -
acc: 0.9086 - val_loss: 0.1923 - val_acc: 0.9425
Epoch 8/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.2917 -
acc: 0.9113 - val_loss: 0.1841 - val_acc: 0.9452
Epoch 9/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.2819 -
acc: 0.9151 - val_loss: 0.1771 - val_acc: 0.9465
Epoch 10/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.2719 -
acc: 0.9177 - val_loss: 0.1690 - val_acc: 0.9499
Epoch 11/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.2587 -
acc: 0.9216 - val_loss: 0.1586 - val_acc: 0.9511
Epoch 12/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.2460 -
acc: 0.9258 - val_loss: 0.1535 - val_acc: 0.9547
Epoch 13/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.2326 -
acc: 0.9294 - val_loss: 0.1485 - val_acc: 0.9555
Epoch 14/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.2277 -
acc: 0.9308 - val_loss: 0.1417 - val_acc: 0.9578
Epoch 15/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.2162 -
acc: 0.9349 - val_loss: 0.1375 - val_acc: 0.9588
Epoch 16/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.2077 -
acc: 0.9376 - val_loss: 0.1326 - val_acc: 0.9599
Epoch 17/20
```

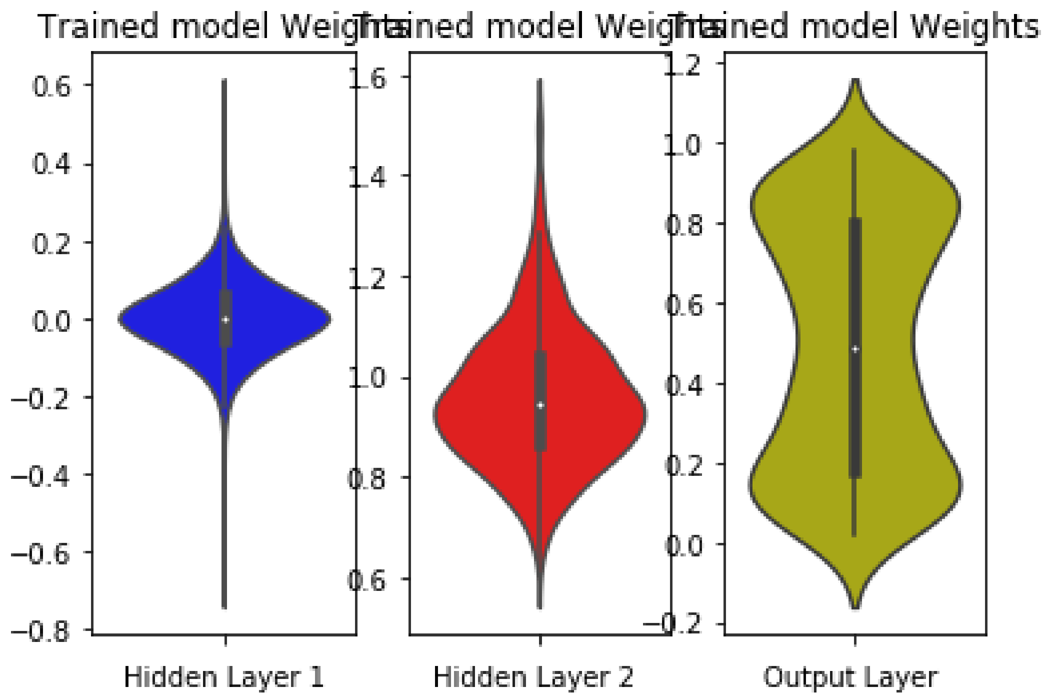
```
60000/60000 [==============================] - 4s 73us/step - loss: 0.1984 -
acc: 0.9394 - val_loss: 0.1288 - val_acc: 0.9617
Epoch 18/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.1937 -
acc: 0.9424 - val_loss: 0.1198 - val_acc: 0.9649
Epoch 19/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.1850 -
acc: 0.9441 - val_loss: 0.1182 - val_acc: 0.9653
Epoch 20/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.1762 -
acc: 0.9459 - val_loss: 0.1107 - val_acc: 0.9678
```

[52]:
```python
score = model_drop.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

# 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, validation_data=(X_test, Y_test))

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

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

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

```
Test score: 0.11071725258678197
Test accuracy: 0.9678
```



[53]:
```
w_after = model_drop.get_weights()

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


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

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

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



Hyper-parameter tuning of Keras models using Sklearn

[0]:
```
from keras.optimizers import Adam,RMSprop,SGD
def best_hyperparameters(activ):

    model = Sequential()
    model.add(Dense(512, activation=activ, input_shape=(input_dim,),␣
 ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
    model.add(Dense(128, activation=activ,␣
 ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
    model.add(Dense(output_dim, activation='softmax'))


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

    return model
```

[0]:
```
# https://machinelearningmastery.com/
 ↪grid-search-hyperparameters-deep-learning-models-python-keras/

activ = ['sigmoid','relu']

from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
```

```
model = KerasClassifier(build_fn=best_hyperparameters, epochs=nb_epoch,␣
↪batch_size=batch_size, verbose=0)
param_grid = dict(activ=activ)

# if you are using CPU
# grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
# if you are using GPU dont use the n_jobs parameter

grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid_result = grid.fit(X_train, Y_train)
```

[56]:
```
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.976467 using {'activ': 'relu'}
0.975167 (0.001694) with: {'activ': 'sigmoid'}
0.976467 (0.000655) with: {'activ': 'relu'}
```

Assignment Work

- Keras
- Google Colab
- 3 Different Architecture, I will use below hidden layers:
  - 2 Hidden Layer 256, 256
  - 3 Hidden Layer 512, 256, 128
  - 5 Hidden Layer 512, 256, 128, 64, 32
- Activation RELU, Optimizator Adam with Batch Normalization and Dropout

[57]:
```
# MLP + RELU + Adam + BN + Dropout (2 Hidden Layer)
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout

model_relu_bn_dropout = Sequential()

model_relu_bn_dropout.add(Dense(256, activation='relu',␣
↪input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.
↪039, seed=None)))
model_relu_bn_dropout.add(BatchNormalization())
model_relu_bn_dropout.add(Dropout(0.5))

model_relu_bn_dropout.add(Dense(256, activation='relu',␣
↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )
model_relu_bn_dropout.add(BatchNormalization())
model_relu_bn_dropout.add(Dropout(0.5))

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

print(model_relu_bn_dropout.summary())

model_relu_bn_dropout.compile(optimizer='adam',␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])

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

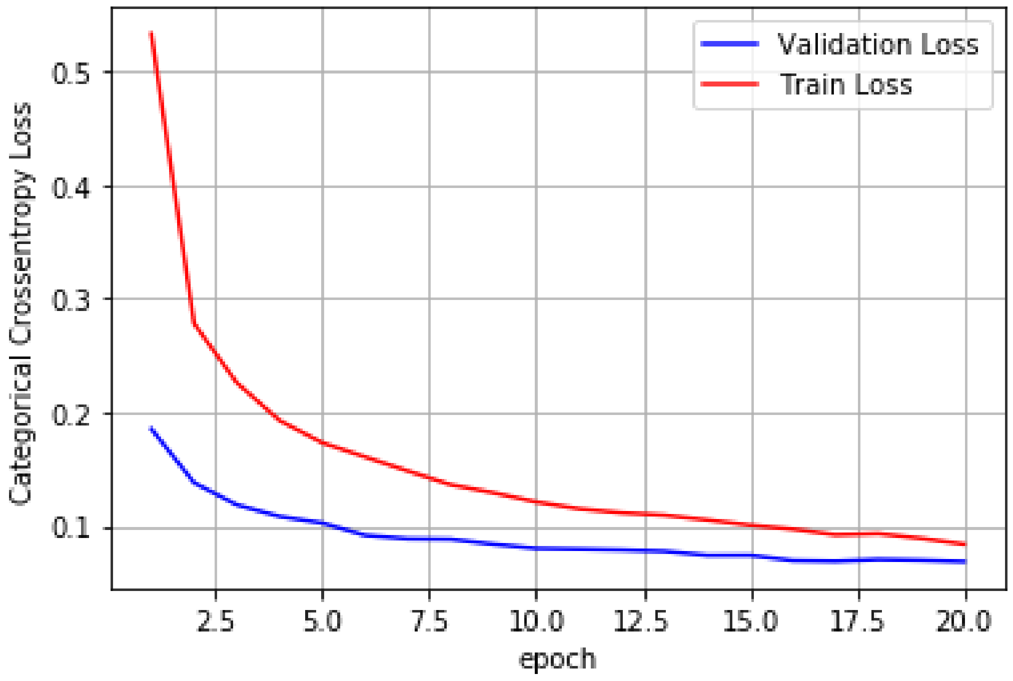
```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_54 (Dense)             (None, 256)               200960
_________________________________________________________________
batch_normalization_5 (Batch (None, 256)               1024
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_55 (Dense)             (None, 256)               65792
_________________________________________________________________
batch_normalization_6 (Batch (None, 256)               1024
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_56 (Dense)             (None, 10)                2570
=================================================================
Total params: 271,370
Trainable params: 270,346
Non-trainable params: 1,024
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.5328 -
acc: 0.8379 - val_loss: 0.1855 - val_acc: 0.9453
Epoch 2/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.2781 -
acc: 0.9163 - val_loss: 0.1382 - val_acc: 0.9575
Epoch 3/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.2261 -
acc: 0.9321 - val_loss: 0.1188 - val_acc: 0.9636
Epoch 4/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.1929 -
```

```
acc: 0.9422 - val_loss: 0.1086 - val_acc: 0.9678
Epoch 5/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.1733 -
acc: 0.9477 - val_loss: 0.1028 - val_acc: 0.9677
Epoch 6/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.1608 -
acc: 0.9502 - val_loss: 0.0919 - val_acc: 0.9722
Epoch 7/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.1484 -
acc: 0.9544 - val_loss: 0.0889 - val_acc: 0.9724
Epoch 8/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.1365 -
acc: 0.9584 - val_loss: 0.0885 - val_acc: 0.9721
Epoch 9/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.1294 -
acc: 0.9605 - val_loss: 0.0841 - val_acc: 0.9749
Epoch 10/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.1213 -
acc: 0.9632 - val_loss: 0.0804 - val_acc: 0.9753
Epoch 11/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.1153 -
acc: 0.9642 - val_loss: 0.0798 - val_acc: 0.9758
Epoch 12/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.1118 -
acc: 0.9659 - val_loss: 0.0791 - val_acc: 0.9775
Epoch 13/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.1096 -
acc: 0.9654 - val_loss: 0.0779 - val_acc: 0.9764
Epoch 14/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.1055 -
acc: 0.9669 - val_loss: 0.0743 - val_acc: 0.9781
Epoch 15/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.1009 -
acc: 0.9686 - val_loss: 0.0744 - val_acc: 0.9785
Epoch 16/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.0974 -
acc: 0.9697 - val_loss: 0.0697 - val_acc: 0.9794
Epoch 17/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.0924 -
acc: 0.9706 - val_loss: 0.0692 - val_acc: 0.9794
Epoch 18/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.0934 -
acc: 0.9707 - val_loss: 0.0707 - val_acc: 0.9786
Epoch 19/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.0892 -
acc: 0.9723 - val_loss: 0.0700 - val_acc: 0.9789
Epoch 20/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0842 -
```

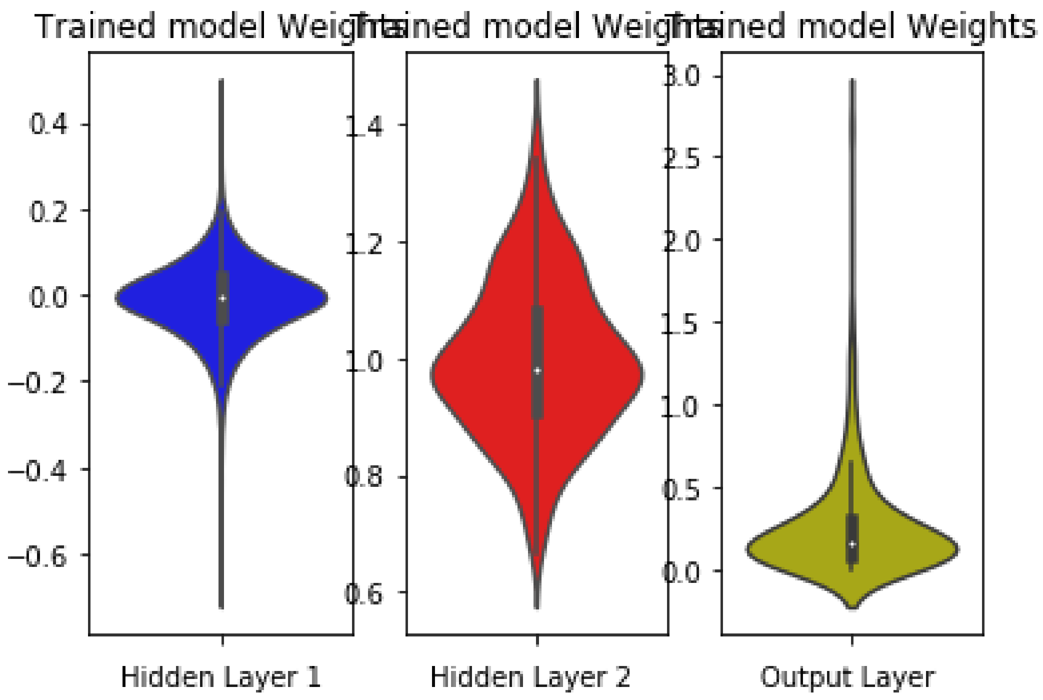
```
acc: 0.9732 - val_loss: 0.0688 - val_acc: 0.9799
```

[58]:
```python
score = model_relu_bn_dropout.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

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


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

```
Test score: 0.06881213052356616
Test accuracy: 0.9799
```



[59]:
```
w_after = model_relu_bn_dropout.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()
```



[60]:
```
# MLP + RELU + Adam + BN + Dropout (3 Hidden Layer)
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout

model_relu_bn_dropout = Sequential()

model_relu_bn_dropout.add(Dense(512, activation='relu',␣
 ↪input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.
 ↪039, seed=None)))
model_relu_bn_dropout.add(BatchNormalization())
model_relu_bn_dropout.add(Dropout(0.5))

model_relu_bn_dropout.add(Dense(256, activation='relu',␣
 ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )
model_relu_bn_dropout.add(BatchNormalization())
model_relu_bn_dropout.add(Dropout(0.5))

model_relu_bn_dropout.add(Dense(128, activation='relu',␣
 ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )
model_relu_bn_dropout.add(BatchNormalization())
model_relu_bn_dropout.add(Dropout(0.5))

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

```
print(model_relu_bn_dropout.summary())

model_relu_bn_dropout.compile(optimizer='adam',␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])

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

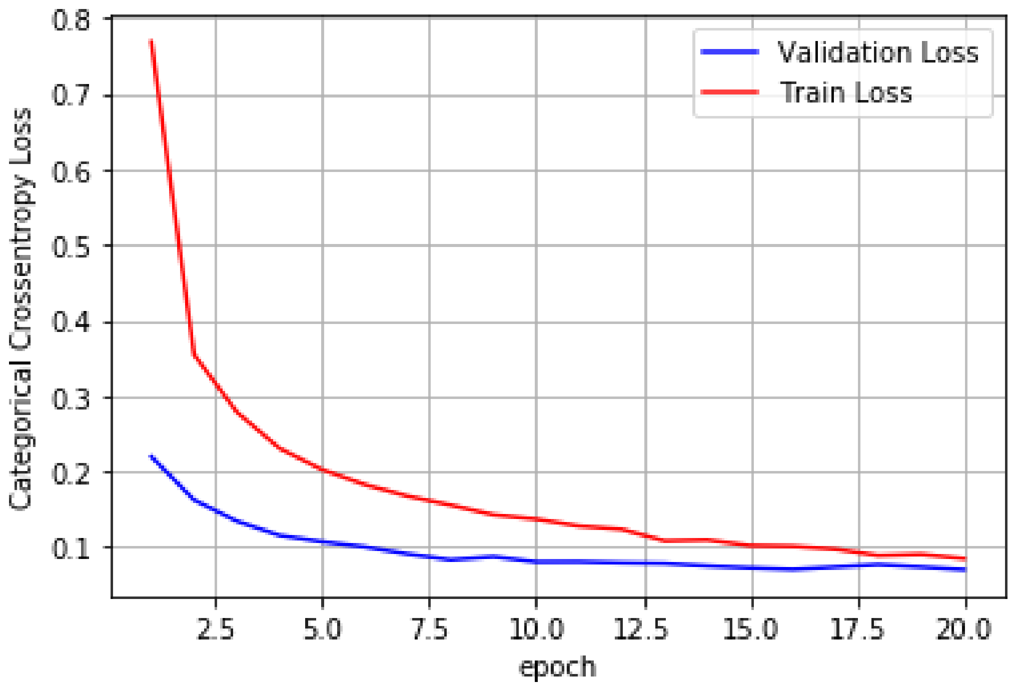
```
Model: "sequential_19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_57 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_7 (Batch (None, 512)               2048
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_58 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_8 (Batch (None, 256)               1024
_________________________________________________________________
dropout_6 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_59 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_9 (Batch (None, 128)               512
_________________________________________________________________
dropout_7 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_60 (Dense)             (None, 10)                1290
=================================================================
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.7689 -
acc: 0.7618 - val_loss: 0.2198 - val_acc: 0.9332
Epoch 2/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.3549 -
acc: 0.8947 - val_loss: 0.1625 - val_acc: 0.9495
Epoch 3/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.2788 -
acc: 0.9191 - val_loss: 0.1344 - val_acc: 0.9594
```

```
Epoch 4/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.2307 -
acc: 0.9321 - val_loss: 0.1154 - val_acc: 0.9644
Epoch 5/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.2023 -
acc: 0.9398 - val_loss: 0.1071 - val_acc: 0.9670
Epoch 6/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.1826 -
acc: 0.9468 - val_loss: 0.1002 - val_acc: 0.9698
Epoch 7/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.1675 -
acc: 0.9504 - val_loss: 0.0908 - val_acc: 0.9731
Epoch 8/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.1556 -
acc: 0.9542 - val_loss: 0.0831 - val_acc: 0.9747
Epoch 9/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.1428 -
acc: 0.9578 - val_loss: 0.0874 - val_acc: 0.9742
Epoch 10/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.1371 -
acc: 0.9595 - val_loss: 0.0804 - val_acc: 0.9777
Epoch 11/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.1279 -
acc: 0.9620 - val_loss: 0.0805 - val_acc: 0.9754
Epoch 12/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.1236 -
acc: 0.9634 - val_loss: 0.0791 - val_acc: 0.9777
Epoch 13/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.1083 -
acc: 0.9683 - val_loss: 0.0784 - val_acc: 0.9775
Epoch 14/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.1095 -
acc: 0.9673 - val_loss: 0.0751 - val_acc: 0.9786
Epoch 15/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.1020 -
acc: 0.9688 - val_loss: 0.0725 - val_acc: 0.9791
Epoch 16/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.1011 -
acc: 0.9696 - val_loss: 0.0705 - val_acc: 0.9813
Epoch 17/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.0973 -
acc: 0.9711 - val_loss: 0.0737 - val_acc: 0.9795
Epoch 18/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0889 -
acc: 0.9735 - val_loss: 0.0769 - val_acc: 0.9791
Epoch 19/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0903 -
acc: 0.9726 - val_loss: 0.0737 - val_acc: 0.9804
```

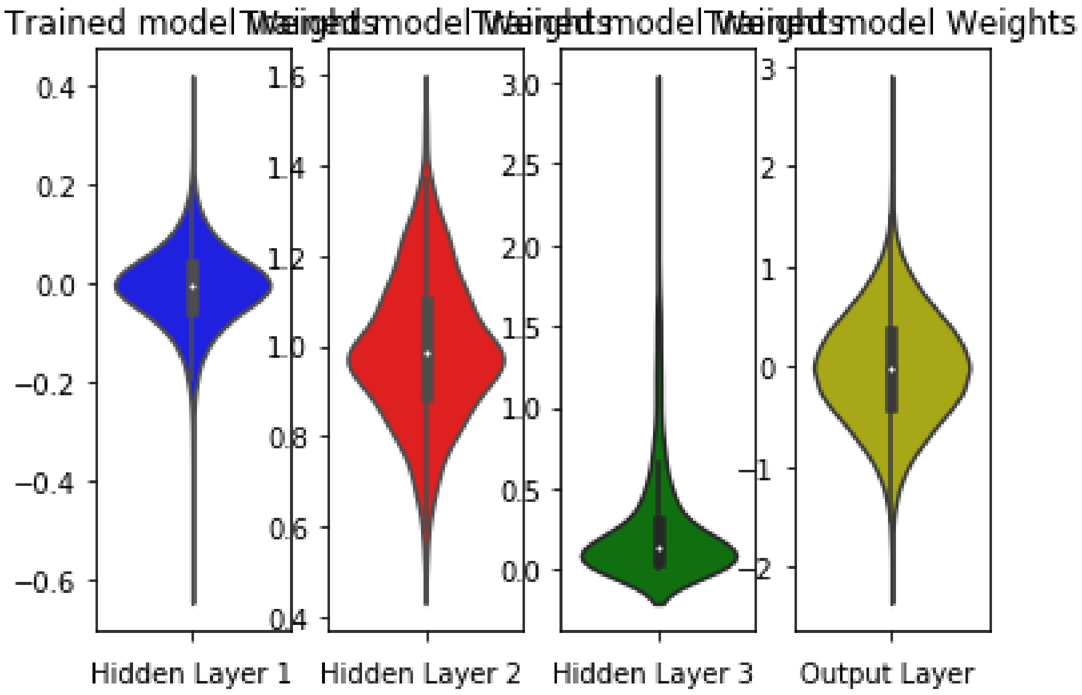
```
Epoch 20/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0849 -
acc: 0.9748 - val_loss: 0.0701 - val_acc: 0.9816
```

[61]:
```
score = model_relu_bn_dropout.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

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


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

```
Test score: 0.07013871098109521
Test accuracy: 0.9816
```



[62]:
```
w_after = model_relu_bn_dropout.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()
```



[63]:
```
# MLP + RELU + Adam + BN + Dropout (5 Hidden Layer)
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout

model_relu_bn_dropout = Sequential()

model_relu_bn_dropout.add(Dense(512, activation='relu',␣
 ↪input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.
 ↪039, seed=None)))
model_relu_bn_dropout.add(BatchNormalization())
model_relu_bn_dropout.add(Dropout(0.5))

model_relu_bn_dropout.add(Dense(256, activation='relu',␣
 ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )
model_relu_bn_dropout.add(BatchNormalization())
model_relu_bn_dropout.add(Dropout(0.5))
model_relu_bn_dropout.add(Dense(128, activation='relu',␣
 ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )
model_relu_bn_dropout.add(BatchNormalization())
model_relu_bn_dropout.add(Dropout(0.5))

model_relu_bn_dropout.add(Dense(64, activation='relu',␣
 ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )
model_relu_bn_dropout.add(BatchNormalization())
model_relu_bn_dropout.add(Dropout(0.5))

model_relu_bn_dropout.add(Dense(32, activation='relu',␣
 ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )
model_relu_bn_dropout.add(BatchNormalization())
model_relu_bn_dropout.add(Dropout(0.5))

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

print(model_relu_bn_dropout.summary())

model_relu_bn_dropout.compile(optimizer='adam',␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])

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

```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_61 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_10 (Batc (None, 512)               2048
_________________________________________________________________
dropout_8 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_62 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_11 (Batc (None, 256)               1024
_________________________________________________________________
dropout_9 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_63 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_12 (Batc (None, 128)               512
_________________________________________________________________
dropout_10 (Dropout)         (None, 128)               0
_________________________________________________________________
```

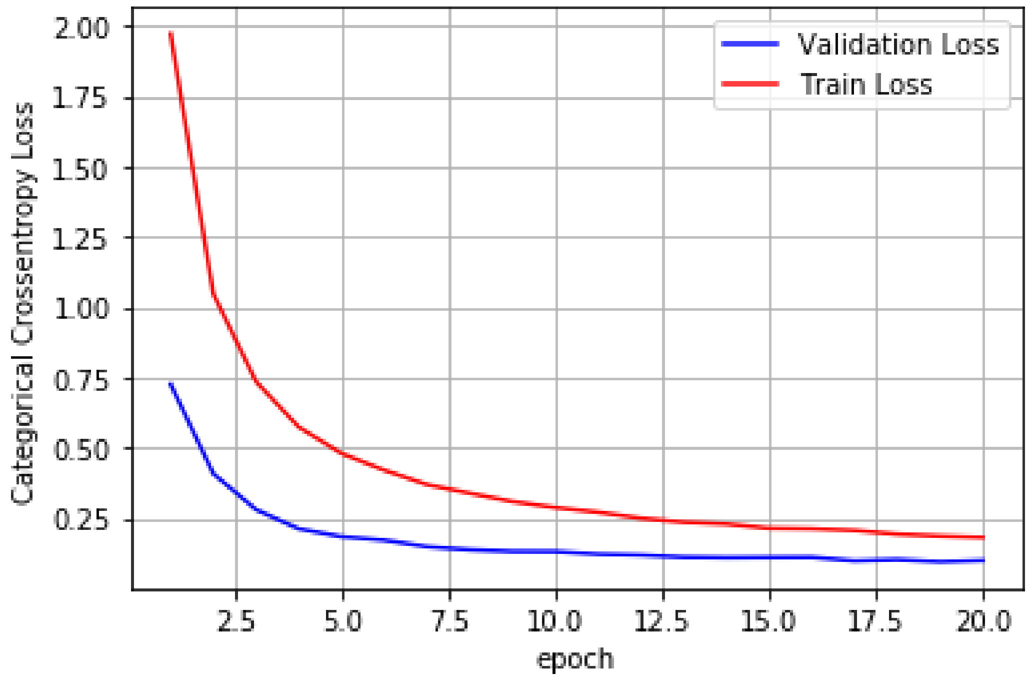
```
dense_64 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_13 (Batc (None, 64)                256
_________________________________________________________________
dropout_11 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_65 (Dense)             (None, 32)                2080
_________________________________________________________________
batch_normalization_14 (Batc (None, 32)                128
_________________________________________________________________
dropout_12 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_66 (Dense)             (None, 10)                330
=================================================================
Total params: 580,778
Trainable params: 578,794
Non-trainable params: 1,984
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 178us/step - loss: 1.9713 -
acc: 0.3386 - val_loss: 0.7261 - val_acc: 0.8496
Epoch 2/20
60000/60000 [==============================] - 8s 136us/step - loss: 1.0472 -
acc: 0.6450 - val_loss: 0.4078 - val_acc: 0.9005
Epoch 3/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.7352 -
acc: 0.7663 - val_loss: 0.2801 - val_acc: 0.9270
Epoch 4/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.5736 -
acc: 0.8321 - val_loss: 0.2125 - val_acc: 0.9435
Epoch 5/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.4798 -
acc: 0.8674 - val_loss: 0.1845 - val_acc: 0.9520
Epoch 6/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.4202 -
acc: 0.8876 - val_loss: 0.1720 - val_acc: 0.9542
Epoch 7/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.3689 -
acc: 0.9039 - val_loss: 0.1487 - val_acc: 0.9591
Epoch 8/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.3382 -
acc: 0.9152 - val_loss: 0.1379 - val_acc: 0.9654
Epoch 9/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.3086 -
acc: 0.9226 - val_loss: 0.1319 - val_acc: 0.9667
Epoch 10/20
```

```
60000/60000 [==============================] - 8s 128us/step - loss: 0.2874 -
acc: 0.9288 - val_loss: 0.1314 - val_acc: 0.9678
Epoch 11/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.2707 -
acc: 0.9353 - val_loss: 0.1222 - val_acc: 0.9699
Epoch 12/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.2500 -
acc: 0.9392 - val_loss: 0.1190 - val_acc: 0.9702
Epoch 13/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.2358 -
acc: 0.9440 - val_loss: 0.1118 - val_acc: 0.9720
Epoch 14/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.2295 -
acc: 0.9462 - val_loss: 0.1101 - val_acc: 0.9737
Epoch 15/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.2143 -
acc: 0.9490 - val_loss: 0.1108 - val_acc: 0.9740
Epoch 16/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.2126 -
acc: 0.9503 - val_loss: 0.1113 - val_acc: 0.9732
Epoch 17/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.2067 -
acc: 0.9526 - val_loss: 0.0987 - val_acc: 0.9771
Epoch 18/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.1933 -
acc: 0.9558 - val_loss: 0.1029 - val_acc: 0.9764
Epoch 19/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.1864 -
acc: 0.9569 - val_loss: 0.0954 - val_acc: 0.9780
Epoch 20/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.1819 -
acc: 0.9575 - val_loss: 0.1001 - val_acc: 0.9768
```

[64]:
```
score = model_relu_bn_dropout.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

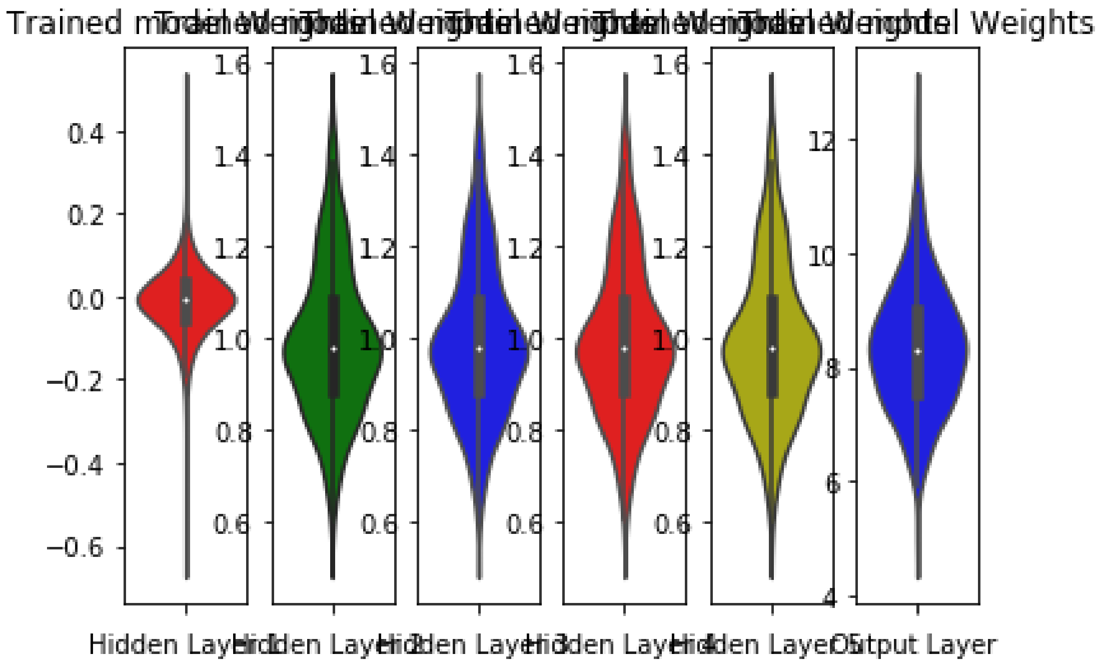
```
# we will get val_loss and val_acc only when you pass the paramter␣
↪validation_data
# val_loss : validation loss
# val_acc : validation accuracy

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


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

```
Test score: 0.10009752484490164
Test accuracy: 0.9768
```



[65]:
```
w_after = model_relu_bn_dropout.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='r')
plt.xlabel('Hidden Layer 1')

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

plt.subplot(1, 6, 3)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='b')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 4 ')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h2_w, color='y')
plt.xlabel('Hidden Layer 5 ')

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



[0]:
```
# MLP + RELU + Adam + BN + Dropout (3 hidden layers is best)
from keras.optimizers import Adam,RMSprop,SGD
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout
def best_hyperparameters(activ):

    model_relu_bn_dropout = Sequential()

    model_relu_bn_dropout.add(Dense(512, activation='relu',␣
↪input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.
↪039, seed=None)))
    model_relu_bn_dropout.add(BatchNormalization())
    model_relu_bn_dropout.add(Dropout(0.5))

    model_relu_bn_dropout.add(Dense(256, activation='relu',␣
↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )
    model_relu_bn_dropout.add(BatchNormalization())
    model_relu_bn_dropout.add(Dropout(0.5))

    model_relu_bn_dropout.add(Dense(128, activation='relu',␣
↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )
    model_relu_bn_dropout.add(BatchNormalization())
    model_relu_bn_dropout.add(Dropout(0.5))

    model_relu_bn_dropout.add(Dense(output_dim, activation='softmax'))
    print(model_relu_bn_dropout.summary())

    model_relu_bn_dropout.compile(optimizer='adam',␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])

    return model_relu_bn_dropout
```

[68]:
```
# https://machinelearningmastery.com/
 ↪grid-search-hyperparameters-deep-learning-models-python-keras/

activ = ['sigmoid','relu']

from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV

model = KerasClassifier(build_fn=best_hyperparameters, epochs=nb_epoch,␣
 ↪batch_size=batch_size, verbose=0)
param_grid = dict(activ=activ)

# if you are using CPU
# grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
# if you are using GPU dont use the n_jobs parameter

grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid_result = grid.fit(X_train, Y_train)
```

```
Model: "sequential_21"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_67 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_15 (Batc (None, 512)               2048
_________________________________________________________________
dropout_13 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_68 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_16 (Batc (None, 256)               1024
_________________________________________________________________
dropout_14 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_69 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_17 (Batc (None, 128)               512
_________________________________________________________________
dropout_15 (Dropout)         (None, 128)               0
```

```
_________________________________________________________________
dense_70 (Dense)             (None, 10)                1290
=================================================================
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
_________________________________________________________________
None
Model: "sequential_22"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_71 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_18 (Batc (None, 512)               2048
_________________________________________________________________
dropout_16 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_72 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_19 (Batc (None, 256)               1024
_________________________________________________________________
dropout_17 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_73 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_20 (Batc (None, 128)               512
_________________________________________________________________
dropout_18 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_74 (Dense)             (None, 10)                1290
=================================================================
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
_________________________________________________________________
None
Model: "sequential_23"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_75 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_21 (Batc (None, 512)               2048
_________________________________________________________________
dropout_19 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_76 (Dense)             (None, 256)               131328
```

```
_________________________________________________________________
batch_normalization_22 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_20 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_77 (Dense)             (None, 128)               32896     
_________________________________________________________________
batch_normalization_23 (Batc (None, 128)               512       
_________________________________________________________________
dropout_21 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_78 (Dense)             (None, 10)                1290      
=================================================================
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
_________________________________________________________________
None
Model: "sequential_24"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_79 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_24 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_22 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_80 (Dense)             (None, 256)               131328    
_________________________________________________________________
batch_normalization_25 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_23 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_81 (Dense)             (None, 128)               32896     
_________________________________________________________________
batch_normalization_26 (Batc (None, 128)               512       
_________________________________________________________________
dropout_24 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_82 (Dense)             (None, 10)                1290      
=================================================================
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
_________________________________________________________________
None
Model: "sequential_25"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_83 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_27 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_25 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_84 (Dense)             (None, 256)               131328    
_________________________________________________________________
batch_normalization_28 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_26 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_85 (Dense)             (None, 128)               32896     
_________________________________________________________________
batch_normalization_29 (Batc (None, 128)               512       
_________________________________________________________________
dropout_27 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_86 (Dense)             (None, 10)                1290      
=================================================================
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
_________________________________________________________________
None
Model: "sequential_26"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_87 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_30 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_28 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_88 (Dense)             (None, 256)               131328    
_________________________________________________________________
batch_normalization_31 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_29 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_89 (Dense)             (None, 128)               32896     
_________________________________________________________________
batch_normalization_32 (Batc (None, 128)               512       
_________________________________________________________________
```

```
dropout_30 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_90 (Dense)             (None, 10)                1290
=================================================================
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
_________________________________________________________________
None
Model: "sequential_27"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_91 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_33 (Batc (None, 512)               2048
_________________________________________________________________
dropout_31 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_92 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_34 (Batc (None, 256)               1024
_________________________________________________________________
dropout_32 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_93 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_35 (Batc (None, 128)               512
_________________________________________________________________
dropout_33 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_94 (Dense)             (None, 10)                1290
=================================================================
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
_________________________________________________________________
None
```

[69]:
```python
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.975800 using {'activ': 'sigmoid'}
```

```
0.975800 (0.001575) with: {'activ': 'sigmoid'}
0.974600 (0.001350) with: {'activ': 'relu'}
```