Implementing various MLP Architectures using Keras.

In [1]:

```
import tensorflow as tf
```

In [2]:

```
# checking if the gpu is connected or not

from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 9299138210179744872
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 4840685568
locality {
  bus_id: 1
  links {
  }
}
incarnation: 2320203755516720165
physical_device_desc: "device: 0, name: GeForce RTX 2060, pci bus id: 000
0:01:00.0, compute capability: 7.5"
]
```

In [3]:

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

```
Using TensorFlow backend.
```

In [4]:

```
%matplotlib notebook
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-- which we will be
calling in the later part of the code.
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()
```

In [5]:

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

In [6]:

```
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 test 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 test examples : 10000 and each image is of shape (28, 28)
```

Now we can see that our input is in the form of a 2-D vector hence we will convert it into a 1-d vector right now that is from 28*28 ---> 1* 784.

In [7]:

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

After conversion

In [8]:

```
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)
```

In [9]:

```
# sample data point
X_train[5]
```

Out[9]:

```
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,   0,   0,  13,
        25, 100, 122,   7,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  33,
       151, 208, 252, 252, 252, 146,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  40,
       152, 244, 252, 253, 224, 211, 252, 232,  40,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  15,
       152, 239, 252, 252, 252, 216,  31,  37, 252, 252,  60,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,  96, 252, 252, 252, 252, 217,  29,   0,  37, 252, 252,  60,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0, 181, 252, 252, 220, 167,  30,   0,   0,  77, 252,
       252,  60,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,  26, 128,  58,  22,   0,   0,   0,   0,
       100, 252, 252,  60,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0, 157, 252, 252,  60,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
       110, 121, 122, 121, 202, 252, 194,   3,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  10,
        53, 179, 253, 253, 255, 253, 253, 228,  35,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   5,
        54, 227, 252, 243, 228, 170, 242, 252, 252, 231, 117,   6,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         6,  78, 252, 252, 125,  59,   0,  18, 208, 252, 252, 252, 252,
        87,   7,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   5, 135, 252, 252, 180,  16,   0,  21, 203, 253, 247, 129,
       173, 252, 252, 184,  66,  49,  49,   0,   0,   0,   0,   0,   0,
         0,   0,   3, 136, 252, 241, 106,  17,   0,  53, 200, 252, 216,
        65,   0,  14,  72, 163, 241, 252, 252, 223,   0,   0,   0,   0,
         0,   0,   0,   0, 105, 252, 242,  88,  18,  73, 170, 244, 252,
       126,  29,   0,   0,   0,   0,   0,  89, 180, 180,  37,   0,   0,
         0,   0,   0,   0,   0,   0, 231, 252, 245, 205, 216, 252, 252,
       252, 124,   3,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0, 207, 252, 252, 252, 252,
       178, 116,  36,   4,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  13,  93, 143,
       121,  23,   6,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
        0,   0,   0,   0,   0,   0,   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)
```

Each cell in the above matrix has value between 0 to 255 but as we will be using these cell values for our ML tasks hence lets normalize these values and bring them between 0 and 1

In [10]:

```
# X => (X - Xmin)/(Xmax-Xmin) = X/255   --- >> note that Xmin = 0

X_train = X_train/255
X_test = X_test/255
```

In [11]:

```
#normalized sample data point
X_train[5]
```

Out[11]:

```
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.        , 0.        , 0.        ,
       0.05098039, 0.09803922, 0.39215686, 0.47843137, 0.02745098,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.12941176, 0.59215686, 0.81568627, 0.98823529,
       0.98823529, 0.98823529, 0.57254902, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.15686275, 0.59607843, 0.95686275,
       0.98823529, 0.99215686, 0.87843137, 0.82745098, 0.98823529,
       0.90980392, 0.15686275, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.05882353, 0.59607843,
       0.9372549 , 0.98823529, 0.98823529, 0.98823529, 0.84705882,
       0.12156863, 0.14509804, 0.98823529, 0.98823529, 0.23529412,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.37647059, 0.98823529, 0.98823529, 0.98823529,
       0.98823529, 0.85098039, 0.11372549, 0.        , 0.14509804,
       0.98823529, 0.98823529, 0.23529412, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.70980392,
       0.98823529, 0.98823529, 0.8627451 , 0.65490196, 0.11764706,
```

```
0.        , 0.        , 0.30196078, 0.98823529, 0.98823529,
0.23529412, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.10196078, 0.50196078, 0.22745098,
0.08627451, 0.        , 0.        , 0.        , 0.        ,
0.39215686, 0.98823529, 0.98823529, 0.23529412, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.61568627, 0.98823529,
0.98823529, 0.23529412, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.43137255, 0.4745098 , 0.47843137,
0.4745098 , 0.79215686, 0.98823529, 0.76078431, 0.01176471,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.03921569, 0.20784314, 0.70196078,
0.99215686, 0.99215686, 1.        , 0.99215686, 0.99215686,
0.89411765, 0.1372549 , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.01960784, 0.21176471,
0.89019608, 0.98823529, 0.95294118, 0.89411765, 0.66666667,
0.94901961, 0.98823529, 0.98823529, 0.90588235, 0.45882353,
0.02352941, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.02352941, 0.30588235, 0.98823529, 0.98823529, 0.49019608,
0.23137255, 0.        , 0.07058824, 0.81568627, 0.98823529,
0.98823529, 0.98823529, 0.98823529, 0.34117647, 0.02745098,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.01960784, 0.52941176, 0.98823529,
0.98823529, 0.70588235, 0.0627451 , 0.        , 0.08235294,
0.79607843, 0.99215686, 0.96862745, 0.50588235, 0.67843137,
0.98823529, 0.98823529, 0.72156863, 0.25882353, 0.19215686,
0.19215686, 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.01176471,
0.53333333, 0.98823529, 0.94509804, 0.41568627, 0.06666667,
0.        , 0.20784314, 0.78431373, 0.98823529, 0.84705882,
0.25490196, 0.        , 0.05490196, 0.28235294, 0.63921569,
0.94509804, 0.98823529, 0.98823529, 0.8745098 , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.41176471, 0.98823529, 0.94901961,
0.34509804, 0.07058824, 0.28627451, 0.66666667, 0.95686275,
0.98823529, 0.49411765, 0.11372549, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.34901961, 0.70588235,
0.70588235, 0.14509804, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.90588235, 0.98823529, 0.96078431, 0.80392157, 0.84705882,
0.98823529, 0.98823529, 0.98823529, 0.48627451, 0.01176471,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.81176471, 0.98823529,
0.98823529, 0.98823529, 0.98823529, 0.69803922, 0.45490196,
```

```
       0.14117647, 0.01568627, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.05098039, 0.36470588, 0.56078431, 0.4745098 ,
       0.09019608, 0.02352941, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        ])
```

Now let us check the class labels.

In [12]:

```
# given label
y_train[5]
```

Out[12]:

```
2
```

Each class label will be a number between 0 to 9 hence let us convert them into vector form using one hot encoding ---> ex: consider an image is 5 convert it into 5 => [0, 0, 0, 0, 0, 1, 0, 0, 0, 0] We are doing so because we will be using the categorical cross entropy loss for our model

In [13]:

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

In [14]:

```
#label after one hot encoding
Y_train[5]
```

Out[14]:

```
array([0., 0., 1., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

In [15]:

```
len(Y_train[5])
```

Out[15]:

```
10
```

Using Keras to build our Sequential Model

In [16]:

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

In [17]:

```
X_train.shape[1]
```

Out[17]:

```
784
```

In [18]:

```
# some model parameters

output_dim = 10 # as our output will be any number from 0 to 9
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

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.

Start building the model

Model-1 Type -> Models with Relu activation + Adam optimizer + Dropouts + Batch Normalization

Model - 1 --> 2 Hidden Layers

Model Architecture --> [(input)--(hidden layer 1 (512 neurons))---(hidden layer 2 (128 neurons))---(output layer)]

In [31]:

```
# Multilayer perceptron

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

In [25]:

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

In [48]:

```
model_final = Sequential()

model_final.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_final.add(BatchNormalization())
model_final.add(Dropout(0.5))

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

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


model_final.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_8 (Dense)              (None, 512)               401920    
_________________________________________________________________
batch_normalization_6 (Batch (None, 512)               2048      
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_9 (Dense)              (None, 128)               65664     
_________________________________________________________________
batch_normalization_7 (Batch (None, 128)               512       
_________________________________________________________________
dropout_7 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_10 (Dense)             (None, 10)                1290      
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

In [49]:

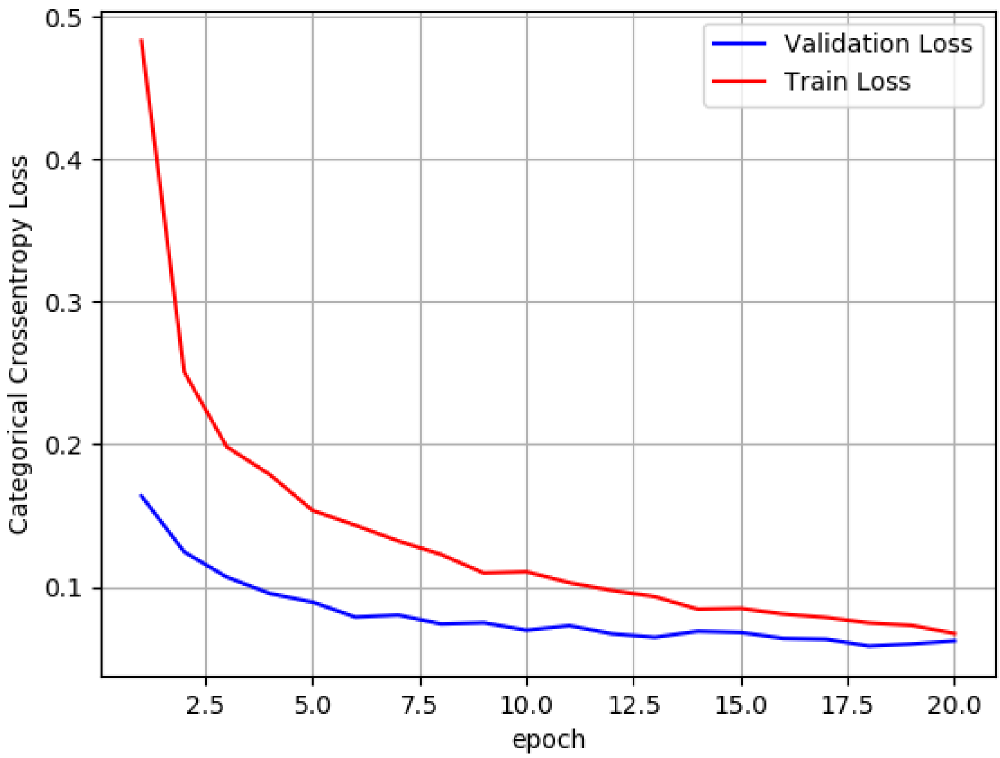
```
model_final.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura
cy'])

history = model_final.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ver
bose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 3s 51us/step - loss: 0.4832
- acc: 0.8522 - val_loss: 0.1638 - val_acc: 0.9504
Epoch 2/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.2508
- acc: 0.9243 - val_loss: 0.1246 - val_acc: 0.9612
Epoch 3/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.1981
- acc: 0.9406 - val_loss: 0.1068 - val_acc: 0.9675
Epoch 4/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.1788
- acc: 0.9466 - val_loss: 0.0953 - val_acc: 0.9707
Epoch 5/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.1536
- acc: 0.9532 - val_loss: 0.0893 - val_acc: 0.9720
Epoch 6/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.1432
- acc: 0.9565 - val_loss: 0.0788 - val_acc: 0.9767
Epoch 7/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.1322
- acc: 0.9591 - val_loss: 0.0802 - val_acc: 0.9750
Epoch 8/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.1228
- acc: 0.9626 - val_loss: 0.0739 - val_acc: 0.9777
Epoch 9/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.1097
- acc: 0.9665 - val_loss: 0.0748 - val_acc: 0.9770
Epoch 10/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.1106
- acc: 0.9663 - val_loss: 0.0696 - val_acc: 0.9794
Epoch 11/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.1029
- acc: 0.9680 - val_loss: 0.0728 - val_acc: 0.9785
Epoch 12/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0974
- acc: 0.9697 - val_loss: 0.0670 - val_acc: 0.9798
Epoch 13/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.0931
- acc: 0.9704 - val_loss: 0.0646 - val_acc: 0.9804
Epoch 14/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0843
- acc: 0.9741 - val_loss: 0.0688 - val_acc: 0.9802
Epoch 15/20
60000/60000 [==============================] - 3s 43us/step - loss: 0.0848
- acc: 0.9735 - val_loss: 0.0680 - val_acc: 0.9790
Epoch 16/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.0809
- acc: 0.9749 - val_loss: 0.0638 - val_acc: 0.9800
Epoch 17/20
60000/60000 [==============================] - 2s 39us/step - loss: 0.0785
- acc: 0.9749 - val_loss: 0.0633 - val_acc: 0.9814
Epoch 18/20
60000/60000 [==============================] - 3s 44us/step - loss: 0.0747
- acc: 0.9760 - val_loss: 0.0585 - val_acc: 0.9821
Epoch 19/20
60000/60000 [==============================] - 2s 41us/step - loss: 0.0730
- acc: 0.9771 - val_loss: 0.0599 - val_acc: 0.9819
Epoch 20/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.0672
- acc: 0.9785 - val_loss: 0.0621 - val_acc: 0.9825
```

In [50]:

```
score = model_final.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, ve
rbose=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 ep
ochs

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

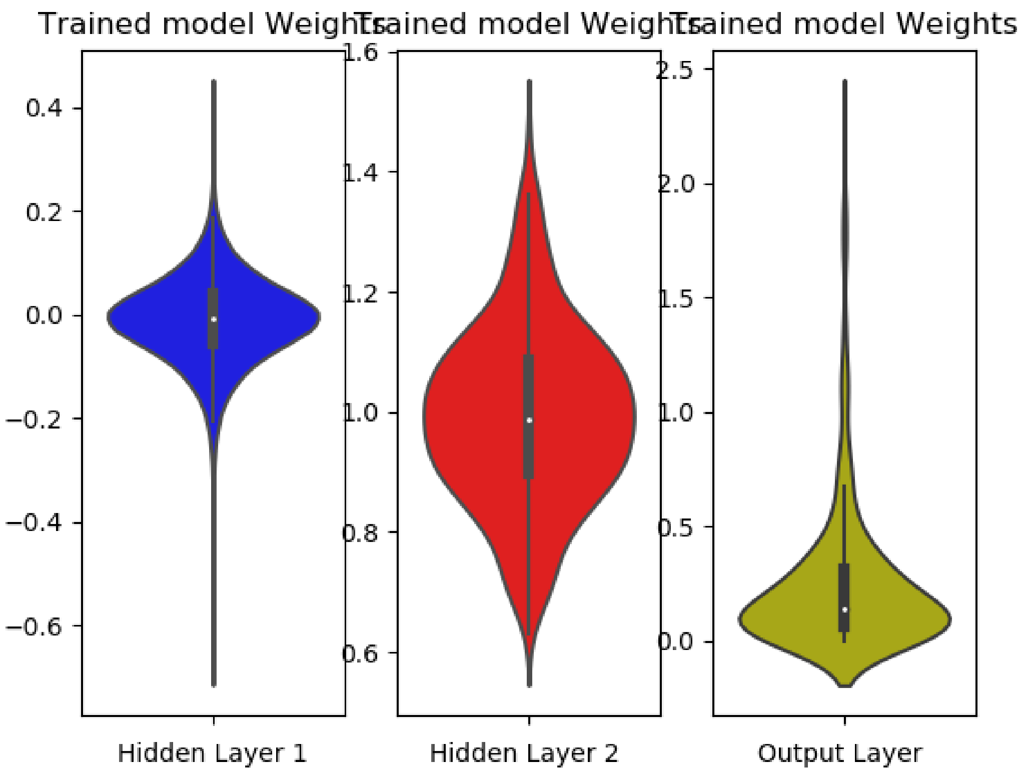
```
Test score: 0.06209656530671637
Test accuracy: 0.9825
```



Let's give a look at the weights which lead to this accuracy of our model.

In [51]:

```
w_after = model_final.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()
```



Model - 1 --> 3 Hidden Layers

Model Architecture --> [(input)--(hidden layer 1 (512 neurons))--(hidden layer 2 (128 neurons))---(hidden layer 3 (256 neurons))---(output layer)]

In [52]:

```
model_final = Sequential()

model_final.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initiali
zer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_final.add(BatchNormalization())
model_final.add(Dropout(0.5))

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

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

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


model_final.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_11 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_8 (Batch (None, 512)               2048      
_________________________________________________________________
dropout_8 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_12 (Dense)             (None, 128)               65664     
_________________________________________________________________
batch_normalization_9 (Batch (None, 128)               512       
_________________________________________________________________
dropout_9 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_13 (Dense)             (None, 256)               33024     
_________________________________________________________________
batch_normalization_10 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_10 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_14 (Dense)             (None, 10)                2570      
=================================================================
Total params: 506,762
Trainable params: 504,970
Non-trainable params: 1,792
_________________________________________________________________
```

In [53]:

```
batch_size = 128
nb_epoch = 40
```

In [54]:

```
model_final.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura
cy'])

history = model_final.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ver
bose=1, validation_data=(X_test, Y_test))
```

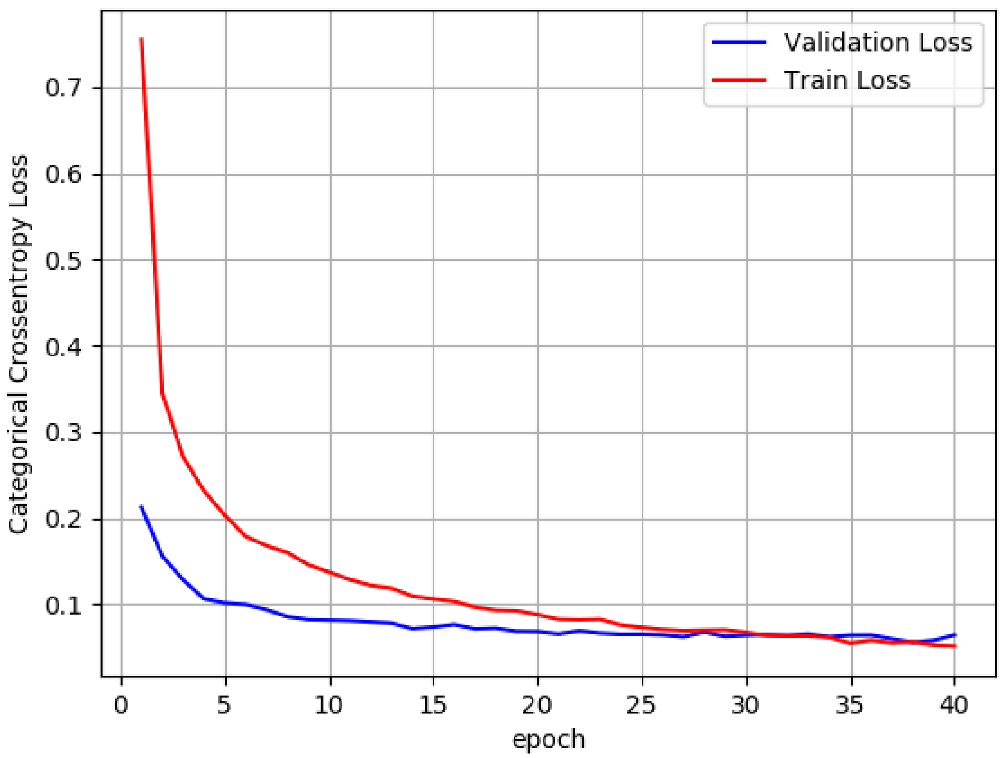
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/40
60000/60000 [==============================] - 4s 69us/step - loss: 0.7553
- acc: 0.7651 - val_loss: 0.2128 - val_acc: 0.9367
Epoch 2/40
60000/60000 [==============================] - 3s 52us/step - loss: 0.3450
- acc: 0.8973 - val_loss: 0.1563 - val_acc: 0.9502
Epoch 3/40
60000/60000 [==============================] - 3s 54us/step - loss: 0.2711
- acc: 0.9201 - val_loss: 0.1286 - val_acc: 0.9595
Epoch 4/40
60000/60000 [==============================] - 3s 52us/step - loss: 0.2319
- acc: 0.9310 - val_loss: 0.1067 - val_acc: 0.9671
Epoch 5/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.2034
- acc: 0.9406 - val_loss: 0.1019 - val_acc: 0.9674
Epoch 6/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.1789
- acc: 0.9473 - val_loss: 0.1002 - val_acc: 0.9708
Epoch 7/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.1682
- acc: 0.9505 - val_loss: 0.0938 - val_acc: 0.9720
Epoch 8/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.1603
- acc: 0.9526 - val_loss: 0.0859 - val_acc: 0.9748
Epoch 9/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.1463
- acc: 0.9561 - val_loss: 0.0824 - val_acc: 0.9745
Epoch 10/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.1376
- acc: 0.9584 - val_loss: 0.0818 - val_acc: 0.9758
Epoch 11/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.1288
- acc: 0.9611 - val_loss: 0.0812 - val_acc: 0.9761
Epoch 12/40
60000/60000 [==============================] - 3s 53us/step - loss: 0.1222
- acc: 0.9626 - val_loss: 0.0798 - val_acc: 0.9757
Epoch 13/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.1189
- acc: 0.9655 - val_loss: 0.0784 - val_acc: 0.9769
Epoch 14/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.1096
- acc: 0.9671 - val_loss: 0.0717 - val_acc: 0.9796
Epoch 15/40
60000/60000 [==============================] - 3s 49us/step - loss: 0.1064
- acc: 0.9683 - val_loss: 0.0737 - val_acc: 0.9788
Epoch 16/40
60000/60000 [==============================] - 3s 49us/step - loss: 0.1036
- acc: 0.9695 - val_loss: 0.0768 - val_acc: 0.9782
Epoch 17/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.0972
- acc: 0.9705 - val_loss: 0.0716 - val_acc: 0.9801
Epoch 18/40
60000/60000 [==============================] - 3s 52us/step - loss: 0.0933
- acc: 0.9714 - val_loss: 0.0723 - val_acc: 0.9807
Epoch 19/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.0927
- acc: 0.9719 - val_loss: 0.0687 - val_acc: 0.9805
Epoch 20/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.0882
- acc: 0.9734 - val_loss: 0.0685 - val_acc: 0.9812
```

```
Epoch 21/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.0828
- acc: 0.9745 - val_loss: 0.0657 - val_acc: 0.9821
Epoch 22/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.0822
- acc: 0.9749 - val_loss: 0.0692 - val_acc: 0.9811
Epoch 23/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.0827
- acc: 0.9755 - val_loss: 0.0666 - val_acc: 0.9812
Epoch 24/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.0763
- acc: 0.9766 - val_loss: 0.0654 - val_acc: 0.9809
Epoch 25/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.0734
- acc: 0.9775 - val_loss: 0.0654 - val_acc: 0.9826
Epoch 26/40
60000/60000 [==============================] - 3s 52us/step - loss: 0.0710
- acc: 0.9781 - val_loss: 0.0646 - val_acc: 0.9823
Epoch 27/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.0696
- acc: 0.9795 - val_loss: 0.0624 - val_acc: 0.9812
Epoch 28/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.0702
- acc: 0.9786 - val_loss: 0.0680 - val_acc: 0.9815
Epoch 29/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.0705
- acc: 0.9789 - val_loss: 0.0628 - val_acc: 0.9827
Epoch 30/40
60000/60000 [==============================] - 3s 52us/step - loss: 0.0675
- acc: 0.9797 - val_loss: 0.0643 - val_acc: 0.9832
Epoch 31/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.0639
- acc: 0.9804 - val_loss: 0.0650 - val_acc: 0.9831
Epoch 32/40
60000/60000 [==============================] - 3s 52us/step - loss: 0.0634
- acc: 0.9800 - val_loss: 0.0642 - val_acc: 0.9828
Epoch 33/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.0632
- acc: 0.9806 - val_loss: 0.0658 - val_acc: 0.9825
Epoch 34/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.0616
- acc: 0.9807 - val_loss: 0.0623 - val_acc: 0.9831
Epoch 35/40
60000/60000 [==============================] - 3s 52us/step - loss: 0.0553
- acc: 0.9830 - val_loss: 0.0643 - val_acc: 0.9828
Epoch 36/40
60000/60000 [==============================] - 3s 52us/step - loss: 0.0584
- acc: 0.9825 - val_loss: 0.0644 - val_acc: 0.9830
Epoch 37/40
60000/60000 [==============================] - 3s 52us/step - loss: 0.0557
- acc: 0.9828 - val_loss: 0.0601 - val_acc: 0.9843
Epoch 38/40
60000/60000 [==============================] - 3s 51us/step - loss: 0.0568
- acc: 0.9821 - val_loss: 0.0559 - val_acc: 0.9847
Epoch 39/40
60000/60000 [==============================] - 3s 52us/step - loss: 0.0531
- acc: 0.9832 - val_loss: 0.0583 - val_acc: 0.9849
Epoch 40/40
60000/60000 [==============================] - 3s 52us/step - loss: 0.0520
- acc: 0.9839 - val_loss: 0.0645 - val_acc: 0.9833
```

In [55]:

```
score = model_final.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, ve
rbose=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 ep
ochs

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

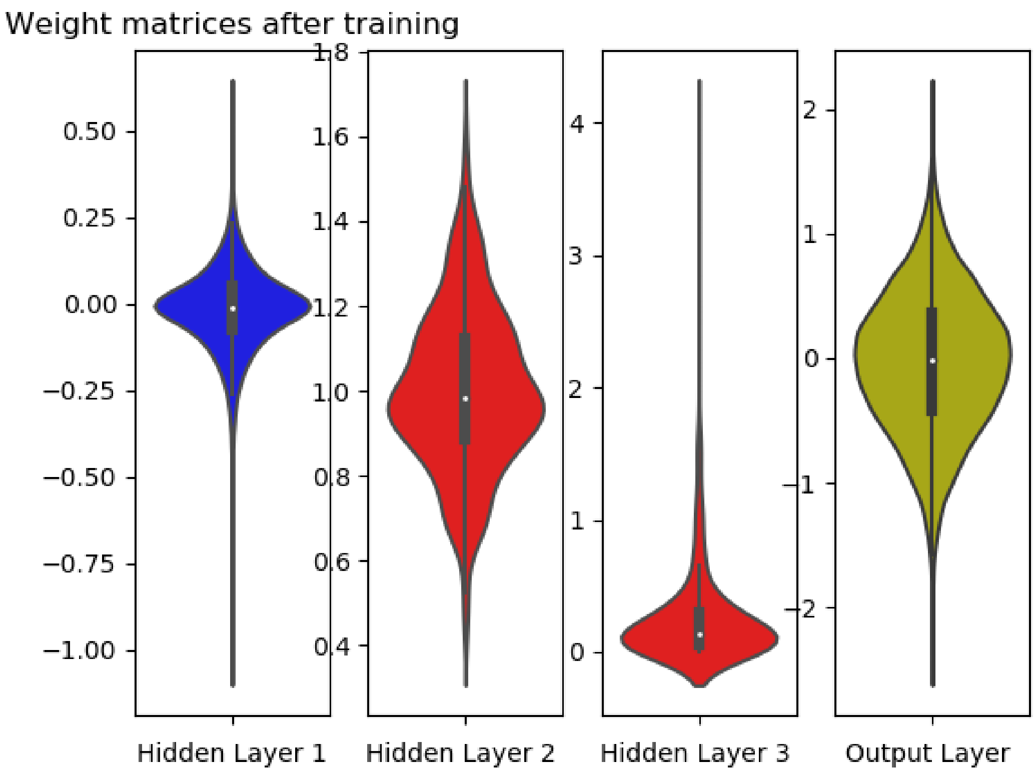
```
Test score: 0.06454481310489937
Test accuracy: 0.9833
```



Let's give a look at the weights which lead to this accuracy of our model.

In [56]:

```
w_after = model_final.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.subplot(1, 4, 1)
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('Hidden Layer 1')

plt.title("Weight matrices after training")
plt.subplot(1, 4, 2)
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot(1, 4, 3)
ax = sns.violinplot(y=h3_w, color='r')
plt.xlabel('Hidden Layer 3 ')

plt.subplot(1, 4, 4)
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')

plt.show()
```



Model - 1 --> 5 Hidden Layers

Model Architecture --> [(input)--(hidden layer 1 (512 neurons))--(hidden layer 2 (128 neurons))---(hidden layer 3 (256 neurons))--(hidden layer 4 (1024 neurons))----(hidden layer 5 (2048 neurons))---(output layer)]

In [57]:

```
model_final = Sequential()

model_final.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initiali
zer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_final.add(BatchNormalization())
model_final.add(Dropout(0.5))

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

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

model_final.add(Dense(1024, activation='relu', kernel_initializer=RandomNormal(mean=0.0
, stddev=0.55, seed=None)) )
model_final.add(BatchNormalization())
model_final.add(Dropout(0.5))

model_final.add(Dense(2048, activation='relu', kernel_initializer=RandomNormal(mean=0.0
, stddev=0.55, seed=None)) )
model_final.add(BatchNormalization())
model_final.add(Dropout(0.5))

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


model_final.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_15 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_11 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_11 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_16 (Dense)             (None, 128)               65664     
_________________________________________________________________
batch_normalization_12 (Batc (None, 128)               512       
_________________________________________________________________
dropout_12 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_17 (Dense)             (None, 256)               33024     
_________________________________________________________________
batch_normalization_13 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_13 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_18 (Dense)             (None, 1024)              263168    
_________________________________________________________________
batch_normalization_14 (Batc (None, 1024)              4096      
_________________________________________________________________
dropout_14 (Dropout)         (None, 1024)              0         
_________________________________________________________________
dense_19 (Dense)             (None, 2048)              2099200   
_________________________________________________________________
batch_normalization_15 (Batc (None, 2048)              8192      
_________________________________________________________________
dropout_15 (Dropout)         (None, 2048)              0         
_________________________________________________________________
dense_20 (Dense)             (None, 10)                20490     
=================================================================
Total params: 2,899,338
Trainable params: 2,891,402
Non-trainable params: 7,936
_________________________________________________________________
```

In [58]:

```
batch_size = 256
nb_epoch = 60
```

In [59]:

```
model_final.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura
cy'])

history = model_final.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ver
bose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/60
60000/60000 [==============================] - 5s 76us/step - loss: 1.3446
- acc: 0.5888 - val_loss: 0.4966 - val_acc: 0.8424
Epoch 2/60
60000/60000 [==============================] - 3s 48us/step - loss: 0.5576
- acc: 0.8277 - val_loss: 0.2985 - val_acc: 0.9067
Epoch 3/60
60000/60000 [==============================] - 3s 47us/step - loss: 0.4277
- acc: 0.8710 - val_loss: 0.2539 - val_acc: 0.9254
Epoch 4/60
60000/60000 [==============================] - 3s 47us/step - loss: 0.3590
- acc: 0.8942 - val_loss: 0.2306 - val_acc: 0.9334
Epoch 5/60
60000/60000 [==============================] - 3s 47us/step - loss: 0.3114
- acc: 0.9078 - val_loss: 0.1989 - val_acc: 0.9455
Epoch 6/60
60000/60000 [==============================] - 3s 49us/step - loss: 0.2770
- acc: 0.9191 - val_loss: 0.1746 - val_acc: 0.9520
Epoch 7/60
60000/60000 [==============================] - 3s 47us/step - loss: 0.2518
- acc: 0.9277 - val_loss: 0.1642 - val_acc: 0.9563
Epoch 8/60
60000/60000 [==============================] - 3s 47us/step - loss: 0.2302
- acc: 0.9326 - val_loss: 0.1355 - val_acc: 0.9624
Epoch 9/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.2146
- acc: 0.9383 - val_loss: 0.1395 - val_acc: 0.9619
Epoch 10/60
60000/60000 [==============================] - 3s 47us/step - loss: 0.1981
- acc: 0.9425 - val_loss: 0.1298 - val_acc: 0.9642
Epoch 11/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.1894
- acc: 0.9453 - val_loss: 0.1244 - val_acc: 0.9677
Epoch 12/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.1773
- acc: 0.9484 - val_loss: 0.1222 - val_acc: 0.9696
Epoch 13/60
60000/60000 [==============================] - 3s 47us/step - loss: 0.1686
- acc: 0.9519 - val_loss: 0.1227 - val_acc: 0.9693
Epoch 14/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.1620
- acc: 0.9530 - val_loss: 0.1176 - val_acc: 0.9706
Epoch 15/60
60000/60000 [==============================] - 3s 47us/step - loss: 0.1559
- acc: 0.9550 - val_loss: 0.1112 - val_acc: 0.9726
Epoch 16/60
60000/60000 [==============================] - 3s 47us/step - loss: 0.1481
- acc: 0.9572 - val_loss: 0.1167 - val_acc: 0.9700
Epoch 17/60
60000/60000 [==============================] - 3s 47us/step - loss: 0.1424
- acc: 0.9594 - val_loss: 0.0982 - val_acc: 0.9758
Epoch 18/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.1356
- acc: 0.9608 - val_loss: 0.0977 - val_acc: 0.9752
Epoch 19/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.1335
- acc: 0.9608 - val_loss: 0.0914 - val_acc: 0.9766
Epoch 20/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.1278
- acc: 0.9639 - val_loss: 0.0896 - val_acc: 0.9786
```

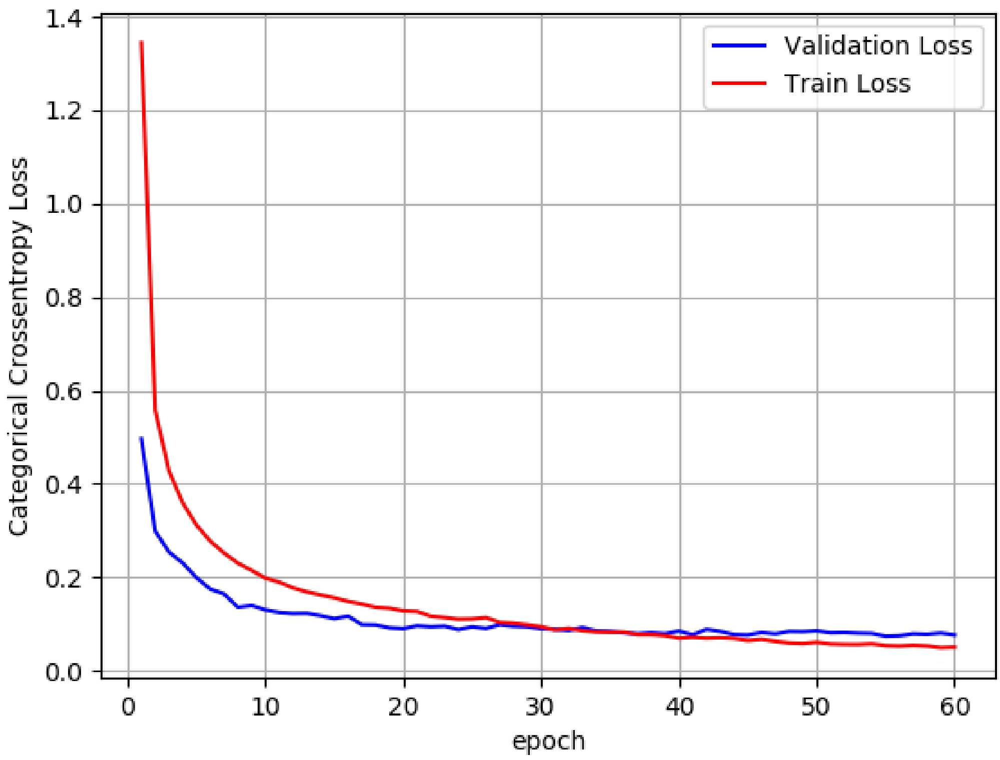
```
Epoch 21/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.1266
- acc: 0.9641 - val_loss: 0.0959 - val_acc: 0.9762
Epoch 22/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.1165
- acc: 0.9665 - val_loss: 0.0937 - val_acc: 0.9780
Epoch 23/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.1137
- acc: 0.9669 - val_loss: 0.0951 - val_acc: 0.9780
Epoch 24/60
60000/60000 [==============================] - 3s 47us/step - loss: 0.1101
- acc: 0.9682 - val_loss: 0.0877 - val_acc: 0.9797
Epoch 25/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.1106
- acc: 0.9676 - val_loss: 0.0937 - val_acc: 0.9779
Epoch 26/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.1135
- acc: 0.9680 - val_loss: 0.0900 - val_acc: 0.9794
Epoch 27/60
60000/60000 [==============================] - 3s 47us/step - loss: 0.1032
- acc: 0.9701 - val_loss: 0.0982 - val_acc: 0.9782
Epoch 28/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.1015
- acc: 0.9705 - val_loss: 0.0950 - val_acc: 0.9777
Epoch 29/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.0981
- acc: 0.9724 - val_loss: 0.0938 - val_acc: 0.9780
Epoch 30/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.0940
- acc: 0.9732 - val_loss: 0.0894 - val_acc: 0.9798
Epoch 31/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.0869
- acc: 0.9750 - val_loss: 0.0885 - val_acc: 0.9800
Epoch 32/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.0901
- acc: 0.9738 - val_loss: 0.0859 - val_acc: 0.9805
Epoch 33/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.0853
- acc: 0.9754 - val_loss: 0.0921 - val_acc: 0.9793
Epoch 34/60
60000/60000 [==============================] - 3s 47us/step - loss: 0.0827
- acc: 0.9757 - val_loss: 0.0850 - val_acc: 0.9811
Epoch 35/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.0820
- acc: 0.9757 - val_loss: 0.0839 - val_acc: 0.9800
Epoch 36/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.0823
- acc: 0.9767 - val_loss: 0.0811 - val_acc: 0.9817
Epoch 37/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.0772
- acc: 0.9782 - val_loss: 0.0789 - val_acc: 0.9820
Epoch 38/60
60000/60000 [==============================] - 3s 47us/step - loss: 0.0772
- acc: 0.9776 - val_loss: 0.0810 - val_acc: 0.9815
Epoch 39/60
60000/60000 [==============================] - 3s 47us/step - loss: 0.0748
- acc: 0.9781 - val_loss: 0.0790 - val_acc: 0.9817
Epoch 40/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.0694
- acc: 0.9794 - val_loss: 0.0846 - val_acc: 0.9812
Epoch 41/60
```

```
60000/60000 [==============================] - 3s 46us/step - loss: 0.0711
- acc: 0.9790 - val_loss: 0.0765 - val_acc: 0.9824
Epoch 42/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.0694
- acc: 0.9796 - val_loss: 0.0883 - val_acc: 0.9816
Epoch 43/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.0703
- acc: 0.9795 - val_loss: 0.0839 - val_acc: 0.9819
Epoch 44/60
60000/60000 [==============================] - 3s 47us/step - loss: 0.0689
- acc: 0.9796 - val_loss: 0.0771 - val_acc: 0.9821
Epoch 45/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.0645
- acc: 0.9812 - val_loss: 0.0761 - val_acc: 0.9829
Epoch 46/60
60000/60000 [==============================] - 3s 47us/step - loss: 0.0671
- acc: 0.9798 - val_loss: 0.0816 - val_acc: 0.9819
Epoch 47/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.0625
- acc: 0.9813 - val_loss: 0.0786 - val_acc: 0.9826
Epoch 48/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.0590
- acc: 0.9825 - val_loss: 0.0838 - val_acc: 0.9817
Epoch 49/60
60000/60000 [==============================] - 3s 47us/step - loss: 0.0579
- acc: 0.9831 - val_loss: 0.0833 - val_acc: 0.9807
Epoch 50/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.0604
- acc: 0.9825 - val_loss: 0.0851 - val_acc: 0.9814
Epoch 51/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.0572
- acc: 0.9833 - val_loss: 0.0809 - val_acc: 0.9834
Epoch 52/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.0563
- acc: 0.9829 - val_loss: 0.0815 - val_acc: 0.9834
Epoch 53/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.0561
- acc: 0.9834 - val_loss: 0.0802 - val_acc: 0.9835
Epoch 54/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.0575
- acc: 0.9831 - val_loss: 0.0796 - val_acc: 0.9830
Epoch 55/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.0536
- acc: 0.9843 - val_loss: 0.0735 - val_acc: 0.9838
Epoch 56/60
60000/60000 [==============================] - 3s 47us/step - loss: 0.0527
- acc: 0.9844 - val_loss: 0.0745 - val_acc: 0.9847
Epoch 57/60
60000/60000 [==============================] - 3s 46us/step - loss: 0.0542
- acc: 0.9842 - val_loss: 0.0787 - val_acc: 0.9832
Epoch 58/60
60000/60000 [==============================] - 3s 47us/step - loss: 0.0528
- acc: 0.9850 - val_loss: 0.0776 - val_acc: 0.9835
Epoch 59/60
60000/60000 [==============================] - 3s 47us/step - loss: 0.0491
- acc: 0.9853 - val_loss: 0.0804 - val_acc: 0.9843
Epoch 60/60
60000/60000 [==============================] - 3s 48us/step - loss: 0.0502
- acc: 0.9852 - val_loss: 0.0763 - val_acc: 0.9841
```

In [60]:

```
score = model_final.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, ve
rbose=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 ep
ochs

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

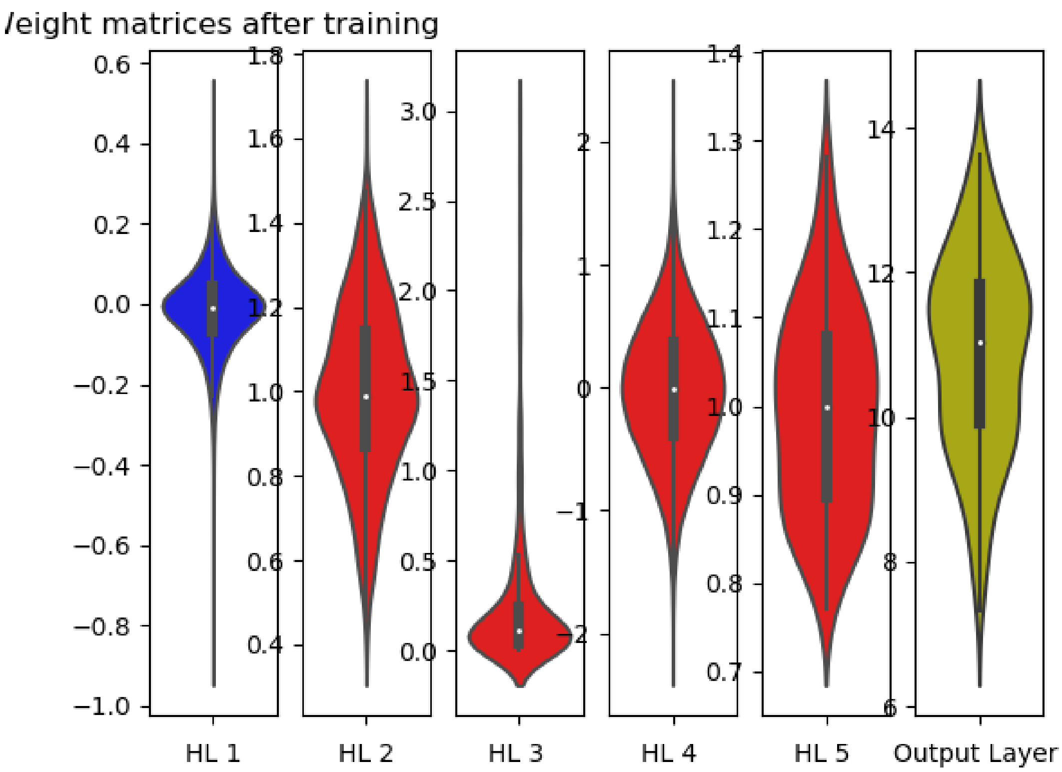
```
Test score: 0.07631430160264717
Test accuracy: 0.9841
```



Let's give a look at the weights which lead to this accuracy of our model.

In [61]:

```
w_after = model_final.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.subplot(1, 6, 1)
ax = sns.violinplot(y=h1_w,color='b')
plt.xlabel('HL 1')

plt.title("Weight matrices after training")
plt.subplot(1, 6, 2)
ax = sns.violinplot(y=h2_w, color='r')
plt.xlabel('HL 2')

plt.subplot(1, 6, 3)
ax = sns.violinplot(y=h3_w, color='r')
plt.xlabel('HL 3 ')

plt.subplot(1, 6, 4)
ax = sns.violinplot(y=h4_w, color='r')
plt.xlabel('HL 4 ')

plt.subplot(1, 6, 5)
ax = sns.violinplot(y=h5_w, color='r')
plt.xlabel('HL 5 ')

plt.subplot(1, 6, 6)
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')

plt.show()
```



Now Randomly trying different MLP architectures with different configurations.

RELU ACTIVATION + ADAM OPTIMIZER + Model Architecture --> [(input)--(hidden layer 1 (512 neurons))--(hidden layer 2 (128 neurons))---(hidden layer 3 (256 neurons))--(hidden layer 4 (1024 neurons))----(hidden layer 5 (2048 neurons))---(output layer)]

In [62]:

```
model_final = Sequential()

model_final.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))


model_final.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )


model_final.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )


model_final.add(Dense(1024, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )


model_final.add(Dense(2048, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )


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


model_final.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_21 (Dense)             (None, 512)               401920    
_________________________________________________________________
dense_22 (Dense)             (None, 128)               65664     
_________________________________________________________________
dense_23 (Dense)             (None, 256)               33024     
_________________________________________________________________
dense_24 (Dense)             (None, 1024)              263168    
_________________________________________________________________
dense_25 (Dense)             (None, 2048)              2099200   
_________________________________________________________________
dense_26 (Dense)             (None, 10)                20490     
=================================================================
Total params: 2,883,466
Trainable params: 2,883,466
Non-trainable params: 0
_________________________________________________________________
```

In [63]:

```
nb_epoch = 40
```

In [64]:

```
model_final.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura
cy'])

history = model_final.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ver
bose=1, validation_data=(X_test, Y_test))
```

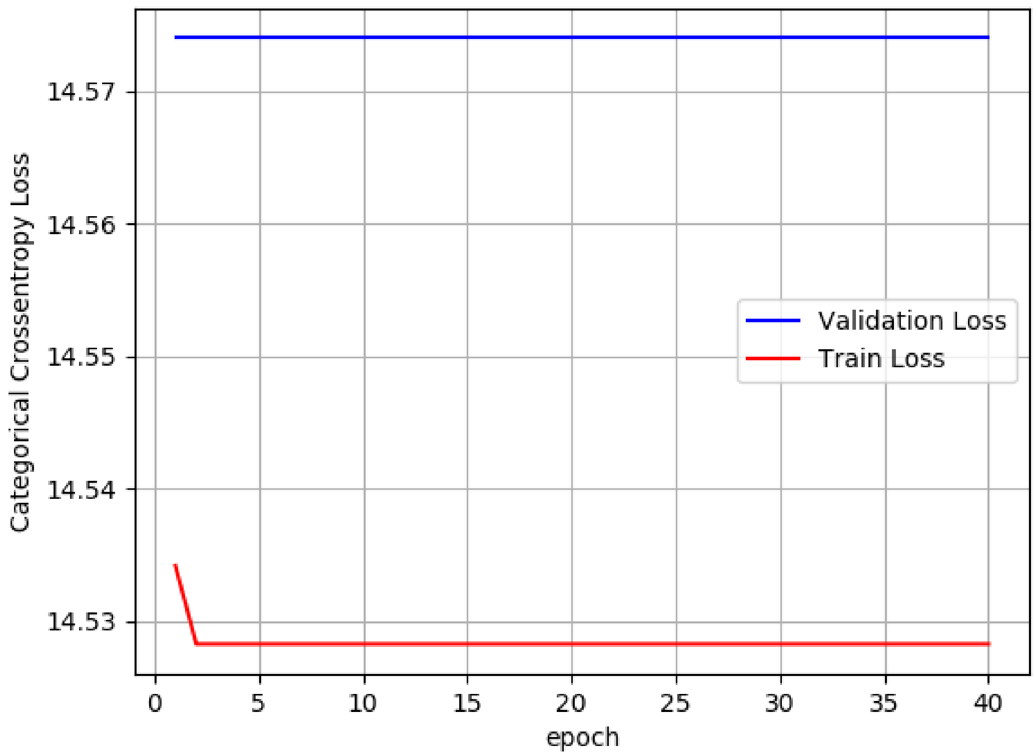
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/40
60000/60000 [==============================] - 3s 44us/step - loss: 14.534
2 - acc: 0.0983 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 2/40
60000/60000 [==============================] - 2s 27us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 3/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 4/40
60000/60000 [==============================] - 2s 29us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 5/40
60000/60000 [==============================] - 2s 29us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 6/40
60000/60000 [==============================] - 2s 29us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 7/40
60000/60000 [==============================] - 2s 29us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 8/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 9/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 10/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 11/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 12/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 13/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 14/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 15/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 16/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 17/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 18/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 19/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 20/40
60000/60000 [==============================] - 2s 29us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
```

```
Epoch 21/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 22/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 23/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 24/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 25/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 26/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 27/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 28/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 29/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 30/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 31/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 32/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 33/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 34/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 35/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 36/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 37/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 38/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 39/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
Epoch 40/40
60000/60000 [==============================] - 2s 28us/step - loss: 14.528
3 - acc: 0.0986 - val_loss: 14.5740 - val_acc: 0.0958
```

In [65]:

```
score = model_final.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, ve
rbose=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 ep
ochs

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

```
Test score: 14.573981648254394
Test accuracy: 0.0958
```



We can clearly see that as we removed the drop out layers and batch normalization we noticed a very poor accuracy for same number of epochs.

RELU ACTIVATION + SGD OPTIMIZER + Batch Normalization + Model Architecture --> [(input)--(hidden layer 1 (512 neurons))--(hidden layer 2 (128 neurons))---(hidden layer 3 (256 neurons))--(hidden layer 4 (1024 neurons))----(hidden layer 5 (2048 neurons))---(output layer)]

In [67]:

```
model_final = Sequential()

model_final.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initiali
zer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_final.add(BatchNormalization())

model_final.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
stddev=0.55, seed=None)) )
model_final.add(BatchNormalization())

model_final.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
stddev=0.55, seed=None)) )
model_final.add(BatchNormalization())

model_final.add(Dense(1024, activation='relu', kernel_initializer=RandomNormal(mean=0.0
, stddev=0.55, seed=None)) )
model_final.add(BatchNormalization())

model_final.add(Dense(2048, activation='relu', kernel_initializer=RandomNormal(mean=0.0
, stddev=0.55, seed=None)) )
model_final.add(BatchNormalization())

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


model_final.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_28 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_16 (Batc (None, 512)               2048      
_________________________________________________________________
dense_29 (Dense)             (None, 128)               65664     
_________________________________________________________________
batch_normalization_17 (Batc (None, 128)               512       
_________________________________________________________________
dense_30 (Dense)             (None, 256)               33024     
_________________________________________________________________
batch_normalization_18 (Batc (None, 256)               1024      
_________________________________________________________________
dense_31 (Dense)             (None, 1024)              263168    
_________________________________________________________________
batch_normalization_19 (Batc (None, 1024)              4096      
_________________________________________________________________
dense_32 (Dense)             (None, 2048)              2099200   
_________________________________________________________________
batch_normalization_20 (Batc (None, 2048)              8192      
_________________________________________________________________
dense_33 (Dense)             (None, 10)                20490     
=================================================================
Total params: 2,899,338
Trainable params: 2,891,402
Non-trainable params: 7,936
_________________________________________________________________
```

In [68]:

```
nb_epoch = 40
batch_size = 128
```

In [69]:

```
model_final.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accurac
y'])

history = model_final.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ver
bose=1, validation_data=(X_test, Y_test))
```

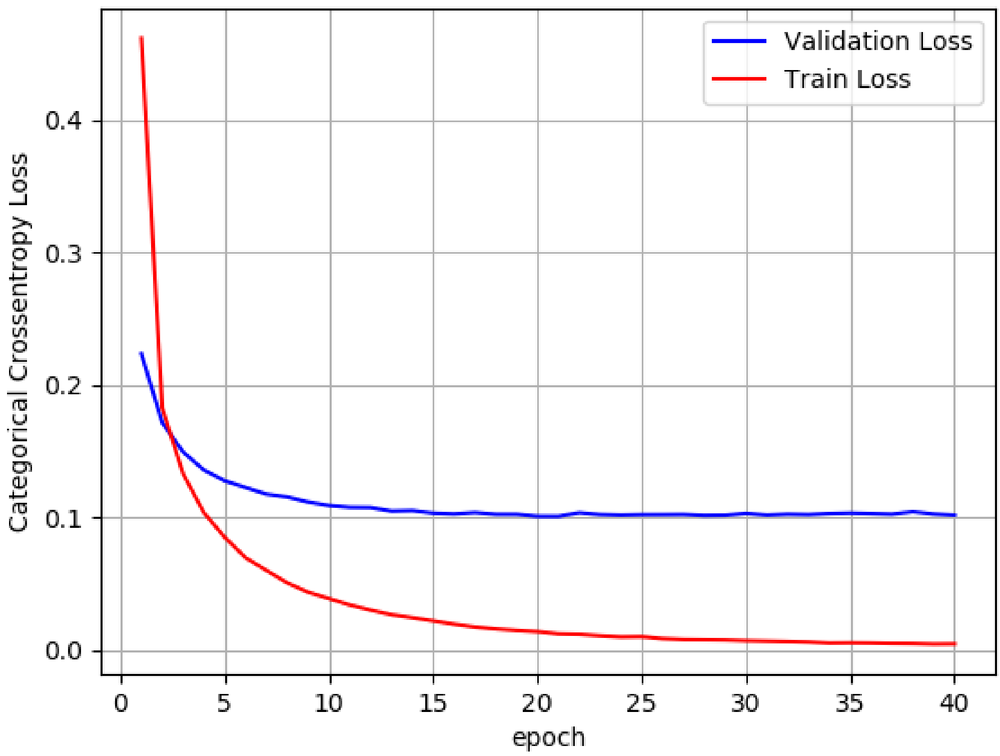
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/40
60000/60000 [==============================] - 5s 90us/step - loss: 0.4617
- acc: 0.8571 - val_loss: 0.2238 - val_acc: 0.9308
Epoch 2/40
60000/60000 [==============================] - 4s 68us/step - loss: 0.1828
- acc: 0.9455 - val_loss: 0.1717 - val_acc: 0.9482
Epoch 3/40
60000/60000 [==============================] - 4s 66us/step - loss: 0.1333
- acc: 0.9603 - val_loss: 0.1496 - val_acc: 0.9549
Epoch 4/40
60000/60000 [==============================] - 4s 66us/step - loss: 0.1038
- acc: 0.9696 - val_loss: 0.1361 - val_acc: 0.9595
Epoch 5/40
60000/60000 [==============================] - 4s 67us/step - loss: 0.0853
- acc: 0.9755 - val_loss: 0.1279 - val_acc: 0.9608
Epoch 6/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0699
- acc: 0.9804 - val_loss: 0.1229 - val_acc: 0.9628
Epoch 7/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0603
- acc: 0.9825 - val_loss: 0.1178 - val_acc: 0.9646
Epoch 8/40
60000/60000 [==============================] - 4s 66us/step - loss: 0.0509
- acc: 0.9857 - val_loss: 0.1157 - val_acc: 0.9658
Epoch 9/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0439
- acc: 0.9884 - val_loss: 0.1119 - val_acc: 0.9673
Epoch 10/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0391
- acc: 0.9898 - val_loss: 0.1093 - val_acc: 0.9681
Epoch 11/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0343
- acc: 0.9917 - val_loss: 0.1079 - val_acc: 0.9689
Epoch 12/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0304
- acc: 0.9925 - val_loss: 0.1078 - val_acc: 0.9680
Epoch 13/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0269
- acc: 0.9936 - val_loss: 0.1051 - val_acc: 0.9686
Epoch 14/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0247
- acc: 0.9945 - val_loss: 0.1054 - val_acc: 0.9691
Epoch 15/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0223
- acc: 0.9951 - val_loss: 0.1035 - val_acc: 0.9700
Epoch 16/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0198
- acc: 0.9961 - val_loss: 0.1030 - val_acc: 0.9700
Epoch 17/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0176
- acc: 0.9969 - val_loss: 0.1040 - val_acc: 0.9702
Epoch 18/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0162
- acc: 0.9971 - val_loss: 0.1027 - val_acc: 0.9703
Epoch 19/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0151
- acc: 0.9975 - val_loss: 0.1027 - val_acc: 0.9709
Epoch 20/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0142
- acc: 0.9974 - val_loss: 0.1012 - val_acc: 0.9720
```

```
Epoch 21/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0125
- acc: 0.9981 - val_loss: 0.1011 - val_acc: 0.9715
Epoch 22/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0122
- acc: 0.9978 - val_loss: 0.1039 - val_acc: 0.9710
Epoch 23/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0111
- acc: 0.9983 - val_loss: 0.1025 - val_acc: 0.9714
Epoch 24/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0103
- acc: 0.9986 - val_loss: 0.1020 - val_acc: 0.9719
Epoch 25/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0105
- acc: 0.9983 - val_loss: 0.1024 - val_acc: 0.9710
Epoch 26/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0089
- acc: 0.9989 - val_loss: 0.1024 - val_acc: 0.9718
Epoch 27/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0083
- acc: 0.9990 - val_loss: 0.1025 - val_acc: 0.9709
Epoch 28/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0081
- acc: 0.9988 - val_loss: 0.1017 - val_acc: 0.9714
Epoch 29/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0079
- acc: 0.9990 - val_loss: 0.1019 - val_acc: 0.9713
Epoch 30/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0074
- acc: 0.9992 - val_loss: 0.1034 - val_acc: 0.9707
Epoch 31/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0071
- acc: 0.9992 - val_loss: 0.1021 - val_acc: 0.9717
Epoch 32/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0068
- acc: 0.9992 - val_loss: 0.1027 - val_acc: 0.9719
Epoch 33/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0064
- acc: 0.9993 - val_loss: 0.1024 - val_acc: 0.9720
Epoch 34/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0056
- acc: 0.9995 - val_loss: 0.1032 - val_acc: 0.9713
Epoch 35/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0058
- acc: 0.9994 - val_loss: 0.1036 - val_acc: 0.9716
Epoch 36/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0056
- acc: 0.9993 - val_loss: 0.1032 - val_acc: 0.9725
Epoch 37/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0054
- acc: 0.9993 - val_loss: 0.1028 - val_acc: 0.9719
Epoch 38/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0052
- acc: 0.9994 - val_loss: 0.1046 - val_acc: 0.9718
Epoch 39/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0049
- acc: 0.9995 - val_loss: 0.1029 - val_acc: 0.9724
Epoch 40/40
60000/60000 [==============================] - 4s 65us/step - loss: 0.0050
- acc: 0.9996 - val_loss: 0.1020 - val_acc: 0.9729
```

In [70]:

```
score = model_final.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, ve
rbose=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 ep
ochs

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

```
Test score: 0.10197011248232156
Test accuracy: 0.9729
```



Sigmoid activation +adam optimizer+ Dropout(with rate = 0.75) + Model Architecture --> [(input)--(hidden layer 1 (512 neurons))--(hidden layer 2(128 neurons))---(hidden layer 3 (256 neurons))---(output layer)]

In [34]:

```
model_final = Sequential()
model_final.add(Dense(512, activation='sigmoid', input_shape=(input_dim,), kernel_initi
alizer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_final.add(Dropout(0.75))

model_final.add(Dense(128, activation='sigmoid', kernel_initializer=RandomNormal(mean=
0.0,stddev=0.55, seed=None)) )
model_final.add(Dropout(0.75))

model_final.add(Dense(256, activation='sigmoid', kernel_initializer=RandomNormal(mean=
0.0,stddev=0.55, seed=None)) )
model_final.add(Dropout(0.75))

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

```
WARNING: Logging before flag parsing goes to stderr.
W1002 10:26:23.843827  9664 deprecation_wrapper.py:119] From C:\Users\RASH
U TYAGI\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:7
4: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.ge
t_default_graph instead.

W1002 10:26:23.882996  9664 deprecation_wrapper.py:119] From C:\Users\RASH
U TYAGI\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:51
7: The name tf.placeholder is deprecated. Please use tf.compat.v1.placehol
der instead.

W1002 10:26:23.893732  9664 deprecation_wrapper.py:119] From C:\Users\RASH
U TYAGI\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:41
15: The name tf.random_normal is deprecated. Please use tf.random.normal i
nstead.

W1002 10:26:23.910495  9664 deprecation_wrapper.py:119] From C:\Users\RASH
U TYAGI\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:13
3: The name tf.placeholder_with_default is deprecated. Please use tf.compa
t.v1.placeholder_with_default instead.

W1002 10:26:23.916351  9664 deprecation.py:506] From C:\Users\RASHU TYAGI
\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:3445: cal
ling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is depreca
ted 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`.
W1002 10:26:23.917327  9664 nn_ops.py:4224] Large dropout rate: 0.75 (>0.
5). In TensorFlow 2.x, dropout() uses dropout rate instead of keep_prob. P
lease ensure that this is intended.
W1002 10:26:23.939802  9664 nn_ops.py:4224] Large dropout rate: 0.75 (>0.
5). In TensorFlow 2.x, dropout() uses dropout rate instead of keep_prob. P
lease ensure that this is intended.
W1002 10:26:23.959618  9664 nn_ops.py:4224] Large dropout rate: 0.75 (>0.
5). In TensorFlow 2.x, dropout() uses dropout rate instead of keep_prob. P
lease ensure that this is intended.
W1002 10:26:23.968427  9664 deprecation_wrapper.py:119] From C:\Users\RASH
U TYAGI\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:41
38: The name tf.random_uniform is deprecated. Please use tf.random.uniform
instead.
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 512)               401920    
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 128)               65664     
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 256)               33024     
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 10)                2570      
=================================================================
Total params: 503,178
Trainable params: 503,178
Non-trainable params: 0
_________________________________________________________________
```

In [35]:

```
nb_epoch = 40
```

In [36]:

```
model_final.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura
cy'])
history = model_final.fit(X_train, Y_train, epochs=nb_epoch, verbose=1, validation_data
=(X_test, Y_test))
```

```
W1002 10:28:01.442343  9664 deprecation_wrapper.py:119] From C:\Users\RASH
U TYAGI\Anaconda3\lib\site-packages\keras\optimizers.py:790: The name tf.t
rain.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer inst
ead.

W1002 10:28:01.545196  9664 deprecation.py:323] From C:\Users\RASHU TYAGI
\Anaconda3\lib\site-packages\tensorflow\python\ops\math_grad.py:1250: add_
dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) i
s 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
```

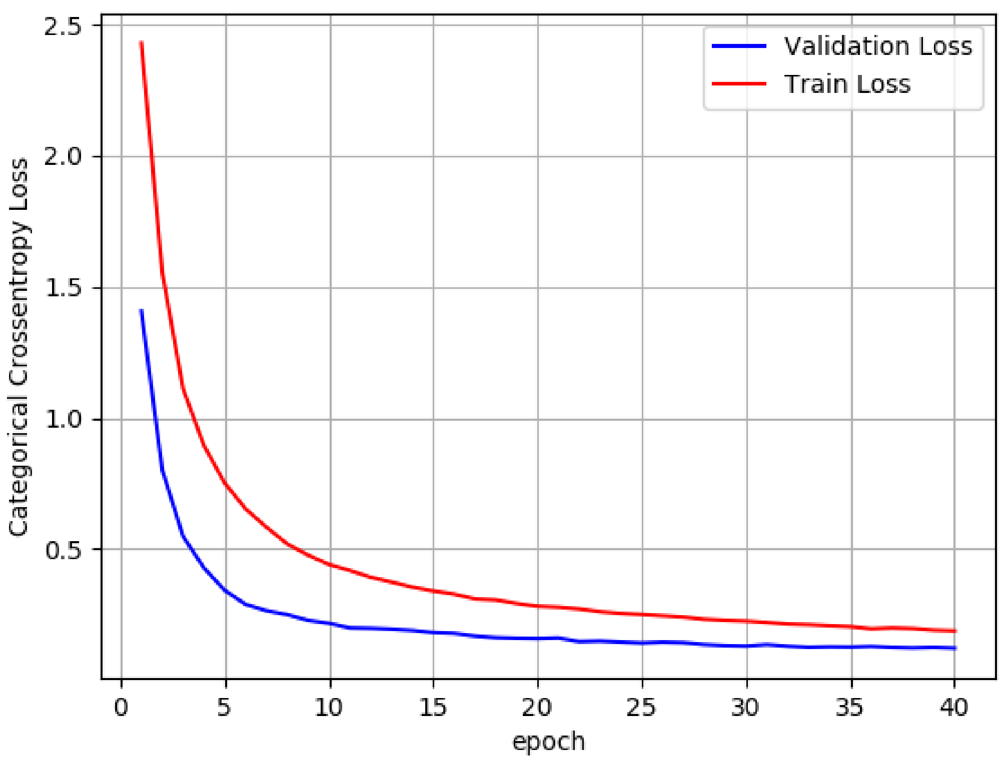
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/40
60000/60000 [==============================] - 8s 128us/step - loss: 2.428
7 - acc: 0.1483 - val_loss: 1.4073 - val_acc: 0.5483
Epoch 2/40
60000/60000 [==============================] - 6s 99us/step - loss: 1.5549
- acc: 0.4082 - val_loss: 0.8020 - val_acc: 0.7208
Epoch 3/40
60000/60000 [==============================] - 6s 99us/step - loss: 1.1108
- acc: 0.5945 - val_loss: 0.5479 - val_acc: 0.8673
Epoch 4/40
60000/60000 [==============================] - 6s 101us/step - loss: 0.894
7 - acc: 0.6925 - val_loss: 0.4284 - val_acc: 0.8903
Epoch 5/40
60000/60000 [==============================] - 6s 99us/step - loss: 0.7512
- acc: 0.7584 - val_loss: 0.3424 - val_acc: 0.9213
Epoch 6/40
60000/60000 [==============================] - 6s 100us/step - loss: 0.652
9 - acc: 0.8037 - val_loss: 0.2887 - val_acc: 0.9314
Epoch 7/40
60000/60000 [==============================] - 6s 99us/step - loss: 0.5830
- acc: 0.8317 - val_loss: 0.2646 - val_acc: 0.9368
Epoch 8/40
60000/60000 [==============================] - 6s 97us/step - loss: 0.5192
- acc: 0.8549 - val_loss: 0.2496 - val_acc: 0.9413
Epoch 9/40
60000/60000 [==============================] - 6s 97us/step - loss: 0.4765
- acc: 0.8712 - val_loss: 0.2278 - val_acc: 0.9447
Epoch 10/40
60000/60000 [==============================] - 6s 99us/step - loss: 0.4411
- acc: 0.8819 - val_loss: 0.2166 - val_acc: 0.9492
Epoch 11/40
60000/60000 [==============================] - 6s 99us/step - loss: 0.4183
- acc: 0.8907 - val_loss: 0.1989 - val_acc: 0.9534
Epoch 12/40
60000/60000 [==============================] - 6s 99us/step - loss: 0.3925
- acc: 0.9003 - val_loss: 0.1973 - val_acc: 0.9544
Epoch 13/40
60000/60000 [==============================] - 6s 99us/step - loss: 0.3740
- acc: 0.9039 - val_loss: 0.1941 - val_acc: 0.9564
Epoch 14/40
60000/60000 [==============================] - 6s 98us/step - loss: 0.3547
- acc: 0.9098 - val_loss: 0.1894 - val_acc: 0.9559
Epoch 15/40
60000/60000 [==============================] - 6s 99us/step - loss: 0.3401
- acc: 0.9157 - val_loss: 0.1811 - val_acc: 0.9591
Epoch 16/40
60000/60000 [==============================] - 6s 98us/step - loss: 0.3280
- acc: 0.9181 - val_loss: 0.1781 - val_acc: 0.9592
Epoch 17/40
60000/60000 [==============================] - 6s 98us/step - loss: 0.3098
- acc: 0.9238 - val_loss: 0.1679 - val_acc: 0.9614
Epoch 18/40
60000/60000 [==============================] - 6s 99us/step - loss: 0.3059
- acc: 0.9255 - val_loss: 0.1616 - val_acc: 0.9638
Epoch 19/40
60000/60000 [==============================] - 6s 99us/step - loss: 0.2924
- acc: 0.9280 - val_loss: 0.1594 - val_acc: 0.9635
Epoch 20/40
60000/60000 [==============================] - 6s 100us/step - loss: 0.281
4 - acc: 0.9311 - val_loss: 0.1583 - val_acc: 0.9642
```

```
Epoch 21/40
60000/60000 [==============================] - 6s 99us/step - loss: 0.2781
- acc: 0.9325 - val_loss: 0.1605 - val_acc: 0.9650
Epoch 22/40
60000/60000 [==============================] - 6s 99us/step - loss: 0.2714
- acc: 0.9347 - val_loss: 0.1472 - val_acc: 0.9663
Epoch 23/40
60000/60000 [==============================] - 6s 100us/step - loss: 0.260
8 - acc: 0.9365 - val_loss: 0.1492 - val_acc: 0.9677
Epoch 24/40
60000/60000 [==============================] - 6s 100us/step - loss: 0.254
4 - acc: 0.9382 - val_loss: 0.1452 - val_acc: 0.9686
Epoch 25/40
60000/60000 [==============================] - 6s 99us/step - loss: 0.2501
- acc: 0.9405 - val_loss: 0.1416 - val_acc: 0.9688
Epoch 26/40
60000/60000 [==============================] - 6s 98us/step - loss: 0.2453
- acc: 0.9404 - val_loss: 0.1449 - val_acc: 0.9682
Epoch 27/40
60000/60000 [==============================] - 6s 97us/step - loss: 0.2405
- acc: 0.9423 - val_loss: 0.1428 - val_acc: 0.9685
Epoch 28/40
60000/60000 [==============================] - 6s 99us/step - loss: 0.2320
- acc: 0.9443 - val_loss: 0.1358 - val_acc: 0.9705
Epoch 29/40
60000/60000 [==============================] - 6s 98us/step - loss: 0.2278
- acc: 0.9451 - val_loss: 0.1319 - val_acc: 0.9698
Epoch 30/40
60000/60000 [==============================] - 6s 97us/step - loss: 0.2251
- acc: 0.9462 - val_loss: 0.1301 - val_acc: 0.9706
Epoch 31/40
60000/60000 [==============================] - 6s 97us/step - loss: 0.2198
- acc: 0.9470 - val_loss: 0.1358 - val_acc: 0.9710
Epoch 32/40
60000/60000 [==============================] - 6s 97us/step - loss: 0.2136
- acc: 0.9489 - val_loss: 0.1295 - val_acc: 0.9713
Epoch 33/40
60000/60000 [==============================] - 6s 97us/step - loss: 0.2114
- acc: 0.9481 - val_loss: 0.1252 - val_acc: 0.9718
Epoch 34/40
60000/60000 [==============================] - 6s 98us/step - loss: 0.2074
- acc: 0.9498 - val_loss: 0.1268 - val_acc: 0.9720
Epoch 35/40
60000/60000 [==============================] - 6s 98us/step - loss: 0.2042
- acc: 0.9505 - val_loss: 0.1261 - val_acc: 0.9722
Epoch 36/40
60000/60000 [==============================] - 6s 98us/step - loss: 0.1955
- acc: 0.9524 - val_loss: 0.1285 - val_acc: 0.9741
Epoch 37/40
60000/60000 [==============================] - 6s 99us/step - loss: 0.1988
- acc: 0.9526 - val_loss: 0.1247 - val_acc: 0.9730
Epoch 38/40
60000/60000 [==============================] - 6s 99us/step - loss: 0.1964
- acc: 0.9519 - val_loss: 0.1229 - val_acc: 0.9731
Epoch 39/40
60000/60000 [==============================] - 6s 99us/step - loss: 0.1906
- acc: 0.9535 - val_loss: 0.1247 - val_acc: 0.9728
Epoch 40/40
60000/60000 [==============================] - 6s 99us/step - loss: 0.1880
- acc: 0.9551 - val_loss: 0.1221 - val_acc: 0.9721
```

In [38]:

```
score = model_final.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, ve
rbose=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 ep
ochs

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

```
Test score: 0.12213036285452544
Test accuracy: 0.9721
```



Tanh activation +adam optimizer+ Batch Normalization + Model Architecture --> [(input)--(hidden layer 1 (512 neurons))--(hidden layer 2(128 neurons))---(hidden layer 3 (256 neurons))---(output layer)]

In [26]:

```
model_final = Sequential()
model_final.add(Dense(512, activation='tanh', input_shape=(input_dim,), kernel_initiali
zer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_final.add(BatchNormalization())

model_final.add(Dense(128, activation='tanh', kernel_initializer=RandomNormal(mean=0.0,
stddev=0.55, seed=None)) )
model_final.add(BatchNormalization())

model_final.add(Dense(256, activation='tanh', kernel_initializer=RandomNormal(mean=0.0,
stddev=0.55, seed=None)) )
model_final.add(BatchNormalization())

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

```
W1003 04:23:29.100877 14324 deprecation_wrapper.py:119] From C:\Users\RASH
U TYAGI\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:13
3: The name tf.placeholder_with_default is deprecated. Please use tf.compa
t.v1.placeholder_with_default instead.
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 512)               401920
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048
_________________________________________________________________
dense_4 (Dense)              (None, 128)               65664
_________________________________________________________________
batch_normalization_2 (Batch (None, 128)               512
_________________________________________________________________
dense_5 (Dense)              (None, 256)               33024
_________________________________________________________________
batch_normalization_3 (Batch (None, 256)               1024
_________________________________________________________________
dense_6 (Dense)              (None, 10)                2570
=================================================================
Total params: 506,762
Trainable params: 504,970
Non-trainable params: 1,792
_________________________________________________________________
```

In [27]:

```
nb_epoch = 30
batch_size = 64
```

In [28]:

```
model_final.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura
cy'])
history = model_final.fit(X_train, Y_train, epochs=nb_epoch, verbose=1, validation_data
=(X_test, Y_test))
```

```
W1003 04:23:33.710057 14324 deprecation.py:323] From C:\Users\RASHU TYAGI
\Anaconda3\lib\site-packages\tensorflow\python\ops\math_grad.py:1250: add_
dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) i
s 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
```

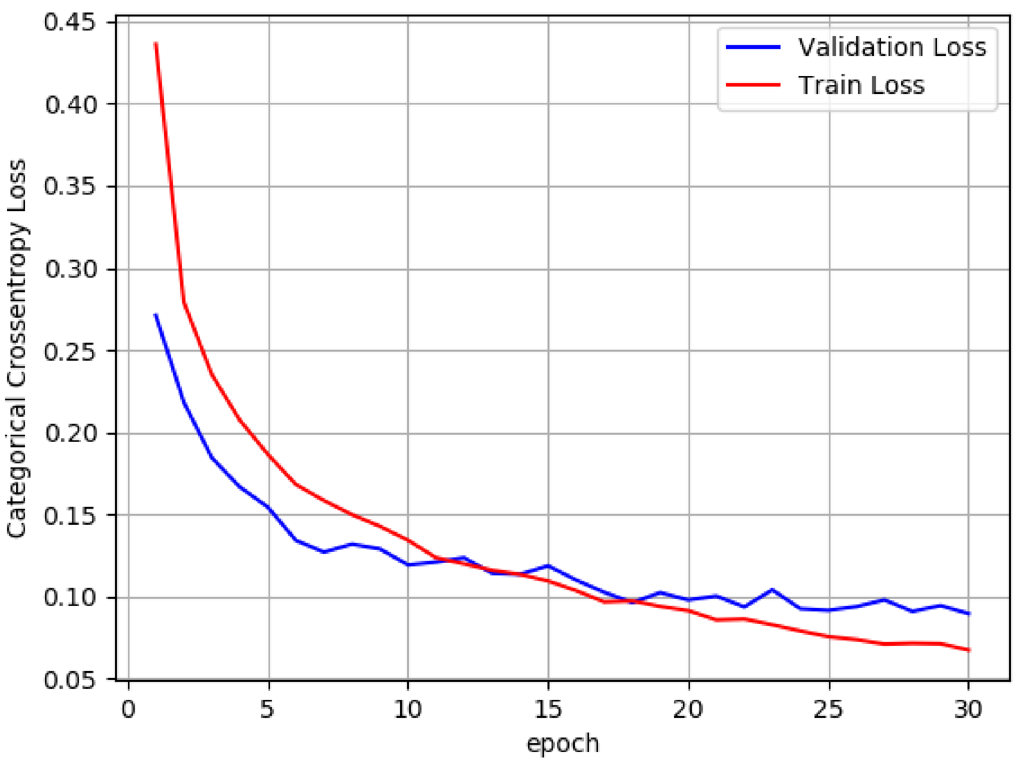
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 13s 216us/step - loss: 0.43
63 - acc: 0.8650 - val_loss: 0.2711 - val_acc: 0.9171
Epoch 2/30
60000/60000 [==============================] - 11s 182us/step - loss: 0.27
92 - acc: 0.9136 - val_loss: 0.2183 - val_acc: 0.9335
Epoch 3/30
60000/60000 [==============================] - 11s 183us/step - loss: 0.23
51 - acc: 0.9277 - val_loss: 0.1845 - val_acc: 0.9446
Epoch 4/30
60000/60000 [==============================] - 11s 181us/step - loss: 0.20
73 - acc: 0.9361 - val_loss: 0.1667 - val_acc: 0.9490
Epoch 5/30
60000/60000 [==============================] - 11s 181us/step - loss: 0.18
65 - acc: 0.9439 - val_loss: 0.1544 - val_acc: 0.9537
Epoch 6/30
60000/60000 [==============================] - 11s 181us/step - loss: 0.16
82 - acc: 0.9478 - val_loss: 0.1342 - val_acc: 0.9585
Epoch 7/30
60000/60000 [==============================] - 11s 182us/step - loss: 0.15
84 - acc: 0.9523 - val_loss: 0.1270 - val_acc: 0.9631
Epoch 8/30
60000/60000 [==============================] - 11s 183us/step - loss: 0.14
99 - acc: 0.9532 - val_loss: 0.1319 - val_acc: 0.9611
Epoch 9/30
60000/60000 [==============================] - 11s 181us/step - loss: 0.14
27 - acc: 0.9554 - val_loss: 0.1290 - val_acc: 0.9593
Epoch 10/30
60000/60000 [==============================] - 11s 183us/step - loss: 0.13
42 - acc: 0.9582 - val_loss: 0.1193 - val_acc: 0.9643
Epoch 11/30
60000/60000 [==============================] - 11s 182us/step - loss: 0.12
36 - acc: 0.9606 - val_loss: 0.1210 - val_acc: 0.9629
Epoch 12/30
60000/60000 [==============================] - 11s 181us/step - loss: 0.12
00 - acc: 0.9621 - val_loss: 0.1236 - val_acc: 0.9624
Epoch 13/30
60000/60000 [==============================] - 11s 180us/step - loss: 0.11
60 - acc: 0.9641 - val_loss: 0.1142 - val_acc: 0.9643
Epoch 14/30
60000/60000 [==============================] - 11s 183us/step - loss: 0.11
35 - acc: 0.9649 - val_loss: 0.1136 - val_acc: 0.9672
Epoch 15/30
60000/60000 [==============================] - 11s 180us/step - loss: 0.10
95 - acc: 0.9655 - val_loss: 0.1187 - val_acc: 0.9642
Epoch 16/30
60000/60000 [==============================] - 11s 181us/step - loss: 0.10
36 - acc: 0.9672 - val_loss: 0.1100 - val_acc: 0.9653
Epoch 17/30
60000/60000 [==============================] - 11s 182us/step - loss: 0.09
68 - acc: 0.9700 - val_loss: 0.1026 - val_acc: 0.9673
Epoch 18/30
60000/60000 [==============================] - 11s 183us/step - loss: 0.09
74 - acc: 0.9692 - val_loss: 0.0963 - val_acc: 0.9703
Epoch 19/30
60000/60000 [==============================] - 11s 183us/step - loss: 0.09
40 - acc: 0.9707 - val_loss: 0.1024 - val_acc: 0.9702
Epoch 20/30
60000/60000 [==============================] - 11s 183us/step - loss: 0.09
14 - acc: 0.9698 - val_loss: 0.0979 - val_acc: 0.9720
```

```
Epoch 21/30
60000/60000 [==============================] - 11s 179us/step - loss: 0.08
58 - acc: 0.9730 - val_loss: 0.1003 - val_acc: 0.9719
Epoch 22/30
60000/60000 [==============================] - 11s 182us/step - loss: 0.08
63 - acc: 0.9726 - val_loss: 0.0937 - val_acc: 0.9727
Epoch 23/30
60000/60000 [==============================] - 11s 182us/step - loss: 0.08
28 - acc: 0.9742 - val_loss: 0.1041 - val_acc: 0.9678
Epoch 24/30
60000/60000 [==============================] - 11s 179us/step - loss: 0.07
90 - acc: 0.9749 - val_loss: 0.0926 - val_acc: 0.9730
Epoch 25/30
60000/60000 [==============================] - 11s 180us/step - loss: 0.07
56 - acc: 0.9764 - val_loss: 0.0916 - val_acc: 0.9719
Epoch 26/30
60000/60000 [==============================] - 11s 181us/step - loss: 0.07
38 - acc: 0.9760 - val_loss: 0.0939 - val_acc: 0.9721
Epoch 27/30
60000/60000 [==============================] - 11s 183us/step - loss: 0.07
11 - acc: 0.9779 - val_loss: 0.0980 - val_acc: 0.9716
Epoch 28/30
60000/60000 [==============================] - 11s 182us/step - loss: 0.07
16 - acc: 0.9771 - val_loss: 0.0909 - val_acc: 0.9732
Epoch 29/30
60000/60000 [==============================] - 11s 181us/step - loss: 0.07
13 - acc: 0.9773 - val_loss: 0.0945 - val_acc: 0.9720
Epoch 30/30
60000/60000 [==============================] - 11s 180us/step - loss: 0.06
75 - acc: 0.9786 - val_loss: 0.0896 - val_acc: 0.9736
```

In [29]:

```
score = model_final.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, ve
rbose=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 ep
ochs

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

```
Test score: 0.08960605073939078
Test accuracy: 0.9736
```



Next architecture

glorot_uniform initialization + sgd optimizer + relu activation function + Dropout(0.33) + no batch normalization + Model Architecture --> [(input)--(hidden layer 1 (512 neurons))--(hidden layer 2 (128 neurons))---(hidden layer 3 (256 neurons))--(hidden layer 4 (1024 neurons))----(hidden layer 5 (2048 neurons))---(output layer)]

In [36]:

```
import keras.utils

model_final = Sequential()

model_final.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initiali
zer = keras.initializers.glorot_normal(seed=None)))
model_final.add(Dropout(0.33))

model_final.add(Dense(128, activation='relu', kernel_initializer = keras.initializers.g
lorot_normal(seed=None)))
model_final.add(Dropout(0.33))

model_final.add(Dense(256, activation='relu', kernel_initializer = keras.initializers.g
lorot_normal(seed=None)))
model_final.add(Dropout(0.33))

model_final.add(Dense(1024, activation='relu', kernel_initializer = keras.initializers.
glorot_normal(seed=None)))
model_final.add(Dropout(0.33))

model_final.add(Dense(2048, activation='relu', kernel_initializer = keras.initializers.
glorot_normal(seed=None)))
model_final.add(Dropout(0.33))

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


model_final.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_8 (Dense)              (None, 512)               401920    
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_9 (Dense)              (None, 128)               65664     
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_10 (Dense)             (None, 256)               33024     
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_11 (Dense)             (None, 1024)              263168    
_________________________________________________________________
dropout_5 (Dropout)          (None, 1024)              0         
_________________________________________________________________
dense_12 (Dense)             (None, 2048)              2099200   
_________________________________________________________________
dropout_6 (Dropout)          (None, 2048)              0         
_________________________________________________________________
dense_13 (Dense)             (None, 10)                20490     
=================================================================
Total params: 2,883,466
Trainable params: 2,883,466
Non-trainable params: 0
_________________________________________________________________
```

In [37]:

```
nb_epoch = 60
```

In [38]:

```
model_final.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accurac
y'])

history = model_final.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ver
bose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/60
60000/60000 [==============================] - 4s 68us/step - loss: 1.8581
- acc: 0.3439 - val_loss: 0.7089 - val_acc: 0.7973
Epoch 2/60
60000/60000 [==============================] - 3s 58us/step - loss: 0.7309
- acc: 0.7606 - val_loss: 0.3608 - val_acc: 0.8915
Epoch 3/60
60000/60000 [==============================] - 4s 60us/step - loss: 0.4883
- acc: 0.8503 - val_loss: 0.2734 - val_acc: 0.9179
Epoch 4/60
60000/60000 [==============================] - 3s 58us/step - loss: 0.3872
- acc: 0.8858 - val_loss: 0.2224 - val_acc: 0.9335
Epoch 5/60
60000/60000 [==============================] - 3s 58us/step - loss: 0.3271
- acc: 0.9028 - val_loss: 0.1916 - val_acc: 0.9418
Epoch 6/60
60000/60000 [==============================] - 3s 58us/step - loss: 0.2824
- acc: 0.9163 - val_loss: 0.1719 - val_acc: 0.9490
Epoch 7/60
60000/60000 [==============================] - 3s 58us/step - loss: 0.2523
- acc: 0.9250 - val_loss: 0.1520 - val_acc: 0.9535
Epoch 8/60
60000/60000 [==============================] - 3s 58us/step - loss: 0.2295
- acc: 0.9324 - val_loss: 0.1409 - val_acc: 0.9566
Epoch 9/60
60000/60000 [==============================] - 3s 58us/step - loss: 0.2080
- acc: 0.9387 - val_loss: 0.1292 - val_acc: 0.9607
Epoch 10/60
60000/60000 [==============================] - 4s 60us/step - loss: 0.1938
- acc: 0.9428 - val_loss: 0.1225 - val_acc: 0.9638
Epoch 11/60
60000/60000 [==============================] - 3s 57us/step - loss: 0.1786
- acc: 0.9467 - val_loss: 0.1157 - val_acc: 0.9645
Epoch 12/60
60000/60000 [==============================] - 3s 57us/step - loss: 0.1687
- acc: 0.9501 - val_loss: 0.1106 - val_acc: 0.9675
Epoch 13/60
60000/60000 [==============================] - 3s 57us/step - loss: 0.1554
- acc: 0.9542 - val_loss: 0.1050 - val_acc: 0.9681
Epoch 14/60
60000/60000 [==============================] - 3s 57us/step - loss: 0.1510
- acc: 0.9558 - val_loss: 0.0996 - val_acc: 0.9708
Epoch 15/60
60000/60000 [==============================] - 3s 58us/step - loss: 0.1392
- acc: 0.9590 - val_loss: 0.0981 - val_acc: 0.9707
Epoch 16/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.1329
- acc: 0.9613 - val_loss: 0.0967 - val_acc: 0.9708
Epoch 17/60
60000/60000 [==============================] - 3s 57us/step - loss: 0.1241
- acc: 0.9637 - val_loss: 0.0930 - val_acc: 0.9725
Epoch 18/60
60000/60000 [==============================] - 3s 57us/step - loss: 0.1197
- acc: 0.9646 - val_loss: 0.0902 - val_acc: 0.9734
Epoch 19/60
60000/60000 [==============================] - 3s 57us/step - loss: 0.1135
- acc: 0.9668 - val_loss: 0.0868 - val_acc: 0.9748
Epoch 20/60
60000/60000 [==============================] - 4s 60us/step - loss: 0.1095
- acc: 0.9673 - val_loss: 0.0877 - val_acc: 0.9745
```

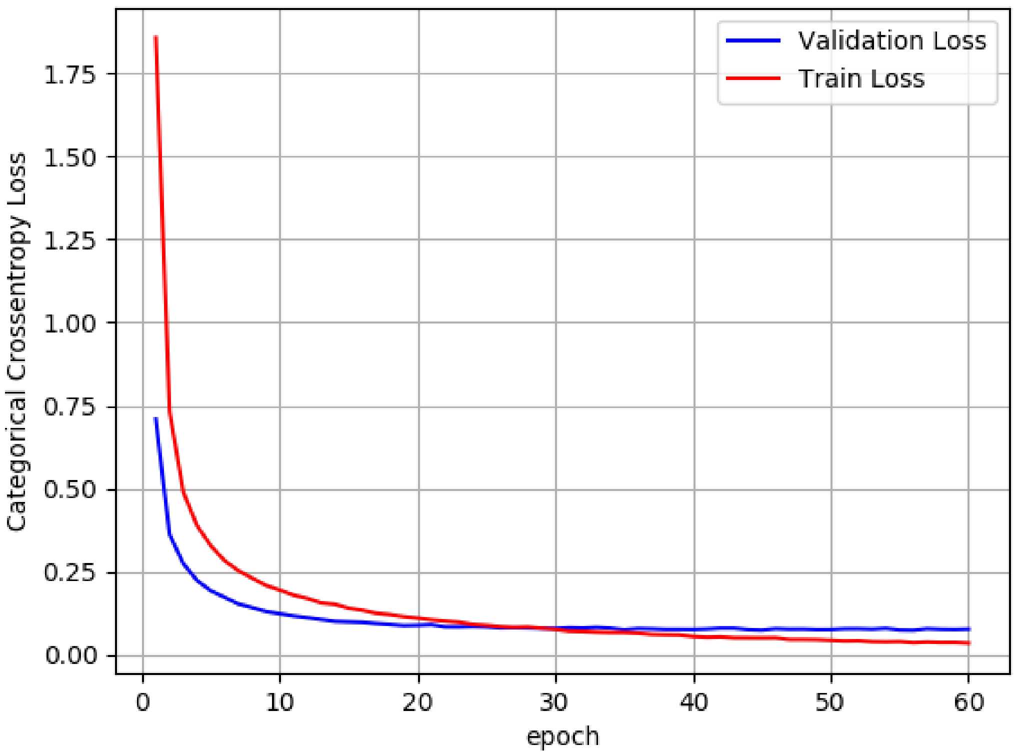
```
Epoch 21/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.1042
- acc: 0.9685 - val_loss: 0.0897 - val_acc: 0.9737
Epoch 22/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.1006
- acc: 0.9704 - val_loss: 0.0831 - val_acc: 0.9761
Epoch 23/60
60000/60000 [==============================] - 4s 58us/step - loss: 0.0971
- acc: 0.9711 - val_loss: 0.0829 - val_acc: 0.9767
Epoch 24/60
60000/60000 [==============================] - 4s 58us/step - loss: 0.0904
- acc: 0.9729 - val_loss: 0.0842 - val_acc: 0.9760
Epoch 25/60
60000/60000 [==============================] - 4s 60us/step - loss: 0.0878
- acc: 0.9730 - val_loss: 0.0828 - val_acc: 0.9770
Epoch 26/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0839
- acc: 0.9748 - val_loss: 0.0795 - val_acc: 0.9779
Epoch 27/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0817
- acc: 0.9761 - val_loss: 0.0798 - val_acc: 0.9780
Epoch 28/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0829
- acc: 0.9749 - val_loss: 0.0794 - val_acc: 0.9783
Epoch 29/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0789
- acc: 0.9766 - val_loss: 0.0783 - val_acc: 0.9796
Epoch 30/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0754
- acc: 0.9772 - val_loss: 0.0780 - val_acc: 0.9788
Epoch 31/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0699
- acc: 0.9793 - val_loss: 0.0805 - val_acc: 0.9782
Epoch 32/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0690
- acc: 0.9792 - val_loss: 0.0795 - val_acc: 0.9785
Epoch 33/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0671
- acc: 0.9801 - val_loss: 0.0815 - val_acc: 0.9776
Epoch 34/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0660
- acc: 0.9796 - val_loss: 0.0791 - val_acc: 0.9802
Epoch 35/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0659
- acc: 0.9800 - val_loss: 0.0741 - val_acc: 0.9800
Epoch 36/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0640
- acc: 0.9808 - val_loss: 0.0777 - val_acc: 0.9798
Epoch 37/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0603
- acc: 0.9818 - val_loss: 0.0768 - val_acc: 0.9801
Epoch 38/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0594
- acc: 0.9817 - val_loss: 0.0752 - val_acc: 0.9812
Epoch 39/60
60000/60000 [==============================] - 4s 60us/step - loss: 0.0591
- acc: 0.9822 - val_loss: 0.0752 - val_acc: 0.9801
Epoch 40/60
60000/60000 [==============================] - 4s 63us/step - loss: 0.0543
- acc: 0.9835 - val_loss: 0.0750 - val_acc: 0.9806
Epoch 41/60
```

```
60000/60000 [==============================] - 4s 59us/step - loss: 0.0518
- acc: 0.9849 - val_loss: 0.0762 - val_acc: 0.9808
Epoch 42/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0529
- acc: 0.9832 - val_loss: 0.0789 - val_acc: 0.9795
Epoch 43/60
60000/60000 [==============================] - 4s 60us/step - loss: 0.0500
- acc: 0.9843 - val_loss: 0.0788 - val_acc: 0.9804
Epoch 44/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0498
- acc: 0.9844 - val_loss: 0.0746 - val_acc: 0.9818
Epoch 45/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0497
- acc: 0.9844 - val_loss: 0.0731 - val_acc: 0.9817
Epoch 46/60
60000/60000 [==============================] - 4s 58us/step - loss: 0.0499
- acc: 0.9841 - val_loss: 0.0773 - val_acc: 0.9808
Epoch 47/60
60000/60000 [==============================] - 4s 58us/step - loss: 0.0450
- acc: 0.9860 - val_loss: 0.0758 - val_acc: 0.9816
Epoch 48/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0450
- acc: 0.9856 - val_loss: 0.0760 - val_acc: 0.9811
Epoch 49/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0445
- acc: 0.9859 - val_loss: 0.0747 - val_acc: 0.9824
Epoch 50/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0427
- acc: 0.9868 - val_loss: 0.0748 - val_acc: 0.9815
Epoch 51/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0409
- acc: 0.9875 - val_loss: 0.0769 - val_acc: 0.9826
Epoch 52/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0415
- acc: 0.9870 - val_loss: 0.0771 - val_acc: 0.9823
Epoch 53/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0386
- acc: 0.9881 - val_loss: 0.0758 - val_acc: 0.9825
Epoch 54/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0379
- acc: 0.9881 - val_loss: 0.0783 - val_acc: 0.9812
Epoch 55/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0387
- acc: 0.9878 - val_loss: 0.0735 - val_acc: 0.9828
Epoch 56/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0353
- acc: 0.9890 - val_loss: 0.0728 - val_acc: 0.9826
Epoch 57/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0373
- acc: 0.9884 - val_loss: 0.0771 - val_acc: 0.9826
Epoch 58/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0358
- acc: 0.9891 - val_loss: 0.0754 - val_acc: 0.9828
Epoch 59/60
60000/60000 [==============================] - 4s 59us/step - loss: 0.0361
- acc: 0.9890 - val_loss: 0.0749 - val_acc: 0.9832
Epoch 60/60
60000/60000 [==============================] - 3s 58us/step - loss: 0.0341
- acc: 0.9892 - val_loss: 0.0759 - val_acc: 0.9832
```

In [39]:

```
score = model_final.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, ve
rbose=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 ep
ochs

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

```
Test score: 0.07590167486594146
Test accuracy: 0.9832
```



Next Architecture

he_normal initialization + adam optimizer + softmax activation function + Dropout(0.66) + batch normalization(to only 2 layers) + Model Architecture --> [(input)--(hidden layer 1 (512 neurons))--(hidden layer 2 (128 neurons))---(hidden layer 3 (256 neurons))--(hidden layer 4 (1024 neurons))----(hidden layer 5 (2048 neurons))---(output layer)]

In [40]:

```
# refer this for he_normal initialization --> https://keras.io/activations/
import keras.utils

model_final = Sequential()

model_final.add(Dense(512, activation='softmax', input_shape=(input_dim,), kernel_initi
alizer = keras.initializers.he_normal(seed=None)))
model_final.add(Dropout(0.66))


model_final.add(Dense(128, activation='softmax', kernel_initializer = keras.initializer
s.he_normal(seed=None)))
model_final.add(Dropout(0.66))
model_final.add(BatchNormalization())

model_final.add(Dense(256, activation='softmax', kernel_initializer = keras.initializer
s.he_normal(seed=None)))
model_final.add(Dropout(0.66))


model_final.add(Dense(1024, activation='softmax', kernel_initializer = keras.initialize
rs.he_normal(seed=None)))
model_final.add(Dropout(0.66))


model_final.add(Dense(2048, activation='softmax', kernel_initializer = keras.initialize
rs.he_normal(seed=None)))
model_final.add(Dropout(0.66))
model_final.add(BatchNormalization())

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


model_final.summary()
```

```
W1003 06:40:12.831539 14324 nn_ops.py:4224] Large dropout rate: 0.66 (>0.
5). In TensorFlow 2.x, dropout() uses dropout rate instead of keep_prob. P
lease ensure that this is intended.
W1003 06:40:12.853988 14324 nn_ops.py:4224] Large dropout rate: 0.66 (>0.
5). In TensorFlow 2.x, dropout() uses dropout rate instead of keep_prob. P
lease ensure that this is intended.
W1003 06:40:12.924259 14324 nn_ops.py:4224] Large dropout rate: 0.66 (>0.
5). In TensorFlow 2.x, dropout() uses dropout rate instead of keep_prob. P
lease ensure that this is intended.
W1003 06:40:12.944783 14324 nn_ops.py:4224] Large dropout rate: 0.66 (>0.
5). In TensorFlow 2.x, dropout() uses dropout rate instead of keep_prob. P
lease ensure that this is intended.
W1003 06:40:12.964304 14324 nn_ops.py:4224] Large dropout rate: 0.66 (>0.
5). In TensorFlow 2.x, dropout() uses dropout rate instead of keep_prob. P
lease ensure that this is intended.
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_14 (Dense)             (None, 512)               401920    
_________________________________________________________________
dropout_7 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_15 (Dense)             (None, 128)               65664     
_________________________________________________________________
dropout_8 (Dropout)          (None, 128)               0         
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512       
_________________________________________________________________
dense_16 (Dense)             (None, 256)               33024     
_________________________________________________________________
dropout_9 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_17 (Dense)             (None, 1024)              263168    
_________________________________________________________________
dropout_10 (Dropout)         (None, 1024)              0         
_________________________________________________________________
dense_18 (Dense)             (None, 2048)              2099200   
_________________________________________________________________
dropout_11 (Dropout)         (None, 2048)              0         
_________________________________________________________________
batch_normalization_5 (Batch (None, 2048)              8192      
_________________________________________________________________
dense_19 (Dense)             (None, 10)                20490     
=================================================================
Total params: 2,892,170
Trainable params: 2,887,818
Non-trainable params: 4,352
_________________________________________________________________
```

In [42]:

```
nb_epoch = 50
batch_size = 256
```

In [43]:

```
model_final.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura
cy'])

history = model_final.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ver
bose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 3s 56us/step - loss: 2.3027
- acc: 0.1090 - val_loss: 2.3015 - val_acc: 0.1028
Epoch 2/50
60000/60000 [==============================] - 2s 38us/step - loss: 2.3031
- acc: 0.1086 - val_loss: 2.3016 - val_acc: 0.1135
Epoch 3/50
60000/60000 [==============================] - 2s 38us/step - loss: 2.3031
- acc: 0.1083 - val_loss: 2.3014 - val_acc: 0.1135
Epoch 4/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3028
- acc: 0.1077 - val_loss: 2.3014 - val_acc: 0.1135
Epoch 5/50
60000/60000 [==============================] - 2s 38us/step - loss: 2.3027
- acc: 0.1073 - val_loss: 2.3025 - val_acc: 0.1135
Epoch 6/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3028
- acc: 0.1098 - val_loss: 2.3018 - val_acc: 0.1028
Epoch 7/50
60000/60000 [==============================] - 2s 38us/step - loss: 2.3030
- acc: 0.1076 - val_loss: 2.3031 - val_acc: 0.1135
Epoch 8/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3031
- acc: 0.1094 - val_loss: 2.3036 - val_acc: 0.1135
Epoch 9/50
60000/60000 [==============================] - 2s 38us/step - loss: 2.3031
- acc: 0.1077 - val_loss: 2.3021 - val_acc: 0.1135
Epoch 10/50
60000/60000 [==============================] - 2s 38us/step - loss: 2.3031
- acc: 0.1077 - val_loss: 2.3017 - val_acc: 0.1135
Epoch 11/50
60000/60000 [==============================] - 2s 38us/step - loss: 2.3031
- acc: 0.1080 - val_loss: 2.3014 - val_acc: 0.1135
Epoch 12/50
60000/60000 [==============================] - 2s 39us/step - loss: 2.3031
- acc: 0.1073 - val_loss: 2.3030 - val_acc: 0.1028
Epoch 13/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3031
- acc: 0.1081 - val_loss: 2.3013 - val_acc: 0.1135
Epoch 14/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3034
- acc: 0.1084 - val_loss: 2.3015 - val_acc: 0.1135
Epoch 15/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3032
- acc: 0.1073 - val_loss: 2.3024 - val_acc: 0.1135
Epoch 16/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3036
- acc: 0.1070 - val_loss: 2.3023 - val_acc: 0.1135
Epoch 17/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3032
- acc: 0.1077 - val_loss: 2.3015 - val_acc: 0.1009
Epoch 18/50
60000/60000 [==============================] - 2s 38us/step - loss: 2.3033
- acc: 0.1062 - val_loss: 2.3027 - val_acc: 0.1135
Epoch 19/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3035
- acc: 0.1063 - val_loss: 2.3025 - val_acc: 0.1135
Epoch 20/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3033
- acc: 0.1075 - val_loss: 2.3014 - val_acc: 0.1135
```

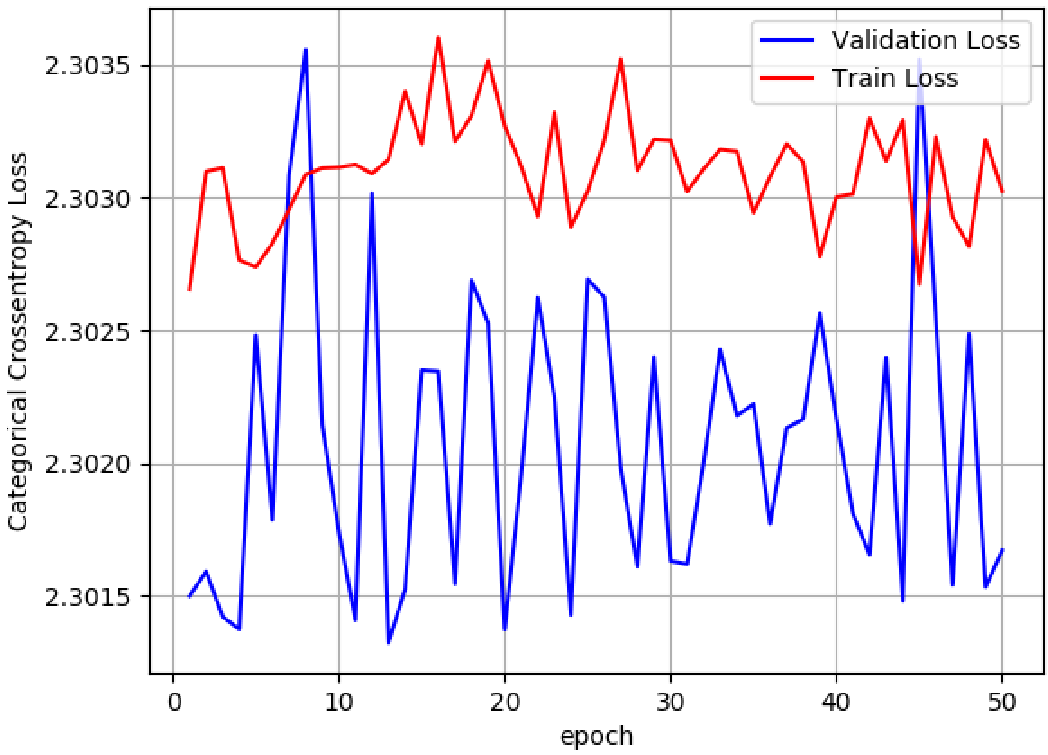
```
Epoch 21/50
60000/60000 [==============================] - 2s 38us/step - loss: 2.3031
- acc: 0.1067 - val_loss: 2.3020 - val_acc: 0.1135
Epoch 22/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3029
- acc: 0.1083 - val_loss: 2.3026 - val_acc: 0.1135
Epoch 23/50
60000/60000 [==============================] - 2s 38us/step - loss: 2.3033
- acc: 0.1078 - val_loss: 2.3022 - val_acc: 0.1010
Epoch 24/50
60000/60000 [==============================] - 2s 38us/step - loss: 2.3029
- acc: 0.1087 - val_loss: 2.3014 - val_acc: 0.1135
Epoch 25/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3030
- acc: 0.1084 - val_loss: 2.3027 - val_acc: 0.1135
Epoch 26/50
60000/60000 [==============================] - 2s 38us/step - loss: 2.3032
- acc: 0.1071 - val_loss: 2.3026 - val_acc: 0.1032
Epoch 27/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3035
- acc: 0.1079 - val_loss: 2.3020 - val_acc: 0.1010
Epoch 28/50
60000/60000 [==============================] - 2s 38us/step - loss: 2.3031
- acc: 0.1080 - val_loss: 2.3016 - val_acc: 0.1135
Epoch 29/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3032
- acc: 0.1073 - val_loss: 2.3024 - val_acc: 0.1135
Epoch 30/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3032
- acc: 0.1086 - val_loss: 2.3016 - val_acc: 0.1135
Epoch 31/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3030
- acc: 0.1077 - val_loss: 2.3016 - val_acc: 0.1135
Epoch 32/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3031
- acc: 0.1077 - val_loss: 2.3020 - val_acc: 0.1135
Epoch 33/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3032
- acc: 0.1087 - val_loss: 2.3024 - val_acc: 0.1135
Epoch 34/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3032
- acc: 0.1076 - val_loss: 2.3022 - val_acc: 0.1135
Epoch 35/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3029
- acc: 0.1081 - val_loss: 2.3022 - val_acc: 0.1135
Epoch 36/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3031
- acc: 0.1089 - val_loss: 2.3018 - val_acc: 0.1135
Epoch 37/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3032
- acc: 0.1080 - val_loss: 2.3021 - val_acc: 0.1028
Epoch 38/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3031
- acc: 0.1070 - val_loss: 2.3022 - val_acc: 0.1028
Epoch 39/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3028
- acc: 0.1066 - val_loss: 2.3026 - val_acc: 0.1135
Epoch 40/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3030
- acc: 0.1068 - val_loss: 2.3022 - val_acc: 0.1135
Epoch 41/50
```

```
60000/60000 [==============================] - 2s 37us/step - loss: 2.3030
- acc: 0.1075 - val_loss: 2.3018 - val_acc: 0.1135
Epoch 42/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3033
- acc: 0.1065 - val_loss: 2.3017 - val_acc: 0.1135
Epoch 43/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3031
- acc: 0.1091 - val_loss: 2.3024 - val_acc: 0.1028
Epoch 44/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3033
- acc: 0.1067 - val_loss: 2.3015 - val_acc: 0.1135
Epoch 45/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3027
- acc: 0.1081 - val_loss: 2.3035 - val_acc: 0.1135
Epoch 46/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3032
- acc: 0.1068 - val_loss: 2.3025 - val_acc: 0.1135
Epoch 47/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3029
- acc: 0.1072 - val_loss: 2.3015 - val_acc: 0.1135
Epoch 48/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3028
- acc: 0.1091 - val_loss: 2.3025 - val_acc: 0.1028
Epoch 49/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3032
- acc: 0.1079 - val_loss: 2.3015 - val_acc: 0.1135
Epoch 50/50
60000/60000 [==============================] - 2s 37us/step - loss: 2.3030
- acc: 0.1079 - val_loss: 2.3017 - val_acc: 0.1010
```

In [44]:

```
score = model_final.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, ve
rbose=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 ep
ochs

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

```
Test score: 2.3016698822021486
Test accuracy: 0.101
```



We received a very poor accuracy over there let us try the same code with relu activation

In [46]:

```
# refer this for he_normal initialization --> https://keras.io/activations/
import keras.utils

model_final = Sequential()

model_final.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initiali
zer = keras.initializers.he_normal(seed=None)))
model_final.add(Dropout(0.66))


model_final.add(Dense(128, activation='relu', kernel_initializer = keras.initializers.h
e_normal(seed=None)))
model_final.add(Dropout(0.66))
model_final.add(BatchNormalization())

model_final.add(Dense(256, activation='relu', kernel_initializer = keras.initializers.h
e_normal(seed=None)))
model_final.add(Dropout(0.66))


model_final.add(Dense(1024, activation='relu', kernel_initializer = keras.initializers.
he_normal(seed=None)))
model_final.add(Dropout(0.66))


model_final.add(Dense(2048, activation='relu', kernel_initializer = keras.initializers.
he_normal(seed=None)))
model_final.add(Dropout(0.66))
model_final.add(BatchNormalization())

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


model_final.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_20 (Dense)             (None, 512)               401920    
_________________________________________________________________
dropout_12 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_21 (Dense)             (None, 128)               65664     
_________________________________________________________________
dropout_13 (Dropout)         (None, 128)               0         
_________________________________________________________________
batch_normalization_6 (Batch (None, 128)               512       
_________________________________________________________________
dense_22 (Dense)             (None, 256)               33024     
_________________________________________________________________
dropout_14 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_23 (Dense)             (None, 1024)              263168    
_________________________________________________________________
dropout_15 (Dropout)         (None, 1024)              0         
_________________________________________________________________
dense_24 (Dense)             (None, 2048)              2099200   
_________________________________________________________________
dropout_16 (Dropout)         (None, 2048)              0         
_________________________________________________________________
batch_normalization_7 (Batch (None, 2048)              8192      
_________________________________________________________________
dense_25 (Dense)             (None, 10)                20490     
=================================================================
Total params: 2,892,170
Trainable params: 2,887,818
Non-trainable params: 4,352
_________________________________________________________________
```

In [47]:

```
nb_epoch = 50
batch_size = 256
```

In [48]:

```
model_final.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura
cy'])

history = model_final.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ver
bose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 3s 55us/step - loss: 1.9138
- acc: 0.3571 - val_loss: 1.3399 - val_acc: 0.4876
Epoch 2/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.9259
- acc: 0.6837 - val_loss: 0.5729 - val_acc: 0.7837
Epoch 3/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.6319
- acc: 0.8045 - val_loss: 0.4046 - val_acc: 0.8502
Epoch 4/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.5001
- acc: 0.8528 - val_loss: 0.2862 - val_acc: 0.9020
Epoch 5/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.4270
- acc: 0.8785 - val_loss: 0.2521 - val_acc: 0.9145
Epoch 6/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.3747
- acc: 0.8951 - val_loss: 0.1859 - val_acc: 0.9466
Epoch 7/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.3425
- acc: 0.9070 - val_loss: 0.1850 - val_acc: 0.9443
Epoch 8/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.3117
- acc: 0.9154 - val_loss: 0.1650 - val_acc: 0.9538
Epoch 9/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.2962
- acc: 0.9197 - val_loss: 0.1580 - val_acc: 0.9577
Epoch 10/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.2744
- acc: 0.9270 - val_loss: 0.1428 - val_acc: 0.9624
Epoch 11/50
60000/60000 [==============================] - 2s 37us/step - loss: 0.2594
- acc: 0.9315 - val_loss: 0.1284 - val_acc: 0.9661
Epoch 12/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.2456
- acc: 0.9355 - val_loss: 0.1291 - val_acc: 0.9655
Epoch 13/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.2358
- acc: 0.9380 - val_loss: 0.1303 - val_acc: 0.9679
Epoch 14/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.2246
- acc: 0.9410 - val_loss: 0.1205 - val_acc: 0.9693
Epoch 15/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.2226
- acc: 0.9420 - val_loss: 0.1140 - val_acc: 0.9704
Epoch 16/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.2121
- acc: 0.9449 - val_loss: 0.1218 - val_acc: 0.9692
Epoch 17/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.2031
- acc: 0.9496 - val_loss: 0.1091 - val_acc: 0.9726
Epoch 18/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.2023
- acc: 0.9479 - val_loss: 0.1066 - val_acc: 0.9731
Epoch 19/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.1920
- acc: 0.9505 - val_loss: 0.1035 - val_acc: 0.9730
Epoch 20/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.1936
- acc: 0.9506 - val_loss: 0.1092 - val_acc: 0.9728
```

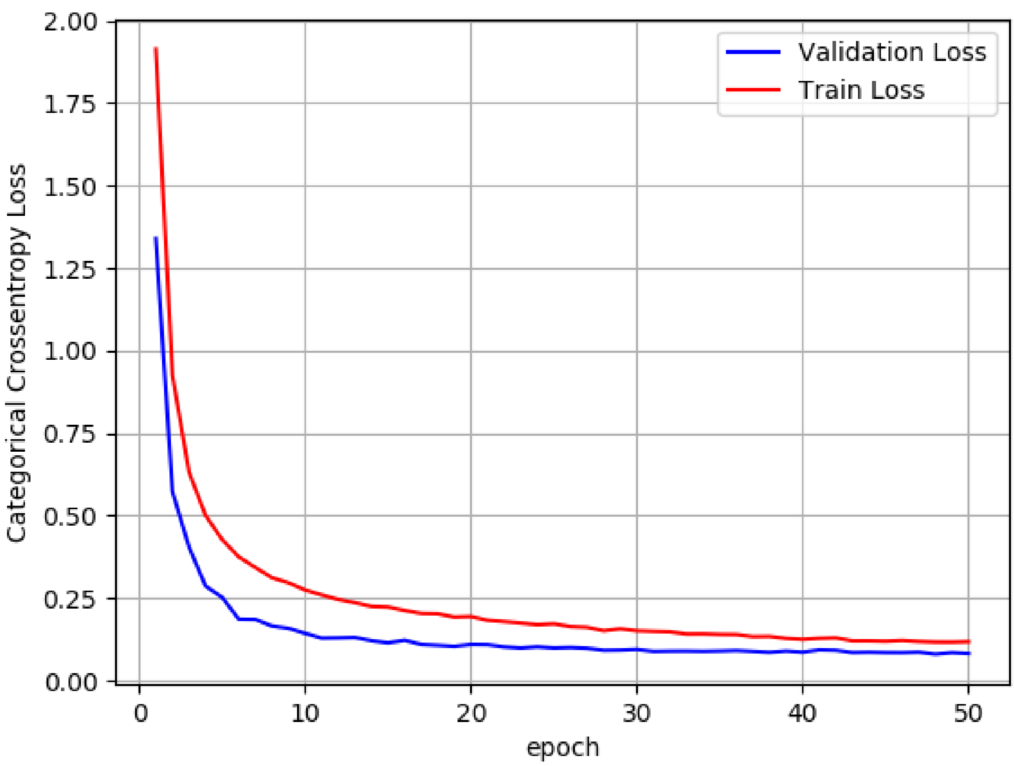
```
Epoch 21/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.1823
- acc: 0.9532 - val_loss: 0.1085 - val_acc: 0.9733
Epoch 22/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.1784
- acc: 0.9547 - val_loss: 0.1020 - val_acc: 0.9730
Epoch 23/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.1742
- acc: 0.9534 - val_loss: 0.0984 - val_acc: 0.9756
Epoch 24/50
60000/60000 [==============================] - 2s 38us/step - loss: 0.1691
- acc: 0.9572 - val_loss: 0.1022 - val_acc: 0.9739
Epoch 25/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.1717
- acc: 0.9562 - val_loss: 0.0985 - val_acc: 0.9740
Epoch 26/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.1632
- acc: 0.9579 - val_loss: 0.1000 - val_acc: 0.9758
Epoch 27/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.1612
- acc: 0.9577 - val_loss: 0.0976 - val_acc: 0.9758
Epoch 28/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.1507
- acc: 0.9603 - val_loss: 0.0916 - val_acc: 0.9773
Epoch 29/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.1564
- acc: 0.9597 - val_loss: 0.0921 - val_acc: 0.9776
Epoch 30/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.1506
- acc: 0.9613 - val_loss: 0.0941 - val_acc: 0.9764
Epoch 31/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.1488
- acc: 0.9619 - val_loss: 0.0884 - val_acc: 0.9792
Epoch 32/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.1476
- acc: 0.9614 - val_loss: 0.0891 - val_acc: 0.9775
Epoch 33/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.1408
- acc: 0.9639 - val_loss: 0.0893 - val_acc: 0.9778
Epoch 34/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.1410
- acc: 0.9637 - val_loss: 0.0886 - val_acc: 0.9779
Epoch 35/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.1394
- acc: 0.9649 - val_loss: 0.0895 - val_acc: 0.9778
Epoch 36/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.1389
- acc: 0.9648 - val_loss: 0.0909 - val_acc: 0.9772
Epoch 37/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.1329
- acc: 0.9658 - val_loss: 0.0883 - val_acc: 0.9782
Epoch 38/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.1335
- acc: 0.9660 - val_loss: 0.0856 - val_acc: 0.9784
Epoch 39/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.1273
- acc: 0.9676 - val_loss: 0.0894 - val_acc: 0.9790
Epoch 40/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.1244
- acc: 0.9679 - val_loss: 0.0860 - val_acc: 0.9792
Epoch 41/50
```

```
60000/60000 [==============================] - 2s 35us/step - loss: 0.1275
- acc: 0.9684 - val_loss: 0.0930 - val_acc: 0.9778
Epoch 42/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.1289
- acc: 0.9673 - val_loss: 0.0917 - val_acc: 0.9783
Epoch 43/50
60000/60000 [==============================] - 2s 35us/step - loss: 0.1200
- acc: 0.9695 - val_loss: 0.0849 - val_acc: 0.9805
Epoch 44/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.1201
- acc: 0.9697 - val_loss: 0.0858 - val_acc: 0.9791
Epoch 45/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.1190
- acc: 0.9702 - val_loss: 0.0848 - val_acc: 0.9791
Epoch 46/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.1211
- acc: 0.9691 - val_loss: 0.0846 - val_acc: 0.9795
Epoch 47/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.1182
- acc: 0.9695 - val_loss: 0.0860 - val_acc: 0.9801
Epoch 48/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.1164
- acc: 0.9708 - val_loss: 0.0798 - val_acc: 0.9802
Epoch 49/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.1163
- acc: 0.9701 - val_loss: 0.0843 - val_acc: 0.9795
Epoch 50/50
60000/60000 [==============================] - 2s 36us/step - loss: 0.1175
- acc: 0.9702 - val_loss: 0.0817 - val_acc: 0.9802
```

In [49]:

```
score = model_final.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, ve
rbose=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 ep
ochs

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

```
Test score: 0.08166552726062946
Test accuracy: 0.9802
```



In [53]:

```
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable

#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prett
ytable

x = PrettyTable()

x.field_names = ["Architecture used","Hidden layers","Activation", "Optimizer used", "e
pochs","Dropouts","BN(batch size)"," accuracy"]

x.add_row(["784-512-128-10", 2,"RELU","Adam" , 20,"yes","yes(128)",0.9825])
x.add_row(["784-512-128-256-10", 3,"RELU","Adam" , 40,"yes","yes(128)",0.9833])
x.add_row(["784-512-128-256-1024-2048-10", 5,"RELU","Adam" , 60,"yes","yes(256)",0.9841
])
x.add_row(["784-512-128-256-1024-2048-10", 5,"RELU","Adam" , 40,"no","no",0.0958])
x.add_row(["784-512-128-256-1024-2048-10", 5,"RELU","SGD" , 40,"no","yes(128)",0.9729])
x.add_row(["784-512-128-256-10", 3,"sigmoid","Adam" , 40,"yes","no",0.9721])
x.add_row(["784-512-128-256-10", 3,"tanh","Adam" , 30,"no","yes(64)",0.9736])
x.add_row(["784-512-128-256-1024-2048-10", 5,"RELU","SGD" , 60,"yes","no",0.9832])
x.add_row(["784-512-128-256-1024-2048-10", 5,"softmax","Adam" , 50,"yes","yes(256)",0.1
01])
x.add_row(["784-512-128-256-1024-2048-10", 5,"relu","Adam" , 50,"yes","yes(256)",0.9802
])


print(x)
```

```
+------------------------------+---------------+------------+-------------
---+--------+----------+----------------+-----------+
|      Architecture used       | Hidden layers | Activation | Optimizer us
ed | epochs | Dropouts | BN(batch size) |  accuracy |
+------------------------------+---------------+------------+-------------
---+--------+----------+----------------+-----------+
|        784-512-128-10        |       2       |    RELU    |      Adam
|   20   |   yes    |    yes(128)    |   0.9825  |
|      784-512-128-256-10      |       3       |    RELU    |      Adam
|   40   |   yes    |    yes(128)    |   0.9833  |
| 784-512-128-256-1024-2048-10 |       5       |    RELU    |      Adam
|   60   |   yes    |    yes(256)    |   0.9841  |
| 784-512-128-256-1024-2048-10 |       5       |    RELU    |      Adam
|   40   |    no    |       no       |   0.0958  |
| 784-512-128-256-1024-2048-10 |       5       |    RELU    |      SGD
|   40   |    no    |    yes(128)    |   0.9729  |
|      784-512-128-256-10      |       3       |  sigmoid   |      Adam
|   40   |   yes    |       no       |   0.9721  |
|      784-512-128-256-10      |       3       |    tanh    |      Adam
|   30   |    no    |    yes(64)     |   0.9736  |
| 784-512-128-256-1024-2048-10 |       5       |    RELU    |      SGD
|   60   |   yes    |       no       |   0.9832  |
| 784-512-128-256-1024-2048-10 |       5       |  softmax   |      Adam
|   50   |   yes    |    yes(256)    |   0.101   |
| 784-512-128-256-1024-2048-10 |       5       |    relu    |      Adam
|   50   |   yes    |    yes(256)    |   0.9802  |
+------------------------------+---------------+------------+-------------
---+--------+----------+----------------+-----------+
```

Summary :-

1.) We tried various types of architectures for MLP with 2 hidden layers,3 hidden layers and 5 hidden layers also.

2.) We measured the accuracy for all the cases we considered

3.) Along with different architectures we tried different initializations like xavier/cohort initialization,he_normal initialization etc and varied them with applying batch normalization and sometimes removing batch normalizations.

4.) We also did experiments with dropout layers and we also changed the dropout rates as well.

5.) We also did experiments with the optimizers which are avaliable to us like adam,sgd etc.

6.) Final conclusion that can be said is that choosing which will be the right architecture for us and choosing the best initialization,dropout rates,epochs,optimizers actually depends on data also and also the initialized parameters also we can see in the above experiments that some architectures gave us a very good accuracy at very low epoch aslo while some couldn't give us a good accuracy even at high epoch numbers hence it can be said that only increasing the epochs does not gurantee us a good accuracy afterall.

7.) the graphs between the number of epochs and train/validation accuracy shows us how our model converges the loss to a value which is as minimum as possible in that epoch range.

Best results - If we go with kaggle results methodilogy which is highly dependent upon the validation accuracy considering about all our experiments although some have very close accuracy still the best accuracy was found to be 98.32% for the following configuration :-

glorot_uniform initialization + sgd optimizer + relu activation function + Dropout(0.33) + no batch normalization + Model Architecture --> [(input)--(hidden layer 1 (512 neurons))--(hidden layer 2 (128 neurons))---(hidden layer 3 (256 neurons))--(hidden layer 4 (1024 neurons))----(hidden layer 5 (2048 neurons))---(output layer)]

In [ ]:

```

```