MLP in MNIST dataset -- Keras Implementation

Importing packages

In [1]:

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

```
Using TensorFlow backend.
```

Function for plotting dynamic graph

In [2]:

```
%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
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()
```

Loading mnist data

In [3]:

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

In [4]:

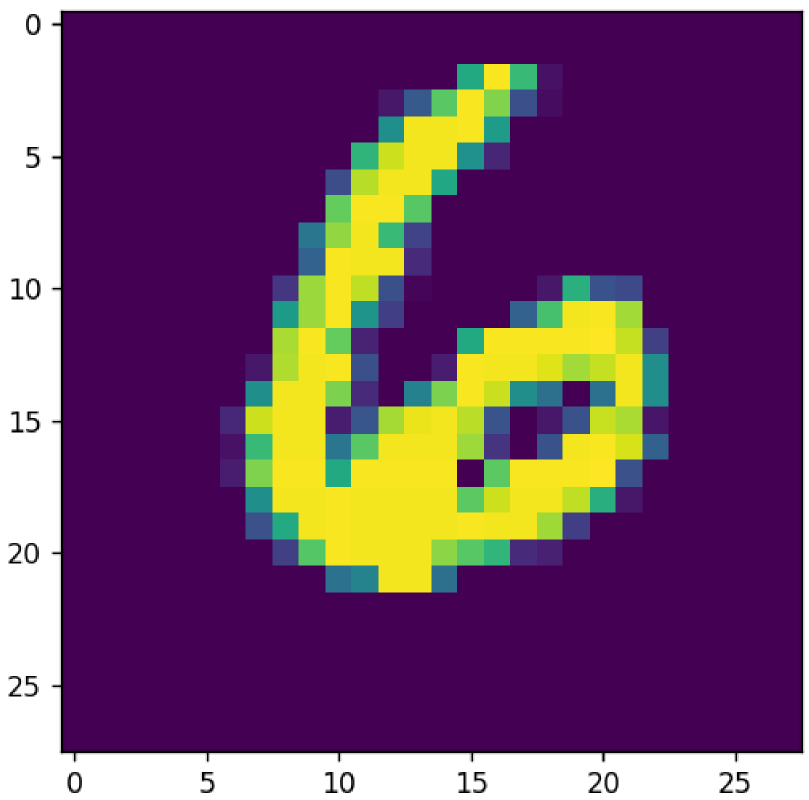
```
print("Number of training examples :", X_train.shape[0], "and each image is of shape (%d, %
print("Number of training examples :", X_test.shape[0], "and each image is of shape (%d, %c
```

```
Number of training examples : 60000 and each image is of shape (28, 28)
Number of training examples : 10000 and each image is of shape (28, 28)
```

In [5]:

```
plt.imshow(X_train[299])
```

Figure 1



Out[5]:

```
<matplotlib.image.AxesImage at 0x1f0adaa87f0>
```

In [6]:

```
# if you observe the input shape its 2 dimensional vector
# for each image we have a (28*28) vector
# we will convert the (28*28) vector into single dimensional vector of 1 * 784

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

In [7]:

```
print(X_train.shape, y_train)
print(X_test.shape, y_test)
```

```
(60000, 784) [5 0 4 ... 5 6 8]
(10000, 784) [7 2 1 ... 4 5 6]
```

In [8]:

```
# after converting the input images from 3d to 2d vectors

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

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

sample data

In [9]:

```
# An example data point
print(X_train[0])
```

```
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 255
 247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  36  94 154
 170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0   0   0
   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82
  82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 253
 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  35 241
 225 160 108   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
 253 207   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253
 253 201  78   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0  18 171 219 253 253 253 253 195
  80   9   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0 136 253 253 253 212 135 132  16
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
```

Normalizing pixel values

In [9]:

```
# if we observe the above matrix each cell is having a value between 0-255
# before we move to apply machine learning algorithms lets try to normalize the data
# X => (X - Xmin)/(Xmax-Xmin) = X/255

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

After normalization (pixel value ranges from 0-1)

In [10]:

```
# example data point after normlizing
print(X_train[0])
```

```
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 255
 247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  36  94 154
 170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0   0   0
   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82
  82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 253
 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  35 241
 225 160 108   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
 253 207   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253
 253 201  78   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0  18 171 219 253 253 253 253 195
  80   9   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0 136 253 253 253 212 135 132  16
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
```

Encoding class labels

In [11]:

```
# here we are having a class number for each image
print("Class label of first image :", y_train[0])

# lets convert this into a 10 dimensional vector
# ex: consider an image is 5 convert it into 5 => [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
# this conversion needed for MLPs

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

print("After converting the output into a vector : ",Y_train[0])
```

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

Model 1

In [12]:

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

# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

In [13]:

```
model1 = Sequential()
model1.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=Randc

model1.add(Dropout(0.5))

model1.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=6

model1.add(Dropout(0.5))

model1.add(Dense(output_dim, activation='softmax'))

print(model1.summary())

model1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model1.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, v
```

```
WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\Deep_learning\lib\site
-packages\keras\backend\tensorflow_backend.py:74: The name tf.get_default_gr
aph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\Deep_learning\lib\site
-packages\keras\backend\tensorflow_backend.py:517: The name tf.placeholder i
s deprecated. Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\Deep_learning\lib\site
-packages\keras\backend\tensorflow_backend.py:4115: The name tf.random_norma
l is deprecated. Please use tf.random.normal instead.

WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\Deep_learning\lib\site
-packages\keras\backend\tensorflow_backend.py:133: The name tf.placeholder_w
ith_default is deprecated. Please use tf.compat.v1.placeholder_with_default
instead.

WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\Deep_learning\lib\site
-packages\keras\backend\tensorflow_backend.py:3445: calling dropout (from te
nsorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed
in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\Deep_learning\lib\site
-packages\keras\backend\tensorflow_backend.py:4138: The name tf.random_unifo
rm 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, 10)                1290      
=================================================================
```

```
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
None
WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\Deep_learning\lib\site
-packages\keras\optimizers.py:790: The name tf.train.Optimizer is deprecate
d. Please use tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From C:\Users\vansh\Anaconda3\envs\Deep_learning\lib\site
-packages\tensorflow\python\ops\math_grad.py:1250: add_dispatch_support.<loc
als>.wrapper (from tensorflow.python.ops.array_ops) is deprecated and will b
e 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 1/20
60000/60000 [==============================] - 5s 80us/step - loss: 12.5850
- acc: 0.2181 - val_loss: 9.8416 - val_acc: 0.3886
Epoch 2/20
60000/60000 [==============================] - 3s 48us/step - loss: 10.1937
- acc: 0.3669 - val_loss: 8.5453 - val_acc: 0.4693
Epoch 3/20
60000/60000 [==============================] - 3s 49us/step - loss: 9.3444 -
acc: 0.4199 - val_loss: 8.0839 - val_acc: 0.4981
Epoch 4/20
60000/60000 [==============================] - 3s 47us/step - loss: 9.1723 -
acc: 0.4304 - val_loss: 8.0043 - val_acc: 0.5030
Epoch 5/20
60000/60000 [==============================] - 3s 48us/step - loss: 8.3629 -
acc: 0.4807 - val_loss: 7.0649 - val_acc: 0.5615
Epoch 6/20
60000/60000 [==============================] - 3s 47us/step - loss: 8.1204 -
acc: 0.4960 - val_loss: 7.0859 - val_acc: 0.5599
Epoch 7/20
60000/60000 [==============================] - 3s 48us/step - loss: 7.8107 -
acc: 0.5152 - val_loss: 6.6105 - val_acc: 0.5896
Epoch 8/20
60000/60000 [==============================] - 3s 47us/step - loss: 7.9137 -
acc: 0.5088 - val_loss: 6.6857 - val_acc: 0.5851
Epoch 9/20
60000/60000 [==============================] - 3s 49us/step - loss: 8.2320 -
acc: 0.4890 - val_loss: 7.5361 - val_acc: 0.5324
Epoch 10/20
60000/60000 [==============================] - 3s 49us/step - loss: 8.0205 -
acc: 0.5021 - val_loss: 7.1956 - val_acc: 0.5535
Epoch 11/20
60000/60000 [==============================] - 3s 48us/step - loss: 8.0914 -
acc: 0.4978 - val_loss: 6.2347 - val_acc: 0.6126
Epoch 12/20
60000/60000 [==============================] - 3s 48us/step - loss: 7.3911 -
acc: 0.5411 - val_loss: 6.2756 - val_acc: 0.6105
Epoch 13/20
60000/60000 [==============================] - 3s 48us/step - loss: 7.5374 -
acc: 0.5321 - val_loss: 8.4569 - val_acc: 0.4751
Epoch 14/20
60000/60000 [==============================] - 3s 48us/step - loss: 6.9246 -
acc: 0.5702 - val_loss: 5.3398 - val_acc: 0.6686
Epoch 15/20
60000/60000 [==============================] - 3s 48us/step - loss: 6.5722 -
acc: 0.5921 - val_loss: 5.6229 - val_acc: 0.6509
```

```
Epoch 16/20
60000/60000 [==============================] - 3s 49us/step - loss: 6.5891 -
acc: 0.5909 - val_loss: 5.2502 - val_acc: 0.6742
Epoch 17/20
60000/60000 [==============================] - 3s 49us/step - loss: 6.2670 -
acc: 0.6110 - val_loss: 4.5675 - val_acc: 0.7165
Epoch 18/20
60000/60000 [==============================] - 3s 48us/step - loss: 6.1065 -
acc: 0.6209 - val_loss: 4.6888 - val_acc: 0.7088
Epoch 19/20
60000/60000 [==============================] - 3s 48us/step - loss: 6.3317 -
acc: 0.6070 - val_loss: 7.4789 - val_acc: 0.5359
Epoch 20/20
60000/60000 [==============================] - 3s 48us/step - loss: 6.6072 -
acc: 0.5899 - val_loss: 5.4056 - val_acc: 0.6646
```

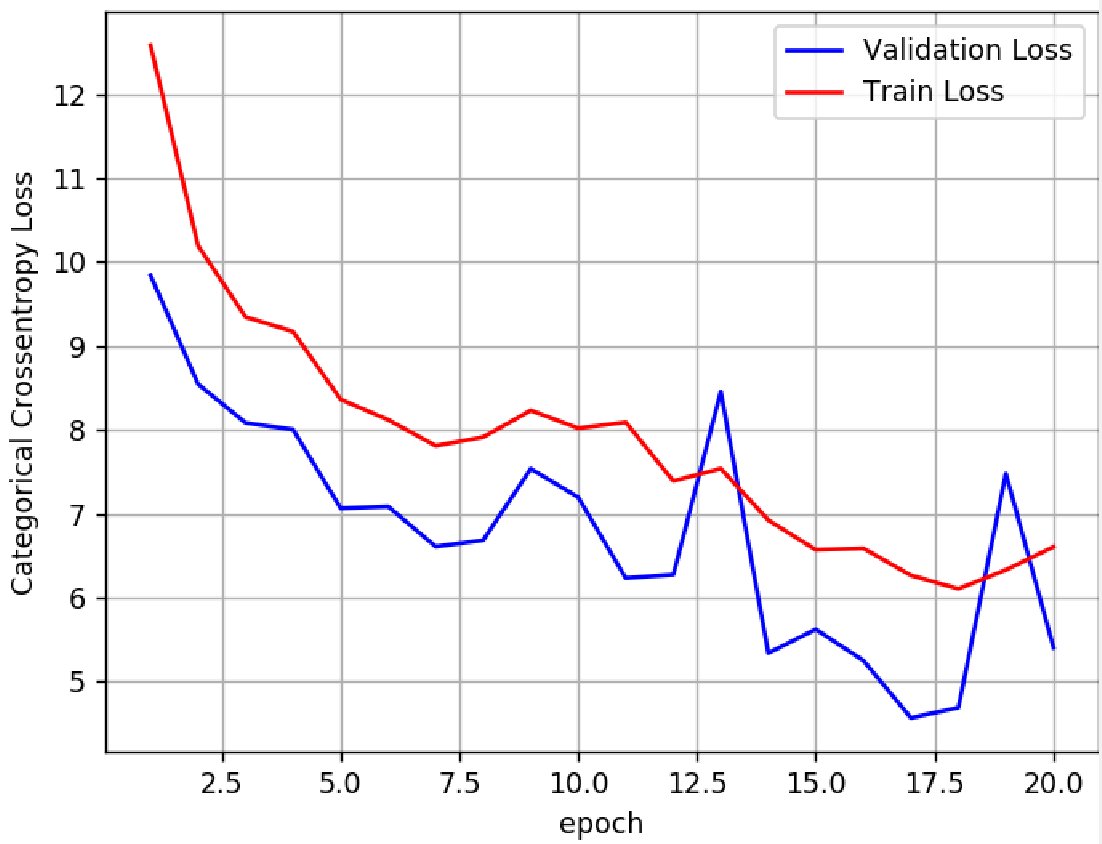
train and test loss vs Epochs

In [16]:

```
score = model1.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, verbos

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

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


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

```
Test score: 5.405633866882324
Test accuracy: 0.6646
```

Figure 4

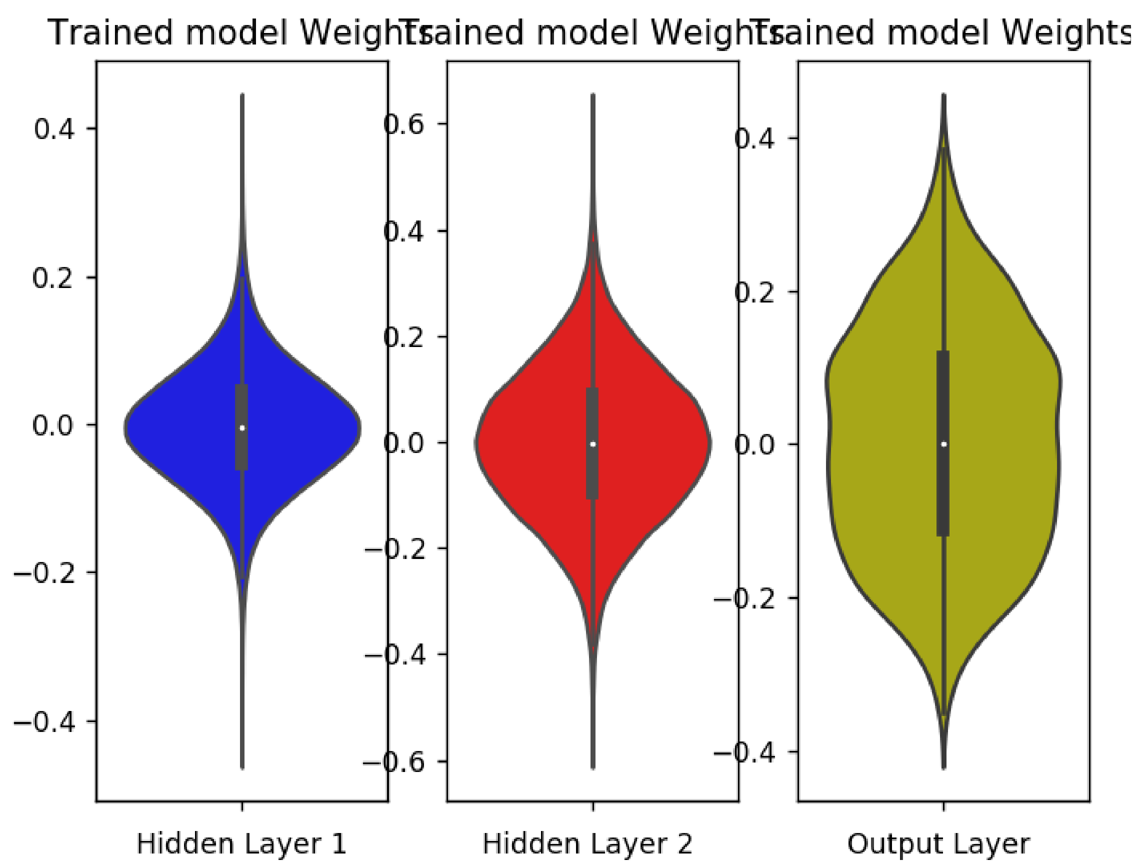


Violin plots of weights

In [17]:

```
w_after = model1.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()
```

Figure 5



1. Weights at hidden layer 1 ranges (-0.2 to 0.2).
2. Weights at hidden layer 2 have more variance (-0.4 to 0.4).
3. Weights at output layer are off-centered (-0.4 to 0.4).

Model 2

In [18]:

```
model2 = Sequential()
model2.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=Randc

model2.add(Dense(128, activation='relu', input_shape=(input_dim,), kernel_initializer=Randc

model2.add(Dropout(0.5))

model2.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.

model2.add(Dropout(0.5))

model2.add(Dense(output_dim, activation='softmax'))

print(model2.summary())

model2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model2.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, v
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_5 (Dense)              (None, 128)               65664
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_6 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_7 (Dense)              (None, 10)                650
=================================================================
Total params: 476,490
Trainable params: 476,490
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 61us/step - loss: 13.4442
- acc: 0.1647 - val_loss: 12.6372 - val_acc: 0.2158
Epoch 2/20
60000/60000 [==============================] - 3s 51us/step - loss: 11.6720
- acc: 0.2752 - val_loss: 9.9816 - val_acc: 0.3807
Epoch 3/20
60000/60000 [==============================] - 3s 51us/step - loss: 10.5429
- acc: 0.3455 - val_loss: 9.9067 - val_acc: 0.3851
Epoch 4/20
60000/60000 [==============================] - 3s 53us/step - loss: 10.5595
- acc: 0.3446 - val_loss: 9.9687 - val_acc: 0.3814
Epoch 5/20
60000/60000 [==============================] - 3s 53us/step - loss: 10.3295
- acc: 0.3588 - val_loss: 9.9273 - val_acc: 0.3839
Epoch 6/20
60000/60000 [==============================] - 3s 50us/step - loss: 10.0814
- acc: 0.3743 - val_loss: 9.9309 - val_acc: 0.3836
```

```
Epoch 7/20
60000/60000 [==============================] - 3s 51us/step - loss: 10.2240
- acc: 0.3655 - val_loss: 9.8755 - val_acc: 0.3872
Epoch 8/20
60000/60000 [==============================] - 3s 50us/step - loss: 9.8280 -
acc: 0.3900 - val_loss: 8.9418 - val_acc: 0.4450
Epoch 9/20
60000/60000 [==============================] - 3s 50us/step - loss: 9.4864 -
acc: 0.4113 - val_loss: 9.7467 - val_acc: 0.3952
Epoch 10/20
60000/60000 [==============================] - 3s 52us/step - loss: 9.4932 -
acc: 0.4108 - val_loss: 9.0106 - val_acc: 0.4409
Epoch 11/20
60000/60000 [==============================] - 3s 54us/step - loss: 8.9193 -
acc: 0.4464 - val_loss: 8.6056 - val_acc: 0.4660
Epoch 12/20
60000/60000 [==============================] - 3s 52us/step - loss: 8.9756 -
acc: 0.4430 - val_loss: 8.4394 - val_acc: 0.4764
Epoch 13/20
60000/60000 [==============================] - 3s 51us/step - loss: 8.9882 -
acc: 0.4421 - val_loss: 8.3750 - val_acc: 0.4804
Epoch 14/20
60000/60000 [==============================] - 3s 51us/step - loss: 8.7018 -
acc: 0.4599 - val_loss: 8.2476 - val_acc: 0.4883
Epoch 15/20
60000/60000 [==============================] - 3s 51us/step - loss: 8.6530 -
acc: 0.4630 - val_loss: 8.2132 - val_acc: 0.4902
Epoch 16/20
60000/60000 [==============================] - 3s 54us/step - loss: 8.6741 -
acc: 0.4617 - val_loss: 8.0635 - val_acc: 0.4996
Epoch 17/20
60000/60000 [==============================] - 3s 52us/step - loss: 9.0568 -
acc: 0.4379 - val_loss: 8.3073 - val_acc: 0.4846
Epoch 18/20
60000/60000 [==============================] - 3s 53us/step - loss: 8.8086 -
acc: 0.4533 - val_loss: 8.2073 - val_acc: 0.4908
Epoch 19/20
60000/60000 [==============================] - 3s 53us/step - loss: 8.8016 -
acc: 0.4538 - val_loss: 8.3379 - val_acc: 0.4827
Epoch 20/20
60000/60000 [==============================] - 3s 53us/step - loss: 8.7108 -
acc: 0.4595 - val_loss: 8.5877 - val_acc: 0.4672
```

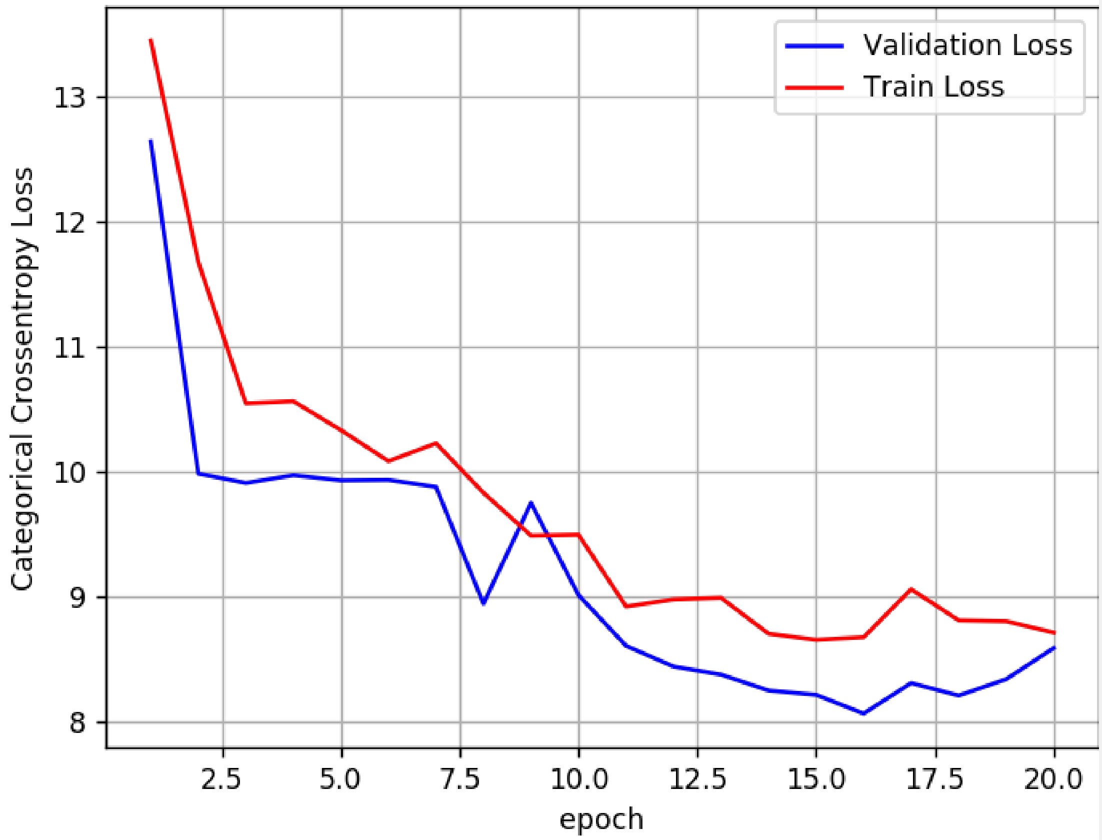
train and test loss vs Epochs

In [19]:

```
score = model2.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, verbos

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

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


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

```
Test score: 8.587725143432618
Test accuracy: 0.4672
```

Figure 6

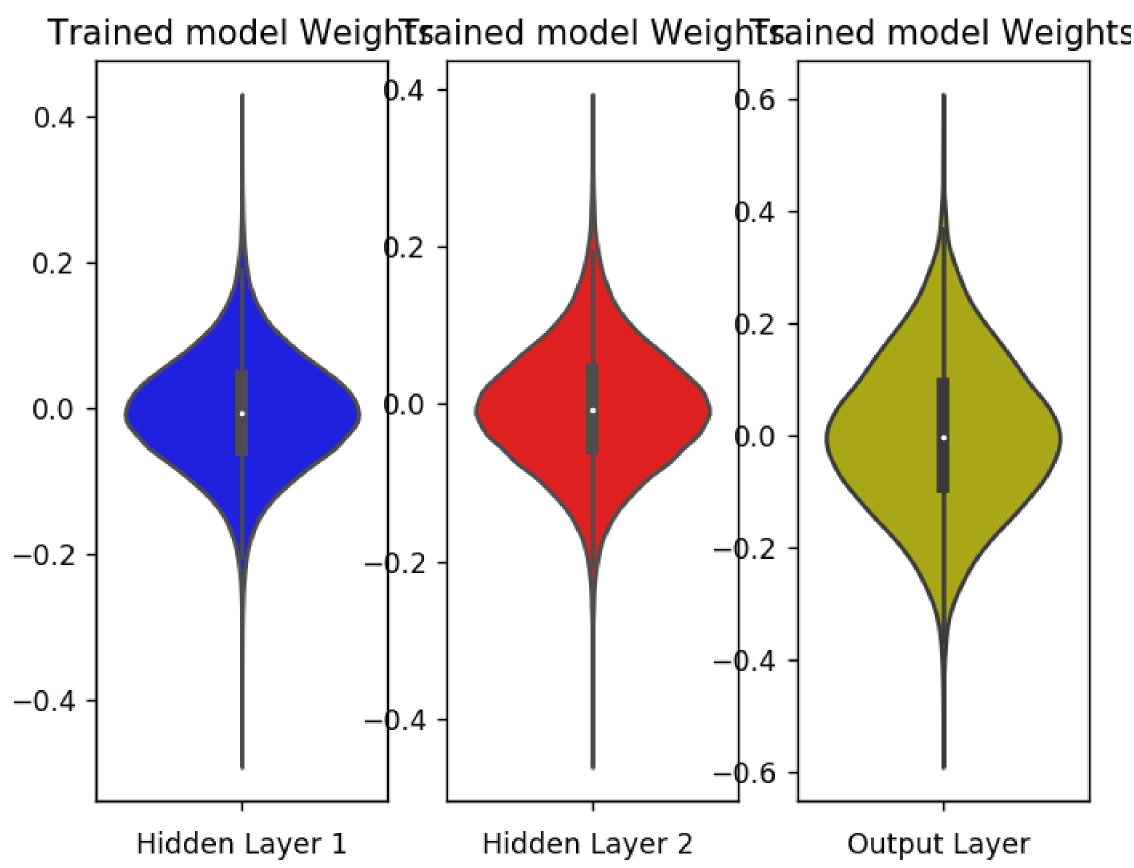


Visualizing weights with violin plot

In [20]:

```
w_after = model2.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()
```

Figure 7



1. Weights at hidden layer 1 ranges(-0.2 to 0.2).
2. Weights at hidden layer 2 ranges (-0.3 to 0.3).
3. Weights at output layer ranges (-0.4 to 0.4).

Model 3

In [21]:

```
from keras.initializers import he_normal
from keras.layers.normalization import BatchNormalization

model3 = Sequential()

model3.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=he_nc

model3.add(Dense(128, activation='relu', input_shape=(input_dim,), kernel_initializer=he_nc

model3.add(Dropout(0.5))

model3.add(Dense(128, activation='relu', input_shape=(input_dim,), kernel_initializer=he_nc

model3.add(Dense(64, activation='relu', input_shape=(input_dim,), kernel_initializer=he_nor

model3.add(Dense(32, activation='relu', kernel_initializer=he_normal(seed=None)))

model3.add(Dropout(0.5))

model3.add(BatchNormalization())

model3.add(Dense(output_dim, activation='softmax'))

print(model3.summary())

model3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model3.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, v
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_8 (Dense)              (None, 512)               401920    
_________________________________________________________________
dense_9 (Dense)              (None, 128)               65664     
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_10 (Dense)             (None, 128)               16512     
_________________________________________________________________
dense_11 (Dense)             (None, 64)                8256      
_________________________________________________________________
dense_12 (Dense)             (None, 32)                2080      
_________________________________________________________________
dropout_6 (Dropout)          (None, 32)                0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 32)                128       
_________________________________________________________________
dense_13 (Dense)             (None, 10)                330       
=================================================================
Total params: 494,890
Trainable params: 494,826
Non-trainable params: 64
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.9664 -
acc: 0.7120 - val_loss: 0.2113 - val_acc: 0.9447
```

```
Epoch 2/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.3508 -
acc: 0.9156 - val_loss: 0.1334 - val_acc: 0.9623
Epoch 3/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.2472 -
acc: 0.9411 - val_loss: 0.1125 - val_acc: 0.9684
Epoch 4/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.1941 -
acc: 0.9531 - val_loss: 0.1159 - val_acc: 0.9704
Epoch 5/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.1642 -
acc: 0.9596 - val_loss: 0.0865 - val_acc: 0.9775
Epoch 6/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.1388 -
acc: 0.9652 - val_loss: 0.0983 - val_acc: 0.9747
Epoch 7/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.1247 -
acc: 0.9679 - val_loss: 0.0996 - val_acc: 0.9764
Epoch 8/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.1184 -
acc: 0.9687 - val_loss: 0.0945 - val_acc: 0.9766
Epoch 9/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.1051 -
acc: 0.9713 - val_loss: 0.1006 - val_acc: 0.9777
Epoch 10/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0984 -
acc: 0.9736 - val_loss: 0.0903 - val_acc: 0.9790
Epoch 11/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0850 -
acc: 0.9769 - val_loss: 0.0896 - val_acc: 0.9794
Epoch 12/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0816 -
acc: 0.9781 - val_loss: 0.0981 - val_acc: 0.9798
Epoch 13/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0774 -
acc: 0.9793 - val_loss: 0.0944 - val_acc: 0.9796
Epoch 14/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0719 -
acc: 0.9802 - val_loss: 0.1023 - val_acc: 0.9785
Epoch 15/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0708 -
acc: 0.9800 - val_loss: 0.1031 - val_acc: 0.9792
Epoch 16/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0713 -
acc: 0.9799 - val_loss: 0.1086 - val_acc: 0.9784
Epoch 17/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.0647 -
acc: 0.9822 - val_loss: 0.0949 - val_acc: 0.9806
Epoch 18/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0642 -
acc: 0.9815 - val_loss: 0.1004 - val_acc: 0.9797
Epoch 19/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0576 -
acc: 0.9833 - val_loss: 0.1076 - val_acc: 0.9807
Epoch 20/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0567 -
acc: 0.9835 - val_loss: 0.1011 - val_acc: 0.9815
```

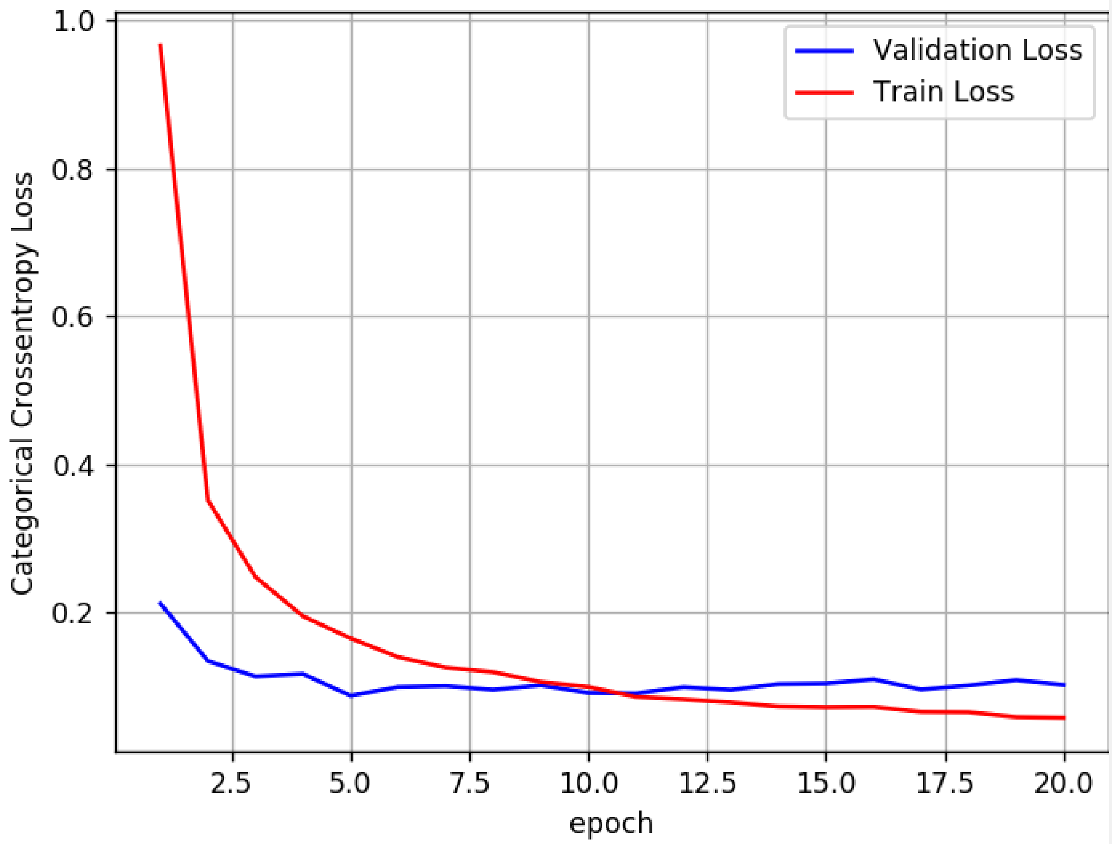
Train and test loss vs epochs

In [22]:

```
score = model3.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, verbos

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

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


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

```
Test score: 0.10108382558252847
Test accuracy: 0.9815
```

Figure 8



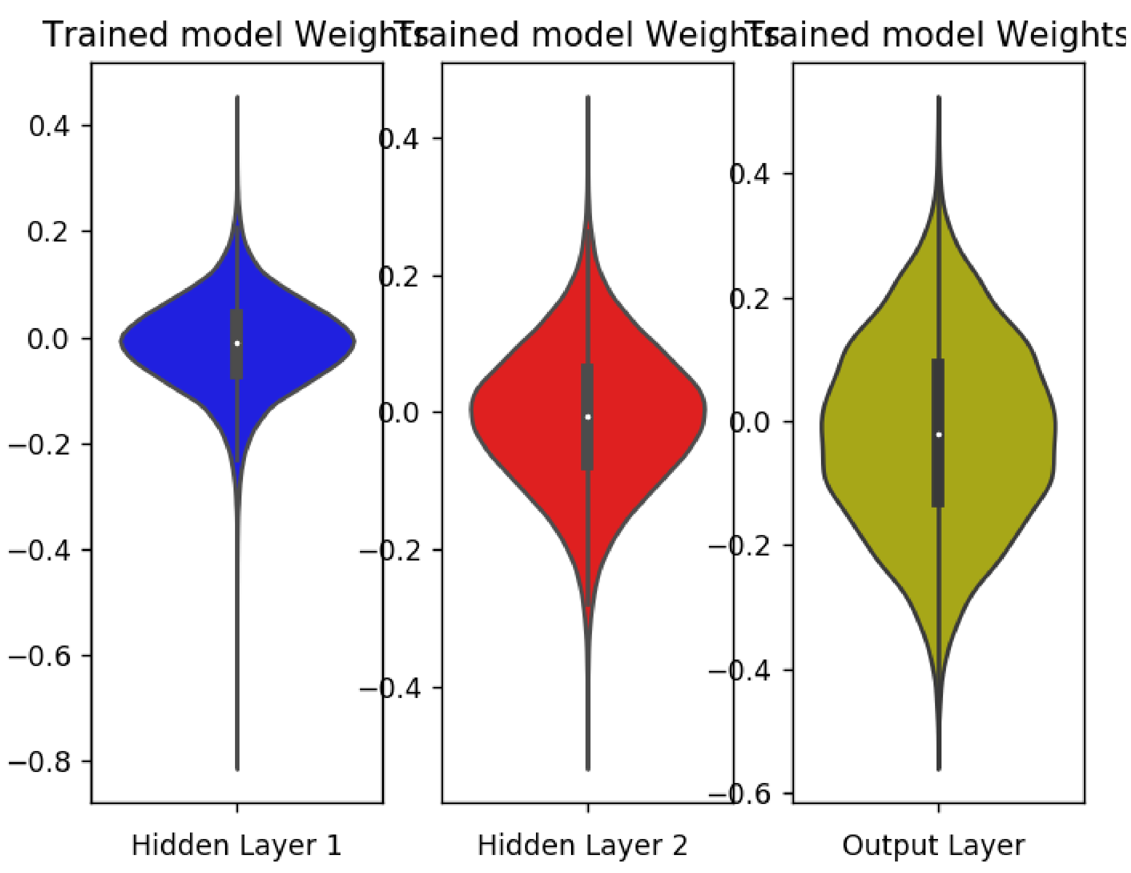
1. validation loss was not decreasing after 2 epochs while train loss was decreasing slowly.

Visualizing weights with violin plot

In [23]:

```
w_after = model3.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()
```

Figure 9



1. Weights at hidden layer 1 ranges(-0.2 to 0.2).
2. Weights at hidden layer 2 ranges (-0.3 to 0.3).
3. Weights at output layer ranges (-0.6 to 0.4).

In [24]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["model", "No. dense layers","dropout layers and rate", "weight initilizati
x.add_row(["Model 1","2 (512,128)"," 2 (0.5)","Random", "58.99%","66.46%%"])
x.add_row(["Model 2","3 (512,128,64)","2 (0.5)", "Random", "45.9%%","46.7%"])
x.add_row(["Model 3","5(512,128,128,64,32)","2 (0.5)", "He_normal","98%","98%"])

print(x)
```

```
+---------+---------------------+-----------------------+---------------
------+---------------+--------------+
|  model  |   No. dense layers  | dropout layers and rate | weight initiliz
ation | Train accuracy | Test accuracy |
+---------+---------------------+-----------------------+---------------
------+---------------+--------------+
| Model 1 |     2 (512,128)     |         2 (0.5)         |        Random
|     58.99%     |    66.46%%    |
| Model 2 |    3 (512,128,64)   |        2 (0.5)          |        Random
|     45.9%%     |     46.7%     |
| Model 3 | 5(512,128,128,64,32) |        2 (0.5)          |      He_normal
|      98%       |      98%      |
+---------+---------------------+-----------------------+---------------
------+---------------+--------------+
```

Obeservations:

1. Model 1 has high train loss and low validation loss.
2. Model 1 has performed worst.
3. Slight decrese in model accuracy has been obeserved while adding dropouts.
4. He_normal weight initialization has slightly fasten the convergence of the model.
5. Model 4 is slightly overfitting with train loss of 0.05 and test loss of 0.1.