MLP on MNIST Data Set

In [0]:

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

In [0]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np
import time
plt.style.use('classic')
# 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 Data

In [9]:

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

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 1s 0us/step
```

In [10]:

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

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

In [0]:

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

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

In [12]:

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

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

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

In [13]:

```python
# 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]
```

In [0]:

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

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

In [15]:

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

```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
```

```
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.01176471 0.07058824 0.07058824 0.07058824
0.49411765 0.53333333 0.68627451 0.10196078 0.65098039 1.
0.96862745 0.49803922 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.11764706 0.14117647 0.36862745 0.60392157
0.66666667 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
0.88235294 0.6745098  0.99215686 0.94901961 0.76470588 0.25098039
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.19215686
0.93333333 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
0.99215686 0.99215686 0.99215686 0.98431373 0.36470588 0.32156863
0.32156863 0.21960784 0.15294118 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.07058824 0.85882353 0.99215686
0.99215686 0.99215686 0.99215686 0.99215686 0.77647059 0.71372549
0.96862745 0.94509804 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.31372549 0.61176471 0.41960784 0.99215686
0.99215686 0.80392157 0.04313725 0.         0.16862745 0.60392157
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.05490196 0.00392157 0.60392157 0.99215686 0.35294118
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.54509804 0.99215686 0.74509804 0.00784314 0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.04313725
0.74509804 0.99215686 0.2745098  0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.1372549  0.94509804
0.88235294 0.62745098 0.42352941 0.00392157 0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.31764706 0.94117647 0.99215686
0.99215686 0.46666667 0.09803922 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.17647059 0.72941176 0.99215686 0.99215686
0.58823529 0.10588235 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.0627451  0.36470588 0.98823529 0.99215686 0.73333333
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
```

```
 0.         0.         0.         0.         0.         0.
 0.         0.97647059 0.99215686 0.97647059 0.25098039 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.18039216 0.50980392 0.71764706 0.99215686
 0.99215686 0.81176471 0.00784314 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.15294118 0.58039216
 0.89803922 0.99215686 0.99215686 0.99215686 0.98039216 0.71372549
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.09411765 0.44705882 0.86666667 0.99215686 0.99215686 0.99215686
 0.99215686 0.78823529 0.30588235 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.09019608 0.25882353 0.83529412 0.99215686
 0.99215686 0.99215686 0.99215686 0.77647059 0.31764706 0.00784314
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.07058824 0.67058824
 0.85882353 0.99215686 0.99215686 0.99215686 0.99215686 0.76470588
 0.31372549 0.03529412 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.21568627 0.6745098  0.88627451 0.99215686 0.99215686 0.99215686
 0.99215686 0.95686275 0.52156863 0.04313725 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.53333333 0.99215686
 0.99215686 0.99215686 0.83137255 0.52941176 0.51764706 0.0627451
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
```

In [16]:

```
# 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.]
```

In [0]:

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

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

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

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

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

###

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

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

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

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

####

# https://keras.io/activations/

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

# from keras.layers import Activation, Dense

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

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

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


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

In [0]:

```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

MLP + RELU activation + adam optimizer + 2 hidden layer

In [14]:

```
# Multilayer perceptron

model_relu = Sequential()
model_relu.add(Dense(392, activation='relu', input_shape=(input_dim,),kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(Dense(196, activation='relu',kernel_initializer=RandomNormal(mean=0.0, stddev=0.062,
seed=None)))
model_relu.add(Dense(output_dim, activation='softmax'))
```

```
model_relu.summary()

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

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

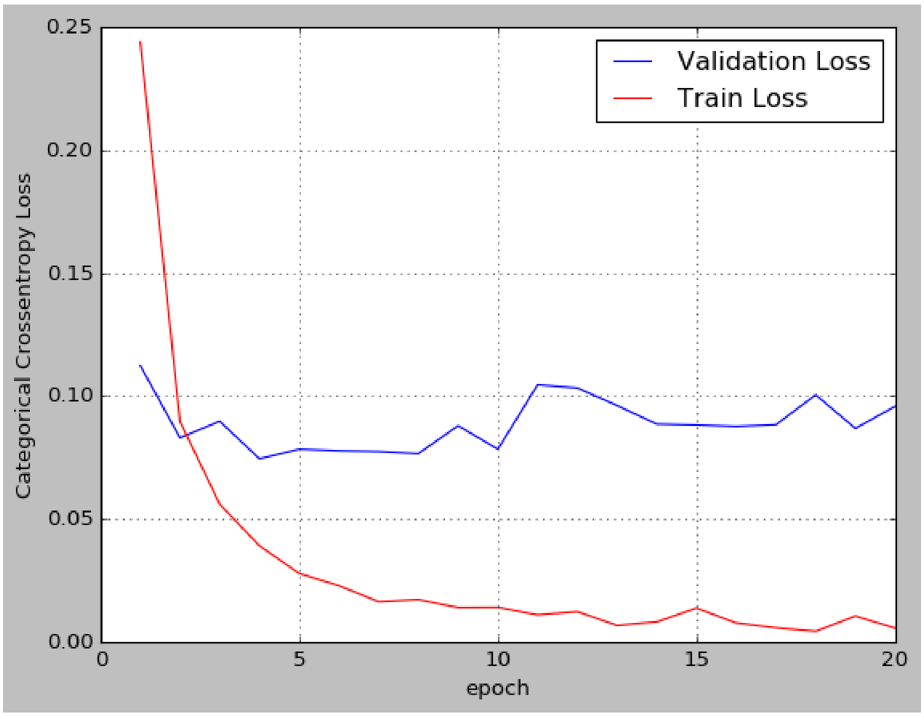
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 392)               307720    
_________________________________________________________________
dense_5 (Dense)              (None, 196)               77028     
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1970      
=================================================================
Total params: 386,718
Trainable params: 386,718
Non-trainable params: 0
_________________________________________________________________
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.2439 - acc: 0.9285 -
val_loss: 0.1123 - val_acc: 0.9662
Epoch 2/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0896 - acc: 0.9724 -
val_loss: 0.0830 - val_acc: 0.9744
Epoch 3/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0560 - acc: 0.9825 -
val_loss: 0.0897 - val_acc: 0.9717
Epoch 4/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0391 - acc: 0.9876 -
val_loss: 0.0745 - val_acc: 0.9776
Epoch 5/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0279 - acc: 0.9909 -
val_loss: 0.0783 - val_acc: 0.9793
Epoch 6/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0229 - acc: 0.9930 -
val_loss: 0.0777 - val_acc: 0.9779
Epoch 7/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0164 - acc: 0.9947 -
val_loss: 0.0774 - val_acc: 0.9795
Epoch 8/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0172 - acc: 0.9944 -
val_loss: 0.0766 - val_acc: 0.9793
Epoch 9/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0140 - acc: 0.9953 -
val_loss: 0.0879 - val_acc: 0.9785
Epoch 10/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0141 - acc: 0.9954 -
val_loss: 0.0784 - val_acc: 0.9811
Epoch 11/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0111 - acc: 0.9960 -
val_loss: 0.1045 - val_acc: 0.9755
Epoch 12/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0124 - acc: 0.9961 -
val_loss: 0.1032 - val_acc: 0.9768
Epoch 13/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0068 - acc: 0.9978 -
val_loss: 0.0961 - val_acc: 0.9800
Epoch 14/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0082 - acc: 0.9973 -
val_loss: 0.0886 - val_acc: 0.9820
Epoch 15/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0138 - acc: 0.9951 -
val_loss: 0.0882 - val_acc: 0.9798
Epoch 16/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0078 - acc: 0.9974 -
val_loss: 0.0876 - val_acc: 0.9810
Epoch 17/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0058 - acc: 0.9979 -
```

```
val_loss: 0.0883 - val_acc: 0.9816
Epoch 18/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0044 - acc: 0.9986 -
val_loss: 0.1004 - val_acc: 0.9806
Epoch 19/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0105 - acc: 0.9965 -
val_loss: 0.0868 - val_acc: 0.9828
Epoch 20/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0057 - acc: 0.9982 -
val_loss: 0.0958 - val_acc: 0.9818
```

In [15]:

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

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

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

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

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

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

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

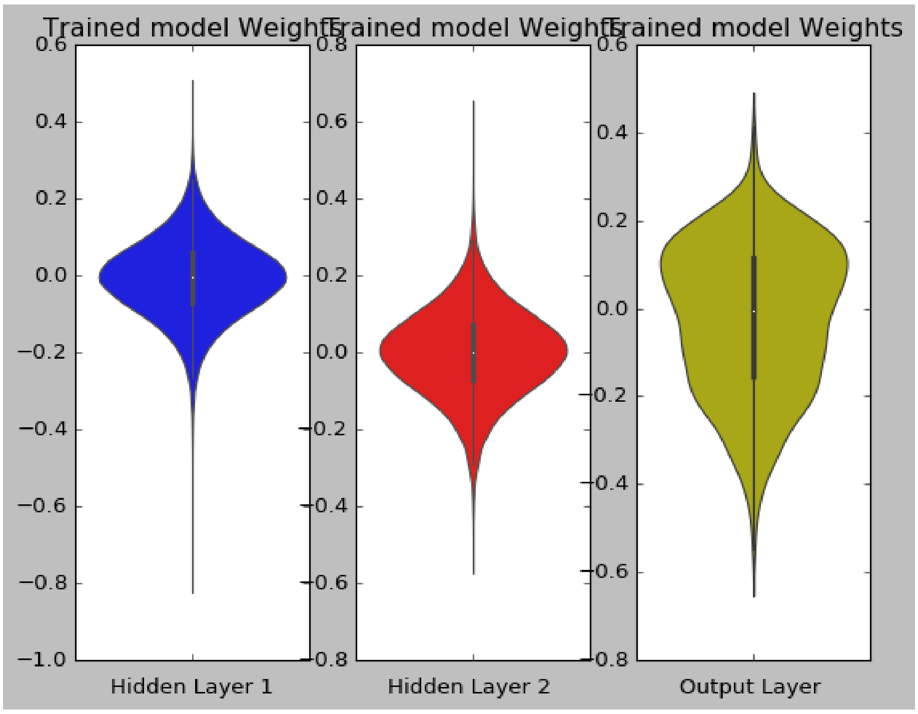
```
Test score: 0.09581032517525569
Test accuracy: 0.9818
```



In [16]:

```python
w_after = model_relu.get_weights()
```

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


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

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

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



MLP + RELU activation + adam optimizer + 2 hidden layer + Batch normalization and Drop out

In [4]:

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

```
Using TensorFlow backend.
```

In [18]:

```
# Multilayer perceptron

model_relu = Sequential()

model_relu.add(Dense(392, activation='relu', input_shape=(input_dim,),kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

model_relu.add(Dense(196, activation='relu',kernel_initializer=RandomNormal(mean=0.0, stddev=0.062,
```

```
seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

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

model_relu.summary()

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

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

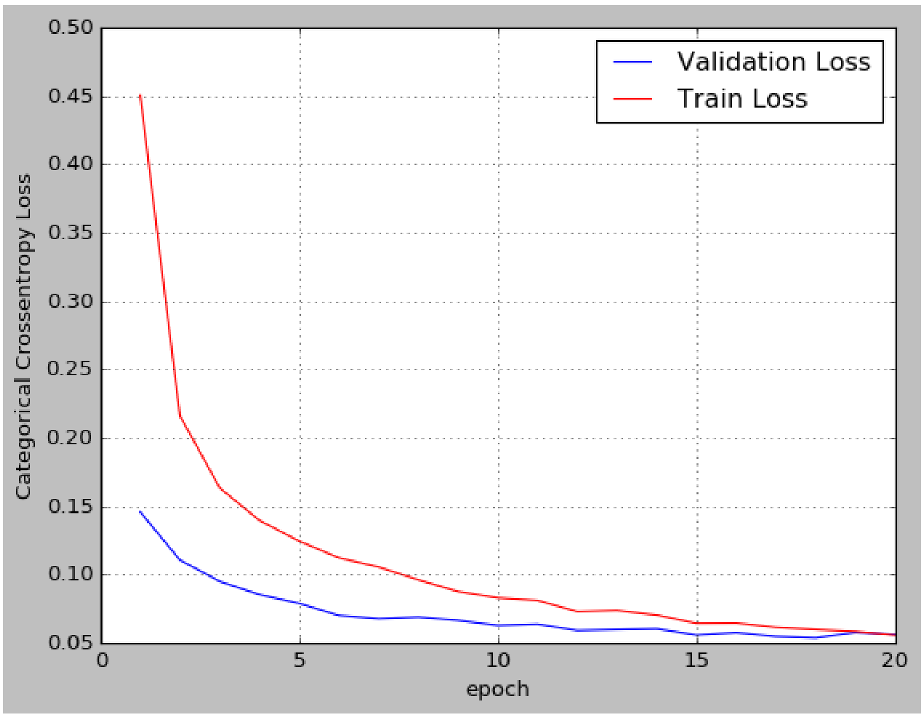
```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:3445: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future
version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_7 (Dense)              (None, 392)               307720    
_________________________________________________________________
batch_normalization_1 (Batch (None, 392)               1568      
_________________________________________________________________
dropout_1 (Dropout)          (None, 392)               0         
_________________________________________________________________
dense_8 (Dense)              (None, 196)               77028     
_________________________________________________________________
batch_normalization_2 (Batch (None, 196)               784       
_________________________________________________________________
dropout_2 (Dropout)          (None, 196)               0         
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1970      
=================================================================
Total params: 389,070
Trainable params: 387,894
Non-trainable params: 1,176
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.4503 - acc: 0.8642 -
val_loss: 0.1457 - val_acc: 0.9537
Epoch 2/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.2161 - acc: 0.9353 -
val_loss: 0.1105 - val_acc: 0.9658
Epoch 3/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.1633 - acc: 0.9502 -
val_loss: 0.0950 - val_acc: 0.9702
Epoch 4/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.1396 - acc: 0.9575 -
val_loss: 0.0854 - val_acc: 0.9727
Epoch 5/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.1244 - acc: 0.9611 -
val_loss: 0.0790 - val_acc: 0.9759
Epoch 6/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.1122 - acc: 0.9652 -
val_loss: 0.0701 - val_acc: 0.9778
Epoch 7/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.1055 - acc: 0.9670 -
val_loss: 0.0677 - val_acc: 0.9798
Epoch 8/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0961 - acc: 0.9706 -
val_loss: 0.0689 - val_acc: 0.9795
Epoch 9/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0877 - acc: 0.9729 -
val_loss: 0.0665 - val_acc: 0.9791
Epoch 10/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0830 - acc: 0.9739 -
val_loss: 0.0629 - val_acc: 0.9815
Epoch 11/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0811 - acc: 0.9745 -
val_loss: 0.0637 - val_acc: 0.9806
Epoch 12/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0730 - acc: 0.9773 -
val_loss: 0.0592 - val_acc: 0.9806
```

```
Epoch 13/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0738 - acc: 0.9766 -
val_loss: 0.0599 - val_acc: 0.9814
Epoch 14/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0704 - acc: 0.9777 -
val_loss: 0.0604 - val_acc: 0.9813
Epoch 15/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0644 - acc: 0.9793 -
val_loss: 0.0558 - val_acc: 0.9829
Epoch 16/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0646 - acc: 0.9789 -
val_loss: 0.0576 - val_acc: 0.9822
Epoch 17/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0614 - acc: 0.9803 -
val_loss: 0.0547 - val_acc: 0.9841
Epoch 18/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.0599 - acc: 0.9812 -
val_loss: 0.0537 - val_acc: 0.9837
Epoch 19/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0584 - acc: 0.9813 -
val_loss: 0.0577 - val_acc: 0.9816
Epoch 20/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0557 - acc: 0.9818 -
val_loss: 0.0562 - val_acc: 0.9831
```

In [19]:

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

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

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

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

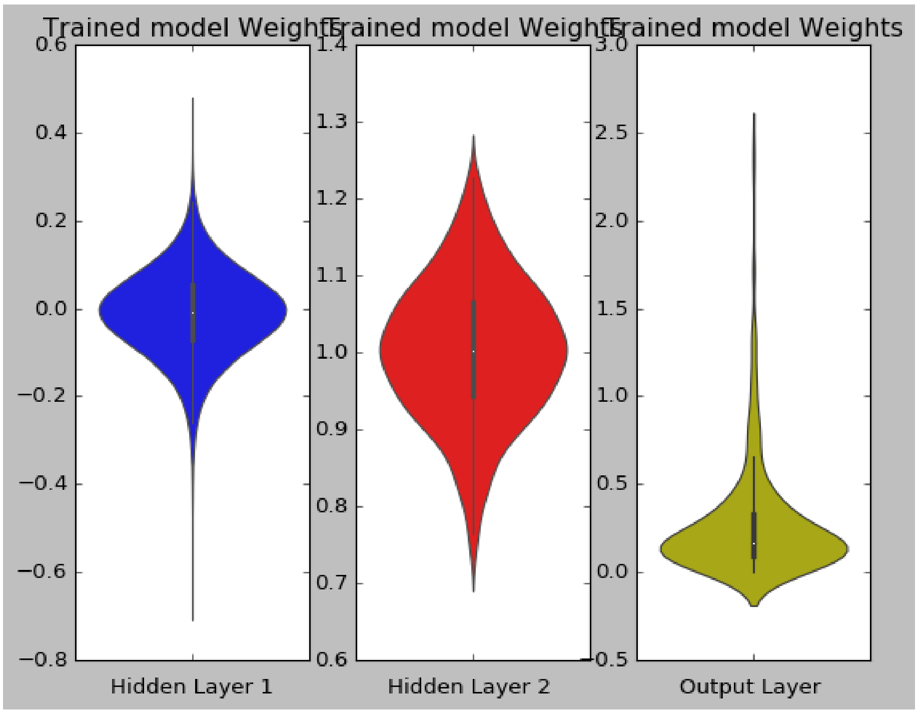
```
Test score: 0.05622168810109433
Test accuracy: 0.9831
```



In [20]:

```
w_after = model_relu.get_weights()
```

```
w_after = model_relu.get_weights()

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


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

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

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



MLP + RELU activation + adam optimizer + 3 hidden layer

In [19]:

```
# Multilayer perceptron

model_relu = Sequential()
model_relu.add(Dense(261, activation='relu', input_shape=(input_dim,),kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(Dense(87, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.062,
seed=None)))
model_relu.add(Dense(29, activation='relu',kernel_initializer=RandomNormal(mean=0.0, stddev=0.062,
seed=None)))
model_relu.add(Dense(output_dim, activation='softmax'))

model_relu.summary()

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

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

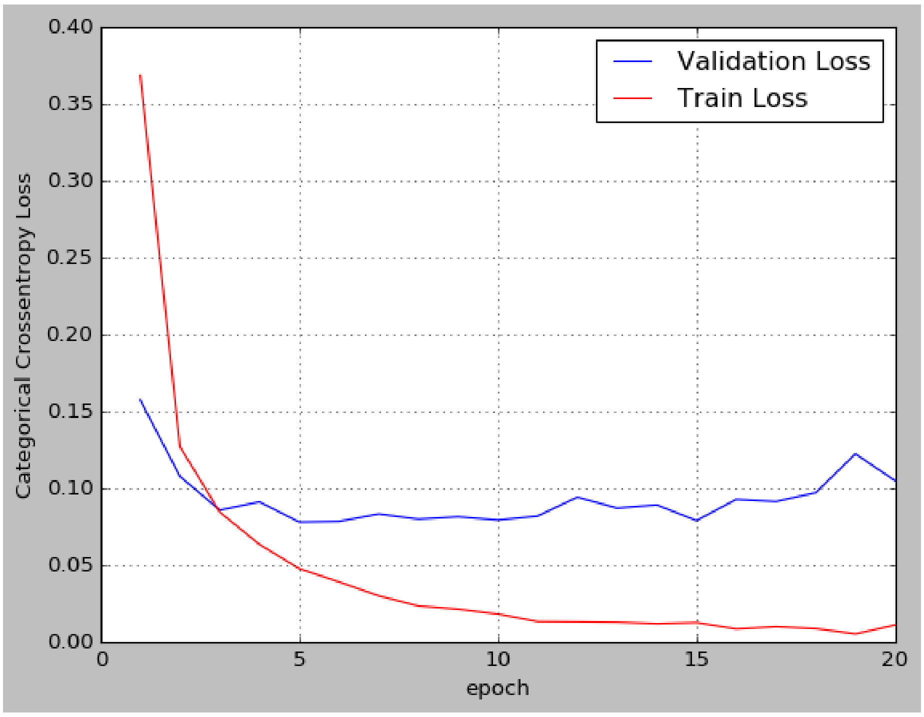
```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow/python/framework/op_def_library.py:263: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 261)               204885
_________________________________________________________________
dense_2 (Dense)              (None, 87)                22794
_________________________________________________________________
dense_3 (Dense)              (None, 29)                2552
_________________________________________________________________
dense_4 (Dense)              (None, 10)                300
=================================================================
Total params: 230,531
Trainable params: 230,531
Non-trainable params: 0
_________________________________________________________________
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.3684 - acc: 0.8915 -
val_loss: 0.1574 - val_acc: 0.9520
Epoch 2/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.1270 - acc: 0.9626 -
val_loss: 0.1077 - val_acc: 0.9687
Epoch 3/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0845 - acc: 0.9746 -
val_loss: 0.0858 - val_acc: 0.9741
Epoch 4/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0634 - acc: 0.9807 -
val_loss: 0.0911 - val_acc: 0.9706
Epoch 5/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0477 - acc: 0.9850 -
val_loss: 0.0780 - val_acc: 0.9756
Epoch 6/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0391 - acc: 0.9873 -
val_loss: 0.0784 - val_acc: 0.9757
Epoch 7/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0301 - acc: 0.9903 -
val_loss: 0.0833 - val_acc: 0.9762
Epoch 8/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0234 - acc: 0.9930 -
val_loss: 0.0800 - val_acc: 0.9785
Epoch 9/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0214 - acc: 0.9931 -
val_loss: 0.0815 - val_acc: 0.9785
Epoch 10/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0182 - acc: 0.9938 -
val_loss: 0.0793 - val_acc: 0.9785
Epoch 11/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0133 - acc: 0.9955 -
val_loss: 0.0819 - val_acc: 0.9791
Epoch 12/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0131 - acc: 0.9958 -
val_loss: 0.0942 - val_acc: 0.9766
Epoch 13/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0129 - acc: 0.9958 -
val_loss: 0.0871 - val_acc: 0.9776
Epoch 14/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0119 - acc: 0.9961 -
val_loss: 0.0891 - val_acc: 0.9797
Epoch 15/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0126 - acc: 0.9961 -
val_loss: 0.0790 - val_acc: 0.9809
Epoch 16/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0085 - acc: 0.9973 -
val_loss: 0.0928 - val_acc: 0.9806
Epoch 17/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0100 - acc: 0.9969 -
val_loss: 0.0914 - val_acc: 0.9778
```

```
Epoch 18/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0088 - acc: 0.9971 -
val_loss: 0.0971 - val_acc: 0.9783
Epoch 19/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0053 - acc: 0.9986 -
val_loss: 0.1223 - val_acc: 0.9771
Epoch 20/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0111 - acc: 0.9962 -
val_loss: 0.1049 - val_acc: 0.9796
```

In [20]:

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

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

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

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

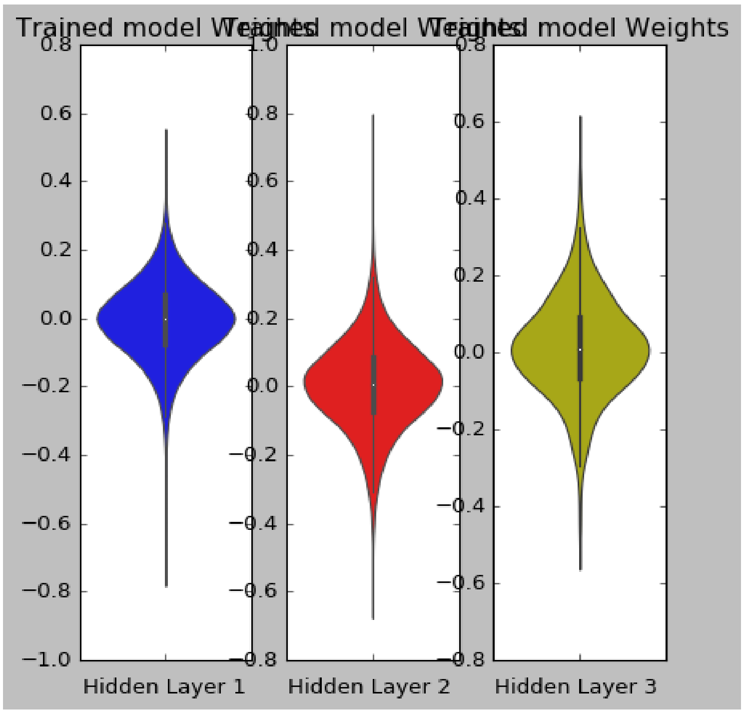
```
Test score: 0.10485377672798267
Test accuracy: 0.9796
```

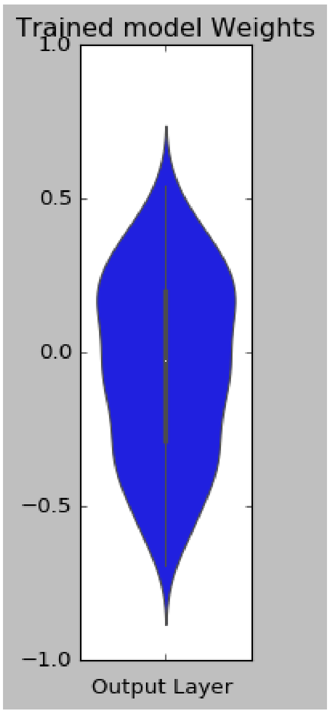


In [21]:

```
w_after = model_relu.get_weights()

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


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

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

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

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

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





MLP + RELU activation + adam optimizer + 3 hidden layer + Batch normalization and Drop out

In [27]:

```
# Multilayer perceptron

model_relu = Sequential()

model_relu.add(Dense(261, activation='relu', input_shape=(input_dim,),kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

model_relu.add(Dense(87, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.062,
seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

model_relu.add(Dense(29, activation='relu',kernel_initializer=RandomNormal(mean=0.0, stddev=0.062,
seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))


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


model_relu.summary()

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

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

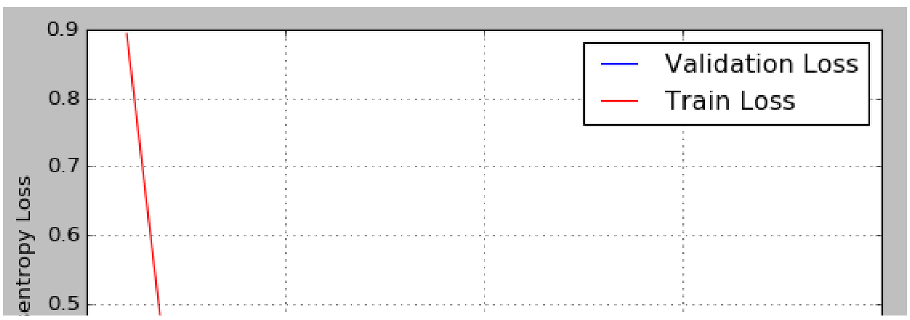
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_14 (Dense)             (None, 261)               204885
_________________________________________________________________
batch_normalization_3 (Batch (None, 261)               1044
_________________________________________________________________
dropout_3 (Dropout)          (None, 261)               0
_________________________________________________________________
dense_15 (Dense)             (None, 87)                22794
_________________________________________________________________
batch_normalization_4 (Batch (None, 87)                348
_________________________________________________________________
dropout_4 (Dropout)          (None, 87)                0
_________________________________________________________________
dense_16 (Dense)             (None, 29)                2552
_________________________________________________________________
batch_normalization_5 (Batch (None, 29)                116
_________________________________________________________________
dropout_5 (Dropout)          (None, 29)                0
_________________________________________________________________
dense_17 (Dense)             (None, 10)                300
=================================================================
Total params: 232,039
Trainable params: 231,285
Non-trainable params: 754
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.8934 - acc: 0.7249 -
val_loss: 0.2338 - val_acc: 0.9335
Epoch 2/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.4027 - acc: 0.8902 -
val_loss: 0.1656 - val_acc: 0.9506
Epoch 3/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.3096 - acc: 0.9173 -
val_loss: 0.1398 - val_acc: 0.9585
Epoch 4/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.2681 - acc: 0.9302 -
val_loss: 0.1143 - val_acc: 0.9678
Epoch 5/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.2352 - acc: 0.9380 -
val_loss: 0.1081 - val_acc: 0.9701
Epoch 6/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.2099 - acc: 0.9454 -
```

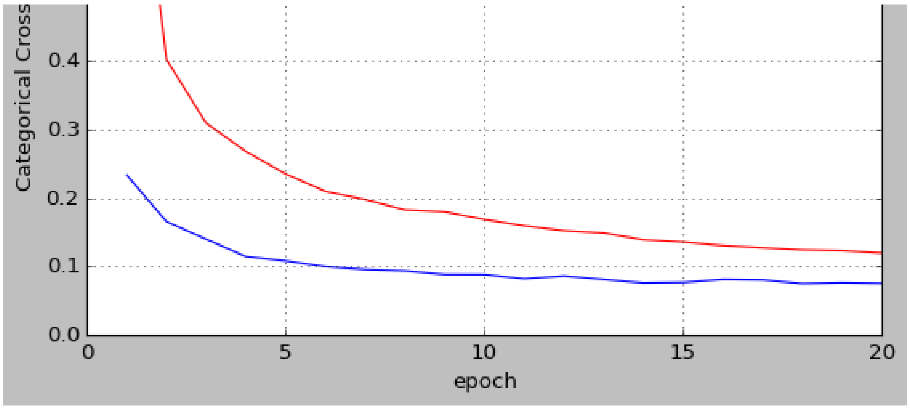
```
60000/60000 [==============================] - 5s 86us/step - loss: 0.2099 - acc: 0.9454 -
val_loss: 0.0999 - val_acc: 0.9716
Epoch 7/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.1978 - acc: 0.9490 -
val_loss: 0.0955 - val_acc: 0.9718
Epoch 8/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.1826 - acc: 0.9532 -
val_loss: 0.0936 - val_acc: 0.9734
Epoch 9/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.1798 - acc: 0.9538 -
val_loss: 0.0884 - val_acc: 0.9758
Epoch 10/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.1686 - acc: 0.9557 -
val_loss: 0.0882 - val_acc: 0.9771
Epoch 11/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.1595 - acc: 0.9583 -
val_loss: 0.0822 - val_acc: 0.9770
Epoch 12/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.1523 - acc: 0.9610 -
val_loss: 0.0862 - val_acc: 0.9770
Epoch 13/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.1489 - acc: 0.9623 -
val_loss: 0.0811 - val_acc: 0.9782
Epoch 14/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.1390 - acc: 0.9642 -
val_loss: 0.0761 - val_acc: 0.9790
Epoch 15/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.1357 - acc: 0.9656 -
val_loss: 0.0768 - val_acc: 0.9792
Epoch 16/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.1304 - acc: 0.9655 -
val_loss: 0.0812 - val_acc: 0.9797
Epoch 17/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.1273 - acc: 0.9675 -
val_loss: 0.0804 - val_acc: 0.9790
Epoch 18/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.1242 - acc: 0.9682 -
val_loss: 0.0750 - val_acc: 0.9789
Epoch 19/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.1230 - acc: 0.9686 -
val_loss: 0.0763 - val_acc: 0.9791
Epoch 20/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.1196 - acc: 0.9686 -
val_loss: 0.0754 - val_acc: 0.9814
```

In [28]:

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

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

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

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

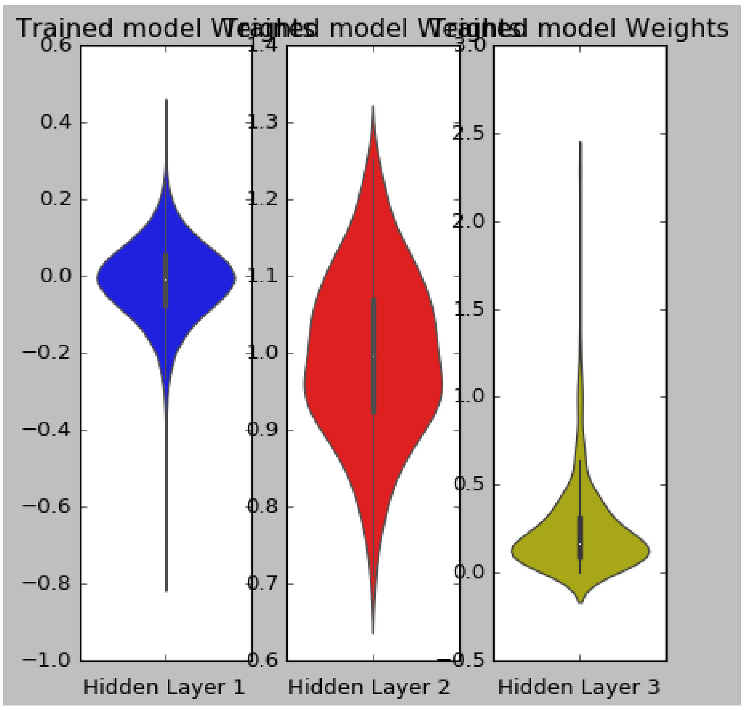
```
Test score: 0.07536343550827004
Test accuracy: 0.9814
```

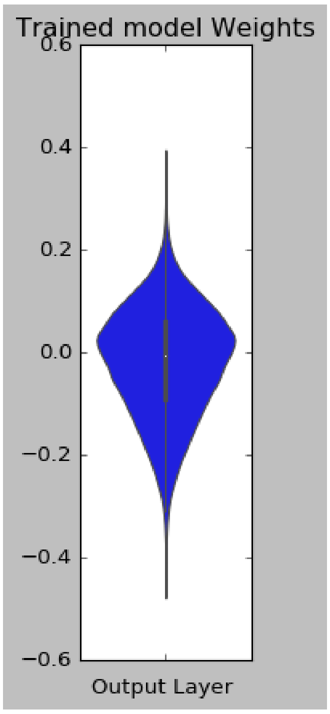




In [30]:

```
w_after = model_relu.get_weights()

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


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

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

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

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





MLP + RELU activation + adam optimizer + 5 hidden layer

In [22]:

```
# Multilayer perceptron

model_relu = Sequential()
model_relu.add(Dense(684, activation='relu', input_shape=(input_dim,),kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(Dense(426, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.062
, seed=None)))
model_relu.add(Dense(256, activation='relu',kernel_initializer=RandomNormal(mean=0.0, stddev=0.062,
seed=None)))
model_relu.add(Dense(128, activation='relu',kernel_initializer=RandomNormal(mean=0.0, stddev=0.062,
seed=None)))
model_relu.add(Dense(64, activation='relu',kernel_initializer=RandomNormal(mean=0.0, stddev=0.062,
seed=None)))
model_relu.add(Dense(output_dim, activation='softmax'))

model_relu.summary()

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

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 684)               536940
_________________________________________________________________
dense_6 (Dense)              (None, 426)               291810
_________________________________________________________________
dense_7 (Dense)              (None, 256)               109312
_________________________________________________________________
dense_8 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_9 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_10 (Dense)             (None, 10)                650
=================================================================
Total params: 979,864
Trainable params: 979,864
Non-trainable params: 0
_________________________________________________________________
```

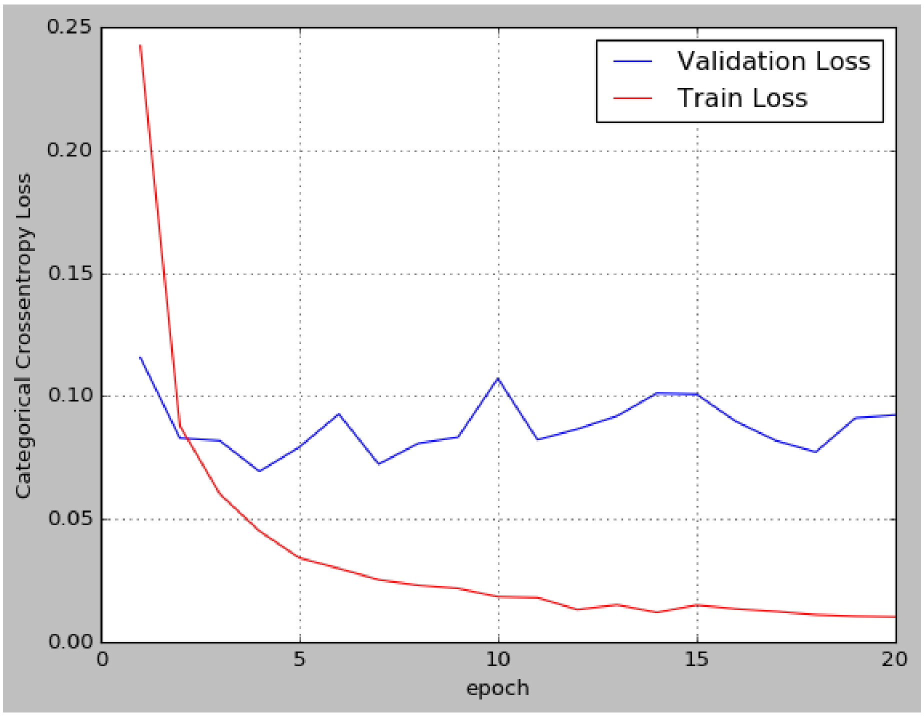
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 15s 249us/step - loss: 0.2423 - acc: 0.9259 - val_l
oss: 0.1156 - val_acc: 0.9628
Epoch 2/20
60000/60000 [==============================] - 14s 238us/step - loss: 0.0878 - acc: 0.9731 - val_l
oss: 0.0829 - val_acc: 0.9745
Epoch 3/20
60000/60000 [==============================] - 14s 238us/step - loss: 0.0602 - acc: 0.9810 - val_l
oss: 0.0819 - val_acc: 0.9748
Epoch 4/20
60000/60000 [==============================] - 14s 236us/step - loss: 0.0451 - acc: 0.9857 - val_l
oss: 0.0693 - val_acc: 0.9806
Epoch 5/20
60000/60000 [==============================] - 15s 242us/step - loss: 0.0342 - acc: 0.9892 - val_l
oss: 0.0792 - val_acc: 0.9771
Epoch 6/20
60000/60000 [==============================] - 14s 242us/step - loss: 0.0299 - acc: 0.9908 - val_l
oss: 0.0927 - val_acc: 0.9740
Epoch 7/20
60000/60000 [==============================] - 14s 240us/step - loss: 0.0253 - acc: 0.9922 - val_l
oss: 0.0723 - val_acc: 0.9800
Epoch 8/20
60000/60000 [==============================] - 15s 243us/step - loss: 0.0231 - acc: 0.9930 - val_l
oss: 0.0807 - val_acc: 0.9807
Epoch 9/20
60000/60000 [==============================] - 15s 242us/step - loss: 0.0218 - acc: 0.9929 - val_l
oss: 0.0833 - val_acc: 0.9799
Epoch 10/20
60000/60000 [==============================] - 14s 241us/step - loss: 0.0184 - acc: 0.9942 - val_l
oss: 0.1071 - val_acc: 0.9752
Epoch 11/20
60000/60000 [==============================] - 14s 240us/step - loss: 0.0180 - acc: 0.9945 - val_l
oss: 0.0822 - val_acc: 0.9793
Epoch 12/20
60000/60000 [==============================] - 14s 238us/step - loss: 0.0132 - acc: 0.9960 - val_l
oss: 0.0866 - val_acc: 0.9809
Epoch 13/20
60000/60000 [==============================] - 14s 239us/step - loss: 0.0151 - acc: 0.9955 - val_l
oss: 0.0919 - val_acc: 0.9790
Epoch 14/20
60000/60000 [==============================] - 14s 237us/step - loss: 0.0120 - acc: 0.9966 - val_l
oss: 0.1011 - val_acc: 0.9787
Epoch 15/20
60000/60000 [==============================] - 14s 241us/step - loss: 0.0150 - acc: 0.9954 - val_l
oss: 0.1007 - val_acc: 0.9776
Epoch 16/20
60000/60000 [==============================] - 14s 238us/step - loss: 0.0135 - acc: 0.9957 - val_l
oss: 0.0896 - val_acc: 0.9826
Epoch 17/20
60000/60000 [==============================] - 14s 240us/step - loss: 0.0124 - acc: 0.9968 - val_l
oss: 0.0818 - val_acc: 0.9825
Epoch 18/20
60000/60000 [==============================] - 14s 241us/step - loss: 0.0111 - acc: 0.9970 - val_l
oss: 0.0772 - val_acc: 0.9818
Epoch 19/20
60000/60000 [==============================] - 14s 241us/step - loss: 0.0105 - acc: 0.9971 - val_l
oss: 0.0911 - val_acc: 0.9799
Epoch 20/20
60000/60000 [==============================] - 14s 241us/step - loss: 0.0103 - acc: 0.9971 - val_l
oss: 0.0923 - val_acc: 0.9813
```

In [23]:

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

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

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

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

```
plt_dynamic(x, vy, ty, ax)
```

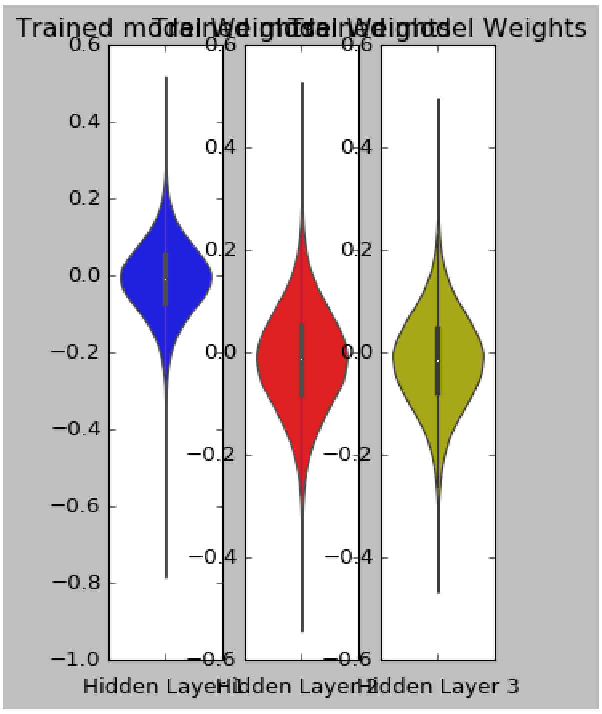
```
Test score: 0.09227842169566275
Test accuracy: 0.9813
```

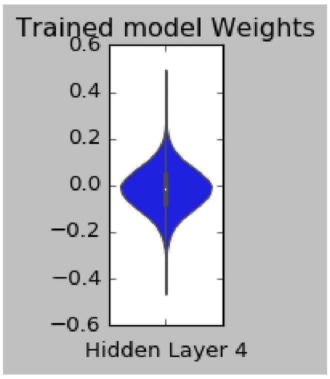


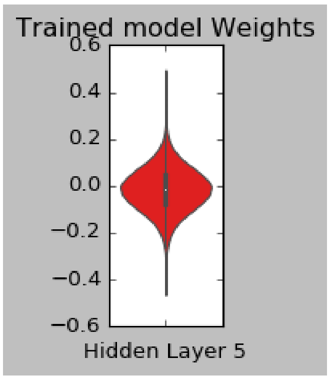
In [26]:

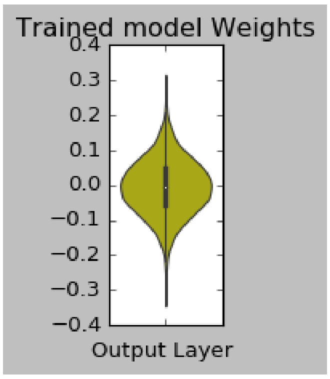
```
w_after = model_relu.get_weights()

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


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

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

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

plt.subplot(2, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w,color='b')
plt.xlabel('Hidden Layer 4')
plt.show()

plt.subplot(2, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w,color='r')
plt.xlabel('Hidden Layer 5')
plt.show()

plt.subplot(2, 6, 6)
plt.title("Trained model Weights")
```

```
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```









MLP + RELU activation + adam optimizer + 5 hidden layer + Batch normalization and Drop out

In [27]:

```
# Multilayer perceptron

model_relu = Sequential()
model_relu.add(Dense(684, activation='relu', input_shape=(input_dim,),kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

model_relu.add(Dense(426, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.062
, seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

model_relu.add(Dense(256, activation='relu',kernel_initializer=RandomNormal(mean=0.0, stddev=0.062,
seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

model_relu.add(Dense(128, activation='relu',kernel_initializer=RandomNormal(mean=0.0, stddev=0.062,
seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

model_relu.add(Dense(64, activation='relu',kernel_initializer=RandomNormal(mean=0.0, stddev=0.062,
seed=None)))
model_relu.add(BatchNormalization())
model_relu.add(Dropout(0.5))

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

model_relu.summary()

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

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

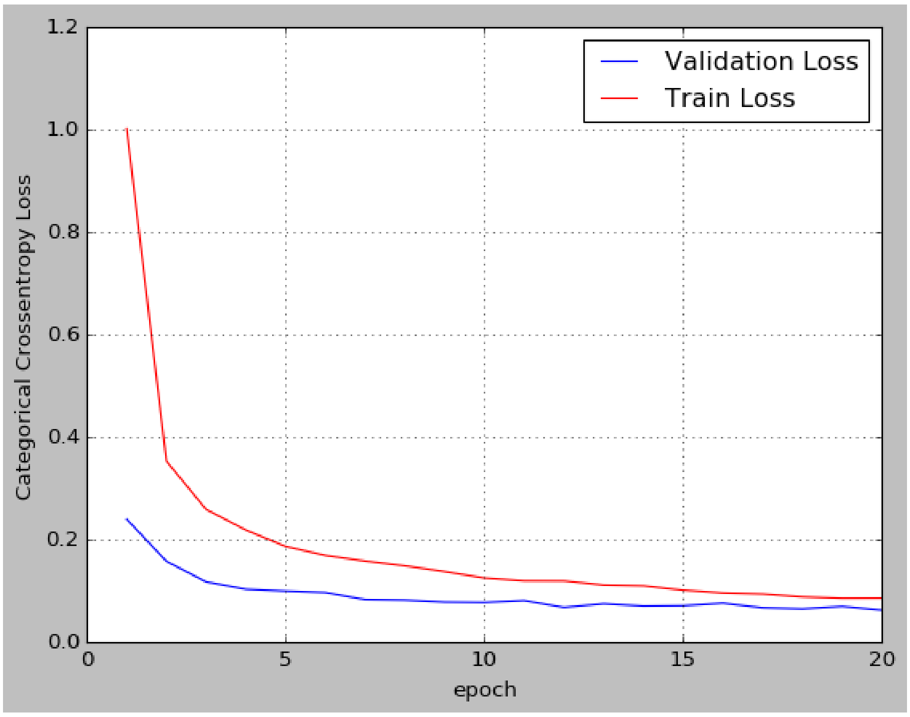
```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:3445: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future
version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_11 (Dense)             (None, 684)               536940
_________________________________________________________________
batch_normalization_1 (Batch (None, 684)               2736
_________________________________________________________________
dropout_1 (Dropout)          (None, 684)               0
_________________________________________________________________
dense_12 (Dense)             (None, 426)               291810
_________________________________________________________________
batch_normalization_2 (Batch (None, 426)               1704
_________________________________________________________________
dropout_2 (Dropout)          (None, 426)               0
_________________________________________________________________
dense_13 (Dense)             (None, 256)               109312
_________________________________________________________________
batch_normalization_3 (Batch (None, 256)               1024
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_14 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
```

```
_________________________________________________________________
dense_15 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_5 (Batch (None, 64)                256
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_16 (Dense)             (None, 10)                650
=================================================================
Total params: 986,096
Trainable params: 982,980
Non-trainable params: 3,116
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 19s 319us/step - loss: 1.0018 - acc: 0.6862 - val_l
oss: 0.2397 - val_acc: 0.9278
Epoch 2/20
60000/60000 [==============================] - 18s 293us/step - loss: 0.3527 - acc: 0.9032 - val_l
oss: 0.1574 - val_acc: 0.9541
Epoch 3/20
60000/60000 [==============================] - 18s 296us/step - loss: 0.2587 - acc: 0.9309 - val_l
oss: 0.1173 - val_acc: 0.9672
Epoch 4/20
60000/60000 [==============================] - 18s 295us/step - loss: 0.2184 - acc: 0.9431 - val_l
oss: 0.1031 - val_acc: 0.9708
Epoch 5/20
60000/60000 [==============================] - 18s 292us/step - loss: 0.1864 - acc: 0.9505 - val_l
oss: 0.0994 - val_acc: 0.9732
Epoch 6/20
60000/60000 [==============================] - 18s 292us/step - loss: 0.1689 - acc: 0.9556 - val_l
oss: 0.0965 - val_acc: 0.9748
Epoch 7/20
60000/60000 [==============================] - 18s 293us/step - loss: 0.1579 - acc: 0.9586 - val_l
oss: 0.0828 - val_acc: 0.9788
Epoch 8/20
60000/60000 [==============================] - 18s 292us/step - loss: 0.1486 - acc: 0.9619 - val_l
oss: 0.0818 - val_acc: 0.9777
Epoch 9/20
60000/60000 [==============================] - 18s 295us/step - loss: 0.1373 - acc: 0.9646 - val_l
oss: 0.0780 - val_acc: 0.9785
Epoch 10/20
60000/60000 [==============================] - 18s 294us/step - loss: 0.1248 - acc: 0.9674 - val_l
oss: 0.0773 - val_acc: 0.9802
Epoch 11/20
60000/60000 [==============================] - 17s 291us/step - loss: 0.1194 - acc: 0.9688 - val_l
oss: 0.0809 - val_acc: 0.9779
Epoch 12/20
60000/60000 [==============================] - 18s 292us/step - loss: 0.1192 - acc: 0.9686 - val_l
oss: 0.0676 - val_acc: 0.9816
Epoch 13/20
60000/60000 [==============================] - 17s 290us/step - loss: 0.1112 - acc: 0.9709 - val_l
oss: 0.0748 - val_acc: 0.9804
Epoch 14/20
60000/60000 [==============================] - 18s 294us/step - loss: 0.1096 - acc: 0.9704 - val_l
oss: 0.0703 - val_acc: 0.9815
Epoch 15/20
60000/60000 [==============================] - 18s 296us/step - loss: 0.1012 - acc: 0.9733 - val_l
oss: 0.0710 - val_acc: 0.9821
Epoch 16/20
60000/60000 [==============================] - 18s 296us/step - loss: 0.0958 - acc: 0.9747 - val_l
oss: 0.0759 - val_acc: 0.9799
Epoch 17/20
60000/60000 [==============================] - 18s 295us/step - loss: 0.0937 - acc: 0.9760 - val_l
oss: 0.0668 - val_acc: 0.9821
Epoch 18/20
60000/60000 [==============================] - 18s 294us/step - loss: 0.0882 - acc: 0.9766 - val_l
oss: 0.0648 - val_acc: 0.9834
Epoch 19/20
60000/60000 [==============================] - 18s 295us/step - loss: 0.0855 - acc: 0.9778 - val_l
oss: 0.0694 - val_acc: 0.9823
Epoch 20/20
60000/60000 [==============================] - 17s 291us/step - loss: 0.0857 - acc: 0.9774 - val_l
oss: 0.0623 - val_acc: 0.9834
```

In [28]:

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

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

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

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

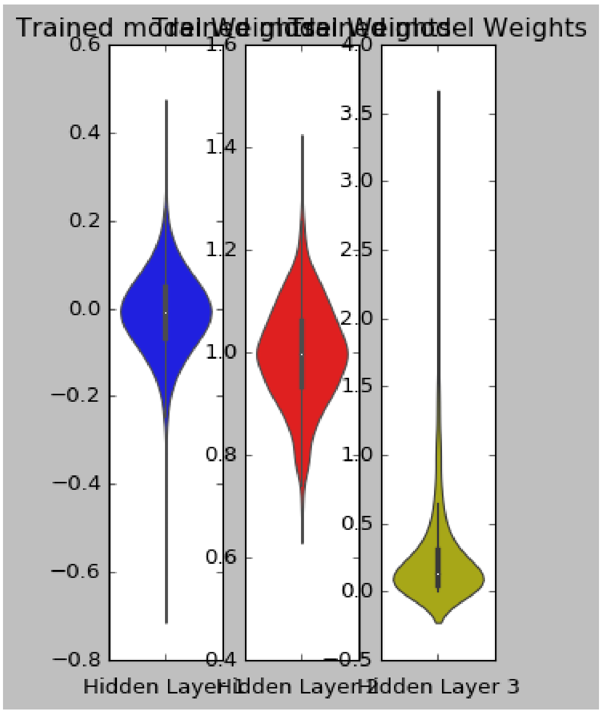
```
Test score: 0.06228468422386795
Test accuracy: 0.9834
```

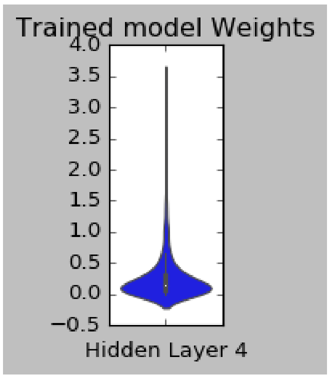


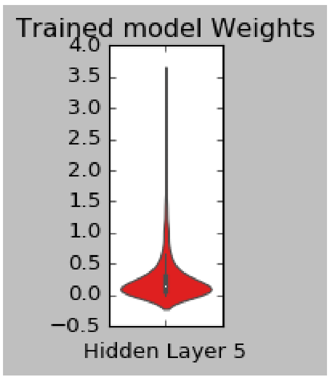
In [29]:

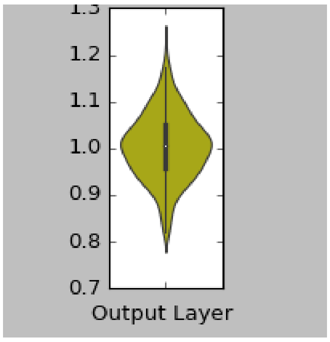
```
w_after = model_relu.get_weights()

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


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

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

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

plt.subplot(2, 6, 4)
```

```
plt.subplot(2, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w,color='b')
plt.xlabel('Hidden Layer 4')
plt.show()

plt.subplot(2, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h3_w,color='r')
plt.xlabel('Hidden Layer 5')
plt.show()

plt.subplot(2, 6, 6)
plt.title("Trained model Weights")
ax = sns.violinplot(y=out_w,color='y')
plt.xlabel('Output Layer ')
plt.show()
```









In [7]:

```
#code copied from -http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["No. of Hidden layers", "Normalization and dropout applied?", "Accuracy"]
x.add_row([2, "No", 0.9818])
x.add_row([2, "Yes", 0.9831])
x.add_row([3, "No", 0.9796])
x.add_row([3, "Yes", 0.9814])
x.add_row([5, "No", 0.9813])
x.add_row([5, "Yes", 0.9834])
print(x)
```

```
+----------------------+------------------------------------+----------+
| No. of Hidden layers | Normalization and dropout applied? | Accuracy |
+----------------------+------------------------------------+----------+
|          2           |                 No                 |  0.9818  |
|          2           |                Yes                 |  0.9831  |
|          3           |                 No                 |  0.9796  |
|          3           |                Yes                 |  0.9814  |
|          5           |                 No                 |  0.9813  |
|          5           |                Yes                 |  0.9834  |
+----------------------+------------------------------------+----------+
```

The best MLP architecture for MNIST Data Set is when we applied 5 hidden layers with Normalization and Drop Out applied and accuracy obtained is 98.34 percent