Different_MLP_architectures_on_MNIST_dataset

January 21, 2019

In [0]:
```
# if you keras is not using tensorflow as backend set "KERAS_BACKEND=tensorflow" use t
from keras.utils import np_utils
import pandas as pd
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
import keras
keras.backend.backend()
```

```
Using TensorFlow backend.
```

Out[0]: 'tensorflow'

In [0]:
```
%matplotlib inline
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()
```

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

```
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]:
```python
# if you observe the input shape its 3 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 [0]:
```python
# 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 (%
print("Number of training examples :", X_test.shape[0], "and each image is of shape (%
```

```
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 [0]:
```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]:

In [0]:
```
# 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 [0]:
```
# 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.        ]
```

In [0]:
```
# 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.]
```

Softmax classifier

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 constr

# 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_
# bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_r
# 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 activatio

# 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
```

In [0]:
```
# start building a model
model = Sequential()

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

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

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

In [0]:
```
# Before training a model, you need to configure the learning process, which is done v

# It receives three arguments:
# An optimizer. This could be the string identifier of an existing optimizer , https:/
# A loss function. This is the objective that the model will try to minimize., https:/
# A list of metrics. For any classification problem you will want to set this to metri


# Note: when using the categorical_crossentropy loss, your targets should be in catego
# (e.g. if you have 10 classes, the target for each sample should be a 10-dimensional 
# for a 1 at the index corresponding to the class of the sample).

# that is why we converted out labels into vectors

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

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

# fit(self, x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=None, vali
# validation_data=None, shuffle=True, class_weight=None, sample_weight=None, initial_e
# validation_steps=None)

# fit() function Trains the model for a fixed number of epochs (iterations on a datase

# it returns A History object. Its History.history attribute is a record of training l
# metrics values at successive epochs, as well as validation loss values and validatio

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

history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=
```

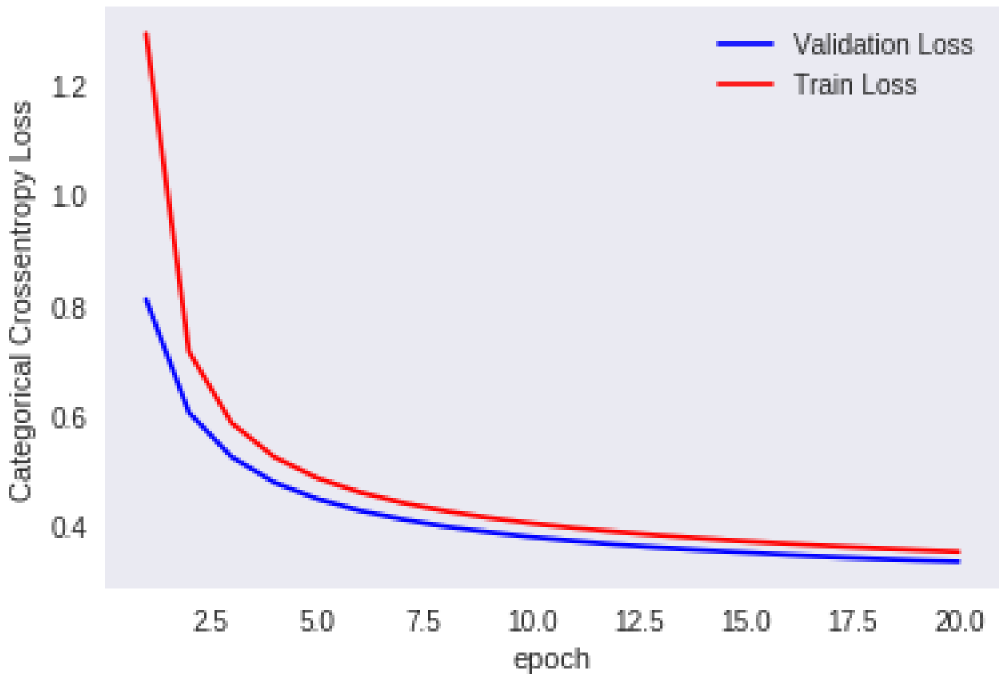
```
Train on 60000 samples, validate on 10000 samples
```

```
Epoch 1/20
60000/60000 [==============================] - 2s 27us/step - loss: 1.2892 - acc: 0.6962 - val_
Epoch 2/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.7151 - acc: 0.8415 - val_
Epoch 3/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.5861 - acc: 0.8599 - val_
Epoch 4/20
60000/60000 [==============================] - 1s 20us/step - loss: 0.5245 - acc: 0.8690 - val_
Epoch 5/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.4870 - acc: 0.8756 - val_
Epoch 6/20
60000/60000 [==============================] - 1s 20us/step - loss: 0.4612 - acc: 0.8800 - val_
Epoch 7/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.4422 - acc: 0.8836 - val_
Epoch 8/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.4273 - acc: 0.8871 - val_
Epoch 9/20
60000/60000 [==============================] - 1s 19us/step - loss: 0.4153 - acc: 0.8892 - val_
Epoch 10/20
60000/60000 [==============================] - 1s 17us/step - loss: 0.4053 - acc: 0.8910 - val_
Epoch 11/20
60000/60000 [==============================] - 1s 17us/step - loss: 0.3969 - acc: 0.8932 - val_
Epoch 12/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.3897 - acc: 0.8946 - val_
Epoch 13/20
60000/60000 [==============================] - 1s 17us/step - loss: 0.3833 - acc: 0.8960 - val_
Epoch 14/20
60000/60000 [==============================] - 1s 17us/step - loss: 0.3777 - acc: 0.8971 - val_
Epoch 15/20
60000/60000 [==============================] - 1s 17us/step - loss: 0.3727 - acc: 0.8981 - val_
Epoch 16/20
60000/60000 [==============================] - 1s 17us/step - loss: 0.3682 - acc: 0.8991 - val_
Epoch 17/20
60000/60000 [==============================] - 1s 17us/step - loss: 0.3641 - acc: 0.9000 - val_
Epoch 18/20
60000/60000 [==============================] - 1s 17us/step - loss: 0.3604 - acc: 0.9007 - val_
Epoch 19/20
60000/60000 [==============================] - 1s 17us/step - loss: 0.3569 - acc: 0.9017 - val_
Epoch 20/20
60000/60000 [==============================] - 1s 17us/step - loss: 0.3538 - acc: 0.9021 - val_
```

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

# 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 e

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

```
Test score: 0.33589279960989954
Test accuracy: 0.9093
```



Observations 1. We did not use any hidden layer and have only input and output layer and got accuracy ~90% which is best.

1 1. MLP + ReLU + ADAM with 2 hidden layers

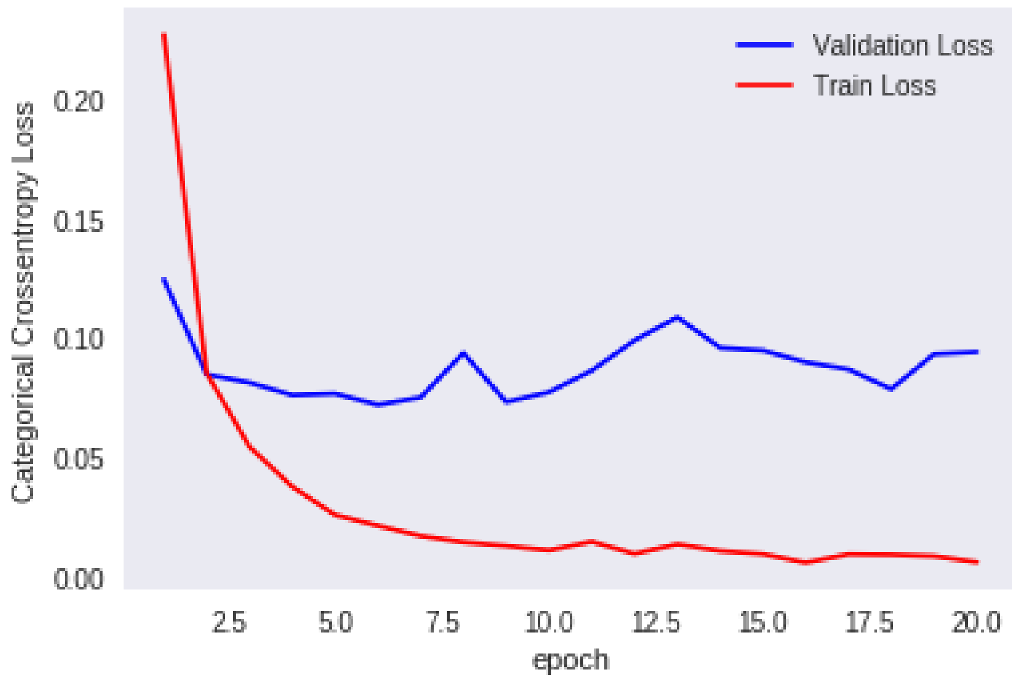
In [0]:
```
model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initiali
model_relu.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ver
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_3 (Dense)              (None, 128)               65664
_________________________________________________________________
dense_4 (Dense)              (None, 10)                1290
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 108us/step - loss: 0.2267 - acc: 0.9321 - va
Epoch 2/20
60000/60000 [==============================] - 6s 104us/step - loss: 0.0849 - acc: 0.9751 - va
Epoch 3/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.0540 - acc: 0.9832 - va
Epoch 4/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.0374 - acc: 0.9884 - va
Epoch 5/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.0255 - acc: 0.9921 - va
Epoch 6/20
60000/60000 [==============================] - 6s 100us/step - loss: 0.0209 - acc: 0.9931 - va
Epoch 7/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.0166 - acc: 0.9948 - va
Epoch 8/20
60000/60000 [==============================] - 6s 103us/step - loss: 0.0140 - acc: 0.9951 - va
Epoch 9/20
60000/60000 [==============================] - 6s 101us/step - loss: 0.0125 - acc: 0.9960 - va
Epoch 10/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0106 - acc: 0.9970 - val_
```

```
Epoch 11/20
60000/60000 [==============================] - 6s 100us/step - loss: 0.0142 - acc: 0.9952 - val
Epoch 12/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.0092 - acc: 0.9970 - val
Epoch 13/20
60000/60000 [==============================] - 6s 100us/step - loss: 0.0132 - acc: 0.9960 - val
Epoch 14/20
60000/60000 [==============================] - 6s 100us/step - loss: 0.0103 - acc: 0.9963 - val
Epoch 15/20
60000/60000 [==============================] - 6s 101us/step - loss: 0.0090 - acc: 0.9972 - val
Epoch 16/20
60000/60000 [==============================] - 6s 100us/step - loss: 0.0055 - acc: 0.9982 - val
Epoch 17/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0090 - acc: 0.9971 - val_
Epoch 18/20
60000/60000 [==============================] - 6s 101us/step - loss: 0.0088 - acc: 0.9972 - val
Epoch 19/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.0082 - acc: 0.9975 - val_
Epoch 20/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0057 - acc: 0.9983 - val_
```

In [0]:
```
score_relu2_adam = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score_relu2_adam[0])
print('Test accuracy:', score_relu2_adam[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, v

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

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


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

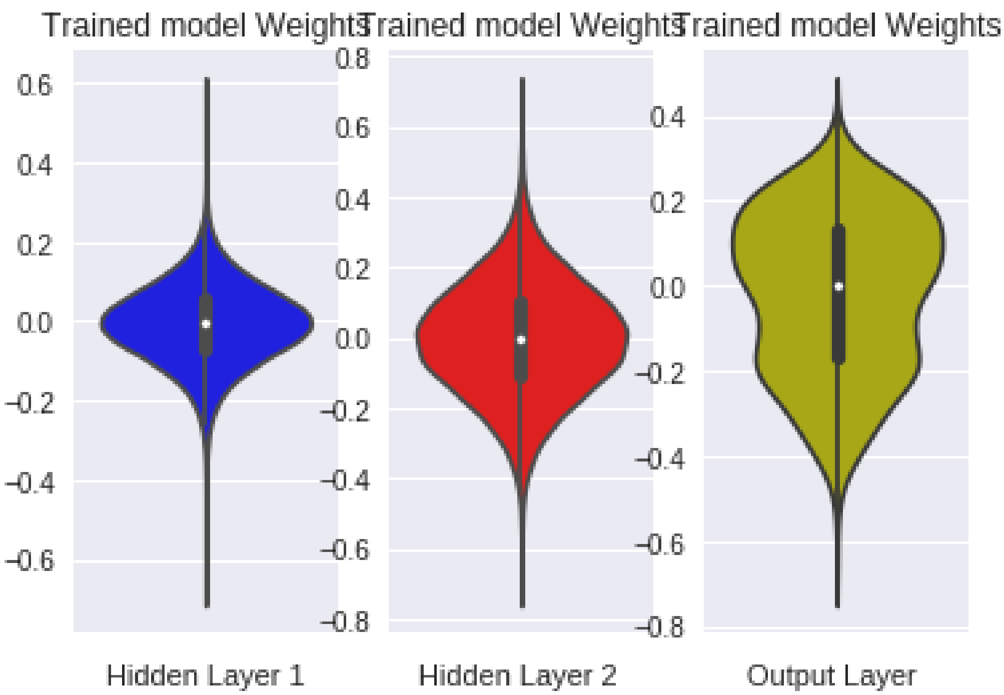
```
Test score: 0.0938020015134396
Test accuracy: 0.9807
```




```
In [0]: 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()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is
  violin_data = remove_na(group_data)
```



2 MLP + ReLU + ADAM with 2 hidden layers

```
In [0]: model_relu = Sequential()

        model_relu.add(Dense(672, activation='relu', input_shape=(input_dim,), kernel_initializ

        model_relu.add(Dense(325, activation='relu', kernel_initializer=RandomNormal(mean=0.0,

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

        print(model_relu.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
dense_5 (Dense)              (None, 672)               527520
_________________________________________________________________
dense_6 (Dense)              (None, 325)               218725
_________________________________________________________________
dense_7 (Dense)              (None, 10)                3260
=================================================================
Total params: 749,505
Trainable params: 749,505
Non-trainable params: 0
_________________________________________________________________
None
```

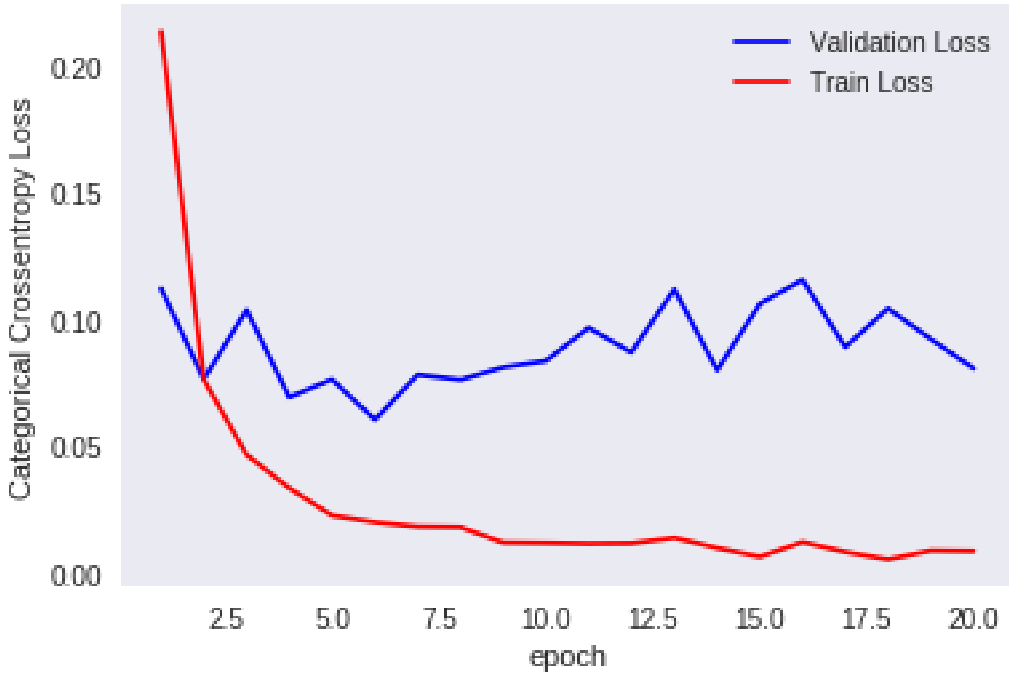
```
In [0]: model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

        history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verb
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s 165us/step - loss: 0.2132 - acc: 0.9354 - va
Epoch 2/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.0761 - acc: 0.9764 - val
Epoch 3/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0465 - acc: 0.9849 - val
Epoch 4/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0335 - acc: 0.9894 - val
Epoch 5/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0227 - acc: 0.9928 - val
Epoch 6/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0200 - acc: 0.9929 - val
Epoch 7/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0183 - acc: 0.9935 - val
Epoch 8/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.0181 - acc: 0.9941 - val
Epoch 9/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0120 - acc: 0.9959 - val
Epoch 10/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.0118 - acc: 0.9961 - val
Epoch 11/20
60000/60000 [==============================] - 9s 158us/step - loss: 0.0115 - acc: 0.9962 - val
Epoch 12/20
60000/60000 [==============================] - 10s 159us/step - loss: 0.0117 - acc: 0.9961 - va
Epoch 13/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.0139 - acc: 0.9956 - val
Epoch 14/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0099 - acc: 0.9966 - val
Epoch 15/20
```

```
60000/60000 [==============================] - 9s 155us/step - loss: 0.0064 - acc: 0.9979 - va
Epoch 16/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.0122 - acc: 0.9960 - va
Epoch 17/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0083 - acc: 0.9974 - va
Epoch 18/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.0053 - acc: 0.9983 - v
Epoch 19/20
60000/60000 [==============================] - 9s 158us/step - loss: 0.0088 - acc: 0.9974 - va
Epoch 20/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0086 - acc: 0.9974 - va
```

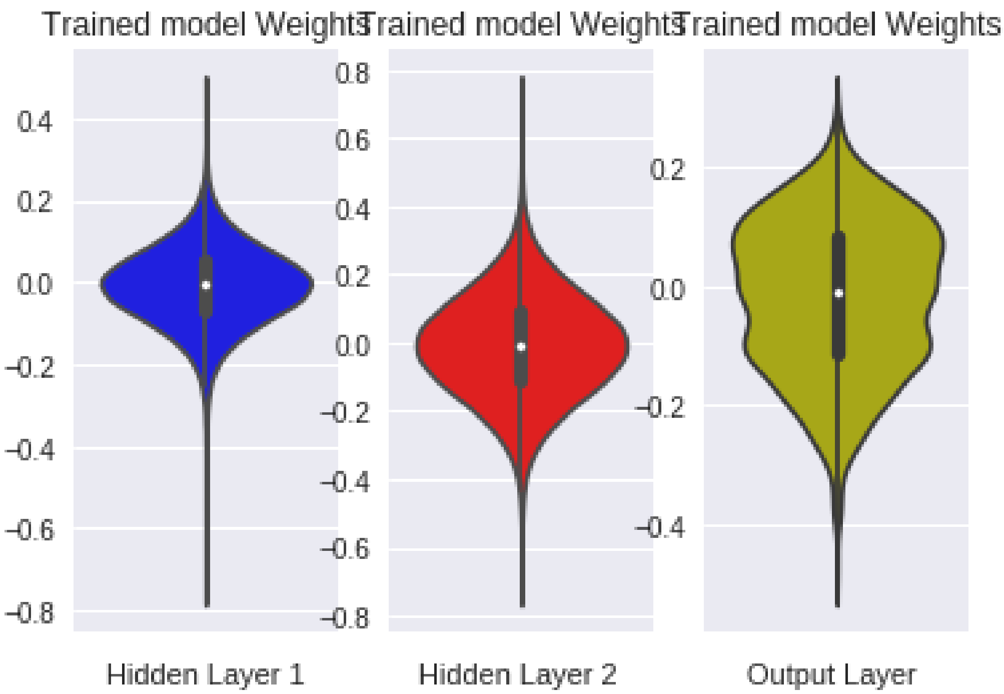
In [0]:
```
score_relu2_adam_diff = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score_relu2_adam_diff[0])
print('Test accuracy:', score_relu2_adam_diff[1])

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, v

# 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 e


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

```
Test score: 0.08052913746033337
Test accuracy: 0.9845
```




```
In [0]: 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()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is
  violin_data = remove_na(group_data)
```



3 MLP + ReLU + ADAM with 3 hidden layers

```
In [0]: model_relu = Sequential()

        model_relu.add(Dense(532, activation='relu', input_shape=(input_dim,), kernel_initiali

        model_relu.add(Dense(291, activation='relu', kernel_initializer=RandomNormal(mean=0.0,

        model_relu.add(Dense(187, activation='relu', kernel_initializer=RandomNormal(mean=0.0,

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

        print(model_relu.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 532)               417620
```

```
_________________________________________________________________
dense_9 (Dense)              (None, 291)               155103
_________________________________________________________________
dense_10 (Dense)             (None, 187)               54604
_________________________________________________________________
dense_11 (Dense)             (None, 10)                1880
=================================================================
Total params: 629,207
Trainable params: 629,207
Non-trainable params: 0
_________________________________________________________________
None
```

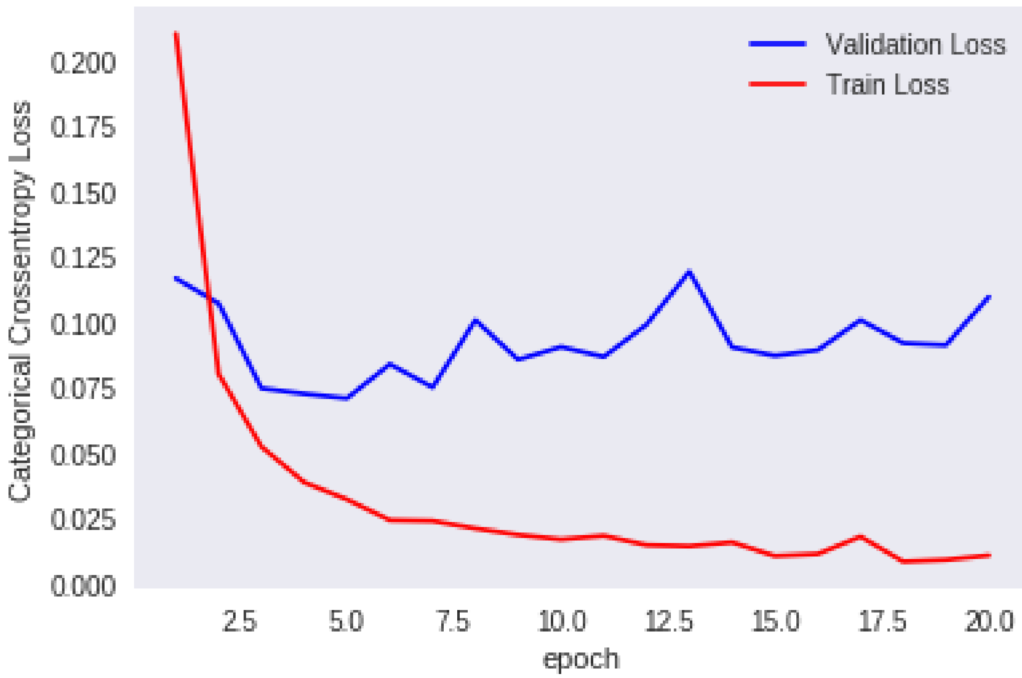
In [0]:
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accurac

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verb
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.2101 - acc: 0.9357 - val
Epoch 2/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0799 - acc: 0.9754 - val
Epoch 3/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.0521 - acc: 0.9835 - val
Epoch 4/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.0385 - acc: 0.9880 - val
Epoch 5/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.0320 - acc: 0.9888 - val
Epoch 6/20
60000/60000 [==============================] - 8s 140us/step - loss: 0.0241 - acc: 0.9918 - val
Epoch 7/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0238 - acc: 0.9922 - val
Epoch 8/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0209 - acc: 0.9935 - val
Epoch 9/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0184 - acc: 0.9941 - val
Epoch 10/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.0167 - acc: 0.9948 - val
Epoch 11/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.0181 - acc: 0.9938 - val
Epoch 12/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.0145 - acc: 0.9955 - val
Epoch 13/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0141 - acc: 0.9956 - val
Epoch 14/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0156 - acc: 0.9951 - val
Epoch 15/20
```

```
60000/60000 [==============================] - 9s 147us/step - loss: 0.0103 - acc: 0.9967 - val
Epoch 16/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0111 - acc: 0.9963 - val
Epoch 17/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0178 - acc: 0.9944 - val
Epoch 18/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0082 - acc: 0.9974 - val
Epoch 19/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0089 - acc: 0.9971 - val
Epoch 20/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0106 - acc: 0.9969 - val
```

In [0]:
```
score_relu3_adam = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score_relu3_adam[0])
print('Test accuracy:', score_relu3_adam[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, v

# 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 e


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

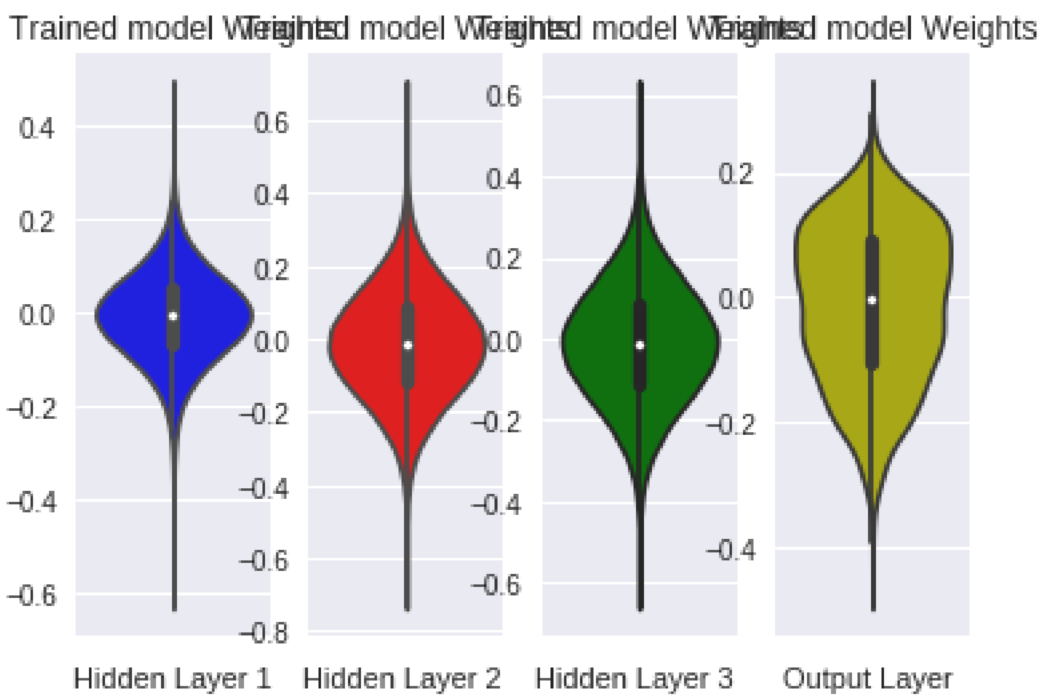
```
Test score: 0.10949685554472845
Test accuracy: 0.9773
```



In [0]:
```
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='g')
plt.xlabel('Hidden Layer 3 ')
```

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

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is
  violin_data = remove_na(group_data)
```



4 MLP + ReLU + ADAM with 5 hidden layers

```
In [0]: model_relu = Sequential()

        model_relu.add(Dense(532, activation='relu', input_shape=(input_dim,), kernel_initiali

        model_relu.add(Dense(443, activation='relu', kernel_initializer=RandomNormal(mean=0.0,

        model_relu.add(Dense(291, activation='relu', kernel_initializer=RandomNormal(mean=0.0,

        model_relu.add(Dense(167, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
```

```
model_relu.add(Dense(125, activation='relu', kernel_initializer=RandomNormal(mean=0.0,

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

print(model_relu.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 532)               417620
_________________________________________________________________
dense_13 (Dense)             (None, 443)               236119
_________________________________________________________________
dense_14 (Dense)             (None, 291)               129204
_________________________________________________________________
dense_15 (Dense)             (None, 167)               48764
_________________________________________________________________
dense_16 (Dense)             (None, 125)               21000
_________________________________________________________________
dense_17 (Dense)             (None, 10)                1260
=================================================================
Total params: 853,967
Trainable params: 853,967
Non-trainable params: 0
_________________________________________________________________
None
```

In [0]:
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verb
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 12s 208us/step - loss: 0.2453 - acc: 0.9270 - va
Epoch 2/20
60000/60000 [==============================] - 12s 200us/step - loss: 0.0974 - acc: 0.9696 - va
Epoch 3/20
60000/60000 [==============================] - 12s 199us/step - loss: 0.0683 - acc: 0.9781 - va
Epoch 4/20
60000/60000 [==============================] - 12s 203us/step - loss: 0.0529 - acc: 0.9826 - va
Epoch 5/20
60000/60000 [==============================] - 12s 202us/step - loss: 0.0419 - acc: 0.9868 - va
Epoch 6/20
60000/60000 [==============================] - 12s 204us/step - loss: 0.0391 - acc: 0.9875 - va
Epoch 7/20
60000/60000 [==============================] - 12s 201us/step - loss: 0.0338 - acc: 0.9890 - va
```

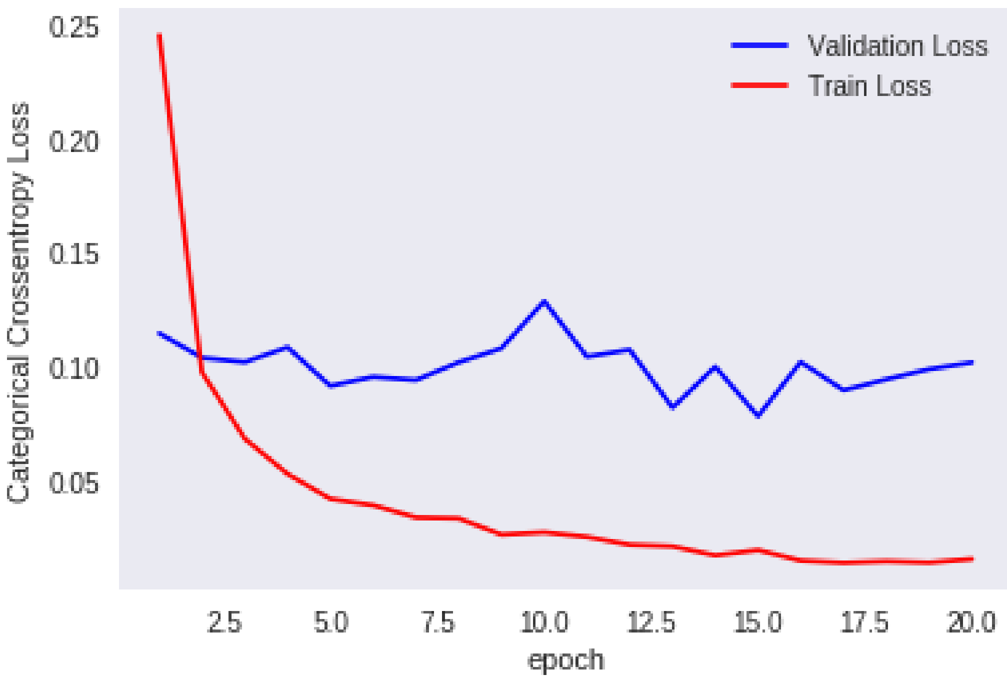
```
Epoch 8/20
60000/60000 [==============================] - 12s 204us/step - loss: 0.0334 - acc: 0.9895 - va
Epoch 9/20
60000/60000 [==============================] - 13s 210us/step - loss: 0.0263 - acc: 0.9920 - va
Epoch 10/20
60000/60000 [==============================] - 12s 205us/step - loss: 0.0273 - acc: 0.9918 - va
Epoch 11/20
60000/60000 [==============================] - 12s 203us/step - loss: 0.0253 - acc: 0.9921 - va
Epoch 12/20
60000/60000 [==============================] - 12s 203us/step - loss: 0.0219 - acc: 0.9928 - va
Epoch 13/20
60000/60000 [==============================] - 12s 201us/step - loss: 0.0212 - acc: 0.9934 - va
Epoch 14/20
60000/60000 [==============================] - 12s 202us/step - loss: 0.0172 - acc: 0.9946 - va
Epoch 15/20
60000/60000 [==============================] - 12s 205us/step - loss: 0.0195 - acc: 0.9938 - va
Epoch 16/20
60000/60000 [==============================] - 12s 205us/step - loss: 0.0149 - acc: 0.9953 - va
Epoch 17/20
60000/60000 [==============================] - 12s 202us/step - loss: 0.0140 - acc: 0.9957 - va
Epoch 18/20
60000/60000 [==============================] - 12s 200us/step - loss: 0.0146 - acc: 0.9957 - va
Epoch 19/20
60000/60000 [==============================] - 12s 197us/step - loss: 0.0141 - acc: 0.9960 - va
Epoch 20/20
60000/60000 [==============================] - 12s 196us/step - loss: 0.0156 - acc: 0.9954 - va
```

In [0]:
```python
score_relu5_adam = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score_relu5_adam[0])
print('Test accuracy:', score_relu5_adam[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, v

# 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 e
```

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

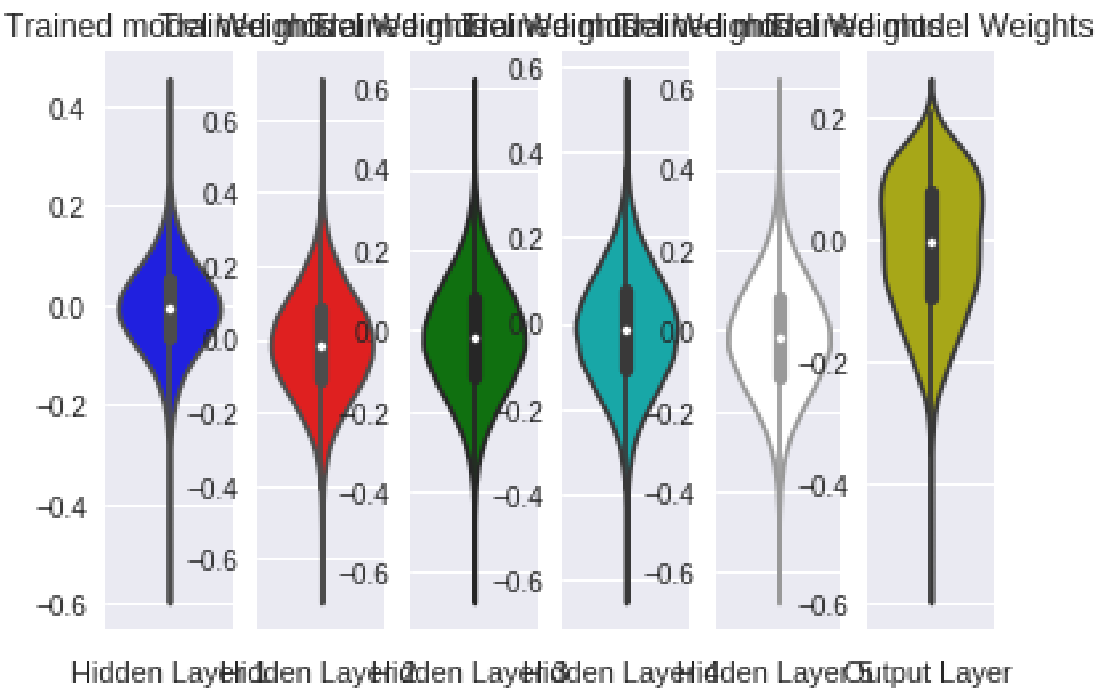
```
Test score: 0.1017726518217185
Test accuracy: 0.9787
```




```
In [0]: w_after = model_relu.get_weights()

        h1_w = w_after[0].flatten().reshape(-1,1)
        h2_w = w_after[2].flatten().reshape(-1,1)
        h3_w = w_after[4].flatten().reshape(-1,1)
        h4_w = w_after[6].flatten().reshape(-1,1)
        h5_w = w_after[4].flatten().reshape(-1,1)
        out_w = w_after[10].flatten().reshape(-1,1)

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

```python
    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='g')
    plt.xlabel('Hidden Layer 3 ')

    plt.subplot(1, 6, 4)
    plt.title("Trained model Weights")
    ax = sns.violinplot(y=h4_w, color='c')
    plt.xlabel('Hidden Layer 4 ')

    plt.subplot(1, 6, 5)
    plt.title("Trained model Weights")
    ax = sns.violinplot(y=h5_w, color='w')
    plt.xlabel('Hidden Layer 5 ')

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

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is
  violin_data = remove_na(group_data)
```



Observations 1. We used MLP with 2, 3 and 5 different layer and got accuracy ~98% but the difference between train loss and validataion loss is slightly high and could so happen that we are overfitting. 2. Weight distributions is not too small and not too large and mean is at 0, which is a good sign of not getting into problem of vanishing or exploding gradient.

5 2. MLP + Batch-Norm + AdamOptimizer with 2 hidden layer

```
In [0]: # Multilayer perceptron

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

        from keras.layers.normalization import BatchNormalization

        model_batch = Sequential()

        model_batch.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initiali
        model_batch.add(BatchNormalization())

        model_batch.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0
        model_batch.add(BatchNormalization())
```

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

model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_18 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048      
_________________________________________________________________
dense_19 (Dense)             (None, 128)               65664     
_________________________________________________________________
batch_normalization_2 (Batch (None, 128)               512       
_________________________________________________________________
dense_20 (Dense)             (None, 10)                1290      
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

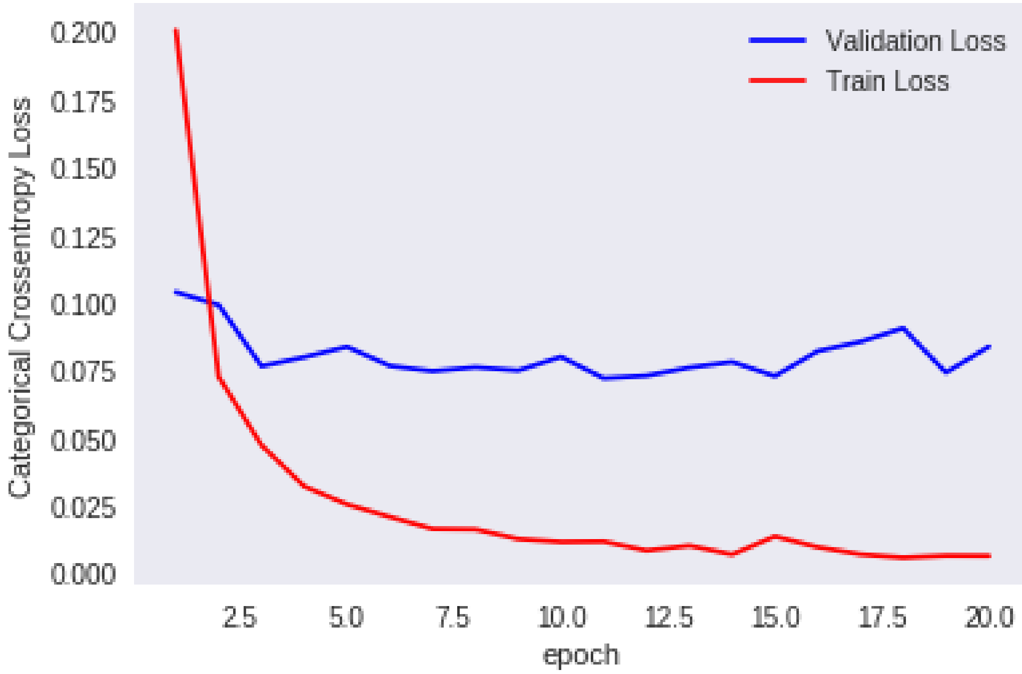
```
In [0]: model_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

        history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.2004 - acc: 0.9403 - val
Epoch 2/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.0724 - acc: 0.9788 - val
Epoch 3/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0470 - acc: 0.9858 - val
Epoch 4/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0319 - acc: 0.9903 - val
Epoch 5/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0252 - acc: 0.9921 - val
Epoch 6/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.0206 - acc: 0.9934 - val
Epoch 7/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0161 - acc: 0.9952 - val
Epoch 8/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0160 - acc: 0.9951 - val
Epoch 9/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0124 - acc: 0.9962 - val
Epoch 10/20
```

```
60000/60000 [==============================] - 8s 125us/step - loss: 0.0113 - acc: 0.9966 - va
Epoch 11/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0115 - acc: 0.9965 - va
Epoch 12/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0081 - acc: 0.9978 - va
Epoch 13/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0099 - acc: 0.9968 - va
Epoch 14/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.0067 - acc: 0.9978 - va
Epoch 15/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0134 - acc: 0.9953 - va
Epoch 16/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0094 - acc: 0.9968 - va
Epoch 17/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0067 - acc: 0.9976 - va
Epoch 18/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0055 - acc: 0.9983 - va
Epoch 19/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.0062 - acc: 0.9981 - va
Epoch 20/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.0062 - acc: 0.9981 - va
```

In [0]:
```python
score_relu_bn2 = model_batch.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score_relu_bn2[0])
print('Test accuracy:', score_relu_bn2[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, v

# 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 e

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

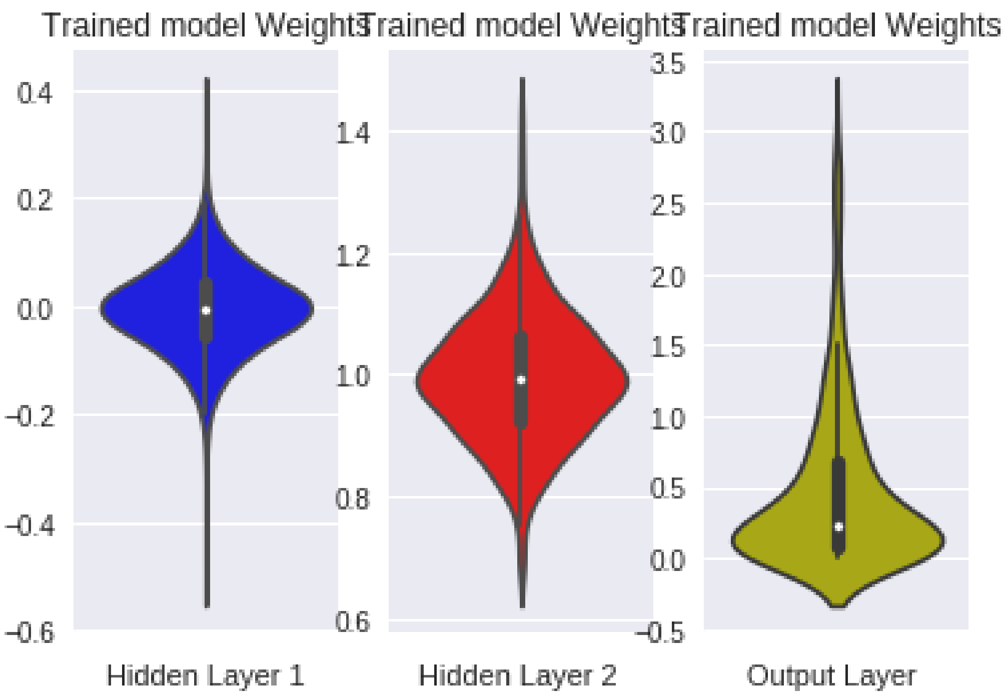
```
Test score: 0.08353556353475215
Test accuracy: 0.9796
```




```
In [0]: w_after = model_batch.get_weights()

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


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

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

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

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

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is
  violin_data = remove_na(group_data)
```



6 MLP + Batch-Norm + AdamOptimizer with 2 hidden layer

```
In [0]: from keras.layers.normalization import BatchNormalization

        model = Sequential()

        model.add(Dense(526, activation = 'relu', input_shape=(input_dim,), kernel_initializer
        model_batch.add(BatchNormalization())

        model.add(Dense(207, activation = 'relu', kernel_initializer = RandomNormal(mean = 0.0
        model.add(BatchNormalization())

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

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_21 (Dense)             (None, 526)               412910
_________________________________________________________________
dense_22 (Dense)             (None, 207)               109089
_________________________________________________________________
batch_normalization_4 (Batch (None, 207)               828
_________________________________________________________________
dense_23 (Dense)             (None, 10)                2080
=================================================================
Total params: 524,907
Trainable params: 524,493
Non-trainable params: 414
_________________________________________________________________
```

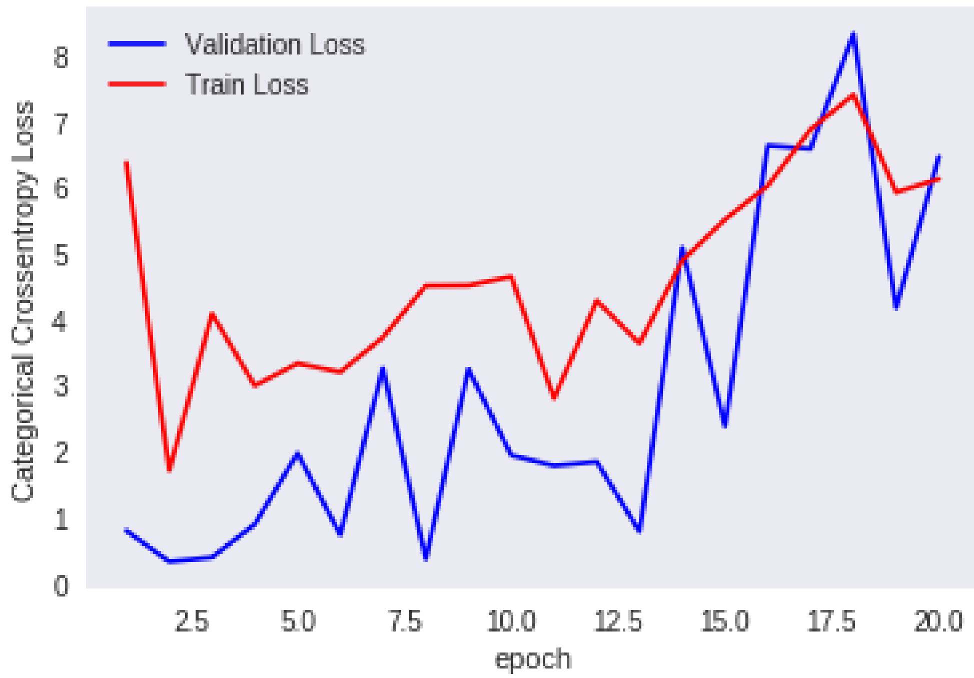
```
In [0]: model_batch.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = [

        history = model_batch.fit(X_train, Y_train, batch_size = batch_size, epochs = nb_epoch
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 151us/step - loss: 6.3680 - acc: 0.9433 - va
Epoch 2/20
60000/60000 [==============================] - 8s 138us/step - loss: 1.6970 - acc: 0.9770 - va
Epoch 3/20
60000/60000 [==============================] - 8s 129us/step - loss: 4.0755 - acc: 0.9641 - va
Epoch 4/20
60000/60000 [==============================] - 8s 127us/step - loss: 2.9868 - acc: 0.9269 - va
Epoch 5/20
60000/60000 [==============================] - 8s 127us/step - loss: 3.3244 - acc: 0.9279 - va
Epoch 6/20
60000/60000 [==============================] - 8s 129us/step - loss: 3.1933 - acc: 0.9081 - va
Epoch 7/20
60000/60000 [==============================] - 8s 129us/step - loss: 3.7201 - acc: 0.9356 - va
Epoch 8/20
60000/60000 [==============================] - 8s 128us/step - loss: 4.5023 - acc: 0.9560 - va
Epoch 9/20
60000/60000 [==============================] - 8s 130us/step - loss: 4.5078 - acc: 0.9594 - va
Epoch 10/20
60000/60000 [==============================] - 8s 130us/step - loss: 4.6388 - acc: 0.9547 - va
Epoch 11/20
60000/60000 [==============================] - 8s 128us/step - loss: 2.7944 - acc: 0.9380 - va
Epoch 12/20
```

```
60000/60000 [==============================] - 8s 128us/step - loss: 4.2741 - acc: 0.9495 - val
Epoch 13/20
60000/60000 [==============================] - 8s 127us/step - loss: 3.6275 - acc: 0.8936 - val
Epoch 14/20
60000/60000 [==============================] - 8s 128us/step - loss: 4.8827 - acc: 0.9140 - val
Epoch 15/20
60000/60000 [==============================] - 8s 127us/step - loss: 5.5093 - acc: 0.8756 - val
Epoch 16/20
60000/60000 [==============================] - 8s 127us/step - loss: 6.0189 - acc: 0.8542 - val
Epoch 17/20
60000/60000 [==============================] - 8s 127us/step - loss: 6.8726 - acc: 0.8445 - val
Epoch 18/20
60000/60000 [==============================] - 8s 128us/step - loss: 7.4012 - acc: 0.8366 - val
Epoch 19/20
60000/60000 [==============================] - 8s 128us/step - loss: 5.9204 - acc: 0.7755 - val
Epoch 20/20
60000/60000 [==============================] - 8s 127us/step - loss: 6.1215 - acc: 0.7644 - val
```

In [0]:
```
score_relu_bn2_diff = model_batch.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score_relu_bn2_diff[0])
print('Test accuracy:', score_relu_bn2_diff[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, v

# 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 e

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

```
Test score: 6.459873273468018
Test accuracy: 0.8002
```



In [0]:
```
w_after = model_batch.get_weights()

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


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

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

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

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is
  violin_data = remove_na(group_data)
```

7 MLP + Batch-Norm + AdamOptimizer with 3 hidden layer

In [0]:
```
from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_batch.add(Dense(345, activation = 'relu', input_shape=(input_dim,), kernel_initi
model_batch.add(BatchNormalization())

model_batch.add(Dense(198, activation = 'relu', kernel_initializer = RandomNormal(mean
model_batch.add(BatchNormalization())

model_batch.add(Dense(57, activation = 'relu', kernel_initializer = RandomNormal(mean =
model_batch.add(BatchNormalization())

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

```
model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_24 (Dense)             (None, 345)               270825
_________________________________________________________________
batch_normalization_5 (Batch (None, 345)               1380
_________________________________________________________________
dense_25 (Dense)             (None, 198)               68508
_________________________________________________________________
batch_normalization_6 (Batch (None, 198)               792
_________________________________________________________________
dense_26 (Dense)             (None, 57)                11343
_________________________________________________________________
batch_normalization_7 (Batch (None, 57)                228
_________________________________________________________________
dense_27 (Dense)             (None, 10)                580
=================================================================
Total params: 353,656
Trainable params: 352,456
Non-trainable params: 1,200
_________________________________________________________________
```

```
In [0]: model_batch.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = [

        history = model_batch.fit(X_train, Y_train, batch_size = batch_size, epochs = nb_epoch
```

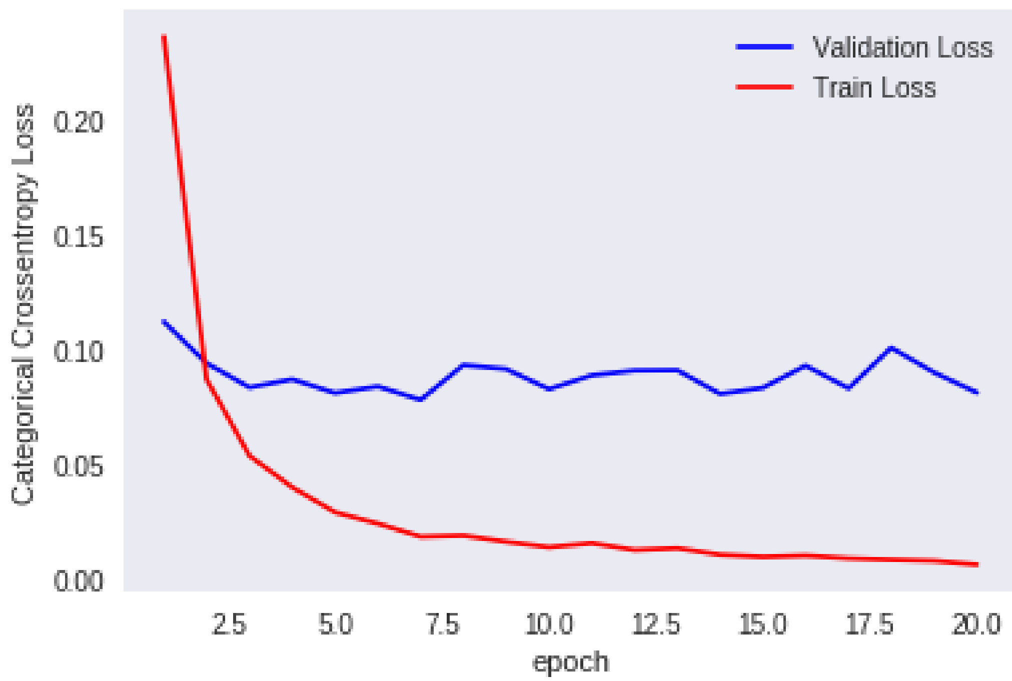
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.2364 - acc: 0.9325 - val
Epoch 2/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.0867 - acc: 0.9745 - val
Epoch 3/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.0532 - acc: 0.9839 - val
Epoch 4/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.0395 - acc: 0.9875 - val
Epoch 5/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.0286 - acc: 0.9913 - val
Epoch 6/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.0237 - acc: 0.9926 - val
Epoch 7/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.0179 - acc: 0.9944 - val
Epoch 8/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.0184 - acc: 0.9940 - val
Epoch 9/20
```

```
60000/60000 [==============================] - 7s 111us/step - loss: 0.0157 - acc: 0.9949 - val
Epoch 10/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.0133 - acc: 0.9957 - val
Epoch 11/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0151 - acc: 0.9951 - val
Epoch 12/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.0121 - acc: 0.9960 - val
Epoch 13/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0129 - acc: 0.9958 - val
Epoch 14/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0100 - acc: 0.9969 - val
Epoch 15/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.0091 - acc: 0.9972 - val
Epoch 16/20
60000/60000 [==============================] - 7s 113us/step - loss: 0.0096 - acc: 0.9969 - val
Epoch 17/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.0084 - acc: 0.9970 - val
Epoch 18/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.0078 - acc: 0.9974 - val
Epoch 19/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.0074 - acc: 0.9977 - val
Epoch 20/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.0057 - acc: 0.9982 - val
```

In [0]:
```
score_relu_bn3 = model_batch.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score_relu_bn3[0])
print('Test accuracy:', score_relu_bn3[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, v

# 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 e
```

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

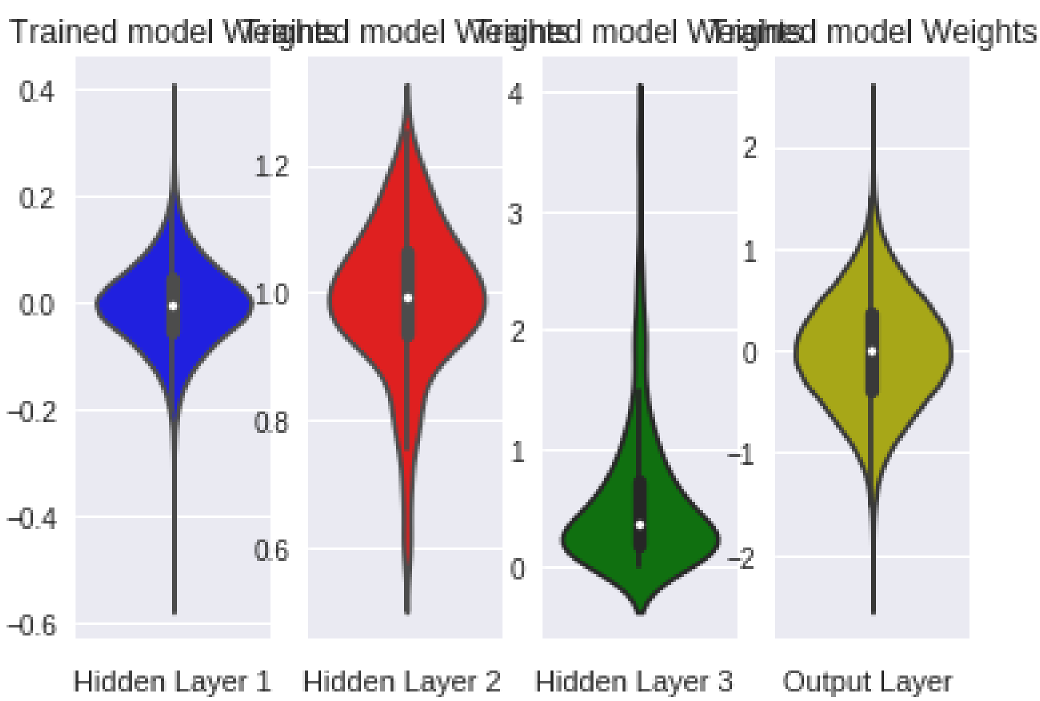
```
Test score: 0.08087161136912183
Test accuracy: 0.9798
```




```
In [0]: w_after = model_batch.get_weights()

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


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

        plt.subplot(1, 4, 2)
        plt.title("Trained model Weights")
```

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

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

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

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is
  violin_data = remove_na(group_data)
```



8 MLP + Batch-Norm + AdamOptimizer with 5 hidden layer

```
In [0]: from keras.layers.normalization import BatchNormalization
```

```
model_batch = Sequential()

model_batch.add(Dense(451, activation = 'relu', input_shape=(input_dim,), kernel_initia
model_batch.add(BatchNormalization())

model_batch.add(Dense(272, activation = 'relu', kernel_initializer = RandomNormal(mean
model_batch.add(BatchNormalization())

model_batch.add(Dense(180, activation = 'relu', kernel_initializer = RandomNormal(mean
model_batch.add(BatchNormalization())

model_batch.add(Dense(97, activation = 'relu', kernel_initializer = RandomNormal(mean =
model_batch.add(BatchNormalization())

model_batch.add(Dense(56, activation = 'relu', kernel_initializer = RandomNormal(mean =
model_batch.add(BatchNormalization())


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


model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_28 (Dense)             (None, 451)               354035
_________________________________________________________________
batch_normalization_8 (Batch (None, 451)               1804
_________________________________________________________________
dense_29 (Dense)             (None, 272)               122944
_________________________________________________________________
batch_normalization_9 (Batch (None, 272)               1088
_________________________________________________________________
dense_30 (Dense)             (None, 180)               49140
_________________________________________________________________
batch_normalization_10 (Batc (None, 180)               720
_________________________________________________________________
dense_31 (Dense)             (None, 97)                17557
_________________________________________________________________
batch_normalization_11 (Batc (None, 97)                388
_________________________________________________________________
dense_32 (Dense)             (None, 56)                5488
_________________________________________________________________
batch_normalization_12 (Batc (None, 56)                224
_________________________________________________________________
dense_33 (Dense)             (None, 10)                570
=================================================================
```

```
Total params: 553,958
Trainable params: 551,846
Non-trainable params: 2,112
_________________________________________________________________
```

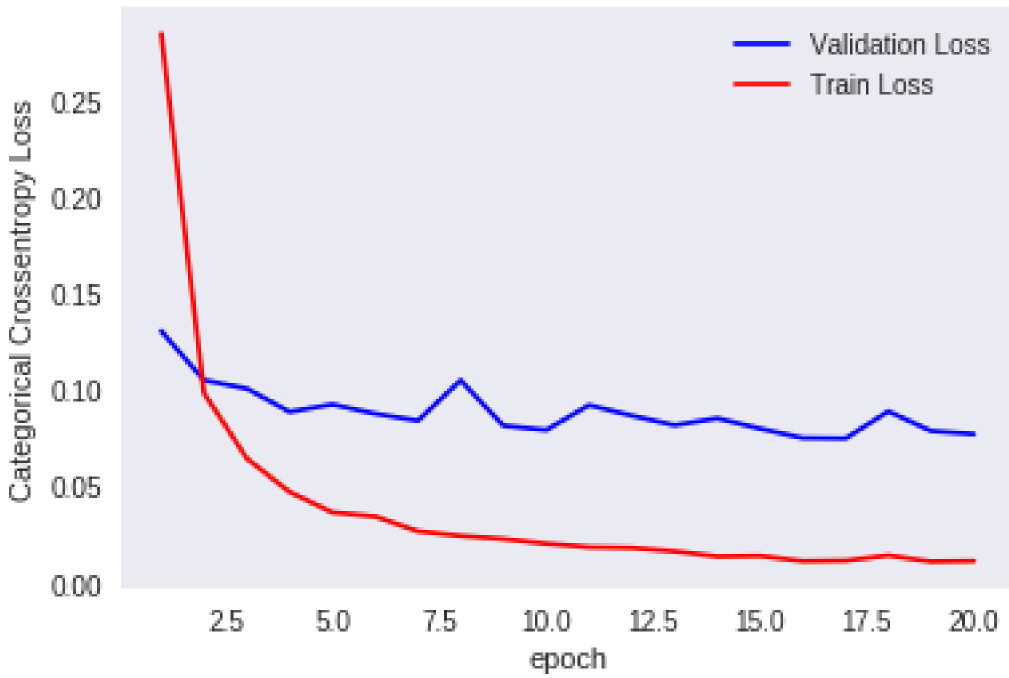
```
In [0]: model_batch.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = [

        history = model_batch.fit(X_train, Y_train, batch_size = batch_size, epochs = nb_epoch
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 13s 209us/step - loss: 0.2851 - acc: 0.9181 - va
Epoch 2/20
60000/60000 [==============================] - 11s 182us/step - loss: 0.0984 - acc: 0.9705 - va
Epoch 3/20
60000/60000 [==============================] - 11s 179us/step - loss: 0.0643 - acc: 0.9797 - va
Epoch 4/20
60000/60000 [==============================] - 11s 178us/step - loss: 0.0470 - acc: 0.9852 - va
Epoch 5/20
60000/60000 [==============================] - 11s 178us/step - loss: 0.0364 - acc: 0.9880 - va
Epoch 6/20
60000/60000 [==============================] - 11s 178us/step - loss: 0.0343 - acc: 0.9886 - va
Epoch 7/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.0266 - acc: 0.9913 - va
Epoch 8/20
60000/60000 [==============================] - 10s 173us/step - loss: 0.0242 - acc: 0.9919 - va
Epoch 9/20
60000/60000 [==============================] - 10s 174us/step - loss: 0.0226 - acc: 0.9927 - va
Epoch 10/20
60000/60000 [==============================] - 10s 173us/step - loss: 0.0201 - acc: 0.9934 - va
Epoch 11/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.0183 - acc: 0.9940 - va
Epoch 12/20
60000/60000 [==============================] - 10s 174us/step - loss: 0.0180 - acc: 0.9940 - va
Epoch 13/20
60000/60000 [==============================] - 10s 175us/step - loss: 0.0161 - acc: 0.9947 - va
Epoch 14/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.0135 - acc: 0.9956 - va
Epoch 15/20
60000/60000 [==============================] - 10s 173us/step - loss: 0.0137 - acc: 0.9956 - va
Epoch 16/20
60000/60000 [==============================] - 11s 180us/step - loss: 0.0110 - acc: 0.9965 - va
Epoch 17/20
60000/60000 [==============================] - 11s 180us/step - loss: 0.0113 - acc: 0.9964 - va
Epoch 18/20
60000/60000 [==============================] - 11s 180us/step - loss: 0.0140 - acc: 0.9953 - va
Epoch 19/20
```

```
60000/60000 [==============================] - 11s 181us/step - loss: 0.0108 - acc: 0.9965 - va
Epoch 20/20
60000/60000 [==============================] - 11s 181us/step - loss: 0.0111 - acc: 0.9964 - va
```

```
In [0]: score_relu_bn5 = model_batch.evaluate(X_test, Y_test, verbose=0)
        print('Test score:', score_relu_bn5[0])
        print('Test accuracy:', score_relu_bn5[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, v

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

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

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

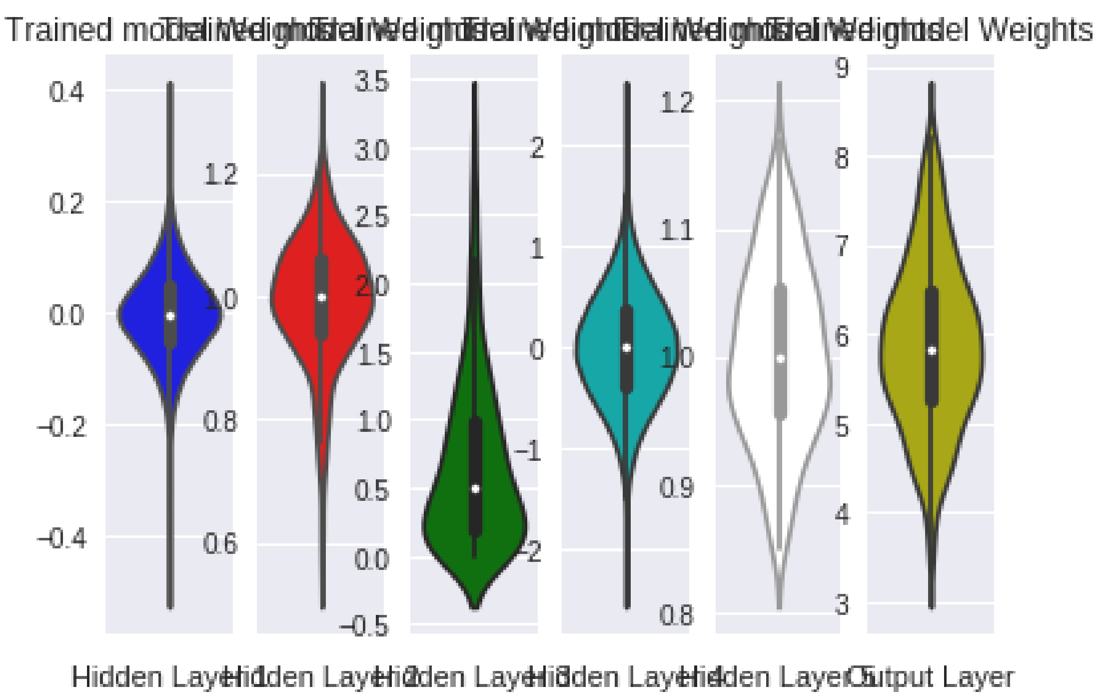
```
Test score: 0.0770734781166655
Test accuracy: 0.9815
```




```
In [0]: w_after = model_batch.get_weights()

        h1_w = w_after[0].flatten().reshape(-1,1)
        h2_w = w_after[2].flatten().reshape(-1,1)
        h3_w = w_after[4].flatten().reshape(-1,1)
        h4_w = w_after[6].flatten().reshape(-1,1)
        h5_w = w_after[8].flatten().reshape(-1,1)
        out_w = w_after[10].flatten().reshape(-1,1)


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

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

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='c')
plt.xlabel('Hidden Layer 4')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='w')
plt.xlabel('Hidden Layer 5')

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

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is
  violin_data = remove_na(group_data)
```



Observations 1. We did not get good accuracy with 2 and 3 hidden layer and also mean of weights deviated from 0 to 1. 2. With 5 hidden layer we got slightly good accuracy than 2 and 3 hidden layer.

9 3. MLP + Dropout + AdamOptimizer with 2 hidden layer

In [0]:
```
from keras.layers import Dropout

model_drop = Sequential()


model_drop.add(Dense(370, activation='relu', input_shape=(input_dim,), kernel_initiali
model_drop.add(Dropout(0.5))

model_drop.add(Dense(112, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
model_drop.add(Dropout(0.5))

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

model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_34 (Dense)             (None, 370)               290450
_________________________________________________________________
dropout_1 (Dropout)          (None, 370)               0
_________________________________________________________________
dense_35 (Dense)             (None, 112)               41552
_________________________________________________________________
dropout_2 (Dropout)          (None, 112)               0
_________________________________________________________________
dense_36 (Dense)             (None, 10)                1130
=================================================================
Total params: 333,132
Trainable params: 333,132
Non-trainable params: 0
_________________________________________________________________
```

In [0]:
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verl
```

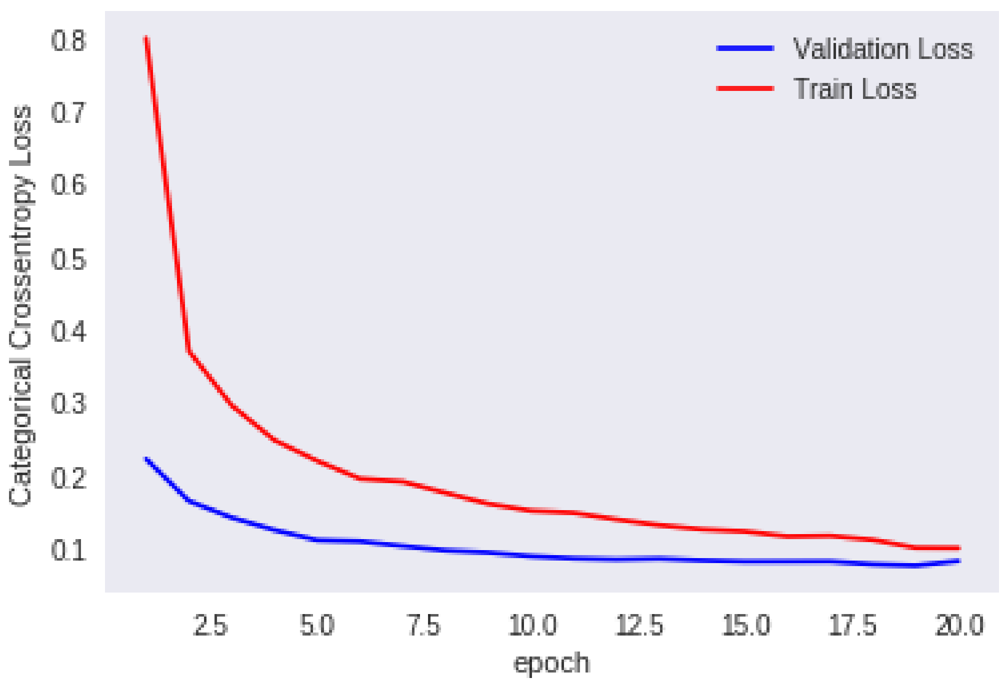
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.7979 - acc: 0.7623 - val
Epoch 2/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.3699 - acc: 0.8913 - val_
Epoch 3/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.2955 - acc: 0.9159 - val_
Epoch 4/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.2479 - acc: 0.9279 - val_
```

```
Epoch 5/20
60000/60000 [==============================] - 6s 100us/step - loss: 0.2196 - acc: 0.9374 - val
Epoch 6/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.1952 - acc: 0.9437 - val_
Epoch 7/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.1912 - acc: 0.9458 - val_
Epoch 8/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.1760 - acc: 0.9496 - val_
Epoch 9/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.1610 - acc: 0.9533 - val_
Epoch 10/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.1514 - acc: 0.9563 - val_
Epoch 11/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.1486 - acc: 0.9580 - val_
Epoch 12/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.1395 - acc: 0.9600 - val_
Epoch 13/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.1316 - acc: 0.9619 - val_
Epoch 14/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.1259 - acc: 0.9637 - val_
Epoch 15/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.1232 - acc: 0.9640 - val_
Epoch 16/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.1162 - acc: 0.9666 - val_
Epoch 17/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.1171 - acc: 0.9664 - val_
Epoch 18/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.1115 - acc: 0.9670 - val_
Epoch 19/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.1006 - acc: 0.9709 - val_
Epoch 20/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.1003 - acc: 0.9707 - val
```

In [0]:
```
score_relu_drop2 = model_drop.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score_relu_drop2[0])
print('Test accuracy:', score_relu_drop2[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, v
```

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

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

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

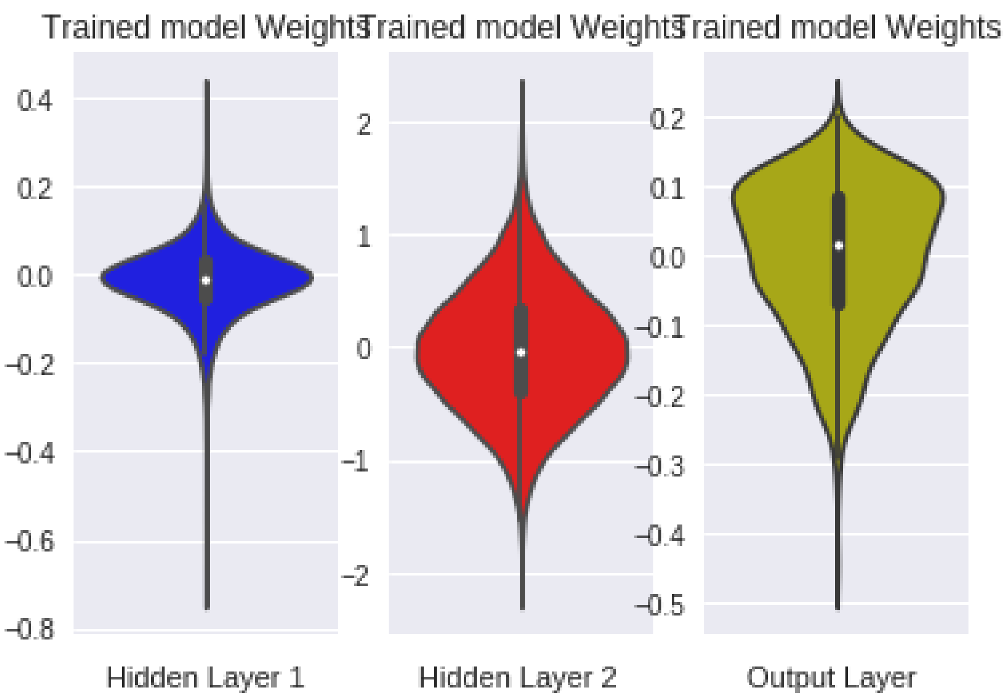
```
Test score: 0.0825668051331857
Test accuracy: 0.9783
```




```
In [0]: w_after = model_drop.get_weights()

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


        fig = plt.figure()
        plt.title("Weight matrices after model trained")
```

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

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

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

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is
  violin_data = remove_na(group_data)
```



10 MLP + Dropout + AdamOptimizer with 3 hidden layer

In [0]:
```
from keras.layers import Dropout

model_drop = Sequential()


model_drop.add(Dense(531, activation='relu', input_shape=(input_dim,), kernel_initiali
model_drop.add(Dropout(0.5))

model_drop.add(Dense(375, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
model_drop.add(Dropout(0.5))

model_drop.add(Dense(130, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
model_drop.add(Dropout(0.5))

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

model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_37 (Dense)             (None, 531)               416835
_________________________________________________________________
dropout_3 (Dropout)          (None, 531)               0
_________________________________________________________________
dense_38 (Dense)             (None, 375)               199500
_________________________________________________________________
dropout_4 (Dropout)          (None, 375)               0
_________________________________________________________________
dense_39 (Dense)             (None, 130)               48880
_________________________________________________________________
dropout_5 (Dropout)          (None, 130)               0
_________________________________________________________________
dense_40 (Dense)             (None, 10)                1310
=================================================================
Total params: 666,525
Trainable params: 666,525
Non-trainable params: 0
_________________________________________________________________
```

In [0]:
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verl
```

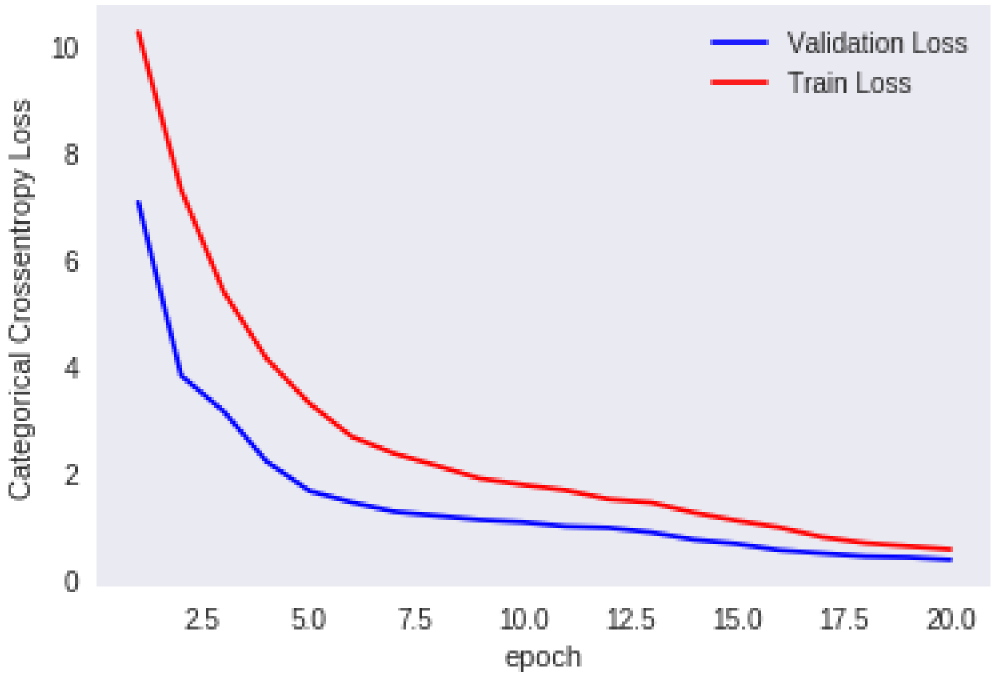
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
```

```
60000/60000 [==============================] - 12s 192us/step - loss: 10.2589 - acc: 0.3486 - v
Epoch 2/20
60000/60000 [==============================] - 10s 169us/step - loss: 7.3090 - acc: 0.5348 - va
Epoch 3/20
60000/60000 [==============================] - 10s 170us/step - loss: 5.3924 - acc: 0.6534 - va
Epoch 4/20
60000/60000 [==============================] - 10s 170us/step - loss: 4.1462 - acc: 0.7304 - va
Epoch 5/20
60000/60000 [==============================] - 10s 170us/step - loss: 3.3093 - acc: 0.7818 - va
Epoch 6/20
60000/60000 [==============================] - 10s 170us/step - loss: 2.6803 - acc: 0.8215 - va
Epoch 7/20
60000/60000 [==============================] - 10s 171us/step - loss: 2.3632 - acc: 0.8423 - va
Epoch 8/20
60000/60000 [==============================] - 10s 170us/step - loss: 2.1355 - acc: 0.8575 - va
Epoch 9/20
60000/60000 [==============================] - 10s 173us/step - loss: 1.8977 - acc: 0.8719 - va
Epoch 10/20
60000/60000 [==============================] - 10s 172us/step - loss: 1.7798 - acc: 0.8790 - va
Epoch 11/20
60000/60000 [==============================] - 10s 171us/step - loss: 1.6786 - acc: 0.8857 - va
Epoch 12/20
60000/60000 [==============================] - 10s 171us/step - loss: 1.5114 - acc: 0.8957 - va
Epoch 13/20
60000/60000 [==============================] - 10s 172us/step - loss: 1.4467 - acc: 0.8977 - va
Epoch 14/20
60000/60000 [==============================] - 10s 170us/step - loss: 1.2593 - acc: 0.9055 - va
Epoch 15/20
60000/60000 [==============================] - 10s 170us/step - loss: 1.1119 - acc: 0.9063 - va
Epoch 16/20
60000/60000 [==============================] - 10s 169us/step - loss: 0.9847 - acc: 0.9029 - va
Epoch 17/20
60000/60000 [==============================] - 10s 168us/step - loss: 0.8042 - acc: 0.8968 - va
Epoch 18/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.6922 - acc: 0.8995 - va
Epoch 19/20
60000/60000 [==============================] - 10s 168us/step - loss: 0.6329 - acc: 0.9009 - va
Epoch 20/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.5741 - acc: 0.9026 - va
```

In [0]:
```python
score_relu_drop3 = model_drop.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score_relu_drop3[0])
print('Test accuracy:', score_relu_drop3[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, v

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

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

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

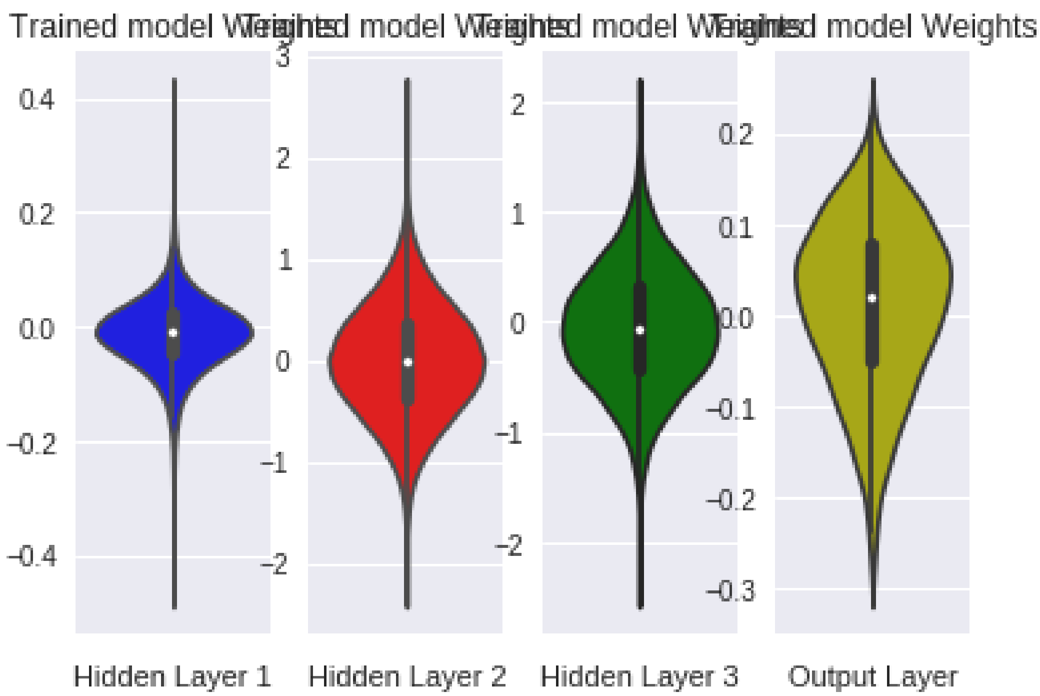
```
Test score: 0.37853595131413315
Test accuracy: 0.9328
```




```
In [0]: w_after = model_drop.get_weights()
```

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


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

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

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


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

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is
  violin_data = remove_na(group_data)
```



11 MLP + Dropout + AdamOptimizer with 5 hidden layer

```
In [0]: from keras.layers import Dropout

        model_drop = Sequential()

        model_drop.add(Dense(681, activation='relu', input_shape=(input_dim,), kernel_initiali
        model_drop.add(Dropout(0.5))

        model_drop.add(Dense(475, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
        model_drop.add(Dropout(0.5))

        model_drop.add(Dense(230, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
        model_drop.add(Dropout(0.5))

        model_drop.add(Dense(102, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
        model_drop.add(Dropout(0.5))

        model_drop.add(Dense(28, activation='relu', kernel_initializer=RandomNormal(mean=0.0, 
        model_drop.add(Dropout(0.5))

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

        model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_41 (Dense)             (None, 681)               534585
_________________________________________________________________
dropout_6 (Dropout)          (None, 681)               0
_________________________________________________________________
dense_42 (Dense)             (None, 475)               323950
_________________________________________________________________
dropout_7 (Dropout)          (None, 475)               0
_________________________________________________________________
dense_43 (Dense)             (None, 230)               109480
_________________________________________________________________
dropout_8 (Dropout)          (None, 230)               0
_________________________________________________________________
dense_44 (Dense)             (None, 102)               23562
_________________________________________________________________
dropout_9 (Dropout)          (None, 102)               0
_________________________________________________________________
dense_45 (Dense)             (None, 28)                2884
_________________________________________________________________
dropout_10 (Dropout)         (None, 28)                0
_________________________________________________________________
dense_46 (Dense)             (None, 10)                290
=================================================================
Total params: 994,751
Trainable params: 994,751
Non-trainable params: 0
_________________________________________________________________
```

```
In [0]: model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

        history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verb
```

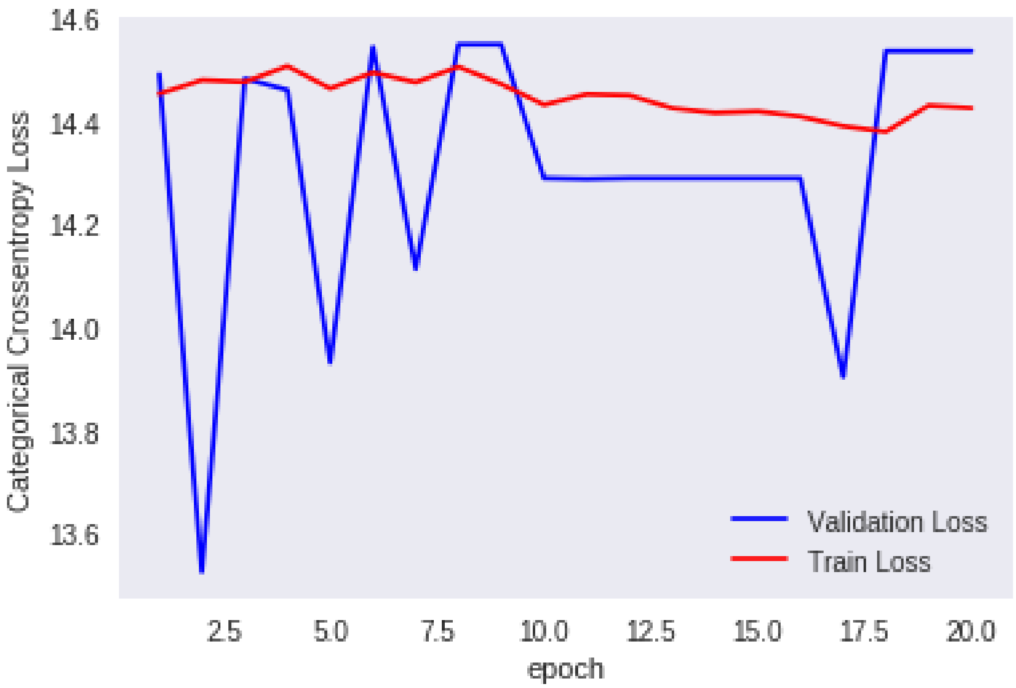
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 16s 274us/step - loss: 14.4528 - acc: 0.1023 - v
Epoch 2/20
60000/60000 [==============================] - 15s 249us/step - loss: 14.4791 - acc: 0.1008 - v
Epoch 3/20
60000/60000 [==============================] - 15s 250us/step - loss: 14.4756 - acc: 0.1010 - v
Epoch 4/20
60000/60000 [==============================] - 15s 250us/step - loss: 14.5060 - acc: 0.0991 - v
Epoch 5/20
60000/60000 [==============================] - 15s 249us/step - loss: 14.4631 - acc: 0.1014 - v
Epoch 6/20
60000/60000 [==============================] - 15s 253us/step - loss: 14.4937 - acc: 0.0995 - v
```

```
Epoch 7/20
60000/60000 [==============================] - 15s 252us/step - loss: 14.4749 - acc: 0.1003 - v
Epoch 8/20
60000/60000 [==============================] - 15s 248us/step - loss: 14.5048 - acc: 0.0985 - v
Epoch 9/20
60000/60000 [==============================] - 15s 246us/step - loss: 14.4717 - acc: 0.1007 - v
Epoch 10/20
60000/60000 [==============================] - 15s 248us/step - loss: 14.4314 - acc: 0.1035 - v
Epoch 11/20
60000/60000 [==============================] - 15s 246us/step - loss: 14.4519 - acc: 0.1023 - v
Epoch 12/20
60000/60000 [==============================] - 15s 245us/step - loss: 14.4497 - acc: 0.1024 - v
Epoch 13/20
60000/60000 [==============================] - 15s 244us/step - loss: 14.4248 - acc: 0.1043 - v
Epoch 14/20
60000/60000 [==============================] - 15s 247us/step - loss: 14.4156 - acc: 0.1049 - v
Epoch 15/20
60000/60000 [==============================] - 15s 245us/step - loss: 14.4187 - acc: 0.1047 - v
Epoch 16/20
60000/60000 [==============================] - 15s 247us/step - loss: 14.4085 - acc: 0.1054 - v
Epoch 17/20
60000/60000 [==============================] - 15s 249us/step - loss: 14.3898 - acc: 0.1066 - v
Epoch 18/20
60000/60000 [==============================] - 15s 247us/step - loss: 14.3788 - acc: 0.1071 - v
Epoch 19/20
60000/60000 [==============================] - 15s 249us/step - loss: 14.4300 - acc: 0.1036 - v
Epoch 20/20
60000/60000 [==============================] - 15s 247us/step - loss: 14.4250 - acc: 0.1037 - v
```

```
In [0]: score_relu_drop5 = model_drop.evaluate(X_test, Y_test, verbose=0)
        print('Test score:', score_relu_drop5[0])
        print('Test accuracy:', score_relu_drop5[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, v

        # 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 e

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

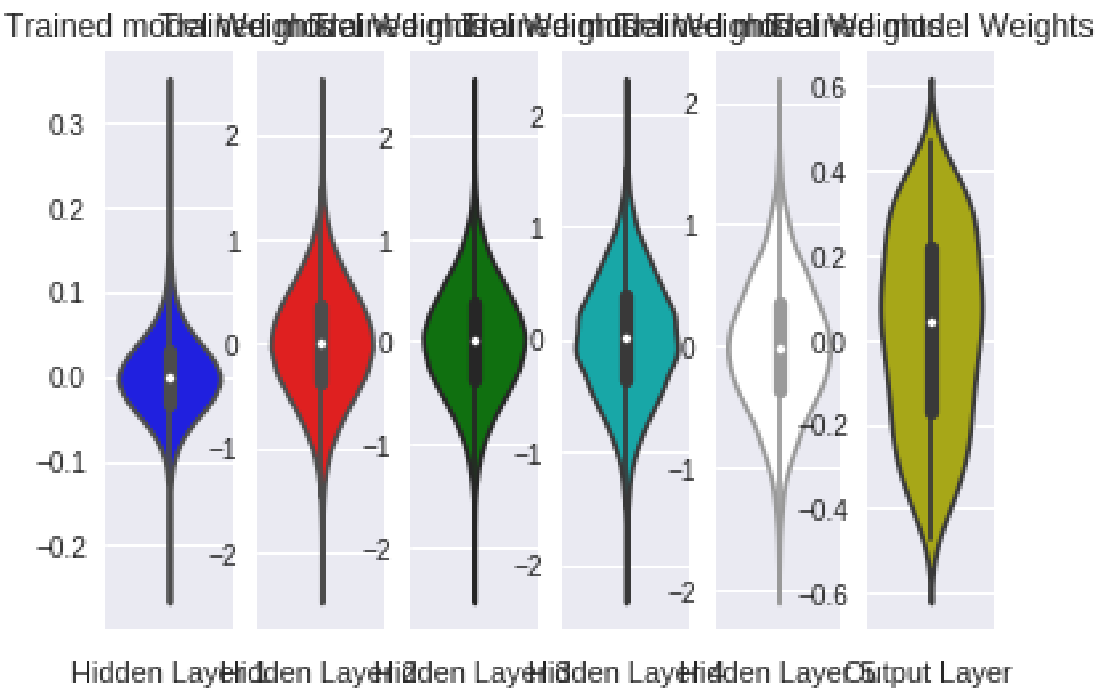
```
Test score: 14.535298265075683
Test accuracy: 0.0982
```




```
In [0]: w_after = model_drop.get_weights()

        h1_w = w_after[0].flatten().reshape(-1,1)
        h2_w = w_after[2].flatten().reshape(-1,1)
        h3_w = w_after[4].flatten().reshape(-1,1)
        h4_w = w_after[6].flatten().reshape(-1,1)
        h5_w = w_after[8].flatten().reshape(-1,1)
        out_w = w_after[10].flatten().reshape(-1,1)


        fig = plt.figure()
        plt.title("Weight matrices after model trained")
        plt.subplot(1, 6, 1)
```

```
plt.title("Trained model Weights")
ax = sns.violinplot(y=h1_w,color='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='g')
plt.xlabel('Hidden Layer 3')

plt.subplot(1, 6, 4)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h4_w, color='c')
plt.xlabel('Hidden Layer 4')

plt.subplot(1, 6, 5)
plt.title("Trained model Weights")
ax = sns.violinplot(y=h5_w, color='w')
plt.xlabel('Hidden Layer 5')

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

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is
  violin_data = remove_na(group_data)
```



Observations 1. We use 2, 3 and 5 different layer architecture with dropout and did not get good accuracy in 5 layer architecture whereas first 2 model architecture works very well.

12 4. MLP + RELU + Dropout + BatchNormalization + AdamOptimizer with 2 hidden layers

In [0]:
```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-

from keras.layers import Dropout

model_drop = Sequential()


model_drop.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initiali
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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


model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_47 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_13 (Batc (None, 512)               2048
_________________________________________________________________
dropout_11 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_48 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_14 (Batc (None, 128)               512
_________________________________________________________________
dropout_12 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_49 (Dense)             (None, 10)                1290
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

```
In [0]: model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

        history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verb
```

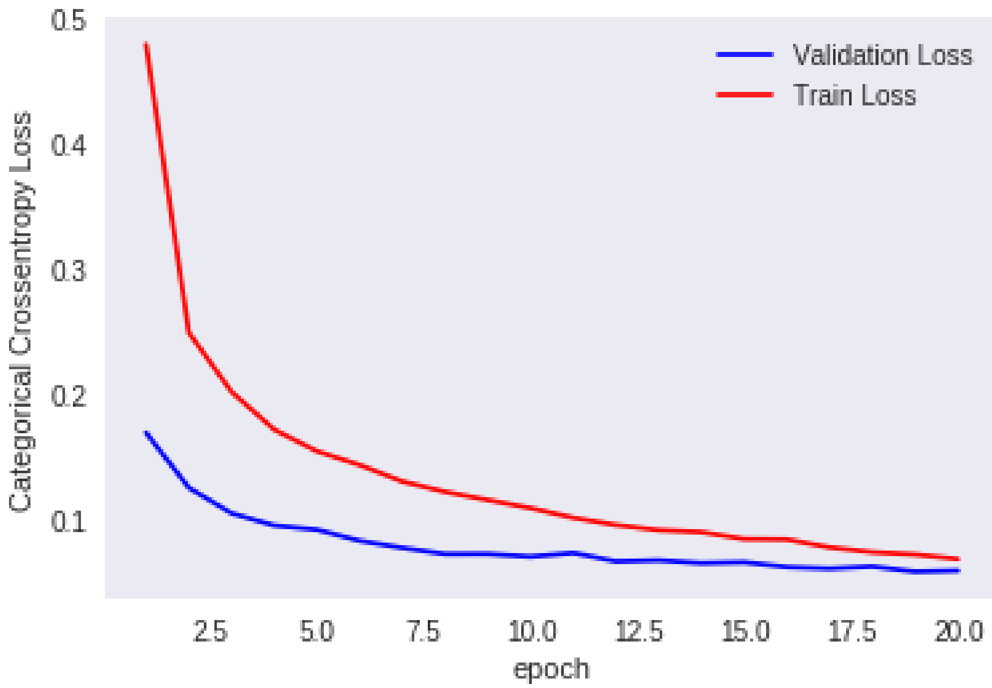
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s 169us/step - loss: 0.4788 - acc: 0.8538 - va
Epoch 2/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.2481 - acc: 0.9256 - val
Epoch 3/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.2007 - acc: 0.9398 - val
Epoch 4/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.1705 - acc: 0.9483 - val
Epoch 5/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.1532 - acc: 0.9536 - val
Epoch 6/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.1423 - acc: 0.9571 - val
Epoch 7/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.1290 - acc: 0.9613 - val
Epoch 8/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.1207 - acc: 0.9627 - val
Epoch 9/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.1142 - acc: 0.9656 - val
Epoch 10/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.1076 - acc: 0.9674 - val
```

```
Epoch 11/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.0999 - acc: 0.9692 - val
Epoch 12/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.0941 - acc: 0.9710 - val
Epoch 13/20
60000/60000 [==============================] - 8s 140us/step - loss: 0.0900 - acc: 0.9712 - val
Epoch 14/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0884 - acc: 0.9721 - val
Epoch 15/20
60000/60000 [==============================] - 8s 140us/step - loss: 0.0829 - acc: 0.9741 - val
Epoch 16/20
60000/60000 [==============================] - 8s 140us/step - loss: 0.0826 - acc: 0.9745 - val
Epoch 17/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0762 - acc: 0.9761 - val
Epoch 18/20
60000/60000 [==============================] - 8s 140us/step - loss: 0.0724 - acc: 0.9775 - val
Epoch 19/20
60000/60000 [==============================] - 8s 140us/step - loss: 0.0705 - acc: 0.9781 - val
Epoch 20/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.0669 - acc: 0.9792 - val
```

In [0]:
```
score_relu_bn_drop2 = model_drop.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score_relu_bn_drop2[0])
print('Test accuracy:', score_relu_bn_drop2[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, v

# 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 e

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

```
Test score: 0.0575819099090164
```

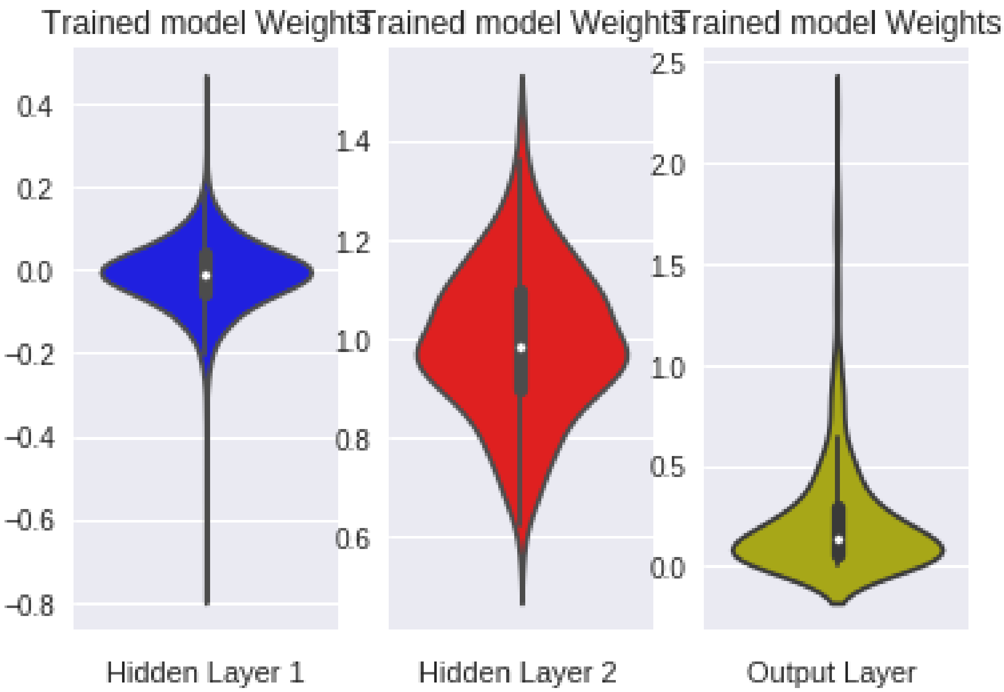
```
Test accuracy: 0.9831
```




```
In [0]: w_after = model_drop.get_weights()

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


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

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

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

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

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is
  violin_data = remove_na(group_data)
```



13 MLP + RELU + Dropout + BatchNormalization with 2 hidden layers

```
In [0]: from keras.layers.normalization import BatchNormalization
        from keras.layers import Dropout
        from keras.models import Sequential
        from keras.layers import Dense, Activation

        model = Sequential()

        model.add(Dense(389, activation = "relu", input_shape = (input_dim,), kernel_initialize
        model.add(BatchNormalization())
        model.add(Dropout(0.5))

        model.add(Dense(258, activation = "relu", kernel_initializer = RandomNormal(mean=0.0, s
        model.add(BatchNormalization())
```

```
model.add(Dropout(0.5))

model.add(Dense(output_dim, activation = "softmax"))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_50 (Dense)             (None, 389)               305365
_________________________________________________________________
batch_normalization_15 (Batc (None, 389)               1556
_________________________________________________________________
dropout_13 (Dropout)         (None, 389)               0
_________________________________________________________________
dense_51 (Dense)             (None, 258)               100620
_________________________________________________________________
batch_normalization_16 (Batc (None, 258)               1032
_________________________________________________________________
dropout_14 (Dropout)         (None, 258)               0
_________________________________________________________________
dense_52 (Dense)             (None, 10)                2590
=================================================================
Total params: 411,163
Trainable params: 409,869
Non-trainable params: 1,294
_________________________________________________________________
```

```
In [0]: model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accura

        history = model.fit(X_train, Y_train, batch_size = batch_size, epochs = nb_epoch, verbo
```

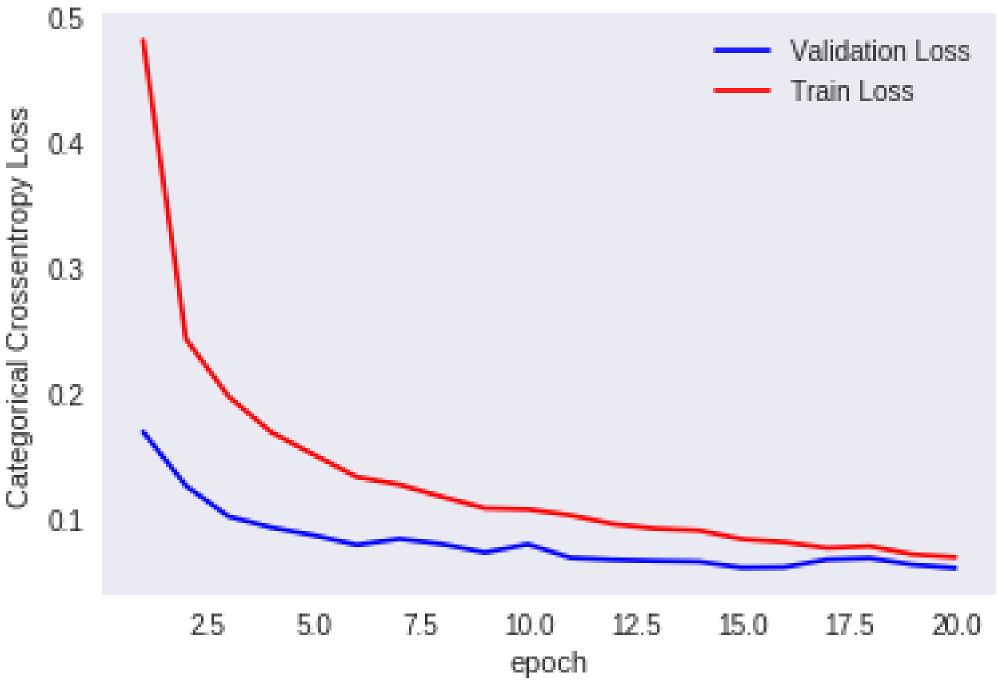
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s 166us/step - loss: 0.4825 - acc: 0.8528 - va
Epoch 2/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.2447 - acc: 0.9261 - val
Epoch 3/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.1989 - acc: 0.9403 - val
Epoch 4/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.1704 - acc: 0.9486 - val
Epoch 5/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.1525 - acc: 0.9535 - val
Epoch 6/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.1348 - acc: 0.9576 - val
Epoch 7/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.1287 - acc: 0.9605 - val
```

```
Epoch 8/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.1191 - acc: 0.9629 - val
Epoch 9/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.1101 - acc: 0.9658 - val
Epoch 10/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.1092 - acc: 0.9666 - val
Epoch 11/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.1044 - acc: 0.9681 - val
Epoch 12/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0975 - acc: 0.9693 - val
Epoch 13/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0937 - acc: 0.9706 - val
Epoch 14/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0924 - acc: 0.9707 - val
Epoch 15/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0855 - acc: 0.9734 - val
Epoch 16/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0830 - acc: 0.9734 - val
Epoch 17/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0784 - acc: 0.9753 - val
Epoch 18/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0798 - acc: 0.9753 - val
Epoch 19/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0733 - acc: 0.9763 - val
Epoch 20/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0710 - acc: 0.9775 - val
```

In [0]:
```
score_relu_bn_drop2_diff = model.evaluate(X_test, Y_test, verbose = 0)
print('Test score:', score_relu_bn_drop2_diff[0])
print('Test accuracy:', score_relu_bn_drop2_diff[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, v

# 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 e

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

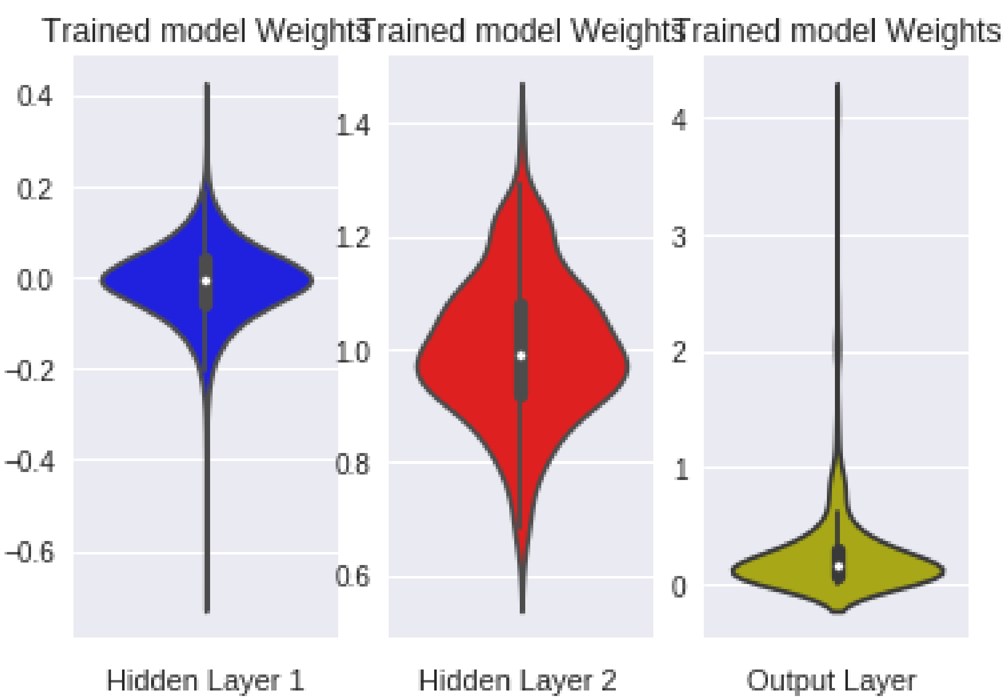
```
Test score: 0.06233537118713721
Test accuracy: 0.982
```




```
In [0]: w_after = model.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()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is
  violin_data = remove_na(group_data)
```



14 MLP + RELU + Dropout + BatchNormalization with 3 hidden layers

```
In [0]: from keras.layers.normalization import BatchNormalization
        from keras.layers import Dropout
        from keras.models import Sequential
        from keras.layers import Dense, Activation
```

```
model = Sequential()

model.add(Dense(434, activation = "relu", input_shape = (input_dim,), kernel_initialize
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(391, activation = "relu", kernel_initializer = RandomNormal(mean=0.0, s
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(141, activation = "relu", kernel_initializer = RandomNormal(mean=0.0, s
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(output_dim, activation = "softmax"))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_53 (Dense)             (None, 434)               340690
_________________________________________________________________
batch_normalization_17 (Batc (None, 434)               1736
_________________________________________________________________
dropout_15 (Dropout)         (None, 434)               0
_________________________________________________________________
dense_54 (Dense)             (None, 391)               170085
_________________________________________________________________
batch_normalization_18 (Batc (None, 391)               1564
_________________________________________________________________
dropout_16 (Dropout)         (None, 391)               0
_________________________________________________________________
dense_55 (Dense)             (None, 141)               55272
_________________________________________________________________
batch_normalization_19 (Batc (None, 141)               564
_________________________________________________________________
dropout_17 (Dropout)         (None, 141)               0
_________________________________________________________________
dense_56 (Dense)             (None, 10)                1420
=================================================================
Total params: 571,331
Trainable params: 569,399
Non-trainable params: 1,932
_________________________________________________________________
```

```
In [0]: model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accura
```

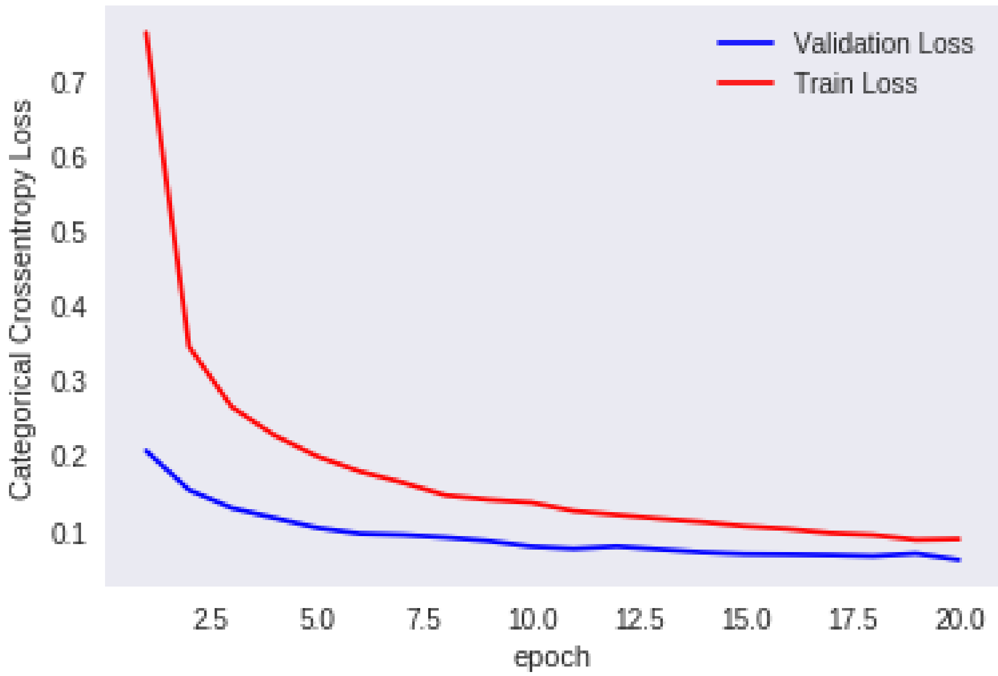
```
history = model.fit(X_train, Y_train, batch_size = batch_size, epochs = nb_epoch, verbo
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 13s 224us/step - loss: 0.7627 - acc: 0.7634 - va
Epoch 2/20
60000/60000 [==============================] - 11s 185us/step - loss: 0.3444 - acc: 0.8963 - va
Epoch 3/20
60000/60000 [==============================] - 11s 185us/step - loss: 0.2640 - acc: 0.9214 - va
Epoch 4/20
60000/60000 [==============================] - 11s 185us/step - loss: 0.2262 - acc: 0.9333 - va
Epoch 5/20
60000/60000 [==============================] - 11s 186us/step - loss: 0.1982 - acc: 0.9404 - va
Epoch 6/20
60000/60000 [==============================] - 12s 194us/step - loss: 0.1781 - acc: 0.9472 - va
Epoch 7/20
60000/60000 [==============================] - 11s 185us/step - loss: 0.1633 - acc: 0.9515 - va
Epoch 8/20
60000/60000 [==============================] - 11s 186us/step - loss: 0.1464 - acc: 0.9557 - va
Epoch 9/20
60000/60000 [==============================] - 11s 187us/step - loss: 0.1406 - acc: 0.9583 - va
Epoch 10/20
60000/60000 [==============================] - 11s 187us/step - loss: 0.1367 - acc: 0.9592 - va
Epoch 11/20
60000/60000 [==============================] - 11s 186us/step - loss: 0.1256 - acc: 0.9618 - va
Epoch 12/20
60000/60000 [==============================] - 11s 186us/step - loss: 0.1205 - acc: 0.9647 - va
Epoch 13/20
60000/60000 [==============================] - 11s 186us/step - loss: 0.1155 - acc: 0.9656 - va
Epoch 14/20
60000/60000 [==============================] - 11s 186us/step - loss: 0.1110 - acc: 0.9672 - va
Epoch 15/20
60000/60000 [==============================] - 11s 186us/step - loss: 0.1054 - acc: 0.9677 - va
Epoch 16/20
60000/60000 [==============================] - 11s 189us/step - loss: 0.1017 - acc: 0.9691 - va
Epoch 17/20
60000/60000 [==============================] - 11s 187us/step - loss: 0.0962 - acc: 0.9712 - va
Epoch 18/20
60000/60000 [==============================] - 11s 187us/step - loss: 0.0937 - acc: 0.9725 - va
Epoch 19/20
60000/60000 [==============================] - 11s 187us/step - loss: 0.0873 - acc: 0.9729 - va
Epoch 20/20
60000/60000 [==============================] - 11s 187us/step - loss: 0.0884 - acc: 0.9729 - va
```

In [0]:
```
score_relu_bn_drop3 = model.evaluate(X_test, Y_test, verbose = 0)
print('Test score:', score_relu_bn_drop3[0])
```

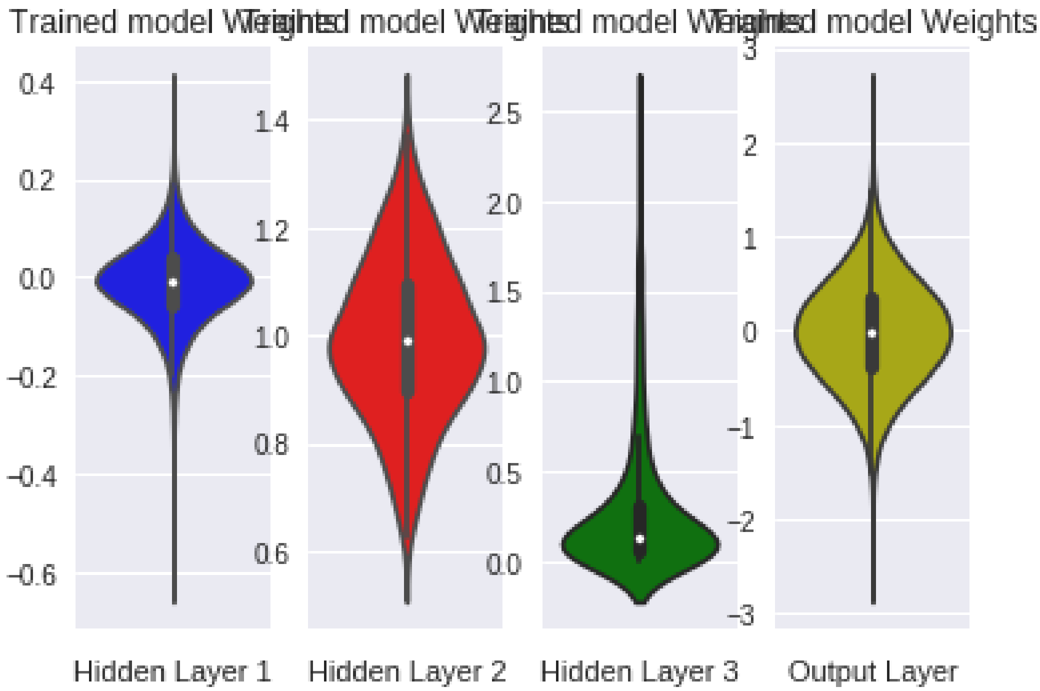
```python
print('Test accuracy:', score_relu_bn_drop3[1])

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, v

# 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 e

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

```
Test score: 0.0605601929532073
Test accuracy: 0.9824
```



```
In [0]: w_after = model.get_weights()

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


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

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

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

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

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is
  violin_data = remove_na(group_data)
```



15 MLP + RELU + Dropout + BatchNormalization with 5 hidden layers

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

        model.add(Dense(697, activation = "relu", input_shape = (input_dim,), kernel_initialize
        model.add(BatchNormalization())
        model.add(Dropout(0.5))

        model.add(Dense(458, activation = "relu", kernel_initializer = RandomNormal(mean=0.0, s
        model.add(BatchNormalization())
        model.add(Dropout(0.5))

        model.add(Dense(246, activation = "relu", kernel_initializer = RandomNormal(mean=0.0, s
        model.add(BatchNormalization())
        model.add(Dropout(0.5))

        model.add(Dense(111, activation = "relu", kernel_initializer = RandomNormal(mean=0.0, s
        model.add(BatchNormalization())
        model.add(Dropout(0.5))

        model.add(Dense(58, activation = "relu", kernel_initializer = RandomNormal(mean=0.0, st
        model.add(BatchNormalization())
        model.add(Dropout(0.5))
```

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

        model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_57 (Dense)             (None, 697)               547145
_________________________________________________________________
batch_normalization_20 (Batc (None, 697)               2788
_________________________________________________________________
dropout_18 (Dropout)         (None, 697)               0
_________________________________________________________________
dense_58 (Dense)             (None, 458)               319684
_________________________________________________________________
batch_normalization_21 (Batc (None, 458)               1832
_________________________________________________________________
dropout_19 (Dropout)         (None, 458)               0
_________________________________________________________________
dense_59 (Dense)             (None, 246)               112914
_________________________________________________________________
batch_normalization_22 (Batc (None, 246)               984
_________________________________________________________________
dropout_20 (Dropout)         (None, 246)               0
_________________________________________________________________
dense_60 (Dense)             (None, 111)               27417
_________________________________________________________________
batch_normalization_23 (Batc (None, 111)               444
_________________________________________________________________
dropout_21 (Dropout)         (None, 111)               0
_________________________________________________________________
dense_61 (Dense)             (None, 58)                6496
_________________________________________________________________
batch_normalization_24 (Batc (None, 58)                232
_________________________________________________________________
dropout_22 (Dropout)         (None, 58)                0
_________________________________________________________________
dense_62 (Dense)             (None, 10)                590
=================================================================
Total params: 1,020,526
Trainable params: 1,017,386
Non-trainable params: 3,140
_________________________________________________________________
```

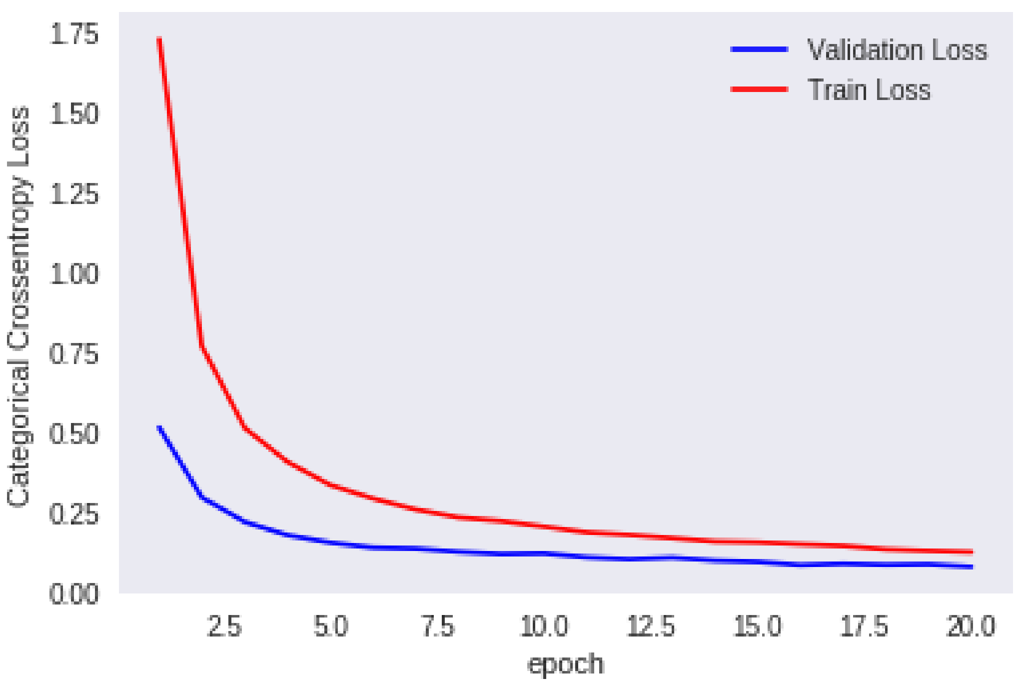
```
In [0]: model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accura

        history = model.fit(X_train, Y_train, batch_size = batch_size, epochs = nb_epoch, verbo
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 22s 361us/step - loss: 1.7259 - acc: 0.4318 - va
Epoch 2/20
60000/60000 [==============================] - 19s 315us/step - loss: 0.7668 - acc: 0.7482 - va
Epoch 3/20
60000/60000 [==============================] - 18s 305us/step - loss: 0.5112 - acc: 0.8477 - va
Epoch 4/20
60000/60000 [==============================] - 18s 304us/step - loss: 0.4055 - acc: 0.8850 - va
Epoch 5/20
60000/60000 [==============================] - 18s 306us/step - loss: 0.3332 - acc: 0.9080 - va
Epoch 6/20
60000/60000 [==============================] - 18s 306us/step - loss: 0.2908 - acc: 0.9211 - va
Epoch 7/20
60000/60000 [==============================] - 18s 305us/step - loss: 0.2566 - acc: 0.9311 - va
Epoch 8/20
60000/60000 [==============================] - 19s 314us/step - loss: 0.2320 - acc: 0.9391 - va
Epoch 9/20
60000/60000 [==============================] - 18s 304us/step - loss: 0.2208 - acc: 0.9429 - va
Epoch 10/20
60000/60000 [==============================] - 18s 305us/step - loss: 0.2035 - acc: 0.9466 - va
Epoch 11/20
60000/60000 [==============================] - 18s 304us/step - loss: 0.1856 - acc: 0.9511 - va
Epoch 12/20
60000/60000 [==============================] - 18s 305us/step - loss: 0.1777 - acc: 0.9539 - va
Epoch 13/20
60000/60000 [==============================] - 18s 306us/step - loss: 0.1676 - acc: 0.9565 - va
Epoch 14/20
60000/60000 [==============================] - 18s 306us/step - loss: 0.1569 - acc: 0.9589 - va
Epoch 15/20
60000/60000 [==============================] - 18s 305us/step - loss: 0.1544 - acc: 0.9611 - va
Epoch 16/20
60000/60000 [==============================] - 18s 308us/step - loss: 0.1485 - acc: 0.9624 - va
Epoch 17/20
60000/60000 [==============================] - 18s 306us/step - loss: 0.1435 - acc: 0.9630 - va
Epoch 18/20
60000/60000 [==============================] - 18s 306us/step - loss: 0.1323 - acc: 0.9654 - va
Epoch 19/20
60000/60000 [==============================] - 18s 307us/step - loss: 0.1278 - acc: 0.9675 - va
Epoch 20/20
60000/60000 [==============================] - 18s 306us/step - loss: 0.1241 - acc: 0.9685 - va
```

In [0]:
```
score_relu_bn_drop5 = model.evaluate(X_test, Y_test, verbose = 0)
print('Test score:', score_relu_bn_drop5[0])
print('Test accuracy:', score_relu_bn_drop5[1])

fig, ax = plt.subplots(1, 1)
```

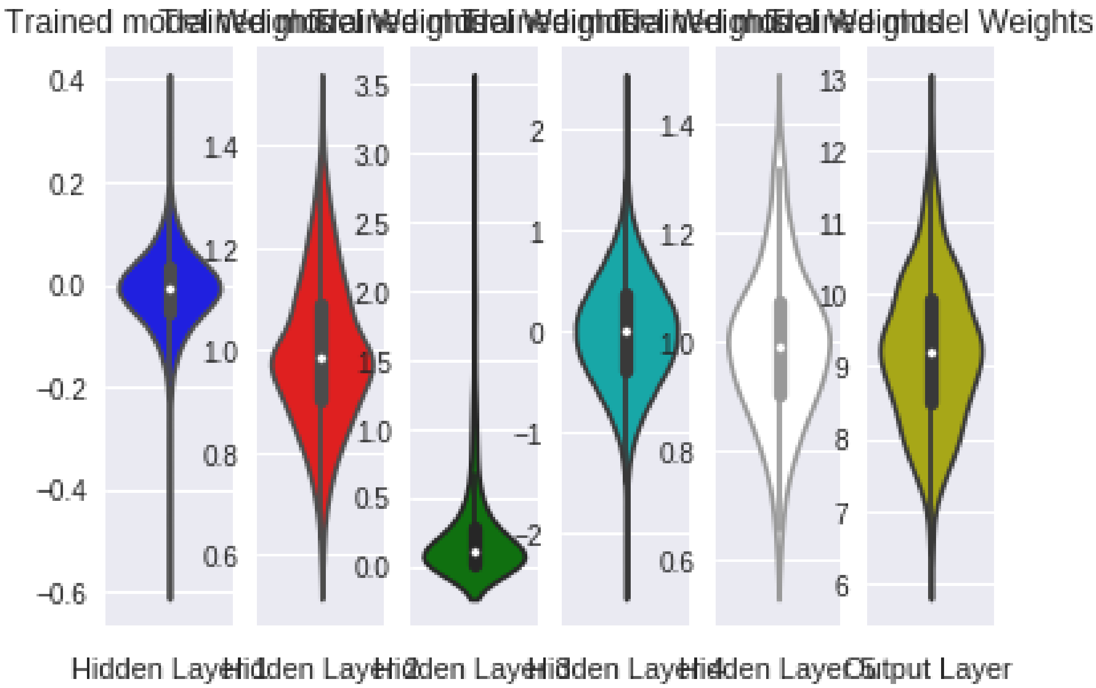
```
    ax.set_xlabel('epoch')
    ax.set_ylabel('Categorical Crossentropy Loss')

    # 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, v

    # 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 e

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

```
Test score: 0.07801552013880574
Test accuracy: 0.9801
```



```
In [0]: w_after = model.get_weights()

        h1_w = w_after[0].flatten().reshape(-1,1)
        h2_w = w_after[2].flatten().reshape(-1,1)
        h3_w = w_after[4].flatten().reshape(-1,1)
        h4_w = w_after[6].flatten().reshape(-1,1)
        h5_w = w_after[8].flatten().reshape(-1,1)
        out_w = w_after[10].flatten().reshape(-1,1)


        fig = plt.figure()
        plt.title("Weight matrices after model trained")
        plt.subplot(1, 6, 1)
        plt.title("Trained model Weights")
        ax = sns.violinplot(y=h1_w,color='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='g')
        plt.xlabel('Hidden Layer 3')

        plt.subplot(1, 6, 4)
        plt.title("Trained model Weights")
        ax = sns.violinplot(y=h4_w, color='c')
        plt.xlabel('Hidden Layer 4')

        plt.subplot(1, 6, 5)
        plt.title("Trained model Weights")
        ax = sns.violinplot(y=h5_w, color='w')
        plt.xlabel('Hidden Layer 5')

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

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:588: FutureWarning: remove_na is
  kde_data = remove_na(group_data)
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:816: FutureWarning: remove_na is
  violin_data = remove_na(group_data)
```



Observations

1. As we can see when apply bn + dropout each and every model performs well.

Conclusions 1. We perform various different types of model architecture and with different layers and also plotted train and validation error graph. We observe that when applied BN + Dropout with three different architecture models perfoms very well. And, when apply only dropout with 5 hidden layer, BN with 3 hidden layer, it works worse than all. 2. We also check for the weights of model and found that weights are slightly small in BN and Droupout, when applied individually. Other than this, model with only I/O layer also performs quite well. 3. We can also see the model performence chart in below table.

```
In [0]: from prettytable import PrettyTable

        x = PrettyTable()

        x.field_names = ["MODEL", "ACCURACY"]

        x.add_row(["MLP with only I/O layer", score_in_out[1]])
        x.add_row(["MLP with 2 hidden layer", score_relu2_adam[1]])
        x.add_row(["MLP with 2 hidden layer", score_relu2_adam_diff[1]])
        x.add_row(["MLP with 3 hidden layer", score_relu3_adam[1]])
        x.add_row(["MLP + BN with 2 hidden layer", score_relu5_adam[1]])
        x.add_row(["MLP + BN with 2 hidden layer", score_relu_bn2[1]])
        x.add_row(["MLP + BN with 3 hidden layer", score_relu_bn2_diff[1]])
        x.add_row(["MLP + BN with 5 hidden layer", score_relu_bn3[1]])
```

```
x.add_row(["MLP + Dropout with 2 hidden layer", score_relu_bn5[1]])
x.add_row(["MLP + Dropout with 2 hidden layer", score_relu_drop2[1]])
x.add_row(["MLP + Dropout with 3 hidden layer", score_relu_drop3[1]])
x.add_row(["MLP + Dropout with 5 hidden layer", score_relu_drop5[1]])
x.add_row(["MLP + BN + Dropout with 2 hidden layers", score_relu_bn_drop2[1]])
x.add_row(["MLP + BN + Dropout with 2 hidden layers", score_relu_bn_drop2_diff[1]])
x.add_row(["MLP + BN + Dropout with 3 hidden layers", score_relu_bn_drop3[1]])
x.add_row(["MLP + BN + Dropout with 5 hidden layers", score_relu_bn_drop5[1]])


print(x)
```

```
+-----------------------------------------+----------+
|                  MODEL                  | ACCURACY |
+-----------------------------------------+----------+
|          MLP with only I/O layer        |  0.9093  |
|          MLP with 2 hidden layer        |  0.9807  |
|          MLP with 2 hidden layer        |  0.9845  |
|          MLP with 3 hidden layer        |  0.9773  |
|        MLP + BN with 2 hidden layer     |  0.9787  |
|        MLP + BN with 2 hidden layer     |  0.9796  |
|        MLP + BN with 3 hidden layer     |  0.8002  |
|        MLP + BN with 5 hidden layer     |  0.9798  |
|     MLP + Dropout with 2 hidden layer   |  0.9815  |
|     MLP + Dropout with 2 hidden layer   |  0.9783  |
|     MLP + Dropout with 3 hidden layer   |  0.9328  |
|     MLP + Dropout with 5 hidden layer   |  0.0982  |
| MLP + BN + Dropout with 2 hidden layers |  0.9831  |
| MLP + BN + Dropout with 2 hidden layers |  0.982   |
| MLP + BN + Dropout with 3 hidden layers |  0.9824  |
| MLP + BN + Dropout with 5 hidden layers |  0.9801  |
+-----------------------------------------+----------+
```