TensorFlow and Keras Build various MLP architectures for MNIST dataset

In [2]:

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

In [3]:

```
%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

In [4]:

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

In [5]:

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

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

In [6]:

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

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

In [7]:

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

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

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

In [8]:

```
# 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 [9]:

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

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

In [10]:

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

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

```
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         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 [11]:

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

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

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

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

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

Softmax classifier

In [12]:

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

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

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

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

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

###

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

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

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

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

####

# https://keras.io/activations/

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

# from keras.layers import Activation, Dense

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

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

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


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

In [14]:

```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

In [16]:

```
# 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 [19]:

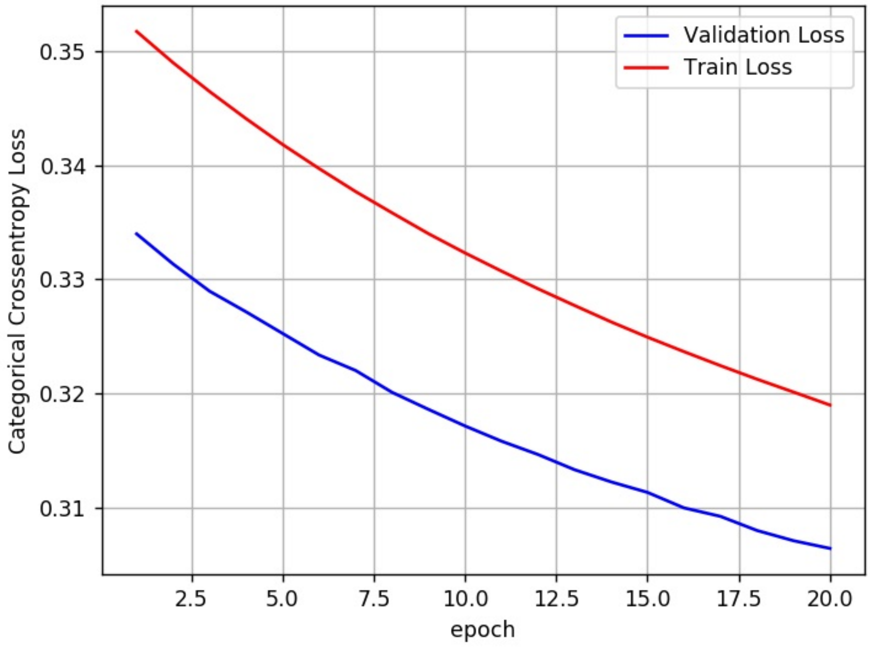
```
# Before training a model, you need to configure the learning process, which is done via the compile method

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


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

# that is why we converted out labels into vectors

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

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

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

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

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

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

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.3517 - acc: 0.9026 - val_loss:
0.3340 - val_acc: 0.9095
Epoch 2/20
60000/60000 [==============================] - 4s 60us/step - loss: 0.3490 - acc: 0.9030 - val_loss:
0.3313 - val_acc: 0.9097
Epoch 3/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.3464 - acc: 0.9039 - val_loss:
0.3289 - val_acc: 0.9101
Epoch 4/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.3441 - acc: 0.9041 - val_loss:
0.3271 - val_acc: 0.9113
Epoch 5/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.3418 - acc: 0.9045 - val_loss:
0.3253 - val_acc: 0.9114
Epoch 6/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.3397 - acc: 0.9054 - val_loss:
0.3234 - val_acc: 0.9116
Epoch 7/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.3377 - acc: 0.9056 - val_loss:
0.3220 - val_acc: 0.9111
Epoch 8/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.3358 - acc: 0.9062 - val_loss:
0.3201 - val_acc: 0.9125
Epoch 9/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.3340 - acc: 0.9064 - val_loss:
0.3186 - val_acc: 0.9125
Epoch 10/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.3323 - acc: 0.9075 - val_loss:
0.3171 - val_acc: 0.9129
Epoch 11/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.3307 - acc: 0.9076 - val_loss:
0.3158 - val_acc: 0.9135
Epoch 12/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.3292 - acc: 0.9081 - val_loss:
0.3147 - val_acc: 0.9136
Epoch 13/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.3277 - acc: 0.9086 - val_loss:
0.3133 - val_acc: 0.9137
Epoch 14/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.3263 - acc: 0.9089 - val_loss:
0.3123 - val_acc: 0.9143
Epoch 15/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.3249 - acc: 0.9096 - val_loss:
0.3113 - val_acc: 0.9142
Epoch 16/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.3237 - acc: 0.9101 - val_loss
: 0.3100 - val_acc: 0.9147
Epoch 17/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.3224 - acc: 0.9100 - val_loss:
0.3092 - val_acc: 0.9143
Epoch 18/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.3212 - acc: 0.9105 - val_loss:
0.3080 - val_acc: 0.9146
Epoch 19/20
60000/60000 [==============================] - 4s 62us/step - loss: 0.3201 - acc: 0.9108 - val_loss:
0.3071 - val_acc: 0.9148
Epoch 20/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.3190 - acc: 0.9109 - val_loss:
0.3064 - val_acc: 0.9148
```

In [20]:

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

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

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

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

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

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

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

```
Test score: 0.30642640863657
Test accuracy: 0.9148
```



MLP + Sigmoid activation + SGDOptimizer

In [21]:

```
# Multilayer perceptron

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

model_sigmoid.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_3 (Dense)              (None, 512)               401920    
_________________________________________________________________
dense_4 (Dense)              (None, 128)               65664     
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1290      
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
```

In [22]:

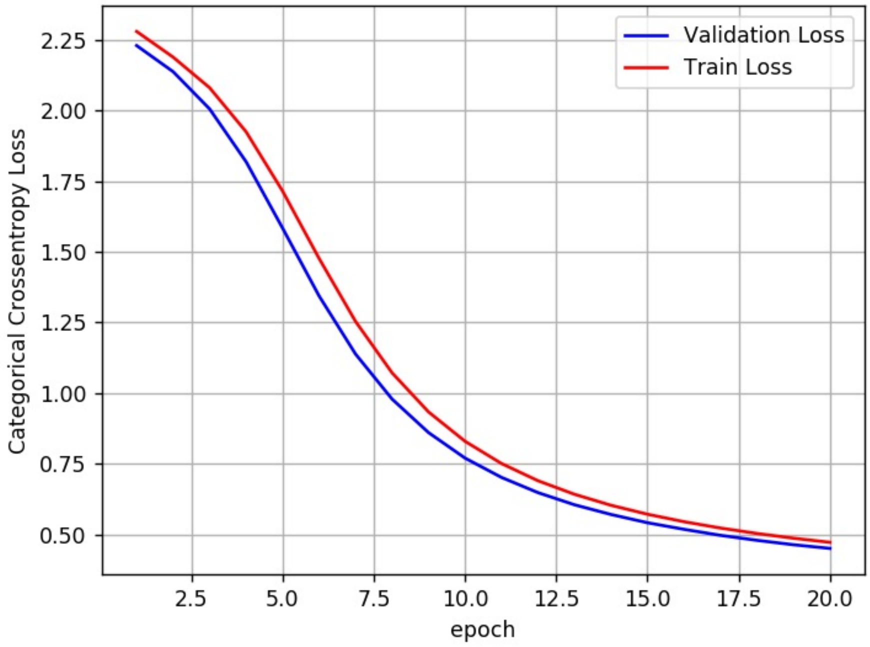
```
model_sigmoid.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 17s 288us/step - loss: 2.2790 - acc: 0.2040 - val_los
s: 2.2290 - val_acc: 0.3838
Epoch 2/20
60000/60000 [==============================] - 13s 213us/step - loss: 2.1881 - acc: 0.4278 - val_los
s: 2.1367 - val_acc: 0.5669
Epoch 3/20
60000/60000 [==============================] - 13s 208us/step - loss: 2.0800 - acc: 0.5721 - val_los
s: 2.0050 - val_acc: 0.6305
Epoch 4/20
60000/60000 [==============================] - 14s 237us/step - loss: 1.9242 - acc: 0.6430 - val_los
s: 1.8182 - val_acc: 0.7021
Epoch 5/20
60000/60000 [==============================] - 13s 220us/step - loss: 1.7145 - acc: 0.6899 - val_los
s: 1.5834 - val_acc: 0.7200
Epoch 6/20
60000/60000 [==============================] - 19s 323us/step - loss: 1.4763 - acc: 0.7325 - val_los
s: 1.3437 - val_acc: 0.7585
Epoch 7/20
60000/60000 [==============================] - 11s 180us/step - loss: 1.2534 - acc: 0.7681 - val_los
s: 1.1381 - val_acc: 0.7845
Epoch 8/20
60000/60000 [==============================] - 9s 152us/step - loss: 1.0719 - acc: 0.7916 - val_loss
: 0.9791 - val_acc: 0.8075
Epoch 9/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.9336 - acc: 0.8088 - val_los
s: 0.8605 - val_acc: 0.8235
Epoch 10/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.8298 - acc: 0.8212 - val_loss
: 0.7704 - val_acc: 0.8356
Epoch 11/20
60000/60000 [==============================] - 11s 180us/step - loss: 0.7511 - acc: 0.8311 - val_los
s: 0.7021 - val_acc: 0.8455
Epoch 12/20
60000/60000 [==============================] - 8s 142us/step - loss: 0.6902 - acc: 0.8399 - val_loss
: 0.6482 - val_acc: 0.8518
Epoch 13/20
60000/60000 [==============================] - 12s 201us/step - loss: 0.6423 - acc: 0.8466 - val_los
s: 0.6053 - val_acc: 0.8588
Epoch 14/20
60000/60000 [==============================] - 10s 173us/step - loss: 0.6035 - acc: 0.8525 - val_los
s: 0.5710 - val_acc: 0.8640
Epoch 15/20
60000/60000 [==============================] - 13s 216us/step - loss: 0.5719 - acc: 0.8574 - val_los
s: 0.5418 - val_acc: 0.8665
Epoch 16/20
60000/60000 [==============================] - 14s 238us/step - loss: 0.5454 - acc: 0.8612 - val_los
s: 0.5185 - val_acc: 0.8697
Epoch 17/20
60000/60000 [==============================] - 12s 201us/step - loss: 0.5230 - acc: 0.8657 - val_los
s: 0.4973 - val_acc: 0.8744
Epoch 18/20
60000/60000 [==============================] - 10s 165us/step - loss: 0.5038 - acc: 0.8692 - val_los
s: 0.4796 - val_acc: 0.8767
Epoch 19/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.4871 - acc: 0.8724 - val_los
s: 0.4640 - val_acc: 0.8801
Epoch 20/20
60000/60000 [==============================] - 10s 162us/step - loss: 0.4725 - acc: 0.8750 - val_los
s: 0.4510 - val_acc: 0.8820
```

In [23]:

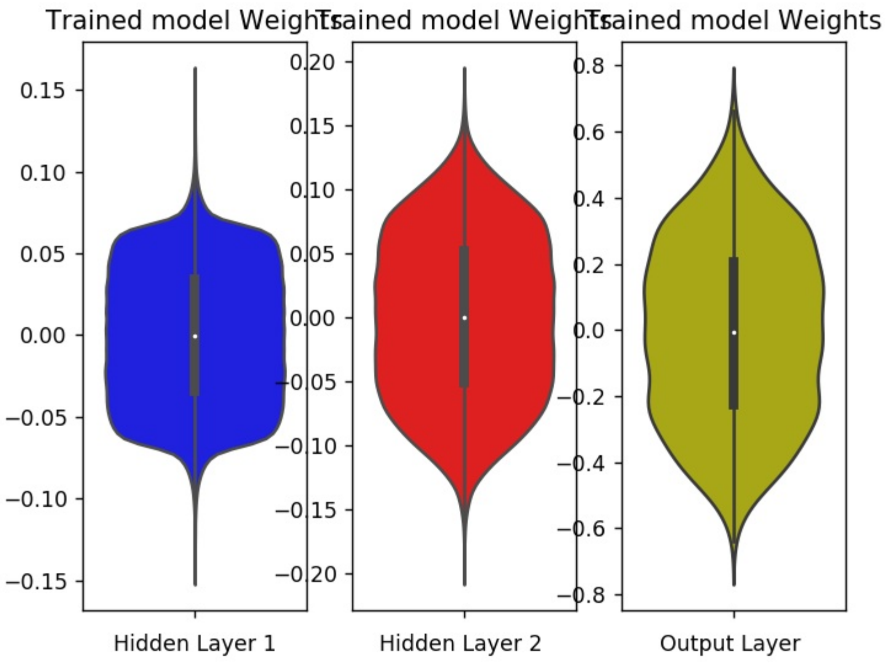
```
score = model_sigmoid.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

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

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

```
Test score: 0.4510295778989792
Test accuracy: 0.882
```



In [24]:

```
w_after = model_sigmoid.get_weights()

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


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

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

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



MLP + Sigmoid activation + ADAM

In [25]:

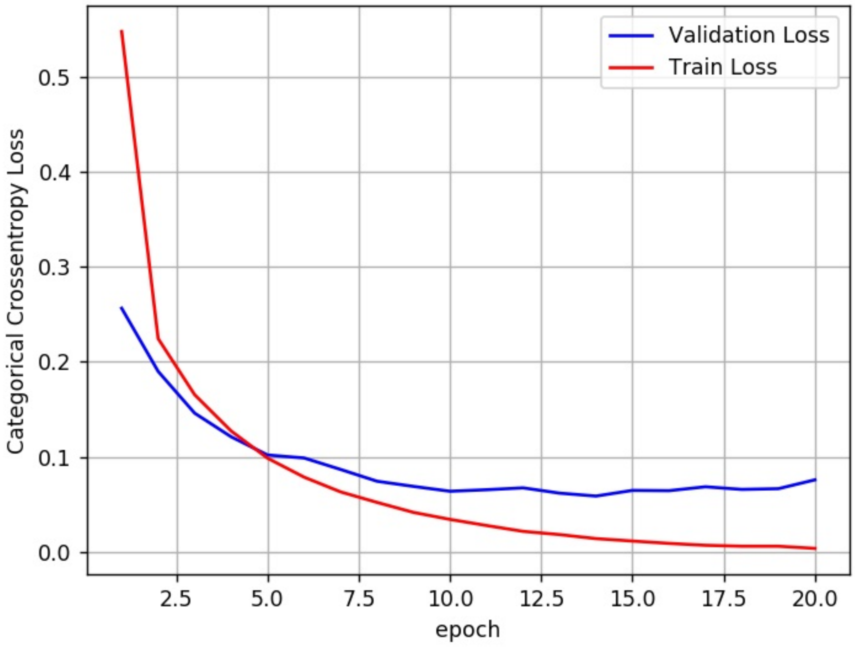
```
model_sigmoid = Sequential()
model_sigmoid.add(Dense(512, activation='sigmoid', input_shape=(input_dim,)))
model_sigmoid.add(Dense(128, activation='sigmoid'))
model_sigmoid.add(Dense(output_dim, activation='softmax'))

model_sigmoid.summary()

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

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_7 (Dense)              (None, 128)               65664
_________________________________________________________________
dense_8 (Dense)              (None, 10)                1290
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 22s 370us/step - loss: 0.5477 - acc: 0.8564 - val_los
s: 0.2563 - val_acc: 0.9257
Epoch 2/20
60000/60000 [==============================] - 18s 306us/step - loss: 0.2243 - acc: 0.9345 - val_los
s: 0.1899 - val_acc: 0.9413
Epoch 3/20
60000/60000 [==============================] - 16s 266us/step - loss: 0.1653 - acc: 0.9513 - val_los
s: 0.1459 - val_acc: 0.9558
Epoch 4/20
60000/60000 [==============================] - 15s 256us/step - loss: 0.1271 - acc: 0.9629 - val_los
s: 0.1210 - val_acc: 0.9634
Epoch 5/20
60000/60000 [==============================] - 13s 218us/step - loss: 0.0984 - acc: 0.9711 - val_los
s: 0.1018 - val_acc: 0.9679
Epoch 6/20
60000/60000 [==============================] - 12s 203us/step - loss: 0.0785 - acc: 0.9764 - val_los
s: 0.0987 - val_acc: 0.9699
Epoch 7/20
60000/60000 [==============================] - 12s 207us/step - loss: 0.0630 - acc: 0.9811 - val_los
s: 0.0866 - val_acc: 0.9729
Epoch 8/20
60000/60000 [==============================] - 13s 210us/step - loss: 0.0519 - acc: 0.9847 - val_los
s: 0.0742 - val_acc: 0.9769
Epoch 9/20
60000/60000 [==============================] - 14s 231us/step - loss: 0.0414 - acc: 0.9880 - val_los
s: 0.0688 - val_acc: 0.9783
Epoch 10/20
60000/60000 [==============================] - 16s 262us/step - loss: 0.0340 - acc: 0.9904 - val_los
s: 0.0637 - val_acc: 0.9808
Epoch 11/20
60000/60000 [==============================] - 17s 282us/step - loss: 0.0276 - acc: 0.9926 - val_los
s: 0.0653 - val_acc: 0.9789
Epoch 12/20
60000/60000 [==============================] - 17s 277us/step - loss: 0.0215 - acc: 0.9947 - val_los
s: 0.0673 - val_acc: 0.9801
Epoch 13/20
60000/60000 [==============================] - 17s 283us/step - loss: 0.0181 - acc: 0.9951 - val_los
s: 0.0617 - val_acc: 0.98105 - ETA: 5s - loss: 0.0178   - ETA: 1s - loss: 0.0180 -  - ETA: 1s - loss
: 0.0182
Epoch 14/20
60000/60000 [==============================] - 13s 214us/step - loss: 0.0138 - acc: 0.9970 - val_los
s: 0.0586 - val_acc: 0.9819
Epoch 15/20
60000/60000 [==============================] - 15s 245us/step - loss: 0.0113 - acc: 0.9974 - val_los
s: 0.0646 - val_acc: 0.9813
Epoch 16/20
60000/60000 [==============================] - 15s 254us/step - loss: 0.0088 - acc: 0.9982 - val_los
s: 0.0643 - val_acc: 0.9809
Epoch 17/20
60000/60000 [==============================] - 14s 234us/step - loss: 0.0068 - acc: 0.9987 - val_los
s: 0.0684 - val_acc: 0.9810
Epoch 18/20
60000/60000 [==============================] - 16s 259us/step - loss: 0.0058 - acc: 0.9988 - val_los
s: 0.0656 - val_acc: 0.98050059 - acc:
Epoch 19/20
60000/60000 [==============================] - 17s 277us/step - loss: 0.0058 - acc: 0.9987 - val_los
s: 0.0664 - val_acc: 0.9817
Epoch 20/20
60000/60000 [==============================] - 17s 277us/step - loss: 0.0035 - acc: 0.9994 - val_los
s: 0.0757 - val_acc: 0.9806
```

In [26]:

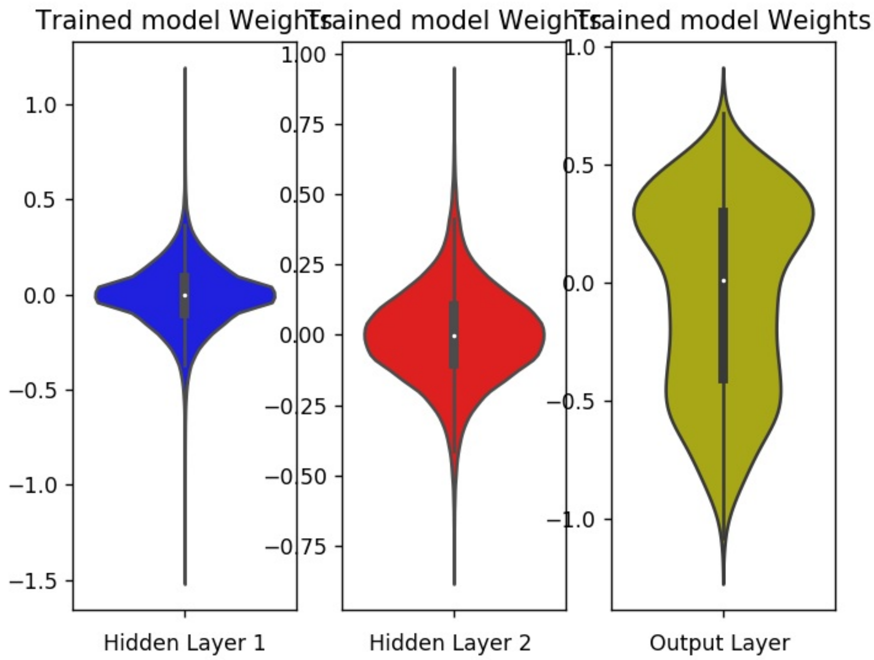
```
score = model_sigmoid.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

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

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

```
Test score: 0.07570469840772276
Test accuracy: 0.9806
```



In [27]:

```
w_after = model_sigmoid.get_weights()

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


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

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

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



MLP + ReLU +SGD

In [28]:

```
# Multilayer perceptron

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

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

model_relu.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_9 (Dense)              (None, 512)               401920    
_________________________________________________________________
dense_10 (Dense)             (None, 128)               65664     
_________________________________________________________________
dense_11 (Dense)             (None, 10)                1290      
=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
_________________________________________________________________
```

In [29]:

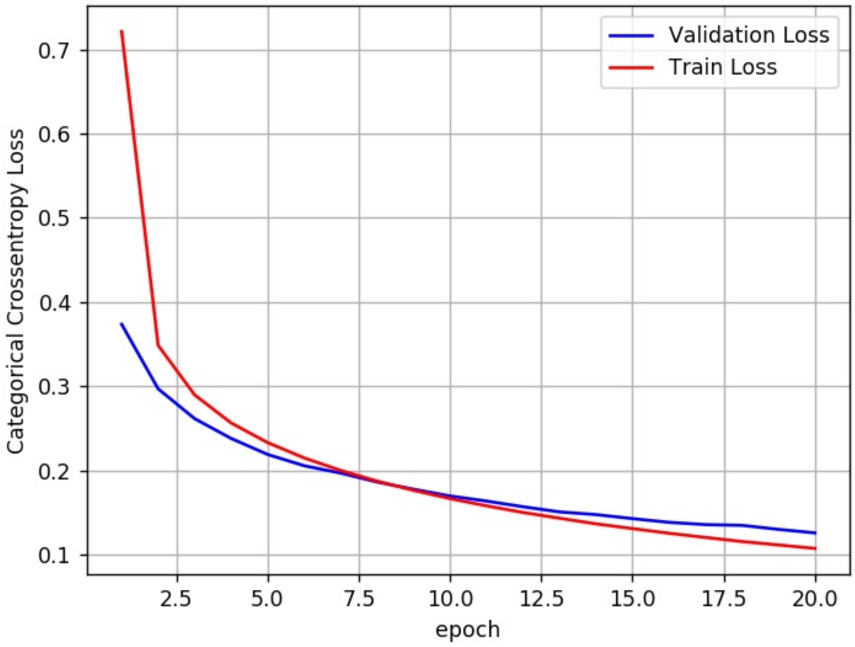
```
model_relu.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_
test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 18s 307us/step - loss: 0.7215 - acc: 0.7982 - val_los
s: 0.3736 - val_acc: 0.9006
Epoch 2/20
60000/60000 [==============================] - 14s 237us/step - loss: 0.3485 - acc: 0.9010 - val_los
s: 0.2969 - val_acc: 0.9170
Epoch 3/20
60000/60000 [==============================] - 14s 226us/step - loss: 0.2897 - acc: 0.9175 - val_los
s: 0.2614 - val_acc: 0.9263
Epoch 4/20
60000/60000 [==============================] - 11s 190us/step - loss: 0.2563 - acc: 0.9264 - val_los
s: 0.2379 - val_acc: 0.9339
Epoch 5/20
60000/60000 [==============================] - 11s 184us/step - loss: 0.2330 - acc: 0.9330 - val_los
s: 0.2189 - val_acc: 0.9397
Epoch 6/20
60000/60000 [==============================] - 11s 178us/step - loss: 0.2148 - acc: 0.9385 - val_los
s: 0.2055 - val_acc: 0.9422
Epoch 7/20
60000/60000 [==============================] - 11s 175us/step - loss: 0.1999 - acc: 0.9430 - val_los
s: 0.1968 - val_acc: 0.9444
Epoch 8/20
60000/60000 [==============================] - 11s 186us/step - loss: 0.1872 - acc: 0.9464 - val_los
s: 0.1861 - val_acc: 0.9467
Epoch 9/20
60000/60000 [==============================] - 13s 218us/step - loss: 0.1762 - acc: 0.9498 - val_los
s: 0.1775 - val_acc: 0.9489
Epoch 10/20
60000/60000 [==============================] - 12s 206us/step - loss: 0.1664 - acc: 0.9530 - val_los
s: 0.1695 - val_acc: 0.9507
Epoch 11/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.1578 - acc: 0.9552 - val_los
s: 0.1635 - val_acc: 0.9521
Epoch 12/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.1500 - acc: 0.9572 - val_los
s: 0.1568 - val_acc: 0.9542
Epoch 13/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.1431 - acc: 0.9592 - val_los
s: 0.1506 - val_acc: 0.9547
Epoch 14/20
60000/60000 [==============================] - 10s 163us/step - loss: 0.1364 - acc: 0.9613 - val_los
s: 0.1474 - val_acc: 0.9558
Epoch 15/20
60000/60000 [==============================] - 10s 168us/step - loss: 0.1308 - acc: 0.9629 - val_los
s: 0.1426 - val_acc: 0.9578
Epoch 16/20
60000/60000 [==============================] - 10s 163us/step - loss: 0.1253 - acc: 0.9649 - val_los
s: 0.1382 - val_acc: 0.9600
Epoch 17/20
60000/60000 [==============================] - 10s 166us/step - loss: 0.1203 - acc: 0.9663 - val_los
s: 0.1355 - val_acc: 0.9600
Epoch 18/20
60000/60000 [==============================] - 10s 173us/step - loss: 0.1154 - acc: 0.9676 - val_los
s: 0.1346 - val_acc: 0.9603
Epoch 19/20
60000/60000 [==============================] - 11s 182us/step - loss: 0.1113 - acc: 0.9688 - val_los
s: 0.1299 - val_acc: 0.9614
Epoch 20/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.1072 - acc: 0.9705 - val_los
s: 0.1256 - val_acc: 0.9634
```

In [30]:

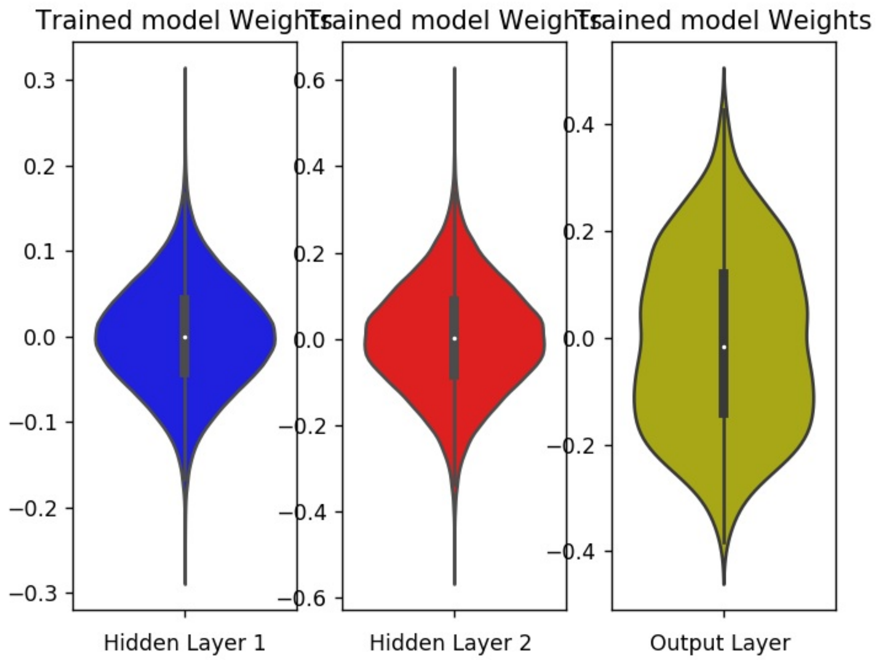
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

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

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

```
Test score: 0.1256313980385661
Test accuracy: 0.9634
```



In [31]:

```
w_after = model_relu.get_weights()

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


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

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

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



MLP + ReLU + ADAM

In [32]:

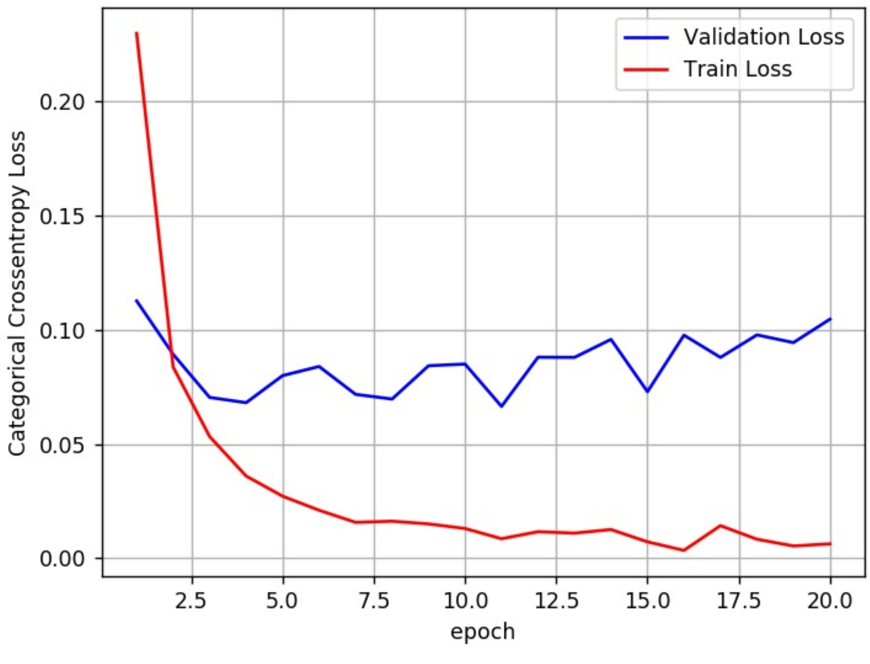
```
model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None))
)
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_12 (Dense)             (None, 512)               401920    
_________________________________________________________________
dense_13 (Dense)             (None, 128)               65664     
_________________________________________________________________
dense_14 (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 [==============================] - 19s 324us/step - loss: 0.2299 - acc: 0.9316 - val_los
s: 0.1127 - val_acc: 0.9641 1s - loss: 0.2427 - acc: 0.927 - ETA: 1s - loss:
Epoch 2/20
60000/60000 [==============================] - 15s 252us/step - loss: 0.0837 - acc: 0.9749 - val_los
s: 0.0893 - val_acc: 0.9718
Epoch 3/20
60000/60000 [==============================] - 15s 245us/step - loss: 0.0533 - acc: 0.9838 - val_los
s: 0.0704 - val_acc: 0.9778
Epoch 4/20
60000/60000 [==============================] - 15s 255us/step - loss: 0.0360 - acc: 0.9887 - val_los
s: 0.0681 - val_acc: 0.9784
Epoch 5/20
60000/60000 [==============================] - 15s 246us/step - loss: 0.0271 - acc: 0.9915 - val_los
s: 0.0800 - val_acc: 0.9759
Epoch 6/20
60000/60000 [==============================] - 12s 194us/step - loss: 0.0210 - acc: 0.9932 - val_los
s: 0.0839 - val_acc: 0.9769
Epoch 7/20
60000/60000 [==============================] - 14s 228us/step - loss: 0.0157 - acc: 0.9947 - val_los
s: 0.0717 - val_acc: 0.9798
Epoch 8/20
60000/60000 [==============================] - 13s 220us/step - loss: 0.0162 - acc: 0.9945 - val_los
s: 0.0697 - val_acc: 0.9817
Epoch 9/20
60000/60000 [==============================] - 14s 227us/step - loss: 0.0150 - acc: 0.9956 - val_los
s: 0.0843 - val_acc: 0.9782
Epoch 10/20
60000/60000 [==============================] - 13s 216us/step - loss: 0.0130 - acc: 0.9956 - val_los
s: 0.0850 - val_acc: 0.9779
Epoch 11/20
60000/60000 [==============================] - 13s 215us/step - loss: 0.0085 - acc: 0.9974 - val_los
s: 0.0664 - val_acc: 0.9836
Epoch 12/20
60000/60000 [==============================] - 14s 233us/step - loss: 0.0115 - acc: 0.9960 - val_los
s: 0.0880 - val_acc: 0.9797
Epoch 13/20
60000/60000 [==============================] - 14s 238us/step - loss: 0.0109 - acc: 0.9964 - val_los
s: 0.0879 - val_acc: 0.9803
Epoch 14/20
60000/60000 [==============================] - 20s 340us/step - loss: 0.0125 - acc: 0.9957 - val_los
s: 0.0958 - val_acc: 0.9782
Epoch 15/20
60000/60000 [==============================] - 15s 256us/step - loss: 0.0071 - acc: 0.9976 - val_los
s: 0.0729 - val_acc: 0.9838
Epoch 16/20
60000/60000 [==============================] - 18s 293us/step - loss: 0.0034 - acc: 0.9989 - val_los
s: 0.0977 - val_acc: 0.9803 - ETA: 9s -  - ETA: 8s - loss - ETA: 6s - loss: 0.0023 - acc: 0.999 - ET
A: 6s - loss: 0.002 - ETA: 1s
Epoch 17/20
60000/60000 [==============================] - 17s 281us/step - loss: 0.0142 - acc: 0.9952 - val_los
s: 0.0879 - val_acc: 0.9814
Epoch 18/20
60000/60000 [==============================] - 19s 316us/step - loss: 0.0083 - acc: 0.9975 - val_los
s: 0.0978 - val_acc: 0.9788
Epoch 19/20
60000/60000 [==============================] - 16s 275us/step - loss: 0.0053 - acc: 0.9983 - val_los
s: 0.0944 - val_acc: 0.9812
Epoch 20/20
60000/60000 [==============================] - 16s 274us/step - loss: 0.0062 - acc: 0.9980 - val_los
s: 0.1047 - val_acc: 0.9801
```

In [33]:

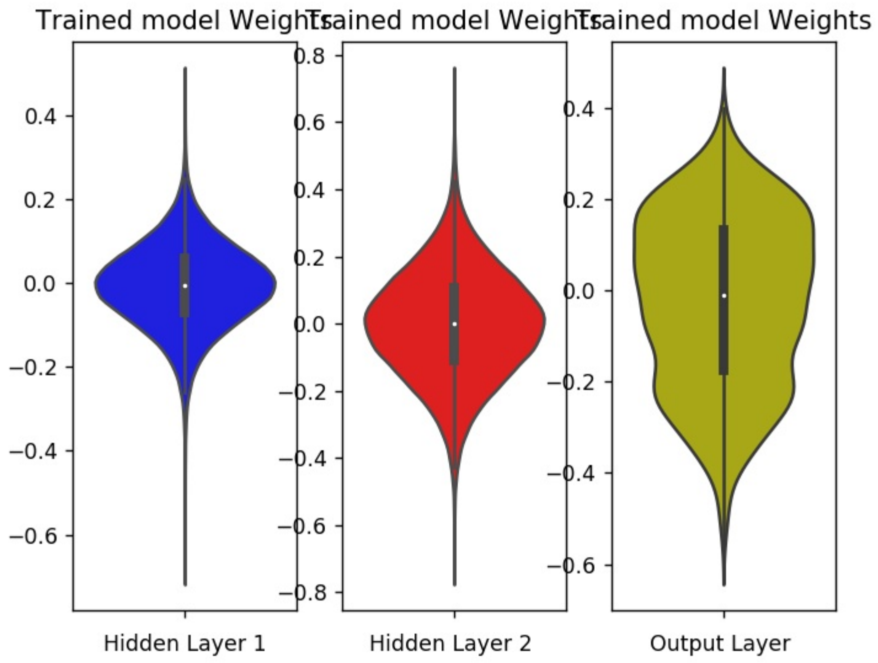
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

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


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

```
Test score: 0.10466666469420727
Test accuracy: 0.9801
```



In [34]:

```
w_after = model_relu.get_weights()

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


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

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

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



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

In [35]:

```
# Multilayer perceptron

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

from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_batch.add(Dense(512, activation='sigmoid', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0
.0, stddev=0.039, seed=None)))
model_batch.add(BatchNormalization())

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

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


model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_15 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048      
_________________________________________________________________
dense_16 (Dense)             (None, 128)               65664     
_________________________________________________________________
batch_normalization_2 (Batch (None, 128)               512       
_________________________________________________________________
dense_17 (Dense)             (None, 10)                1290      
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

In [36]:

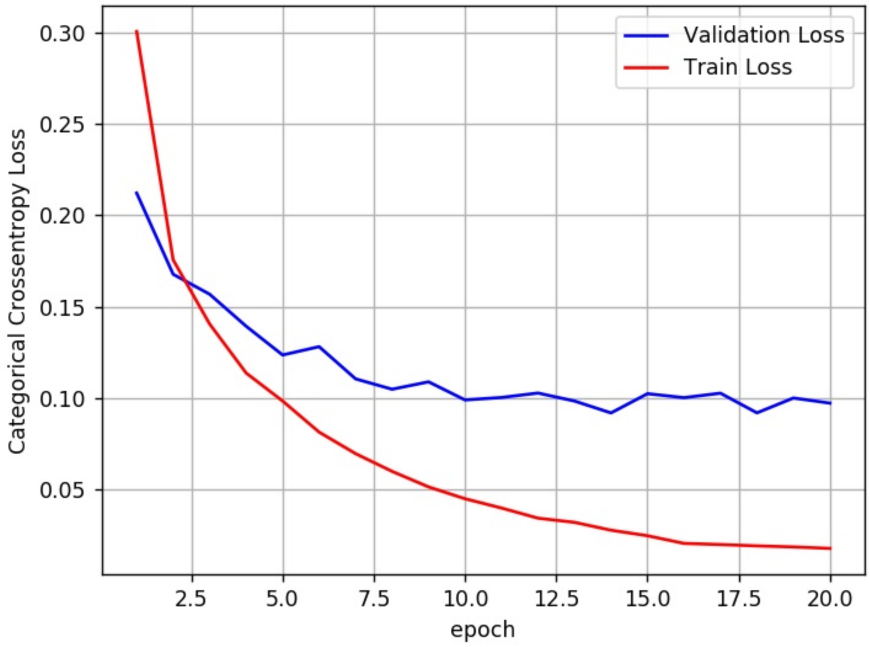
```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X
_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 29s 486us/step - loss: 0.3008 - acc: 0.9113 - val_los
s: 0.2122 - val_acc: 0.9382
Epoch 2/20
60000/60000 [==============================] - 33s 542us/step - loss: 0.1756 - acc: 0.9489 - val_los
s: 0.1677 - val_acc: 0.9516
Epoch 3/20
60000/60000 [==============================] - 26s 440us/step - loss: 0.1405 - acc: 0.9592 - val_los
s: 0.1568 - val_acc: 0.9538
Epoch 4/20
60000/60000 [==============================] - 21s 352us/step - loss: 0.1136 - acc: 0.9667 - val_los
s: 0.1392 - val_acc: 0.9581
Epoch 5/20
60000/60000 [==============================] - 24s 392us/step - loss: 0.0983 - acc: 0.9701 - val_los
s: 0.1235 - val_acc: 0.9632
Epoch 6/20
60000/60000 [==============================] - 20s 332us/step - loss: 0.0812 - acc: 0.9748 - val_los
s: 0.1280 - val_acc: 0.96100 - acc:
Epoch 7/20
60000/60000 [==============================] - 19s 324us/step - loss: 0.0694 - acc: 0.9788 - val_los
s: 0.1104 - val_acc: 0.9671
Epoch 8/20
60000/60000 [==============================] - 24s 404us/step - loss: 0.0597 - acc: 0.9812 - val_los
s: 0.1047 - val_acc: 0.9686
Epoch 9/20
60000/60000 [==============================] - 15s 244us/step - loss: 0.0512 - acc: 0.9837 - val_los
s: 0.1087 - val_acc: 0.9684
Epoch 10/20
60000/60000 [==============================] - 20s 337us/step - loss: 0.0447 - acc: 0.9853 - val_los
s: 0.0988 - val_acc: 0.9714
Epoch 11/20
60000/60000 [==============================] - 20s 340us/step - loss: 0.0396 - acc: 0.9876 - val_los
s: 0.1002 - val_acc: 0.9697
Epoch 12/20
60000/60000 [==============================] - 18s 301us/step - loss: 0.0340 - acc: 0.9889 - val_los
s: 0.1026 - val_acc: 0.9695
Epoch 13/20
60000/60000 [==============================] - 19s 316us/step - loss: 0.0317 - acc: 0.9896 - val_los
s: 0.0982 - val_acc: 0.9703
Epoch 14/20
60000/60000 [==============================] - 19s 311us/step - loss: 0.0274 - acc: 0.9915 - val_los
s: 0.0917 - val_acc: 0.9730
Epoch 15/20
60000/60000 [==============================] - 20s 335us/step - loss: 0.0244 - acc: 0.9919 - val_los
s: 0.1022 - val_acc: 0.9731oss: 0
Epoch 16/20
60000/60000 [==============================] - 22s 365us/step - loss: 0.0202 - acc: 0.9937 - val_los
s: 0.1001 - val_acc: 0.9730
Epoch 17/20
60000/60000 [==============================] - 19s 325us/step - loss: 0.0196 - acc: 0.9937 - val_los
s: 0.1025 - val_acc: 0.9713
Epoch 18/20
60000/60000 [==============================] - 21s 354us/step - loss: 0.0188 - acc: 0.9939 - val_los
s: 0.0917 - val_acc: 0.97402s -
Epoch 19/20
60000/60000 [==============================] - 24s 408us/step - loss: 0.0183 - acc: 0.9941 - val_los
s: 0.0999 - val_acc: 0.9737
Epoch 20/20
60000/60000 [==============================] - 19s 315us/step - loss: 0.0175 - acc: 0.9939 - val_los
s: 0.0971 - val_acc: 0.9761
```

In [37]:

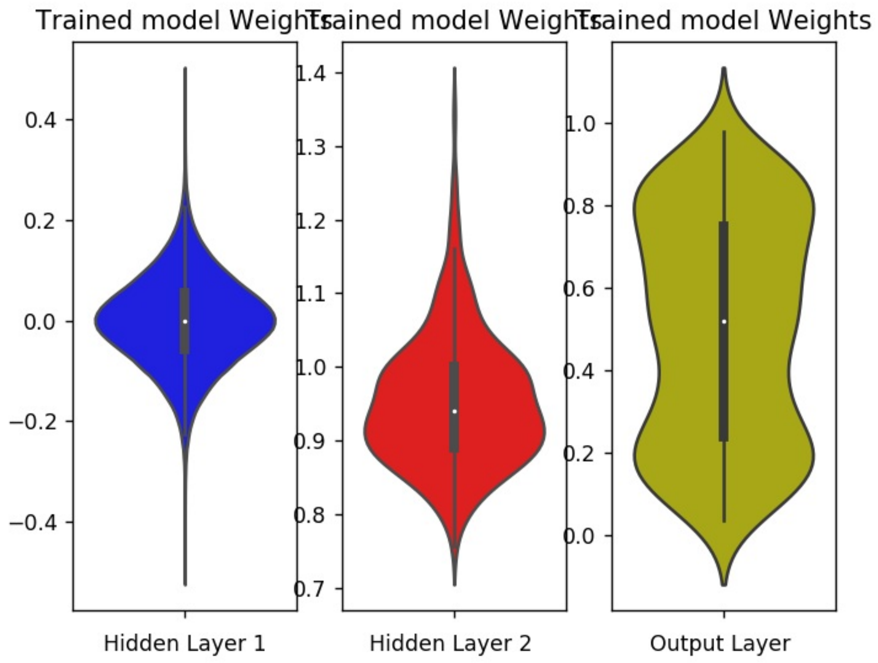
```
score = model_batch.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

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

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

```
Test score: 0.0970754934698838
Test accuracy: 0.9761
```



In [38]:

```
w_after = model_batch.get_weights()

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


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

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

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



5. MLP + Dropout + AdamOptimizer

In [39]:

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

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(512, activation='sigmoid', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.
0, stddev=0.039, seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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

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


model_drop.summary()
```

```
WARNING:tensorflow:From C:\Users\user\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.p
y:3445: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be
removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_18 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_3 (Batch (None, 512)               2048      
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_19 (Dense)             (None, 128)               65664     
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512       
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_20 (Dense)             (None, 10)                1290      
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

In [40]:

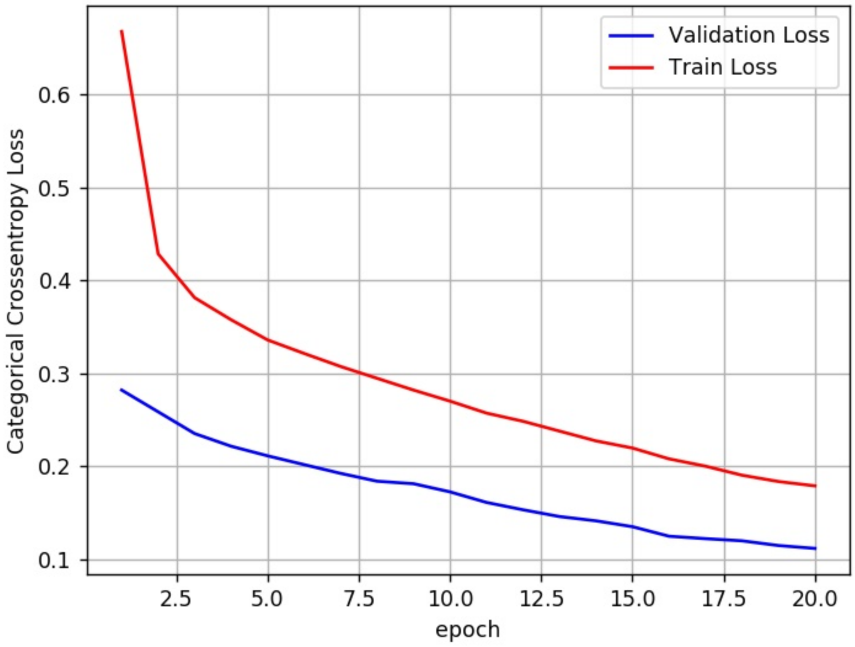
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_
test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 30s 499us/step - loss: 0.6681 - acc: 0.7933 - val_los
s: 0.2821 - val_acc: 0.9161
Epoch 2/20
60000/60000 [==============================] - 20s 335us/step - loss: 0.4287 - acc: 0.8693 - val_los
s: 0.2587 - val_acc: 0.9219
Epoch 3/20
60000/60000 [==============================] - 21s 348us/step - loss: 0.3816 - acc: 0.8843 - val_los
s: 0.2353 - val_acc: 0.9277
Epoch 4/20
60000/60000 [==============================] - 19s 323us/step - loss: 0.3578 - acc: 0.8914 - val_los
s: 0.2217 - val_acc: 0.9340
Epoch 5/20
60000/60000 [==============================] - 21s 351us/step - loss: 0.3361 - acc: 0.8992 - val_los
s: 0.2114 - val_acc: 0.9394
Epoch 6/20
60000/60000 [==============================] - 23s 376us/step - loss: 0.3216 - acc: 0.9020 - val_los
s: 0.2019 - val_acc: 0.9406
Epoch 7/20
60000/60000 [==============================] - 20s 335us/step - loss: 0.3075 - acc: 0.9063 - val_los
s: 0.1925 - val_acc: 0.9420
Epoch 8/20
60000/60000 [==============================] - 19s 320us/step - loss: 0.2948 - acc: 0.9108 - val_los
s: 0.1840 - val_acc: 0.9461
Epoch 9/20
60000/60000 [==============================] - 22s 369us/step - loss: 0.2821 - acc: 0.9159 - val_los
s: 0.1813 - val_acc: 0.9482
Epoch 10/20
60000/60000 [==============================] - 21s 351us/step - loss: 0.2702 - acc: 0.9185 - val_los
s: 0.1726 - val_acc: 0.9490
Epoch 11/20
60000/60000 [==============================] - 20s 333us/step - loss: 0.2573 - acc: 0.9222 - val_los
s: 0.1612 - val_acc: 0.9521
Epoch 12/20
60000/60000 [==============================] - 23s 380us/step - loss: 0.2485 - acc: 0.9247 - val_los
s: 0.1533 - val_acc: 0.9547
Epoch 13/20
60000/60000 [==============================] - 21s 356us/step - loss: 0.2380 - acc: 0.9290 - val_los
s: 0.1460 - val_acc: 0.9565
Epoch 14/20
60000/60000 [==============================] - 21s 357us/step - loss: 0.2274 - acc: 0.9308 - val_los
s: 0.1414 - val_acc: 0.9580
Epoch 15/20
60000/60000 [==============================] - 22s 365us/step - loss: 0.2197 - acc: 0.9335 - val_los
s: 0.1351 - val_acc: 0.9590
Epoch 16/20
60000/60000 [==============================] - 23s 385us/step - loss: 0.2081 - acc: 0.9379 - val_los
s: 0.1248 - val_acc: 0.9625
Epoch 17/20
60000/60000 [==============================] - 19s 320us/step - loss: 0.2002 - acc: 0.9394 - val_los
s: 0.1222 - val_acc: 0.9629
Epoch 18/20
60000/60000 [==============================] - 22s 371us/step - loss: 0.1905 - acc: 0.9429 - val_los
s: 0.1199 - val_acc: 0.9643
Epoch 19/20
60000/60000 [==============================] - 20s 336us/step - loss: 0.1837 - acc: 0.9454 - val_los
s: 0.1148 - val_acc: 0.9658
Epoch 20/20
60000/60000 [==============================] - 19s 323us/step - loss: 0.1790 - acc: 0.9464 - val_los
s: 0.1117 - val_acc: 0.9652
```

In [41]:

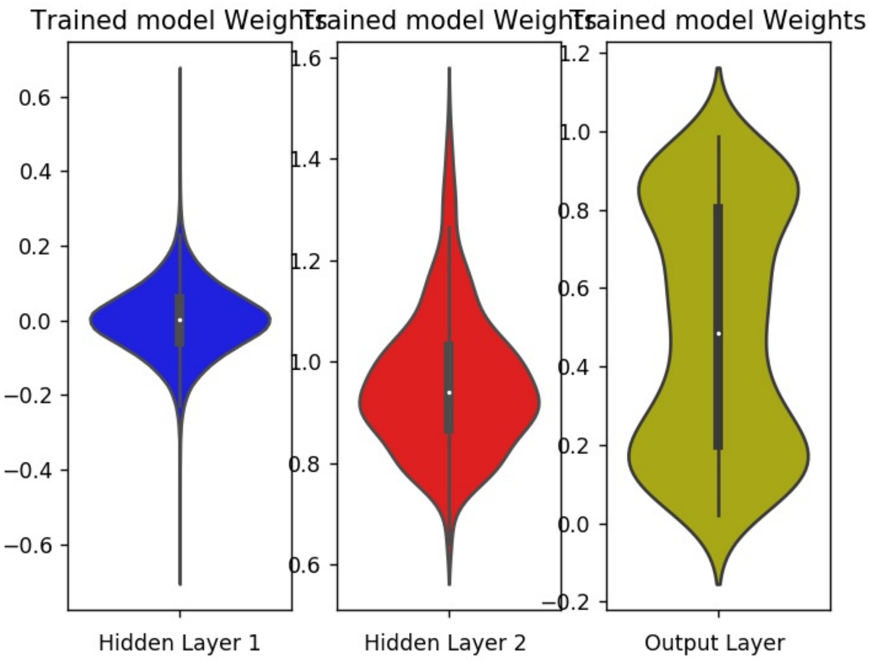
```
score = model_drop.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

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

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

```
Test score: 0.11170987114515156
Test accuracy: 0.9652
```



In [42]:

```
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()
```



Task 1: 2 hidden layers

MLP + ReLU + ADAM (784-360-50-10)

In [45]:

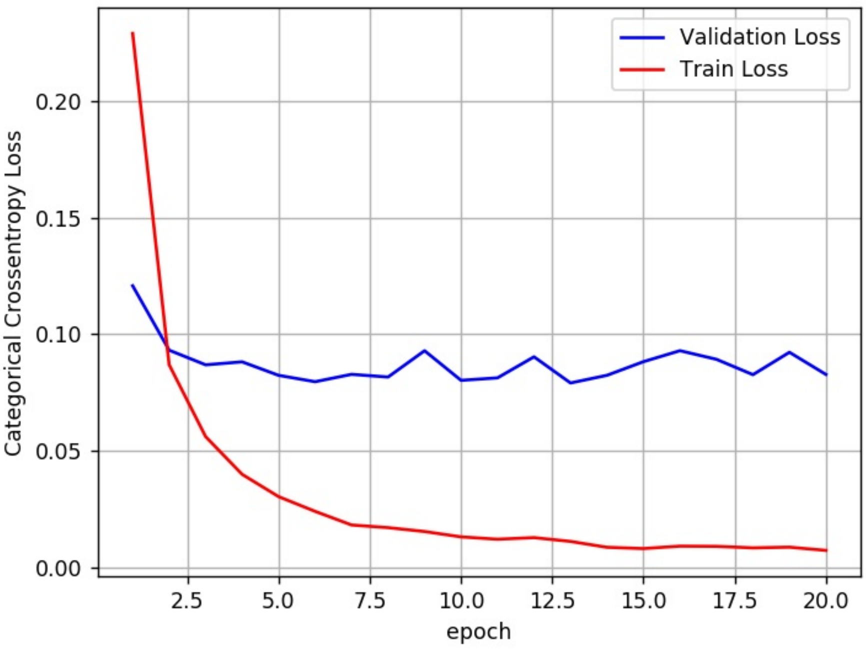
```
model_relu = Sequential()
model_relu.add(Dense(360, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0,
stddev=0.062, seed=None)))
model_relu.add(Dense(50, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_
test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_24 (Dense)             (None, 360)               282600
_________________________________________________________________
dense_25 (Dense)             (None, 50)                18050
_________________________________________________________________
dense_26 (Dense)             (None, 10)                510
=================================================================
Total params: 301,160
Trainable params: 301,160
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 19s 313us/step - loss: 0.2617 - acc: 0.9223 - val_los
s: 0.1235 - val_acc: 0.9629
Epoch 2/20
60000/60000 [==============================] - 14s 238us/step - loss: 0.1006 - acc: 0.9704 - val_los
s: 0.0853 - val_acc: 0.9745
Epoch 3/20
60000/60000 [==============================] - 13s 221us/step - loss: 0.0656 - acc: 0.9794 - val_los
s: 0.0876 - val_acc: 0.9719
Epoch 4/20
60000/60000 [==============================] - 14s 238us/step - loss: 0.0468 - acc: 0.9857 - val_los
s: 0.0814 - val_acc: 0.9740
Epoch 5/20
60000/60000 [==============================] - 16s 272us/step - loss: 0.0335 - acc: 0.9894 - val_los
s: 0.0796 - val_acc: 0.9759
Epoch 6/20
60000/60000 [==============================] - 14s 230us/step - loss: 0.0254 - acc: 0.9921 - val_los
s: 0.0655 - val_acc: 0.9793
Epoch 7/20
60000/60000 [==============================] - 14s 235us/step - loss: 0.0186 - acc: 0.9943 - val_los
s: 0.0743 - val_acc: 0.9788
Epoch 8/20
60000/60000 [==============================] - 15s 244us/step - loss: 0.0148 - acc: 0.9954 - val_los
s: 0.0748 - val_acc: 0.9790
Epoch 9/20
60000/60000 [==============================] - 14s 229us/step - loss: 0.0134 - acc: 0.9963 - val_los
s: 0.0714 - val_acc: 0.9801
Epoch 10/20
60000/60000 [==============================] - 15s 242us/step - loss: 0.0112 - acc: 0.9967 - val_los
s: 0.0752 - val_acc: 0.9808
Epoch 11/20
60000/60000 [==============================] - 14s 229us/step - loss: 0.0097 - acc: 0.9969 - val_los
s: 0.0807 - val_acc: 0.9802
Epoch 12/20
60000/60000 [==============================] - 13s 216us/step - loss: 0.0135 - acc: 0.9957 - val_los
s: 0.0850 - val_acc: 0.9779
Epoch 13/20
60000/60000 [==============================] - 11s 185us/step - loss: 0.0119 - acc: 0.9960 - val_los
s: 0.0911 - val_acc: 0.9782
Epoch 14/20
60000/60000 [==============================] - 12s 206us/step - loss: 0.0055 - acc: 0.9984 - val_los
s: 0.0869 - val_acc: 0.9790
Epoch 15/20
60000/60000 [==============================] - 12s 199us/step - loss: 0.0088 - acc: 0.9973 - val_los
s: 0.0905 - val_acc: 0.9799
Epoch 16/20
60000/60000 [==============================] - 11s 188us/step - loss: 0.0078 - acc: 0.9975 - val_los
s: 0.0897 - val_acc: 0.9790
Epoch 17/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.0087 - acc: 0.9972 - val_los
s: 0.1014 - val_acc: 0.9779
Epoch 18/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.0075 - acc: 0.9974 - val_loss
: 0.0899 - val_acc: 0.9794
Epoch 19/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.0041 - acc: 0.9987 - val_loss
: 0.0847 - val_acc: 0.9821
Epoch 20/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0056 - acc: 0.9983 - val_loss
: 0.1050 - val_acc: 0.9766
```

In [49]:

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

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

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

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

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

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


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

```
Test score: 0.10503946823413621
Test accuracy: 0.9766
```



MLP + Batch-Norm on hidden Layers + AdamOptimizer (784-360-50-10)

In [50]:

```
# Multilayer perceptron

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

from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_batch.add(Dense(360, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0,
stddev=0.039, seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(50, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )
model_batch.add(BatchNormalization())

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


model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_30 (Dense)             (None, 360)               282600    
_________________________________________________________________
batch_normalization_7 (Batch (None, 360)               1440      
_________________________________________________________________
dense_31 (Dense)             (None, 50)                18050     
_________________________________________________________________
batch_normalization_8 (Batch (None, 50)                200       
_________________________________________________________________
dense_32 (Dense)             (None, 10)                510       
=================================================================
Total params: 302,800
Trainable params: 301,980
Non-trainable params: 820
_________________________________________________________________
```

In [53]:

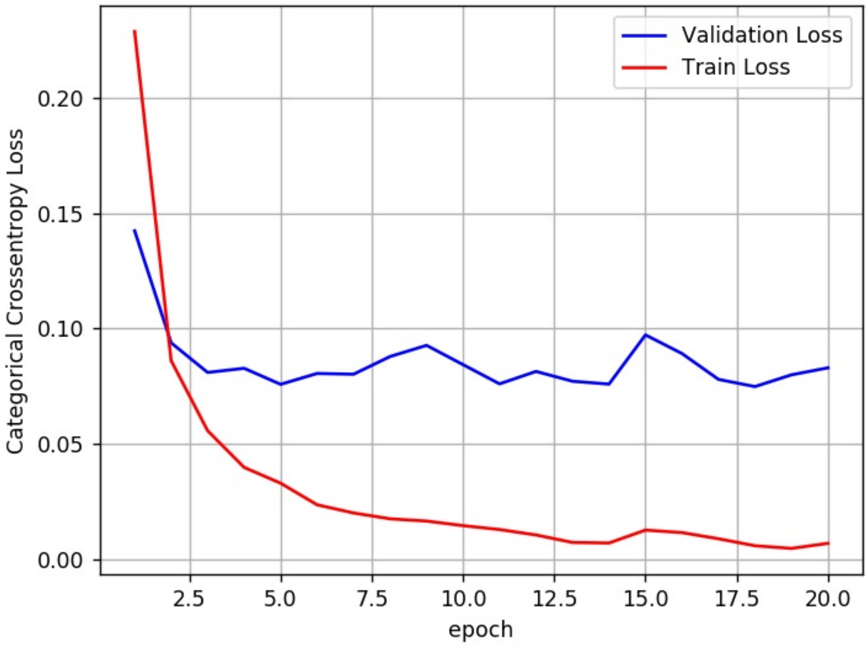
```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X
_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 25s 412us/step - loss: 0.2286 - acc: 0.9360 - val_los
s: 0.1423 - val_acc: 0.9547
Epoch 2/20
60000/60000 [==============================] - 20s 332us/step - loss: 0.0861 - acc: 0.9754 - val_los
s: 0.0938 - val_acc: 0.9708
Epoch 3/20
60000/60000 [==============================] - 19s 320us/step - loss: 0.0558 - acc: 0.9837 - val_los
s: 0.0810 - val_acc: 0.9758
Epoch 4/20
60000/60000 [==============================] - 19s 322us/step - loss: 0.0399 - acc: 0.9884 - val_los
s: 0.0828 - val_acc: 0.9750
Epoch 5/20
60000/60000 [==============================] - 23s 380us/step - loss: 0.0330 - acc: 0.9899 - val_los
s: 0.0759 - val_acc: 0.9768
Epoch 6/20
60000/60000 [==============================] - 18s 306us/step - loss: 0.0237 - acc: 0.9928 - val_los
s: 0.0805 - val_acc: 0.9751
Epoch 7/20
60000/60000 [==============================] - 17s 281us/step - loss: 0.0201 - acc: 0.9937 - val_los
s: 0.0802 - val_acc: 0.9761
Epoch 8/20
60000/60000 [==============================] - 17s 291us/step - loss: 0.0176 - acc: 0.9947 - val_los
s: 0.0879 - val_acc: 0.9743
Epoch 9/20
60000/60000 [==============================] - 17s 287us/step - loss: 0.0166 - acc: 0.9947 - val_los
s: 0.0927 - val_acc: 0.9744
Epoch 10/20
60000/60000 [==============================] - 18s 308us/step - loss: 0.0147 - acc: 0.9952 - val_los
s: 0.0844 - val_acc: 0.9768
Epoch 11/20
60000/60000 [==============================] - 21s 346us/step - loss: 0.0130 - acc: 0.9958 - val_los
s: 0.0761 - val_acc: 0.9788
Epoch 12/20
60000/60000 [==============================] - 22s 362us/step - loss: 0.0106 - acc: 0.9967 - val_los
s: 0.0814 - val_acc: 0.9783
Epoch 13/20
60000/60000 [==============================] - 20s 327us/step - loss: 0.0074 - acc: 0.9980 - val_los
s: 0.0772 - val_acc: 0.9790
Epoch 14/20
60000/60000 [==============================] - 23s 377us/step - loss: 0.0071 - acc: 0.9979 - val_los
s: 0.0759 - val_acc: 0.9795
Epoch 15/20
60000/60000 [==============================] - 18s 304us/step - loss: 0.0127 - acc: 0.9959 - val_los
s: 0.0972 - val_acc: 0.9761
Epoch 16/20
60000/60000 [==============================] - 19s 313us/step - loss: 0.0116 - acc: 0.9962 - val_los
s: 0.0892 - val_acc: 0.9782
Epoch 17/20
60000/60000 [==============================] - 20s 339us/step - loss: 0.0090 - acc: 0.9974 - val_los
s: 0.0780 - val_acc: 0.9799
Epoch 18/20
60000/60000 [==============================] - 24s 405us/step - loss: 0.0060 - acc: 0.9983 - val_los
s: 0.0749 - val_acc: 0.9815
Epoch 19/20
60000/60000 [==============================] - 2747s 46ms/step - loss: 0.0048 - acc: 0.9987 - val_lo
ss: 0.0800 - val_acc: 0.9810
Epoch 20/20
60000/60000 [==============================] - 14s 238us/step - loss: 0.0070 - acc: 0.9976 - val_los
s: 0.0830 - val_acc: 0.9794
```

In [54]:

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

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

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

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

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

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

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

```
Test score: 0.08298968671116527
Test accuracy: 0.9794
```



MLP + Dropout + AdamOptimizer (784-360-50-10)

In [55]:

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

from keras.layers import Dropout

model_drop = Sequential()

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

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

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


model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_33 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_9 (Batch (None, 512)               2048      
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_34 (Dense)             (None, 128)               65664     
_________________________________________________________________
batch_normalization_10 (Batc (None, 128)               512       
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_35 (Dense)             (None, 10)                1290      
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

In [56]:

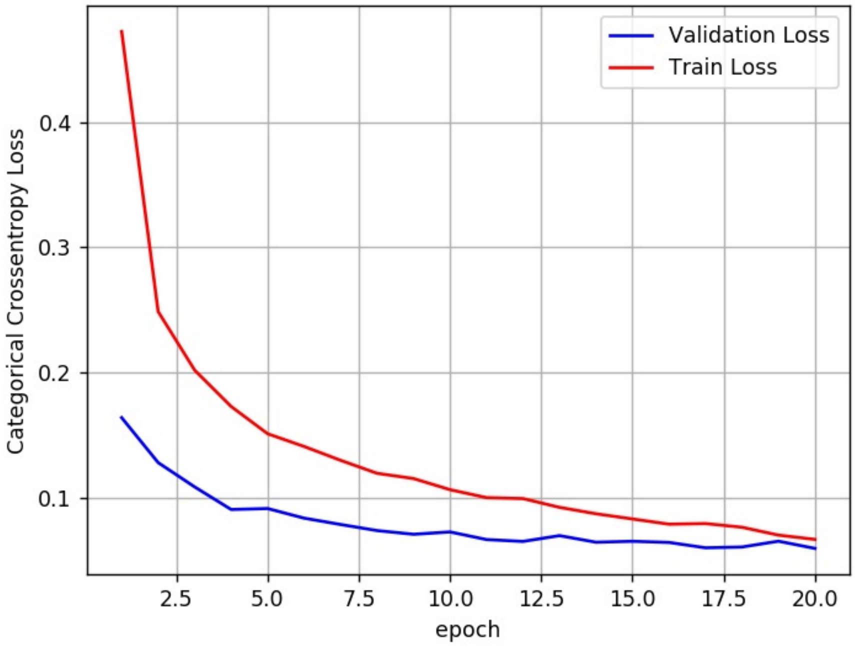
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_
test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 26s 436us/step - loss: 0.4722 - acc: 0.8563 - val_los
s: 0.1639 - val_acc: 0.9489
Epoch 2/20
60000/60000 [==============================] - 18s 308us/step - loss: 0.2486 - acc: 0.9257 - val_los
s: 0.1280 - val_acc: 0.9616
Epoch 3/20
60000/60000 [==============================] - 17s 278us/step - loss: 0.2019 - acc: 0.9393 - val_los
s: 0.1086 - val_acc: 0.9663
Epoch 4/20
60000/60000 [==============================] - 18s 304us/step - loss: 0.1728 - acc: 0.9478 - val_los
s: 0.0906 - val_acc: 0.9709
Epoch 5/20
60000/60000 [==============================] - 19s 312us/step - loss: 0.1511 - acc: 0.9547 - val_los
s: 0.0914 - val_acc: 0.9710
Epoch 6/20
60000/60000 [==============================] - 21s 352us/step - loss: 0.1409 - acc: 0.9580 - val_los
s: 0.0837 - val_acc: 0.9741
Epoch 7/20
60000/60000 [==============================] - 18s 300us/step - loss: 0.1299 - acc: 0.9597 - val_los
s: 0.0786 - val_acc: 0.9769
Epoch 8/20
60000/60000 [==============================] - 19s 317us/step - loss: 0.1195 - acc: 0.9630 - val_los
s: 0.0738 - val_acc: 0.9773
Epoch 9/20
60000/60000 [==============================] - 19s 312us/step - loss: 0.1153 - acc: 0.9655 - val_los
s: 0.0708 - val_acc: 0.9772
Epoch 10/20
60000/60000 [==============================] - 18s 304us/step - loss: 0.1064 - acc: 0.9671 - val_los
s: 0.0727 - val_acc: 0.9778
Epoch 11/20
60000/60000 [==============================] - 18s 306us/step - loss: 0.1001 - acc: 0.9696 - val_los
s: 0.0666 - val_acc: 0.9802
Epoch 12/20
60000/60000 [==============================] - 17s 290us/step - loss: 0.0993 - acc: 0.9691 - val_los
s: 0.0651 - val_acc: 0.9793
Epoch 13/20
60000/60000 [==============================] - 18s 304us/step - loss: 0.0923 - acc: 0.9714 - val_los
s: 0.0697 - val_acc: 0.9788
Epoch 14/20
60000/60000 [==============================] - 18s 292us/step - loss: 0.0872 - acc: 0.9729 - val_los
s: 0.0645 - val_acc: 0.9804
Epoch 15/20
60000/60000 [==============================] - 21s 349us/step - loss: 0.0830 - acc: 0.9739 - val_los
s: 0.0652 - val_acc: 0.9804
Epoch 16/20
60000/60000 [==============================] - 19s 324us/step - loss: 0.0789 - acc: 0.9754 - val_los
s: 0.0643 - val_acc: 0.9806
Epoch 17/20
60000/60000 [==============================] - 18s 298us/step - loss: 0.0794 - acc: 0.9751 - val_los
s: 0.0600 - val_acc: 0.9821
Epoch 18/20
60000/60000 [==============================] - 18s 301us/step - loss: 0.0764 - acc: 0.9754 - val_los
s: 0.0606 - val_acc: 0.9823
Epoch 19/20
60000/60000 [==============================] - 20s 340us/step - loss: 0.0701 - acc: 0.9785 - val_los
s: 0.0653 - val_acc: 0.9821
Epoch 20/20
60000/60000 [==============================] - 19s 314us/step - loss: 0.0667 - acc: 0.9789 - val_los
s: 0.0595 - val_acc: 0.9834
```

In [57]:

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

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

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

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

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

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

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

```
Test score: 0.059524212178983724
Test accuracy: 0.9834
```



Task 2: 3 Hidden layers (784-405-250-115-10)

MLP + ReLU + ADAM

In [72]:

```
model_relu = Sequential()
model_relu.add(Dense(405, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(Dense(250, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(115, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.15, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_83 (Dense)             (None, 405)               317925    
_________________________________________________________________
dense_84 (Dense)             (None, 250)               101500    
_________________________________________________________________
dense_85 (Dense)             (None, 115)               28865     
_________________________________________________________________
dense_86 (Dense)             (None, 10)                1160      
=================================================================
Total params: 449,450
Trainable params: 449,450
Non-trainable params: 0
_________________________________________________________________
None
```

In [73]:

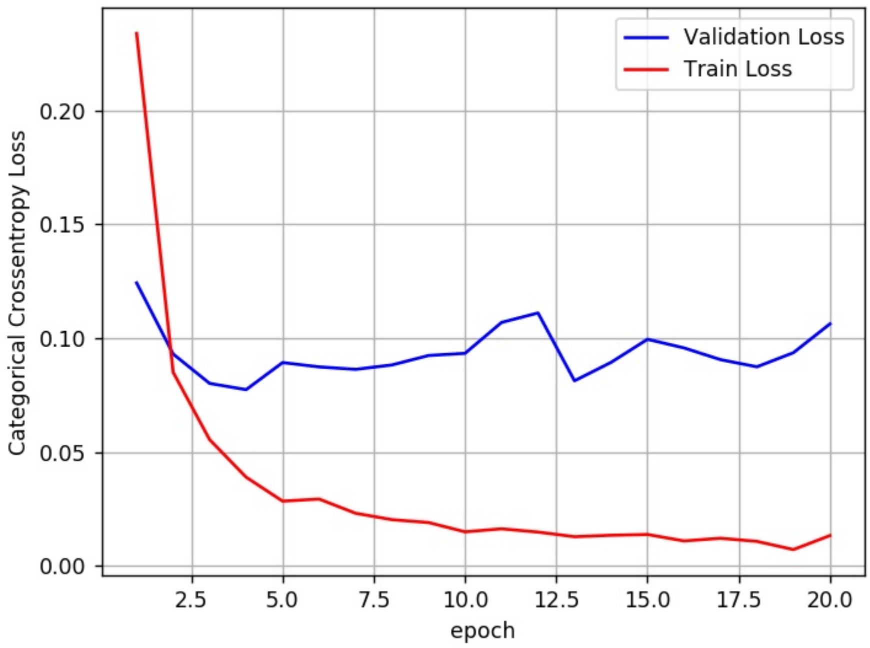
```
model_relu.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 42s 698us/step - loss: 0.2338 - acc: 0.9281 - val_los
s: 0.1243 - val_acc: 0.9616
Epoch 2/20
60000/60000 [==============================] - 24s 402us/step - loss: 0.0851 - acc: 0.9734 - val_los
s: 0.0931 - val_acc: 0.9701
Epoch 3/20
60000/60000 [==============================] - 21s 345us/step - loss: 0.0555 - acc: 0.9821 - val_los
s: 0.0802 - val_acc: 0.9744
Epoch 4/20
60000/60000 [==============================] - 18s 294us/step - loss: 0.0390 - acc: 0.9874 - val_los
s: 0.0774 - val_acc: 0.9773
Epoch 5/20
60000/60000 [==============================] - 16s 275us/step - loss: 0.0285 - acc: 0.9904 - val_los
s: 0.0893 - val_acc: 0.9741
Epoch 6/20
60000/60000 [==============================] - 14s 238us/step - loss: 0.0294 - acc: 0.9903 - val_los
s: 0.0874 - val_acc: 0.9751
Epoch 7/20
60000/60000 [==============================] - 22s 362us/step - loss: 0.0232 - acc: 0.9923 - val_los
s: 0.0863 - val_acc: 0.9761
Epoch 8/20
60000/60000 [==============================] - 25s 420us/step - loss: 0.0203 - acc: 0.9932 - val_los
s: 0.0883 - val_acc: 0.9774
Epoch 9/20
60000/60000 [==============================] - 22s 372us/step - loss: 0.0191 - acc: 0.9940 - val_los
s: 0.0924 - val_acc: 0.9764
Epoch 10/20
60000/60000 [==============================] - 19s 314us/step - loss: 0.0150 - acc: 0.9949 - val_los
s: 0.0934 - val_acc: 0.9773
Epoch 11/20
60000/60000 [==============================] - 20s 332us/step - loss: 0.0163 - acc: 0.9944 - val_los
s: 0.1069 - val_acc: 0.9766
Epoch 12/20
60000/60000 [==============================] - 19s 320us/step - loss: 0.0149 - acc: 0.9951 - val_los
s: 0.1111 - val_acc: 0.9762
Epoch 13/20
60000/60000 [==============================] - 22s 364us/step - loss: 0.0129 - acc: 0.9959 - val_los
s: 0.0813 - val_acc: 0.9821
Epoch 14/20
60000/60000 [==============================] - 17s 289us/step - loss: 0.0135 - acc: 0.9955 - val_los
s: 0.0894 - val_acc: 0.9811
Epoch 15/20
60000/60000 [==============================] - 15s 247us/step - loss: 0.0138 - acc: 0.9959 - val_los
s: 0.0995 - val_acc: 0.9783
Epoch 16/20
60000/60000 [==============================] - 15s 256us/step - loss: 0.0110 - acc: 0.9964 - val_los
s: 0.0958 - val_acc: 0.9796
Epoch 17/20
60000/60000 [==============================] - 14s 227us/step - loss: 0.0122 - acc: 0.9960 - val_los
s: 0.0906 - val_acc: 0.9807
Epoch 18/20
60000/60000 [==============================] - 22s 362us/step - loss: 0.0108 - acc: 0.9966 - val_los
s: 0.0874 - val_acc: 0.9821
Epoch 19/20
60000/60000 [==============================] - 25s 418us/step - loss: 0.0072 - acc: 0.9976 - val_los
s: 0.0937 - val_acc: 0.9830
Epoch 20/20
60000/60000 [==============================] - 18s 296us/step - loss: 0.0134 - acc: 0.9960 - val_los
s: 0.1063 - val_acc: 0.9787
```

In [75]:

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

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

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

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

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

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


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

```
Test score: 0.10629093127994114
Test accuracy: 0.9787
```



MLP + Batch-Norm on hidden Layers + AdamOptimizer (784-405-250-115-10) </2>

In [76]:

```
from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_batch.add(Dense(405, activation='relu', input_shape=(input_dim,),
                      kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(250, activation='relu',
                      kernel_initializer=RandomNormal(mean=0.0, stddev=0.15, seed=None)) )
model_batch.add(BatchNormalization())


model_batch.add(Dense(115, activation='relu',
                      kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )
model_batch.add(BatchNormalization())

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


model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_91 (Dense)             (None, 405)               317925
_________________________________________________________________
batch_normalization_24 (Batc (None, 405)               1620
_________________________________________________________________
dense_92 (Dense)             (None, 250)               101500
_________________________________________________________________
batch_normalization_25 (Batc (None, 250)               1000
_________________________________________________________________
dense_93 (Dense)             (None, 115)               28865
_________________________________________________________________
batch_normalization_26 (Batc (None, 115)               460
_________________________________________________________________
dense_94 (Dense)             (None, 10)                1160
=================================================================
Total params: 452,530
Trainable params: 450,990
Non-trainable params: 1,540
_________________________________________________________________
```

In [77]:

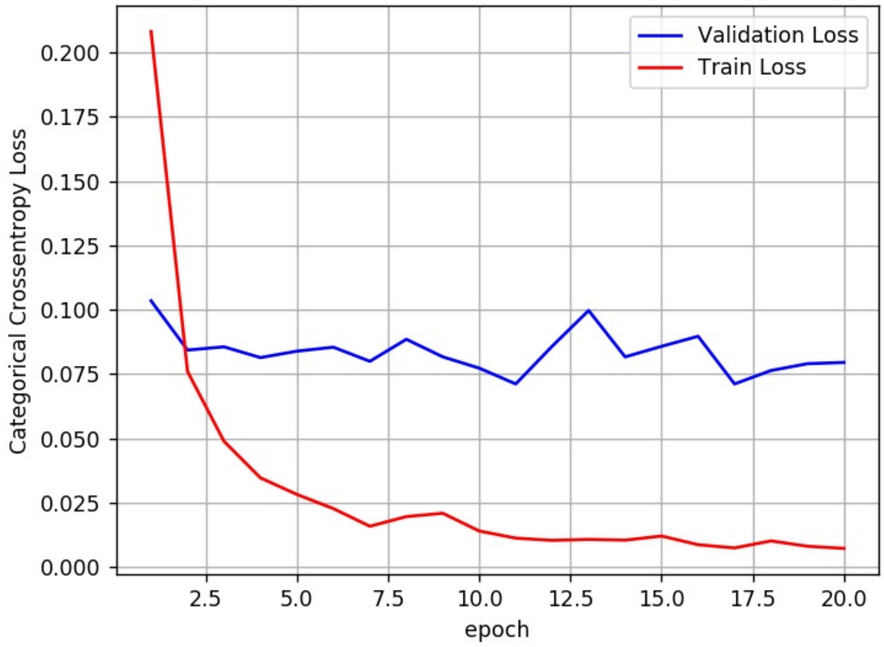
```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X
_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 39s 650us/step - loss: 0.2081 - acc: 0.9384 - val_los
s: 0.1035 - val_acc: 0.9664
Epoch 2/20
60000/60000 [==============================] - 19s 321us/step - loss: 0.0760 - acc: 0.9770 - val_los
s: 0.0844 - val_acc: 0.9730
Epoch 3/20
60000/60000 [==============================] - 19s 319us/step - loss: 0.0489 - acc: 0.9845 - val_los
s: 0.0856 - val_acc: 0.9723
Epoch 4/20
60000/60000 [==============================] - 27s 449us/step - loss: 0.0347 - acc: 0.9890 - val_los
s: 0.0814 - val_acc: 0.9732
Epoch 5/20
60000/60000 [==============================] - 35s 581us/step - loss: 0.0283 - acc: 0.9910 - val_los
s: 0.0839 - val_acc: 0.9749
Epoch 6/20
60000/60000 [==============================] - 35s 587us/step - loss: 0.0228 - acc: 0.9927 - val_los
s: 0.0855 - val_acc: 0.9744
Epoch 7/20
60000/60000 [==============================] - 42s 702us/step - loss: 0.0159 - acc: 0.9951 - val_los
s: 0.0800 - val_acc: 0.9776
Epoch 8/20
60000/60000 [==============================] - 29s 477us/step - loss: 0.0197 - acc: 0.9932 - val_los
s: 0.0885 - val_acc: 0.9744
Epoch 9/20
60000/60000 [==============================] - 28s 465us/step - loss: 0.0210 - acc: 0.9930 - val_los
s: 0.0817 - val_acc: 0.9787
Epoch 10/20
60000/60000 [==============================] - 37s 611us/step - loss: 0.0141 - acc: 0.9954 - val_los
s: 0.0773 - val_acc: 0.9773
Epoch 11/20
60000/60000 [==============================] - 33s 544us/step - loss: 0.0113 - acc: 0.9966 - val_los
s: 0.0712 - val_acc: 0.9798
Epoch 12/20
60000/60000 [==============================] - 36s 608us/step - loss: 0.0105 - acc: 0.9968 - val_los
s: 0.0859 - val_acc: 0.9771
Epoch 13/20
60000/60000 [==============================] - 36s 598us/step - loss: 0.0108 - acc: 0.9963 - val_los
s: 0.0997 - val_acc: 0.9761
Epoch 14/20
60000/60000 [==============================] - 33s 551us/step - loss: 0.0105 - acc: 0.9963 - val_los
s: 0.0817 - val_acc: 0.9778
Epoch 15/20
60000/60000 [==============================] - 22s 359us/step - loss: 0.0121 - acc: 0.9960 - val_los
s: 0.0858 - val_acc: 0.9793
Epoch 16/20
60000/60000 [==============================] - 34s 568us/step - loss: 0.0088 - acc: 0.9971 - val_los
s: 0.0897 - val_acc: 0.9794
Epoch 17/20
60000/60000 [==============================] - 29s 484us/step - loss: 0.0075 - acc: 0.9973 - val_los
s: 0.0712 - val_acc: 0.9811
Epoch 18/20
60000/60000 [==============================] - 25s 412us/step - loss: 0.0103 - acc: 0.9965 - val_los
s: 0.0764 - val_acc: 0.9817
Epoch 19/20
60000/60000 [==============================] - 31s 511us/step - loss: 0.0082 - acc: 0.9970 - val_los
s: 0.0791 - val_acc: 0.9806
Epoch 20/20
60000/60000 [==============================] - 33s 546us/step - loss: 0.0073 - acc: 0.9974 - val_los
s: 0.0795 - val_acc: 0.9800
```

In [78]:

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

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

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

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

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

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

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

```
Test score: 0.0795468446892528
Test accuracy: 0.98
```



MLP + Dropout + AdamOptimizer (784-405-250-115-10)

In [79]:

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

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(405, activation='relu', input_shape=(input_dim,),
                     kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(250, activation='relu',
                     kernel_initializer=RandomNormal(mean=0.0, stddev=0.5, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(115, activation='relu',
                     kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))


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


model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_95 (Dense)             (None, 405)               317925    
_________________________________________________________________
batch_normalization_27 (Batc (None, 405)               1620      
_________________________________________________________________
dropout_18 (Dropout)         (None, 405)               0         
_________________________________________________________________
dense_96 (Dense)             (None, 250)               101500    
_________________________________________________________________
batch_normalization_28 (Batc (None, 250)               1000      
_________________________________________________________________
dropout_19 (Dropout)         (None, 250)               0         
_________________________________________________________________
dense_97 (Dense)             (None, 115)               28865     
_________________________________________________________________
batch_normalization_29 (Batc (None, 115)               460       
_________________________________________________________________
dropout_20 (Dropout)         (None, 115)               0         
_________________________________________________________________
dense_98 (Dense)             (None, 10)                1160      
=================================================================
Total params: 452,530
Trainable params: 450,990
Non-trainable params: 1,540
_________________________________________________________________
```

In [80]:

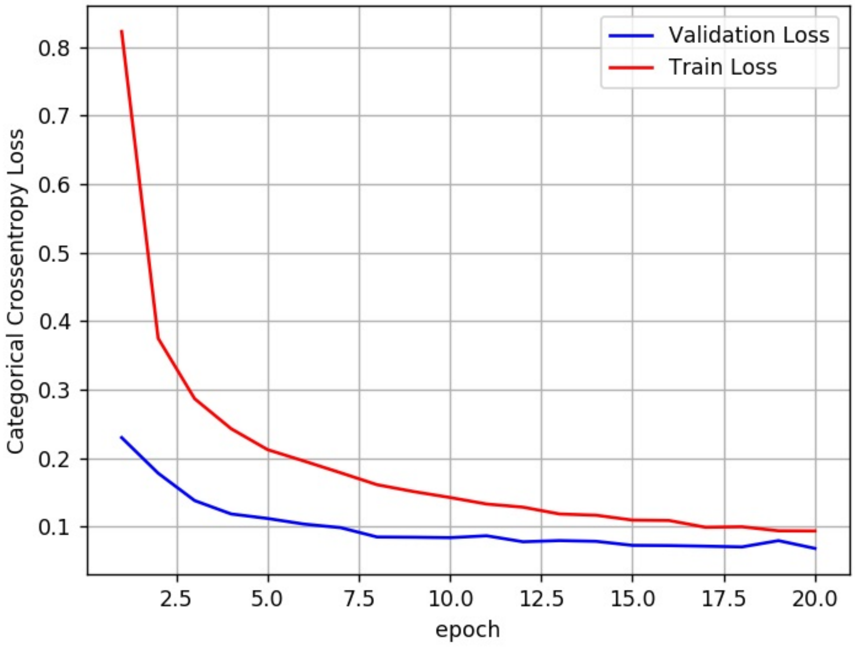
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_
test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 46s 762us/step - loss: 0.8226 - acc: 0.7439 - val_los
s: 0.2299 - val_acc: 0.9308
Epoch 2/20
60000/60000 [==============================] - 26s 429us/step - loss: 0.3748 - acc: 0.8882 - val_los
s: 0.1779 - val_acc: 0.9428
Epoch 3/20
60000/60000 [==============================] - 26s 428us/step - loss: 0.2866 - acc: 0.9163 - val_los
s: 0.1379 - val_acc: 0.9570
Epoch 4/20
60000/60000 [==============================] - 25s 419us/step - loss: 0.2428 - acc: 0.9288 - val_los
s: 0.1184 - val_acc: 0.9636
Epoch 5/20
60000/60000 [==============================] - 25s 418us/step - loss: 0.2122 - acc: 0.9377 - val_los
s: 0.1118 - val_acc: 0.9653
Epoch 6/20
60000/60000 [==============================] - 26s 441us/step - loss: 0.1956 - acc: 0.9428 - val_los
s: 0.1036 - val_acc: 0.9695
Epoch 7/20
60000/60000 [==============================] - 25s 418us/step - loss: 0.1785 - acc: 0.9476 - val_los
s: 0.0984 - val_acc: 0.9708
Epoch 8/20
60000/60000 [==============================] - 25s 417us/step - loss: 0.1611 - acc: 0.9532 - val_los
s: 0.0848 - val_acc: 0.9731
Epoch 9/20
60000/60000 [==============================] - 21s 358us/step - loss: 0.1511 - acc: 0.9556 - val_los
s: 0.0844 - val_acc: 0.9751
Epoch 10/20
60000/60000 [==============================] - 23s 379us/step - loss: 0.1424 - acc: 0.9579 - val_los
s: 0.0838 - val_acc: 0.9755
Epoch 11/20
60000/60000 [==============================] - 30s 498us/step - loss: 0.1329 - acc: 0.9605 - val_los
s: 0.0866 - val_acc: 0.9754
Epoch 12/20
60000/60000 [==============================] - 26s 428us/step - loss: 0.1283 - acc: 0.9628 - val_los
s: 0.0778 - val_acc: 0.9774
Epoch 13/20
60000/60000 [==============================] - 28s 461us/step - loss: 0.1184 - acc: 0.9648 - val_los
s: 0.0795 - val_acc: 0.9761
Epoch 14/20
60000/60000 [==============================] - 28s 461us/step - loss: 0.1165 - acc: 0.9657 - val_los
s: 0.0785 - val_acc: 0.9776
Epoch 15/20
60000/60000 [==============================] - 27s 451us/step - loss: 0.1094 - acc: 0.9675 - val_los
s: 0.0726 - val_acc: 0.9789
Epoch 16/20
60000/60000 [==============================] - 29s 490us/step - loss: 0.1089 - acc: 0.9678 - val_los
s: 0.0722 - val_acc: 0.9798
Epoch 17/20
60000/60000 [==============================] - 29s 485us/step - loss: 0.0990 - acc: 0.9702 - val_los
s: 0.0713 - val_acc: 0.9803
Epoch 18/20
60000/60000 [==============================] - 30s 504us/step - loss: 0.0997 - acc: 0.9703 - val_los
s: 0.0703 - val_acc: 0.9808
Epoch 19/20
60000/60000 [==============================] - 30s 503us/step - loss: 0.0938 - acc: 0.9723 - val_los
s: 0.0795 - val_acc: 0.9789
Epoch 20/20
60000/60000 [==============================] - 31s 523us/step - loss: 0.0936 - acc: 0.9723 - val_los
s: 0.0681 - val_acc: 0.9809
```

In [81]:

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

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

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

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

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

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

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

```
Test score: 0.06811148248039535
Test accuracy: 0.9809
```



Task 3: 5 Hidden Layers(784-405-350-215-110-55-10)

MLP + ReLU + ADAM

In [82]:

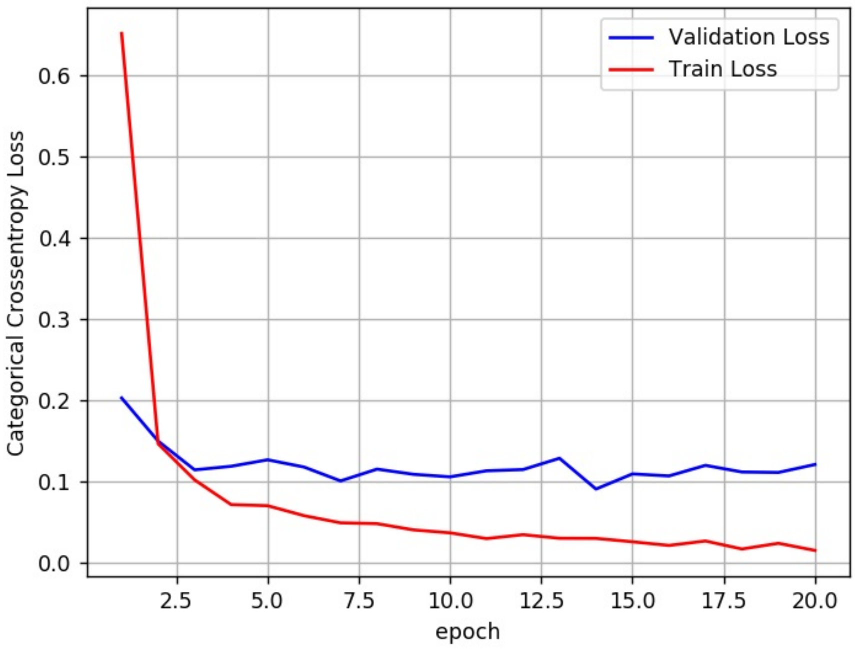
```
model_relu = Sequential()
model_relu.add(Dense(405, activation='relu', input_shape=(input_dim,),
                     kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(Dense(350, activation='relu',
                     kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )

model_relu.add(Dense(215, activation='relu',
                     kernel_initializer=RandomNormal(mean=0.0, stddev=0.65, seed=None)) )

model_relu.add(Dense(110, activation='relu',
                     kernel_initializer=RandomNormal(mean=0.0, stddev=0.15, seed=None)) )

model_relu.add(Dense(55, activation='relu',
                     kernel_initializer=RandomNormal(mean=0.0, stddev=0.25, seed=None)) )

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

print(model_relu.summary())

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

history = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X
_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_99 (Dense)             (None, 405)               317925
_________________________________________________________________
dense_100 (Dense)            (None, 350)               142100
_________________________________________________________________
dense_101 (Dense)            (None, 215)               75465
_________________________________________________________________
dense_102 (Dense)            (None, 110)               23760
_________________________________________________________________
dense_103 (Dense)            (None, 55)                6105
_________________________________________________________________
dense_104 (Dense)            (None, 10)                560
=================================================================
Total params: 565,915
Trainable params: 565,915
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 38s 628us/step - loss: 0.6514 - acc: 0.8835 - val_los
s: 0.2031 - val_acc: 0.9400
Epoch 2/20
60000/60000 [==============================] - 28s 460us/step - loss: 0.1467 - acc: 0.9575 - val_los
s: 0.1501 - val_acc: 0.9571
Epoch 3/20
60000/60000 [==============================] - 27s 448us/step - loss: 0.1024 - acc: 0.9696 - val_los
s: 0.1147 - val_acc: 0.9657
Epoch 4/20
60000/60000 [==============================] - 26s 433us/step - loss: 0.0720 - acc: 0.9778 - val_los
s: 0.1191 - val_acc: 0.9674
Epoch 5/20
60000/60000 [==============================] - 27s 446us/step - loss: 0.0706 - acc: 0.9789 - val_los
s: 0.1270 - val_acc: 0.9649
Epoch 6/20
60000/60000 [==============================] - 27s 448us/step - loss: 0.0583 - acc: 0.9816 - val_los
s: 0.1181 - val_acc: 0.9692
Epoch 7/20
60000/60000 [==============================] - 28s 460us/step - loss: 0.0495 - acc: 0.9846 - val_los
s: 0.1011 - val_acc: 0.9732
Epoch 8/20
60000/60000 [==============================] - 25s 410us/step - loss: 0.0485 - acc: 0.9849 - val_los
s: 0.1156 - val_acc: 0.9716
Epoch 9/20
60000/60000 [==============================] - 25s 419us/step - loss: 0.0408 - acc: 0.9869 - val_los
s: 0.1092 - val_acc: 0.9708
Epoch 10/20
60000/60000 [==============================] - 28s 461us/step - loss: 0.0372 - acc: 0.9883 - val_los
s: 0.1061 - val_acc: 0.9743
Epoch 11/20
60000/60000 [==============================] - 32s 525us/step - loss: 0.0302 - acc: 0.9910 - val_los
s: 0.1135 - val_acc: 0.9727
Epoch 12/20
60000/60000 [==============================] - 29s 480us/step - loss: 0.0350 - acc: 0.9895 - val_los
s: 0.1151 - val_acc: 0.9742
Epoch 13/20
60000/60000 [==============================] - 26s 439us/step - loss: 0.0306 - acc: 0.9904 - val_los
s: 0.1289 - val_acc: 0.9707
Epoch 14/20
60000/60000 [==============================] - 26s 434us/step - loss: 0.0304 - acc: 0.9909 - val_los
s: 0.0911 - val_acc: 0.9779
Epoch 15/20
60000/60000 [==============================] - 27s 457us/step - loss: 0.0262 - acc: 0.9919 - val_los
s: 0.1097 - val_acc: 0.9766
Epoch 16/20
60000/60000 [==============================] - 27s 445us/step - loss: 0.0218 - acc: 0.9932 - val_los
s: 0.1072 - val_acc: 0.9770
Epoch 17/20
60000/60000 [==============================] - 23s 386us/step - loss: 0.0272 - acc: 0.9920 - val_los
s: 0.1202 - val_acc: 0.975270
Epoch 18/20
60000/60000 [==============================] - 22s 359us/step - loss: 0.0174 - acc: 0.9944 - val_los
s: 0.1121 - val_acc: 0.9769
Epoch 19/20
60000/60000 [==============================] - 24s 392us/step - loss: 0.0244 - acc: 0.9924 - val_los
s: 0.1115 - val_acc: 0.9754
Epoch 20/20
60000/60000 [==============================] - 25s 412us/step - loss: 0.0156 - acc: 0.9951 - val_los
s: 0.1212 - val_acc: 0.9767
```

In [83]:

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

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

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

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

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

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


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

```
Test score: 0.12124889276220338
Test accuracy: 0.9767
```



MLP + Batch-Norm on hidden Layers + AdamOptimizer (784-405-350-215-110-55-10)

In [84]:

```
from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_batch.add(Dense(405, activation='relu', input_shape=(input_dim,),
                      kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(350, activation='relu',
                      kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_batch.add(BatchNormalization())

model_batch.add(Dense(215, activation='relu',
                      kernel_initializer=RandomNormal(mean=0.0, stddev=0.25, seed=None)) )

model_batch.add(Dense(110, activation='relu',
                      kernel_initializer=RandomNormal(mean=0.0, stddev=0.15, seed=None)) )
model_batch.add(BatchNormalization())

model_batch.add(Dense(55, activation='relu',
                      kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )
model_batch.add(BatchNormalization())

model_batch.add(BatchNormalization())

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


model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_105 (Dense)            (None, 405)               317925
_________________________________________________________________
batch_normalization_30 (Batc (None, 405)               1620
_________________________________________________________________
dense_106 (Dense)            (None, 350)               142100
_________________________________________________________________
batch_normalization_31 (Batc (None, 350)               1400
_________________________________________________________________
dense_107 (Dense)            (None, 215)               75465
_________________________________________________________________
dense_108 (Dense)            (None, 110)               23760
_________________________________________________________________
batch_normalization_32 (Batc (None, 110)               440
_________________________________________________________________
dense_109 (Dense)            (None, 55)                6105
_________________________________________________________________
batch_normalization_33 (Batc (None, 55)                220
_________________________________________________________________
batch_normalization_34 (Batc (None, 55)                220
_________________________________________________________________
dense_110 (Dense)            (None, 10)                560
=================================================================
Total params: 569,815
Trainable params: 567,865
Non-trainable params: 1,950
_________________________________________________________________
```

In [85]:

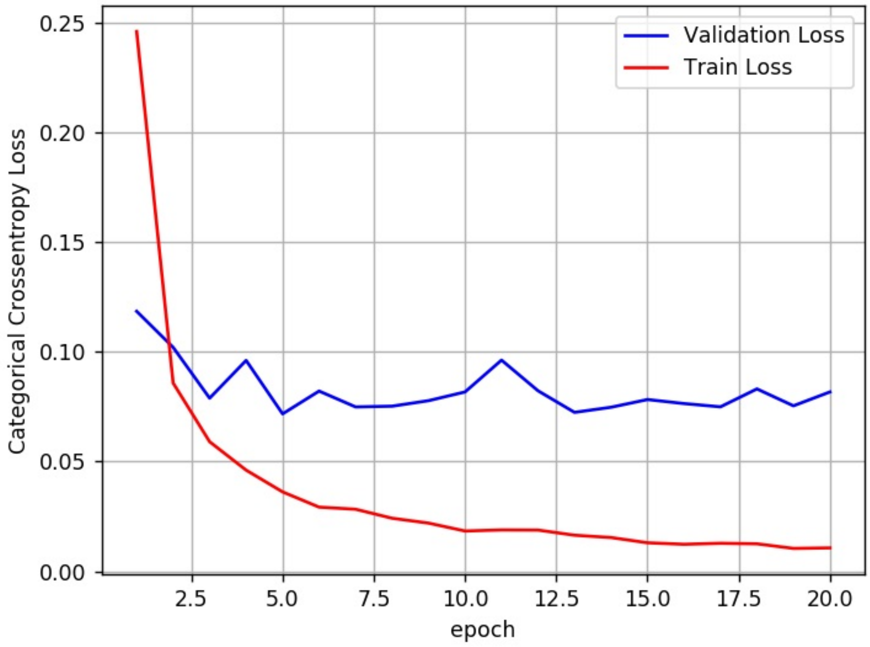
```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X
_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 65s 1ms/step - loss: 0.2459 - acc: 0.9294 - val_loss:
0.1185 - val_acc: 0.9640
Epoch 2/20
60000/60000 [==============================] - 43s 711us/step - loss: 0.0858 - acc: 0.9739 - val_los
s: 0.1020 - val_acc: 0.9695
Epoch 3/20
60000/60000 [==============================] - 40s 673us/step - loss: 0.0590 - acc: 0.9815 - val_los
s: 0.0789 - val_acc: 0.9779
Epoch 4/20
60000/60000 [==============================] - 35s 587us/step - loss: 0.0461 - acc: 0.9853 - val_los
s: 0.0961 - val_acc: 0.9692
Epoch 5/20
60000/60000 [==============================] - 34s 563us/step - loss: 0.0362 - acc: 0.9884 - val_los
s: 0.0717 - val_acc: 0.9786
Epoch 6/20
60000/60000 [==============================] - 40s 660us/step - loss: 0.0292 - acc: 0.9905 - val_los
s: 0.0821 - val_acc: 0.9771
Epoch 7/20
60000/60000 [==============================] - 39s 647us/step - loss: 0.0283 - acc: 0.9909 - val_los
s: 0.0749 - val_acc: 0.9784
Epoch 8/20
60000/60000 [==============================] - 48s 794us/step - loss: 0.0242 - acc: 0.9916 - val_los
s: 0.0752 - val_acc: 0.9785
Epoch 9/20
60000/60000 [==============================] - 44s 741us/step - loss: 0.0220 - acc: 0.9924 - val_los
s: 0.0777 - val_acc: 0.9786
Epoch 10/20
60000/60000 [==============================] - 38s 627us/step - loss: 0.0184 - acc: 0.9938 - val_los
s: 0.0817 - val_acc: 0.9793
Epoch 11/20
60000/60000 [==============================] - 44s 726us/step - loss: 0.0189 - acc: 0.9938 - val_los
s: 0.0962 - val_acc: 0.9744- loss: 0.0188 - acc: 0.9
Epoch 12/20
60000/60000 [==============================] - 41s 677us/step - loss: 0.0188 - acc: 0.9937 - val_los
s: 0.0822 - val_acc: 0.9794
Epoch 13/20
60000/60000 [==============================] - 33s 545us/step - loss: 0.0164 - acc: 0.9943 - val_los
s: 0.0724 - val_acc: 0.9805 14s - loss: 0.0152  - ETA: 9s - loss: 0.0 - ETA: 5s  - ETA: 3s - loss: 0
.0
Epoch 14/20
60000/60000 [==============================] - 32s 531us/step - loss: 0.0154 - acc: 0.9949 - val_los
s: 0.0747 - val_acc: 0.9799 26s -  - ETA: 26s - loss: 0.0134 - ETA: 25s - loss: 0.0128 - ETA: 25s  -
ETA: 19s - loss: 0. - ETA: 18s - loss: 0.01 - ETA: 18s - loss: 0.0153 - acc: 0.99 - ETA:  - ETA: 12s
- loss: 0.0152 - a - ETA: 12s - loss: 0.01 - ETA: 11s - loss: 0.0153 - acc: 0.99 - ETA: 11s - loss:
0.0152 - ETA: 9s - loss: 0.0151 - acc - ETA: 4s - loss: 0.0147 - ac - ETA: 4s - ETA: 2s - loss: 0.01
50 - acc: 0.994 - ETA: 2s - loss: 0.0150 - acc: 0.994 - ETA: 2s - loss: 0.0151 - a - ETA: 1s - l
Epoch 15/20
60000/60000 [==============================] - 42s 702us/step - loss: 0.0130 - acc: 0.9959 - val_los
s: 0.0783 - val_acc: 0.9801
Epoch 16/20
60000/60000 [==============================] - 39s 643us/step - loss: 0.0123 - acc: 0.9958 - val_los
s: 0.0764 - val_acc: 0.9806
Epoch 17/20
60000/60000 [==============================] - 48s 804us/step - loss: 0.0128 - acc: 0.9956 - val_los
s: 0.0749 - val_acc: 0.9799
Epoch 18/20
60000/60000 [==============================] - 47s 792us/step - loss: 0.0125 - acc: 0.9958 - val_los
s: 0.0831 - val_acc: 0.9795
Epoch 19/20
60000/60000 [==============================] - 45s 757us/step - loss: 0.0104 - acc: 0.9964 - val_los
s: 0.0754 - val_acc: 0.9823
Epoch 20/20
60000/60000 [==============================] - 46s 761us/step - loss: 0.0107 - acc: 0.9966 - val_los
s: 0.0817 - val_acc: 0.9798
```

In [87]:

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

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

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

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

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

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

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

```
Test score: 0.08171314015269746
Test accuracy: 0.9798
```



MLP + Dropout + AdamOptimizer (784-405-350-215-110-55-10)

In [88]:

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

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(405, activation='relu', input_shape=(input_dim,),
                     kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(350, activation='relu',
                     kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(215, activation='relu',
                     kernel_initializer=RandomNormal(mean=0.0, stddev=0.15, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(110, activation='relu',
                     kernel_initializer=RandomNormal(mean=0.0, stddev=0.25, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(55, activation='relu',
                     kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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


model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_111 (Dense)            (None, 405)               317925    
_________________________________________________________________
batch_normalization_35 (Batc (None, 405)               1620      
_________________________________________________________________
dropout_21 (Dropout)         (None, 405)               0         
_________________________________________________________________
dense_112 (Dense)            (None, 350)               142100    
_________________________________________________________________
batch_normalization_36 (Batc (None, 350)               1400      
_________________________________________________________________
dropout_22 (Dropout)         (None, 350)               0         
_________________________________________________________________
dense_113 (Dense)            (None, 215)               75465     
_________________________________________________________________
batch_normalization_37 (Batc (None, 215)               860       
_________________________________________________________________
dropout_23 (Dropout)         (None, 215)               0         
_________________________________________________________________
dense_114 (Dense)            (None, 110)               23760     
_________________________________________________________________
batch_normalization_38 (Batc (None, 110)               440       
_________________________________________________________________
dropout_24 (Dropout)         (None, 110)               0         
_________________________________________________________________
dense_115 (Dense)            (None, 55)                6105      
_________________________________________________________________
batch_normalization_39 (Batc (None, 55)                220       
_________________________________________________________________
dropout_25 (Dropout)         (None, 55)                0         
_________________________________________________________________
dense_116 (Dense)            (None, 10)                560       
=================================================================
Total params: 570,455
Trainable params: 568,185
Non-trainable params: 2,270
_________________________________________________________________
```

In [89]:

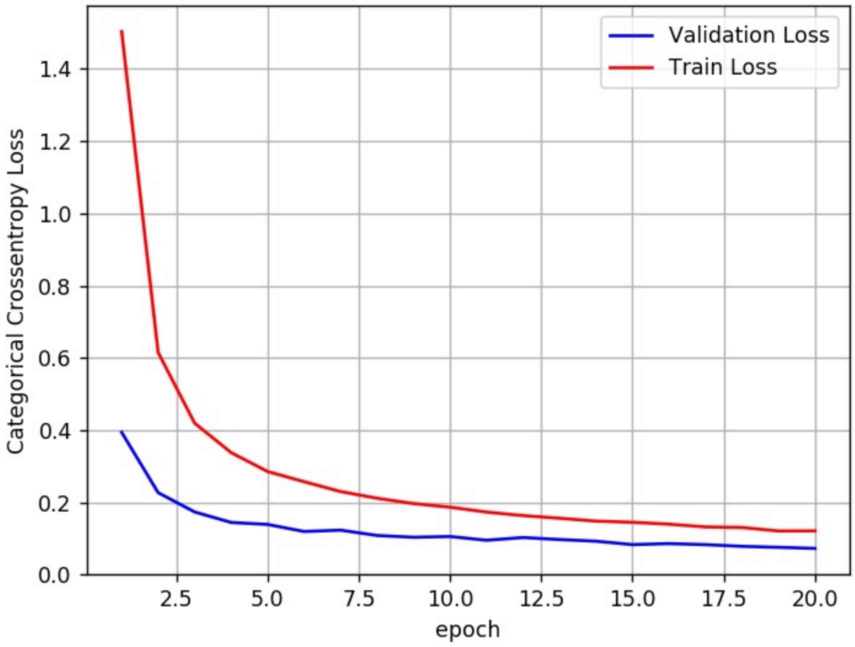
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_
test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 107s 2ms/step - loss: 1.5035 - acc: 0.5129 - val_loss
: 0.3943 - val_acc: 0.8932
Epoch 2/20
60000/60000 [==============================] - 66s 1ms/step - loss: 0.6153 - acc: 0.8143 - val_loss:
0.2274 - val_acc: 0.9368
Epoch 3/20
60000/60000 [==============================] - 67s 1ms/step - loss: 0.4197 - acc: 0.8846 - val_loss:
0.1740 - val_acc: 0.9519
Epoch 4/20
60000/60000 [==============================] - 64s 1ms/step - loss: 0.3382 - acc: 0.9105 - val_loss:
0.1450 - val_acc: 0.9605
Epoch 5/20
60000/60000 [==============================] - 66s 1ms/step - loss: 0.2859 - acc: 0.9262 - val_loss:
0.1394 - val_acc: 0.9631
Epoch 6/20
60000/60000 [==============================] - 62s 1ms/step - loss: 0.2577 - acc: 0.9341 - val_loss:
0.1199 - val_acc: 0.9695
Epoch 7/20
60000/60000 [==============================] - 66s 1ms/step - loss: 0.2304 - acc: 0.9416 - val_loss:
0.1236 - val_acc: 0.9675
Epoch 8/20
60000/60000 [==============================] - 67s 1ms/step - loss: 0.2119 - acc: 0.9470 - val_loss:
0.1089 - val_acc: 0.9723
Epoch 9/20
60000/60000 [==============================] - 63s 1ms/step - loss: 0.1971 - acc: 0.9509 - val_loss:
0.1040 - val_acc: 0.9736
Epoch 10/20
60000/60000 [==============================] - 66s 1ms/step - loss: 0.1872 - acc: 0.9526 - val_loss:
0.1060 - val_acc: 0.9732
Epoch 11/20
60000/60000 [==============================] - 62s 1ms/step - loss: 0.1736 - acc: 0.9572 - val_loss:
0.0957 - val_acc: 0.9753
Epoch 12/20
60000/60000 [==============================] - 69s 1ms/step - loss: 0.1639 - acc: 0.9596 - val_loss:
0.1032 - val_acc: 0.9746: 0.95
Epoch 13/20
60000/60000 [==============================] - 67s 1ms/step - loss: 0.1566 - acc: 0.9621 - val_loss:
0.0977 - val_acc: 0.9761
Epoch 14/20
60000/60000 [==============================] - 70s 1ms/step - loss: 0.1486 - acc: 0.9634 - val_loss:
0.0931 - val_acc: 0.9770
Epoch 15/20
60000/60000 [==============================] - 72s 1ms/step - loss: 0.1454 - acc: 0.9637 - val_loss:
0.0836 - val_acc: 0.9788
Epoch 16/20
60000/60000 [==============================] - 77s 1ms/step - loss: 0.1401 - acc: 0.9655 - val_loss:
0.0864 - val_acc: 0.9785
Epoch 17/20
60000/60000 [==============================] - 78s 1ms/step - loss: 0.1325 - acc: 0.9665 - val_loss:
0.0835 - val_acc: 0.9780
Epoch 18/20
60000/60000 [==============================] - 78s 1ms/step - loss: 0.1310 - acc: 0.9671 - val_loss:
0.0789 - val_acc: 0.9802
Epoch 19/20
60000/60000 [==============================] - 84s 1ms/step - loss: 0.1214 - acc: 0.9697 - val_loss:
0.0761 - val_acc: 0.9812
Epoch 20/20
60000/60000 [==============================] - 75s 1ms/step - loss: 0.1216 - acc: 0.9698 - val_loss:
0.0731 - val_acc: 0.9822
```

In [91]:

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

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

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

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

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

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

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

```
Test score: 0.07309297594130039
Test accuracy: 0.9822
```



Summary

In [1]:

```python
from prettytable import PrettyTable

x=PrettyTable()

x.field_names=["Hidden Layers", "Model",  "Test score", "Test Accuracy"]
x.add_row(["2", "MLP+Relu+Adam",  '0.1050', '0.9766'])
x.add_row(['2',  "MLP+Batch Normalization+ Adam", '0.082', '0.9794'])
x.add_row(['2',  "MLP+Drop Out+Adam",  '0.0605', '0.9834'])
x.add_row(['3', "MLP+Relu+Adam", '0.1062', '0.9787'])
x.add_row(['3', "MLP+Batch Normalization", '0.0795', '98'])
x.add_row(['3', "MLP+Drop Out", '0.0681', '0.9809'])
x.add_row(['5', "MLP+Relu+Adam", '0.1212', '0.9767'])
x.add_row(['5', "MLP+Batch Normaliztion", '0.0817', '0.9798'])
x.add_row(['5', "MLP+Drop Out", '0.073', '0.9822'])

print(x)
```

```
+---------------+-------------------------------+------------+---------------+
| Hidden Layers |             Model             | Test score | Test Accuracy |
+---------------+-------------------------------+------------+---------------+
|       2       |         MLP+Relu+Adam         |   0.1050   |     0.9766    |
|       2       | MLP+Batch Normalization+ Adam |   0.082    |     0.9794    |
|       2       |       MLP+Drop Out+Adam       |   0.0605   |     0.9834    |
|       3       |         MLP+Relu+Adam         |   0.1062   |     0.9787    |
|       3       |    MLP+Batch Normalization    |   0.0795   |       98      |
|       3       |          MLP+Drop Out         |   0.0681   |     0.9809    |
|       5       |         MLP+Relu+Adam         |   0.1212   |     0.9767    |
|       5       |     MLP+Batch Normaliztion    |   0.0817   |     0.9798    |
|       5       |          MLP+Drop Out         |   0.073    |     0.9822    |
+---------------+-------------------------------+------------+---------------+
```

In [ ]: