Keras -- MLPs on MNIST

In [1]:

```
# 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
```

```
Using TensorFlow backend.
```

In [2]:

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

In [3]:

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

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 55s 5us/step
```

In [4]:

```
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 [5]:

```
# 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 [6]:

```
# 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 [7]:

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

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

In [8]:

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

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

In [9]:

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

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

```
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.01176471 0.07058824 0.07058824 0.07058824
0.49411765 0.53333333 0.68627451 0.10196078 0.65098039 1.
0.96862745 0.49803922 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.11764706 0.14117647 0.36862745 0.60392157
0.66666667 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
0.88235294 0.6745098  0.99215686 0.94901961 0.76470588 0.25098039
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.19215686
0.93333333 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
0.99215686 0.99215686 0.99215686 0.98431373 0.36470588 0.32156863
0.32156863 0.21960784 0.15294118 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.07058824 0.85882353 0.99215686
0.99215686 0.99215686 0.99215686 0.99215686 0.77647059 0.71372549
0.96862745 0.94509804 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.31372549 0.61176471 0.41960784 0.99215686
0.99215686 0.80392157 0.04313725 0.         0.16862745 0.60392157
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.05490196 0.00392157 0.60392157 0.99215686 0.35294118
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.54509804 0.99215686 0.74509804 0.00784314 0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.04313725
0.74509804 0.99215686 0.2745098  0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.1372549  0.94509804
0.88235294 0.62745098 0.42352941 0.00392157 0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.31764706 0.94117647 0.99215686
0.99215686 0.46666667 0.09803922 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.17647059 0.72941176 0.99215686 0.99215686
0.58823529 0.10588235 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.0627451  0.36470588 0.98823529 0.99215686 0.73333333
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.97647059 0.99215686 0.97647059 0.25098039 0.
0.         0.         0.         0.         0.         0.
```

```
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          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 [10]:

```python
# 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 [14]:

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

# The Sequential model is a linear stack of layers.
```

```
# you can create a Sequential model by passing a list of layer instances to the constructor:

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

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

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

###

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

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

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

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

####

# https://keras.io/activations/

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

# from keras.layers import Activation, Dense

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

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

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


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

In [12]:

```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

1. 2-hidden layer architecture (784-352-124-10)

1.1 MLP + ReLU activation + ADAM Optimizer

In [15]:

```
model_relu = Sequential()
model_relu.add(Dense(352, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal
(seed=None)))
```

```
model_relu.add(Dense(124, activation='relu', kernel_initializer=he_normal(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, valid
ation_data=(X_test, Y_test))
```

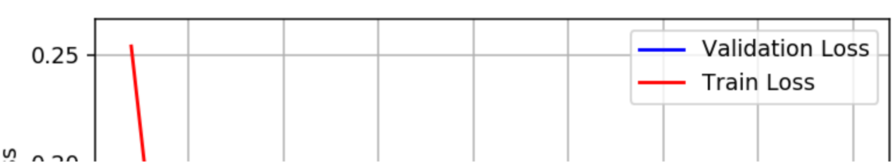
```
WARNING:tensorflow:From C:\Users\NIKHITHA\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:517: The name tf.placeholder is deprecated. Please us
e tf.compat.v1.placeholder instead.

WARNING:tensorflow:From C:\Users\NIKHITHA\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:4185: The name tf.truncated_normal is deprecated. Ple
ase use tf.random.truncated_normal instead.

WARNING:tensorflow:From C:\Users\NIKHITHA\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:4138: The name tf.random_uniform is deprecated. Pleas
e use tf.random.uniform instead.

_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 352)               276320    
_________________________________________________________________
dense_2 (Dense)              (None, 124)               43772     
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1250      
=================================================================
Total params: 321,342
Trainable params: 321,342
Non-trainable params: 0
_________________________________________________________________
None
WARNING:tensorflow:From C:\Users\NIKHITHA\Anaconda3\lib\site-packages\keras\optimizers.py:790: The
name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From C:\Users\NIKHITHA\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:3295: The name tf.log is deprecated. Please use tf.ma
th.log instead.

WARNING:tensorflow:From C:\Users\NIKHITHA\Anaconda3\lib\site-
packages\tensorflow\python\ops\math_grad.py:1250: add_dispatch_support.<locals>.wrapper (from
tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.2540 - acc: 0.9269 -
val_loss: 0.1213 - val_acc: 0.9621
Epoch 2/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.0975 - acc: 0.9714 -
val_loss: 0.0903 - val_acc: 0.9711
Epoch 3/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.0621 - acc: 0.9803 -
val_loss: 0.0731 - val_acc: 0.9783
Epoch 4/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0429 - acc: 0.9869 -
val_loss: 0.0655 - val_acc: 0.9788
Epoch 5/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.0318 - acc: 0.9898 -
val_loss: 0.0664 - val_acc: 0.9782
Epoch 6/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.0231 - acc: 0.9928 -
val_loss: 0.0749 - val_acc: 0.9789
Epoch 7/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0191 - acc: 0.9943 -
val_loss: 0.0820 - val_acc: 0.9769
Epoch 8/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0187 - acc: 0.9941 -
val_loss: 0.0753 - val_acc: 0.9783
Epoch 9/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.0139 - acc: 0.9953 -
val_loss: 0.0653 - val_acc: 0.9806
Epoch 10/20
```

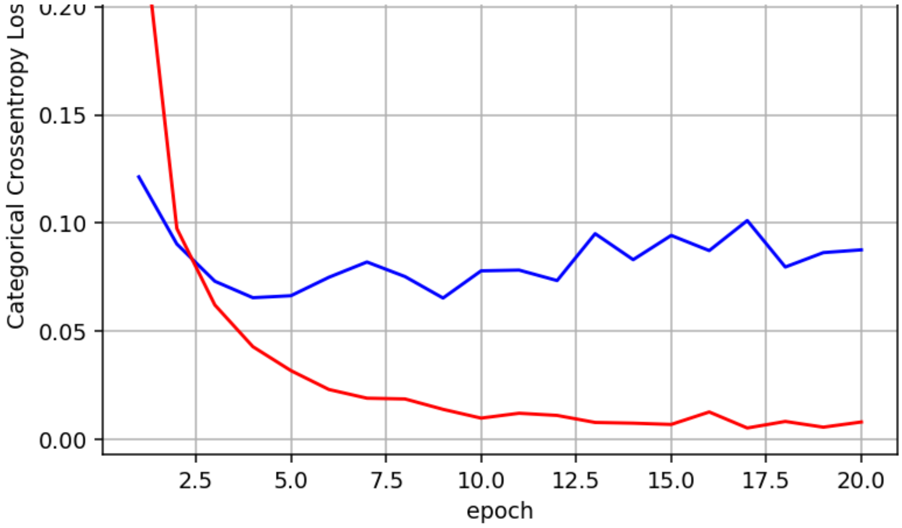
```
Epoch 10/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.0099 - acc: 0.9970 -
val_loss: 0.0779 - val_acc: 0.9804
Epoch 11/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0121 - acc: 0.9960 -
val_loss: 0.0783 - val_acc: 0.9800
Epoch 12/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0111 - acc: 0.9968 -
val_loss: 0.0734 - val_acc: 0.9806
Epoch 13/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0078 - acc: 0.9973 -
val_loss: 0.0951 - val_acc: 0.9801
Epoch 14/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0075 - acc: 0.9975 -
val_loss: 0.0831 - val_acc: 0.9817
Epoch 15/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0069 - acc: 0.9975 -
val_loss: 0.0943 - val_acc: 0.9791
Epoch 16/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0127 - acc: 0.9957 -
val_loss: 0.0873 - val_acc: 0.9821
Epoch 17/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0053 - acc: 0.9980 -
val_loss: 0.1011 - val_acc: 0.9803
Epoch 18/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.0083 - acc: 0.9974 -
val_loss: 0.0797 - val_acc: 0.9837
Epoch 19/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0057 - acc: 0.9980 -
val_loss: 0.0863 - val_acc: 0.9801
Epoch 20/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.0081 - acc: 0.9974 -
val_loss: 0.0876 - val_acc: 0.9816
```

In [16]:

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

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

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

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

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

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

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

```
Test score: 0.08760498947395635
Test accuracy: 0.9816
```





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

In [17]:

```
# 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(352, activation='relu', input_shape=(input_dim,), kernel_initializer=he_norma
l(seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(124, activation='relu', kernel_initializer=he_normal(seed=None)))
model_batch.add(BatchNormalization())

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


model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 352)               276320    
_________________________________________________________________
batch_normalization_1 (Batch (None, 352)               1408      
_________________________________________________________________
dense_5 (Dense)              (None, 124)               43772     
_________________________________________________________________
batch_normalization_2 (Batch (None, 124)               496       
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1250      
=================================================================
Total params: 323,246
Trainable params: 322,294
Non-trainable params: 952
_________________________________________________________________
```

In [18]:

```
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, vali
dation_data=(X_test, Y_test))
```

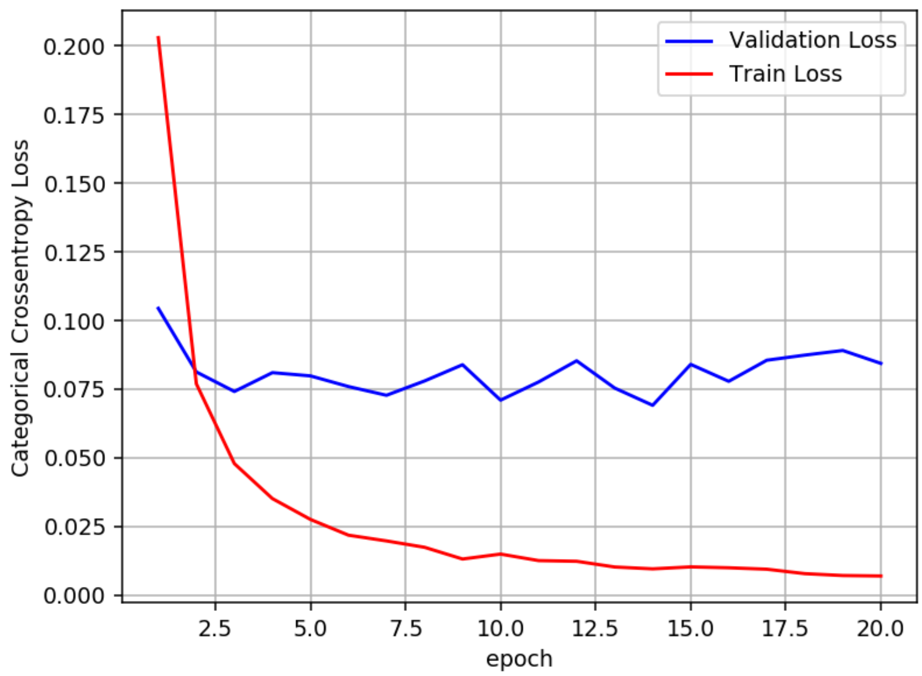
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.2029 - acc: 0.9397 -
val_loss: 0.1044 - val_acc: 0.9672
Epoch 2/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0770 - acc: 0.9766 -
val_loss: 0.0813 - val_acc: 0.9744
Epoch 3/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.0479 - acc: 0.9853 -
val_loss: 0.0742 - val_acc: 0.9777
Epoch 4/20
60000/60000 [==============================] - 5s 75us/step - loss: 0.0352 - acc: 0.9887 -
val_loss: 0.0810 - val_acc: 0.9763
Epoch 5/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0276 - acc: 0.9907 -
val_loss: 0.0798 - val_acc: 0.9760
Epoch 6/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0219 - acc: 0.9934 -
val_loss: 0.0760 - val_acc: 0.9770
Epoch 7/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.0198 - acc: 0.9937 -
val_loss: 0.0728 - val_acc: 0.9815
Epoch 8/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0175 - acc: 0.9944 -
val_loss: 0.0780 - val_acc: 0.9784
Epoch 9/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0132 - acc: 0.9958 -
val_loss: 0.0839 - val_acc: 0.9767
Epoch 10/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0150 - acc: 0.9953 -
val_loss: 0.0710 - val_acc: 0.9810
Epoch 11/20
60000/60000 [==============================] - 6s 100us/step - loss: 0.0126 - acc: 0.9954 -
val_loss: 0.0776 - val_acc: 0.9789
Epoch 12/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.0124 - acc: 0.9962 -
val_loss: 0.0853 - val_acc: 0.9776
Epoch 13/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.0103 - acc: 0.9969 -
val_loss: 0.0754 - val_acc: 0.9807
Epoch 14/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0096 - acc: 0.9968 -
val_loss: 0.0691 - val_acc: 0.9820
Epoch 15/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0103 - acc: 0.9966 -
val_loss: 0.0840 - val_acc: 0.9782
Epoch 16/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.0100 - acc: 0.9966 -
val_loss: 0.0779 - val_acc: 0.9806
Epoch 17/20
60000/60000 [==============================] - 6s 100us/step - loss: 0.0095 - acc: 0.9970 -
val_loss: 0.0855 - val_acc: 0.9796
Epoch 18/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0079 - acc: 0.9972 -
val_loss: 0.0874 - val_acc: 0.9794
Epoch 19/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0072 - acc: 0.9976 -
val_loss: 0.0891 - val_acc: 0.9800
Epoch 20/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0070 - acc: 0.9978 -
val_loss: 0.0844 - val_acc: 0.9788
```

In [19]:

```
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))
```

```
x = list(range(1,nb_epoch+1))

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

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

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

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

```
Test score: 0.08438539689127975
Test accuracy: 0.9788
```



1.3 MLP + Dropout + Adam Optimizer

In [20]:

```
# 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=he_normal
(seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(128, activation='relu', kernel_initializer=he_normal(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\NIKHITHA\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_7 (Dense)              (None, 512)               401920    
_________________________________________________________________
batch_normalization_3 (Batch (None, 512)               2048      
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_8 (Dense)              (None, 128)               65664     
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512       
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1290      
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

In [21]:

```
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))
```

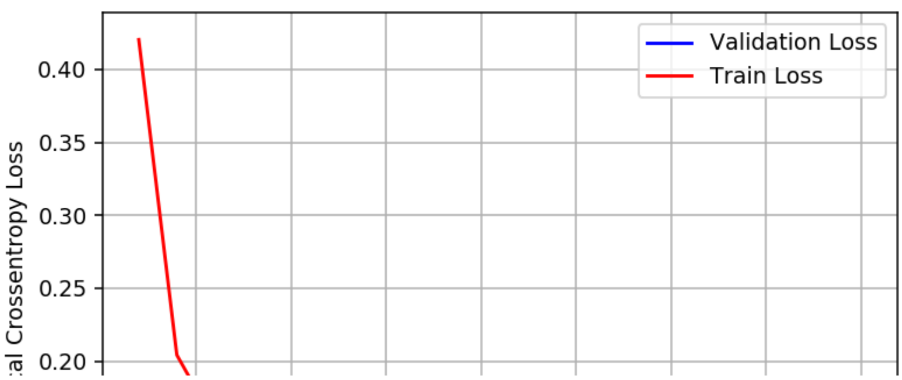
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.4204 - acc: 0.8730 - val_loss: 0.1374 - val_acc: 0.9567
Epoch 2/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.2043 - acc: 0.9384 - val_loss: 0.1108 - val_acc: 0.9642
Epoch 3/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.1579 - acc: 0.9527 - val_loss: 0.0897 - val_acc: 0.9726
Epoch 4/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.1358 - acc: 0.9590 - val_loss: 0.0782 - val_acc: 0.9759
Epoch 5/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.1168 - acc: 0.9640 - val_loss: 0.0707 - val_acc: 0.9777
Epoch 6/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.1066 - acc: 0.9676 - val_loss: 0.0752 - val_acc: 0.9759
Epoch 7/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.1008 - acc: 0.9686 - val_loss: 0.0669 - val_acc: 0.9798
Epoch 8/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0923 - acc: 0.9714 - val_loss: 0.0657 - val_acc: 0.9792
Epoch 9/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.0885 - acc: 0.9727 - val_loss: 0.0638 - val_acc: 0.9803
Epoch 10/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0840 - acc: 0.9740 - val_loss: 0.0603 - val_acc: 0.9810
Epoch 11/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.0796 - acc: 0.9755 - val_loss: 0.0602 - val_acc: 0.9823
Epoch 12/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0712 - acc: 0.9778 - val_loss: 0.0559 - val_acc: 0.9822
```

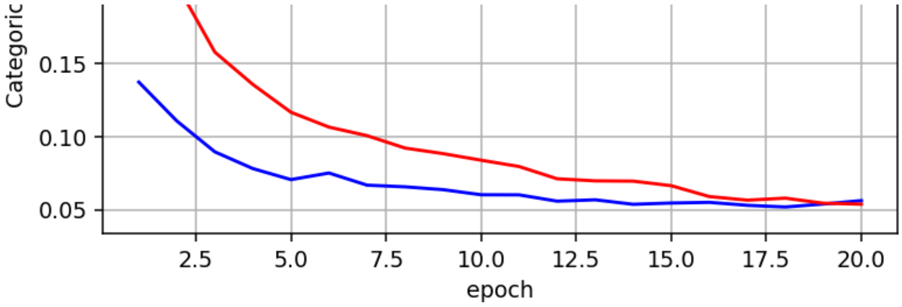
```
val_loss: 0.0559 - val_acc: 0.9822
Epoch 13/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0698 - acc: 0.9779 -
val_loss: 0.0568 - val_acc: 0.9823
Epoch 14/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.0696 - acc: 0.9779 -
val_loss: 0.0538 - val_acc: 0.9839
Epoch 15/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0665 - acc: 0.9790 -
val_loss: 0.0546 - val_acc: 0.9829
Epoch 16/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0591 - acc: 0.9815 -
val_loss: 0.0551 - val_acc: 0.9839
Epoch 17/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.0566 - acc: 0.9820 -
val_loss: 0.0531 - val_acc: 0.9841
Epoch 18/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0580 - acc: 0.9813 -
val_loss: 0.0519 - val_acc: 0.9838
Epoch 19/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0545 - acc: 0.9833 -
val_loss: 0.0540 - val_acc: 0.9846
Epoch 20/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0539 - acc: 0.9822 -
val_loss: 0.0563 - val_acc: 0.9830
```

In [22]:

```python
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, va
lidation_data=(X_test, Y_test))

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

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

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

```
Test score: 0.05628199922235217
Test accuracy: 0.983
```





2. 3-hidden layer architecture (784-454-232-120-10)

2.1 MLP + ReLU activation + ADAM Optimizer

In [23]:

```
model_relu = Sequential()
model_relu.add(Dense(454, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal
(seed=None)))
model_relu.add(Dense(232, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(120, activation='relu', kernel_initializer=he_normal(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, valid
ation_data=(X_test, Y_test))
```

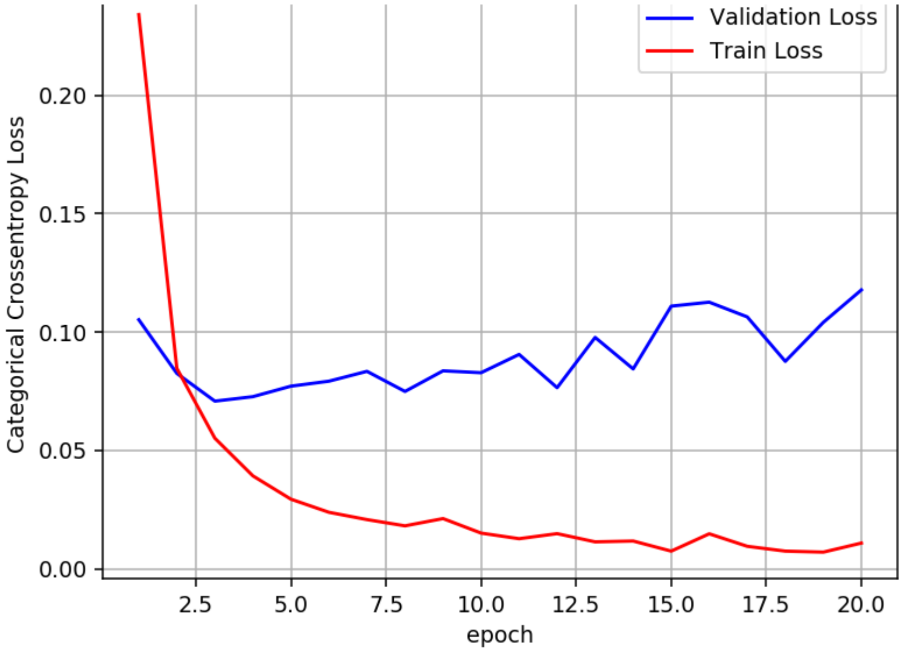
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 454)               356390
_________________________________________________________________
dense_11 (Dense)             (None, 232)               105560
_________________________________________________________________
dense_12 (Dense)             (None, 120)               27960
_________________________________________________________________
dense_13 (Dense)             (None, 10)                1210
=================================================================
Total params: 491,120
Trainable params: 491,120
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.2340 - acc: 0.9310 -
val_loss: 0.1052 - val_acc: 0.9669
Epoch 2/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0847 - acc: 0.9736 -
val_loss: 0.0826 - val_acc: 0.9733
Epoch 3/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.0552 - acc: 0.9820 -
val_loss: 0.0708 - val_acc: 0.9772
Epoch 4/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0393 - acc: 0.9872 -
val_loss: 0.0727 - val_acc: 0.9781
Epoch 5/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0294 - acc: 0.9906 -
val_loss: 0.0772 - val_acc: 0.9790
Epoch 6/20
60000/60000 [==============================] - 5s 92us/step - loss: 0.0239 - acc: 0.9920 -
val_loss: 0.0793 - val_acc: 0.9768
Epoch 7/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0208 - acc: 0.9930 -
val_loss: 0.0834 - val_acc: 0.9787
Epoch 8/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0182 - acc: 0.9943 -
```

```
val_loss: 0.0749 - val_acc: 0.9806
Epoch 9/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0213 - acc: 0.9931 -
val_loss: 0.0836 - val_acc: 0.9803
Epoch 10/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0151 - acc: 0.9948 -
val_loss: 0.0828 - val_acc: 0.9806
Epoch 11/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0128 - acc: 0.9961 -
val_loss: 0.0906 - val_acc: 0.9783
Epoch 12/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0149 - acc: 0.9955 -
val_loss: 0.0765 - val_acc: 0.9828
Epoch 13/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0114 - acc: 0.9964 -
val_loss: 0.0977 - val_acc: 0.9783
Epoch 14/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.0118 - acc: 0.9963 -
val_loss: 0.0844 - val_acc: 0.9806
Epoch 15/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0075 - acc: 0.9975 -
val_loss: 0.1109 - val_acc: 0.9792
Epoch 16/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0148 - acc: 0.9954 -
val_loss: 0.1126 - val_acc: 0.9785
Epoch 17/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0095 - acc: 0.9973 -
val_loss: 0.1064 - val_acc: 0.9791
Epoch 18/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.0075 - acc: 0.9978 -
val_loss: 0.0876 - val_acc: 0.9820
Epoch 19/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0071 - acc: 0.9975 -
val_loss: 0.1041 - val_acc: 0.9798
Epoch 20/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0109 - acc: 0.9967 -
val_loss: 0.1178 - val_acc: 0.9778
```

In [24]:

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

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

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

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

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

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

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

```
Test score: 0.11779277188464211
Test accuracy: 0.9778
```



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

In [25]:

```
# 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(454, activation='relu', input_shape=(input_dim,), kernel_initializer=he_norma
l(seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(232, activation='relu', kernel_initializer=he_normal(seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(120, activation='relu', kernel_initializer=he_normal(seed=None)))
model_batch.add(BatchNormalization())

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


model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_14 (Dense)             (None, 454)               356390    
_________________________________________________________________
batch_normalization_5 (Batch (None, 454)               1816      
_________________________________________________________________
dense_15 (Dense)             (None, 232)               105560    
_________________________________________________________________
batch_normalization_6 (Batch (None, 232)               928       
_________________________________________________________________
dense_16 (Dense)             (None, 120)               27960     
_________________________________________________________________
batch_normalization_7 (Batch (None, 120)               480       
_________________________________________________________________
dense_17 (Dense)             (None, 10)                1210      
```

```
=================================================================
Total params: 494,344
Trainable params: 492,732
Non-trainable params: 1,612
_________________________________________________________________
```

In [26]:

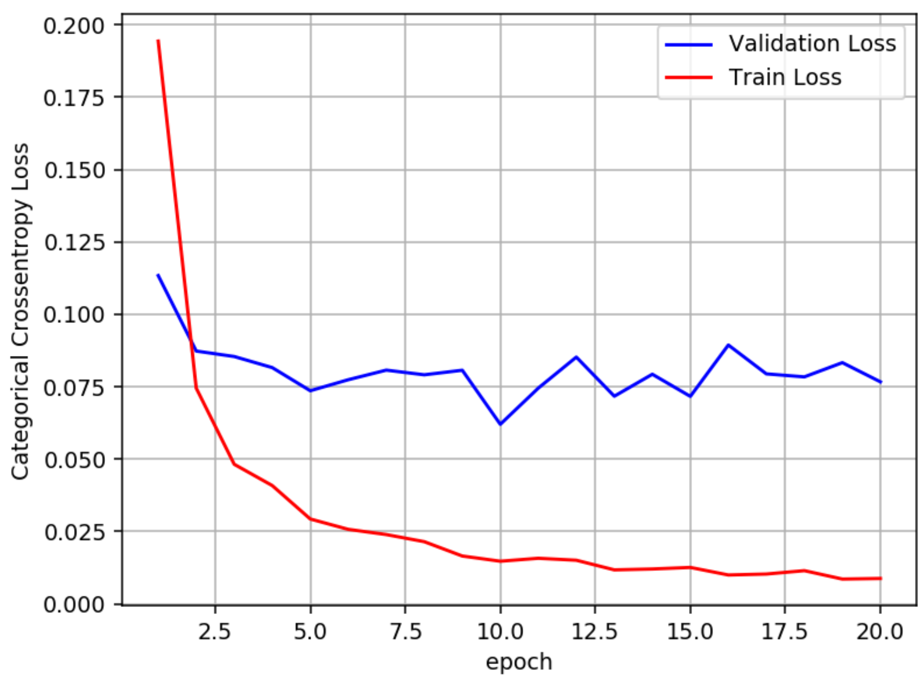
```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, vali
dation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.1944 - acc: 0.9408 - val_l
oss: 0.1134 - val_acc: 0.9649
Epoch 2/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.0745 - acc: 0.9767 -
val_loss: 0.0873 - val_acc: 0.9727
Epoch 3/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0482 - acc: 0.9852 -
val_loss: 0.0854 - val_acc: 0.9732
Epoch 4/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0408 - acc: 0.9872 -
val_loss: 0.0816 - val_acc: 0.9747
Epoch 5/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0293 - acc: 0.9908 -
val_loss: 0.0736 - val_acc: 0.9769
Epoch 6/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0257 - acc: 0.9919 -
val_loss: 0.0774 - val_acc: 0.9754
Epoch 7/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0239 - acc: 0.9921 -
val_loss: 0.0807 - val_acc: 0.9762
Epoch 8/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.0215 - acc: 0.9929 -
val_loss: 0.0791 - val_acc: 0.9771
Epoch 9/20
60000/60000 [==============================] - 6s 103us/step - loss: 0.0165 - acc: 0.9943 -
val_loss: 0.0807 - val_acc: 0.9780
Epoch 10/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0147 - acc: 0.9951 -
val_loss: 0.0620 - val_acc: 0.9810
Epoch 11/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0157 - acc: 0.9944 -
val_loss: 0.0745 - val_acc: 0.9798
Epoch 12/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0150 - acc: 0.9945 -
val_loss: 0.0852 - val_acc: 0.9779
Epoch 13/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0117 - acc: 0.9962 -
val_loss: 0.0717 - val_acc: 0.9817
Epoch 14/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0120 - acc: 0.9961 -
val_loss: 0.0793 - val_acc: 0.9794
Epoch 15/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0126 - acc: 0.9959 -
val_loss: 0.0717 - val_acc: 0.9817
Epoch 16/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.0100 - acc: 0.9964 -
val_loss: 0.0894 - val_acc: 0.9776
Epoch 17/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.0103 - acc: 0.9964 -
val_loss: 0.0794 - val_acc: 0.9809
Epoch 18/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0114 - acc: 0.9958 -
val_loss: 0.0784 - val_acc: 0.9812
Epoch 19/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0085 - acc: 0.9973 -
val_loss: 0.0833 - val_acc: 0.9814
Epoch 20/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0087 - acc: 0.9972 -
val_loss: 0.0767 - val_acc: 0.9809
```

In [27]:

```
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, va
lidation_data=(X_test, Y_test))

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

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

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

```
Test score: 0.07667597182277495
Test accuracy: 0.9809
```



2.3 MLP + Dropout + Adam Optimizer

In [28]:

```
# 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(454, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal
(seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(232, activation='relu', kernel_initializer=he_normal(seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(120, activation='relu', kernel_initializer=he_normal(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_18 (Dense)             (None, 454)               356390    
_________________________________________________________________
batch_normalization_8 (Batch (None, 454)               1816      
_________________________________________________________________
dropout_3 (Dropout)          (None, 454)               0         
_________________________________________________________________
dense_19 (Dense)             (None, 232)               105560    
_________________________________________________________________
batch_normalization_9 (Batch (None, 232)               928       
_________________________________________________________________
dropout_4 (Dropout)          (None, 232)               0         
_________________________________________________________________
dense_20 (Dense)             (None, 120)               27960     
_________________________________________________________________
batch_normalization_10 (Batc (None, 120)               480       
_________________________________________________________________
dropout_5 (Dropout)          (None, 120)               0         
_________________________________________________________________
dense_21 (Dense)             (None, 10)                1210      
=================================================================
Total params: 494,344
Trainable params: 492,732
Non-trainable params: 1,612
_________________________________________________________________
```

In [29]:

```
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, valid
ation_data=(X_test, Y_test))
```

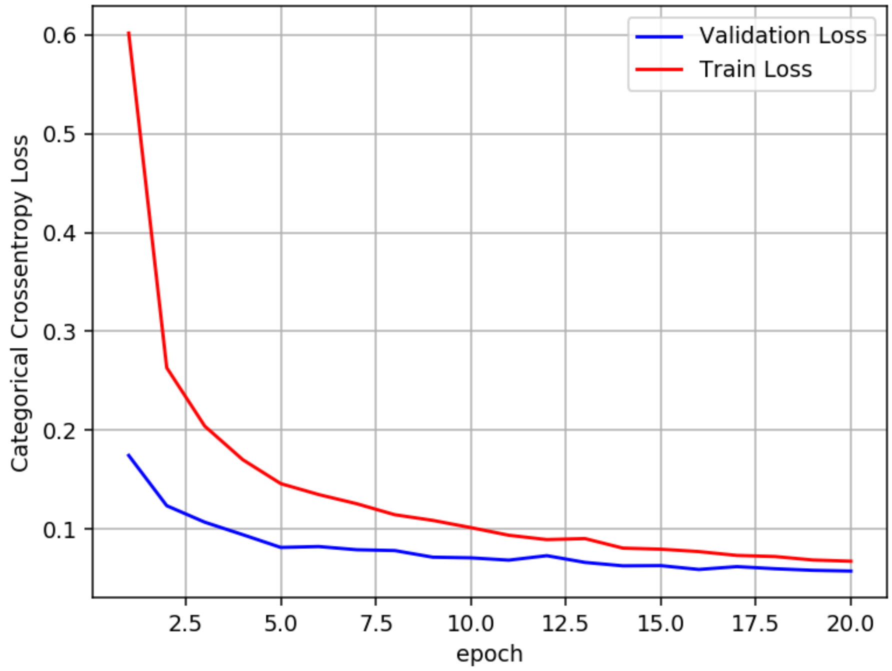
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 12s 193us/step - loss: 0.6018 - acc: 0.8176 - val_l
oss: 0.1744 - val_acc: 0.9470
Epoch 2/20
60000/60000 [==============================] - 10s 162us/step - loss: 0.2632 - acc: 0.9222 - val_l
oss: 0.1235 - val_acc: 0.9624
Epoch 3/20
60000/60000 [==============================] - 10s 162us/step - loss: 0.2041 - acc: 0.9395 - val_l
oss: 0.1069 - val_acc: 0.9677
Epoch 4/20
60000/60000 [==============================] - 10s 162us/step - loss: 0.1701 - acc: 0.9496 - val_l
oss: 0.0942 - val_acc: 0.9716
Epoch 5/20
60000/60000 [==============================] - 10s 162us/step - loss: 0.1459 - acc: 0.9572 - val_l
oss: 0.0813 - val_acc: 0.9748
Epoch 6/20
60000/60000 [==============================] - 10s 163us/step - loss: 0.1348 - acc: 0.9599 - val_l
oss: 0.0822 - val_acc: 0.9751
Epoch 7/20
60000/60000 [==============================] - 10s 162us/step - loss: 0.1255 - acc: 0.9629 - val_l
```

```
oss: 0.0790 - val_acc: 0.9759
Epoch 8/20
60000/60000 [==============================] - 10s 162us/step - loss: 0.1144 - acc: 0.9666 - val_l
oss: 0.0781 - val_acc: 0.9756
Epoch 9/20
60000/60000 [==============================] - 10s 162us/step - loss: 0.1087 - acc: 0.9685 - val_l
oss: 0.0714 - val_acc: 0.9782
Epoch 10/20
60000/60000 [==============================] - 10s 162us/step - loss: 0.1014 - acc: 0.9693 - val_l
oss: 0.0708 - val_acc: 0.9787
Epoch 11/20
60000/60000 [==============================] - 10s 158us/step - loss: 0.0937 - acc: 0.9721 - val_l
oss: 0.0685 - val_acc: 0.9814
Epoch 12/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.0893 - acc: 0.9732 - val_l
oss: 0.0730 - val_acc: 0.9795
Epoch 13/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.0903 - acc: 0.9730 - val_l
oss: 0.0662 - val_acc: 0.9811
Epoch 14/20
60000/60000 [==============================] - 10s 162us/step - loss: 0.0806 - acc: 0.9759 - val_l
oss: 0.0627 - val_acc: 0.9824
Epoch 15/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.0795 - acc: 0.9766 -
val_loss: 0.0629 - val_acc: 0.9812
Epoch 16/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0771 - acc: 0.9766 -
val_loss: 0.0590 - val_acc: 0.9829
Epoch 17/20
60000/60000 [==============================] - 10s 159us/step - loss: 0.0732 - acc: 0.9769 - val_l
oss: 0.0619 - val_acc: 0.9824
Epoch 18/20
60000/60000 [==============================] - 10s 159us/step - loss: 0.0721 - acc: 0.9777 - val_l
oss: 0.0598 - val_acc: 0.9825
Epoch 19/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.0686 - acc: 0.9784 - val_l
oss: 0.0582 - val_acc: 0.9821
Epoch 20/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.0674 - acc: 0.9788 - val_l
oss: 0.0574 - val_acc: 0.9857
```

In [30]:

```
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, va
lidation_data=(X_test, Y_test))

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

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

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

```
Test score: 0.05740614543475094
Test accuracy: 0.9857
```



3. 5-hidden layer architecture (784-320-280-230-160-80-10)

3.1 MLP + ReLU activation + ADAM Optimizer

In [31]:

```
model_relu = Sequential()
model_relu.add(Dense(320, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal
(seed=None)))
model_relu.add(Dense(280, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(230, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(160, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(80, activation='relu', kernel_initializer=he_normal(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, valid
ation_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_22 (Dense)             (None, 320)               251200    
_________________________________________________________________
dense_23 (Dense)             (None, 280)               89880     
_________________________________________________________________
dense_24 (Dense)             (None, 230)               64630     
_________________________________________________________________
dense_25 (Dense)             (None, 160)               36960     
_________________________________________________________________
dense_26 (Dense)             (None, 80)                12880     
_________________________________________________________________
dense_27 (Dense)             (None, 10)                810       
=================================================================
Total params: 456,360
Trainable params: 456,360
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
```

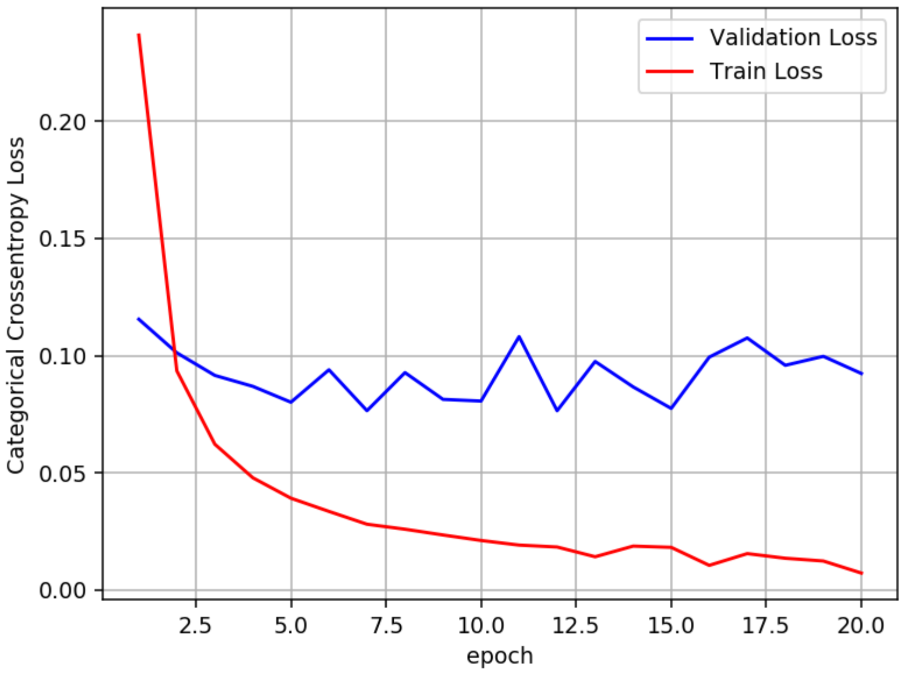
```
60000/60000 [==============================] - 7s 115us/step - loss: 0.2368 - acc: 0.9289 -
val_loss: 0.1155 - val_acc: 0.9640
Epoch 2/20
60000/60000 [==============================] - 6s 100us/step - loss: 0.0935 - acc: 0.9714 -
val_loss: 0.1012 - val_acc: 0.9694
Epoch 3/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0622 - acc: 0.9806 -
val_loss: 0.0916 - val_acc: 0.9725
Epoch 4/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0479 - acc: 0.9849 -
val_loss: 0.0869 - val_acc: 0.9759
Epoch 5/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0392 - acc: 0.9869 -
val_loss: 0.0801 - val_acc: 0.9761
Epoch 6/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0335 - acc: 0.9892 -
val_loss: 0.0940 - val_acc: 0.9729
Epoch 7/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0280 - acc: 0.9914 -
val_loss: 0.0765 - val_acc: 0.9800
Epoch 8/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0259 - acc: 0.9914 -
val_loss: 0.0928 - val_acc: 0.9758
Epoch 9/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.0235 - acc: 0.9924 -
val_loss: 0.0813 - val_acc: 0.9782
Epoch 10/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0211 - acc: 0.9933 -
val_loss: 0.0806 - val_acc: 0.9801
Epoch 11/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0191 - acc: 0.9935 -
val_loss: 0.1081 - val_acc: 0.9749
Epoch 12/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.0183 - acc: 0.9944 -
val_loss: 0.0765 - val_acc: 0.9807
Epoch 13/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0142 - acc: 0.9956 -
val_loss: 0.0975 - val_acc: 0.9775
Epoch 14/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0187 - acc: 0.9944 -
val_loss: 0.0867 - val_acc: 0.9782
Epoch 15/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0182 - acc: 0.9946 -
val_loss: 0.0775 - val_acc: 0.9813
Epoch 16/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0105 - acc: 0.9968 -
val_loss: 0.0994 - val_acc: 0.9798
Epoch 17/20
60000/60000 [==============================] - 6s 101us/step - loss: 0.0155 - acc: 0.9950 -
val_loss: 0.1075 - val_acc: 0.9779
Epoch 18/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0135 - acc: 0.9959 -
val_loss: 0.0959 - val_acc: 0.9795
Epoch 19/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0123 - acc: 0.9963 -
val_loss: 0.0997 - val_acc: 0.9781
Epoch 20/20
60000/60000 [==============================] - 6s 100us/step - loss: 0.0072 - acc: 0.9979 -
val_loss: 0.0924 - val_acc: 0.9812
```

In [32]:

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

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

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

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

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

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

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

```
Test score: 0.09238322618362185
Test accuracy: 0.9812
```



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

In [33]:

```
# 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(320, activation='relu', input_shape=(input_dim,), kernel_initializer=he_norma
l(seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(280, activation='relu', kernel_initializer=he_normal(seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(230, activation='relu', kernel_initializer=he_normal(seed=None)))
model_batch.add(BatchNormalization())
```

```
model_batch.add(Dense(160, activation='relu', kernel_initializer=he_normal(seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(80, activation='relu', kernel_initializer=he_normal(seed=None)))
model_batch.add(BatchNormalization())

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


model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_28 (Dense)             (None, 320)               251200    
_________________________________________________________________
batch_normalization_11 (Batc (None, 320)               1280      
_________________________________________________________________
dense_29 (Dense)             (None, 280)               89880     
_________________________________________________________________
batch_normalization_12 (Batc (None, 280)               1120      
_________________________________________________________________
dense_30 (Dense)             (None, 230)               64630     
_________________________________________________________________
batch_normalization_13 (Batc (None, 230)               920       
_________________________________________________________________
dense_31 (Dense)             (None, 160)               36960     
_________________________________________________________________
batch_normalization_14 (Batc (None, 160)               640       
_________________________________________________________________
dense_32 (Dense)             (None, 80)                12880     
_________________________________________________________________
batch_normalization_15 (Batc (None, 80)                320       
_________________________________________________________________
dense_33 (Dense)             (None, 10)                810       
=================================================================
Total params: 460,640
Trainable params: 458,500
Non-trainable params: 2,140
_________________________________________________________________
```

In [34]:

```
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, vali
dation_data=(X_test, Y_test))
```

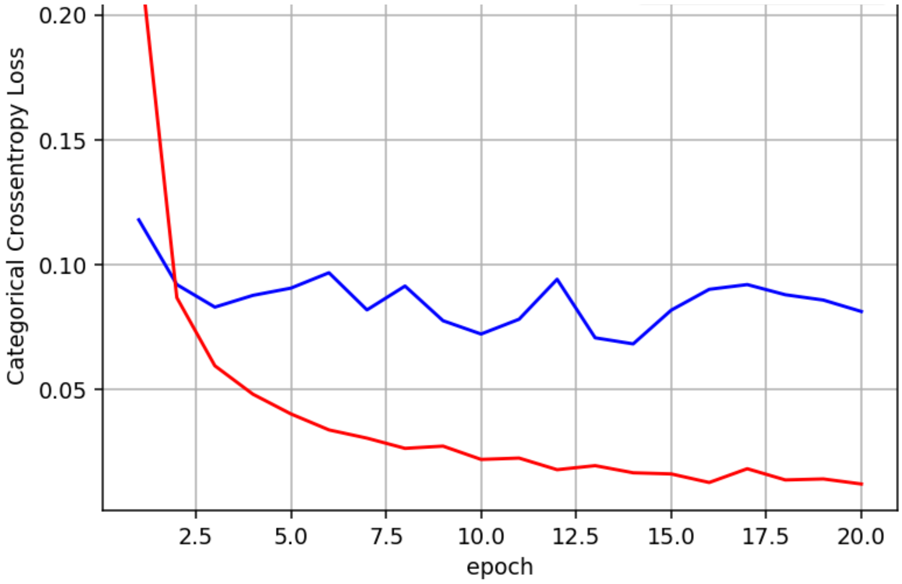
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 12s 201us/step - loss: 0.2278 - acc: 0.9324 - val_l
oss: 0.1180 - val_acc: 0.9658
Epoch 2/20
60000/60000 [==============================] - 9s 158us/step - loss: 0.0869 - acc: 0.9731 -
val_loss: 0.0921 - val_acc: 0.9710
Epoch 3/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.0596 - acc: 0.9808 -
val_loss: 0.0831 - val_acc: 0.9722
Epoch 4/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.0482 - acc: 0.9842 -
val_loss: 0.0878 - val_acc: 0.9730
Epoch 5/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.0403 - acc: 0.9863 -
val_loss: 0.0907 - val_acc: 0.9729
Epoch 6/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.0339 - acc: 0.9892 -
val_loss: 0.0968 - val_acc: 0.9739
Epoch 7/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.0306 - acc: 0.9895 -
val_loss: 0.0820 - val_acc: 0.9774
Epoch 8/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.0265 - acc: 0.9913 -
val_loss: 0.0915 - val_acc: 0.9760
Epoch 9/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0274 - acc: 0.9909
```

```
60000/60000 [==============================] - 8s 126us/step - loss: 0.0274 - acc: 0.9909 -
val_loss: 0.0776 - val_acc: 0.9779
Epoch 10/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.0221 - acc: 0.9927 -
val_loss: 0.0723 - val_acc: 0.9795
Epoch 11/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.0227 - acc: 0.9925 -
val_loss: 0.0782 - val_acc: 0.9791
Epoch 12/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.0180 - acc: 0.9937 -
val_loss: 0.0942 - val_acc: 0.9752
Epoch 13/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.0196 - acc: 0.9936 -
val_loss: 0.0708 - val_acc: 0.9803
Epoch 14/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.0168 - acc: 0.9943 -
val_loss: 0.0684 - val_acc: 0.9825
Epoch 15/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.0163 - acc: 0.9945 -
val_loss: 0.0819 - val_acc: 0.9794
Epoch 16/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.0129 - acc: 0.9959 -
val_loss: 0.0902 - val_acc: 0.9789
Epoch 17/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0184 - acc: 0.9938 -
val_loss: 0.0921 - val_acc: 0.9765
Epoch 18/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0139 - acc: 0.9953 -
val_loss: 0.0880 - val_acc: 0.9790
Epoch 19/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0143 - acc: 0.9949 -
val_loss: 0.0859 - val_acc: 0.9806
Epoch 20/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.0123 - acc: 0.9960 -
val_loss: 0.0813 - val_acc: 0.9796
```

In [35]:

```python
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, va
lidation_data=(X_test, Y_test))

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

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

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

```
Test score: 0.0813284714843132
Test accuracy: 0.9796
```



3.3 MLP + Dropout + Adam Optimizer

In [36]:

```
# 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(320, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(280, activation='relu', kernel_initializer=he_normal(seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(230, activation='relu', kernel_initializer=he_normal(seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(160, activation='relu', kernel_initializer=he_normal(seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(80, activation='relu', kernel_initializer=he_normal(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_34 (Dense)             (None, 320)               251200
_________________________________________________________________
batch_normalization_16 (Batc (None, 320)               1280
_________________________________________________________________
dropout_6 (Dropout)          (None, 320)               0
_________________________________________________________________
dense_35 (Dense)             (None, 280)               89880
_________________________________________________________________
batch_normalization_17 (Batc (None, 280)               1120
_________________________________________________________________
dropout_7 (Dropout)          (None, 280)               0
```

```
_________________________________________________________________
dense_36 (Dense)             (None, 230)               64630
_________________________________________________________________
batch_normalization_18 (Batc (None, 230)               920
_________________________________________________________________
dropout_8 (Dropout)          (None, 230)               0
_________________________________________________________________
dense_37 (Dense)             (None, 160)               36960
_________________________________________________________________
batch_normalization_19 (Batc (None, 160)               640
_________________________________________________________________
dropout_9 (Dropout)          (None, 160)               0
_________________________________________________________________
dense_38 (Dense)             (None, 80)                12880
_________________________________________________________________
batch_normalization_20 (Batc (None, 80)                320
_________________________________________________________________
dropout_10 (Dropout)         (None, 80)                0
_________________________________________________________________
dense_39 (Dense)             (None, 10)                810
=================================================================
Total params: 460,640
Trainable params: 458,500
Non-trainable params: 2,140
_________________________________________________________________
```

In [37]:

```python
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, valid
ation_data=(X_test, Y_test))
```

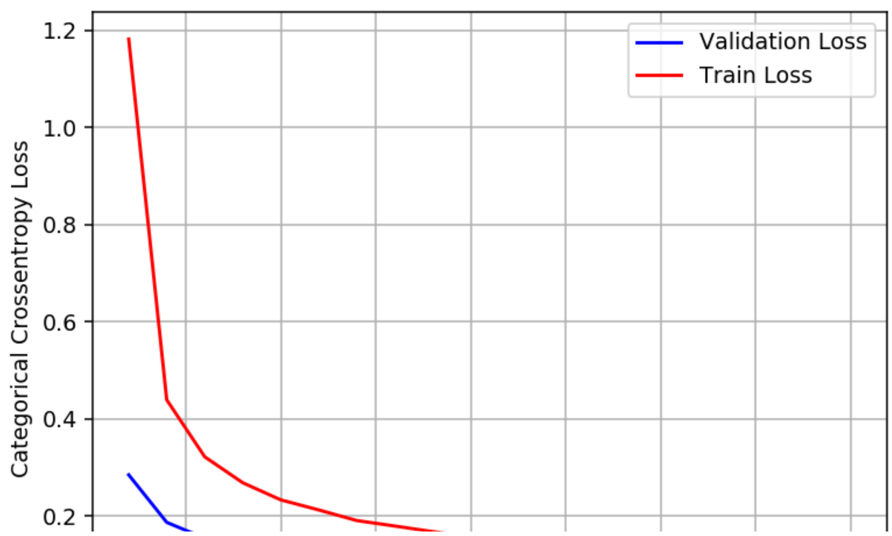
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 17s 280us/step - loss: 1.1816 - acc: 0.6242 - val_l
oss: 0.2850 - val_acc: 0.9147
Epoch 2/20
60000/60000 [==============================] - 12s 204us/step - loss: 0.4393 - acc: 0.8740 - val_l
oss: 0.1872 - val_acc: 0.9476
Epoch 3/20
60000/60000 [==============================] - 12s 208us/step - loss: 0.3221 - acc: 0.9100 - val_l
oss: 0.1546 - val_acc: 0.9544
Epoch 4/20
60000/60000 [==============================] - 12s 203us/step - loss: 0.2687 - acc: 0.9280 - val_l
oss: 0.1426 - val_acc: 0.9600
Epoch 5/20
60000/60000 [==============================] - 12s 207us/step - loss: 0.2335 - acc: 0.9384 - val_l
oss: 0.1239 - val_acc: 0.9668
Epoch 6/20
60000/60000 [==============================] - 12s 204us/step - loss: 0.2128 - acc: 0.9434 - val_l
oss: 0.1096 - val_acc: 0.9716
Epoch 7/20
60000/60000 [==============================] - 12s 207us/step - loss: 0.1909 - acc: 0.9496 - val_l
oss: 0.1030 - val_acc: 0.9736
Epoch 8/20
60000/60000 [==============================] - 12s 204us/step - loss: 0.1797 - acc: 0.9533 - val_l
oss: 0.0979 - val_acc: 0.9743
Epoch 9/20
60000/60000 [==============================] - 12s 202us/step - loss: 0.1680 - acc: 0.9556 - val_l
oss: 0.0988 - val_acc: 0.9745
Epoch 10/20
60000/60000 [==============================] - 12s 202us/step - loss: 0.1566 - acc: 0.9589 - val_l
oss: 0.0965 - val_acc: 0.9741
Epoch 11/20
60000/60000 [==============================] - 12s 203us/step - loss: 0.1488 - acc: 0.9612 - val_l
oss: 0.0908 - val_acc: 0.9772
Epoch 12/20
60000/60000 [==============================] - 12s 203us/step - loss: 0.1420 - acc: 0.9620 - val_l
oss: 0.0880 - val_acc: 0.9781
Epoch 13/20
60000/60000 [==============================] - 12s 204us/step - loss: 0.1372 - acc: 0.9635 - val_l
oss: 0.0856 - val_acc: 0.9778
Epoch 14/20
60000/60000 [==============================] - 12s 199us/step - loss: 0.1392 - acc: 0.9639 - val_l
oss: 0.0850 - val_acc: 0.9781
```

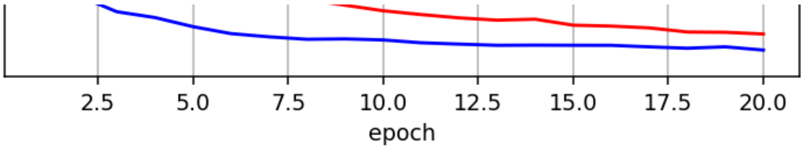
```
oss: 0.0859 - val_acc: 0.9781
Epoch 15/20
60000/60000 [==============================] - 12s 198us/step - loss: 0.1269 - acc: 0.9662 - val_l
oss: 0.0856 - val_acc: 0.9771
Epoch 16/20
60000/60000 [==============================] - 12s 197us/step - loss: 0.1250 - acc: 0.9675 - val_l
oss: 0.0855 - val_acc: 0.9792
Epoch 17/20
60000/60000 [==============================] - 12s 199us/step - loss: 0.1213 - acc: 0.9679 - val_l
oss: 0.0823 - val_acc: 0.9797
Epoch 18/20
60000/60000 [==============================] - 12s 199us/step - loss: 0.1129 - acc: 0.9692 - val_l
oss: 0.0795 - val_acc: 0.9792
Epoch 19/20
60000/60000 [==============================] - 12s 199us/step - loss: 0.1123 - acc: 0.9700 - val_l
oss: 0.0824 - val_acc: 0.9773
Epoch 20/20
60000/60000 [==============================] - 12s 199us/step - loss: 0.1088 - acc: 0.9710 - val_l
oss: 0.0758 - val_acc: 0.9806
```

In [38]:

```
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, va
lidation_data=(X_test, Y_test))

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

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

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

```
Test score: 0.07577368428185582
Test accuracy: 0.9806
```





4. CONCLUSION

In [40]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Numer of Layers", "BN","Dropout", "Accuracy"]
x.add_row(["2", 'NO',"NO", 0.981])
x.add_row(["2", 'YES','NO', 0.978])
x.add_row(["2", 'NO',0.5, 0.983])

x.add_row(["3", 'NO',"NO", 0.977])
x.add_row(["3", 'YES','NO', 0.980])
x.add_row(["3", 'NO',0.5, 0.985])

x.add_row(["5", 'NO','NO', 0.981])
x.add_row(["5", 'YES','NO', 0.979])
x.add_row(["5", 'NO',0.5, 0.980])
print(x)
```

```
+-----------------+-----+---------+----------+
| Numer of Layers |  BN | Dropout | Accuracy |
+-----------------+-----+---------+----------+
|        2        |  NO |    NO   |  0.981   |
|        2        | YES |    NO   |  0.978   |
|        2        |  NO |   0.5   |  0.983   |
|        3        |  NO |    NO   |  0.977   |
|        3        | YES |    NO   |   0.98   |
|        3        |  NO |   0.5   |  0.985   |
|        5        |  NO |    NO   |  0.981   |
|        5        | YES |    NO   |  0.979   |
|        5        |  NO |   0.5   |   0.98   |
+-----------------+-----+---------+----------+
```

In [ ]: