In [1]:
```
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
```

```
E:\conda\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversi
on of the second argument of issubdtype from `float` to `np.floating` i
s deprecated. In future, it will be treated as `np.float64 == np.dtype
(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

In [2]:
```
def plt_dynamic(X,vy,ty,ax,colors=['b']):
    ax.plot(X,vy,'b',label='validation_test')
    ax.plot(X,ty,'r',label='Train loss')
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

In [3]:
```
(X_train,y_train),(X_test,y_test)=mnist.load_data()
```

In [4]:
```
print(X_train.shape);print(y_train.shape);print("each image is shape of
 ",X_train.shape[1],X_train.shape[2])
```

```
(60000, 28, 28)
(60000,)
each image is shape of  28 28
```

In [5]:
```
print(X_test.shape);print(y_test.shape);print("each image is shape of "
,X_test.shape[1],X_test.shape[2])
```

```
(10000, 28, 28)
(10000,)
each image is shape of  28 28
```

In [6]:
```
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 3d to 2d
print("number of training examples:",X_train.shape[0],"each images is s
hape of :",X_train.shape[1])
print("number of test examples:",X_test.shape[0],"each images is shape
 of :",X_test.shape[1])
```

```
number of training examples: 60000 each images is shape of : 784
number of test examples: 10000 each images is shape of : 784
```

In [8]:
```
X_train[0]
```

Out[8]:
```
array([  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   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], dtype=uint8)
```

In [9]:

```
# we need to normalize this data
```

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

In [10]:
```
#the data is normalized
X_train[0]
```

Out[10]:
```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 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]:

```
print("class label of first image",y_train[0])
y_train=np_utils.to_categorical(y_train)
y_test=np_utils.to_categorical(y_test)
print("after converting into one hot encoding :",y_train[0])
```

```
class label of first image 5
after converting into one hot encoding : [0. 0. 0. 0. 0. 1. 0. 0. 0.
0.]
```

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 instance
s 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_init
ializer='glorot_uniform',
# bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=N
one, activity_regularizer=None,
# kernel_constraint=None, bias_constraint=None)

# Dense implements the operation: output = activation(dot(input, kerne
l) + bias) where
# activation is the element-wise activation function passed as the acti
vation argument,
# kernel is a weights matrix created by the layer, and
# bias is a bias vector created by the layer (only applicable if use_bi
as 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 throug
h 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, soft
max
from keras.models import Sequential
from keras.layers import Dense,Activation
from keras.initializers import he_normal
```

In [13]:
```
X_train.shape[1]
```
Out[13]: 784

In [14]:
```
output_dim=10
input_dim=X_train.shape[1]
batch_size=128
nb_epoch=20
```

1.0 TWO HIDDEN LAYER MLP 492-160

1.1 MODEL 1+ADAM+RELU

In [27]:
```
model_relu=Sequential()

model_relu.add(Dense(492,activation="relu",input_shape=(input_dim,),ker
nel_initializer=he_normal(seed=None)))
model_relu.add(Dense(160,activation="relu",kernel_initializer=he_normal
(seed=None)))
model_relu.add(Dense(output_dim,activation='softmax'))

model_relu.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 492)               386220
_________________________________________________________________
dense_8 (Dense)              (None, 160)               78880
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1610
=================================================================
Total params: 466,710
Trainable params: 466,710
Non-trainable params: 0
_________________________________________________________________
```

In [28]:
```
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 [==============================] - 12s 198us/step - loss: 0.2274 - acc: 0.9328 - val_loss: 0.1013 - val_acc: 0.9694
Epoch 2/20
60000/60000 [==============================] - 11s 177us/step - loss: 0.0846 - acc: 0.9744 - val_loss: 0.1001 - val_acc: 0.9682
Epoch 3/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.0531 - acc: 0.9836 - val_loss: 0.0673 - val_acc: 0.9794
Epoch 4/20
60000/60000 [==============================] - 10s 166us/step - loss: 0.0370 - acc: 0.9885 - val_loss: 0.0783 - val_acc: 0.9747
Epoch 5/20
60000/60000 [==============================] - 10s 163us/step - loss: 0.0259 - acc: 0.9920 - val_loss: 0.0664 - val_acc: 0.9804
Epoch 6/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.0218 - acc: 0.9928 - val_loss: 0.0715 - val_acc: 0.9802
Epoch 7/20
60000/60000 [==============================] - 10s 164us/step - loss: 0.0182 - acc: 0.9938 - val_loss: 0.0867 - val_acc: 0.9784
Epoch 8/20
60000/60000 [==============================] - 10s 175us/step - loss: 0.0154 - acc: 0.9953 - val_loss: 0.0793 - val_acc: 0.9799
Epoch 9/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.0124 - acc: 0.9956 - val_loss: 0.0872 - val_acc: 0.9770
Epoch 10/20
60000/60000 [==============================] - 10s 167us/step - loss: 0.0136 - acc: 0.9955 - val_loss: 0.0869 - val_acc: 0.9797
Epoch 11/20
60000/60000 [==============================] - 10s 166us/step - loss: 0.0077 - acc: 0.9975 - val_loss: 0.0822 - val_acc: 0.9807
```

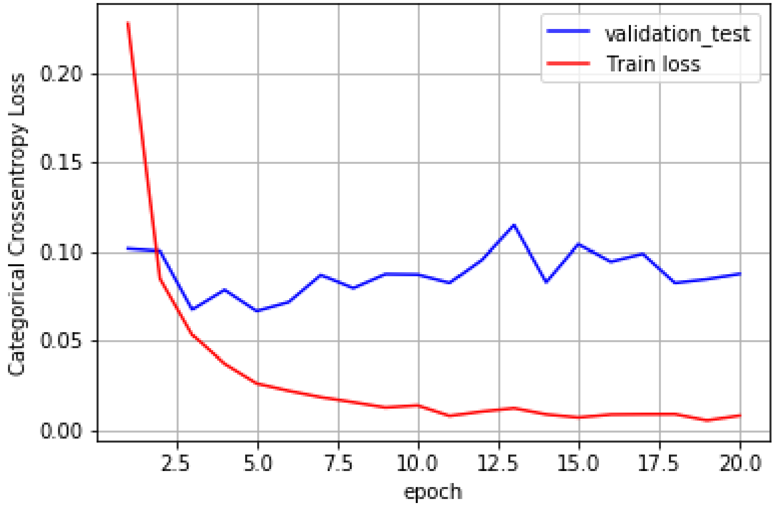
```
Epoch 12/20
60000/60000 [==============================] - 11s 183us/step - loss:
0.0102 - acc: 0.9963 - val_loss: 0.0952 - val_acc: 0.9783
Epoch 13/20
60000/60000 [==============================] - 10s 168us/step - loss:
0.0120 - acc: 0.9959 - val_loss: 0.1146 - val_acc: 0.9753
Epoch 14/20
60000/60000 [==============================] - 10s 173us/step - loss:
0.0086 - acc: 0.9969 - val_loss: 0.0826 - val_acc: 0.9812
Epoch 15/20
60000/60000 [==============================] - 10s 168us/step - loss:
0.0068 - acc: 0.9978 - val_loss: 0.1038 - val_acc: 0.9790
Epoch 16/20
60000/60000 [==============================] - 10s 165us/step - loss:
0.0085 - acc: 0.9972 - val_loss: 0.0941 - val_acc: 0.9801
Epoch 17/20
60000/60000 [==============================] - 10s 172us/step - loss:
0.0086 - acc: 0.9973 - val_loss: 0.0985 - val_acc: 0.9805
Epoch 18/20
60000/60000 [==============================] - 10s 166us/step - loss:
0.0087 - acc: 0.9971 - val_loss: 0.0822 - val_acc: 0.9820
Epoch 19/20
60000/60000 [==============================] - 11s 185us/step - loss:
0.0052 - acc: 0.9985 - val_loss: 0.0843 - val_acc: 0.9821: 0.998 - ETA:
0s - loss: 0.0050 - acc:
Epoch 20/20
60000/60000 [==============================] - 10s 169us/step - loss:
0.0079 - acc: 0.9974 - val_loss: 0.0873 - val_acc: 0.9813
```

In [32]:

```
score = model_relu.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
import matplotlib.pyplot as plt
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, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter vali
dation_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: 0.08728731730800837
Test accuracy: 0.9813
```



1.2 ADAM+RELU+BATCHNORMALIZATION

In [14]:
```
from keras.layers.normalization import BatchNormalization
model_batch=Sequential()

model_batch.add(Dense(492,activation="relu",input_shape=(input_dim,),ke
rnel_initializer=he_normal(seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(160,activation="relu",kernel_initializer=he_norma
l(seed=None)))
model_batch.add(BatchNormalization())

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

model_batch.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 492)               386220
_________________________________________________________________
batch_normalization_1 (Batch (None, 492)               1968
_________________________________________________________________
dense_2 (Dense)              (None, 160)               78880
_________________________________________________________________
batch_normalization_2 (Batch (None, 160)               640
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1610
=================================================================
Total params: 469,318
Trainable params: 468,014
Non-trainable params: 1,304
_________________________________________________________________
```

In [15]:
```
model_batch.compile(optimizer='adam',loss="categorical_crossentropy",me
trics=['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 [==============================] - 12s 205us/step - loss:
0.1866 - acc: 0.9436 - val_loss: 0.0930 - val_acc: 0.9712
Epoch 2/20
60000/60000 [==============================] - 12s 199us/step - loss:
0.0691 - acc: 0.9789 - val_loss: 0.0811 - val_acc: 0.9753
Epoch 3/20
60000/60000 [==============================] - 12s 195us/step - loss:
0.0463 - acc: 0.9853 - val_loss: 0.0794 - val_acc: 0.9742
Epoch 4/20
60000/60000 [==============================] - 12s 197us/step - loss:
0.0325 - acc: 0.9904 - val_loss: 0.0871 - val_acc: 0.9734
Epoch 5/20
60000/60000 [==============================] - 12s 199us/step - loss:
0.0240 - acc: 0.9925 - val_loss: 0.0781 - val_acc: 0.9758
Epoch 6/20
60000/60000 [==============================] - 12s 199us/step - loss:
0.0218 - acc: 0.9930 - val_loss: 0.0767 - val_acc: 0.9793
Epoch 7/20
60000/60000 [==============================] - 12s 204us/step - loss:
0.0176 - acc: 0.9945 - val_loss: 0.0871 - val_acc: 0.9772
Epoch 8/20
60000/60000 [==============================] - 12s 204us/step - loss:
0.0178 - acc: 0.9941 - val_loss: 0.0872 - val_acc: 0.9777
Epoch 9/20
60000/60000 [==============================] - 12s 204us/step - loss:
0.0190 - acc: 0.9937 - val_loss: 0.0704 - val_acc: 0.9801
Epoch 10/20
60000/60000 [==============================] - 12s 204us/step - loss:
0.0106 - acc: 0.9966 - val_loss: 0.0809 - val_acc: 0.9786
Epoch 11/20
60000/60000 [==============================] - 12s 204us/step - loss:
0.0108 - acc: 0.9965 - val_loss: 0.0801 - val_acc: 0.9779
Epoch 12/20
60000/60000 [==============================] - 12s 206us/step - loss:
0.0122 - acc: 0.9959 - val_loss: 0.0871 - val_acc: 0.9776
```

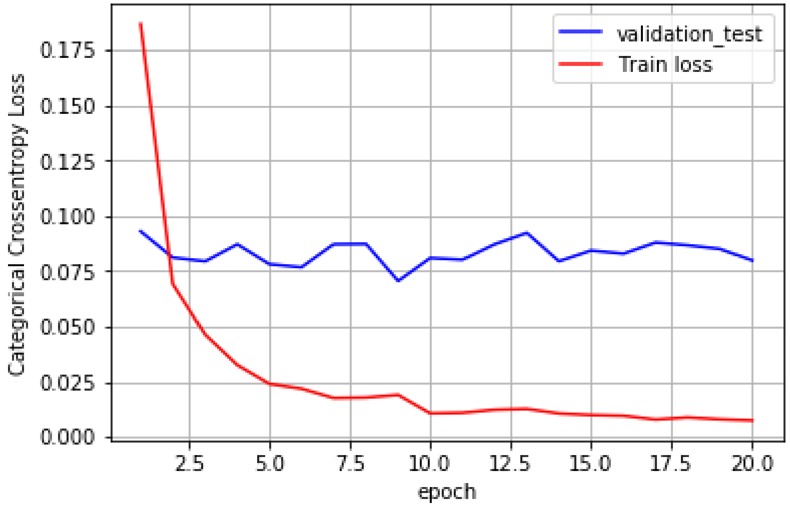
```
Epoch 13/20
60000/60000 [==============================] - 12s 206us/step - loss:

0.0126 - acc: 0.9955 - val_loss: 0.0923 - val_acc: 0.9768
Epoch 14/20
60000/60000 [==============================] - 12s 204us/step - loss:
0.0105 - acc: 0.9965 - val_loss: 0.0794 - val_acc: 0.9812
Epoch 15/20
60000/60000 [==============================] - 12s 204us/step - loss:
0.0098 - acc: 0.9966 - val_loss: 0.0842 - val_acc: 0.9796
Epoch 16/20
60000/60000 [==============================] - 12s 204us/step - loss:
0.0095 - acc: 0.9970 - val_loss: 0.0828 - val_acc: 0.9813
Epoch 17/20
60000/60000 [==============================] - 13s 210us/step - loss:
0.0078 - acc: 0.9975 - val_loss: 0.0879 - val_acc: 0.9796
Epoch 18/20
60000/60000 [==============================] - 12s 206us/step - loss:
0.0088 - acc: 0.9971 - val_loss: 0.0866 - val_acc: 0.9786
Epoch 19/20
60000/60000 [==============================] - 12s 204us/step - loss:
0.0079 - acc: 0.9973 - val_loss: 0.0850 - val_acc: 0.9815
Epoch 20/20
60000/60000 [==============================] - 12s 205us/step - loss:
0.0074 - acc: 0.9976 - val_loss: 0.0798 - val_acc: 0.9820
```

In [16]:

```
score = model_batch.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
import matplotlib.pyplot as plt
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, epo
```

```
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter vali
dation_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: 0.079772907704144475
Test accuracy: 0.982
```



1.3 ADAM+RELU+BATCHNORMALIZATION+DROPOU

In [17]:

```
from keras.layers import Dropout
model_drop=Sequential()

model_drop.add(Dense(492,activation="relu",input_shape=(input_dim,),ker
nel_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(output_dim,activation='softmax'))

model_drop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 492)               386220    
_________________________________________________________________
batch_normalization_3 (Batch (None, 492)               1968      
_________________________________________________________________
dropout_1 (Dropout)          (None, 492)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 160)               78880     
_________________________________________________________________
batch_normalization_4 (Batch (None, 160)               640       
_________________________________________________________________
dropout_2 (Dropout)          (None, 160)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1610      
=================================================================
Total params: 469,318
Trainable params: 468,014
Non-trainable params: 1,304
_________________________________________________________________
```

In [18]:

```
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 [==============================] - 14s 229us/step - loss: 0.4224 - acc: 0.8723 - val_loss: 0.1457 - val_acc: 0.9530
Epoch 2/20
60000/60000 [==============================] - 13s 210us/step - loss: 0.2056 - acc: 0.9388 - val_loss: 0.1057 - val_acc: 0.9666
Epoch 3/20
60000/60000 [==============================] - 13s 211us/step - loss: 0.1588 - acc: 0.9516 - val_loss: 0.0941 - val_acc: 0.9707
Epoch 4/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.1334 - acc: 0.9597 - val_loss: 0.0788 - val_acc: 0.9750
Epoch 5/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.1208 - acc: 0.9636 - val_loss: 0.0727 - val_acc: 0.9776
Epoch 6/20
60000/60000 [==============================] - 13s 214us/step - loss: 0.1082 - acc: 0.9670 - val_loss: 0.0721 - val_acc: 0.9775
Epoch 7/20
60000/60000 [==============================] - 13s 222us/step - loss: 0.0990 - acc: 0.9692 - val_loss: 0.0695 - val_acc: 0.9781
Epoch 8/20
60000/60000 [==============================] - 13s 214us/step - loss: 0.0919 - acc: 0.9719 - val_loss: 0.0634 - val_acc: 0.9791
Epoch 9/20
60000/60000 [==============================] - 13s 224us/step - loss: 0.0877 - acc: 0.9725 - val_loss: 0.0691 - val_acc: 0.9788
Epoch 10/20
60000/60000 [==============================] - 13s 214us/step - loss: 0.0807 - acc: 0.9748 - val_loss: 0.0643 - val_acc: 0.9810
Epoch 11/20
60000/60000 [==============================] - 13s 214us/step - loss: 0.0774 - acc: 0.9759 - val_loss: 0.0571 - val_acc: 0.9815
Epoch 12/20
```

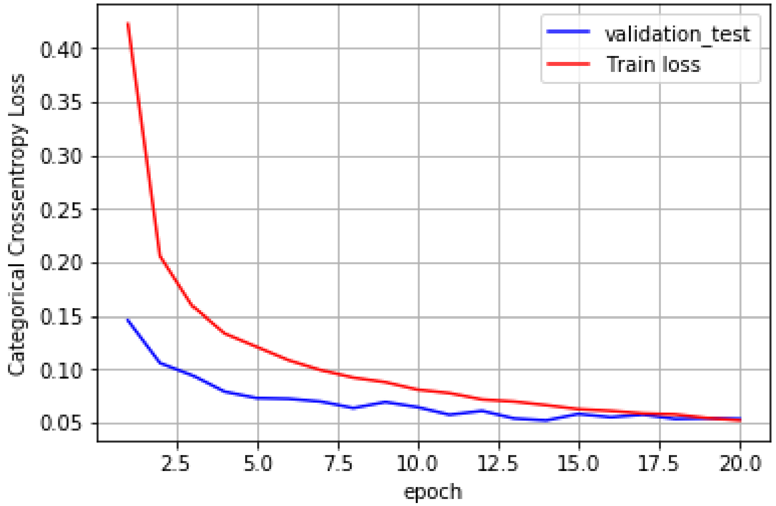
```
Epoch 12/20
60000/60000 [==============================] - 13s 214us/step - loss:
0.0714 - acc: 0.9768 - val_loss: 0.0608 - val_acc: 0.9814
Epoch 13/20
60000/60000 [==============================] - 13s 213us/step - loss:
0.0695 - acc: 0.9781 - val_loss: 0.0536 - val_acc: 0.9832
Epoch 14/20
60000/60000 [==============================] - 13s 214us/step - loss:
0.0661 - acc: 0.9776 - val_loss: 0.0518 - val_acc: 0.9841
Epoch 15/20
60000/60000 [==============================] - 13s 214us/step - loss:
0.0624 - acc: 0.9798 - val_loss: 0.0579 - val_acc: 0.9823
Epoch 16/20
60000/60000 [==============================] - 13s 214us/step - loss:
0.0608 - acc: 0.9804 - val_loss: 0.0548 - val_acc: 0.9841
Epoch 17/20
60000/60000 [==============================] - 13s 217us/step - loss:
0.0583 - acc: 0.9812 - val_loss: 0.0572 - val_acc: 0.9834
Epoch 18/20
60000/60000 [==============================] - 13s 221us/step - loss:
0.0574 - acc: 0.9819 - val_loss: 0.0533 - val_acc: 0.9828
Epoch 19/20
60000/60000 [==============================] - 13s 217us/step - loss:
0.0540 - acc: 0.9828 - val_loss: 0.0538 - val_acc: 0.9839
Epoch 20/20
60000/60000 [==============================] - 13s 215us/step - loss:
0.0519 - acc: 0.9830 - val_loss: 0.0532 - val_acc: 0.9829
```

In [19]:

```
score = model_drop.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
import matplotlib.pyplot as plt
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, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter vali
dation_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: 0.053227471913574846
Test accuracy: 0.9829
```



1.4 ADAM+RELU+DROPOUT

In [20]:
```
from keras.layers import Dropout
model_onlydrop=Sequential()

model_onlydrop.add(Dense(492,activation="relu",input_shape=(input_dim
,),kernel_initializer=he_normal(seed=None)))
model_onlydrop.add(Dropout(0.5))

model_onlydrop.add(Dense(160,activation="relu",kernel_initializer=he_no
rmal(seed=None)))
model_onlydrop.add(Dropout(0.5))

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

model_onlydrop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 492)               386220
_________________________________________________________________
dropout_3 (Dropout)          (None, 492)               0
_________________________________________________________________
dense_8 (Dense)              (None, 160)               78880
_________________________________________________________________
dropout_4 (Dropout)          (None, 160)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1610
=================================================================
Total params: 466,710
Trainable params: 466,710
Non-trainable params: 0
_________________________________________________________________
```

In [21]:
```
model_onlydrop.compile(optimizer='adam',loss="categorical_crossentropy"
,metrics=['accuracy'])
history=model_onlydrop.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 [==============================] - 11s 188us/step - loss:
0.4408 - acc: 0.8625 - val_loss: 0.1396 - val_acc: 0.9579
Epoch 2/20
60000/60000 [==============================] - 10s 174us/step - loss:
0.1992 - acc: 0.9420 - val_loss: 0.1050 - val_acc: 0.9677
Epoch 3/20
60000/60000 [==============================] - 11s 178us/step - loss:
0.1542 - acc: 0.9548 - val_loss: 0.0905 - val_acc: 0.9706
Epoch 4/20
60000/60000 [==============================] - 11s 179us/step - loss:
0.1304 - acc: 0.9615 - val_loss: 0.0787 - val_acc: 0.9757
Epoch 5/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.
1126 - acc: 0.9661 - val_loss: 0.0754 - val_acc: 0.9770
Epoch 6/20
60000/60000 [==============================] - 9s 158us/step - loss: 0.
1004 - acc: 0.9699 - val_loss: 0.0724 - val_acc: 0.9773
Epoch 7/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.
0911 - acc: 0.9722 - val_loss: 0.0683 - val_acc: 0.9783
Epoch 8/20
60000/60000 [==============================] - 10s 162us/step - loss:
0.0874 - acc: 0.9736 - val_loss: 0.0677 - val_acc: 0.9804
Epoch 9/20
60000/60000 [==============================] - 9s 158us/step - loss: 0.
0795 - acc: 0.9754 - val_loss: 0.0653 - val_acc: 0.9811
Epoch 10/20
60000/60000 [==============================] - 10s 158us/step - loss:
0.0741 - acc: 0.9775 - val_loss: 0.0596 - val_acc: 0.9812
Epoch 11/20
60000/60000 [==============================] - 11s 187us/step - loss:
0.0699 - acc: 0.9778 - val_loss: 0.0639 - val_acc: 0.9825
Epoch 12/20
60000/60000 [==============================] - 11s 182us/step - loss:
0.0669 - acc: 0.9791 - val_loss: 0.0665 - val_acc: 0.9812
Epoch 13/20
60000/60000 [==============================] - 11s 187us/step - loss:
```

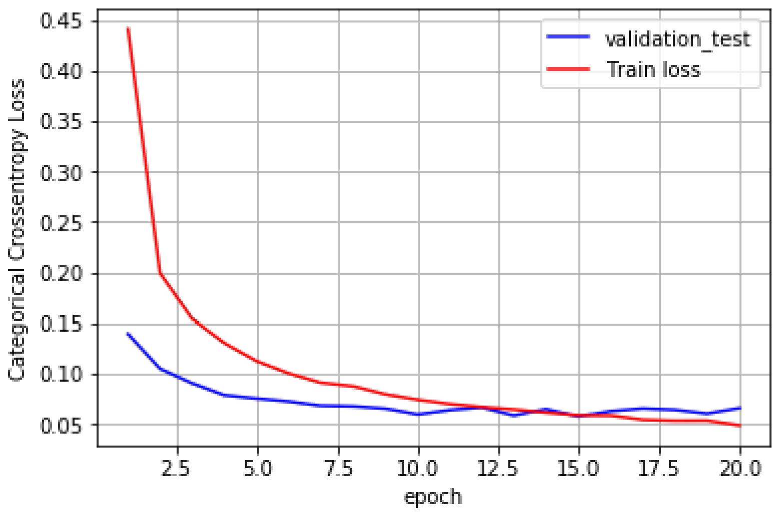
```
0.0645 - acc: 0.9798 - val_loss: 0.0585 - val_acc: 0.9830
Epoch 14/20
60000/60000 [==============================] - 10s 169us/step - loss:
0.0614 - acc: 0.9809 - val_loss: 0.0648 - val_acc: 0.9815
Epoch 15/20
60000/60000 [==============================] - 11s 175us/step - loss:
0.0588 - acc: 0.9812 - val_loss: 0.0577 - val_acc: 0.9844
Epoch 16/20
60000/60000 [==============================] - 10s 172us/step - loss:
0.0584 - acc: 0.9816 - val_loss: 0.0628 - val_acc: 0.9828
Epoch 17/20
60000/60000 [==============================] - 10s 173us/step - loss:
0.0543 - acc: 0.9829 - val_loss: 0.0655 - val_acc: 0.9815
Epoch 18/20
60000/60000 [==============================] - 10s 175us/step - loss:
0.0533 - acc: 0.9832 - val_loss: 0.0641 - val_acc: 0.9821
Epoch 19/20
60000/60000 [==============================] - 11s 176us/step - loss:
0.0534 - acc: 0.9838 - val_loss: 0.0604 - val_acc: 0.9838
Epoch 20/20
60000/60000 [==============================] - 11s 177us/step - loss:
0.0487 - acc: 0.9844 - val_loss: 0.0659 - val_acc: 0.9838
```

In [22]:

```
score = model_onlydrop.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
import matplotlib.pyplot as plt
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, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter vali
```

```
dation_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: 0.06586753905236692
Test accuracy: 0.9838
```



2.0 THREE layers input-412-298-89-output

2.1 ADAM+RELU

In [23]:
```
model_relu3=Sequential()

model_relu3.add(Dense(412,activation="relu",input_shape=(input_dim,),kernel_initializer=he_normal(seed=None)))

model_relu3.add(Dense(298,activation="relu",kernel_initializer=he_normal(seed=None)))

model_relu3.add(Dense(89,activation='relu',kernel_initializer=he_normal(seed=None)))

model_relu3.add(Dense(output_dim,activation='softmax'))

model_relu3.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_10 (Dense)             (None, 412)               323420    
_________________________________________________________________
dense_11 (Dense)             (None, 298)               123074    
_________________________________________________________________
dense_12 (Dense)             (None, 89)                26611     
_________________________________________________________________
dense_13 (Dense)             (None, 10)                900       
=================================================================
Total params: 474,005
Trainable params: 474,005
Non-trainable params: 0
_________________________________________________________________
```

In [24]:
```
model_relu3.compile(optimizer='adam',loss="categorical_crossentropy",metrics=['accuracy'])
history=model_relu3.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 [==============================] - 10s 166us/step - loss:
0.2252 - acc: 0.9334 - val_loss: 0.1095 - val_acc: 0.9660
```

```
Epoch 2/20
60000/60000 [==============================] - 10s 172us/step - loss:
0.0861 - acc: 0.9734 - val_loss: 0.0888 - val_acc: 0.9705
Epoch 3/20
60000/60000 [==============================] - 10s 161us/step - loss:
0.0557 - acc: 0.9825 - val_loss: 0.0793 - val_acc: 0.9756
Epoch 4/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.
0370 - acc: 0.9878 - val_loss: 0.0758 - val_acc: 0.9782
Epoch 5/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.
0323 - acc: 0.9899 - val_loss: 0.0665 - val_acc: 0.9799
Epoch 6/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.
0270 - acc: 0.9909 - val_loss: 0.0820 - val_acc: 0.9773
Epoch 7/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.
0211 - acc: 0.9930 - val_loss: 0.0750 - val_acc: 0.9810
Epoch 8/20
60000/60000 [==============================] - 10s 160us/step - loss:
0.0191 - acc: 0.9938 - val_loss: 0.0866 - val_acc: 0.9777
Epoch 9/20
60000/60000 [==============================] - 10s 160us/step - loss:
0.0148 - acc: 0.9950 - val_loss: 0.0919 - val_acc: 0.9784
Epoch 10/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.
0160 - acc: 0.9946 - val_loss: 0.0862 - val_acc: 0.9792
Epoch 11/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.
0160 - acc: 0.9947 - val_loss: 0.0863 - val_acc: 0.9807
Epoch 12/20
60000/60000 [==============================] - 10s 172us/step - loss:
0.0150 - acc: 0.9952 - val_loss: 0.0961 - val_acc: 0.9778
Epoch 13/20
60000/60000 [==============================] - 9s 158us/step - loss: 0.
0097 - acc: 0.9969 - val_loss: 0.0826 - val_acc: 0.9810
Epoch 14/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.
0106 - acc: 0.9966 - val_loss: 0.0975 - val_acc: 0.9802
Epoch 15/20
```

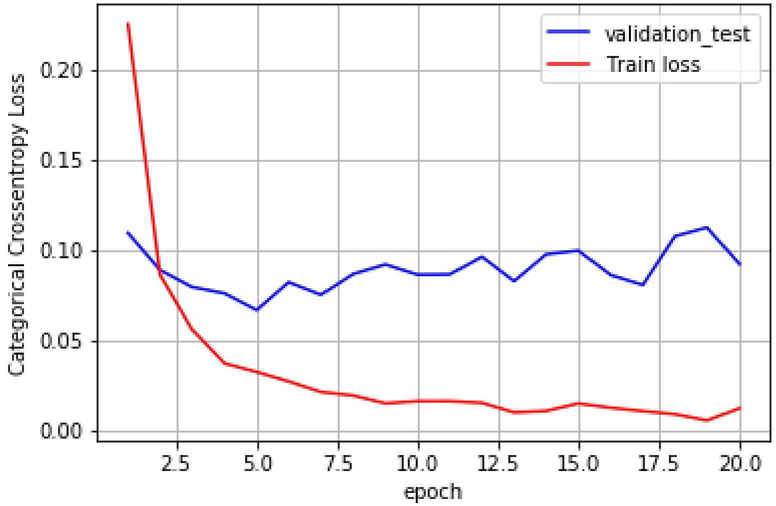
```
60000/60000 [==============================] - 9s 150us/step - loss: 0.
0147 - acc: 0.9953 - val_loss: 0.0996 - val_acc: 0.9759
Epoch 16/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.
0123 - acc: 0.9960 - val_loss: 0.0860 - val_acc: 0.9820
Epoch 17/20
60000/60000 [==============================] - 11s 178us/step - loss:
0.0105 - acc: 0.9970 - val_loss: 0.0805 - val_acc: 0.9828
Epoch 18/20
60000/60000 [==============================] - 10s 160us/step - loss:
0.0087 - acc: 0.9972 - val_loss: 0.1076 - val_acc: 0.9784
Epoch 19/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.
0053 - acc: 0.9983 - val_loss: 0.1125 - val_acc: 0.9801
Epoch 20/20
60000/60000 [==============================] - 10s 165us/step - loss:
0.0120 - acc: 0.9963 - val_loss: 0.0921 - val_acc: 0.9807
```

In [25]:

```
score = model_relu3.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
import matplotlib.pyplot as plt
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, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter vali
dation_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: 0.09208794380759855
Test accuracy: 0.9807
```



2.2 ADAM+RELU+BATCHNORMALIZATION

In [26]:

```
model_batch3=Sequential()

model_batch3.add(Dense(412,activation="relu",input_shape=(input_dim,),k
ernel_initializer=he_normal(seed=None)))
model_batch3.add(BatchNormalization())

model_batch3.add(Dense(298,activation="relu",kernel_initializer=he_norm
```

```python
al(seed=None)))
model_batch3.add(BatchNormalization())


model_batch3.add(Dense(89,activation='relu',kernel_initializer=he_norma
l(seed=None)))
model_batch3.add(BatchNormalization())

model_batch3.add(Dense(output_dim,activation='softmax'))

model_batch3.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_14 (Dense)             (None, 412)               323420    
_________________________________________________________________
batch_normalization_5 (Batch (None, 412)               1648      
_________________________________________________________________
dense_15 (Dense)             (None, 298)               123074    
_________________________________________________________________
batch_normalization_6 (Batch (None, 298)               1192      
_________________________________________________________________
dense_16 (Dense)             (None, 89)                26611     
_________________________________________________________________
batch_normalization_7 (Batch (None, 89)                356       
_________________________________________________________________
dense_17 (Dense)             (None, 10)                900       
=================================================================
Total params: 477,201
Trainable params: 475,603
Non-trainable params: 1,598
_________________________________________________________________
```

In [27]:

```python
model_batch3.compile(optimizer='adam',loss="categorical_crossentropy",m
etrics=['accuracy'])
history=model_batch3.fit(X_train,y_train,batch_size=batch_size,epochs=n
b_epoch,verbose=1,validation_data=(X_test,y_test))
```

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

```
Epoch 1/20
60000/60000 [==============================] - 12s 205us/step - loss:
0.1971 - acc: 0.9413 - val_loss: 0.0990 - val_acc: 0.9710
Epoch 2/20
60000/60000 [==============================] - 11s 182us/step - loss:
0.0741 - acc: 0.9777 - val_loss: 0.0865 - val_acc: 0.9729
Epoch 3/20
60000/60000 [==============================] - 10s 173us/step - loss:
0.0493 - acc: 0.9846 - val_loss: 0.0879 - val_acc: 0.9735
Epoch 4/20
60000/60000 [==============================] - 10s 174us/step - loss:
0.0378 - acc: 0.9876 - val_loss: 0.1050 - val_acc: 0.9684
Epoch 5/20
60000/60000 [==============================] - 10s 172us/step - loss:
0.0281 - acc: 0.9912 - val_loss: 0.0894 - val_acc: 0.9756
Epoch 6/20
60000/60000 [==============================] - 11s 191us/step - loss:
0.0235 - acc: 0.9918 - val_loss: 0.0664 - val_acc: 0.9805
Epoch 7/20
60000/60000 [==============================] - 11s 183us/step - loss:
0.0229 - acc: 0.9925 - val_loss: 0.0838 - val_acc: 0.9765
Epoch 8/20
60000/60000 [==============================] - 11s 175us/step - loss:
0.0203 - acc: 0.9932 - val_loss: 0.0837 - val_acc: 0.9748
Epoch 9/20
60000/60000 [==============================] - 10s 174us/step - loss:
0.0171 - acc: 0.9943 - val_loss: 0.0721 - val_acc: 0.9799
Epoch 10/20
60000/60000 [==============================] - 11s 177us/step - loss:
0.0166 - acc: 0.9945 - val_loss: 0.0772 - val_acc: 0.9784
Epoch 11/20
60000/60000 [==============================] - 11s 177us/step - loss:
0.0127 - acc: 0.9958 - val_loss: 0.0734 - val_acc: 0.9801
Epoch 12/20
60000/60000 [==============================] - 11s 176us/step - loss:
0.0170 - acc: 0.9942 - val_loss: 0.0764 - val_acc: 0.9805
Epoch 13/20
60000/60000 [==============================] - 11s 191us/step - loss:
0.0144 - acc: 0.9951 - val_loss: 0.0658 - val_acc: 0.9825
Epoch 14/20
```

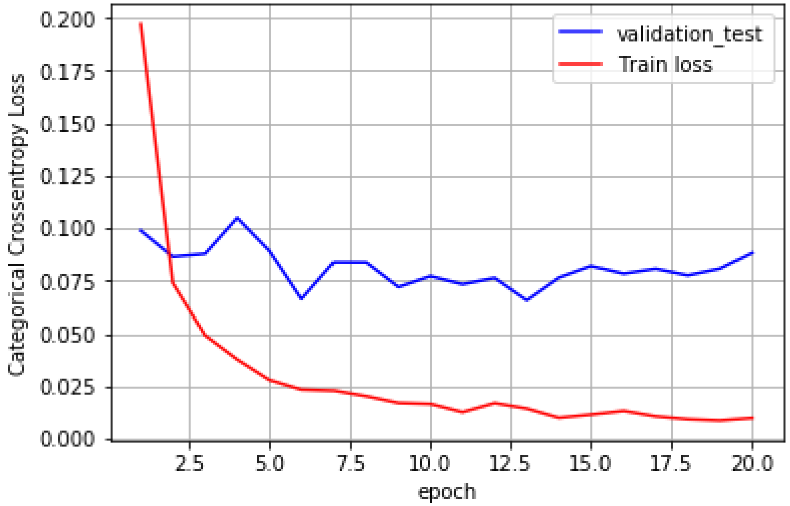
```
60000/60000 [==============================] - 11s 177us/step - loss: 0.0100 - acc: 0.9966 - val_loss: 0.0765 - val_acc: 0.9803
Epoch 15/20
60000/60000 [==============================] - 11s 185us/step - loss: 0.0115 - acc: 0.9960 - val_loss: 0.0820 - val_acc: 0.9793
Epoch 16/20
60000/60000 [==============================] - 12s 207us/step - loss: 0.0133 - acc: 0.9955 - val_loss: 0.0784 - val_acc: 0.9796
Epoch 17/20
60000/60000 [==============================] - 12s 204us/step - loss: 0.0107 - acc: 0.9965 - val_loss: 0.0807 - val_acc: 0.9816
Epoch 18/20
60000/60000 [==============================] - 11s 179us/step - loss: 0.0093 - acc: 0.9969 - val_loss: 0.0776 - val_acc: 0.9811
Epoch 19/20
60000/60000 [==============================] - 11s 179us/step - loss: 0.0087 - acc: 0.9971 - val_loss: 0.0807 - val_acc: 0.9818
Epoch 20/20
60000/60000 [==============================] - 12s 203us/step - loss: 0.0099 - acc: 0.9966 - val_loss: 0.0883 - val_acc: 0.9788
```

In [28]:

```
score = model_batch3.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
import matplotlib.pyplot as plt
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: 0.08829731181473471
Test accuracy: 0.9788
```



2.3 ADAM+RELU+DROPOUT

In [48]:

```
model_dropout=Sequential()

model_dropout.add(Dense(412,activation="relu",input_shape=(input_dim,),
kernel_initializer=he_normal(seed=None)))
```

```
model_dropout.add(Dropout(0.5))

model_dropout.add(Dense(298,activation="relu",kernel_initializer=he_nor
mal(seed=None)))
model_dropout.add(Dropout(0.5))

model_dropout.add(Dense(89,activation="relu",kernel_initializer=he_norm
al(seed=None)))
model_dropout.add(Dropout(0.5))

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

model_dropout.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_22 (Dense)             (None, 412)               323420    
_________________________________________________________________
dropout_8 (Dropout)          (None, 412)               0         
_________________________________________________________________
dense_23 (Dense)             (None, 298)               123074    
_________________________________________________________________
dropout_9 (Dropout)          (None, 298)               0         
_________________________________________________________________
dense_24 (Dense)             (None, 89)                26611     
_________________________________________________________________
dropout_10 (Dropout)         (None, 89)                0         
_________________________________________________________________
dense_25 (Dense)             (None, 10)                900       
=================================================================
Total params: 474,005
Trainable params: 474,005
Non-trainable params: 0
_________________________________________________________________
```

In [50]:
```
model_dropout.compile(optimizer='adam',loss="categorical_crossentropy",
```

```
metrics=['accuracy'])
history=model_dropout.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 [==============================] - 12s 201us/step - loss:
0.6521 - acc: 0.7956 - val_loss: 0.1816 - val_acc: 0.9486
Epoch 2/20
60000/60000 [==============================] - 11s 179us/step - loss:
0.2733 - acc: 0.9267 - val_loss: 0.1277 - val_acc: 0.9631
Epoch 3/20
60000/60000 [==============================] - 10s 169us/step - loss:
0.2112 - acc: 0.9435 - val_loss: 0.1168 - val_acc: 0.9647
Epoch 4/20
60000/60000 [==============================] - 10s 169us/step - loss:
0.1804 - acc: 0.9520 - val_loss: 0.1064 - val_acc: 0.9713
Epoch 5/20
60000/60000 [==============================] - 10s 168us/step - loss:
0.1596 - acc: 0.9565 - val_loss: 0.0910 - val_acc: 0.9750
Epoch 6/20
60000/60000 [==============================] - 10s 167us/step - loss:
0.1414 - acc: 0.9622 - val_loss: 0.0837 - val_acc: 0.9770
Epoch 7/20
60000/60000 [==============================] - 10s 168us/step - loss:
0.1314 - acc: 0.9647 - val_loss: 0.0836 - val_acc: 0.9766
Epoch 8/20
60000/60000 [==============================] - 10s 169us/step - loss:
0.1254 - acc: 0.9663 - val_loss: 0.0836 - val_acc: 0.9772
Epoch 9/20
60000/60000 [==============================] - 10s 171us/step - loss:
0.1109 - acc: 0.9689 - val_loss: 0.0712 - val_acc: 0.9801
Epoch 10/20
60000/60000 [==============================] - 10s 171us/step - loss:
0.1074 - acc: 0.9703 - val_loss: 0.0814 - val_acc: 0.9783
Epoch 11/20
60000/60000 [==============================] - 11s 181us/step - loss:
0.0994 - acc: 0.9719 - val_loss: 0.0791 - val_acc: 0.9783
Epoch 12/20
60000/60000 [==============================] - 10s 172us/step - loss:
```

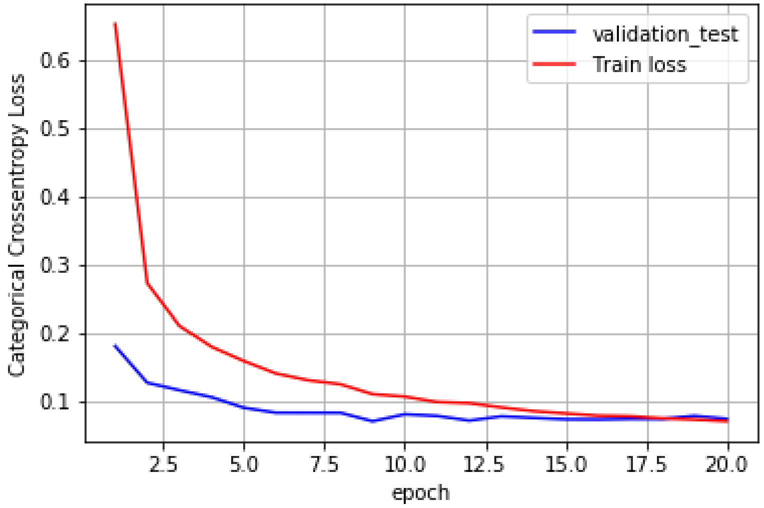
```
0.0978 - acc: 0.9725 - val_loss: 0.0722 - val_acc: 0.9811
Epoch 13/20
60000/60000 [==============================] - 10s 166us/step - loss:
0.0915 - acc: 0.9751 - val_loss: 0.0784 - val_acc: 0.9796
Epoch 14/20
60000/60000 [==============================] - 10s 167us/step - loss:
0.0860 - acc: 0.9754 - val_loss: 0.0761 - val_acc: 0.9807
Epoch 15/20
60000/60000 [==============================] - 10s 168us/step - loss:
0.0827 - acc: 0.9760 - val_loss: 0.0741 - val_acc: 0.9824
Epoch 16/20
60000/60000 [==============================] - 10s 167us/step - loss:
0.0793 - acc: 0.9775 - val_loss: 0.0738 - val_acc: 0.9820
Epoch 17/20
60000/60000 [==============================] - 10s 166us/step - loss:
0.0785 - acc: 0.9774 - val_loss: 0.0746 - val_acc: 0.9823
Epoch 18/20
60000/60000 [==============================] - 10s 167us/step - loss:
0.0755 - acc: 0.9782 - val_loss: 0.0746 - val_acc: 0.9818
Epoch 19/20
60000/60000 [==============================] - 10s 173us/step - loss:
0.0737 - acc: 0.9801 - val_loss: 0.0791 - val_acc: 0.9815
Epoch 20/20
60000/60000 [==============================] - 12s 193us/step - loss:
0.0713 - acc: 0.9792 - val_loss: 0.0747 - val_acc: 0.9816
```

In [51]:

```
score = model_dropout.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
import matplotlib.pyplot as plt
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, epo
```

```
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter vali
dation_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: 0.07473588717993752
Test accuracy: 0.9816
```



2.4 ADAM+RELU+BN+DROPOUT

```
In [16]: from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
model_final=Sequential()

model_final.add(Dense(412,activation="relu",input_shape=(input_dim,),ke
rnel_initializer=he_normal(seed=None)))
model_final.add(BatchNormalization())
model_final.add(Dropout(0.5))

model_final.add(Dense(298,activation="relu",kernel_initializer=he_norma
l(seed=None)))
model_final.add(BatchNormalization())
model_final.add(Dropout(0.5))

model_final.add(Dense(89,activation="relu",kernel_initializer=he_normal
(seed=None)))
model_final.add(BatchNormalization())
model_final.add(Dropout(0.5))

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

model_final.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_2 (Dense)              (None, 412)               323420    
_________________________________________________________________
batch_normalization_1 (Batch (None, 412)               1648      
_________________________________________________________________
dropout_1 (Dropout)          (None, 412)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 298)               123074    
_________________________________________________________________
batch_normalization_2 (Batch (None, 298)               1192      
_________________________________________________________________
dropout_2 (Dropout)          (None, 298)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 89)                26611     
_________________________________________________________________
```

```
batch_normalization_3 (Batch (None, 89)              356
_________________________________________________________________
dropout_3 (Dropout)          (None, 89)              0
_________________________________________________________________
dense_5 (Dense)              (None, 10)              900
=================================================================
Total params: 477,201
Trainable params: 475,603
Non-trainable params: 1,598
_________________________________________________________________
```

In [17]:

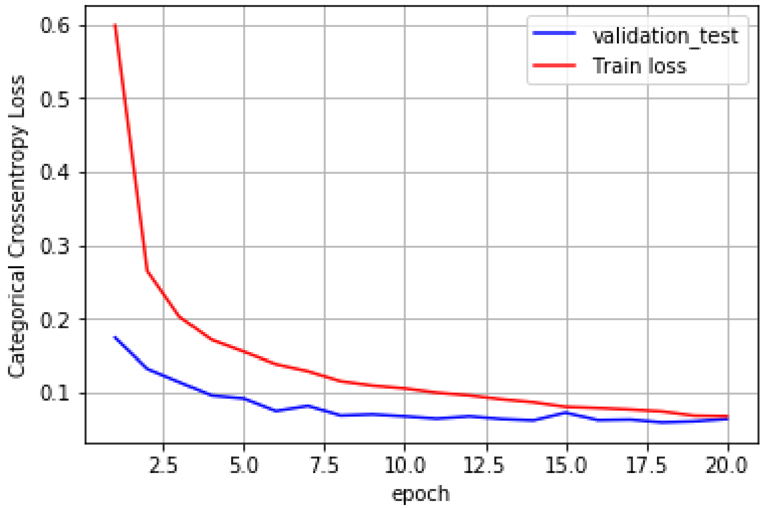
```
model_final.compile(optimizer='adam',loss="categorical_crossentropy",metrics=['accuracy'])
history=model_final.fit(X_train,y_train,batch_size=batch_size,epochs=nb_epoch,verbose=1,validation_data=(X_test,y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 15s 255us/step - loss: 0.5984 - acc: 0.8176 - val_loss: 0.1747 - val_acc: 0.9461
Epoch 2/20
60000/60000 [==============================] - 13s 216us/step - loss: 0.2657 - acc: 0.9229 - val_loss: 0.1326 - val_acc: 0.9593
Epoch 3/20
60000/60000 [==============================] - 13s 218us/step - loss: 0.2031 - acc: 0.9411 - val_loss: 0.1144 - val_acc: 0.9655
Epoch 4/20
60000/60000 [==============================] - 13s 219us/step - loss: 0.1719 - acc: 0.9512 - val_loss: 0.0964 - val_acc: 0.9702
Epoch 5/20
60000/60000 [==============================] - 13s 223us/step - loss: 0.1557 - acc: 0.9547 - val_loss: 0.0922 - val_acc: 0.9720
Epoch 6/20
60000/60000 [==============================] - 14s 230us/step - loss: 0.1389 - acc: 0.9597 - val_loss: 0.0752 - val_acc: 0.9783
Epoch 7/20
60000/60000 [==============================] - 14s 233us/step - loss: 0.1293 - acc: 0.9620 - val_loss: 0.0823 - val_acc: 0.9754
Epoch 8/20
```

```
60000/60000 [==============================] - 14s 227us/step - loss:
0.1158 - acc: 0.9657 - val_loss: 0.0692 - val_acc: 0.9802
Epoch 9/20
60000/60000 [==============================] - 14s 226us/step - loss:
0.1099 - acc: 0.9676 - val_loss: 0.0706 - val_acc: 0.9788
Epoch 10/20
60000/60000 [==============================] - 13s 223us/step - loss:
0.1061 - acc: 0.9691 - val_loss: 0.0680 - val_acc: 0.9795
Epoch 11/20
60000/60000 [==============================] - 12s 195us/step - loss:
0.1002 - acc: 0.9700 - val_loss: 0.0648 - val_acc: 0.9813
Epoch 12/20
60000/60000 [==============================] - 11s 188us/step - loss:
0.0965 - acc: 0.9714 - val_loss: 0.0680 - val_acc: 0.9805
Epoch 13/20
60000/60000 [==============================] - 12s 197us/step - loss:
0.0913 - acc: 0.9731 - val_loss: 0.0644 - val_acc: 0.9819
Epoch 14/20
60000/60000 [==============================] - 12s 194us/step - loss:
0.0872 - acc: 0.9742 - val_loss: 0.0625 - val_acc: 0.9824
Epoch 15/20
60000/60000 [==============================] - 13s 214us/step - loss:
0.0810 - acc: 0.9759 - val_loss: 0.0730 - val_acc: 0.9792
Epoch 16/20
60000/60000 [==============================] - 13s 216us/step - loss:
0.0793 - acc: 0.9761 - val_loss: 0.0628 - val_acc: 0.9825
Epoch 17/20
60000/60000 [==============================] - 13s 218us/step - loss:
0.0773 - acc: 0.9767 - val_loss: 0.0636 - val_acc: 0.9817
Epoch 18/20
60000/60000 [==============================] - 12s 196us/step - loss:
0.0746 - acc: 0.9775 - val_loss: 0.0597 - val_acc: 0.9823
Epoch 19/20
60000/60000 [==============================] - 11s 188us/step - loss:
0.0689 - acc: 0.9794 - val_loss: 0.0612 - val_acc: 0.9825
Epoch 20/20
60000/60000 [==============================] - 11s 187us/step - loss:
0.0682 - acc: 0.9800 - val_loss: 0.0643 - val_acc: 0.9828
```

In [18]:
```
score = model_final.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
import matplotlib.pyplot as plt
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, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter vali
dation_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: 0.0643334603364463
Test accuracy: 0.9828
```



3.0 FIVE layers input-415-286-145-78-52-output

3.1 ADAM+RELU

In [57]:
```
model_relu5=Sequential()

model_relu5.add(Dense(415,activation="relu",input_shape=(input_dim,),kernel_initializer=he_normal(seed=None)))

model_relu5.add(Dense(286,activation="relu",kernel_initializer=he_normal(seed=None)))

model_relu5.add(Dense(145,activation="relu",kernel_initializer=he_normal(seed=None)))
```

```
model_relu5.add(Dense(78,activation="relu",kernel_initializer=he_normal
(seed=None)))

model_relu5.add(Dense(52,activation="relu",kernel_initializer=he_normal
(seed=None)))


model_relu5.add(Dense(output_dim,activation='softmax'))

model_relu5.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_34 (Dense)             (None, 415)               325775
_________________________________________________________________
dense_35 (Dense)             (None, 286)               118976
_________________________________________________________________
dense_36 (Dense)             (None, 145)               41615
_________________________________________________________________
dense_37 (Dense)             (None, 78)                11388
_________________________________________________________________
dense_38 (Dense)             (None, 52)                4108
_________________________________________________________________
dense_39 (Dense)             (None, 10)                530
=================================================================
Total params: 502,392
Trainable params: 502,392
Non-trainable params: 0
_________________________________________________________________
```

In [58]:

```
model_relu5.compile(optimizer='adam',loss="categorical_crossentropy",me
trics=['accuracy'])
history=model_relu5.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 [==============================] - 11s 186us/step - loss:
0.2507 - acc: 0.9246 - val_loss: 0.1168 - val_acc: 0.9646
```

```
Epoch 2/20
60000/60000 [==============================] - 11s 183us/step - loss:
0.0935 - acc: 0.9716 - val_loss: 0.0878 - val_acc: 0.9729
Epoch 3/20
60000/60000 [==============================] - 11s 178us/step - loss:
0.0634 - acc: 0.9804 - val_loss: 0.0921 - val_acc: 0.9724
Epoch 4/20
60000/60000 [==============================] - 11s 180us/step - loss:
0.0481 - acc: 0.9849 - val_loss: 0.0809 - val_acc: 0.9750
Epoch 5/20
60000/60000 [==============================] - 10s 167us/step - loss:
0.0365 - acc: 0.9881 - val_loss: 0.0735 - val_acc: 0.9774
Epoch 6/20
60000/60000 [==============================] - 10s 165us/step - loss:
0.0312 - acc: 0.9897 - val_loss: 0.0937 - val_acc: 0.9752
Epoch 7/20
60000/60000 [==============================] - 11s 176us/step - loss:
0.0282 - acc: 0.9908 - val_loss: 0.0693 - val_acc: 0.9810
Epoch 8/20
60000/60000 [==============================] - 11s 178us/step - loss:
0.0249 - acc: 0.9918 - val_loss: 0.0779 - val_acc: 0.9793
Epoch 9/20
60000/60000 [==============================] - 10s 162us/step - loss:
0.0202 - acc: 0.9937 - val_loss: 0.0988 - val_acc: 0.9746
Epoch 10/20
60000/60000 [==============================] - 10s 161us/step - loss:
0.0195 - acc: 0.9937 - val_loss: 0.0823 - val_acc: 0.9801
Epoch 11/20
60000/60000 [==============================] - 9s 158us/step - loss: 0.
0171 - acc: 0.9942 - val_loss: 0.0896 - val_acc: 0.9786
Epoch 12/20
60000/60000 [==============================] - 10s 159us/step - loss:
0.0165 - acc: 0.9948 - val_loss: 0.0810 - val_acc: 0.9798
Epoch 13/20
60000/60000 [==============================] - 10s 160us/step - loss:
0.0147 - acc: 0.9957 - val_loss: 0.0835 - val_acc: 0.9808
Epoch 14/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.
0156 - acc: 0.9951 - val_loss: 0.0786 - val_acc: 0.9796
Epoch 15/20
```

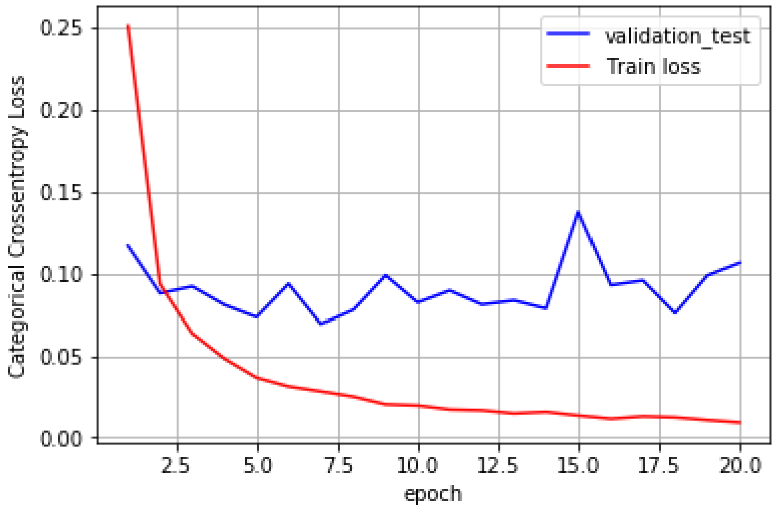
```
60000/60000 [==============================] - 10s 160us/step - loss: 0.0134 - acc: 0.9957 - val_loss: 0.1375 - val_acc: 0.9710
Epoch 16/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.0114 - acc: 0.9965 - val_loss: 0.0928 - val_acc: 0.9806
Epoch 17/20
60000/60000 [==============================] - 9s 158us/step - loss: 0.0129 - acc: 0.9960 - val_loss: 0.0956 - val_acc: 0.9809
Epoch 18/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.0123 - acc: 0.9963 - val_loss: 0.0757 - val_acc: 0.9823
Epoch 19/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.0106 - acc: 0.9970 - val_loss: 0.0986 - val_acc: 0.9763
Epoch 20/20
60000/60000 [==============================] - 10s 163us/step - loss: 0.0092 - acc: 0.9971 - val_loss: 0.1063 - val_acc: 0.9787
```

In [59]:

```
score = model_relu5.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
import matplotlib.pyplot as plt
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: 0.10631057699779321
Test accuracy: 0.9787
```



3.2 ADAM+RELU+BATCHNORMALIZATION

In [60]:
```
model_batch5=Sequential()

model_batch5.add(Dense(415,activation="relu",input_shape=(input_dim,),k
ernel_initializer=he_normal(seed=None)))
model_batch5.add(BatchNormalization())

model_batch5.add(Dense(286,activation="relu",kernel_initializer=he_norm
```

```
al(seed=None)))
model_batch5.add(BatchNormalization())

model_batch5.add(Dense(145,activation="relu",kernel_initializer=he_norm
al(seed=None)))
model_batch5.add(BatchNormalization())

model_batch5.add(Dense(78,activation="relu",kernel_initializer=he_norma
l(seed=None)))
model_batch5.add(BatchNormalization())

model_batch5.add(Dense(52,activation="relu",kernel_initializer=he_norma
l(seed=None)))
model_batch5.add(BatchNormalization())

model_batch5.add(Dense(output_dim,activation='softmax'))

model_batch5.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_40 (Dense)             (None, 415)               325775    
_________________________________________________________________
batch_normalization_14 (Batc (None, 415)               1660      
_________________________________________________________________
dense_41 (Dense)             (None, 286)               118976    
_________________________________________________________________
batch_normalization_15 (Batc (None, 286)               1144      
_________________________________________________________________
dense_42 (Dense)             (None, 145)               41615     
_________________________________________________________________
batch_normalization_16 (Batc (None, 145)               580       
_________________________________________________________________
dense_43 (Dense)             (None, 78)                11388     
_________________________________________________________________
batch_normalization_17 (Batc (None, 78)                312       
_________________________________________________________________
dense_44 (Dense)             (None, 52)                4108      
_________________________________________________________________
batch_normalization_18 (Batc (None, 52)                208       
```

```
batch_normalization_18 (Batc (None, 52)              208
_________________________________________________________________
dense_45 (Dense)             (None, 10)              530
=================================================================
Total params: 506,296
Trainable params: 504,344
Non-trainable params: 1,952
_________________________________________________________________
```

In [61]:

```
model_batch5.compile(optimizer='adam',loss="categorical_crossentropy",m
etrics=['accuracy'])
history=model_batch5.fit(X_train,y_train,batch_size=batch_size,epochs=n
b_epoch,verbose=1,validation_data=(X_test,y_test))
```

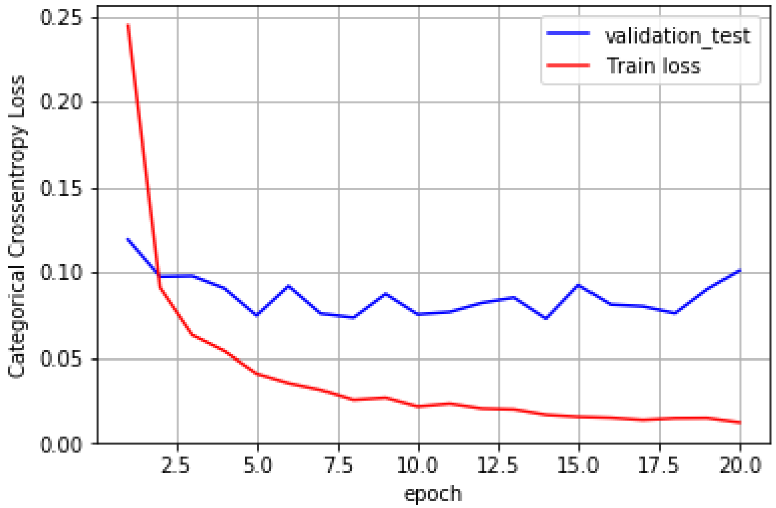
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 15s 243us/step - loss:
0.2445 - acc: 0.9281 - val_loss: 0.1192 - val_acc: 0.9644
Epoch 2/20
60000/60000 [==============================] - 12s 202us/step - loss:
0.0911 - acc: 0.9721 - val_loss: 0.0974 - val_acc: 0.9690
Epoch 3/20
60000/60000 [==============================] - 12s 198us/step - loss:
0.0632 - acc: 0.9802 - val_loss: 0.0978 - val_acc: 0.9692
Epoch 4/20
60000/60000 [==============================] - 12s 196us/step - loss:
0.0538 - acc: 0.9829 - val_loss: 0.0905 - val_acc: 0.9741
Epoch 5/20
60000/60000 [==============================] - 12s 197us/step - loss:
0.0407 - acc: 0.9869 - val_loss: 0.0746 - val_acc: 0.9781
Epoch 6/20
60000/60000 [==============================] - 12s 203us/step - loss:
0.0351 - acc: 0.9888 - val_loss: 0.0919 - val_acc: 0.9729
Epoch 7/20
60000/60000 [==============================] - 12s 197us/step - loss:
0.0312 - acc: 0.9902 - val_loss: 0.0758 - val_acc: 0.9783
Epoch 8/20
60000/60000 [==============================] - 12s 196us/step - loss:
0.0254 - acc: 0.9916 - val_loss: 0.0733 - val_acc: 0.9784
```

```
Epoch 9/20
60000/60000 [==============================] - 12s 196us/step - loss:
0.0266 - acc: 0.9916 - val_loss: 0.0873 - val_acc: 0.9762
Epoch 10/20
60000/60000 [==============================] - 12s 197us/step - loss:
0.0215 - acc: 0.9930 - val_loss: 0.0752 - val_acc: 0.9790
Epoch 11/20
60000/60000 [==============================] - 12s 199us/step - loss:
0.0231 - acc: 0.9926 - val_loss: 0.0767 - val_acc: 0.9780
Epoch 12/20
60000/60000 [==============================] - 12s 203us/step - loss:
0.0202 - acc: 0.9931 - val_loss: 0.0820 - val_acc: 0.9786
Epoch 13/20
60000/60000 [==============================] - 13s 211us/step - loss:
0.0197 - acc: 0.9937 - val_loss: 0.0851 - val_acc: 0.9775
Epoch 14/20
60000/60000 [==============================] - 12s 204us/step - loss:
0.0166 - acc: 0.9948 - val_loss: 0.0727 - val_acc: 0.9801
Epoch 15/20
60000/60000 [==============================] - 12s 204us/step - loss:
0.0154 - acc: 0.9947 - val_loss: 0.0925 - val_acc: 0.9764
Epoch 16/20
60000/60000 [==============================] - 12s 199us/step - loss:
0.0148 - acc: 0.9952 - val_loss: 0.0811 - val_acc: 0.9793
Epoch 17/20
60000/60000 [==============================] - 12s 199us/step - loss:
0.0136 - acc: 0.9956 - val_loss: 0.0800 - val_acc: 0.9801
Epoch 18/20
60000/60000 [==============================] - 12s 200us/step - loss:
0.0146 - acc: 0.9951 - val_loss: 0.0759 - val_acc: 0.9811
Epoch 19/20
60000/60000 [==============================] - 12s 201us/step - loss:
0.0147 - acc: 0.9952 - val_loss: 0.0900 - val_acc: 0.9786
Epoch 20/20
60000/60000 [==============================] - 12s 198us/step - loss:
0.0121 - acc: 0.9960 - val_loss: 0.1010 - val_acc: 0.9764
```

In [62]:
```
score = model_batch5.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
```

```
print('Test accuracy:', score[1])
import matplotlib.pyplot as plt
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, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter vali
dation_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: 0.10098484183535911
Test accuracy: 0.9764
```



3.3 ADAM+RELU+DROPOUT

In [63]:
```
model_drop5=Sequential()

model_drop5.add(Dense(415,activation="relu",input_shape=(input_dim,),kernel_initializer=he_normal(seed=None)))
model_drop5.add(Dropout(0.5))

model_drop5.add(Dense(286,activation="relu",kernel_initializer=he_normal(seed=None)))
model_drop5.add(Dropout(0.5))

model_drop5.add(Dense(145,activation="relu",kernel_initializer=he_normal(seed=None)))
model_drop5.add(Dropout(0.5))

model_drop5.add(Dense(78,activation="relu",kernel_initializer=he_normal(seed=None)))
```

```
model_drop5.add(Dropout(0.5))

model_drop5.add(Dense(52,activation="relu",kernel_initializer=he_normal
(seed=None)))
model_drop5.add(Dropout(0.5))


model_drop5.add(Dense(output_dim,activation='softmax'))

model_drop5.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_46 (Dense)             (None, 415)               325775    
_________________________________________________________________
dropout_17 (Dropout)         (None, 415)               0         
_________________________________________________________________
dense_47 (Dense)             (None, 286)               118976    
_________________________________________________________________
dropout_18 (Dropout)         (None, 286)               0         
_________________________________________________________________
dense_48 (Dense)             (None, 145)               41615     
_________________________________________________________________
dropout_19 (Dropout)         (None, 145)               0         
_________________________________________________________________
dense_49 (Dense)             (None, 78)                11388     
_________________________________________________________________
dropout_20 (Dropout)         (None, 78)                0         
_________________________________________________________________
dense_50 (Dense)             (None, 52)                4108      
_________________________________________________________________
dropout_21 (Dropout)         (None, 52)                0         
_________________________________________________________________
dense_51 (Dense)             (None, 10)                530       
=================================================================
Total params: 502,392
Trainable params: 502,392
Non-trainable params: 0
_________________________________________________________________
```

In [64]:
```
model_drop5.compile(optimizer='adam',loss="categorical_crossentropy",me
trics=['accuracy'])
history=model_drop5.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 [==============================] - 13s 214us/step - loss:
1.6559 - acc: 0.4006 - val_loss: 0.5646 - val_acc: 0.8356
Epoch 2/20
60000/60000 [==============================] - 12s 198us/step - loss:
0.6610 - acc: 0.8079 - val_loss: 0.2800 - val_acc: 0.9358
Epoch 3/20
60000/60000 [==============================] - 11s 191us/step - loss:
0.4404 - acc: 0.8884 - val_loss: 0.2217 - val_acc: 0.9467
Epoch 4/20
60000/60000 [==============================] - 12s 192us/step - loss:
0.3650 - acc: 0.9129 - val_loss: 0.1894 - val_acc: 0.9559
Epoch 5/20
60000/60000 [==============================] - 12s 192us/step - loss:
0.3160 - acc: 0.9267 - val_loss: 0.1725 - val_acc: 0.9586
Epoch 6/20
60000/60000 [==============================] - 12s 194us/step - loss:
0.2913 - acc: 0.9334 - val_loss: 0.1478 - val_acc: 0.9644
Epoch 7/20
60000/60000 [==============================] - 12s 197us/step - loss:
0.2616 - acc: 0.9394 - val_loss: 0.1490 - val_acc: 0.9640
Epoch 8/20
60000/60000 [==============================] - 12s 192us/step - loss:
0.2386 - acc: 0.9445 - val_loss: 0.1532 - val_acc: 0.9658
Epoch 9/20
60000/60000 [==============================] - 12s 193us/step - loss:
0.2289 - acc: 0.9471 - val_loss: 0.1426 - val_acc: 0.9672
Epoch 10/20
60000/60000 [==============================] - 12s 195us/step - loss:
0.2160 - acc: 0.9498 - val_loss: 0.1395 - val_acc: 0.9676
Epoch 11/20
60000/60000 [==============================] - 12s 193us/step - loss:
0.2060 - acc: 0.9526 - val_loss: 0.1325 - val_acc: 0.9707
```

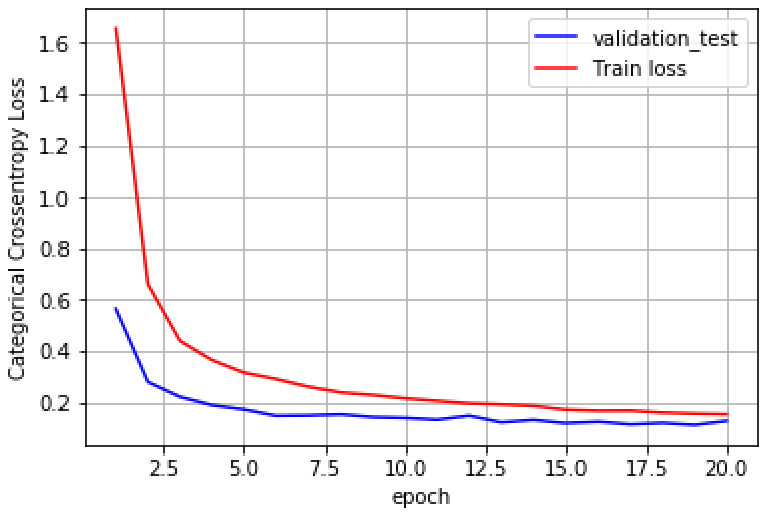
```
Epoch 12/20
60000/60000 [==============================] - 11s 191us/step - loss:
0.1963 - acc: 0.9550 - val_loss: 0.1481 - val_acc: 0.9688
Epoch 13/20
60000/60000 [==============================] - 11s 191us/step - loss:
0.1921 - acc: 0.9558 - val_loss: 0.1222 - val_acc: 0.9728
Epoch 14/20
60000/60000 [==============================] - 13s 209us/step - loss:
0.1866 - acc: 0.9577 - val_loss: 0.1317 - val_acc: 0.9721
Epoch 15/20
60000/60000 [==============================] - 12s 205us/step - loss:
0.1718 - acc: 0.9600 - val_loss: 0.1190 - val_acc: 0.9735
Epoch 16/20
60000/60000 [==============================] - 12s 199us/step - loss:
0.1674 - acc: 0.9613 - val_loss: 0.1247 - val_acc: 0.9735
Epoch 17/20
60000/60000 [==============================] - 12s 201us/step - loss:
0.1680 - acc: 0.9620 - val_loss: 0.1144 - val_acc: 0.9759
Epoch 18/20
60000/60000 [==============================] - 12s 200us/step - loss:
0.1600 - acc: 0.9631 - val_loss: 0.1199 - val_acc: 0.9750
Epoch 19/20
60000/60000 [==============================] - 11s 189us/step - loss:
0.1562 - acc: 0.9639 - val_loss: 0.1121 - val_acc: 0.9763
Epoch 20/20
60000/60000 [==============================] - 11s 188us/step - loss:
0.1538 - acc: 0.9647 - val_loss: 0.1277 - val_acc: 0.9745
```

In [65]:

```
score = model_drop5.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
import matplotlib.pyplot as plt
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, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter vali
dation_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: 0.127720822353533
Test accuracy: 0.9745
```



3.4

ADAM+RELU+BATCHNORMALIZATION+DROPOU

In [66]:

```
model_final5=Sequential()

model_final5.add(Dense(415,activation="relu",input_shape=(input_dim,),k
ernel_initializer=he_normal(seed=None)))
model_final5.add(BatchNormalization())
model_final5.add(Dropout(0.5))

model_final5.add(Dense(286,activation="relu",kernel_initializer=he_norm
al(seed=None)))
model_final5.add(BatchNormalization())
model_final5.add(Dropout(0.5))

model_final5.add(Dense(145,activation="relu",kernel_initializer=he_norm
al(seed=None)))
model_final5.add(BatchNormalization())
model_final5.add(Dropout(0.5))

model_final5.add(Dense(78,activation="relu",kernel_initializer=he_norma
l(seed=None)))
model_final5.add(BatchNormalization())
model_final5.add(Dropout(0.5))

model_final5.add(Dense(52,activation="relu",kernel_initializer=he_norma
l(seed=None)))
model_final5.add(BatchNormalization())
model_final5.add(Dropout(0.5))


model_final5.add(Dense(output_dim,activation='softmax'))

model_final5.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_52 (Dense)             (None, 415)               325775
```

```
batch_normalization_19 (Batc (None, 415)               1660
_________________________________________________________________
dropout_22 (Dropout)         (None, 415)               0
_________________________________________________________________
dense_53 (Dense)             (None, 286)               118976
_________________________________________________________________
batch_normalization_20 (Batc (None, 286)               1144
_________________________________________________________________
dropout_23 (Dropout)         (None, 286)               0
_________________________________________________________________
dense_54 (Dense)             (None, 145)               41615
_________________________________________________________________
batch_normalization_21 (Batc (None, 145)               580
_________________________________________________________________
dropout_24 (Dropout)         (None, 145)               0
_________________________________________________________________
dense_55 (Dense)             (None, 78)                11388
_________________________________________________________________
batch_normalization_22 (Batc (None, 78)                312
_________________________________________________________________
dropout_25 (Dropout)         (None, 78)                0
_________________________________________________________________
dense_56 (Dense)             (None, 52)                4108
_________________________________________________________________
batch_normalization_23 (Batc (None, 52)                208
_________________________________________________________________
dropout_26 (Dropout)         (None, 52)                0
_________________________________________________________________
dense_57 (Dense)             (None, 10)                530
=================================================================
Total params: 506,296
Trainable params: 504,344
Non-trainable params: 1,952
_________________________________________________________________
```

In [67]:

```
model_final5.compile(optimizer='adam',loss="categorical_crossentropy",metrics=['accuracy'])
```

```
history=model_final5.fit(X_train,y_train,batch_size=batch_size,epochs=n
b_epoch,verbose=1,validation_data=(X_test,y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 17s 283us/step - loss:
1.4048 - acc: 0.5410 - val_loss: 0.3185 - val_acc: 0.9126
Epoch 2/20
60000/60000 [==============================] - 13s 219us/step - loss:
0.5268 - acc: 0.8505 - val_loss: 0.2047 - val_acc: 0.9430
Epoch 3/20
60000/60000 [==============================] - 14s 226us/step - loss:
0.3682 - acc: 0.9040 - val_loss: 0.1618 - val_acc: 0.9578
Epoch 4/20
60000/60000 [==============================] - 14s 232us/step - loss:
0.3009 - acc: 0.9226 - val_loss: 0.1454 - val_acc: 0.9621
Epoch 5/20
60000/60000 [==============================] - 13s 224us/step - loss:
0.2606 - acc: 0.9349 - val_loss: 0.1287 - val_acc: 0.9665
Epoch 6/20
60000/60000 [==============================] - 13s 223us/step - loss:
0.2331 - acc: 0.9426 - val_loss: 0.1139 - val_acc: 0.9707
Epoch 7/20
60000/60000 [==============================] - 13s 223us/step - loss:
0.2094 - acc: 0.9470 - val_loss: 0.1129 - val_acc: 0.9708
Epoch 8/20
60000/60000 [==============================] - 14s 226us/step - loss:
0.1984 - acc: 0.9516 - val_loss: 0.1133 - val_acc: 0.9731
Epoch 9/20
60000/60000 [==============================] - 14s 226us/step - loss:
0.1843 - acc: 0.9545 - val_loss: 0.0988 - val_acc: 0.9751
Epoch 10/20
60000/60000 [==============================] - 14s 227us/step - loss:
0.1713 - acc: 0.9584 - val_loss: 0.0995 - val_acc: 0.9745
Epoch 11/20
60000/60000 [==============================] - 14s 237us/step - loss:
0.1648 - acc: 0.9593 - val_loss: 0.0916 - val_acc: 0.9760
Epoch 12/20
60000/60000 [==============================] - 15s 255us/step - loss:
0.1591 - acc: 0.9610 - val_loss: 0.0848 - val_acc: 0.9786
```

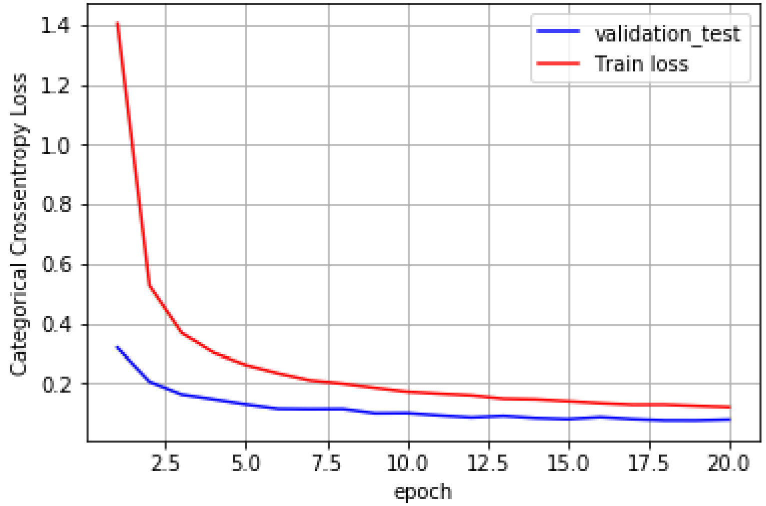
```
Epoch 13/20
60000/60000 [==============================] - 15s 251us/step - loss: 0.1478 - acc: 0.9641 - val_loss: 0.0899 - val_acc: 0.9794
Epoch 14/20
60000/60000 [==============================] - 14s 238us/step - loss: 0.1457 - acc: 0.9650 - val_loss: 0.0826 - val_acc: 0.9799
Epoch 15/20
60000/60000 [==============================] - 14s 234us/step - loss: 0.1394 - acc: 0.9661 - val_loss: 0.0790 - val_acc: 0.9802
Epoch 16/20
60000/60000 [==============================] - 14s 226us/step - loss: 0.1328 - acc: 0.9673 - val_loss: 0.0860 - val_acc: 0.9785
Epoch 17/20
60000/60000 [==============================] - 14s 226us/step - loss: 0.1278 - acc: 0.9689 - val_loss: 0.0791 - val_acc: 0.9797
Epoch 18/20
60000/60000 [==============================] - 14s 226us/step - loss: 0.1279 - acc: 0.9693 - val_loss: 0.0749 - val_acc: 0.9824
Epoch 19/20
60000/60000 [==============================] - 14s 229us/step - loss: 0.1237 - acc: 0.9700 - val_loss: 0.0747 - val_acc: 0.9813
Epoch 20/20
60000/60000 [==============================] - 14s 233us/step - loss: 0.1198 - acc: 0.9710 - val_loss: 0.0776 - val_acc: 0.9816
```

In [68]:

```
score = model_final5.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
import matplotlib.pyplot as plt
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 vali
dation_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: 0.07763815396269784
Test accuracy: 0.9816
```



Conclusion

2 hidden layer

In [20]:
```
from tabulate import tabulate
print("2HIDDEN LAYER I/P-492-160-O/P")
print(tabulate ([['ADam+relu', 0.087,98.13],['adam+relu+batchnormalizat
ion',0.079,98.2],['adam+relu+dropout',0.065,98.38] , ['adam+relu+batchn
ormalization+dropout',0.053,98.29]],    headers=['2hiddenlayer MLP', 't
est score','test accuracy']))
```

```
2HIDDEN LAYER I/P-492-160-O/P
2hiddenlayer MLP                      test score    test accuracy
------------------------------------  ------------  ---------------
ADam+relu                                    0.087            98.13
adam+relu+batchnormalization                 0.079            98.2
adam+relu+dropout                            0.065            98.38
adam+relu+batchnormalization+dropout         0.053            98.29
```

3 hidden layer

In [21]:
```
# from tabulate import tabulate
print("3HIDDEN LAYER I/P-412-298-89-O/P")
print(tabulate ([['ADam+relu', 0.092, 98],['adam+relu+batchnormalizatio
n',0.088,97],['adam+relu+dropout',0.074,98.16] , ['adam+relu+batchnorma
lization+dropout',0.064,98.28]],    headers=['3hiddenlayer MLP', 'test
 score','test accuracy']))
```

```
3HIDDEN LAYER I/P-412-298-89-O/P
3hiddenlayer MLP                      test score    test accuracy
------------------------------------  ------------  ---------------
ADam+relu                                    0.092            98
adam+relu+batchnormalization                 0.088            97
adam+relu+dropout                            0.074            98.16
adam+relu+batchnormalization+dropout         0.064            98.28
```

5 hidden layer

In [22]:

```
from tabulate import tabulate
print("5HIDDEN LAYER I/P-415-286-145-78-52-O/P")
print(tabulate ([['ADam+relu', 0.1063, 97.8],['adam+relu+batchnormalization',0.10,97.6],['adam+relu+dropout',0.127,98.45] , ['adam+relu+batchnormalization+dropout',0.077,85.72]],    headers=['5hiddenlayer MLP', 'test score','test accuracy']))
```

```
5HIDDEN LAYER I/P-415-286-145-78-52-O/P
5hiddenlayer MLP                          test score    test accuracy
------------------------------------  ------------  ---------------
ADam+relu                                   0.1063            97.8
adam+relu+batchnormalization                0.1               97.6
adam+relu+dropout                           0.127             98.45
adam+relu+batchnormalization+dropout        0.077             85.72
```