In [1]:
```
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
from keras.initializers import he_normal
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import Dropout
```

```
Using TensorFlow backend.
```

Plotting for each Epoch and Loss

In [2]:
```
%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

In [3]:
```
# the data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

In [4]:
```
print("Number of training examples :", X_train.shape[0], "and each imag
e is of shape (%d, %d)"%(X_train.shape[1], X_train.shape[2]))
print("Number of training examples :", X_test.shape[0], "and each image
 is of shape (%d, %d)"%(X_test.shape[1], X_test.shape[2]))
```

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

In [5]:

```
# if you observe the input shape its 3 dimensional vector
# for each image we have a (28*28) vector
# we will convert the (28*28) vector into single dimensional vector of
 1 * 784

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

In [6]:

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

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

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

In [7]:

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

```
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0
```

```
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 25
5
 247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  36  94 15
4
 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  8
2
  82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 25
3
 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 15
4
   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 24
1
 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 18
```

```
7
   0   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 25
3
 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 25
3
 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 19
5
  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   1
6
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0]
```

Normalize the data

In [8]:
```
# if we observe the above matrix each cell is having a value between 0-
255
# before we move to apply machine learning algorithms lets try to norma
lize the data
# X => (X - Xmin)/(Xmax-Xmin) = X/255

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

In [9]:
```
# example data point after normlizing
print(X_train[0])
```

```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.01176471 0.07058824 0.07058824 0.07058824
```

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

```
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.31764706 0.94117647 0.99215686
0.99215686 0.46666667 0.09803922 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.17647059 0.72941176 0.99215686 0.99215686
0.58823529 0.10588235 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.0627451  0.36470588 0.98823529 0.99215686 0.73333333
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.97647059 0.99215686 0.97647059 0.25098039 0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.18039216 0.50980392 0.71764706 0.99215686
0.99215686 0.81176471 0.00784314 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.15294118 0.58039216
0.89803922 0.99215686 0.99215686 0.99215686 0.98039216 0.71372549
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.09411765 0.44705882 0.86666667 0.99215686 0.99215686 0.99215686
0.99215686 0.78823529 0.30588235 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.09019608 0.25882353 0.83529412 0.99215686
0.99215686 0.99215686 0.99215686 0.77647059 0.31764706 0.00784314
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.07058824 0.67058824
```

```
 0.85882353 0.99215686 0.99215686 0.99215686 0.99215686 0.76470588
 0.31372549 0.03529412 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.21568627 0.6745098  0.88627451 0.99215686 0.99215686 0.99215686
 0.99215686 0.95686275 0.52156863 0.04313725 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.53333333 0.99215686
 0.99215686 0.99215686 0.83137255 0.52941176 0.51764706 0.0627451
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
```

Vectorizing the Class Label to 10 Dimensions

In [10]:

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

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

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

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

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

Defining some model parameters

In [11]:
```
output_dim = 10
input_dim = X_train.shape[1]
batch_size = 128
nb_epoch = 20
```

Model 1: --> 2 - Hidden Layers

MLP + ReLU activation + ADAMOptimizer

In [12]:
```
model_relu = Sequential()
model_relu.add(Dense(364, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(Dense(64, activation='relu', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

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

```
WARNING:tensorflow:From C:\Users\kingsubham27091995\Anaconda3\lib\site-
packages\tensorflow\python\framework\op_def_library.py:263: colocate_wi
th (from tensorflow.python.framework.ops) is deprecated and will be rem
oved in a future version.
Instructions for updating:
Colocations handled automatically by placer.
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 364)               285740    
_________________________________________________________________
dense_2 (Dense)              (None, 64)                23360     
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650       
=================================================================
Total params: 309,750
Trainable params: 309,750
Non-trainable params: 0
_________________________________________________________________
None
WARNING:tensorflow:From C:\Users\kingsubham27091995\Anaconda3\lib\site-
packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tensorf
low.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 16s 264us/step - loss:
0.2589 - acc: 0.9235 - val_loss: 0.1318 - val_acc: 0.9604
Epoch 2/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.
1014 - acc: 0.9698 - val_loss: 0.0906 - val_acc: 0.9707
Epoch 3/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.
0637 - acc: 0.9805 - val_loss: 0.0858 - val_acc: 0.9732
Epoch 4/20
```

```
60000/60000 [==============================] - 8s 130us/step - loss: 0.
0466 - acc: 0.9853 - val_loss: 0.0662 - val_acc: 0.9790
Epoch 5/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.
0317 - acc: 0.9904 - val_loss: 0.0700 - val_acc: 0.9779
Epoch 6/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.
0255 - acc: 0.9919 - val_loss: 0.0709 - val_acc: 0.9773
Epoch 7/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.
0179 - acc: 0.9944 - val_loss: 0.0747 - val_acc: 0.9785
Epoch 8/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.
0148 - acc: 0.9953 - val_loss: 0.0702 - val_acc: 0.9807
Epoch 9/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.
0132 - acc: 0.9957 - val_loss: 0.0700 - val_acc: 0.9812
Epoch 10/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.
0117 - acc: 0.9962 - val_loss: 0.0751 - val_acc: 0.9790
Epoch 11/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.
0101 - acc: 0.9968 - val_loss: 0.0709 - val_acc: 0.9810
Epoch 12/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.
0097 - acc: 0.9969 - val_loss: 0.0719 - val_acc: 0.9824
Epoch 13/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.
0056 - acc: 0.9984 - val_loss: 0.0966 - val_acc: 0.9770
Epoch 14/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.
0116 - acc: 0.9961 - val_loss: 0.0863 - val_acc: 0.9793
Epoch 15/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.
0069 - acc: 0.9978 - val_loss: 0.0934 - val_acc: 0.9783
Epoch 16/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.
0102 - acc: 0.9966 - val_loss: 0.0865 - val_acc: 0.9809
Epoch 17/20
```

```
60000/60000 [==============================] - 8s 129us/step - loss: 0.
0075 - acc: 0.9974 - val_loss: 0.0846 - val_acc: 0.9817
Epoch 18/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.
0070 - acc: 0.9976 - val_loss: 0.0820 - val_acc: 0.9813
Epoch 19/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.
0065 - acc: 0.9979 - val_loss: 0.0883 - val_acc: 0.9812
Epoch 20/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.
0052 - acc: 0.9983 - val_loss: 0.0937 - val_acc: 0.9805
```

Results:

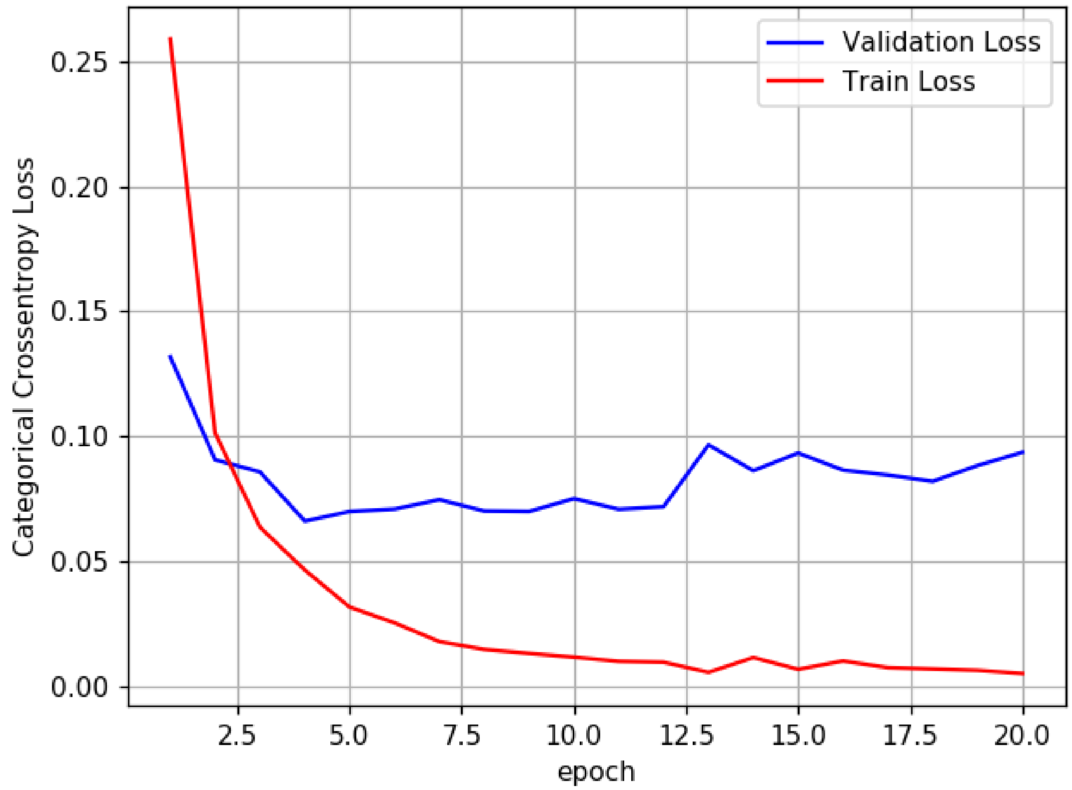
1. Train Accuracy= 99.83%

Plotting Each Epoch vs Loss

In [13]:

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

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

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



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

```
Test score: 0.09365907231679503
Test accuracy: 0.9805
```



MLP + Batch-Norm on hidden Layers + AdamOptimizer

In [14]:
```
from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_batch.add(Dense(364, activation='relu', input_shape=(input_dim,),
 kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
```

```
model_batch.add(BatchNormalization())

model_batch.add(Dense(64, activation='relu', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.125, seed=None)) )
model_batch.add(BatchNormalization())

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


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

history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epoc
hs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.
2429 - acc: 0.9305 - val_loss: 0.1228 - val_acc: 0.9621
Epoch 2/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.
0843 - acc: 0.9756 - val_loss: 0.0890 - val_acc: 0.9724
Epoch 3/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.
0528 - acc: 0.9848 - val_loss: 0.0782 - val_acc: 0.9756
Epoch 4/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.
0368 - acc: 0.9888 - val_loss: 0.0766 - val_acc: 0.9772
Epoch 5/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.
0286 - acc: 0.9913 - val_loss: 0.0706 - val_acc: 0.9772
Epoch 6/20
60000/60000 [==============================] - 9s 158us/step - loss: 0.
0216 - acc: 0.9933 - val_loss: 0.0783 - val_acc: 0.9762
Epoch 7/20
60000/60000 [==============================] - 10s 162us/step - loss:
0.0185 - acc: 0.9943 - val_loss: 0.0738 - val_acc: 0.9798
Epoch 8/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.
0160 - acc: 0.9949 - val_loss: 0.0749 - val_acc: 0.9776
```

```
Epoch 9/20
60000/60000 [==============================] - 10s 159us/step - loss:
0.0131 - acc: 0.9958 - val_loss: 0.0767 - val_acc: 0.9780
Epoch 10/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.
0126 - acc: 0.9960 - val_loss: 0.0789 - val_acc: 0.9781
Epoch 11/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.
0113 - acc: 0.9965 - val_loss: 0.0819 - val_acc: 0.9770
Epoch 12/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.
0124 - acc: 0.9960 - val_loss: 0.0848 - val_acc: 0.9768
Epoch 13/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.
0118 - acc: 0.9961 - val_loss: 0.0823 - val_acc: 0.9796
Epoch 14/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.
0107 - acc: 0.9963 - val_loss: 0.0848 - val_acc: 0.9772
Epoch 15/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.
0079 - acc: 0.9972 - val_loss: 0.0824 - val_acc: 0.9808
Epoch 16/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.
0071 - acc: 0.9974 - val_loss: 0.0850 - val_acc: 0.9782
Epoch 17/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.
0091 - acc: 0.9971 - val_loss: 0.0886 - val_acc: 0.9796
Epoch 18/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.
0078 - acc: 0.9974 - val_loss: 0.0825 - val_acc: 0.9780
Epoch 19/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.
0069 - acc: 0.9980 - val_loss: 0.0722 - val_acc: 0.9825
Epoch 20/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.
0073 - acc: 0.9976 - val_loss: 0.0839 - val_acc: 0.9793
```

Results:

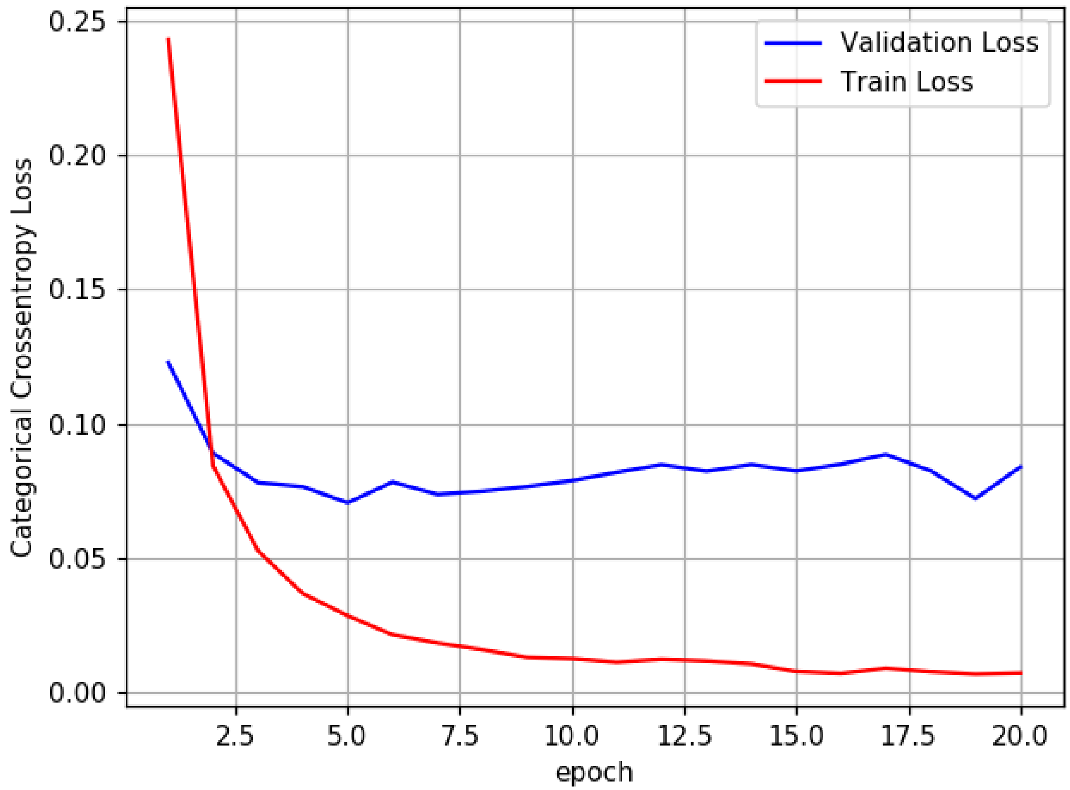
1. Train Accuracy= 99.76%

Plotting Each Epoch vs Loss

In [15]:

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

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

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



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

```
Test score: 0.08393578212396678
Test accuracy: 0.9793
```



MLP + Dropout(0.5) + AdamOptimizer

In [16]:
```
from keras.layers.normalization import BatchNormalization

model_drop = Sequential()

model_drop.add(Dense(364, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
```

```
model_drop.add(Dense(64, activation='relu', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.125, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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


model_drop.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epoch
s=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s 169us/step - loss:
0.5970 - acc: 0.8187 - val_loss: 0.1836 - val_acc: 0.9441
Epoch 2/20
60000/60000 [==============================] - 10s 174us/step - loss:
0.2818 - acc: 0.9191 - val_loss: 0.1288 - val_acc: 0.9602
Epoch 3/20
60000/60000 [==============================] - 10s 174us/step - loss:
0.2140 - acc: 0.9375 - val_loss: 0.1104 - val_acc: 0.9664
Epoch 4/20
60000/60000 [==============================] - 10s 164us/step - loss:
0.1797 - acc: 0.9470 - val_loss: 0.0951 - val_acc: 0.9714
Epoch 5/20
60000/60000 [==============================] - 10s 161us/step - loss:
0.1577 - acc: 0.9539 - val_loss: 0.0871 - val_acc: 0.9734
Epoch 6/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.
1435 - acc: 0.9582 - val_loss: 0.0786 - val_acc: 0.9757
Epoch 7/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.
1327 - acc: 0.9611 - val_loss: 0.0758 - val_acc: 0.9763
Epoch 8/20
60000/60000 [==============================] - 10s 164us/step - loss:
0.1216 - acc: 0.9633 - val_loss: 0.0727 - val_acc: 0.9776
```

```
Epoch 9/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.
1154 - acc: 0.9658 - val_loss: 0.0686 - val_acc: 0.9792
Epoch 10/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.
1078 - acc: 0.9683 - val_loss: 0.0690 - val_acc: 0.9793
Epoch 11/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.
0997 - acc: 0.9700 - val_loss: 0.0704 - val_acc: 0.9781
Epoch 12/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.
0934 - acc: 0.9714 - val_loss: 0.0713 - val_acc: 0.9782
Epoch 13/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.
0899 - acc: 0.9729 - val_loss: 0.0637 - val_acc: 0.9808
Epoch 14/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.
0864 - acc: 0.9736 - val_loss: 0.0626 - val_acc: 0.9813
Epoch 15/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.
0824 - acc: 0.9745 - val_loss: 0.0674 - val_acc: 0.9813
Epoch 16/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.
0815 - acc: 0.9752 - val_loss: 0.0673 - val_acc: 0.9814
Epoch 17/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.
0763 - acc: 0.9768 - val_loss: 0.0632 - val_acc: 0.9810
Epoch 18/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.
0747 - acc: 0.9768 - val_loss: 0.0662 - val_acc: 0.9817
Epoch 19/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.
0767 - acc: 0.9765 - val_loss: 0.0590 - val_acc: 0.9825
Epoch 20/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.
0704 - acc: 0.9787 - val_loss: 0.0658 - val_acc: 0.9805
```

Results:

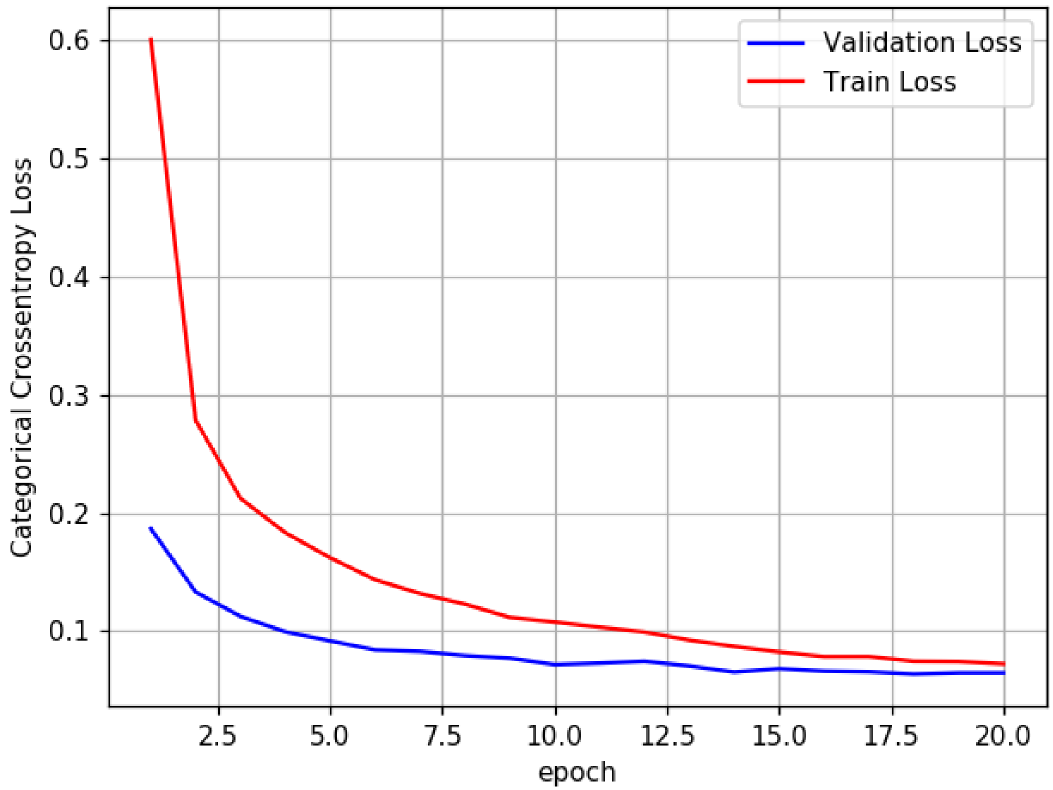
1. Train Accuracy= 97.87%

Plotting Each Epoch vs Loss

In [17]:

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

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

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



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

```
Test score: 0.06485723318307719
Test accuracy: 0.9811
```



MLP + Dropout(0.1) + AdamOptimizer

In [12]:
```
from keras.layers.normalization import BatchNormalization

model_drop = Sequential()

model_drop.add(Dense(364, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.1))
```

```
model_drop.add(Dense(64, activation='relu', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.125, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.1))

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


model_drop.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epoch
s=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
WARNING:tensorflow:From C:\Users\kingsubham27091995\Anaconda3\lib\site-
packages\tensorflow\python\framework\op_def_library.py:263: colocate_wi
th (from tensorflow.python.framework.ops) is deprecated and will be rem
oved in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From C:\Users\kingsubham27091995\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:3445: calling dropout (fro
m tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will b
e removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate =
1 - keep_prob`.
WARNING:tensorflow:From C:\Users\kingsubham27091995\Anaconda3\lib\site-
packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tensorf
low.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 12s 207us/step - loss:
0.2750 - acc: 0.9192 - val_loss: 0.1166 - val_acc: 0.9655
Epoch 2/20
60000/60000 [==============================] - 9s 158us/step - loss: 0.
```

```
1118 - acc: 0.9666 - val_loss: 0.0909 - val_acc: 0.9706
Epoch 3/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.
0768 - acc: 0.9770 - val_loss: 0.0804 - val_acc: 0.9752
Epoch 4/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.
0585 - acc: 0.9817 - val_loss: 0.0731 - val_acc: 0.9774
Epoch 5/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.
0458 - acc: 0.9856 - val_loss: 0.0739 - val_acc: 0.9775
Epoch 6/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.
0393 - acc: 0.9877 - val_loss: 0.0761 - val_acc: 0.9764
Epoch 7/20
60000/60000 [==============================] - 9s 142us/step - loss: 0.
0369 - acc: 0.9878 - val_loss: 0.0766 - val_acc: 0.9770
Epoch 8/20
60000/60000 [==============================] - 9s 142us/step - loss: 0.
0333 - acc: 0.9888 - val_loss: 0.0706 - val_acc: 0.9783
Epoch 9/20
60000/60000 [==============================] - 8s 142us/step - loss: 0.
0285 - acc: 0.9907 - val_loss: 0.0724 - val_acc: 0.9788
Epoch 10/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.
0243 - acc: 0.9922 - val_loss: 0.0713 - val_acc: 0.9796
Epoch 11/20
60000/60000 [==============================] - 10s 162us/step - loss:
0.0247 - acc: 0.9916 - val_loss: 0.0724 - val_acc: 0.9806
Epoch 12/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.
0200 - acc: 0.9932 - val_loss: 0.0748 - val_acc: 0.9790
Epoch 13/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.
0195 - acc: 0.9931 - val_loss: 0.0753 - val_acc: 0.9809
Epoch 14/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.
0193 - acc: 0.9934 - val_loss: 0.0720 - val_acc: 0.9816
Epoch 15/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.
```

```
0151 - acc: 0.9950 - val_loss: 0.0735 - val_acc: 0.9826
Epoch 16/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.
0166 - acc: 0.9945 - val_loss: 0.0676 - val_acc: 0.9829
Epoch 17/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.
0158 - acc: 0.9948 - val_loss: 0.0774 - val_acc: 0.9813
Epoch 18/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.
0143 - acc: 0.9951 - val_loss: 0.0724 - val_acc: 0.9811
Epoch 19/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.
0149 - acc: 0.9950 - val_loss: 0.0704 - val_acc: 0.9821
Epoch 20/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.
0126 - acc: 0.9957 - val_loss: 0.0691 - val_acc: 0.9830
```

Results:

1. Train Accuracy= 99.57%

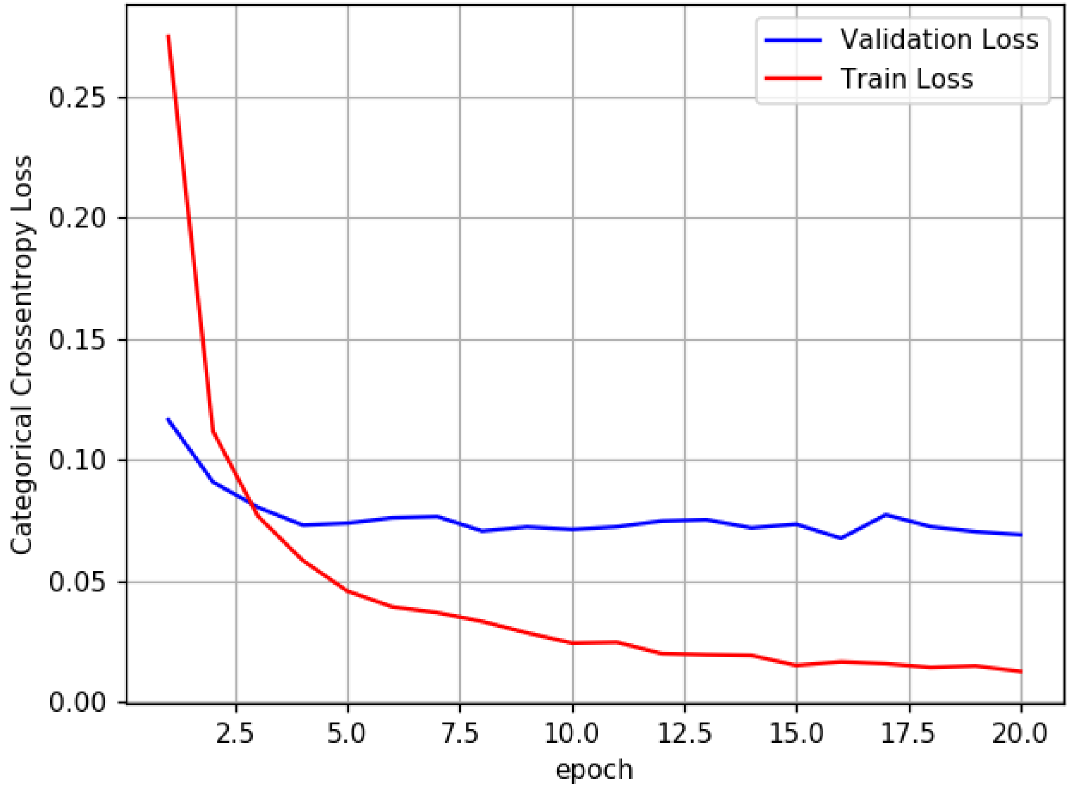
Plotting each Epoch vs Loss

In [13]:

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

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

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



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

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

```
Test score: 0.06910191163946729
Test accuracy: 0.983
```



MLP + Dropout (0.7)+ AdamOptimizer

In [14]:
```
from keras.layers.normalization import BatchNormalization
```

```
model_drop = Sequential()

model_drop.add(Dense(364, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.7))

model_drop.add(Dense(64, activation='relu', kernel_initializer=RandomNo
rmal(mean=0.0, stddev=0.125, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.7))

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


model_drop.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epoch
s=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s 173us/step - loss:
1.0427 - acc: 0.6779 - val_loss: 0.2808 - val_acc: 0.9202
Epoch 2/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.
5151 - acc: 0.8481 - val_loss: 0.2065 - val_acc: 0.9393
Epoch 3/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.
4049 - acc: 0.8848 - val_loss: 0.1728 - val_acc: 0.9478
Epoch 4/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.
3492 - acc: 0.9026 - val_loss: 0.1515 - val_acc: 0.9528
Epoch 5/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.
3088 - acc: 0.9136 - val_loss: 0.1388 - val_acc: 0.9590
Epoch 6/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.
2811 - acc: 0.9212 - val_loss: 0.1274 - val_acc: 0.9623
```

```
Epoch 7/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.
2594 - acc: 0.9278 - val_loss: 0.1134 - val_acc: 0.9667
Epoch 8/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.
2422 - acc: 0.9337 - val_loss: 0.1147 - val_acc: 0.9664
Epoch 9/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.
2323 - acc: 0.9372 - val_loss: 0.1084 - val_acc: 0.9689
Epoch 10/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.
2219 - acc: 0.9398 - val_loss: 0.1012 - val_acc: 0.9711
Epoch 11/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.
2076 - acc: 0.9430 - val_loss: 0.1011 - val_acc: 0.9715
Epoch 12/20
60000/60000 [==============================] - 8s 142us/step - loss: 0.
2073 - acc: 0.9441 - val_loss: 0.0937 - val_acc: 0.9722
Epoch 13/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.
1949 - acc: 0.9469 - val_loss: 0.0900 - val_acc: 0.9727
Epoch 14/20
60000/60000 [==============================] - 11s 188us/step - loss:
0.1941 - acc: 0.9472 - val_loss: 0.0925 - val_acc: 0.9734
Epoch 15/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.
1870 - acc: 0.9492 - val_loss: 0.0912 - val_acc: 0.9738
Epoch 16/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.
1793 - acc: 0.9515 - val_loss: 0.0877 - val_acc: 0.9745
Epoch 17/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.
1759 - acc: 0.9524 - val_loss: 0.0842 - val_acc: 0.9749
Epoch 18/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.
1708 - acc: 0.9534 - val_loss: 0.0854 - val_acc: 0.9750
Epoch 19/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.
1655 - acc: 0.9546 - val_loss: 0.0825 - val_acc: 0.9774
```

```
Epoch 20/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.
1632 - acc: 0.9558 - val_loss: 0.0842 - val_acc: 0.9750
```

Results:

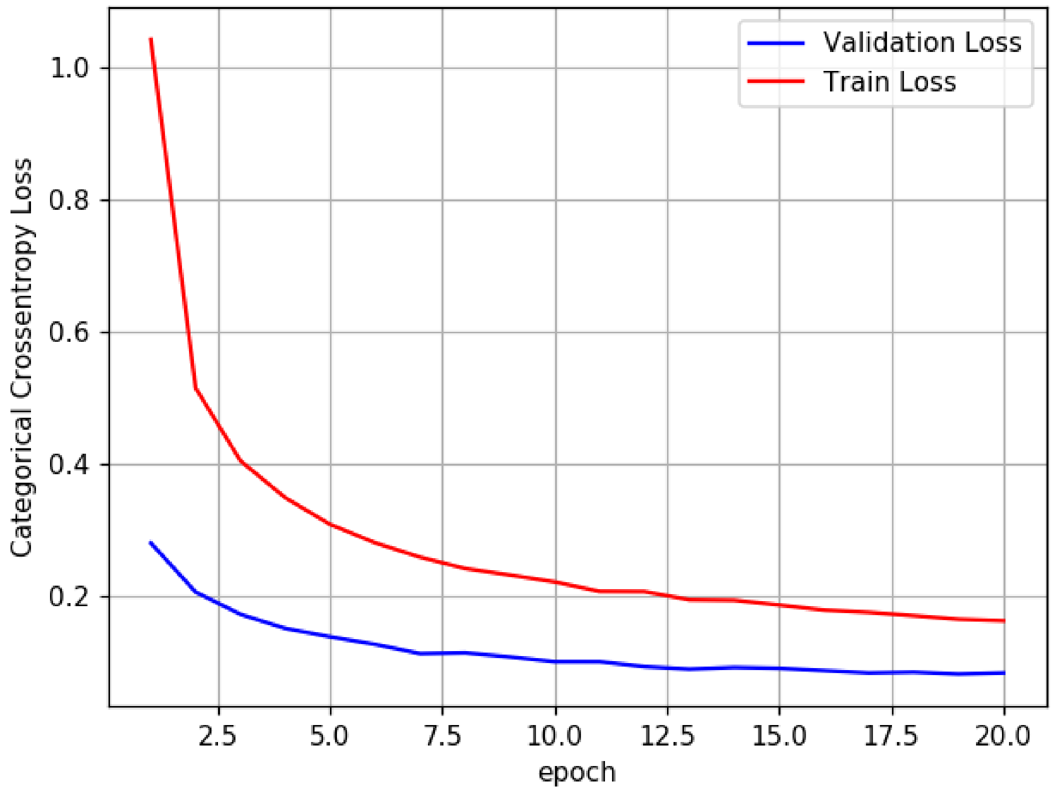
1. Train Accuracy= 95.58%

Plotting each Epoch vs Loss

In [15]:

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

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

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



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

```
Test score: 0.08424389982710127
Test accuracy: 0.975
```



Model 2:----> 3 Hidden Layers

MLP + ReLU activation + ADAMOptimizer

In [18]:
```
model_relu = Sequential()

model_relu.add(Dense(364, activation='relu', input_shape=(input_dim,),
kernel_initializer=he_normal(seed=None)))
```

```
model_relu.add(Dense(150, activation='relu', kernel_initializer=he_norm
al(seed=None)) )
model_relu.add(Dense(70, activation='relu', kernel_initializer=he_norma
l(seed=None)) )

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


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

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.
2564 - acc: 0.9242 - val_loss: 0.1124 - val_acc: 0.9647
Epoch 2/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.
0908 - acc: 0.9725 - val_loss: 0.0857 - val_acc: 0.9744
Epoch 3/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.
0595 - acc: 0.9815 - val_loss: 0.0762 - val_acc: 0.9750
Epoch 4/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.
0429 - acc: 0.9862 - val_loss: 0.0689 - val_acc: 0.9788
Epoch 5/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.
0328 - acc: 0.9891 - val_loss: 0.0709 - val_acc: 0.9775
Epoch 6/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.
0255 - acc: 0.9918 - val_loss: 0.0690 - val_acc: 0.9805
Epoch 7/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.
0210 - acc: 0.9933 - val_loss: 0.0825 - val_acc: 0.9787
Epoch 8/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.
0191 - acc: 0.9935 - val_loss: 0.1016 - val_acc: 0.9734
Epoch 9/20
```

```
60000/60000 [==============================] - 8s 131us/step - loss: 0.
0143 - acc: 0.9952 - val_loss: 0.0754 - val_acc: 0.9813
Epoch 10/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.
0159 - acc: 0.9944 - val_loss: 0.0830 - val_acc: 0.9794
Epoch 11/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.
0166 - acc: 0.9944 - val_loss: 0.0955 - val_acc: 0.9780
Epoch 12/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.
0112 - acc: 0.9962 - val_loss: 0.0851 - val_acc: 0.9798
Epoch 13/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.
0124 - acc: 0.9959 - val_loss: 0.1014 - val_acc: 0.9762
Epoch 14/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.
0096 - acc: 0.9968 - val_loss: 0.0935 - val_acc: 0.9786
Epoch 15/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.
0094 - acc: 0.9968 - val_loss: 0.0921 - val_acc: 0.9795
Epoch 16/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.
0133 - acc: 0.9958 - val_loss: 0.0889 - val_acc: 0.9799
Epoch 17/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.
0071 - acc: 0.9978 - val_loss: 0.0807 - val_acc: 0.9824
Epoch 18/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.
0093 - acc: 0.9972 - val_loss: 0.0880 - val_acc: 0.9805
Epoch 19/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.
0090 - acc: 0.9972 - val_loss: 0.1011 - val_acc: 0.9786
Epoch 20/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.
0089 - acc: 0.9972 - val_loss: 0.1021 - val_acc: 0.9799
```

Results:

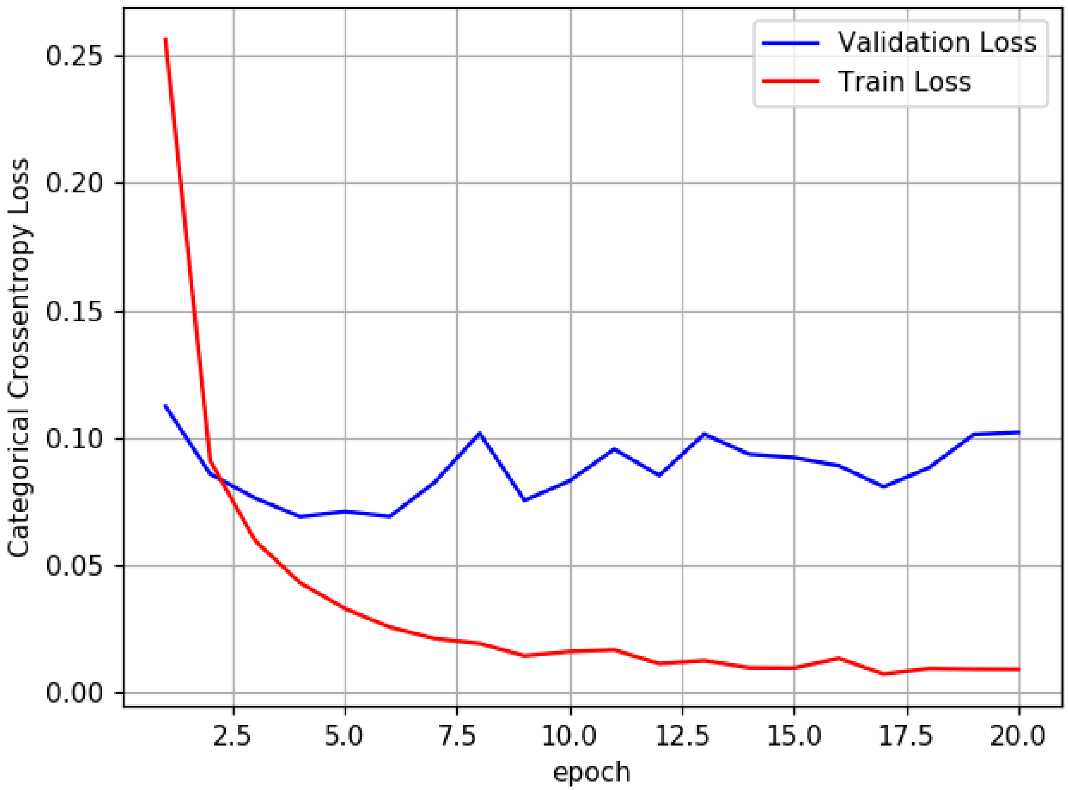
1. Train Accuracy= 99.72%

Plotting each Epoch vs Loss

In [19]:

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

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

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


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

```
Test score: 0.10209789935903654
Test accuracy: 0.9799
```



MLP + Batch-Norm on hidden Layers + AdamOptimizer

In [20]:
```
model_batch = Sequential()

model_batch.add(Dense(364, activation='relu', input_shape=(input_dim,),
 kernel_initializer=he_normal(seed=None)))
model_batch.add(BatchNormalization())
model_batch.add(Dense(150, activation='relu', kernel_initializer=he_nor
mal(seed=None)) )
```

```
model_batch.add(BatchNormalization())
model_batch.add(Dense(70, activation='relu', kernel_initializer=he_norm
al(seed=None)) )
model_batch.add(BatchNormalization())
model_batch.add(Dense(output_dim, activation='softmax'))


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

history = model_batch.fit(X_train, Y_train, batch_size=batch_size, epoc
hs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 14s 225us/step - loss:
0.2211 - acc: 0.9350 - val_loss: 0.1058 - val_acc: 0.9665
Epoch 2/20
60000/60000 [==============================] - 10s 172us/step - loss:
0.0809 - acc: 0.9756 - val_loss: 0.0890 - val_acc: 0.9733
Epoch 3/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.
0513 - acc: 0.9839 - val_loss: 0.0836 - val_acc: 0.9752
Epoch 4/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.
0404 - acc: 0.9868 - val_loss: 0.0796 - val_acc: 0.9747
Epoch 5/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.
0324 - acc: 0.9899 - val_loss: 0.0864 - val_acc: 0.9746
Epoch 6/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.
0266 - acc: 0.9914 - val_loss: 0.0832 - val_acc: 0.9765
Epoch 7/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.
0205 - acc: 0.9931 - val_loss: 0.0716 - val_acc: 0.9801
Epoch 8/20
60000/60000 [==============================] - 10s 160us/step - loss:
0.0201 - acc: 0.9932 - val_loss: 0.0902 - val_acc: 0.9755
Epoch 9/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.
```

```
0206 - acc: 0.9927 - val_loss: 0.0734 - val_acc: 0.9790
Epoch 10/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.
0172 - acc: 0.9940 - val_loss: 0.0723 - val_acc: 0.9794
Epoch 11/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.
0147 - acc: 0.9954 - val_loss: 0.0816 - val_acc: 0.9801
Epoch 12/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.
0124 - acc: 0.9962 - val_loss: 0.0788 - val_acc: 0.9798
Epoch 13/20
60000/60000 [==============================] - 10s 161us/step - loss:
0.0124 - acc: 0.9960 - val_loss: 0.0798 - val_acc: 0.9811
Epoch 14/20
60000/60000 [==============================] - 10s 159us/step - loss:
0.0137 - acc: 0.9953 - val_loss: 0.0815 - val_acc: 0.9789
Epoch 15/20
60000/60000 [==============================] - 10s 159us/step - loss:
0.0098 - acc: 0.9969 - val_loss: 0.0754 - val_acc: 0.9802
Epoch 16/20
60000/60000 [==============================] - 9s 158us/step - loss: 0.
0108 - acc: 0.9966 - val_loss: 0.0904 - val_acc: 0.9780
Epoch 17/20
60000/60000 [==============================] - 10s 167us/step - loss:
0.0137 - acc: 0.9952 - val_loss: 0.0726 - val_acc: 0.9823
Epoch 18/20
60000/60000 [==============================] - 10s 171us/step - loss:
0.0109 - acc: 0.9964 - val_loss: 0.0835 - val_acc: 0.9795
Epoch 19/20
60000/60000 [==============================] - 10s 166us/step - loss:
0.0079 - acc: 0.9973 - val_loss: 0.0736 - val_acc: 0.9811
Epoch 20/20
60000/60000 [==============================] - 10s 163us/step - loss:
0.0079 - acc: 0.9974 - val_loss: 0.0791 - val_acc: 0.9801
```

Results:

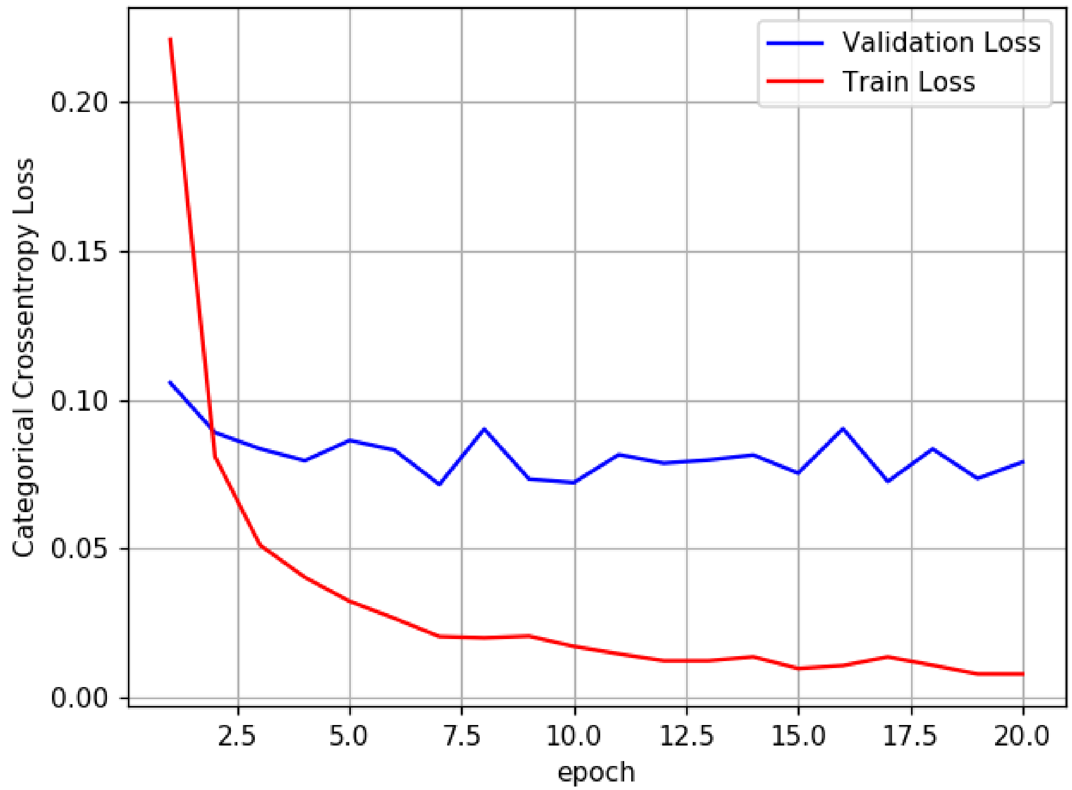
1. Train Accuracy: 99.74%

Plotting each Epoch vs Loss

In [21]:

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

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

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

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

```
Test score: 0.07912658260073877
Test accuracy: 0.9801
```



MLP + Dropout (0.5)+ AdamOptimizer

In [22]:
```
model_drop = Sequential()

model_drop.add(Dense(364, activation='relu', input_shape=(input_dim,),
kernel_initializer=he_normal(seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(150, activation='relu', kernel_initializer=he_norm
```

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

model_drop.add(Dense(70, activation='relu', kernel_initializer=he_norma
l(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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


model_drop.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epoch
s=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 12s 208us/step - loss:
0.7288 - acc: 0.7770 - val_loss: 0.1964 - val_acc: 0.9408
Epoch 2/20
60000/60000 [==============================] - 10s 173us/step - loss:
0.3055 - acc: 0.9129 - val_loss: 0.1416 - val_acc: 0.9572
Epoch 3/20
60000/60000 [==============================] - 11s 175us/step - loss:
0.2356 - acc: 0.9340 - val_loss: 0.1102 - val_acc: 0.9683
Epoch 4/20
60000/60000 [==============================] - 11s 175us/step - loss:
0.1935 - acc: 0.9450 - val_loss: 0.1066 - val_acc: 0.9698
Epoch 5/20
60000/60000 [==============================] - 11s 176us/step - loss:
0.1763 - acc: 0.9503 - val_loss: 0.0935 - val_acc: 0.9721
Epoch 6/20
60000/60000 [==============================] - 11s 176us/step - loss:
0.1553 - acc: 0.9563 - val_loss: 0.0905 - val_acc: 0.9744
Epoch 7/20
60000/60000 [==============================] - 10s 175us/step - loss:
0.1476 - acc: 0.9584 - val_loss: 0.0866 - val_acc: 0.9746
```

```
Epoch 8/20
60000/60000 [==============================] - 11s 176us/step - loss:
0.1436 - acc: 0.9593 - val_loss: 0.0785 - val_acc: 0.9786
Epoch 9/20
60000/60000 [==============================] - 11s 178us/step - loss:
0.1262 - acc: 0.9641 - val_loss: 0.0800 - val_acc: 0.9777
Epoch 10/20
60000/60000 [==============================] - 11s 177us/step - loss:
0.1218 - acc: 0.9662 - val_loss: 0.0775 - val_acc: 0.9791
Epoch 11/20
60000/60000 [==============================] - 11s 177us/step - loss:
0.1146 - acc: 0.9670 - val_loss: 0.0693 - val_acc: 0.9792
Epoch 12/20
60000/60000 [==============================] - 11s 177us/step - loss:
0.1103 - acc: 0.9687 - val_loss: 0.0703 - val_acc: 0.9789
Epoch 13/20
60000/60000 [==============================] - 11s 182us/step - loss:
0.1031 - acc: 0.9704 - val_loss: 0.0673 - val_acc: 0.9803
Epoch 14/20
60000/60000 [==============================] - 11s 177us/step - loss:
0.1019 - acc: 0.9710 - val_loss: 0.0686 - val_acc: 0.9810
Epoch 15/20
60000/60000 [==============================] - 11s 179us/step - loss:
0.0966 - acc: 0.9721 - val_loss: 0.0642 - val_acc: 0.9823
Epoch 16/20
60000/60000 [==============================] - 11s 177us/step - loss:
0.0941 - acc: 0.9728 - val_loss: 0.0631 - val_acc: 0.9822
Epoch 17/20
60000/60000 [==============================] - 11s 179us/step - loss:
0.0940 - acc: 0.9722 - val_loss: 0.0684 - val_acc: 0.9819
Epoch 18/20
60000/60000 [==============================] - 11s 177us/step - loss:
0.0881 - acc: 0.9748 - val_loss: 0.0652 - val_acc: 0.9821
Epoch 19/20
60000/60000 [==============================] - 11s 185us/step - loss:
0.0853 - acc: 0.9757 - val_loss: 0.0658 - val_acc: 0.9812
Epoch 20/20
60000/60000 [==============================] - 11s 183us/step - loss:
0.0801 - acc: 0.9765 - val_loss: 0.0667 - val_acc: 0.9809
```

Results:

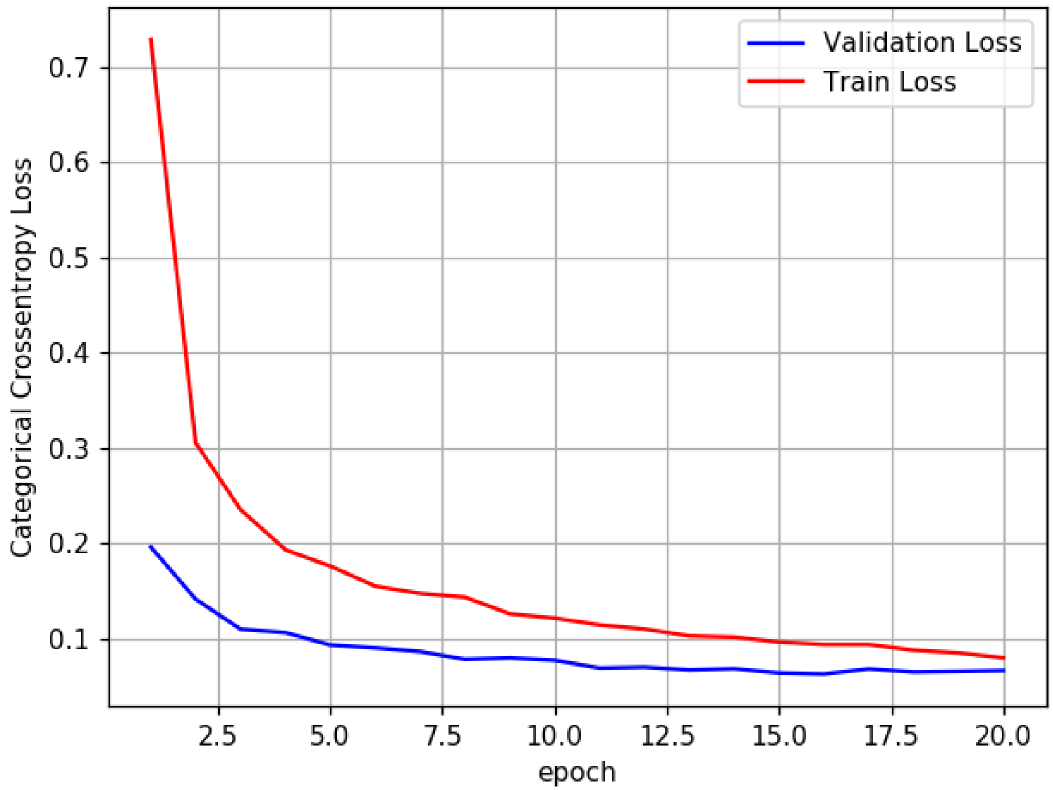
1. Train Accuracy = 97.65%

Plotting each Epoch vs Loss

In [23]:

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

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

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


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

```
Test score: 0.06665783619710709
Test accuracy: 0.9809
```



MLP + Dropout(0.1) + AdamOptimizer

In [17]:
```
model_drop = Sequential()

model_drop.add(Dense(364, activation='relu', input_shape=(input_dim,),
kernel_initializer=he_normal(seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.1))

model_drop.add(Dense(150, activation='relu', kernel_initializer=he_norm
```

```
al(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.1))

model_drop.add(Dense(70, activation='relu', kernel_initializer=he_norma
l(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.1))

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


model_drop.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epoch
s=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 12s 208us/step - loss:
0.2738 - acc: 0.9181 - val_loss: 0.1113 - val_acc: 0.9662
Epoch 2/20
60000/60000 [==============================] - 10s 169us/step - loss:
0.1145 - acc: 0.9649 - val_loss: 0.0873 - val_acc: 0.9733
Epoch 3/20
60000/60000 [==============================] - 10s 174us/step - loss:
0.0809 - acc: 0.9747 - val_loss: 0.0866 - val_acc: 0.9725
Epoch 4/20
60000/60000 [==============================] - 10s 169us/step - loss:
0.0647 - acc: 0.9798 - val_loss: 0.0715 - val_acc: 0.9790
Epoch 5/20
60000/60000 [==============================] - 11s 177us/step - loss:
0.0538 - acc: 0.9831 - val_loss: 0.0697 - val_acc: 0.9787
Epoch 6/20
60000/60000 [==============================] - 10s 172us/step - loss:
0.0470 - acc: 0.9846 - val_loss: 0.0646 - val_acc: 0.9794
Epoch 7/20
60000/60000 [==============================] - 11s 182us/step - loss:
0.0408 - acc: 0.9871 - val_loss: 0.0896 - val_acc: 0.9764
```

```
Epoch 8/20
60000/60000 [==============================] - 10s 168us/step - loss:
0.0361 - acc: 0.9881 - val_loss: 0.0669 - val_acc: 0.9809
Epoch 9/20
60000/60000 [==============================] - 10s 170us/step - loss:
0.0334 - acc: 0.9892 - val_loss: 0.0615 - val_acc: 0.9825
Epoch 10/20
60000/60000 [==============================] - 10s 175us/step - loss:
0.0297 - acc: 0.9904 - val_loss: 0.0744 - val_acc: 0.9790
Epoch 11/20
60000/60000 [==============================] - 10s 175us/step - loss:
0.0275 - acc: 0.9912 - val_loss: 0.0670 - val_acc: 0.9811
Epoch 12/20
60000/60000 [==============================] - 14s 225us/step - loss:
0.0249 - acc: 0.9919 - val_loss: 0.0621 - val_acc: 0.9830
Epoch 13/20
60000/60000 [==============================] - 12s 208us/step - loss:
0.0253 - acc: 0.9916 - val_loss: 0.0779 - val_acc: 0.9799
Epoch 14/20
60000/60000 [==============================] - 10s 175us/step - loss:
0.0237 - acc: 0.9920 - val_loss: 0.0651 - val_acc: 0.9819
Epoch 15/20
60000/60000 [==============================] - 11s 178us/step - loss:
0.0211 - acc: 0.9927 - val_loss: 0.0681 - val_acc: 0.9821
Epoch 16/20
60000/60000 [==============================] - 11s 178us/step - loss:
0.0177 - acc: 0.9941 - val_loss: 0.0698 - val_acc: 0.9824
Epoch 17/20
60000/60000 [==============================] - 11s 190us/step - loss:
0.0176 - acc: 0.9939 - val_loss: 0.0668 - val_acc: 0.9822
Epoch 18/20
60000/60000 [==============================] - 11s 179us/step - loss:
0.0196 - acc: 0.9933 - val_loss: 0.0696 - val_acc: 0.9827
Epoch 19/20
60000/60000 [==============================] - 11s 182us/step - loss:
0.0170 - acc: 0.9941 - val_loss: 0.0698 - val_acc: 0.9831
Epoch 20/20
60000/60000 [==============================] - 11s 176us/step - loss:
0.0170 - acc: 0.9943 - val_loss: 0.0638 - val_acc: 0.9837
```

Results:

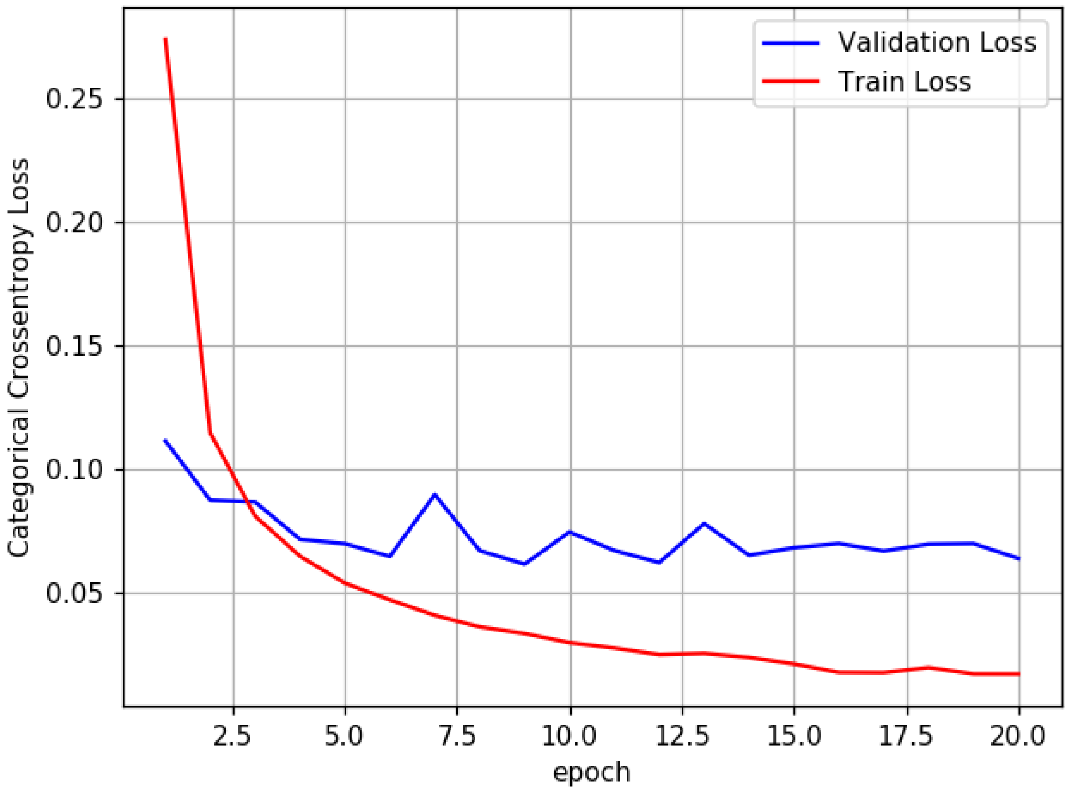
1. Accuracy= 99.43%

Plotting each Epoch vs Loss

In [18]:

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

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

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


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

```
Test score: 0.06381631516393027
Test accuracy: 0.9837
```



MLP + Dropout(0.7) + AdamOptimizer

In [19]:
```
model_drop = Sequential()

model_drop.add(Dense(364, activation='relu', input_shape=(input_dim,),
kernel_initializer=he_normal(seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.7))

model_drop.add(Dense(150, activation='relu', kernel_initializer=he_norm
```

```
al(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.7))

model_drop.add(Dense(70, activation='relu', kernel_initializer=he_norma
l(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.7))

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


model_drop.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epoch
s=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

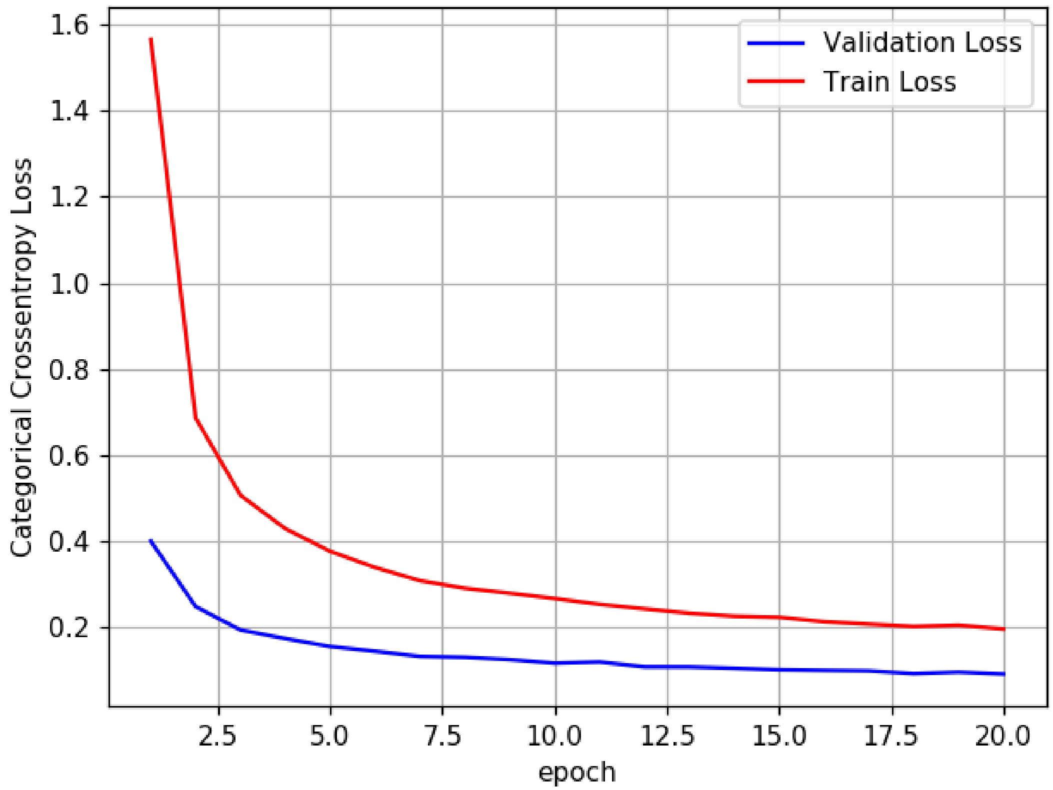
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 13s 214us/step - loss:
1.5645 - acc: 0.5105 - val_loss: 0.4003 - val_acc: 0.9014
Epoch 2/20
60000/60000 [==============================] - 10s 161us/step - loss:
0.6860 - acc: 0.7904 - val_loss: 0.2483 - val_acc: 0.9290
Epoch 3/20
60000/60000 [==============================] - 10s 170us/step - loss:
0.5067 - acc: 0.8556 - val_loss: 0.1941 - val_acc: 0.9427
Epoch 4/20
60000/60000 [==============================] - 10s 168us/step - loss:
0.4291 - acc: 0.8845 - val_loss: 0.1741 - val_acc: 0.9496
Epoch 5/20
60000/60000 [==============================] - 10s 168us/step - loss:
0.3767 - acc: 0.8991 - val_loss: 0.1557 - val_acc: 0.9545
Epoch 6/20
60000/60000 [==============================] - 10s 168us/step - loss:
0.3392 - acc: 0.9112 - val_loss: 0.1447 - val_acc: 0.9592
Epoch 7/20
60000/60000 [==============================] - 10s 164us/step - loss:
0.3086 - acc: 0.9199 - val_loss: 0.1329 - val_acc: 0.9616
```

```
Epoch 8/20
60000/60000 [==============================] - 10s 168us/step - loss:
0.2910 - acc: 0.9240 - val_loss: 0.1308 - val_acc: 0.9626
Epoch 9/20
60000/60000 [==============================] - 10s 174us/step - loss:
0.2795 - acc: 0.9286 - val_loss: 0.1253 - val_acc: 0.9660
Epoch 10/20
60000/60000 [==============================] - 10s 171us/step - loss:
0.2672 - acc: 0.9303 - val_loss: 0.1170 - val_acc: 0.9674
Epoch 11/20
60000/60000 [==============================] - 10s 170us/step - loss:
0.2537 - acc: 0.9345 - val_loss: 0.1198 - val_acc: 0.9664
Epoch 12/20
60000/60000 [==============================] - 10s 172us/step - loss:
0.2431 - acc: 0.9382 - val_loss: 0.1086 - val_acc: 0.9698
Epoch 13/20
60000/60000 [==============================] - 11s 181us/step - loss:
0.2331 - acc: 0.9403 - val_loss: 0.1083 - val_acc: 0.9711
Epoch 14/20
60000/60000 [==============================] - 11s 176us/step - loss:
0.2262 - acc: 0.9421 - val_loss: 0.1053 - val_acc: 0.9707
Epoch 15/20
60000/60000 [==============================] - 10s 170us/step - loss:
0.2237 - acc: 0.9431 - val_loss: 0.1018 - val_acc: 0.9716
Epoch 16/20
60000/60000 [==============================] - 10s 174us/step - loss:
0.2135 - acc: 0.9457 - val_loss: 0.1003 - val_acc: 0.9723
Epoch 17/20
60000/60000 [==============================] - 10s 173us/step - loss:
0.2080 - acc: 0.9479 - val_loss: 0.0992 - val_acc: 0.9741
Epoch 18/20
60000/60000 [==============================] - 10s 172us/step - loss:
0.2023 - acc: 0.9486 - val_loss: 0.0928 - val_acc: 0.9747
Epoch 19/20
60000/60000 [==============================] - 10s 170us/step - loss:
0.2053 - acc: 0.9486 - val_loss: 0.0963 - val_acc: 0.9745
Epoch 20/20
60000/60000 [==============================] - 10s 174us/step - loss:
0.1966 - acc: 0.9494 - val_loss: 0.0919 - val_acc: 0.9748
```

Plotting each Epoch vs Loss

In [20]:
```
score = model_drop.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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


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

```
Test score: 0.091905259068124
Test accuracy: 0.9748
```



Model 3:----> 5 Hidden Layers

MLP + ReLU activation + ADAMOptimizer

In [24]:
```
model_relu = Sequential()

model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(seed=None)))
```

```
model_relu.add(Dense(350, activation='relu', kernel_initializer=he_norm
al(seed=None)) )
model_relu.add(Dense(230, activation='relu', kernel_initializer=he_norm
al(seed=None)) )
model_relu.add(Dense(145, activation='relu', kernel_initializer=he_norm
al(seed=None)) )
model_relu.add(Dense(64, activation='relu', kernel_initializer=he_norma
l(seed=None)) )

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


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

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 18s 292us/step - loss:
0.2215 - acc: 0.9331 - val_loss: 0.1031 - val_acc: 0.9682
Epoch 2/20
60000/60000 [==============================] - 15s 252us/step - loss:
0.0883 - acc: 0.9730 - val_loss: 0.0971 - val_acc: 0.9702
Epoch 3/20
60000/60000 [==============================] - 15s 254us/step - loss:
0.0622 - acc: 0.9802 - val_loss: 0.0762 - val_acc: 0.9764
Epoch 4/20
60000/60000 [==============================] - 15s 257us/step - loss:
0.0462 - acc: 0.9850 - val_loss: 0.1113 - val_acc: 0.9670
Epoch 5/20
60000/60000 [==============================] - 15s 255us/step - loss:
0.0380 - acc: 0.9881 - val_loss: 0.0800 - val_acc: 0.9776
Epoch 6/20
60000/60000 [==============================] - 15s 255us/step - loss:
0.0302 - acc: 0.9904 - val_loss: 0.0881 - val_acc: 0.9769
Epoch 7/20
60000/60000 [==============================] - 16s 260us/step - loss:
0.0286 - acc: 0.9909 - val_loss: 0.0882 - val_acc: 0.9778
```

```
Epoch 8/20
60000/60000 [==============================] - 15s 249us/step - loss:
0.0240 - acc: 0.9922 - val_loss: 0.0919 - val_acc: 0.9778 loss: 0.
Epoch 9/20
60000/60000 [==============================] - 15s 253us/step - loss:
0.0232 - acc: 0.9931 - val_loss: 0.0877 - val_acc: 0.9803
Epoch 10/20
60000/60000 [==============================] - 15s 252us/step - loss:
0.0200 - acc: 0.9936 - val_loss: 0.0727 - val_acc: 0.9810
Epoch 11/20
60000/60000 [==============================] - 15s 254us/step - loss:
0.0182 - acc: 0.9944 - val_loss: 0.0785 - val_acc: 0.9804
Epoch 12/20
60000/60000 [==============================] - 15s 252us/step - loss:
0.0166 - acc: 0.9948 - val_loss: 0.1036 - val_acc: 0.9741
Epoch 13/20
60000/60000 [==============================] - 15s 253us/step - loss:
0.0150 - acc: 0.9955 - val_loss: 0.0988 - val_acc: 0.9766
Epoch 14/20
60000/60000 [==============================] - 15s 256us/step - loss:
0.0166 - acc: 0.9953 - val_loss: 0.1014 - val_acc: 0.9781
Epoch 15/20
60000/60000 [==============================] - 15s 255us/step - loss:
0.0138 - acc: 0.9959 - val_loss: 0.0999 - val_acc: 0.9785
Epoch 16/20
60000/60000 [==============================] - 15s 258us/step - loss:
0.0130 - acc: 0.9961 - val_loss: 0.0887 - val_acc: 0.9789
Epoch 17/20
60000/60000 [==============================] - 15s 257us/step - loss:
0.0122 - acc: 0.9962 - val_loss: 0.0973 - val_acc: 0.9792
Epoch 18/20
60000/60000 [==============================] - 16s 266us/step - loss:
0.0121 - acc: 0.9962 - val_loss: 0.0907 - val_acc: 0.9823
Epoch 19/20
60000/60000 [==============================] - 15s 258us/step - loss:
0.0113 - acc: 0.9965 - val_loss: 0.1023 - val_acc: 0.9822
Epoch 20/20
60000/60000 [==============================] - 15s 256us/step - loss:
0.0093 - acc: 0.9972 - val_loss: 0.0759 - val_acc: 0.9842
```

Results:

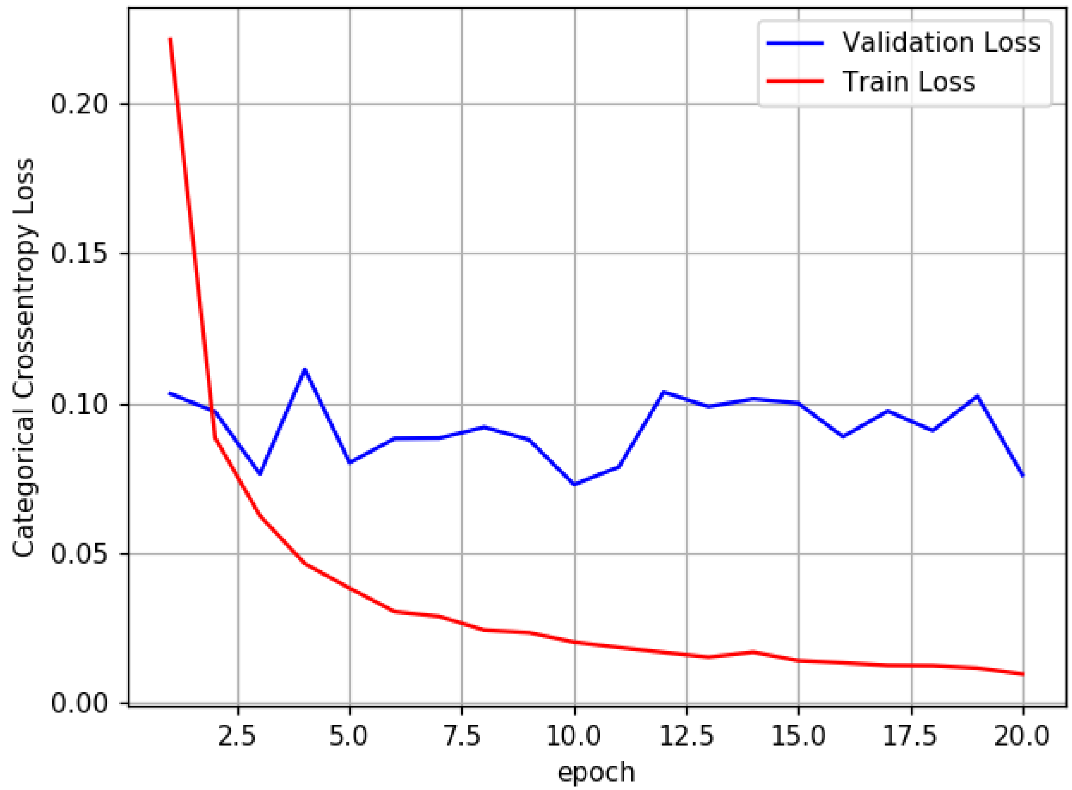
1. Train Accuracy= 99.72%

Plotting each Epoch vs Loss

In [25]:

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

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

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



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

```
Test score: 0.07591210639261503
Test accuracy: 0.9842
```



MLP + Batch-Norm on hidden Layers + AdamOptimizer

In [26]:
```
model_batch = Sequential()

model_batch.add(Dense(512, activation='relu', input_shape=(input_dim,),
 kernel_initializer=he_normal(seed=None)))
model_batch.add(BatchNormalization())
model_batch.add(Dense(350, activation='relu', kernel_initializer=he_nor
mal(seed=None)) )
```

```
model_batch.add(BatchNormalization())
model_batch.add(Dense(230, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())
model_batch.add(Dense(145, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())
model_batch.add(Dense(64, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())
model_batch.add(Dense(output_dim, activation='softmax'))


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

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 24s 394us/step - loss: 0.2161 - acc: 0.9365 - val_loss: 0.1082 - val_acc: 0.9670
Epoch 2/20
60000/60000 [==============================] - 18s 292us/step - loss: 0.0830 - acc: 0.9746 - val_loss: 0.0832 - val_acc: 0.9718
Epoch 3/20
60000/60000 [==============================] - 17s 281us/step - loss: 0.0602 - acc: 0.9808 - val_loss: 0.0904 - val_acc: 0.9730
Epoch 4/20
60000/60000 [==============================] - 17s 281us/step - loss: 0.0494 - acc: 0.9839 - val_loss: 0.0814 - val_acc: 0.9745
Epoch 5/20
60000/60000 [==============================] - 17s 286us/step - loss: 0.0389 - acc: 0.9874 - val_loss: 0.1550 - val_acc: 0.9535
Epoch 6/20
60000/60000 [==============================] - 17s 281us/step - loss: 0.0358 - acc: 0.9883 - val_loss: 0.0828 - val_acc: 0.9757
Epoch 7/20
60000/60000 [==============================] - 17s 282us/step - loss:
```

```
0.0296 - acc: 0.9903 - val_loss: 0.0739 - val_acc: 0.9793
Epoch 8/20
60000/60000 [==============================] - 17s 282us/step - loss:
0.0280 - acc: 0.9906 - val_loss: 0.0827 - val_acc: 0.9775
Epoch 9/20
60000/60000 [==============================] - 17s 284us/step - loss:
0.0270 - acc: 0.9910 - val_loss: 0.0776 - val_acc: 0.9790
Epoch 10/20
60000/60000 [==============================] - 17s 287us/step - loss:
0.0228 - acc: 0.9923 - val_loss: 0.0829 - val_acc: 0.9782
Epoch 11/20
60000/60000 [==============================] - 17s 283us/step - loss:
0.0219 - acc: 0.9931 - val_loss: 0.0756 - val_acc: 0.9797
Epoch 12/20
60000/60000 [==============================] - 19s 310us/step - loss:
0.0204 - acc: 0.9933 - val_loss: 0.0705 - val_acc: 0.9822
Epoch 13/20
60000/60000 [==============================] - 17s 291us/step - loss:
0.0203 - acc: 0.9934 - val_loss: 0.0914 - val_acc: 0.9779
Epoch 14/20
60000/60000 [==============================] - 17s 291us/step - loss:
0.0162 - acc: 0.9947 - val_loss: 0.0713 - val_acc: 0.9806
Epoch 15/20
60000/60000 [==============================] - 17s 291us/step - loss:
0.0152 - acc: 0.9951 - val_loss: 0.0827 - val_acc: 0.9796
Epoch 16/20
60000/60000 [==============================] - 17s 291us/step - loss:
0.0146 - acc: 0.9948 - val_loss: 0.0827 - val_acc: 0.9787
Epoch 17/20
60000/60000 [==============================] - 18s 292us/step - loss:
0.0152 - acc: 0.9946 - val_loss: 0.0908 - val_acc: 0.9785
Epoch 18/20
60000/60000 [==============================] - 18s 293us/step - loss:
0.0130 - acc: 0.9957 - val_loss: 0.0760 - val_acc: 0.9837
Epoch 19/20
60000/60000 [==============================] - 18s 295us/step - loss:
0.0128 - acc: 0.9961 - val_loss: 0.0849 - val_acc: 0.9784
Epoch 20/20
```

```
60000/60000 [==============================] - 18s 293us/step - loss:
0.0118 - acc: 0.9961 - val_loss: 0.0800 - val_acc: 0.9808
```

Results:

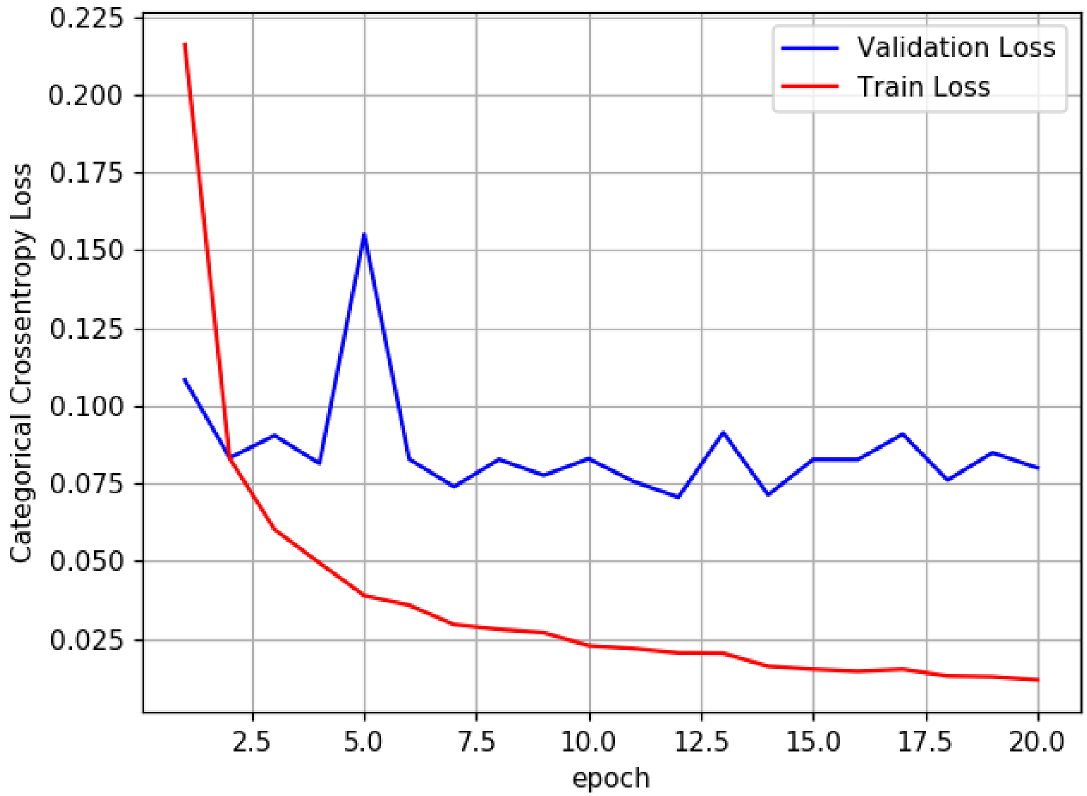
1. Train Accuracy= 99.61%

Plotting each Epoch vs Loss

In [27]:

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

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

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


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

```
Test score: 0.07999502164848526
Test accuracy: 0.9808
```



MLP + Dropout (0.5)+ AdamOptimizer

In [28]:
```
model_drop = Sequential()

model_drop.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(350, activation='relu', kernel_initializer=he_norm
```

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

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

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

model_drop.add(Dense(64, activation='relu', kernel_initializer=he_norma
l(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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


model_drop.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epoch
s=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 23s 385us/step - loss:
1.1365 - acc: 0.6434 - val_loss: 0.2582 - val_acc: 0.9255
Epoch 2/20
60000/60000 [==============================] - 19s 320us/step - loss:
0.3932 - acc: 0.8887 - val_loss: 0.1707 - val_acc: 0.9523
Epoch 3/20
60000/60000 [==============================] - 19s 321us/step - loss:
0.2857 - acc: 0.9224 - val_loss: 0.1386 - val_acc: 0.9625
Epoch 4/20
60000/60000 [==============================] - 20s 333us/step - loss:
```

```
0.2380 - acc: 0.9370 - val_loss: 0.1252 - val_acc: 0.9647
Epoch 5/20
60000/60000 [==============================] - 19s 321us/step - loss:
0.2131 - acc: 0.9448 - val_loss: 0.1101 - val_acc: 0.9711
Epoch 6/20
60000/60000 [==============================] - 19s 324us/step - loss:
0.1929 - acc: 0.9496 - val_loss: 0.1078 - val_acc: 0.9720
Epoch 7/20
60000/60000 [==============================] - 19s 321us/step - loss:
0.1778 - acc: 0.9538 - val_loss: 0.0945 - val_acc: 0.9754
Epoch 8/20
60000/60000 [==============================] - 19s 324us/step - loss:
0.1631 - acc: 0.9562 - val_loss: 0.0941 - val_acc: 0.9753
Epoch 9/20
60000/60000 [==============================] - 19s 324us/step - loss:
0.1555 - acc: 0.9594 - val_loss: 0.0845 - val_acc: 0.9779
Epoch 10/20
60000/60000 [==============================] - 20s 332us/step - loss:
0.1455 - acc: 0.9621 - val_loss: 0.0888 - val_acc: 0.9773
Epoch 11/20
60000/60000 [==============================] - 19s 323us/step - loss:
0.1391 - acc: 0.9642 - val_loss: 0.0817 - val_acc: 0.9775
Epoch 12/20
60000/60000 [==============================] - 19s 322us/step - loss:
0.1320 - acc: 0.9664 - val_loss: 0.0794 - val_acc: 0.9791
Epoch 13/20
60000/60000 [==============================] - 20s 326us/step - loss:
0.1218 - acc: 0.9684 - val_loss: 0.0741 - val_acc: 0.9809
Epoch 14/20
60000/60000 [==============================] - 19s 324us/step - loss:
0.1224 - acc: 0.9691 - val_loss: 0.0740 - val_acc: 0.9804
Epoch 15/20
60000/60000 [==============================] - 19s 323us/step - loss:
0.1105 - acc: 0.9713 - val_loss: 0.0785 - val_acc: 0.9806
Epoch 16/20
60000/60000 [==============================] - 20s 329us/step - loss:
0.1099 - acc: 0.9720 - val_loss: 0.0711 - val_acc: 0.9815
Epoch 17/20
60000/60000 [==============================] - 19s 325us/step - loss:
```

```
0.1069 - acc: 0.9721 - val_loss: 0.0702 - val_acc: 0.9811
Epoch 18/20
60000/60000 [==============================] - 20s 331us/step - loss:
0.1011 - acc: 0.9735 - val_loss: 0.0770 - val_acc: 0.9800
Epoch 19/20
60000/60000 [==============================] - 20s 330us/step - loss:
0.0989 - acc: 0.9741 - val_loss: 0.0696 - val_acc: 0.9812
Epoch 20/20
60000/60000 [==============================] - 19s 323us/step - loss:
0.0954 - acc: 0.9751 - val_loss: 0.0709 - val_acc: 0.9824
```

Results:

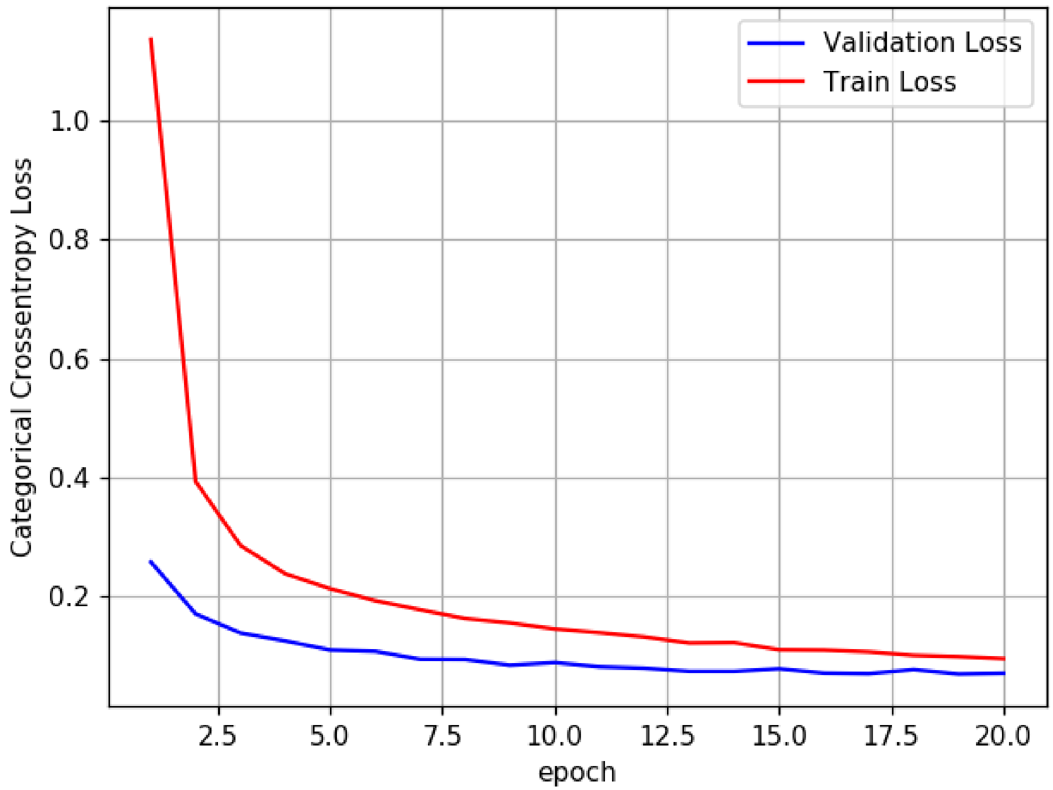
1. Train Accuracy= 97.51%

Plotting each Epoch vs Loss

In [29]:

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

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

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

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

```
Test score: 0.07089627963798 12
Test accuracy: 0.9824
```



MLP + Dropout (0.1)+ AdamOptimizer

In [21]:
```
model_drop = Sequential()

model_drop.add(Dense(512, activation='relu', input_shape=(input_dim,),
kernel_initializer=he_normal(seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.1))

model_drop.add(Dense(350, activation='relu', kernel_initializer=he_norm
```

```
al(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.1))

model_drop.add(Dense(230, activation='relu', kernel_initializer=he_norm
al(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.1))

model_drop.add(Dense(145, activation='relu', kernel_initializer=he_norm
al(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.1))

model_drop.add(Dense(64, activation='relu', kernel_initializer=he_norma
l(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.1))

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


model_drop.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epoch
s=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 24s 404us/step - loss:
0.2872 - acc: 0.9145 - val_loss: 0.1030 - val_acc: 0.9681
Epoch 2/20
60000/60000 [==============================] - 20s 328us/step - loss:
0.1203 - acc: 0.9640 - val_loss: 0.1022 - val_acc: 0.9692
Epoch 3/20
60000/60000 [==============================] - 19s 323us/step - loss:
0.0906 - acc: 0.9723 - val_loss: 0.0835 - val_acc: 0.9748
Epoch 4/20
60000/60000 [==============================] - 19s 324us/step - loss:
```

```
0.0722 - acc: 0.9779 - val_loss: 0.0843 - val_acc: 0.9752
Epoch 5/20
60000/60000 [==============================] - 21s 346us/step - loss:
0.0620 - acc: 0.9806 - val_loss: 0.0723 - val_acc: 0.9793
Epoch 6/20
60000/60000 [==============================] - 20s 325us/step - loss:
0.0549 - acc: 0.9826 - val_loss: 0.0737 - val_acc: 0.9792
Epoch 7/20
60000/60000 [==============================] - 19s 325us/step - loss:
0.0494 - acc: 0.9845 - val_loss: 0.0717 - val_acc: 0.9789
Epoch 8/20
60000/60000 [==============================] - 20s 327us/step - loss:
0.0424 - acc: 0.9867 - val_loss: 0.0682 - val_acc: 0.9823
Epoch 9/20
60000/60000 [==============================] - 19s 324us/step - loss:
0.0389 - acc: 0.9873 - val_loss: 0.0731 - val_acc: 0.9799
Epoch 10/20
60000/60000 [==============================] - 19s 323us/step - loss:
0.0380 - acc: 0.9877 - val_loss: 0.0734 - val_acc: 0.9804
Epoch 11/20
60000/60000 [==============================] - 19s 324us/step - loss:
0.0324 - acc: 0.9895 - val_loss: 0.0656 - val_acc: 0.9815
Epoch 12/20
60000/60000 [==============================] - 19s 325us/step - loss:
0.0313 - acc: 0.9899 - val_loss: 0.0739 - val_acc: 0.9781
Epoch 13/20
60000/60000 [==============================] - 19s 323us/step - loss:
0.0284 - acc: 0.9908 - val_loss: 0.0697 - val_acc: 0.9812
Epoch 14/20
60000/60000 [==============================] - 19s 323us/step - loss:
0.0293 - acc: 0.9909 - val_loss: 0.0732 - val_acc: 0.9801
Epoch 15/20
60000/60000 [==============================] - 20s 326us/step - loss:
0.0257 - acc: 0.9918 - val_loss: 0.0728 - val_acc: 0.9815
Epoch 16/20
60000/60000 [==============================] - 20s 328us/step - loss:
0.0225 - acc: 0.9927 - val_loss: 0.0755 - val_acc: 0.9811
Epoch 17/20
60000/60000 [==============================] - 20s 326us/step - loss:
```

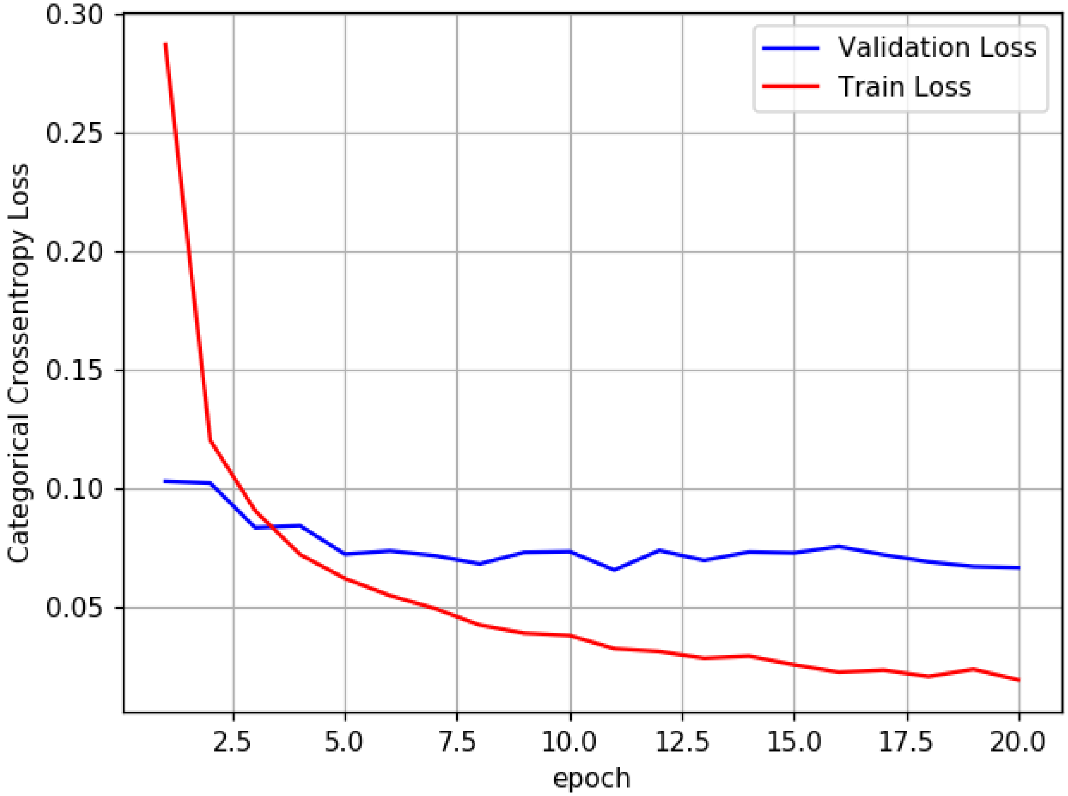
```
0.0234 - acc: 0.9926 - val_loss: 0.0720 - val_acc: 0.9825
Epoch 18/20
60000/60000 [==============================] - 20s 326us/step - loss:
0.0207 - acc: 0.9931 - val_loss: 0.0691 - val_acc: 0.9817
Epoch 19/20
60000/60000 [==============================] - 20s 327us/step - loss:
0.0238 - acc: 0.9929 - val_loss: 0.0671 - val_acc: 0.9832
Epoch 20/20
60000/60000 [==============================] - 20s 331us/step - loss:
0.0193 - acc: 0.9940 - val_loss: 0.0666 - val_acc: 0.9837
```

Plotting each Epoch vs Loss

In [22]:

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

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

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

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

```
Test score: 0.06661306326177437
Test accuracy: 0.9837
```



MLP + Dropout (0.7)+ AdamOptimizer

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

model_drop.add(Dense(512, activation='relu', input_shape=(input_dim,),
kernel_initializer=he_normal(seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.7))

model_drop.add(Dense(350, activation='relu', kernel_initializer=he_norm
```

```
al(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.7))

model_drop.add(Dense(230, activation='relu', kernel_initializer=he_norm
al(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.7))

model_drop.add(Dense(145, activation='relu', kernel_initializer=he_norm
al(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.7))

model_drop.add(Dense(64, activation='relu', kernel_initializer=he_norma
l(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.7))

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


model_drop.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epoch
s=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 25s 413us/step - loss:
2.4775 - acc: 0.2108 - val_loss: 1.6731 - val_acc: 0.3405
Epoch 2/20
60000/60000 [==============================] - 21s 348us/step - loss:
1.4687 - acc: 0.4547 - val_loss: 0.8386 - val_acc: 0.7370
Epoch 3/20
60000/60000 [==============================] - 21s 347us/step - loss:
1.0088 - acc: 0.6378 - val_loss: 0.5046 - val_acc: 0.8555
Epoch 4/20
60000/60000 [==============================] - 21s 356us/step - loss:
```

```
0.7767 - acc: 0.7416 - val_loss: 0.3513 - val_acc: 0.9134
Epoch 5/20
60000/60000 [==============================] - 21s 343us/step - loss:
0.6363 - acc: 0.8077 - val_loss: 0.2584 - val_acc: 0.9343
Epoch 6/20
60000/60000 [==============================] - 20s 336us/step - loss:
0.5368 - acc: 0.8471 - val_loss: 0.2157 - val_acc: 0.9464
Epoch 7/20
60000/60000 [==============================] - 20s 337us/step - loss:
0.4730 - acc: 0.8754 - val_loss: 0.1895 - val_acc: 0.9517
Epoch 8/20
60000/60000 [==============================] - 21s 344us/step - loss:
0.4182 - acc: 0.8947 - val_loss: 0.1649 - val_acc: 0.9577
Epoch 9/20
60000/60000 [==============================] - 20s 336us/step - loss:
0.3824 - acc: 0.9063 - val_loss: 0.1598 - val_acc: 0.9601
Epoch 10/20
60000/60000 [==============================] - 20s 339us/step - loss:
0.3534 - acc: 0.9148 - val_loss: 0.1423 - val_acc: 0.9631
Epoch 11/20
60000/60000 [==============================] - 21s 343us/step - loss:
0.3420 - acc: 0.9179 - val_loss: 0.1436 - val_acc: 0.9631
Epoch 12/20
60000/60000 [==============================] - 20s 337us/step - loss:
0.3192 - acc: 0.9255 - val_loss: 0.1379 - val_acc: 0.9656
Epoch 13/20
60000/60000 [==============================] - 21s 348us/step - loss:
0.3083 - acc: 0.9289 - val_loss: 0.1298 - val_acc: 0.9694
Epoch 14/20
60000/60000 [==============================] - 20s 336us/step - loss:
0.2885 - acc: 0.9326 - val_loss: 0.1262 - val_acc: 0.9697
Epoch 15/20
60000/60000 [==============================] - 20s 332us/step - loss:
0.2780 - acc: 0.9369 - val_loss: 0.1230 - val_acc: 0.9711
Epoch 16/20
60000/60000 [==============================] - 20s 339us/step - loss:
0.2724 - acc: 0.9382 - val_loss: 0.1170 - val_acc: 0.9728
Epoch 17/20
60000/60000 [==============================] - 20s 335us/step - loss:
```

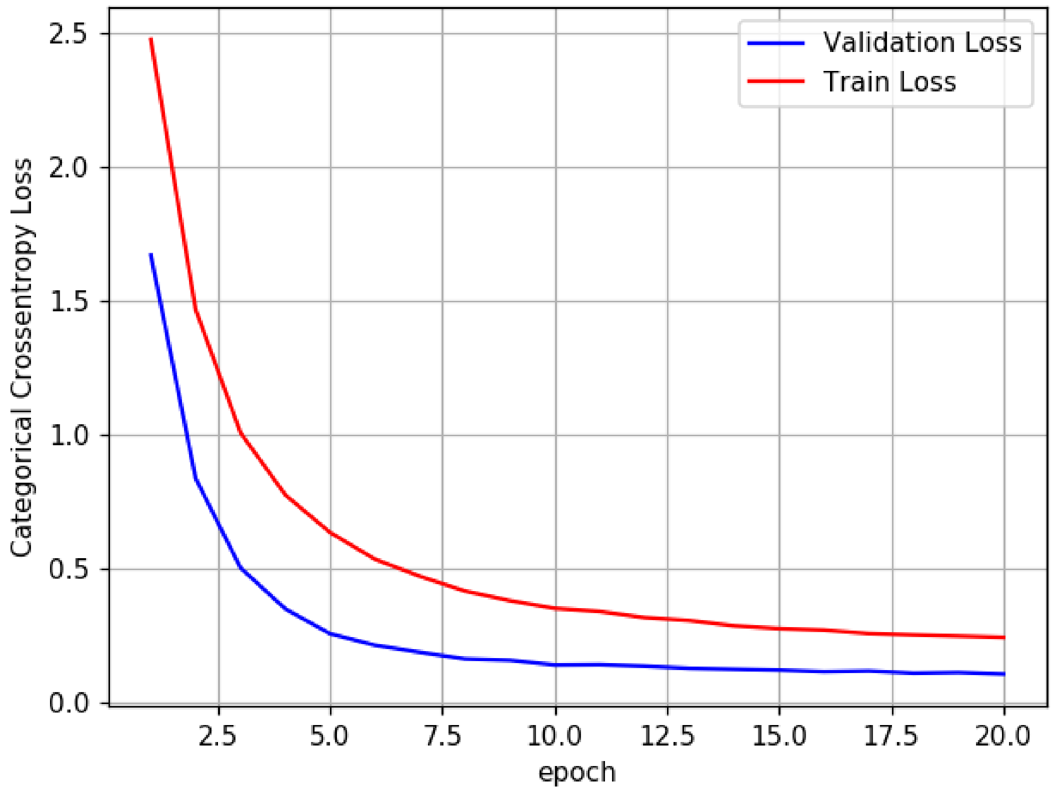
```
0.2595 - acc: 0.9420 - val_loss: 0.1197 - val_acc: 0.9706
Epoch 18/20
60000/60000 [==============================] - 20s 336us/step - loss:
0.2542 - acc: 0.9419 - val_loss: 0.1116 - val_acc: 0.9741
Epoch 19/20
60000/60000 [==============================] - 20s 336us/step - loss:
0.2499 - acc: 0.9440 - val_loss: 0.1141 - val_acc: 0.9728
Epoch 20/20
60000/60000 [==============================] - 20s 337us/step - loss:
0.2452 - acc: 0.9444 - val_loss: 0.1083 - val_acc: 0.9750
```

Plotting each Epoch vs Loss

In [24]:

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

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

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

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

```
Test score: 0.10832664410285651
Test accuracy: 0.975
```



Pretty Table

In [25]:

```
# Please compare all your models using Prettytable library
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable

#If you get a ModuleNotFoundError error , install prettytable using: pi
p3 install prettytable
```

```python
x = PrettyTable()
x.field_names = ["No. of Hidden Layers Used", "Activation Unit","Optimiser", "Batch Normalisation", "DropOuts"," Train Accuracy","Test_Accuracy"]

x.add_row(["2", "ReLU","Adam","No","No" ,"99.83%", "98.05%"])
x.add_row(["2", "ReLU","Adam","Yes","No" ,"99.76%", "97.93%"])
x.add_row(["2", "ReLU","Adam","Yes","0.5" ,"97.87%", "98.11%"])
x.add_row(["2", "ReLU","Adam","Yes","0.1" ,"99.57%", "98.3%"])
x.add_row(["2", "ReLU","Adam","Yes","0.7" ,"95.58%", "97.5%"])

x.add_row(["3", "ReLU","Adam","No","No", "99.72%", "97.99%"])
x.add_row(["3", "ReLU","Adam","Yes","No" ,"99.74%", "98.01%"])
x.add_row(["3", "ReLU","Adam","Yes","0.5","97.65%", "98.09%"])
x.add_row(["2", "ReLU","Adam","Yes","0.1" ,"99.43%", "98.37%"])
x.add_row(["2", "ReLU","Adam","Yes","0.7" ,"94.94%", "97.48%"])

x.add_row(["5", "ReLU","Adam","No","No" ,"99.72%", "98.42%"])
x.add_row(["5", "ReLU","Adam","Yes","No" ,"99.61%", "98.08%"])
x.add_row(["5", "ReLU","Adam","Yes","0.5" ,"97.51%", "98.24%"])
x.add_row(["2", "ReLU","Adam","Yes","0.1" ,"99.4%", "98.37%"])
x.add_row(["2", "ReLU","Adam","Yes","0.7" ,"94.44%", "97.5%"])


print(x)
```

```
+---------------------------+-----------------+-----------+---------------------+----------+-----------------+---------------+
| No. of Hidden Layers Used | Activation Unit | Optimiser | Batch Normalisation | DropOuts |  Train Accuracy | Test_Accuracy |
+---------------------------+-----------------+-----------+---------------------+----------+-----------------+---------------+
|             2             |       ReLU      |    Adam   |          No         |    No    |      99.83%     |     98.05%    |
|             2             |       ReLU      |    Adam   |         Yes         |    No    |      99.76%     |     97.93%    |
|             2             |       ReLU      |    Adam   |         Yes         |   0.5    |      97.87%     |     98.11%    |
|             2             |       ReLU      |    Adam   |         Yes         |   0.1    |      99.57%     |     98.3%     |
```

2	ReLU	Adam	Yes	0.7	95.58%	97.5%
3	ReLU	Adam	No	No	99.72%	97.99%
3	ReLU	Adam	Yes	No	99.74%	98.01%
3	ReLU	Adam	Yes	0.5	97.65%	98.09%
2	ReLU	Adam	Yes	0.1	99.43%	98.37%
2	ReLU	Adam	Yes	0.7	94.94%	97.48%
5	ReLU	Adam	No	No	99.72%	98.42%
5	ReLU	Adam	Yes	No	99.61%	98.08%
5	ReLU	Adam	Yes	0.5	97.51%	98.24%
2	ReLU	Adam	Yes	0.1	99.4%	98.37%
2	ReLU	Adam	Yes	0.7	94.44%	97.5%

Conclusion:

1. We can see that using Batch Normalization and DropOuts gave better Accuracy.
2. Batch Normalization helps in Faster Convergence , since it prevents Internal Covariance Shift
3. We are further avoiding Overfitting by using randomisation as Regularisation i.e. DROPOUTS.
4. The best results are seen while using Batch Normalisation and Dropouts together.

DropOut Rates

1. In Model 1: Dropout=0.5 , it converged well

```
        Dropout=0.1 , didn't converge well, Overfitted
        DropOut=0.7 , converged but not better than Dropou
t=0.5
```

2. In Model 2: Dropout=0.5 , it converged well

```
        Dropout=0.1 , didn't converge well, Overfitted
        DropOut=0.7 , converged but not better than Dropou
t=0.5
```

3. In Model 3: Dropout=0.5 , it converged well

```
        Dropout=0.1 , didn't converge well, Overfitted
        DropOut=0.7 , converged but not better than Dropou
t=0.5
```

Model 3 gave a better results than Model 2 followed by Model 1 , in terms of Accuracy and Loss vs Epoch Curves