In [2]:
```
# if you keras is not using tensorflow as backend set "KERAS_BACKEND=tensorflow" use this command
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is dep
recated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [0]:
```
%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.grid()

    fig.canvas.draw()
```

In [4]:
```
# the data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

In [5]:
```
print("Number of training examples :", X_train.shape[0], "and each image is of shape (%d, %d)"%(X_train.shape[1], X
_train.shape[2]))
print("Number of training examples :", X_test.shape[0], "and each image is of shape (%d, %d)"%(X_test.shape[1], X_t
est.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 [0]:
```
# if you observe the input shape its 2 dimensional vector
# for each image we have a (28*28) vector
# we will convert the (28*28) vector into single dimensional vector of 1 * 784

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

In [7]:
```
# after converting the input images from 3d to 2d vectors

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

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

In [8]:
```python
# An example data point
print(X_train[0])
```

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

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

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

In [10]:
```
# example data point after normlizing
print(X_train[0])
```

```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.01176471 0.07058824 0.07058824 0.07058824
 0.49411765 0.53333333 0.68627451 0.10196078 0.65098039 1.
 0.96862745 0.49803922 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.11764706 0.14117647 0.36862745 0.60392157
 0.66666667 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.88235294 0.6745098  0.99215686 0.94901961 0.76470588 0.25098039
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.19215686
 0.93333333 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.99215686 0.99215686 0.99215686 0.98431373 0.36470588 0.32156863
 0.32156863 0.21960784 0.15294118 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.07058824 0.85882353 0.99215686
 0.99215686 0.99215686 0.99215686 0.99215686 0.77647059 0.71372549
 0.96862745 0.94509804 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.31372549 0.61176471 0.41960784 0.99215686
 0.99215686 0.80392157 0.04313725 0.         0.16862745 0.60392157
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.05490196 0.00392157 0.60392157 0.99215686 0.35294118
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.54509804 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.         0.97647059 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]:

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

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

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

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

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

Softmax classifier

In [0]:
```python
# https://keras.io/getting-started/sequential-model-guide/

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

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

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

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

###

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

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

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

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

####

# https://keras.io/activations/

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

# from keras.layers import Activation, Dense

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

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

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


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

In [0]:
```python
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

In [43]:
```python
X_train.shape[1]
```

Out[43]: 784

1. MLP 2-Hidden layer architecture

1.1. MLP + ReLU + ADAM

In [29]:

```
model_relu = Sequential()
model_relu.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_relu.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

history1 = model_relu.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_6 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_7 (Dense)              (None, 10)                2570
=================================================================
Total params: 535,818
Trainable params: 535,818
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.2122 - accuracy: 0.9362 - val_loss: 0.1199
- val_accuracy: 0.9620
Epoch 2/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.0793 - accuracy: 0.9759 - val_loss: 0.0749
- val_accuracy: 0.9760
Epoch 3/20
60000/60000 [==============================] - 6s 103us/step - loss: 0.0512 - accuracy: 0.9841 - val_loss: 0.0709
- val_accuracy: 0.9777
Epoch 4/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0336 - accuracy: 0.9893 - val_loss: 0.0733
- val_accuracy: 0.9777
Epoch 5/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0267 - accuracy: 0.9914 - val_loss: 0.0747
- val_accuracy: 0.9777
Epoch 6/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.0175 - accuracy: 0.9947 - val_loss: 0.0910
- val_accuracy: 0.9754
Epoch 7/20
60000/60000 [==============================] - 6s 104us/step - loss: 0.0170 - accuracy: 0.9942 - val_loss: 0.0713
- val_accuracy: 0.9796
Epoch 8/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0145 - accuracy: 0.9953 - val_loss: 0.0985
- val_accuracy: 0.9738
Epoch 9/20
60000/60000 [==============================] - 6s 104us/step - loss: 0.0141 - accuracy: 0.9954 - val_loss: 0.0736
- val_accuracy: 0.9815
Epoch 10/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0117 - accuracy: 0.9959 - val_loss: 0.0878
- val_accuracy: 0.9797
Epoch 11/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.0138 - accuracy: 0.9957 - val_loss: 0.0906
- val_accuracy: 0.9788
Epoch 12/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0115 - accuracy: 0.9960 - val_loss: 0.0676
- val_accuracy: 0.9839
Epoch 13/20
60000/60000 [==============================] - 6s 103us/step - loss: 0.0104 - accuracy: 0.9963 - val_loss: 0.0970
- val_accuracy: 0.9782
Epoch 14/20
60000/60000 [==============================] - 6s 104us/step - loss: 0.0099 - accuracy: 0.9965 - val_loss: 0.1023
- val_accuracy: 0.9792
Epoch 15/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0111 - accuracy: 0.9964 - val_loss: 0.0926
- val_accuracy: 0.9803
Epoch 16/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0071 - accuracy: 0.9976 - val_loss: 0.0873
- val_accuracy: 0.9805
Epoch 17/20
60000/60000 [==============================] - 6s 104us/step - loss: 0.0078 - accuracy: 0.9972 - val_loss: 0.1013
- val_accuracy: 0.9788
Epoch 18/20
60000/60000 [==============================] - 6s 104us/step - loss: 0.0096 - accuracy: 0.9968 - val_loss: 0.0973
- val_accuracy: 0.9822
Epoch 19/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0063 - accuracy: 0.9979 - val_loss: 0.1016
- val_accuracy: 0.9811
Epoch 20/20
60000/60000 [==============================] - 6s 103us/step - loss: 0.0049 - accuracy: 0.9985 - val_loss: 0.1079
- val_accuracy: 0.9796
```

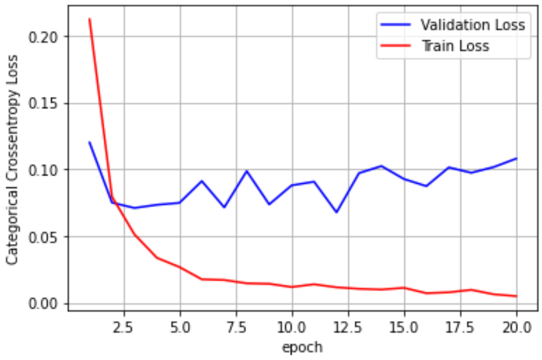
In [33]:
```
score = model_relu.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

vy = history1.history['val_loss']
ty = history1.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.10785088968896661
Test accuracy: 0.9796000123023987
```



%matplotlib inline

plt.close('all')

1.2. MLP + ReLU + ADAM + Batch Normalization + HE normal initialization

In [36]:
```
from keras.layers.normalization import BatchNormalization
from keras.initializers import he_normal

model_2 = Sequential()

model_2.add(Dense(256, activation='relu', input_shape=(input_dim,),kernel_initializer=he_normal(seed=None)))
model_2.add(Dense(128, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_2.add(BatchNormalization())

model_2.add(Dense(output_dim, activation='softmax'))


model_2.summary()

model_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history2 = model_2.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test
, Y_test))
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_11 (Dense)             (None, 256)               200960
_________________________________________________________________
dense_12 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_3 (Batch (None, 128)               512
_________________________________________________________________
dense_13 (Dense)             (None, 10)                1290
=================================================================
Total params: 235,658
Trainable params: 235,402
Non-trainable params: 256
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 58us/step - loss: 0.2228 - accuracy: 0.9351 - val_loss: 0.1234 - val_accuracy: 0.9608
Epoch 2/20
60000/60000 [==============================] - 3s 54us/step - loss: 0.0859 - accuracy: 0.9741 - val_loss: 0.0993 - val_accuracy: 0.9699
Epoch 3/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.0556 - accuracy: 0.9831 - val_loss: 0.0821 - val_accuracy: 0.9736
Epoch 4/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0405 - accuracy: 0.9874 - val_loss: 0.0836 - val_accuracy: 0.9733
Epoch 5/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0285 - accuracy: 0.9910 - val_loss: 0.0760 - val_accuracy: 0.9769
Epoch 6/20
60000/60000 [==============================] - 3s 57us/step - loss: 0.0219 - accuracy: 0.9929 - val_loss: 0.0801 - val_accuracy: 0.9769
Epoch 7/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0222 - accuracy: 0.9925 - val_loss: 0.0749 - val_accuracy: 0.9795
Epoch 8/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0179 - accuracy: 0.9940 - val_loss: 0.0730 - val_accuracy: 0.9796
Epoch 9/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0167 - accuracy: 0.9948 - val_loss: 0.0778 - val_accuracy: 0.9785
Epoch 10/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0092 - accuracy: 0.9971 - val_loss: 0.0807 - val_accuracy: 0.9790
Epoch 11/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0099 - accuracy: 0.9970 - val_loss: 0.0852 - val_accuracy: 0.9781
Epoch 12/20
60000/60000 [==============================] - 3s 58us/step - loss: 0.0161 - accuracy: 0.9949 - val_loss: 0.0730 - val_accuracy: 0.9799
Epoch 13/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0113 - accuracy: 0.9962 - val_loss: 0.0860 - val_accuracy: 0.9796
Epoch 14/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0075 - accuracy: 0.9977 - val_loss: 0.0869 - val_accuracy: 0.9793
Epoch 15/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0070 - accuracy: 0.9977 - val_loss: 0.0905 - val_accuracy: 0.9783
Epoch 16/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0073 - accuracy: 0.9978 - val_loss: 0.0867 - val_accuracy: 0.9803
Epoch 17/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0091 - accuracy: 0.9969 - val_loss: 0.1147 - val_accuracy: 0.9765
Epoch 18/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0111 - accuracy: 0.9962 - val_loss: 0.0913 - val_accuracy: 0.9778
Epoch 19/20
60000/60000 [==============================] - 3s 56us/step - loss: 0.0049 - accuracy: 0.9985 - val_loss: 0.0907 - val_accuracy: 0.9791
Epoch 20/20
60000/60000 [==============================] - 3s 55us/step - loss: 0.0066 - accuracy: 0.9978 - val_loss: 0.1016 - val_accuracy: 0.9766
```

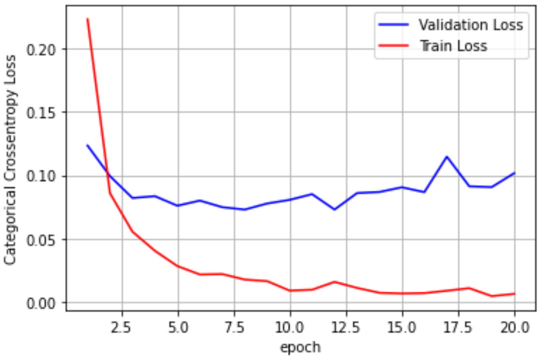
In [37]:
```
score = model_2.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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 = history2.history['val_loss']
ty = history2.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.10163859208421328
Test accuracy: 0.9765999913215637
```



1.3. MLP + ReLU + ADAM + Batch Normalization + Dropout + HE normal initialization

In [39]:

```
from keras.layers.normalization import BatchNormalization
from keras.initializers import he_normal
from keras.layers import Dropout

model_3 = Sequential()

model_3.add(Dense(512, activation='relu', input_shape=(input_dim,),kernel_initializer=he_normal(seed=None)))
model_3.add(Dropout(0.5))

model_3.add(Dense(384, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_3.add(Dropout(0.5))

model_3.add(BatchNormalization())

model_3.add(Dense(output_dim, activation='softmax'))


model_3.summary()

model_3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history3 = model_3.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test
, Y_test))
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_15 (Dense)             (None, 512)               401920    
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_16 (Dense)             (None, 384)               196992    
_________________________________________________________________
dropout_2 (Dropout)          (None, 384)               0         
_________________________________________________________________
batch_normalization_4 (Batch (None, 384)               1536      
_________________________________________________________________
dense_17 (Dense)             (None, 10)                3850      
=================================================================
Total params: 604,298
Trainable params: 603,530
Non-trainable params: 768
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.4228 - accuracy: 0.8695 - val_loss: 0.1439
- val_accuracy: 0.9581
Epoch 2/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.1880 - accuracy: 0.9439 - val_loss: 0.0995
- val_accuracy: 0.9707
Epoch 3/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.1437 - accuracy: 0.9567 - val_loss: 0.0856
- val_accuracy: 0.9728
Epoch 4/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.1225 - accuracy: 0.9618 - val_loss: 0.0757
- val_accuracy: 0.9772
Epoch 5/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.1061 - accuracy: 0.9678 - val_loss: 0.0780
- val_accuracy: 0.9756
Epoch 6/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0988 - accuracy: 0.9699 - val_loss: 0.0666
- val_accuracy: 0.9797
Epoch 7/20
60000/60000 [==============================] - 8s 140us/step - loss: 0.0872 - accuracy: 0.9723 - val_loss: 0.0653
- val_accuracy: 0.9805
Epoch 8/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0814 - accuracy: 0.9744 - val_loss: 0.0651
- val_accuracy: 0.9805
Epoch 9/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0767 - accuracy: 0.9762 - val_loss: 0.0605
- val_accuracy: 0.9816
Epoch 10/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0701 - accuracy: 0.9774 - val_loss: 0.0641
- val_accuracy: 0.9822
Epoch 11/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0676 - accuracy: 0.9786 - val_loss: 0.0593
- val_accuracy: 0.9830
Epoch 12/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0648 - accuracy: 0.9792 - val_loss: 0.0596
- val_accuracy: 0.9824
Epoch 13/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0570 - accuracy: 0.9817 - val_loss: 0.0569
- val_accuracy: 0.9824
Epoch 14/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0574 - accuracy: 0.9811 - val_loss: 0.0578
- val_accuracy: 0.9829
Epoch 15/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0534 - accuracy: 0.9821 - val_loss: 0.0550
- val_accuracy: 0.9839
Epoch 16/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0492 - accuracy: 0.9838 - val_loss: 0.0598
- val_accuracy: 0.9835
Epoch 17/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0499 - accuracy: 0.9836 - val_loss: 0.0585
- val_accuracy: 0.9828
Epoch 18/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0476 - accuracy: 0.9841 - val_loss: 0.0553
- val_accuracy: 0.9841
Epoch 19/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0440 - accuracy: 0.9856 - val_loss: 0.0557
- val_accuracy: 0.9850
Epoch 20/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0451 - accuracy: 0.9854 - val_loss: 0.0566
- val_accuracy: 0.9834
```

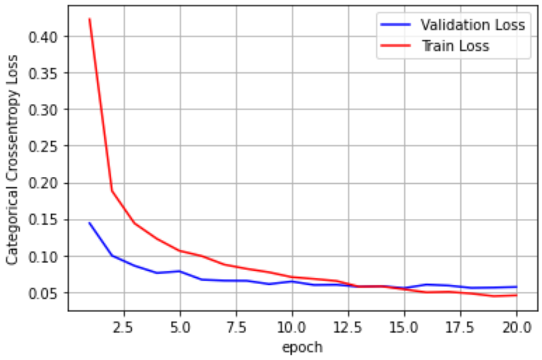
In [49]:
```
score = model_3.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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 = history3.history['val_loss']
ty = history3.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.05660308131032507
Test accuracy: 0.9833999872207642
```



2. 3-Hidden layer architecture

2.1. MLP + ReLU + ADAM

In [41]:
```
model_4 = Sequential()
model_4.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stdde
v=0.062, seed=None)))
model_4.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_4.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_4.add(Dense(output_dim, activation='softmax'))

print(model_4.summary())

model_4.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history4 = model_4.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test
, Y_test))
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_18 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_19 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_20 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_21 (Dense)             (None, 10)                1290
=================================================================
Total params: 567,434
Trainable params: 567,434
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 113us/step - loss: 0.2268 - accuracy: 0.9305 - val_loss: 0.1094
- val_accuracy: 0.9665
Epoch 2/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.0803 - accuracy: 0.9756 - val_loss: 0.0887
- val_accuracy: 0.9729
Epoch 3/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.0528 - accuracy: 0.9828 - val_loss: 0.0744
- val_accuracy: 0.9773
Epoch 4/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.0376 - accuracy: 0.9880 - val_loss: 0.0915
- val_accuracy: 0.9713
Epoch 5/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.0289 - accuracy: 0.9909 - val_loss: 0.0723
- val_accuracy: 0.9798
Epoch 6/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.0258 - accuracy: 0.9915 - val_loss: 0.0854
- val_accuracy: 0.9762
Epoch 7/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.0227 - accuracy: 0.9926 - val_loss: 0.0809
- val_accuracy: 0.9786
Epoch 8/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.0218 - accuracy: 0.9930 - val_loss: 0.0894
- val_accuracy: 0.9785
Epoch 9/20
60000/60000 [==============================] - 7s 113us/step - loss: 0.0164 - accuracy: 0.9945 - val_loss: 0.0983
- val_accuracy: 0.9749
Epoch 10/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.0164 - accuracy: 0.9947 - val_loss: 0.0867
- val_accuracy: 0.9804
Epoch 11/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.0174 - accuracy: 0.9941 - val_loss: 0.0852
- val_accuracy: 0.9791
Epoch 12/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.0139 - accuracy: 0.9956 - val_loss: 0.0852
- val_accuracy: 0.9792
Epoch 13/20
60000/60000 [==============================] - 6s 108us/step - loss: 0.0147 - accuracy: 0.9958 - val_loss: 0.1261
- val_accuracy: 0.9714
Epoch 14/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.0150 - accuracy: 0.9954 - val_loss: 0.0893
- val_accuracy: 0.9782
Epoch 15/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.0068 - accuracy: 0.9978 - val_loss: 0.0830
- val_accuracy: 0.9824
Epoch 16/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.0130 - accuracy: 0.9957 - val_loss: 0.0793
- val_accuracy: 0.9836
Epoch 17/20
60000/60000 [==============================] - 6s 108us/step - loss: 0.0118 - accuracy: 0.9964 - val_loss: 0.0808
- val_accuracy: 0.9807
Epoch 18/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.0079 - accuracy: 0.9976 - val_loss: 0.0984
- val_accuracy: 0.9799
Epoch 19/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.0149 - accuracy: 0.9953 - val_loss: 0.1010
- val_accuracy: 0.9799
Epoch 20/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.0083 - accuracy: 0.9972 - val_loss: 0.1068
- val_accuracy: 0.9775
```

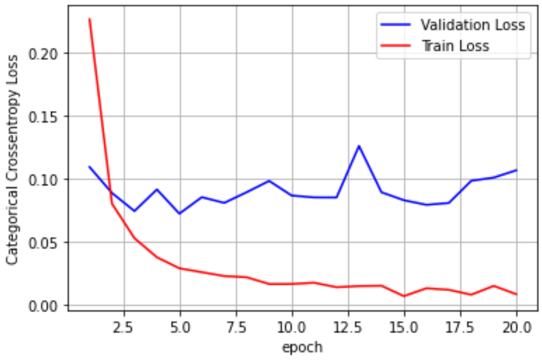
In [50]:
```
score = model_4.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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 = history4.history['val_loss']
ty = history4.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.10676916894102763
Test accuracy: 0.9775000214576721
```



%matplotlib inline

plt.close('all')

1.2. MLP + ReLU + ADAM + Batch Normalization + HE normal initialization

In [44]:
```
from keras.layers.normalization import BatchNormalization
from keras.initializers import he_normal

model_5 = Sequential()

model_5.add(Dense(256, activation='relu', input_shape=(input_dim,),kernel_initializer=he_normal(seed=None)))
model_5.add(Dense(384, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_5.add(Dense(512, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_5.add(BatchNormalization())

model_5.add(Dense(output_dim, activation='softmax'))


model_5.summary()

model_5.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history5 = model_5.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test
, Y_test))
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_22 (Dense)             (None, 256)               200960    
_________________________________________________________________
dense_23 (Dense)             (None, 384)               98688     
_________________________________________________________________
dense_24 (Dense)             (None, 512)               197120    
_________________________________________________________________
batch_normalization_5 (Batch (None, 512)               2048      
_________________________________________________________________
dense_25 (Dense)             (None, 10)                5130      
=================================================================
Total params: 503,946
Trainable params: 502,922
Non-trainable params: 1,024
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.2097 - accuracy: 0.9366 - val_loss: 0.1158
- val_accuracy: 0.9610
Epoch 2/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0848 - accuracy: 0.9737 - val_loss: 0.1262
- val_accuracy: 0.9619
Epoch 3/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0627 - accuracy: 0.9805 - val_loss: 0.0781
- val_accuracy: 0.9760
Epoch 4/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0477 - accuracy: 0.9846 - val_loss: 0.0862
- val_accuracy: 0.9738
Epoch 5/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0351 - accuracy: 0.9883 - val_loss: 0.1040
- val_accuracy: 0.9689
Epoch 6/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0316 - accuracy: 0.9893 - val_loss: 0.0931
- val_accuracy: 0.9733
Epoch 7/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0261 - accuracy: 0.9916 - val_loss: 0.0805
- val_accuracy: 0.9796
Epoch 8/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0210 - accuracy: 0.9929 - val_loss: 0.0913
- val_accuracy: 0.9755
Epoch 9/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0207 - accuracy: 0.9930 - val_loss: 0.1076
- val_accuracy: 0.9734
Epoch 10/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.0182 - accuracy: 0.9939 - val_loss: 0.0949
- val_accuracy: 0.9745
Epoch 11/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0157 - accuracy: 0.9945 - val_loss: 0.0788
- val_accuracy: 0.9782
Epoch 12/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0167 - accuracy: 0.9945 - val_loss: 0.0946
- val_accuracy: 0.9779
Epoch 13/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.0181 - accuracy: 0.9943 - val_loss: 0.0817
- val_accuracy: 0.9799
Epoch 14/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0115 - accuracy: 0.9961 - val_loss: 0.0830
- val_accuracy: 0.9794
Epoch 15/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0115 - accuracy: 0.9963 - val_loss: 0.0980
- val_accuracy: 0.9766
Epoch 16/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.0113 - accuracy: 0.9963 - val_loss: 0.1014
- val_accuracy: 0.9783
Epoch 17/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.0127 - accuracy: 0.9960 - val_loss: 0.1022
- val_accuracy: 0.9784
Epoch 18/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0105 - accuracy: 0.9965 - val_loss: 0.0914
- val_accuracy: 0.9793
Epoch 19/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0094 - accuracy: 0.9966 - val_loss: 0.1374
- val_accuracy: 0.9735
Epoch 20/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0124 - accuracy: 0.9959 - val_loss: 0.1005
- val_accuracy: 0.9800
```

In [51]:

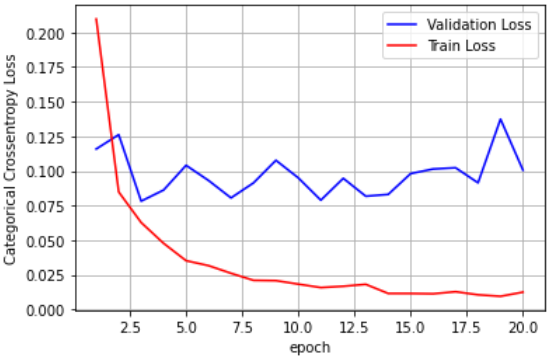
```
score = model_5.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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 = history5.history['val_loss']
ty = history5.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.10054782296594231
Test accuracy: 0.9800000190734863
```



1.3. MLP + ReLU + ADAM + Batch Normalization + Dropout + HE normal initialization

In [46]:
```
from keras.layers.normalization import BatchNormalization
from keras.initializers import he_normal
from keras.layers import Dropout

model_6 = Sequential()

model_6.add(Dense(256, activation='relu', input_shape=(input_dim,),kernel_initializer=he_normal(seed=None)))
model_6.add(BatchNormalization())
model_6.add(Dropout(0.3))

model_6.add(Dense(384, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_6.add(BatchNormalization())
model_6.add(Dropout(0.4))

model_6.add(Dense(256, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_6.add(BatchNormalization())
model_6.add(Dropout(0.5))

model_6.add(Dense(output_dim, activation='softmax'))


model_6.summary()

model_6.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history6 = model_6.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test
, Y_test))
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_26 (Dense)             (None, 256)               200960    
_________________________________________________________________
batch_normalization_6 (Batch (None, 256)               1024      
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_27 (Dense)             (None, 384)               98688     
_________________________________________________________________
batch_normalization_7 (Batch (None, 384)               1536      
_________________________________________________________________
dropout_4 (Dropout)          (None, 384)               0         
_________________________________________________________________
dense_28 (Dense)             (None, 256)               98560     
_________________________________________________________________
batch_normalization_8 (Batch (None, 256)               1024      
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_29 (Dense)             (None, 10)                2570      
=================================================================
Total params: 404,362
Trainable params: 402,570
Non-trainable params: 1,792
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.4752 - accuracy: 0.8572 - val_loss: 0.1460
- val_accuracy: 0.9537
Epoch 2/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.2098 - accuracy: 0.9369 - val_loss: 0.1075
- val_accuracy: 0.9673
Epoch 3/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.1569 - accuracy: 0.9522 - val_loss: 0.0933
- val_accuracy: 0.9719
Epoch 4/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.1319 - accuracy: 0.9596 - val_loss: 0.0828
- val_accuracy: 0.9727
Epoch 5/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.1123 - accuracy: 0.9656 - val_loss: 0.0785
- val_accuracy: 0.9764
Epoch 6/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.1039 - accuracy: 0.9676 - val_loss: 0.0812
- val_accuracy: 0.9757
Epoch 7/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0946 - accuracy: 0.9713 - val_loss: 0.0721
- val_accuracy: 0.9773
Epoch 8/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0840 - accuracy: 0.9739 - val_loss: 0.0687
- val_accuracy: 0.9787
Epoch 9/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.0800 - accuracy: 0.9747 - val_loss: 0.0696
- val_accuracy: 0.9788
Epoch 10/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.0735 - accuracy: 0.9770 - val_loss: 0.0661
- val_accuracy: 0.9815
Epoch 11/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.0703 - accuracy: 0.9779 - val_loss: 0.0590
- val_accuracy: 0.9814
Epoch 12/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0661 - accuracy: 0.9791 - val_loss: 0.0628
- val_accuracy: 0.9803
Epoch 13/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0622 - accuracy: 0.9804 - val_loss: 0.0603
- val_accuracy: 0.9811
Epoch 14/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0582 - accuracy: 0.9822 - val_loss: 0.0594
- val_accuracy: 0.9835
Epoch 15/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0544 - accuracy: 0.9828 - val_loss: 0.0548
- val_accuracy: 0.9849
Epoch 16/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.0528 - accuracy: 0.9826 - val_loss: 0.0594
- val_accuracy: 0.9825
Epoch 17/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0506 - accuracy: 0.9834 - val_loss: 0.0605
- val_accuracy: 0.9827
Epoch 18/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.0486 - accuracy: 0.9838 - val_loss: 0.0573
- val_accuracy: 0.9827
Epoch 19/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.0480 - accuracy: 0.9852 - val_loss: 0.0561
- val_accuracy: 0.9843
Epoch 20/20
```

```
60000/60000 [==============================] - 7s 117us/step - loss: 0.0415 - accuracy: 0.9870 - val_loss: 0.0638
- val_accuracy: 0.9829
```

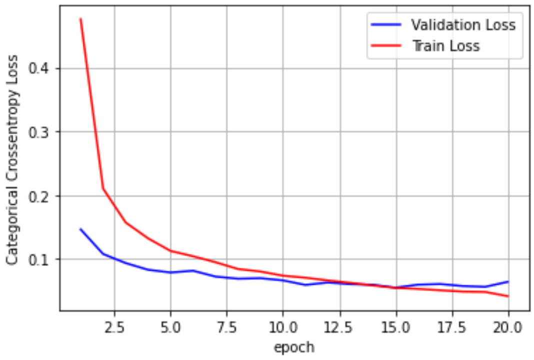
In [52]:
```python
score = model_6.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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 = history6.history['val_loss']
ty = history6.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.06382637676765153
Test accuracy: 0.9829000234603882
```



In [0]:

3. 5-Hidden layer architecture

3.1. MLP + ReLU + ADAM

In [48]:
```
model_7 = Sequential()
model_7.add(Dense(128, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stdde
v=0.062, seed=None)))
model_7.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_7.add(Dense(512, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_7.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_7.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_7.add(Dense(output_dim, activation='softmax'))

print(model_7.summary())

model_7.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history7 = model_7.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test
, Y_test))
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_30 (Dense)             (None, 128)               100480    
_________________________________________________________________
dense_31 (Dense)             (None, 256)               33024     
_________________________________________________________________
dense_32 (Dense)             (None, 512)               131584    
_________________________________________________________________
dense_33 (Dense)             (None, 256)               131328    
_________________________________________________________________
dense_34 (Dense)             (None, 128)               32896     
_________________________________________________________________
dense_35 (Dense)             (None, 10)                1290      
=================================================================
Total params: 430,602
Trainable params: 430,602
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 101us/step - loss: 0.2822 - accuracy: 0.9142 - val_loss: 0.1245
- val_accuracy: 0.9630
Epoch 2/20
60000/60000 [==============================] - 6s 100us/step - loss: 0.1103 - accuracy: 0.9662 - val_loss: 0.1154
- val_accuracy: 0.9662
Epoch 3/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0743 - accuracy: 0.9771 - val_loss: 0.0957 -
val_accuracy: 0.9721
Epoch 4/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.0599 - accuracy: 0.9810 - val_loss: 0.0966 -
val_accuracy: 0.9720
Epoch 5/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0456 - accuracy: 0.9854 - val_loss: 0.1049 -
val_accuracy: 0.9707
Epoch 6/20
60000/60000 [==============================] - 6s 101us/step - loss: 0.0402 - accuracy: 0.9873 - val_loss: 0.0943
- val_accuracy: 0.9723
Epoch 7/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.0369 - accuracy: 0.9880 - val_loss: 0.0884 -
val_accuracy: 0.9753
Epoch 8/20
60000/60000 [==============================] - 6s 100us/step - loss: 0.0333 - accuracy: 0.9895 - val_loss: 0.1322
- val_accuracy: 0.9684
Epoch 9/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0282 - accuracy: 0.9912 - val_loss: 0.1010 -
val_accuracy: 0.9741
Epoch 10/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.0281 - accuracy: 0.9911 - val_loss: 0.1097 -
val_accuracy: 0.9757
Epoch 11/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.0235 - accuracy: 0.9927 - val_loss: 0.1033 -
val_accuracy: 0.9743
Epoch 12/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0234 - accuracy: 0.9927 - val_loss: 0.1020 -
val_accuracy: 0.9764
Epoch 13/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.0199 - accuracy: 0.9935 - val_loss: 0.1223 -
val_accuracy: 0.9738
Epoch 14/20
60000/60000 [==============================] - 6s 101us/step - loss: 0.0219 - accuracy: 0.9931 - val_loss: 0.1211
- val_accuracy: 0.9751
Epoch 15/20
60000/60000 [==============================] - 6s 100us/step - loss: 0.0169 - accuracy: 0.9946 - val_loss: 0.1082
- val_accuracy: 0.9751
Epoch 16/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0190 - accuracy: 0.9941 - val_loss: 0.1063 -
val_accuracy: 0.9767
Epoch 17/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0157 - accuracy: 0.9952 - val_loss: 0.1223 -
val_accuracy: 0.9736
Epoch 18/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0171 - accuracy: 0.9947 - val_loss: 0.1059
- val_accuracy: 0.9779
Epoch 19/20
60000/60000 [==============================] - 6s 103us/step - loss: 0.0150 - accuracy: 0.9956 - val_loss: 0.1050
- val_accuracy: 0.9779
Epoch 20/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0178 - accuracy: 0.9948 - val_loss: 0.1090 -
val_accuracy: 0.9771
```

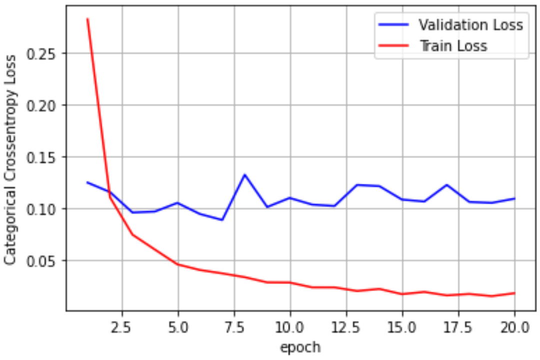
In [53]:
```
score = model_7.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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 = history7.history['val_loss']
ty = history7.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.10898555530682802
Test accuracy: 0.9771000146865845
```



%matplotlib inline

plt.close('all')

3.2. MLP + ReLU + ADAM + Batch Normalization + HE normal initialization

In [54]:

```
from keras.layers.normalization import BatchNormalization
from keras.initializers import he_normal

model_8 = Sequential()

model_8.add(Dense(128, activation='relu', input_shape=(input_dim,),kernel_initializer=he_normal(seed=None)))
model_8.add(Dense(256, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_8.add(BatchNormalization())
model_8.add(Dense(384, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_8.add(BatchNormalization())
model_8.add(Dense(512, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_8.add(BatchNormalization())
model_8.add(Dense(384, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_8.add(BatchNormalization())

model_8.add(Dense(output_dim, activation='softmax'))


model_8.summary()

model_8.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history8 = model_8.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test
, Y_test))
```

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_36 (Dense)             (None, 128)               100480    
_________________________________________________________________
dense_37 (Dense)             (None, 256)               33024     
_________________________________________________________________
batch_normalization_9 (Batch (None, 256)               1024      
_________________________________________________________________
dense_38 (Dense)             (None, 384)               98688     
_________________________________________________________________
batch_normalization_10 (Batc (None, 384)               1536      
_________________________________________________________________
dense_39 (Dense)             (None, 512)               197120    
_________________________________________________________________
batch_normalization_11 (Batc (None, 512)               2048      
_________________________________________________________________
dense_40 (Dense)             (None, 384)               196992    
_________________________________________________________________
batch_normalization_12 (Batc (None, 384)               1536      
_________________________________________________________________
dense_41 (Dense)             (None, 10)                3850      
=================================================================
Total params: 636,298
Trainable params: 633,226
Non-trainable params: 3,072
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s 163us/step - loss: 0.2359 - accuracy: 0.9273 - val_loss: 0.1322
- val_accuracy: 0.9598
Epoch 2/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.0929 - accuracy: 0.9713 - val_loss: 0.1262
- val_accuracy: 0.9613
Epoch 3/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.0638 - accuracy: 0.9793 - val_loss: 0.1127
- val_accuracy: 0.9660
Epoch 4/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.0509 - accuracy: 0.9826 - val_loss: 0.1158
- val_accuracy: 0.9670
Epoch 5/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.0412 - accuracy: 0.9856 - val_loss: 0.1240
- val_accuracy: 0.9640
Epoch 6/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.0378 - accuracy: 0.9873 - val_loss: 0.0794
- val_accuracy: 0.9740
Epoch 7/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.0318 - accuracy: 0.9888 - val_loss: 0.0763
- val_accuracy: 0.9777
Epoch 8/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.0317 - accuracy: 0.9890 - val_loss: 0.0817
- val_accuracy: 0.9769
Epoch 9/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0282 - accuracy: 0.9908 - val_loss: 0.0792
- val_accuracy: 0.9782
Epoch 10/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0256 - accuracy: 0.9912 - val_loss: 0.0851
- val_accuracy: 0.9773
Epoch 11/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0232 - accuracy: 0.9921 - val_loss: 0.0930
- val_accuracy: 0.9760
Epoch 12/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0199 - accuracy: 0.9931 - val_loss: 0.0995
- val_accuracy: 0.9756
Epoch 13/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.0215 - accuracy: 0.9925 - val_loss: 0.0923
- val_accuracy: 0.9756
Epoch 14/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.0216 - accuracy: 0.9929 - val_loss: 0.1121
- val_accuracy: 0.9732
Epoch 15/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0168 - accuracy: 0.9943 - val_loss: 0.0971
- val_accuracy: 0.9753
Epoch 16/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0163 - accuracy: 0.9945 - val_loss: 0.1065
- val_accuracy: 0.9747
Epoch 17/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.0195 - accuracy: 0.9937 - val_loss: 0.0850
- val_accuracy: 0.9797
Epoch 18/20
60000/60000 [==============================] - 10s 163us/step - loss: 0.0147 - accuracy: 0.9957 - val_loss: 0.0887
- val_accuracy: 0.9788
Epoch 19/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.0152 - accuracy: 0.9952 - val_loss: 0.0815
- val_accuracy: 0.9797
Epoch 20/20
```

```
60000/60000 [==============================] - 10s 160us/step - loss: 0.0126 - accuracy: 0.9959 - val_loss: 0.0885
- val_accuracy: 0.9790
```

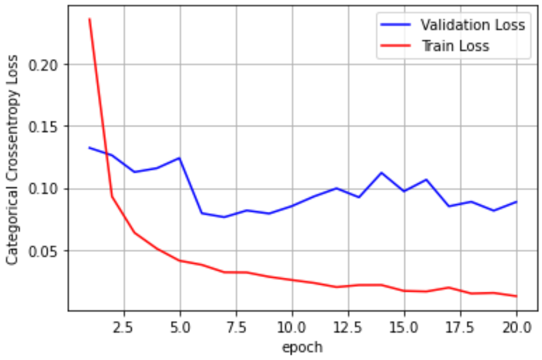
In [55]:
```
score = model_8.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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 = history8.history['val_loss']
ty = history8.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.088547984336391l6
Test accuracy: 0.9789999723434448
```



3.3. MLP + ReLU + ADAM + Batch Normalization + Dropout + HE normal initialization

In [57]:
```
from keras.layers.normalization import BatchNormalization
from keras.initializers import he_normal
from keras.layers import Dropout

model_9 = Sequential()

model_9.add(Dense(256, activation='relu', input_shape=(input_dim,),kernel_initializer=he_normal(seed=None)))
model_9.add(BatchNormalization())
model_9.add(Dropout(0.3))

model_9.add(Dense(384, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_9.add(BatchNormalization())
model_9.add(Dropout(0.3))

model_9.add(Dense(512, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_9.add(BatchNormalization())
model_9.add(Dropout(0.3))

model_9.add(Dense(384, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_9.add(BatchNormalization())
model_9.add(Dropout(0.3))

model_9.add(Dense(256, activation='relu', kernel_initializer=he_normal(seed=None)) )
model_9.add(BatchNormalization())
model_9.add(Dropout(0.3))

model_9.add(Dense(output_dim, activation='softmax'))


model_9.summary()

model_9.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history9 = model_9.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test
, Y_test))
```

```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_46 (Dense)             (None, 256)               200960    
_________________________________________________________________
batch_normalization_16 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_9 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_47 (Dense)             (None, 384)               98688     
_________________________________________________________________
batch_normalization_17 (Batc (None, 384)               1536      
_________________________________________________________________
dropout_10 (Dropout)         (None, 384)               0         
_________________________________________________________________
dense_48 (Dense)             (None, 512)               197120    
_________________________________________________________________
batch_normalization_18 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_11 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_49 (Dense)             (None, 384)               196992    
_________________________________________________________________
batch_normalization_19 (Batc (None, 384)               1536      
_________________________________________________________________
dropout_12 (Dropout)         (None, 384)               0         
_________________________________________________________________
dense_50 (Dense)             (None, 256)               98560     
_________________________________________________________________
batch_normalization_20 (Batc (None, 256)               1024      
_________________________________________________________________
dropout_13 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_51 (Dense)             (None, 10)                2570      
=================================================================
Total params: 802,058
Trainable params: 798,474
Non-trainable params: 3,584
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 13s 218us/step - loss: 0.4796 - accuracy: 0.8537 - val_loss: 0.1411 
- val_accuracy: 0.9555
Epoch 2/20
60000/60000 [==============================] - 12s 204us/step - loss: 0.2065 - accuracy: 0.9383 - val_loss: 0.1054 
- val_accuracy: 0.9679
Epoch 3/20
60000/60000 [==============================] - 12s 207us/step - loss: 0.1593 - accuracy: 0.9514 - val_loss: 0.0895 
- val_accuracy: 0.9733
Epoch 4/20
60000/60000 [==============================] - 12s 201us/step - loss: 0.1370 - accuracy: 0.9571 - val_loss: 0.0891 
- val_accuracy: 0.9733
Epoch 5/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.1178 - accuracy: 0.9639 - val_loss: 0.0760 
- val_accuracy: 0.9773
Epoch 6/20
60000/60000 [==============================] - 12s 203us/step - loss: 0.1082 - accuracy: 0.9677 - val_loss: 0.0766 
- val_accuracy: 0.9766
Epoch 7/20
60000/60000 [==============================] - 12s 200us/step - loss: 0.0964 - accuracy: 0.9698 - val_loss: 0.0726 
- val_accuracy: 0.9776
Epoch 8/20
60000/60000 [==============================] - 12s 199us/step - loss: 0.0913 - accuracy: 0.9722 - val_loss: 0.0757 
- val_accuracy: 0.9767
Epoch 9/20
60000/60000 [==============================] - 12s 199us/step - loss: 0.0860 - accuracy: 0.9728 - val_loss: 0.0779 
- val_accuracy: 0.9776
Epoch 10/20
60000/60000 [==============================] - 12s 201us/step - loss: 0.0809 - accuracy: 0.9753 - val_loss: 0.0684 
- val_accuracy: 0.9796
Epoch 11/20
60000/60000 [==============================] - 12s 195us/step - loss: 0.0759 - accuracy: 0.9755 - val_loss: 0.0673 
- val_accuracy: 0.9796
Epoch 12/20
60000/60000 [==============================] - 12s 194us/step - loss: 0.0718 - accuracy: 0.9772 - val_loss: 0.0732 
- val_accuracy: 0.9776
Epoch 13/20
60000/60000 [==============================] - 12s 194us/step - loss: 0.0687 - accuracy: 0.9784 - val_loss: 0.0665 
- val_accuracy: 0.9786
Epoch 14/20
60000/60000 [==============================] - 12s 200us/step - loss: 0.0611 - accuracy: 0.9804 - val_loss: 0.0631 
- val_accuracy: 0.9820
Epoch 15/20
60000/60000 [==============================] - 12s 199us/step - loss: 0.0602 - accuracy: 0.9814 - val_loss: 0.0641 
- val_accuracy: 0.9819
Epoch 16/20
```

```
60000/60000 [==============================] - 12s 196us/step - loss: 0.0574 - accuracy: 0.9822 - val_loss: 0.0645
- val_accuracy: 0.9820
Epoch 17/20
60000/60000 [==============================] - 12s 204us/step - loss: 0.0550 - accuracy: 0.9827 - val_loss: 0.0563
- val_accuracy: 0.9837
Epoch 18/20
60000/60000 [==============================] - 13s 214us/step - loss: 0.0539 - accuracy: 0.9829 - val_loss: 0.0619
- val_accuracy: 0.9841
Epoch 19/20
60000/60000 [==============================] - 13s 215us/step - loss: 0.0513 - accuracy: 0.9838 - val_loss: 0.0584
- val_accuracy: 0.9840
Epoch 20/20
60000/60000 [==============================] - 13s 218us/step - loss: 0.0516 - accuracy: 0.9835 - val_loss: 0.0643
- val_accuracy: 0.9817
```

In [58]:

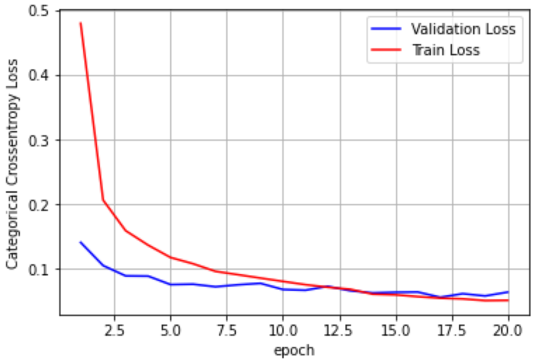
```python
score = model_9.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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 = history9.history['val_loss']
ty = history9.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.06429058624608443
Test accuracy: 0.9817000031471252
```



In [64]:

```python
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Hidden Layers","Parameters","Loss"]
x.add_row([2,'MLP + ReLU + ADAM ', 0.979])
x.add_row([2,'MLP + ReLU + ADAM + Batch Normalization + HE normal initialization', 0.976])
x.add_row([2,'MLP + ReLU + ADAM + Batch Normalization + Dropout + HE normal initialization', 0.983])
x.add_row([3,'MLP + ReLU + ADAM ', 0.977])
x.add_row([3,'MLP + ReLU + ADAM + Batch Normalization + HE normal initialization', 0.980])
x.add_row([3,'MLP + ReLU + ADAM + Batch Normalization + Dropout + HE normal initialization', 0.982])
x.add_row([5,'MLP + ReLU + ADAM ', 0.977])
x.add_row([5,'MLP + ReLU + ADAM + Batch Normalization + HE normal initialization', 0.978])
x.add_row([5,'MLP + ReLU + ADAM + Batch Normalization + Dropout + HE normal initialization', 0.981])
print(x)
```

Hidden Layers	Parameters	Loss
2	MLP + ReLU + ADAM	0.979
2	MLP + ReLU + ADAM + Batch Normalization + HE normal initialization	0.976
2	MLP + ReLU + ADAM + Batch Normalization + Dropout + HE normal initialization	0.983
3	MLP + ReLU + ADAM	0.977
3	MLP + ReLU + ADAM + Batch Normalization + HE normal initialization	0.98
3	MLP + ReLU + ADAM + Batch Normalization + Dropout + HE normal initialization	0.982
5	MLP + ReLU + ADAM	0.977
5	MLP + ReLU + ADAM + Batch Normalization + HE normal initialization	0.978
5	MLP + ReLU + ADAM + Batch Normalization + Dropout + HE normal initialization	0.981

In [0]: