Assignment 12: TensorFlow and Keras Build various MLP architectures for MNIST dataset

Objective:

1. Building Models with 3 different architectures:

   i) 2-Hidden layer architecture (784-472-168-10 architecture)

   ii) 3-Hidden layer architecture (784-352-164-124-10 architecture)

   iii) 5-Hidden layer architecture (784-216-170-136-80-38-10 architecture)

1. Train-Test error plot

1. Activation='relu'+ Adam Optimizer+Batch_Normalization +Drop_out

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
```

```
Using TensorFlow backend.
```

In [3]:

```
%matplotlib inline
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 [==============================] - 4s 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 testing 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 testing examples : 10000 and each image is of shape (28, 28)
```

In [6]:

```
# Each image we have is a (28*28) vector
# Let's 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 testing 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 testing examples : 10000 and each image is of shape (784)
```

In [8]:
```
# Let's print the first entry
print(X_train[0])
```

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

In [9]:
```
# Each cell of above matrix is having a value between 0-255
# before applying machine learning algorithms, let's normalize the data
# X => (X - Xmin)/(Xmax-Xmin) = X/255

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

In [11]:
```
# Let's print first entry after normlizing
print(X_train[0])
```

```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.01176471 0.07058824 0.07058824 0.07058824
 0.49411765 0.53333333 0.68627451 0.10196078 0.65098039 1.
 0.96862745 0.49803922 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.11764706 0.14117647 0.36862745 0.60392157
 0.66666667 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.88235294 0.6745098  0.99215686 0.94901961 0.76470588 0.25098039
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.19215686
 0.93333333 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.99215686 0.99215686 0.99215686 0.98431373 0.36470588 0.32156863
 0.32156863 0.21960784 0.15294118 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.07058824 0.85882353 0.99215686
 0.99215686 0.99215686 0.99215686 0.99215686 0.77647059 0.71372549
 0.96862745 0.94509804 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.31372549 0.61176471 0.41960784 0.99215686
 0.99215686 0.80392157 0.04313725 0.         0.16862745 0.60392157
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.05490196 0.00392157 0.60392157 0.99215686 0.35294118
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.54509804 0.99215686 0.74509804 0.00784314 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.04313725
 0.74509804 0.99215686 0.2745098  0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.1372549  0.94509804
 0.88235294 0.62745098 0.42352941 0.00392157 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.31764706 0.94117647 0.99215686
 0.99215686 0.46666667 0.09803922 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.17647059 0.72941176 0.99215686 0.99215686
 0.58823529 0.10588235 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.0627451  0.36470588 0.98823529 0.99215686 0.73333333
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.97647059 0.99215686 0.97647059 0.25098039 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.18039216 0.50980392 0.71764706 0.99215686
 0.99215686 0.81176471 0.00784314 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.15294118 0.58039216
 0.89803922 0.99215686 0.99215686 0.99215686 0.98039216 0.71372549
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.09411765 0.44705882 0.86666667 0.99215686 0.99215686 0.99215686
 0.99215686 0.78823529 0.30588235 0.         0.         0.
```

```
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.09019608 0.25882353 0.83529412 0.99215686
 0.99215686 0.99215686 0.99215686 0.77647059 0.31764706 0.00784314
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.07058824 0.67058824
 0.85882353 0.99215686 0.99215686 0.99215686 0.99215686 0.76470588
 0.31372549 0.03529412 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.21568627 0.6745098  0.88627451 0.99215686 0.99215686 0.99215686
 0.99215686 0.95686275 0.52156863 0.04313725 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.53333333 0.99215686
 0.99215686 0.99215686 0.83137255 0.52941176 0.51764706 0.0627451
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
```

In [13]:

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

# let's convert this into a 10 dimensional vector as it is needed for MLPs

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

print("After converting, class label of first image: ",y_train[0])
```

```
Class label of first image : 5
After converting, class label of first image:  [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

In [14]:

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

In [15]:

```
# Setting model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

1) 2-Hidden layer architecture (784-472-168-10 architecture)

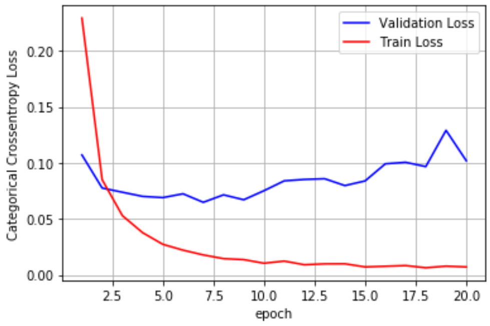
1.1 MLP + ReLU + ADAM

In [16]:
```
model_relu = Sequential()
model_relu.add(Dense(472, activation='relu', input_shape=(input_dim,),
                     kernel_initializer=he_normal(seed=None)))
model_relu.add(Dense(168, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

history11 = model_relu.fit(X_train, y_train,
                         batch_size=batch_size,
                         epochs=nb_epoch, verbose=1,
                         validation_data=(X_test, y_test))
```

```
WARNING:tensorflow:From C:\Users\pratyush.acharya\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorflow\python\ops\resou
rce_variable_ops.py:435: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future versi
on.
Instructions for updating:
Colocations handled automatically by placer.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 472)               370520
_________________________________________________________________
dense_2 (Dense)              (None, 168)               79464
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1690
=================================================================
Total params: 451,674
Trainable params: 451,674
Non-trainable params: 0
_________________________________________________________________
None
WARNING:tensorflow:From C:\Users\pratyush.acharya\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorflow\python\ops\math_
ops.py:3066: to_int32 (from tensorflow.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 [==============================] - 8s 134us/step - loss: 0.2293 - accuracy: 0.9337 - val_loss: 0.1071 - val_accuracy:
0.9665
Epoch 2/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.0851 - accuracy: 0.9744 - val_loss: 0.0776 - val_accuracy:
0.9745
Epoch 3/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0530 - accuracy: 0.9838 - val_loss: 0.0739 - val_accuracy:
0.9767
Epoch 4/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0378 - accuracy: 0.9880 - val_loss: 0.0702 - val_accuracy:
0.9786
Epoch 5/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0274 - accuracy: 0.9914 - val_loss: 0.0691 - val_accuracy:
0.9790
Epoch 6/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0222 - accuracy: 0.9926 - val_loss: 0.0725 - val_accuracy:
0.9793
Epoch 7/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.0179 - accuracy: 0.9940 - val_loss: 0.0649 - val_accuracy:
0.9814
Epoch 8/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0146 - accuracy: 0.9953 - val_loss: 0.0715 - val_accuracy:
0.9806
Epoch 9/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.0137 - accuracy: 0.9954 - val_loss: 0.0671 - val_accuracy:
0.9825
Epoch 10/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.0105 - accuracy: 0.9966 - val_loss: 0.0752 - val_accuracy:
0.9804
Epoch 11/20
60000/60000 [==============================] - 8s 140us/step - loss: 0.0124 - accuracy: 0.9959 - val_loss: 0.0840 - val_accuracy:
0.9801
Epoch 12/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0091 - accuracy: 0.9970 - val_loss: 0.0853 - val_accuracy:
0.9806
Epoch 13/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0100 - accuracy: 0.9968 - val_loss: 0.0859 - val_accuracy:
0.9801
Epoch 14/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.0100 - accuracy: 0.9968 - val_loss: 0.0798 - val_accuracy:
0.9822
Epoch 15/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0072 - accuracy: 0.9977 - val_loss: 0.0840 - val_accuracy:
0.9805
Epoch 16/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0078 - accuracy: 0.9977 - val_loss: 0.0992 - val_accuracy:
0.9788
Epoch 17/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0084 - accuracy: 0.9971 - val_loss: 0.1006 - val_accuracy:
0.9786
Epoch 18/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.0065 - accuracy: 0.9979 - val_loss: 0.0967 - val_accuracy:
0.9814
Epoch 19/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0079 - accuracy: 0.9976 - val_loss: 0.1291 - val_accuracy:
0.9771
Epoch 20/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0072 - accuracy: 0.9978 - val_loss: 0.1020 - val_accuracy:
0.9792
```

In [20]:
```
score = model_relu.evaluate(X_test, y_test, verbose=0)
score1=score[0]
score2=score[1]
train_acc1=history11.history['accuracy']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax11 = plt.subplots(1,1)
ax11.set_xlabel('epoch') ; ax11.set_ylabel('Categorical Crossentropy Loss')

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

# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy

vy11 = history11.history['val_loss']
ty11 = history11.history['loss']
plt_dynamic(x, vy11, ty11, ax11)
```

```
Test score: 0.10199347244282822
Test accuracy: 0.979200005531311
```



1.2 MLP + Batch-Norm on hidden Layers + AdamOptimizer

In [21]:
```
from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_batch.add(Dense(472, activation='relu',
                      input_shape=(input_dim,),
                      kernel_initializer=he_normal(seed=None)))
model_batch.add(BatchNormalization())

model_batch.add(Dense(168, activation='relu',
                      kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

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


model_batch.summary()
```

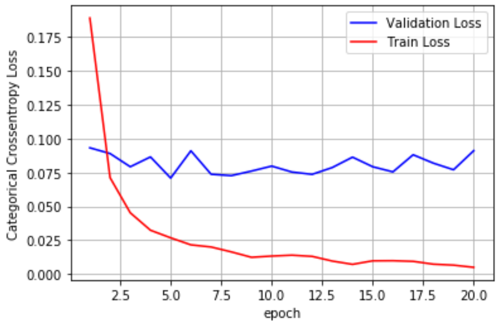
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 472)               370520    
_________________________________________________________________
batch_normalization_1 (Batch (None, 472)               1888      
_________________________________________________________________
dense_5 (Dense)              (None, 168)               79464     
_________________________________________________________________
batch_normalization_2 (Batch (None, 168)               672       
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1690      
=================================================================
Total params: 454,234
Trainable params: 452,954
Non-trainable params: 1,280
_________________________________________________________________
```

In [22]:
```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy',
                    metrics=['accuracy'])

history12 = 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 [==============================] - 10s 166us/step - loss: 0.1890 - accuracy: 0.9433 - val_loss: 0.0932 - val_accurac
y: 0.9721
Epoch 2/20
60000/60000 [==============================] - 9s 158us/step - loss: 0.0712 - accuracy: 0.9788 - val_loss: 0.0890 - val_accuracy:
0.9709
Epoch 3/20
60000/60000 [==============================] - 9s 158us/step - loss: 0.0453 - accuracy: 0.9858 - val_loss: 0.0792 - val_accuracy:
0.9743
Epoch 4/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.0324 - accuracy: 0.9900 - val_loss: 0.0865 - val_accuracy:
0.9746
Epoch 5/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.0268 - accuracy: 0.9917 - val_loss: 0.0709 - val_accuracy:
0.9789
Epoch 6/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0216 - accuracy: 0.9930 - val_loss: 0.0910 - val_accuracy:
0.9747
Epoch 7/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.0200 - accuracy: 0.9936 - val_loss: 0.0737 - val_accuracy:
0.9777
Epoch 8/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.0164 - accuracy: 0.9948 - val_loss: 0.0727 - val_accuracy:
0.9793
Epoch 9/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0124 - accuracy: 0.9959 - val_loss: 0.0760 - val_accuracy:
0.9784
Epoch 10/20
60000/60000 [==============================] - 9s 158us/step - loss: 0.0134 - accuracy: 0.9957 - val_loss: 0.0797 - val_accuracy:
0.9794
Epoch 11/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0140 - accuracy: 0.9954 - val_loss: 0.0753 - val_accuracy:
0.9783
Epoch 12/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0131 - accuracy: 0.9956 - val_loss: 0.0736 - val_accuracy:
0.9804
Epoch 13/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.0097 - accuracy: 0.9969 - val_loss: 0.0787 - val_accuracy:
0.9806
Epoch 14/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0072 - accuracy: 0.9976 - val_loss: 0.0863 - val_accuracy:
0.9797
Epoch 15/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0099 - accuracy: 0.9969 - val_loss: 0.0792 - val_accuracy:
0.9801
Epoch 16/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.0099 - accuracy: 0.9966 - val_loss: 0.0754 - val_accuracy:
0.9819
Epoch 17/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.0095 - accuracy: 0.9968 - val_loss: 0.0881 - val_accuracy:
0.9809
Epoch 18/20
60000/60000 [==============================] - 9s 149us/step - loss: 0.0074 - accuracy: 0.9977 - val_loss: 0.0819 - val_accuracy:
0.9814
Epoch 19/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.0067 - accuracy: 0.9980 - val_loss: 0.0770 - val_accuracy:
0.9829
Epoch 20/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0050 - accuracy: 0.9983 - val_loss: 0.0911 - val_accuracy:
0.9800
```

In [28]:
```
score = model_batch.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
score3=score[0]
score4=score[1]
train_acc2=history12.history['accuracy']

fig,ax12 = plt.subplots(1,1)
ax12.set_xlabel('epoch') ; ax12.set_ylabel('Categorical Crossentropy Loss')

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


vy12 = history12.history['val_loss']
ty12 = history12.history['loss']
plt_dynamic(x, vy12, ty12, ax12)
```

```
Test score: 0.09108898642615058
Test accuracy: 0.9800000190734863
```



1.3 MLP + Dropout + AdamOptimizer

In [24]:
```
from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(472, 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(168, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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


model_drop.summary()
```

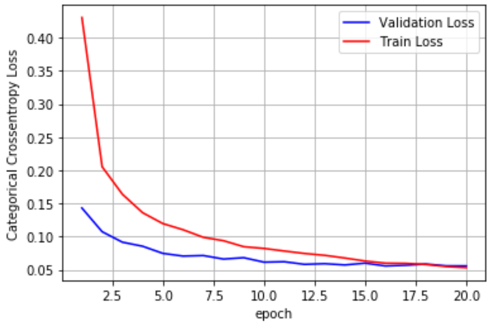
```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 472)               370520
_________________________________________________________________
batch_normalization_3 (Batch (None, 472)               1888
_________________________________________________________________
dropout_1 (Dropout)          (None, 472)               0
_________________________________________________________________
dense_8 (Dense)              (None, 168)               79464
_________________________________________________________________
batch_normalization_4 (Batch (None, 168)               672
_________________________________________________________________
dropout_2 (Dropout)          (None, 168)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1690
=================================================================
Total params: 454,234
Trainable params: 452,954
Non-trainable params: 1,280
_________________________________________________________________
```

In [25]:
```
model_drop.compile(optimizer='adam',
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])

history13 = model_drop.fit(X_train, y_train,
                         batch_size=batch_size,
                         epochs=nb_epoch, verbose=1,
                         validation_data=(X_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s 175us/step - loss: 0.4305 - accuracy: 0.8688 - val_loss: 0.1433 - val_accurac
y: 0.9556
Epoch 2/20
60000/60000 [==============================] - 10s 175us/step - loss: 0.2054 - accuracy: 0.9384 - val_loss: 0.1075 - val_accurac
y: 0.9667
Epoch 3/20
60000/60000 [==============================] - 11s 176us/step - loss: 0.1641 - accuracy: 0.9507 - val_loss: 0.0917 - val_accurac
y: 0.9695
Epoch 4/20
60000/60000 [==============================] - 11s 176us/step - loss: 0.1362 - accuracy: 0.9586 - val_loss: 0.0856 - val_accurac
y: 0.9727
Epoch 5/20
60000/60000 [==============================] - 10s 163us/step - loss: 0.1198 - accuracy: 0.9634 - val_loss: 0.0748 - val_accurac
y: 0.9769
Epoch 6/20
60000/60000 [==============================] - 10s 166us/step - loss: 0.1105 - accuracy: 0.9656 - val_loss: 0.0707 - val_accurac
y: 0.9773
Epoch 7/20
60000/60000 [==============================] - 10s 162us/step - loss: 0.0991 - accuracy: 0.9693 - val_loss: 0.0716 - val_accurac
y: 0.9771
Epoch 8/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.0939 - accuracy: 0.9708 - val_loss: 0.0663 - val_accurac
y: 0.9790
Epoch 9/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.0849 - accuracy: 0.9732 - val_loss: 0.0683 - val_accurac
y: 0.9774
Epoch 10/20
60000/60000 [==============================] - 11s 177us/step - loss: 0.0823 - accuracy: 0.9741 - val_loss: 0.0616 - val_accurac
y: 0.9816
Epoch 11/20
60000/60000 [==============================] - 11s 181us/step - loss: 0.0784 - accuracy: 0.9756 - val_loss: 0.0622 - val_accurac
y: 0.9804
Epoch 12/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.0747 - accuracy: 0.9762 - val_loss: 0.0583 - val_accurac
y: 0.9826
Epoch 13/20
60000/60000 [==============================] - 11s 182us/step - loss: 0.0719 - accuracy: 0.9772 - val_loss: 0.0593 - val_accurac
y: 0.9814
Epoch 14/20
60000/60000 [==============================] - 10s 168us/step - loss: 0.0678 - accuracy: 0.9784 - val_loss: 0.0574 - val_accurac
y: 0.9832
Epoch 15/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.0633 - accuracy: 0.9797 - val_loss: 0.0601 - val_accurac
y: 0.9826
Epoch 16/20
60000/60000 [==============================] - 11s 181us/step - loss: 0.0601 - accuracy: 0.9798 - val_loss: 0.0558 - val_accurac
y: 0.9828
Epoch 17/20
60000/60000 [==============================] - 10s 173us/step - loss: 0.0598 - accuracy: 0.9808 - val_loss: 0.0569 - val_accurac
y: 0.9839
Epoch 18/20
60000/60000 [==============================] - 10s 167us/step - loss: 0.0581 - accuracy: 0.9811 - val_loss: 0.0587 - val_accurac
y: 0.9824
Epoch 19/20
60000/60000 [==============================] - 10s 174us/step - loss: 0.0550 - accuracy: 0.9818 - val_loss: 0.0560 - val_accurac
y: 0.9847
Epoch 20/20
60000/60000 [==============================] - 11s 184us/step - loss: 0.0533 - accuracy: 0.9829 - val_loss: 0.0559 - val_accurac
y: 0.9840
```

In [29]:
```
score = model_drop.evaluate(X_test, y_test, verbose=0)
score5=score[0]
score6=score[1]
train_acc3=history13.history['accuracy']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax13 = plt.subplots(1,1)
ax13.set_xlabel('epoch') ; ax13.set_ylabel('Categorical Crossentropy Loss')

vy13 = history13.history['val_loss']
ty13 = history13.history['loss']
plt_dynamic(x, vy13, ty13, ax13)
```

```
Test score: 0.055900052050140224
Test accuracy: 0.984000027179718
```



2) 3-Hidden layer architecture (784-352-164-124 architecture)

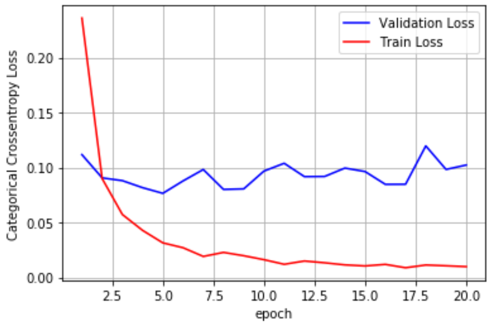
2.1 MLP + ReLU + ADAM

In [27]:
```
model_relu = Sequential()
model_relu.add(Dense(352, activation='relu', input_shape=(input_dim,),
                     kernel_initializer=he_normal(seed=None)))
model_relu.add(Dense(164, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )

model_relu.add(Dense(124, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

history21 = model_relu.fit(X_train, y_train,
                           batch_size=batch_size,
                           epochs=nb_epoch, verbose=1,
                           validation_data=(X_test, y_test))
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_10 (Dense)             (None, 352)               276320    
_________________________________________________________________
dense_11 (Dense)             (None, 164)               57892     
_________________________________________________________________
dense_12 (Dense)             (None, 124)               20460     
_________________________________________________________________
dense_13 (Dense)             (None, 10)                1250      
=================================================================
Total params: 355,922
Trainable params: 355,922
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.2361 - accuracy: 0.9291 - val_loss: 0.1118 - val_accuracy: 0.9678
Epoch 2/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.0900 - accuracy: 0.9721 - val_loss: 0.0906 - val_accuracy: 0.9712 0.0900 - accura
Epoch 3/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0573 - accuracy: 0.9821 - val_loss: 0.0881 - val_accuracy: 0.9738
Epoch 4/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0428 - accuracy: 0.9858 - val_loss: 0.0817 - val_accuracy: 0.9767
Epoch 5/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.0315 - accuracy: 0.9896 - val_loss: 0.0765 - val_accuracy: 0.9784
Epoch 6/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.0270 - accuracy: 0.9912 - val_loss: 0.0878 - val_accuracy: 0.9756
Epoch 7/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0191 - accuracy: 0.9941 - val_loss: 0.0982 - val_accuracy: 0.9737
Epoch 8/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.0228 - accuracy: 0.9923 - val_loss: 0.0801 - val_accuracy: 0.9788
Epoch 9/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0197 - accuracy: 0.9934 - val_loss: 0.0806 - val_accuracy: 0.9800
Epoch 10/20
60000/60000 [==============================] - 7s 109us/step - loss: 0.0161 - accuracy: 0.9948 - val_loss: 0.0968 - val_accuracy: 0.9780
Epoch 11/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.0119 - accuracy: 0.9964 - val_loss: 0.1039 - val_accuracy: 0.9771
Epoch 12/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.0149 - accuracy: 0.9951 - val_loss: 0.0917 - val_accuracy: 0.9785
Epoch 13/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.0133 - accuracy: 0.9954 - val_loss: 0.0919 - val_accuracy: 0.9798
Epoch 14/20
60000/60000 [==============================] - 7s 110us/step - loss: 0.0114 - accuracy: 0.9964 - val_loss: 0.0995 - val_accuracy: 0.9789
Epoch 15/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.0104 - accuracy: 0.9966 - val_loss: 0.0963 - val_accuracy: 0.9805
Epoch 16/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.0119 - accuracy: 0.9961 - val_loss: 0.0847 - val_accuracy: 0.9828
Epoch 17/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.0088 - accuracy: 0.9974 - val_loss: 0.0848 - val_accuracy: 0.9827
Epoch 18/20
60000/60000 [==============================] - 7s 113us/step - loss: 0.0113 - accuracy: 0.9964 - val_loss: 0.1197 - val_accuracy: 0.9771
Epoch 19/20
60000/60000 [==============================] - 7s 112us/step - loss: 0.0106 - accuracy: 0.9964 - val_loss: 0.0983 - val_accuracy: 0.9804
Epoch 20/20
60000/60000 [==============================] - 7s 113us/step - loss: 0.0098 - accuracy: 0.9969 - val_loss: 0.1023 - val_accuracy: 0.9790
```

In [30]:
```
score = model_relu.evaluate(X_test, y_test, verbose=0)
score7=score[0]
score8=score[1]
train_acc4=history21.history['accuracy']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax21 = plt.subplots(1,1)
ax21.set_xlabel('epoch') ; ax21.set_ylabel('Categorical Crossentropy Loss')

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


vy21 = history21.history['val_loss']
ty21 = history21.history['loss']
plt_dynamic(x, vy21, ty21, ax21)
```

```
Test score: 0.10229123631868356
Test accuracy: 0.9789999723434448
```



2.2 MLP + Batch-Norm on hidden Layers + AdamOptimizer

In [31]:
```
from keras.layers.normalization import BatchNormalization

model_batch = Sequential()

model_relu.add(Dense(352, activation='relu', input_shape=(input_dim,),
                     kernel_initializer=he_normal(seed=None)))
model_batch.add(BatchNormalization())
model_relu.add(Dense(164, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

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

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

In [33]:
```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy',
                    metrics=['accuracy'])

history22 = model_batch.fit(X_train, y_train,
                           batch_size=batch_size,
                           epochs=nb_epoch, verbose=1,
                           validation_data=(X_test, y_test))
```

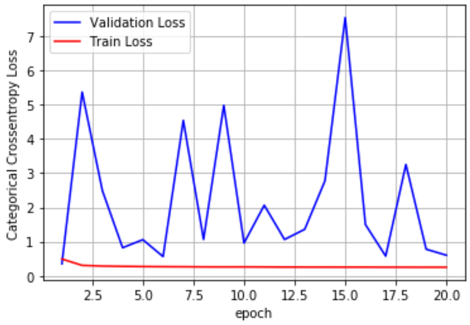
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.4919 - accuracy: 0.8541 - val_loss: 0.3529 - val_accuracy:
0.9093
Epoch 2/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.3065 - accuracy: 0.9117 - val_loss: 5.3663 - val_accuracy:
0.4360
Epoch 3/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.2863 - accuracy: 0.9199 - val_loss: 2.4895 - val_accuracy:
0.5634
Epoch 4/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.2774 - accuracy: 0.9216 - val_loss: 0.8210 - val_accuracy:
0.7968
Epoch 5/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.2714 - accuracy: 0.9236 - val_loss: 1.0564 - val_accuracy:
0.7441
Epoch 6/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.2681 - accuracy: 0.9251 - val_loss: 0.5624 - val_accuracy:
0.8672
Epoch 7/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.2653 - accuracy: 0.9253 - val_loss: 4.5390 - val_accuracy:
0.3114
Epoch 8/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.2616 - accuracy: 0.9261 - val_loss: 1.0641 - val_accuracy:
0.7624
Epoch 9/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.2605 - accuracy: 0.9280 - val_loss: 4.9788 - val_accuracy:
0.3448
Epoch 10/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.2614 - accuracy: 0.9271 - val_loss: 0.9586 - val_accuracy:
0.7472
Epoch 11/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.2590 - accuracy: 0.9283 - val_loss: 2.0622 - val_accuracy:
0.6219
Epoch 12/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.2562 - accuracy: 0.9283 - val_loss: 1.0619 - val_accuracy:
0.6985
Epoch 13/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.2553 - accuracy: 0.9280 - val_loss: 1.3594 - val_accuracy:
0.6367
Epoch 14/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.2550 - accuracy: 0.9290 - val_loss: 2.7772 - val_accuracy:
0.5876
Epoch 15/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.2540 - accuracy: 0.9284 - val_loss: 7.5427 - val_accuracy:
0.2344
Epoch 16/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.2543 - accuracy: 0.9291 - val_loss: 1.4970 - val_accuracy:
0.6992
Epoch 17/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.2530 - accuracy: 0.9295 - val_loss: 0.5769 - val_accuracy:
0.8851
Epoch 18/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.2530 - accuracy: 0.9279 - val_loss: 3.2516 - val_accuracy:
0.5636
Epoch 19/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.2512 - accuracy: 0.9297 - val_loss: 0.7772 - val_accuracy:
0.8149
Epoch 20/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.2512 - accuracy: 0.9300 - val_loss: 0.6023 - val_accuracy:
0.8690
```

In [34]:
```
model_batch.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
batch_normalization_5 (Batch (None, 784)               3136
_________________________________________________________________
batch_normalization_6 (Batch (None, 784)               3136
_________________________________________________________________
batch_normalization_7 (Batch (None, 784)               3136
_________________________________________________________________
dense_17 (Dense)             (None, 10)                7850
=================================================================
Total params: 17,258
Trainable params: 12,554
Non-trainable params: 4,704
_________________________________________________________________
```

In [35]:
```
score = model_batch.evaluate(X_test, y_test, verbose=0)
score9=score[0]
score10=score[1]
train_acc5=history22.history['accuracy']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax22 = plt.subplots(1,1)
ax22.set_xlabel('epoch') ; ax22.set_ylabel('Categorical Crossentropy Loss')

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

vy22 = history22.history['val_loss']
ty22 = history22.history['loss']
plt_dynamic(x, vy22, ty22, ax22)
```

```
Test score: 0.6022788046717644
Test accuracy: 0.8690000176429749
```



2.3 MLP + Dropout + AdamOptimizer

In [36]:
```
from keras.layers import Dropout

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

model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
model_relu.add(Dense(164, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))


model_relu.add(Dense(124, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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

In [37]:
```
model_drop.compile(optimizer='adam',
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])

history23 = model_drop.fit(X_train, y_train,
                         batch_size=batch_size,
                         epochs=nb_epoch, verbose=1,
                         validation_data=(X_test, y_test))
```

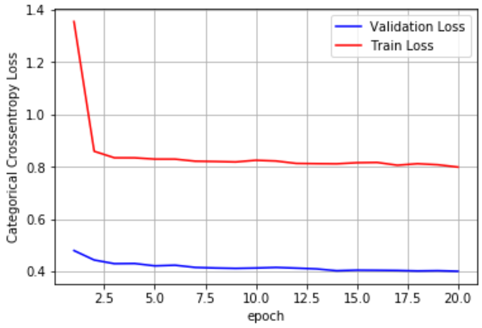
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 135us/step - loss: 1.3556 - accuracy: 0.5682 - val_loss: 0.4809 - val_accuracy:
0.8837
Epoch 2/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.8602 - accuracy: 0.7193 - val_loss: 0.4449 - val_accuracy:
0.8909
Epoch 3/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.8353 - accuracy: 0.7277 - val_loss: 0.4308 - val_accuracy:
0.8929
Epoch 4/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.8351 - accuracy: 0.7289 - val_loss: 0.4315 - val_accuracy:
0.8932
Epoch 5/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.8302 - accuracy: 0.7289 - val_loss: 0.4225 - val_accuracy:
0.8955
Epoch 6/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.8301 - accuracy: 0.7298 - val_loss: 0.4249 - val_accuracy:
0.8951
Epoch 7/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.8223 - accuracy: 0.7341 - val_loss: 0.4165 - val_accuracy:
0.8958
Epoch 8/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.8211 - accuracy: 0.7341 - val_loss: 0.4143 - val_accuracy:
0.8962
Epoch 9/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.8194 - accuracy: 0.7341 - val_loss: 0.4128 - val_accuracy:
0.8959
Epoch 10/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.8258 - accuracy: 0.7321 - val_loss: 0.4142 - val_accuracy:
0.8978
Epoch 11/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.8229 - accuracy: 0.7322 - val_loss: 0.4165 - val_accuracy:
0.8936
Epoch 12/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.8138 - accuracy: 0.7372 - val_loss: 0.4138 - val_accuracy:
0.8978
Epoch 13/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.8128 - accuracy: 0.7363 - val_loss: 0.4108 - val_accuracy:
0.8987
Epoch 14/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.8123 - accuracy: 0.7379 - val_loss: 0.4039 - val_accuracy:
0.8985
Epoch 15/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.8164 - accuracy: 0.7355 - val_loss: 0.4061 - val_accuracy:
0.8978
Epoch 16/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.8174 - accuracy: 0.7362 - val_loss: 0.4056 - val_accuracy:
0.8993
Epoch 17/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.8069 - accuracy: 0.7393 - val_loss: 0.4048 - val_accuracy:
0.8953
Epoch 18/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.8126 - accuracy: 0.7373 - val_loss: 0.4030 - val_accuracy:
0.8994
Epoch 19/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.8087 - accuracy: 0.7380 - val_loss: 0.4040 - val_accuracy:
0.8994
Epoch 20/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.7998 - accuracy: 0.7410 - val_loss: 0.4019 - val_accuracy:
0.8968
```

In [38]:
```
model_drop.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
batch_normalization_8 (Batch (None, 784)               3136      
_________________________________________________________________
dropout_3 (Dropout)          (None, 784)               0         
_________________________________________________________________
batch_normalization_9 (Batch (None, 784)               3136      
_________________________________________________________________
dropout_4 (Dropout)          (None, 784)               0         
_________________________________________________________________
batch_normalization_10 (Batc (None, 784)               3136      
_________________________________________________________________
dropout_5 (Dropout)          (None, 784)               0         
_________________________________________________________________
dense_21 (Dense)             (None, 10)                7850      
=================================================================
Total params: 17,258
Trainable params: 12,554
Non-trainable params: 4,704
_________________________________________________________________
```

In [40]:
```
score = model_drop.evaluate(X_test, y_test, verbose=0)
score11=score[0]
score12=score[1]
train_acc6=history23.history['accuracy']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax23 = plt.subplots(1,1)
ax23.set_xlabel('epoch') ; ax23.set_ylabel('Categorical Crossentropy Loss')

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


vy23 = history23.history['val_loss']
ty23 = history23.history['loss']
plt_dynamic(x, vy23, ty23, ax23)
```

```
Test score: 0.40186178770065306
Test accuracy: 0.8967999815940857
```



3) 5-Hidden layer architecture (784-216-170-136-80-38-10 architecture)

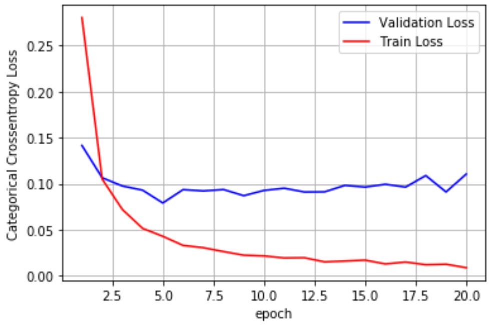
3.1 MLP + ReLU + ADAM

In [41]:
```
model_relu = Sequential()
model_relu.add(Dense(216, activation='relu', input_shape=(input_dim,),
                     kernel_initializer=he_normal(seed=None)))
model_relu.add(Dense(170, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )

model_relu.add(Dense(136, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(80, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )

model_relu.add(Dense(38, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_relu.add(Dense(output_dim, activation='softmax'))

print(model_relu.summary())

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

history31 = model_relu.fit(X_train, y_train,
                           batch_size=batch_size,
                           epochs=nb_epoch, verbose=1,
                           validation_data=(X_test, y_test))
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_22 (Dense)             (None, 216)               169560    
_________________________________________________________________
dense_23 (Dense)             (None, 170)               36890     
_________________________________________________________________
dense_24 (Dense)             (None, 136)               23256     
_________________________________________________________________
dense_25 (Dense)             (None, 80)                10960     
_________________________________________________________________
dense_26 (Dense)             (None, 38)                3078      
_________________________________________________________________
dense_27 (Dense)             (None, 10)                390       
=================================================================
Total params: 244,134
Trainable params: 244,134
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.2802 - accuracy: 0.9160 - val_loss: 0.1414 - val_accuracy:
0.9583
Epoch 2/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.1050 - accuracy: 0.9682 - val_loss: 0.1064 - val_accuracy:
0.9660
Epoch 3/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.0720 - accuracy: 0.9775 - val_loss: 0.0974 - val_accuracy:
0.9697
Epoch 4/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0515 - accuracy: 0.9841 - val_loss: 0.0929 - val_accuracy:
0.9727
Epoch 5/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.0429 - accuracy: 0.9864 - val_loss: 0.0790 - val_accuracy:
0.9780
Epoch 6/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.0330 - accuracy: 0.9892 - val_loss: 0.0935 - val_accuracy:
0.9762
Epoch 7/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0305 - accuracy: 0.9901 - val_loss: 0.0921 - val_accuracy:
0.9766
Epoch 8/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.0263 - accuracy: 0.9918 - val_loss: 0.0936 - val_accuracy:
0.9735
Epoch 9/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.0224 - accuracy: 0.9927 - val_loss: 0.0869 - val_accuracy:
0.9791
Epoch 10/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0216 - accuracy: 0.9930 - val_loss: 0.0926 - val_accuracy:
0.9774
Epoch 11/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0194 - accuracy: 0.9937 - val_loss: 0.0950 - val_accuracy:
0.9802
Epoch 12/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.0196 - accuracy: 0.9937 - val_loss: 0.0909 - val_accuracy:
0.9767
Epoch 13/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0151 - accuracy: 0.9951 - val_loss: 0.0911 - val_accuracy:
0.9792
Epoch 14/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0160 - accuracy: 0.9948 - val_loss: 0.0981 - val_accuracy:
0.9775
Epoch 15/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.0170 - accuracy: 0.9950 - val_loss: 0.0963 - val_accuracy:
0.9794
Epoch 16/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0129 - accuracy: 0.9959 - val_loss: 0.0993 - val_accuracy:
0.9774
Epoch 17/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0149 - accuracy: 0.9954 - val_loss: 0.0963 - val_accuracy:
0.9809
Epoch 18/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.0121 - accuracy: 0.9963 - val_loss: 0.1088 - val_accuracy:
0.9779
Epoch 19/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0126 - accuracy: 0.9958 - val_loss: 0.0910 - val_accuracy:
0.9817
Epoch 20/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.0088 - accuracy: 0.9972 - val_loss: 0.1104 - val_accuracy:
0.9751
```

In [43]:
```
score = model_relu.evaluate(X_test, y_test, verbose=0)
score13=score[0]
score14=score[1]
train_acc7=history31.history['accuracy']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax31 = plt.subplots(1,1)
ax31.set_xlabel('epoch') ; ax31.set_ylabel('Categorical Crossentropy Loss')

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


vy31 = history31.history['val_loss']
ty31 = history31.history['loss']
plt_dynamic(x, vy31, ty31, ax31)
```

```
Test score: 0.110397712179703474
Test accuracy: 0.9750999808311462
```



3.2 MLP + Batch-Norm on hidden Layers + AdamOptimizer

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

model_batch = Sequential()

model_relu.add(Dense(216, activation='relu', input_shape=(input_dim,),
                     kernel_initializer=he_normal(seed=None)))
model_batch.add(BatchNormalization())
model_relu.add(Dense(170, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

model_relu.add(Dense(136, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())
model_relu.add(Dense(80, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

model_relu.add(Dense(38, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_batch.add(BatchNormalization())

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

In [45]:
```
model_batch.compile(optimizer='adam', loss='categorical_crossentropy',
                    metrics=['accuracy'])

history32 = model_batch.fit(X_train, y_train,
                           batch_size=batch_size,
                           epochs=nb_epoch, verbose=1,
                           validation_data=(X_test, y_test))
```

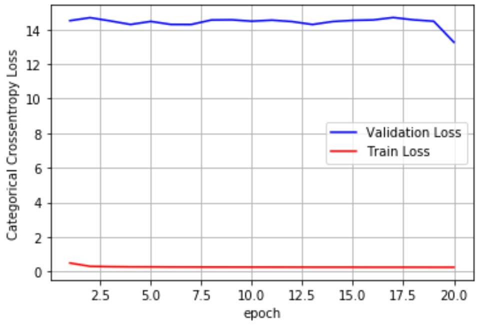
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 180us/step - loss: 0.4971 - accuracy: 0.8524 - val_loss: 14.5068 - val_accurac
y: 0.0992
Epoch 2/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.3063 - accuracy: 0.9121 - val_loss: 14.6796 - val_accurac
y: 0.0892
Epoch 3/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.2869 - accuracy: 0.9177 - val_loss: 14.4918 - val_accurac
y: 0.1009
Epoch 4/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.2753 - accuracy: 0.9219 - val_loss: 14.2871 - val_accurac
y: 0.1136
Epoch 5/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.2730 - accuracy: 0.9228 - val_loss: 14.4644 - val_accurac
y: 0.1026
Epoch 6/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.2678 - accuracy: 0.9259 - val_loss: 14.2887 - val_accurac
y: 0.1135
Epoch 7/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.2663 - accuracy: 0.9249 - val_loss: 14.2855 - val_accurac
y: 0.1137
Epoch 8/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.2630 - accuracy: 0.9259 - val_loss: 14.5450 - val_accurac
y: 0.0976
Epoch 9/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.2615 - accuracy: 0.9266 - val_loss: 14.5563 - val_accurac
y: 0.0969
Epoch 10/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.2605 - accuracy: 0.9276 - val_loss: 14.4762 - val_accurac
y: 0.1018
Epoch 11/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.2596 - accuracy: 0.9272 - val_loss: 14.5353 - val_accurac
y: 0.0982
Epoch 12/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.2581 - accuracy: 0.9282 - val_loss: 14.4534 - val_accurac
y: 0.1032
Epoch 13/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.2559 - accuracy: 0.9286 - val_loss: 14.2855 - val_accurac
y: 0.1137
Epoch 14/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.2561 - accuracy: 0.9289 - val_loss: 14.4563 - val_accurac
y: 0.1031
Epoch 15/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.2551 - accuracy: 0.9288 - val_loss: 14.5240 - val_accurac
y: 0.0989
Epoch 16/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.2529 - accuracy: 0.9294 - val_loss: 14.5482 - val_accurac
y: 0.0974
Epoch 17/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.2524 - accuracy: 0.9299 - val_loss: 14.6884 - val_accurac
y: 0.0887
Epoch 18/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.2533 - accuracy: 0.9290 - val_loss: 14.5546 - val_accurac
y: 0.0970
Epoch 19/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.2512 - accuracy: 0.9293 - val_loss: 14.4747 - val_accurac
y: 0.1019
Epoch 20/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.2515 - accuracy: 0.9297 - val_loss: 13.2606 - val_accurac
y: 0.1016
```

In [46]:
```
model_batch.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
batch_normalization_11 (Batc (None, 784)               3136      
_________________________________________________________________
batch_normalization_12 (Batc (None, 784)               3136      
_________________________________________________________________
batch_normalization_13 (Batc (None, 784)               3136      
_________________________________________________________________
batch_normalization_14 (Batc (None, 784)               3136      
_________________________________________________________________
batch_normalization_15 (Batc (None, 784)               3136      
_________________________________________________________________
dense_33 (Dense)             (None, 10)                7850      
=================================================================
Total params: 23,530
Trainable params: 15,690
Non-trainable params: 7,840
_________________________________________________________________
```

In [47]:
```
score = model_batch.evaluate(X_test, y_test, verbose=0)
score15=score[0]
score16=score[1]
train_acc8=history32.history['accuracy']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax32 = plt.subplots(1,1)
ax32.set_xlabel('epoch') ; ax32.set_ylabel('Categorical Crossentropy Loss')

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

vy32 = history32.history['val_loss']
ty32 = history32.history['loss']
plt_dynamic(x, vy32, ty32, ax32)
```

```
Test score: 13.260629919433594
Test accuracy: 0.10159999877214432
```



3.3 MLP + Dropout + AdamOptimizer

In [48]:
```
from keras.layers import Dropout

model_drop = Sequential()
model_relu.add(Dense(216, activation='relu', input_shape=(input_dim,),
                     kernel_initializer=he_normal(seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
model_relu.add(Dense(170, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_relu.add(Dense(136, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))
model_relu.add(Dense(80, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_relu.add(Dense(38, activation='relu',
                     kernel_initializer=he_normal(seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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

In [49]:
```
model_drop.compile(optimizer='adam',
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])

history33 = model_drop.fit(X_train, y_train,
                         batch_size=batch_size,
                         epochs=nb_epoch, verbose=1,
                         validation_data=(X_test, y_test))
```

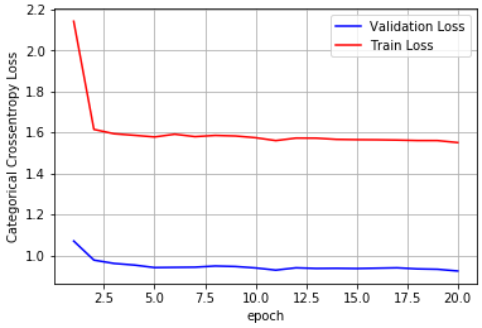
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 15s 249us/step - loss: 2.1432 - accuracy: 0.3119 - val_loss: 1.0702 - val_accurac
y: 0.8327
Epoch 2/20
60000/60000 [==============================] - 13s 220us/step - loss: 1.6152 - accuracy: 0.4400 - val_loss: 0.9769 - val_accurac
y: 0.8457
Epoch 3/20
60000/60000 [==============================] - 13s 220us/step - loss: 1.5940 - accuracy: 0.4475 - val_loss: 0.9606 - val_accurac
y: 0.8457
Epoch 4/20
60000/60000 [==============================] - 14s 231us/step - loss: 1.5865 - accuracy: 0.4529 - val_loss: 0.9527 - val_accurac
y: 0.8472
Epoch 5/20
60000/60000 [==============================] - 12s 207us/step - loss: 1.5784 - accuracy: 0.4535 - val_loss: 0.9404 - val_accurac
y: 0.8495
Epoch 6/20
60000/60000 [==============================] - 13s 221us/step - loss: 1.5918 - accuracy: 0.4486 - val_loss: 0.9414 - val_accurac
y: 0.8485
Epoch 7/20
60000/60000 [==============================] - 13s 217us/step - loss: 1.5800 - accuracy: 0.4522 - val_loss: 0.9423 - val_accurac
y: 0.8506
Epoch 8/20
60000/60000 [==============================] - 13s 210us/step - loss: 1.5859 - accuracy: 0.4500 - val_loss: 0.9480 - val_accurac
y: 0.8464
Epoch 9/20
60000/60000 [==============================] - 13s 221us/step - loss: 1.5832 - accuracy: 0.4530 - val_loss: 0.9457 - val_accurac
y: 0.8468
Epoch 10/20
60000/60000 [==============================] - 13s 214us/step - loss: 1.5747 - accuracy: 0.4529 - val_loss: 0.9386 - val_accurac
y: 0.8490
Epoch 11/20
60000/60000 [==============================] - 13s 210us/step - loss: 1.5605 - accuracy: 0.4601 - val_loss: 0.9280 - val_accurac
y: 0.8552
Epoch 12/20
60000/60000 [==============================] - 13s 213us/step - loss: 1.5729 - accuracy: 0.4553 - val_loss: 0.9393 - val_accurac
y: 0.8517
Epoch 13/20
60000/60000 [==============================] - 13s 212us/step - loss: 1.5725 - accuracy: 0.4517 - val_loss: 0.9361 - val_accurac
y: 0.8456
Epoch 14/20
60000/60000 [==============================] - 14s 226us/step - loss: 1.5665 - accuracy: 0.4575 - val_loss: 0.9368 - val_accurac
y: 0.8520
Epoch 15/20
60000/60000 [==============================] - 13s 215us/step - loss: 1.5652 - accuracy: 0.4577 - val_loss: 0.9357 - val_accurac
y: 0.8520
Epoch 16/20
60000/60000 [==============================] - 13s 216us/step - loss: 1.5646 - accuracy: 0.4608 - val_loss: 0.9374 - val_accurac
y: 0.8516
Epoch 17/20
60000/60000 [==============================] - 13s 221us/step - loss: 1.5632 - accuracy: 0.4571 - val_loss: 0.9393 - val_accurac
y: 0.8546
Epoch 18/20
60000/60000 [==============================] - 14s 230us/step - loss: 1.5609 - accuracy: 0.4591 - val_loss: 0.9339 - val_accurac
y: 0.8494
Epoch 19/20
60000/60000 [==============================] - 14s 226us/step - loss: 1.5607 - accuracy: 0.4599 - val_loss: 0.9320 - val_accurac
y: 0.8470
Epoch 20/20
60000/60000 [==============================] - 13s 219us/step - loss: 1.5509 - accuracy: 0.4657 - val_loss: 0.9236 - val_accurac
y: 0.8496
```

In [50]:
```
model_drop.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
batch_normalization_16 (Batc (None, 784)               3136      
_________________________________________________________________
dropout_6 (Dropout)          (None, 784)               0         
_________________________________________________________________
batch_normalization_17 (Batc (None, 784)               3136      
_________________________________________________________________
dropout_7 (Dropout)          (None, 784)               0         
_________________________________________________________________
batch_normalization_18 (Batc (None, 784)               3136      
_________________________________________________________________
dropout_8 (Dropout)          (None, 784)               0         
_________________________________________________________________
batch_normalization_19 (Batc (None, 784)               3136      
_________________________________________________________________
dropout_9 (Dropout)          (None, 784)               0         
_________________________________________________________________
batch_normalization_20 (Batc (None, 784)               3136      
_________________________________________________________________
dropout_10 (Dropout)         (None, 784)               0         
_________________________________________________________________
dense_39 (Dense)             (None, 10)                7850      
=================================================================
Total params: 23,530
Trainable params: 15,690
Non-trainable params: 7,840
_________________________________________________________________
```

In [53]:
```
score = model_drop.evaluate(X_test, y_test, verbose=0)
score17=score[0]
score18=score[1]
train_acc9=history33.history['accuracy']
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax33 = plt.subplots(1,1)
ax33.set_xlabel('epoch') ; ax33.set_ylabel('Categorical Crossentropy Loss')

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


vy33 = history33.history['val_loss']
ty33 = history33.history['loss']
plt_dynamic(x, vy33, ty33, ax33)
```

```
Test score: 0.923618637752533
Test accuracy: 0.8496000170707703
```



Final Summary:

In [62]:
```
from prettytable import PrettyTable

models=['2_hidden_layer MLP+ReLu+Adam',
        '2_hidden_layer MLP+Relu+adam+BN',
        '2_hidden_layer MLP+reLu+Adam+BN+Drop-out',
       '3_hidden_layer MLP+ReLu+Adam',
        '3_hidden_layer MLP+Relu+adam+BN',
        '3_hidden_layer MLP+reLu+Adam+BN+Drop-out',
       '5_hidden_layer MLP+ReLu+Adam',
        '5_hidden_layer MLP+Relu+adam+BN',
        '5_hidden_layer MLP+reLu+Adam+BN+Drop-out']

training_accuracy=[np.mean(train_acc1),np.mean(train_acc2),np.mean(train_acc3),np.mean(train_acc4),
                   np.mean(train_acc5),np.mean(train_acc6),np.mean(train_acc7),np.mean(train_acc8),
                   np.mean(train_acc9)]

test_score=[score1,score3,score5,score7,score9,score11,score13,score15,
           score17]

test_accuracy=[score2,score4,score6,score8,score10,score12,score14,
              score16,
              score18]
INDEX = [1,2,3,4,5,6,7,8,9]

# Initializing prettytable
Model_Performance = PrettyTable()
# Adding columns
Model_Performance.add_column("INDEX.",INDEX)
Model_Performance.add_column("MODEL_NAME",models)
Model_Performance.add_column("TRAINING ACCURACY",np.around(training_accuracy,2))
Model_Performance.add_column("TESTING ACCURACY",np.around(test_accuracy,2))
Model_Performance.add_column("TEST SCORE",np.around(test_score,2))

# Printing the Model_Performance
print(Model_Performance)
```

```
+--------+------------------------------------------+-------------------+------------------+------------+
| INDEX. |                MODEL_NAME                | TRAINING ACCURACY | TESTING ACCURACY | TEST SCORE |
+--------+------------------------------------------+-------------------+------------------+------------+
|   1    |       2_hidden_layer MLP+ReLu+Adam       |        0.99       |       0.98       |    0.1     |
|   2    |     2_hidden_layer MLP+Relu+adam+BN      |        0.99       |       0.98       |    0.09    |
|   3    | 2_hidden_layer MLP+reLu+Adam+BN+Drop-out |        0.97       |       0.98       |    0.06    |
|   4    |       3_hidden_layer MLP+ReLu+Adam       |        0.99       |       0.98       |    0.1     |
|   5    |     3_hidden_layer MLP+Relu+adam+BN      |        0.92       |       0.87       |    0.6     |
|   6    | 3_hidden_layer MLP+reLu+Adam+BN+Drop-out |        0.73       |       0.9        |    0.4     |
|   7    |       5_hidden_layer MLP+ReLu+Adam       |        0.99       |       0.98       |    0.11    |
|   8    |     5_hidden_layer MLP+Relu+adam+BN      |        0.92       |       0.1        |   13.26    |
|   9    | 5_hidden_layer MLP+reLu+Adam+BN+Drop-out |        0.45       |       0.85       |    0.92    |
+--------+------------------------------------------+-------------------+------------------+------------+
```