prabhudayala@gmail.com_12

July 15, 2019

0.1 Keras – MLPs on MNIST

[1]:
```
# if you keras is not using tensorflow as backend set␣
↪"KERAS_BACKEND=tensorflow" use this command
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
```

```
Using TensorFlow backend.
```

[2]:
```
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()
```

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

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

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

[5]:
```
# if you observe the input shape its 2 dimensional vector
# for each image we have a (28*28) vector
```

```python
# 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])
```

[6]:

```python
# 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)
```

[7]:

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

```
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    3   18   18   18  126  136  175   26  166  255
  247  127    0    0    0    0    0    0    0    0    0    0    0    0   30   36   94  154
  170  253  253  253  253  253  225  172  253  242  195   64    0    0    0    0    0    0
    0    0    0    0    0   49  238  253  253  253  253  253  253  253  253  251   93   82
   82   56   39    0    0    0    0    0    0    0    0    0    0    0    0   18  219  253
  253  253  253  253  198  182  247  241    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0   80  156  107  253  253  205   11    0   43  154
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0   14    1  154  253   90    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0  139  253  190    2    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0   11  190  253   70    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0   35  241
  225  160  108    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0   81  240  253  253  119   25    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0   45  186  253  253  150   27    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0   16   93  252  253  187
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0  249  253  249   64    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0   46  130  183  253
```

```
 253 207   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253
 253 201  78   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0  18 171 219 253 253 253 253 195
  80   9   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0 136 253 253 253 212 135 132  16
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
```

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

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

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

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

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

```
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.17647059 0.72941176 0.99215686 0.99215686
0.58823529 0.10588235 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.0627451  0.36470588 0.98823529 0.99215686 0.73333333
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.97647059 0.99215686 0.97647059 0.25098039 0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.18039216 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.        ]
```

[10]:
```
# here we are having a class number for each image
print("Class label of first image :", y_train[0])

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

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

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

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

[11]:
```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

[12]:
```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
```

Softmax classifier
Model 1. 2 hidden layer, ReLU, Dropout, BatchNormalization and AdamOptimizer

[13]:
```
# https://stackoverflow.com/questions/34716454/
 ↪where-do-i-call-the-batchnormalization-function-in-keras
```

```python
model_1 = Sequential()

model_1.add(Dense(128, activation='relu',␣
↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None),␣
↪input_shape=(input_dim,)) )
model_1.add(BatchNormalization())
model_1.add(Dropout(0.5))

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.
↪0, stddev=0.176, seed=None)) )
model_1.add(BatchNormalization())
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

```
WARNING: Logging before flag parsing goes to stderr.
W0715 18:42:29.220162 11376 deprecation_wrapper.py:119] From
C:\Users\user\Anaconda3\envs\tensorflow_cpu\lib\site-
packages\keras\backend\tensorflow_backend.py:74: The name tf.get_default_graph
is deprecated. Please use tf.compat.v1.get_default_graph instead.

W0715 18:42:29.232118 11376 deprecation_wrapper.py:119] From
C:\Users\user\Anaconda3\envs\tensorflow_cpu\lib\site-
packages\keras\backend\tensorflow_backend.py:517: The name tf.placeholder is
deprecated. Please use tf.compat.v1.placeholder instead.

W0715 18:42:29.234127 11376 deprecation_wrapper.py:119] From
C:\Users\user\Anaconda3\envs\tensorflow_cpu\lib\site-
packages\keras\backend\tensorflow_backend.py:4115: The name tf.random_normal is
deprecated. Please use tf.random.normal instead.

W0715 18:42:29.294940 11376 deprecation_wrapper.py:119] From
C:\Users\user\Anaconda3\envs\tensorflow_cpu\lib\site-
packages\keras\backend\tensorflow_backend.py:133: The name
tf.placeholder_with_default is deprecated. Please use
tf.compat.v1.placeholder_with_default instead.

W0715 18:42:29.310896 11376 deprecation.py:506] From
C:\Users\user\Anaconda3\envs\tensorflow_cpu\lib\site-
packages\keras\backend\tensorflow_backend.py:3445: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed
in a future version.
```

```
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
W0715 18:42:29.391703 11376 deprecation_wrapper.py:119] From
C:\Users\user\Anaconda3\envs\tensorflow_cpu\lib\site-
packages\keras\backend\tensorflow_backend.py:4138: The name tf.random_uniform is
deprecated. Please use tf.random.uniform instead.
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
batch_normalization_1 (Batch (None, 128)               512
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
batch_normalization_2 (Batch (None, 64)                256
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650
=================================================================
Total params: 110,154
Trainable params: 109,770
Non-trainable params: 384
_________________________________________________________________
```

[14]:
```
model_1.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,␣
 ↪verbose=1, validation_data=(X_test, Y_test))
```

```
W0715 18:42:31.010054 11376 deprecation_wrapper.py:119] From
C:\Users\user\Anaconda3\envs\tensorflow_cpu\lib\site-
packages\keras\optimizers.py:790: The name tf.train.Optimizer is deprecated.
Please use tf.compat.v1.train.Optimizer instead.

W0715 18:42:31.090867 11376 deprecation.py:323] From
C:\Users\user\Anaconda3\envs\tensorflow_cpu\lib\site-
packages\tensorflow\python\ops\math_grad.py:1250:
add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is
deprecated and will be removed in a future version.
```

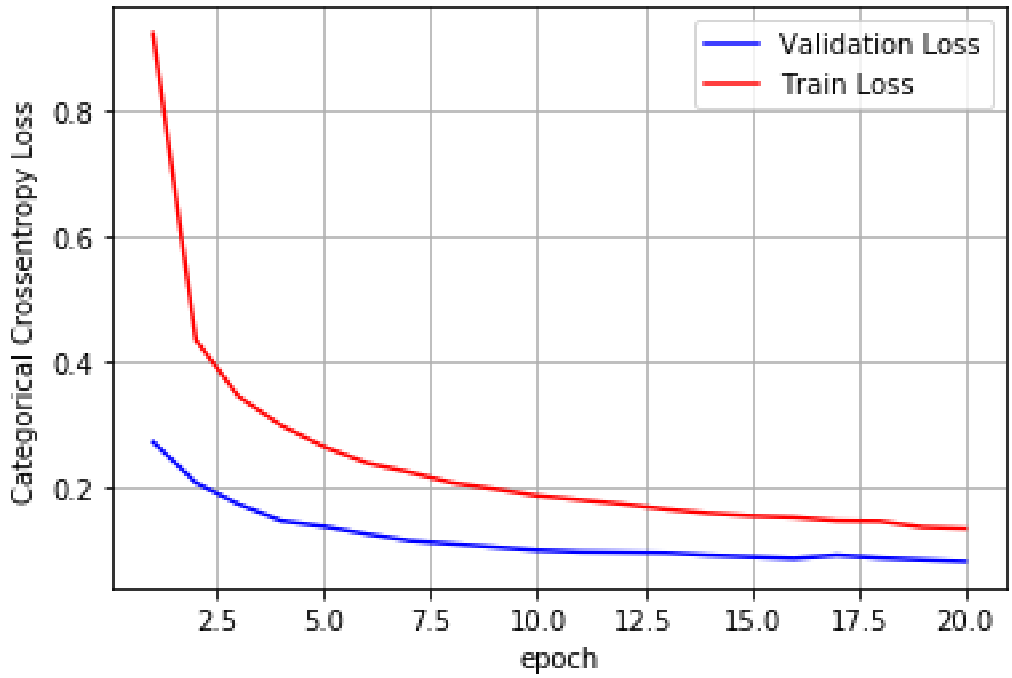
```
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 2s 30us/step - loss: 0.9239 -
acc: 0.7141 - val_loss: 0.2713 - val_acc: 0.9185
Epoch 2/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.4345 -
acc: 0.8711 - val_loss: 0.2065 - val_acc: 0.9355
Epoch 3/20
60000/60000 [==============================] - 1s 22us/step - loss: 0.3440 -
acc: 0.8991 - val_loss: 0.1721 - val_acc: 0.9464
Epoch 4/20
60000/60000 [==============================] - 1s 22us/step - loss: 0.2976 -
acc: 0.9140 - val_loss: 0.1457 - val_acc: 0.9540
Epoch 5/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.2635 -
acc: 0.9225 - val_loss: 0.1367 - val_acc: 0.9558
Epoch 6/20
60000/60000 [==============================] - 1s 22us/step - loss: 0.2377 -
acc: 0.9308 - val_loss: 0.1243 - val_acc: 0.9610
Epoch 7/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.2229 -
acc: 0.9361 - val_loss: 0.1137 - val_acc: 0.9641
Epoch 8/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.2063 -
acc: 0.9394 - val_loss: 0.1085 - val_acc: 0.9660
Epoch 9/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.1965 -
acc: 0.9433 - val_loss: 0.1033 - val_acc: 0.9676
Epoch 10/20
60000/60000 [==============================] - 1s 22us/step - loss: 0.1855 -
acc: 0.9469 - val_loss: 0.0983 - val_acc: 0.9708
Epoch 11/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.1790 -
acc: 0.9478 - val_loss: 0.0957 - val_acc: 0.9707
Epoch 12/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.1723 -
acc: 0.9490 - val_loss: 0.0950 - val_acc: 0.9719
Epoch 13/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.1640 -
acc: 0.9512 - val_loss: 0.0941 - val_acc: 0.9716
Epoch 14/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.1575 -
acc: 0.9540 - val_loss: 0.0905 - val_acc: 0.9726
Epoch 15/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.1534 -
```

```
acc: 0.9549 - val_loss: 0.0879 - val_acc: 0.9734
Epoch 16/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.1511 -
acc: 0.9555 - val_loss: 0.0853 - val_acc: 0.9741
Epoch 17/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.1459 -
acc: 0.9570 - val_loss: 0.0907 - val_acc: 0.9736
Epoch 18/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.1450 -
acc: 0.9575 - val_loss: 0.0861 - val_acc: 0.9740
Epoch 19/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.1356 -
acc: 0.9597 - val_loss: 0.0833 - val_acc: 0.9756
Epoch 20/20
60000/60000 [==============================] - 1s 21us/step - loss: 0.1334 -
acc: 0.9597 - val_loss: 0.0806 - val_acc: 0.9764
```

[15]:

```
score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

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

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

```
Test score: 0.08059193601340521
Test accuracy: 0.9764
```



Model 1. 2 hidden layer, ReLU, Dropout and AdamOptimizer

[16]:
```
# https://stackoverflow.com/questions/34716454/
 ↪where-do-i-call-the-batchnormalization-function-in-keras



model_1 = Sequential()

model_1.add(Dense(128, activation='relu',␣
 ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None),␣
 ↪input_shape=(input_dim,)) )
model_1.add(Dropout(0.5))

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.
 ↪0, stddev=0.176, seed=None)) )
model_1.add(Dropout(0.5))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 128)               100480
```

```
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                650
=================================================================
Total params: 109,386
Trainable params: 109,386
Non-trainable params: 0
_________________________________________________________________
```

[17]:
```
model_1.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,␣
 ↪verbose=1, validation_data=(X_test, Y_test))
```

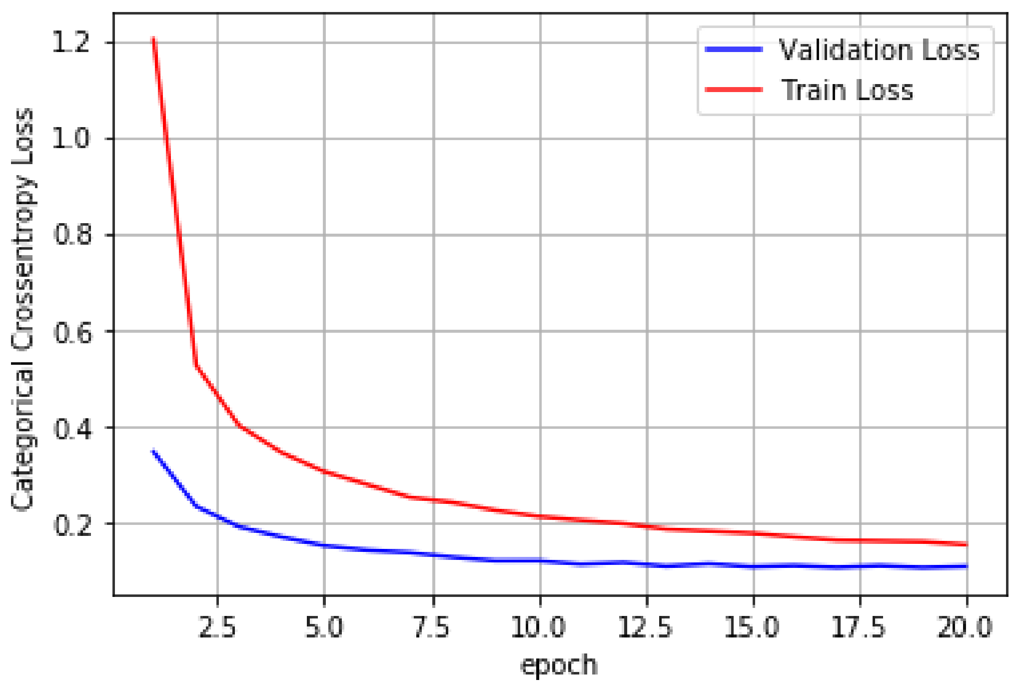
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 1s 22us/step - loss: 1.2017 -
acc: 0.6102 - val_loss: 0.3453 - val_acc: 0.9066
Epoch 2/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.5247 -
acc: 0.8416 - val_loss: 0.2322 - val_acc: 0.9323
Epoch 3/20
60000/60000 [==============================] - 1s 17us/step - loss: 0.4000 -
acc: 0.8858 - val_loss: 0.1899 - val_acc: 0.9445
Epoch 4/20
60000/60000 [==============================] - 1s 17us/step - loss: 0.3436 -
acc: 0.9016 - val_loss: 0.1686 - val_acc: 0.9523
Epoch 5/20
60000/60000 [==============================] - 1s 17us/step - loss: 0.3039 -
acc: 0.9153 - val_loss: 0.1504 - val_acc: 0.9574
Epoch 6/20
60000/60000 [==============================] - 1s 17us/step - loss: 0.2777 -
acc: 0.9220 - val_loss: 0.1411 - val_acc: 0.9607
Epoch 7/20
60000/60000 [==============================] - 1s 17us/step - loss: 0.2511 -
acc: 0.9294 - val_loss: 0.1360 - val_acc: 0.9617
Epoch 8/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2404 -
acc: 0.9340 - val_loss: 0.1266 - val_acc: 0.9651
Epoch 9/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2240 -
```

```
acc: 0.9375 - val_loss: 0.1191 - val_acc: 0.9672
Epoch 10/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2114 -
acc: 0.9424 - val_loss: 0.1192 - val_acc: 0.9669
Epoch 11/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2034 -
acc: 0.9430 - val_loss: 0.1119 - val_acc: 0.9690
Epoch 12/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.1964 -
acc: 0.9446 - val_loss: 0.1155 - val_acc: 0.9685
Epoch 13/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.1847 -
acc: 0.9475 - val_loss: 0.1077 - val_acc: 0.9690
Epoch 14/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.1807 -
acc: 0.9495 - val_loss: 0.1129 - val_acc: 0.9685
Epoch 15/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.1763 -
acc: 0.9503 - val_loss: 0.1066 - val_acc: 0.9713
Epoch 16/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.1687 -
acc: 0.9525 - val_loss: 0.1092 - val_acc: 0.9701
Epoch 17/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.1618 -
acc: 0.9547 - val_loss: 0.1053 - val_acc: 0.9736
Epoch 18/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.1603 -
acc: 0.9538 - val_loss: 0.1090 - val_acc: 0.9725
Epoch 19/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.1588 -
acc: 0.9558 - val_loss: 0.1049 - val_acc: 0.9724
Epoch 20/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.1526 -
acc: 0.9562 - val_loss: 0.1074 - val_acc: 0.9727
```

[18]:
```
score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

```
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size,␣
 ↪epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

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

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

```
Test score: 0.10739179229954025
Test accuracy: 0.9727
```



Model 1. 2 hidden layer, ReLU, Dropout, BatchNormalization and AdamOptimizer

[19]:
```
# https://stackoverflow.com/questions/34716454/
 ↪where-do-i-call-the-batchnormalization-function-in-keras
```

```
model_1 = Sequential()

model_1.add(Dense(128, activation='relu',␣
 ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None),␣
 ↪input_shape=(input_dim,)) )
model_1.add(BatchNormalization())
model_1.add(Dropout(0.2))

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.
 ↪0, stddev=0.176, seed=None)) )
model_1.add(BatchNormalization())
model_1.add(Dropout(0.2))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 128)               100480
_________________________________________________________________
batch_normalization_3 (Batch (None, 128)               512
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_8 (Dense)              (None, 64)                8256
_________________________________________________________________
batch_normalization_4 (Batch (None, 64)                256
_________________________________________________________________
dropout_6 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                650
=================================================================
Total params: 110,154
Trainable params: 109,770
Non-trainable params: 384
_________________________________________________________________
```

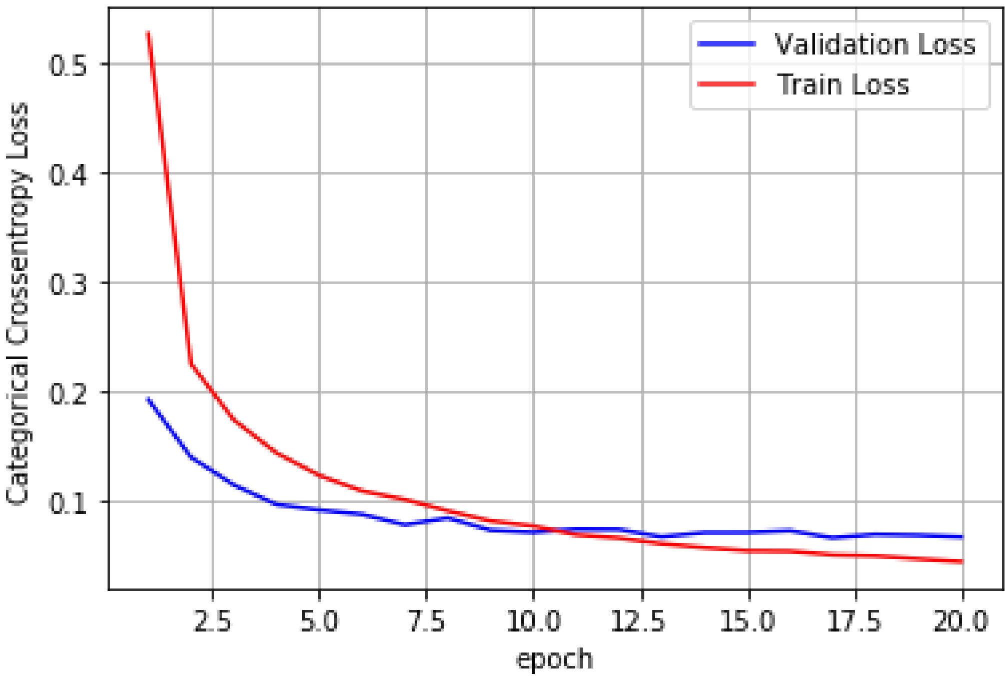
[20]:
```
model_1.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

history = model_1.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 [==============================] - 2s 34us/step - loss: 0.5259 -
acc: 0.8396 - val_loss: 0.1926 - val_acc: 0.9411
Epoch 2/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.2251 -
acc: 0.9331 - val_loss: 0.1399 - val_acc: 0.9580
Epoch 3/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.1740 -
acc: 0.9471 - val_loss: 0.1148 - val_acc: 0.9629
Epoch 4/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.1438 -
acc: 0.9562 - val_loss: 0.0969 - val_acc: 0.9704
Epoch 5/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.1234 -
acc: 0.9619 - val_loss: 0.0919 - val_acc: 0.9718
Epoch 6/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.1093 -
acc: 0.9660 - val_loss: 0.0879 - val_acc: 0.9730
Epoch 7/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.1014 -
acc: 0.9682 - val_loss: 0.0784 - val_acc: 0.9754
Epoch 8/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.0911 -
acc: 0.9715 - val_loss: 0.0847 - val_acc: 0.9753
Epoch 9/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.0820 -
acc: 0.9742 - val_loss: 0.0734 - val_acc: 0.9780
Epoch 10/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.0773 -
acc: 0.9757 - val_loss: 0.0716 - val_acc: 0.9774
Epoch 11/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.0693 -
acc: 0.9778 - val_loss: 0.0744 - val_acc: 0.9782
Epoch 12/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.0663 -
acc: 0.9785 - val_loss: 0.0741 - val_acc: 0.9781
Epoch 13/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.0611 -
acc: 0.9802 - val_loss: 0.0675 - val_acc: 0.9800
Epoch 14/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.0576 -
acc: 0.9809 - val_loss: 0.0713 - val_acc: 0.9787
Epoch 15/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.0547 -
acc: 0.9818 - val_loss: 0.0713 - val_acc: 0.9791
Epoch 16/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.0543 -
```

```
acc: 0.9820 - val_loss: 0.0731 - val_acc: 0.9792
Epoch 17/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.0510 -
acc: 0.9832 - val_loss: 0.0665 - val_acc: 0.9804
Epoch 18/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.0500 -
acc: 0.9829 - val_loss: 0.0697 - val_acc: 0.9801
Epoch 19/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.0473 -
acc: 0.9845 - val_loss: 0.0689 - val_acc: 0.9810
Epoch 20/20
60000/60000 [==============================] - 1s 23us/step - loss: 0.0448 -
acc: 0.9852 - val_loss: 0.0674 - val_acc: 0.9807
```

[21]:
```
score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

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

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

```
Test score: 0.06737108307137678
Test accuracy: 0.9807
```



This model is stable after 10 epochs. training further will make the model overfit to train data.

Model 1. 2 hidden layer, ReLU, Dropout, BatchNormalization and AdamOptimizer

[22]:
```
# https://stackoverflow.com/questions/34716454/
↪where-do-i-call-the-batchnormalization-function-in-keras




model_1 = Sequential()

model_1.add(Dense(128, activation='relu',␣
↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None),␣
↪input_shape=(input_dim,)) )
model_1.add(BatchNormalization())
model_1.add(Dropout(0.8))

model_1.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.
↪0, stddev=0.176, seed=None)) )
model_1.add(BatchNormalization())
model_1.add(Dropout(0.8))

model_1.add(Dense(output_dim, activation='softmax'))

model_1.summary()
```

```
W0715 18:44:06.593636 11376 nn_ops.py:4224] Large dropout rate: 0.8 (>0.5). In
TensorFlow 2.x, dropout() uses dropout rate instead of keep_prob. Please ensure
that this is intended.
W0715 18:44:06.665421 11376 nn_ops.py:4224] Large dropout rate: 0.8 (>0.5). In
TensorFlow 2.x, dropout() uses dropout rate instead of keep_prob. Please ensure
that this is intended.
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 128)               100480
_________________________________________________________________
batch_normalization_5 (Batch (None, 128)               512
_________________________________________________________________
dropout_7 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_11 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_6 (Batch (None, 64)                256
_________________________________________________________________
dropout_8 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_12 (Dense)             (None, 10)                650
=================================================================
Total params: 110,154
Trainable params: 109,770
Non-trainable params: 384
_________________________________________________________________
```

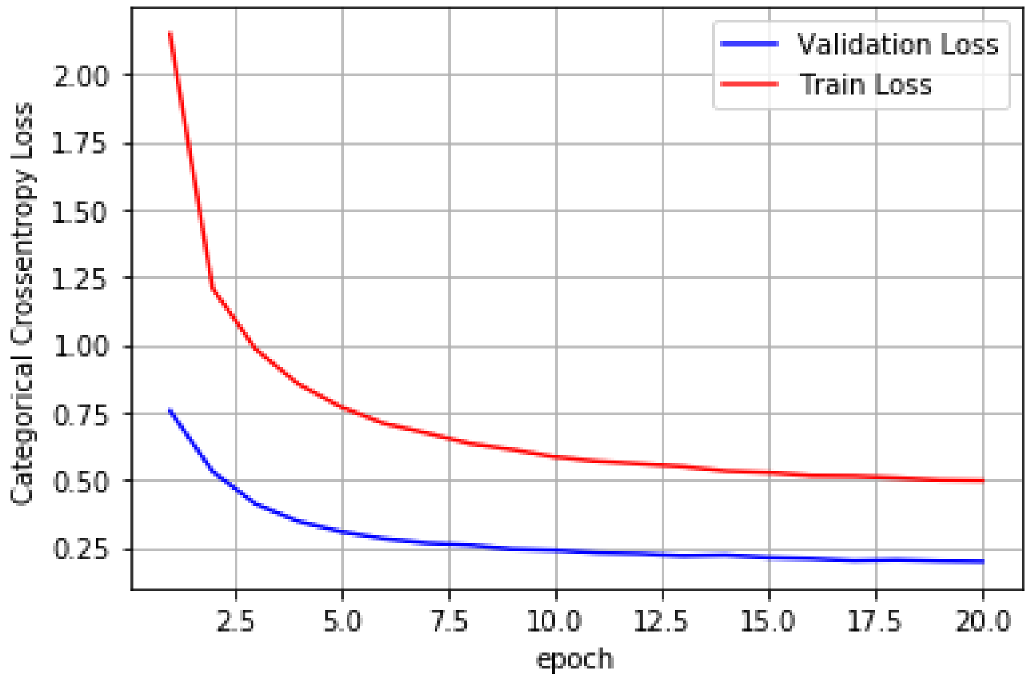
[23]:
```python
model_1.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

history = model_1.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 [==============================] - 2s 39us/step - loss: 2.1479 -
acc: 0.3531 - val_loss: 0.7555 - val_acc: 0.8517
Epoch 2/20
60000/60000 [==============================] - 2s 28us/step - loss: 1.2069 -
acc: 0.5817 - val_loss: 0.5305 - val_acc: 0.8823
Epoch 3/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.9829 -
acc: 0.6711 - val_loss: 0.4108 - val_acc: 0.8977
Epoch 4/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.8540 -
acc: 0.7230 - val_loss: 0.3469 - val_acc: 0.9060
```

```
Epoch 5/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.7693 -
acc: 0.7580 - val_loss: 0.3085 - val_acc: 0.9151
Epoch 6/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.7085 -
acc: 0.7846 - val_loss: 0.2826 - val_acc: 0.9192
Epoch 7/20
60000/60000 [==============================] - 2s 29us/step - loss: 0.6728 -
acc: 0.7994 - val_loss: 0.2663 - val_acc: 0.9245
Epoch 8/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.6345 -
acc: 0.8139 - val_loss: 0.2589 - val_acc: 0.9251
Epoch 9/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.6115 -
acc: 0.8207 - val_loss: 0.2438 - val_acc: 0.9298
Epoch 10/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.5844 -
acc: 0.8289 - val_loss: 0.2397 - val_acc: 0.9298
Epoch 11/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.5683 -
acc: 0.8353 - val_loss: 0.2299 - val_acc: 0.9320
Epoch 12/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.5589 -
acc: 0.8387 - val_loss: 0.2266 - val_acc: 0.9326
Epoch 13/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.5487 -
acc: 0.8437 - val_loss: 0.2190 - val_acc: 0.9345
Epoch 14/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.5323 -
acc: 0.8486 - val_loss: 0.2221 - val_acc: 0.9364
Epoch 15/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.5266 -
acc: 0.8495 - val_loss: 0.2125 - val_acc: 0.9379
Epoch 16/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.5153 -
acc: 0.8545 - val_loss: 0.2090 - val_acc: 0.9400
Epoch 17/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.5137 -
acc: 0.8551 - val_loss: 0.2014 - val_acc: 0.9419
Epoch 18/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.5073 -
acc: 0.8561 - val_loss: 0.2042 - val_acc: 0.9410
Epoch 19/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.4995 -
acc: 0.8579 - val_loss: 0.2001 - val_acc: 0.9438
Epoch 20/20
60000/60000 [==============================] - 2s 28us/step - loss: 0.4976 -
acc: 0.8595 - val_loss: 0.1978 - val_acc: 0.9443
```

[24]:
```
score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

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

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

```
Test score: 0.19782737234905362
Test accuracy: 0.9443
```




```
Due to high dropout rate the model became stable at a high loss than other
model and after 10 epoch it is not moving towards the minima
```

Model 2. 3 hidden layer, ReLU, Dropout, BatchNormalization and AdamOptimizer

[25]:
```
# https://stackoverflow.com/questions/34716454/
↪where-do-i-call-the-batchnormalization-function-in-keras




model_2 = Sequential()

model_2.add(Dense(512, activation='relu', input_shape=(input_dim,),␣
↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_2.add(BatchNormalization())
model_2.add(Dropout(0.5))

model_2.add(Dense(128, activation='relu',␣
↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_2.add(BatchNormalization())
model_2.add(Dropout(0.5))

model_2.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.
↪0, stddev=0.176, seed=None)) )
model_2.add(BatchNormalization())
model_2.add(Dropout(0.5))
```

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


model_2.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_13 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_7 (Batch (None, 512)               2048
_________________________________________________________________
dropout_9 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_14 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_8 (Batch (None, 128)               512
_________________________________________________________________
dropout_10 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_15 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_9 (Batch (None, 64)                256
_________________________________________________________________
dropout_11 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_16 (Dense)             (None, 10)                650
=================================================================
Total params: 479,306
Trainable params: 477,898
Non-trainable params: 1,408
_________________________________________________________________
```

[26]:
```
model_2.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

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

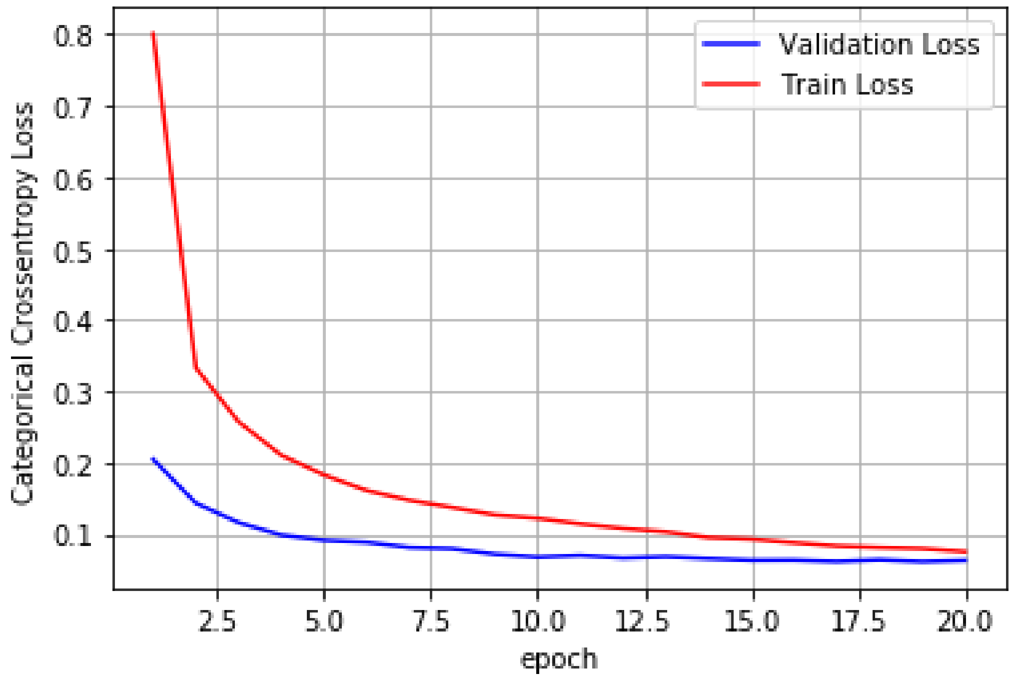
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.8009 -
acc: 0.7527 - val_loss: 0.2047 - val_acc: 0.9364
Epoch 2/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.3331 -
acc: 0.9038 - val_loss: 0.1434 - val_acc: 0.9528
Epoch 3/20
```

```
60000/60000 [==============================] - 5s 82us/step - loss: 0.2573 -
acc: 0.9271 - val_loss: 0.1162 - val_acc: 0.9629
Epoch 4/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.2104 -
acc: 0.9410 - val_loss: 0.0986 - val_acc: 0.9690
Epoch 5/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.1828 -
acc: 0.9490 - val_loss: 0.0913 - val_acc: 0.9751
Epoch 6/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.1611 -
acc: 0.9559 - val_loss: 0.0885 - val_acc: 0.9738
Epoch 7/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.1474 -
acc: 0.9576 - val_loss: 0.0812 - val_acc: 0.9766
Epoch 8/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.1373 -
acc: 0.9615 - val_loss: 0.0796 - val_acc: 0.9756
Epoch 9/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.1270 -
acc: 0.9643 - val_loss: 0.0718 - val_acc: 0.9788
Epoch 10/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.1222 -
acc: 0.9661 - val_loss: 0.0678 - val_acc: 0.9805
Epoch 11/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.1143 -
acc: 0.9681 - val_loss: 0.0701 - val_acc: 0.9795
Epoch 12/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.1080 -
acc: 0.9700 - val_loss: 0.0665 - val_acc: 0.9816
Epoch 13/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.1032 -
acc: 0.9711 - val_loss: 0.0684 - val_acc: 0.9804
Epoch 14/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0951 -
acc: 0.9730 - val_loss: 0.0657 - val_acc: 0.9819
Epoch 15/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0929 -
acc: 0.9742 - val_loss: 0.0633 - val_acc: 0.9828
Epoch 16/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0878 -
acc: 0.9751 - val_loss: 0.0635 - val_acc: 0.9835
Epoch 17/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0833 -
acc: 0.9767 - val_loss: 0.0616 - val_acc: 0.9836
Epoch 18/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0812 -
acc: 0.9769 - val_loss: 0.0640 - val_acc: 0.9819
Epoch 19/20
```

```
60000/60000 [==============================] - 5s 82us/step - loss: 0.0797 -
acc: 0.9778 - val_loss: 0.0614 - val_acc: 0.9819
Epoch 20/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0755 -
acc: 0.9777 - val_loss: 0.0632 - val_acc: 0.9823
```

[27]:
```
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))

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

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

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

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

```
Test score: 0.06315016652900377
Test accuracy: 0.9823
```



Model 2. 3 hidden layer, ReLU, Dropout and AdamOptimizer

[28]:
```
# https://stackoverflow.com/questions/34716454/
 ↪where-do-i-call-the-batchnormalization-function-in-keras



model_2 = Sequential()

model_2.add(Dense(512, activation='relu', input_shape=(input_dim,),␣
 ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_2.add(Dropout(0.5))

model_2.add(Dense(128, activation='relu',␣
 ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_2.add(Dropout(0.5))

model_2.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.
 ↪0, stddev=0.176, seed=None)) )
model_2.add(Dropout(0.5))

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


model_2.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_17 (Dense)             (None, 512)               401920
_________________________________________________________________
dropout_12 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_18 (Dense)             (None, 128)               65664
_________________________________________________________________
dropout_13 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_19 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout_14 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_20 (Dense)             (None, 10)                650
=================================================================
Total params: 476,490
Trainable params: 476,490
Non-trainable params: 0
_________________________________________________________________
```

[29]:

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

history = model_2.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 [==============================] - 5s 84us/step - loss: 1.1420 -
acc: 0.6327 - val_loss: 0.2604 - val_acc: 0.9287
Epoch 2/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.4306 -
acc: 0.8802 - val_loss: 0.1797 - val_acc: 0.9508
Epoch 3/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.3154 -
acc: 0.9168 - val_loss: 0.1521 - val_acc: 0.9590
Epoch 4/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.2582 -
acc: 0.9323 - val_loss: 0.1386 - val_acc: 0.9640
Epoch 5/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.2209 -
acc: 0.9430 - val_loss: 0.1261 - val_acc: 0.9664
Epoch 6/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.2000 -
acc: 0.9481 - val_loss: 0.1072 - val_acc: 0.9723
```

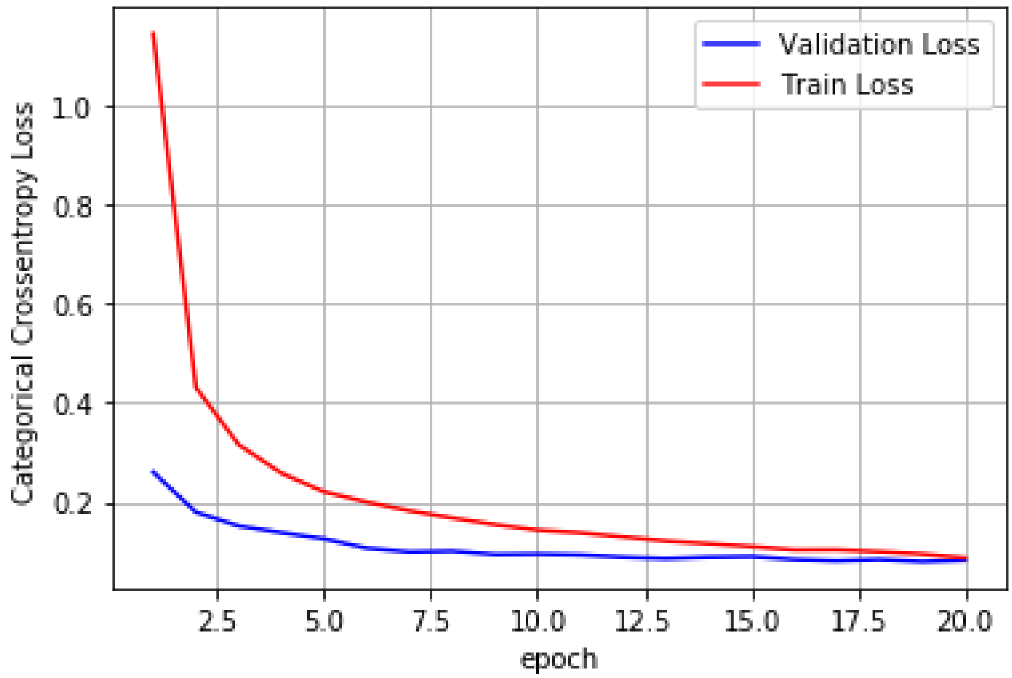
```
Epoch 7/20
60000/60000 [==============================] - ETA: 0s - loss: 0.1823 - acc:
0.952 - 4s 72us/step - loss: 0.1826 - acc: 0.9527 - val_loss: 0.0997 - val_acc:
0.9737
Epoch 8/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.1682 -
acc: 0.9566 - val_loss: 0.1014 - val_acc: 0.9734
Epoch 9/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.1548 -
acc: 0.9606 - val_loss: 0.0945 - val_acc: 0.9763
Epoch 10/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.1437 -
acc: 0.9630 - val_loss: 0.0952 - val_acc: 0.9754
Epoch 11/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.1382 -
acc: 0.9647 - val_loss: 0.0938 - val_acc: 0.9760
Epoch 12/20
60000/60000 [==============================] - 5s 75us/step - loss: 0.1300 -
acc: 0.9667 - val_loss: 0.0891 - val_acc: 0.9778
Epoch 13/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.1217 -
acc: 0.9685 - val_loss: 0.0863 - val_acc: 0.9786
Epoch 14/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.1162 -
acc: 0.9703 - val_loss: 0.0892 - val_acc: 0.9780
Epoch 15/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.1105 -
acc: 0.9711 - val_loss: 0.0904 - val_acc: 0.9790
Epoch 16/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.1038 -
acc: 0.9720 - val_loss: 0.0845 - val_acc: 0.9794
Epoch 17/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.1040 -
acc: 0.9728 - val_loss: 0.0818 - val_acc: 0.9784
Epoch 18/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.1001 -
acc: 0.9741 - val_loss: 0.0846 - val_acc: 0.9800
Epoch 19/20
60000/60000 [==============================] - 4s 72us/step - loss: 0.0953 -
acc: 0.9750 - val_loss: 0.0802 - val_acc: 0.9796
Epoch 20/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0877 -
acc: 0.9768 - val_loss: 0.0831 - val_acc: 0.9807
```

[30]:

```python
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))

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

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

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

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

```
Test score: 0.08312617581425139
Test accuracy: 0.9807
```



Model 3. 3 hidden layer, ReLU, Dropout, BatchNormalization and AdamOptimizer

[31]:
```
# https://stackoverflow.com/questions/34716454/
↪where-do-i-call-the-batchnormalization-function-in-keras



model_2 = Sequential()

model_2.add(Dense(512, activation='relu', input_shape=(input_dim,),␣
↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_2.add(BatchNormalization())
model_2.add(Dropout(0.2))

model_2.add(Dense(128, activation='relu',␣
↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_2.add(BatchNormalization())
model_2.add(Dropout(0.2))

model_2.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.
↪0, stddev=0.176, seed=None)) )
model_2.add(BatchNormalization())
model_2.add(Dropout(0.2))

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

```
model_2.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_21 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_10 (Batc (None, 512)               2048
_________________________________________________________________
dropout_15 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_22 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_11 (Batc (None, 128)               512
_________________________________________________________________
dropout_16 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_23 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_12 (Batc (None, 64)                256
_________________________________________________________________
dropout_17 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_24 (Dense)             (None, 10)                650
=================================================================
Total params: 479,306
Trainable params: 477,898
Non-trainable params: 1,408
_________________________________________________________________
```

[32]:
```
model_2.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

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

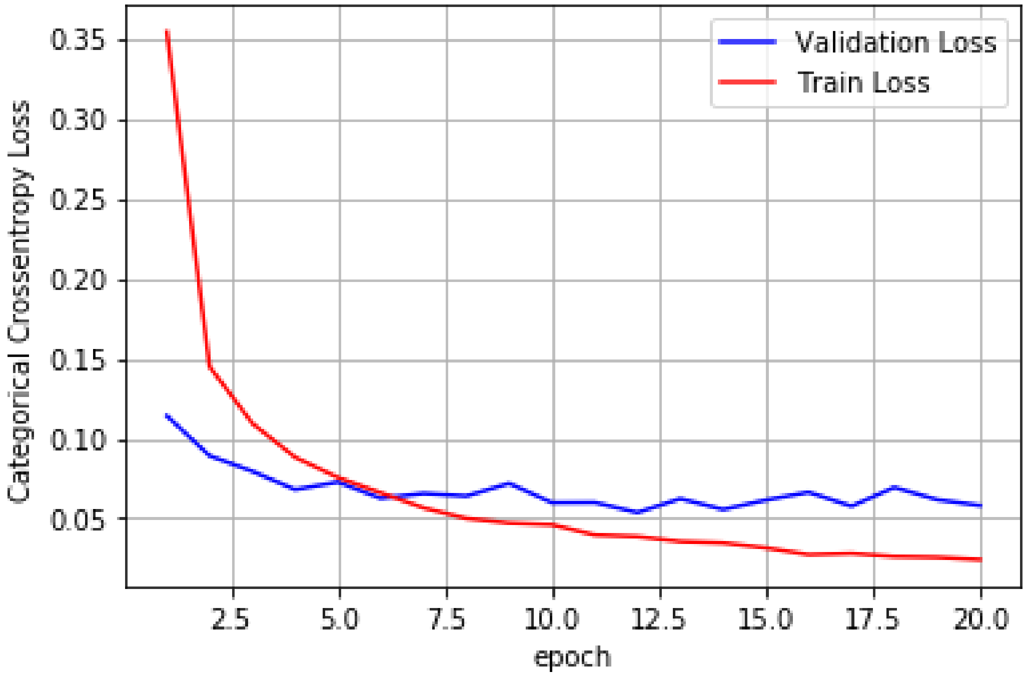
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 104us/step - loss: 0.3539 -
acc: 0.8956 - val_loss: 0.1141 - val_acc: 0.9654
Epoch 2/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.1448 -
acc: 0.9573 - val_loss: 0.0891 - val_acc: 0.9732
Epoch 3/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.1093 -
acc: 0.9661 - val_loss: 0.0794 - val_acc: 0.9747
```

```
Epoch 4/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0881 -
acc: 0.9730 - val_loss: 0.0681 - val_acc: 0.9787
Epoch 5/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.0756 -
acc: 0.9760 - val_loss: 0.0728 - val_acc: 0.9780
Epoch 6/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0660 -
acc: 0.9797 - val_loss: 0.0626 - val_acc: 0.9802
Epoch 7/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0566 -
acc: 0.9823 - val_loss: 0.0657 - val_acc: 0.9810
Epoch 8/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0501 -
acc: 0.9838 - val_loss: 0.0640 - val_acc: 0.9812
Epoch 9/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0472 -
acc: 0.9847 - val_loss: 0.0719 - val_acc: 0.9793
Epoch 10/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0461 -
acc: 0.9853 - val_loss: 0.0599 - val_acc: 0.9829
Epoch 11/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0397 -
acc: 0.9870 - val_loss: 0.0600 - val_acc: 0.9814
Epoch 12/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0387 -
acc: 0.9870 - val_loss: 0.0537 - val_acc: 0.9841
Epoch 13/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0356 -
acc: 0.9885 - val_loss: 0.0624 - val_acc: 0.9825
Epoch 14/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.0347 -
acc: 0.9884 - val_loss: 0.0557 - val_acc: 0.9847
Epoch 15/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0316 -
acc: 0.9898 - val_loss: 0.0615 - val_acc: 0.9836
Epoch 16/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0274 -
acc: 0.9906 - val_loss: 0.0664 - val_acc: 0.9829
Epoch 17/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0282 -
acc: 0.9906 - val_loss: 0.0577 - val_acc: 0.9841
Epoch 18/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0262 -
acc: 0.9908 - val_loss: 0.0694 - val_acc: 0.9828
Epoch 19/20
60000/60000 [==============================] - 5s 84us/step - loss: 0.0258 -
acc: 0.9914 - val_loss: 0.0618 - val_acc: 0.9830
```

```
Epoch 20/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.0243 -
acc: 0.9918 - val_loss: 0.0581 - val_acc: 0.9854
```

[33]:
```
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))

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

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

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

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

```
Test score: 0.058136306594208875
Test accuracy: 0.9854
```



Model 3. 3 hidden layer, ReLU, Dropout, BatchNormalization and AdamOptimizer

[34]:
```
# https://stackoverflow.com/questions/34716454/
↪where-do-i-call-the-batchnormalization-function-in-keras



model_2 = Sequential()

model_2.add(Dense(512, activation='relu', input_shape=(input_dim,),␣
↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_2.add(BatchNormalization())
model_2.add(Dropout(0.8))

model_2.add(Dense(128, activation='relu',␣
↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_2.add(BatchNormalization())
model_2.add(Dropout(0.8))

model_2.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.
↪0, stddev=0.176, seed=None)) )
model_2.add(BatchNormalization())
model_2.add(Dropout(0.8))

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

```
model_2.summary()
```

```
W0715 18:49:34.213879 11376 nn_ops.py:4224] Large dropout rate: 0.8 (>0.5). In
TensorFlow 2.x, dropout() uses dropout rate instead of keep_prob. Please ensure
that this is intended.
W0715 18:49:34.288651 11376 nn_ops.py:4224] Large dropout rate: 0.8 (>0.5). In
TensorFlow 2.x, dropout() uses dropout rate instead of keep_prob. Please ensure
that this is intended.
W0715 18:49:34.363479 11376 nn_ops.py:4224] Large dropout rate: 0.8 (>0.5). In
TensorFlow 2.x, dropout() uses dropout rate instead of keep_prob. Please ensure
that this is intended.

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_25 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_13 (Batc (None, 512)               2048
_________________________________________________________________
dropout_18 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_26 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_14 (Batc (None, 128)               512
_________________________________________________________________
dropout_19 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_27 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_15 (Batc (None, 64)                256
_________________________________________________________________
dropout_20 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_28 (Dense)             (None, 10)                650
=================================================================
Total params: 479,306
Trainable params: 477,898
Non-trainable params: 1,408
_________________________________________________________________
```

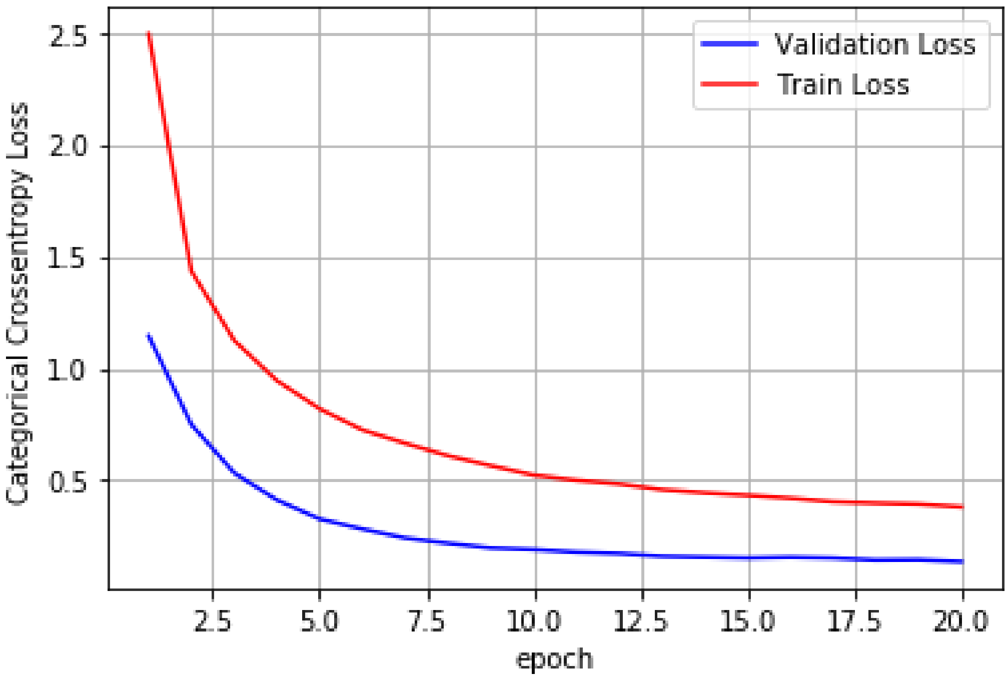
[35]:
```
model_2.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

history = model_2.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: 2.4937 -
acc: 0.2566 - val_loss: 1.1451 - val_acc: 0.7325
Epoch 2/20
60000/60000 [==============================] - 5s 85us/step - loss: 1.4336 -
acc: 0.4918 - val_loss: 0.7489 - val_acc: 0.8238
Epoch 3/20
60000/60000 [==============================] - 5s 85us/step - loss: 1.1254 -
acc: 0.6133 - val_loss: 0.5330 - val_acc: 0.8762
Epoch 4/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.9454 -
acc: 0.6819 - val_loss: 0.4127 - val_acc: 0.9009
Epoch 5/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.8193 -
acc: 0.7359 - val_loss: 0.3276 - val_acc: 0.9199
Epoch 6/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.7250 -
acc: 0.7723 - val_loss: 0.2828 - val_acc: 0.9273
Epoch 7/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.6651 -
acc: 0.7997 - val_loss: 0.2425 - val_acc: 0.9375
Epoch 8/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.6095 -
acc: 0.8204 - val_loss: 0.2189 - val_acc: 0.9425
Epoch 9/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.5651 -
acc: 0.8388 - val_loss: 0.1981 - val_acc: 0.9469
Epoch 10/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.5232 -
acc: 0.8530 - val_loss: 0.1911 - val_acc: 0.9496
Epoch 11/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.5006 -
acc: 0.8616 - val_loss: 0.1793 - val_acc: 0.9520
Epoch 12/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.4841 -
acc: 0.8678 - val_loss: 0.1732 - val_acc: 0.9540
Epoch 13/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.4584 -
acc: 0.8784 - val_loss: 0.1608 - val_acc: 0.9574
Epoch 14/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.4446 -
acc: 0.8804 - val_loss: 0.1570 - val_acc: 0.9583
Epoch 15/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.4335 -
acc: 0.8841 - val_loss: 0.1536 - val_acc: 0.9584
Epoch 16/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.4205 -
```

```
acc: 0.8886 - val_loss: 0.1567 - val_acc: 0.9588
Epoch 17/20
60000/60000 [==============================] - 5s 87us/step - loss: 0.4047 -
acc: 0.8940 - val_loss: 0.1533 - val_acc: 0.9594
Epoch 18/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.3985 -
acc: 0.8953 - val_loss: 0.1441 - val_acc: 0.9626
Epoch 19/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.3938 -
acc: 0.8989 - val_loss: 0.1454 - val_acc: 0.9637
Epoch 20/20
60000/60000 [==============================] - 5s 85us/step - loss: 0.3815 -
acc: 0.8997 - val_loss: 0.1369 - val_acc: 0.9648
```

[36]:
```
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))

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

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

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

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

```
Test score: 0.1369463347967714
Test accuracy: 0.9648
```



Model 3. 5 hidden layer, ReLU, Dropout, BatchNormalization and AdamOptimizer

[37]:
```
# https://stackoverflow.com/questions/34716454/
 ↪where-do-i-call-the-batchnormalization-function-in-keras

from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization

model_3 = Sequential()

model_3.add(Dense(512, activation='relu', input_shape=(input_dim,),␣
 ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(128, activation='relu',␣
 ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.
 ↪0, stddev=0.176, seed=None)) )
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))
```

```
model_3.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.
 ↪0, stddev=0.176, seed=None)) )
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

model_3.add(Dense(128, activation='relu',␣
 ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))

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


model_3.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_29 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_16 (Batc (None, 512)               2048
_________________________________________________________________
dropout_21 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_30 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_17 (Batc (None, 128)               512
_________________________________________________________________
dropout_22 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_31 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_18 (Batc (None, 64)                256
_________________________________________________________________
dropout_23 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_32 (Dense)             (None, 64)                4160
_________________________________________________________________
batch_normalization_19 (Batc (None, 64)                256
_________________________________________________________________
dropout_24 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_33 (Dense)             (None, 128)               8320
_________________________________________________________________
batch_normalization_20 (Batc (None, 128)               512
_________________________________________________________________
dropout_25 (Dropout)         (None, 128)               0
```

```
_________________________________________________________________
dense_34 (Dense)             (None, 10)                1290
=================================================================
Total params: 493,194
Trainable params: 491,402
Non-trainable params: 1,792
_________________________________________________________________
```

[38]:
```
model_3.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

history = model_3.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 128us/step - loss: 1.6475 -
acc: 0.4693 - val_loss: 0.4033 - val_acc: 0.8867
Epoch 2/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.6705 -
acc: 0.7905 - val_loss: 0.2357 - val_acc: 0.9349
Epoch 3/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.4587 -
acc: 0.8699 - val_loss: 0.1786 - val_acc: 0.9511
Epoch 4/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.3607 -
acc: 0.9007 - val_loss: 0.1519 - val_acc: 0.9588
Epoch 5/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.3025 -
acc: 0.9191 - val_loss: 0.1399 - val_acc: 0.9642
Epoch 6/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.2703 -
acc: 0.9301 - val_loss: 0.1225 - val_acc: 0.9695
Epoch 7/20
60000/60000 [==============================] - 6s 99us/step - loss: 0.2407 -
acc: 0.9386 - val_loss: 0.1176 - val_acc: 0.9689
Epoch 8/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.2219 -
acc: 0.9451 - val_loss: 0.1185 - val_acc: 0.9700
Epoch 9/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.2048 -
acc: 0.9494 - val_loss: 0.1026 - val_acc: 0.9740
Epoch 10/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.1918 -
acc: 0.9522 - val_loss: 0.1019 - val_acc: 0.9751
Epoch 11/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.1762 -
```

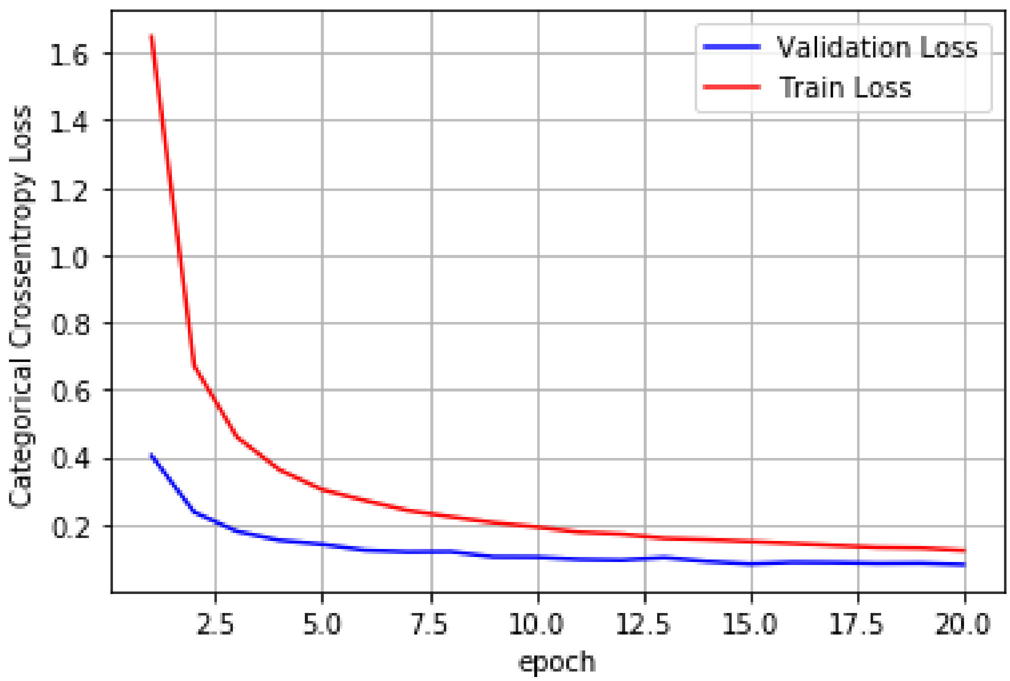
```
acc: 0.9569 - val_loss: 0.0951 - val_acc: 0.9767
Epoch 12/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.1703 -
acc: 0.9581 - val_loss: 0.0930 - val_acc: 0.9769
Epoch 13/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.1579 -
acc: 0.9616 - val_loss: 0.1007 - val_acc: 0.9762
Epoch 14/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.1539 -
acc: 0.9613 - val_loss: 0.0892 - val_acc: 0.9781
Epoch 15/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.1483 -
acc: 0.9635 - val_loss: 0.0816 - val_acc: 0.9803
Epoch 16/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.1425 -
acc: 0.9636 - val_loss: 0.0866 - val_acc: 0.9797
Epoch 17/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.1366 -
acc: 0.9661 - val_loss: 0.0857 - val_acc: 0.9795
Epoch 18/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.1309 -
acc: 0.9678 - val_loss: 0.0832 - val_acc: 0.9803
Epoch 19/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.1290 -
acc: 0.9681 - val_loss: 0.0840 - val_acc: 0.9812
Epoch 20/20
60000/60000 [==============================] - 6s 98us/step - loss: 0.1218 -
acc: 0.9696 - val_loss: 0.0797 - val_acc: 0.9823
```

[39]:

```python
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))

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

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

```
# val_acc : validation accuracy

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

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

```
Test score: 0.07966478181201965
Test accuracy: 0.9823
```



Model 3. 5 hidden layer, ReLU, Dropout and AdamOptimizer

[40]:
```
# https://stackoverflow.com/questions/34716454/
 ↪where-do-i-call-the-batchnormalization-function-in-keras

from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization

model_3 = Sequential()

model_3.add(Dense(512, activation='relu', input_shape=(input_dim,),
 ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
```

```
model_3.add(Dropout(0.5))

model_3.add(Dense(128, activation='relu',␣
 ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_3.add(Dropout(0.5))

model_3.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.
 ↪0, stddev=0.176, seed=None)) )
model_3.add(Dropout(0.5))

model_3.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.
 ↪0, stddev=0.176, seed=None)) )
model_3.add(Dropout(0.5))

model_3.add(Dense(128, activation='relu',␣
 ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_3.add(Dropout(0.5))

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


model_3.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_35 (Dense)             (None, 512)               401920
_________________________________________________________________
dropout_26 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_36 (Dense)             (None, 128)               65664
_________________________________________________________________
dropout_27 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_37 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout_28 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_38 (Dense)             (None, 64)                4160
_________________________________________________________________
dropout_29 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_39 (Dense)             (None, 128)               8320
_________________________________________________________________
dropout_30 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_40 (Dense)             (None, 10)                1290
```

```
=================================================================
Total params: 489,610
Trainable params: 489,610
Non-trainable params: 0
_________________________________________________________________
```

[41]:
```
model_3.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

history = model_3.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 103us/step - loss: 2.1719 -
acc: 0.2328 - val_loss: 1.4112 - val_acc: 0.4797
Epoch 2/20
60000/60000 [==============================] - 5s 82us/step - loss: 1.2927 -
acc: 0.5287 - val_loss: 0.8277 - val_acc: 0.6854
Epoch 3/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.9126 -
acc: 0.6637 - val_loss: 0.5805 - val_acc: 0.8139
Epoch 4/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.7038 -
acc: 0.7515 - val_loss: 0.4486 - val_acc: 0.8323
Epoch 5/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.5840 -
acc: 0.7958 - val_loss: 0.3832 - val_acc: 0.8513
Epoch 6/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.5204 -
acc: 0.8157 - val_loss: 0.3473 - val_acc: 0.8589
Epoch 7/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.4657 -
acc: 0.8338 - val_loss: 0.3145 - val_acc: 0.8690
Epoch 8/20
60000/60000 [==============================] - 5s 86us/step - loss: 0.4365 -
acc: 0.8521 - val_loss: 0.2935 - val_acc: 0.9065
Epoch 9/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.3839 -
acc: 0.8963 - val_loss: 0.2154 - val_acc: 0.9530
Epoch 10/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.3319 -
acc: 0.9199 - val_loss: 0.1913 - val_acc: 0.9574
Epoch 11/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.3073 -
acc: 0.9251 - val_loss: 0.1842 - val_acc: 0.9573
Epoch 12/20
```

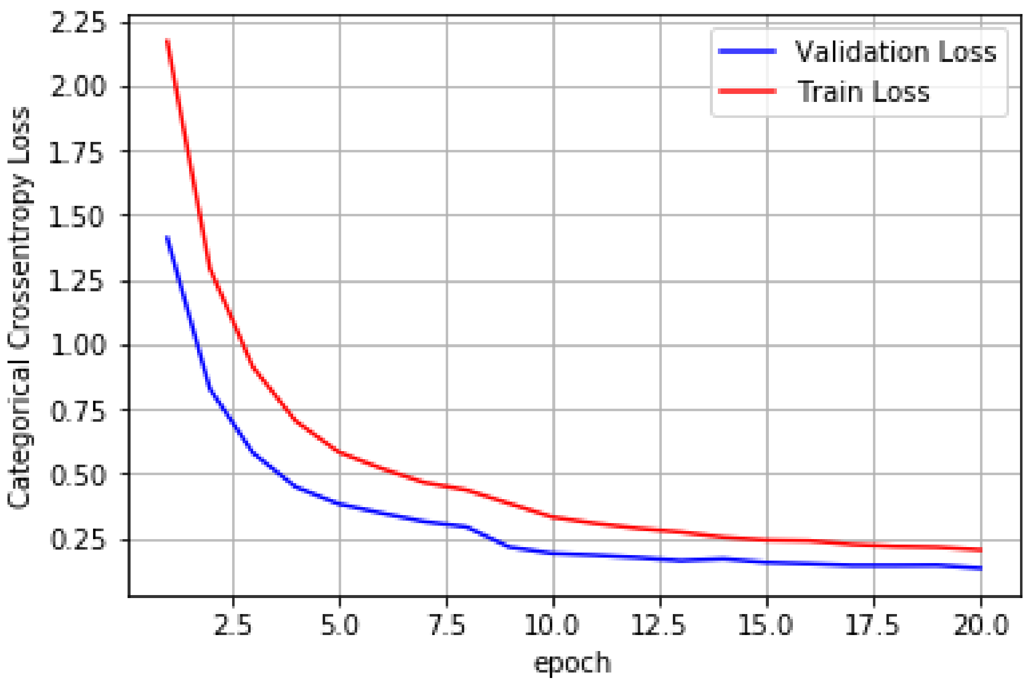
```
60000/60000 [==============================] - 5s 83us/step - loss: 0.2882 -
acc: 0.9305 - val_loss: 0.1751 - val_acc: 0.9607
Epoch 13/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.2737 -
acc: 0.9349 - val_loss: 0.1632 - val_acc: 0.9627
Epoch 14/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.2538 -
acc: 0.9404 - val_loss: 0.1705 - val_acc: 0.9638
Epoch 15/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.2426 -
acc: 0.9431 - val_loss: 0.1562 - val_acc: 0.9662
Epoch 16/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.2400 -
acc: 0.9442 - val_loss: 0.1511 - val_acc: 0.9655
Epoch 17/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.2259 -
acc: 0.9467 - val_loss: 0.1447 - val_acc: 0.9681
Epoch 18/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.2184 -
acc: 0.9498 - val_loss: 0.1446 - val_acc: 0.9691
Epoch 19/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.2153 -
acc: 0.9485 - val_loss: 0.1456 - val_acc: 0.9665
Epoch 20/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.2050 -
acc: 0.9515 - val_loss: 0.1340 - val_acc: 0.9692
```

[42]:

```
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))

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

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

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

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

```
Test score: 0.13398695126250387
Test accuracy: 0.9692
```



Model 3. 5 hidden layer, ReLU, Dropout, BatchNormalization and AdamOptimizer

[43]:
```
# https://stackoverflow.com/questions/34716454/
 ↪where-do-i-call-the-batchnormalization-function-in-keras

from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization

model_3 = Sequential()

model_3.add(Dense(512, activation='relu', input_shape=(input_dim,),␣
 ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_3.add(BatchNormalization())
```

```
model_3.add(Dropout(0.2))

model_3.add(Dense(128, activation='relu',␣
    ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_3.add(BatchNormalization())
model_3.add(Dropout(0.2))

model_3.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.
    ↪0, stddev=0.176, seed=None)) )
model_3.add(BatchNormalization())
model_3.add(Dropout(0.2))

model_3.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.
    ↪0, stddev=0.176, seed=None)) )
model_3.add(BatchNormalization())
model_3.add(Dropout(0.2))

model_3.add(Dense(128, activation='relu',␣
    ↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_3.add(BatchNormalization())
model_3.add(Dropout(0.2))

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


model_3.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_41 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_21 (Batc (None, 512)               2048
_________________________________________________________________
dropout_31 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_42 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_22 (Batc (None, 128)               512
_________________________________________________________________
dropout_32 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_43 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_23 (Batc (None, 64)                256
_________________________________________________________________
dropout_33 (Dropout)         (None, 64)                0
```

```
_________________________________________________________________
dense_44 (Dense)             (None, 64)                4160
_________________________________________________________________
batch_normalization_24 (Batc (None, 64)                256
_________________________________________________________________
dropout_34 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_45 (Dense)             (None, 128)               8320
_________________________________________________________________
batch_normalization_25 (Batc (None, 128)               512
_________________________________________________________________
dropout_35 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_46 (Dense)             (None, 10)                1290
=================================================================
Total params: 493,194
Trainable params: 491,402
Non-trainable params: 1,792
_________________________________________________________________
```

[44]:
```
model_3.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

history = model_3.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 139us/step - loss: 0.5371 -
acc: 0.8352 - val_loss: 0.1464 - val_acc: 0.9552
Epoch 2/20
60000/60000 [==============================] - 6s 104us/step - loss: 0.2128 -
acc: 0.9385 - val_loss: 0.1091 - val_acc: 0.9672
Epoch 3/20
60000/60000 [==============================] - 6s 104us/step - loss: 0.1594 -
acc: 0.9534 - val_loss: 0.0922 - val_acc: 0.9725
Epoch 4/20
60000/60000 [==============================] - 6s 104us/step - loss: 0.1300 -
acc: 0.9622 - val_loss: 0.0867 - val_acc: 0.9734
Epoch 5/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.1093 -
acc: 0.9679 - val_loss: 0.0753 - val_acc: 0.9784
Epoch 6/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.0980 -
acc: 0.9712 - val_loss: 0.0803 - val_acc: 0.9770
Epoch 7/20
60000/60000 [==============================] - 6s 104us/step - loss: 0.0893 -
```

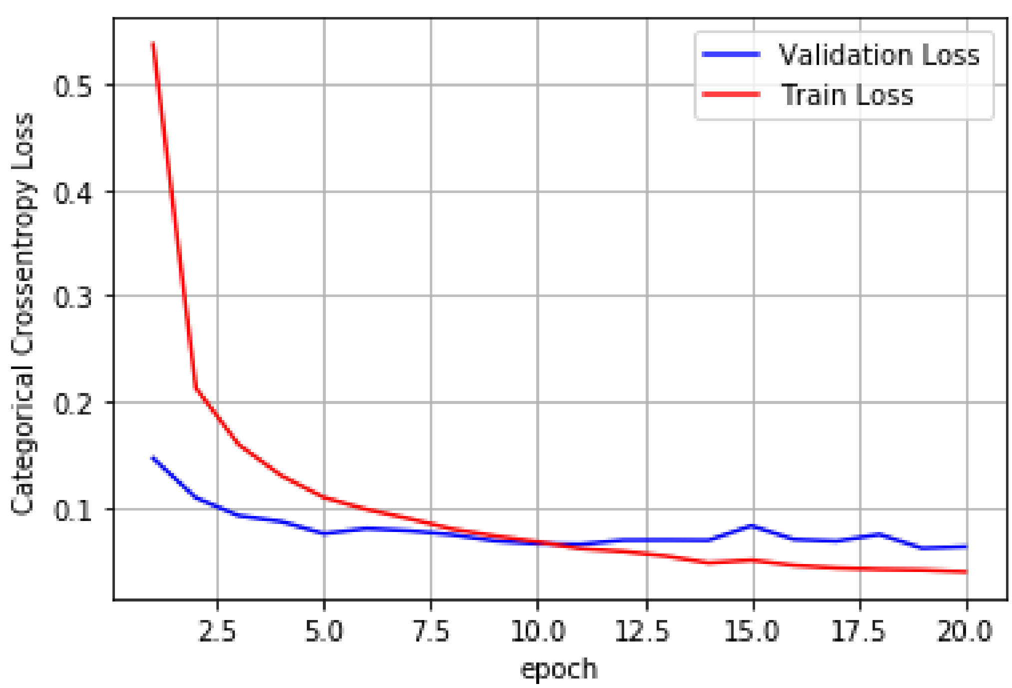
```
acc: 0.9731 - val_loss: 0.0781 - val_acc: 0.9778
Epoch 8/20
60000/60000 [==============================] - 7s 111us/step - loss: 0.0797 -
acc: 0.9765 - val_loss: 0.0743 - val_acc: 0.9797
Epoch 9/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.0732 -
acc: 0.9779 - val_loss: 0.0685 - val_acc: 0.9815
Epoch 10/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.0677 -
acc: 0.9803 - val_loss: 0.0658 - val_acc: 0.9817
Epoch 11/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0612 -
acc: 0.9820 - val_loss: 0.0652 - val_acc: 0.9815
Epoch 12/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0586 -
acc: 0.9824 - val_loss: 0.0692 - val_acc: 0.9810
Epoch 13/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.0542 -
acc: 0.9837 - val_loss: 0.0694 - val_acc: 0.9811
Epoch 14/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.0477 -
acc: 0.9855 - val_loss: 0.0691 - val_acc: 0.9822
Epoch 15/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0503 -
acc: 0.9850 - val_loss: 0.0828 - val_acc: 0.9785
Epoch 16/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.0454 -
acc: 0.9861 - val_loss: 0.0699 - val_acc: 0.9813
Epoch 17/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.0429 -
acc: 0.9868 - val_loss: 0.0683 - val_acc: 0.9830
Epoch 18/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.0417 -
acc: 0.9870 - val_loss: 0.0746 - val_acc: 0.9804
Epoch 19/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.0410 -
acc: 0.9874 - val_loss: 0.0616 - val_acc: 0.9832
Epoch 20/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.0393 -
acc: 0.9880 - val_loss: 0.0631 - val_acc: 0.9829
```

[45]:

```python
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))

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

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

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

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

```
Test score: 0.06305098229350987
Test accuracy: 0.9829
```



Model 3. 5 hidden layer, ReLU, Dropout, BatchNormalization and AdamOptimizer

[46]:
```
# https://stackoverflow.com/questions/34716454/
↪where-do-i-call-the-batchnormalization-function-in-keras

from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization

model_3 = Sequential()

model_3.add(Dense(512, activation='relu', input_shape=(input_dim,),␣
↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None)))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.8))

model_3.add(Dense(128, activation='relu',␣
↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_3.add(BatchNormalization())
model_3.add(Dropout(0.8))

model_3.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.
↪0, stddev=0.176, seed=None)) )
model_3.add(BatchNormalization())
model_3.add(Dropout(0.8))

model_3.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean=0.
↪0, stddev=0.176, seed=None)) )
model_3.add(BatchNormalization())
model_3.add(Dropout(0.8))

model_3.add(Dense(128, activation='relu',␣
↪kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None)) )
model_3.add(BatchNormalization())
model_3.add(Dropout(0.8))

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


model_3.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_47 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_26 (Batc (None, 512)               2048
_________________________________________________________________
dropout_36 (Dropout)         (None, 512)               0
```

```
_________________________________________________________________
dense_48 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_27 (Batc (None, 128)               512
_________________________________________________________________
dropout_37 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_49 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_28 (Batc (None, 64)                256
_________________________________________________________________
dropout_38 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_50 (Dense)             (None, 64)                4160
_________________________________________________________________
batch_normalization_29 (Batc (None, 64)                256
_________________________________________________________________
dropout_39 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_51 (Dense)             (None, 128)               8320
_________________________________________________________________
batch_normalization_30 (Batc (None, 128)               512
_________________________________________________________________
dropout_40 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_52 (Dense)             (None, 10)                1290
=================================================================
Total params: 493,194
Trainable params: 491,402
Non-trainable params: 1,792
_________________________________________________________________
```

[47]:

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

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

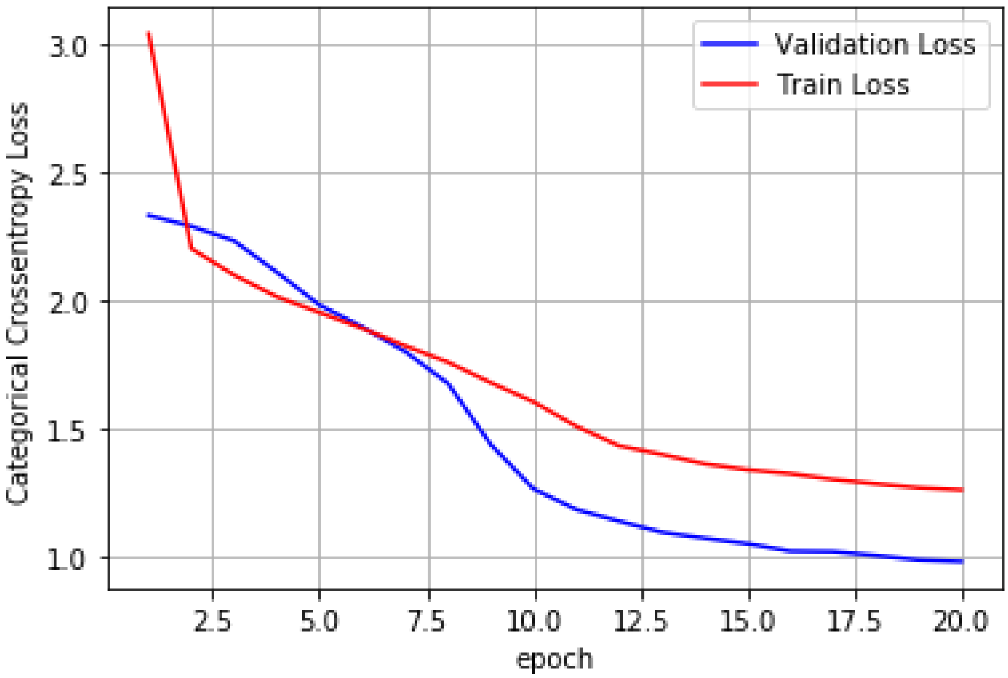
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 144us/step - loss: 3.0380 -
acc: 0.1137 - val_loss: 2.3314 - val_acc: 0.1145
Epoch 2/20
60000/60000 [==============================] - 6s 106us/step - loss: 2.2017 -
acc: 0.1747 - val_loss: 2.2887 - val_acc: 0.1366
Epoch 3/20
60000/60000 [==============================] - 6s 103us/step - loss: 2.0982 -
```

```
acc: 0.2019 - val_loss: 2.2321 - val_acc: 0.1283
Epoch 4/20
60000/60000 [==============================] - 6s 103us/step - loss: 2.0141 -
acc: 0.2138 - val_loss: 2.1096 - val_acc: 0.1461
Epoch 5/20
60000/60000 [==============================] - 7s 112us/step - loss: 1.9522 -
acc: 0.2232 - val_loss: 1.9815 - val_acc: 0.1942
Epoch 6/20
60000/60000 [==============================] - 6s 104us/step - loss: 1.8904 -
acc: 0.2424 - val_loss: 1.8959 - val_acc: 0.2227
Epoch 7/20
60000/60000 [==============================] - 6s 106us/step - loss: 1.8227 -
acc: 0.2633 - val_loss: 1.8004 - val_acc: 0.2619
Epoch 8/20
60000/60000 [==============================] - 6s 105us/step - loss: 1.7593 -
acc: 0.2829 - val_loss: 1.6761 - val_acc: 0.3008
Epoch 9/20
60000/60000 [==============================] - 6s 105us/step - loss: 1.6793 -
acc: 0.3099 - val_loss: 1.4377 - val_acc: 0.4360
Epoch 10/20
60000/60000 [==============================] - 6s 104us/step - loss: 1.6044 -
acc: 0.3410 - val_loss: 1.2655 - val_acc: 0.4952
Epoch 11/20
60000/60000 [==============================] - 6s 103us/step - loss: 1.5093 -
acc: 0.3793 - val_loss: 1.1853 - val_acc: 0.4824
Epoch 12/20
60000/60000 [==============================] - 6s 104us/step - loss: 1.4331 -
acc: 0.4036 - val_loss: 1.1400 - val_acc: 0.5061
Epoch 13/20
60000/60000 [==============================] - 7s 111us/step - loss: 1.3994 -
acc: 0.4158 - val_loss: 1.0967 - val_acc: 0.5307
Epoch 14/20
60000/60000 [==============================] - 7s 109us/step - loss: 1.3628 -
acc: 0.4262 - val_loss: 1.0733 - val_acc: 0.5401
Epoch 15/20
60000/60000 [==============================] - 7s 110us/step - loss: 1.3399 -
acc: 0.4352 - val_loss: 1.0520 - val_acc: 0.5453
Epoch 16/20
60000/60000 [==============================] - 6s 106us/step - loss: 1.3246 -
acc: 0.4398 - val_loss: 1.0232 - val_acc: 0.6262
Epoch 17/20
60000/60000 [==============================] - 7s 108us/step - loss: 1.3028 -
acc: 0.4462 - val_loss: 1.0212 - val_acc: 0.5690
Epoch 18/20
60000/60000 [==============================] - 6s 108us/step - loss: 1.2857 -
acc: 0.4552 - val_loss: 1.0059 - val_acc: 0.5607
Epoch 19/20
60000/60000 [==============================] - 7s 110us/step - loss: 1.2704 -
```

```
acc: 0.4587 - val_loss: 0.9887 - val_acc: 0.5915
Epoch 20/20
60000/60000 [==============================] - 7s 110us/step - loss: 1.2616 -
acc: 0.4644 - val_loss: 0.9821 - val_acc: 0.5892
```

[48]:
```
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))

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

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

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

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

```
Test score: 0.9821138044357299
Test accuracy: 0.5892
```



[49]:
```
from prettytable import PrettyTable
y = PrettyTable()
y.field_names = ["Model 1", "Model 2","Model 3"]
y.add_row(["128 - 64", "512 - 128 - 64", "512 - 128 - 64 - 64 - 128"])
y.add_row(["2 layer NN", "3 layer NN", "5 layer NN"])
print(y)



x = PrettyTable()
x.field_names = ["Model", "Dropout","Batch Normalization", "optimizer", "Train␣
 ↪Accuracy", "Test Accuracy"]
x.add_row(["Model 1.1", "0.5", "Yes", "adam",0.959,0.976])
x.add_row(["Model 1.2", "0.5" , "No", "adam",0.956,0.979])
x.add_row(["Model 1.3", "0.2", "Yes", "adam",0.985,0.979])
x.add_row(["Model 1.4", "0.8", "Yes", "adam",0.859,0.945])

x.add_row(["-------", "---", "---", "----","----","----"])

x.add_row(["Model 2.1", "0.5" , "Yes", "adam",0.978,0.983])
x.add_row(["Model 2.2", "0.5", "No", "adam",0.975,0.978])
x.add_row(["Model 2.3", "0.2" , "Yes", "adam",0.992,0.981])
x.add_row(["Model 2.4", "0.8", "Yes", "adam",0.902,0.965])

x.add_row(["-------", "---", "---", "----","----","----"])
```

```
x.add_row(["Model 3.1", "0.5", "Yes", "adam",0.971,0.979])
x.add_row(["Model 3.2", "0.5" , "No", "adam",0.951,0.967])
x.add_row(["Model 3.3", "0.2", "Yes", "adam",0.989,0.984])
x.add_row(["Model 3.4", "0.8", "Yes", "adam",0.406,0.502])

x.border=True
print(x)
```

```
+-------------+----------------+---------------------------+
|  Model 1    |    Model 2     |          Model 3          |
+-------------+----------------+---------------------------+
|  128 - 64   | 512 - 128 - 64 | 512 - 128 - 64 - 64 - 128 |
| 2 layer NN  |   3 layer NN   |         5 layer NN        |
+-------------+----------------+---------------------------+
+------------+---------+---------------------+-----------+----------------+------
---------+
|   Model    | Dropout | Batch Normalization | optimizer | Train Accuracy | Test
Accuracy |
+------------+---------+---------------------+-----------+----------------+------
---------+
| Model 1.1 |   0.5   |         Yes         |    adam   |     0.959      |
0.976      |
| Model 1.2 |   0.5   |          No         |    adam   |     0.956      |
0.979      |
| Model 1.3 |   0.2   |         Yes         |    adam   |     0.985      |
0.979      |
| Model 1.4 |   0.8   |         Yes         |    adam   |     0.859      |
0.945      |
|  -------  |   ---   |         ---         |    ----   |      ----      |
----      |
| Model 2.1 |   0.5   |         Yes         |    adam   |     0.978      |
0.983      |
| Model 2.2 |   0.5   |          No         |    adam   |     0.975      |
0.978      |
| Model 2.3 |   0.2   |         Yes         |    adam   |     0.992      |
0.981      |
| Model 2.4 |   0.8   |         Yes         |    adam   |     0.902      |
0.965      |
|  -------  |   ---   |         ---         |    ----   |      ----      |
----      |
| Model 3.1 |   0.5   |         Yes         |    adam   |     0.971      |
0.979      |
| Model 3.2 |   0.5   |          No         |    adam   |     0.951      |
0.967      |
| Model 3.3 |   0.2   |         Yes         |    adam   |     0.989      |
0.984      |
| Model 3.4 |   0.8   |         Yes         |    adam   |     0.406      |
```

```
0.502     |
+----------+--------+------------------+----------+--------------+------
---------+
```

Best model is Model 3.3 as it has both train and test score as 0.989 and 0.984 respectively
Drop out of value 0.8 hurts the model. 0.5 dropout is reasonable.
From result 1.2, 2.2, 3.2 its pretty clear that batch Normalization is needed in later stages of deep neural network, especially having more than 3 hidden layers. It is always good to do batch Normalization.

[ ]:

```

```