Assignment:Different MLP architectures on MNIST dataset

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

```
Using TensorFlow backend.
```

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

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

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 1s 0us/step
```

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

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

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

In [0]:

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

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

In [6]:

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

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

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

In [7]:

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

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

```
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 25
5
 247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  36  94 15
4
 170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0   0
0
   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  8
2
  82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 25
3
 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 15
4
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  35 24
1
 225 160 108   1   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0   0
```

```
0
   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 18
7
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 25
3
 253 207   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 25
3
 253 201  78   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0  18 171 219 253 253 253 253 19
5
  80   9   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0 136 253 253 253 212 135 132   1
6
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0
   0   0   0   0   0   0   0   0   0   0]
```

In [0]:

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

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

In [9]:

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

```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.01176471 0.07058824 0.07058824 0.07058824
 0.49411765 0.53333333 0.68627451 0.10196078 0.65098039 1.
```

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

```
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.31764706 0.94117647 0.99215686
0.99215686 0.46666667 0.09803922 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.17647059 0.72941176 0.99215686 0.99215686
0.58823529 0.10588235 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.0627451  0.36470588 0.98823529 0.99215686 0.73333333
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.97647059 0.99215686 0.97647059 0.25098039 0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.18039216 0.50980392 0.71764706 0.99215686
0.99215686 0.81176471 0.00784314 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.15294118 0.58039216
0.89803922 0.99215686 0.99215686 0.99215686 0.98039216 0.71372549
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.09411765 0.44705882 0.86666667 0.99215686 0.99215686 0.99215686
0.99215686 0.78823529 0.30588235 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.09019608 0.25882353 0.83529412 0.99215686
0.99215686 0.99215686 0.99215686 0.77647059 0.31764706 0.00784314
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.07058824 0.67058824
0.85882353 0.99215686 0.99215686 0.99215686 0.99215686 0.76470588
```

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

In [10]:

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

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

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

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

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

In [0]:

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

In [0]:

```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

MLP + Batch-Norm on 2-hidden Layers + Dropout + AdamOptimizer

In [13]:

```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batc
hnormalization-function-in-keras

from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
model_drop = Sequential()

model_drop.add(Dense(450, activation='relu', input_shape=(input_dim,),
kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(108, activation='relu', kernel_initializer=RandomN
ormal(mean=0.0, stddev=0.55, seed=None)) )
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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

```
model_drop.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorfl
ow/python/framework/op_def_library.py:263: colocate_with (from tensorfl
ow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/ba
ckend/tensorflow_backend.py:3445: calling dropout (from tensorflow.pyth
on.ops.nn_ops) with keep_prob is deprecated and will be removed in a fu
ture version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate =
1 - keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 450)               353250
_________________________________________________________________
batch_normalization_1 (Batch (None, 450)               1800
_________________________________________________________________
dropout_1 (Dropout)          (None, 450)               0
_________________________________________________________________
dense_2 (Dense)              (None, 108)               48708
_________________________________________________________________
batch_normalization_2 (Batch (None, 108)               432
_________________________________________________________________
dropout_2 (Dropout)          (None, 108)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1090
=================================================================
Total params: 405,280
Trainable params: 404,164
Non-trainable params: 1,116
_________________________________________________________________
```

In [14]:
```
model_drop.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['accuracy'])

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

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorfl
ow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.ma
th_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 140us/step - loss: 0.
5117 - acc: 0.8455 - val_loss: 0.1711 - val_acc: 0.9443
Epoch 2/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.
2657 - acc: 0.9211 - val_loss: 0.1310 - val_acc: 0.9590
Epoch 3/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.
2107 - acc: 0.9360 - val_loss: 0.1172 - val_acc: 0.9647
Epoch 4/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.
1868 - acc: 0.9443 - val_loss: 0.1019 - val_acc: 0.9684
Epoch 5/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.
1647 - acc: 0.9505 - val_loss: 0.0913 - val_acc: 0.9723
Epoch 6/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.
1509 - acc: 0.9540 - val_loss: 0.0869 - val_acc: 0.9738
Epoch 7/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.
1395 - acc: 0.9569 - val_loss: 0.0831 - val_acc: 0.9748
Epoch 8/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.
1311 - acc: 0.9598 - val_loss: 0.0859 - val_acc: 0.9734
Epoch 9/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.
1236 - acc: 0.9624 - val_loss: 0.0778 - val_acc: 0.9759
Epoch 10/20
```

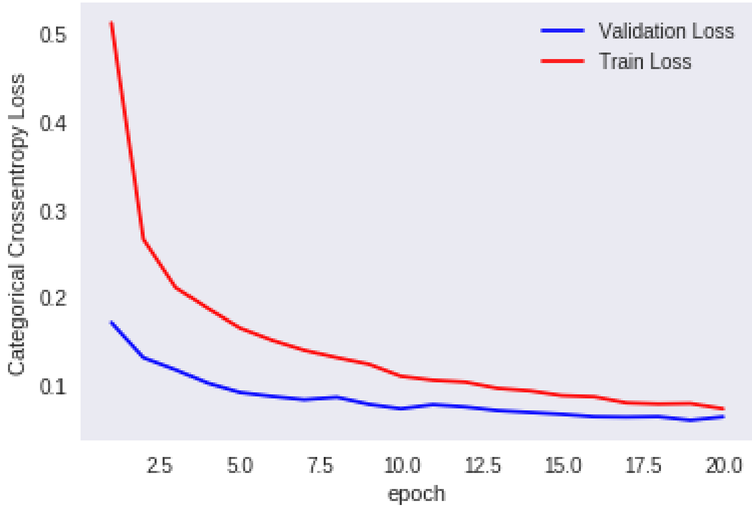
```
60000/60000 [==============================] - 7s 121us/step - loss: 0.
1099 - acc: 0.9664 - val_loss: 0.0727 - val_acc: 0.9773
Epoch 11/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.
1053 - acc: 0.9680 - val_loss: 0.0775 - val_acc: 0.9764
Epoch 12/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.
1032 - acc: 0.9679 - val_loss: 0.0747 - val_acc: 0.9774
Epoch 13/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.
0960 - acc: 0.9700 - val_loss: 0.0706 - val_acc: 0.9787
Epoch 14/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.
0932 - acc: 0.9709 - val_loss: 0.0685 - val_acc: 0.9789
Epoch 15/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.
0878 - acc: 0.9723 - val_loss: 0.0665 - val_acc: 0.9797
Epoch 16/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.
0864 - acc: 0.9729 - val_loss: 0.0636 - val_acc: 0.9810
Epoch 17/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.
0794 - acc: 0.9747 - val_loss: 0.0632 - val_acc: 0.9804
Epoch 18/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.
0782 - acc: 0.9758 - val_loss: 0.0637 - val_acc: 0.9807
Epoch 19/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.
0787 - acc: 0.9752 - val_loss: 0.0596 - val_acc: 0.9808
Epoch 20/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.
0727 - acc: 0.9771 - val_loss: 0.0635 - val_acc: 0.9812
```

In [15]:

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

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

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

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

# we will get val_loss and val_acc only when you pass the paramter vali
dation_data
# val_loss : validation loss
# val_acc : validation accuracy

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

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

```
Test score: 0.06352490943533484
Test accuracy: 0.9812
```



MLP of 2-hidden Layers + AdamOptimizer

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

model_nodrop.add(Dense(450, activation='relu', input_shape=(input_dim
,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None
)))


model_nodrop.add(Dense(108, activation='relu', kernel_initializer=Rando
mNormal(mean=0.0, stddev=0.55, seed=None)) )

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


model_nodrop.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
```

```
dense_4 (Dense)              (None, 450)               353250
_________________________________________________________________
dense_5 (Dense)              (None, 108)               48708
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1090
=================================================================
Total params: 403,048
Trainable params: 403,048
Non-trainable params: 0
_________________________________________________________________
```

In [17]:

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

history = model_nodrop.fit(X_train, Y_train, batch_size=batch_size, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

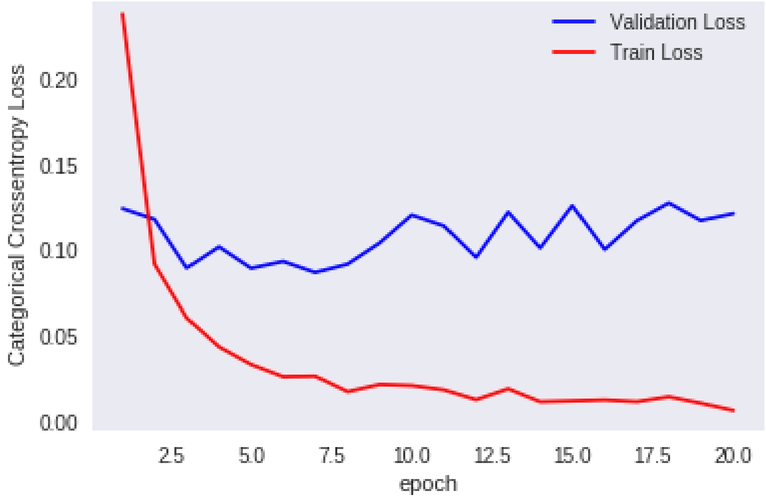
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 100us/step - loss: 0.
2367 - acc: 0.9307 - val_loss: 0.1238 - val_acc: 0.9629
Epoch 2/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
915 - acc: 0.9723 - val_loss: 0.1176 - val_acc: 0.9628
Epoch 3/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
603 - acc: 0.9808 - val_loss: 0.0892 - val_acc: 0.9753
Epoch 4/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
435 - acc: 0.9861 - val_loss: 0.1016 - val_acc: 0.9712
Epoch 5/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0
332 - acc: 0.9892 - val_loss: 0.0891 - val_acc: 0.9751
Epoch 6/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0
261 - acc: 0.9911 - val_loss: 0.0930 - val_acc: 0.9759
Epoch 7/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0

264 - acc: 0.9913 - val_loss: 0.0866 - val_acc: 0.9768
```

```
Epoch 8/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0
174 - acc: 0.9939 - val_loss: 0.0915 - val_acc: 0.9777
Epoch 9/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0
215 - acc: 0.9931 - val_loss: 0.1038 - val_acc: 0.9758
Epoch 10/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
209 - acc: 0.9925 - val_loss: 0.1200 - val_acc: 0.9731
Epoch 11/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0
184 - acc: 0.9939 - val_loss: 0.1138 - val_acc: 0.9752
Epoch 12/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0
127 - acc: 0.9957 - val_loss: 0.0954 - val_acc: 0.9799
Epoch 13/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0
190 - acc: 0.9938 - val_loss: 0.1217 - val_acc: 0.9744
Epoch 14/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0
115 - acc: 0.9960 - val_loss: 0.1008 - val_acc: 0.9788
Epoch 15/20
60000/60000 [==============================] - 6s 100us/step - loss: 0.
0120 - acc: 0.9957 - val_loss: 0.1255 - val_acc: 0.9753
Epoch 16/20
60000/60000 [==============================] - 6s 100us/step - loss: 0.
0125 - acc: 0.9958 - val_loss: 0.1000 - val_acc: 0.9793
Epoch 17/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0
115 - acc: 0.9963 - val_loss: 0.1167 - val_acc: 0.9759
Epoch 18/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
144 - acc: 0.9951 - val_loss: 0.1271 - val_acc: 0.9778
Epoch 19/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
106 - acc: 0.9965 - val_loss: 0.1169 - val_acc: 0.9787
Epoch 20/20


60000/60000 [==============================] - 5s 90us/step - loss: 0.0
```

```
064 - acc: 0.9977 - val_loss: 0.1209 - val_acc: 0.9783
```

In [18]:
```python
score = model_nodrop.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, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter vali
dation_data
# val_loss : validation loss
# val_acc : validation accuracy

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

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

```
Test score: 0.12091251760286323
Test accuracy: 0.9783
```



MLP + Batch-Norm on 3-hidden Layers + Dropout + AdamOptimizer

In [19]:

```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batc
hnormalization-function-in-keras

from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
model_drop_3 = Sequential()

model_drop_3.add(Dense(420, activation='relu', input_shape=(input_dim
,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None
)))
model_drop_3.add(BatchNormalization())
model_drop_3.add(Dropout(0.5))

model_drop_3.add(Dense(150, activation='relu', kernel_initializer=Rando
mNormal(mean=0.0, stddev=0.55, seed=None)) )
model_drop_3.add(BatchNormalization())
model_drop_3.add(Dropout(0.6))
```

```
model_drop_3.add(Dense(45, activation='relu', kernel_initializer=Random
Normal(mean=0.0, stddev=0.55, seed=None)) )
model_drop_3.add(BatchNormalization())
model_drop_3.add(Dropout(0.7))

model_drop_3.add(Dense(output_dim, activation='softmax'))


model_drop_3.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 420)               329700
_________________________________________________________________
batch_normalization_3 (Batch (None, 420)               1680
_________________________________________________________________
dropout_3 (Dropout)          (None, 420)               0
_________________________________________________________________
dense_8 (Dense)              (None, 150)               63150
_________________________________________________________________
batch_normalization_4 (Batch (None, 150)               600
_________________________________________________________________
dropout_4 (Dropout)          (None, 150)               0
_________________________________________________________________
dense_9 (Dense)              (None, 45)                6795
_________________________________________________________________
batch_normalization_5 (Batch (None, 45)                180
_________________________________________________________________
dropout_5 (Dropout)          (None, 45)                0
_________________________________________________________________
dense_10 (Dense)             (None, 10)                460
=================================================================
Total params: 402,565
Trainable params: 401,335
Non-trainable params: 1,230
_________________________________________________________________
```

In [20]:
```
model_drop_3.compile(optimizer='adam', loss='categorical_crossentropy',
```

```
 metrics=['accuracy'])

history = model_drop_3.fit(X_train, Y_train, batch_size=batch_size, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 142us/step - loss: 1.
4435 - acc: 0.5397 - val_loss: 0.3629 - val_acc: 0.9110
Epoch 2/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.
7449 - acc: 0.7655 - val_loss: 0.2329 - val_acc: 0.9336
Epoch 3/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.
5633 - acc: 0.8330 - val_loss: 0.1909 - val_acc: 0.9441
Epoch 4/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.
4642 - acc: 0.8677 - val_loss: 0.1632 - val_acc: 0.9530
Epoch 5/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.
3970 - acc: 0.8895 - val_loss: 0.1486 - val_acc: 0.9569
Epoch 6/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.
3590 - acc: 0.9024 - val_loss: 0.1374 - val_acc: 0.9606
Epoch 7/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.
3229 - acc: 0.9143 - val_loss: 0.1278 - val_acc: 0.9652
Epoch 8/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.
2996 - acc: 0.9209 - val_loss: 0.1230 - val_acc: 0.9656
Epoch 9/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.
2777 - acc: 0.9262 - val_loss: 0.1210 - val_acc: 0.9687
Epoch 10/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.
2655 - acc: 0.9330 - val_loss: 0.1125 - val_acc: 0.9686
Epoch 11/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.
2462 - acc: 0.9367 - val_loss: 0.1103 - val_acc: 0.9712
Epoch 12/20
```

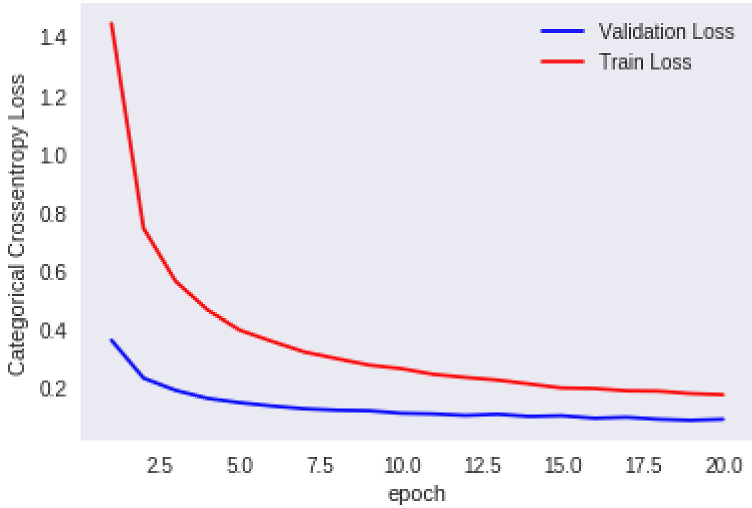
```
60000/60000 [==============================] - 7s 120us/step - loss: 0.
2353 - acc: 0.9394 - val_loss: 0.1048 - val_acc: 0.9741
Epoch 13/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.
2263 - acc: 0.9422 - val_loss: 0.1091 - val_acc: 0.9714
Epoch 14/20
60000/60000 [==============================] - 8s 130us/step - loss: 0.
2128 - acc: 0.9457 - val_loss: 0.1016 - val_acc: 0.9740
Epoch 15/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.
1992 - acc: 0.9483 - val_loss: 0.1040 - val_acc: 0.9736
Epoch 16/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.
1973 - acc: 0.9488 - val_loss: 0.0953 - val_acc: 0.9761
Epoch 17/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.
1894 - acc: 0.9532 - val_loss: 0.0987 - val_acc: 0.9747
Epoch 18/20
60000/60000 [==============================] - 7s 125us/step - loss: 0.
1883 - acc: 0.9531 - val_loss: 0.0918 - val_acc: 0.9770
Epoch 19/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.
1800 - acc: 0.9548 - val_loss: 0.0882 - val_acc: 0.9771
Epoch 20/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.
1761 - acc: 0.9555 - val_loss: 0.0919 - val_acc: 0.9764
```

In [21]:

```python
score = model_drop_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, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter vali
dation_data
# val_loss : validation loss
# val_acc : validation accuracy

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

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

```
Test score: 0.09186852280043531
Test accuracy: 0.9764
```



MLP of 3-hidden Layers + AdamOptimizer

In [22]:
```
from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
model_nodrop_3 = Sequential()

model_nodrop_3.add(Dense(420, activation='relu', input_shape=(input_dim
,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None
)))


model_nodrop_3.add(Dense(150, activation='relu', kernel_initializer=Ran
domNormal(mean=0.0, stddev=0.55, seed=None)) )


model_nodrop_3.add(Dense(45, activation='relu', kernel_initializer=Rand
omNormal(mean=0.0, stddev=0.55, seed=None)) )

model_nodrop_3.add(Dense(output_dim, activation='softmax'))


model_nodrop_3.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_11 (Dense)             (None, 420)               329700
_________________________________________________________________
dense_12 (Dense)             (None, 150)               63150
_________________________________________________________________
dense_13 (Dense)             (None, 45)                6795
_________________________________________________________________
dense_14 (Dense)             (None, 10)                460
=================================================================
Total params: 400,105
Trainable params: 400,105
Non-trainable params: 0
_________________________________________________________________
```

In [23]:
```
model_nodrop_3.compile(optimizer='adam', loss='categorical_crossentrop
y', metrics=['accuracy'])
```

```python
history = model_nodrop_3.fit(X_train, Y_train, batch_size=batch_size, e
pochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 102us/step - loss: 0.
4762 - acc: 0.8962 - val_loss: 0.2028 - val_acc: 0.9430
Epoch 2/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.1
408 - acc: 0.9594 - val_loss: 0.1511 - val_acc: 0.9585
Epoch 3/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0
950 - acc: 0.9712 - val_loss: 0.1398 - val_acc: 0.9612
Epoch 4/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0
762 - acc: 0.9767 - val_loss: 0.1478 - val_acc: 0.9639
Epoch 5/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0
601 - acc: 0.9814 - val_loss: 0.1271 - val_acc: 0.9689
Epoch 6/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0
485 - acc: 0.9845 - val_loss: 0.1482 - val_acc: 0.9659
Epoch 7/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0
463 - acc: 0.9850 - val_loss: 0.1419 - val_acc: 0.9669
Epoch 8/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0
352 - acc: 0.9886 - val_loss: 0.1405 - val_acc: 0.9680
Epoch 9/20
60000/60000 [==============================] - 6s 100us/step - loss: 0.
0349 - acc: 0.9887 - val_loss: 0.1313 - val_acc: 0.9712
Epoch 10/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.0
362 - acc: 0.9887 - val_loss: 0.1227 - val_acc: 0.9730
Epoch 11/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
275 - acc: 0.9911 - val_loss: 0.1287 - val_acc: 0.9723
Epoch 12/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0
```

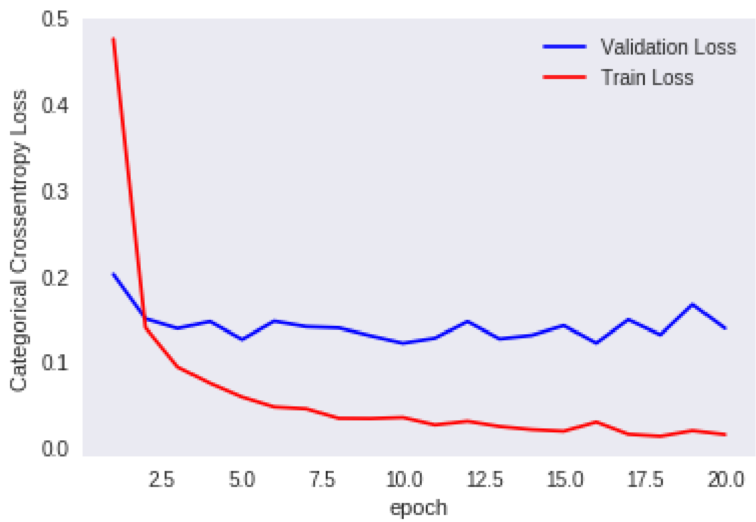
```
318 - acc: 0.9900 - val_loss: 0.1477 - val_acc: 0.9684
Epoch 13/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.0
257 - acc: 0.9919 - val_loss: 0.1277 - val_acc: 0.9736
Epoch 14/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0
221 - acc: 0.9929 - val_loss: 0.1319 - val_acc: 0.9721
Epoch 15/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
202 - acc: 0.9934 - val_loss: 0.1432 - val_acc: 0.9743
Epoch 16/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
309 - acc: 0.9912 - val_loss: 0.1229 - val_acc: 0.9756
Epoch 17/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
167 - acc: 0.9949 - val_loss: 0.1499 - val_acc: 0.9699
Epoch 18/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0
143 - acc: 0.9954 - val_loss: 0.1324 - val_acc: 0.9733
Epoch 19/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
208 - acc: 0.9934 - val_loss: 0.1676 - val_acc: 0.9721
Epoch 20/20
60000/60000 [==============================] - 5s 92us/step - loss: 0.0
163 - acc: 0.9950 - val_loss: 0.1399 - val_acc: 0.9747
```

In [24]:

```
score = model_nodrop_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, epo
```

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

# we will get val_loss and val_acc only when you pass the paramter vali
dation_data
# val_loss : validation loss
# val_acc : validation accuracy

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

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

```
Test score: 0.13994209184171222
Test accuracy: 0.9747
```



MLP + Batch-Norm on 5-hidden Layers + Dropout + AdamOptimizer

In [0]:
```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batc
hnormalization-function-in-keras

from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
model_drop_5 = Sequential()

model_drop_5.add(Dense(512, activation='relu', input_shape=(input_dim
,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None
)))
model_drop_5.add(BatchNormalization())
model_drop_5.add(Dropout(0.8))

model_drop_5.add(Dense(256, activation='relu', kernel_initializer=Rando
mNormal(mean=0.0, stddev=0.55, seed=None)) )
model_drop_5.add(BatchNormalization())
model_drop_5.add(Dropout(0.7))

model_drop_5.add(Dense(128, activation='relu', kernel_initializer=Rando
mNormal(mean=0.0, stddev=0.55, seed=None)) )
model_drop_5.add(BatchNormalization())
model_drop_5.add(Dropout(0.6))

model_drop_5.add(Dense(64, activation='relu', kernel_initializer=Random
Normal(mean=0.0, stddev=0.55, seed=None)) )
model_drop_5.add(BatchNormalization())
model_drop_5.add(Dropout(0.5))

model_drop_5.add(Dense(32, activation='relu', kernel_initializer=Random
Normal(mean=0.0, stddev=0.55, seed=None)) )
model_drop_5.add(BatchNormalization())
model_drop_5.add(Dropout(0.4))

model_drop_5.add(Dense(output_dim, activation='softmax'))
```

In [0]:
```
model_drop_5.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
dense_15 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_6 (Batch (None, 512)               2048
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_16 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_7 (Batch (None, 256)               1024
_________________________________________________________________
dropout_7 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_17 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_8 (Batch (None, 128)               512
_________________________________________________________________
dropout_8 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_18 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_9 (Batch (None, 64)                256
_________________________________________________________________
dropout_9 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_19 (Dense)             (None, 32)                2080
_________________________________________________________________
batch_normalization_10 (Batc (None, 32)                128
_________________________________________________________________
dropout_10 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_20 (Dense)             (None, 10)                330
=================================================================
Total params: 580,778
Trainable params: 578,794
Non-trainable params: 1,984
_________________________________________________________________
```

In [26]:
```
model_drop_5.compile(optimizer='adam', loss='categorical_crossentropy',
```

```
 metrics=['accuracy'])

history = model_drop_5.fit(X_train, Y_train, batch_size=batch_size, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 13s 209us/step - loss:
2.4181 - acc: 0.1676 - val_loss: 1.9354 - val_acc: 0.4156
Epoch 2/20
60000/60000 [==============================] - 11s 182us/step - loss:
1.8200 - acc: 0.3338 - val_loss: 1.1060 - val_acc: 0.6734
Epoch 3/20
60000/60000 [==============================] - 11s 187us/step - loss:
1.3506 - acc: 0.4993 - val_loss: 0.8121 - val_acc: 0.7470
Epoch 4/20
60000/60000 [==============================] - 11s 183us/step - loss:
1.1110 - acc: 0.5875 - val_loss: 0.6596 - val_acc: 0.7935
Epoch 5/20
60000/60000 [==============================] - 11s 181us/step - loss:
0.9731 - acc: 0.6512 - val_loss: 0.5571 - val_acc: 0.8273
Epoch 6/20
60000/60000 [==============================] - 11s 183us/step - loss:
0.8757 - acc: 0.6937 - val_loss: 0.4738 - val_acc: 0.8666
Epoch 7/20
60000/60000 [==============================] - 11s 183us/step - loss:
0.7992 - acc: 0.7297 - val_loss: 0.4131 - val_acc: 0.8917
Epoch 8/20
60000/60000 [==============================] - 11s 186us/step - loss:
0.7349 - acc: 0.7582 - val_loss: 0.3691 - val_acc: 0.9019
Epoch 9/20
60000/60000 [==============================] - 11s 184us/step - loss:
0.6883 - acc: 0.7830 - val_loss: 0.3261 - val_acc: 0.9145
Epoch 10/20
60000/60000 [==============================] - 11s 183us/step - loss:
0.6301 - acc: 0.8059 - val_loss: 0.3023 - val_acc: 0.9180
Epoch 11/20
60000/60000 [==============================] - 11s 183us/step - loss:
0.5869 - acc: 0.8256 - val_loss: 0.2608 - val_acc: 0.9299
Epoch 12/20
```

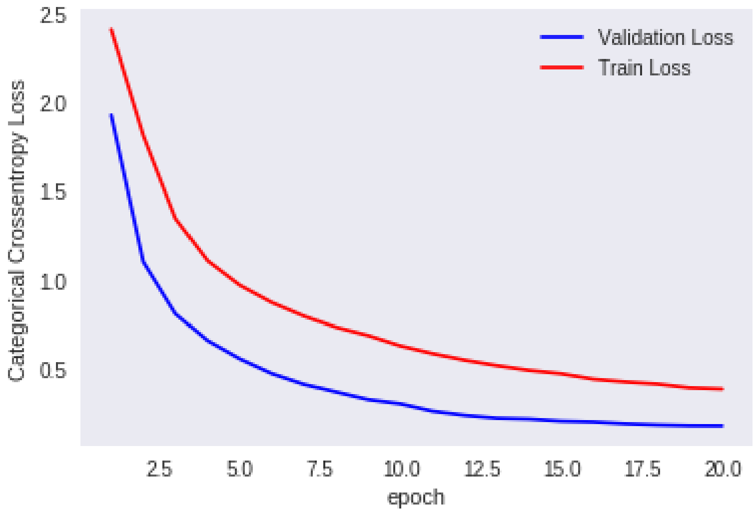
```
60000/60000 [==============================] - 11s 184us/step - loss:
0.5497 - acc: 0.8404 - val_loss: 0.2367 - val_acc: 0.9353
Epoch 13/20
60000/60000 [==============================] - 11s 183us/step - loss:
0.5190 - acc: 0.8523 - val_loss: 0.2205 - val_acc: 0.9409
Epoch 14/20
60000/60000 [==============================] - 11s 186us/step - loss:
0.4923 - acc: 0.8640 - val_loss: 0.2160 - val_acc: 0.9426
Epoch 15/20
60000/60000 [==============================] - 11s 187us/step - loss:
0.4735 - acc: 0.8736 - val_loss: 0.2039 - val_acc: 0.9459
Epoch 16/20
60000/60000 [==============================] - 11s 184us/step - loss:
0.4423 - acc: 0.8818 - val_loss: 0.1992 - val_acc: 0.9473
Epoch 17/20
60000/60000 [==============================] - 11s 183us/step - loss:
0.4267 - acc: 0.8888 - val_loss: 0.1888 - val_acc: 0.9511
Epoch 18/20
60000/60000 [==============================] - 11s 183us/step - loss:
0.4148 - acc: 0.8916 - val_loss: 0.1823 - val_acc: 0.9540
Epoch 19/20
60000/60000 [==============================] - 11s 182us/step - loss:
0.3932 - acc: 0.8989 - val_loss: 0.1781 - val_acc: 0.9551
Epoch 20/20
60000/60000 [==============================] - 12s 196us/step - loss:
0.3867 - acc: 0.9021 - val_loss: 0.1772 - val_acc: 0.9543
```

In [27]:

```
score = model_drop_5.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

```
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter vali
dation_data
# val_loss : validation loss
# val_acc : validation accuracy

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

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

```
Test score: 0.1772317446306348
Test accuracy: 0.9543
```



MLP of 5-hidden Layers + AdamOptimizer

In [28]:
```
# 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
from keras.initializers import he_normal
model_nodrop_5 = Sequential()

model_nodrop_5.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal( seed=None)))

model_nodrop_5.add(Dense(256, activation='relu', kernel_initializer=he_normal(seed=None)) )

model_nodrop_5.add(Dense(128, activation='relu', kernel_initializer=he_normal(seed=None)) )

model_nodrop_5.add(Dense(64, activation='relu', kernel_initializer=he_normal(seed=None)) )


model_nodrop_5.add(Dense(32, activation='relu', kernel_initializer=he_normal(seed=None)) )

model_nodrop_5.add(Dense(output_dim, activation='softmax'))
model_nodrop_5.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_21 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_22 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_23 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_24 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_25 (Dense)             (None, 32)                2080
```

```
dense_26 (Dense)             (None, 10)                330
=================================================================
Total params: 576,810
Trainable params: 576,810
Non-trainable params: 0
_________________________________________________________________
```

In [29]:

```
model_nodrop_5.compile(optimizer='adam', loss='categorical_crossentrop
y', metrics=['accuracy'])

history = model_nodrop_5.fit(X_train, Y_train, batch_size=batch_size, e
pochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.
2507 - acc: 0.9238 - val_loss: 0.1110 - val_acc: 0.9677
Epoch 2/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.
0908 - acc: 0.9722 - val_loss: 0.1099 - val_acc: 0.9668
Epoch 3/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.
0612 - acc: 0.9810 - val_loss: 0.1000 - val_acc: 0.9708
Epoch 4/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.
0460 - acc: 0.9854 - val_loss: 0.0849 - val_acc: 0.9779
Epoch 5/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.
0379 - acc: 0.9879 - val_loss: 0.0842 - val_acc: 0.9773
Epoch 6/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.
0288 - acc: 0.9907 - val_loss: 0.0949 - val_acc: 0.9745
Epoch 7/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.
0297 - acc: 0.9908 - val_loss: 0.0709 - val_acc: 0.9819
Epoch 8/20
60000/60000 [==============================] - 8s 141us/step - loss: 0.
0190 - acc: 0.9938 - val_loss: 0.0953 - val_acc: 0.9778
Epoch 9/20
```

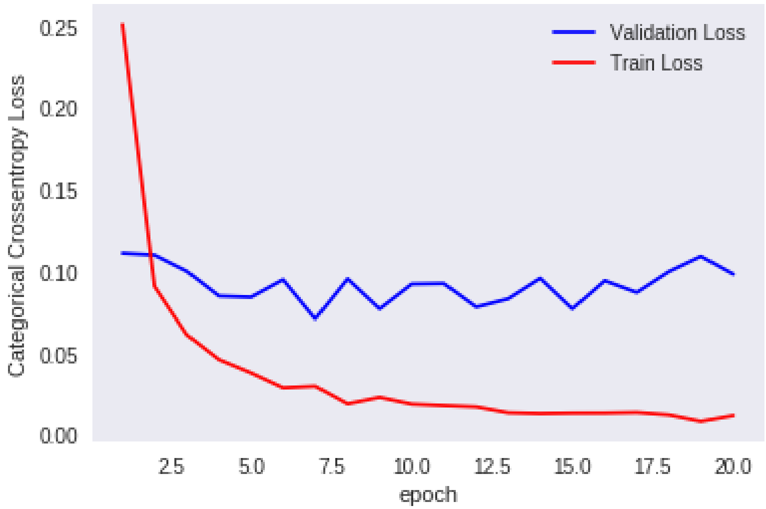
```
60000/60000 [==============================] - 8s 138us/step - loss: 0.
0229 - acc: 0.9927 - val_loss: 0.0770 - val_acc: 0.9791
Epoch 10/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.
0188 - acc: 0.9937 - val_loss: 0.0922 - val_acc: 0.9793
Epoch 11/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.
0179 - acc: 0.9943 - val_loss: 0.0925 - val_acc: 0.9781
Epoch 12/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.
0171 - acc: 0.9944 - val_loss: 0.0781 - val_acc: 0.9807
Epoch 13/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.
0135 - acc: 0.9957 - val_loss: 0.0831 - val_acc: 0.9787
Epoch 14/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.
0130 - acc: 0.9960 - val_loss: 0.0957 - val_acc: 0.9805
Epoch 15/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.
0133 - acc: 0.9958 - val_loss: 0.0770 - val_acc: 0.9809
Epoch 16/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.
0133 - acc: 0.9959 - val_loss: 0.0942 - val_acc: 0.9765
Epoch 17/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.
0136 - acc: 0.9956 - val_loss: 0.0870 - val_acc: 0.9821
Epoch 18/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.
0122 - acc: 0.9963 - val_loss: 0.0998 - val_acc: 0.9787
Epoch 19/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.
0082 - acc: 0.9974 - val_loss: 0.1091 - val_acc: 0.9765
Epoch 20/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.
0118 - acc: 0.9962 - val_loss: 0.0982 - val_acc: 0.9805
```

In [30]:

```
score = model_nodrop_5.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

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

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

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

# we will get val_loss and val_acc only when you pass the paramter vali
dation_data
# val_loss : validation loss
# val_acc : validation accuracy

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

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

```
Test score: 0.09818555498997499
Test accuracy: 0.9805
```



MLP with Batch Normalization and Dropout

In [33]:
```
import pandas as pd
models = pd.DataFrame({'Hidden layers': ['2', '3', "5"], 'Test score' :
 [0.06,0.09,0.17],'Accuracy': [0.9812,0.9764,0.9543]}, columns = ["Hidd
en layers","Test score","Accuracy"])
models
```

Out[33]:

	Hidden layers	Test score	Accuracy
0	2	0.06	0.9812
1	3	0.09	0.9764
2	5	0.17	0.9543

MLP without Batch Normalization and Dropout

In [34]:
```
models = pd.DataFrame({'Hidden layers': ['2', '3', "5"], 'Test score' :
 [0.12,0.13,0.09],'Accuracy': [0.9783,0.9747,0.9805]}, columns = ["Hidd
```

```
en layers","Test score","Accuracy"])
models
```

Out[34]:

	Hidden layers	Test score	Accuracy
0	2	0.12	0.9783
1	3	0.13	0.9747
2	5	0.09	0.9805

Conclusion

1.Load mnist dataset.

2.Split the dataset into train and test.

3.converting the input images from 3d to 2d vectors.

4.Normalize the data.

5.Implement Softmax classifier with 2 , 3 and 5 hidden layers with batchnormalizer and different dropoutrates.

6.Implement Softmax classifier with 2 , 3 and 5 hidden layers without batchnormalizer and dropoutrates.

7.Ploting Categorical Crossentropy Loss VS No.of Epochs plot .