OBJECTIVE :- Apply different MLP Architectures on MNIST dataset

In [1]:

```
# Importing libraries
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import time

# the data, shuffled and split between train and test sets
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()

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

```
Using TensorFlow backend.

Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz (http
s://s3.amazonaws.com/img-datasets/mnist.npz)
11493376/11490434 [==============================] - 1s 0us/step
Number of training examples : 60000 and each image is of shape (28, 28)
Number of test examples : 10000 and each image is of shape (28, 28)
```

In [2]:

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

# 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)"%
print("Number of test examples :", X_test.shape[0], "and each image is of shape (%d)"%(X_te
```

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

In [3]:

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

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

In [5]:

```
# 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

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

```
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
```

In [6]:

```
# 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]:

```
# this function is used draw Categorical Crossentropy Loss VS No. of epochs plot
def plt_dynamic(x, vy, ty):
  plt.figure(figsize=(10,5))
  plt.plot(x, vy, 'b', label="Validation Loss")
  plt.plot(x, ty, 'r', label="Train Loss")
  plt.xlabel('Epochs')
  plt.ylabel('Categorical Crossentropy Loss')
  plt.title('\nCategorical Crossentropy Loss VS Epochs')
  plt.legend()
  plt.grid()
  plt.show()
```

(1). Softmax Classifier with 2 hidden layers

(1.a) Without dropout and Batch normalization

In [8]:

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

# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20

# Initialising model
model_2 = Sequential()

# Adding first hidden layer
model_2.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=he_r

# Adding second hidden layer
model_2.add(Dense(256, activation='relu', kernel_initializer=he_normal(seed=None)))

# Adding output layer
model_2.add(Dense(output_dim, activation='softmax'))

# Printing model Summary
print("Model Summary :- \n",model_2.summary())

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

# Fitting the data to the model
history_2 = model_2.fit(X_train, y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1
```

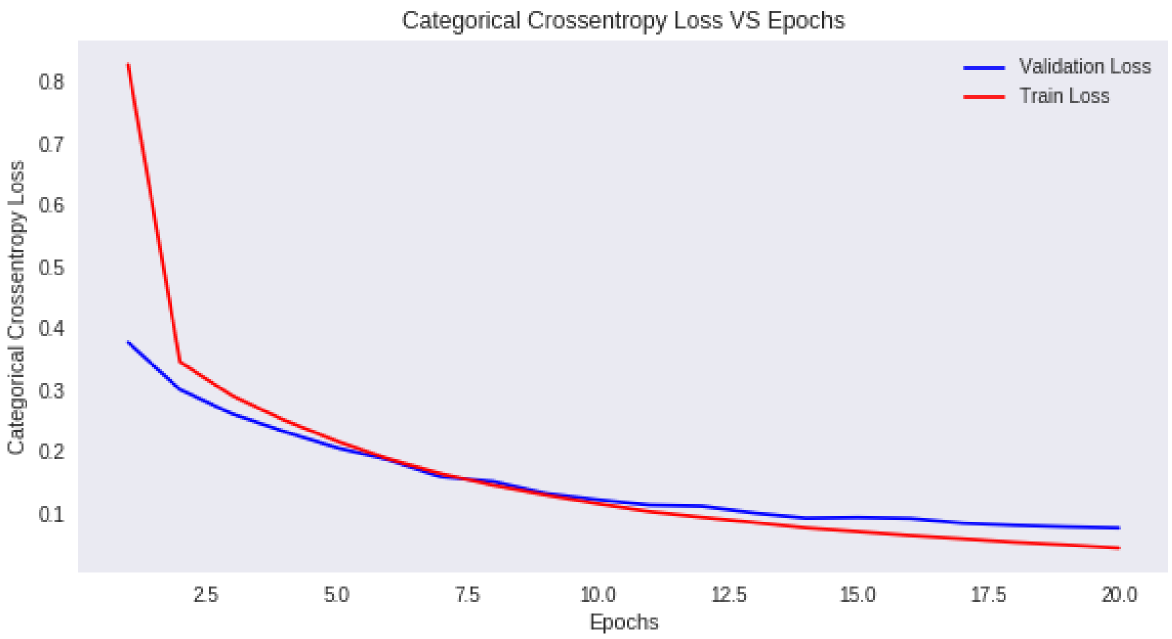
```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/py
thon/framework/op_def_library.py:263: colocate_with (from tensorflow.python.
framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 512)               401920    
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328    
_________________________________________________________________
dense_3 (Dense)              (None, 10)                2570      
=================================================================
Total params: 535,818
Trainable params: 535,818
Non-trainable params: 0
_________________________________________________________________
Model Summary :- 
 None
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/py
thon/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 60000 samples, validate on 10000 samples
```

```
Epoch 1/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.8276
- acc: 0.7720 - val_loss: 0.3774 - val_acc: 0.8896
Epoch 2/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.3455
- acc: 0.8997 - val_loss: 0.3008 - val_acc: 0.9122
Epoch 3/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.2907
- acc: 0.9147 - val_loss: 0.2613 - val_acc: 0.9213
Epoch 4/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.2510
- acc: 0.9266 - val_loss: 0.2327 - val_acc: 0.9303
Epoch 5/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.2174
- acc: 0.9358 - val_loss: 0.2067 - val_acc: 0.9391
Epoch 6/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.1882
- acc: 0.9445 - val_loss: 0.1868 - val_acc: 0.9446
Epoch 7/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.1647
- acc: 0.9520 - val_loss: 0.1594 - val_acc: 0.9523
Epoch 8/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.1457
- acc: 0.9576 - val_loss: 0.1517 - val_acc: 0.9530
Epoch 9/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.1302
- acc: 0.9618 - val_loss: 0.1327 - val_acc: 0.9594
Epoch 10/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.1158
- acc: 0.9662 - val_loss: 0.1224 - val_acc: 0.9617
Epoch 11/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.1030
- acc: 0.9694 - val_loss: 0.1135 - val_acc: 0.9646
Epoch 12/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0939
- acc: 0.9722 - val_loss: 0.1117 - val_acc: 0.9674
Epoch 13/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0855
- acc: 0.9742 - val_loss: 0.1008 - val_acc: 0.9694
Epoch 14/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0766
- acc: 0.9769 - val_loss: 0.0925 - val_acc: 0.9710
Epoch 15/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0706
- acc: 0.9788 - val_loss: 0.0936 - val_acc: 0.9711
Epoch 16/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0643
- acc: 0.9801 - val_loss: 0.0921 - val_acc: 0.9723
Epoch 17/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0590
- acc: 0.9818 - val_loss: 0.0844 - val_acc: 0.9746
Epoch 18/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0535
- acc: 0.9841 - val_loss: 0.0808 - val_acc: 0.9746
Epoch 19/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0490
- acc: 0.9853 - val_loss: 0.0783 - val_acc: 0.9761
Epoch 20/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0439
- acc: 0.9872 - val_loss: 0.0764 - val_acc: 0.9759
```

In [9]:

```
# Evaluating the model
score = model_2.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

# Test and train accuracy of the model
model_2_test = score[1]
model_2_train = history_2.history['acc']

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,nb_epoch+1))

# Validation loss
vy = history_2.history['val_loss']
# Training loss
ty = history_2.history['loss']

# Calling the function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.07642286013420671
Test accuracy: 0.9759
```



(1.b) With dropout and Batch Normalization

In [10]:

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

# Initialising model
model_2d = Sequential()

# Adding first hidden layer
model_2d.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=he_
# Adding Batch Normalization
model_2d.add(BatchNormalization())
# Adding dropout to first hidden layer
model_2d.add(Dropout(0.4))

# Adding second hidden layer
model_2d.add(Dense(256, activation='relu', kernel_initializer=he_normal(seed=None)))
# Adding Batch Normalization
model_2d.add(BatchNormalization())
# Adding dropout to second hidden layer
model_2d.add(Dropout(0.4))

# Adding output layer
model_2d.add(Dense(output_dim, activation='softmax'))

# Printing model Summary
print("Model Summary :- \n",model_2d.summary())

# Compiling the model
model_2d.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Fitting the data to the model
history_2d = model_2d.fit(X_train, y_train, batch_size=batch_size, epochs=nb_epoch, verbose
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backen
d/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`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 512)               401920
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               131328
_________________________________________________________________
batch_normalization_2 (Batch (None, 256)               1024
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                2570
=================================================================
Total params: 538,890
Trainable params: 537,354
Non-trainable params: 1,536
_________________________________________________________________
```

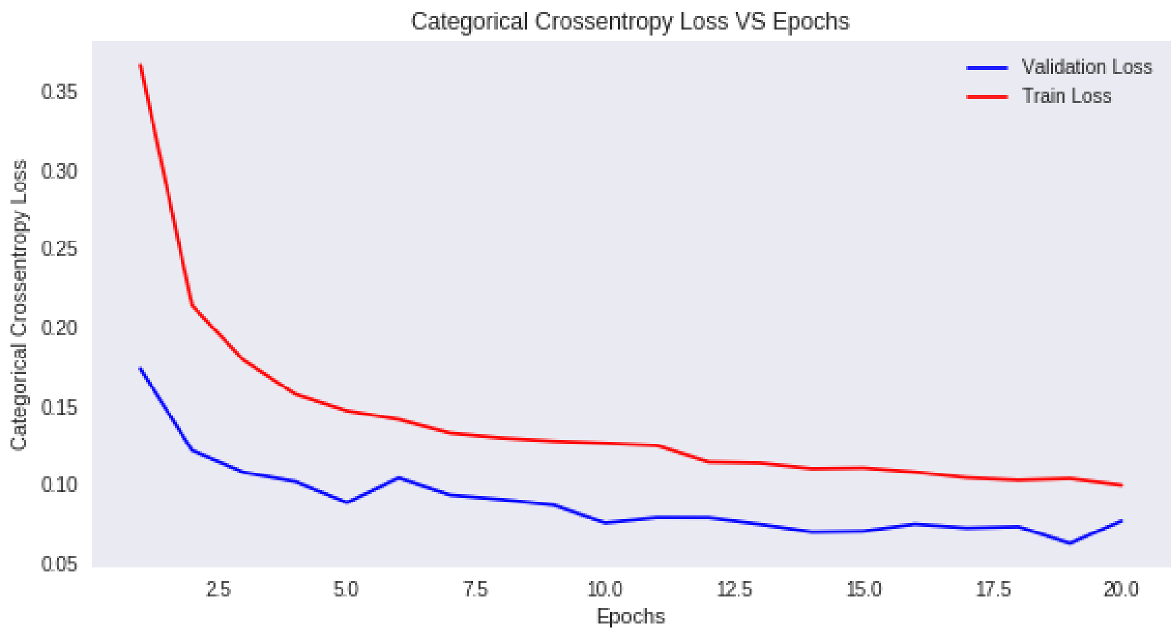
```
Model Summary :-
 None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 14s 229us/step - loss: 0.3672
- acc: 0.8881 - val_loss: 0.1736 - val_acc: 0.9480
Epoch 2/20
60000/60000 [==============================] - 13s 218us/step - loss: 0.2139
- acc: 0.9354 - val_loss: 0.1216 - val_acc: 0.9627
Epoch 3/20
60000/60000 [==============================] - 12s 203us/step - loss: 0.1791
- acc: 0.9451 - val_loss: 0.1077 - val_acc: 0.9673
Epoch 4/20
60000/60000 [==============================] - 12s 203us/step - loss: 0.1575
- acc: 0.9523 - val_loss: 0.1018 - val_acc: 0.9676
Epoch 5/20
60000/60000 [==============================] - 12s 201us/step - loss: 0.1469
- acc: 0.9544 - val_loss: 0.0884 - val_acc: 0.9717
Epoch 6/20
60000/60000 [==============================] - 12s 202us/step - loss: 0.1414
- acc: 0.9567 - val_loss: 0.1041 - val_acc: 0.9669
Epoch 7/20
60000/60000 [==============================] - 12s 200us/step - loss: 0.1328
- acc: 0.9585 - val_loss: 0.0933 - val_acc: 0.9722
Epoch 8/20
60000/60000 [==============================] - 12s 194us/step - loss: 0.1296
- acc: 0.9600 - val_loss: 0.0903 - val_acc: 0.9726
Epoch 9/20
60000/60000 [==============================] - 11s 187us/step - loss: 0.1275
- acc: 0.9601 - val_loss: 0.0869 - val_acc: 0.9738
Epoch 10/20
60000/60000 [==============================] - 12s 201us/step - loss: 0.1262
- acc: 0.9614 - val_loss: 0.0755 - val_acc: 0.9762
Epoch 11/20
60000/60000 [==============================] - 12s 198us/step - loss: 0.1248
- acc: 0.9611 - val_loss: 0.0790 - val_acc: 0.9753
Epoch 12/20
60000/60000 [==============================] - 11s 190us/step - loss: 0.1143
- acc: 0.9640 - val_loss: 0.0789 - val_acc: 0.9771
Epoch 13/20
60000/60000 [==============================] - 11s 191us/step - loss: 0.1137
- acc: 0.9650 - val_loss: 0.0746 - val_acc: 0.9760
Epoch 14/20
60000/60000 [==============================] - 11s 182us/step - loss: 0.1100
- acc: 0.9658 - val_loss: 0.0697 - val_acc: 0.9797
Epoch 15/20
60000/60000 [==============================] - 11s 186us/step - loss: 0.1105
- acc: 0.9648 - val_loss: 0.0701 - val_acc: 0.9784
Epoch 16/20
60000/60000 [==============================] - 12s 192us/step - loss: 0.1079
- acc: 0.9658 - val_loss: 0.0747 - val_acc: 0.9775
Epoch 17/20
60000/60000 [==============================] - 11s 189us/step - loss: 0.1043
- acc: 0.9676 - val_loss: 0.0722 - val_acc: 0.9788
Epoch 18/20
60000/60000 [==============================] - 11s 184us/step - loss: 0.1027
- acc: 0.9674 - val_loss: 0.0730 - val_acc: 0.9788
Epoch 19/20
60000/60000 [==============================] - 12s 195us/step - loss: 0.1037
- acc: 0.9678 - val_loss: 0.0625 - val_acc: 0.9804
Epoch 20/20
```

```
60000/60000 [==============================] - 11s 191us/step - loss: 0.0995
- acc: 0.9690 - val_loss: 0.0767 - val_acc: 0.9764
```

In [12]:

```
# Evaluating the model
score = model_2d.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

# Test and train accuracy of the model
model_2d_test = score[1]
model_2d_train = history_2d.history['acc']

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,nb_epoch+1))

# Validation loss
vy = history_2d.history['val_loss']
# Training loss
ty = history_2d.history['loss']

# Calling the function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.07667584380609915
Test accuracy: 0.9764
```



(2). Softmax Classifier with 3 hidden layers

(2.a) Without Dropout and Batch Normalization

In [13]:

```
# Initialising model
model_3 = Sequential()

# Adding first hidden layer
model_3.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=he_r

# Adding second hidden layer
model_3.add(Dense(256, activation='relu', kernel_initializer=he_normal(seed=None)))

# Adding third hidden layer
model_3.add(Dense(128, activation='relu', kernel_initializer=he_normal(seed=None)))

# Adding output layer
model_3.add(Dense(output_dim, activation='softmax'))

# Printing model Summary
print(model_3.summary())

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

# Fitting the data to the model
history_3 = model_3.fit(X_train, y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1
```

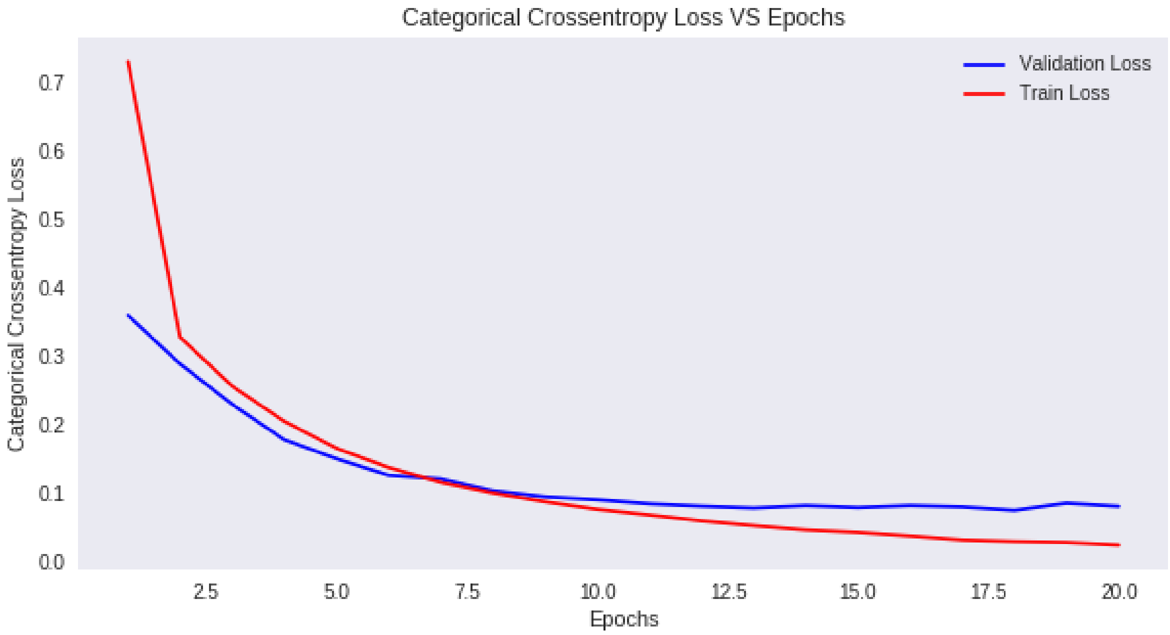
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_7 (Dense)              (None, 512)               401920    
_________________________________________________________________
dense_8 (Dense)              (None, 256)               131328    
_________________________________________________________________
dense_9 (Dense)              (None, 128)               32896     
_________________________________________________________________
dense_10 (Dense)             (None, 10)                1290      
=================================================================
Total params: 567,434
Trainable params: 567,434
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 187us/step - loss: 0.7319
- acc: 0.7790 - val_loss: 0.3602 - val_acc: 0.8931
Epoch 2/20
60000/60000 [==============================] - 11s 179us/step - loss: 0.3282
- acc: 0.9025 - val_loss: 0.2890 - val_acc: 0.9137
Epoch 3/20
60000/60000 [==============================] - 11s 181us/step - loss: 0.2564
- acc: 0.9239 - val_loss: 0.2298 - val_acc: 0.9279
Epoch 4/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.2048
- acc: 0.9384 - val_loss: 0.1779 - val_acc: 0.9459
Epoch 5/20
60000/60000 [==============================] - 11s 182us/step - loss: 0.1654
- acc: 0.9505 - val_loss: 0.1503 - val_acc: 0.9540
Epoch 6/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.1374
- acc: 0.9585 - val_loss: 0.1259 - val_acc: 0.9617
```

```
Epoch 7/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.1156
- acc: 0.9656 - val_loss: 0.1206 - val_acc: 0.9635
Epoch 8/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.1000
- acc: 0.9700 - val_loss: 0.1032 - val_acc: 0.9676
Epoch 9/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.0872
- acc: 0.9736 - val_loss: 0.0944 - val_acc: 0.9712
Epoch 10/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.0761
- acc: 0.9765 - val_loss: 0.0901 - val_acc: 0.9730
Epoch 11/20
60000/60000 [==============================] - 10s 169us/step - loss: 0.0677
- acc: 0.9793 - val_loss: 0.0842 - val_acc: 0.9731
Epoch 12/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.0592
- acc: 0.9817 - val_loss: 0.0806 - val_acc: 0.9756
Epoch 13/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.0523
- acc: 0.9838 - val_loss: 0.0779 - val_acc: 0.9756
Epoch 14/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.0460
- acc: 0.9857 - val_loss: 0.0816 - val_acc: 0.9754
Epoch 15/20
60000/60000 [==============================] - 11s 176us/step - loss: 0.0423
- acc: 0.9863 - val_loss: 0.0786 - val_acc: 0.9768
Epoch 16/20
60000/60000 [==============================] - 11s 178us/step - loss: 0.0370
- acc: 0.9881 - val_loss: 0.0816 - val_acc: 0.9750
Epoch 17/20
60000/60000 [==============================] - 11s 175us/step - loss: 0.0312
- acc: 0.9906 - val_loss: 0.0796 - val_acc: 0.9764
Epoch 18/20
60000/60000 [==============================] - 11s 178us/step - loss: 0.0288
- acc: 0.9907 - val_loss: 0.0746 - val_acc: 0.9774
Epoch 19/20
60000/60000 [==============================] - 11s 179us/step - loss: 0.0275
- acc: 0.9913 - val_loss: 0.0851 - val_acc: 0.9747
Epoch 20/20
60000/60000 [==============================] - 11s 180us/step - loss: 0.0237
- acc: 0.9925 - val_loss: 0.0804 - val_acc: 0.9771
```

In [14]:

```
# Evaluating the model
score = model_3.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

# Test and train accuracy of the model
model_3_test = score[1]
model_3_train = history_3.history['acc']

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,nb_epoch+1))

# Validation loss
vy = history_3.history['val_loss']
# Training loss
ty = history_3.history['loss']

# Calling the function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.08041134356782713
Test accuracy: 0.9771
```



(2.b) With Droput and Batch Normalization

In [15]:

```
model_3d = Sequential()

# Adding first hidden layer
model_3d.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=he_
# Adding Batch Normalization
model_3d.add(BatchNormalization())
# Adding dropout
model_3d.add(Dropout(0.4))

# Adding second hidden layer
model_3d.add(Dense(256, activation='relu', kernel_initializer=he_normal(seed=None)))
# Adding Batch Normalization
model_3d.add(BatchNormalization())
# Adding dropout
model_3d.add(Dropout(0.4))

# Adding third hidden layer
model_3d.add(Dense(128, activation='relu', kernel_initializer=he_normal(seed=None)))
# Adding Batch Normalization
model_3d.add(BatchNormalization())
# Adding dropout
model_3d.add(Dropout(0.4))

# Adding output layer
model_3d.add(Dense(output_dim, activation='softmax'))

# Printing model Summary
print(model_3d.summary())

# Compiling the model
model_3d.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Fitting the data to the model
history_3d = model_3d.fit(X_train, y_train, batch_size=batch_size, epochs=nb_epoch, verbose
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_11 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_3 (Batch (None, 512)               2048      
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_12 (Dense)             (None, 256)               131328    
_________________________________________________________________
batch_normalization_4 (Batch (None, 256)               1024      
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_13 (Dense)             (None, 128)               32896     
_________________________________________________________________
batch_normalization_5 (Batch (None, 128)               512       
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_14 (Dense)             (None, 10)                1290      
=================================================================
```

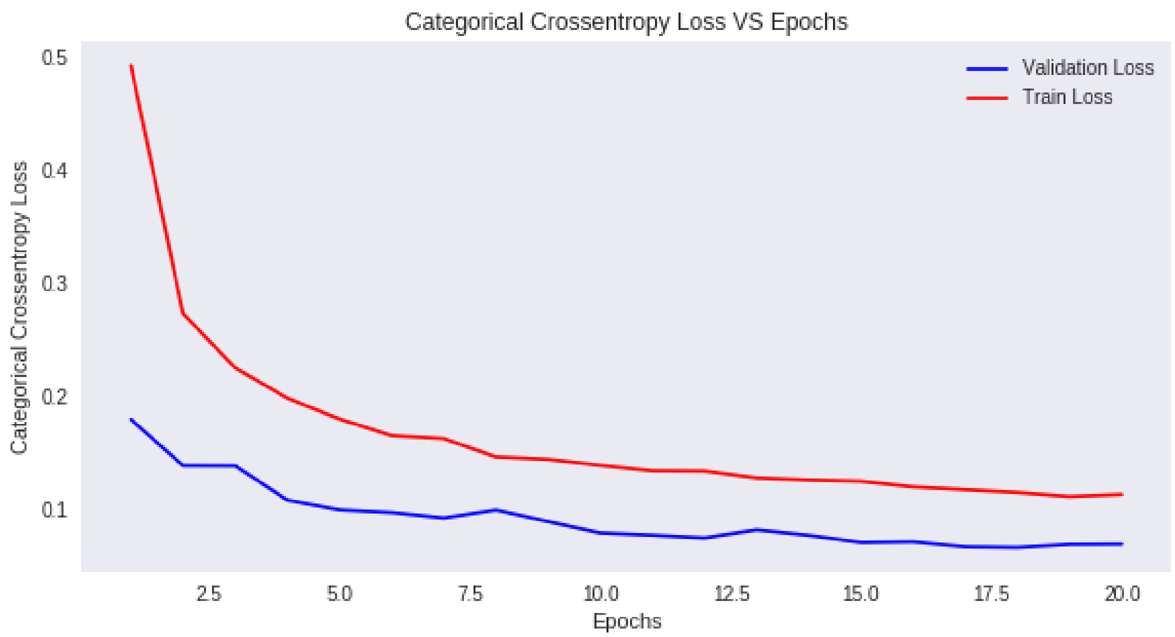
```
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 13s 225us/step - loss: 0.4926
- acc: 0.8490 - val_loss: 0.1797 - val_acc: 0.9451
Epoch 2/20
60000/60000 [==============================] - 12s 198us/step - loss: 0.2733
- acc: 0.9176 - val_loss: 0.1389 - val_acc: 0.9561
Epoch 3/20
60000/60000 [==============================] - 12s 200us/step - loss: 0.2253
- acc: 0.9319 - val_loss: 0.1387 - val_acc: 0.9557
Epoch 4/20
60000/60000 [==============================] - 12s 207us/step - loss: 0.1986
- acc: 0.9391 - val_loss: 0.1084 - val_acc: 0.9660
Epoch 5/20
60000/60000 [==============================] - 12s 200us/step - loss: 0.1800
- acc: 0.9454 - val_loss: 0.0997 - val_acc: 0.9699
Epoch 6/20
60000/60000 [==============================] - 13s 216us/step - loss: 0.1655
- acc: 0.9482 - val_loss: 0.0973 - val_acc: 0.9691
Epoch 7/20
60000/60000 [==============================] - 13s 210us/step - loss: 0.1628
- acc: 0.9503 - val_loss: 0.0925 - val_acc: 0.9698
Epoch 8/20
60000/60000 [==============================] - 12s 205us/step - loss: 0.1466
- acc: 0.9549 - val_loss: 0.0996 - val_acc: 0.9666
Epoch 9/20
60000/60000 [==============================] - 12s 203us/step - loss: 0.1444
- acc: 0.9556 - val_loss: 0.0894 - val_acc: 0.9720
Epoch 10/20
60000/60000 [==============================] - 12s 200us/step - loss: 0.1391
- acc: 0.9566 - val_loss: 0.0793 - val_acc: 0.9767
Epoch 11/20
60000/60000 [==============================] - 12s 201us/step - loss: 0.1343
- acc: 0.9587 - val_loss: 0.0773 - val_acc: 0.9756
Epoch 12/20
60000/60000 [==============================] - 12s 195us/step - loss: 0.1340
- acc: 0.9587 - val_loss: 0.0749 - val_acc: 0.9761
Epoch 13/20
60000/60000 [==============================] - 12s 201us/step - loss: 0.1280
- acc: 0.9600 - val_loss: 0.0821 - val_acc: 0.9749
Epoch 14/20
60000/60000 [==============================] - 12s 193us/step - loss: 0.1261
- acc: 0.9607 - val_loss: 0.0772 - val_acc: 0.9744
Epoch 15/20
60000/60000 [==============================] - 12s 204us/step - loss: 0.1250
- acc: 0.9608 - val_loss: 0.0709 - val_acc: 0.9768
Epoch 16/20
60000/60000 [==============================] - 12s 208us/step - loss: 0.1201
- acc: 0.9619 - val_loss: 0.0715 - val_acc: 0.9776
Epoch 17/20
60000/60000 [==============================] - 12s 207us/step - loss: 0.1176
- acc: 0.9638 - val_loss: 0.0671 - val_acc: 0.9794
Epoch 18/20
60000/60000 [==============================] - 12s 208us/step - loss: 0.1152
- acc: 0.9642 - val_loss: 0.0665 - val_acc: 0.9785
Epoch 19/20
```

```
60000/60000 [==============================] - 12s 205us/step - loss: 0.1115
- acc: 0.9649 - val_loss: 0.0692 - val_acc: 0.9774
Epoch 20/20
60000/60000 [==============================] - 13s 209us/step - loss: 0.1134
- acc: 0.9644 - val_loss: 0.0695 - val_acc: 0.9780
```

In [16]:

```
# Evaluating the model
score = model_3d.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

# Test and train accuracy of the model
model_3d_test = score[1]
model_3d_train = history_3d.history['acc']

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,nb_epoch+1))

# Validation loss
vy = history_3d.history['val_loss']
# Training loss
ty = history_3d.history['loss']

# Calling the function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.06945714832819067
Test accuracy: 0.978
```



(3). Softmax Classifier with 5 hidden layers

(3.a) Without Dropout and Batch Normalization

In [17]:

```
# Initialising model
model_5 = Sequential()

# Adding first hidden layer
model_5.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=he_r

# Adding second hidden layer
model_5.add(Dense(256, activation='relu', kernel_initializer=he_normal(seed=None)))

# Adding third hidden layer
model_5.add(Dense(128, activation='relu', kernel_initializer=he_normal(seed=None)))

# Adding fourth hidden layer
model_5.add(Dense(64, activation='relu', kernel_initializer=he_normal(seed=None)))

# Adding fifth hidden layer
model_5.add(Dense(32, activation='relu', kernel_initializer=he_normal(seed=None)))

# Adding output layer
model_5.add(Dense(output_dim, activation='softmax'))

# Printing model Summary
print(model_5.summary())

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

# Fitting the data to the model
history_5 = model_5.fit(X_train, y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1
```

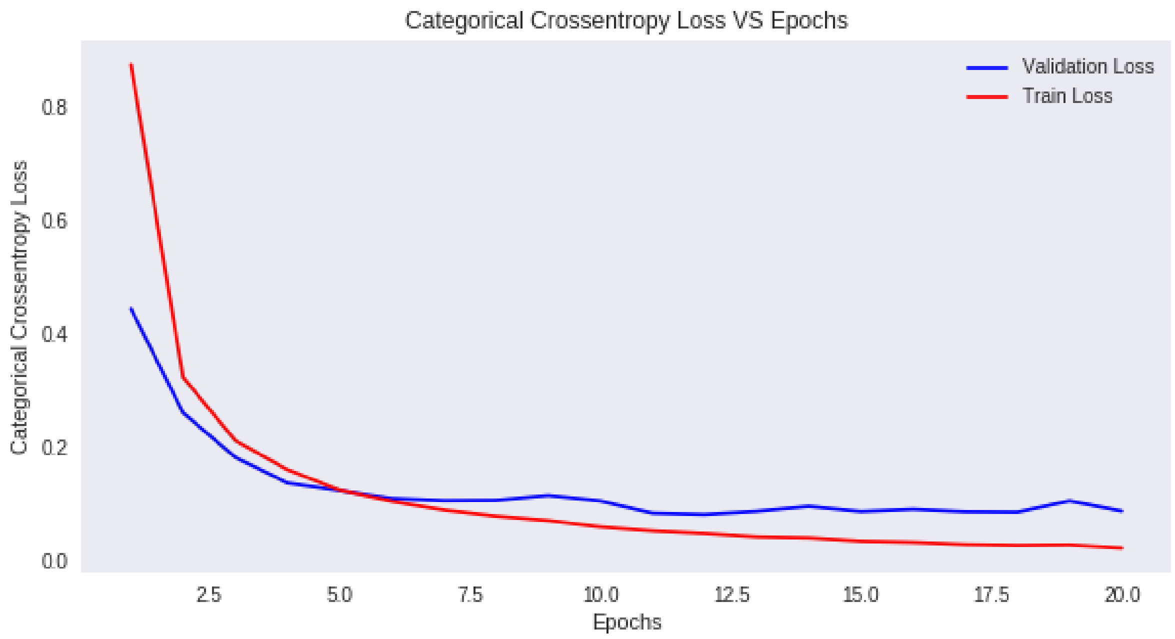
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_15 (Dense)             (None, 512)               401920    
_________________________________________________________________
dense_16 (Dense)             (None, 256)               131328    
_________________________________________________________________
dense_17 (Dense)             (None, 128)               32896     
_________________________________________________________________
dense_18 (Dense)             (None, 64)                8256      
_________________________________________________________________
dense_19 (Dense)             (None, 32)                2080      
_________________________________________________________________
dense_20 (Dense)             (None, 10)                330       
=================================================================
Total params: 576,810
Trainable params: 576,810
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 12s 192us/step - loss: 0.8731
- acc: 0.7113 - val_loss: 0.4438 - val_acc: 0.8727
Epoch 2/20
60000/60000 [==============================] - 11s 180us/step - loss: 0.3234
- acc: 0.9059 - val_loss: 0.2611 - val_acc: 0.9229
Epoch 3/20
60000/60000 [==============================] - 11s 182us/step - loss: 0.2123
```

```
- acc: 0.9378 - val_loss: 0.1829 - val_acc: 0.9456
Epoch 4/20
60000/60000 [==============================] - 11s 180us/step - loss: 0.1610
- acc: 0.9521 - val_loss: 0.1379 - val_acc: 0.9589
Epoch 5/20
60000/60000 [==============================] - 11s 176us/step - loss: 0.1254
- acc: 0.9620 - val_loss: 0.1246 - val_acc: 0.9621
Epoch 6/20
60000/60000 [==============================] - 11s 178us/step - loss: 0.1055
- acc: 0.9674 - val_loss: 0.1105 - val_acc: 0.9659
Epoch 7/20
60000/60000 [==============================] - 11s 181us/step - loss: 0.0903
- acc: 0.9731 - val_loss: 0.1066 - val_acc: 0.9695
Epoch 8/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.0793
- acc: 0.9753 - val_loss: 0.1072 - val_acc: 0.9662
Epoch 9/20
60000/60000 [==============================] - 11s 184us/step - loss: 0.0716
- acc: 0.9778 - val_loss: 0.1158 - val_acc: 0.9657
Epoch 10/20
60000/60000 [==============================] - 11s 186us/step - loss: 0.0606
- acc: 0.9812 - val_loss: 0.1062 - val_acc: 0.9683
Epoch 11/20
60000/60000 [==============================] - 11s 184us/step - loss: 0.0536
- acc: 0.9832 - val_loss: 0.0847 - val_acc: 0.9732
Epoch 12/20
60000/60000 [==============================] - 11s 185us/step - loss: 0.0490
- acc: 0.9845 - val_loss: 0.0822 - val_acc: 0.9765
Epoch 13/20
60000/60000 [==============================] - 11s 185us/step - loss: 0.0429
- acc: 0.9856 - val_loss: 0.0879 - val_acc: 0.9756
Epoch 14/20
60000/60000 [==============================] - 11s 175us/step - loss: 0.0411
- acc: 0.9865 - val_loss: 0.0967 - val_acc: 0.9753
Epoch 15/20
60000/60000 [==============================] - 11s 176us/step - loss: 0.0353
- acc: 0.9888 - val_loss: 0.0875 - val_acc: 0.9766
Epoch 16/20
60000/60000 [==============================] - 11s 177us/step - loss: 0.0334
- acc: 0.9892 - val_loss: 0.0915 - val_acc: 0.9761
Epoch 17/20
60000/60000 [==============================] - 11s 183us/step - loss: 0.0295
- acc: 0.9900 - val_loss: 0.0871 - val_acc: 0.9770
Epoch 18/20
60000/60000 [==============================] - 10s 169us/step - loss: 0.0280
- acc: 0.9908 - val_loss: 0.0865 - val_acc: 0.9769
Epoch 19/20
60000/60000 [==============================] - 10s 167us/step - loss: 0.0288
- acc: 0.9907 - val_loss: 0.1064 - val_acc: 0.9729
Epoch 20/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.0233
- acc: 0.9921 - val_loss: 0.0884 - val_acc: 0.9783
```

In [19]:

```
# Evaluating the model
score = model_5.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

# Test and train accuracy of the model
model_5_test = score[1]
model_5_train = history_5.history['acc']

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,nb_epoch+1))

# Validation loss
vy = history_5.history['val_loss']
# Training loss
ty = history_5.history['loss']

# Calling the function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.08839814674687223
Test accuracy: 0.9783
```



(3.b) With Dropout and Batch Normalisation

In [20]:

```
# Initialising model
model_5d = Sequential()

# Adding first hidden layer
model_5d.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer=he_
# Adding Batch Normalization
model_5d.add(BatchNormalization())
# Adding dropout
model_5d.add(Dropout(0.4))

# Adding second hidden layer
model_5d.add(Dense(256, activation='relu', kernel_initializer=he_normal(seed=None)))
# Adding Batch Normalization
model_5d.add(BatchNormalization())
# Adding dropout
model_5d.add(Dropout(0.4))

# Adding third hidden layer
model_5d.add(Dense(128, activation='relu', kernel_initializer=he_normal(seed=None)))
# Adding Batch Normalization
model_5d.add(BatchNormalization())
# Adding dropout
model_5d.add(Dropout(0.4))

# Adding fourth hidden layer
model_5d.add(Dense(64, activation='relu', kernel_initializer=he_normal(seed=None)))
# Adding Batch Normalization
model_5d.add(BatchNormalization())
# Adding dropout
model_5d.add(Dropout(0.4))

# Adding fifth hidden layer
model_5d.add(Dense(32, activation='relu', kernel_initializer=he_normal(seed=None)))
# Adding Batch Normalization
model_5d.add(BatchNormalization())
# Adding dropout
model_5d.add(Dropout(0.4))

# Adding output layer
model_5d.add(Dense(output_dim, activation='softmax'))

# Printing model Summary
print(model_5d.summary())

# Compiling the model
model_5d.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Fitting the data to the model
history_5d = model_5d.fit(X_train, y_train, batch_size=batch_size, epochs=nb_epoch, verbose
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_21 (Dense)             (None, 512)               401920    
_________________________________________________________________
batch_normalization_6 (Batch (None, 512)               2048      
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0         
_________________________________________________________________
```

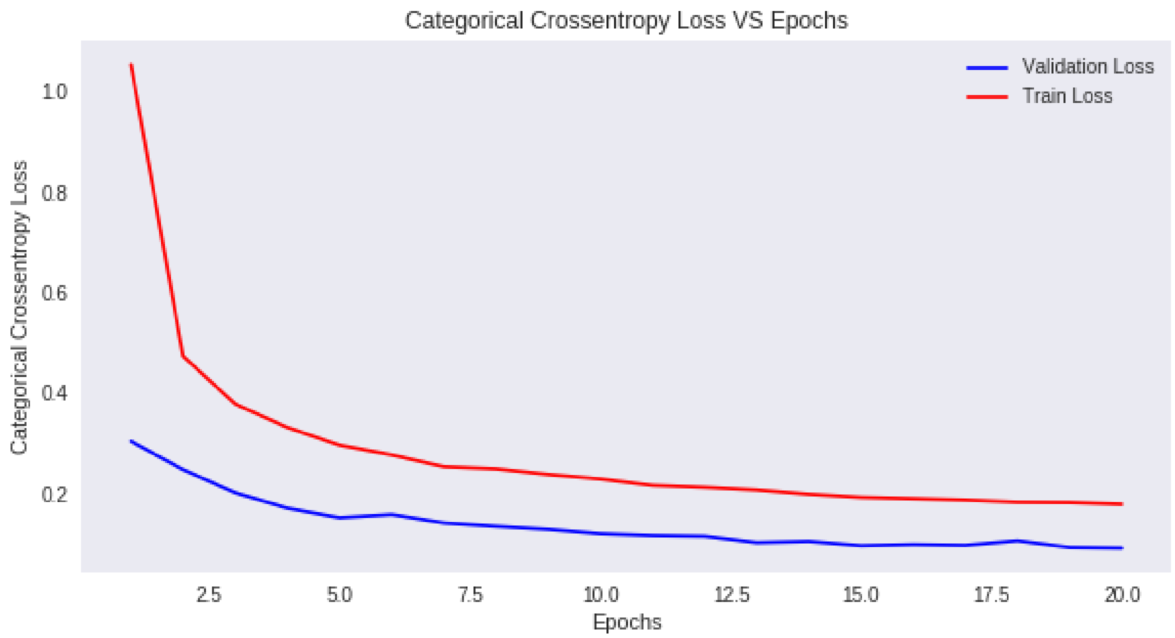
```
dense_22 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_7 (Batch (None, 256)               1024
_________________________________________________________________
dropout_7 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_23 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_8 (Batch (None, 128)               512
_________________________________________________________________
dropout_8 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_24 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_9 (Batch (None, 64)                256
_________________________________________________________________
dropout_9 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_25 (Dense)             (None, 32)                2080
_________________________________________________________________
batch_normalization_10 (Batc (None, 32)                128
_________________________________________________________________
dropout_10 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_26 (Dense)             (None, 10)                330
=================================================================
Total params: 580,778
Trainable params: 578,794
Non-trainable params: 1,984
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 15s 243us/step - loss: 1.0536
- acc: 0.6618 - val_loss: 0.3050 - val_acc: 0.9124
Epoch 2/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.4741
- acc: 0.8712 - val_loss: 0.2487 - val_acc: 0.9302
Epoch 3/20
60000/60000 [==============================] - 13s 223us/step - loss: 0.3790
- acc: 0.9002 - val_loss: 0.2027 - val_acc: 0.9430
Epoch 4/20
60000/60000 [==============================] - 13s 222us/step - loss: 0.3324
- acc: 0.9130 - val_loss: 0.1723 - val_acc: 0.9490
Epoch 5/20
60000/60000 [==============================] - 13s 221us/step - loss: 0.2972
- acc: 0.9239 - val_loss: 0.1530 - val_acc: 0.9568
Epoch 6/20
60000/60000 [==============================] - 13s 221us/step - loss: 0.2780
- acc: 0.9283 - val_loss: 0.1594 - val_acc: 0.9556
Epoch 7/20
60000/60000 [==============================] - 13s 213us/step - loss: 0.2545
- acc: 0.9347 - val_loss: 0.1426 - val_acc: 0.9602
Epoch 8/20
60000/60000 [==============================] - 13s 221us/step - loss: 0.2501
- acc: 0.9358 - val_loss: 0.1360 - val_acc: 0.9595
Epoch 9/20
60000/60000 [==============================] - 13s 218us/step - loss: 0.2390
- acc: 0.9388 - val_loss: 0.1300 - val_acc: 0.9611
Epoch 10/20
60000/60000 [==============================] - 13s 218us/step - loss: 0.2301
```

```
- acc: 0.9408 - val_loss: 0.1213 - val_acc: 0.9656
Epoch 11/20
60000/60000 [==============================] - 13s 217us/step - loss: 0.2175
- acc: 0.9436 - val_loss: 0.1177 - val_acc: 0.9677
Epoch 12/20
60000/60000 [==============================] - 13s 219us/step - loss: 0.2136
- acc: 0.9445 - val_loss: 0.1163 - val_acc: 0.9678
Epoch 13/20
60000/60000 [==============================] - 13s 209us/step - loss: 0.2083
- acc: 0.9459 - val_loss: 0.1036 - val_acc: 0.9727
Epoch 14/20
60000/60000 [==============================] - 13s 214us/step - loss: 0.1997
- acc: 0.9482 - val_loss: 0.1060 - val_acc: 0.9703
Epoch 15/20
60000/60000 [==============================] - 13s 219us/step - loss: 0.1936
- acc: 0.9494 - val_loss: 0.0975 - val_acc: 0.9743
Epoch 16/20
60000/60000 [==============================] - 13s 218us/step - loss: 0.1910
- acc: 0.9509 - val_loss: 0.0997 - val_acc: 0.9729
Epoch 17/20
60000/60000 [==============================] - 13s 218us/step - loss: 0.1883
- acc: 0.9506 - val_loss: 0.0982 - val_acc: 0.9730
Epoch 18/20
60000/60000 [==============================] - 13s 221us/step - loss: 0.1842
- acc: 0.9521 - val_loss: 0.1073 - val_acc: 0.9694
Epoch 19/20
60000/60000 [==============================] - 13s 217us/step - loss: 0.1835
- acc: 0.9526 - val_loss: 0.0940 - val_acc: 0.9739
Epoch 20/20
60000/60000 [==============================] - 13s 218us/step - loss: 0.1801
- acc: 0.9535 - val_loss: 0.0928 - val_acc: 0.9751
```

In [21]:

```
# Evaluating the model
score = model_5d.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

# Test and train accuracy of the model
model_5d_test = score[1]
model_5d_train = history_5d.history['acc']

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,nb_epoch+1))

# Validation loss
vy = history_5d.history['val_loss']
# Training loss
ty = history_5d.history['loss']

# Calling the function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.09278820200497284
Test accuracy: 0.9751
```



CONCLUSION

(a). Procedure Followed :

1. Load MNIST dataset
2. Split the dataset into train and test
3. Normalize the train and test data
4. Convert class variable into categorical data vector
5. Implement Softmax classifier with 2 , 3 and 5 hidden layers(512,256,128,64,32 respectively) .
6. Add Dropout(rate 40%) and Batch Normalization to the hidden layers .

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

(b) Table (Different models with their train and test accuracies):

In [22]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["MLP_Hidden_Layers", "MODEL", "Training Accuracy", "Test Accuracy"]

x.add_row([2, "Without Dropout and Batch Normalization",0.95,0.95])
x.add_row([2, "With Dropout and Batch Normalization",0.96,0.97])
x.add_row([3, "Without Dropout and Batch Normalization",0.95,0.95])
x.add_row([3,"With Dropout and Batch Normalization",0.95,0.95])
x.add_row([5, "Without Dropout and Batch Normalization",0.96,0.96])
x.add_row([5,"With Dropout and Batch Normalization",0.95,0.96])

print(x)
```

```
+-------------------+-----------------------------------------+-------------
------+---------------+
| MLP_Hidden_Layers |                  MODEL                  | Training Acc
uracy | Test Accuracy |
+-------------------+-----------------------------------------+-------------
------+---------------+
|         2         | Without Dropout and Batch Normalization |        0.95
|      0.95     |
|         2         |   With Dropout and Batch Normalization  |        0.96
|      0.97     |
|         3         | Without Dropout and Batch Normalization |        0.95
|      0.95     |
|         3         |   With Dropout and Batch Normalization  |        0.95
|      0.95     |
|         5         | Without Dropout and Batch Normalization |        0.96
|      0.96     |
|         5         |   With Dropout and Batch Normalization  |        0.95
|      0.96     |
+-------------------+-----------------------------------------+-------------
------+---------------+
```

In [0]:

```
```