Assignment-12(Different MLP architectures on MNIST dataset)

September 17, 2018

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

Using TensorFlow backend.


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

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

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

```
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
```

```
In [4]: # 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.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.01176471 0.07058824 0.07058824 0.07058824
 0.49411765 0.53333333 0.68627451 0.10196078 0.65098039 1.
 0.96862745 0.49803922 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.11764706 0.14117647 0.36862745 0.60392157
 0.66666667 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.88235294 0.6745098  0.99215686 0.94901961 0.76470588 0.25098039
 0.         0.         0.         0.         0.         0.
```

```
0.         0.         0.         0.         0.         0.19215686
0.93333333 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
0.99215686 0.99215686 0.99215686 0.98431373 0.36470588 0.32156863
0.32156863 0.21960784 0.15294118 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.07058824 0.85882353 0.99215686
0.99215686 0.99215686 0.99215686 0.99215686 0.77647059 0.71372549
0.96862745 0.94509804 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.31372549 0.61176471 0.41960784 0.99215686
0.99215686 0.80392157 0.04313725 0.         0.16862745 0.60392157
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.05490196 0.00392157 0.60392157 0.99215686 0.35294118
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.54509804 0.99215686 0.74509804 0.00784314 0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.04313725
0.74509804 0.99215686 0.2745098  0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.1372549  0.94509804
0.88235294 0.62745098 0.42352941 0.00392157 0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.31764706 0.94117647 0.99215686
0.99215686 0.46666667 0.09803922 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.17647059 0.72941176 0.99215686 0.99215686
0.58823529 0.10588235 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.0627451  0.36470588 0.98823529 0.99215686 0.73333333
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
```

```
0.         0.         0.         0.         0.         0.
0.         0.97647059 0.99215686 0.97647059 0.25098039 0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.18039216 0.50980392 0.71764706 0.99215686
0.99215686 0.81176471 0.00784314 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.15294118 0.58039216
0.89803922 0.99215686 0.99215686 0.99215686 0.98039216 0.71372549
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.09411765 0.44705882 0.86666667 0.99215686 0.99215686 0.99215686
0.99215686 0.78823529 0.30588235 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.09019608 0.25882353 0.83529412 0.99215686
0.99215686 0.99215686 0.99215686 0.77647059 0.31764706 0.00784314
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.07058824 0.67058824
0.85882353 0.99215686 0.99215686 0.99215686 0.99215686 0.76470588
0.31372549 0.03529412 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.21568627 0.6745098  0.88627451 0.99215686 0.99215686 0.99215686
0.99215686 0.95686275 0.52156863 0.04313725 0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.53333333 0.99215686
0.99215686 0.99215686 0.83137255 0.52941176 0.51764706 0.0627451
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
```

```
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.         ]
```

```
In [5]: # 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()
```

1.1 (1). Softmax Classifier with 2 hidden layers

1.1.1 (1.a) Without dropout and Batch normalization

```
In [7]: 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
```

```python
model_2 = Sequential()

# Adding first hidden layer
model_2.add(Dense(364, activation='relu', input_shape=(input_dim,), kernel_initializer=

# Adding second hidden layer
model_2.add(Dense(52, 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, verbo
```

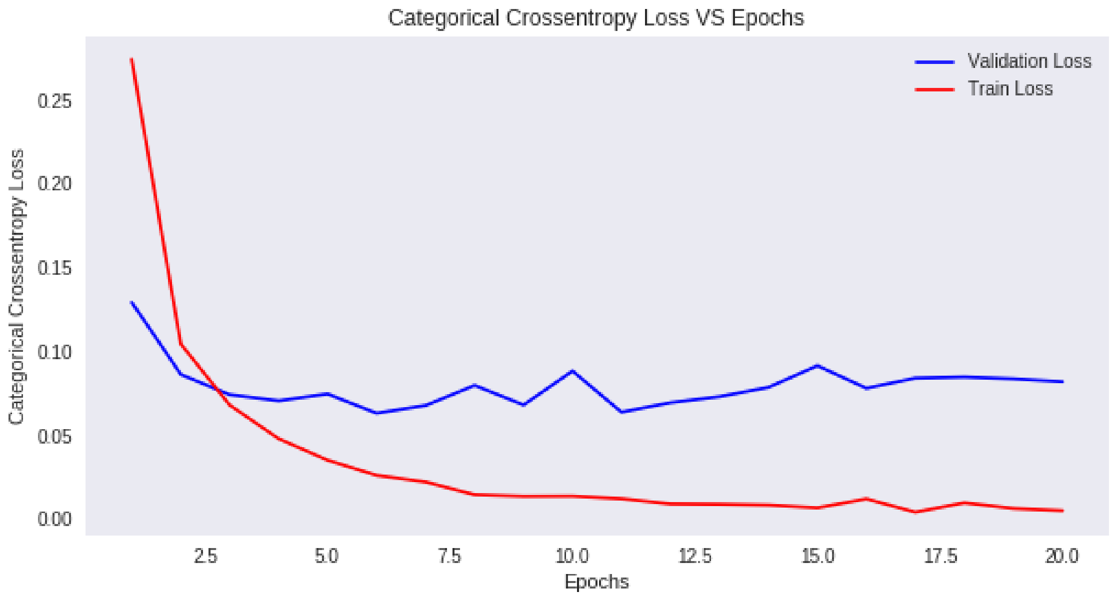
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 364)               285740
_________________________________________________________________
dense_2 (Dense)              (None, 52)                18980
_________________________________________________________________
dense_3 (Dense)              (None, 10)                530
=================================================================
Total params: 305,250
Trainable params: 305,250
Non-trainable params: 0
_________________________________________________________________
Model Summary :-
 None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 6s 104us/step - loss: 0.2745 - acc: 0.9217 - va
Epoch 2/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.1046 - acc: 0.9684 - val_
Epoch 3/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0681 - acc: 0.9792 - val_
Epoch 4/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0481 - acc: 0.9852 - val_
Epoch 5/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0352 - acc: 0.9892 - val_
Epoch 6/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0262 - acc: 0.9921 - val_
```

```
Epoch 7/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0222 - acc: 0.9929 - val_
Epoch 8/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0146 - acc: 0.9958 - val_
Epoch 9/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0137 - acc: 0.9961 - val_
Epoch 10/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0138 - acc: 0.9957 - val_
Epoch 11/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0122 - acc: 0.9962 - val_
Epoch 12/20
60000/60000 [==============================] - 6s 97us/step - loss: 0.0091 - acc: 0.9970 - val_
Epoch 13/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0089 - acc: 0.9970 - val_
Epoch 14/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0084 - acc: 0.9973 - val_
Epoch 15/20
60000/60000 [==============================] - 6s 96us/step - loss: 0.0068 - acc: 0.9979 - val_
Epoch 16/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.0121 - acc: 0.9960 - val_
Epoch 17/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.0042 - acc: 0.9989 - val_
Epoch 18/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0098 - acc: 0.9970 - val_
Epoch 19/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0064 - acc: 0.9980 - va
Epoch 20/20
60000/60000 [==============================] - 6s 103us/step - loss: 0.0051 - acc: 0.9984 - va
```

In [8]:
```python
# 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: 0.08213182862031254
Test accuracy: 0.9815
```



1.1.2 (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(364, activation='relu', input_shape=(input_dim,), kernel_initialize
         # Adding Batch Normalization
         model_2d.add(BatchNormalization())
         # Adding dropout to first hidden layer
         model_2d.add(Dropout(0.5))

         # Adding second hidden layer
         model_2d.add(Dense(52, activation='relu', kernel_initializer=he_normal(seed=None)))
         # Adding Batch Normalization
         model_2d.add(BatchNormalization())
         # Adding dropout to second hidden layer
```

```python
model_2d.add(Dropout(0.5))

# 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, ve
```

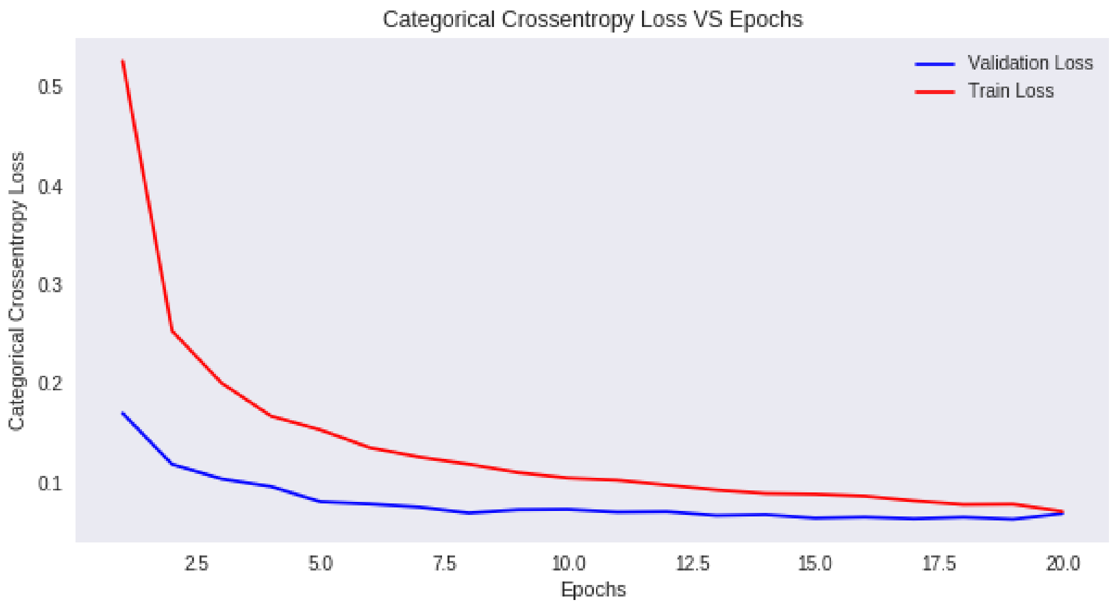
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 364)               285740
_________________________________________________________________
batch_normalization_1 (Batch (None, 364)               1456
_________________________________________________________________
dropout_1 (Dropout)          (None, 364)               0
_________________________________________________________________
dense_6 (Dense)              (None, 52)                18980
_________________________________________________________________
batch_normalization_2 (Batch (None, 52)                208
_________________________________________________________________
dropout_2 (Dropout)          (None, 52)                0
_________________________________________________________________
dense_7 (Dense)              (None, 10)                530
=================================================================
Total params: 306,914
Trainable params: 306,082
Non-trainable params: 832
_________________________________________________________________
Model Summary :-
 None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.5260 - acc: 0.8436 - val
Epoch 2/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.2536 - acc: 0.9274 - val
Epoch 3/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.2009 - acc: 0.9425 - val
Epoch 4/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.1676 - acc: 0.9521 - val
Epoch 5/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.1536 - acc: 0.9557 - val
```

```
Epoch 6/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.1357 - acc: 0.9603 - val
Epoch 7/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.1261 - acc: 0.9636 - val
Epoch 8/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.1189 - acc: 0.9648 - val
Epoch 9/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.1106 - acc: 0.9676 - val
Epoch 10/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.1050 - acc: 0.9693 - val
Epoch 11/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.1029 - acc: 0.9691 - val
Epoch 12/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0979 - acc: 0.9702 - val
Epoch 13/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0930 - acc: 0.9724 - val
Epoch 14/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.0894 - acc: 0.9736 - val
Epoch 15/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0887 - acc: 0.9735 - val
Epoch 16/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.0867 - acc: 0.9738 - val
Epoch 17/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0820 - acc: 0.9749 - val
Epoch 18/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0783 - acc: 0.9762 - val
Epoch 19/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0787 - acc: 0.9764 - val
Epoch 20/20
60000/60000 [==============================] - 7s 118us/step - loss: 0.0714 - acc: 0.9781 - val
```

In [11]:
```
# 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: 0.06914302359962603
Test accuracy: 0.9817
```



1.2 (2). Softmax Classifier with 3 hidden layers

1.2.1 (2.a) Without Dropout and Batch Normalization

In [12]:
```
# Initialising model
model_3 = Sequential()

# Adding first hidden layer
model_3.add(Dense(392, activation='relu', input_shape=(input_dim,), kernel_initialize

# Adding second hidden layer
model_3.add(Dense(196, activation='relu', kernel_initializer=he_normal(seed=None)))

# Adding third hidden layer
model_3.add(Dense(98, activation='relu', kernel_initializer=he_normal(seed=None)))

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

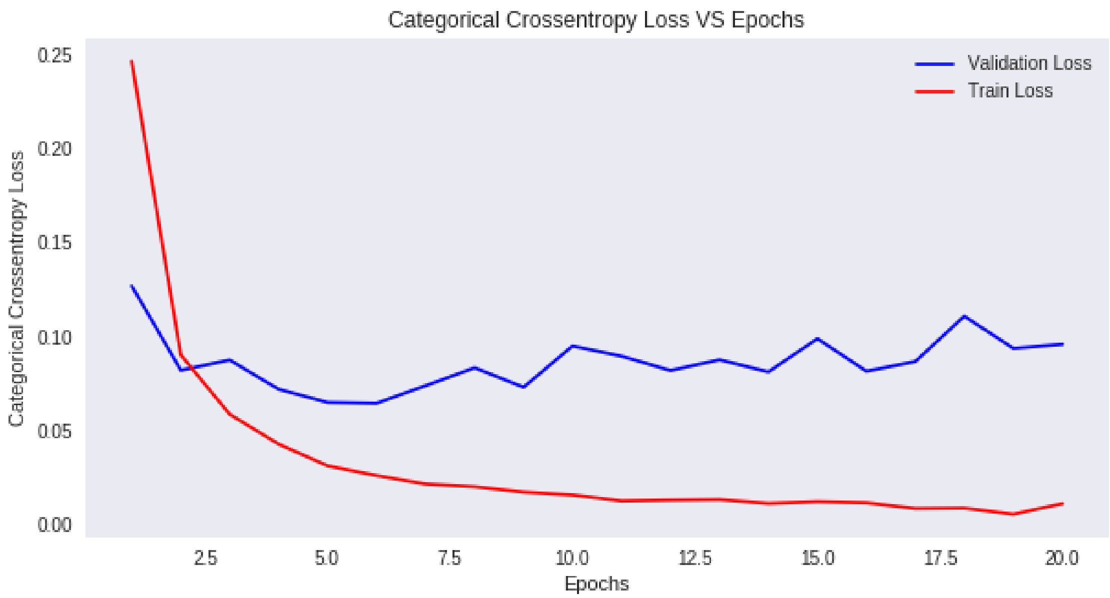
```
# Printing model Summary
print(model_3.summary())

# 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, verl
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 392)               307720
_________________________________________________________________
dense_9 (Dense)              (None, 196)               77028
_________________________________________________________________
dense_10 (Dense)             (None, 98)                19306
_________________________________________________________________
dense_11 (Dense)             (None, 10)                990
=================================================================
Total params: 405,044
Trainable params: 405,044
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.2469 - acc: 0.9283 - val
Epoch 2/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0902 - acc: 0.9726 - val
Epoch 3/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.0585 - acc: 0.9817 - val
Epoch 4/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0426 - acc: 0.9867 - val
Epoch 5/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.0310 - acc: 0.9899 - val
Epoch 6/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.0258 - acc: 0.9912 - val
Epoch 7/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0213 - acc: 0.9929 - val
Epoch 8/20
60000/60000 [==============================] - 7s 116us/step - loss: 0.0198 - acc: 0.9935 - val
Epoch 9/20
60000/60000 [==============================] - 7s 117us/step - loss: 0.0170 - acc: 0.9941 - val
Epoch 10/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0154 - acc: 0.9949 - val
Epoch 11/20
```

```
60000/60000 [==============================] - 7s 115us/step - loss: 0.0123 - acc: 0.9960 - val
Epoch 12/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0127 - acc: 0.9956 - val
Epoch 13/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0130 - acc: 0.9960 - val
Epoch 14/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.0109 - acc: 0.9963 - val
Epoch 15/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.0119 - acc: 0.9962 - val
Epoch 16/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0113 - acc: 0.9964 - val
Epoch 17/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0082 - acc: 0.9974 - val
Epoch 18/20
60000/60000 [==============================] - 7s 115us/step - loss: 0.0084 - acc: 0.9973 - val
Epoch 19/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.0052 - acc: 0.9983 - val
Epoch 20/20
60000/60000 [==============================] - 7s 114us/step - loss: 0.0108 - acc: 0.9967 - val
```

In [13]:
```
# 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: 0.09583447469817774
Test accuracy: 0.9782
```



1.2.2 (2.b) With Droput and Batch Normalization

In [14]:
```
model_3d = Sequential()

# Adding first hidden layer
model_3d.add(Dense(392, activation='relu', input_shape=(input_dim,), kernel_initialize
# Adding Batch Normalization
model_3d.add(BatchNormalization())
# Adding dropout
model_3d.add(Dropout(0.5))

# Adding second hidden layer
model_3d.add(Dense(196, activation='relu', kernel_initializer=he_normal(seed=None)))
# Adding Batch Normalization
model_3d.add(BatchNormalization())
# Adding dropout
model_3d.add(Dropout(0.5))

# Adding third hidden layer
model_3d.add(Dense(98, activation='relu', kernel_initializer=he_normal(seed=None)))
# Adding Batch Normalization
model_3d.add(BatchNormalization())
# Adding dropout
model_3d.add(Dropout(0.5))

# Adding output layer
```

```python
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, ve
```

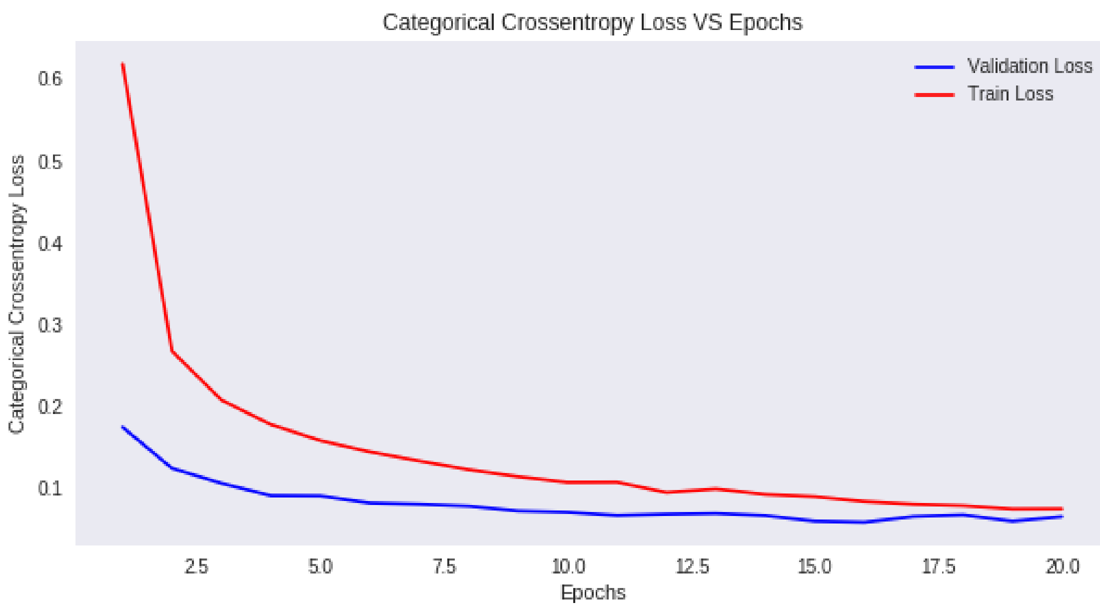
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 392)               307720
_________________________________________________________________
batch_normalization_3 (Batch (None, 392)               1568
_________________________________________________________________
dropout_3 (Dropout)          (None, 392)               0
_________________________________________________________________
dense_13 (Dense)             (None, 196)               77028
_________________________________________________________________
batch_normalization_4 (Batch (None, 196)               784
_________________________________________________________________
dropout_4 (Dropout)          (None, 196)               0
_________________________________________________________________
dense_14 (Dense)             (None, 98)                19306
_________________________________________________________________
batch_normalization_5 (Batch (None, 98)                392
_________________________________________________________________
dropout_5 (Dropout)          (None, 98)                0
_________________________________________________________________
dense_15 (Dense)             (None, 10)                990
=================================================================
Total params: 407,788
Trainable params: 406,416
Non-trainable params: 1,372
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 180us/step - loss: 0.6184 - acc: 0.8117 - va
Epoch 2/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.2680 - acc: 0.9222 - va
Epoch 3/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.2083 - acc: 0.9392 - va
Epoch 4/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.1784 - acc: 0.9484 - va
```

```
Epoch 5/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.1588 - acc: 0.9538 - va
Epoch 6/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.1451 - acc: 0.9581 - va
Epoch 7/20
60000/60000 [==============================] - 10s 159us/step - loss: 0.1340 - acc: 0.9605 - va
Epoch 8/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.1233 - acc: 0.9636 - va
Epoch 9/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.1148 - acc: 0.9662 - va
Epoch 10/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.1078 - acc: 0.9685 - va
Epoch 11/20
60000/60000 [==============================] - 10s 166us/step - loss: 0.1080 - acc: 0.9684 - va
Epoch 12/20
60000/60000 [==============================] - 10s 166us/step - loss: 0.0955 - acc: 0.9715 - va
Epoch 13/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.0998 - acc: 0.9701 - va
Epoch 14/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.0931 - acc: 0.9718 - va
Epoch 15/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.0904 - acc: 0.9733 - va
Epoch 16/20
60000/60000 [==============================] - 10s 159us/step - loss: 0.0846 - acc: 0.9754 - va
Epoch 17/20
60000/60000 [==============================] - 10s 159us/step - loss: 0.0811 - acc: 0.9750 - va
Epoch 18/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.0794 - acc: 0.9762 - va
Epoch 19/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.0753 - acc: 0.9776 - va
Epoch 20/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.0755 - acc: 0.9778 - va
```

In [15]:
```
# 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: 0.06592972582291114
Test accuracy: 0.9822
```



1.3 (3). Softmax Classifier with 5 hidden layers

(3.a) Without Dropout and Batch Normalization

```
In [16]: # Initialising model
         model_5 = Sequential()

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

         # 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)))
```

```python
# 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, verl
```

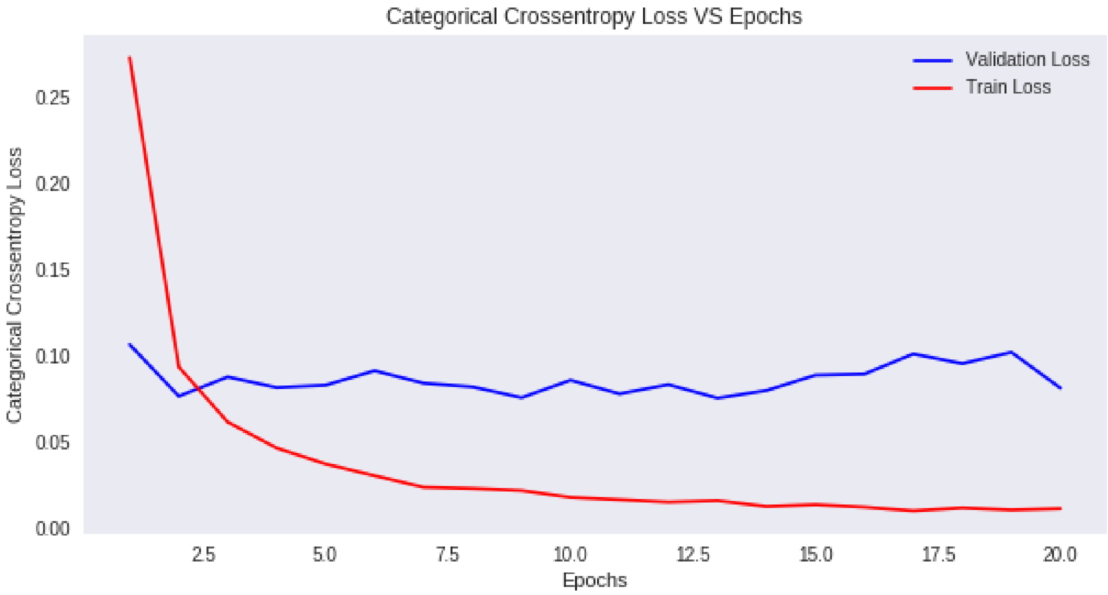
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_16 (Dense)             (None, 512)               401920
_________________________________________________________________
dense_17 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_18 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_19 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_20 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_21 (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 [==============================] - 10s 169us/step - loss: 0.2726 - acc: 0.9163 - va
Epoch 2/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0936 - acc: 0.9719 - val
Epoch 3/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0616 - acc: 0.9810 - val
Epoch 4/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.0466 - acc: 0.9852 - val
Epoch 5/20
```

```
60000/60000 [==============================] - 9s 158us/step - loss: 0.0374 - acc: 0.9879 - val
Epoch 6/20
60000/60000 [==============================] - 10s 169us/step - loss: 0.0306 - acc: 0.9901 - va
Epoch 7/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.0239 - acc: 0.9921 - va
Epoch 8/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.0232 - acc: 0.9927 - val
Epoch 9/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.0220 - acc: 0.9931 - val
Epoch 10/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.0181 - acc: 0.9944 - val
Epoch 11/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.0167 - acc: 0.9950 - val
Epoch 12/20
60000/60000 [==============================] - 10s 159us/step - loss: 0.0153 - acc: 0.9954 - va
Epoch 13/20
60000/60000 [==============================] - 9s 158us/step - loss: 0.0161 - acc: 0.9946 - val
Epoch 14/20
60000/60000 [==============================] - 9s 158us/step - loss: 0.0128 - acc: 0.9960 - val
Epoch 15/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.0138 - acc: 0.9957 - val
Epoch 16/20
60000/60000 [==============================] - 9s 155us/step - loss: 0.0124 - acc: 0.9961 - val
Epoch 17/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.0103 - acc: 0.9966 - val
Epoch 18/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0120 - acc: 0.9963 - val
Epoch 19/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.0108 - acc: 0.9970 - val
Epoch 20/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0116 - acc: 0.9968 - val
```

In [17]:
```
# 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: 0.08139225203025052
Test accuracy: 0.9827
```



1.3.1 (3.b) With Dropout and Batch Normalisation

```
In [18]: # Initialising model
model_5d = Sequential()

# Adding first hidden layer
model_5d.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initialize
# Adding Batch Normalization
model_5d.add(BatchNormalization())
# Adding dropout
model_5d.add(Dropout(0.5))

# 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(0.5))

# 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(0.5))

# 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(0.5))

# 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(0.5))

# 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, ve
```

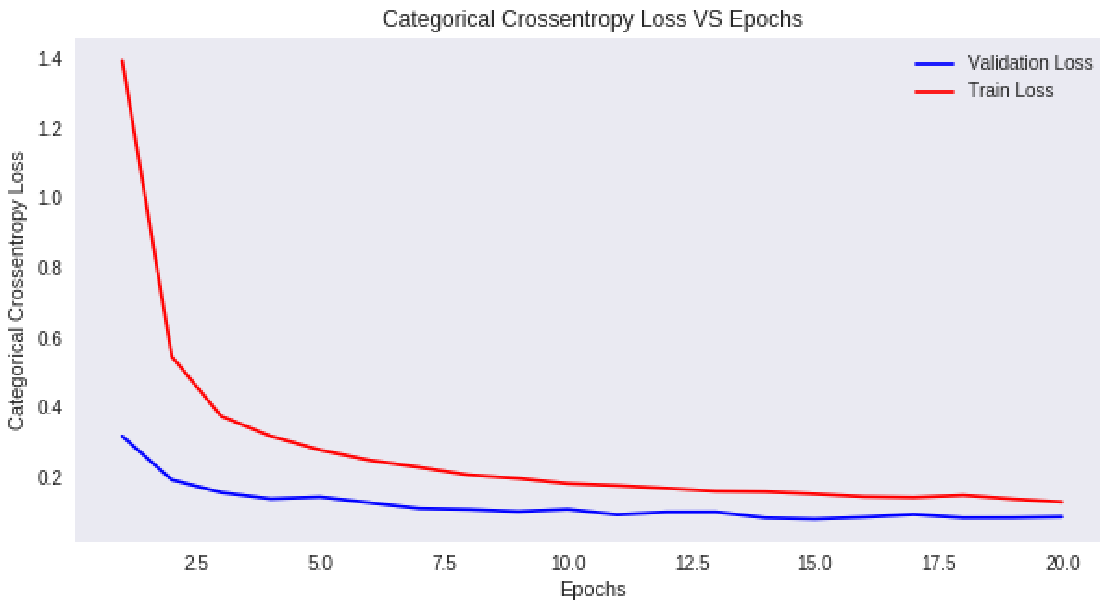
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_22 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_6 (Batch (None, 512)               2048
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_23 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_7 (Batch (None, 256)               1024
_________________________________________________________________
```

```
dropout_7 (Dropout)          (None, 256)              0
_________________________________________________________________
dense_24 (Dense)             (None, 128)              32896
_________________________________________________________________
batch_normalization_8 (Batch (None, 128)              512
_________________________________________________________________
dropout_8 (Dropout)          (None, 128)              0
_________________________________________________________________
dense_25 (Dense)             (None, 64)               8256
_________________________________________________________________
batch_normalization_9 (Batch (None, 64)               256
_________________________________________________________________
dropout_9 (Dropout)          (None, 64)               0
_________________________________________________________________
dense_26 (Dense)             (None, 32)               2080
_________________________________________________________________
batch_normalization_10 (Batc (None, 32)               128
_________________________________________________________________
dropout_10 (Dropout)         (None, 32)               0
_________________________________________________________________
dense_27 (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 253us/step - loss: 1.3940 - acc: 0.5474 - va
Epoch 2/20
60000/60000 [==============================] - 13s 213us/step - loss: 0.5452 - acc: 0.8511 - va
Epoch 3/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.3737 - acc: 0.9058 - va
Epoch 4/20
60000/60000 [==============================] - 13s 217us/step - loss: 0.3167 - acc: 0.9237 - va
Epoch 5/20
60000/60000 [==============================] - 13s 220us/step - loss: 0.2769 - acc: 0.9335 - va
Epoch 6/20
60000/60000 [==============================] - 13s 213us/step - loss: 0.2476 - acc: 0.9423 - va
Epoch 7/20
60000/60000 [==============================] - 13s 214us/step - loss: 0.2276 - acc: 0.9461 - va
Epoch 8/20
60000/60000 [==============================] - 13s 211us/step - loss: 0.2056 - acc: 0.9533 - va
Epoch 9/20
60000/60000 [==============================] - 13s 213us/step - loss: 0.1955 - acc: 0.9558 - va
Epoch 10/20
60000/60000 [==============================] - 13s 214us/step - loss: 0.1809 - acc: 0.9584 - va
```

```
Epoch 11/20
60000/60000 [==============================] - 13s 213us/step - loss: 0.1750 - acc: 0.9603 - va
Epoch 12/20
60000/60000 [==============================] - 13s 213us/step - loss: 0.1672 - acc: 0.9622 - va
Epoch 13/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.1590 - acc: 0.9644 - va
Epoch 14/20
60000/60000 [==============================] - 13s 213us/step - loss: 0.1573 - acc: 0.9649 - va
Epoch 15/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.1511 - acc: 0.9660 - va
Epoch 16/20
60000/60000 [==============================] - 13s 209us/step - loss: 0.1432 - acc: 0.9679 - va
Epoch 17/20
60000/60000 [==============================] - 13s 210us/step - loss: 0.1411 - acc: 0.9688 - va
Epoch 18/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.1472 - acc: 0.9671 - va
Epoch 19/20
60000/60000 [==============================] - 13s 211us/step - loss: 0.1365 - acc: 0.9697 - va
Epoch 20/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.1277 - acc: 0.9713 - va
```

In [19]:
```python
# 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: 0.08531591616126243
Test accuracy: 0.981
```



1.4 CONCLUSION

1.5 (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 .
6. Add Dropout and Batch Normalization to the hidden layers .
7. Draw Categorical Crossentropy Loss VS No.of Epochs plot .

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

In [22]:
```
# Installing the library prettytable
!pip install prettytable

# Creating table using PrettyTable library
from prettytable import PrettyTable

# Names of models
names = ['MLP(2-hidden layers) Without Dropout and Batch Normalization','MLP(2-hidden
         'MLP(3-hidden layers) Without Dropout and Batch Normalization','MLP(3-hidden
         'MLP(5-hidden layers) Without Dropout and Batch Normalization','MLP(5-hidden

# Training accuracies
train_acc = [model_2_train[19],model_2d_train[19],model_3_train[19],model_3d_train[19]
```

```python
# Test accuracies
test_acc = [model_2_test,model_2d_test,model_3_test,model_3d_test,model_5_test,model_5

numbering = [1,2,3,4,5,6]

# Initializing prettytable
ptable = PrettyTable()

# Adding columns
ptable.add_column("S.NO.",numbering)
ptable.add_column("MODEL",names)
ptable.add_column("Training Accuracy",train_acc)
ptable.add_column("Test Accuracy",test_acc)

# Printing the Table
print(ptable)
```

```
Requirement already satisfied: prettytable in /usr/local/lib/python3.6/dist-packages (0.7.2)
+-------+--------------------------------------------------------------+--------------------+--
| S.NO. |                            MODEL                             | Training Accuracy  | T
+-------+--------------------------------------------------------------+--------------------+--
|   1   | MLP(2-hidden layers) Without Dropout and Batch Normalization | 0.9983666666666666 |
|   2   |   MLP(2-hidden layers) With Dropout and Batch Normalization   | 0.9780666666348775 |
|   3   | MLP(3-hidden layers) Without Dropout and Batch Normalization | 0.9966666666666667 |
|   4   |   MLP(3-hidden layers) With Dropout and Batch Normalization   | 0.9777999999682109 |
|   5   | MLP(5-hidden layers) Without Dropout and Batch Normalization | 0.9967666666666667 |
|   6   |   MLP(5-hidden layers) With Dropout and Batch Normalization   | 0.9712833333651225 |
+-------+--------------------------------------------------------------+--------------------+--
```