Exercise: Try different MLP architectures on MNIST dataset.

Step By Step Process

- Lets start working on this Assignents and as we know our main objective in assignment to implemants Multi Layer Perceptron using Keras on MNIST Dataset And in this we will work with Multi Layer Perceptron and also MLP with Dropout and Batch Normalization.
- First we will start with importing multiple important librarues with is being used in this assignments and after that we will load our MNIST Dataset.
- After loading MNIST Dataset with 100000 data points and then we will split our dataset into train and test and we know our dataset i.e image is in the shape of (28 X 28) so we will first reshape our dataset i.e we will convert the (28*28) vector into single dimensional vector of 1* 784 .
- Now in this step we will do data point normlization and as we observe the our data in which each cell is having a value between 0-255 so before we move to apply machine learning algorithms lets try to normalize the data.
- And as we know our class lables in this dataset is in numbers between {0,1,2,3......9} and now we will convert it into one hot encoded vector
- After doing all above now we will start working with our models in this we will work with softmax classifier with multiple hidden layers that means we will works with softmax with multiple hidden layers and try to observe the performance by changing the no of layers and in this we will also work with dropout and batch normalization.
- And after doing all this we will try to observe the performance of train and test val so that we will be able to know our model should not overfit.

In [1]:

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

# importing other lib
%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
import time
```

```
C:\Users\nisha\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of the s
econd argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be
treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

In [14]:

```
def plt_dynamic_model(x, vy, ty):
    plt.figure(figsize=(10,5))
    plt.plot(x, vy, 'b', label="val Loss")
    plt.plot(x, ty, 'r', label="Train Loss")
    plt.xlabel('Epochs val')
    plt.ylabel('Categorical Crossentropy Loss')
    plt.title('\nCategorical Crossentropy Loss VS Epochs')
    plt.legend()
    plt.show()
```

Loading MNIST Dataset

- And then spliting it into train and test

In [2]:

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

```
%d)"%(X_test.shape[1],X_test.shape[2]))
```

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

In [3]:

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

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

In [4]:

```
# 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]
```

Data point normlization

In [5]:

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

X_train = X_train/255
X_test = X_test/255

# 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.          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.        ]
```

Converting class lables into One hot Encoded

- As we know our class lables in this dataset is in numbers between {0,1,2,3......9} and now we will convert it into one hot encoded vector

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 [31]:

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

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

1. Softmax classifier with 2-hidden layers Without dropout and Batch Normalization

In [8]:

```
model2 = Sequential()

model2.add(Dense(385, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(see
d=None)))
model2.add(Dense(120, activation='relu', kernel_initializer=he_normal(seed=None)))
model2.add(Dense(output_dim, activation='softmax'))

# model Summary
print("Model Summary :- \n",model2.summary())

# Compiling the model
model2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Fitting the data to the model
history2 = model2.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validati
on_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 385)               302225    
_________________________________________________________________
dense_5 (Dense)              (None, 120)               46320     
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1210      
=================================================================
Total params: 349,755
Trainable params: 349,755
Non-trainable params: 0
```

```
Model Summary :-
 None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 142us/step - loss: 0.2441 - acc: 0.9290 -
val_loss: 0.1123 - val_acc: 0.9653
Epoch 2/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0918 - acc: 0.9718 -
val_loss: 0.0919 - val_acc: 0.9717
Epoch 3/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0573 - acc: 0.9827 -
val_loss: 0.0746 - val_acc: 0.9765
Epoch 4/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.0389 - acc: 0.9877 -
val_loss: 0.0668 - val_acc: 0.9798
Epoch 5/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.0304 - acc: 0.9905 -
val_loss: 0.0685 - val_acc: 0.9799
Epoch 6/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0226 - acc: 0.9932 -
val_loss: 0.0684 - val_acc: 0.9810
Epoch 7/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0185 - acc: 0.9941 -
val_loss: 0.0821 - val_acc: 0.9773
Epoch 8/20
60000/60000 [==============================] - 9s 142us/step - loss: 0.0157 - acc: 0.9948 -
val_loss: 0.0779 - val_acc: 0.9794
Epoch 9/20
60000/60000 [==============================] - 8s 129us/step - loss: 0.0162 - acc: 0.9945 -
val_loss: 0.0768 - val_acc: 0.9794
Epoch 10/20
60000/60000 [==============================] - 8s 128us/step - loss: 0.0103 - acc: 0.9966 -
val_loss: 0.0705 - val_acc: 0.9818
Epoch 11/20
60000/60000 [==============================] - 8s 127us/step - loss: 0.0081 - acc: 0.9972 -
val_loss: 0.0818 - val_acc: 0.9794
Epoch 12/20
60000/60000 [==============================] - 8s 126us/step - loss: 0.0145 - acc: 0.9951 -
val_loss: 0.0845 - val_acc: 0.9788
Epoch 13/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.0088 - acc: 0.9971 -
val_loss: 0.0924 - val_acc: 0.9790
Epoch 14/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0090 - acc: 0.9970 -
val_loss: 0.0959 - val_acc: 0.9797
Epoch 15/20
60000/60000 [==============================] - 9s 145us/step - loss: 0.0072 - acc: 0.9976 -
val_loss: 0.0945 - val_acc: 0.9804
Epoch 16/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0056 - acc: 0.9982 -
val_loss: 0.0855 - val_acc: 0.9816
Epoch 17/20
60000/60000 [==============================] - 8s 132us/step - loss: 0.0093 - acc: 0.9969 -
val_loss: 0.1034 - val_acc: 0.9781s - l
Epoch 18/20
60000/60000 [==============================] - 8s 125us/step - loss: 0.0095 - acc: 0.9967 -
val_loss: 0.0933 - val_acc: 0.9796
Epoch 19/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0050 - acc: 0.9984 -
val_loss: 0.0809 - val_acc: 0.9809
Epoch 20/20
60000/60000 [==============================] - 7s 124us/step - loss: 0.0104 - acc: 0.9967 -
val_loss: 0.0834 - val_acc: 0.9820
```

In [22]:

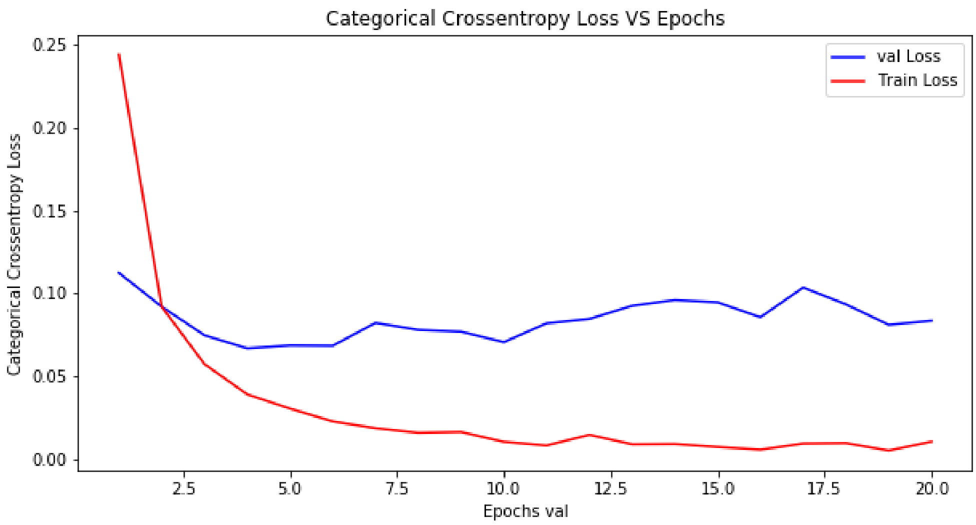
```
x = list(range(1,nb_epoch+1))

# getting Val loss
vy = history2.history['val_loss']
# getting Train loss
ty = history2.history['loss']

# function call
plt_dynamic_model(x, vy, ty)
```

```
# Evaluating the model
model_score = model2.evaluate(X_test, Y_test, verbose=0)
print('Test score:', model_score[0])
print('Test accuracy:', model_score[1])

# saving train and test accuracy of the model
model2_test_acc = model_score[1]
model2_train_acc = history2.history['acc']
```




```
Test score: 0.08338531920399564
Test accuracy: 0.982
```

Softmax classifier with 2-hidden layers With dropout and Batch Normalization

In [9]:

```
model2dn = Sequential()

model2dn.add(Dense(375, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(s
eed=None)))
model2dn.add(BatchNormalization())
model2dn.add(Dropout(0.5))

model2dn.add(Dense(112, activation='relu', kernel_initializer=he_normal(seed=None)))
model2dn.add(BatchNormalization())
model2dn.add(Dropout(0.5))

model2dn.add(Dense(output_dim, activation='softmax'))

# model Summary
print("model summary :- \n",model2dn.summary())

# Compiling the model
model2dn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Fitting the data to the model
history2dn = model2dn.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, vali
dation_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_7 (Dense)              (None, 375)               294375    
_________________________________________________________________
batch_normalization_1 (Batch (None, 375)               1500      
_________________________________________________________________
dropout_1 (Dropout)          (None, 375)               0         
_________________________________________________________________
dense_8 (Dense)              (None, 112)               42112     
_________________________________________________________________
batch_normalization_2 (Batch (None, 112)               448       
_________________________________________________________________
```

```
dropout_2 (Dropout)          (None, 112)              0
_________________________________________________________________
dense_9 (Dense)              (None, 10)               1130
=================================================================
Total params: 339,565
Trainable params: 338,591
Non-trainable params: 974
_________________________________________________________________
model summary :-
 None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 179us/step - loss: 0.4790 - acc: 0.8549 - val_l
oss: 0.1542 - val_acc: 0.9516
Epoch 2/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.2275 - acc: 0.9320 -
val_loss: 0.1145 - val_acc: 0.9659
Epoch 3/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.1773 - acc: 0.9467 -
val_loss: 0.0980 - val_acc: 0.9693
Epoch 4/20
60000/60000 [==============================] - 10s 168us/step - loss: 0.1521 - acc: 0.9540 - val_l
oss: 0.0871 - val_acc: 0.9734
Epoch 5/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.1332 - acc: 0.9601 -
val_loss: 0.0823 - val_acc: 0.9754
Epoch 6/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.1187 - acc: 0.9638 -
val_loss: 0.0872 - val_acc: 0.9731
Epoch 7/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.1115 - acc: 0.9651 -
val_loss: 0.0727 - val_acc: 0.9767
Epoch 8/20
60000/60000 [==============================] - 10s 158us/step - loss: 0.1049 - acc: 0.9677 - val_l
oss: 0.0706 - val_acc: 0.9785
Epoch 9/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.0966 - acc: 0.9700 -
val_loss: 0.0667 - val_acc: 0.9791
Epoch 10/20
60000/60000 [==============================] - 10s 160us/step - loss: 0.0910 - acc: 0.9715 - val_l
oss: 0.0655 - val_acc: 0.9790
Epoch 11/20
60000/60000 [==============================] - 10s 163us/step - loss: 0.0853 - acc: 0.9740 - val_l
oss: 0.0680 - val_acc: 0.9796
Epoch 12/20
60000/60000 [==============================] - 11s 175us/step - loss: 0.0820 - acc: 0.9738 - val_l
oss: 0.0644 - val_acc: 0.9804
Epoch 13/20
60000/60000 [==============================] - 10s 165us/step - loss: 0.0805 - acc: 0.9754 - val_l
oss: 0.0633 - val_acc: 0.9806
Epoch 14/20
60000/60000 [==============================] - 9s 157us/step - loss: 0.0779 - acc: 0.9760 -
val_loss: 0.0681 - val_acc: 0.9801
Epoch 15/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0755 - acc: 0.9765 -
val_loss: 0.0674 - val_acc: 0.9806
Epoch 16/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0697 - acc: 0.9779 -
val_loss: 0.0620 - val_acc: 0.9821
Epoch 17/20
60000/60000 [==============================] - 10s 165us/step - loss: 0.0688 - acc: 0.9788 - val_l
oss: 0.0615 - val_acc: 0.9820
Epoch 18/20
60000/60000 [==============================] - 10s 165us/step - loss: 0.0643 - acc: 0.9794 - val_l
oss: 0.0630 - val_acc: 0.9808
Epoch 19/20
60000/60000 [==============================] - 10s 165us/step - loss: 0.0655 - acc: 0.9791 - val_l
oss: 0.0651 - val_acc: 0.9800
Epoch 20/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.0615 - acc: 0.9808 -
val_loss: 0.0583 - val_acc: 0.9818
```

In [23]:

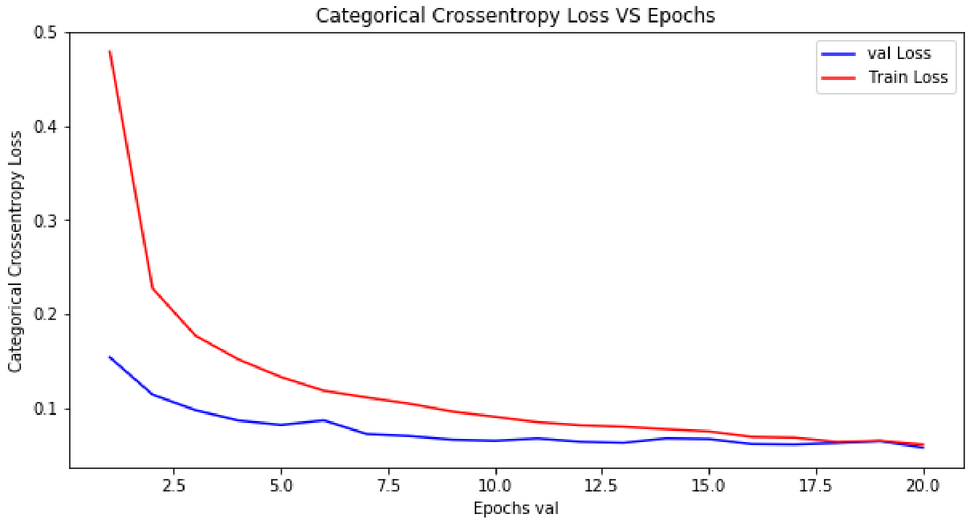
```
x = list(range(1,nb_epoch+1))
```

```
# getting Val loss
vy = history2dn.history['val_loss']
# getting Train loss
ty = history2dn.history['loss']

# function call
plt_dynamic_model(x, vy, ty)

# Evaluating the model
model_score = model2dn.evaluate(X_test, Y_test, verbose=0)
print('Test score:', model_score[0])
print('Test accuracy:', model_score[1])

# saving train and test accuracy of the model
model2dn_test_acc = model_score[1]
model2dn_train_acc = history2dn.history['acc']
```




```
Test score: 0.05833159902372281
Test accuracy: 0.9818
```

2. Softmax classifier with 3-hidden layers Without dropout and Batch Normalization

In [21]:

```
model3 = Sequential()

model3.add(Dense(465, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(see
d=None)))
model3.add(Dense(141, activation='relu', kernel_initializer=he_normal(seed=None)))
model3.add(Dense(65, activation='relu', kernel_initializer=he_normal(seed=None)))

model3.add(Dense(output_dim, activation='softmax'))

print(model3.summary())
model3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history3 = model3.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validati
on_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_26 (Dense)             (None, 465)               365025
_________________________________________________________________
dense_27 (Dense)             (None, 141)               65706
_________________________________________________________________
dense_28 (Dense)             (None, 65)                9230
_________________________________________________________________
dense_29 (Dense)             (None, 10)                660
=================================================================
Total params: 440,621
Trainable params: 440,621
```

```
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 177us/step - loss: 0.2408 - acc: 0.9295 - val_l
oss: 0.1255 - val_acc: 0.9601
Epoch 2/20
60000/60000 [==============================] - 9s 154us/step - loss: 0.0889 - acc: 0.9729 -
val_loss: 0.0912 - val_acc: 0.9725
Epoch 3/20
60000/60000 [==============================] - 10s 168us/step - loss: 0.0570 - acc: 0.9820 - val_l
oss: 0.0684 - val_acc: 0.9787
Epoch 4/20
60000/60000 [==============================] - 10s 167us/step - loss: 0.0401 - acc: 0.9867 - val_l
oss: 0.0881 - val_acc: 0.9742
Epoch 5/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.0287 - acc: 0.9905 - val_l
oss: 0.0784 - val_acc: 0.9781
Epoch 6/20
60000/60000 [==============================] - 9s 153us/step - loss: 0.0243 - acc: 0.9924 -
val_loss: 0.0783 - val_acc: 0.9783
Epoch 7/20
60000/60000 [==============================] - 10s 173us/step - loss: 0.0206 - acc: 0.9931 - val_l
oss: 0.0766 - val_acc: 0.9798
Epoch 8/20
60000/60000 [==============================] - 9s 152us/step - loss: 0.0168 - acc: 0.9946 -
val_loss: 0.0844 - val_acc: 0.9791
Epoch 9/20
60000/60000 [==============================] - 10s 159us/step - loss: 0.0191 - acc: 0.9937 - val_l
oss: 0.0804 - val_acc: 0.9788
Epoch 10/20
60000/60000 [==============================] - 10s 165us/step - loss: 0.0135 - acc: 0.9957 - val_l
oss: 0.0807 - val_acc: 0.9809
Epoch 11/20
60000/60000 [==============================] - 10s 162us/step - loss: 0.0142 - acc: 0.9953 - val_l
oss: 0.0707 - val_acc: 0.9823
Epoch 12/20
60000/60000 [==============================] - 11s 178us/step - loss: 0.0114 - acc: 0.9964 - val_l
oss: 0.0849 - val_acc: 0.9812
Epoch 13/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0098 - acc: 0.9969 -
val_loss: 0.0859 - val_acc: 0.9797
Epoch 14/20
60000/60000 [==============================] - 9s 156us/step - loss: 0.0117 - acc: 0.9962 -
val_loss: 0.0986 - val_acc: 0.9789
Epoch 15/20
60000/60000 [==============================] - 10s 169us/step - loss: 0.0135 - acc: 0.9955 - val_l
oss: 0.0783 - val_acc: 0.9831
Epoch 16/20
60000/60000 [==============================] - 11s 183us/step - loss: 0.0099 - acc: 0.9968 - val_l
oss: 0.0885 - val_acc: 0.9825
Epoch 17/20
60000/60000 [==============================] - 11s 186us/step - loss: 0.0083 - acc: 0.9972 - val_l
oss: 0.0940 - val_acc: 0.9814s:
Epoch 18/20
60000/60000 [==============================] - 8s 136us/step - loss: 0.0088 - acc: 0.9976 -
val_loss: 0.0995 - val_acc: 0.9811
Epoch 19/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.0095 - acc: 0.9972 - val_l
oss: 0.1119 - val_acc: 0.9760
Epoch 20/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.0086 - acc: 0.9973 -
val_loss: 0.0825 - val_acc: 0.9829
```

In [24]:

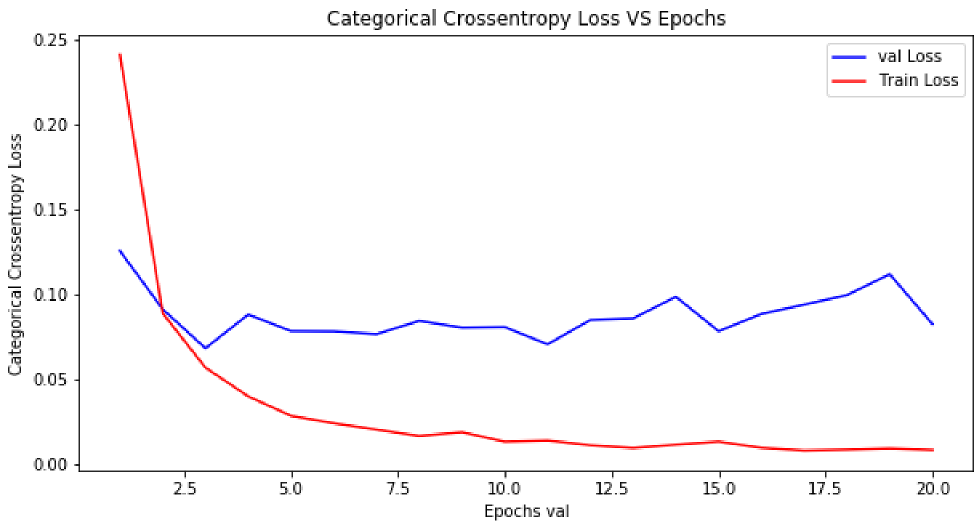
```
x = list(range(1,nb_epoch+1))

# getting Val loss
vy = history3.history['val_loss']
# getting Train loss
ty = history3.history['loss']

# function call
plt_dynamic_model(x, vy, ty)
```

```
# Evaluating the model
model_score = model3.evaluate(X_test, Y_test, verbose=0)
print('Test score:', model_score[0])
print('Test accuracy:', model_score[1])

# saving train and test accuracy of the model
model3_test_acc = model_score[1]
model3_train_acc = history3.history['acc']
```




```
Test score: 0.08253725015399864
Test accuracy: 0.9829
```

Softmax classifier with 3-hidden layers With Droput and Batch Normalization

In [10]:

```
model3dn = Sequential()

model3dn.add(Dense(465, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(s
eed=None)))
model3dn.add(BatchNormalization())
model3dn.add(Dropout(0.5))

model3dn.add(Dense(141, activation='relu', kernel_initializer=he_normal(seed=None)))
model3dn.add(BatchNormalization())
model3dn.add(Dropout(0.5))

model3dn.add(Dense(65, activation='relu', kernel_initializer=he_normal(seed=None)))
model3dn.add(BatchNormalization())
model3dn.add(Dropout(0.5))

model3dn.add(Dense(output_dim, activation='softmax'))
print(model3dn.summary())
model3dn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history3dn = model3dn.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, vali
dation_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_10 (Dense)             (None, 465)               365025    
_________________________________________________________________
batch_normalization_3 (Batch (None, 465)               1860      
_________________________________________________________________
dropout_3 (Dropout)          (None, 465)               0         
_________________________________________________________________
dense_11 (Dense)             (None, 141)               65706     
_________________________________________________________________
batch_normalization_4 (Batch (None, 141)               564       
_________________________________________________________________
dropout_4 (Dropout)          (None, 141)               0         
```

```
_________________________________________________________________
dense_12 (Dense)             (None, 65)                9230      
_________________________________________________________________
batch_normalization_5 (Batch (None, 65)                260       
_________________________________________________________________
dropout_5 (Dropout)          (None, 65)                0         
_________________________________________________________________
dense_13 (Dense)             (None, 10)                660       
=================================================================
Total params: 443,305
Trainable params: 441,963
Non-trainable params: 1,342
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 15s 249us/step - loss: 0.6878 - acc: 0.7869 - val_l
oss: 0.1866 - val_acc: 0.9435
Epoch 2/20
60000/60000 [==============================] - 12s 198us/step - loss: 0.2941 - acc: 0.9160 - val_l
oss: 0.1342 - val_acc: 0.9599
Epoch 3/20
60000/60000 [==============================] - 12s 196us/step - loss: 0.2237 - acc: 0.9382 - val_l
oss: 0.1131 - val_acc: 0.9666
Epoch 4/20
60000/60000 [==============================] - 12s 194us/step - loss: 0.1874 - acc: 0.9471 - val_l
oss: 0.0936 - val_acc: 0.9719
Epoch 5/20
60000/60000 [==============================] - 12s 198us/step - loss: 0.1680 - acc: 0.9526 - val_l
oss: 0.0836 - val_acc: 0.9739
Epoch 6/20
60000/60000 [==============================] - 12s 195us/step - loss: 0.1513 - acc: 0.9578 - val_l
oss: 0.0823 - val_acc: 0.9744
Epoch 7/20
60000/60000 [==============================] - 11s 186us/step - loss: 0.1370 - acc: 0.9615 - val_l
oss: 0.0824 - val_acc: 0.9765
Epoch 8/20
60000/60000 [==============================] - 11s 186us/step - loss: 0.1331 - acc: 0.9627 - val_l
oss: 0.0792 - val_acc: 0.9777
Epoch 9/20
60000/60000 [==============================] - 11s 189us/step - loss: 0.1241 - acc: 0.9657 - val_l
oss: 0.0701 - val_acc: 0.9792
Epoch 10/20
60000/60000 [==============================] - 11s 188us/step - loss: 0.1115 - acc: 0.9685 - val_l
oss: 0.0680 - val_acc: 0.9814
Epoch 11/20
60000/60000 [==============================] - 12s 202us/step - loss: 0.1044 - acc: 0.9698 - val_l
oss: 0.0720 - val_acc: 0.9793
Epoch 12/20
60000/60000 [==============================] - 11s 191us/step - loss: 0.1035 - acc: 0.9713 - val_l
oss: 0.0702 - val_acc: 0.9798
Epoch 13/20
60000/60000 [==============================] - 11s 190us/step - loss: 0.0996 - acc: 0.9722 - val_l
oss: 0.0714 - val_acc: 0.9802
Epoch 14/20
60000/60000 [==============================] - 12s 198us/step - loss: 0.0957 - acc: 0.9717 - val_l
oss: 0.0608 - val_acc: 0.9821
Epoch 15/20
60000/60000 [==============================] - 12s 193us/step - loss: 0.0905 - acc: 0.9746 - val_l
oss: 0.0658 - val_acc: 0.9806
Epoch 16/20
60000/60000 [==============================] - 12s 197us/step - loss: 0.0862 - acc: 0.9759 - val_l
oss: 0.0633 - val_acc: 0.9825
Epoch 17/20
60000/60000 [==============================] - 12s 201us/step - loss: 0.0857 - acc: 0.9749 - val_l
oss: 0.0631 - val_acc: 0.9815
Epoch 18/20
60000/60000 [==============================] - 12s 197us/step - loss: 0.0810 - acc: 0.9763 - val_l
oss: 0.0597 - val_acc: 0.9830
Epoch 19/20
60000/60000 [==============================] - 12s 194us/step - loss: 0.0776 - acc: 0.9778 - val_l
oss: 0.0616 - val_acc: 0.9829
Epoch 20/20
60000/60000 [==============================] - 12s 204us/step - loss: 0.0734 - acc: 0.9786 - val_l
oss: 0.0572 - val_acc: 0.9834
```

In [19]:

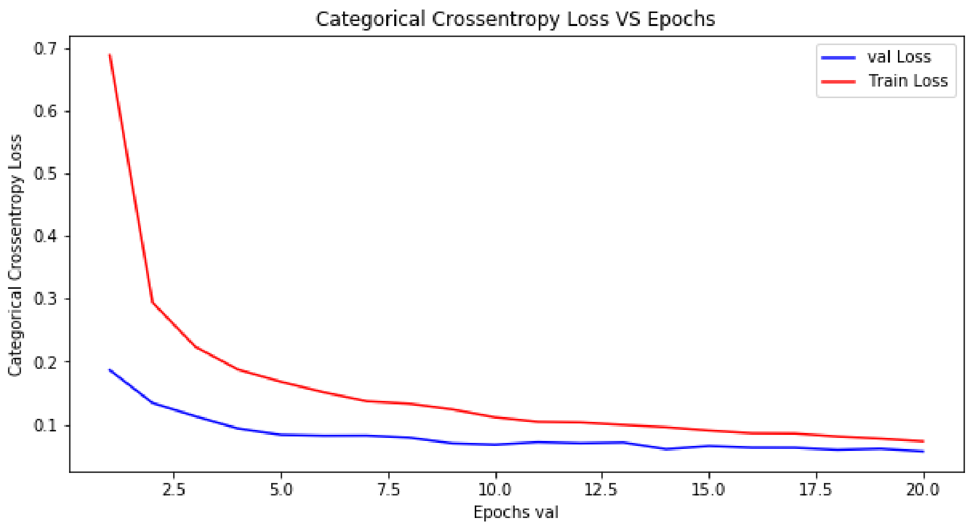
```
x = list(range(1,nb_epoch+1))

# getting Val loss
vy = history3dn.history['val_loss']
# getting Train loss
ty = history3dn.history['loss']

# function call
plt_dynamic_model(x, vy, ty)

# Evaluating the model
model_score = model3dn.evaluate(X_test, Y_test, verbose=0)
print('Test score:', model_score[0])
print('Test accuracy:', model_score[1])

# saving train and test accuracy of the model
model3dn_test_acc = model_score[1]
model3dn_train_acc = history3dn.history['acc']
```




```
Test score: 0.05717137544195284
Test accuracy: 0.9834
```

3. Softmax classifier with 5-hidden layers without dropout and Batch Normalization

In [11]:

```
model5 = Sequential()

model5.add(Dense(513, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(see
d=None)))
model5.add(Dense(252, activation='relu', kernel_initializer=he_normal(seed=None)))
model5.add(Dense(124, activation='relu', kernel_initializer=he_normal(seed=None)))
model5.add(Dense(68, activation='relu', kernel_initializer=he_normal(seed=None)))
model5.add(Dense(31, activation='relu', kernel_initializer=he_normal(seed=None)))

model5.add(Dense(output_dim, activation='softmax'))
print(model5.summary())
model5.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history5 = model5.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validati
on_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_14 (Dense)             (None, 513)               402705    
_________________________________________________________________
dense_15 (Dense)             (None, 252)               129528    
_________________________________________________________________
dense_16 (Dense)             (None, 124)               31372     
```

```
_________________________________________________________________
dense_17 (Dense)             (None, 68)                8500
_________________________________________________________________
dense_18 (Dense)             (None, 31)                2139
_________________________________________________________________
dense_19 (Dense)             (None, 10)                320
=================================================================
Total params: 574,564
Trainable params: 574,564
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 12s 205us/step - loss: 0.2576 - acc: 0.9213 - val_l
oss: 0.1284 - val_acc: 0.9606s: 0.2580 - acc: 0.921
Epoch 2/20
60000/60000 [==============================] - 12s 199us/step - loss: 0.0908 - acc: 0.9722 - val_l
oss: 0.0920 - val_acc: 0.9714
Epoch 3/20
60000/60000 [==============================] - 11s 185us/step - loss: 0.0620 - acc: 0.9814 - val_l
oss: 0.0863 - val_acc: 0.9727
Epoch 4/20
60000/60000 [==============================] - 12s 199us/step - loss: 0.0492 - acc: 0.9841 - val_l
oss: 0.0750 - val_acc: 0.9788
Epoch 5/20
60000/60000 [==============================] - 12s 207us/step - loss: 0.0350 - acc: 0.9891 - val_l
oss: 0.0834 - val_acc: 0.9766
Epoch 6/20
60000/60000 [==============================] - 11s 187us/step - loss: 0.0320 - acc: 0.9899 - val_l
oss: 0.0706 - val_acc: 0.9808
Epoch 7/20
60000/60000 [==============================] - 12s 207us/step - loss: 0.0243 - acc: 0.9920 - val_l
oss: 0.0812 - val_acc: 0.9806
Epoch 8/20
60000/60000 [==============================] - 12s 201us/step - loss: 0.0247 - acc: 0.9922 - val_l
oss: 0.0835 - val_acc: 0.9785
Epoch 9/20
60000/60000 [==============================] - 11s 182us/step - loss: 0.0203 - acc: 0.9936 - val_l
oss: 0.0936 - val_acc: 0.9785
Epoch 10/20
60000/60000 [==============================] - 11s 191us/step - loss: 0.0189 - acc: 0.9939 - val_l
oss: 0.0854 - val_acc: 0.9794
Epoch 11/20
60000/60000 [==============================] - 13s 223us/step - loss: 0.0184 - acc: 0.9939 - val_l
oss: 0.0839 - val_acc: 0.9804
Epoch 12/20
60000/60000 [==============================] - 13s 215us/step - loss: 0.0164 - acc: 0.9947 - val_l
oss: 0.0763 - val_acc: 0.9820
Epoch 13/20
60000/60000 [==============================] - 13s 218us/step - loss: 0.0134 - acc: 0.9960 - val_l
oss: 0.1041 - val_acc: 0.9769
Epoch 14/20
60000/60000 [==============================] - 11s 188us/step - loss: 0.0152 - acc: 0.9954 - val_l
oss: 0.0660 - val_acc: 0.9843
Epoch 15/20
60000/60000 [==============================] - 11s 184us/step - loss: 0.0133 - acc: 0.9959 - val_l
oss: 0.0851 - val_acc: 0.9819
Epoch 16/20
60000/60000 [==============================] - 11s 179us/step - loss: 0.0127 - acc: 0.9961 - val_l
oss: 0.0964 - val_acc: 0.9785 -
Epoch 17/20
60000/60000 [==============================] - 11s 180us/step - loss: 0.0104 - acc: 0.9969 - val_l
oss: 0.0784 - val_acc: 0.9825
Epoch 18/20
60000/60000 [==============================] - 11s 183us/step - loss: 0.0107 - acc: 0.9969 - val_l
oss: 0.1170 - val_acc: 0.9781
Epoch 19/20
60000/60000 [==============================] - 12s 194us/step - loss: 0.0110 - acc: 0.9966 - val_l
oss: 0.1014 - val_acc: 0.9786
Epoch 20/20
60000/60000 [==============================] - 11s 189us/step - loss: 0.0098 - acc: 0.9972 - val_l
oss: 0.1006 - val_acc: 0.9809
```

In [18]:

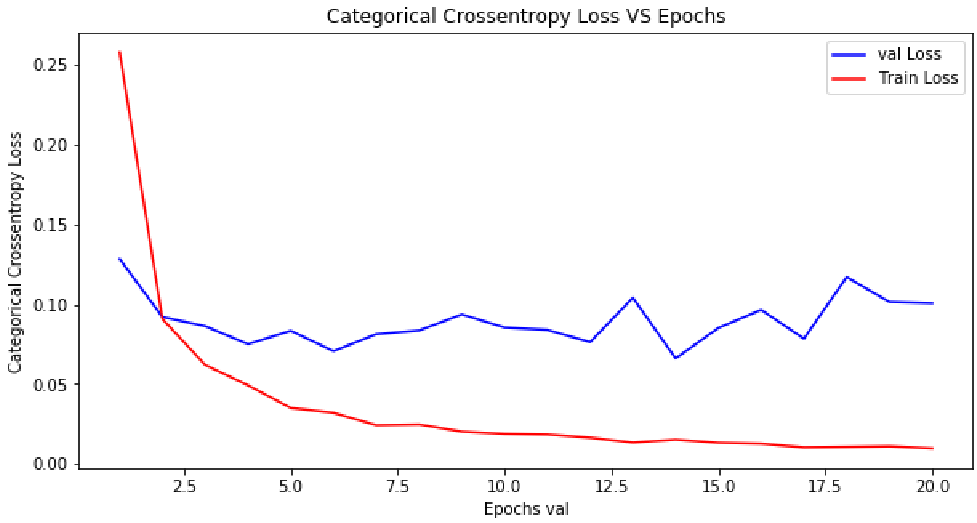
```
x = list(range(1,nb_epoch+1))

# getting Val loss
vy = history5.history['val_loss']
# getting Train loss
ty = history5.history['loss']

# function call
plt_dynamic_model(x, vy, ty)

# Evaluating the model
model_score = model5.evaluate(X_test, Y_test, verbose=0)
print('Test score:', model_score[0])
print('Test accuracy:', model_score[1])

# saving train and test accuracy of the model
model5_test_acc = model_score[1]
model5_train_acc = history5.history['acc']
```




```
Test score: 0.10062611548545246
Test accuracy: 0.9809
```

Softmax classifier with 5-hidden layers With Dropout and Batch Normalisation

In [12]:

```
model5dn = Sequential()

model5dn.add(Dense(513, activation='relu', input_shape=(input_dim,), kernel_initializer=he_normal(s
eed=None)))
model5dn.add(BatchNormalization())
model5dn.add(Dropout(0.5))

model5dn.add(Dense(252, activation='relu', kernel_initializer=he_normal(seed=None)))
model5dn.add(BatchNormalization())
model5dn.add(Dropout(0.5))

model5dn.add(Dense(124, activation='relu', kernel_initializer=he_normal(seed=None)))
model5dn.add(BatchNormalization())
model5dn.add(Dropout(0.5))

model5dn.add(Dense(68, activation='relu', kernel_initializer=he_normal(seed=None)))
model5dn.add(BatchNormalization())
model5dn.add(Dropout(0.5))

model5dn.add(Dense(31, activation='relu', kernel_initializer=he_normal(seed=None)))
model5dn.add(BatchNormalization())
model5dn.add(Dropout(0.5))

model5dn.add(Dense(output_dim, activation='softmax'))
print(model5dn.summary())
model5dn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history5dn = model5dn.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, vali
```

```
dation_data=(X_test, Y_test))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_20 (Dense)             (None, 513)               402705    
_________________________________________________________________
batch_normalization_6 (Batch (None, 513)               2052      
_________________________________________________________________
dropout_6 (Dropout)          (None, 513)               0         
_________________________________________________________________
dense_21 (Dense)             (None, 252)               129528    
_________________________________________________________________
batch_normalization_7 (Batch (None, 252)               1008      
_________________________________________________________________
dropout_7 (Dropout)          (None, 252)               0         
_________________________________________________________________
dense_22 (Dense)             (None, 124)               31372     
_________________________________________________________________
batch_normalization_8 (Batch (None, 124)               496       
_________________________________________________________________
dropout_8 (Dropout)          (None, 124)               0         
_________________________________________________________________
dense_23 (Dense)             (None, 68)                8500      
_________________________________________________________________
batch_normalization_9 (Batch (None, 68)                272       
_________________________________________________________________
dropout_9 (Dropout)          (None, 68)                0         
_________________________________________________________________
dense_24 (Dense)             (None, 31)                2139      
_________________________________________________________________
batch_normalization_10 (Batc (None, 31)                124       
_________________________________________________________________
dropout_10 (Dropout)         (None, 31)                0         
_________________________________________________________________
dense_25 (Dense)             (None, 10)                320       
=================================================================
Total params: 578,516
Trainable params: 576,540
Non-trainable params: 1,976
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 19s 319us/step - loss: 1.4600 - acc: 0.5162 - val_l
oss: 0.3729 - val_acc: 0.9069
Epoch 2/20
60000/60000 [==============================] - 18s 294us/step - loss: 0.6099 - acc: 0.8234 - val_l
oss: 0.2034 - val_acc: 0.9439
Epoch 3/20
60000/60000 [==============================] - 16s 272us/step - loss: 0.4101 - acc: 0.8950 - val_l
oss: 0.1691 - val_acc: 0.9560
Epoch 4/20
60000/60000 [==============================] - 17s 276us/step - loss: 0.3345 - acc: 0.9194 - val_l
oss: 0.1409 - val_acc: 0.9644
Epoch 5/20
60000/60000 [==============================] - 16s 274us/step - loss: 0.2907 - acc: 0.9314 - val_l
oss: 0.1330 - val_acc: 0.9682
Epoch 6/20
60000/60000 [==============================] - 16s 269us/step - loss: 0.2616 - acc: 0.9389 - val_l
oss: 0.1170 - val_acc: 0.9709
Epoch 7/20
60000/60000 [==============================] - 16s 265us/step - loss: 0.2326 - acc: 0.9469 - val_l
oss: 0.1168 - val_acc: 0.9729
Epoch 8/20
60000/60000 [==============================] - 16s 273us/step - loss: 0.2174 - acc: 0.9508 - val_l
oss: 0.1052 - val_acc: 0.9750
Epoch 9/20
60000/60000 [==============================] - 16s 270us/step - loss: 0.2037 - acc: 0.9538 - val_l
oss: 0.1045 - val_acc: 0.9743
Epoch 10/20
60000/60000 [==============================] - 16s 268us/step - loss: 0.1946 - acc: 0.9564 - val_l
oss: 0.0987 - val_acc: 0.9768
Epoch 11/20
60000/60000 [==============================] - 16s 270us/step - loss: 0.1800 - acc: 0.9591 - val_l
oss: 0.0994 - val_acc: 0.9758
Epoch 12/20
```

```
60000/60000 [==============================] - 16s 270us/step - loss: 0.1758 - acc: 0.9611 - val_l
oss: 0.0921 - val_acc: 0.9783ss: 0. - ETA: 2s - loss: 0 - ETA: 1s - loss: 0.1739 - acc: 0. - ETA:
1s - loss: 0.1
Epoch 13/20
60000/60000 [==============================] - 16s 265us/step - loss: 0.1656 - acc: 0.9624 - val_l
oss: 0.0898 - val_acc: 0.9794
Epoch 14/20
60000/60000 [==============================] - 16s 265us/step - loss: 0.1603 - acc: 0.9634 - val_l
oss: 0.0887 - val_acc: 0.9785
Epoch 15/20
60000/60000 [==============================] - 17s 280us/step - loss: 0.1555 - acc: 0.9651 - val_l
oss: 0.0808 - val_acc: 0.9803
Epoch 16/20
60000/60000 [==============================] - 17s 279us/step - loss: 0.1532 - acc: 0.9655 - val_l
oss: 0.0898 - val_acc: 0.9795
Epoch 17/20
60000/60000 [==============================] - 16s 268us/step - loss: 0.1445 - acc: 0.9679 - val_l
oss: 0.0806 - val_acc: 0.9815
Epoch 18/20
60000/60000 [==============================] - 16s 269us/step - loss: 0.1384 - acc: 0.9698 - val_l
oss: 0.0776 - val_acc: 0.9822
Epoch 19/20
60000/60000 [==============================] - 16s 271us/step - loss: 0.1328 - acc: 0.9707 - val_l
oss: 0.0866 - val_acc: 0.9810
Epoch 20/20
60000/60000 [==============================] - 16s 274us/step - loss: 0.1320 - acc: 0.9704 - val_l
oss: 0.0807 - val_acc: 0.9825
```

In [17]:

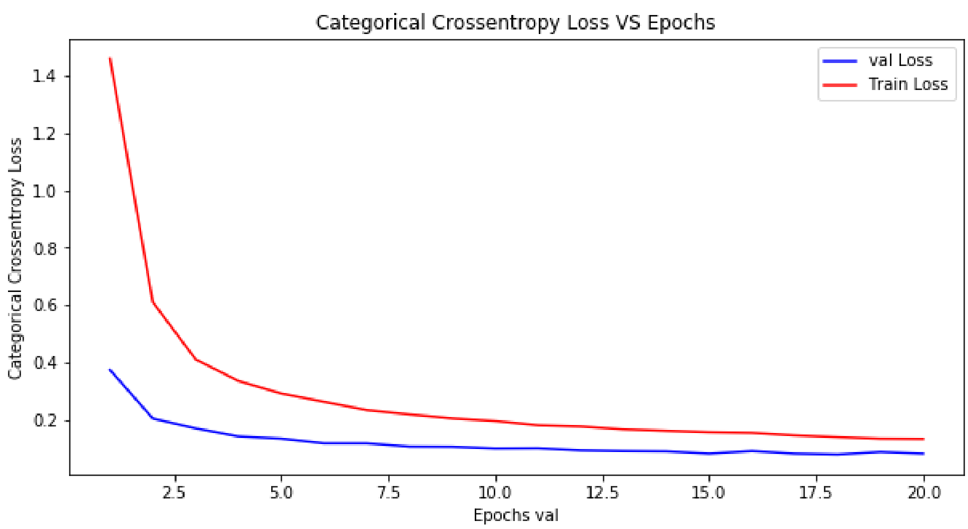
```
x = list(range(1,nb_epoch+1))

# getting Val loss
vy = history5dn.history['val_loss']
# getting Train loss
ty = history5dn.history['loss']

# function call
plt_dynamic_model(x, vy, ty)

# Evaluating the model
model_score = model5dn.evaluate(X_test, Y_test, verbose=0)
print('Test score:', model_score[0])
print('Test accuracy:', model_score[1])

# saving train and test accuracy of the model
model5dn_test_acc = model_score[1]
model5dn_train_acc = history5dn.history['acc']
```




```
Test score: 0.0807438613414066
Test accuracy: 0.9825
```

Conclustion

In [30]:

```
from prettytable import PrettyTable

print('Performance Table')
x = PrettyTable()
x.field_names =["Models","Train","Test"]

x.add_row(["2-Layer softmax without Dropout and BN",model2_train_acc[-1],model2_test_acc])
x.add_row(["2-Layer softmax with Dropout and BN ",model2dn_train_acc[-1],model2dn_test_acc])
x.add_row(["3-Layer softmax without Dropout and BN",model3_train_acc[-1],model3_test_acc])
x.add_row(["3-Layer softmax with Dropout and BN",model3dn_train_acc[-1],model3dn_test_acc])
x.add_row(["5-Layer softmax without Dropout and BN",model5_train_acc[-1],model5_test_acc])
x.add_row(["5-Layer softmax with Dropout and BN",model5dn_train_acc[-1],model5dn_test_acc])

print(x)
```

```
Performance Table
+----------------------------------------+--------------------+--------+
|                 Models                 |       Train        |  Test  |
+----------------------------------------+--------------------+--------+
| 2-Layer softmax without Dropout and BN | 0.9967166666666667 | 0.982  |
|  2-Layer softmax with Dropout and BN   | 0.9808166666666667 | 0.9818 |
| 3-Layer softmax without Dropout and BN | 0.9972833333333333 | 0.9829 |
|  3-Layer softmax with Dropout and BN   | 0.9785833333651225 | 0.9834 |
| 5-Layer softmax without Dropout and BN | 0.9971666666666666 | 0.9809 |
|  5-Layer softmax with Dropout and BN   | 0.9703666666666667 | 0.9825 |
+----------------------------------------+--------------------+--------+
```