In [0]:

```
# if your 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
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
```

In [0]:

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

In [3]:

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

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

In [4]:

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

```
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 [0]:

```
# if you observe the input shape its 2 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])
```

In [6]:

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

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

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

In [7]:

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

```
[   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 255
  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 [0]:

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

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

In [9]:

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

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

```
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.01176471 0.07058824 0.07058824 0.07058824
 0.49411765 0.53333333 0.68627451 0.10196078 0.65098039 1.
 0.96862745 0.49803922 0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.11764706 0.14117647 0.36862745 0.60392157
 0.66666667 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.88235294 0.6745098  0.99215686 0.94901961 0.76470588 0.25098039
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.19215686
 0.93333333 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.99215686 0.99215686 0.99215686 0.98431373 0.36470588 0.32156863
 0.32156863 0.21960784 0.15294118 0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.07058824 0.85882353 0.99215686
 0.99215686 0.99215686 0.99215686 0.99215686 0.77647059 0.71372549
 0.96862745 0.94509804 0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.31372549 0.61176471 0.41960784 0.99215686
 0.99215686 0.80392157 0.04313725 0.          0.16862745 0.60392157
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.05490196 0.00392157 0.60392157 0.99215686 0.35294118
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.54509804 0.99215686 0.74509804 0.00784314 0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.04313725
 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.          0.31764706 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.          0.17647059 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.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.        ]
```

In [10]:

```python
# 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]:

```python
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

1. Architecture 1: Two Hidden Layers (784-256-64-10)

With Batch Normalization and Dropout

In [19]:

```
model_relu_1 = Sequential()

model_relu_1.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_initializer='glorot
_normal'))
model_relu_1.add(BatchNormalization())
model_relu_1.add(Dropout(0.5))

model_relu_1.add(Dense(64, activation='relu', kernel_initializer='glorot_normal'))
model_relu_1.add(BatchNormalization())
model_relu_1.add(Dropout(0.5))

model_relu_1.add(Dense(output_dim, activation='softmax'))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:4479: The name tf.truncated_normal is deprecated. Ple
ase use tf.random.truncated_normal instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:148: The name tf.placeholder_with_default is
deprecated. Please use tf.compat.v1.placeholder_with_default instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:3733: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future
version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:4432: The name tf.random_uniform is deprecated. Pleas
e use tf.random.uniform instead.
```

In [22]:

```
model_relu_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_relu_1.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, val
idation_data=(X_test, Y_test))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow_core/python/ops/math_grad.py:1424: where (from
tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:1033: The name tf.assign_add is deprecated. Please us
e tf.compat.v1.assign_add instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:1020: The name tf.assign is deprecated. Please use tf
.compat.v1.assign instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:3005: The name tf.Session is deprecated. Please use t
f.compat.v1.Session instead.

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:190: The name tf.get_default_session is deprecated. P
lease use tf.compat.v1.get_default_session instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:197: The name tf.ConfigProto is deprecated. Please us
e tf.compat.v1.ConfigProto instead.
```

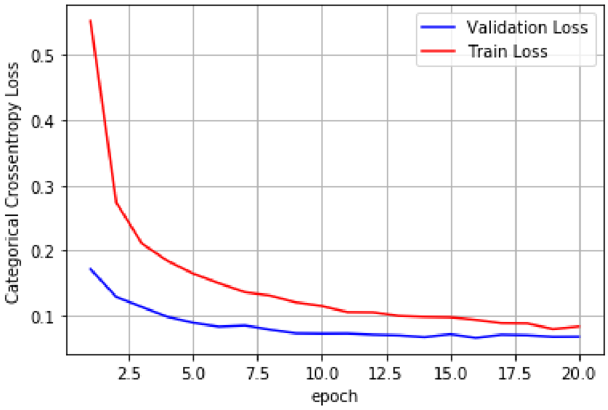
```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:207: The name tf.global_variables is deprecated. Plea
se use tf.compat.v1.global_variables instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:216: The name tf.is_variable_initialized is
deprecated. Please use tf.compat.v1.is_variable_initialized instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/keras/backend/tensorflow_backend.py:223: The name tf.variables_initializer is deprecated.
Please use tf.compat.v1.variables_initializer instead.

60000/60000 [==============================] - 6s 98us/step - loss: 0.5515 - acc: 0.8337 -
val_loss: 0.1715 - val_acc: 0.9484
Epoch 2/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.2734 - acc: 0.9208 -
val_loss: 0.1290 - val_acc: 0.9592
Epoch 3/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.2108 - acc: 0.9386 -
val_loss: 0.1136 - val_acc: 0.9663
Epoch 4/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.1841 - acc: 0.9476 -
val_loss: 0.0983 - val_acc: 0.9706
Epoch 5/20
60000/60000 [==============================] - 5s 82us/step - loss: 0.1646 - acc: 0.9522 -
val_loss: 0.0894 - val_acc: 0.9735
Epoch 6/20
60000/60000 [==============================] - 5s 83us/step - loss: 0.1500 - acc: 0.9566 -
val_loss: 0.0833 - val_acc: 0.9745
Epoch 7/20
60000/60000 [==============================] - 5s 81us/step - loss: 0.1365 - acc: 0.9597 -
val_loss: 0.0853 - val_acc: 0.9733
Epoch 8/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.1310 - acc: 0.9615 -
val_loss: 0.0787 - val_acc: 0.9755
Epoch 9/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.1206 - acc: 0.9644 -
val_loss: 0.0733 - val_acc: 0.9775
Epoch 10/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.1151 - acc: 0.9657 -
val_loss: 0.0728 - val_acc: 0.9782
Epoch 11/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.1057 - acc: 0.9685 -
val_loss: 0.0730 - val_acc: 0.9782
Epoch 12/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.1054 - acc: 0.9691 -
val_loss: 0.0710 - val_acc: 0.9785
Epoch 13/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.0999 - acc: 0.9701 -
val_loss: 0.0700 - val_acc: 0.9784
Epoch 14/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.0984 - acc: 0.9704 -
val_loss: 0.0674 - val_acc: 0.9798
Epoch 15/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.0978 - acc: 0.9708 -
val_loss: 0.0718 - val_acc: 0.9795
Epoch 16/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.0938 - acc: 0.9722 -
val_loss: 0.0662 - val_acc: 0.9806
Epoch 17/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.0888 - acc: 0.9725 -
val_loss: 0.0710 - val_acc: 0.9800
Epoch 18/20
60000/60000 [==============================] - 5s 79us/step - loss: 0.0885 - acc: 0.9729 -
val_loss: 0.0702 - val_acc: 0.9791
Epoch 19/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.0797 - acc: 0.9758 -
val_loss: 0.0679 - val_acc: 0.9801
Epoch 20/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.0836 - acc: 0.9746 -
val_loss: 0.0681 - val_acc: 0.9798
```

In [23]:

```
score = model_relu_1.evaluate(X_test, Y_test, verbose=0)
print('Test Score: ', score[0])
```

```
print('Test Accuracy: ', score[1])

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

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

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

```
Test Score:  0.06807973722481983
Test Accuracy:  0.9798
```



Without Batch Normalization and Dropout

In [0]:

```
model_relu_1 = Sequential()

model_relu_1.add(Dense(256, activation='relu', input_shape=(input_dim,), kernel_initializer='glorot_normal'))

model_relu_1.add(Dense(64, activation='relu', kernel_initializer='glorot_normal'))

model_relu_1.add(Dense(output_dim, activation='softmax'))
```

In [25]:

```
model_relu_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_relu_1.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

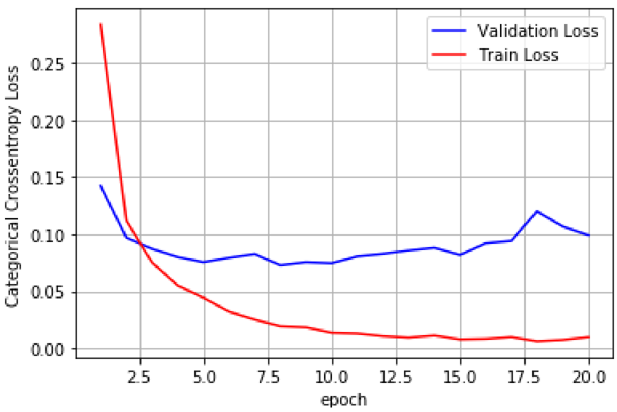
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 71us/step - loss: 0.2834 - acc: 0.9189 -
val_loss: 0.1421 - val_acc: 0.9574
Epoch 2/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.1117 - acc: 0.9670 -
val_loss: 0.0968 - val_acc: 0.9692
Epoch 3/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0751 - acc: 0.9775 -
val_loss: 0.0871 - val_acc: 0.9743
Epoch 4/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0550 - acc: 0.9836 -
val_loss: 0.0799 - val_acc: 0.9766
Epoch 5/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0443 - acc: 0.9860 -
val_loss: 0.0753 - val_acc: 0.9766
Epoch 6/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0320 - acc: 0.9903 -
val_loss: 0.0792 - val_acc: 0.9773
Epoch 7/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0251 - acc: 0.9924 -
```

```
val_loss: 0.0825 - val_acc: 0.9762
Epoch 8/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0193 - acc: 0.9940 -
val_loss: 0.0726 - val_acc: 0.9788
Epoch 9/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0183 - acc: 0.9942 -
val_loss: 0.0753 - val_acc: 0.9801
Epoch 10/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0135 - acc: 0.9959 -
val_loss: 0.0744 - val_acc: 0.9806
Epoch 11/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0129 - acc: 0.9960 -
val_loss: 0.0805 - val_acc: 0.9779
Epoch 12/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0106 - acc: 0.9966 -
val_loss: 0.0826 - val_acc: 0.9794
Epoch 13/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0093 - acc: 0.9970 -
val_loss: 0.0856 - val_acc: 0.9799
Epoch 14/20
60000/60000 [==============================] - 4s 63us/step - loss: 0.0112 - acc: 0.9964 -
val_loss: 0.0881 - val_acc: 0.9789
Epoch 15/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0077 - acc: 0.9974 -
val_loss: 0.0816 - val_acc: 0.9824
Epoch 16/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0081 - acc: 0.9972 -
val_loss: 0.0920 - val_acc: 0.9789
Epoch 17/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0098 - acc: 0.9968 -
val_loss: 0.0941 - val_acc: 0.9792
Epoch 18/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0060 - acc: 0.9980 -
val_loss: 0.1199 - val_acc: 0.9743
Epoch 19/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0072 - acc: 0.9974 -
val_loss: 0.1067 - val_acc: 0.9781
Epoch 20/20
60000/60000 [==============================] - 4s 64us/step - loss: 0.0097 - acc: 0.9967 -
val_loss: 0.0991 - val_acc: 0.9794
```

In [26]:

```python
score = model_relu_1.evaluate(X_test, Y_test, verbose=0)
print('Test Score: ', score[0])
print('Test Accuracy: ', score[1])

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

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

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

```
Test Score:  0.09909216562398315
Test Accuracy:  0.9794
```



2. Architecture: 3 Hidden Layers (784-512-256-128-10)

With Batch Normalization

In [0]:

```
model_relu_2 = Sequential()

model_relu_2.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer='glorot
_normal'))
model_relu_2.add(BatchNormalization())
model_relu_2.add(Dropout(0.5))

model_relu_2.add(Dense(256, activation='relu', kernel_initializer='glorot_normal'))
model_relu_2.add(BatchNormalization())
model_relu_2.add(Dropout(0.5))

model_relu_2.add(Dense(128, activation='relu', kernel_initializer='glorot_normal'))
model_relu_2.add(BatchNormalization())
model_relu_2.add(Dropout(0.5))

model_relu_2.add(Dense(output_dim, activation='softmax'))
```

In [30]:

```
model_relu_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_relu_2.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1,vali
dation_data=(X_test,Y_test))
```

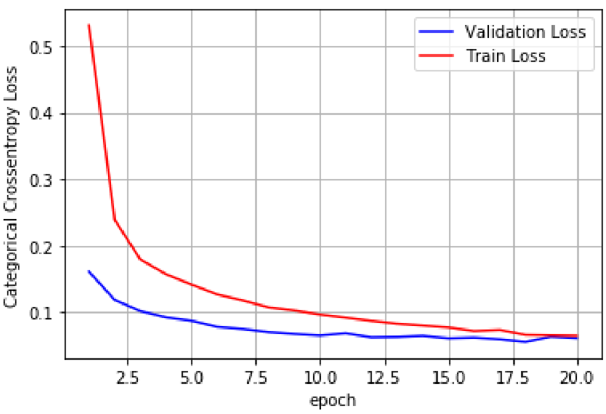
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 187us/step - loss: 0.5308 - acc: 0.8401 - val_l
oss: 0.1612 - val_acc: 0.9524
Epoch 2/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.2390 - acc: 0.9292 - val_l
oss: 0.1186 - val_acc: 0.9644
Epoch 3/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.1795 - acc: 0.9472 - val_l
oss: 0.1018 - val_acc: 0.9699
Epoch 4/20
60000/60000 [==============================] - 10s 173us/step - loss: 0.1573 - acc: 0.9536 - val_l
oss: 0.0925 - val_acc: 0.9722
Epoch 5/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.1416 - acc: 0.9577 - val_l
oss: 0.0872 - val_acc: 0.9743
Epoch 6/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.1269 - acc: 0.9619 - val_l
oss: 0.0781 - val_acc: 0.9773
Epoch 7/20
60000/60000 [==============================] - 10s 172us/step - loss: 0.1178 - acc: 0.9648 - val_l
oss: 0.0749 - val_acc: 0.9780
Epoch 8/20
60000/60000 [==============================] - 10s 174us/step - loss: 0.1074 - acc: 0.9686 - val_l
oss: 0.0702 - val_acc: 0.9783
Epoch 9/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.1028 - acc: 0.9696 - val_l
oss: 0.0675 - val_acc: 0.9794
Epoch 10/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.0966 - acc: 0.9712 - val_l
oss: 0.0652 - val_acc: 0.9801
Epoch 11/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.0920 - acc: 0.9720 - val_l
oss: 0.0685 - val_acc: 0.9807
Epoch 12/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.0871 - acc: 0.9744 - val_l
oss: 0.0624 - val_acc: 0.9814
Epoch 13/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.0829 - acc: 0.9755 - val_l
oss: 0.0630 - val_acc: 0.9808
Epoch 14/20
```

```
60000/60000 [==============================] - 10s 170us/step - loss: 0.0802 - acc: 0.9761 - val_l
oss: 0.0647 - val_acc: 0.9809
Epoch 15/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.0773 - acc: 0.9759 - val_l
oss: 0.0609 - val_acc: 0.9828
Epoch 16/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.0718 - acc: 0.9779 - val_l
oss: 0.0619 - val_acc: 0.9819
Epoch 17/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.0735 - acc: 0.9773 - val_l
oss: 0.0595 - val_acc: 0.9828
Epoch 18/20
60000/60000 [==============================] - 10s 170us/step - loss: 0.0661 - acc: 0.9802 - val_l
oss: 0.0554 - val_acc: 0.9830
Epoch 19/20
60000/60000 [==============================] - 10s 171us/step - loss: 0.0655 - acc: 0.9793 - val_l
oss: 0.0633 - val_acc: 0.9820
Epoch 20/20
60000/60000 [==============================] - 10s 169us/step - loss: 0.0649 - acc: 0.9806 - val_l
oss: 0.0612 - val_acc: 0.9818
```

In [31]:

```
score = model_relu_2.evaluate(X_test,Y_test,verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

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

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

```
Test score: 0.06115838211502996
Test accuracy: 0.9818
```



Without Batch Normalization

In [0]:

```
model_relu_2 = Sequential()

model_relu_2.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer='glorot_normal'))

model_relu_2.add(Dense(256, activation='relu', kernel_initializer='glorot_normal'))

model_relu_2.add(Dense(128, activation='relu', kernel_initializer='glorot_normal'))

model_relu_2.add(Dense(output_dim, activation='softmax'))
```

In [33]:

```
model_relu_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_relu_2.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1,validation_data=(X_test,Y_test))
```

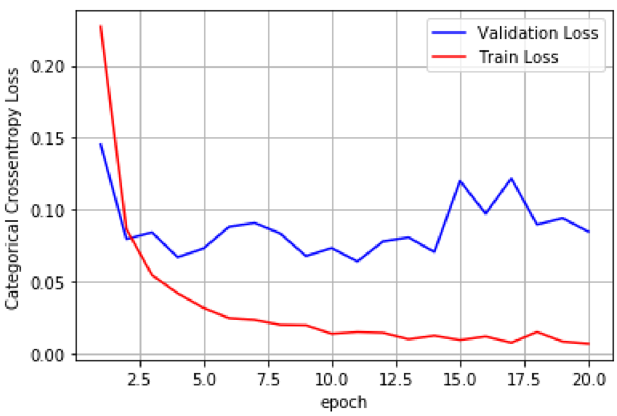
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.2272 - acc: 0.9325 -
val_loss: 0.1453 - val_acc: 0.9525
Epoch 2/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0865 - acc: 0.9739 -
val_loss: 0.0795 - val_acc: 0.9729
Epoch 3/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0544 - acc: 0.9829 -
val_loss: 0.0841 - val_acc: 0.9728
Epoch 4/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0418 - acc: 0.9867 -
val_loss: 0.0668 - val_acc: 0.9800
Epoch 5/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0316 - acc: 0.9898 -
val_loss: 0.0729 - val_acc: 0.9795
Epoch 6/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0244 - acc: 0.9917 -
val_loss: 0.0880 - val_acc: 0.9746
Epoch 7/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0233 - acc: 0.9922 -
val_loss: 0.0907 - val_acc: 0.9776
Epoch 8/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0198 - acc: 0.9933 -
val_loss: 0.0833 - val_acc: 0.9780
Epoch 9/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0195 - acc: 0.9936 -
val_loss: 0.0676 - val_acc: 0.9821
Epoch 10/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0136 - acc: 0.9956 -
val_loss: 0.0732 - val_acc: 0.9828
Epoch 11/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0148 - acc: 0.9951 -
val_loss: 0.0639 - val_acc: 0.9844
Epoch 12/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0143 - acc: 0.9955 -
val_loss: 0.0779 - val_acc: 0.9820
Epoch 13/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0098 - acc: 0.9968 -
val_loss: 0.0806 - val_acc: 0.9818
Epoch 14/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0123 - acc: 0.9962 -
val_loss: 0.0705 - val_acc: 0.9824
Epoch 15/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0092 - acc: 0.9969 -
val_loss: 0.1200 - val_acc: 0.9725
Epoch 16/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0117 - acc: 0.9963 -
val_loss: 0.0971 - val_acc: 0.9803
Epoch 17/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0073 - acc: 0.9979 -
val_loss: 0.1216 - val_acc: 0.9765
Epoch 18/20
60000/60000 [==============================] - 8s 135us/step - loss: 0.0149 - acc: 0.9953 -
val_loss: 0.0895 - val_acc: 0.9813
Epoch 19/20
```

```
Epoch 19/20
60000/60000 [==============================] - 8s 134us/step - loss: 0.0081 - acc: 0.9975 -
val_loss: 0.0940 - val_acc: 0.9819
Epoch 20/20
60000/60000 [==============================] - 8s 133us/step - loss: 0.0066 - acc: 0.9979 -
val_loss: 0.0846 - val_acc: 0.9821
```

In [34]:

```
score = model_relu_2.evaluate(X_test,Y_test,verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

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

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

```
Test score: 0.084612847658376
Test accuracy: 0.9821
```



In [0]:

```

```

3. Architecture: 5 Hidden Layers (784-512-256-128-64-32-10)

With Batch Normalization

In [0]:

```
model_relu_3 = Sequential()

model_relu_3.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer='glorot
_normal'))
model_relu_3.add(BatchNormalization())
model_relu_3.add(Dropout(0.5))
```

```
model_relu_3.add(Dropout(0.5))

model_relu_3.add(Dense(256, activation='relu', kernel_initializer='glorot_normal'))
model_relu_3.add(BatchNormalization())
model_relu_3.add(Dropout(0.5))

model_relu_3.add(Dense(128, activation='relu', kernel_initializer='glorot_normal'))
model_relu_3.add(BatchNormalization())
model_relu_3.add(Dropout(0.5))

model_relu_3.add(Dense(64, activation='relu', kernel_initializer='glorot_normal'))
model_relu_3.add(BatchNormalization())
model_relu_3.add(Dropout(0.5))

model_relu_3.add(Dense(32, activation='relu', kernel_initializer='glorot_normal'))
model_relu_3.add(BatchNormalization())
model_relu_3.add(Dropout(0.5))

model_relu_3.add(Dense(output_dim, activation='softmax'))
```

In [36]:

```
model_relu_3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_relu_3.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1,vali
dation_data=(X_test,Y_test))
```

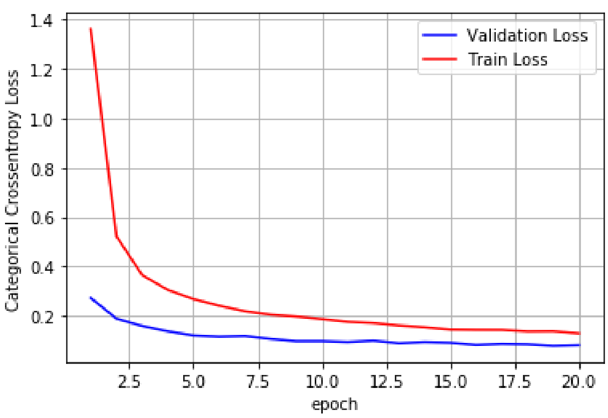
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 13s 220us/step - loss: 1.3623 - acc: 0.5673 - val_l
oss: 0.2719 - val_acc: 0.9246
Epoch 2/20
60000/60000 [==============================] - 11s 180us/step - loss: 0.5217 - acc: 0.8631 - val_l
oss: 0.1877 - val_acc: 0.9488
Epoch 3/20
60000/60000 [==============================] - 11s 179us/step - loss: 0.3646 - acc: 0.9108 - val_l
oss: 0.1580 - val_acc: 0.9578
Epoch 4/20
60000/60000 [==============================] - 11s 183us/step - loss: 0.3042 - acc: 0.9275 - val_l
oss: 0.1363 - val_acc: 0.9671
Epoch 5/20
60000/60000 [==============================] - 11s 182us/step - loss: 0.2670 - acc: 0.9376 - val_l
oss: 0.1194 - val_acc: 0.9709
Epoch 6/20
60000/60000 [==============================] - 11s 183us/step - loss: 0.2404 - acc: 0.9441 - val_l
oss: 0.1148 - val_acc: 0.9704
Epoch 7/20
60000/60000 [==============================] - 11s 182us/step - loss: 0.2177 - acc: 0.9492 - val_l
oss: 0.1173 - val_acc: 0.9717
Epoch 8/20
60000/60000 [==============================] - 11s 182us/step - loss: 0.2040 - acc: 0.9524 - val_l
oss: 0.1054 - val_acc: 0.9745
Epoch 9/20
60000/60000 [==============================] - 11s 183us/step - loss: 0.1965 - acc: 0.9549 - val_l
oss: 0.0963 - val_acc: 0.9747
Epoch 10/20
60000/60000 [==============================] - 11s 180us/step - loss: 0.1859 - acc: 0.9579 - val_l
oss: 0.0964 - val_acc: 0.9768
Epoch 11/20
60000/60000 [==============================] - 11s 180us/step - loss: 0.1749 - acc: 0.9603 - val_l
oss: 0.0925 - val_acc: 0.9784
Epoch 12/20
60000/60000 [==============================] - 11s 186us/step - loss: 0.1699 - acc: 0.9618 - val_l
oss: 0.0983 - val_acc: 0.9758
Epoch 13/20
60000/60000 [==============================] - 11s 180us/step - loss: 0.1599 - acc: 0.9640 - val_l
oss: 0.0878 - val_acc: 0.9788
Epoch 14/20
60000/60000 [==============================] - 11s 185us/step - loss: 0.1525 - acc: 0.9648 - val_l
oss: 0.0926 - val_acc: 0.9778
Epoch 15/20
60000/60000 [==============================] - 11s 186us/step - loss: 0.1434 - acc: 0.9675 - val_l
oss: 0.0898 - val_acc: 0.9791
Epoch 16/20
60000/60000 [==============================] - 11s 184us/step - loss: 0.1423 - acc: 0.9680 - val_l
oss: 0.0812 - val_acc: 0.9811
```

```
Epoch 17/20
60000/60000 [==============================] - 11s 185us/step - loss: 0.1421 - acc: 0.9669 - val_l
oss: 0.0852 - val_acc: 0.9805
Epoch 18/20
60000/60000 [==============================] - 11s 184us/step - loss: 0.1359 - acc: 0.9691 - val_l
oss: 0.0836 - val_acc: 0.9813
Epoch 19/20
60000/60000 [==============================] - 11s 184us/step - loss: 0.1365 - acc: 0.9691 - val_l
oss: 0.0772 - val_acc: 0.9806
Epoch 20/20
60000/60000 [==============================] - 11s 184us/step - loss: 0.1280 - acc: 0.9703 - val_l
oss: 0.0800 - val_acc: 0.9816
```

In [37]:

```
score = model_relu_3.evaluate(X_test,Y_test,verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

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

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

```
Test score: 0.08003878279228228
Test accuracy: 0.9816
```



Without Batch Normalization

In [0]:

```
model_relu_3 = Sequential()

model_relu_3.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initializer='glorot
_normal'))

model_relu_3.add(Dense(256, activation='relu', kernel_initializer='glorot_normal'))
```

```
model_relu_3.add(Dense(128, activation='relu', kernel_initializer='glorot_normal'))

model_relu_3.add(Dense(64, activation='relu', kernel_initializer='glorot_normal'))

model_relu_3.add(Dense(32, activation='relu', kernel_initializer='glorot_normal'))

model_relu_3.add(Dense(output_dim, activation='softmax'))
```

In [39]:

```
model_relu_3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_relu_3.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1,vali
dation_data=(X_test,Y_test))
```

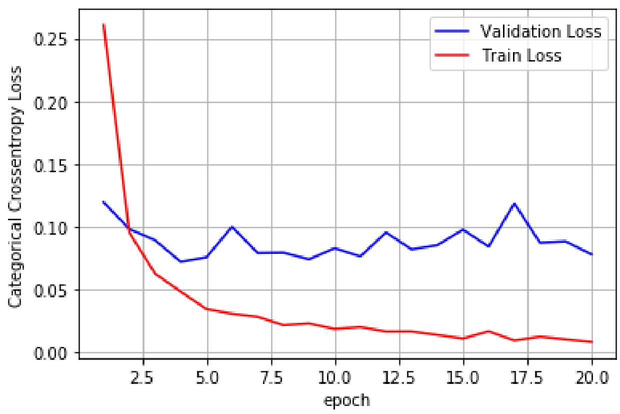
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s 162us/step - loss: 0.2607 - acc: 0.9216 - val_l
oss: 0.1193 - val_acc: 0.9627
Epoch 2/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0951 - acc: 0.9711 -
val_loss: 0.0980 - val_acc: 0.9683
Epoch 3/20
60000/60000 [==============================] - 8s 140us/step - loss: 0.0625 - acc: 0.9802 -
val_loss: 0.0891 - val_acc: 0.9707
Epoch 4/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.0480 - acc: 0.9846 -
val_loss: 0.0718 - val_acc: 0.9784
Epoch 5/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.0341 - acc: 0.9889 -
val_loss: 0.0754 - val_acc: 0.9788
Epoch 6/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.0303 - acc: 0.9903 -
val_loss: 0.0998 - val_acc: 0.9712
Epoch 7/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0279 - acc: 0.9911 -
val_loss: 0.0789 - val_acc: 0.9784
Epoch 8/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.0213 - acc: 0.9934 -
val_loss: 0.0792 - val_acc: 0.9783
Epoch 9/20
60000/60000 [==============================] - 8s 137us/step - loss: 0.0226 - acc: 0.9928 -
val_loss: 0.0738 - val_acc: 0.9791
Epoch 10/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.0182 - acc: 0.9939 -
val_loss: 0.0827 - val_acc: 0.9792
Epoch 11/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0198 - acc: 0.9938 -
val_loss: 0.0762 - val_acc: 0.9808
Epoch 12/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0162 - acc: 0.9948 -
val_loss: 0.0952 - val_acc: 0.9773
Epoch 13/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0162 - acc: 0.9949 -
val_loss: 0.0817 - val_acc: 0.9813
Epoch 14/20
60000/60000 [==============================] - 8s 140us/step - loss: 0.0135 - acc: 0.9959 -
val_loss: 0.0851 - val_acc: 0.9795
Epoch 15/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0106 - acc: 0.9969 -
val_loss: 0.0975 - val_acc: 0.9787
Epoch 16/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0164 - acc: 0.9951 -
val_loss: 0.0840 - val_acc: 0.9819
Epoch 17/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0090 - acc: 0.9972 -
val_loss: 0.1184 - val_acc: 0.9760
Epoch 18/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0121 - acc: 0.9964 -
val_loss: 0.0870 - val_acc: 0.9826
Epoch 19/20
60000/60000 [==============================] - 8s 138us/step - loss: 0.0099 - acc: 0.9970 -
val_loss: 0.0881 - val_acc: 0.9826
Epoch 20/20
60000/60000 [==============================] - 8s 139us/step - loss: 0.0079 - acc: 0.9976 -
```

```
val_loss: 0.0779 - val_acc: 0.9829
```

In [40]:

```python
score = model_relu_3.evaluate(X_test,Y_test,verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

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

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

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

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

```
Test score: 0.07785627461677641
Test accuracy: 0.9829
```



In [2]:

```python
from prettytable import PrettyTable

table = PrettyTable()

table.field_names = ['No. of Layers', 'Batch Normalisation and Dropouts', 'Test Score', 'Test Accur
acy', 'Convergence']

table.add_row(['2', 'Yes', '6.81', '97.98', 'Quick'])
table.add_row(['2', 'No', '9.91', '97.94', 'Slow'])
table.add_row(['3', 'Yes', '6.12', '98.18', 'Quick'])
table.add_row(['3', 'No', '8.46', '98.21', 'Slow'])
table.add_row(['5', 'Yes', '8.00', '98.16', 'Quick'])
table.add_row(['5', 'No', '7.79', '98.29', 'Slow'])

print(table)
```

```
+---------------+----------------------------------+------------+---------------+-------------+
| No. of Layers | Batch Normalisation and Dropouts | Test Score | Test Accuracy | Convergence |
+---------------+----------------------------------+------------+---------------+-------------+
|       2       |               Yes                |    6.81    |     97.98     |    Quick    |
```

```
|       2       |                 No                 |    9.91    |     97.94     |    Slow     |
|       3       |                Yes                 |    6.12    |     98.18     |    Quick    |
|       3       |                 No                 |    8.46    |     98.21     |    Slow     |
|       5       |                Yes                 |    8.00    |     98.16     |    Quick    |
|       5       |                 No                 |    7.79    |     98.29     |    Slow     |
+---------------+------------------------------------+------------+---------------+-------------+
```

In [0]:

```

```