In [0]:

```
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
import matplotlib.pyplot as plt
import numpy as np
import time
```

In [0]:

```
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 [0]:

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

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

In [0]:

```
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 [0]:

```
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   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]:

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

In [0]:

```
print(X_train[0])
```

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

```
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.04313725
0.74509804 0.99215686 0.2745098  0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.1372549  0.94509804
0.88235294 0.62745098 0.42352941 0.00392157 0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.31764706 0.94117647 0.99215686
0.99215686 0.46666667 0.09803922 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.17647059 0.72941176 0.99215686 0.99215686
0.58823529 0.10588235 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.0627451  0.36470588 0.98823529 0.99215686 0.73333333
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.97647059 0.99215686 0.97647059 0.25098039 0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.18039216 0.50980392 0.71764706 0.99215686
0.99215686 0.81176471 0.00784314 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.15294118 0.58039216
0.89803922 0.99215686 0.99215686 0.99215686 0.98039216 0.71372549
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.09411765 0.44705882 0.86666667 0.99215686 0.99215686 0.99215686
0.99215686 0.78823529 0.30588235 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.09019608 0.25882353 0.83529412 0.99215686
0.99215686 0.99215686 0.99215686 0.77647059 0.31764706 0.00784314
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.07058824 0.67058824
0.85882353 0.99215686 0.99215686 0.99215686 0.99215686 0.76470588
0.31372549 0.03529412 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.21568627 0.6745098  0.88627451 0.99215686 0.99215686 0.99215686
0.99215686 0.95686275 0.52156863 0.04313725 0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.53333333 0.99215686
0.99215686 0.99215686 0.83137255 0.52941176 0.51764706 0.0627451
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.        ]
```

In [0]:

```
print("Class label of first image :", y_train[0])
```

```
Class label of first image : 5
```

In [0]:

```
from keras.utils import np_utils
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])
```

```
After converting the output into a vector :  [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

In [0]:

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

In [0]:

```
output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

For (784 - 488 - 256 - 10) Architecture

In [0]:

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

model = Sequential()

model.add(Dense(488, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(m
ean=0.0, stddev=0.039, seed=None)))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed
=None)) )
model.add(BatchNormalization())
model.add(Dropout(0.5))

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


model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 488)               383080
_________________________________________________________________
batch_normalization_5 (Batch (None, 488)               1952
_________________________________________________________________
dropout_5 (Dropout)          (None, 488)               0
_________________________________________________________________
dense_8 (Dense)              (None, 256)               125184
_________________________________________________________________
batch_normalization_6 (Batch (None, 256)               1024
_________________________________________________________________
dropout_6 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                2570
```

```
=================================================================
Total params: 513,810
Trainable params: 512,322
Non-trainable params: 1,488
_________________________________________________________________
```

In [0]:

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

history = model.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 [==============================] - 11s 179us/step - loss: 0.4587 - acc: 0.8601 - val_l
oss: 0.1523 - val_acc: 0.9538
Epoch 2/20
60000/60000 [==============================] - 10s 163us/step - loss: 0.2336 - acc: 0.9285 - val_l
oss: 0.1202 - val_acc: 0.9620
Epoch 3/20
60000/60000 [==============================] - 10s 164us/step - loss: 0.1864 - acc: 0.9437 - val_l
oss: 0.1012 - val_acc: 0.9686
Epoch 4/20
60000/60000 [==============================] - 10s 166us/step - loss: 0.1616 - acc: 0.9516 - val_l
oss: 0.0855 - val_acc: 0.9728
Epoch 5/20
60000/60000 [==============================] - 10s 165us/step - loss: 0.1467 - acc: 0.9553 - val_l
oss: 0.0863 - val_acc: 0.9723
Epoch 6/20
60000/60000 [==============================] - 10s 164us/step - loss: 0.1282 - acc: 0.9610 - val_l
oss: 0.0823 - val_acc: 0.9739
Epoch 7/20
60000/60000 [==============================] - 10s 165us/step - loss: 0.1258 - acc: 0.9615 - val_l
oss: 0.0802 - val_acc: 0.9740
Epoch 8/20
60000/60000 [==============================] - 10s 165us/step - loss: 0.1143 - acc: 0.9644 - val_l
oss: 0.0752 - val_acc: 0.9764
Epoch 9/20
60000/60000 [==============================] - 10s 165us/step - loss: 0.1080 - acc: 0.9660 - val_l
oss: 0.0709 - val_acc: 0.9780
Epoch 10/20
60000/60000 [==============================] - 10s 167us/step - loss: 0.1030 - acc: 0.9677 - val_l
oss: 0.0679 - val_acc: 0.9783
Epoch 11/20
60000/60000 [==============================] - 10s 167us/step - loss: 0.0971 - acc: 0.9699 - val_l
oss: 0.0657 - val_acc: 0.9796
Epoch 12/20
60000/60000 [==============================] - 10s 168us/step - loss: 0.0935 - acc: 0.9713 - val_l
oss: 0.0674 - val_acc: 0.9804
Epoch 13/20
60000/60000 [==============================] - 10s 166us/step - loss: 0.0841 - acc: 0.9735 - val_l
oss: 0.0637 - val_acc: 0.9802
Epoch 14/20
60000/60000 [==============================] - 10s 166us/step - loss: 0.0854 - acc: 0.9733 - val_l
oss: 0.0637 - val_acc: 0.9807
Epoch 15/20
60000/60000 [==============================] - 10s 168us/step - loss: 0.0810 - acc: 0.9746 - val_l
oss: 0.0652 - val_acc: 0.9819
Epoch 16/20
60000/60000 [==============================] - 10s 168us/step - loss: 0.0779 - acc: 0.9753 - val_l
oss: 0.0603 - val_acc: 0.9826
Epoch 17/20
60000/60000 [==============================] - 10s 166us/step - loss: 0.0762 - acc: 0.9757 - val_l
oss: 0.0608 - val_acc: 0.9825
Epoch 18/20
60000/60000 [==============================] - 10s 168us/step - loss: 0.0709 - acc: 0.9774 - val_l
oss: 0.0603 - val_acc: 0.9830
Epoch 19/20
60000/60000 [==============================] - 10s 166us/step - loss: 0.0658 - acc: 0.9783 - val_l
oss: 0.0621 - val_acc: 0.9833
Epoch 20/20
60000/60000 [==============================] - 10s 166us/step - loss: 0.0681 - acc: 0.9785 - val_l
oss: 0.0618 - val_acc: 0.9819
```

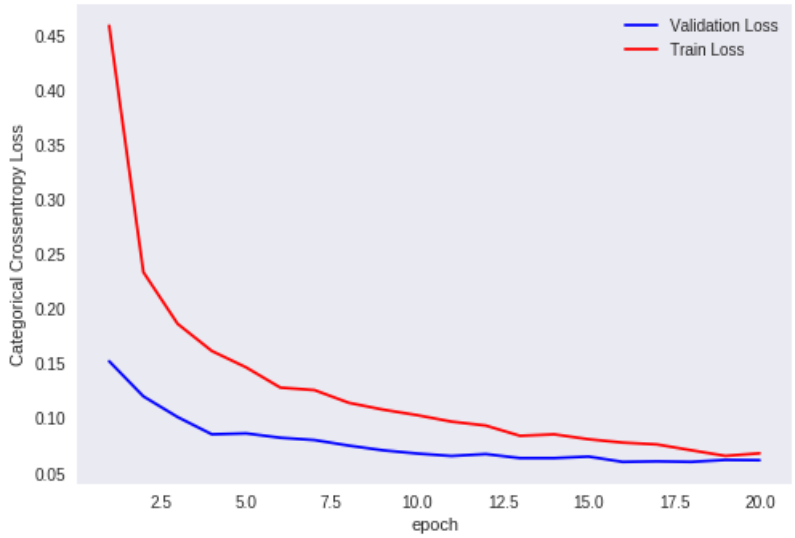
In [0]:

```
score = model.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: 0.0618182118332712
Test accuracy: 0.9819
```



For (784 - 515 - 319 - 127 - 10) 3 hidden layer Architecture

In [0]:

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

model = Sequential()

model.add(Dense(515, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(319, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.039, seed=None)))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(127, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed=None)) )
model.add(BatchNormalization())
model.add(Dropout(0.5))

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


model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_16 (Dense)             (None, 515)               404275
```

```
_________________________________________________________________
batch_normalization_11 (Batc (None, 515)               2060
_________________________________________________________________
dropout_11 (Dropout)         (None, 515)               0
_________________________________________________________________
dense_17 (Dense)             (None, 319)               164604
_________________________________________________________________
batch_normalization_12 (Batc (None, 319)               1276
_________________________________________________________________
dropout_12 (Dropout)         (None, 319)               0
_________________________________________________________________
dense_18 (Dense)             (None, 127)               40640
_________________________________________________________________
batch_normalization_13 (Batc (None, 127)               508
_________________________________________________________________
dropout_13 (Dropout)         (None, 127)               0
_________________________________________________________________
dense_19 (Dense)             (None, 10)                1280
=================================================================
Total params: 614,643
Trainable params: 612,721
Non-trainable params: 1,922
_________________________________________________________________
```

In [0]:

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

history = model.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 [==============================] - 14s 240us/step - loss: 0.5505 - acc: 0.8335 - val_l
oss: 0.1642 - val_acc: 0.9487
Epoch 2/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.2447 - acc: 0.9290 - val_l
oss: 0.1198 - val_acc: 0.9626
Epoch 3/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.1930 - acc: 0.9433 - val_l
oss: 0.0975 - val_acc: 0.9711
Epoch 4/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.1625 - acc: 0.9525 - val_l
oss: 0.0862 - val_acc: 0.9740
Epoch 5/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.1450 - acc: 0.9571 - val_l
oss: 0.0858 - val_acc: 0.9740
Epoch 6/20
60000/60000 [==============================] - 13s 212us/step - loss: 0.1307 - acc: 0.9614 - val_l
oss: 0.0752 - val_acc: 0.9769
Epoch 7/20
60000/60000 [==============================] - 13s 214us/step - loss: 0.1244 - acc: 0.9629 - val_l
oss: 0.0725 - val_acc: 0.9788
Epoch 8/20
60000/60000 [==============================] - 13s 214us/step - loss: 0.1111 - acc: 0.9670 - val_l
oss: 0.0683 - val_acc: 0.9781
Epoch 9/20
60000/60000 [==============================] - 13s 214us/step - loss: 0.1070 - acc: 0.9677 - val_l
oss: 0.0667 - val_acc: 0.9803
Epoch 10/20
60000/60000 [==============================] - 13s 216us/step - loss: 0.1007 - acc: 0.9699 - val_l
oss: 0.0686 - val_acc: 0.9805
Epoch 11/20
60000/60000 [==============================] - 13s 215us/step - loss: 0.0945 - acc: 0.9720 - val_l
oss: 0.0656 - val_acc: 0.9813
Epoch 12/20
60000/60000 [==============================] - 13s 215us/step - loss: 0.0893 - acc: 0.9733 - val_l
oss: 0.0691 - val_acc: 0.9803
Epoch 13/20
60000/60000 [==============================] - 13s 219us/step - loss: 0.0854 - acc: 0.9743 - val_l
oss: 0.0667 - val_acc: 0.9811
Epoch 14/20
60000/60000 [==============================] - 13s 215us/step - loss: 0.0830 - acc: 0.9753 - val_l
oss: 0.0680 - val_acc: 0.9813
Epoch 15/20
60000/60000 [==============================] - 13s 215us/step - loss: 0.0799 - acc: 0.9759 - val_l
```

```
oss: 0.0592 - val_acc: 0.9819
Epoch 16/20
60000/60000 [==============================] - 13s 224us/step - loss: 0.0746 - acc: 0.9772 - val_l
oss: 0.0580 - val_acc: 0.9830
Epoch 17/20
60000/60000 [==============================] - 13s 218us/step - loss: 0.0746 - acc: 0.9775 - val_l
oss: 0.0608 - val_acc: 0.9827
Epoch 18/20
60000/60000 [==============================] - 13s 215us/step - loss: 0.0704 - acc: 0.9783 - val_l
oss: 0.0616 - val_acc: 0.9830
Epoch 19/20
60000/60000 [==============================] - 13s 215us/step - loss: 0.0646 - acc: 0.9798 - val_l
oss: 0.0595 - val_acc: 0.9841
Epoch 20/20
60000/60000 [==============================] - 13s 215us/step - loss: 0.0648 - acc: 0.9804 - val_l
oss: 0.0529 - val_acc: 0.9858
```

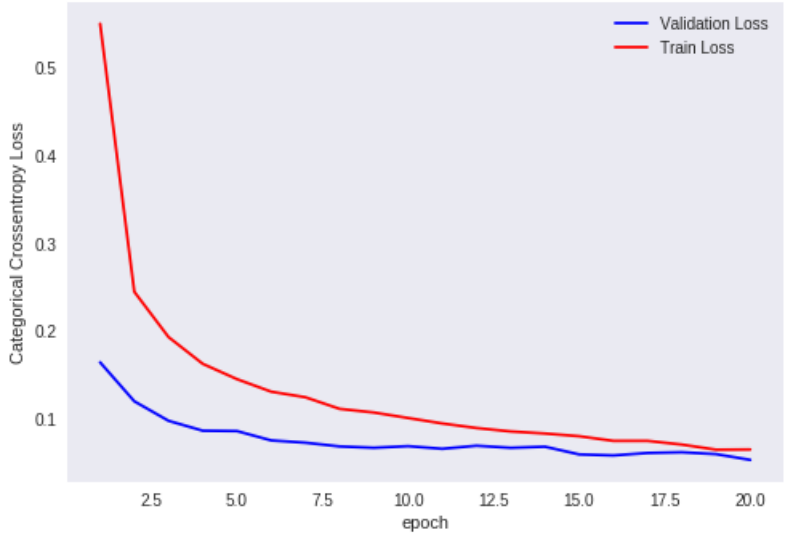
In [0]:

```
score = model.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: 0.052927311327529605
Test accuracy: 0.9858
```



For (784 - 600 - 567-428-385 -250- 10) 5 hidden layer Architecture

In [0]:

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

model = Sequential()

model.add(Dense(600, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(m
ean=0.0, stddev=0.039, seed=None)))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(567, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(m
ean=0.0, stddev=0.039, seed=None)))
model.add(BatchNormalization())
```

```
model.add(Dropout(0.5))

model.add(Dense(428, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(m
ean=0.0, stddev=0.039, seed=None)))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(385, activation='relu', input_shape=(input_dim,), kernel_initializer=RandomNormal(m
ean=0.0, stddev=0.039, seed=None)))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(250, activation='relu', kernel_initializer=RandomNormal(mean=0.0, stddev=0.55, seed
=None)) )
model.add(BatchNormalization())
model.add(Dropout(0.5))

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


model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_20 (Dense)             (None, 600)               471000
_________________________________________________________________
batch_normalization_14 (Batc (None, 600)               2400
_________________________________________________________________
dropout_14 (Dropout)         (None, 600)               0
_________________________________________________________________
dense_21 (Dense)             (None, 567)               340767
_________________________________________________________________
batch_normalization_15 (Batc (None, 567)               2268
_________________________________________________________________
dropout_15 (Dropout)         (None, 567)               0
_________________________________________________________________
dense_22 (Dense)             (None, 428)               243104
_________________________________________________________________
batch_normalization_16 (Batc (None, 428)               1712
_________________________________________________________________
dropout_16 (Dropout)         (None, 428)               0
_________________________________________________________________
dense_23 (Dense)             (None, 385)               165165
_________________________________________________________________
batch_normalization_17 (Batc (None, 385)               1540
_________________________________________________________________
dropout_17 (Dropout)         (None, 385)               0
_________________________________________________________________
dense_24 (Dense)             (None, 250)               96500
_________________________________________________________________
batch_normalization_18 (Batc (None, 250)               1000
_________________________________________________________________
dropout_18 (Dropout)         (None, 250)               0
_________________________________________________________________
dense_25 (Dense)             (None, 10)                2510
=================================================================
Total params: 1,327,966
Trainable params: 1,323,506
Non-trainable params: 4,460
_________________________________________________________________
```

In [0]:

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

history = model.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 [==============================] - 29s 481us/step - loss: 0.0717 - acc: 0.9793 - val_l
oss: 0.0594 - val_acc: 0.9850
Epoch 2/20
60000/60000 [==============================] - 26s 436us/step - loss: 0.0706 - acc: 0.9798 - val_l
```

```
oss: 0.0578 - val_acc: 0.9838
Epoch 3/20
60000/60000 [==============================] - 26s 435us/step - loss: 0.0695 - acc: 0.9798 - val_l
oss: 0.0601 - val_acc: 0.9833
Epoch 4/20
60000/60000 [==============================] - 26s 433us/step - loss: 0.0631 - acc: 0.9816 - val_l
oss: 0.0557 - val_acc: 0.9848
Epoch 5/20
60000/60000 [==============================] - 26s 433us/step - loss: 0.0616 - acc: 0.9816 - val_l
oss: 0.0511 - val_acc: 0.9857
Epoch 6/20
60000/60000 [==============================] - 26s 434us/step - loss: 0.0603 - acc: 0.9823 - val_l
oss: 0.0531 - val_acc: 0.9849
Epoch 7/20
60000/60000 [==============================] - 26s 435us/step - loss: 0.0605 - acc: 0.9827 - val_l
oss: 0.0578 - val_acc: 0.9846
Epoch 8/20
60000/60000 [==============================] - 26s 438us/step - loss: 0.0570 - acc: 0.9829 - val_l
oss: 0.0587 - val_acc: 0.9849
Epoch 9/20
60000/60000 [==============================] - 26s 436us/step - loss: 0.0571 - acc: 0.9836 - val_l
oss: 0.0520 - val_acc: 0.9857
Epoch 10/20
60000/60000 [==============================] - 27s 450us/step - loss: 0.0540 - acc: 0.9841 - val_l
oss: 0.0541 - val_acc: 0.9854
Epoch 11/20
60000/60000 [==============================] - 26s 437us/step - loss: 0.0513 - acc: 0.9839 - val_l
oss: 0.0558 - val_acc: 0.9854
Epoch 12/20
60000/60000 [==============================] - 26s 440us/step - loss: 0.0545 - acc: 0.9838 - val_l
oss: 0.0535 - val_acc: 0.9853
Epoch 13/20
60000/60000 [==============================] - 26s 436us/step - loss: 0.0501 - acc: 0.9851 - val_l
oss: 0.0533 - val_acc: 0.9864
Epoch 14/20
60000/60000 [==============================] - 26s 437us/step - loss: 0.0512 - acc: 0.9850 - val_l
oss: 0.0505 - val_acc: 0.9866
Epoch 15/20
60000/60000 [==============================] - 26s 439us/step - loss: 0.0490 - acc: 0.9854 - val_l
oss: 0.0530 - val_acc: 0.9863
Epoch 16/20
60000/60000 [==============================] - 26s 439us/step - loss: 0.0438 - acc: 0.9866 - val_l
oss: 0.0533 - val_acc: 0.9859
Epoch 17/20
60000/60000 [==============================] - 26s 438us/step - loss: 0.0455 - acc: 0.9862 - val_l
oss: 0.0528 - val_acc: 0.9857
Epoch 18/20
60000/60000 [==============================] - 26s 436us/step - loss: 0.0439 - acc: 0.9868 - val_l
oss: 0.0504 - val_acc: 0.9869
Epoch 19/20
60000/60000 [==============================] - 27s 442us/step - loss: 0.0427 - acc: 0.9874 - val_l
oss: 0.0546 - val_acc: 0.9857
Epoch 20/20
60000/60000 [==============================] - 27s 442us/step - loss: 0.0432 - acc: 0.9871 - val_l
oss: 0.0531 - val_acc: 0.9861
```

In [0]:

```
score = model.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: 0.05311666791146854
Test accuracy: 0.9861
```

In [1]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["MOdel","AUC"]

x.add_row([" (784 - 488 - 256 - 10) Architecture","98.19%"])
x.add_row(["(784 - 515 - 319 - 127 - 10) Architecture","98.58%"])
x.add_row(["(784 - 600 - 567-428-385 -250- 10) ","98.61%"])

print(x)
```

```
+-----------------------------------------+--------+
|                   MOdel                 |  AUC   |
+-----------------------------------------+--------+
|     (784 - 488 - 256 - 10) Architecture   | 98.19% |
| (784 - 515 - 319 - 127 - 10) Architecture | 98.58% |
|    (784 - 600 - 567-428-385 -250- 10)     | 98.61% |
+-----------------------------------------+--------+
```

Observation: 1) Accuracy increasing slightly as number of hidden layer increases. 2) As in graph of third architecture of number of 5 hidden layers its overtitting after 13 epochs.