In [0]:
```
import keras
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
import time
from keras.models import Sequential
from keras.layers import Dense, Activation
```

```
Using TensorFlow backend.
```

In [0]:
```
!pip install PTable
```

```
Requirement already satisfied: PTable in /usr/local/lib/python3.6/dist-packages (0.9.2)
```

In [0]:
```
from prettytable import PrettyTable
table = PrettyTable()
table.field_names = ["model_name", "Test Accuracy", "no_of_activation_units", "Droput rate of each Hidden Layer", "incl
ude/Not include BatchNorm - each hidden Layer"]
```

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

In [0]:
```
print("Number of training examples :", X_train.shape[0], "and each image is of shape (%d, %d)"%(X_train.shape[1], X_tra
in.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 [0]:
```
# An example data point
print(X_train[632])
```

```
[[   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0  15  18  18  32
   136 166 255 106   2   0   0   0   0   0]
 [   0   0   0  28 137  36  36   7   0   0   4  36  36  72 237 253 253 253
   253 253 253 253  73   0   0   0   0   0]
 [   0   0   0 216 253 253 253 186 171 171 179 253 253 253 253 253 253 253
   253 253 253 253  17   0   0   0   0   0]
 [   0   0   0 133 253 253 253 253 253 253 253 253 253 253 253 188 177  65
   100 253 253 225  12   0   0   0   0   0]
 [   0   0   0  12 164 225 253 253 253 253 253 253 191 128  47   5   0   0
   144 253 253  57   0   0   0   0   0   0]
 [   0   0   0   0   0  20  29  71  98  29  29  29   9   0   0   0   0  13
   202 253 253  35   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1  91
   253 253 213  18   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  79 253
   253 217  76   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 205 253
   253 169   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  79 252 253
   220  21   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 204 253 253
   115   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0  36 239 253 242
    55   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0 202 253 248  70
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0  40 234 253 180   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0  91 253 253  55   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0  39 233 253 180   2   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0  60 253 253 111   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0  14 178 253 222  16   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0 144 253 253  57   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0  12 203 253 217  22   0   0   0
     0   0   0   0   0   0   0   0   0   0]]
```

In [0]:
```
#converting 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 [0]:
```
#lets try to normalize the data
# X => (X - Xmin)/(Xmax-Xmin) = X/255

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

In [0]:
```
# 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]

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]:
```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

In [0]:
```
def buildModel(no_hidden_layers,neuron_in_each_layer,dropouts=None,batchNormalization=None):
  if len(neuron_in_each_layer) == no_hidden_layers:
    output_dim = 10
    input_dim = 784
    model = Sequential()
    model.add(Dense(neuron_in_each_layer[0],activation="relu",input_shape=(input_dim,),kernel_initializer='he_normal'))
    model.add(keras.layers.Dropout(rate=dropouts[0]))
    if batchNormalization[0]:
      model.add(keras.layers.BatchNormalization())
    for i in range(1,no_hidden_layers):
      model.add(Dense(neuron_in_each_layer[i],activation="relu",kernel_initializer='he_normal'))
      model.add(keras.layers.Dropout(rate=dropouts[i]))
      if batchNormalization[i]:
        model.add(keras.layers.BatchNormalization())
    model.add(Dense(output_dim, activation='softmax'))
    return model
  else:
    print("please enter correct values: no of neuron in each layer should match with no of hidden layer")
```

In [0]:
```
model_with_2_layer = buildModel(2,(512,128),(0.3,0.5),(0,1))
model_with_2_layer.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:66: The name tf.ge
t_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:541: The name tf.p
laceholder is deprecated. Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:4479: The name tf.
truncated_normal is deprecated. Please 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.p
laceholder_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 drop
out (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. Please use tf.random.uniform instead.

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               65664
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
batch_normalization_1 (Batch (None, 128)               512
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 469,386
Trainable params: 469,130
Non-trainable params: 256
_________________________________________________________________
```

In [0]:
```
model_with_2_layer.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_with_2_layer.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=
(X_test, Y_test))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:793: The name tf.train.Optimizer i
s deprecated. Please use tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3576: The name tf.
log is deprecated. Please use tf.math.log instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops/math_grad.py:1424: where (f
rom 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 use 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 tf.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.g
et_default_session is deprecated. Please 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.C
onfigProto is deprecated. Please use tf.compat.v1.ConfigProto instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:207: The name tf.g
lobal_variables is deprecated. Please 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.i
s_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.v
ariables_initializer is deprecated. Please use tf.compat.v1.variables_initializer instead.

60000/60000 [==============================] - 5s 88us/step - loss: 0.3984 - acc: 0.8822 - val_loss: 0.1269 - val_ac
c: 0.9608
Epoch 2/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.1686 - acc: 0.9508 - val_loss: 0.1044 - val_ac
c: 0.9668
Epoch 3/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.1257 - acc: 0.9634 - val_loss: 0.0785 - val_ac
c: 0.9767
Epoch 4/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.1029 - acc: 0.9690 - val_loss: 0.0768 - val_ac
c: 0.9763
Epoch 5/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0885 - acc: 0.9737 - val_loss: 0.0699 - val_ac
c: 0.9795
Epoch 6/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0755 - acc: 0.9772 - val_loss: 0.0686 - val_ac
c: 0.9791
Epoch 7/20
60000/60000 [==============================] - 4s 69us/step - loss: 0.0684 - acc: 0.9794 - val_loss: 0.0725 - val_ac
c: 0.9798
Epoch 8/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0612 - acc: 0.9816 - val_loss: 0.0627 - val_ac
c: 0.9810
Epoch 9/20
60000/60000 [==============================] - 4s 65us/step - loss: 0.0548 - acc: 0.9832 - val_loss: 0.0640 - val_ac
c: 0.9815
Epoch 10/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0501 - acc: 0.9846 - val_loss: 0.0592 - val_ac
c: 0.9822
Epoch 11/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0468 - acc: 0.9855 - val_loss: 0.0615 - val_ac
c: 0.9844
Epoch 12/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0438 - acc: 0.9865 - val_loss: 0.0650 - val_ac
c: 0.9818
Epoch 13/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0423 - acc: 0.9866 - val_loss: 0.0636 - val_ac
c: 0.9828
Epoch 14/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0377 - acc: 0.9878 - val_loss: 0.0649 - val_ac
c: 0.9813
Epoch 15/20
60000/60000 [==============================] - 4s 66us/step - loss: 0.0352 - acc: 0.9888 - val_loss: 0.0588 - val_ac
c: 0.9841
Epoch 16/20
60000/60000 [==============================] - 4s 70us/step - loss: 0.0352 - acc: 0.9889 - val_loss: 0.0610 - val_ac
c: 0.9827
Epoch 17/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0322 - acc: 0.9896 - val_loss: 0.0604 - val_ac
c: 0.9826
Epoch 18/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0328 - acc: 0.9898 - val_loss: 0.0631 - val_ac
c: 0.9843
Epoch 19/20
60000/60000 [==============================] - 4s 68us/step - loss: 0.0290 - acc: 0.9907 - val_loss: 0.0603 - val_ac
c: 0.9834
Epoch 20/20
60000/60000 [==============================] - 4s 67us/step - loss: 0.0294 - acc: 0.9908 - val_loss: 0.0674 - val_ac
c: 0.9832
```

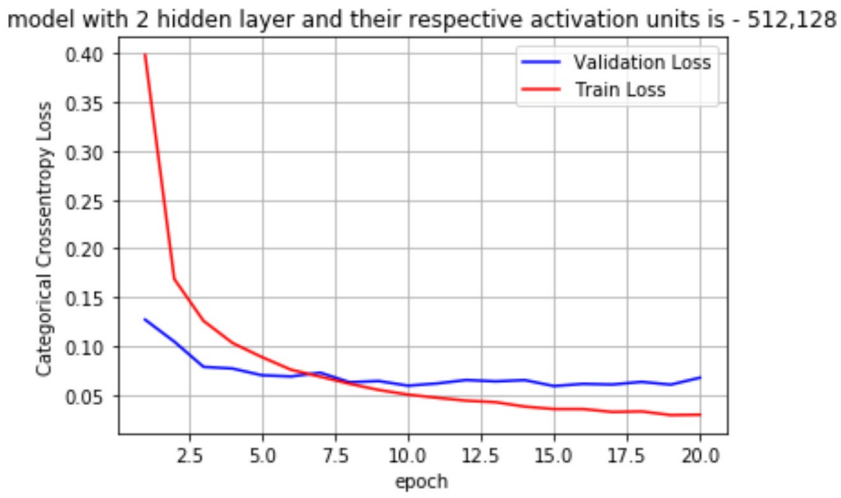
In [0]:
```
score = model_with_2_layer.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax = plt.subplots(1,1)
ax.set_title("model with 2 hidden layer and their respective activation units is - 512,128")
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)
table.add_row(["model_with_2_layer",score[1],"(512,128)","(0.3,0.5)", "(0,1)"])
```

```
Test score: 0.06735987591733755
Test accuracy: 0.9832
```



In [0]:
```
model_with_3_layer = buildModel(3,(512,256,128),(0,0.2,0.5),(1,1,1))
model_with_3_layer.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 512)               401920
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
batch_normalization_2 (Batch (None, 512)               2048
_________________________________________________________________
dense_5 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
batch_normalization_3 (Batch (None, 256)               1024
_________________________________________________________________
dense_6 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1290
=================================================================
Total params: 571,018
Trainable params: 569,226
Non-trainable params: 1,792
_________________________________________________________________
```

In [0]:
```
model_with_3_layer.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_with_3_layer.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 [==============================] - 7s 120us/step - loss: 0.3263 - acc: 0.9039 - val_loss: 0.1204 - val_ac
c: 0.9634
Epoch 2/20
60000/60000 [==============================] - 6s 104us/step - loss: 0.1267 - acc: 0.9637 - val_loss: 0.0977 - val_ac
c: 0.9707
Epoch 3/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0932 - acc: 0.9726 - val_loss: 0.0820 - val_ac
c: 0.9759
Epoch 4/20
60000/60000 [==============================] - 6s 104us/step - loss: 0.0694 - acc: 0.9795 - val_loss: 0.0841 - val_ac
c: 0.9742
Epoch 5/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.0573 - acc: 0.9828 - val_loss: 0.0782 - val_ac
c: 0.9782
Epoch 6/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0483 - acc: 0.9853 - val_loss: 0.0695 - val_ac
c: 0.9795
Epoch 7/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.0406 - acc: 0.9881 - val_loss: 0.0754 - val_ac
c: 0.9776
Epoch 8/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0380 - acc: 0.9884 - val_loss: 0.0709 - val_ac
c: 0.9787
Epoch 9/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.0342 - acc: 0.9895 - val_loss: 0.0720 - val_ac
c: 0.9796
Epoch 10/20
60000/60000 [==============================] - 6s 104us/step - loss: 0.0310 - acc: 0.9908 - val_loss: 0.0746 - val_ac
c: 0.9798
Epoch 11/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0254 - acc: 0.9924 - val_loss: 0.0748 - val_ac
c: 0.9803
Epoch 12/20
60000/60000 [==============================] - 6s 107us/step - loss: 0.0245 - acc: 0.9922 - val_loss: 0.0721 - val_ac
c: 0.9821
Epoch 13/20
60000/60000 [==============================] - 6s 103us/step - loss: 0.0213 - acc: 0.9930 - val_loss: 0.0789 - val_ac
c: 0.9808
Epoch 14/20
60000/60000 [==============================] - 6s 103us/step - loss: 0.0232 - acc: 0.9929 - val_loss: 0.0763 - val_ac
c: 0.9815
Epoch 15/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.0199 - acc: 0.9940 - val_loss: 0.0717 - val_ac
c: 0.9819
Epoch 16/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0211 - acc: 0.9934 - val_loss: 0.0750 - val_ac
c: 0.9805
Epoch 17/20
60000/60000 [==============================] - 6s 103us/step - loss: 0.0177 - acc: 0.9943 - val_loss: 0.0750 - val_ac
c: 0.9825
Epoch 18/20
60000/60000 [==============================] - 6s 106us/step - loss: 0.0170 - acc: 0.9945 - val_loss: 0.0876 - val_ac
c: 0.9786
Epoch 19/20
60000/60000 [==============================] - 6s 105us/step - loss: 0.0142 - acc: 0.9956 - val_loss: 0.0870 - val_ac
c: 0.9790
Epoch 20/20
60000/60000 [==============================] - 6s 104us/step - loss: 0.0142 - acc: 0.9955 - val_loss: 0.1051 - val_ac
c: 0.9778
```

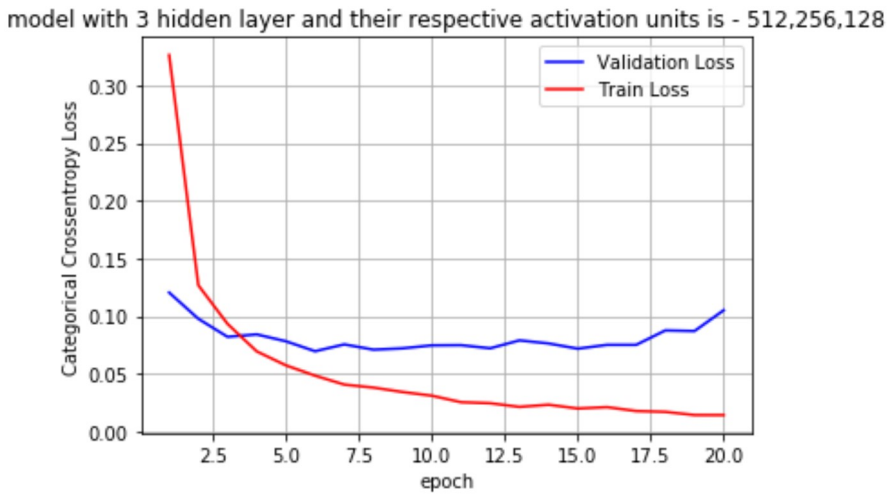
In [0]:
```
score = model_with_3_layer.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax = plt.subplots(1,1)
ax.set_title("model with 3 hidden layer and their respective activation units is - 512,256,128")
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)
table.add_row(["model_with_3_layer",score[1],"(512,256,128)","(0,0.2,0.5)", "(1,1,1)"])
```

```
Test score: 0.10507028651929867
Test accuracy: 0.9778
```



In [0]:
```
model_with_5_layer = buildModel(5,(600,300,150,75,32),(0.5,0,0,0.2,0.1),(0,0,1,1,1))
model_with_5_layer.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 600)               471000
_________________________________________________________________
dropout_6 (Dropout)          (None, 600)               0
_________________________________________________________________
dense_9 (Dense)              (None, 300)               180300
_________________________________________________________________
dropout_7 (Dropout)          (None, 300)               0
_________________________________________________________________
dense_10 (Dense)             (None, 150)               45150
_________________________________________________________________
dropout_8 (Dropout)          (None, 150)               0
_________________________________________________________________
batch_normalization_5 (Batch (None, 150)               600
_________________________________________________________________
dense_11 (Dense)             (None, 75)                11325
_________________________________________________________________
dropout_9 (Dropout)          (None, 75)                0
_________________________________________________________________
batch_normalization_6 (Batch (None, 75)                300
_________________________________________________________________
dense_12 (Dense)             (None, 32)                2432
_________________________________________________________________
dropout_10 (Dropout)         (None, 32)                0
_________________________________________________________________
batch_normalization_7 (Batch (None, 32)                128
_________________________________________________________________
dense_13 (Dense)             (None, 10)                330
=================================================================
Total params: 711,565
Trainable params: 711,051
Non-trainable params: 514
_________________________________________________________________
```

In [0]:
```
model_with_5_layer.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model_with_5_layer.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 [==============================] - 8s 141us/step - loss: 0.5587 - acc: 0.8350 - val_loss: 0.1534 - val_ac
c: 0.9538
Epoch 2/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.2174 - acc: 0.9381 - val_loss: 0.1150 - val_ac
c: 0.9668
Epoch 3/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.1679 - acc: 0.9522 - val_loss: 0.0934 - val_ac
c: 0.9723
Epoch 4/20
60000/60000 [==============================] - 8s 131us/step - loss: 0.1400 - acc: 0.9600 - val_loss: 0.0838 - val_ac
c: 0.9753
Epoch 5/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.1233 - acc: 0.9662 - val_loss: 0.0855 - val_ac
c: 0.9751
Epoch 6/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.1070 - acc: 0.9695 - val_loss: 0.0757 - val_ac
c: 0.9793
Epoch 7/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.1000 - acc: 0.9711 - val_loss: 0.0791 - val_ac
c: 0.9784
Epoch 8/20
60000/60000 [==============================] - 7s 123us/step - loss: 0.0915 - acc: 0.9741 - val_loss: 0.0730 - val_ac
c: 0.9788
Epoch 9/20
60000/60000 [==============================] - 7s 122us/step - loss: 0.0841 - acc: 0.9765 - val_loss: 0.0655 - val_ac
c: 0.9802
Epoch 10/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0803 - acc: 0.9770 - val_loss: 0.0713 - val_ac
c: 0.9796
Epoch 11/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0743 - acc: 0.9781 - val_loss: 0.0658 - val_ac
c: 0.9796
Epoch 12/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0720 - acc: 0.9792 - val_loss: 0.0610 - val_ac
c: 0.9819
Epoch 13/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0683 - acc: 0.9806 - val_loss: 0.0659 - val_ac
c: 0.9807
Epoch 14/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0668 - acc: 0.9806 - val_loss: 0.0641 - val_ac
c: 0.9823
Epoch 15/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0608 - acc: 0.9827 - val_loss: 0.0678 - val_ac
c: 0.9810
Epoch 16/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0581 - acc: 0.9830 - val_loss: 0.0628 - val_ac
c: 0.9833
Epoch 17/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0566 - acc: 0.9838 - val_loss: 0.0636 - val_ac
c: 0.9835
Epoch 18/20
60000/60000 [==============================] - 7s 119us/step - loss: 0.0521 - acc: 0.9853 - val_loss: 0.0601 - val_ac
c: 0.9836
Epoch 19/20
60000/60000 [==============================] - 7s 121us/step - loss: 0.0521 - acc: 0.9847 - val_loss: 0.0581 - val_ac
c: 0.9843
Epoch 20/20
60000/60000 [==============================] - 7s 120us/step - loss: 0.0467 - acc: 0.9860 - val_loss: 0.0633 - val_ac
c: 0.9836
```

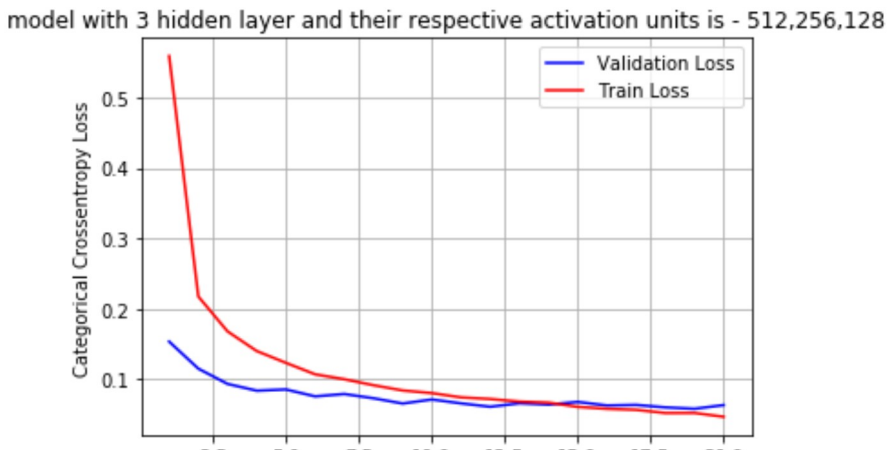
In [0]:
```
score = model_with_5_layer.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax = plt.subplots(1,1)
ax.set_title("model with 3 hidden layer and their respective activation units is - 512,256,128")
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)
#(600,300,150,75,32),(0.5,0,0,0.2,0.1),(0,0,1,1,1)
table.add_row(["model_with_5_layer",score[1],"(600,300,150,75,32)","(0.5,0,0,0.2,0.1)", "(0,0,1,1,1)"])
```

```
Test score: 0.06331695338832215
Test accuracy: 0.9836
```



In [0]:
```
print(table)
```

```
+--------------------+---------------+------------------------+----------------------------------+-------------------
---------------------------------+
|     model_name     | Test Accuracy | no_of_activation_units | Droput rate of each Hidden Layer | include/Not includ
e BatchNorm - each hidden Layer |
+--------------------+---------------+------------------------+----------------------------------+-------------------
---------------------------------+
| model_with_2_layer |     0.9832    |       (512,128)        |            (0.3,0.5)             |
(0,1)                           |
| model_with_3_layer |     0.9778    |     (512,256,128)      |           (0,0.2,0.5)            |
(1,1,1)                         |
| model_with_5_layer |     0.9836    |  (600,300,150,75,32)   |        (0.5,0,0,0.2,0.1)         |
(0,0,1,1,1)                        |
+--------------------+---------------+------------------------+----------------------------------+-------------------
---------------------------------+
```