In [1]:
```
# if you keras is not using tensorflow as backend set "KERAS_BACKEND=tensorflow"
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
```

```
Using TensorFlow backend.
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade (https://www.tensorflow.org/guide/migrate) now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: more info (https://colab.research.google.com/notebooks/tensorflow_version.ipynb).

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

In [4]:
```
print("Number of training examples :", X_train.shape[0], "and each image is of sl
print("Number of training examples :", X_test.shape[0], "and each image is of sha
```

```
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]:
```
# 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 sl
print("Number of training examples :", X_test.shape[0], "and each image is of sha
```

```
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 d
# 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.
```

In [10]:
```
# 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]:
```
from keras.models import Sequential
from keras.layers import Dense, Activation
```

In [0]:
```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

`Type-I` MLP_Relu_Adam_2Layer_1024_64

In [15]:
```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormaliz

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

model_drop = Sequential()

model_drop.add(Dense(1024, activation='relu', input_shape=(input_dim,), kernel_i
# model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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

# print(model_relu.summary())

model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoc

model_drop.summary()
```

```
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 d
eprecated 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/te
nsorflow_backend.py:1033: The name tf.assign_add is deprecated. Please use tf.c
ompat.v1.assign_add instead.

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

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/te
nsorflow_backend.py:3005: The name tf.Session is deprecated. Please use tf.comp
at.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/te
nsorflow_backend.py:190: The name tf.get_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/te
nsorflow_backend.py:197: The name tf.ConfigProto is deprecated. Please use tf.c
ompat.v1.ConfigProto instead.

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

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

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

60000/60000 [==============================] - 15s 248us/step - loss: 0.5077 -
acc: 0.8482 - val_loss: 0.1529 - val_acc: 0.9554
Epoch 2/20
60000/60000 [==============================] - 14s 233us/step - loss: 0.2440 -
acc: 0.9279 - val_loss: 0.1150 - val_acc: 0.9644
Epoch 3/20
60000/60000 [==============================] - 14s 234us/step - loss: 0.1868 -
acc: 0.9450 - val_loss: 0.0933 - val_acc: 0.9700
Epoch 4/20
60000/60000 [==============================] - 14s 230us/step - loss: 0.1617 -
acc: 0.9528 - val_loss: 0.0907 - val_acc: 0.9739
Epoch 5/20
60000/60000 [==============================] - 15s 243us/step - loss: 0.1422 -
acc: 0.9581 - val_loss: 0.0852 - val_acc: 0.9737
Epoch 6/20
60000/60000 [==============================] - 14s 234us/step - loss: 0.1256 -
acc: 0.9631 - val_loss: 0.0749 - val_acc: 0.9782
Epoch 7/20
60000/60000 [==============================] - 14s 241us/step - loss: 0.1128 -
acc: 0.9667 - val_loss: 0.0750 - val_acc: 0.9775
Epoch 8/20
60000/60000 [==============================] - 15s 251us/step - loss: 0.1051 -
acc: 0.9690 - val_loss: 0.0719 - val_acc: 0.9790
Epoch 9/20
60000/60000 [==============================] - 14s 231us/step - loss: 0.1005 -
acc: 0.9702 - val_loss: 0.0670 - val_acc: 0.9801
Epoch 10/20
60000/60000 [==============================] - 14s 228us/step - loss: 0.0908 -
acc: 0.9717 - val_loss: 0.0624 - val_acc: 0.9817
Epoch 11/20
60000/60000 [==============================] - 14s 231us/step - loss: 0.0869 -
acc: 0.9741 - val_loss: 0.0623 - val_acc: 0.9832
Epoch 12/20
60000/60000 [==============================] - 14s 231us/step - loss: 0.0758 -
acc: 0.9771 - val_loss: 0.0609 - val_acc: 0.9816
Epoch 13/20
60000/60000 [==============================] - 14s 230us/step - loss: 0.0744 -
acc: 0.9770 - val_loss: 0.0618 - val_acc: 0.9829
Epoch 14/20
60000/60000 [==============================] - 14s 227us/step - loss: 0.0725 -
acc: 0.9778 - val_loss: 0.0604 - val_acc: 0.9830
Epoch 15/20
60000/60000 [==============================] - 14s 236us/step - loss: 0.0663 -
acc: 0.9801 - val_loss: 0.0585 - val_acc: 0.9835
Epoch 16/20
60000/60000 [==============================] - 14s 231us/step - loss: 0.0622 -
acc: 0.9809 - val_loss: 0.0609 - val_acc: 0.9828
Epoch 17/20
60000/60000 [==============================] - 14s 228us/step - loss: 0.0593 -
```

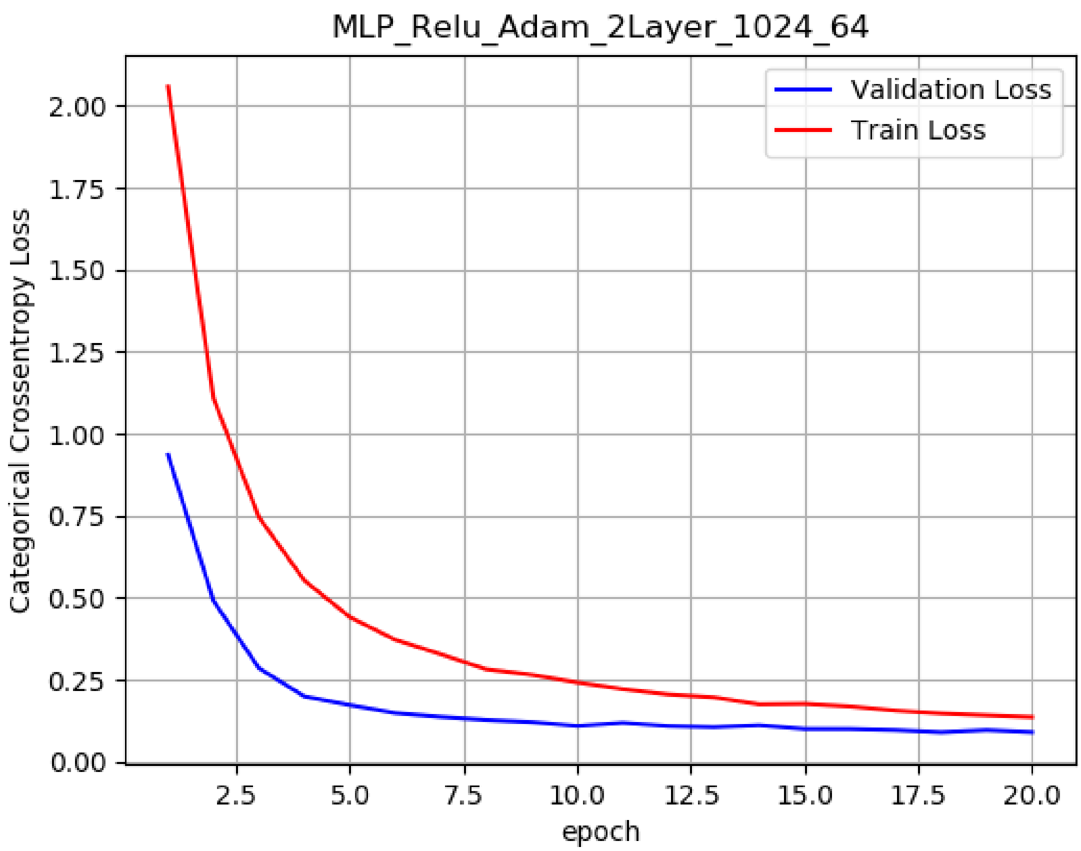
```
acc: 0.9823 - val_loss: 0.0595 - val_acc: 0.9845
Epoch 18/20
60000/60000 [==============================] - 14s 231us/step - loss: 0.0583 -
acc: 0.9823 - val_loss: 0.0631 - val_acc: 0.9832
Epoch 19/20
60000/60000 [==============================] - 14s 228us/step - loss: 0.0550 -
acc: 0.9831 - val_loss: 0.0607 - val_acc: 0.9835
Epoch 20/20
60000/60000 [==============================] - 13s 225us/step - loss: 0.0533 -
acc: 0.9834 - val_loss: 0.0570 - val_acc: 0.9845
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 1024)              803840
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_7 (Dense)              (None, 64)                65600
_________________________________________________________________
batch_normalization_2 (Batch (None, 64)                256
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_8 (Dense)              (None, 10)                650
=================================================================
Total params: 870,346
Trainable params: 870,218
Non-trainable params: 128
_________________________________________________________________
```

Accuracy

In [51]:
```
score = model_drop.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_epo

# we will get val_loss and val_acc only when you pass the paramter validation_dat
# 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 numbe


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

```
Test score: 0.09080233381008729
Test accuracy: 0.9804

<IPython.core.display.Javascript object>


[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
[0.9364621857643127, 0.49075753841400144, 0.2855819786310196, 0.199641513609886
16, 0.17320310388207436, 0.14879324183166026, 0.1371548559397459, 0.12790572729
706765, 0.12123643999770284, 0.10966003392711282, 0.11899310316890478, 0.109499
49248600752, 0.10622592614218593, 0.11184748293552547, 0.10049898426607251, 0.1
0034355153460056, 0.09705943623417988, 0.09053028036076576, 0.0970813074545003
5, 0.09080233482245821]
[2.058573489634196, 1.1087739948590596, 0.7454398405392965, 0.5526444519678751,
0.44117149138450623, 0.37196457761128743, 0.32815683867136636, 0.28217501312891
64, 0.26549448329607644, 0.24212648111979165, 0.22209778400262198, 0.2057082060
8933768, 0.1968686815738678, 0.1755473879337311, 0.1770206745068232, 0.16906047
227780024, 0.15631902202765147, 0.14764749088684717, 0.14192116012970607, 0.136
5976075251897]
```

In [3]:
```
%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
import time
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ;
ax.set_ylabel('Categorical Crossentropy Loss')
ax.set_title(label="MLP_Relu_Adam_2Layer_1024_64")
plt_dynamic(x, vy, ty, ax)
```



Type-II MLP_Relu_Adam_3Layer_512_128_64

In [25]:
```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormaliz

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_in
# model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mea
# model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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

# print(model_relu.summary())

model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoc

model_drop.summary()
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s 162us/step - loss: 1.0693 -
acc: 0.6588 - val_loss: 0.2757 - val_acc: 0.9233
Epoch 2/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.4742 - a
cc: 0.8602 - val_loss: 0.1845 - val_acc: 0.9446
Epoch 3/20
60000/60000 [==============================] - 9s 150us/step - loss: 0.3542 - a
cc: 0.8994 - val_loss: 0.1449 - val_acc: 0.9573
Epoch 4/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.2920 - a
cc: 0.9179 - val_loss: 0.1317 - val_acc: 0.9612
Epoch 5/20
60000/60000 [==============================] - 9s 143us/step - loss: 0.2549 - a
cc: 0.9285 - val_loss: 0.1135 - val_acc: 0.9672
Epoch 6/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.2297 - a
cc: 0.9379 - val_loss: 0.1090 - val_acc: 0.9682
Epoch 7/20
60000/60000 [==============================] - 9s 142us/step - loss: 0.2101 - a
cc: 0.9415 - val_loss: 0.1017 - val_acc: 0.9728
Epoch 8/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.1892 - a
cc: 0.9485 - val_loss: 0.1001 - val_acc: 0.9720
Epoch 9/20
60000/60000 [==============================] - 10s 161us/step - loss: 0.1752 -
acc: 0.9520 - val_loss: 0.0922 - val_acc: 0.9747
```

```
Epoch 10/20
60000/60000 [==============================] - 9s 151us/step - loss: 0.1650 - a
cc: 0.9553 - val_loss: 0.0913 - val_acc: 0.9747
Epoch 11/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.1473 - a
cc: 0.9586 - val_loss: 0.0832 - val_acc: 0.9761
Epoch 12/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.1492 - a
cc: 0.9597 - val_loss: 0.0862 - val_acc: 0.9767
Epoch 13/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.1375 - a
cc: 0.9625 - val_loss: 0.0830 - val_acc: 0.9771
Epoch 14/20
60000/60000 [==============================] - 9s 142us/step - loss: 0.1322 - a
cc: 0.9637 - val_loss: 0.0860 - val_acc: 0.9757
Epoch 15/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.1264 - a
cc: 0.9656 - val_loss: 0.0805 - val_acc: 0.9789
Epoch 16/20
60000/60000 [==============================] - 9s 147us/step - loss: 0.1196 - a
cc: 0.9669 - val_loss: 0.0805 - val_acc: 0.9791
Epoch 17/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.1127 - a
cc: 0.9698 - val_loss: 0.0819 - val_acc: 0.9781
Epoch 18/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.1100 - a
cc: 0.9700 - val_loss: 0.0820 - val_acc: 0.9785
Epoch 19/20
60000/60000 [==============================] - 9s 144us/step - loss: 0.1027 - a
cc: 0.9712 - val_loss: 0.0817 - val_acc: 0.9798
Epoch 20/20
60000/60000 [==============================] - 8s 142us/step - loss: 0.1001 - a
cc: 0.9721 - val_loss: 0.0781 - val_acc: 0.9794
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_13 (Dense)             (None, 512)               401920    
_________________________________________________________________
dropout_9 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_14 (Dense)             (None, 128)               65664     
_________________________________________________________________
dropout_10 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_15 (Dense)             (None, 64)                8256      
_________________________________________________________________
batch_normalization_4 (Batch (None, 64)                256       
_________________________________________________________________
dropout_11 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_16 (Dense)             (None, 10)                650       
=================================================================
Total params: 476,746
Trainable params: 476,618
Non-trainable params: 128
_________________________________________________________________
```

Accuracy

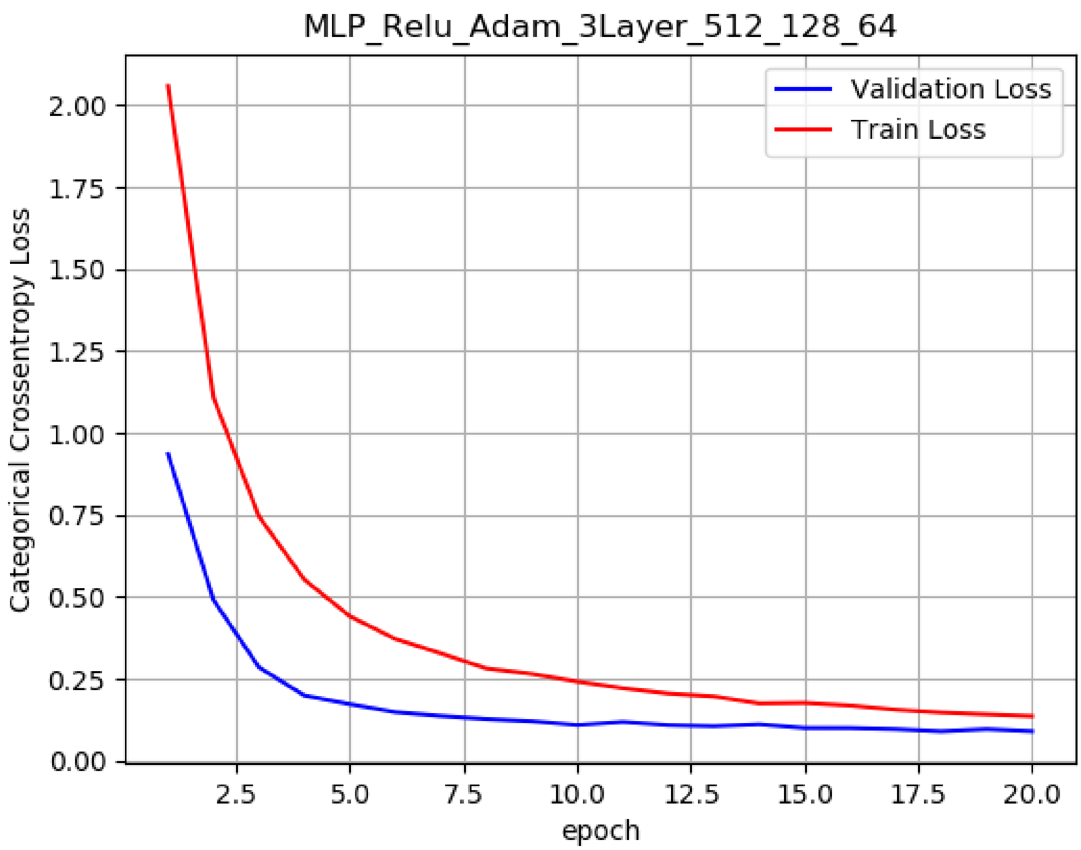
In [52]:
```
score = model_drop.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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_epo

# we will get val_loss and val_acc only when you pass the paramter validation_dat
# 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


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

```
Test score: 0.09080233381008729
Test accuracy: 0.9804
```

```
<IPython.core.display.Javascript object>
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
[0.9364621857643127, 0.49075753841400144, 0.2855819786310196, 0.199641513609886
16, 0.17320310388207436, 0.14879324183166026, 0.1371548559397459, 0.12790572729
706765, 0.12123643999770284, 0.10966003392711282, 0.11899310316890478, 0.109499
49248600752, 0.10622592614218593, 0.11184748293552547, 0.10049898426607251, 0.1
0034355153460056, 0.09705943623417988, 0.09053028036076576, 0.0970813074545003
5, 0.09080233482245821]
[2.058573489634196, 1.1087739948590596, 0.7454398405392965, 0.5526444519678751,
0.44117149138450623, 0.37196457761128743, 0.32815683867136636, 0.28217501312891
64, 0.26549448329607644, 0.24212648111979165, 0.22209778400262198, 0.2057082060
8933768, 0.1968686815738678, 0.1755473879337311, 0.1770206745068232, 0.16906047
227780024, 0.15631902202765147, 0.14764749088684717, 0.14192116012970607, 0.136
5976075251897]
```

In [5]:
```
%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
import time
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ;
ax.set_ylabel('Categorical Crossentropy Loss')
ax.set_title(label="MLP_Relu_Adam_3Layer_512_128_64")
plt_dynamic(x, vy, ty, ax)
```



Type-III MLP_Relu_Adam_5Layer_1024_512_128_64_32

In [28]:
```
# https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormaliz

from keras.layers import Dropout

model_drop = Sequential()

model_drop.add(Dense(1024, activation='relu', input_shape=(input_dim,), kernel_i
# model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(512, activation='relu', kernel_initializer=RandomNormal(mea
# model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mea
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(64, activation='relu', kernel_initializer=RandomNormal(mean
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

model_drop.add(Dense(32, activation='relu', kernel_initializer=RandomNormal(mean
model_drop.add(BatchNormalization())
model_drop.add(Dropout(0.5))

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

# print(model_relu.summary())

model_drop.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['

history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoc

model_drop.summary()
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 25s 415us/step - loss: 2.0586 -
acc: 0.3012 - val_loss: 0.9365 - val_acc: 0.7289
Epoch 2/20
60000/60000 [==============================] - 23s 391us/step - loss: 1.1088 -
acc: 0.6104 - val_loss: 0.4908 - val_acc: 0.8497
Epoch 3/20
60000/60000 [==============================] - 23s 384us/step - loss: 0.7454 -
acc: 0.7522 - val_loss: 0.2856 - val_acc: 0.9286
Epoch 4/20
60000/60000 [==============================] - 23s 385us/step - loss: 0.5526 -
acc: 0.8320 - val_loss: 0.1996 - val_acc: 0.9486
Epoch 5/20
60000/60000 [==============================] - 24s 394us/step - loss: 0.4412 -
acc: 0.8744 - val_loss: 0.1732 - val_acc: 0.9566
Epoch 6/20
60000/60000 [==============================] - 23s 380us/step - loss: 0.3720 -
acc: 0.8974 - val_loss: 0.1488 - val_acc: 0.9605
```

```
Epoch 7/20
60000/60000 [==============================] - 24s 395us/step - loss: 0.3282 -
acc: 0.9149 - val_loss: 0.1372 - val_acc: 0.9650
Epoch 8/20
60000/60000 [==============================] - 23s 378us/step - loss: 0.2822 -
acc: 0.9281 - val_loss: 0.1279 - val_acc: 0.9671
Epoch 9/20
60000/60000 [==============================] - 23s 387us/step - loss: 0.2655 -
acc: 0.9334 - val_loss: 0.1212 - val_acc: 0.9700
Epoch 10/20
60000/60000 [==============================] - 23s 376us/step - loss: 0.2421 -
acc: 0.9406 - val_loss: 0.1097 - val_acc: 0.9722
Epoch 11/20
60000/60000 [==============================] - 23s 387us/step - loss: 0.2221 -
acc: 0.9461 - val_loss: 0.1190 - val_acc: 0.9708
Epoch 12/20
60000/60000 [==============================] - 24s 400us/step - loss: 0.2057 -
acc: 0.9511 - val_loss: 0.1095 - val_acc: 0.9740
Epoch 13/20
60000/60000 [==============================] - 24s 395us/step - loss: 0.1969 -
acc: 0.9532 - val_loss: 0.1062 - val_acc: 0.9739
Epoch 14/20
60000/60000 [==============================] - 24s 393us/step - loss: 0.1755 -
acc: 0.9572 - val_loss: 0.1118 - val_acc: 0.9740
Epoch 15/20
60000/60000 [==============================] - 22s 375us/step - loss: 0.1770 -
acc: 0.9590 - val_loss: 0.1005 - val_acc: 0.9766
Epoch 16/20
60000/60000 [==============================] - 23s 384us/step - loss: 0.1691 -
acc: 0.9609 - val_loss: 0.1003 - val_acc: 0.9775
Epoch 17/20
60000/60000 [==============================] - 24s 403us/step - loss: 0.1563 -
acc: 0.9635 - val_loss: 0.0971 - val_acc: 0.9778
Epoch 18/20
60000/60000 [==============================] - 23s 385us/step - loss: 0.1476 -
acc: 0.9653 - val_loss: 0.0905 - val_acc: 0.9785
Epoch 19/20
60000/60000 [==============================] - 23s 381us/step - loss: 0.1419 -
acc: 0.9676 - val_loss: 0.0971 - val_acc: 0.9797
Epoch 20/20
60000/60000 [==============================] - 23s 390us/step - loss: 0.1366 -
acc: 0.9684 - val_loss: 0.0908 - val_acc: 0.9804
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_17 (Dense)             (None, 1024)              803840
_________________________________________________________________
dropout_12 (Dropout)         (None, 1024)              0
_________________________________________________________________
dense_18 (Dense)             (None, 512)               524800
_________________________________________________________________
dropout_13 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_19 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_5 (Batch (None, 128)               512
```

```
_________________________________________________________________
dropout_14 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_20 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_6 (Batch (None, 64)                256
_________________________________________________________________
dropout_15 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_21 (Dense)             (None, 32)                2080
_________________________________________________________________
batch_normalization_7 (Batch (None, 32)                128
_________________________________________________________________
dropout_16 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_22 (Dense)             (None, 10)                330
=================================================================
Total params: 1,405,866
Trainable params: 1,405,418
Non-trainable params: 448
_________________________________________________________________
```

Accuracy

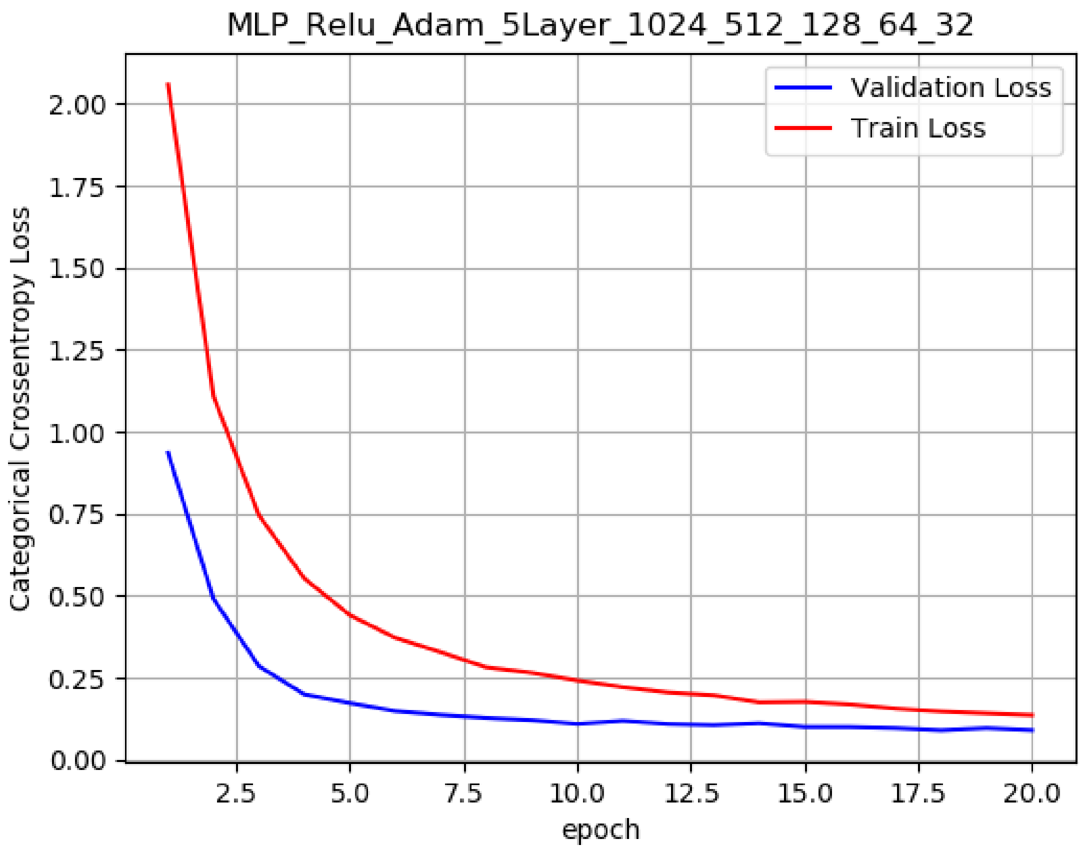
In [53]:
```
score = model_drop.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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_epo

# we will get val_loss and val_acc only when you pass the paramter validation_dat
# 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


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

```
Test score: 0.09080233381008729
Test accuracy: 0.9804
```

```
<IPython.core.display.Javascript object>
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
[0.9364621857643127, 0.49075753841400144, 0.2855819786310196, 0.199641513609886
16, 0.17320310388207436, 0.14879324183166026, 0.1371548559397459, 0.12790572729
706765, 0.12123643999770284, 0.10966003392711282, 0.11899310316890478, 0.109499
49248600752, 0.10622592614218593, 0.11184748293552547, 0.10049898426607251, 0.1
0034355153460056, 0.09705943623417988, 0.09053028036076576, 0.0970813074545003
5, 0.09080233482245821]
[2.058573489634196, 1.1087739948590596, 0.7454398405392965, 0.5526444519678751,
0.44117149138450623, 0.37196457761128743, 0.32815683867136636, 0.28217501312891
64, 0.26549448329607644, 0.24212648111979165, 0.22209778400262198, 0.2057082060
8933768, 0.1968686815738678, 0.1755473879337311, 0.1770206745068232, 0.16906047
227780024, 0.15631902202765147, 0.14764749088684717, 0.14192116012970607, 0.136
5976075251897]
```

In [7]:
```
%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
import time
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ;
ax.set_ylabel('Categorical Crossentropy Loss')
ax.set_title(label="MLP_Relu_Adam_5Layer_1024_512_128_64_32")
plt_dynamic(x, vy, ty, ax)
```



In [ ]: