HW4

1-Multi-Layer Perceptron

In [1]:

```
# import tensorflow as tf
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
import numpy as np
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/pytho
n/compat/v2_compat.py:96: disable_resource_variables (from tensorflow.python.op
s.variable_scope) is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
```

In [2]:

```
# tf.test.is_gpu_available(
#     cuda_only=False, min_cuda_compute_capability=None
# )
```

In [3]:

```
#Load the data
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

In [4]:

```
#define class names
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']
```

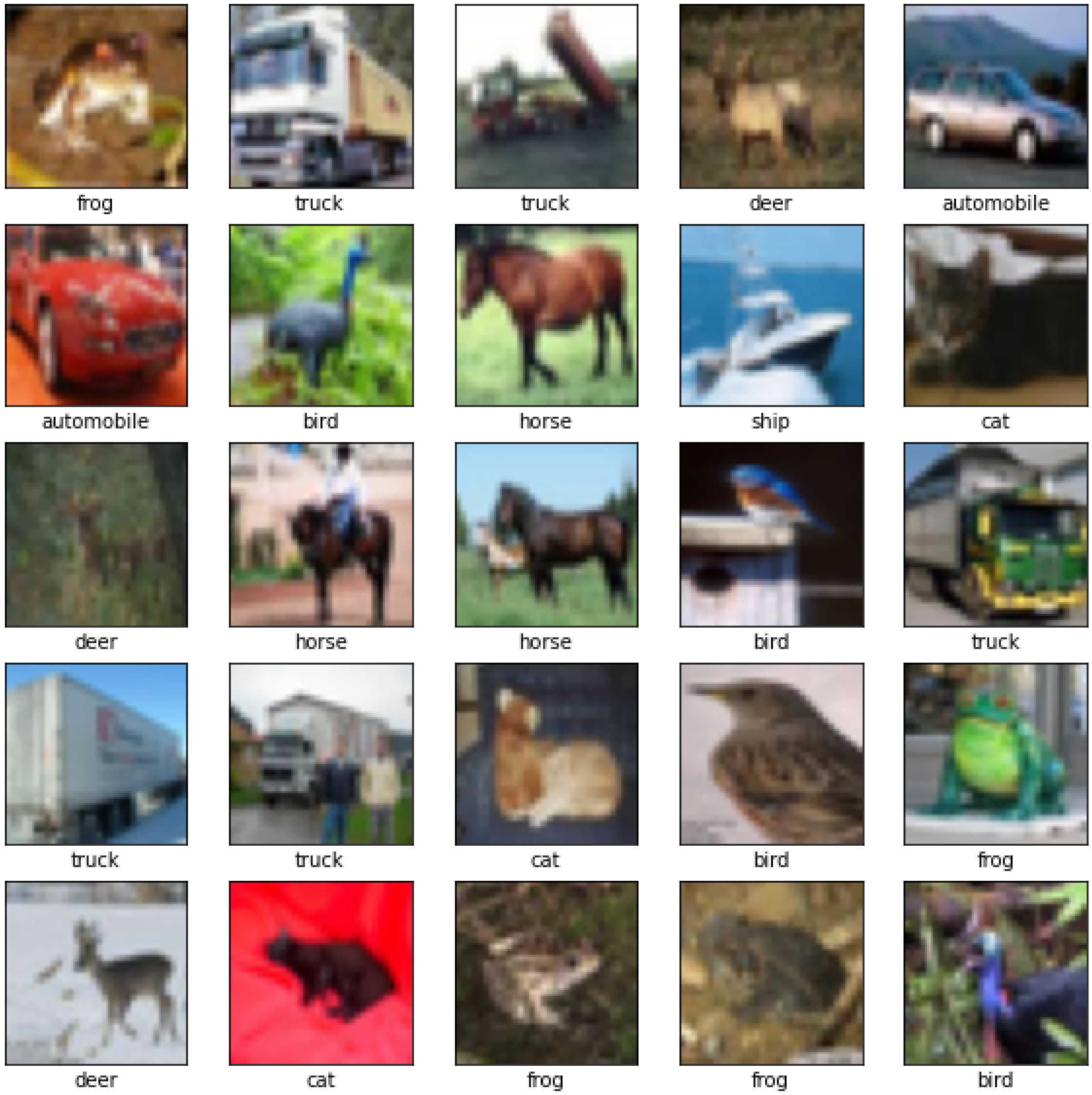
In [5]:

```
print('Train', x_train.shape, y_train.shape)
print('Test', (x_test.shape, y_test.shape))
# normalize pixel values
x_train, x_test = x_train/255, x_test/255
```

```
Train (50000, 32, 32, 3) (50000, 1)
Test ((10000, 32, 32, 3), (10000, 1))
```

In [6]:

```
#Plot some of the images
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train[i], cmap=plt.cm.binary)
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[y_train[i][0]])
plt.show()
```



In [7]:

```
#flatting the training data to (50000*3072)
#where 3072 = 32*32*3
# u = x_train.reshape(-1, 3072)
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1]*x_train.shape[2]*x_
x_test = x_test.reshape(x_test.shape[0], x_test.shape[1]*x_test.shape[2]*x_test.
```

In [8]:

```
#One Hot Encode with Keras for the labels
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

In [9]:

```
#Randomize the data
def  shuffle_data(x, y):
    permutation = np.random.permutation(x.shape[0])
    shuffled_x, shuffled_y  = x[permutation], y[permutation]
    return shuffled_x, shuffled_y

def get_next_batch(x, y, start, end):
    x_batch, y_batch = x[start:end], y[start:end]
    return x_batch, y_batch
```

In [10]:
```
#Function for creating layers
def layer(x, num_units, name, use_relu=True):
    """
    Create a fully-connected layer
    :param x: input from previous layer
    :param num_units: number of hidden units in the fully-connected layer
    :param name: layer name
    :param use_relu: boolean to add ReLU non-linearity (or not)
    :return: The output array
    """
    in_dim = x.get_shape()[1]
    shape=[in_dim, num_units]

    W = tf.get_variable('W_' + name,
        dtype=tf.float32,
        shape=shape,
        initializer=tf.truncated_normal_initializer(stddev=0.01))

    b =tf.get_variable('b_' + name,
        dtype=tf.float32,
        initializer=tf.constant(0., shape=[num_units], dtype=tf.float32))

    layer = tf.matmul(x, W)
    layer += b
    if use_relu:
        layer = tf.nn.relu(layer)
    return layer, W
```

In [52]:
```
#Network configuration
h1 = 250 #250                # Number of nodes in the first hidden layer
h2 = 250 #250                   # Number of nodes in the second hidden layer
#input vector size
feature_vector_size = x_train.shape[1]
num_classes = len(class_names)
```

In [53]:
```
# Parameters
learning_rate = 0.001  # The optimization initial learning rate
epochs = 50            # Total number of training epochs
batch_size = 100       # Training batch size
display_freq = 100       # Frequency of displaying the training results
```

In [54]:
```
# Remove previous weights, bias, inputs, etc..
tf.reset_default_graph()
```

In [55]:
```
#creating the network
# Create the graph for the linear model
# Placeholders for inputs (x) and outputs(y)
x = tf.placeholder(tf.float32, shape=[None, feature_vector_size], name='X')
y = tf.placeholder(tf.float32, shape=[None, num_classes], name='Y')
```

In [56]:
```
#Create the network layers
layer_h1, hidden_weights_1 = layer(x, h1, 'h1', use_relu=True)
layer_h2, hidden_weights_2 = layer(layer_h1, h2, 'h2', use_relu=True)
output_logits, hidden_weights_out = layer(layer_h2, num_classes, 'OUT', use_relu
```

In [57]:

```
# Network predictions
cls_prediction = tf.argmax(output_logits, axis=1, name='predictions')

#the loss function, optimizer, accuracy, and predicted class

# Loss function with No Regularization
# loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y, logits

# Loss function using L2 Regularization
loss = (tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y, logits=
        0.001*tf.nn.l2_loss(hidden_weights_1) + \
        0.001*tf.nn.l2_loss(hidden_weights_2) + \
        0.001*tf.nn.l2_loss(hidden_weights_out))


optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate, name='Adam-op').
# optimizer = tf.train.GradientDescentOptimizer(learning_rate = learning_rate).m

correct_prediction = tf.equal(tf.argmax(output_logits, 1), tf.argmax(y, 1), name
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32), name='accurac
```

In [58]:

```
# Create the op for initializing all variables
init = tf.global_variables_initializer()
```

In [59]:

```
#train
sess = tf.InteractiveSession()
sess.run(init)
global_step = 0
# Number of training iterations in each epoch
num_tr_iter = int(len(y_train) / batch_size)
```

In [61]:

```
for epoch in range(epochs):
    print('Training epoch: {}'.format(epoch + 1))
    x_train, y_train = shuffle_data(x_train, y_train)
    for iteration in range(num_tr_iter):
        global_step += 1
        start = iteration * batch_size
        end = (iteration + 1) * batch_size
        x_batch, y_batch = get_next_batch(x_train, y_train, start, end)

        # Run optimization op (backprop)
        feed_dict_batch = {x: x_batch, y: y_batch}
        sess.run(optimizer, feed_dict=feed_dict_batch)

        if iteration % display_freq == 0:
            # Calculate and display the batch loss and accuracy
            loss_batch, acc_batch = sess.run([loss, accuracy],
                                             feed_dict=feed_dict_batch)

            print("iter {0:3d}:\t Loss={1:.2f},\tTraining Accuracy={2:.01%}".
                  format(iteration, loss_batch, acc_batch))

    # Run validation after every epoch
    feed_dict_valid = {x: x_test[:10000], y: y_test[:10000]}
    loss_valid, acc_valid = sess.run([loss, accuracy], feed_dict=feed_dict_valic
```

```
    print('-------------------------------------------------------')
    print("Epoch: {0}, validation loss: {1:.2f}, validation accuracy: {2:.01%}".
          format(epoch + 1, loss_valid, acc_valid))
    print('-------------------------------------------------------')
```

```
Training epoch: 1
iter   0:        Loss=1.47,     Training Accuracy=54.0%
iter 100:        Loss=1.27,     Training Accuracy=64.0%
iter 200:        Loss=1.53,     Training Accuracy=46.0%
iter 300:        Loss=1.30,     Training Accuracy=57.0%
iter 400:        Loss=1.43,     Training Accuracy=49.0%
---------------------------------------------------------
Epoch: 1, validation loss: 1.53, validation accuracy: 49.8%
---------------------------------------------------------
Training epoch: 2
iter   0:        Loss=1.60,     Training Accuracy=46.0%
iter 100:        Loss=1.37,     Training Accuracy=55.0%
iter 200:        Loss=1.46,     Training Accuracy=55.0%
iter 300:        Loss=1.33,     Training Accuracy=59.0%
iter 400:        Loss=1.42,     Training Accuracy=48.0%
---------------------------------------------------------
Epoch: 2, validation loss: 1.54, validation accuracy: 49.7%
---------------------------------------------------------
Training epoch: 3
iter   0:        Loss=1.35,     Training Accuracy=63.0%
iter 100:        Loss=1.32,     Training Accuracy=58.0%
iter 200:        Loss=1.15,     Training Accuracy=63.0%
iter 300:        Loss=1.38,     Training Accuracy=59.0%
iter 400:        Loss=1.39,     Training Accuracy=60.0%
---------------------------------------------------------
Epoch: 3, validation loss: 1.58, validation accuracy: 49.6%
---------------------------------------------------------
Training epoch: 4
iter   0:        Loss=1.26,     Training Accuracy=61.0%
iter 100:        Loss=1.29,     Training Accuracy=67.0%
iter 200:        Loss=1.22,     Training Accuracy=66.0%
iter 300:        Loss=1.38,     Training Accuracy=50.0%
iter 400:        Loss=1.34,     Training Accuracy=54.0%
---------------------------------------------------------
Epoch: 4, validation loss: 1.53, validation accuracy: 51.3%
---------------------------------------------------------
Training epoch: 5
iter   0:        Loss=1.27,     Training Accuracy=59.0%
iter 100:        Loss=1.28,     Training Accuracy=58.0%
iter 200:        Loss=1.36,     Training Accuracy=56.0%
iter 300:        Loss=1.21,     Training Accuracy=55.0%
iter 400:        Loss=1.50,     Training Accuracy=53.0%
---------------------------------------------------------
Epoch: 5, validation loss: 1.53, validation accuracy: 49.8%
---------------------------------------------------------
Training epoch: 6
iter   0:        Loss=1.58,     Training Accuracy=50.0%
iter 100:        Loss=1.56,     Training Accuracy=52.0%
iter 200:        Loss=1.26,     Training Accuracy=60.0%
iter 300:        Loss=1.52,     Training Accuracy=46.0%
iter 400:        Loss=1.67,     Training Accuracy=47.0%
---------------------------------------------------------
Epoch: 6, validation loss: 1.57, validation accuracy: 49.0%
---------------------------------------------------------
Training epoch: 7
iter   0:        Loss=1.38,     Training Accuracy=56.0%
iter 100:        Loss=1.34,     Training Accuracy=58.0%
iter 200:        Loss=1.19,     Training Accuracy=60.0%
```

```
iter 300:        Loss=1.28,     Training Accuracy=58.0%
iter 400:        Loss=1.62,     Training Accuracy=38.0%
---------------------------------------------------------
Epoch: 7, validation loss: 1.54, validation accuracy: 50.2%
---------------------------------------------------------
Training epoch: 8
iter   0:        Loss=1.44,     Training Accuracy=48.0%
iter 100:        Loss=1.19,     Training Accuracy=59.0%
iter 200:        Loss=1.35,     Training Accuracy=56.0%
iter 300:        Loss=1.36,     Training Accuracy=54.0%
iter 400:        Loss=1.50,     Training Accuracy=50.0%
---------------------------------------------------------
Epoch: 8, validation loss: 1.50, validation accuracy: 51.5%
---------------------------------------------------------
Training epoch: 9
iter   0:        Loss=1.34,     Training Accuracy=53.0%
iter 100:        Loss=1.34,     Training Accuracy=54.0%
iter 200:        Loss=1.16,     Training Accuracy=65.0%
iter 300:        Loss=1.54,     Training Accuracy=51.0%
iter 400:        Loss=1.29,     Training Accuracy=61.0%
---------------------------------------------------------
Epoch: 9, validation loss: 1.50, validation accuracy: 51.9%
---------------------------------------------------------
Training epoch: 10
iter   0:        Loss=1.25,     Training Accuracy=59.0%
iter 100:        Loss=1.28,     Training Accuracy=52.0%
iter 200:        Loss=1.54,     Training Accuracy=53.0%
iter 300:        Loss=1.46,     Training Accuracy=51.0%
iter 400:        Loss=1.31,     Training Accuracy=56.0%
---------------------------------------------------------
Epoch: 10, validation loss: 1.50, validation accuracy: 51.2%
---------------------------------------------------------
Training epoch: 11
iter   0:        Loss=1.03,     Training Accuracy=74.0%
iter 100:        Loss=1.31,     Training Accuracy=62.0%
iter 200:        Loss=1.50,     Training Accuracy=49.0%
iter 300:        Loss=1.24,     Training Accuracy=60.0%
iter 400:        Loss=1.39,     Training Accuracy=50.0%
---------------------------------------------------------
Epoch: 11, validation loss: 1.56, validation accuracy: 49.9%
---------------------------------------------------------
Training epoch: 12
iter   0:        Loss=1.53,     Training Accuracy=51.0%
iter 100:        Loss=1.32,     Training Accuracy=60.0%
iter 200:        Loss=1.52,     Training Accuracy=49.0%
iter 300:        Loss=1.22,     Training Accuracy=58.0%
iter 400:        Loss=1.39,     Training Accuracy=52.0%
---------------------------------------------------------
Epoch: 12, validation loss: 1.54, validation accuracy: 50.5%
---------------------------------------------------------
Training epoch: 13
iter   0:        Loss=1.58,     Training Accuracy=49.0%
iter 100:        Loss=1.25,     Training Accuracy=59.0%
iter 200:        Loss=1.22,     Training Accuracy=61.0%
iter 300:        Loss=1.38,     Training Accuracy=57.0%
iter 400:        Loss=1.50,     Training Accuracy=48.0%
---------------------------------------------------------
Epoch: 13, validation loss: 1.53, validation accuracy: 50.9%
---------------------------------------------------------
Training epoch: 14
iter   0:        Loss=1.54,     Training Accuracy=52.0%
iter 100:        Loss=1.24,     Training Accuracy=60.0%
iter 200:        Loss=1.40,     Training Accuracy=53.0%
iter 300:        Loss=1.48,     Training Accuracy=56.0%
iter 400:        Loss=1.26,     Training Accuracy=62.0%
```

```
------------------------------------------------------------
Epoch: 14, validation loss: 1.57, validation accuracy: 48.9%
------------------------------------------------------------
Training epoch: 15
iter   0:        Loss=1.35,     Training Accuracy=56.0%
iter 100:        Loss=1.45,     Training Accuracy=53.0%
iter 200:        Loss=1.40,     Training Accuracy=54.0%
iter 300:        Loss=1.38,     Training Accuracy=56.0%
iter 400:        Loss=1.23,     Training Accuracy=64.0%
------------------------------------------------------------
Epoch: 15, validation loss: 1.48, validation accuracy: 52.7%
------------------------------------------------------------
Training epoch: 16
iter   0:        Loss=1.44,     Training Accuracy=49.0%
iter 100:        Loss=1.37,     Training Accuracy=52.0%
iter 200:        Loss=1.38,     Training Accuracy=56.0%
iter 300:        Loss=1.41,     Training Accuracy=61.0%
iter 400:        Loss=1.31,     Training Accuracy=61.0%
------------------------------------------------------------
Epoch: 16, validation loss: 1.56, validation accuracy: 49.4%
------------------------------------------------------------
Training epoch: 17
iter   0:        Loss=1.31,     Training Accuracy=55.0%
iter 100:        Loss=1.29,     Training Accuracy=58.0%
iter 200:        Loss=1.36,     Training Accuracy=54.0%
iter 300:        Loss=1.32,     Training Accuracy=53.0%
iter 400:        Loss=1.33,     Training Accuracy=60.0%
------------------------------------------------------------
Epoch: 17, validation loss: 1.54, validation accuracy: 49.8%
------------------------------------------------------------
Training epoch: 18
iter   0:        Loss=1.39,     Training Accuracy=57.0%
iter 100:        Loss=1.34,     Training Accuracy=58.0%
iter 200:        Loss=1.28,     Training Accuracy=66.0%
iter 300:        Loss=1.51,     Training Accuracy=46.0%
iter 400:        Loss=1.44,     Training Accuracy=53.0%
------------------------------------------------------------
Epoch: 18, validation loss: 1.50, validation accuracy: 51.5%
------------------------------------------------------------
Training epoch: 19
iter   0:        Loss=1.42,     Training Accuracy=59.0%
iter 100:        Loss=1.25,     Training Accuracy=61.0%
iter 200:        Loss=1.43,     Training Accuracy=55.0%
iter 300:        Loss=1.24,     Training Accuracy=59.0%
iter 400:        Loss=1.37,     Training Accuracy=56.0%
------------------------------------------------------------
Epoch: 19, validation loss: 1.50, validation accuracy: 52.6%
------------------------------------------------------------
Training epoch: 20
iter   0:        Loss=1.31,     Training Accuracy=59.0%
iter 100:        Loss=1.34,     Training Accuracy=68.0%
iter 200:        Loss=1.40,     Training Accuracy=49.0%
iter 300:        Loss=1.21,     Training Accuracy=57.0%
iter 400:        Loss=1.38,     Training Accuracy=56.0%
------------------------------------------------------------
Epoch: 20, validation loss: 1.49, validation accuracy: 52.1%
------------------------------------------------------------
Training epoch: 21
iter   0:        Loss=1.34,     Training Accuracy=57.0%
iter 100:        Loss=1.26,     Training Accuracy=64.0%
iter 200:        Loss=1.30,     Training Accuracy=60.0%
iter 300:        Loss=1.40,     Training Accuracy=62.0%
iter 400:        Loss=1.53,     Training Accuracy=49.0%
------------------------------------------------------------
Epoch: 21, validation loss: 1.50, validation accuracy: 51.7%
```

```
-----------------------------------------------------------
Training epoch: 22
iter   0:        Loss=1.33,     Training Accuracy=62.0%
iter 100:        Loss=1.41,     Training Accuracy=57.0%
iter 200:        Loss=1.49,     Training Accuracy=53.0%
iter 300:        Loss=1.46,     Training Accuracy=51.0%
iter 400:        Loss=1.43,     Training Accuracy=50.0%
-----------------------------------------------------------
Epoch: 22, validation loss: 1.51, validation accuracy: 51.2%
-----------------------------------------------------------
Training epoch: 23
iter   0:        Loss=1.19,     Training Accuracy=66.0%
iter 100:        Loss=1.36,     Training Accuracy=54.0%
iter 200:        Loss=1.40,     Training Accuracy=59.0%
iter 300:        Loss=1.31,     Training Accuracy=57.0%
iter 400:        Loss=1.38,     Training Accuracy=55.0%
-----------------------------------------------------------
Epoch: 23, validation loss: 1.58, validation accuracy: 48.8%
-----------------------------------------------------------
Training epoch: 24
iter   0:        Loss=1.32,     Training Accuracy=59.0%
iter 100:        Loss=1.26,     Training Accuracy=60.0%
iter 200:        Loss=1.36,     Training Accuracy=58.0%
iter 300:        Loss=1.30,     Training Accuracy=58.0%
iter 400:        Loss=1.42,     Training Accuracy=56.0%
-----------------------------------------------------------
Epoch: 24, validation loss: 1.52, validation accuracy: 51.1%
-----------------------------------------------------------
Training epoch: 25
iter   0:        Loss=1.29,     Training Accuracy=59.0%
iter 100:        Loss=1.35,     Training Accuracy=60.0%
iter 200:        Loss=1.30,     Training Accuracy=57.0%
iter 300:        Loss=1.43,     Training Accuracy=56.0%
iter 400:        Loss=1.26,     Training Accuracy=59.0%
-----------------------------------------------------------
Epoch: 25, validation loss: 1.53, validation accuracy: 50.3%
-----------------------------------------------------------
Training epoch: 26
iter   0:        Loss=1.38,     Training Accuracy=52.0%
iter 100:        Loss=1.34,     Training Accuracy=57.0%
iter 200:        Loss=1.45,     Training Accuracy=54.0%
iter 300:        Loss=1.47,     Training Accuracy=50.0%
iter 400:        Loss=1.42,     Training Accuracy=58.0%
-----------------------------------------------------------
Epoch: 26, validation loss: 1.53, validation accuracy: 50.7%
-----------------------------------------------------------
Training epoch: 27
iter   0:        Loss=1.30,     Training Accuracy=52.0%
iter 100:        Loss=1.41,     Training Accuracy=57.0%
iter 200:        Loss=1.21,     Training Accuracy=62.0%
iter 300:        Loss=1.19,     Training Accuracy=60.0%
iter 400:        Loss=1.22,     Training Accuracy=64.0%
-----------------------------------------------------------
Epoch: 27, validation loss: 1.53, validation accuracy: 50.9%
-----------------------------------------------------------
Training epoch: 28
iter   0:        Loss=1.36,     Training Accuracy=58.0%
iter 100:        Loss=1.35,     Training Accuracy=53.0%
iter 200:        Loss=1.42,     Training Accuracy=53.0%
iter 300:        Loss=1.34,     Training Accuracy=61.0%
iter 400:        Loss=1.31,     Training Accuracy=58.0%
-----------------------------------------------------------
Epoch: 28, validation loss: 1.54, validation accuracy: 50.6%
-----------------------------------------------------------
Training epoch: 29
```

```
iter   0:        Loss=1.43,     Training Accuracy=58.0%
iter 100:        Loss=1.21,     Training Accuracy=57.0%
iter 200:        Loss=1.53,     Training Accuracy=56.0%
iter 300:        Loss=1.33,     Training Accuracy=58.0%
iter 400:        Loss=1.17,     Training Accuracy=63.0%
---------------------------------------------------------
Epoch: 29, validation loss: 1.54, validation accuracy: 50.2%
---------------------------------------------------------
Training epoch: 30
iter   0:        Loss=1.24,     Training Accuracy=59.0%
iter 100:        Loss=1.23,     Training Accuracy=66.0%
iter 200:        Loss=1.09,     Training Accuracy=69.0%
iter 300:        Loss=1.20,     Training Accuracy=67.0%
iter 400:        Loss=1.27,     Training Accuracy=56.0%
---------------------------------------------------------
Epoch: 30, validation loss: 1.57, validation accuracy: 49.8%
---------------------------------------------------------
Training epoch: 31
iter   0:        Loss=1.42,     Training Accuracy=58.0%
iter 100:        Loss=1.37,     Training Accuracy=53.0%
iter 200:        Loss=1.21,     Training Accuracy=64.0%
iter 300:        Loss=1.37,     Training Accuracy=48.0%
iter 400:        Loss=1.42,     Training Accuracy=57.0%
---------------------------------------------------------
Epoch: 31, validation loss: 1.51, validation accuracy: 51.8%
---------------------------------------------------------
Training epoch: 32
iter   0:        Loss=1.32,     Training Accuracy=57.0%
iter 100:        Loss=1.33,     Training Accuracy=57.0%
iter 200:        Loss=1.48,     Training Accuracy=59.0%
iter 300:        Loss=1.41,     Training Accuracy=54.0%
iter 400:        Loss=1.51,     Training Accuracy=61.0%
---------------------------------------------------------
Epoch: 32, validation loss: 1.51, validation accuracy: 52.2%
---------------------------------------------------------
Training epoch: 33
iter   0:        Loss=1.34,     Training Accuracy=55.0%
iter 100:        Loss=1.43,     Training Accuracy=51.0%
iter 200:        Loss=1.45,     Training Accuracy=51.0%
iter 300:        Loss=1.47,     Training Accuracy=51.0%
iter 400:        Loss=1.34,     Training Accuracy=55.0%
---------------------------------------------------------
Epoch: 33, validation loss: 1.50, validation accuracy: 51.5%
---------------------------------------------------------
Training epoch: 34
iter   0:        Loss=1.22,     Training Accuracy=61.0%
iter 100:        Loss=1.26,     Training Accuracy=61.0%
iter 200:        Loss=1.31,     Training Accuracy=57.0%
iter 300:        Loss=1.21,     Training Accuracy=66.0%
iter 400:        Loss=1.26,     Training Accuracy=61.0%
---------------------------------------------------------
Epoch: 34, validation loss: 1.53, validation accuracy: 50.9%
---------------------------------------------------------
Training epoch: 35
iter   0:        Loss=1.33,     Training Accuracy=61.0%
iter 100:        Loss=1.21,     Training Accuracy=65.0%
iter 200:        Loss=1.25,     Training Accuracy=58.0%
iter 300:        Loss=1.42,     Training Accuracy=57.0%
iter 400:        Loss=1.18,     Training Accuracy=59.0%
---------------------------------------------------------
Epoch: 35, validation loss: 1.51, validation accuracy: 51.6%
---------------------------------------------------------
Training epoch: 36
iter   0:        Loss=1.29,     Training Accuracy=67.0%
iter 100:        Loss=1.13,     Training Accuracy=65.0%
```

```
iter 200:        Loss=1.32,     Training Accuracy=56.0%
iter 300:        Loss=1.31,     Training Accuracy=56.0%
iter 400:        Loss=1.35,     Training Accuracy=61.0%
---------------------------------------------------------
Epoch: 36, validation loss: 1.56, validation accuracy: 50.1%
---------------------------------------------------------
Training epoch: 37
iter   0:        Loss=1.49,     Training Accuracy=55.0%
iter 100:        Loss=1.25,     Training Accuracy=58.0%
iter 200:        Loss=1.14,     Training Accuracy=60.0%
iter 300:        Loss=1.38,     Training Accuracy=55.0%
iter 400:        Loss=1.26,     Training Accuracy=50.0%
---------------------------------------------------------
Epoch: 37, validation loss: 1.54, validation accuracy: 50.3%
---------------------------------------------------------
Training epoch: 38
iter   0:        Loss=1.43,     Training Accuracy=45.0%
iter 100:        Loss=1.52,     Training Accuracy=48.0%
iter 200:        Loss=1.20,     Training Accuracy=62.0%
iter 300:        Loss=1.28,     Training Accuracy=61.0%
iter 400:        Loss=1.38,     Training Accuracy=51.0%
---------------------------------------------------------
Epoch: 38, validation loss: 1.50, validation accuracy: 52.6%
---------------------------------------------------------
Training epoch: 39
iter   0:        Loss=1.26,     Training Accuracy=61.0%
iter 100:        Loss=1.18,     Training Accuracy=64.0%
iter 200:        Loss=1.25,     Training Accuracy=59.0%
iter 300:        Loss=1.65,     Training Accuracy=44.0%
iter 400:        Loss=1.54,     Training Accuracy=52.0%
---------------------------------------------------------
Epoch: 39, validation loss: 1.48, validation accuracy: 52.6%
---------------------------------------------------------
Training epoch: 40
iter   0:        Loss=1.52,     Training Accuracy=51.0%
iter 100:        Loss=1.39,     Training Accuracy=51.0%
iter 200:        Loss=1.35,     Training Accuracy=60.0%
iter 300:        Loss=1.46,     Training Accuracy=52.0%
iter 400:        Loss=1.35,     Training Accuracy=57.0%
---------------------------------------------------------
Epoch: 40, validation loss: 1.51, validation accuracy: 51.7%
---------------------------------------------------------
Training epoch: 41
iter   0:        Loss=1.32,     Training Accuracy=57.0%
iter 100:        Loss=1.25,     Training Accuracy=62.0%
iter 200:        Loss=1.37,     Training Accuracy=51.0%
iter 300:        Loss=1.29,     Training Accuracy=61.0%
iter 400:        Loss=1.37,     Training Accuracy=54.0%
---------------------------------------------------------
Epoch: 41, validation loss: 1.50, validation accuracy: 52.2%
---------------------------------------------------------
Training epoch: 42
iter   0:        Loss=1.39,     Training Accuracy=51.0%
iter 100:        Loss=1.39,     Training Accuracy=60.0%
iter 200:        Loss=1.23,     Training Accuracy=56.0%
iter 300:        Loss=1.42,     Training Accuracy=57.0%
iter 400:        Loss=1.19,     Training Accuracy=64.0%
---------------------------------------------------------
Epoch: 42, validation loss: 1.52, validation accuracy: 51.6%
---------------------------------------------------------
Training epoch: 43
iter   0:        Loss=1.36,     Training Accuracy=56.0%
iter 100:        Loss=1.38,     Training Accuracy=54.0%
iter 200:        Loss=1.21,     Training Accuracy=59.0%
iter 300:        Loss=1.41,     Training Accuracy=57.0%
```

```
iter 400:        Loss=1.39,     Training Accuracy=50.0%
----------------------------------------------------------
Epoch: 43, validation loss: 1.56, validation accuracy: 49.6%
----------------------------------------------------------
Training epoch: 44
iter   0:        Loss=1.30,     Training Accuracy=54.0%
iter 100:        Loss=1.11,     Training Accuracy=68.0%
iter 200:        Loss=1.31,     Training Accuracy=64.0%
iter 300:        Loss=1.40,     Training Accuracy=52.0%
iter 400:        Loss=1.47,     Training Accuracy=53.0%
----------------------------------------------------------
Epoch: 44, validation loss: 1.54, validation accuracy: 50.3%
----------------------------------------------------------
Training epoch: 45
iter   0:        Loss=1.12,     Training Accuracy=65.0%
iter 100:        Loss=1.64,     Training Accuracy=44.0%
iter 200:        Loss=1.33,     Training Accuracy=60.0%
iter 300:        Loss=1.28,     Training Accuracy=63.0%
iter 400:        Loss=1.41,     Training Accuracy=51.0%
----------------------------------------------------------
Epoch: 45, validation loss: 1.54, validation accuracy: 51.3%
----------------------------------------------------------
Training epoch: 46
iter   0:        Loss=1.33,     Training Accuracy=55.0%
iter 100:        Loss=1.29,     Training Accuracy=63.0%
iter 200:        Loss=1.43,     Training Accuracy=52.0%
iter 300:        Loss=1.34,     Training Accuracy=54.0%
iter 400:        Loss=1.31,     Training Accuracy=57.0%
----------------------------------------------------------
Epoch: 46, validation loss: 1.55, validation accuracy: 50.4%
----------------------------------------------------------
Training epoch: 47
iter   0:        Loss=1.45,     Training Accuracy=53.0%
iter 100:        Loss=1.46,     Training Accuracy=51.0%
iter 200:        Loss=1.45,     Training Accuracy=51.0%
iter 300:        Loss=1.33,     Training Accuracy=60.0%
iter 400:        Loss=1.47,     Training Accuracy=51.0%
----------------------------------------------------------
Epoch: 47, validation loss: 1.52, validation accuracy: 51.2%
----------------------------------------------------------
Training epoch: 48
iter   0:        Loss=1.46,     Training Accuracy=54.0%
iter 100:        Loss=1.19,     Training Accuracy=62.0%
iter 200:        Loss=1.33,     Training Accuracy=51.0%
iter 300:        Loss=1.18,     Training Accuracy=63.0%
iter 400:        Loss=1.30,     Training Accuracy=57.0%
----------------------------------------------------------
Epoch: 48, validation loss: 1.51, validation accuracy: 51.6%
----------------------------------------------------------
Training epoch: 49
iter   0:        Loss=1.39,     Training Accuracy=56.0%
iter 100:        Loss=1.58,     Training Accuracy=51.0%
iter 200:        Loss=1.30,     Training Accuracy=57.0%
iter 300:        Loss=1.39,     Training Accuracy=55.0%
iter 400:        Loss=1.44,     Training Accuracy=48.0%
----------------------------------------------------------
Epoch: 49, validation loss: 1.55, validation accuracy: 49.9%
----------------------------------------------------------
Training epoch: 50
iter   0:        Loss=1.39,     Training Accuracy=51.0%
iter 100:        Loss=1.24,     Training Accuracy=57.0%
iter 200:        Loss=1.45,     Training Accuracy=53.0%
iter 300:        Loss=1.54,     Training Accuracy=48.0%
iter 400:        Loss=1.36,     Training Accuracy=50.0%
----------------------------------------------------------
```

```
Epoch: 50, validation loss: 1.53, validation accuracy: 50.8%
----------------------------------------------------------
```

Results

Config.	Classification Error		
***	Training	Testing	Accuracy
50HLN+no regularization	1.33	1.42	49.5
50HLN+L2 regularization	1.24	1.54	48.8
250HLN+no regularization	0.80	1.50	52.0
250HLN+L2 regularization	1.36	1.53	50.8

From the above results, we can conclude that MLP does not do a good job on CIFAR-10 dataset. The next step is to use CNN and compare the results.

In [ ]:

```

```

2-Adaboost

In [1]:

```
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from random import sample
import math
import random
from sklearn import tree
from sklearn.metrics import confusion_matrix
pd.set_option('display.max_rows', None)
```

In [2]:

```
class adaboost:
    def __init__(self):
        pass

    def sample(self, N, p):
        random_sample = np.zeros(N)
        p_estimate = np.zeros(len(p))
        p_cdf = np.cumsum(p)
        counts = np.zeros(len(p))

        for n in range(N):
            # generate a random number on [0,1]
            x = np.random.rand()
            random_sample[n] = np.where(((p_cdf > x)*1.0) == 1.)[0][0]
            counts[int(random_sample[n])] += 1

        p_estimate = counts/counts.sum()
        return random_sample, p_estimate

    def weakLearn(self, D, dataSet, DT=None):
        # DT model with depth one
        clf_gini = DecisionTreeClassifier(criterion = "gini", random_state = 10(
        #random distribution for the samples
        random.seed(10)
        rDataSet = dataSet.sample(len(D), replace = True, weights = D)

        X_train = rDataSet.iloc[:,0:4]
        y_train = rDataSet.iloc[:,4]

        #fitting the DT model
        stump = clf_gini.fit(X_train, y_train)

        return stump


    def fit(self, X, y, T):
        """
        X: feature vector
        y: labels
        T: number of iterations
        """
        self.trainingData = pd.DataFrame(X)
        self.trainingData['Label'] = y
```

```
        self.stumps = []
        self.alphas = []

        #Initially assign same weights to each records in the dataset
        self.trainingData['weights'] = 1/(self.trainingData.shape[0])

        for i in range(T):
            print("iteration {} ...".format(i+1))
            #create weak classifier
            stump = self.weakLearn(self.trainingData['weights'], self.trainingDa

            #append stumps
            self.stumps.append(stump)

            #make a prediction with the weak model
            y_pred = stump.predict(self.trainingData.iloc[:,0:4])


            #save the prediction
            self.trainingData['pred'] = y_pred

            #find the misclassified samples
            self.trainingData['misclassified'] = \
                                np.where(self.trainingData['Label'] == self.trai


            #calulating the error
            e = sum(self.trainingData['misclassified'] * self.trainingData['weig

            #calculation of alpha
            alpha = 0.5*math.log((1-e)/e)

            self.alphas.append(alpha)

            #update weights
            new_weights = self.trainingData['weights']*np.exp(-1*alpha*self.trai
                                *self.trainingData['pred'])

            #normalized weight
            z = sum(new_weights)
            normalized_weights = new_weights/z


            #add the new weights
            self.trainingData['weights'] = normalized_weights


    def predict(self, X):
        """
        Make prediction using the fitted model
        """
        stump_preds = np.array([stump.predict(X) for stump in self.stumps])
        return np.sign(np.dot(np.asarray(self.alphas), stump_preds))
```

Data preparation

In [3]:
```
blood = pd.read_csv("blood.csv", header=None)
blood.iloc[:,-1:] = blood.iloc[:,-1:].replace(to_replace = [1,0], value=[1,-1])
```

In [4]:
```
X = blood.iloc[:,0:4].values
X.shape
```

Out[4]:
```
(748, 4)
```

In [5]:
```
y = blood.iloc[:,4].values
y.shape
```

Out[5]:
```
(748,)
```

Training and predicting

In [6]:
```
obj = adaboost()
```

In [7]:
```
obj.fit(X, y, 20)
```

```
iteration 1 ...
iteration 2 ...
iteration 3 ...
iteration 4 ...
iteration 5 ...
iteration 6 ...
iteration 7 ...
iteration 8 ...
iteration 9 ...
iteration 10 ...
iteration 11 ...
iteration 12 ...
iteration 13 ...
iteration 14 ...
iteration 15 ...
iteration 16 ...
iteration 17 ...
iteration 18 ...
iteration 19 ...
iteration 20 ...
```

In [8]:
```
stump_preds = obj.predict(X)
```

Comparing accuracy to sklearn’simplementation

In [9]:
```
#Using the confusion matrix for evaluating the accuracy
c=confusion_matrix(y, stump_preds)
c
```

Out[9]:
```
array([[543,  27],
       [118,  60]])
```

In [10]:
```
#Accuracy
(c[0,0]+c[1,1])/np.sum(c)*100
```

Out[10]: 80.61497326203208

In [11]:
```
X_train = X
y_train = y
```

In [12]:
```
clf = AdaBoostClassifier(n_estimators=20, random_state=0)
clf.fit(X_train, y_train)
```

Out[12]: AdaBoostClassifier(n_estimators=20, random_state=0)

In [13]:
```
clf.score(X_train, y_train)
```

Out[13]: 0.8074866310160428

In [ ]:

Recurrent Neural Networks for Languange Modeling

Model modification

The plan is to modify the model build by adding L1 and L2 regularizations to the GRU layer and increase the number of RNN from 1024 to 2048. Reultes:

I noticed that the loss after 40th epoch the loss function was 3.

In [5]:
```
import tensorflow as tf
from tensorflow.keras.layers.experimental import preprocessing

import numpy as np
import os
import time
```

In [6]:
```
path_to_file = tf.keras.utils.get_file('shakespeare.txt', 'https://storage.googl
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/dat
a/shakespeare.txt
1122304/1115394 [==============================] - 0s 0us/step
```

In [7]:
```
path_to_file
```

Out[7]:
```
'/root/.keras/datasets/shakespeare.txt'
```

In [8]:
```
# Read, then decode for py2 compat.
text = open(path_to_file, 'rb').read().decode(encoding='utf-8')
# length of text is the number of characters in it
print(f'Length of text: {len(text)} characters')
```

```
Length of text: 1115394 characters
```

In [9]:
```
# print(text)
```

In [10]:
```
# Take a look at the first 250 characters in text
print(text[:250])
```

```
First Citizen:
Before we proceed any further, hear me speak.

All:
Speak, speak.

First Citizen:
You are all resolved rather to die than to famish?

All:
```

```
Resolved. resolved.

First Citizen:
First, you know Caius Marcius is chief enemy to the people.
```

In [11]:
```
# The unique characters in the file
vocab = sorted(set(text))
print(f'{len(vocab)} unique characters')
```

```
65 unique characters
```

In [12]:
```
print(vocab)
```

```
['\n', ' ', '!', '$', '&', "'", ',', '-', '.', '3', ':', ';', '?', 'A', 'B',
'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R',
'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x',
'y', 'z']
```

Process the text

In [13]:
```
#The preprocessing.StringLookup layer can convert each character into
#a numeric ID. It just needs the text to be split into tokens first.
example_texts = ['abcdefg', 'xyz']

chars = tf.strings.unicode_split(example_texts, input_encoding='UTF-8')
chars
```

Out[13]:
```
<tf.RaggedTensor [[b'a', b'b', b'c', b'd', b'e', b'f', b'g'], [b'x', b'y',
b'z']]>
```

In [14]:
```
#Now create the preprocessing.StringLookup layer:
ids_from_chars = preprocessing.StringLookup(
    vocabulary=list(vocab))
```

In [15]:
```
#It converts form tokens to character IDs, padding with 0
ids = ids_from_chars(chars)
ids
```

Out[15]:
```
<tf.RaggedTensor [[41, 42, 43, 44, 45, 46, 47], [64, 65, 66]]>
```

In [16]:
```
#invert this representation and recover human-readable strings
chars_from_ids = tf.keras.layers.experimental.preprocessing.StringLookup(
    vocabulary=ids_from_chars.get_vocabulary(), invert=True)
```

In [17]:
```
#This layer recovers the characters from the vectors of IDs, and returns them as
chars = chars_from_ids(ids)
chars
```

Out[17]:
```
<tf.RaggedTensor [[b'a', b'b', b'c', b'd', b'e', b'f', b'g'], [b'x', b'y',
b'z']]>
```

In [18]:
```
#You can tf.strings.reduce_join to join the characters back into strings.
def text_from_ids(ids):
  return tf.strings.reduce_join(chars_from_ids(ids), axis=-1)
```

In [19]:
```
text_from_ids(ids)
```

Out[19]:
```
<tf.Tensor: shape=(2,), dtype=string, numpy=array([b'abcdefg', b'xyz'], dtype=ob
ject)>
```

The prediction task

In [20]:
```
#Create training examples and targets
all_ids = ids_from_chars(tf.strings.unicode_split(text, 'UTF-8'))
all_ids
```

Out[20]:
```
<tf.Tensor: shape=(1115394,), dtype=int64, numpy=array([20, 49, 58, ..., 47, 10,
2])>
```

In [21]:
```
#use the tf.data.Dataset.from_tensor_slices function to convert the text vector
#a stream of character indices.
ids_dataset = tf.data.Dataset.from_tensor_slices(all_ids)
```

In [22]:
```
for ids in ids_dataset.take(10):
    print(chars_from_ids(ids).numpy().decode('utf-8'))
```

```
F
i
r
s
t
 
C
i
t
i
```

In [23]:
```
seq_length = 100
examples_per_epoch = len(text)//(seq_length+1)
```

In [24]:
```
#batch method lets you easily convert these individual characters
#to sequences of the desired size.
sequences = ids_dataset.batch(seq_length+1, drop_remainder=True)

for seq in sequences.take(1):
  print(chars_from_ids(seq))
```

```
tf.Tensor(
[b'F' b'i' b'r' b's' b't' b' ' b'C' b'i' b't' b'i' b'z' b'e' b'n' b':'
 b'\n' b'B' b'e' b'f' b'o' b'r' b'e' b' ' b'w' b'e' b' ' b'p' b'r' b'o'
 b'c' b'e' b'e' b'd' b' ' b'a' b'n' b'y' b' ' b'f' b'u' b'r' b't' b'h'
 b'e' b'r' b',' b' ' b'h' b'e' b'a' b'r' b' ' b'm' b'e' b' ' b's' b'p'
 b'e' b'a' b'k' b'.' b'\n' b'\n' b'A' b'l' b'l' b':' b'\n' b'S' b'p' b'e'
 b'a' b'k' b',' b' ' b's' b'p' b'e' b'a' b'k' b'.' b'\n' b'\n' b'F' b'i'
```

```
b'r' b's' b't' b' ' b'C' b'i' b't' b'i' b'z' b'e' b'n' b':' b'\n' b'Y'
b'o' b'u' b' '], shape=(101,), dtype=string)
```

In [25]:

```
#It's easier to see what this is doing if you join the tokens back into strings:
for seq in sequences.take(5):
  print(text_from_ids(seq).numpy())
```

```
b'First Citizen:\nBefore we proceed any further, hear me speak.\n\nAll:\nSpeak,
speak.\n\nFirst Citizen:\nYou '
b'are all resolved rather to die than to famish?\n\nAll:\nResolved. resolved.\n
\nFirst Citizen:\nFirst, you k'
b"now Caius Marcius is chief enemy to the people.\n\nAll:\nWe know't, we kno
w't.\n\nFirst Citizen:\nLet us ki"
b"ll him, and we'll have corn at our own price.\nIs't a verdict?\n\nAll:\nNo mor
e talking on't; let it be d"
b'one: away, away!\n\nSecond Citizen:\nOne word, good citizens.\n\nFirst Citize
n:\nWe are accounted poor citi'
```

In [26]:

```
#function that takes a sequence as input, duplicates, and shifts it to align the
#label for each timestep
def split_input_target(sequence):
    input_text = sequence[:-1]
    target_text = sequence[1:]
    return input_text, target_text
```

In [27]:

```
dataset = sequences.map(split_input_target)
```

In [28]:

```
for input_example, target_example in dataset.take(1):
    print("Input :", text_from_ids(input_example).numpy())
    print("Target:", text_from_ids(target_example).numpy())
```

```
Input : b'First Citizen:\nBefore we proceed any further, hear me speak.\n\nAl
l:\nSpeak, speak.\n\nFirst Citizen:\nYou'
Target: b'irst Citizen:\nBefore we proceed any further, hear me speak.\n\nAll:\n
Speak, speak.\n\nFirst Citizen:\nYou '
```

Create training batches

In [29]:

```
# Batch size
BATCH_SIZE = 64

# Buffer size to shuffle the dataset
# (TF data is designed to work with possibly infinite sequences,
# so it doesn't attempt to shuffle the entire sequence in memory. Instead,
# it maintains a buffer in which it shuffles elements).
BUFFER_SIZE = 10000

dataset = (
    dataset
    .shuffle(BUFFER_SIZE)
    .batch(BATCH_SIZE, drop_remainder=True)
    .prefetch(tf.data.experimental.AUTOTUNE))

dataset
```

Out[29]: <PrefetchDataset shapes: ((64, 100), (64, 100)), types: (tf.int64, tf.int64)>

Build The Model

In [30]:

```
# Length of the vocabulary in chars
vocab_size = len(vocab)

# The embedding dimension
embedding_dim = 256

# Number of RNN units
rnn_units = 2048 #incease from 1024
```

In [31]:

```
class MyModel(tf.keras.Model):
  def __init__(self, vocab_size, embedding_dim, rnn_units):
    super().__init__(self)
    self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
    self.gru = tf.keras.layers.GRU(rnn_units,
                                   kernel_regularizer=tf.keras.regularizers.L1(0
                                   activity_regularizer=tf.keras.regularizers.L2
                                   return_sequences=True,
                                   return_state=True)
    self.dense = tf.keras.layers.Dense(vocab_size)

  def call(self, inputs, states=None, return_state=False, training=False):
    x = inputs
    x = self.embedding(x, training=training)
    if states is None:
      states = self.gru.get_initial_state(x)
    x, states = self.gru(x, initial_state=states, training=training)
    x = self.dense(x, training=training)

    if return_state:
      return x, states
    else:
      return x
```

In [32]:

```
model = MyModel(
    # Be sure the vocabulary size matches the `StringLookup` layers.
    vocab_size=len(ids_from_chars.get_vocabulary()),
    embedding_dim=embedding_dim,
    rnn_units=rnn_units)
```

In [33]:

```
for input_example_batch, target_example_batch in dataset.take(1):
    example_batch_predictions = model(input_example_batch)
    print(example_batch_predictions.shape, "# (batch_size, sequence_length, voca
```

(64, 100, 67) # (batch_size, sequence_length, vocab_size)

In [34]:

```
model.summary()
```

Model: "my_model"

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        multiple                  17152
_________________________________________________________________
gru (GRU)                    multiple                  14168064
_________________________________________________________________
dense (Dense)                multiple                  137283
=================================================================
Total params: 14,322,499
Trainable params: 14,322,499
Non-trainable params: 0
_________________________________________________________________
```

In [35]:

```
sampled_indices = tf.random.categorical(example_batch_predictions[0], num_sample
sampled_indices = tf.squeeze(sampled_indices, axis=-1).numpy()
```

In [36]:

```
sampled_indices
```

Out[36]:

```
array([52,  9,  2, 58,  0, 44,  2, 56,  6, 11,  0, 39, 29, 59, 56,  7, 60,
       52, 25, 18, 57, 16, 11, 28, 53, 64, 21, 54, 42,  2, 10, 44, 28, 34,
       35,  3, 33, 45, 59, 18, 49, 38, 53,  9, 53, 27,  5, 35, 15, 53, 62,
       39, 30, 23, 38, 57,  9, 33, 10, 36,  7, 54, 52, 12,  5, 39, 38, 64,
       52, 22, 57, 31,  1, 42, 63,  0, 25,  1,  2, 11, 60,  7, 51, 34, 30,
        1,  6,  7, 20, 35, 11, 33, 38, 12, 51,  8, 13, 57, 31, 32])
```

In [37]:

```
print("Input:\n", text_from_ids(input_example_batch[0]).numpy())
print()
print("Next Char Predictions:\n", text_from_ids(sampled_indices).numpy())
```

```
Input:
 b"th Baptista ta'en,\nThat none shall have access unto Bianca\nTill Katharina t
he curst have got a husba"

Next Char Predictions:
 b"l-\nrd\np&3YOsp'tlKDqB3NmxGnb\n.dNTU SesDiXm-mM$UAmvYPIXq-S.V'nl:$YXxlHqQ[UN
K]bwK[UNK]\n3t'kTP[UNK]&'FU3SX:k,;qQR"
```

Train the model

In [38]:

```
#Attach an optimizer, and a loss function
loss = tf.losses.SparseCategoricalCrossentropy(from_logits=True)
```

In [39]:

```
example_batch_loss = loss(target_example_batch, example_batch_predictions)
mean_loss = example_batch_loss.numpy().mean()
print("Prediction shape: ", example_batch_predictions.shape, " # (batch_size, se
print("Mean loss:        ", mean_loss)
```

```
Prediction shape:  (64, 100, 67)  # (batch_size, sequence_length, vocab_size)
Mean loss:         4.2048993
```

In [40]:

```
tf.exp(mean_loss).numpy()
```

Out[40]:

```
67.01385
```

In [41]:
```
#Configure the training procedure using the tf.keras.Model.compile method. Use t
#arguments and the loss function
model.compile(optimizer='adam', loss=loss)
```

Configure checkpoints

In [42]:
```
#Use a tf.keras.callbacks.ModelCheckpoint to ensure that checkpoints are saved d
# Directory where the checkpoints will be saved
checkpoint_dir = './training_checkpoints'
# Name of the checkpoint files
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_{epoch}")

checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_prefix,
    save_weights_only=True)
```

Execute the training

In [43]:
```
EPOCHS = 20
```

In [44]:
```
history = model.fit(dataset, epochs=EPOCHS, callbacks=[checkpoint_callback])
```

```
Epoch 1/20
172/172 [==============================] - 30s 159ms/step - loss: 4.6363
Epoch 2/20
172/172 [==============================] - 29s 164ms/step - loss: 2.5367
Epoch 3/20
172/172 [==============================] - 30s 168ms/step - loss: 2.2997
Epoch 4/20
172/172 [==============================] - 31s 172ms/step - loss: 2.1452
Epoch 5/20
172/172 [==============================] - 31s 176ms/step - loss: 2.0310
Epoch 6/20
172/172 [==============================] - 32s 180ms/step - loss: 1.9454
Epoch 7/20
172/172 [==============================] - 32s 182ms/step - loss: 1.8738
Epoch 8/20
172/172 [==============================] - 32s 183ms/step - loss: 1.8186
Epoch 9/20
172/172 [==============================] - 32s 183ms/step - loss: 1.7724
Epoch 10/20
172/172 [==============================] - 32s 183ms/step - loss: 1.7284
Epoch 11/20
172/172 [==============================] - 32s 183ms/step - loss: 1.6892
Epoch 12/20
172/172 [==============================] - 32s 183ms/step - loss: 1.6509
Epoch 13/20
172/172 [==============================] - 32s 183ms/step - loss: 1.6161
Epoch 14/20
172/172 [==============================] - 32s 183ms/step - loss: 1.5840
Epoch 15/20
172/172 [==============================] - 32s 183ms/step - loss: 1.5507
Epoch 16/20
172/172 [==============================] - 33s 183ms/step - loss: 1.5194
Epoch 17/20
```

```
172/172 [==============================] - 32s 183ms/step - loss: 1.4825
Epoch 18/20
172/172 [==============================] - 32s 183ms/step - loss: 1.4470
Epoch 19/20
172/172 [==============================] - 32s 183ms/step - loss: 1.4064
Epoch 20/20
172/172 [==============================] - 32s 183ms/step - loss: 1.3621
```

Generate text

In [ ]:

```python
#The following makes a single step prediction:
class OneStep(tf.keras.Model):
  def __init__(self, model, chars_from_ids, ids_from_chars, temperature=1.0):
    super().__init__()
    self.temperature = temperature
    self.model = model
    self.chars_from_ids = chars_from_ids
    self.ids_from_chars = ids_from_chars

    # Create a mask to prevent "" or "[UNK]" from being generated.
    skip_ids = self.ids_from_chars(['', '[UNK]'])[:, None]
    sparse_mask = tf.SparseTensor(
        # Put a -inf at each bad index.
        values=[-float('inf')]*len(skip_ids),
        indices=skip_ids,
        # Match the shape to the vocabulary
        dense_shape=[len(ids_from_chars.get_vocabulary())])
    self.prediction_mask = tf.sparse.to_dense(sparse_mask)

  @tf.function
  def generate_one_step(self, inputs, states=None):
    # Convert strings to token IDs.
    input_chars = tf.strings.unicode_split(inputs, 'UTF-8')
    input_ids = self.ids_from_chars(input_chars).to_tensor()

    # Run the model.
    # predicted_logits.shape is [batch, char, next_char_logits]
    predicted_logits, states = self.model(inputs=input_ids, states=states,
                                          return_state=True)
    # Only use the last prediction.
    predicted_logits = predicted_logits[:, -1, :]
    predicted_logits = predicted_logits/self.temperature
    # Apply the prediction mask: prevent "" or "[UNK]" from being generated.
    predicted_logits = predicted_logits + self.prediction_mask

    # Sample the output logits to generate token IDs.
    predicted_ids = tf.random.categorical(predicted_logits, num_samples=1)
    predicted_ids = tf.squeeze(predicted_ids, axis=-1)

    # Convert from token ids to characters
    predicted_chars = self.chars_from_ids(predicted_ids)

    # Return the characters and model state.
    return predicted_chars, states
```

In [ ]:

```python
one_step_model = OneStep(model, chars_from_ids, ids_from_chars)
```

In [ ]:
```python
start = time.time()
states = None
next_char = tf.constant(['ROMEO:'])
result = [next_char]

for n in range(1000):
  next_char, states = one_step_model.generate_one_step(next_char, states=states)
  result.append(next_char)

result = tf.strings.join(result)
end = time.time()
print(result[0].numpy().decode('utf-8'), '\n\n' + '_'*80)
print('\nRun time:', end - start)
```

```
ROMEO:
The last of Warwick call.

CAMILLO:
Nay, your office is dead.

LORTS:
So doth the death, they have a hearten mouth:
And how she'll sooner in his mother's.
The foul sir's news.

BIANCA:
Adgend and grief say he is colder,
For such a hand of her, of my heart
And hell in sucken me Above a cleagures
That you shall unto Lonnon o'er the land, whose hap that Clifford,
Was sentence of my life; for he dishollow'd attenting music and
personal, sir.

GLOUCESTER:
Sir Richard, Or little tongue
For beying foot to such absolate.

LEONTES:
What willoub children how?

ROMEO:
I pray thee, my lord.

DUKE VINCENTIO:
It is not meet him well: you are dishonour'd between out
Where'er the people of this iclan careless.
Say that she lives.

TRASALLE:
I do become him; for I think, let me hear
The wind sid little eye o' the people, who haviness
my wag's faithful fearful long, and nothing else.'
Hast thou behold it straight degree?

HORTENSIO:
For this affection given him hence,
And he will muck in promise beaute

________________________________________________________________________________

Run time: 1.754654884338379
```

In [ ]: