HW_04 Problem #1

1 Multi-Layer Perceptron [20pts]

In class we discussed the derivation of the backpropagation algorithm for neural networks. In this problem, you will train a neural network on the CIFAR10 data set. Train a Multi-Layer Perceptron (MLP) neural network on the CIFAR10 data set. This is an opened implementation problem, but I expect that you implement the MLP with at least two different hidden layer sizes and use regularization.

- Report the classification error on the training and testing data each configuration of the neural network. For example, you should report the results in the form of a table

	Classification Error	
	training	**testing**
50HLN+no regularization	0.234	0.253
50HLN+L_2 regularization	0.192	0.203
250HLN+no regularization	0.134	0.153
250HLN+L_2 regularization	0.092	0.013

  List all the parameters that you are using (i.e., number of learning rounds, regularization parameters, learning rate, etc.)
- I would suggest using Google's TensorFlow, PyTorch or Keras library to implement the MLP; however, you are free to use whatever library you'd like. If that is the case, here is a link to the data
- I recommend using a cloud platform such as Google Colab to run the code.

This Code was written by Robert 'Quinn' Hull, and borrowed elements from several other resources:

- Resources used to make HW_04 Problem #1:
  - The 'load CIFAR' dataset heavilty borrowed from pytorch tutorial - https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
  - A refresher on the MLP borrowed from machine learning mastery - https://machinelearningmastery.com/neural-networks-crash-course/
  - Some code borrowed from medium . com about putting MLP into a pytorch - https://medium.com/@aungkyawmyint_26195/multi-layer-perceptron-mnist-pytorch-463f795b897a
  - Some notes on regularization in MLP:
    - https://cedar.buffalo.edu/~srihari/CSE574/Chap5/Chap5.5-Regularization.pdf
  - Very helpful for avoiding a blocker with batch_size dimensions. Be very careful with linear layers as the first parameter is batch_size, this is different than for convolutional layers
    - https://discuss.pytorch.org/t/valueerror-expected-input-batch-size-324-to-match-target-batch-size-4/24498valueerror-expected-input-batch-size-324-to-match-target-batch-size-4/24498
    - https://towardsdatascience.com/pytorch-layer-dimensions-what-sizes-should-they-be-and-why-4265a41e01fd
  - L2 normalization is implemented in the optimizer

    - https://pytorch.org/docs/stable/optim.html
  - How to save a model object
    - https://pytorch.org/tutorials/beginner/saving_loading_models.html

In [1]:
```
%matplotlib inline
```

In [2]:
```
import torch
import torchvision
import torchvision.transforms as transforms
from torch.utils.data.sampler import SubsetRandomSampler
import matplotlib.pyplot as plt
import numpy as np
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

import pandas as pd
```

HW_04_Problem 1

MLP

- Pseudocode:
  1. Load CIFAR Data
     - Show an image
  2. Build two MLP models
     - Multiple layer sizes
     - Use regularization
  3. For each model
     A. Train a multi-layer perceptron (MLP) on the CIFAR10 data set
        - Report error on training data
          - w/ regularization
          - w/o regularization
        - include report of hyperparameters (epochs, regularization, learning rate)
     B. Test a multi-layer perceptron (MLP) on the CIFAR10 data set
        - Report error on testing data
          - w/ regularization
          - w/o regularization
        - include report of hyperparameters (epochs, regularization, learning rate)

1. Load CIFAR Dataset

In [4]:
```
# # number of subprocesses to use for data loading
# num_workers = 0
# how many samples per batch to load
batch_size = 12
# # percentage of training set to use as validation
# valid_size = 0.2
```

```
# convert data to torch.FloatTensor
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

# choose the training and testing datasets
train_data = torchvision.datasets.CIFAR10(root = './data', train = True, downloa
test_data = torchvision.datasets.CIFAR10(root = './data', train = False, downloa

train_loader = torch.utils.data.DataLoader(train_data, batch_size=batch_size,
                                          shuffle=True, num_workers=0)
test_loader = torch.utils.data.DataLoader(test_data, batch_size=batch_size,
                                         shuffle=False, num_workers=0)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

Show an Image

In [5]:

```
classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

# functions to show an image
def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# get some random training images
dataiter = iter(train_loader)
images, labels = dataiter.next()

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join('%5s' % classes[labels[j]] for j in range(batch_size)))
```


```
  car   cat  ship truck  bird  frog   car  frog truck horse  bird  bird
```

Data Preparation

2. Build an MLP model

In [6]:

```
# from pytorch tutorial (see resources)
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
```

```
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

# modified from Net2 in medium article (see resources)
class Net3(nn.Module):
    '''a two-layer linear model
      dropout: with default dropout = True
      do_const: with default dropout countant = 0.2
      hidden: with a default hidden dimension = 100
    '''

    def __init__(self, hidden=100, dropout=True, do_const=0.2):
        super(Net3,self).__init__()

        # characteristics of input
        dim_1 = 32 # x dimensions of the figure
        dim_2 = 32 # y dimensions of the figure
        num_classes = 10 # number of output classes
        num_col = 3 # since color images, for greyscale = 1

        # number of hidden nodes in each layer
        self.hidden_1  = hidden # hidden layer dimension (user defined)
        self.hidden_2  = hidden # hidden layer dimension (user defined)

        # dropout
        self.dropout = dropout
        self.do_const = do_const

        # linear layer (num_col*dim_1 * dim_2 -> hidden_1) *NOTE Adding num_col
        self.fc1 = nn.Linear(3*32*32, self.hidden_1)
        # linear layer (n_hidden -> hidden_2)
        self.fc2 = nn.Linear(self.hidden_1,self.hidden_2)
        # linear layer (n_hidden -> num_classes)
        self.fc3 = nn.Linear(self.hidden_2,10)

        # optional dropout
        if self.dropout:
          # dropout layer (p=self.do_const)
          # dropout prevents overfitting of data
          self.droput = nn.Dropout(p=self.do_const)
        else:
          self.droput = nn.Dropout(p=0)

    def forward(self,x):
        # print(x.shape)
        # flatten image input (-1, num_col*dim_1*dim_2) *NOTE Adding num_col ver
        x = x.view(-1,3*32*32)
        # print(x.shape)
        # add hidden layer, with relu activation function
        x = F.relu(self.fc1(x))
        # add dropout layer
```

```
        x = self.dropout(x)
         # add hidden layer, with relu activation function
        x = F.relu(self.fc2(x))

        # optional dropout
        if self.dropout:
          # add dropout layer
          x = self.dropout(x)
        else:
          x = self.dropout(x)

        # add output layer
        x = self.fc3(x)
        return x
```

3. Train each model

- Prelim - Create a pandas array to store all the information re: training and testing
- Prelim - Define a loss function and optimizer

1. First, train the simplest neural net
   - with regularization
   - without regularization
2. Then, test the more complicated neural net (with convolution)
   - with regularization
   - without regularization
3. Save the data

In [7]:
```
# Prelim - Create a pandas array to store all the information re: training and t
model_df = pd.DataFrame(columns=['model_name', 'net', 'train_err', 'test_error',
```

Prelim

In [53]:
```
# Prelim define loss function, optimizer, and other hyperparameters
lr = 0.001
momentum = 0.9
n_epochs = 5

# L2
L2_penalty = True # the L2 penalty is par tof the optimizer
L2_constant = 1e-2 # some L2 constant > 0

# dropout
dropout=False
do_const=0.5

# hidden layers
hidden = 50

# set up net
net_in = Net3(hidden=hidden, dropout=dropout, do_const=do_const)
print(net_in)

# controls optimization
criterion = nn.CrossEntropyLoss()
if L2_penalty: # add regularization if desired
```

```
  optimizer = optim.SGD(net_in.parameters(), lr=0.001, momentum=0.9, weight_deca
else:
  optimizer = optim.SGD(net_in.parameters(), lr=0.001, momentum=0.9, weight_deca

# assembles list of loss for plotting
loss_list = []

# creates model name
model_name = str(hidden)+'HLN+'+'L2='+str(L2_penalty)
print(model_name)
```

```
Net3(
  (fc1): Linear(in_features=3072, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=50, bias=True)
  (fc3): Linear(in_features=50, out_features=10, bias=True)
  (droput): Dropout(p=0, inplace=False)
)
50HLN+L2=True
```

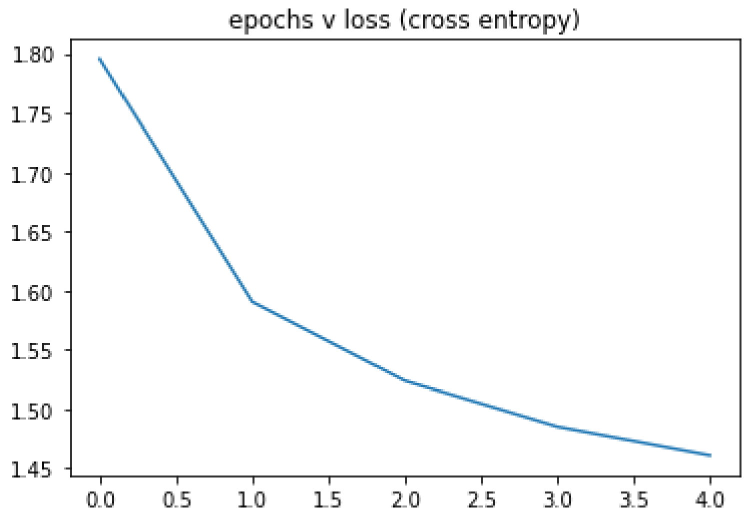
4. Train the neural Net

In [54]:

```
# training
for epoch in range(n_epochs):  # loop over the dataset multiple times

    running_loss = 0.0 # keep track of loss within each epoch

    for i, data in enumerate(train_loader, 0): # loop through batches in train

        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net_in(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # update loss statistics
        running_loss += loss.item()

    # print and save loss statistics
    print(running_loss / i)
    loss_list.append(running_loss / i)
    running_loss = 0.0


# final things
print('Finished Training')

# save some results

# plot
plt.plot(np.arange(0,len(loss_list)), loss_list)
plt.title('epochs v loss (cross entropy)')
plt.show()
```

```
1.7955943031260673
```

```
1.5903667818309593
1.5236713025384492
1.4848844187081118
1.4606699741808085
Finished Training
```



5. test and

6. save

In [55]:

```
# train
correct = 0
total = 0
with torch.no_grad():
    for data in train_loader:
        images, labels = data
        outputs = net_in(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

# create train error, where train error is the percent mis-identified (expressed
train_err = 1 - (correct / total)

# test
correct = 0
total = 0
with torch.no_grad():
    for data in test_loader:
        images, labels = data
        outputs = net_in(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

# create test error, where test error is the percent mis-identified (expressed a
test_err = 1 - (correct / total)
```

In [58]:

```
model_df = model_df.append({'model_name': model_name, 'net': net_in,
                            'train_err': train_err, 'test_error': test_err, 'mod
```

PLOT

In [80]:
```
model_df[['model_name', 'train_err', 'test_error', 'L2_penalty', 'num_nodes', '
```

Out[80]:

	model_name	train_err	test_error	L2_penalty	num_nodes	num_layers
1	200HLN+L2=False	0.39780	0.4666	0.00	200	2
2	200HLN+L2=True	0.47864	0.5067	0.01	200	2
3	200HLN+L2=True	0.90000	0.9000	0.50	200	2
4	50HLN+L2=False	0.45254	0.4956	0.00	50	2
5	50HLN+L2=True	0.49738	0.5181	0.01	50	2
6	50HLN+L2=True	0.90000	0.9000	0.50	50	2
7	50HLN+L2=True	0.90000	0.9000	0.50	50	2
8	5HLN+L2=False	0.65572	0.6646	0.00	5	2

In [98]:
```
i = 0
for model in model_df['model_dict']:
  i = i + 1
  print('Model', i)
  for param_tensor in model:
      print(param_tensor, "\t", model[param_tensor].size())

  print()


i = 0
for optimizer in model_df['optim_dict']:
  i = i + 1
  print('Model', i)
  for var_name in optimizer:
      if var_name == 'state':
        a = 1
      else:
        print((var_name), "\t", optimizer[var_name])

  print()
```

```
Model 1
fc1.weight       torch.Size([200, 3072])
fc1.bias         torch.Size([200])
fc2.weight       torch.Size([200, 200])
fc2.bias         torch.Size([200])
fc3.weight       torch.Size([10, 200])
fc3.bias         torch.Size([10])

Model 2
fc1.weight       torch.Size([200, 3072])
fc1.bias         torch.Size([200])
fc2.weight       torch.Size([200, 200])
fc2.bias         torch.Size([200])
fc3.weight       torch.Size([10, 200])
fc3.bias         torch.Size([10])

Model 3
fc1.weight       torch.Size([200, 3072])
fc1.bias         torch.Size([200])
fc2.weight       torch.Size([200, 200])
fc2.bias         torch.Size([200])
fc3.weight       torch.Size([10, 200])
```

```
fc3.bias         torch.Size([10])

Model 4
fc1.weight       torch.Size([50, 3072])
fc1.bias         torch.Size([50])
fc2.weight       torch.Size([50, 50])
fc2.bias         torch.Size([50])
fc3.weight       torch.Size([10, 50])
fc3.bias         torch.Size([10])

Model 5
fc1.weight       torch.Size([50, 3072])
fc1.bias         torch.Size([50])
fc2.weight       torch.Size([50, 50])
fc2.bias         torch.Size([50])
fc3.weight       torch.Size([10, 50])
fc3.bias         torch.Size([10])

Model 6
fc1.weight       torch.Size([50, 3072])
fc1.bias         torch.Size([50])
fc2.weight       torch.Size([50, 50])
fc2.bias         torch.Size([50])
fc3.weight       torch.Size([10, 50])
fc3.bias         torch.Size([10])

Model 7
fc1.weight       torch.Size([50, 3072])
fc1.bias         torch.Size([50])
fc2.weight       torch.Size([50, 50])
fc2.bias         torch.Size([50])
fc3.weight       torch.Size([10, 50])
fc3.bias         torch.Size([10])

Model 8
fc1.weight       torch.Size([5, 3072])
fc1.bias         torch.Size([5])
fc2.weight       torch.Size([5, 5])
fc2.bias         torch.Size([5])
fc3.weight       torch.Size([10, 5])
fc3.bias         torch.Size([10])

Model 1
param_groups     [{'lr': 0.001, 'momentum': 0.9, 'dampening': 0, 'weight_decay':
0, 'nesterov': False, 'params': [0, 1, 2, 3, 4, 5]}]

Model 2
param_groups     [{'lr': 0.001, 'momentum': 0.9, 'dampening': 0, 'weight_decay':
0.01, 'nesterov': False, 'params': [0, 1, 2, 3, 4, 5]}]

Model 3
param_groups     [{'lr': 0.001, 'momentum': 0.9, 'dampening': 0, 'weight_decay':
0.5, 'nesterov': False, 'params': [0, 1, 2, 3, 4, 5]}]

Model 4
param_groups     [{'lr': 0.001, 'momentum': 0.9, 'dampening': 0, 'weight_decay':
0, 'nesterov': False, 'params': [0, 1, 2, 3, 4, 5]}]

Model 5
param_groups     [{'lr': 0.001, 'momentum': 0.9, 'dampening': 0, 'weight_decay':
0.01, 'nesterov': False, 'params': [0, 1, 2, 3, 4, 5]}]

Model 6
param_groups     [{'lr': 0.001, 'momentum': 0.9, 'dampening': 0, 'weight_decay':
0.5, 'nesterov': False, 'params': [0, 1, 2, 3, 4, 5]}]
```

```
Model 7
param_groups     [{'lr': 0.001, 'momentum': 0.9, 'dampening': 0, 'weight_decay':
0.5, 'nesterov': False, 'params': [0, 1, 2, 3, 4, 5]}]

Model 8
param_groups     [{'lr': 0.001, 'momentum': 0.9, 'dampening': 0, 'weight_decay':
0, 'nesterov': False, 'params': [0, 1, 2, 3, 4, 5]}]
```

BONUS. Fun extra tests

In [13]:

```python
# # check that test data are set up
# dataiter = iter(test_loader)
# images, labels = dataiter.next()

# # print images
# imshow(torchvision.utils.make_grid(images))
# print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(batc

# # preliminary check
# outputs = net_in(images)
# _, predicted = torch.max(outputs, 1)

# print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
#                               for j in range(batch_size)))

# # final check
# correct = 0
# total = 0
# with torch.no_grad():
#     for data in test_loader:
#         images, labels = data
#         outputs = net_in(images)
#         _, predicted = torch.max(outputs.data, 1)
#         total += labels.size(0)
#         correct += (predicted == labels).sum().item()

# print('\n Accuracy of the network on the 10000 test images: %d %%' % (
#     100 * correct / total))

# # What are the classes that performed well, and the classes that did not perfo
# print('')
# class_correct = list(0. for i in range(10))
# class_total = list(0. for i in range(10))
# with torch.no_grad():
#     for data in test_loader:
#         images, labels = data
#         outputs = net_in(images)
#         _, predicted = torch.max(outputs, 1)
#         c = (predicted == labels).squeeze()
#         for i in range(4):
#             label = labels[i]
#             class_correct[label] += c[i].item()
#             class_total[label] += 1

# for i in range(10):
#     print('Accuracy of %5s : %2d %%' % (
#         classes[i], 100 * class_correct[i] / class_total[i]))
```

```
GroundTruth:    cat  ship  ship plane  frog  frog   car  frog   cat   car plane
truck
Predicted:    cat truck plane  bird  deer  frog   cat  frog   dog   car plane tr
uck

 Accuracy of the network on the 10000 test images: 53 %

Accuracy of plane : 61 %
Accuracy of   car : 61 %
Accuracy of  bird : 31 %
Accuracy of   cat : 36 %
Accuracy of  deer : 44 %
Accuracy of   dog : 41 %
Accuracy of  frog : 59 %
Accuracy of horse : 66 %
Accuracy of  ship : 69 %
Accuracy of truck : 61 %
```

HW_04 Problem #2

2 Adaboost [20pts]

Write a class that implements the Adaboost algorithm. Your class should be similar to sklearn's in that it should have a `fit` and `predict` method to train and test the classifier, respectively. You should also use the sampling function from Homework #1 to train the weak learning algorithm, which should be a shallow decision tree. The Adaboost class should be compared to sklearn's implementation on datasets from the course Github page.

This Code was written by Robert 'Quinn' Hull, and borrowed elements from several other resources:

- Understanding AdaBoost - https://towardsdatascience.com/understanding-adaboost-2f94f22d5bfe
- SKLearn - https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
- Machine Learning Mastery
- SKLearn - https://scikit-learn.org/stable/auto_examples/ensemble/plot_adaboost_twoclass.html#sphx-glr-auto-examples-ensemble-plot-adaboost-twoclass-py

P2

How to use the sampling distribution function to get your weak learning classifier

① $h_t = \text{WeakLearn}(D_t, \{x_i, y_i\}, \mathcal{H})$ — tree

② $\epsilon_t = \sum_{i \in \text{wrong}} D_t(i)$

1) c = CART(max_depth=4) - generic classifier

2) idx = rsample(D, len(D)) - find indices of data samples you want to train on

3) X_t, y_t = X[idx], y[idx] - take subset of x and get their labels

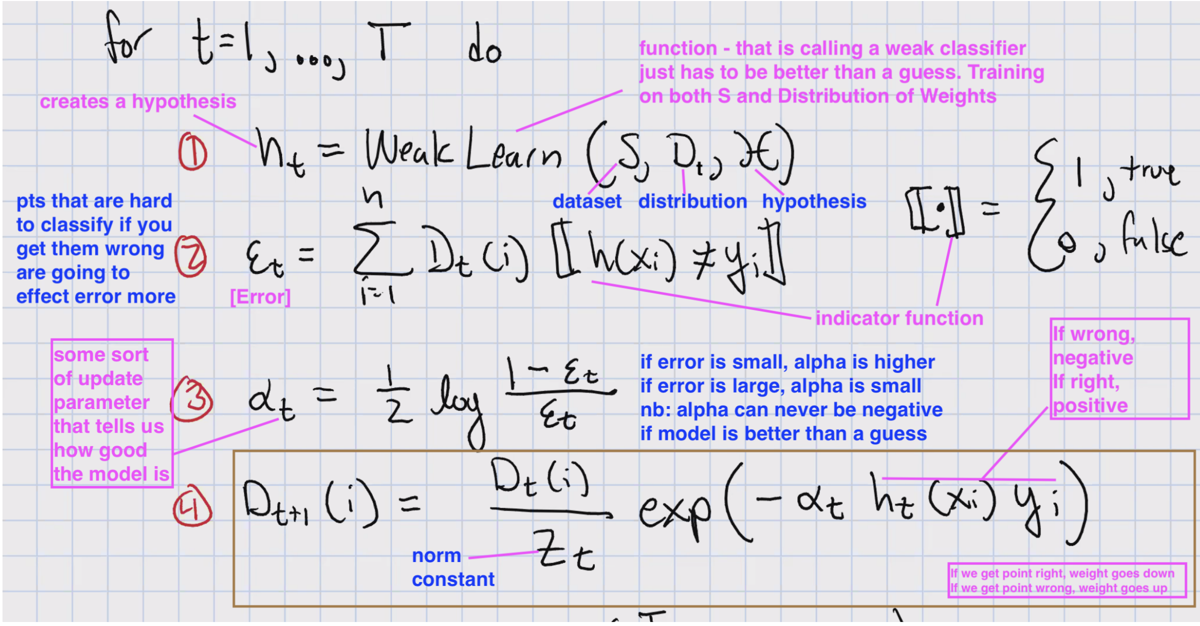
4) h[t] = c.fit(X_t, y_t) - give hypothesis

③ $a_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$

④ $D_{t+1}(i) = \frac{D_t(i)}{Z_t}\exp\{-a_t y_i h_t(x_i)\}$

15:04

15:14



Algorithm 1 Adaboost (Adaptive Boosting)

Input: $\mathcal{S} := \{x_i, y_i\}_{i=1}^{N}$, learning rounds T, and hypothesis class $\mathcal{H}$

Initialize: $\mathcal{D}_1(i) = 1/n$

1: **for** $t = 1, \ldots, T$ **do**

2: $\quad h_t = \textsc{WeakLearn}(h, \mathcal{S}, \mathcal{D}_t)$

3: $\quad \epsilon_t = \sum \mathcal{D}_t(i) [\![h(\mathbf{x}_i) \neq y_i]\!]$

4: $\quad \alpha_t = \frac{1}{2} \log \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$

5: $\quad \mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i)}{Z_t} \exp\left(-\alpha_t y_i h_t(x_i)\right)$

6: **end for**

7: **Output:** $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x) \right)$

AdaBoosting

Pseudocode

- for t=1 to T:
  - 1) execute a weaklearn hypothesis (ht)
    - set generic classifier, c = Cart(max_depth=4)
    - set indices of sample to train, idx = rsample(D, len(D))
    - Take subset of x and get labels, xt, yt = X[idx], y[idx]
    - create hypothesis, h[t] = c.fit(xt, yt)
  - 2) calculate the error (Et)
  - 3) update change parameter
  - 4) reset weights

Modules

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.ensemble import AdaBoostClassifier
```

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import make_gaussian_quantiles
```

Create Data Sets

In [2]:
```python
# Construct dataset
X1, y1 = make_gaussian_quantiles(cov=2.,
                                 n_samples=200, n_features=2,
                                 n_classes=2, random_state=1)
X2, y2 = make_gaussian_quantiles(mean=(3, 3), cov=1.5,
                                 n_samples=300, n_features=2,
                                 n_classes=2, random_state=1)
X = np.concatenate((X1, X2))
y = np.concatenate((y1, - y2 + 1))
y[(y == 0)] = -1 # reset '0' classes to -1
```

AdaBoost

resample

In [3]:
```python
def random(M_in, p_pdf):
  '''
  M_in : the number of samples
  p_pdf : a probability distribution (not cumulative)
  '''
  p_cdf = np.cumsum(p_pdf)
  output = np.zeros(M_in)
  for m in range(M_in): # loop through all elements in array
      p_in = np.random.uniform(0,1) # generate a random probability
      p_i = np.where(p_cdf >= p_in)[0][0] # extract first index of condition
      output[m] = p_i # randomly generate a number in probability interview
  return output
```

WeakLearn

In [4]:
```python
def weaklearn(H, X, y, D):
  '''
  H : Hypothesis Class
  D : Weight Distribution
  X, y : Dataset
  '''
  # set generic classifier
  c = H
  # get indices of samples to train
  idx = random(M_in=len(D), p_pdf=D).astype(np.int16)
  # take subset of x and get labels
  xt, yt = X[idx], y[idx]
  # create hypothesis
  ht = c.fit(xt, yt)
  return ht
```

Error

In [5]:
```python
def Errort(D,ht,X,y):
  '''
  D : distribution of weights
```

```
    ht : weak learning classifier
    X : array of X values (data set)
    y : array of y values (data set)
    '''
    err_t = 0
    for i in range(len(D)):
      # print((ht.predict([X[i]]) != [y[i]]))
      err_i = D[i]*(ht.predict([X[i]]) != [y[i]])
      err_t = err_t + err_i

    return err_t
```

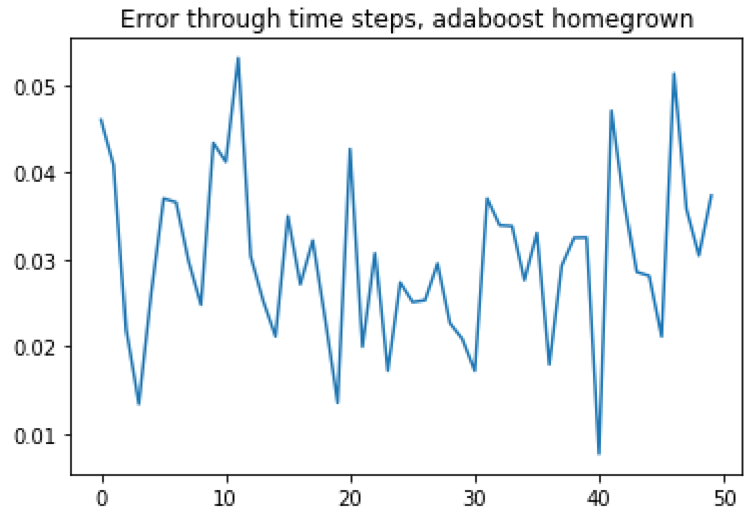
Adaboost Code - homegrown training

In [6]:

```
# Globals
T = 50 # number of rounds
depth = 50 # number of estimators
H = DecisionTreeClassifier(max_depth=depth) # h is a classifier int he hypothesi
n = len(y) # number of samples in data set
N_arr = np.arange(0,n,1) # an array of the different indexes
```

In [7]:

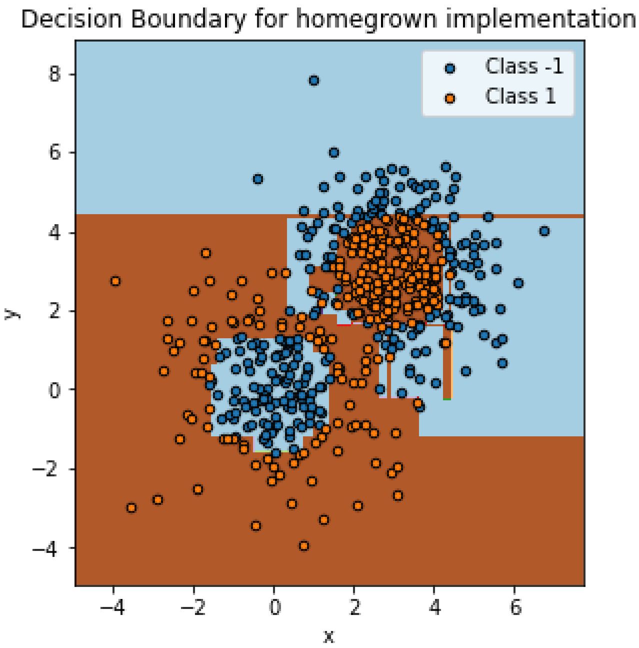
```
err_list = [] # for keeping track of errors
Dt = np.ones(n)*(1/n) # initializing weights, as equal to 1 / number of samples.
# print(Dt.sum())
for t in range(T):
  # generate hypothesis
  ht = weaklearn(H=H,X=X,y=y,D=Dt)
  # print(ht)

  # calculate error
  err_t = Errort(D=Dt,ht=ht,X=X,y=y)
  err_list.append(err_t)
  # print('error is', err_t)

  # calculate update parameter - low error, high alpha
  alpha_t = 0.5*np.log((1-err_t)/err_t)
  # print('alpha is', alpha_t)

  # calculate norm constant - a function of the error, according to lecture note
  Zt = 2*np.sqrt(err_t*(1-err_t))
  # print(Zt)

  # update weights
  Dt = (Dt / Zt) * np.exp(-alpha_t*ht.predict(X)*y)
  # print('Weights summed to', Dt.sum())

plt.plot(np.arange(0,T,1),err_list)
plt.title('Error through time steps, adaboost homegrown')
plt.show()
```



TESTING: Adaboost Code - homegrown

In [9]:
```
y[(y == 0)] = -1

ht.fit(X,y)

plot_step = 0.02

plt.figure(figsize=(10, 5))

# Plot the decision boundaries
plt.subplot(121)
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                     np.arange(y_min, y_max, plot_step))

Z = ht.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
cs = plt.contourf(xx, yy, Z, cmap=plt.cm.Paired)
plt.axis("tight")

# Plot the training points
for i in range(-1,2,2):
    # print(i)
    idx = np.where(y == i)
    # print(idx)
    plt.scatter(X[idx, 0], X[idx, 1],
                cmap=plt.cm.Paired,
                s=20, edgecolor='k',
                label="Class %s" % str(i))
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.legend(loc='upper right')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Decision Boundary for homegrown implementation')
```

Out[9]:
```
Text(0.5, 1.0, 'Decision Boundary for homegrown implementation')
```



TESTING: Adaboost - From SKLearn

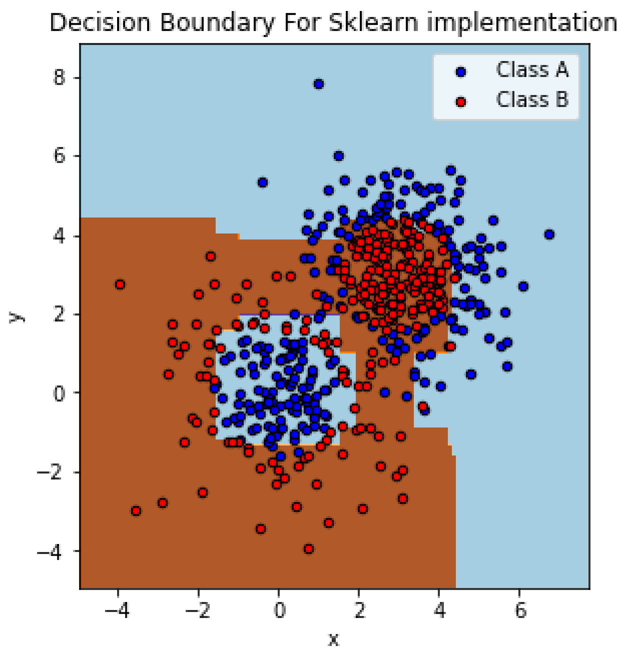
In [10]:
```
print(__doc__)

# Author: Noel Dawe <noel.dawe@gmail.com>
#
# License: BSD 3 clause

y[(y == -1)] = 0

# Create and fit an AdaBoosted decision tree
bdt = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1),
                         algorithm="SAMME",
                         n_estimators=depth)

bdt.fit(X, y)

plot_colors = "br"
plot_step = 0.02
class_names = "AB"

plt.figure(figsize=(10, 5))

# Plot the decision boundaries
plt.subplot(121)
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                     np.arange(y_min, y_max, plot_step))

Z = bdt.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
cs = plt.contourf(xx, yy, Z, cmap=plt.cm.Paired)
plt.axis("tight")

# Plot the training points
for i, n, c in zip(range(2), class_names, plot_colors):
    idx = np.where(y == i)
```

```
    plt.scatter(X[idx, 0], X[idx, 1],
                c=c, cmap=plt.cm.Paired,
                s=20, edgecolor='k',
                label="Class %s" % n)
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.legend(loc='upper right')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Decision Boundary For Sklearn implementation')
```

```
Automatically created module for IPython interactive environment
```

Out[10]:

```
Text(0.5, 1.0, 'Decision Boundary For Sklearn implementation')
```



In [ ]:

HW_04 Problem #3

3 Recurrent Neural Networks for Languange Modeling [20pts]

Read "LSTM: A Search Space Odyssey" (https://arxiv.org/abs/1503.04069). One application of an RNN is the ability model language, which is what your phone does when it is predicting the top three words when you're texting. In this problem, you will need to build a language model.

You are encouraged to start out with the code here. While this code will implement a language model, you are required to modify the code to attempt to beat the baseline for the experiments they have implemented. For example, one modification would be to train multiple language models and average, or weight, their outputs to generate language. Write a couple of paragraphs about what you did and if the results improve the model over the baseline on Github.

This Code was written by Robert 'Quinn' Hull, and borrowed elements from several other resources:

- The bulk of this script is the TensorFlow article about text generation: https://www.tensorflow.org/tutorials/text/text_generation
- Much of the text was removed to make this shorter and more intelligible
- The text describing my work is available at the end of the script.

In [212…

```
#@title Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```

Setup

Import TensorFlow and other libraries

In [265…

```
import tensorflow as tf
from tensorflow.keras.layers.experimental import preprocessing

import numpy as np
import os
import time
```

Download the Shakespeare dataset

In [266…

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

Read the data

First, look in the text:

In [267…

```
# Read, then decode for py2 compat.
text = open(path_to_file, 'rb').read().decode(encoding='utf-8')
# # subset training text (temporary)
# text = text[:11153]
# length of text is the number of characters in it
print('Length of text: {} characters'.format(len(text)))
# The unique characters in the file
vocab = sorted(set(text))
print('{} unique characters'.format(len(vocab)))
```

```
Length of text: 1115394 characters
65 unique characters
```

Process the text

Vectorize the text

In [268…

```
example_texts = ['abcdefg', 'xyz']

chars = tf.strings.unicode_split(example_texts, input_encoding='UTF-8')
print('characters in ', chars)

ids_from_chars = preprocessing.StringLookup(
    vocabulary=list(vocab))

ids = ids_from_chars(chars)
ids

print('IDs ', ids)

chars_from_ids = tf.keras.layers.experimental.preprocessing.StringLookup(
    vocabulary=ids_from_chars.get_vocabulary(), invert=True)

chars = chars_from_ids(ids)
chars

print('characters out ', chars)

# 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)
```

```
characters in  <tf.RaggedTensor [[b'a', b'b', b'c', b'd', b'e', b'f', b'g'],
[b'x', b'y', b'z']]>
IDs  <tf.RaggedTensor [[41, 42, 43, 44, 45, 46, 47], [64, 65, 66]]>
characters out  <tf.RaggedTensor [[b'a', b'b', b'c', b'd', b'e', b'f', b'g'],
[b'x', b'y', b'z']]>
```

Create training examples, targets, batches

In [269…

```
# Create examples and targets
all_ids = ids_from_chars(tf.strings.unicode_split(text, 'UTF-8'))
ids_dataset = tf.data.Dataset.from_tensor_slices(all_ids)

# set constants
seq_length = 100
examples_per_epoch = len(text)//(seq_length+1)
```

```
# set batches
sequences = ids_dataset.batch(seq_length+1, drop_remainder=True)

# split dataset
def split_input_target(sequence):
    input_text = sequence[:-1]
    target_text = sequence[1:]
    return input_text, target_text

dataset = sequences.map(split_input_target)

# create training batches
# 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[269…

```
<PrefetchDataset shapes: ((64, 100), (64, 100)), types: (tf.int64, tf.int64)>
```

Build The Model

In [285…

```
# models
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,
                                   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

# new models, an LSTM
class NewModel1(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.lstm = tf.keras.layers.LSTM(rnn_units,
                                     return_sequences=True,
                                     return_state=False) # change from GRU to LSTM
    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)
    # print(states)
    # print(training)
    if states is None:
      states = self.lstm.get_initial_state(x)
    # print(states)
    x = self.lstm(x) # , initial_state=states) # , training=training)
    x = self.dense(x, training=training)

    if return_state:
      return x, states
    else:
      return x

# new models2, an LSTM
class NewModel2(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.lstm = tf.keras.layers.LSTM(rnn_units,
                                     return_sequences=True,
                                     return_state=True) # change from GRU to LSTM
    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)
    # print(states)
    # print(training)
    if states is None:
      states = self.lstm.get_initial_state(x)
    # print(states)
    x = self.lstm(x , initial_state=states, training=training)
    x = self.dense(x, training=training)

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

In [286…

```
# model chars
# Length of the vocabulary in chars
vocab_size = len(vocab) # QH NOTE: this = 65.

# The embedding dimension
embedding_dim = 256

# Number of RNN units
rnn_units = int(1024) # 1024
```

```
# set model
model = NewModel2(
    # 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)

# optimizer and loss function
loss = tf.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer='adam', loss=loss)

# set number of epochs
EPOCHS = 10
```

Training

Easy Train

In [287…

```
# 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)

# train model (naive)
history = model.fit(dataset, epochs=EPOCHS, callbacks=[checkpoint_callback])
```

```
Epoch 1/10
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-287-28b3bf9f2177> in <module>()
      9 
     10 # train model (naive)
---> 11 history = model.fit(dataset, epochs=EPOCHS, callbacks=[checkpoint_callba
ck])

/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.p
y in fit(self, x, y, batch_size, epochs, verbose, callbacks, validation_split, v
alidation_data, shuffle, class_weight, sample_weight, initial_epoch, steps_per_e
poch, validation_steps, validation_batch_size, validation_freq, max_queue_size,
 workers, use_multiprocessing)
   1098                 _r=1):
   1099               callbacks.on_train_batch_begin(step)
-> 1100               tmp_logs = self.train_function(iterator)
   1101               if data_handler.should_sync:
   1102                 context.async_wait()

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/def_function.py i
n __call__(self, *args, **kwds)
    826     tracing_count = self.experimental_get_tracing_count()
    827     with trace.Trace(self._name) as tm:
--> 828       result = self._call(*args, **kwds)
    829       compiler = "xla" if self._experimental_compile else "nonXla"
    830       new_tracing_count = self.experimental_get_tracing_count()

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/def_function.py i
n _call(self, *args, **kwds)
    869       # This is the first call of __call__, so we have to initialize.
```

```
    870       initializers = []
--> 871       self._initialize(args, kwds, add_initializers_to=initializers)
    872     finally:
    873       # At this point we know that the initialization is complete (or le
ss

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/def_function.py i
n _initialize(self, args, kwds, add_initializers_to)
    724     self._concrete_stateful_fn = (
    725         self._stateful_fn._get_concrete_function_internal_garbage_collec
ted(  # pylint: disable=protected-access
--> 726             *args, **kwds))
    727 
    728     def invalid_creator_scope(*unused_args, **unused_kwds):

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/function.py in _g
et_concrete_function_internal_garbage_collected(self, *args, **kwargs)
   2967       args, kwargs = None, None
   2968     with self._lock:
-> 2969       graph_function, _ = self._maybe_define_function(args, kwargs)
   2970     return graph_function
   2971 

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/function.py in _m
aybe_define_function(self, args, kwargs)
   3359 
   3360           self._function_cache.missed.add(call_context_key)
-> 3361           graph_function = self._create_graph_function(args, kwargs)
   3362           self._function_cache.primary[cache_key] = graph_function
   3363 

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/function.py in _c
reate_graph_function(self, args, kwargs, override_flat_arg_shapes)
   3204             arg_names=arg_names,
   3205             override_flat_arg_shapes=override_flat_arg_shapes,
-> 3206             capture_by_value=self._capture_by_value),
   3207         self._function_attributes,
   3208         function_spec=self.function_spec,

/usr/local/lib/python3.7/dist-packages/tensorflow/python/framework/func_graph.py
in func_graph_from_py_func(name, python_func, args, kwargs, signature, func_grap
h, autograph, autograph_options, add_control_dependencies, arg_names, op_return_
value, collections, capture_by_value, override_flat_arg_shapes)
    988         _, original_func = tf_decorator.unwrap(python_func)
    989 
--> 990       func_outputs = python_func(*func_args, **func_kwargs)
    991 
    992       # invariant: `func_outputs` contains only Tensors, CompositeTensor
s,

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/def_function.py i
n wrapped_fn(*args, **kwds)
    632             xla_context.Exit()
    633         else:
--> 634           out = weak_wrapped_fn().__wrapped__(*args, **kwds)
    635         return out
    636 

/usr/local/lib/python3.7/dist-packages/tensorflow/python/framework/func_graph.py
in wrapper(*args, **kwargs)
    975           except Exception as e:  # pylint:disable=broad-except
    976             if hasattr(e, "ag_error_metadata"):
--> 977               raise e.ag_error_metadata.to_exception(e)
    978             else:
    979               raise
```

```
ValueError: in user code:

    /usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/traini
ng.py:805 train_function  *
        return step_function(self, iterator)
    <ipython-input-279-ea55be33f8d6>:69 call  *
        x = self.dense(x, training=training)
    /usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/base_l
ayer.py:998 __call__  **
        input_spec.assert_input_compatibility(self.input_spec, inputs, self.nam
e)
    /usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/input_
spec.py:207 assert_input_compatibility
        ' input tensors. Inputs received: ' + str(inputs))

    ValueError: Layer dense_28 expects 1 input(s), but it received 3 input tenso
rs. Inputs received: [<tf.Tensor 'new_model2_1/lstm_15/PartitionedCall:1' shape=
(64, 100, 1024) dtype=float32>, <tf.Tensor 'new_model2_1/lstm_15/PartitionedCal
l:2' shape=(64, 1024) dtype=float32>, <tf.Tensor 'new_model2_1/lstm_15/Partition
edCall:3' shape=(64, 1024) dtype=float32>]
```

Generate text

The following makes a single step prediction:

In [273…

```
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 [274…

```
temp = 1.0
one_step_model = OneStep(model, chars_from_ids, ids_from_chars, temperature=temp
```

In [275…

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

pred_char_len = 50

for n in range(pred_char_len):
  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(f"\nRun time: {end - start}")
```

```
ROMEO:
Ag:
Ange notifaloulyo ced bleathird thostll?
Thit

________________________________________________________________________________

Run time: 0.8481040000915527
```

Evaluate

Bleu

- From: https://towardsdatascience.com/how-to-evaluate-text-generation-models-metrics-for-automatic-evaluation-of-nlp-models-e1c251b04ec1
- BLEU is a precision focused metric that calculates n-gram overlap of the reference and generated texts.

In [276…

```
# prep
from nltk.translate.bleu_score import SmoothingFunction, corpus_bleu, sentence_b

def bleu(ref, gen):
    '''
    calculate pair wise bleu score. uses nltk implementation
    Args:
        For word comparison
          references : a list of reference sentences
          candidates : a list of candidate(generated) sentences
```

```
        For character comparison
          references : a list of reference sentences
          candidates : a list of candidate(generated) sentences
    Returns:
        bleu score(float)
    '''
    ref_bleu = []
    gen_bleu = []
    print('test1')
    for l in gen:
        print('test2')
        gen_bleu.append(l.split())
    print(gen_bleu)
    for i,l in enumerate(ref):
        print('test3')
        ref_bleu.append([l.split()])
    print(ref_bleu)
    cc = SmoothingFunction()
    print(cc)
    score_bleu = corpus_bleu(ref_bleu, gen_bleu)
    return score_bleu
```

In [277…

```
bleu_score = 'NA' # bleu(['hillo'],['hello'])
```

Final Comments and Text

- In general, the script experiments with:
  - hyperparameters:
    - embedding_dim, rnn_units, temperature, epochs
  - Model Structure
    - LSTM v RNN
- Lingering questions I have:
  - I was mostly unsuccessful at using an LSTM for this. It's definitely in how I am setting up this model.
  - I never figured out how to properly assess this model. I started to experiment with the BLEU algorithm, but I wasn't sure this is built for character generation in this context. I.E., I wasn't sure how to apply this to our use-case where the text generated was mostly 'random'

In [278…

```
cats = ['bleu_score', 'Training loss (final)', 'Training sequence length',
        'Training Buffer Size', 'Training Epochs', 'Training Batch Size',
        'RNN model: RNN units', 'RNN model: Embedding Dim',
        'RNN model: Name of loss function', 'RNN model: Summary', 'RNN model: hi
        'Prediction: character length', 'Prediction: temp constant', 'Prediction
        'Prediction: model object']

print(cats, '\n')

# save1 = [bleu_score, history.history['loss'][-1], seq_length,
#          BUFFER_SIZE, EPOCHS, BATCH_SIZE,
#          rnn_units, embedding_dim,
#          loss.name, model, history,
#          pred_char_len, temp, result,
#          one_step_model]
```

```
print('Model 1 is the `custom` model using the customized model object in the or
      'There is no calculation of bleu score for this one, because I havent figu
      'how to best evaluate the model. \n')
print(save1, '\n\n')

# save2 = [bleu_score, history.history['loss'][-1], seq_length,
#          BUFFER_SIZE, EPOCHS, BATCH_SIZE,
#          rnn_units, embedding_dim,
#          loss.name, model, history,
#          pred_char_len, temp, result,
#          one_step_model]
print('Model 2 is our baseline model using the class MyModel with architecture t
      'From the persective of training loss, it performs nearly as well as the p
      np.round(save2[1],2), 'versus', np.round(save1[1],2), '\n')
print(save2, '\n\n')

# save3 = [bleu_score, history.history['loss'][-1], seq_length,
#          BUFFER_SIZE, EPOCHS, BATCH_SIZE,
#          rnn_units, embedding_dim,
#          loss.name, model, history,
#          pred_char_len, temp, result,
#          one_step_model]
print('Model 3 is our baseline model using the class MyModel with architecture t
      'to highlight the impact of epochs on training performance, we have reduce
      'From the persective of training loss, it performs worse than before',np.r
      'The predicted text is less coherent, too:\n', 'Model 3 :', save3[13].nump
print(save3, '\n\n')

# save4 = [bleu_score, history.history['loss'][-1], seq_length,
#          BUFFER_SIZE, EPOCHS, BATCH_SIZE,
#          rnn_units, embedding_dim,
#          loss.name, model, history,
#          pred_char_len, temp, result,
#          one_step_model]
print('Model 4 is our baseline model using the class MyModel with architecture t
      'to highlight the impact of epochs on training performance, we have increa
      'From the persective of training loss, it performs better than before',np.
      'The predicted text is way more coherent, almost Shakespearean:\n', 'Model
      'This performance bump does come at the expense of time, though (20s / epo
print(save4, '\n\n')

# save5 = [bleu_score, history.history['loss'][-1], seq_length,
#          BUFFER_SIZE, EPOCHS, BATCH_SIZE,
#          rnn_units, embedding_dim,
#          loss.name, model, history,
#          pred_char_len, temp, result,
#          one_step_model]
print('Model 5 is our baseline model using the class MyModel with architecture t
      'to explore ways to speed up training and preserve text coherence, I reduc
      'by two orders of magnitude: 1115394 characters to 11153 characters and ke
      'From the persective of training loss, it performs far worse than before',
      'The predicted text is nonsense :\n', 'Model 6 :', save5[13].numpy()[0], '
      'This might be because of undertraining, or an issue in how weve indexed t
      'This saves training speed substantially! - 1 s / epoch \n')
print(save5, '\n\n')

# save6 = [bleu_score, history.history['loss'][-1], seq_length,
#          BUFFER_SIZE, EPOCHS, BATCH_SIZE,
#          rnn_units, embedding_dim,
#          loss.name, model, history,
#          pred_char_len, temp, result,
```

```
#           one_step_model]
print('Model 6 is our baseline model using the class MyModel, returning to epoch
      'Here we start to vary the architecture of the model, so rnn_units =', sav
      'From the persective of training loss, it performs better than baseline (M
      'The predicted text is probably more coherent :\n', 'Model 6 :', save6[13]
      'Worth noting that the run-time nearly tripled, from 20 s / epoch to 55 s
print(save6, '\n\n')

# save7 = [bleu_score, history.history['loss'][-1], seq_length,
#           BUFFER_SIZE, EPOCHS, BATCH_SIZE,
#           rnn_units, embedding_dim,
#           loss.name, model, history,
#           pred_char_len, temp, result,
#           one_step_model]
print('Model 7 is our baseline model using the class MyModel \n',
      'Where we decrease the rnn_units =', save7[6], 'down from', save6[6], '\n'
      'From the persective of training loss, it performs worse than baseline (Mo
      'The predicted text is arguably no more or less coherent :\n', 'Model 7 :'
      'Worth noting that the run-time didnt change very much \n ')
print(save7, '\n\n')

# save8 = [bleu_score, history.history['loss'][-1], seq_length,
#           BUFFER_SIZE, EPOCHS, BATCH_SIZE,
#           rnn_units, embedding_dim,
#           loss.name, model, history,
#           pred_char_len, temp, result,
#           one_step_model]
print('Model 8 is our first LSTM model using the class NewModel1 \n',
      'Where we return the rnn_units =', save8[6], '\n',
      'From the persective of training loss, it performs _____ than baseline (Mo
      'The predicted text is totally incoherent :\n', 'Model 8 :', save8[13].num
      'I have definitely made an error in how I set up this LSTM network \n ')
print(save8, '\n\n')

save9 = [bleu_score, history.history['loss'][-1], seq_length,
         BUFFER_SIZE, EPOCHS, BATCH_SIZE,
         rnn_units, embedding_dim,
         loss.name, model, history,
         pred_char_len, temp, result,
         one_step_model]
print('Model 9 is our 2nd LSTM model using the class NewModel2 \n',
      'Ive tried to mess around with the model so that it passes the state\n',
      'From the persective of training loss, it performs _____ than baseline (Mo
      'The predicted text is totally incoherent :\n', 'Model 9 :', save9[13].num
      'I have definitely made an error in how I set up this LSTM network \n ')
print(save9, '\n\n')
```

```
['bleu_score', 'Training loss (final)', 'Training sequence length', 'Training Bu
ffer Size', 'Training Epochs', 'Training Batch Size', 'RNN model: RNN units', 'R
NN model: Embedding Dim', 'RNN model: Name of loss function', 'RNN model: Summar
y', 'RNN model: history object', 'Prediction: character length', 'Prediction: te
mp constant', 'Prediction: result', 'Prediction: model object']

Model 1 is the `custom` model using the customized model object in the original
code
 There is no calculation of bleu score for this one, because I havent figured ou
t at this point
 how to best evaluate the model.

['NA', 1.1910383701324463, 100, 10000, 10, 64, 1024, 256, 'sparse_categorical_cr
ossentropy', <__main__.CustomTraining object at 0x7efee66c6cd0>, <tensorflow.pyt
hon.keras.callbacks.History object at 0x7eff4b279410>, 50, 1.0, <tf.Tensor: shap
```

```
e=(1,), dtype=string, numpy=
array([b'ROMEO:\nO, thy sun doth let\nRend her he should he knife.\n'],
      dtype=object)>, <__main__.OneStep object at 0x7efea8986a50>]


Model 2 is our baseline model using the class MyModel with architecture the same
as in the original code
 From the persective of training loss, it performs nearly as well as the previou
s custom model
 1.2 versus 1.19

['NA', 1.2036826610565186, 100, 10000, 10, 64, 1024, 256, 'sparse_categorical_cr
ossentropy', <__main__.MyModel object at 0x7efee66da950>, <tensorflow.python.ker
as.callbacks.History object at 0x7efee63f06d0>, 50, 1.0, <tf.Tensor: shape=(1,),
dtype=string, numpy=
array([b'ROMEO:\nUntim life is he not to guess.\n\nMARCIUS:\nIs there'],
      dtype=object)>, <__main__.OneStep object at 0x7efea7e72810>]


Model 3 is our baseline model using the class MyModel with architecture the same
as in the original code
 to highlight the impact of epochs on training performance, we have reduced it f
rom 10 to 2
 From the persective of training loss, it performs worse than before 2.0 versus
1.2
 The predicted text is less coherent, too:
 Model 3 : b'ROMEO:\nThe wontely he soum, as\nI sape!\nAnd swant wean I '
 Model 2 : b'ROMEO:\nUntim life is he not to guess.\n\nMARCIUS:\nIs there'

['NA', 2.0047507286071777, 100, 10000, 2, 64, 1024, 256, 'sparse_categorical_cro
ssentropy', <__main__.MyModel object at 0x7efea7c7bf10>, <tensorflow.python.kera
s.callbacks.History object at 0x7efea5fa34d0>, 50, 1.0, <tf.Tensor: shape=(1,),
dtype=string, numpy=
array([b'ROMEO:\nThe wontely he soum, as\nI sape!\nAnd swant wean I '],
      dtype=object)>, <__main__.OneStep object at 0x7efea7986b10>]


Model 4 is our baseline model using the class MyModel with architecture the same
as in the original code
 to highlight the impact of epochs on training performance, we have increased it
from 2 to 30
 From the persective of training loss, it performs better than before 0.48 versu
s 2.0
 The predicted text is way more coherent, almost Shakespearean:
 Model 4 : b'ROMEO:\nThou art well where you will hear none.\n\nGREMIO:\n'
 Model 3 : b'ROMEO:\nThe wontely he soum, as\nI sape!\nAnd swant wean I '
 This performance bump does come at the expense of time, though (20s / epoch * 3
0 epochs = 10 minutes,

['NA', 0.48344069719314575, 100, 10000, 30, 64, 1024, 256, 'sparse_categorical_c
rossentropy', <__main__.MyModel object at 0x7efea7090f50>, <tensorflow.python.ke
ras.callbacks.History object at 0x7efea5fa3250>, 50, 1.0, <tf.Tensor: shape=
(1,), dtype=string, numpy=
array([b'ROMEO:\nThou art well where you will hear none.\n\nGREMIO:\n'],
      dtype=object)>, <__main__.OneStep object at 0x7efea64a4e90>]


Model 5 is our baseline model using the class MyModel with architecture the same
as in the original code
 to explore ways to speed up training and preserve text coherence, I reduce the
size of the input text
 by two orders of magnitude: 1115394 characters to 11153 characters and keep epo
chs = 30
 From the persective of training loss, it performs far worse than before 3.12 ve
rsus 0.48
```

The predicted text is nonsense :
Model 6 : b"ROMEO:hEOl',hbta:zNji t atut sAk :nalrt rey e urn m la t"
Model 5 : b'ROMEO:\nThou art well where you will hear none.\n\nGREMIO:\n'
This might be because of undertraining, or an issue in how weve indexed the vocabulary
This saves training speed substantially! - 1 s / epoch

```
['NA', 3.115530014038086, 100, 10000, 30, 64, 1024, 256, 'sparse_categorical_crossentropy', <__main__.MyModel object at 0x7efea63b7c10>, <tensorflow.python.keras.callbacks.History object at 0x7efea6362390>, 50, 1.0, <tf.Tensor: shape=(1,), dtype=string, numpy=
array([b"ROMEO:hEOl',hbta:zNji t atut sAk :nalrt rey e urn m la t"],
      dtype=object)>, <__main__.OneStep object at 0x7efea60b6850>]
```

Model 6 is our baseline model using the class MyModel, returning to epochs = 10 and characters = 1115394
Here we start to vary the architecture of the model, so rnn_units = 2048 up from 1024
From the persective of training loss, it performs better than baseline (Model 2) 0.97 versus 1.2
The predicted text is probably more coherent :
Model 6 : b'ROMEO:\nThou art not half; he hath not shown but health\na'
Model 2 : b'ROMEO:\nUntim life is he not to guess.\n\nMARCIUS:\nIs there'
Worth noting that the run-time nearly tripled, from 20 s / epoch to 55 s / epoch

```
['NA', 0.9703238010406494, 100, 10000, 10, 64, 2048, 256, 'sparse_categorical_crossentropy', <__main__.MyModel object at 0x7efea4bf0c50>, <tensorflow.python.keras.callbacks.History object at 0x7efea4c0a310>, 50, 1.0, <tf.Tensor: shape=(1,), dtype=string, numpy=
array([b'ROMEO:\nThou art not half; he hath not shown but health\na'],
      dtype=object)>, <__main__.OneStep object at 0x7efea6d12150>]
```

Model 7 is our baseline model using the class MyModel
Where we decrease the rnn_units = 256 down from 2048
From the persective of training loss, it performs worse than baseline (Model 2) 1.43 versus 1.2
The predicted text is arguably no more or less coherent :
Model 7 : b"ROMEO:\nO! therefaleness of 'I thremfort\nWhere is within "
Model 2 : b'ROMEO:\nUntim life is he not to guess.\n\nMARCIUS:\nIs there'
Worth noting that the run-time didnt change very much

```
['NA', 1.4263566732406616, 100, 10000, 10, 64, 256, 256, 'sparse_categorical_crossentropy', <__main__.MyModel object at 0x7efe42a21290>, <tensorflow.python.keras.callbacks.History object at 0x7efe42bf8c90>, 50, 1.0, <tf.Tensor: shape=(1,), dtype=string, numpy=
array([b"ROMEO:\nO! therefaleness of 'I thremfort\nWhere is within "],
      dtype=object)>, <__main__.OneStep object at 0x7efef3be2b10>]
```

Model 8 is our first LSTM model using the class NewModel1
Where we decrease the rnn_units = 1024
From the persective of training loss, it performs _____ than baseline (Model 2) 1.31 versus 1.2
The predicted text is totally incoherent :
Model 8 : b'ROMEO:\nAg:\nAnge notifaloulyo ced bleathird thostll?\nThit'
I have definitely made an error in how I set up this recurrent network

```
['NA', 1.313532829284668, 100, 10000, 10, 64, 1024, 256, 'sparse_categorical_crossentropy', <__main__.NewModel1 object at 0x7efef3f3a090>, <tensorflow.python.keras.callbacks.History object at 0x7efef3ffc090>, 50, 1.0, <tf.Tensor: shape=(1,), dtype=string, numpy=
array([b'ROMEO:\nAg:\nAnge notifaloulyo ced bleathird thostll?\nThit'],
```

```
dtype=object)>, <__main__.OneStep object at 0x7efe42a25810>]
```

In [278…