Google Colaboratory Setup

In [0]:
```
# Install a Drive FUSE wrapper.
# https://github.com/astrada/google-drive-ocamlfuse
!apt-get install -y -qq software-properties-common python-software-properties module-init-tools
!add-apt-repository -y ppa:alessandro-strada/ppa 2>&1 > /dev/null
!apt-get update -qq 2>&1 > /dev/null
!apt-get -y install -qq google-drive-ocamlfuse fuse
```

```
E: Package 'python-software-properties' has no installation candidate
```

In [0]:
```
# Generate auth tokens for Colab
from google.colab import auth
auth.authenticate_user()
```

In [0]:
```
# Generate creds for the Drive FUSE library.
from oauth2client.client import GoogleCredentials
creds = GoogleCredentials.get_application_default()
import getpass
!google-drive-ocamlfuse -headless -id={creds.client_id} -secret={creds.client_secret} < /dev/null 2>&1 | grep URL
vcode = getpass.getpass()
!echo {vcode} | google-drive-ocamlfuse -headless -id={creds.client_id} -secret={creds.client_secret}
```

```
··········
```

In [0]:
```
# Create a directory and mount Google Drive using that directory.
!mkdir -p drive
!google-drive-ocamlfuse drive
```

```
fuse: mountpoint is not empty
fuse: if you are sure this is safe, use the 'nonempty' mount option
```

In [0]:
```
# http://pytorch.org/
from os import path
from wheel.pep425tags import get_abbr_impl, get_impl_ver, get_abi_tag
platform = '{}{}-{}'.format(get_abbr_impl(), get_impl_ver(), get_abi_tag())

accelerator = 'cu80' if path.exists('/opt/bin/nvidia-smi') else 'cpu'

!pip install -q http://download.pytorch.org/whl/{accelerator}/torch-0.4.0-{platform}-linux_x86_64.whl torchvision
```

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

Code

loader.py

In [0]:
```
from torch.utils.data import Dataset, DataLoader
import re
import gzip
from collections import Counter

def clean_str(string):
    string = re.sub(r"[^A-Za-z0-9(),!?\'\`]", " ", string)
    string = re.sub(r"\'s", " \'s", string)
    string = re.sub(r"\'ve", " \'ve", string)
    string = re.sub(r"n\'t", " n\'t", string)
    string = re.sub(r"\'re", " \'re", string)
    string = re.sub(r"\'d", " \'d", string)
    string = re.sub(r"\'ll", " \'ll", string)
    string = re.sub(r",", " , ", string)
    string = re.sub(r"!", " ! ", string)
    string = re.sub(r"\(", " \( ", string)
    string = re.sub(r"\)", " \) ", string)
    string = re.sub(r"\?", " \? ", string)
    string = re.sub(r"\s{2,}", " ", string)
    return string.strip().lower()

class MyData(Dataset):
    def __init__(self, filename, vocab_size):
        total_str = []

        f = open(filename, "r+", encoding='utf-8',errors='ignore')
        for line in f:
            cleaned_str = clean_str(" ".join(line.split('\t')[1:]))
            total_str.append(cleaned_str)
        i
        total_str = " ".join(total_str)
        words = total_str.split()

        self.vocab = dict(Counter(words).most_common(vocab_size-1))
        self.index2word = ['UNK'] + [word for word in self.vocab]
        self.word2index = {word:idx for idx, word in enumerate(self.index2word)}
        self.vocab_size = len(self.index2word)

        # change words not in vocab to 'UNK'
        for i in range(len(words)):
            if words[i] not in self.vocab:
                words[i] = 'UNK'
        # store context_target
        self.context_target = []
        for i in range(0+2, len(words)-2):
            c_t_tuple = (words[i-2:i] + words[i+1:i+3], words[i])
            self.context_target.append(c_t_tuple)


    def __getitem__(self, index):
        context = torch.tensor([self.word2index[word] for word in self.context_target[index][0]])
        target = torch.tensor(self.word2index[self.context_target[index][1]])
        return context, target
    def __len__(self):
        return self.vocab_size
```

In [0]:
```
vocab_size = 10000
filename="drive/reviews_data.txt"
dataset = MyData(filename, vocab_size)

from torch.utils.data import DataLoader
bsz = 32
train_loader = DataLoader(dataset=dataset,
                         batch_size=bsz,
                         shuffle=True)
```

CBOW.py

In [0]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

word2index = dataset.word2index
vocab = dataset.vocab

class CBOW(torch.nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim):
        super(CBOW, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embedding_dim)
        self.hidden = nn.Linear(embedding_dim, hidden_dim)
        self.output = nn.Linear(hidden_dim, vocab_size)

    def forward(self, inputs):
        # print("inputs.shape: ", inputs.shape)
        embeds = self.embeddings(inputs)
        # print("embeds.shape: ", embeds.shape)
        embeds = torch.sum(embeds, dim=1)
        # print("hidden.shape: ", hidden.shape)
        hidden = self.hidden(embeds)
        # print("out.shape: ", out.shape)
        out = self.output(hidden)
        log_probs = F.log_softmax(out, dim=1)
        # print("log_probs.shape: ", log_probs.shape)
        return log_probs

    def get_word_embedding(self, word):
        if word in vocab:
          word_vector = self.embeddings(torch.LongTensor([word2index[word]]).cuda()).view(-1)
        else:
          word_vector = self.embeddings(torch.LongTensor([word2index['UNK']]).cuda()).view(-1)
        return word_vector
```

main.py

Only the best hyperparameter configuration is trained here. Look at the report for other configurations and their resulting losses.

In [0]:
```
from tqdm import tqdm

num_epochs = 40
embedding_size = 300
hidden_size = 150
learning_rate = 0.01

model = CBOW(len(dataset), embedding_size, hidden_size)
model.train()

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=learning_rate, momentum=0.5)

cuda_available = torch.cuda.is_available()

for epoch in range(num_epochs):
  total_loss = 0
  for batch_idx, (context, target) in enumerate(tqdm(train_loader)):

      if cuda_available:
        context = context.cuda()
        target = target.cuda()
        model = model.cuda()

      optimizer.zero_grad()
      output = model(context)
      # target = target.view(-1)
      # print("output: ", output)
      # print("target: ", target)
      # print("torch.max: ", torch.max(output, 1)[1])
      loss = criterion(output, target)
      loss.backward()
      optimizer.step()
      # if batch_idx % 10 == 0:
      total_loss += loss.data

  loss_avg = float(total_loss / len(train_loader))
  print(" {}/{} loss {:.4f}".format(epoch+1, num_epochs, loss_avg))
```

```
100%|██████████| 313/313 [00:01<00:00, 302.09it/s]
 11%|█         | 35/313 [00:00<00:00, 341.99it/s]

 1/40 loss 8.2808

100%|██████████| 313/313 [00:00<00:00, 336.19it/s]
 11%|█         | 34/313 [00:00<00:00, 339.61it/s]

 2/40 loss 6.7534

100%|██████████| 313/313 [00:00<00:00, 336.14it/s]
 11%|█         | 35/313 [00:00<00:00, 344.93it/s]

 3/40 loss 6.0572

100%|██████████| 313/313 [00:00<00:00, 333.30it/s]
 11%|█         | 34/313 [00:00<00:00, 336.40it/s]

 4/40 loss 5.5891

100%|██████████| 313/313 [00:00<00:00, 333.20it/s]
 11%|█         | 35/313 [00:00<00:00, 341.24it/s]

 5/40 loss 5.2183

100%|██████████| 313/313 [00:00<00:00, 334.50it/s]
 11%|█         | 34/313 [00:00<00:00, 335.82it/s]

 6/40 loss 4.8953

100%|██████████| 313/313 [00:00<00:00, 334.09it/s]
 11%|█         | 35/313 [00:00<00:00, 340.82it/s]

 7/40 loss 4.6122

100%|██████████| 313/313 [00:00<00:00, 332.66it/s]
 11%|█         | 34/313 [00:00<00:00, 339.92it/s]

 8/40 loss 4.3513

100%|██████████| 313/313 [00:00<00:00, 334.24it/s]
 11%|█         | 34/313 [00:00<00:00, 339.94it/s]

 9/40 loss 4.1090

100%|██████████| 313/313 [00:00<00:00, 332.96it/s]
 11%|█         | 33/313 [00:00<00:00, 324.09it/s]

 10/40 loss 3.8782

100%|██████████| 313/313 [00:00<00:00, 332.85it/s]
 11%|█         | 34/313 [00:00<00:00, 339.43it/s]

 11/40 loss 3.6659

100%|██████████| 313/313 [00:00<00:00, 332.46it/s]
 11%|█         | 35/313 [00:00<00:00, 341.04it/s]

 12/40 loss 3.4536
```

```
100%|██████████| 313/313 [00:00<00:00, 332.08it/s]
 11%|█         | 35/313 [00:00<00:00, 341.99it/s]

 13/40 loss 3.2625

100%|██████████| 313/313 [00:00<00:00, 333.07it/s]
 11%|█         | 35/313 [00:00<00:00, 343.08it/s]

 14/40 loss 3.0757

100%|██████████| 313/313 [00:00<00:00, 334.77it/s]
 11%|█         | 34/313 [00:00<00:00, 336.11it/s]

 15/40 loss 2.9002

100%|██████████| 313/313 [00:00<00:00, 334.87it/s]
 11%|█         | 35/313 [00:00<00:00, 341.49it/s]

 16/40 loss 2.7356

100%|██████████| 313/313 [00:00<00:00, 333.80it/s]
 11%|█         | 35/313 [00:00<00:00, 341.02it/s]

 17/40 loss 2.5794

100%|██████████| 313/313 [00:00<00:00, 334.62it/s]
 11%|█         | 34/313 [00:00<00:00, 338.70it/s]

 18/40 loss 2.4451

100%|██████████| 313/313 [00:00<00:00, 332.75it/s]
 11%|█         | 34/313 [00:00<00:00, 337.82it/s]

 19/40 loss 2.3157

100%|██████████| 313/313 [00:00<00:00, 333.51it/s]
 11%|█         | 34/313 [00:00<00:00, 337.99it/s]

 20/40 loss 2.1997

100%|██████████| 313/313 [00:00<00:00, 331.76it/s]
 11%|█         | 33/313 [00:00<00:00, 326.91it/s]

 21/40 loss 2.1006

100%|██████████| 313/313 [00:00<00:00, 329.82it/s]
 11%|█         | 33/313 [00:00<00:00, 329.76it/s]

 22/40 loss 2.0055

100%|██████████| 313/313 [00:00<00:00, 334.38it/s]
 11%|█         | 34/313 [00:00<00:00, 338.44it/s]

 23/40 loss 1.9205

100%|██████████| 313/313 [00:00<00:00, 334.69it/s]
 11%|█         | 34/313 [00:00<00:00, 311.94it/s]

 24/40 loss 1.8457
```

```
100%|██████████| 313/313 [00:00<00:00, 331.65it/s]
 11%|█         | 35/313 [00:00<00:00, 342.14it/s]

 25/40 loss 1.7759

100%|██████████| 313/313 [00:00<00:00, 335.20it/s]
 11%|█         | 35/313 [00:00<00:00, 340.35it/s]

 26/40 loss 1.7103

100%|██████████| 313/313 [00:00<00:00, 334.52it/s]
 11%|█         | 34/313 [00:00<00:00, 338.05it/s]

 27/40 loss 1.6581

100%|██████████| 313/313 [00:00<00:00, 332.31it/s]
 11%|█         | 35/313 [00:00<00:00, 342.38it/s]

 28/40 loss 1.6101

100%|██████████| 313/313 [00:00<00:00, 333.91it/s]
 11%|█         | 34/313 [00:00<00:00, 335.81it/s]

 29/40 loss 1.5625

100%|██████████| 313/313 [00:00<00:00, 333.03it/s]
 11%|█         | 34/313 [00:00<00:00, 339.23it/s]

 30/40 loss 1.5210

100%|██████████| 313/313 [00:00<00:00, 331.61it/s]
 11%|█         | 35/313 [00:00<00:00, 345.43it/s]

 31/40 loss 1.4783

100%|██████████| 313/313 [00:00<00:00, 333.71it/s]
 11%|█         | 34/313 [00:00<00:00, 339.93it/s]

 32/40 loss 1.4427

100%|██████████| 313/313 [00:00<00:00, 334.63it/s]
 11%|█         | 33/313 [00:00<00:00, 328.30it/s]

 33/40 loss 1.4087

100%|██████████| 313/313 [00:00<00:00, 333.78it/s]
 11%|█         | 34/313 [00:00<00:00, 338.36it/s]

 34/40 loss 1.3747

100%|██████████| 313/313 [00:00<00:00, 330.43it/s]
 11%|█         | 35/313 [00:00<00:00, 341.60it/s]

 35/40 loss 1.3504

100%|██████████| 313/313 [00:00<00:00, 332.11it/s]
 11%|█         | 34/313 [00:00<00:00, 338.61it/s]

 36/40 loss 1.3174
```

```
100%|██████████| 313/313 [00:00<00:00, 333.77it/s]
 11%|█         | 34/313 [00:00<00:00, 338.18it/s]

 37/40 loss 1.2961

100%|██████████| 313/313 [00:00<00:00, 333.41it/s]
 11%|█         | 35/313 [00:00<00:00, 340.94it/s]

 38/40 loss 1.2697

100%|██████████| 313/313 [00:00<00:00, 333.72it/s]
 11%|█         | 34/313 [00:00<00:00, 337.72it/s]

 39/40 loss 1.2472

100%|██████████| 313/313 [00:00<00:00, 330.78it/s]

 40/40 loss 1.2320
```

In [0]:
```
def get_similarity(word1, word2, model):
  word_vector1 = model.get_word_embedding(word1).view(-1)
  word_vector2 = model.get_word_embedding(word2).view(-1)

  return torch.dot(word_vector1, word_vector2)/(torch.norm(word_vector1) * torch.norm(word_vector2))
```

In [0]:
```
get_similarity('man','woman', model)
```

Out[0]:
```
tensor(1.00000e-02 *
       9.8621, device='cuda:0')
```

In [0]:
```
get_similarity('brother','sister', model)
```

Out[0]:
```
tensor(1.00000e-02 *
       8.9651, device='cuda:0')
```

In [0]:
```
get_similarity('dog','cat', model)
```

Out[0]:
```
tensor(1.00000e-02 *
       7.7766, device='cuda:0')
```

In [0]:
```
get_similarity('chair','throw', model)
```

Out[0]:
```
tensor(1.00000e-02 *
       1.7316, device='cuda:0')
```

In [0]:
```
get_similarity('you','i', model)
```

Out[0]:
```
tensor(1.00000e-02 *
       -9.2692, device='cuda:0')
```

In [0]:
```
import numpy as np

word_emb_mat = model.embeddings.weight.cpu().detach().numpy()
np.savetxt('drive/w_emb_mat.txt', word_emb_mat)
```

In [0]:
```
np.save('drive/word2index.npy', dataset.word2index)
```

In [0]:
```
```