SageMaker Project

May 11, 2021

1 Creating a Sentiment Analysis Web App

1.1 Using PyTorch and SageMaker

Deep Learning Nanodegree Program | Deployment

Now that we have a basic understanding of how SageMaker works we will try to use it to construct a complete project from end to end. Our goal will be to have a simple web page which a user can use to enter a movie review. The web page will then send the review off to our deployed model which will predict the sentiment of the entered review.

1.2 Instructions

Some template code has already been provided for you, and you will need to implement additional functionality to successfully complete this notebook. You will not need to modify the included code beyond what is requested. Sections that begin with '**TODO**' in the header indicate that you need to complete or implement some portion within them. Instructions will be provided for each section and the specifics of the implementation are marked in the code block with a `# TODO: ...` comment. Please be sure to read the instructions carefully!

In addition to implementing code, there will be questions for you to answer which relate to the task and your implementation. Each section where you will answer a question is preceded by a '**Question:**' header. Carefully read each question and provide your answer below the '**Answer:**' header by editing the Markdown cell.

> **Note**: Code and Markdown cells can be executed using the **Shift+Enter** keyboard shortcut. In addition, a cell can be edited by typically clicking it (double-click for Markdown cells) or by pressing **Enter** while it is highlighted.

1.3 General Outline

Recall the general outline for SageMaker projects using a notebook instance.

1. Download or otherwise retrieve the data.
2. Process / Prepare the data.
3. Upload the processed data to S3.
4. Train a chosen model.

5. Test the trained model (typically using a batch transform job).
6. Deploy the trained model.
7. Use the deployed model.

For this project, you will be following the steps in the general outline with some modifications.

First, you will not be testing the model in its own step. You will still be testing the model, however, you will do it by deploying your model and then using the deployed model by sending the test data to it. One of the reasons for doing this is so that you can make sure that your deployed model is working correctly before moving forward.

In addition, you will deploy and use your trained model a second time. In the second iteration you will customize the way that your trained model is deployed by including some of your own code. In addition, your newly deployed model will be used in the sentiment analysis web app.

[1]:
```
# Make sure that we use SageMaker 1.x
!pip install sagemaker==1.72.0
```

```
Collecting sagemaker==1.72.0
  Downloading sagemaker-1.72.0.tar.gz (297 kB)
     || 297 kB 20.7 MB/s eta 0:00:01
Requirement already satisfied: boto3>=1.14.12 in
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from
sagemaker==1.72.0) (1.17.68)
Requirement already satisfied: numpy>=1.9.0 in
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from
sagemaker==1.72.0) (1.19.5)
Requirement already satisfied: protobuf>=3.1 in
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from
sagemaker==1.72.0) (3.15.2)
Requirement already satisfied: scipy>=0.19.0 in
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from
sagemaker==1.72.0) (1.5.3)
Requirement already satisfied: protobuf3-to-dict>=0.1.5 in
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from
sagemaker==1.72.0) (0.1.5)
Collecting smdebug-rulesconfig==0.1.4
  Downloading smdebug_rulesconfig-0.1.4-py2.py3-none-any.whl (10 kB)
Requirement already satisfied: importlib-metadata>=1.4.0 in
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from
sagemaker==1.72.0) (3.7.0)
Requirement already satisfied: packaging>=20.0 in
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from
sagemaker==1.72.0) (20.9)
Requirement already satisfied: s3transfer<0.5.0,>=0.4.0 in
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from
boto3>=1.14.12->sagemaker==1.72.0) (0.4.2)
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from
boto3>=1.14.12->sagemaker==1.72.0) (0.10.0)
```

```
Requirement already satisfied: botocore<1.21.0,>=1.20.68 in
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from
boto3>=1.14.12->sagemaker==1.72.0) (1.20.68)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from
botocore<1.21.0,>=1.20.68->boto3>=1.14.12->sagemaker==1.72.0) (2.8.1)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from
botocore<1.21.0,>=1.20.68->boto3>=1.14.12->sagemaker==1.72.0) (1.26.4)
Requirement already satisfied: zipp>=0.5 in
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from
importlib-metadata>=1.4.0->sagemaker==1.72.0) (3.4.0)
Requirement already satisfied: typing-extensions>=3.6.4 in
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from
importlib-metadata>=1.4.0->sagemaker==1.72.0) (3.7.4.3)
Requirement already satisfied: pyparsing>=2.0.2 in
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from
packaging>=20.0->sagemaker==1.72.0) (2.4.7)
Requirement already satisfied: six>=1.9 in
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from
protobuf>=3.1->sagemaker==1.72.0) (1.15.0)
Building wheels for collected packages: sagemaker
  Building wheel for sagemaker (setup.py) ... done
  Created wheel for sagemaker: filename=sagemaker-1.72.0-py2.py3-none-
any.whl size=386358
sha256=8b54272d8a93db4a0b61d66153b89afe8f2ecc32f86a798229af7532ef3a177e
  Stored in directory: /home/ec2-user/.cache/pip/wheels/c3/58/70/85faf4437568bfa
a4c419937569ba1fe54d44c5db42406bbd7
Successfully built sagemaker
Installing collected packages: smdebug-rulesconfig, sagemaker
  Attempting uninstall: smdebug-rulesconfig
    Found existing installation: smdebug-rulesconfig 1.0.1
    Uninstalling smdebug-rulesconfig-1.0.1:
      Successfully uninstalled smdebug-rulesconfig-1.0.1
  Attempting uninstall: sagemaker
    Found existing installation: sagemaker 2.39.1
    Uninstalling sagemaker-2.39.1:
      Successfully uninstalled sagemaker-2.39.1
Successfully installed sagemaker-1.72.0 smdebug-rulesconfig-0.1.4
```

1.4 Step 1: Downloading the data

As in the XGBoost in SageMaker notebook, we will be using the IMDb dataset

> Maas, Andrew L., et al. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2011.

[2]:
```
%mkdir ../data
!wget -O ../data/aclImdb_v1.tar.gz http://ai.stanford.edu/~amaas/data/sentiment/
 ↪aclImdb_v1.tar.gz
!tar -zxf ../data/aclImdb_v1.tar.gz -C ../data
```

```
--2021-05-12 00:43:28--
http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
Resolving ai.stanford.edu (ai.stanford.edu)... 171.64.68.10
Connecting to ai.stanford.edu (ai.stanford.edu)|171.64.68.10|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 84125825 (80M) [application/x-gzip]
Saving to: ../data/aclImdb_v1.tar.gz

../data/aclImdb_v1. 100%[===================>]  80.23M  24.5MB/s    in 3.9s

2021-05-12 00:43:32 (20.8 MB/s) - ../data/aclImdb_v1.tar.gz saved
[84125825/84125825]
```

1.5 Step 2: Preparing and Processing the data

Also, as in the XGBoost notebook, we will be doing some initial data processing. The first few steps are the same as in the XGBoost example. To begin with, we will read in each of the reviews and combine them into a single input structure. Then, we will split the dataset into a training set and a testing set.

[3]:
```
import os
import glob

def read_imdb_data(data_dir='../data/aclImdb'):
    data = {}
    labels = {}

    for data_type in ['train', 'test']:
        data[data_type] = {}
        labels[data_type] = {}

        for sentiment in ['pos', 'neg']:
            data[data_type][sentiment] = []
            labels[data_type][sentiment] = []

            path = os.path.join(data_dir, data_type, sentiment, '*.txt')
            files = glob.glob(path)

            for f in files:
                with open(f) as review:
                    data[data_type][sentiment].append(review.read())
                    # Here we represent a positive review by '1' and a negative␣
↪review by '0'
                    labels[data_type][sentiment].append(1 if sentiment == 'pos'␣
↪else 0)

            assert len(data[data_type][sentiment]) ==␣
↪len(labels[data_type][sentiment]), \
                    "{}/{} data size does not match labels size".
↪format(data_type, sentiment)

    return data, labels
```

[4]:
```
data, labels = read_imdb_data()
print("IMDB reviews: train = {} pos / {} neg, test = {} pos / {} neg".format(
            len(data['train']['pos']), len(data['train']['neg']),
            len(data['test']['pos']), len(data['test']['neg'])))
```

```
IMDB reviews: train = 12500 pos / 12500 neg, test = 12500 pos / 12500 neg
```

Now that we've read the raw training and testing data from the downloaded dataset, we will combine the positive and negative reviews and shuffle the resulting records.

[5]:
```
from sklearn.utils import shuffle

def prepare_imdb_data(data, labels):
    """Prepare training and test sets from IMDb movie reviews."""

    #Combine positive and negative reviews and labels
    data_train = data['train']['pos'] + data['train']['neg']
    data_test = data['test']['pos'] + data['test']['neg']
    labels_train = labels['train']['pos'] + labels['train']['neg']
    labels_test = labels['test']['pos'] + labels['test']['neg']

    #Shuffle reviews and corresponding labels within training and test sets
    data_train, labels_train = shuffle(data_train, labels_train)
    data_test, labels_test = shuffle(data_test, labels_test)

    # Return a unified training data, test data, training labels, test labets
    return data_train, data_test, labels_train, labels_test
```

[6]:
```
train_X, test_X, train_y, test_y = prepare_imdb_data(data, labels)
print("IMDb reviews (combined): train = {}, test = {}".format(len(train_X),␣
↪len(test_X)))
```

```
IMDb reviews (combined): train = 25000, test = 25000
```

Now that we have our training and testing sets unified and prepared, we should do a quick check and see an example of the data our model will be trained on. This is generally a good idea as it allows you to see how each of the further processing steps affects the reviews and it also ensures that the data has been loaded correctly.

[8]:
```python
print(train_X[100])
print(train_y[100])
```

```
This one is a little better than the first one. It still relies on a lot of its
humor which basically keeps saying that the old Bond movies were not realistic.
That wears thin after so many parodies. The girls were more interesting in this
one.<br /><br />There is a tremendous amount of total gross out humor. Hopefully
one day real comedy will come back.
0
```

The first step in processing the reviews is to make sure that any html tags that appear should be removed. In addition we wish to tokenize our input, that way words such as *entertained* and *entertaining* are considered the same with regard to sentiment analysis.

[9]:
```python
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import *

import re
from bs4 import BeautifulSoup

def review_to_words(review):
    nltk.download("stopwords", quiet=True)
    stemmer = PorterStemmer()

    text = BeautifulSoup(review, "html.parser").get_text() # Remove HTML tags
    text = re.sub(r"[^a-zA-Z0-9]", " ", text.lower()) # Convert to lower case
    words = text.split() # Split string into words
    words = [w for w in words if w not in stopwords.words("english")] # Remove␣
 ↪stopwords
    words = [PorterStemmer().stem(w) for w in words] # stem

    return words
```

The `review_to_words` method defined above uses `BeautifulSoup` to remove any html tags that appear and uses the `nltk` package to tokenize the reviews. As a check to ensure we know how everything is working, try applying `review_to_words` to one of the reviews in the training set.

[10]:
```python
# TODO: Apply review_to_words to a review (train_X[100] or any other review)
print(review_to_words(train_X[100]))
```

```
['one', 'littl', 'better', 'first', 'one', 'still', 'reli', 'lot', 'humor',
'basic', 'keep', 'say', 'old', 'bond', 'movi', 'realist', 'wear', 'thin',
'mani', 'parodi', 'girl', 'interest', 'one', 'tremend', 'amount', 'total',
'gross', 'humor', 'hope', 'one', 'day', 'real', 'comedi', 'come', 'back']
```

Question: Above we mentioned that `review_to_words` method removes html formatting and allows us to tokenize the words found in a review, for example, converting *entertained* and *entertaining* into *entertain* so that they are treated as though they are the same word. What else, if anything, does this method do to the input?

Answer: lower case all words; split sentence strings into single word string list; removes punctuation such as '.'; ',','"','¿; removes pronouns such as 'I', 'it', 'you'.

The method below applies the `review_to_words` method to each of the reviews in the training and testing datasets. In addition it caches the results. This is because performing this processing step can take a long time. This way if you are unable to complete the notebook in the current session, you can come back without needing to process the data a second time.

[11]:
```
import pickle

cache_dir = os.path.join("../cache", "sentiment_analysis")  # where to store␣
 ↪cache files
os.makedirs(cache_dir, exist_ok=True)  # ensure cache directory exists

def preprocess_data(data_train, data_test, labels_train, labels_test,
                    cache_dir=cache_dir, cache_file="preprocessed_data.pkl"):
    """Convert each review to words; read from cache if available."""

    # If cache_file is not None, try to read from it first
    cache_data = None
    if cache_file is not None:
        try:
            with open(os.path.join(cache_dir, cache_file), "rb") as f:
                cache_data = pickle.load(f)
            print("Read preprocessed data from cache file:", cache_file)
        except:
            pass  # unable to read from cache, but that's okay

    # If cache is missing, then do the heavy lifting
    if cache_data is None:
        # Preprocess training and test data to obtain words for each review
        #words_train = list(map(review_to_words, data_train))
        #words_test = list(map(review_to_words, data_test))
        words_train = [review_to_words(review) for review in data_train]
        words_test = [review_to_words(review) for review in data_test]

        # Write to cache file for future runs
        if cache_file is not None:
            cache_data = dict(words_train=words_train, words_test=words_test,
                              labels_train=labels_train,␣
 ↪labels_test=labels_test)
            with open(os.path.join(cache_dir, cache_file), "wb") as f:
                pickle.dump(cache_data, f)
            print("Wrote preprocessed data to cache file:", cache_file)
    else:
        # Unpack data loaded from cache file
        words_train, words_test, labels_train, labels_test =␣
 ↪(cache_data['words_train'],
                cache_data['words_test'], cache_data['labels_train'],␣
 ↪cache_data['labels_test'])

    return words_train, words_test, labels_train, labels_test
```

[12]:
```
# Preprocess data
train_X, test_X, train_y, test_y = preprocess_data(train_X, test_X, train_y,␣
 ↪test_y)
```

```
Wrote preprocessed data to cache file: preprocessed_data.pkl
```

1.6 Transform the data

In the XGBoost notebook we transformed the data from its word representation to a bag-of-words feature representation. For the model we are going to construct in this notebook we will construct a feature representation which is very similar. To start, we will represent each word as an integer. Of course, some of the words that appear in the reviews occur very infrequently and so likely don't contain much information for the purposes of sentiment analysis. The way we will deal with this problem is that we will fix the size of our working vocabulary and we will only include the words that appear most frequently. We will then combine all of the infrequent words into a single category and, in our case, we will label it as `1`.

Since we will be using a recurrent neural network, it will be convenient if the length of each review is the same. To do this, we will fix a size for our reviews and then pad short reviews with the category 'no word' (which we will label `0`) and truncate long reviews.

1.6.1 (TODO) Create a word dictionary

To begin with, we need to construct a way to map words that appear in the reviews to integers. Here we fix the size of our vocabulary (including the 'no word' and 'infrequent' categories) to be `5000` but you may wish to change this to see how it affects the model.

> **TODO:** Complete the implementation for the `build_dict()` method below. Note that even though the vocab_size is set to `5000`, we only want to construct a mapping for the most frequently appearing `4998` words. This is because we want to reserve the special labels `0` for 'no word' and `1` for 'infrequent word'.

[13]:
```
import numpy as np

def build_dict(data, vocab_size = 5000):
    """Construct and return a dictionary mapping each of the most frequently␣
 ↪appearing words to a unique integer."""

    # TODO: Determine how often each word appears in `data`. Note that `data`␣
 ↪is a list of sentences and that a
    #       sentence is a list of words.

    word_count = {} # A dict storing the words that appear in the reviews along␣
 ↪with how often they occur
```

```
    for words in data:
        for word in words:
            if word_count.get(word) == None:
                word_count[word] = 1
            else:
                word_count[word] += 1

    # TODO: Sort the words found in `data` so that sorted_words[0] is the most␣
↪frequently appearing word and
    #       sorted_words[-1] is the least frequently appearing word.

    sorted_words = list(dict(sorted(word_count.items(), key=lambda item:␣
↪item[1], reverse=True)).keys())

    word_dict = {} # This is what we are building, a dictionary that translates␣
↪words into integers
    for idx, word in enumerate(sorted_words[:vocab_size - 2]): # The -2 is so␣
↪that we save room for the 'no word'
        word_dict[word] = idx + 2                              # 'infrequent'␣
↪labels

    return word_dict
```

[14]:
```
word_dict = build_dict(train_X)
```

Question: What are the five most frequently appearing (tokenized) words in the training set? Does it makes sense that these words appear frequently in the training set?

Answer: The most frequently appearing words are shown below. They are reasonable because they are very relevant to film reviews.

[15]:
```
# TODO: Use this space to determine the five most frequently appearing words in␣
↪the training set.
list(word_dict.keys())[:5]
```

[15]:
```
['movi', 'film', 'one', 'like', 'time']
```

1.6.2 Save `word_dict`

Later on when we construct an endpoint which processes a submitted review we will need to make use of the `word_dict` which we have created. As such, we will save it to a file now for future use.

[16]:
```
data_dir = '../data/pytorch' # The folder we will use for storing data
if not os.path.exists(data_dir): # Make sure that the folder exists
    os.makedirs(data_dir)
```

[17]:
```
with open(os.path.join(data_dir, 'word_dict.pkl'), "wb") as f:
    pickle.dump(word_dict, f)
```

1.6.3 Transform the reviews

Now that we have our word dictionary which allows us to transform the words appearing in the reviews into integers, it is time to make use of it and convert our reviews to their integer sequence representation, making sure to pad or truncate to a fixed length, which in our case is 500.

[18]:
```
def convert_and_pad(word_dict, sentence, pad=500):
    NOWORD = 0 # We will use 0 to represent the 'no word' category
    INFREQ = 1 # and we use 1 to represent the infrequent words, i.e., words␣
 ↪not appearing in word_dict

    working_sentence = [NOWORD] * pad

    for word_index, word in enumerate(sentence[:pad]):
        if word in word_dict:
            working_sentence[word_index] = word_dict[word]
        else:
            working_sentence[word_index] = INFREQ

    return working_sentence, min(len(sentence), pad)

def convert_and_pad_data(word_dict, data, pad=500):
    result = []
    lengths = []

    for sentence in data:
        converted, leng = convert_and_pad(word_dict, sentence, pad)
        result.append(converted)
        lengths.append(leng)

    return np.array(result), np.array(lengths)
```

[19]:
```
train_X, train_X_len = convert_and_pad_data(word_dict, train_X)
test_X, test_X_len = convert_and_pad_data(word_dict, test_X)
```

As a quick check to make sure that things are working as intended, check to see what one of the reviews in the training set looks like after having been processeed. Does this look reasonable? What is the length of a review in the training set?

[20]:
```
# Use this cell to examine one of the processed reviews to make sure everything␣
 ↪is working as intended.
train_X_len[100], train_X[100][:100]
```

[20]:
```
(35,
 array([   4,   52,   58,   28,    4,   59, 1808,   70,  351,  392,  190,
          38,   72, 1246,    2,  670,  827, 1382,   46, 1548,   89,   69,
           4, 2354,  916,  274, 1949,  351,  189,    4,   91,   71,  105,
          45,   64,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
           0]))
```

[21]:
```
test_X_len[150], test_X[150][:100]
```

[21]:
```
(99,
 array([   1,     1,   227,     1,  1675,     1,  2441,     1,     1,     1,   496,
           1,     1,     1,  3174,  1249,  4150,     1,   437,   542,     1,     1,
         868,    18,  1073,  1374,   354,  1411,     2,  2885,  3639,  3848,  1011,
        1073,     1,  2501,  2656,     1,  1126,   630,   514,   879,   437,   542,
          33,     1,  3639,     1,   437,  3415,  1106,     1,     1,     1,   224,
           4,   687,   349,     1,   334,  1937,   545,     4,   174,   601,    60,
          56,    47,     2,     1,   598,   166,   427,   866,  1085,    60,  1052,
        2211,   720,   732,  3802,  1073,    18,  1192,  1949,    22,   748,   210,
         289,    83,   210,  1209,  1192,   498,  1073,   495,   961,   556,   462,
           0]))
```

Question: In the cells above we use the `preprocess_data` and `convert_and_pad_data` methods to process both the training and testing set. Why or why not might this be a problem?

Answer: when processing both training set and testing set together, it is likely to accidentally including information from the testing set which would cause inacurate measure of predictive power of the model. However, this would not be a problem here as only training set was used to create `word_dict`.

1.7 Step 3: Upload the data to S3

As in the XGBoost notebook, we will need to upload the training dataset to S3 in order for our training code to access it. For now we will save it locally and we will upload to S3 later on.

1.7.1 Save the processed training dataset locally

It is important to note the format of the data that we are saving as we will need to know it when we write the training code. In our case, each row of the dataset has the form `label`, `length`, `review[500]` where `review[500]` is a sequence of `500` integers representing the words in the review.

[22]:
```
import pandas as pd

pd.concat([pd.DataFrame(train_y), pd.DataFrame(train_X_len), pd.
 ↪DataFrame(train_X)], axis=1) \
        .to_csv(os.path.join(data_dir, 'train.csv'), header=False, index=False)
```

1.7.2 Uploading the training data

Next, we need to upload the training data to the SageMaker default S3 bucket so that we can provide access to it while training our model.

[23]:
```
import sagemaker

sagemaker_session = sagemaker.Session()

bucket = sagemaker_session.default_bucket()
prefix = 'sagemaker/sentiment_rnn'

role = sagemaker.get_execution_role()
```

[24]:
```
input_data = sagemaker_session.upload_data(path=data_dir, bucket=bucket,␣
 ↪key_prefix=prefix)
```

NOTE: The cell above uploads the entire contents of our data directory. This includes the `word_dict.pkl` file. This is fortunate as we will need this later on when we create an endpoint that accepts an arbitrary review. For now, we will just take note of the fact that it resides in the data directory (and so also in the S3 training bucket) and that we will need to make sure it gets saved in the model directory.

1.8 Step 4: Build and Train the PyTorch Model

In the XGBoost notebook we discussed what a model is in the SageMaker framework. In particular, a model comprises three objects

- Model Artifacts,
- Training Code, and
- Inference Code,

each of which interact with one another. In the XGBoost example we used training and inference code that was provided by Amazon. Here we will still be using containers provided by Amazon with the added benefit of being able to include our own custom code.

We will start by implementing our own neural network in PyTorch along with a training script. For the purposes of this project we have provided the necessary model object in the `model.py` file, inside of the `train` folder. You can see the provided implementation by running the cell below.

[25]:
```
!pygmentize train/model.py
```

```
import
torch.nn
as nn

class LSTMClassifier(nn.Module):
    """
    This is the simple RNN model we will be using to perform Sentiment
Analysis.
    """

    def __init__(self,
embedding_dim, hidden_dim, vocab_size):
        """
        Initialize the model by settingg up the various layers.
        """
        super(LSTMClassifier,
self).__init__()

        self.embedding = nn.Embedding(vocab_size, embedding_dim,
padding_idx=0)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim)
        self.dense = nn.Linear(in_features=hidden_dim,
out_features=1)
        self.sig = nn.Sigmoid()

        self.word_dict = None

    def forward(self, x):
        """
        Perform a forward pass of our model on some input.
        """
        x = x.t()
        lengths = x[0,:]
        reviews = x[1:,:]
        embeds = self.embedding(reviews)
        lstm_out, _ = self.lstm(embeds)
        out = self.dense(lstm_out)
        out = out[lengths - 1,
range(len(lengths))]
        return self.sig(out.squeeze())
```

The important takeaway from the implementation provided is that there are three parameters that we may wish to tweak to improve the performance of our model. These are the embedding dimension, the hidden dimension and the size of the vocabulary. We will likely want to make these parameters configurable in the training script so that if we wish to modify them we do not need to modify the script itself. We will see how to do this later on. To start we will write some of the training code in the notebook so that we can more easily diagnose any issues that arise.

First we will load a small portion of the training data set to use as a sample. It would be very time consuming to try and train the model completely in the notebook as we do not have access to a gpu and the compute instance that we are using is not particularly powerful. However, we can work on a small bit of the data to get a feel for how our training script is behaving.

[18]:
```
import torch
import torch.utils.data

# Read in only the first 250 rows
train_sample = pd.read_csv(os.path.join(data_dir, 'train.csv'), header=None,␣
 ↪names=None, nrows=250)

# Turn the input pandas dataframe into tensors
train_sample_y = torch.from_numpy(train_sample[[0]].values).float().squeeze()
```

```
train_sample_X = torch.from_numpy(train_sample.drop([0], axis=1).values).long()

# Build the dataset
train_sample_ds = torch.utils.data.TensorDataset(train_sample_X, train_sample_y)
# Build the dataloader
train_sample_dl = torch.utils.data.DataLoader(train_sample_ds, batch_size=50)
```

1.8.1 (TODO) Writing the training method

Next we need to write the training code itself. This should be very similar to training methods that you have written before to train PyTorch models. We will leave any difficult aspects such as model saving / loading and parameter loading until a little later.

[19]:
```
def train(model, train_loader, epochs, optimizer, loss_fn, device):
    for epoch in range(1, epochs + 1):
        model.train()
        total_loss = 0
        for batch in train_loader:
            batch_X, batch_y = batch

            batch_X = batch_X.to(device)
            batch_y = batch_y.to(device)

            # TODO: Complete this train method to train the model provided.
            optimizer.zero_grad()
            y_pred = model(batch_X)
            loss = loss_fn(y_pred, batch_y)
            loss.backward()
            optimizer.step()

            total_loss += loss.data.item()
        print("Epoch: {}, BCELoss: {}".format(epoch, total_loss /␣
 ↪len(train_loader)))
```

Supposing we have the training method above, we will test that it is working by writing a bit of code in the notebook that executes our training method on the small sample training set that we loaded earlier. The reason for doing this in the notebook is so that we have an opportunity to fix any errors that arise early when they are easier to diagnose.

[20]:
```
import torch.optim as optim
from train.model import LSTMClassifier

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = LSTMClassifier(32, 100, 5000).to(device)
optimizer = optim.Adam(model.parameters())
loss_fn = torch.nn.BCELoss()

train(model, train_sample_dl, 5, optimizer, loss_fn, device)
```

```
Epoch: 1, BCELoss: 0.6934932351112366
Epoch: 2, BCELoss: 0.6854927659034729
Epoch: 3, BCELoss: 0.6790007472038269
Epoch: 4, BCELoss: 0.6721612572669983
Epoch: 5, BCELoss: 0.664274275302887
```

In order to construct a PyTorch model using SageMaker we must provide SageMaker with a training script. We may optionally include a directory which will be copied to the container and from which our training code will be run. When the training container is executed it will check the uploaded directory (if there is one) for a `requirements.txt` file and install any required Python libraries, after which the training script will be run.

1.8.2 (TODO) Training the model

When a PyTorch model is constructed in SageMaker, an entry point must be specified. This is the Python file which will be executed when the model is trained. Inside of the `train` directory is a file called `train.py` which has been provided and which contains most of the necessary code to train our model. The only thing that is missing is the implementation of the `train()` method which you wrote earlier in this notebook.

TODO: Copy the `train()` method written above and paste it into the `train/train.py` file where required.

The way that SageMaker passes hyperparameters to the training script is by way of arguments. These arguments can then be parsed and used in the training script. To see how this is done take a look at the provided `train/train.py` file.

[26]:
```
from sagemaker.pytorch import PyTorch

estimator = PyTorch(entry_point="train.py",
                    source_dir="train",
                    role=role,
                    framework_version='0.4.0',
                    train_instance_count=1,
                    train_instance_type='ml.p2.xlarge',
                    hyperparameters={
                        'epochs': 10,
                        'hidden_dim': 200,
                    })
```

[27]:
```
estimator.fit({'training': input_data})
```

```
'create_image_uri' will be deprecated in favor of 'ImageURIProvider' class in
SageMaker Python SDK v2.
's3_input' class will be renamed to 'TrainingInput' in SageMaker Python SDK v2.
'create_image_uri' will be deprecated in favor of 'ImageURIProvider' class in
SageMaker Python SDK v2.

2021-05-12 01:14:05 Starting - Starting the training job...
2021-05-12 01:14:08 Starting - Launching requested ML instances...
2021-05-12 01:15:20 Starting - Preparing the instances for training...
```

```
2021-05-12 01:16:52 Downloading - Downloading input data...
2021-05-12 01:17:28 Training - Downloading the training image...
2021-05-12 01:17:59 Training - Training image download completed. Training in
progress..bash: cannot set terminal process group (-1): Inappropriate ioctl
for device
bash: no job control in this shell
2021-05-12 01:18:00,692 sagemaker-containers INFO     Imported framework
sagemaker_pytorch_container.training
2021-05-12 01:18:00,719 sagemaker_pytorch_container.training INFO     Block
until all host DNS lookups succeed.
2021-05-12 01:18:00,724 sagemaker_pytorch_container.training INFO
Invoking user training script.
2021-05-12 01:18:01,025 sagemaker-containers INFO     Module train does not
provide a setup.py.
Generating setup.py
2021-05-12 01:18:01,025 sagemaker-containers INFO     Generating
setup.cfg
2021-05-12 01:18:01,025 sagemaker-containers INFO     Generating
MANIFEST.in
2021-05-12 01:18:01,026 sagemaker-containers INFO     Installing module
with the following command:
/usr/bin/python -m pip install -U . -r requirements.txt
Processing /opt/ml/code
Collecting pandas (from -r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/74/24/0cdbf8907e1e3bc5a8da
03345c23cbed7044330bb8f73bb12e711a640a00/pandas-0.24.2-cp35-cp35m-manylinux1_x86
_64.whl (10.0MB)
Collecting numpy (from -r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/b5/36/88723426b4ff576809fe
c7d73594fe17a35c27f8d01f93637637a29ae25b/numpy-1.18.5-cp35-cp35m-manylinux1_x86_
64.whl (19.9MB)
Collecting nltk (from -r requirements.txt (line 3))
  Downloading https://files.pythonhosted.org/packages/5e/37/9532ddd4b1bbb619333d
5708aaad9bf1742f051a664c3c6fa6632a105fd8/nltk-3.6.2-py3-none-any.whl (1.5MB)
Collecting beautifulsoup4 (from -r requirements.txt (line 4))
  Downloading https://files.pythonhosted.org/packages/d1/41/e6495bd7d3781cee623c
e23ea6ac73282a373088fcd0ddc809a047b18eae/beautifulsoup4-4.9.3-py3-none-any.whl
(115kB)
Collecting html5lib (from -r requirements.txt (line 5))
  Downloading https://files.pythonhosted.org/packages/6c/dd/a834df6482147d48e225
a49515aabc28974ad5a4ca3215c18a882565b028/html5lib-1.1-py2.py3-none-any.whl
(112kB)
Collecting pytz>=2011k (from pandas->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/70/94/784178ca5dd892a
98f113cdd923372024dc04b8d40abe77ca76b5fb90ca6/pytz-2021.1-py2.py3-none-any.whl
(510kB)
```

```
Requirement already satisfied, skipping upgrade: python-dateutil>=2.5.0 in
/usr/local/lib/python3.5/dist-packages (from pandas->-r requirements.txt (line
1)) (2.7.5)
Collecting joblib (from nltk->-r requirements.txt (line 3))
  Downloading https://files.pythonhosted.org/packages/28/5c/cf6a2b65a321c4a209ef
cdf64c2689efae2cb62661f8f6f4bb28547cf1bf/joblib-0.14.1-py2.py3-none-any.whl
(294kB)
Requirement already satisfied, skipping upgrade: click in
/usr/local/lib/python3.5/dist-packages (from nltk->-r requirements.txt (line 3))
(7.0)
Collecting tqdm (from nltk->-r requirements.txt (line 3))
  Downloading https://files.pythonhosted.org/packages/72/8a/34efae5cf9924328a8f3
4eeb2fdaae14c011462d9f0e3fcded48e1266d1c/tqdm-4.60.0-py2.py3-none-any.whl
(75kB)
Collecting regex (from nltk->-r requirements.txt (line 3))
  Downloading https://files.pythonhosted.org/packages/38/3f/4c42a98c9ad7d08
c16e7d23b2194a0e4f3b2914662da8bc88986e4e6de1f/regex-2021.4.4.tar.gz (693kB)
Collecting soupsieve>1.2; python_version >= "3.0" (from beautifulsoup4->-r
requirements.txt (line 4))
  Downloading https://files.pythonhosted.org/packages/02/fb/1c65691a9aeb7bd6ac2a
a505b84cb8b49ac29c976411c6ab3659425e045f/soupsieve-2.1-py3-none-any.whl
Collecting webencodings (from html5lib->-r requirements.txt (line 5))
  Downloading https://files.pythonhosted.org/packages/f4/24/2a3e3df732393fed8b3e
bf2ec078f05546de641fe1b667ee316ec1dcf3b7/webencodings-0.5.1-py2.py3-none-
any.whl
Requirement already satisfied, skipping upgrade: six>=1.9 in
/usr/local/lib/python3.5/dist-packages (from html5lib->-r requirements.txt (line
5)) (1.11.0)
Building wheels for collected packages: train, regex
  Running setup.py bdist_wheel for train: started
  Running setup.py bdist_wheel for train: finished with status 'done'
  Stored in directory: /tmp/pip-ephem-wheel-cache-4xwfox4b/wheels/35/24/16/37574
d11bf9bde50616c67372a334f94fa8356bc7164af8ca3
  Running setup.py bdist_wheel for regex: started
  Running setup.py bdist_wheel for regex: finished with status 'done'
  Stored in directory: /root/.cache/pip/wheels/c9/05/a8/b85fa0bd7850b99f9b4f1069
72975f2e3c46412e12f9949b58
Successfully built train regex
Installing collected packages: pytz, numpy, pandas, joblib, tqdm, regex,
nltk, soupsieve, beautifulsoup4, webencodings, html5lib, train
  Found existing installation: numpy 1.15.4
    Uninstalling numpy-1.15.4:
      Successfully uninstalled numpy-1.15.4
Successfully installed beautifulsoup4-4.9.3 html5lib-1.1 joblib-0.14.1
nltk-3.6.2 numpy-1.18.5 pandas-0.24.2 pytz-2021.1 regex-2021.4.4 soupsieve-2.1
tqdm-4.60.0 train-1.0.0 webencodings-0.5.1
You are using pip version 18.1, however version 20.3.4 is available.
You should consider upgrading via the 'pip install --upgrade pip'
command.
2021-05-12 01:18:24,153 sagemaker-containers INFO     Invoking user script

Training Env:
```

```
{
    "input_data_config": {
        "training": {
            "TrainingInputMode": "File",
            "RecordWrapperType": "None",
            "S3DistributionType": "FullyReplicated"
        }
    },
    "resource_config": {
        "current_host": "algo-1",
        "network_interface_name": "eth0",
        "hosts": [
            "algo-1"
        ]
    },
    "framework_module": "sagemaker_pytorch_container.training:main",
    "network_interface_name": "eth0",
    "output_data_dir": "/opt/ml/output/data",
    "hyperparameters": {
        "hidden_dim": 200,
        "epochs": 10
    },
    "output_dir": "/opt/ml/output",
    "module_name": "train",
    "num_gpus": 1,
    "module_dir": "s3://sagemaker-us-east-1-604986012671/sagemaker-
pytorch-2021-05-12-01-14-05-510/source/sourcedir.tar.gz",
    "user_entry_point": "train.py",
    "additional_framework_parameters": {},
    "input_dir": "/opt/ml/input",
    "current_host": "algo-1",
    "job_name": "sagemaker-pytorch-2021-05-12-01-14-05-510",
    "num_cpus": 4,
    "log_level": 20,
    "output_intermediate_dir": "/opt/ml/output/intermediate",
    "hosts": [
        "algo-1"
    ],
    "channel_input_dirs": {
        "training": "/opt/ml/input/data/training"
    },
    "model_dir": "/opt/ml/model",
    "input_config_dir": "/opt/ml/input/config"
}
```

```
Environment variables:
```

```
SM_TRAINING_ENV={"additional_framework_parameters":{},"channel_input_dirs":
{"training":"/opt/ml/input/data/training"},"current_host":"algo-1","framework_mo
dule":"sagemaker_pytorch_container.training:main","hosts":["algo-1"],"hyperparam
eters":{"epochs":10,"hidden_dim":200},"input_config_dir":"/opt/ml/input/config",
"input_data_config":{"training":{"RecordWrapperType":"None","S3DistributionType"
:"FullyReplicated","TrainingInputMode":"File"}},"input_dir":"/opt/ml/input","job
_name":"sagemaker-pytorch-2021-05-12-01-14-05-510","log_level":20,"model_dir":"/
opt/ml/model","module_dir":"s3://sagemaker-us-east-1-604986012671/sagemaker-pyto
rch-2021-05-12-01-14-05-510/source/sourcedir.tar.gz","module_name":"train","netw
ork_interface_name":"eth0","num_cpus":4,"num_gpus":1,"output_data_dir":"/opt/ml/
output/data","output_dir":"/opt/ml/output","output_intermediate_dir":"/opt/ml/ou
tput/intermediate","resource_config":{"current_host":"algo-1","hosts":["algo-1"]
,"network_interface_name":"eth0"},"user_entry_point":"train.py"}
SM_USER_ARGS=["--epochs","10","--hidden_dim","200"]
SM_HOSTS=["algo-1"]
SM_CHANNEL_TRAINING=/opt/ml/input/data/training
SM_INPUT_DIR=/opt/ml/input
SM_RESOURCE_CONFIG={"current_host":"algo-1","hosts":["algo-1"],"network_int
erface_name":"eth0"}
SM_LOG_LEVEL=20
SM_USER_ENTRY_POINT=train.py
PYTHONPATH=/usr/local/bin:/usr/lib/python35.zip:/usr/lib/python3.5:/usr/lib
/python3.5/plat-x86_64-linux-gnu:/usr/lib/python3.5/lib-
dynload:/usr/local/lib/python3.5/dist-packages:/usr/lib/python3/dist-
packages
SM_FRAMEWORK_PARAMS={}
SM_NETWORK_INTERFACE_NAME=eth0
SM_MODEL_DIR=/opt/ml/model
SM_HP_EPOCHS=10
SM_OUTPUT_INTERMEDIATE_DIR=/opt/ml/output/intermediate
SM_FRAMEWORK_MODULE=sagemaker_pytorch_container.training:main
SM_MODULE_NAME=train
SM_HP_HIDDEN_DIM=200
SM_INPUT_CONFIG_DIR=/opt/ml/input/config
SM_HPS={"epochs":10,"hidden_dim":200}
SM_MODULE_DIR=s3://sagemaker-us-east-1-604986012671/sagemaker-
pytorch-2021-05-12-01-14-05-510/source/sourcedir.tar.gz
SM_NUM_GPUS=1
SM_NUM_CPUS=4
SM_INPUT_DATA_CONFIG={"training":{"RecordWrapperType":"None","S3Distributio
nType":"FullyReplicated","TrainingInputMode":"File"}}
SM_CURRENT_HOST=algo-1
SM_CHANNELS=["training"]
SM_OUTPUT_DATA_DIR=/opt/ml/output/data
SM_OUTPUT_DIR=/opt/ml/output
```

```
Invoking script with the following command:

/usr/bin/python -m train --epochs 10 --hidden_dim 200

Using device cuda.
Get train data loader.
Model loaded with embedding_dim 32, hidden_dim 200, vocab_size 5000.
Epoch: 1, BCELoss: 0.6688711010679906
Epoch: 2, BCELoss: 0.6106684134930981
Epoch: 3, BCELoss: 0.5183547485847863
Epoch: 4, BCELoss: 0.4585527680358108
Epoch: 5, BCELoss: 0.4009424204729041
Epoch: 6, BCELoss: 0.3608970976605707
Epoch: 7, BCELoss: 0.35643667894966746
Epoch: 8, BCELoss: 0.33620106930635413
Epoch: 9, BCELoss: 0.3217642167393042
Epoch: 10, BCELoss: 0.3017369685124378
2021-05-12 01:21:25,542 sagemaker-containers INFO     Reporting training
SUCCESS

2021-05-12 01:21:37 Uploading - Uploading generated training model
2021-05-12 01:21:37 Completed - Training job completed
Training seconds: 285
Billable seconds: 285
```

1.9 Step 5: Testing the model

As mentioned at the top of this notebook, we will be testing this model by first deploying it and then sending the testing data to the deployed endpoint. We will do this so that we can make sure that the deployed model is working correctly.

1.10 Step 6: Deploy the model for testing

Now that we have trained our model, we would like to test it to see how it performs. Currently our model takes input of the form `review_length, review[500]` where `review[500]` is a sequence of `500` integers which describe the words present in the review, encoded using `word_dict`. Fortunately for us, SageMaker provides built-in inference code for models with simple inputs such as this.

There is one thing that we need to provide, however, and that is a function which loads the saved model. This function must be called `model_fn()` and takes as its only parameter a path to the directory where the model artifacts are stored. This function must also be present in the python file which we specified as the entry point. In our case the model loading function has been provided and so no changes need to be made.

NOTE: When the built-in inference code is run it must import the `model_fn()` method from the `train.py` file. This is why the training code is wrapped in a main guard (ie, `if __name__ == '__main__':`)

Since we don't need to change anything in the code that was uploaded during training, we can simply deploy the current model as-is.