Introduction

Welcome to the Language modeling Notebook.

In this notebook, we train a neural network to **generate news headlines**. To reduce computational needs, we have reduced it to headlines about technology, and a handful of Tech giants. In this notebook you will:

- Learn to preprocess raw text so it can be fed into an LSTM.
- Make use of the LSTM library of Tensorflow, to train a Language model to generate headlines
- Use your network to generate headlines, and judge which headlines are likely or not

What is a language model?

Language modeling is the task of assigning a probability to sentences in a language. Besides assigning a probability to each sequence of words, the language models also assigns a probability for the likelihood of a given word (or a sequence of words) to follow a sequence of words. — Page 105, **Neural Network Methods in Natural Language Processing (https://www.amazon.com/Language-Processing-Synthesis-Lectures-Technologies/dp/1627052984/)**, 2017.

In terms of neural network, we are training a neural network to produce probabilities (classification) over a fixed vocabulary of words. Concretely, we are training a neural network to produce:

$$P(w_{i+1}|w_1, w_2, w_3, \ldots, w_i), \forall i \in (1, n)$$

Why is language modeling important?

Language modeling is a core problem in NLP.

Language models can either be used as a stand-alone to produce new text that matches the distribution of text the model is trained on, but can also be used at the front-end of a more sophisticated model to produce better results.

Recently for example, the **BERT (https://arxiv.org/abs/1810.04805)** paper show-cased that pretraining a large neural network on a language modeling task can help improve state-of-the-art on many NLP tasks.

How good can the generation of a Language model be?

If you have not seen the latest post by OpenAI, you should read some of the samples they generated from their language model **here (https://blog.openai.com/better-language-models/#sample1)**. Because of computational restrictions, we will not achieve as good text production, but the same algorithm is at the core. They just use more data and compute.

Library imports

Before starting, make sure you have all these libraries.

In [2]:
```
!pip3 install tensorflow
```

```
Requirement already satisfied: tensorflow in /Library/Frameworks/Pytho
n.framework/Versions/3.6/lib/python3.6/site-packages (1.13.1)
Requirement already satisfied: six>=1.10.0 in /Library/Frameworks/Pytho
n.framework/Versions/3.6/lib/python3.6/site-packages (from tensorflow)
(1.12.0)
Requirement already satisfied: absl-py>=0.1.6 in /Library/Frameworks/Py
thon.framework/Versions/3.6/lib/python3.6/site-packages (from tensorflo
w) (0.7.1)
Requirement already satisfied: tensorboard<1.14.0,>=1.13.0 in /Library/
Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages (f
rom tensorflow) (1.13.1)
Requirement already satisfied: keras-applications>=1.0.6 in /Library/Fr
ameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages (fro
m tensorflow) (1.0.7)
Requirement already satisfied: termcolor>=1.1.0 in /Library/Frameworks/
Python.framework/Versions/3.6/lib/python3.6/site-packages (from tensorf
low) (1.1.0)
Requirement already satisfied: grpcio>=1.8.6 in /Library/Frameworks/Pyt
hon.framework/Versions/3.6/lib/python3.6/site-packages (from tensorflo
w) (1.19.0)
Requirement already satisfied: wheel>=0.26 in /Library/Frameworks/Pytho
n.framework/Versions/3.6/lib/python3.6/site-packages (from tensorflow)
(0.33.1)
Requirement already satisfied: tensorflow-estimator<1.14.0rc0,>=1.13.0
in /Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site
-packages (from tensorflow) (1.13.0)
Requirement already satisfied: numpy>=1.13.3 in /Library/Frameworks/Pyt
hon.framework/Versions/3.6/lib/python3.6/site-packages (from tensorflo
w) (1.16.2)
Requirement already satisfied: gast>=0.2.0 in /Library/Frameworks/Pytho
n.framework/Versions/3.6/lib/python3.6/site-packages (from tensorflow)
(0.2.2)
Requirement already satisfied: keras-preprocessing>=1.0.5 in /Library/F
rameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages (fr
om tensorflow) (1.0.9)
Requirement already satisfied: astor>=0.6.0 in /Library/Frameworks/Pyth
on.framework/Versions/3.6/lib/python3.6/site-packages (from tensorflow)
(0.7.1)
Requirement already satisfied: protobuf>=3.6.1 in /Library/Frameworks/P
ython.framework/Versions/3.6/lib/python3.6/site-packages (from tensorfl
ow) (3.7.0)
Requirement already satisfied: markdown>=2.6.8 in /Library/Frameworks/P
ython.framework/Versions/3.6/lib/python3.6/site-packages (from tensorbo
ard<1.14.0,>=1.13.0->tensorflow) (3.0.1)
Requirement already satisfied: werkzeug>=0.11.15 in /Library/Framework
s/Python.framework/Versions/3.6/lib/python3.6/site-packages (from tenso
rboard<1.14.0,>=1.13.0->tensorflow) (0.15.0)
Requirement already satisfied: h5py in /Library/Frameworks/Python.frame
work/Versions/3.6/lib/python3.6/site-packages (from keras-applications>
=1.0.6->tensorflow) (2.9.0)
Requirement already satisfied: mock>=2.0.0 in /Library/Frameworks/Pytho
n.framework/Versions/3.6/lib/python3.6/site-packages (from tensorflow-e
stimator<1.14.0rc0,>=1.13.0->tensorflow) (2.0.0)
Requirement already satisfied: setuptools in /Library/Frameworks/Pytho
n.framework/Versions/3.6/lib/python3.6/site-packages (from protobuf>=3.
6.1->tensorflow) (40.8.0)
Requirement already satisfied: pbr>=0.11 in /Library/Frameworks/Python.
```

```
framework/Versions/3.6/lib/python3.6/site-packages (from mock>=2.0.0->t
ensorflow-estimator<1.14.0rc0,>=1.13.0->tensorflow) (5.1.3)
```

In [41]:
```
from collections import Counter
import tensorflow as tf

import numpy as np
import json
import os
import tokenizer

root_folder = ""
```

Loading the datasets

Make sure the dataset files are all in the `dataset` folder of the assignment.

- If you are using this notebook locally: You should run the `download_data.sh` script.

In [94]:
```
# This cell loads the data for the model
# Run this before working on loading any of the additional data

with open(root_folder+"dataset/headline_generation_dataset_processed.jso
n", "r") as f:
    d_released = json.load(f)

with open(root_folder+"dataset/headline_generation_vocabulary.txt", "r")
as f:
    vocabulary = f.read().split("\n")
w2i = {w: i for i, w in enumerate(vocabulary)} # Word to index
unkI, padI, start_index = w2i['UNK'], w2i['PAD'], w2i['<START>']

vocab_size = len(vocabulary)

input_length = len(d_released[0]['numerized']) # The length of the first
element in the dataset, they are all of the same length
d_train = [d for d in d_released if d['cut'] == 'training']
d_valid = [d for d in d_released if d['cut'] == 'validation']

print("Number of training samples:",len(d_train))
print("Number of validation samples:",len(d_valid))
```

```
10000
Number of training samples: 88568
Number of validation samples: 946
```

Now that we have loaded the data, let's inspect one of the elements. Each sample in our dataset is has a `numerized` vector, that contains the preprocessed headline. This vector is what we will feed in to the neural network. The field `numerized` corresponds to this list of tokens. The already loaded dictionary `vocabulary` maps token lists to the actual string. Use these elements to recover `title` key of entry 1001 in the training dataset.

Here we write the numerized2text function and inspect element 1001 in the training dataset (`entry = d_train[1001]`).

In [209]:
```
def numerized2text(numerized):
    """ Converts an integer sequence in the vocabulary into a string cor
responding to the title.

        Arguments:
            numerized: List[int]  -- The list of vocabulary indices corr
esponding to the string
        Returns:
            title: str -- The string corresponding to the numerized inpu
t, without padding.
    """
    #####
    # Wecover each word from the vocabulary in the list of indices in nu
merized, using the vocabulary variable
    # We use the string.join() function to reconstruct a single string


    words = []
    converted_string = ""
    s=' '
    for num in numerized:
        words.append(vocabulary[num])

    converted_string=s.join(words)
    #####

    return converted_string

entry = d_train[1001]
print("Reversing the numerized: "+numerized2text(entry['numerized']))
print("From the `title` entry: "+ entry['title'])
```

```
Reversing the numerized: microsoft donates cloud computing ' worth $ 1
bn ' PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD
From the `title` entry: Microsoft donates cloud computing 'worth $1 bn'
```

In language modeling, we train a model to produce the next word in the sequence given all previously generated words. This has, in practice, two steps:

```
1. Adding a special <START> token to the start of the sequence for the inpu
t. This "shifts" the input to the right by one. We call this the "source" se
quence
2. Making the network predict the original, unshifted version (we call this
 the "target" sequence)
```

Let's take an example. Say we want to train the network on the sentence: "The cat is great." The input to the network will be " <START> The cat is great." The target will be: "The cat is great".

Therefore the first prediction is to select the word "The" given the <START> token. The second prediction is to produce the word "cat" given the two tokens " <START> The". At each step, the network learns to predict the next word, given all previous ones.

The next step is to write the build_batch function. Given a dataset, we select a random subset of samples, and will build the "inputs" and the "targets" of the batch

Here write the build_batch function.

In [97]:
```
vocabulary
```

Out[97]:
```
['<START>',
 'UNK',
 'PAD',
 'to',
 ',',
 'apple',
 'facebook',
 'google',
 "'",
 'in',
 'on',
 'the',
 'for',
 'twitter',
 'of',
 'amazon',
 ':',
 'and',
 'a',
 'with',
 'microsoft',
 'is',
 'new',
 'as',
 '$',
 'says',
 'over',
 'after',
 'its',
 "apple's",
 'from',
 'iphone',
 'at',
 '?',
 'by',
 'it',
 "google's",
 '-',
 'be',
 'up',
 'data',
 'more',
 'china',
 'will',
 'app',
 'u.s.',
 'how',
 'you',
 "facebook's",
 'that',
 'ceo',
 'are',
 'million',
 '"',
 'users',
 'about',
 'trump',
```

```
'has',
'your',
'deal',
'out',
'billion',
'sales',
'samsung',
"amazon's",
'may',
'an',
'ipad',
'not',
'eu',
'privacy',
'us',
'news',
'now',
'search',
'first',
'tv',
'into',
'service',
'launches',
'could',
'court',
'what',
'qualcomm',
'against',
'mobile',
'shares',
'android',
'tech',
'why',
'tax',
'case',
'can',
'patent',
'watch',
';',
'top',
'but',
'video',
'one',
'this',
'big',
'street',
'ads',
'down',
'buy',
'company',
"microsoft's",
'just',
'pay',
'windows',
'gets',
'takes',
'launch',
```

```
'market',
'&',
'cloud',
'stock',
'music',
'off',
'zuckerberg',
'store',
'said',
'(',
')',
'than',
'phone',
'report',
'apps',
'ipo',
'get',
'antitrust',
'business',
'online',
'have',
'make',
'his',
'ad',
'next',
'plans',
'social',
'jobs',
'no',
'back',
'day',
'account',
'most',
'who',
'like',
'live',
'take',
'help',
'chief',
"it's",
'internet',
'1',
'fake',
'use',
'5',
'web',
'people',
'year',
'10',
'adds',
'uk',
'wall',
'post',
'unveils',
'fight',
's',
'all',
```

```
'india',
'some',
'free',
'tablet',
'plan',
'was',
'profit',
'security',
'africa',
'should',
'media',
'posts',
"here's",
'probe',
'time',
'revenue',
'smartphone',
'growth',
'police',
"twitter's",
'shows',
'world',
'prime',
'software',
'2',
'wants',
'home',
'bid',
'former',
'cook',
'say',
'record',
'maps',
'man',
'accounts',
'their',
'talks',
'ban',
'week',
'property',
'makes',
'.',
'grapples',
'buys',
'mark',
'share',
'lawsuit',
'intellectual',
'earnings',
'user',
'iphones',
'office',
'page',
'%',
'faces',
'battle',
'years',
```

```
'i',
'best',
'wins',
'show',
'startup',
'push',
'hit',
'still',
'own',
'amid',
'results',
'open',
'south',
'ios',
'face',
'car',
'price',
'judge',
'\x80\x99',
'goes',
'or',
'fire',
'steve',
'digital',
'claims',
'group',
'he',
'rules',
'set',
'event',
'sell',
'other',
'seeks',
'two',
'executive',
'view',
'latest',
'8',
'ahead',
'youtube',
'government',
'before',
'streaming',
'go',
'netflix',
'4',
'we',
'work',
'chinese',
'offers',
'7',
'grapple',
'hits',
'review',
'do',
'reports',
'using',
```

```
 'content',
 'site',
 'nokia',
 'tim',
 'calls',
 '3',
 'york',
 'technology',
 'yahoo',
 'feature',
 'ai',
 'war',
 'offer',
 'stocks',
 'announces',
 'support',
 'challenge',
 'attack',
 'russia',
 'access',
 'if',
 'services',
 'under',
 'europe',
 'photos',
 'election',
 'joins',
 'change',
 'boost',
 'devices',
 'project',
 'patents',
 'investors',
 'update',
 'game',
 'brazil',
 'german',
 'sues',
 'stores',
 'russian',
 'global',
 'sale',
 'glass',
 'xbox',
 'opens',
 '!',
 'doodle',
 'biggest',
 'suit',
 'head',
 'french',
 'kindle',
 'phones',
 'raises',
 'high',
 'beats',
 'way',
```

```
'motorola',
'tweets',
'network',
'coming',
'stop',
'firm',
'donald',
'end',
'when',
'working',
'loses',
'employees',
'right',
'+',
'look',
'changes',
'future',
'power',
'campaign',
'ruling',
'build',
'platform',
'team',
"won't",
'reveals',
'sees',
'another',
'searches',
'being',
'made',
'again',
'exec',
'they',
'x',
"trump's",
'used',
'reportedly',
'bill',
'making',
'trial',
'instagram',
'products',
'photo',
'features',
'health',
'win',
'surface',
'rival',
'lead',
'fall',
'board',
'political',
'her',
'trade',
'companies',
'so',
'gives',
```

```
                "don't",
                'threat',
                'reality',
                'expands',
                'obama',
                'want',
                'tool',
                'bezos',
                'smart',
                'delivery',
                'too',
                'self-driving',
                '/',
                'know',
                'warns',
                'developers',
                'alexa',
                'times',
                'move',
                '--',
                'law',
                'product',
                'european',
                'advertising',
                'echo',
                'start',
                'falls',
                'briefing',
                'friends',
                'drop',
                '6',
                'second',
                'woman',
                'house',
                'policy',
                'gains',
                '#',
                'rise',
                'white',
                'test',
                'going',
                'fine',
                'founder',
                'cars',
                'system',
                'appeal',
                'workers',
                'cut',
                'job',
                'keep',
                'better',
                'money',
                'must',
                'cuts',
                'fund',
                'state',
                'hires',
```

```
'would',
'customers',
'dispute',
'fix',
'whatsapp',
'city',
'percent',
'orders',
'lets',
'play',
'boss',
'looks',
'demand',
'president',
'target',
'block',
'whole',
'deals',
'life',
'estimates',
'brings',
'give',
'bug',
'nexus',
'asks',
'secret',
'public',
'need',
'even',
'behind',
'death',
'feed',
'order',
'attacks',
'stake',
'find',
'meet',
'foods',
'competition',
'updates',
'finds',
'apples',
'questions',
'row',
'retail',
'finally',
'through',
'selling',
'partners',
'fans',
'release',
'good',
'despite',
'nasdaq',
'drops',
'games',
'quarter',
```

```
'cash',
'only',
'speech',
'amazon.com',
'let',
'co-founder',
'see',
'virtual',
'cambridge',
'rivals',
'does',
'sells',
'here',
'major',
'book',
'turns',
'tells',
'accused',
'strong',
'legal',
'black',
'france',
'british',
'getting',
'scandal',
'today',
'prices',
'center',
'hate',
'sets',
'seen',
'releases',
'much',
'chip',
'chrome',
'study',
'germany',
'away',
'100',
'them',
'join',
'design',
'women',
'supplier',
'assistant',
'things',
'tests',
'key',
'holiday',
'messenger',
'taking',
'pro',
'program',
'investment',
'real',
'soon',
'jeff',
```

```
'oracle',
'california',
'problem',
'debut',
'tools',
'analytica',
'htc',
'building',
'starts',
"can't",
'japan',
'schmidt',
'alphabet',
'mac',
'names',
'got',
",'",
'add',
'others',
'without',
'moves',
'20',
'three',
'last',
'tops',
'rises',
'headquarters',
'hacked',
'fbi',
'needs',
'siri',
"what's",
'voice',
'did',
'value',
'leave',
'puts',
'shopping',
'sold',
'during',
't',
'abuse',
'unit',
'ftc',
'itunes',
'targets',
'via',
'really',
"china's",
'call',
'concerns',
'congress',
'profits',
'beat',
'buying',
'rolls',
'computer',
```

```
 'bring',
 '500',
 "doesn't",
 'issues',
 'settlement',
 'might',
 'denies',
 'close',
 'five',
 'early',
 'star',
 'local',
 'intel',
 'sued',
 'huge',
 'introduces',
 'my',
 'him',
 'partner',
 '9',
 'become',
 'publishers',
 'bad',
 'acquires',
 'daily',
 'create',
 'bans',
 'expected',
 'aims',
 'button',
 'rights',
 'e-book',
 'signs',
 'action',
 'vs.',
 "isn't",
 'month',
 'testing',
 'staff',
 'list',
 'since',
 'private',
 'korea',
 'series',
 '2016',
 'regulators',
 'focus',
 'giant',
 'following',
 'turn',
 'pages',
 'north',
 'breach',
 'files',
 'hackers',
 'website',
 'helps',
```

```
 'unveil',
 'put',
 'sharing',
 'u.k.',
 'indian',
 'claim',
 'debate',
 'messages',
 'worth',
 'pixel',
 "world's",
 'lawmakers',
 'been',
 'inside',
 'washington',
 'name',
 'employee',
 'london',
 'leads',
 'steps',
 'drive',
 'where',
 'browser',
 'eyes',
 'acquisition',
 'surge',
 'had',
 'tablets',
 'loss',
 'videos',
 'walmart',
 'version',
 'industry',
 'comes',
 'uses',
 'san',
 'confirms',
 'race',
 'trending',
 'partnership',
 'watchdog',
 'maker',
 '2012',
 'while',
 'businesses',
 'uber',
 'picks',
 'dorsey',
 'yet',
 'investigation',
 '50',
 'macbook',
 'long',
 'requests',
 'run',
 'defends',
 'ever',
```

```
'small',
'taxes',
'official',
'p',
'expand',
'analysts',
'removes',
'blocks',
'comments',
'part',
'regulator',
'role',
'books',
'days',
'gmail',
'paid',
'spotify',
'tweet',
'silicon',
'reach',
'jack',
'east',
'hack',
'sony',
'suppliers',
'sandberg',
'months',
'ready',
'rejects',
'lost',
'decision',
'complaint',
'dead',
'ibm',
'strategy',
'drone',
'mini',
'control',
'hq2',
'pressure',
'analyst',
'found',
'suspends',
'love',
'bigger',
'goldman',
'2018',
'oil',
'energy',
'chat',
'leaves',
'friday',
'pc',
'consumer',
'minister',
'vote',
'research',
```

```
'developer',
'galaxy',
'conference',
'our',
'line',
'ireland',
'lower',
'investor',
'ends',
'seek',
'murder',
'less',
'advertisers',
'display',
'hardware',
'bank',
'wireless',
'valley',
'irish',
'device',
'available',
'settle',
'vr',
'followers',
'talk',
'markets',
'charges',
'becomes',
'smartphones',
'check',
'jump',
'firms',
'email',
'space',
'engine',
'linkedin',
'£',
'meeting',
'challenges',
'fears',
'begins',
'higher',
'censorship',
'sites',
'ebay',
'wrong',
'cost',
'boosts',
'family',
'appeals',
'risk',
'details',
'faster',
'groups',
'banks',
'admits',
'american',
```

```
'personal',
'pays',
'...',
'tell',
'many',
'australia',
'spending',
'forecast',
'broadcom',
'rally',
'april',
'popular',
'information',
'bing',
'mln',
'skype',
'kids',
'ipads',
'crisis',
'urges',
'screen',
'30',
'campus',
'hold',
'among',
'images',
'15',
'everything',
'come',
'intelligence',
'protest',
'every',
'air',
'12',
'expect',
'arrested',
'hiring',
'costs',
'ask',
'brexit',
'these',
'debuts',
'threatens',
'brand',
'called',
'complaints',
'backs',
'around',
'explains',
'likely',
'me',
'engineer',
'track',
'teams',
'blocked',
'invest',
'birthday',
```

```
 'supreme',
 'ways',
 'recognition',
 'america',
 "users'",
 'planning',
 'cyber',
 'hopes',
 'stream',
 'camera',
 'party',
 'gain',
 'child',
 'were',
 'os',
 'history',
 'chips',
 'tries',
 'links',
 'morgan',
 'millions',
 'jury',
 'pulls',
 'think',
 'remove',
 'because',
 'outlook',
 'sign',
 '2017',
 'message',
 'pushes',
 'killing',
 'between',
 '11',
 'agency',
 'problems',
 'banned',
 'messaging',
 'little',
 'charge',
 'cities',
 'save',
 'porn',
 'approval',
 'christmas',
 'ballmer',
 'marketing',
 'takeover',
 'sex',
 'battery',
 'looking',
 'nigeria',
 'snapchat',
 'fines',
 'leader',
 'july',
 'which',
```

```
 'diversity',
 'bets',
 'missing',
 'payments',
 'wal-mart',
 'contract',
 'backlash',
 'u.s',
 'great',
 'encryption',
 'e-books',
 'keeps',
 'she',
 'posting',
 'jumps',
 '2015',
 '25',
 'children',
 'quarterly',
 'tracking',
 "didn't",
 'speaker',
 'third',
 'school',
 'turkey',
 'half',
 'growing',
 'step',
 'mobility',
 'return',
 'blackberry',
 ...]
```

In [7]:
```
def build_batch(dataset, batch_size):
    """ Builds a batch of source and target elements from the dataset.

        Arguments:
            dataset: List[db_element] -- A list of dataset elements
            batch_size: int -- The size of the batch that should be crea
ted
        Returns:
            batch_input: List[List[int]] -- List of source sequences
            batch_target: List[List[int]] -- List of target sequences
            batch_target_mask: List[List[int]] -- List of target batch m
asks
    """



    # We get a list of indices we will choose from the dataset.
    # The randint function uses a uniform distribution, giving equal pro
bably to any entry
    # for each batch
    indices = list(np.random.randint(0, len(dataset), size=batch_size))

    # Recover what the entries for the batch are
    batch = [dataset[i] for i in indices]

    # Get the raw numerized for this input, each element of the dataset
 has a 'numerized' key
    batch_numerized = [dataset[i]['numerized'] for i in indices]

    # Create an array of start_index that will be concatenated at positi
on 1 for the input.
    # Should be of shape (batch_size, 1)
    start_tokens = np.zeros((batch_size,1))

    # Concatenate the start_tokens with the rest of the input
    # The np.concatenate function should be useful
    # The output should now be [batch_size, sequence_length+1]
    batch_input = np.concatenate((start_tokens,batch_numerized),axis=1)

    # Remove the last word from each element in the batch
    # To restore the [batch_size, sequence_length] size
    batch_input = batch_input[:,:-1]
    #print(batch_size)
    #print(batch_input.shape)

    # The target should be the un-shifted numerized input
    batch_target = batch_numerized

    # The target-mask is a 0 or 1 filter to note which tokens are
    # padding or not, to give the loss, so the model doesn't get rewarde
d for
    # predicting PAD tokens.
    batch_target_mask = np.array([a['mask'] for a in batch])



    return batch_input, batch_target, batch_target_mask
```

Creating the language model

Now that we've written the data pipelining, we are ready to write the Neural network.

The steps to setting up a neural network to do Language modeling are:

- Creating the placeholders for the model, where we can feed in our inputs and targets.
- Creating an RNN of our choice, size, and with optional parameters
- Using the RNN on our placeholder inputs.
- Getting the output from the RNN, and projecting it into a vocabulary sized dimension, so that we can make word predictions.
- Setting up the loss on the outputs so that the network learns to produce the correct words.
- Finally, choosing an optimizer, and defining a training operation: using the optimizer to minimize the loss.

In [286]:
```
# Using a basic RNN/LSTM for Language modeling
class LanguageModel():
    def __init__(self, input_length, vocab_size, rnn_size, learning_rate
=1e-4):

        # Create the placeholders for the inputs:
        # All three placeholders should be of size [None, input_length]
        # Where None represents a variable batch_size, and input_length
 is the
        # maximal length of a sequence of words, after being padded.
        self.input_num = tf.placeholder(tf.int32, shape=[None, input_len
gth])
        self.targets = tf.placeholder(tf.int32, shape=[None, input_lengt
h])
        self.targets_mask = tf.placeholder(tf.int32, shape=[None, input_
length])

        # Create an embedding variable of shape [vocab_size, rnn_size]
        # That will map each word in our vocab into a vector of rnn_size
size.
        embedding =tf.Variable(tf.random.normal((vocab_size,rnn_size)))
        # Use the tensorflow embedding_lookup function
        # To embed the input_num, using the embedding variable we've cre
ated
        input_emb = tf.nn.embedding_lookup(embedding,self.input_num)

        # Create a an RNN or LSTM cell of rnn_size size.
        # Look into the tf.nn.rnn_cell documentation
        # You can optionally use Tensorflow Add-ons such as the MultiRNN
Cell, or the DropoutWrapper
        lm_cell = tf.nn.rnn_cell.LSTMCell(rnn_size)
        lm_cell = tf.nn.rnn_cell.DropoutWrapper(lm_cell, output_keep_pro
b=0.5)
        # Use the dynamic_rnn function of Tensorflow to run the embedded
inputs
        # using the lm_cell you've created, and obtain the outputs of th
e RNN cell.
        # You have created a cell, which represents a single block (colu
mn) of the RNN.
        # dynamic_rnn will "copy" the cell for each element in your sequ
ence, runs the input you provide through the cell,
        # and returns the outputs and the states of the cell.
        outputs, states = tf.nn.dynamic_rnn(lm_cell, input_emb,dtype=tf.
float32)

        # Use a dense layer to project the outputs of the RNN cell into
 the size of the
        # vocabulary (vocab_size).
        # output_logits should be of shape [None,input_length,vocab_siz
e]
        self.output_logits = tf.layers.dense(outputs,vocab_size)

        # Setup the loss: using the sparse_softmax_cross_entropy.
        # The logits are the output_logits we've computed.
        # The targets are the gold labels we are trying to match
        # Don't forget to use the targets_mask we have, so your loss is
```

```
 not off,
        # And your model doesn't get rewarded for predicting PAD tokens
        # You might have to cast the masks into float32. Look at the tf.
cast function.



        self.loss = tf.losses.sparse_softmax_cross_entropy(self.targets,
self.output_logits,weights=self.targets_mask)

        # Setup an optimizer (SGD, RMSProp, Adam)

        optimizer = tf.train.AdamOptimizer(learning_rate=5.0e-5)

        # We create a train_op that requires the optimizer we've created
to minimize the
        # loss we've defined.

        self.global_step = tf.train.get_or_create_global_step()
        self.train_op = optimizer.minimize(self.loss,global_step=self.gl
obal_step)
        self.saver = tf.train.Saver()
```

In [ ]:

Once you have created the Model class, we should instantiate the model. The line tf.reset_default_graph() resets the graph for the Jupyter notebook, so multiple models aren't floating around. If you have trouble with redefinition of variables, it may be worth re-running the cell below.

In [287]:
```
# We can create our model,
# with parameters of our choosing.

tf.reset_default_graph() # This is so that when you debug, you reset the
graph each time you run this, in essence, cleaning the board
model = LanguageModel(input_length=input_length, vocab_size=vocab_size,
rnn_size=256, learning_rate=5e-4)
```

Training the model

In [334]:
```
old=root_folder+"models/better_language_model"
new_experiment = root_folder+"models/final_language_model"

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    # Here is how you initialize weights of the model according to their
    # Initialization parameters.
    model.saver.restore(sess, old)
    for i in range(0):


        # Here is how you obtain a batch:
        batch_size = 256
        batch_input, batch_target, batch_target_mask = build_batch(d_tra
in, batch_size)
        # Map the values to each tensor in a `feed_dict`
        feed = {model.input_num: batch_input, model.targets: batch_targe
t, model.targets_mask: batch_target_mask}

        # Obtain a single value of the loss for that batch.
        step, train_loss, _ = sess.run([model.global_step, model.loss, m
odel.train_op], feed_dict=feed)
        if i%10==0:
            print(step)
            print(train_loss)
    # Here is how you save the model weights
    model.saver.save(sess, new_experiment)

#     # Here is how you restore the weights previously saved
#     model.saver.restore(sess, experiment)
```

```
INFO:tensorflow:Restoring parameters from models/better_language_model
```

Using the language model

Congratulations, you have now trained a language model! We can now use it to evaluate likely news headlines, as well as generate our very own headlines.

(1) Evaluation loss

To evaluate the language model, we evaluate its loss (ability to predict) on unseen data that is reserved for evaluation. Your first evaluation is to load the model you trained, and obtain a test loss.

In [335]:
```
model_file = root_folder+"models/final_language_model"
```

In [339]:
```
with tf.Session() as sess:
    model.saver.restore(sess, model_file)
    eval_input, eval_target, eval_target_mask = build_batch(d_valid, 500
)
    #print(eval_input,eval_target)
    feed = {model.input_num: eval_input, model.targets: eval_target, mod
el.targets_mask: eval_target_mask}
    eval_loss = sess.run([model.loss], feed_dict=feed)
    print("Evaluation set loss:", eval_loss)
```

```
INFO:tensorflow:Restoring parameters from models/final_language_model
Evaluation set loss: [5.5446362]
```

(2) Evaluation of likelihood of data

One use of a language model is to see what data is more likely to have originated from the training data. Because we have trained our model on news headlines, we can see which of these headlines is more likely:

```
Apple to release another iPhone in September
```

```
Apple and Samsung resolve all lawsuits amicably
```

The first one is obviously more likely since iphones are released yearly but apple and samsung will never amicably resolve lawsuits due to money and competition struggles

In [340]:
```
headline1 = "Apple to release new iPhone in July"
headline2 = "Apple and Samsung resolve all lawsuits"
import numpy as np
headlines = [headline1, headline2]
from nltk.tokenize import TweetTokenizer
tknzr = TweetTokenizer()

with tf.Session() as sess:
    model.saver.restore(sess, model_file)

    for headline in headlines:
        headline = headline.lower() # Our LSTM is trained on lower-cased
headlines

        # From the code in the Preprocessing section at the end of the n
otebook
        # Find out how to tokenize the headline
        tokenized =tknzr.tokenize(headline)

        # Find out how to numerize the tokenized headline
        numerized = numerize_sequence(tokenized)

        # Learn how to pad and obtain the mask of the sequence.
        padded, mask = pad_sequence(numerized, len(numerized), 20)
        #print(tokenized,numerized,padded,mask)

        # Obtain the loss of the sequence, and pring it
        feed = {model.input_num: np.array([0]+padded[:-1]).reshape(1,20
), model.targets: np.array(padded).reshape(1,20), model.targets_mask: np
.array(mask).reshape(1,20)}
        loss,outputlogits = sess.run([model.loss,model.output_logits], f
eed_dict=feed)
#         print(outputlogits)
#         for i in outputlogits[0]:
#             print(np.argmax(i))
        print("----------------------------------------")
        print("Headline:",headline)
        print("Loss of the headline:", loss)
```

```
INFO:tensorflow:Restoring parameters from models/final_language_model
----------------------------------------
Headline: apple to release new iphone in july
Loss of the headline: 3.5208185
----------------------------------------
Headline: apple and samsung resolve all lawsuits
Loss of the headline: 5.736441
```

(3) Generation of headlines

We can use our language model to generate text according to the distribution of our training data. The way generation works is the following:

We seed the model with a beginning of sequence, and obtain the distribution for the next word. We select the most likely word (argmax) and add it to our sequence of words. Now our sequence is one word longer, and we can feed it in again as an input, for the network to produce the next sentence. We do this a fixed number of times (up to 20 words), and obtain automatically generated headlines!

In [341]:
```
with tf.Session() as sess:
    model.saver.restore(sess, model_file)

    # Here are some headline starters.
    # They're all about tech companies, because
    # That is what is in our dataset
    headline_starters = ["apple has released", "google has released", "a
mazon", "tesla to", "google and apple", "tesla sues amazon"]

    for headline_starter in headline_starters:
        print("====================")
        print("Generating headline starting with: "+headline_starter)

        # Tokenize and numerize the headline. Put the numerized headline
        # beginning in `current_build`
        tokenized = tokenized =tknzr.tokenize(headline_starter)
        current_build = [startI] + numerize_sequence(tokenized)

        while len(current_build) < input_length:
            # Pad the current_build into a input_length vector.
            # We do this so that it can be processed by our LanguageMode
l class
            current_padded = current_build[:input_length] + [padI] * (in
put_length - len(current_build))
            padded_out=current_padded[1:]+[2]
            padded_out=np.array([padded_out])
            current_padded = np.array([current_padded])
            #print(current_padded)
            # Obtain the logits for the current padded sequence
            # This involves obtaining the output_logits from our model,
            # and not the loss like we have done so far

            feed = {model.input_num: current_padded.reshape(1,input_leng
th), model.targets: padded_out.reshape(1,input_length), model.targets_ma
sk: np.array(mask).reshape(1,input_length)}
            logits = sess.run([model.output_logits], feed_dict=feed)
            #print(logits)
            # Obtain the row of logits that interest us, the logits for
 the last non-pad
            # inputs
            #print(logits)
            logits = logits[-1][0]
            #print(logits)
            #print(logits)

            last_logits=logits[len(current_build)-1]
            #print(last_logits)

            #print(np.argmax(last_logits))
            # Find the highest scoring word in the last_logits
            # array. The np.argmax function should be useful.
            # Append this word to our current build

            current_build.append(np.argmax(last_logits))

        # Go from the current_build of word_indices
```

```
        # To the headline (string) produced. This should involve
        # the vocabulary, and a string merger.
        produced_sentence = numerized2text(current_build)
        print(produced_sentence)
```

```
INFO:tensorflow:Restoring parameters from models/final_language_model
====================
Generating headline starting with: apple has released
<START> apple has released a UNK to the iphone 8 , but it will be a UNK
UNK for the
====================
Generating headline starting with: google has released
<START> google has released a new version of the UNK of the new york ci
ty , UNK and the cloud
====================
Generating headline starting with: amazon
<START> amazon is launching a new version of the cloud computing ? her
e's what it means to the UNK of
====================
Generating headline starting with: tesla to
<START> tesla to buy apple , UNK UNK , UNK UNK , UNK , UNK , UNK , est.
UNK
====================
Generating headline starting with: google and apple
<START> google and apple are UNK up to UNK the UNK UNK . UNK UNK UNK UN
K UNK UNK UNK
====================
Generating headline starting with: tesla sues amazon
<START> tesla sues amazon to block UNK of UNK iphone sales ban on iphon
e sales , UNK says apple is
```

All done

You are done with the first part of the repo

Next notebook deals with Summarization of text!

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: