CAPSTONE REPORT

Becoming Jane by Deep Learning

Yoko Harada

November 11, 2016
Udacity Machine Learning

INTRODUCTION

Jane? Yes, Jane. Her name is Jane Austen, a famous British novelist in 18th century. The title, “Becoming Jane” is borrowed from her biographical movie released in 2007 (http://www.imdb.com/title/tt0416508/). The next question would be why “becoming.” The experiment here is whether a deep learning can produce or mimic sentences as if Jane wrote. In a catchy phrase, it would be “AI can become Jane?”

There are many areas the deep learning is good at. So called natural language processing (NLP) is among them. In NLP, the data are not independent points. For example, the word “becoming” has some relations to previous or later words, like “becoming Jane.” Because of this nature, the AI should learn from a sequence of data (words). In such a case, the model called Recurrent Neural Networks or RNN is typically used. This model is designed to learn the data based on what have learned so far. As for the simple example, “becoming Jane,” to learn the word, “Jane,” the model uses what have learned by the word, “becoming.” In the next phase of learning, the word comes next to “Jane” will be learned based on the knowledge acquired by “Jane”.

In the field, Shakespeare novels are often used as a source of NLP to generate sentences (“The Unreasonable Effectiveness of Recurrent Neural Networks”, ref. 3). The novels are well known and commonly used in English lectures as well. When I studied English at Community College, Shakespeare and Jane Austen were the textbooks. The reason I chose Jane Austen rather than the Shakespeare novels is Shakespeare’s English is too cryptic for a non-alphabet language speaker. It’s hard to discern whether generated sentences are natural or not. Compared to that, Jane Austen’s English is quite close to the English today. Whether the generated sentences are natural or not is easy to figure out.

Another aspect of the article mentioned above, “The Unreasonable Effectiveness of Recurrent Neural Networks” (ref. 3) is it takes the approach of character based learning. Another approach is a word based. This experiment adopts the word based learning. The character based needs to learn a word itself, while word based starts learning a sentence. If a testing environment is rich, for example, GPU is available, the character based would be nice since it will have ability to generate words also.

However, this experiment was done on a single Laptop. It is unfortunately the best environment available. That said the experiment is a kind of how much we can do on a poor environment.

ALGORITHMS

Before going deeper of the experimentation, let's talk about algorithms to predict sentences. The main algorithm is RNN, which is consists of multiple cells. Each cell has a model called LSTM. Lastly, this section mentions word2vec, which is an algorithm to convert natural language to vector representation.

- Recurrent Neural Networks (RNNs)

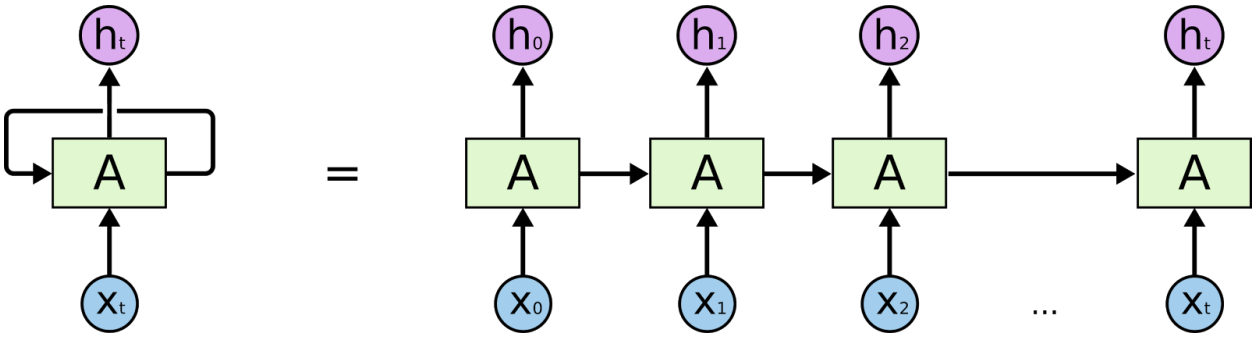
As mentioned earlier, the Recurrent Neural Networks (RNNs) is the algorithm used to train and predict Jane Austen's sentences. Like other neural network models, basically, RNN calculates:

$$y = Wx + b$$

where, x: input, W: weight, b: bias, y: output. Something RNN is different from others is it reuses the output. The training process goes like this:

$$y_0 = Wx_0 + b$$
$$y_1 = W(y_0, x_1) + b$$
$$y_2 = W(y_1, x_2) + b$$

The picture below is describes the RNN learning process. The leftmost network shows all recurrent processes in a single one, so it has a loop. This single network expression is equivalent to the right side. The second, third, ... networks take a new input data and the output from previous training, which is a recurrent connection. This is why the model is called Recurrent Neural Networks. The programming model is the left one, while, what's going on is described on the right side.

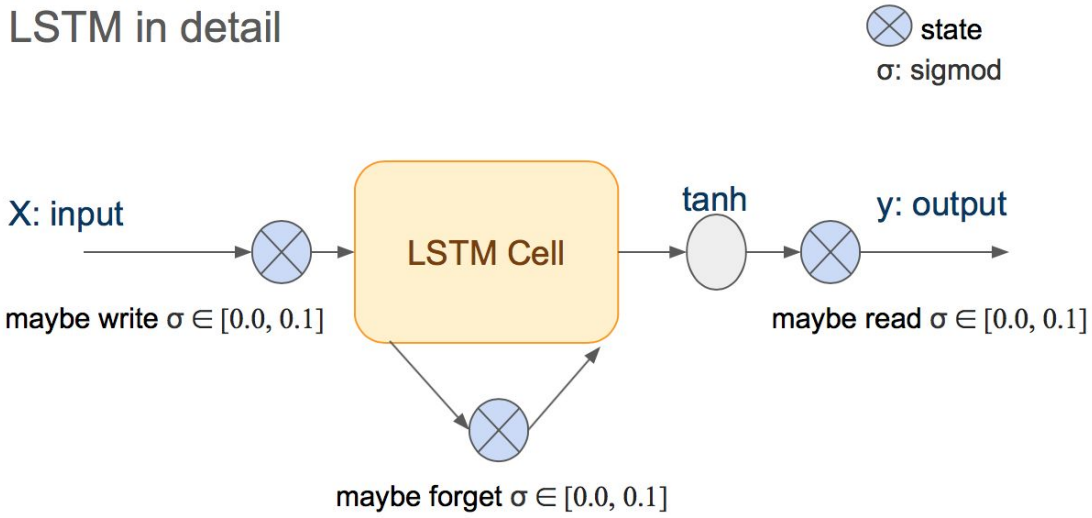


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- Problem of RNN Model

For the natural language processing, the sequence is the matter. As explained earlier, “becoming Jane” makes sense because the word, “Jane,” comes after “becoming.” Often, a sentence makes sense in more than three words. It may be five words or even longer for people to figure out what it means. In this point of view, memorizing words learned so far works well to learn the next word effectively. However, how many should be memorized is to be considered. “Pride and Prejudice” has about 121500 words, “Sense and Sensibility” has 118700. To learn a word in the middle of the novel, all previous 60000 words don’t need. Additionally, too many previous words would be noisy and lower the prediction accuracy. To solve this problem, the idea of “forget it” has been introduced to RNN.

- Long Short Term Memory Networks (LSTMs)



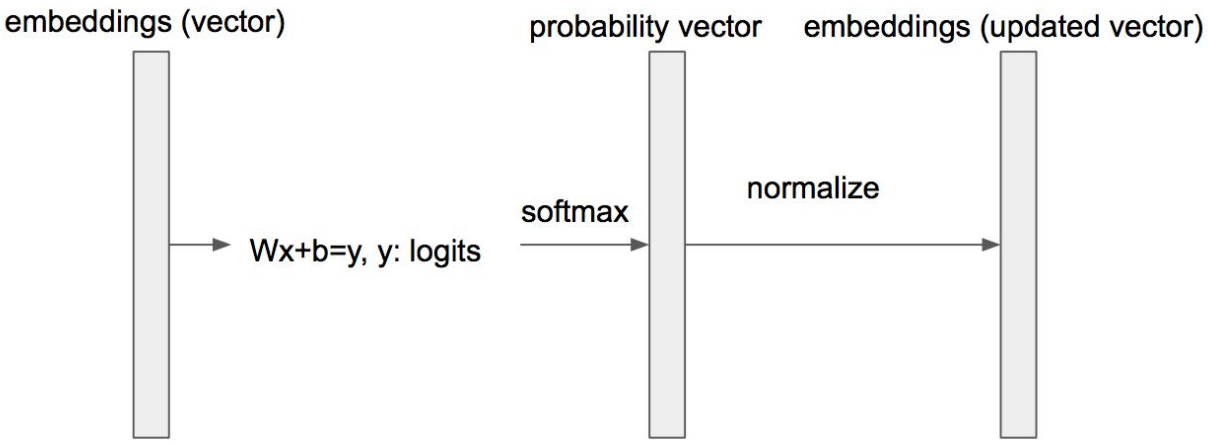
The picture above describes the details of LSTM (based on the lecture, Udacity Deep Learning). Parameters are controlled by a sigmoid function at the three points; input, forget, and output. The tanh (hyperbolic tangent) before output is there to keep output values in the range [-1.0, 1.0]. When maybe write/forget/read is 1.0, parameters will be passed through as those are. When it is 0.0, nothing will be passed. The sigmoid function is expected to produce the value between them. This architecture solves the problem cause by memorizing all words equally.

- Word2vec

In NLP, the process called word2vec is important to measure how much those two words are close. The closeness or distance is calculated by cosine distance. As mentioned in the Expected Results section, all words can be plotted on the two dimensional space based on similarity. The word2vec is

the process to learn similarity. The word similarity is expressed by a vector which is called embeddings. During RNN training, inputs are always looked up from this embeddings.

Roughly, the word2vec works as in the picture below:



The leftmost is an embeddings vector, which is an output from LSTM cell. At the next step, logits will be calculated. Following step creates probability vector using a softmax function. In the end, the normalized probability vector will be the embeddings to feed to RNN for the next training iteration.

Softmax function: $S(y_i) = e^{y_i} / \sum_j e^{y_i}$

LEARNING PROCESS

This section explains how the learning process goes using Jane Austen novels.

1. Convert text to vector of digits
   Since the input data is a text, in another word, a bunch of words, those should be converted to numbers so that RNN can handle. The conversion goes like this:
   I. input text → vector of words
   ```
   "How good it was in you, my dear Mr. Bennet! ...
   [u'"How', u'good', u'it', u'was', u'in', u'you,', u'my', u'dear', u'Mr.', ...
   ```
   II. vector of words → word count map
   ```
   {u'you': 7, u'I': 5, u'it': 4, u'said': 3, u'for': 3, u'was': 3, u'as': 3, ...
   ```
   III. word count map → sortedmap by count
   ```
   [(u'you', 7), (u'I', 5), (u'it', 4), (u'as', 3), (u'for', 3), (u'said', 3), ...
   ```
   IV. sortedmap → word list sorted by frequency, and map of (word, index in sorted word list)
   ```
   (u'you', u'I', u'it', u'as', u'for', u'said', u'the', u'was', u'Mr.', u'a', u'an' ...
   {u'am!': 39, u'matter.': 74, u'pleased': 88, u'till': 99, u'father': 55, u'joke, ...
   ```
   V. vector of words → vector of index in sorted word list
   ```
   [26, 13, 2, 7, 63, 24, 79, 49, 8, 31, 33, 1, 66, 1, 20, 86,...
   ```

The last vector of numbers is the input to create embeddings

2. Trains the model
   As explained in the algorithm section, the embeddings are used to train the RNN. The input data will be divided in batches and each batch is processed one by one. The output of the previous batch process will be fed to the next training.

   When all batches are processed, one iteration finishes. This one iteration is called “epoch.” The training iterates multiple epochs to minimize the cost. The final output will be a probability list of all words in the word list. This machine learning is a regression model. The final output is not yes/no values, but probability.

   During the training, the model tries to minimize a cost. In this model, the cost is an “average negative log probability of the target words,” and calculate by the formula below:

$$loss = -\frac{1}{N}\sum_{i=1}^{N} ln(p_{target_i})$$
$$cost = \frac{loss}{batch\ size}$$

   However, the document, “TensorFlow Tutorial, Recurrent Neural Networks” (ref. 2) mentions the perplexity is used to measure a performance for this kind of learning. A perplexity (average per-word perplexity) sees the same parameter as the cost and is calculate by:

$$perplexity = e^{loss}$$

   Following the TensorFlow Tutorial, the perplexity is used to compare the result.

3. Tests the model
   In this experiment, every time one epoch finishes, a validation runs. When all epochs finish, a test runs. This means the training and validation perplexities will be calculated as many as the number of epochs. While only one test perplexity will be calculated at the end of training.

   The training and validation uses the same RNN setting, but validation doesn’t optimize the result and repeat neither. While the test model processes all data at once without dividing into batches. Also, the test model doesn’t optimize nor repeat.

   While testing, this experiment has changed multiple RNN model parameter settings to find the best combination.

4. Generates (Predicts) a sequence of words
   Once the training finishes, the model will have the trained embeddings and word probability. Based on those trained data, the model can predict a next word to next word. When a prediction continues, say, 200 times, the sentence(s) of 200 words will be produced. This will be a qualitative measurement and no score will be calculated. Whether the outcome is natural or not will be judged.

DATA

Since “Becoming Jane” is the goal, Jane Austen’s three novels are used to train the model;

- “Pride and Prejudice”(http://www.fullbooks.com/Pride-and-Prejudice.html)
- “Sense and Sensibility” (http://www.fullbooks.com/Sense-and-Sensibility-by-Jane-Austen.html)
- “Emma”(http://www.fullbooks.com/Emma-by-Jane-Austen.html)

Each online books are divided in 8-9 parts. The last part of three books are used as validation data. The validation ran after every epoch finished. The rest of 1-7(or 8) parts are used as training data. To test the model, the first part of “Persuasion”(http://www.fullbooks.com/Persuasion-by-Jane-Austen.html) will be used as a test data. After all epochs run, the test data will be evaluated.

Data sizes are:

- training: 2.0 MB (Pride and Prejudice/Sense and Sensibility/Emma)
- validation: 92 KB (Pride and Prejudice/Sense and Sensibility/Emma)
- test: 112 KB (Persuasion)

When the data is loaded, sentences are split by a space to create a list of words. During this process, all line feed (\n) characters are replaced by a space. Since the original input data is a book, every line is formatted to have almost the same line length as in below.

```
Elizabeth Bennet had been obliged, by the scarcity of gentlemen,
to sit down for two dances; and during part of that time,
Mr. Darcy had been standing near enough for her to hear a
conversation between him and Mr. Bingley, who came from the
dance for a few minutes, to press his friend to join it.
```

In this input data, the line feeds don’t contribute to predict the sequence of words. For this reason, those were eliminated.

As explained in the Learning Process section, all words in the whole three novels will be converted in the vector of indices in the word list sorted by frequencies. This indices vector will be used as the training data.

The number of vocabulary acquired from the training data is: 28141. This size is almost the same as the Penn Tree Bank (PTB) data set (http://www.cis.upenn.edu/~treebank/) , 28831, which is popular dataset for this kind of training. Additionally, Jane Austen training data size is almost the same as PTB training dataset of 2.1M. Given that, input data by the three novels of Jane Austen is considered standard.

IMPLEMENTATION

Not many machine learning libraries have RNN model. Theano (http://deeplearning.net/software/theano/) and TensorFlow (https://www.tensorflow.org/) would be the two major libraries that support RNN. Since Udacity Deep Learning courses uses TensorFlow, it is used to implement RNN here.

Tensorflow is an Open Source high level machine learning library. It is designed to run in the big system such as GRU effectively. Probably, because of such computing environment support, TensorFlow has a specific variable caching mechanism. Unfortunately, the variable scope design with caching makes programming complicated on Python notebook. Because of this reason, the RNN implementation code was mostly borrowed from TensorFlow RNN tutorial (ref. 2). As for word2vec, implementation was done followed by TensorFlow word2vec tutorial (ref. 6).

PARAMETER TUNING

TensorFlow RNN model has a few parameters to tune up the model. Among them, parameters below were picked for comparison.

Parameter	Detail	Tested Values
number of layers	the number of RNNs to stack up	2, 3
forget bias	the bias of forget gate in LSTM	0.0, 0.5, 1.0
output keep probability	neuron dropout rate for output	0.5, 0.75, 1.0
hidden size	the number of neurons in hidden layer	200, 300, 400

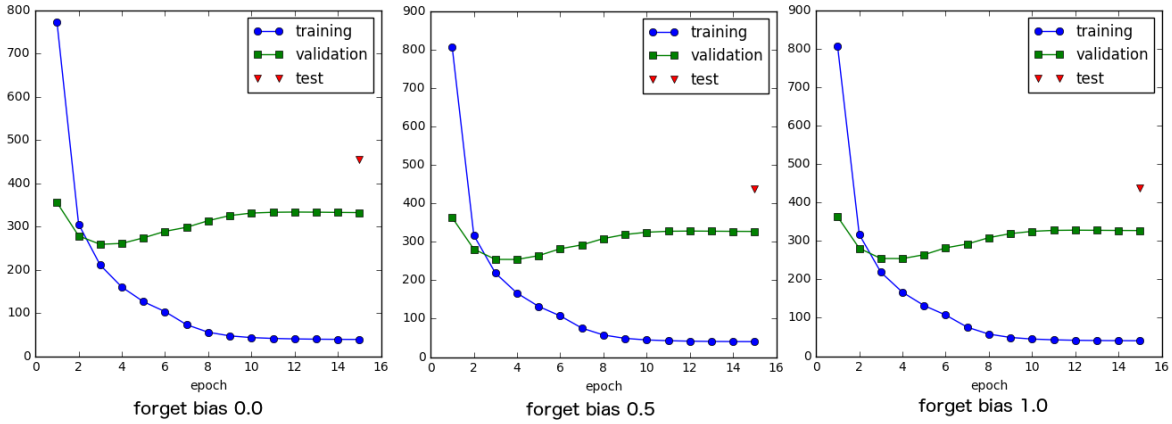
1. forget bias parameter effect on basic model

   The first comparison is done on the forget bias against the basic model. The graphs show the perplexity curves. The table shows perplexities of the training, validation and test when the training finished.

   Parameters: layers: 2, keep prob: 1.0, hidden size: 200

forget bias	0.0	0.5	1.0
Training	38.736	40.061	40.469
Validation	332.277	323.367	323.556

Test	455.205	434.388	425.012



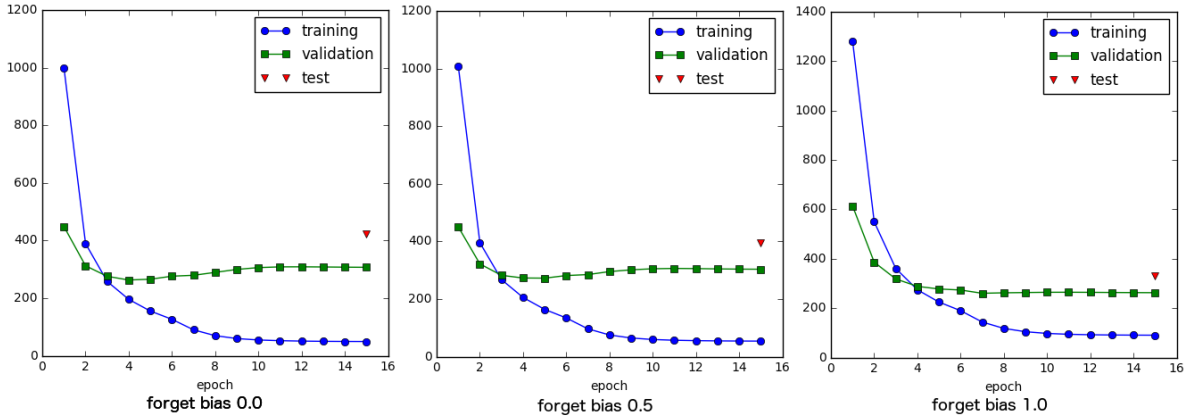
The results show good numbers in training and big gaps in validation and test. This seems an overfitting.

2. layers

   The next parameters is the number of layers. Exactly the same parameters as previous trainings except number of layers were used. Now, the number of layers changed to 3.

   Parameters: layers: 3, keep prob: 1.0, hidden size: 200

forget bias	0.0	0.5	1.0
Training	49.300	53.787	90.301
Validation	306.905	303.334	264.541
Test	421.259	395.809	331.866



Compared the result of 2 layers, apparently, all values improved. However, except forget bias 1.0, still, overfitting tendency exists.

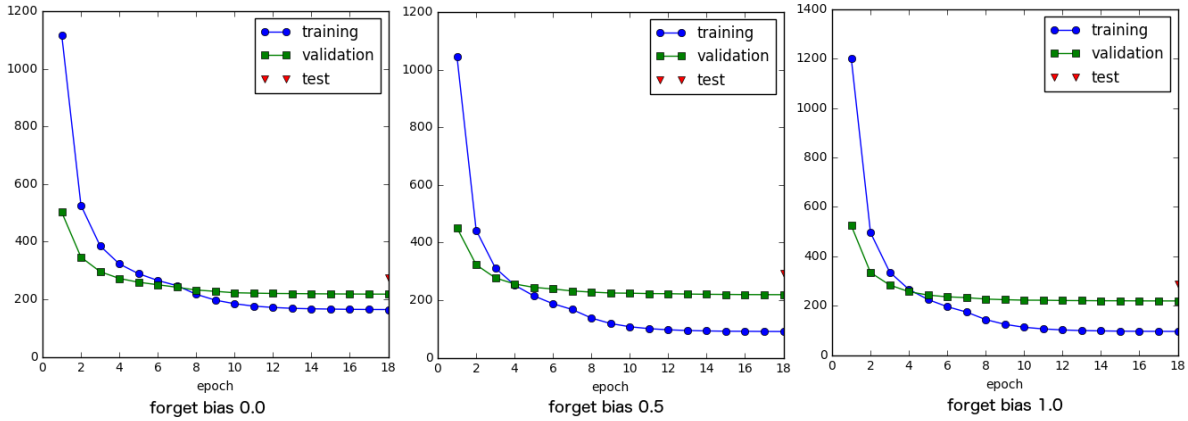
3. output keep probability

   The third parameter was the output keep probability. The previous training used the

probability 1.0 for all. Here, the probabilities 0.75 and 0.5 were tested.

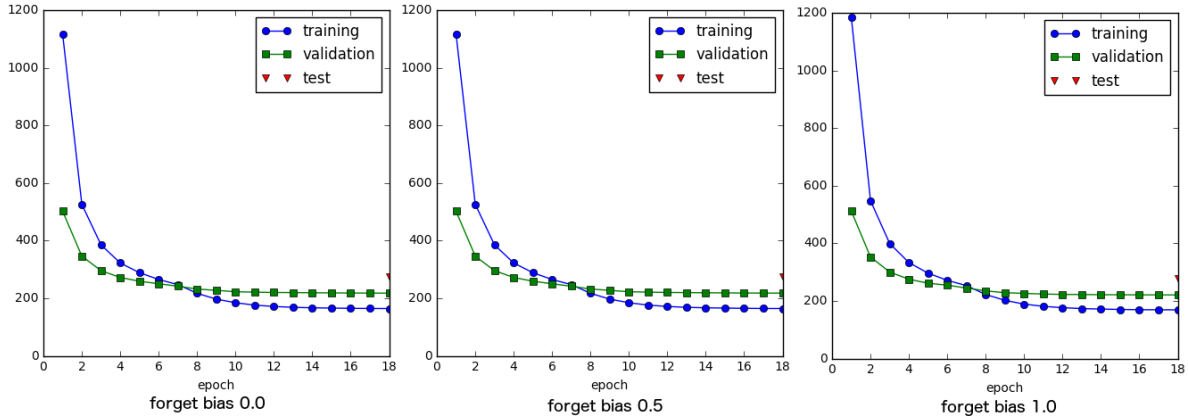
Parameters: layers: 3, keep prob: 0.75, hidden size: 200

forget bias	0.0	0.5	1.0
Training	87.771	90.897	95.613
Validation	222.007	218.876	219.128
Test	293.039	293.200	286.973



Parameters: layers: 3, keep prob: 0.5, hidden size: 200

forget bias	0.0	0.5	1.0
Training	162.184	164.100	168.933
Validation	218.180	217.645	220.649
Test	278.253	274.471	276.033



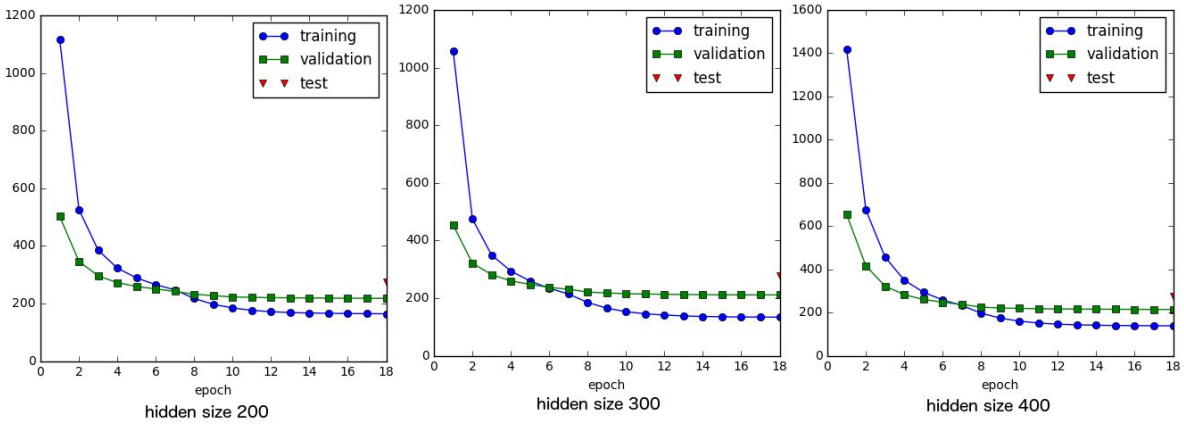
As the above graphs and tables show, when output keep probability decreases, validation and test get much improved. On the contrary, the training accuracy is getting worse. At least, the overfitting problem becomes less. Additionally, the test perplexity 274.471 is the best score among others.

4. hidden size

The last parameter tested was the number of neurons in the hidden layer. For this comparison, forget bias 0.5 and output keep probability 0.5 were used since these showed the best test perplexity so far.

Parameters: layers: 3, forget bias 0.5, keep prob: 0.5

hidden size	200	300	400
Training	164.100	133.553	138.274
Validation	217.645	210.924	214.055
Test	274.471	275.924	274.426



From the training results, the number of neurons in hidden layers doesn't have much impact. When hidden size is 400, the test perplexity was the best in the slight difference. The downside of increasing neurons is: as the number of neurons increases, training time gets longer and longer. Therefore, hidden size of 200 would work well.

The result may change if computing environment is really rich which makes much bigger epochs run in a reasonable time..

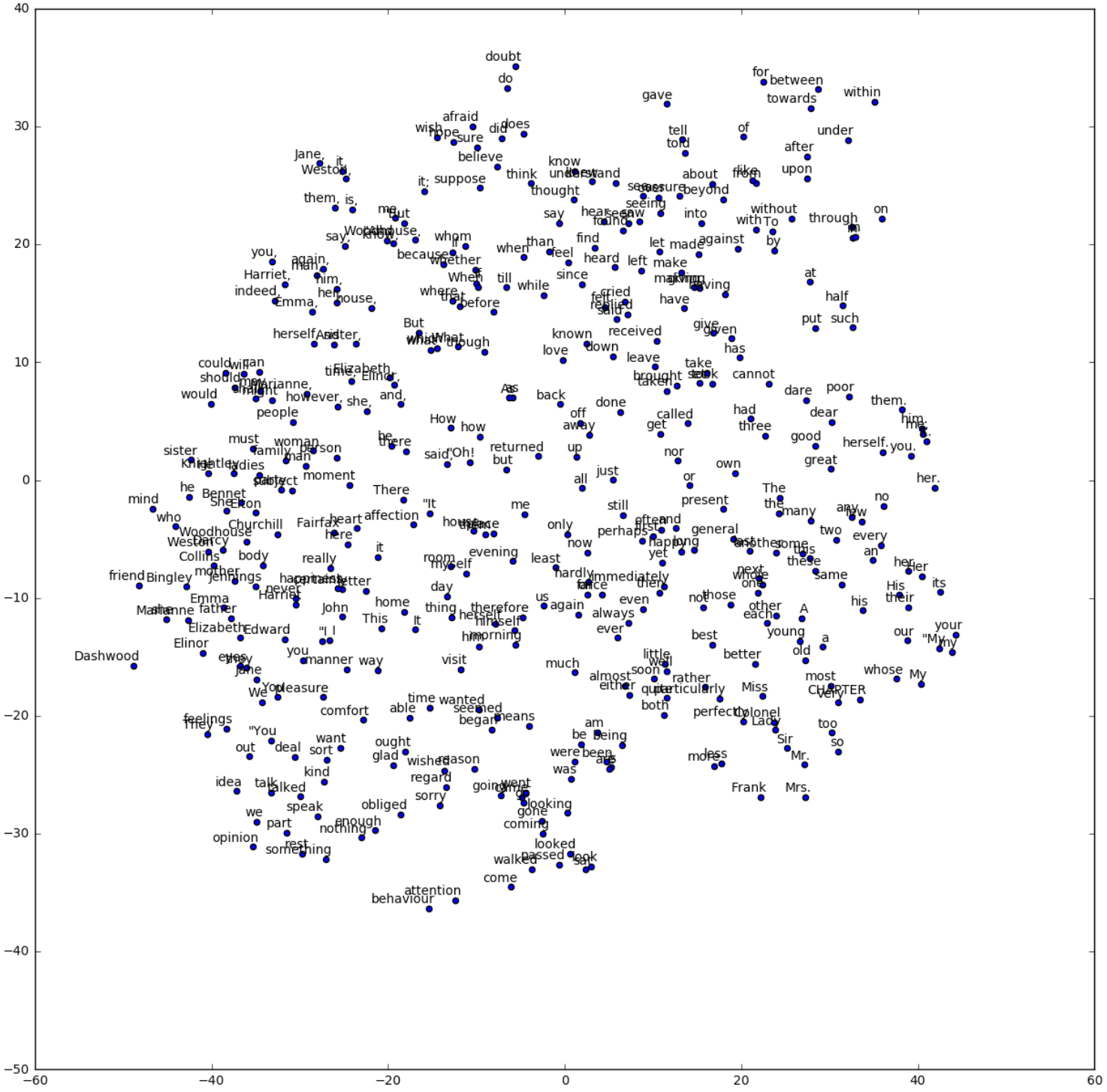
RESULTS

1. Parameters
   From the comparisons in the previous section, these would be the best parameter choice:
   - number of layers: 3
   - forget bias: 0.5
   - output keep probability: 0.5
   - number of neurons: 400

   The number of neuron can be 200. The reason of 400 was chosen is the training perplexity was better than 200.

2. Word2Vec
   This is a two-dimensional visualization of vocabulary, known as word2vec. Similar words should reside in a close area on two-dimensional map after the training. For example, modal verbs such as might, would or can are considered as similar and should be put together in a

particular coordinate on the map. The word2vec representation depicts an internal state when the training finishes.



3. Sequence of words

The sequence of words were generated by a sequential predictions starting from a seed word. In this experiment, the trained model's variables were saved in a file and could be restored for later predictions.
For comparison, two sequences of words are shown below. The first one is the output from the basic model of only one epoch. The second one is from the best parameters combination of 18 epochs.

- basic model of only one epoch

  ```
  Elizabeth looked; married.--Well, a cottage, so; which do walked for by their
  handsomest was joining whose strangers brilliancy she dare wish from the favour
  they and her expected the youths Hertfordshire." however, on every way and the
  concerned, complaint till particulars,--and and were towards the great happy."
  which hope, attaching, and by the disclaim till the second sent. were not
  Charlotte the half.--Five prudently and Mrs. John and haughty. of them, which
  while the borne." much There was income the different shocking! the few
  concealed the contrary, was the room. from the hour walked thousand the
  Imprudence the few son trust; out of the embarrassment or "Ten perhaps, the hope
  are at the 'For well! the fellow every holding "but and I talked was so
  exactly." for the marriage to listen, the wish blast; But I was did How the
  until himself "HER she was distracted." "I must favours I should never add I
  should liked him him heard I should pride--for through the anyhow from Building,
  if and the choose," arrival; expressions to him. the Dashwood was believed but
  "believe like the park, but she affecting said its mother. the little best
  life--" But Marianne is not give the possible
  ```

- best parameters combination of 18 epochs

  ```
  Elizabeth laugh to which any other, that stopt this think?" "Oh, "Yes--no--never
  young man of sure and I really ever could disapprove for me." CHAPTER paused. I
  can't only be happy to extraordinary on all the name of your pleasures, secret
  or very unreasonable; everybody am as going off, to confession at both of Miss
  Woodhouse, it I am afraid suppose; and his modesty, to-morrow of us first--she
  has the table) which gave a mother of it: her would be reasonably afraid of
  explain:--there "Oh!" I think, not altogether to be out of your side, by She as
  this, but you know." Emma saw his comparison without herself. "I lives, could
  another, cannot think on my sister's poor earl's creature!" said to inquiry, and
  I knew unlike his continuance in me: not not to have heard himself talking, I
  would not even happen if your heart to me as assuring you to convince you of Mr.
  Elton. I understood him at last address.-- "Yes, she has been half off Carter
  and expressing his parade to the ladies--married confinement of a income." What
  deplore the unpretending, I refused. Had used all think to see it clear to
  doubt. The meeting immediately. no,
  ```

Obviously, the second one is more natural.

CONCLUSION

By training, RNN could generate somehow readable sentences. However, it is still far from Jane Austen's novels. As mentioned at the beginning, in the field, many trials of this sort have been reported, for example, "Andrej Karpathy blog: The Unreasonable Effectiveness of Recurrent Neural

Networks" (http://karpathy.github.io/2015/05/21/rnn-effectiveness/) . The author attempted to generate Shakespeare, Wikipedia, Linux source code and others. The results are amazingly excellent. The biggest difference is the input data size. According to the blog post, those are Shakespeare: 4.4 MB, Wikipedia: 96MB, and Linux source code: 474 MB. Compared to that, the data size of two Jane Austen novels is only 2.0 MB. Probably, more data would make the sequence of words more natural.

However, a computational resource is limited for a personal project, only one laptop. In the blog post, it mentioned, "several as-large-as-fits-on-my-GPU 3-layer LSTMs over a period of a few days." For deep learning, a good computational environment might be necessary to get excellent results from a vast amount of data.

If the training was done using all Jane Austen's novels, the deep learning would have produced sentences as if she wrote.

REFERENCES

1. Udacity, Deep Learning, https://www.udacity.com/course/deep-learning--ud730
2. TensorFlow Tutorial, Recurrent Neural Networks, https://www.tensorflow.org/versions/r0.11/tutorials/recurrent/index.html#recurrent-neural-networks
3. The Unreasonable Effectiveness of Recurrent Neural Networks, http://karpathy.github.io/2015/05/21/rnn-effectiveness/
4. colah's blog: Understanding LSTM Networks, http://colah.github.io/posts/2015-08-Understanding-LSTMs/
5. word-rnn-tensorflow, https://github.com/hunkim/word-rnn-tensorflow
6. TensorFlow Tutorial, Vector Representations of Words, https://www.tensorflow.org/versions/r0.11/tutorials/word2vec/index.html#vector-representations-of-words
7. YouTube, Sirajology, https://www.youtube.com/channel/UCWN3xxRkmTPmbKwht9FuE5A