CAPSTONE REPORT

Becoming Jane by Deep Learning

Yoko Harada

November 23, 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 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. One more 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. This 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)

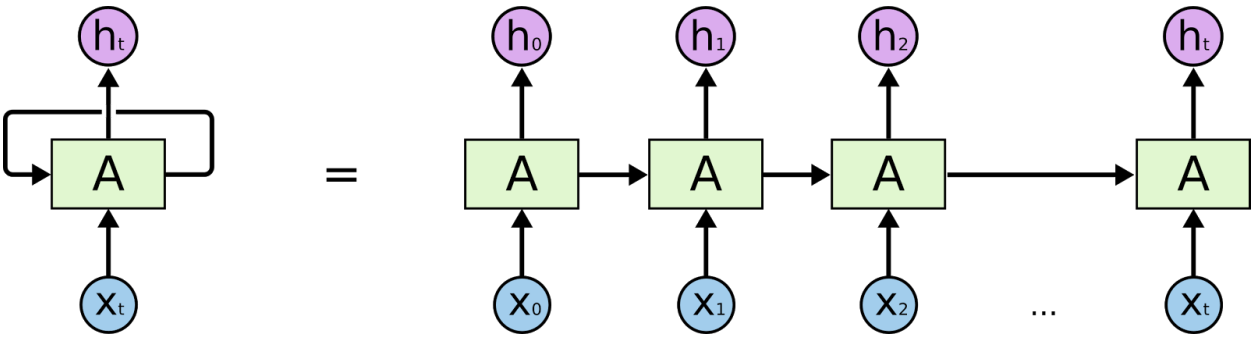
As mentioned earlier, the Recurrent Neural Networks (RNNs) is the algorithm used to train and predict Jane Austen's sentences. Like other neural network models, basically, RNN calculates:

$$y = Wx + b$$

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.



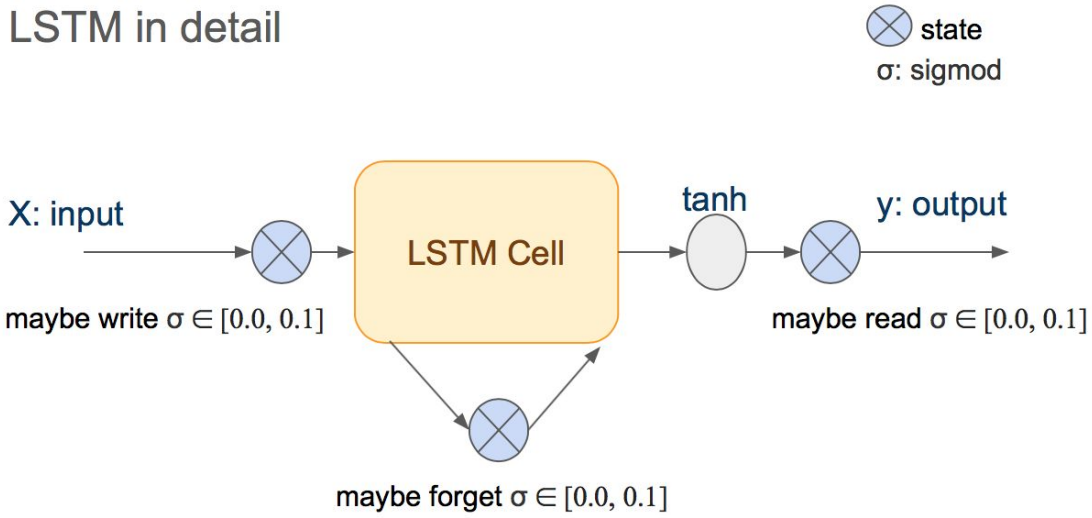
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- 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)

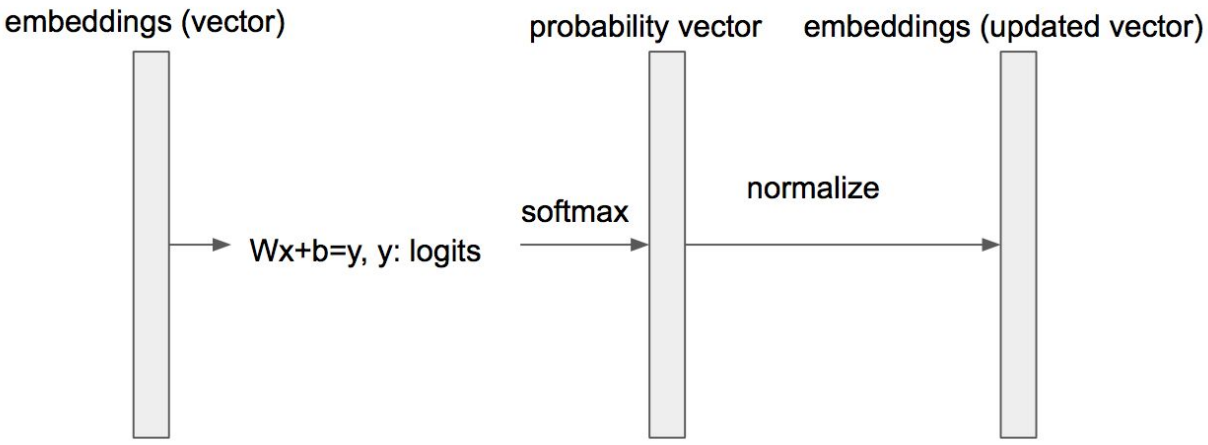


The picture above describes the details of LSTM (based on the lecture, Udacity Deep Learning). Parameters are controlled by a sigmoid function at the three points; input, forget, and output. The tanh (hyperbolic tangent) before output is there to keep output values in the range [-1.0, 1.0]. When maybe 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, in another word, sequential predictions. Starting from a seed 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

Input Source

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 preprocessing

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.

- All double quote characters are deleted.
  This is to treat "I and I as the same word. This preprocessing may cost. The generated sentences may not look like Jane Austen’s writings since she used conversation style a lot. However, in terms of learning appropriate sequence of words, the presence of double quote is considered not relevant.

- All parenthesis are deleted.
  Jane Austen liked to use parenthesis, for example, “`told her one day (and there was a blush as she said it,) that it was.`” However, for the same reason as the double quote, to learn and predict sequence of words, parenthesis are not relevant.

- Except proper nouns, all words are converted to lowercase.
  Without this, word list will have words like: The, the, To, to, And, and. Capitalized or not doesn’t contribute to learn relations between words.

Statistics

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)

After all preprocesses were applied, the number of vocabulary of the training data is: 24770. Without preprocessing, the vocabulary is 28141. This number is close to 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, the three novels of Jane Austen as the input data is considered standard.

As explained in the Learning Process section, all words in the whole three novels will be converted to the vector of indices in the word list sorted by frequencies. This indices vector will be fed to train the model.

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 was the choice to implement RNN here.

Tensorflow is an Open Source machine learning library focused on handling huge dataset. 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 for two reasons:

1. Sharing embeddings among train, validation and test models

   This experiment creates three models to train and test. Those three shares embeddings because it is the matrix to train parameters. The embeddings saved as TensorFlow's Variable, which persists in Python memory and looked up in the TensorFlow manner. TensorFlow has option to reuse model; however, setting is tricky. Also, shape change is not allowed.

2. In session computing

   TensorFlow has an idea of session, which is an environment to run the model. The variables are tied to the session. Even though instances/variables are there from a programmatic view, those are not available outside of the session. There may be a way; however, it's hard to

reuse the session to predict sequence of words after the training. For this reason, this project has two Python notebooks.

Because of these reasons, 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

The training will starts from the very basic model. Parameters used for the basic model are in the table below.

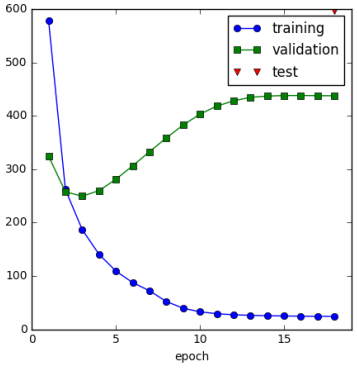
Parameter	Detail	Basic Model
number of layers	the number of RNNs to stack up	1
forget bias	the bias of forget gate in LSTM	0.0
keep probability	neuron dropout rate, 1/keep_prob	1.0
hidden size	the number of neurons in hidden layer	200

Starting from this basic parameter combinations, train, validation and test perplexities will be compared, especially test perplexity. As already explained in Learning Process section, the perplexity measures the performance. Lower the value is, better the model is.

Additionally, the generated sequence of words by the best model will be compared to the basic model. Having “Elizabeth” as the seed word, 200 words will be predicted. This is to check the goal, whether “mimicking Jane Austen” could be accomplished or not.

Result of Basic Model

After the 18 epochs of training, each perplexity of the basic model was: train: 24.162, validation: 437.469, test: 596.622. The graph on the right is perplexity curves during 18 epochs. The graph shows good training result and much worse in validation and test. This is a typical overfitting problem. Trying various parameters, the experiment will find the best model.

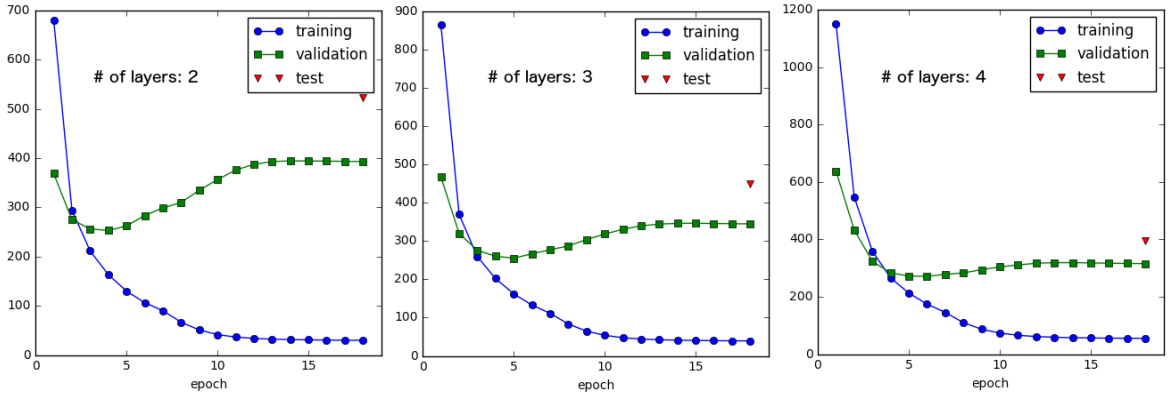


Number of Layers

The first parameter to compare the performance is the number of RNN layers. Other parameters kept the same as the basic model.

Parameters: forget bias: 1.0, keep prob: 1.0, hidden size: 200

# of layers	2	3	4
Training	29.778	39.110	54.844
Validation	393.248	344.903	315.422
Test	522.474	448.959	393.977



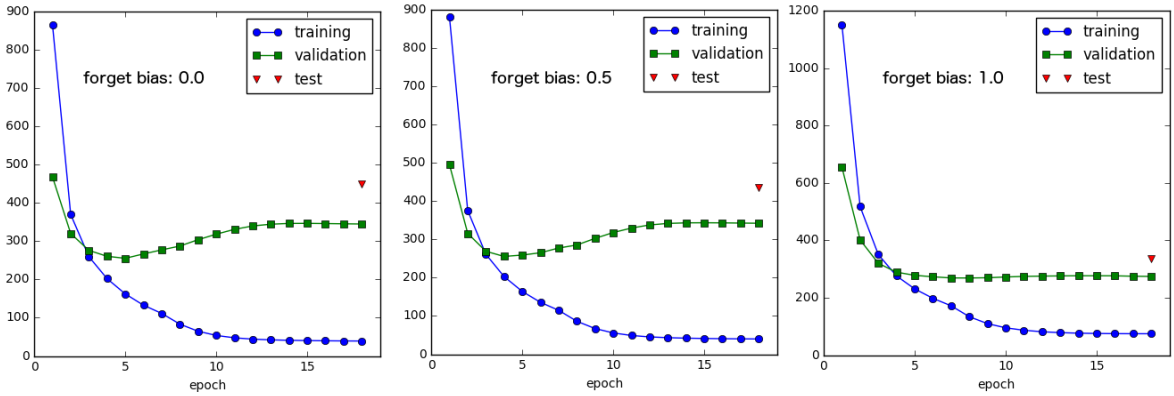
Compared the result of 1 layer (basic model), apparently, validation and test perplexities improved. The overfitting problem got better. Among tested, 4 layers showed the best result. However, on the single laptop, testing on 4 layers is painfully slow. To see other parameters' effect , 3 layers will be used. In the end, 4 layer will be tested with the best parameter combination.

Forget Bias

The second comparison is done on the forget bias keeping the same parameters as the basic model except the number of layers. Now, The number of layers turned to 3.

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

forget bias	0.0	0.5	1.0
Training	39.110	39.873	74.477
Validation	344.903	342.233	274.127
Test	448.959	435.157	337.055



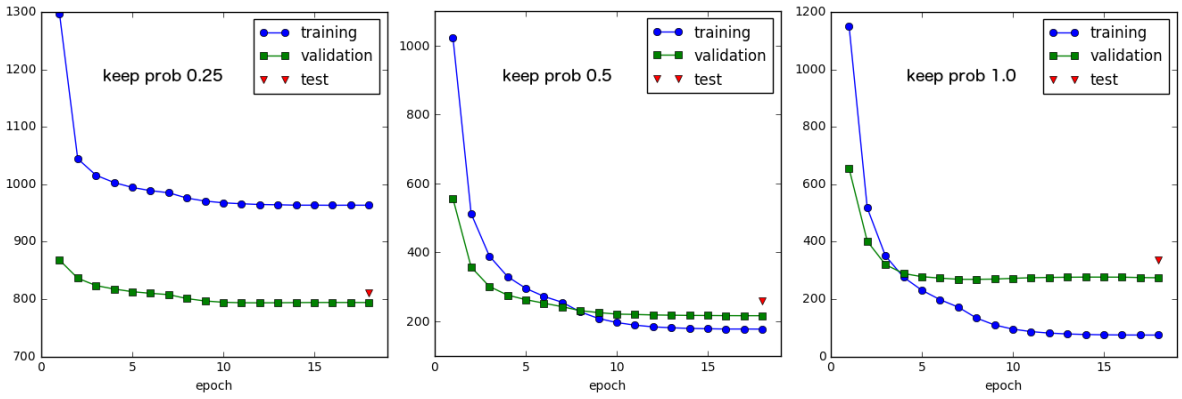
The result of forget bias 1.0 showed the best validation and test perplexities. Ironically, LSTM cell's ability to forget far back inputs didn't help. From this result, following training will use forget bias 1.0.

Keep Probability

The third parameter tested was the keep probability. Looking at previous results, the training was done on 3 layers and forget bias 1.0.

Parameters: layers: 3, forget bias 1.0, hidden size: 200

keep prob	0.25	0.5	1.0
Training	963.408	176.828	74.477
Validation	793.941	215.652	274.127
Test	810.668	259.893	337.055



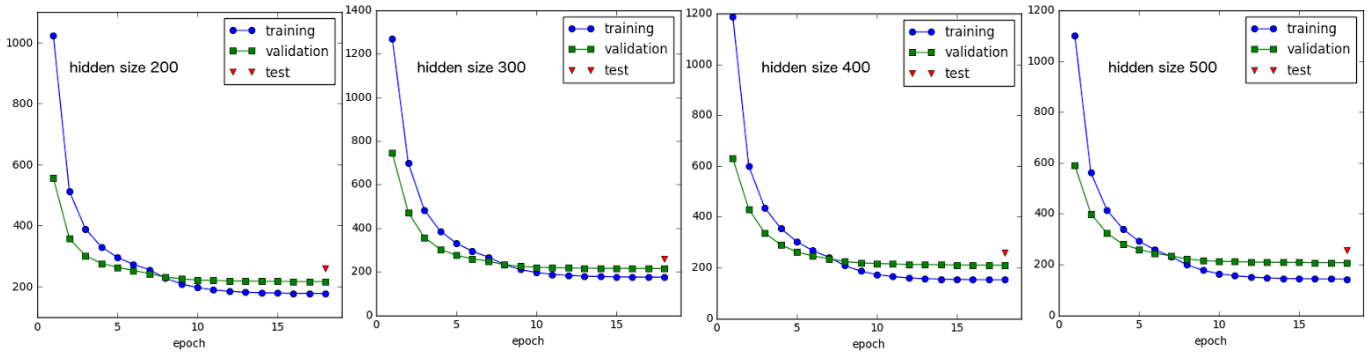
The keep probability is a tricky parameter. The value, 1/keep_prob, will be used to determine dropouts. The best value would depend on the number of hidden layer. For the hidden size 200, the best value was 0.5.

Size of Hidden Layer

The last parameter tested was the number of LSTM cell in the hidden layer. For this comparison, number of layers: 3, forget bias: 1.0, keep probability: 0.5 were used looking at previous results.

Parameters: layers: 3, keep prob: 0.5, forget bias 1.0

hidden size	200	300	400	500
Training	176.828	174.751	151.965	142.349
Validation	215.652	214.271	209.712	206.931
Test	259.893	257.352	257.698	255.721



The results showed the overfitting problem was solved, while the increase in number of hidden layers (LSTM cells) didn't contributed much to improve the performance. Although the hidden size 500 gave the best result, the difference was small. As the number of hidden layers increases, the training time dramatically gets longer. The hidden size 500 took about 6 hours while 200 took about 3 hours for training. Considering the training time length, size 300 may be an affordable size.

4 Layers and Best Parameters So Far

As mentioned in the number of layers comparison section, the last training was done on 4 layers and the best parameter combination -- number of layers: 4, forget bias 1.0, keep probability 0.5, hidden size: 500.

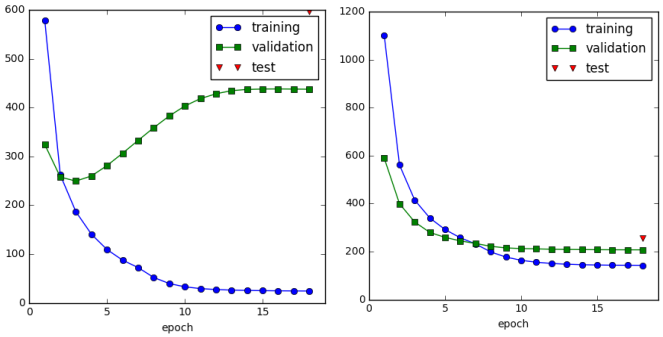
Parameters: layers: 4, forget bias 1.0, keep prob: 0.5, hidden size 500

Training	179.954
Validation	226.650
Test	267.669

The result wasn't good as expected. Probably, too much neurons were used compared to the input data size.

RESULTS

Considering the result in the Parameter Tuning section, the best parameter combination would be -- number of layers: 3, forget bias: 1.0, keep probability: 0.5, number of neurons: 500. For comparison to the basic model, parameters and perplexity curves are put together here. The graph on the left is the basic, right is the best.



Parameter	Basic Model	Best Model
number of layers	1	3
forget bias	0.0	1.0
keep probability	1.0	0.5
hidden size	200	500

Generated sequence of words

The 200 sequence of words were generated starting from “Elizabeth” as the seed word. For comparison, sequences from the basic and best models are shown below. The sequences look like a meaningful sentence fragment are highlighted.

- Basic Model

  Elizabeth consented. soon as she was that she related the lawn to the post, occurrences to Donwell, the carriage did not make Elinor at all Perry!-- out, and Charlotte began to answer Lucy, with increasing astonishment at Hartfield-- she had not been sorry to see him from want. a blush, yet she was convinced that Mr. Darcy loved no other rather for her; and she was obliged to his. saw that she wished it particularly to Jane. real civility independent, except gentle kingdoms, they had hitherto been in a healthfulness of want of pleasure from her behaviour, in asking that she could give her despised with the deepest solicitude of a boys. length, though boundary on him the little one whom he spoke with so little uneasiness as could at Rosings, and entered into her eye, rejoicing in giving zeal to the ladies. Jane, you would give me cruelty to prevent my letter, sights the confession of no concern, though you are not in a way to be engaged her own friends, or express than I am, for a few days came back because it was the compliment I was of removing to me. before? I thought Elizabeth, that some one

- Best Model

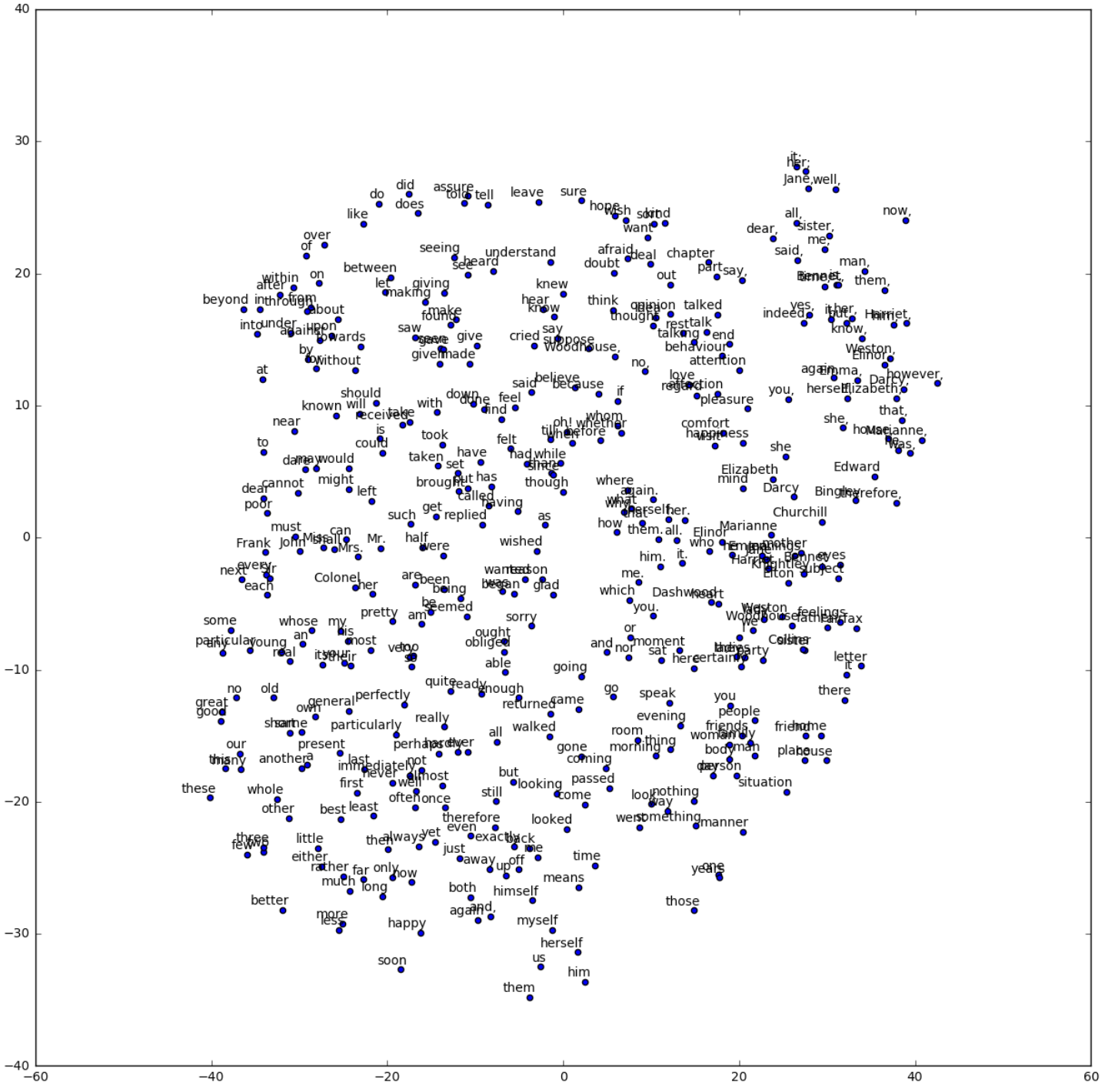
  Elizabeth justify his sister his abhorrence. reproof was an immediate civilities, but Mrs. Elton thought, with Mr. Churchill, but so, and I could certainly say to blush than and my situation, and my hope in questions; but I should speak in seeing of its beauty, reaching your family as now, if he was tranquil I go at the other that much; her brain could not make them well indifferent enough to undeceive education to express his engagement, then, said she, when she comes, she does not avoid wondering as yours. Mr. Darcy was clue, she could not repent. guessed Wickham was obliged to be gone forward again. quitted what passed, he added, in some return; said Marianne, about grave letter and agreed to his chair, affectedly had been, but Miss Woodhouse left me about, almost far to excite an't. the succeeding four acquainted; however, in the arrival, any higher other, and yet express their mother-in-law for advantage allowed repentance, my dear, said Elinor, that all the arrival, that the daughter was left to

```
her, who seemed before; he was forced there, when she fancied so her daughter, and rather were
graciousness, and it was hitherto happy however, was now she doubted
```

The basic mode doesn't have many highlighted areas. Compared to that, in the best model, the highlighted areas were increased and got longer. The result was improved. However, it's still much more to be done to make this sequence of words as if Jane Austen wrote.

Word2Vec

This is a two-dimensional visualization of vocabulary similarity, 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.



CONCLUSION

By training, RNN could generate somehow readable sentences. However, it is still far from mimicking Jane Austen. 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 of 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