A Descriptive Test Auto-Generation and Evaluation

Honglin Bao, Jiaoping Chen

1. Introduction and Motivation

Compared to classical language models that use probabilistic models to predict the next word in the sequence given the words that precede it, neural language model (NLM) is a popular and preferred approach to generate language models. The reason behind is NLM incorporates word embeddings and recurrent neural network (RNN) algorithm. (1) The word embeddings allow words with similar meaning to have similar representation. (Kim et al. 2015) presents that word embeddings obtained in NLMs own the property that semantically close words are likewise close in the induced vector space. (2) RNN allows to learn how to represent memory because neurons with input from recurrent connections represent short term memory, which leads to better performance, especially for Long Short-Term Memory (LSTM).

In sum, because LSTM-based NLM allows to use a distributed representation where different words with similar meanings have similar representation, and also allows to use a large context of recently observed words when making predictions, in this study, *we will utilize the state-of-art LSTM-based NLM approach to implement one of language modeling tasks, the text generation task*. We will answer the following questions: (1) how to prepare text for developing a language model; (2) how to create a generative model for text with a learned word embedding and an LSTM hidden layer; (3) how to generate new text from the source text using the learned language model; (4) how to evaluate the generated text through BLEU and coherence measurements.

2. Problem Definition

The main problem we focus on is that given a sequence of fixed n words, for example, 50, as the input, how to generate a next word which is the n+1 item of the sequence using language models. In other words, the proposed model must learn which is the next most likely word given a sequence of n words. Moreover, we will use the proposed model recursively with output passed as input (50 words to 50 words). That is, it will statistically predict the probability of each word given an input sequence of text. The predicted word will be fed in as input to in turn generate the next word.

The general idea to solve this problem is to apply an Embedding Layer to learn the representation of words, and an LSTM to learn to predict words based on specific contexts.

3. Data

We choose a book, *The Republic* by Plato, a classical Greek philosopher Plato's most famous work. It is one of the world's most influential works of philosophy and political theory. This book contains 10 chapters, and each chapter is structured as a conversation or dialog on the topic of order and justice. The entire book in English is available on the Project Gutenberg website[1], so we could download the ASCII text version directly. We delete standard header and footer texts that do not belong to the original text and then get a text file with about 15,802 lines of text. After that, we did some data preprocessing by splitting words based on white space, removing all punctuations, removing special words that are not alphabetic and normalizing all words to lowercase. After that, a smaller vocabulary set with 118,684 tokens in total is generated. As we mention before, we then rearrange those tokens into sequences of 50 input words and 1 output word by applying line-by-line sequences approach for further modeling processes. We also have an alternative data set which is extracted from a random wikipedia link through BeautifulSoup package. We applied the same procedure to clean the data.

[1] Project Gutenberg website: https://www.gutenberg.org/ebooks/1497

4. Multi-class Classification and Generation

4.1 Word Embedding. Before implementing word embedding for each input sentence, we would first map input words to integers. So that we could convert the prediction to numbers and then check its corresponding words using the same mapping. Therefore, we firstly identify all unique words in data and assign each word a unique integer, and then convert all word-based sequences into a list of integers.

Word embeddings are capable of capturing the context of a word in a document, semantic and syntactic similarity, relation with other words, etc. They are vector representations of particular words to solve the n-gram data sparsity issue (that is, most possible word sequences are not observed in training even though we assume the probability of a word depends on the previous n words). Through word embeddings, words are represented as parameters and as inputs of a neural network, and those parameters can be learned during the training process. As a high-dimensional sparse vector in which all elements are integers. Each cell in the vector represents a separate English word; the value in a cell represents the number of times that word appears in a sentence.

4.2 Multiclass Classification. One-of-many classification. Each sample can belong to one of C classes (assume C classes in total, in this study, C is the size of unique vocabularies). The deep network will have C output neurons that can be gathered in a vector **s** (Scores). The target (ground truth) vector **t** will be *a one-hot vector* with a positive class and C−1 negative classes. This task is treated as a single classification problem of samples in one of C classes. Notice that the reason to implement one-hot encoder for output words is that the model learns to predict the probability distribution for the next word and the ground truth from which to learn from is 0 for all words except the actual word that comes next.

Problems may arise when there is no ordinal relationship, and letting the representation depend on any relationship may impair the learning ability. In these cases, we hope to give the network more expressive power in order to learn probability for each possible label. This helps make the problem easier for network modeling. When a **one hot encode** is used to output variables, it can provide a more detailed set of predictions than a single label.

Regarding the loss function of multi-class classification models, we implement the categorical cross entropy loss, a sum of separate loss for each class per observation, where is the ground truth probability of class c (it is 1 if class c is true; 0 otherwise).

4.3 LSTM. We set the size of the embedding vector space as 50. Two LSTM hidden layers with 100 memory cells each. After that, a dense fully connected layer with 100 neurons and RELU activation function is connected to the LSTM hidden layers to convey extracted features from the sequence. Because the output layer represents the predicted word as a single vector, a softmax activation function is introduced to ensure the outputs have the characteristics of normalized probabilities. Next, because this model is to learn a multi-class classification, the model is compiled by specifying the categorical cross entropy loss. For optimizer, we choose an efficient algorithm that uses mini-batch gradient descent, Adam approach. In our study, we fit on the data for 100 training epochs with a modest batch size of 128 to speed things up.

After completing the training process, we move to the data generation. We select a random line of text from the input text and then generate new words, one at a time. It is noticed that the chosen input words must be encoded to integers using the same mapping and then word embedding techniques.

5. Results and Evaluation

5.1 Result. We generate 2 sentences in Table 1. The left is fed-in original sentence, the right is the corresponding generated sentence. In terms of content, it is interesting that the generated sentence tells a totally different story compared to original content. But we also notice readability is not quite good in our generated sentences.

Table 1. Result of text generation

Original Sentences	Generated Sentences
when he said that a man when he grows old may learn many things for he can no more learn much than he can run much youth is the time for any extraordinary toil of course and therefore calculation and geometry and all the other elements of instruction which are	preparation for dialectic should be presented to the name of idle spendthrifts of whom the other is the manifold and the unjust and is the best and the other which delighted to be the opening of the soul of the soul and the embroiderer will have to be said at
the two philosophers had more in common than they were conscious of and probably some elements of plato remain still undetected in aristotle in english philosophy too many affinities may be traced not only in the works of the cambridge platonists but in great original writers like berkeley or coleridge	other has knowledge are greatest as come on with be conversation to and perhaps any time to platonic keep but by as aristocracies as ideal acknowledge either every sympathy probably become find no so as other words to other century platonic by as well same numbers seem irregular if state

5.2 Evaluation. To further investigate generated results, we apply two kinds of evaluation for text generation, a cross text evaluation (BLEU) and an intrinsic text evaluation (Coherence Score). The cross-text evaluation is to identify the relation between input and output; the intrinsic text evaluation is to measure the quality of artificially generated outputs.

a. BLEU Score

We followed Papineni et al. 2002 to implement the complete version of BLEU where we consider modified n-gram precision on blocks of text, clipped n-gram counts for all the candidate sentences, and Sentence Brevity Penalty. **Evaluation result**: final bleu score is 0.24.

b. Topic Coherence

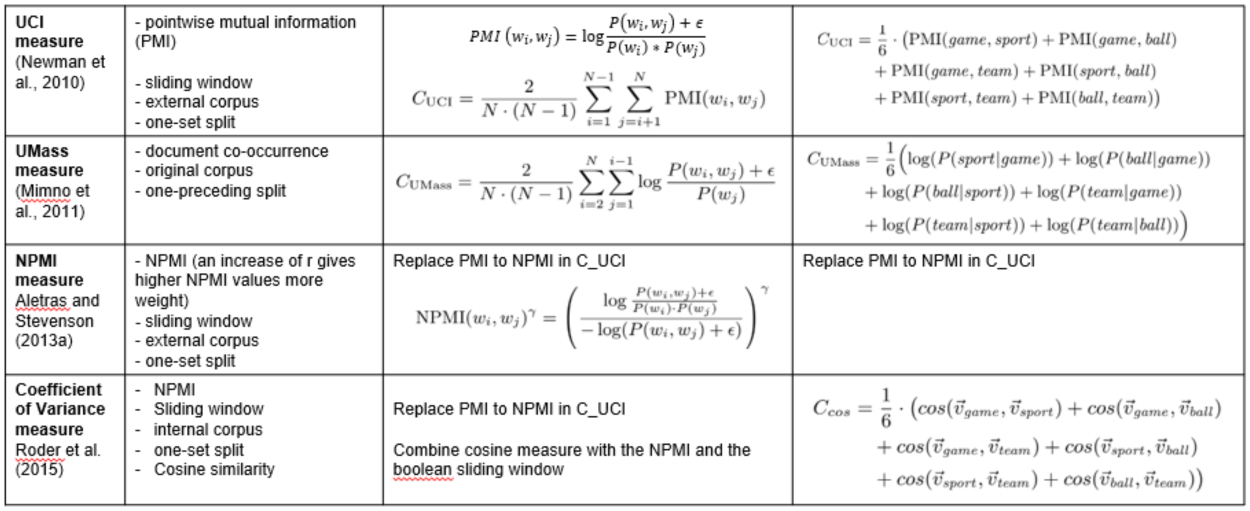
The sentence coherence, for example, perplexity score, is the inverse probability of the test set, normalized by the number of words. It can be regarded as a branching factor (the weighted average number of choices a random variable has). It can also be represented as an exponential of cross-entropy given the assumption of all words have the same probability. However, it is unrealistic to assume all words have the same probability. Past research has shown that predictive likelihood (or equivalently, perplexity) and human judgment are often not correlated, and even sometimes slightly anti-correlated (Hu, Boyd-Graber, and Satinoff 2011). Therefore, we explore more about topic coherence score in this project and see how it can be used to evaluate results.

A relatively coherent topic means words can support each other for a specified topic. That is, top frequent words from the same topic should be well classified by the ideal model. Besides, coherence scores are mainly comparative: if topic B has a higher coherence score than topic A, it is 'better' (more coherent). For example, a coherent fact set: {"the game is a team sport", "the game is played with a ball", "the game demands great physical efforts"}.

Table 2 summarizes 4 types of topic coherence measures. C_uci measure is based on a sliding window and the pointwise mutual information (PMI) of all word pairs of the given top words; C_umass is based on document co-occurrence counts, a one-preceding segmentation and a logarithmic conditional probability as confirmation measure; C_npmi is an enhanced version of the C_uci coherence using normalized pointwise mutual information (NPMI), and C_v measure is based on a sliding window, one-set segmentation of the top words and an indirect confirmation measure that uses NPMI and the cosine similarity.

In this project, we implement C_v measure based on Latent Dirichlet Analysis (LDA). Our ideal goal is to generate thousands of sentences and then compare topic coherence scores between original sentences and generated content. We suppose generated content would have higher topic coherence because they are created by probabilistic models that consider word dependences. However, the unexpected long generation time of each sentence limits us to reach that goal. That's why we choose an alternative approach to test our hypothesis. We compare topic coherence scores between generated contents (2 generated sentences + other 17 original sentences) and original contents (2 fed-in sentences + same 17 original sentences).

Table 2. Topic coherence measurements. Example, a topic with four words {game, sport, ball, team}



UCI measure (Newman et al., 2010)	- pointwise mutual information (PMI) - sliding window - external corpus - one-set split	$PMI\left(w_i, w_j\right) = \log \frac{P(w_i, w_j) + \epsilon}{P(w_i) * P(w_j)}$ $C_{\mathrm{UCI}} = \frac{2}{N \cdot (N-1)} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \mathrm{PMI}(w_i, w_j)$	$C_{\mathrm{UCI}} = \frac{1}{6} \cdot (\mathrm{PMI}(game, sport) + \mathrm{PMI}(game, ball)$ $+ \mathrm{PMI}(game, team) + \mathrm{PMI}(sport, ball)$ $+ \mathrm{PMI}(sport, team) + \mathrm{PMI}(ball, team))$
UMass measure (Mimno et al., 2011)	- document co-occurrence - original corpus - one-preceding split	$C_{\mathrm{UMass}} = \frac{2}{N \cdot (N-1)} \sum_{i=2}^{N} \sum_{j=1}^{i-1} \log \frac{P(w_i, w_j) + \epsilon}{P(w_j)}$	$C_{\mathrm{UMass}} = \frac{1}{6} \Big(\log(P(sport \mid game)) + \log(P(ball \mid game))$ $+ \log(P(ball \mid sport)) + \log(P(team \mid game))$ $+ \log(P(team \mid sport)) + \log(P(team \mid ball)) \Big)$
NPMI measure Aletras and Stevenson (2013a)	- NPMI (an increase of r gives higher NPMI values more weight) - sliding window - external corpus - one-set split	Replace PMI to NPMI in C_UCI $\mathrm{NPMI}(w_i, w_j)^{\gamma} = \left(\frac{\log \frac{P(w_i, w_j) + \epsilon}{P(w_i) \cdot P(w_j)}}{-\log(P(w_i, w_j) + \epsilon)} \right)^{\gamma}$	Replace PMI to NPMI in C_UCI
Coefficient of Variance measure Roder et al. (2015)	- NPMI - Sliding window - internal corpus - one-set split - Cosine similarity	Replace PMI to NPMI in C_UCI Combine cosine measure with the NPMI and the boolean sliding window	$C_{cos} = \frac{1}{6} \cdot (cos(\vec{v}_{game}, \vec{v}_{sport}) + cos(\vec{v}_{game}, \vec{v}_{ball})$ $+ cos(\vec{v}_{game}, \vec{v}_{team}) + cos(\vec{v}_{sport}, \vec{v}_{ball})$ $+ cos(\vec{v}_{sport}, \vec{v}_{team}) + cos(\vec{v}_{ball}, \vec{v}_{team}))$

For LDA modeling, we also generate Bigram and Trigram words and then incorporate those pairs into dictionaries. Besides, we compute the internal corpus, which is represented by the format of "word index-word frequency". After that, we implement LDA topic model with coefficient variance measure C_v to evaluate given contents. It is noted that the number of topics is a hyper parameter which is hard to adjust, in this case, we choose as three due to lack of inputs. Finally, we obtain two topic coherence scores: Evaluation result: for generated texts, 0.19; for original texts: 0.67.

6. Conclusion and Future work

Conclusion. In this project, we first prepare text for developing a language model. Then we design and fit a neural language model with a learned embedding and LSTM. We further generate new text from the source text using the model. Finally, we investigate two kinds of evaluation for text generation results.

Future works. Several potential works can be investigated. First, potential speed-up algorithms are beneficial to generate more texts as population. Second, current supervision is to minimize cross entropy. We should think about combining loss and coherence to create a new optimization target. Third, we can apply this novel topic coherence score as the fitness of evolutionary search, to see if adding topic-level information would lead to better interesting results. Fourth, besides topic coherence, can we generate a better sentence coherence measurement?

Reference:

Hu, Yuening, Jordan Boyd-Graber, and Brianna Satinoff. 2011. “Interactive Topic Modeling.” In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 248–257. Portland, Oregon, USA: Association for Computational Linguistics. https://www.aclweb.org/anthology/P11-1026.

Kim, Yoon, Yacine Jernite, David Sontag, and Alexander M Rush. 2015. “Character-Aware Neural Language Models,”.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. “Bleu: A Method for Automatic Evaluation of Machine Translation.” In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 311–318. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics. https://doi.org/10.3115/1073083.1073135.