Building A Neural Language Model

matthew@cinnamon.is, marc@cinnamon.is

January 2019

1 Language Model

One of the most important task in Natural Language Processing is language modeling which compute the probability distribution of a word $\boldsymbol{x}^t$ given its previous sequence of words $\boldsymbol{x}^1, \boldsymbol{x}^2, ..., \boldsymbol{x}^{t-1}$

$$P(\boldsymbol{x}^t|\boldsymbol{x}^{t-1}, ..., \boldsymbol{x}^1) \tag{1}$$

where $\boldsymbol{x} \in V = \{\boldsymbol{w}_1, ..., \boldsymbol{w}_{|V|}\}$, V is the vocabulary.

With this property, we can see Language Model can appear in search engine suggestion or our smartphone keyboard typing prediction.

Another way to describe language modeling is the computation of occurrence probability of a number words in a sequence $P(\boldsymbol{x}^1, ..., \boldsymbol{x}^T)$. Based on Eq. 1, we can compute the probability of the whole sequence:

$$\begin{aligned} P(\boldsymbol{x}^1, ..., \boldsymbol{x}^T) &= P(\boldsymbol{x}^1) \times P(\boldsymbol{x}^2|\boldsymbol{x}^1) \times ... \times P(\boldsymbol{x}^T|\boldsymbol{x}^{T-1}, ..., \boldsymbol{x}^1) \\ &= \prod_{t=1}^{T} P(\boldsymbol{x}^t|\boldsymbol{x}^{t-1}, ..., \boldsymbol{x}^1) \end{aligned} \tag{2}$$

The above equation is useful for speech recognition or translation systems where we need to determine whether a word sequence is a accurate predictions or not. For example, a language model can help a translation system score alternative word sequences (*I have, I had, I has, me have* or *me had*) to identify the most likely translation sequence.

2 Recurrent Neural Network

Recurrent Neural Network (RNN) (illustrated in Fig. 1) can be seen as a "wide" neural network (in comparison to the Deep Neural Network) where instead of stacking network layers one after another, RNN consists of a dynamic number of hidden layers ($\boldsymbol{h}^t$) shared same parameters ($\boldsymbol{W}_h$, $\boldsymbol{W}_x$, $\boldsymbol{W}_y$) and been fed sequentially in time domain.

$$\boldsymbol{h}^t = \sigma(\boldsymbol{W}_h \boldsymbol{h}^{t-1} + \boldsymbol{W}_x \boldsymbol{x}^t) \tag{3}$$

$$\boldsymbol{y}^t = softmax(\boldsymbol{W}_y h^t) \tag{4}$$

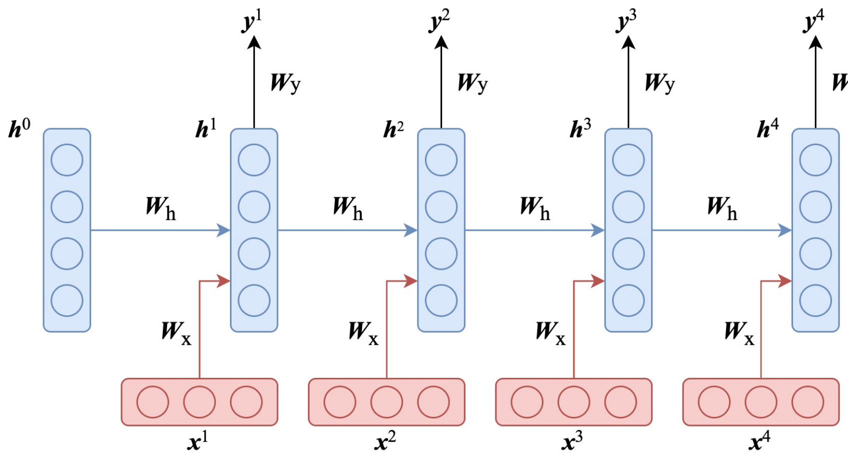

Figure 1: Recurrent Neural Network

where:

- $\boldsymbol{x}^t \in \mathbb{R}^d$ is the vector representation of the word $\boldsymbol{w}^t \in \{\boldsymbol{w}^1, ..., \boldsymbol{w}^T\}$ in sequence of tokens with length T. With d is the dimension of the embedding vectors.
- $\boldsymbol{h}^t \in \mathbb{R}^{D_h}$ are the hidden layer features at timestep t respectively. With D_h is the number of hidden states in each hidden layers.
- $\boldsymbol{y}^t \in \mathbb{R}^{|V|}$ is the probability distribution over the vocabulary V at each time-step t.
- $\boldsymbol{W}_x \in \mathbb{R}^{D_h \times d}$, $\boldsymbol{W}_h \in \mathbb{R}^{D_h \times D_h}$ and $\boldsymbol{W}_x \in \mathbb{R}^{|V| \times D_h}$ are the shared parameters of RNN.

As a results, instead of only accepting a fixed-size vector as input (e.g. an image) and producing a fixed-size vector as output (e.g. probabilities of different classes) like Vanilla Neural Network (as well as Convolution Neural Network), RNN allows us to operate over sequences of vectors and take certain information from the previous steps in the sequence into account when do the current prediction. Another benefit of RNN is the model size will not increase for longer input sequence.

In practical usage, we can stack RNN on top of each other or even stack RNN with different input directions (Bidirectional RNN) to increase performance of the model under certain circumstances. Another factor might affect model's performance is the number of hidden state units which is similar to Vanilla Neural Network.

However, there are still some drawbacks of RNN, one of them is the speed performance is low due to its sequential property. Another problem is the difficulty of accessing information with a long sequence caused by gradient vanishing or exploding, which can be solved with some training tricks or using a fancier types of RNN (LSTM or GRU).

3 Build A LM

That's enough of theory. Our main task in this challenge is to build a Neural Language Model with RNN using your deep learning framework of choice (Tensorflow, Pytorch, Keras, ...). Our LM should be able to **generate a paragraph** on its own, with few input words. Our report should answer the following questions:

- How do we use RNN for language modeling?
- What is the loss function for training our RNN LM?
- How can we evaluate the performance of our LM?
- What factors of RNN affect the performance of our LM? Go into the details how they affect our LM.
- How can we reduce the information missing problem of RNN in training our LM? Show their effects on our LM's performance.

3.1 Data

We have already learnt about data gathering & cleaning in the previous challenge about word embedding, in order to move quickly in this challenges, we can use the pre-processed data from researchers at Salesforce (Wikitext-2 is recommended if we have low resources for computation).

Or if we want to build a true writer, we can crawl the essays from Paul Graham (the training data will be smaller) to train our LM.