Building A Neural Language Model

Phan Viet Hoang

February 28, 2020

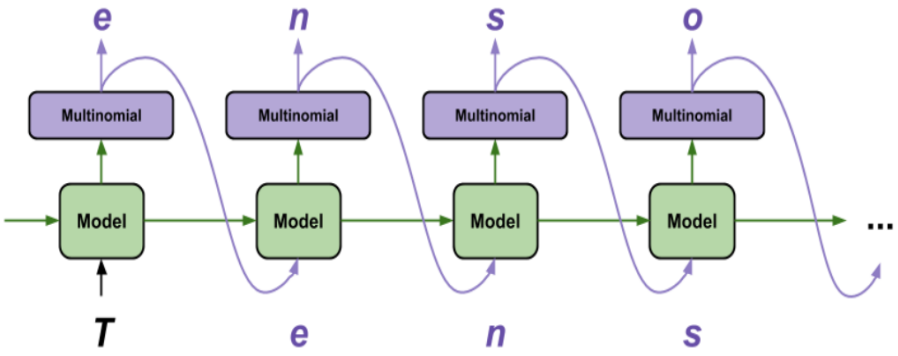
1 My work

- Pre-process data
- Build 2 Neural Language Model using Keras deep learning framework with tw2 different approaches and compare outcomes
- Generate a paragraph with few input words using trained model

• The first model is character-based language model, which means I will divide the given text in to the same length sequences (100 characters in my case).For each input sequence, the corresponding targets contain the same length of text, except shifted one character to the right.

INPUT DATA	TARGET DATA
You want to b	ou want to be
a NLP researche	NLP researcher
try your bes	ry your best

This is how my model look alike

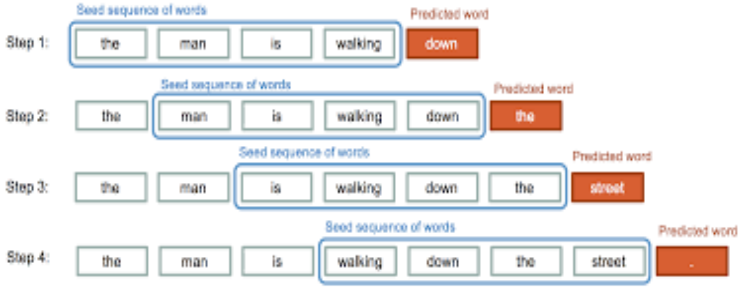


• The second one is little bit different,that I created the paragraph generator working on word-level instead of character-level, as the previous model. As a simplified example, if each sentence is a list of five words, then the

target is a list of only one element, indicating which is the following word in the original text

INPUT DATA	TARGET DATA
You want to be a NLP try your best to	a researcher impress

Detailed text generator model can be interpreted as below



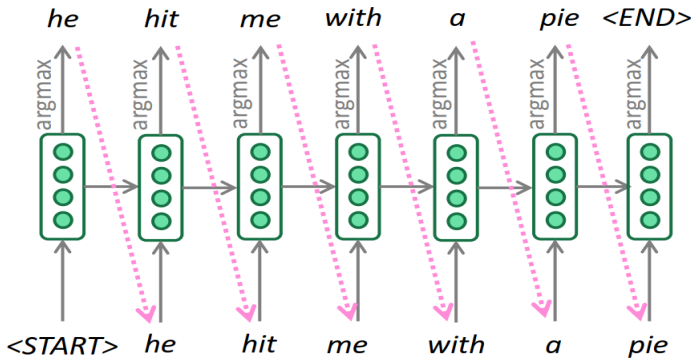
2 Questions

2.1 How do we use RNN for language modeling?

- My approach is slicing the input text into sequences. Given a bunch of sequences and expect that RNN will be trained to predict the output-the following element at each time step thanks to it's ability learning from sequential data and generate similar data

- Traditional RNN's architecture (and also others advanced versions) is perfect for modelling languages because language is a sequence of words and each next word is dependent on the words that come before it, since RNNs can maintain an internal state that depends on the previously seen elements

- For a complete generated data from probability distribution given from our model output, I implement the greedy decoding method: take most probable word on each step and try to decode until the model produces a ¡/x00¿ token (end of string signal added before my data encoding) or when the text has T elements (where T is some pre-defined cutoff threshold):



- But there exist a much more efficient decoding method: Beam search. On each step of decoder,it keeps track of the k most probable candidates and decode until:

- We reach timestep T (where T is some pre-defined cutoff)

or

- We have at least n completed hypotheses (where n is pre-defined cutoff)

- We may encounter this method in the following assignments because of it's popularity

2.2 What is the loss function for training our RNN LM? How can we evaluate the performance of our LM?

- Cross-entropy will be used in my implement like in the case of classification tasks because we've added the dense layer on each time step and treat the modeling language problem as a word prediction on each time step

- Given words $x_1, x_2, .., x_n$; my language model products the x_{t+1} word by taking the $argmax$ probability distribution taken over the vocabulary:

$$P(x_{t+1} = v_j | x_t \cdots, x_1) = \hat{y}_j^t$$

where v_j is a word in the vocabulary.

- The predicted output vector $\hat{y}^t \in R^V$ is a probability distribution

$$CE(y^t, \hat{y}^t) = -\sum_{i=1}^{|V|} y_i^t \log \hat{y}_i^t$$

where y_t is the vector corresponding to the ground truth word

- Other possible LM evaluation metric is perplexity (I've encounter this loss when trying to visualize the word representation with t-SNE method) :

- Mathematically, the perplexity of a language model is defined as:

$$PP^t = \frac{1}{P(x_{t+1}^{pred} = x_{t+1} | x_t \cdots, x_1)} = \frac{1}{\sum_{j=1}^{V} y_j^t \cdot \hat{y}_j^t}$$

suppose y_i^t is the only nonzero element of y^t. Then, note that:

$$CE(y^t, \hat{y}^t) = -\log \hat{y}_i^t = \log \frac{1}{\hat{y}_i^t}$$
$$PP(y^t, \hat{y}^t) = \frac{1}{\hat{y}_i^t}$$

Then, it follows that:

$$CE(y^t, \hat{y}^t) = \log PP(y^t, \hat{y}^t)$$

Which means minimizing the cross-entropy is the same as minimizing the perplexity. In fact, perplexity can be used as the language model evaluation metrics

- In other words, these two loss function try to minimize the "distance" of two distributions. Following that, model will be able to learn, from the sample text, a distribution close to the empirical distribution of the language

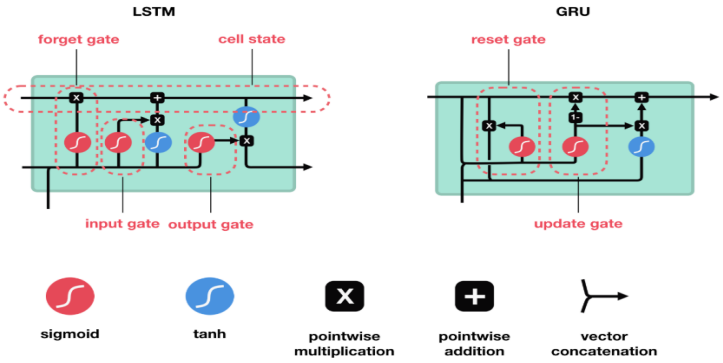
- Metric for the quality of generated text in my assignment is the traditional BLEU score-which is popular in machine-translation tasks. It can easily explained as follow:

$$p_n = \frac{\sum_{S \in C} \sum_{ngram \in S} Count_{matched}(ngram)}{\sum_{S \in C} \sum_{ngram \in S} Count(ngram)}$$

where:
p_n is the n-grams BLEU-precision score taken from every sentence S from corpus C

2.3 What factors of RNN affect the performance of our LM? Go into the details how they affect our LM

- RNN's performance in capturing long term dependencies: advanced architectures seems to be better choice than the vanilla one thanks to update gate (GRU) or both update and a forget gate (LSTM)

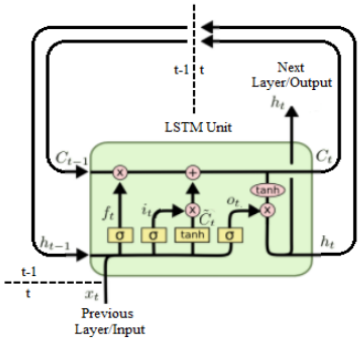


- Number of hidden unit: this is a measure for learning capacity of a neural network, it reflects the number of learned parameters. The bigger it is, the more complex model

2.4 How can we reduce the information missing problem of RNN in training our LM? Show their effects on our LM's performance

We can try other advanced RNN model and compare their performance (just replace one layer and retain not only model architecture but also hyper-parameters combination):

- LSTM



$$
\begin{aligned}
f_t &= \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right) \\
i_t &= \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right) \\
\tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
o_t &= \sigma\left(W_o \left[h_{t-1}, x_t\right] \; + \; b_o\right) \\
h_t &= o_t * \tanh\left(C_t\right)
\end{aligned}
$$

Detailed explain:

• First, the previous hidden state and the current input get concatenated. We'll call it combine.

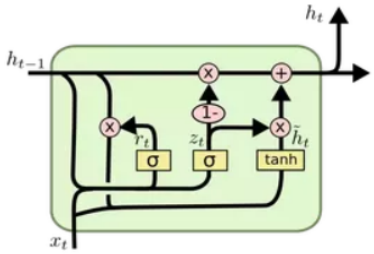
• Combine get's fed into the forget layer. This layer removes non-relevant data. A candidate layer is created using combine. The candidate holds possible values to add to the cell state.

• Combine also get's fed into the input layer. This layer decides what data from the candidate should be added to the new cell state.

• After computing the forget layer, candidate layer, and the input layer, the cell state is calculated using those vectors and the previous cell state.

• The output is then computed.

- GRU



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

• The update gate acts similar to the forget and input gate of an LSTM. It decides what information to throw away and what new information to add.

• The reset gate is another gate is used to decide how much past information to forget.

- GRU's has fewer tensor operations; therefore, they are a little speedier to train then LSTM's. In my assignment, with the same number of hidden units (and much less pararameters) their performances are relatively similar:

- Implement result (on character level model)

Model accuracy (%)				
RNN architecture	Epoch 2	Epoch 3	Epoch 4	Epoch 5
Simple RNN	54.08	57.22	58.95	59.99
GRU	58.78	61.21	62.50	63.26
LSTM	59.14	61.79	63.05	63.86

The word-based language model performs poorly on the test set, let figure out the reasons by comparing to the character-level LM

	Word level LM	Character level LM
Vocabulary size	Large, due to the variety of word, up to many thousands of words	Relatively small about 40 English-language characters (lower case)
Number of hidden units	A small hidden units can capture the paragraph context	Requires much bigger hidden units to successfully model long-term dependencies
Computation cost (no. parameters)	Larger	Smaller
Time cost (second per epoch)	Smaller	Larger
Flexibility	Unusuals word can heavily affect model's performance (also because of out exhausted decoding technique)	Can easily overcome this problem thank to the big size gram (100 in my case) but also usually generate weird characters

The detailed comparision is presented in python notebook file

References

[1] CS224n: Natural Language Processing with Deep Learning

[2] Evaluation Metrics for Language Modeling - The Gradient

[3] Text generation with an RNN

[4] Advantage of character based language models over word based

[5] Perplexity