UNIVERSITY OF WATERLOO

Introduction to Machine Learning
Spring 2019

By Rongrong Su
Statistics and Actuarial Science Department
r2sue@uwaterloo.ca

Department of Statistics and Actuarial Science

Introduction	1
Technique and Theory	1
Evaluation	5
Further work	5
Reference	5
Appendix	6

Introduction

Natural Language Processing, or NLP for short, is broadly defined as the automatic manipulation of natural language, like speech and text, by software. More and more complicated and efficient machine learning model especially neural networks for natural language arise, for instance, RNN (basic model), LSTM, BERT and XLNET.

The dataset I used in project is toxicity classification from Kaggle competition, which requires participators to come up machine models to classify if a comment is toxic or not given comment text. Additionally, a subset of comments have been labelled with a variety of identity attributes, representing the identities that are *mentioned* in the comment.

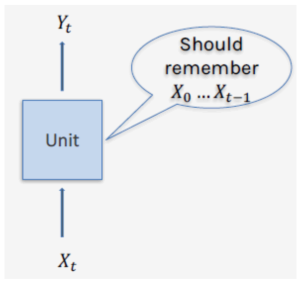
This project introduce basic concepts of RNN, LSTM and BERT, and implement those models. Then evaluation and comparison of above model are finished.

Technique and Theory

1. RNN (Recurrent Neural Network)

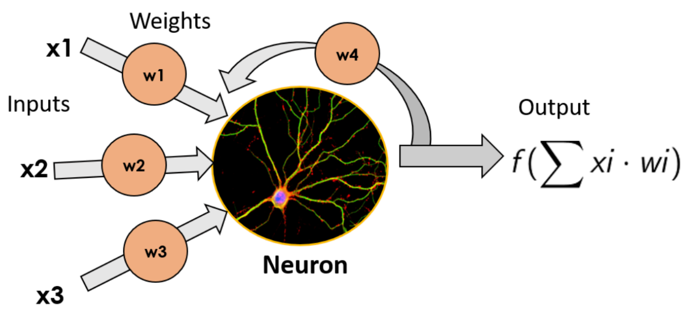
Recurrent neural networks are a special kind of neural networks that are designed to effectively deal with sequential data, which includes time series, text documents (a sequence of word) or audio.

The main difference between RNN and CNN is that RNN contains neurones that are be able to remember what it has been before, which is called unit's state or hidden state or memory.

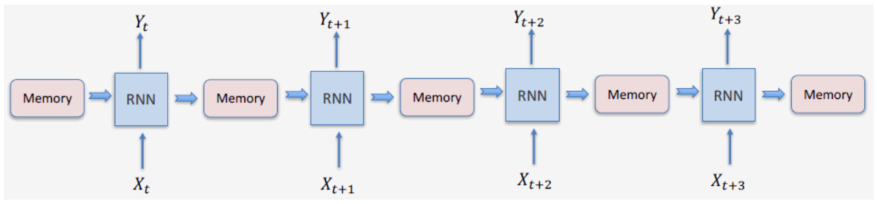


P. Protopapas, CS109b, Harvard FAS

The way RNNs do this, is by taking the output of each neurone, and feed it back to it as an input. Therefore, it does not only receive new pieces of information in every step, but it also adds a weighted previous output as another new information.

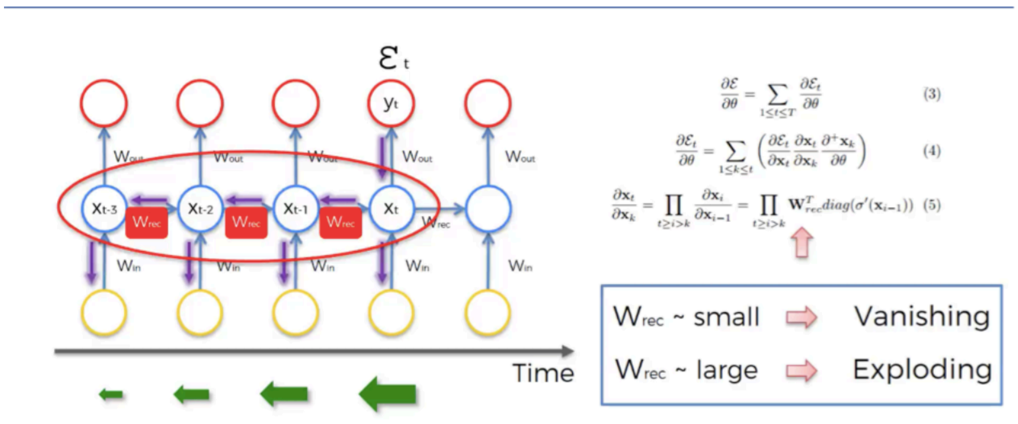


A recurrent neuron, where the output data is multiplied by a weight and fed back into the input



P. Protopapas, CS109b, Harvard FAS

The problem with RNNs is that as time passes by and they get fed more and more new information, they start to 'forget' about the previous information, as it is diluted by new information. Meanwhile, RNNs lead to vanishing gradient problem. As we know, weights assigned at start of the neural network with the value are close to 0, and then the neural network train them up. However, when we start weight recurring (the weight that is used to connect the hidden layers to themselves in the unrolled temporal loop) close to 0 and multiply inputs at each step by those values, the gradients become less and less with each multiplication. But if we start with large weight recurring, we also will have exploding gradient problem. (More details about gradient vanishing problem can be found in appendix.)



$$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{1 \leq t \leq T} \frac{\partial \mathcal{E}_t}{\partial \theta} \quad (3)$$

$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1 \leq k \leq t} \left(\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial^+ \mathbf{x}_k}{\partial \theta} \right) \quad (4)$$

$$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{t \geq i > k} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} = \prod_{t \geq i > k} \mathbf{W}_{rec}^T diag(\sigma'(\mathbf{x}_{i-1})) \quad (5)$$

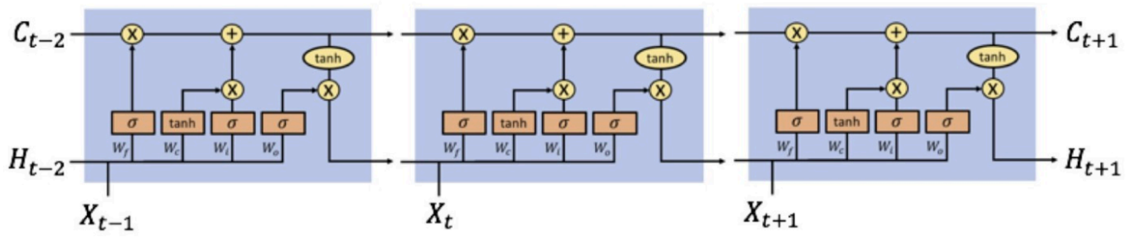
W_{rec} ~ small $\Rightarrow$ Vanishing

W_{rec} ~ large $\Rightarrow$ Exploding

We need some sort of long term memory, which is what LSTMs provide.

2. LSTM (Long Short Term Memory)

LSTMs have a more complex cell structure and three gates (input gate, forget gate and output gate) that allow them to better regulate who to learn or forget from different input sources. And the key to LSTMs is the cell state, the horizontal line running through the top of the diagram.



LSTM network cells at time steps t-1, t, t+1

At each step the cell can decide what to do with the state vector: read from it, write to it, or delete it.
The forget gate controls what information in the cell state to forget, given new information that entered the network. The function is given by: $f_t = \sigma(W_f * [H_t, X_{t+1}] + b_f)$.
The input gate controls what new information will be encoded into the cell state, given the new input information. The output of input's gate is:
$g_t = tanh(W_c * [H_{t-1}, X_t] + b_c) \otimes \sigma(W_i * [H_{t-1}, X_t] + b_i)$.
Then new cell is: $C_t = f_t * C_{t-1} + g_t$. The functioning of an output gate can be seen as three steps:

1. Creating a vector after applying tanh function to the cell state, thereby scaling the values to the range -1 to +1.
2. Making a filter using the values of h_t-1 and x_t, such that it can regulate the values that need to be output from the vector created above. This filter again employs a sigmoid function.
3. Multiplying the value of this regulatory filter to the vector created in step 1, and sending it out as an output and also to the hidden state of the next cell.

The h_t is given by: $o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o), h_t = o_t * tanh(C_t)$

(And the gradient vanishing problem is solved due to cell state. More details can be found in appendix.
)

The above is normal version of LTSMs. There are other variants on LSTMs, for instance, adding 'peephole connection' which is proposed by F.A. Gers and J. Schmidhuber.

There is one more noticeable architecture that can be added in LSTMs: attention layers. The attention mechanism to overcome the limitation that allows the network to learn where to pay attention in the input sequence for each item in the output sequence.

4. BERT (Bidirectional Encoder Representations from Transformers)

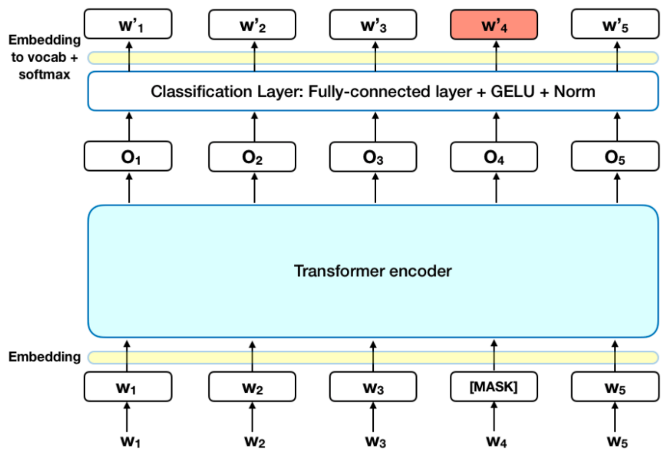
The key technical innovation of BERT model is applying the bidirectional training of transformer, which is in contrast to previous efforts which looked at a text sequence either from left to right or combined left-to-right and right-to-left training.

BERT makes use of transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text. Since BERT's goal is generate a language model, only the encoder mechanism is necessary in transformer. The transformer encoder in BERT reads the entire sequence of words as once. This characteristic allows the model to learn the context of a word based on all its surroundings.

There are two ordinary contribution of BERT: one is novel pre-training tasks called Masked Language Model(MLM) and Next Sentence Prediction (NSP) and a lot of data and compute power to train BERT.

- Masked Language Model (MLM)

In MLM task is converted into tokens, which will be used as an input and output. A random subset of tokens (15% in paper) are masked, and then those tokens are predicted based on non-masked tokens. The final hidden vectors corresponding to the mask tokens are fed into an output softmax over the vocabulary, as in a standard LM. The MLM target allows to represent the context of left and right sides of the fusion, which makes it possible to perform bidirectional learning from the text.

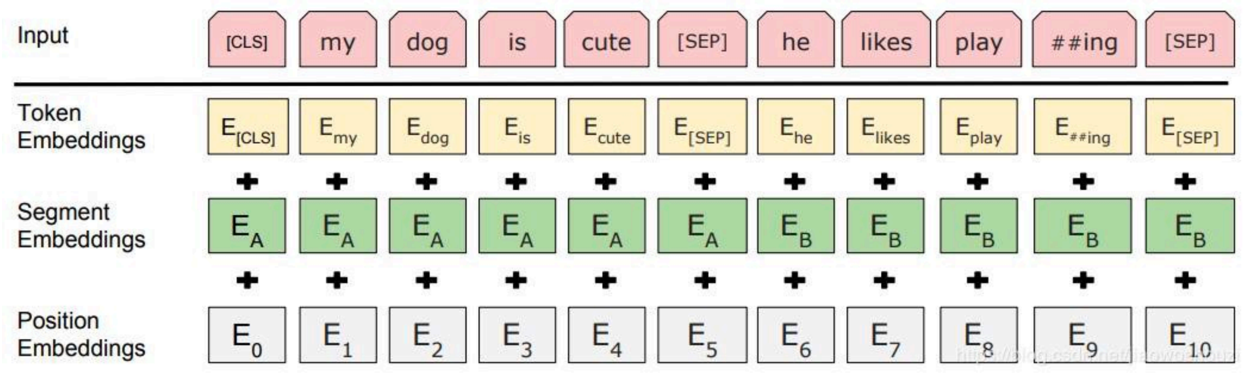


W'1
W'2
W'3
W'4
W'5
Embedding to vocab + softmax
Classification Layer: Fully-connected layer + GELU + Norm
O1
O2
O3
O4
O5
Transformer encoder
Embedding
W1
W2
W3
[MASK]
W5
W1
W2
W3
W4
W5

- Next Sentence Prediction (NSP)

In order to train a model that could understands sentence relationships along with semantic relationships between words, the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original document. There are two datasets: A and B, where 50% of the data B is the next sentence of A, and the remaining 50% of the data B are randomly selected in the corpus. As a result, BERT model will be able to understand relationship between two sentences.

Finally, the inputs after MLM and NSP of BERT are visualized as follows:



Evaluation

As can be seen below, there is no distinct improvements when I tried to use a more complex model. This might due to the random selection of test set. However, overall their performances are great. The current results indicates that to achieve a better model I need to consider embedding methods to combine those basic models' results together.

	LSTM	LSTM - Attention	BERT
Test accuracy	0.9315	0.9322	0.9337

Further work

- More text classification are expected to implement, for instance,TCN, BERT-large-uncased, GPT2, XLNET.
- RNN models could speed up with sequence bucketing method. So, it is a meaningful research topic to compare performance of different sequence bucketing methods.
- Boosting or boosting could be ensemble results from different models (LSTM, BERT-large-uncased,) to achieve a better solution.
- Better text preprocessing could also improve he model performance.

Reference

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Advances in Neural Information Processing Systems, pages 6000–6010.
[2] Devlin, J., Chang, M., Lee, K. and Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. [online] arXiv.org. Available at: https://arxiv.org/abs/1810.04805v2 [Accessed 9 Aug. 2019].
[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Advances in Neural Information Processing Systems, pages 6000–6010.
[4] F.A. Gers ; J. Schmidhuber. Recurrent nets that time and count - IEEE Conference Publication Recurrent nets that time and count - IEEE Conference Publication.
(2000). Ieeexplore.ieee.org.
[5] *Jigsaw Unintended Bias in Toxicity Classification | Kaggle*. (2019). *Kaggle.com*. Available at https://www.kaggle.com/c/jigsaw-unintended-bias-in-toxicity-classification/overview
[6] *Simple LSTM | Kaggle*. (2019). *Kaggle.com*. Available at https://www.kaggle.com/thousandvoices/simple-lstm
[8] *Toxic BERT plain vanila | Kaggle*. (2019). *Kaggle.com*. Available at https://www.kaggle.com/yuval6967/toxic-bert-plain-vanila

Appendix

1. Why there is gradient vanishing problem in RNNs

The gradient of the error at final time step k is: $\frac{\partial E_k}{\partial W}$, which is concerning W = [Wc, Wx].

$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial H_k}\frac{\partial H_k}{\partial C_k}\frac{\partial C_k}{\partial C_{k-1}}\cdots\frac{\partial C_2}{\partial C_1}\frac{\partial C_1}{\partial W} = \frac{\partial E_k}{\partial H_k}\frac{\partial H_k}{\partial C_k}\left(\prod_{t=2}^{k}\frac{\partial C_t}{\partial C_{t-1}}\right)\frac{\partial C_1}{\partial W}$$

Since W = [Wc, Wx], C_t can be written as: $C_t = \tanh\left(W_c C_{t-1} + W_x X_t\right)$.

The derivative of C_t has the form of

$$\frac{\partial c_t}{\partial c_{t-1}} = \tanh'\left(W_c C_{t-1} + W_x X_t\right)\cdot\frac{d}{dc_{t-1}}\left[W_c C_{t-1} + W_x X_t\right] = \tanh'\left(W_c C_{t-1} + W_x X_t\right)\cdot W_c$$

The last expression tends to vanish when k is large. Since the derivative of the activation function tanh which is smaller or equal to 1. As a result, we have:

$$\prod_{t=2}^{k}\tanh'\left(W_c C_{t-1} + W_x X_t\right)\cdot W_c \to 0, \text{ so } \frac{\partial E_k}{\partial W}\to 0$$

2. Why LSTMs solve gradient vanishing problem

As before we have

$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial H_k}\frac{\partial H_k}{\partial C_k}\frac{\partial C_k}{\partial C_{k-1}}\cdots\frac{\partial C_2}{\partial C_1}\frac{\partial C_1}{\partial W} =$$
$$\frac{\partial E_k}{\partial H_k}\frac{\partial H_k}{\partial C_k}\left(\prod_{t=2}^{k}\frac{\partial C_t}{\partial C_{t-1}}\right)\frac{\partial C_1}{\partial W}$$

In LSTMs, the cell vector is:
$C_t = C_{t-1} \otimes \sigma\left(W_f \cdot \left[H_{t-1}, X_t\right]\right) \oplus \tanh\left(W_c \cdot \left[H_{t-1}, X_t\right]\right) \otimes \sigma\left(W_i \cdot \left[H_{t-1}, X_t\right]\right)$.
So, the derivative of C_t has the form of
$\frac{\partial c_t}{\partial c_{t-1}} = \sigma\left(W_f \cdot \left[H_{t-1}, X_t\right]\right) + \frac{d}{dc_{t-1}}\left(\tanh\left(W_c \cdot \left[H_{t-1}, X_t\right]\right) \otimes \sigma\left(W_i \cdot \left[H_{t-1}, X_t\right]\right)\right)$

For simplicity, the computation of $\frac{d}{dc_{t-1}}\left(\tanh\left(W_c \cdot \left[H_{t-1}, X_t\right]\right) \otimes \sigma\left(W_i \cdot \left[H_{t-1}, X_t\right]\right)\right)$ is left out. Since it is enough that the activations of the forget gate are greater than 0.
So, $\frac{\partial c_t}{\partial C_{t-1}} \approx \sigma\left(W_f \cdot \left[H_{t-1}, X_t\right]\right)$. And $\frac{\partial E_k}{\partial W} \approx \frac{\partial E_k}{\partial H_k}\frac{\partial H_k}{\partial C_k}\left(\Pi_{t=2}^{k}\sigma\left(W_f \cdot \left[H_{t-1}, X_t\right]\right)\right)\frac{\partial c_1}{\partial W}$.
The above equation indicates that the gradient behaves similarly to the forget gate, which means if the forget gate decides that a certain piece of information should be remembered, it will be open and have values closer to 1 to allow for information flow. Therefore, for simplicity, the forget gate's action can be viewed as: $\sigma\left(W_f \cdot \left[H_{t-1}, X_t\right]\right) \approx \vec{1}$. Then, we have $\frac{\partial C_t}{\partial C_{t-1}} \nrightarrow 0$.
Finally $\frac{\partial E_k}{\partial W} \approx \frac{\partial E_k}{\partial H_k}\frac{\partial H_k}{\partial C_k}\left(\prod_{t=2}^{k}\sigma\left(W_f \cdot \left[H_{t-1}, X_t\right]\right)\right)\frac{\partial C_1}{\partial W} \nrightarrow 0$