CS 59000SA **Fall 2018**

Assignment 5

Instructor. Ninghui Li

Due: November 19, 2018. [1]

5.1 Introduction

5.1.1 RNN

A recurrent neural network (RNN), is a neural network that contains backward edges so that there are loops in the network. RNN is particular good at handling sequence data.

Figure 5.1 shows a simple RNN with loop where input and output are synchronized. At time t, the network A takes as input x_t and produces output h_t. The loop allows information to be passed from one step of the network to the next. To be more specific, when A takes as input x_{t+1} and computes h_{t+1}. In addition to the input x_{t+1}, the output h_{t+1} also depends upon the state the network is in before taking x_{t+1} as input.

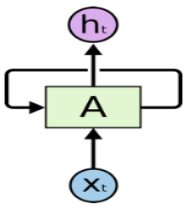

Figure 5.1: Simple RNN

Question 1. Assume we have a sequence of t inputs, namely x_1, x_2, $\cdots$, x_t. Construct neural networks based on the following requirements.

- Draw a feed-forward neural network using A that will perform equivalent computation on the given input as the above RNN. Clearly mark input, output and all connections. **Justify your answer.**
- To use a feed-forward neural network to simulate a RNN, what weight constraints are needed? What's the space complexity for the model of RNN? What about it's equivalent feed-forward network version?

[1] This handout uses lecture notes from CS224d: Deep Learning for Natural Language Processing from Stanford University.`http://web.stanford.edu/class/cs224d/lecture_notes/notes4.pdf`

5.1.2 IDS

An Intrusion Detection System, or IDS, is a device or software application that monitors a network or systems for malicious activity or policy violations. NIDS (Network intrusion detection systems) is a common IDS. NIDS monitors traffic to and from all devices on the network. It performs an analysis of passing traffic on the entire subnet, and matches the traffic that is passed on the subnets to the library of known attacks. Once an attack is identified, or abnormal behavior is sensed, the alert can be sent to the administrator.

There are two popular methodology to detecting malicious and abnormal packages mentioned above. Signature-based NIDS, as the name suggests, detects attacks by looking for specific patterns, such as byte sequences in network traffic, or known malicious instruction sequences used by malware. Anomaly-based NIDS were primarily introduced to detect unknown attacks. The basic approach is to use machine learning to create a model of trustworthy activity, and then compare new behavior against this model.

Naive NIDSs typically make a bad assumption: packages are independent. Such NIDS suffers from time-series based attack. For example, it can be very hard to setup malware with a single package due to the good signature database of NIDS; One way to circumvent it is to send malware piece by piece first. After that, a "biennial program" will be sent to assemble the aforementioned pieces.

You may already realized that using RNN can solve the above problem. Indeed, in this assignment, you are going to implement a simple NIDS using RNN. But before we start coding, let's analyzing an interesting fact of RNN.

5.2 Numerical Instability of Large RNN

One straight forward approach to implement such NIDS is to build a RNN that remember all history. There can be two issues for this. An obvious one is memory consumption. We may run out of memory to store all the history. To relax this constrain, one may choose to remember history up to length H and set H to be a sufficient large number. For example, 2^{20}.

Unfortunately, this is not feasible due to the second drawback of large RNN. Recurrent neural networks propagate weight vectors from one time step to the next. Any large graident-based RNN suffers from **Vanishing Gradient** or **Gradient Explosion** problem.

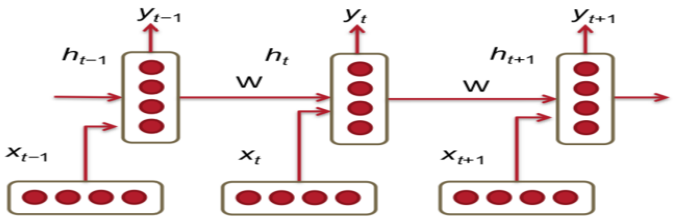

Figure 5.2: three-time-step RNN

Consider the three-time-step word prediction RNN in figure 5.2. It takes words as input and tries to predict the next word. In this RNN, each rectangle box is a hidden layer at a time step. Each layer holds a number of neurons, each of which performing a linear matrix operation on its inputs followed by a non-linear operation like tanh(). At each time-step, the output of the previous step along with the next word vector in the document, x_t, are inputs to the hidden layer to produce a prediction output $\hat{y}$ and output features h_t. Formally speaking:

$$h_t = \text{sigmod}(Wh_{t-1} + W^{|h|\times|x|}x^t)$$
$$\hat{y} = \text{softmax}(W^{(s)}h_t)$$

Where,

- $x_1, x_2, \cdots, x_t, \cdots x_T$ is the word vectors corresponding to a corpus with T wrods
- x_t input word vector at time t.
- $W^{|h|\times|x|}$: weights matrix used to condition the input word vector, x_t. $|h|$ is the dimension of the of hidden layer and $|x|$ is the dimension of the word vector.
- $W^{|h|\times|h|}$: weights matrix used to condition the output of the previous time-step, h_{t1}
- $\hat{y}$: the output probability distribution over the vocabulary at each time-step t. Essentially, y^t is the next predicted word given the document context score so far.

To compute the error, $\frac{\delta E}{\delta W}$, we need sum the error at each time step t from 1 to T.

$$\frac{\delta E}{\delta W} = \sum_{t=1}^{T} \frac{\delta E_t}{\delta W} \tag{5.1}$$
$$= \sum_{t=1}^{T}\sum_{k=1}^{t} \frac{\delta E_t}{\delta y_t}\frac{\delta y_t}{\delta h_t}\frac{\delta h_t}{\delta h_k}\frac{\delta h_k}{\delta W} \tag{5.2}$$
$$= \sum_{t=1}^{T}\sum_{k=1}^{t} \frac{\delta E_t}{\delta y_t}\frac{\delta y_t}{\delta h_t}\frac{\delta h_k}{\delta W}\prod_{i=k+1}^{t}\frac{\delta h_i}{\delta h_{i-1}} \tag{5.3}$$
$$= \sum_{t=1}^{T}\sum_{k=1}^{t} \frac{\delta E_t}{\delta y_t}\frac{\delta y_t}{\delta h_t}\frac{\delta h_k}{\delta W}\prod_{i=k+1}^{t}\frac{\delta h_i}{\delta h_{i-1}} \tag{5.4}$$

equation 5.2 is obtained from chain-rule expansion from 5.1, where $\frac{\delta h_t}{\delta h_k}$ refers to the partial derivative of h_t with respect to all previous k time-steps. equation 5.3 is obtained by applying chain-rule to $\frac{\delta h_t}{\delta h_k}$ again.

If you studied linear algebra (Don't worry if you don't know), $\frac{\delta h_i}{\delta h_{i-1}}$ is a actually a Jacobian matrix for h. To be more specific

$$\frac{\delta h_i}{\delta h_{i-1}} = \begin{bmatrix} \frac{\delta h_{i,1}}{\delta h_{i-1,1}} & \frac{\delta h_{i,1}}{\delta h_{i-1,2}} & \cdot & \cdot & \cdot & \cdot & \frac{\delta h_{i,1}}{\delta h_{i-1,|n|}} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \frac{\delta h_{i,|n|}}{\delta h_{i-1,1}} & \frac{\delta h_{i,|n|}}{\delta h_{i-1,2}} & \cdot & \cdot & \cdot & \cdot & \frac{\delta h_{i,|n|}}{\delta h_{i-1,|n|}} \end{bmatrix}$$

Question 2. In this question, you are going to learn how gradient vanish problem can occur.

- The output range of the sigmoid function is _____
- By linear algebra, the norm of $\frac{\delta h_i}{\delta h_{i-1}}$,$\|\frac{\delta h_i}{\delta h_{i-1}}\|$, is actually$\|W^T\|\|diag(sigmoid(h_{i-1}))\|$. Let α_a be the upperbound of $\|W^T\|$ and α_b be the upperbound $\|diag(sigmoid(h_{i-1}))\|$. Prove why Equation 5.4 can suffer from vanishing gradient or gradient explosion problem. (**Hint: There is a product term in Equation (5.4)**)

5.3 LSTM-RNN

5.3.1 Introduction

If you had solved Problem 2 successfully, you should realize that naive RNN is limited by the precision of the machine. Therefore, naive RNN has to be small. i.e. it can't store a long period of history.

To mitigate this issue, Sepp Hochreiter and Jrgen Schmidhuber propused LSTM(Long Short Term Memory) units. LSTM is a special kind of RNN, capable of learning long-term dependencies. It's explicitly designed to avoid the vanishing gradient problem. All RNN have the form of a chain of repeating of nerual network. LSTMs also have such chain structure, but the repeating module has a different structure. Instead of having a signle neural layer, there are several layers interacting in a very special way. They are five main parts in LSTM unit, namely:

- **New memory generation**: A new memory $\tilde{h_t}$ is generated by consolidating new input word x_t with the past hiddent state h_{t-1}. and output a new memory cell $\tilde{c_t}$
- **Input Gate**: This gate checks whether the input word x_t is worth preserving based on the w_t and past hidden state. It produces i_t as an indicator for this information.
- **Forget Gate**: This gate makes an assessment on whether the past memory cell is useful for the computation for the current memory cell. It looks at word w_t and past hidden state to output f_t to indicate whether the past state has to be erased.
- **Final Memory generation**: This stage first takes the advice from forget gate and accordingly chooses to forget past memory c_{t-1}. It also takes advice from input gate and new memory generation gate as well. It sums these two results to produce the final memory c_t.
- **Output Gate**: This gate separate the final memory from the hidden state. The final memory c_t contains a lot of information is not necessarily required to be saved in the hidden state. Intuitively speaking, it measure how much c_t should be exposed.

Questionm 3. Sketch the structure of LSTM based on the above description and following equation, clearly mark what's the input and output for each gate.

- Input Gate: $i_t = \mathrm{I}(x_t,h_{t-1})$
- Forget Gate: $f_t = \mathrm{F}(x_t,h_{t-1})$
- Output Gate: $o_t = \mathrm{O}(x_t,h_{t-1})$
- New Memory Generation Gate: $\tilde{c_t} = \mathrm{NM}(x_t,h_{t-1})$
- Final Memory Generation Gate: $c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c_t}$
- $h_t = \tanh(c_t) \circ o_t$

Question 4. Which gate(s) of LSTM help prevent gradient vanishing problem? Justify your answer.

5.3.2 LSTM in Tensorflow:

At this step, you should understand the motivation of using LSTM now. Again, as data scientists, our goal is not to implementing the above module. Instead, knowing LSTM, we want to use existing library to build a neural network quickly.

Luckily, Tensorflow does support such type of RNN. The details of it can be found at :

`https://www.tensorflow.org/tutorials/sequences/recurrent`

To be more specific, you should read the description of the following functions: LSTMCell and LSTMBlockCell.

Question 5 What's the difference between a memory cell and a memory block?

5.4 LSTM-RNN based NIDS

In this assignmennt, you are going to impplement an NIDS system based on LSTM-RNN. It detects network intrusions and protects a computer network from unauthorized users, including perhaps insiders. The intrusion detector learning task is to build a predictive model (i.e. a classifier) capable of distinguishing between "bad" connections, called intrusions or attacks, and "good" normal connections.

5.4.1 Dataset

You may download the **data** from `http://kdd.ics.uci.edu/databases/kddcup99`

In this task, you should use *kddcup.data_10_percent.gz* as the training dataset and the rest for testing and validation.

The description of the dataset is available at `http://kdd.ics.uci.edu/databases/kddcup99/task.htmlhttp://kdd.ics.uci.edu/databases/kddcup99/task.html`

5.4.2 Experiment

You are going to implement the following four different LSTM-RNN and measure their performance based on accuracy, recall, and F-Score:

- Two memory blocks with two cells each
- four memory blocks with two cells each
- four memory blocks with four cells each
- eight memory blocks with four cells each.

5.4.3 Fun with Peephole

One popular LSTM variant, introduced by Gers & Schmidhuber (2000), is adding peephole connections. This means that we let the gate layers look at the cell state. Enable Peephole feature in all the neural network mentioned in sectionn 5.4.2. Measure the performance again.

5.5 Grading

You should submit python files and a report in **PDF** including answers for all questions and experiment reports. In addition, you should submit a README.txt telling us how to run your python code. Please zip your submission.

- Question 1: 10 pts
- Question 2: 10 pts
- Question 3: 10 pts
- Question 4: 10 pts
- Question 5: 10 pts
- Experiment: 50 pts
- Coding standard: up to 10 pts deduction