University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 22: November 8

Lecturer: Alexander Schwing Scribe: Ningkai Wu

1 Introduction

The main focus of lecture 22 is autoregressive methods such as Recurrent Neural Nets (RNNs), Long short term memory (LSTMs), Gated recurrent unit (GRUs) and Graph convolutional nets (GCNs). Previously, we have discussed some discriminative and generative models. However, they lack the flexibility regarding inputs and outputs. On the contrary, autoregressive methods, can have sequences of inputs and sequences of outputs as shown in the figure below: the length of sequences may vary.

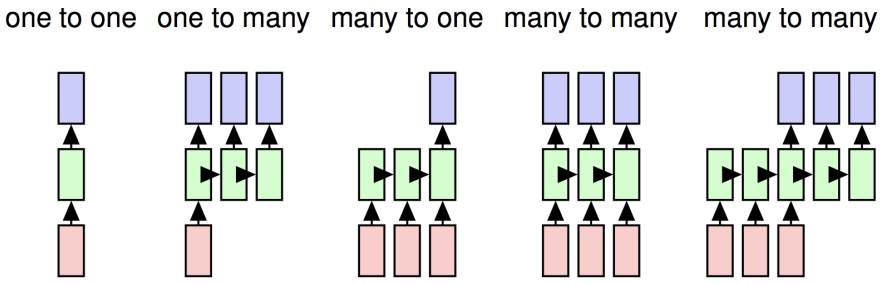

Figure 1: Autoregressive Methods

2 Recurrent Neural Nets

2.1 Overview

In traditional neural networks, inputs and outputs are assumed to be independent of each other. In this way, sequence and time information will not be preserved. For example, if you want to predict the next word in a sentence, it would be useful to know the words before it. RNNs are called recurrent since the networks will perform the same operation for every element in the sequence [1]. They can also be thought of containing information calculated so far. Therefore, RNNs have a good performance in tasks such as Natural Language Processing, Speech recognition, Image processing and Video processing.

2.2 Recurrent structure

Mathematical description and Recurrent structure in general:

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}, \mathbf{w})$$

$$y^{(t)} = g(h^{(t)})$$

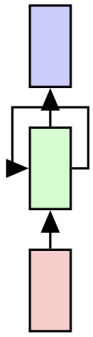

Figure 2: Recurrent Neural Nets

The structure above can also be unfolded to help us better understand the structure. For example, if the sequence consists of 3 words, the network can be unfolded into a 3-layer network, one layer for each word [1]. As we can see, the inputs will depend on previous output. We need to note that the parameters are shared across all steps, which greatly reduces the number of parameters we need to learn. The green blocks are hidden states which capture the information calculated during previous steps and act as "memory". The purple blocks are outputs and they only depend on the memory at time t.

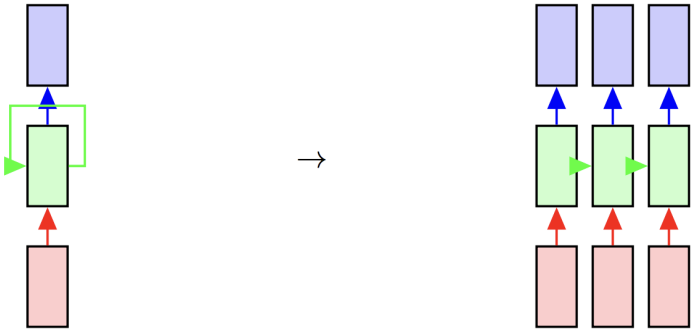

Figure 3: unfolded/unrolled network performs identical operations

2.3 Original recurrent nets (Elman network)

In the above sections, we've talked about the general equations of RNNs. Now, let's talk about the original recurrent nets in detail. An Elman network is a three-layer network. They are arranged horizontally as x, y and z with a set of "context units" u. The hidden layer is connected to these context units with a fixed weight one. In every step, one input is fed forward and the learning rule is applied [4]. Below is the Elman network structure and its mathematical description.

$$h^{(t)} = \sigma_h(W_{hx}x^{(t)} + W_{hh}h^{(t-1)} + w_{hb})$$
$$y^{(t)} = \sigma_y(W_{yh}h^{(t)} + w_{yb})$$

Elman network:

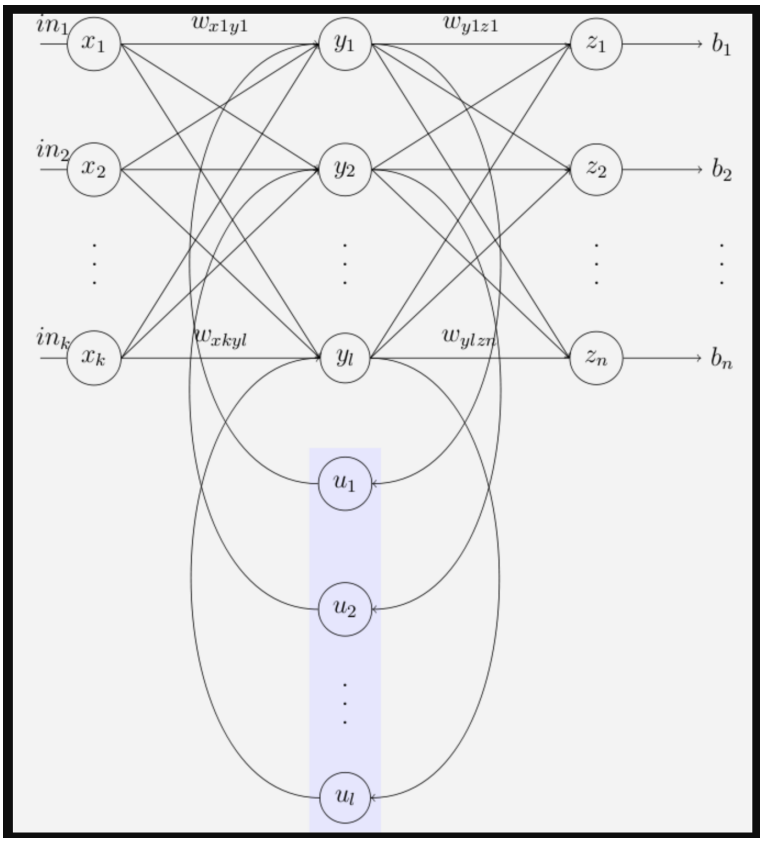

Figure 4: Elman network

- x_t: input vector
- h_t: hidden layer vector
- y_t: output vector
- σ_h and σ_y: activation functions

However, there are two problems with classical recurrent neural nets such as Elman network. The first one is vanishing gradient, which means in some cases, the gradient will become small, thus preventing the weights from changing values. The second one is long term dependency as the network becomes large.

3 Long short term memory (LSTM)

3.1 Overview

LSTM networks are a special kind of RNN. They are much similar to original RNNs mentioned above. However, they use a different function to compute the hidden state. They were refined and popularized since the first introduction by Hochreiter and Schmidhuber (1997) [5]. Specifically, they have the following features:

- Particular functional relation

- Shown to better capture long-term dependencies
- Shown to address the vanishing gradient problem

3.2 LSTM structure

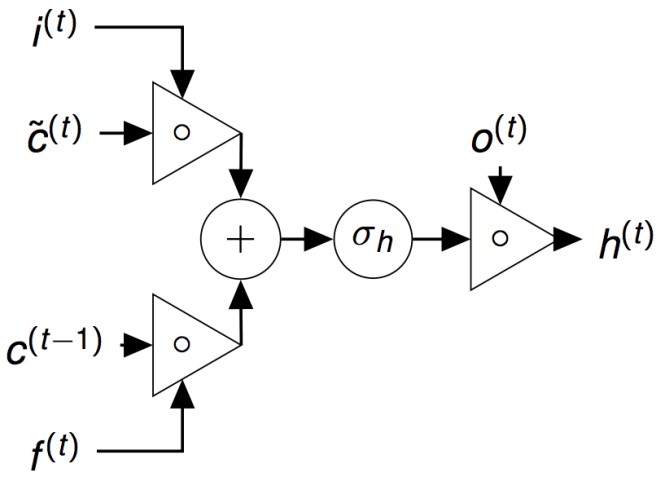

Figure 5: LSTM Intuition

Mathematical description ($\circ$ denotes Hadamard product):
Input gate:

$$i^{(t)} = \sigma_i(W_{ix}x^{(t)} + W_{ih}h^{(t-1)} + w_{bi})$$

Forget gate:

$$f^{(t)} = \sigma_f(W_{fx}x^{(t)} + W_{fh}h^{(t-1)} + w_{bf})$$

Output/Exposure gate:

$$o^{(t)} = \sigma_o(W_{ox}x^{(t)} + W_{oh}h^{(t-1)} + w_{bo})$$

New memory cell:

$$\tilde{c}^{(t)} = \sigma_c(W_{cx}x^{(t)} + W_{ch}h^{(t-1)} + w_{bc})$$

Final memory cell:

$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$$

$$h^{(t)} = o^{(t)} \circ \sigma_h(c^{(t)})$$

The first step in LSTM is to decide whether we should throw away information from cell state. This decision is made by forget gate which looks at h_{t-1} and x_t and outputs a number between 0 and 1 for each number in the cell state. A 1 means "completely keep" while a 0 means "completely throw away". Next, the input gate which tells whether $x^{(t)}$ matters and a sigmoid layer which creates a new set of candidate values $\tilde{c}^{(t)}$ will be combined to create an update to the old cell state [6]. Finally, output gate decides how much memory should be exposed.

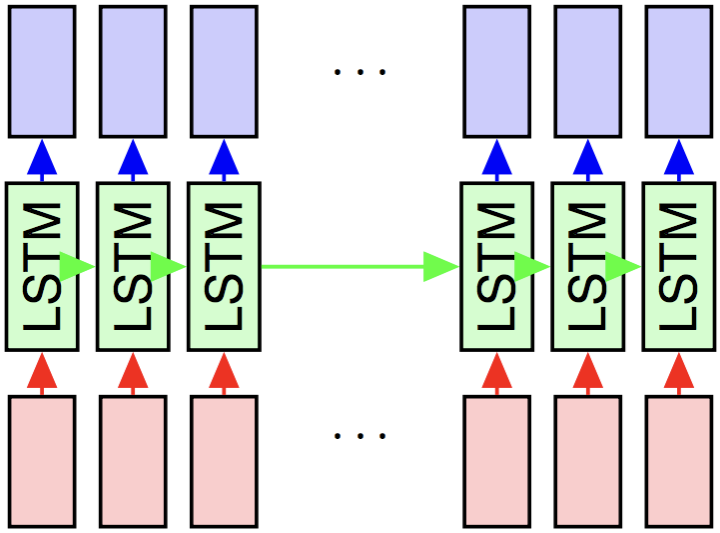

Figure 6: LSTM interpreted as a block in a neural net

4 Gated recurrent unit (GRU)

4.1 Overview

Gated recurrent units, again, is a special type of recurrent neural networks. They were introduced in 2014 by Kyunghyun Cho et al [2]. Their performance on polyphonic music modeling and speech signal modeling was found similar to the LSTM models mentioned above. In addition, they show better performance on smaller datasets [3]. However, since GRUs have no output gate, they have fewer parameters compared to LSTM.

4.2 GRU structure

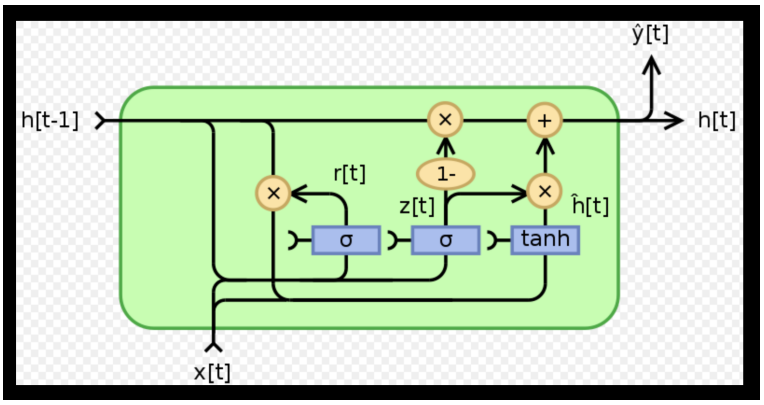

Figure 7: Gated Recurrent Unit, fully gated version

Mathematical description (∘ denotes Hadamard product):
Update gate:

$$z^{(t)} = \sigma_z(W_{zx}x^{(t)} + W_{zh}h^{(t-1)} + w_{bz})$$

Reset gate:

$$r^{(t)} = \sigma_r(W_{rx}x^{(t)} + W_{rh}h^{(t-1)} + w_{br})$$

New memory cell:

$$\tilde{h}^{(t)} = \sigma_h(W_{hx}x^{(t)} + W_{rwh}(r^{(t)} \circ h^{(t-1)} + w_{bh})$$

Hidden cell:

$$h^{(t)} = (1 - z^{(t)} \circ \tilde{h}^{(t)} + z^{(t)} \circ h^{(t-1)})$$

Fully gated unit:

- x_t: input vector
- h_t: output vector
- z_t: update gate vector
- r_t: reset gate vector

r_t decides whether to include $h^{(t-1)}$ in new memory and z_t decides how much $h^{(t-1)}$ in next state. There are several variations for the full gated unit, depending on the combinations of the previous hidden state and the bias [3].

- Type1: Gate depends only on the previous hidden state and the bias
- Type2: Gate depends only on the previous hidden state
- Type3: Gate depends only on the bias

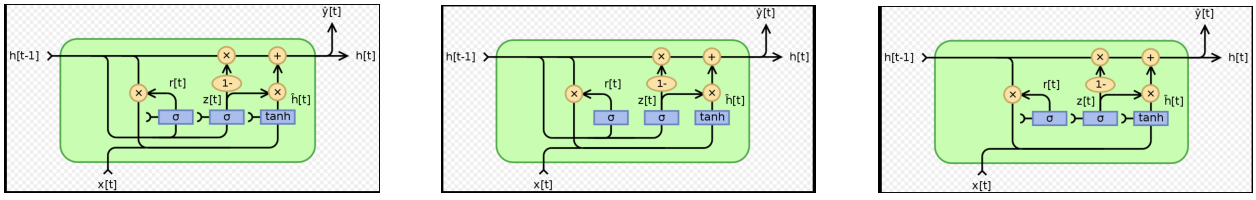

Figure 8: Three types of GRUs

5 Graph Convolutional Neural Nets (GCNs)

5.1 Overview

Nowadays, many important real-world datasets are in the form of graphs or networks such as knowledge graphs and social networks. Graph Convolutional Neural Nets, unlike RNNs or CNNs, are able to work well on these arbitrarily structured graphs. In general, the goal for GCNs is to learn a function of features on a graph that takes in a feature description x_i and a representative description of the graph structure in matrix form. The result is either node-level or graph-level outputs Z.

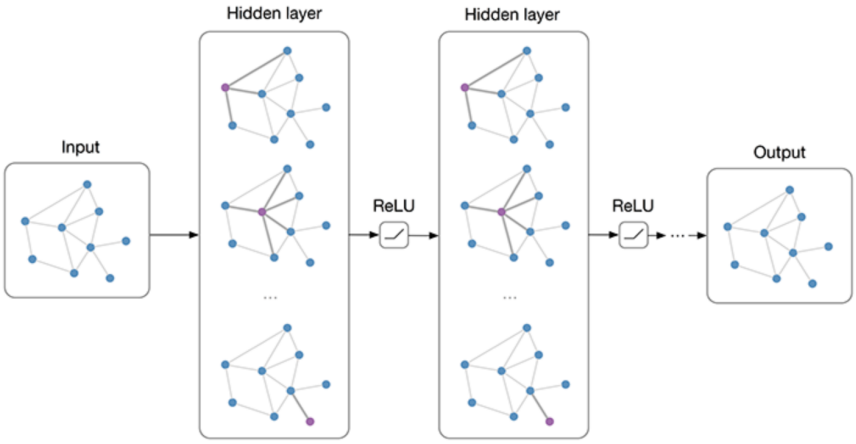

Figure 9: Multi-layer Graph Convolutional Network (GCN) with first-order filters.

5.2 GCN structure

Mathematical description:

$$H^{(I+1)} = f(H^{(I)}, A)$$

- Representation: $H^{(I)}; H^{(0)} = X$
- Graph adjacency matrix: A
- Nonlinearity: f

Example:

$$H^{(I+1)} = \sigma(AH^{(I)}W^{(I)})$$

There are also more complex incarnations.

6 Training of RNN - Backpropagation Through Time

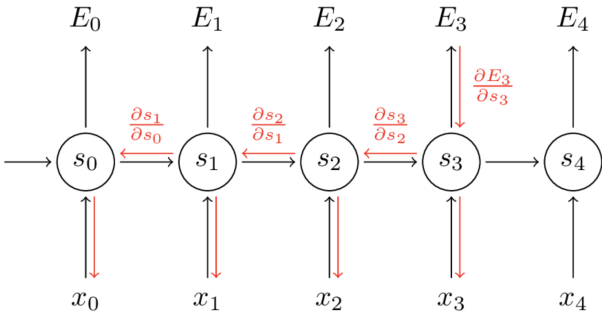

Figure 10: BPTT

As usual, we define our loss function such as cross entropy loss and we treat the entire sequence as one training sample. Therefore, the total error would be sumed over all time steps. In order to learn good parameters, we use gradient descent such as Stochastic Gradient Descent. We still

use backpropagation algorithm to train a RNN as we train a traditional neural network. However, because the parameters are shared by all time steps, the gradient at each output depends on both the calculations of current time step and those of the previous time steps. For example, to calculate the gradient at t = 5 we will need to backpropagate 4 steps and sum the gradients. This is called Backpropagation Through Time or BPTT [4]. As shown in the above figure, we have to sum up the gradients for W at each time step and we have to apply chain rule repeatedly. This is why RNNs are so hard to train and as the sequence becomes longer and longer, the vanishing gradient problem may arise. This is why LSTM and GRU are so popular because these models can deal with the problem.

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y_3}} \frac{\partial \hat{y_3}}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$

7 Generative models overview

- Variational Auto-encoders (VAEs):
  pro: probabilistic graphical model interpretation
  con: slightly blurry examples
- Generative Adversarial Nets (GANs):
  pro: generate sharp images
  con: difficult to optimize (unstable)
- Autoregressive models (RNNs):
  pro: stable training and good likelihoods
  con: inefficient sampling and no low-dimensional codes

8 Summary

In lecture 22, we learned RNNs and its important variants such as LSTM and GRU. We realize their applications in different areas such as NLP. Also, we understand the difference between RNNs and other generative models, e.g. sequence information preserved. At the end, we also had a introduction for GCNs which are used for structured graphs.

References

[1] D. BRITZ. Recurrent neural networks tutorial: Introduction to rnn, 2015.

[2] K. Chor. Empirical evaluation of gated recurrent neural networks on sequence modeling. 2014.

[3] W. contributors. Gated recurrent unit, 2018.

[4] W. contributors. Recurrent neural network, 2018.

[5] S. Hochreiter and J. Schmidhuber. Long short-term memory. 1997.

[6] C. Olah. Understanding lstm networks, 2015.