University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 22: November 8

Lecturer: Alexander Schwing Scribe: Shaowen Wang

1 Overview

This lecture mainly introduces autoregressive methods (RNNS/LSTMS/GRUs) and graph convolution nets (GCNs). Different from previous models, here we pay attention to models for processing sequential data. We will go from multi-layer networks to recurrent networks in this lecture.

The goals of this lecture includes:

- Getting to know Recurrent Neural Nets(RNNs)
- Getting to know Long short term memory(LSTM)
- Getting to know Gated recurrent unit(GRU)
- Getting to know Graph convolutional nets(GCNs)
- Seeing how to apply them

2 Recap

Our models so far:

- Discriminative
$$p(y|x)$$
- Generative
$$p(x)$$

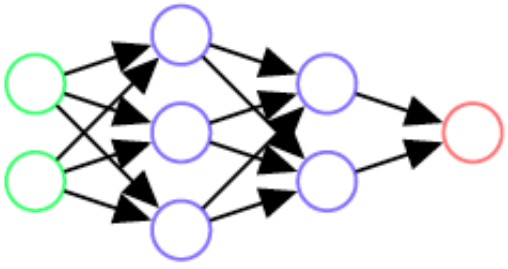

Figure 1: Previous model

When using such kind of model, we are missing time. That's to say, we may want to deal with more flexible data(sequential inputs and outputs). And the length of sequences may vary. For example, we may have:

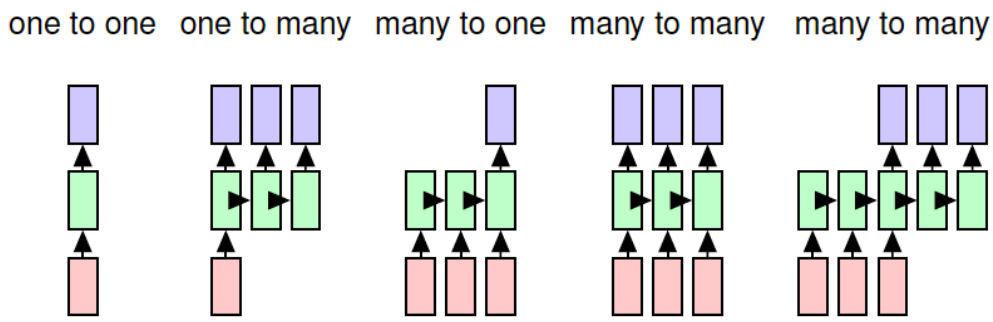

Figure 2: Previous model

one to many: get original picture from occluded picture

many to one: identify original picture from texture features

many to many (4th): autocomplete

many to many (5th): machine translation

3 Recurrent Neural Nets(RNNs)

Recurrent Neural Nets are neural networks for processing sequential data. We know that convolutional networks can scale to images with large width and length. Similarly, recurrent neural networks can scale to longer sequence and even sequence of variable length.

Typical applications of RNNs:

- Natural language processing
- Speech recongnition
- Image Processing
- Video Processing

Different from multi-layer networks, recurrent neworks utilize the idea of parameter sharing (sharing parameters across different parts of a model). Based on this idea (input depends on previous output), Recurrent neural networks can be built in many different ways. Much as almost any function can be considered a feedforward neural network, essentially any function involving recurrence can be considered a RNN. [2]

Here is the general structure for recurrence(unfolded):

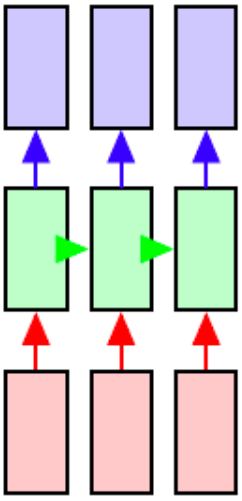

Figure 3: general structure of RNNs

We could use the following equation to define many recurrent neural networks:

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}, w)$$
$$y^{(t)} = g(h^{(t)})$$

We use the variable $h^{(t)}$ to represent the state (green block in the graph)and use $x^{(t)}$ to represent sequential input (red block in the graph).

And we should also know that f function and g function are independent of time. They can be any differentiable functions including original recurrent nets, LSTM nets, GRU nets and etc.

For original recurrent nets (Elman network), generally, we have:

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}, w)$$
$$y^{(t)} = g(h^{(t)})$$

Specifically, we have:

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

Here, σ_h and σ_y are activation functions: tanh, sigmod, usually we don't use relu here.

However, there are problems with classical recurrent neural nets. Gradients propagated over many stages tend to either vanish or explode(rarely). Learning long=term dependencies is difficult because of the exponentially smaller weights given to long-term interactions. [2]

4 Long short term memory(LSTM)

The most effective sequence models utilized in practical are called gated RNNs including the long short term memory(LSTM) and networkd based on the gated recurrent unit(GRU).

Long short term memory model utilizes self-loops to produce paths where the gradient could flow for long durations. Thus, long short term memory model shown to better capture long-term dependencies and address the vanishing gradient problem.

Here is the LSTM block diagram:

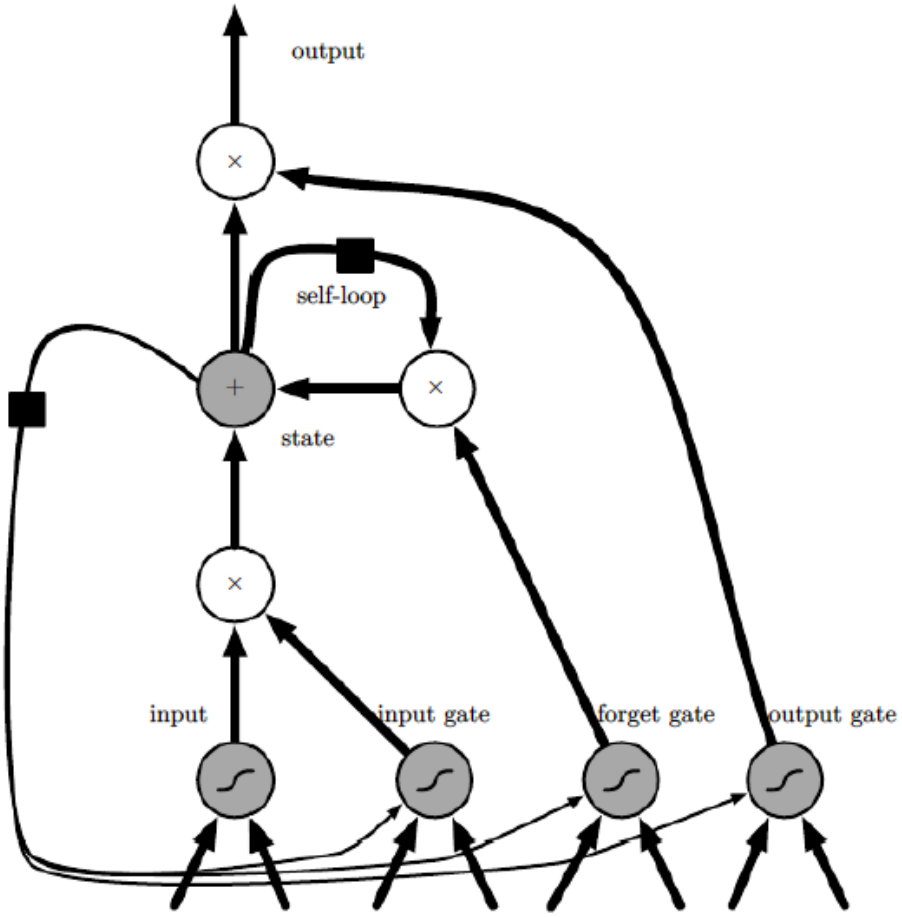

Figure 4: LSTM block diagram [2]

It's block diagram of LSTM recurrent network cell. The input is computed with a regular artificial neuron unit. And its value can be accumulated into the state after getting across the sigmoidal input gate. We can also see that the forget gate controls the weight of state unit's linear self-loop. The black square represents a delay of single time step. [2] The following is the equation for forget gate unit f_i^t (for time step t and cell i):

$$f_i{}^{(t)} = \sigma(b_i{}^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_i^f h_j^{(t-1)})$$

where $x^{(t)}$ is the current input and $h^{(t)}$ is the current hidden layer. And b_f, U_f, W_f are respectively biases, input weights and recurrent weights for the forget gates. Then the update equation of state is:

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)})$$

The external input gate unit $g_i^{(t)}$ is computed as following:

$$g_i^{(t)} = \sigma(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)})$$

The output $h_i^{(t)}$ and the output gate $q_i^{(t)}$:

$$h_i^{(t)} = tanh(s_i^{(t)})q_i^{(t)}$$
$$q_i^{(t)} = \sigma(b_i^o + \sum_j U_i^o x_j^{(t)} + \sum_j W_j^o h_j^{(t-1)})$$

Finally, long short term memory(LSTM) can be interpreted as a block in a neural net:

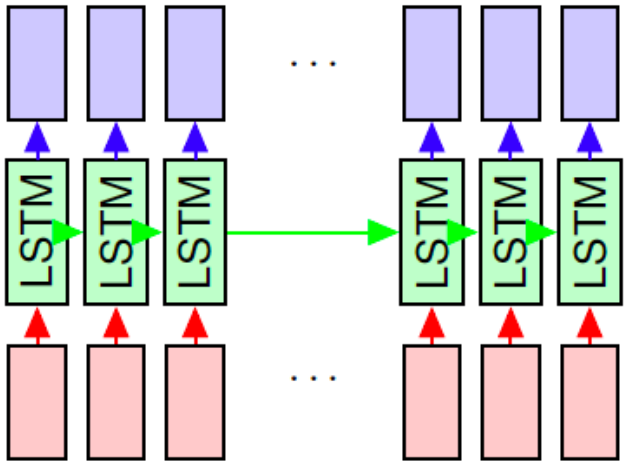

Figure 5: component

5 Gated recurrent unit(GRU)

The units of gated RNNs are known as gated recurrent units or GRUs. The main difference with the LSTM is that a single gating unit simultaneously controls the forgetting factor and the decision to update the state unit. [2]

Here are the equations:

$$h_i^{(t)} = u_i^{(t-1)} h_i^{(t-1)} + (1 - u_i^{(t-1)} \sigma(b_i + \sum_j U_{i,j} x_j^{t-1}) + \sum_j W_{i,j} r_j^{(t-1)} h_j^{(t-1)}))$$

where u represents update gate, r represents reset gate, here are their equations:

$$u_i^{(t)} = \sigma(b_i^u + \sum_j U_{iu} x_j^{(t)} + \sum_j W_{i,j}^u h_j^{(t)})$$
$$r_i^{(t)} = \sigma(b_i^r + \sum_j U_{ir} x_j^{(t)} + \sum_j W_{i,j}^r h_j^{(t)})$$

6 Gradient computation and Overview

6.1 Gradient computation

We could directly apply the generalized back-propagation algorithm to the unrolled computational graph which is rather straightforward. Gradients computed through back-propagation can be used with any other general-purpose gradient-based techniques to train the RNN. [2]

Such kind of back-propagation applied to the unrolled graph is called back-propagation through time or BPTT. To learn more about how BPTT algorithm behaves, we could refer chapter 10.2.2 of [2] for more details.

6.2 Pixel Recurrent Neural Networks [5]

Figure 6: Image completions sampled from a PixelRNN

Generative image modeling is a problem in unsupervised learning. In the paper [5], they advance two-dimensinaol RNNs and apply them to large-scale modeling of natural images. The pixel RNNs are composed of up to twelve, fast two-dimensional LSTM layers.

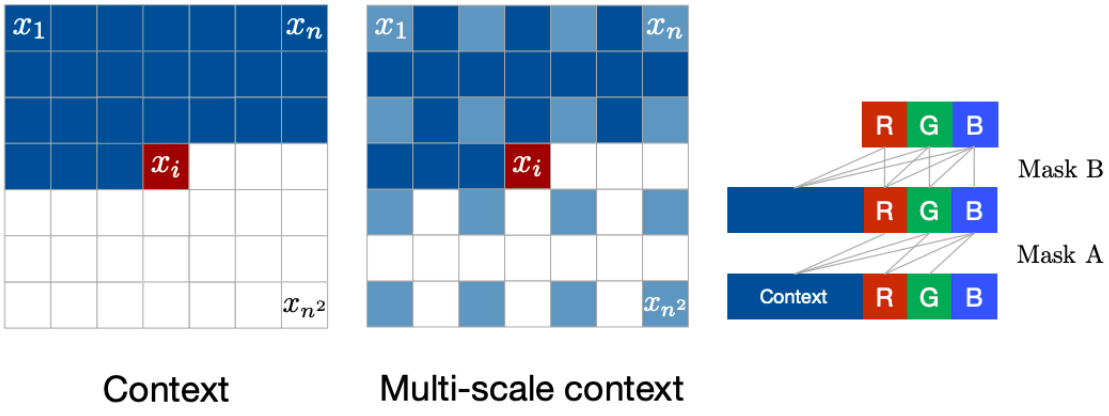

Figure 7: PixelRNN model

The aim is to estimate the distribution over original images which can be utilized to compute and generate new ones. The model will scan the picture row by row, pixel by pixel. For each pixel, it will predict the possible pixel values based on scanned context. And the parameters used in the model are shared across all pixel positions in the image. [5]

Figure 7 helps illustrate the process.

- The left one means to generate pixel x_i, we needs all the previously generated pixels left and above of x_i.
- The center means that to generate a pixel in the multi-scale case we can also condition on the pixels in light blue.
- The rights the is the diagram of the connectivity inside a masked convolution. [5]

For more details about how this Pixel Recurrent Neural Network generates a image pixel by pixel, you could refer [5]

6.3 Generative modesl 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)
- Autogressive models(RNNs):
  Pro: stable training good likelihoods
  Con: inefficient sampling no low-dimensional codes

7 Graph Convolutional Neural Nets(GCNs)

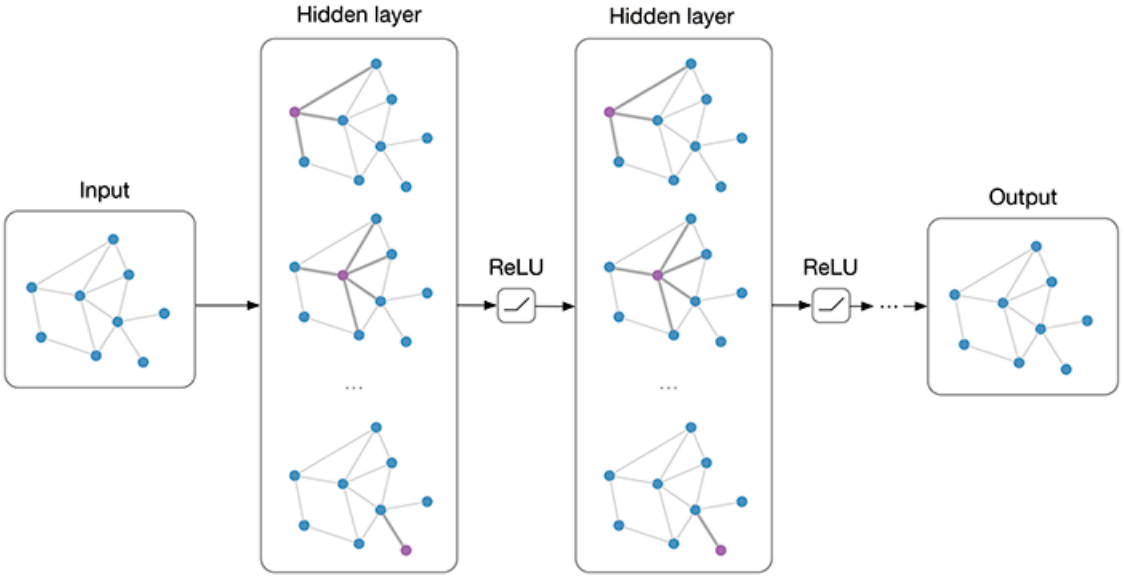

Figure 8: graph convolutional net

7.1 Definitions of GCNs

A lot of real-word problems' datasets are in the format of graphs of network, for example, social network, knowledge graphs etc. Generalizing well-established neural models like RNNs to work on arbitrarily structured graphs is challenging.

Most graph neural network models have a somewhat universal architecture in common. [4] Since filter parameters are shared over all locations in the graph [1], we refer to these models as Graph Convolutional Networks(GCNs).

The goal of those models is to learn a function of features of a graph which take as input:

- A feature vector x_i for every node i; Those feature vectors are summarized in a feature matrix X
- A representative description of the graph structure in matrix form; typically in the form of an adjacency matrix A. [3]

Every neural network layer can be described through a non-linear function:

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

A is the adjacency matrix of graph. $H^{(0)} = X$ and $H^{(L)} = Z$. L is the number of layers. The models then differ in how function $f(.,.)$ is selected.

7.2 Example of GCNs

Considering the following example:

$$f(H^{(l)}, A) = \sigma(AH^{(l)}W^{(l)})$$

$W^{(l)}$ is weight matrix for the l-th neural network layer. σ is activation function.

This simple model have two limitations:

- we sum up all the features of neighboring nodes except itself(unless we could add self-loops in the graph)
- A is not normalized the the scale of feature vectors would be completely changed when the multiplication with A happens.

Normalizing A such that all rows sum to one could help us solve this problem. In practice, we use symmetric normaliztioni, i.e. $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$. (D is the diagonal node degree matrix).

We then could get the propagation rule introduced in [4]:

$$f(H^{(l)}, A) = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$$

Here, $\hat{A} = A + I$ and I is the identity matrix, $\hat{D}$ is the diagonal node degree matrix of $\hat{A}$.

To have a closer look at how this model operates, you could refer [4].

References

[1] D. Duvenaud, D. Maclaurin, and R. P.Adams. Convolutional networks on graphs for learning molecular fingerprints. *NIPS*, 2015.

[2] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. 2016.

[3] T. KIPF. Graph convolutional network. `https://tkipf.github.io/graph-convolutional-networks/`, 2016.

[4] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017.

[5] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. *ICML*, 2016.