Recurrent Neural Networks

Sixtus Dakurah

Outline

Outline

What are RNNs?

- A family of neural networks for processing **sequential** data
- We often consider a sequence $x^{(1)}$, $x(^2)$, ..., $x^{(r)}$, for some index from 1 to r
- The index need not refer to time, but can be the position of any item in a sequence
- There is inherent dependence in this kind of sequence which makes it inefficient to be processed by any of the multilayer networks we've learned so far [1]
- How then do we model this dependence structure?

[1] MLP, FFNN, CNN

Motivating Example

- Consider these two sentences
  (1) I graduated from Umass in 2020
  (2) 2020 was my graduation year from Umass
- If we want wanted to extract the year from each of the sentences
  - Using a traditional FFNN will be quite inefficient. It will have to have separate weights for each input feature, which will literally be all the rules of the english language at each position in the sentence
  - In contrast, a RNN **shares weights** across time/index steps
  - The key idea in RNN is **parameter sharing** in a sequential setting
- Why not apply convolution?
  - In convolution, the parameter sharing is apparent in applying the same convolution kernel at each "convolution index"
  - It turns out this leads to shallow networks. How?

Outline

Computation Graphs

- A computation graph allows us to formalize a sequence of computations such as mapping inputs to parameters and loss
- How to construct a computation graph for a recurring sequence?
- Consider the following recurrent system:
  - $$s^{(t)} = f\left(s^{(t-1)}; \theta\right) \tag{1}$$
  - This equation is recurrent; the definition of the state at time t; $s^{(t)}$ depends on the same definition at time t-1
  - We can unfold this system by removing this dependence
  - That is, for a fixed number of time steps r, we can apply the definition $r-1$ times to unfold (1)
  - Example, consider $r = 3$

$$\begin{aligned} s^{(3)} &= f\left(s^{(2)}; \theta\right) \\ &= f\left(f\left(s^{(1)}; \theta\right); \theta\right) \end{aligned} \tag{2}$$

  - We now have an equation that does not involve recurrence.

Unfolding Computation Graphs

- We can now represent (2) as a directed acyclic graph

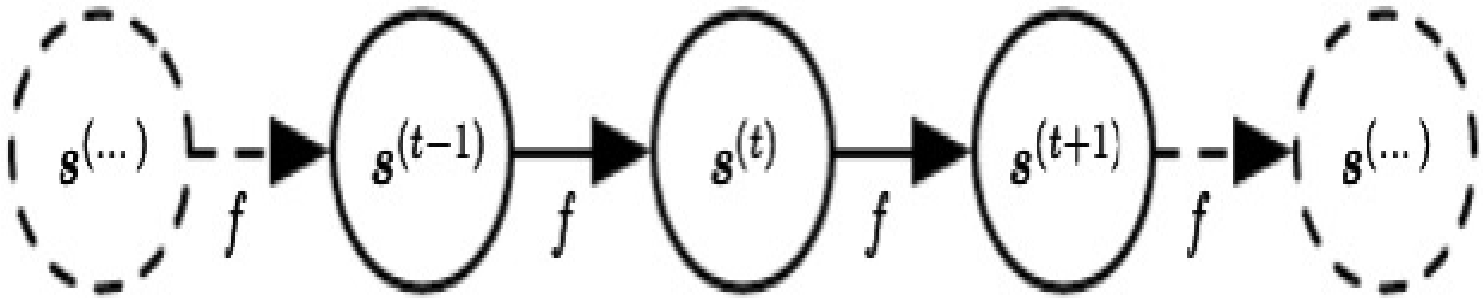

Figure: Unfolded computation graph

- Each node represents the state at time t
- The function f maps the state at time to to time t+1
- The same set of parameters θ as in (2) is used to parameterize f

Adding an external signal

- The state of a system at time t could also incorporate information from external signals
- We can rewrite the state equation as:

$$s^{(t)} = f\left(s^{(t-1)}, x^{(t)}; \theta\right) \tag{3}$$

- $x^{(t)}$ is an external signal upto and including the current time point
- For example if we characterize $x^{(t)}$ as noise, the state can be said to contain all the information about the whole past sequence

Dynamical Systems as RNN

- Recall that any function can be considered a FFNN
- Similarly, any function with recurrence can be considered a RNN
- To convert (3) into a RNN, we need a formulation that will make this equation a hidden unit
- A common formulation is to define h to be the state of the variable:

$$h^{(t)} = f\left(h^{(t-1)}, x^{(t)}; \theta\right) \tag{4}$$

- Here, h is for symbolism. It shows that the state is just the hidden units in the network

Dynamical Systems as RNN

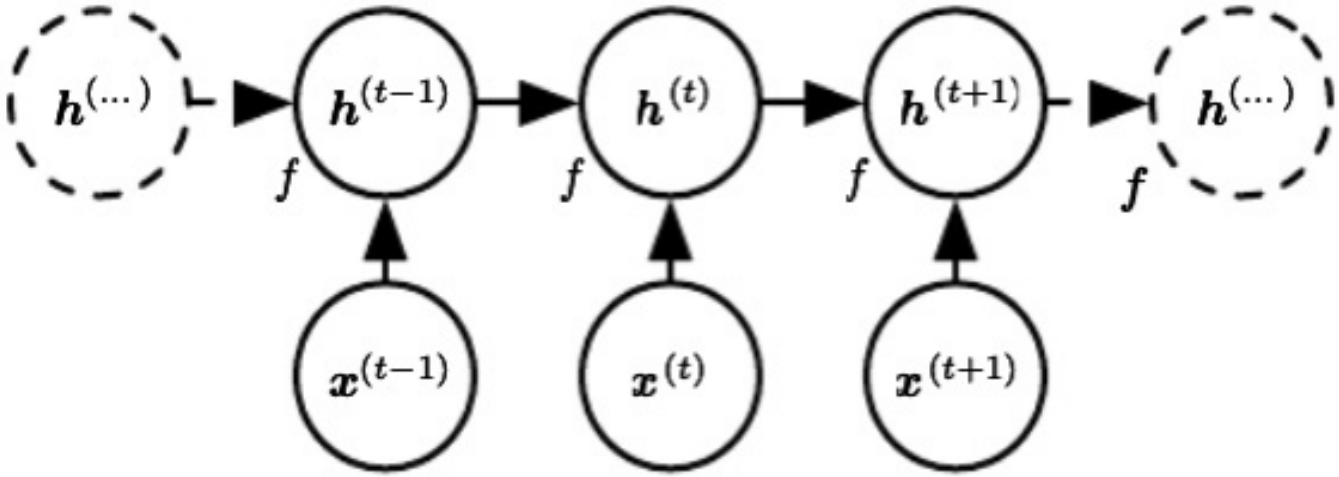

Figure: Unfolded Computation Graph with Signal

- This recurrent network simply takes an input x, incorporate it into the state $h(.)$, and pass it through time
- This is the basic form a RNN
- Typical RNNs will add more architectural features, like output layers that allows for predictions

Advantages of the Unfolded Computation Graph

- Restating the hidden state of the recurrent network:

$$\begin{aligned} h^{(t)} &= g^{(t)}\left(x^{(t)}, x^{(t-1)}, ..., x^{(1)}\right) \\ &= f\left(h^{(t-1)}, x^{(t)}; \theta\right) \end{aligned} \tag{5}$$

- The function $g^{(t)}$ takes the whole past sequence and produces the current state
- The unfolding allows us to factorize $g^{(t)}$ into repeated applications of the function $f(.)$
- This presents two advantages
  (1) Regardless of the sequence length, the learned model always has the same input size
  (2) It's possible to use the same transition function $f(.)$ with the same parameters at each time step

Outline

Recurrent Neural Network (A)

- Graph unrolling and parameter sharing ideas can now be used to build a wide variety of RNNs.
- We will consider three most prominent design patterns.
- I'll term the first design pattern as RNN (A)
- Structure of RNN (A)
  - Produces an output at each time step
  - Have recurrent connections between hidden units
- Parameterization of RNN (A)
  - Input to hidden connection is parameterized by a weight matrix U
  - hidden-to-hidden connection is parameterized by a weight matrix W
  - hidden-to-output connection is parameterized by a weight matrix V

Recurrent Neural Network (A)

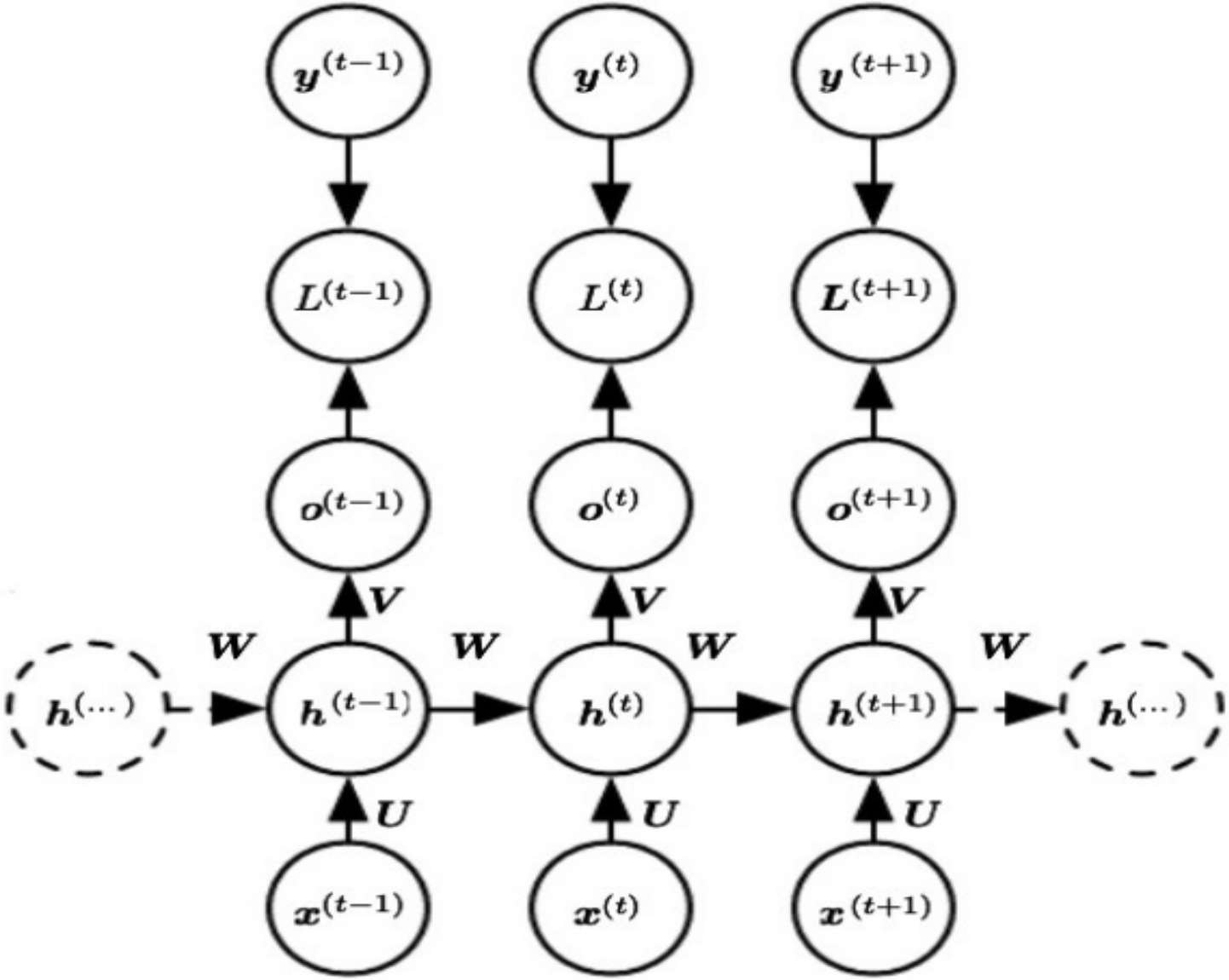

Figure: RNN (A) Computation Graph

Recurrent Neural Network (A)

- If we assume a discrete output, a softmax activation function, then for $t = 1$ to $t = r$, the forward computation is given as:

$$
\begin{aligned}
a^{(t)} &= b + Wh^{(t-1)} + Ux^{(t)} \\
h^{(t)} &= tanh(a^{(t)}) \\
o^{(t)} &= c + Vh^{(t)} \\
\hat{y}^{(t)} &= softmax(o^{(t)})
\end{aligned}
\tag{6}
$$

- We have to specify the initial state $h^{(0)}$
- The total loss for a pair of sequence of x and y values is just the sum of the losses over all time steps

$$
L\left(\{x^{(1)}, ..., x^{(r)}\}, \{y^{(1)}, ..., y^{(r)}\}\right) = \sum_{t} L^{(t)} \tag{7}
$$

Backpropagation for RNN (A)

- The math is highly intractable
- I'm yet to find a reference with detail derivations.

Recurrent Neural Network (B)

- Structure of RNN (B)
  - Produces an output at each time step
  - Have recurrent connection between output and the hidden layer
  - At each time step t, the input is $x^{(t)}$, the hidden layer activations are $h^{(t)}$
- Why it's less powerful compared to RNN (A)
  - This network is trained to put a specific input value into o, and o is the only input it passes forward
  - They're no direct connections from h going forward, only indirectly via it's predictions.
  - Unless the output o is high dimensional, it'll usually contain less information.
- Nonetheless, it's very easy to paralelize since each time step can be trained separately from the other.

Recurrent Neural Network (B)

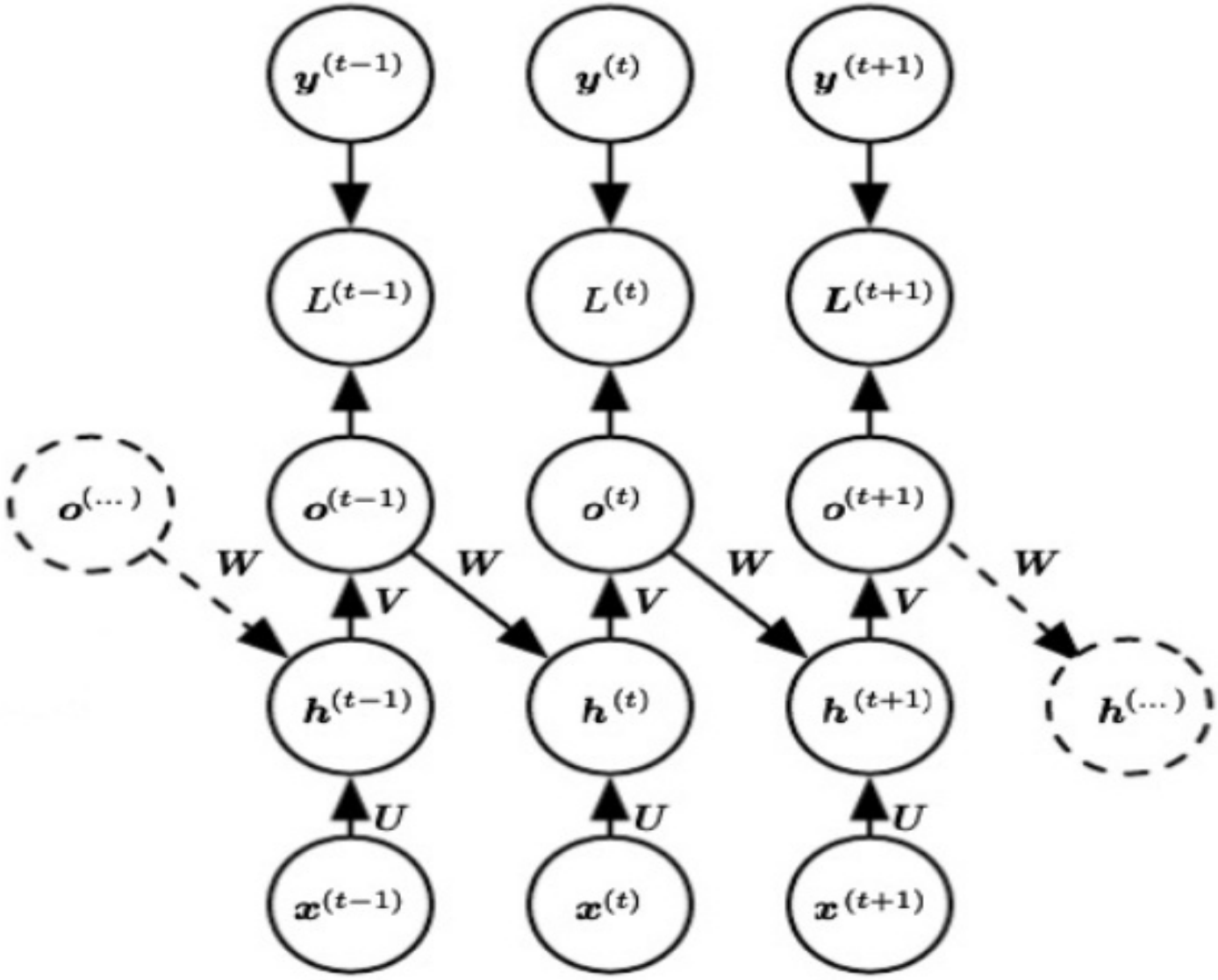

Figure: RNN (B) Computation Graph

Recurrent Neural Network (C)

- This basically has a single output at the end of the sequence.

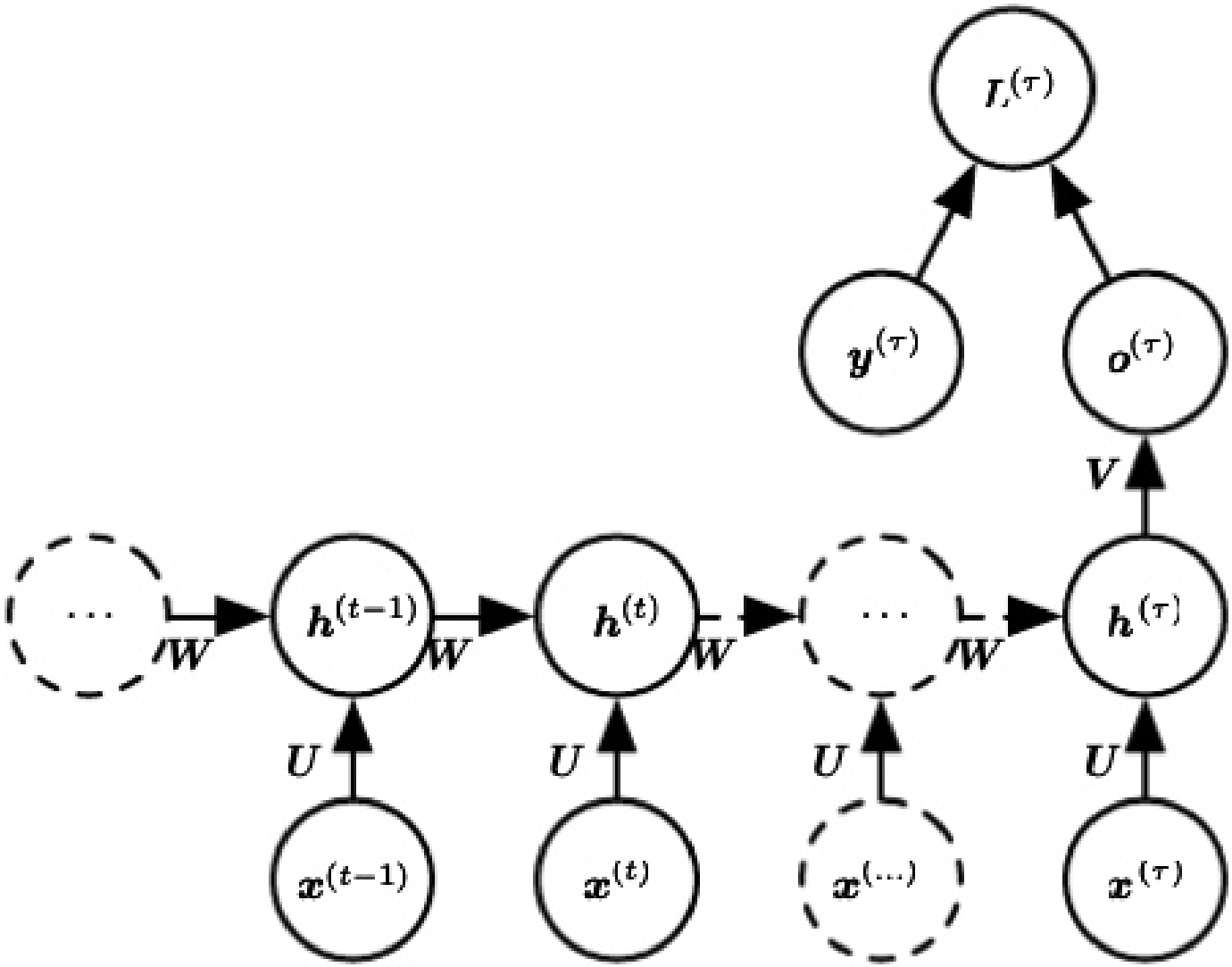

Figure: RNN (C) Computation Graph

Recurrent Neural Network (C)

- This is less powerful than RNN (C) and best suited for various preprocessing task
- For example, it could be used to produce a an input to a network by summarizing a sequence of observations

Next Meeting

- Backpropagation for RNN (A)
- RNN (B) and RNN (C)
- Deep Recurrent Networks

Outline

References

1. Ian Goodfellow and Yoshua Bengio and Aaron Courville
   - Chapter 10 $\sim$ Click to access