

# Deep Learning for NLP

## Lecture 7: Recurrent Neural Networks

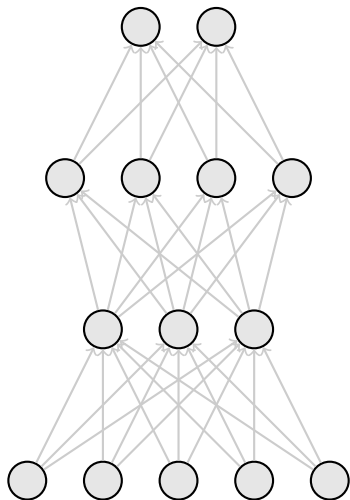
**Dr. Mohsen Mesgar**

**Ubiquitous Knowledge Processing Lab (UKP Lab)**

# This lecture

- ▶ RNNs
- ▶ Vanishing and explosion
- ▶ GRUs
- ▶ LSTMs
- ▶ Application of RNNs in NLP

# Recall: MultiLayer Perceptron (MLP)



$$\hat{\mathbf{y}} = f(\mathbf{h}_2 \mathbf{W}^{(3)} + \mathbf{b}^{(3)}) = [\hat{y}_1, \hat{y}_2]$$

$$\mathbf{W}^{(3)} \in \mathbb{R}^{|\mathbf{h}_2| \times |\hat{\mathbf{y}}|}, \mathbf{b}^{(3)} \in \mathbb{R}^{1 \times |\mathbf{h}_3|}$$

$$\mathbf{h}_2 = f(\mathbf{h}_1 \mathbf{W}^{(2)} + \mathbf{b}^{(2)})$$

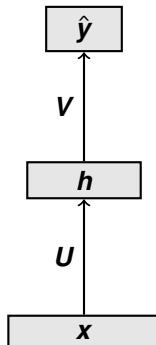
$$\mathbf{W}^{(2)} \in \mathbb{R}^{|\mathbf{h}_1| \times |\mathbf{h}_2|}, \mathbf{b}^{(2)} \in \mathbb{R}^{1 \times |\mathbf{h}_2|}$$

$$\mathbf{h}_1 = f(\mathbf{x} \mathbf{W}^{(1)} + \mathbf{b}^{(1)}),$$

$$\mathbf{W}^{(1)} \in \mathbb{R}^{|\mathbf{x}| \times |\mathbf{h}_1|}, \mathbf{b}^{(1)} \in \mathbb{R}^{1 \times |\mathbf{h}_1|}$$

$$\mathbf{x} = [x_1, x_2, x_3, x_4], \mathbf{x} \in \mathbb{R}^{1 \times |\mathbf{x}|}$$

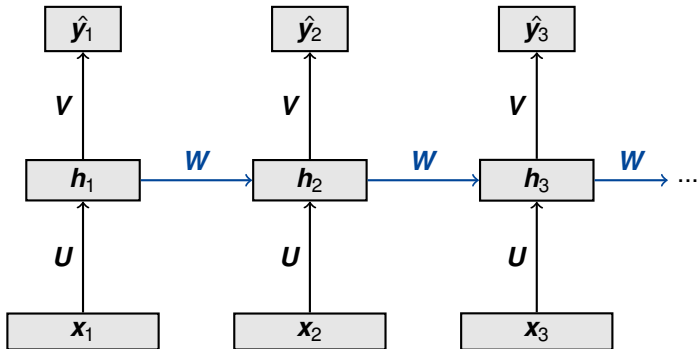
# Recall: Feedforward



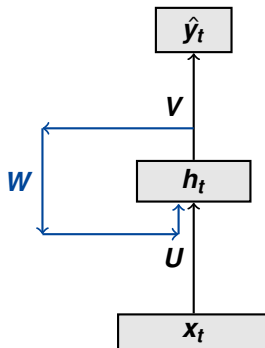
# Motivation

- ▶ Text is a **sequence** of symbols.
- ▶ Symbols show characters, words etc.
- ▶ input = “a persian cat on the mat .”  $\longrightarrow$   
input =  $(x_1 = \text{a}, x_2 = \text{persian}, x_3 = \text{cat}, x_4 = \text{sat}, x_5 = \text{on}, x_6 = \text{the}, x_7 = \text{mat})$
- ▶ As such symbols are related to each other in text, so should symbols' representations.
- ▶ How can we relate representations of symbols to each other?

# Recurrent



# Recurrent



# Recurrent

- ▶ input = a sequence of vectors =  $[\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \dots, \mathbf{x}_n]$ ,  $\mathbf{x}_t \in \mathbb{R}^{1 \times |x|}$



$$\mathbf{h}_t = f(\mathbf{x}_t U + \mathbf{h}_{t-1} W + \mathbf{b}_h)$$

- ▶  $U \in \mathbb{R}^{|x| \times |h|}$ ,  $|h|$  is the dimensionality of hidden states
- ▶  $W \in \mathbb{R}^{|h| \times |h|}$
- ▶  $\mathbf{b}_h \in \mathbb{R}^{1 \times |h|}$



$$\hat{\mathbf{y}}_t = g(\mathbf{h}_t V + \mathbf{b}_{\hat{y}})$$

- ▶  $V \in \mathbb{R}^{|h| \times |\hat{y}|}$
- ▶  $\mathbf{b}_{\hat{y}} \in \mathbb{R}^{1 \times |\hat{y}|}$



# Training

- ▶ input =  $[\mathbf{x}_1, \dots, \mathbf{x}_n]$ ,  $\longrightarrow$  output =  $[\mathbf{y}_1, \dots, \mathbf{y}_n]$
- ▶ The loss of sequence prediction is the mean of step losses,

$$\ell = \frac{1}{n} \sum_{t=1}^n \ell_t(y_t, \hat{y}_t).$$

- ▶ Use backprop to compute gradients.
- ▶ Use SGD to update parameters.

# Some Properties of RNNs

- ▶ Hidden state is known also as memory.
- ▶ In principle the hidden state represents information from the first step until the current step conditioned on current input symbol.
- ▶ So RNNs can capture left-to-right order of input symbols

# Example

- ▶ input = ( $x_1 = a$ ,  $x_2 = \text{persian}$ ,  $x_3 = \text{cat}$ )  $\longrightarrow$   
output = ( $y_1 = \text{DET}$ ,  $y_2 = \text{ADJ}$ ,  $y_3 = \text{NOUN}$ )
- ▶ Assume following embeddings for input:
  - ▶  $\mathbf{x}_1 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$
  - ▶  $\mathbf{x}_2 = \begin{bmatrix} 1 & 1 & 2 \end{bmatrix}$
  - ▶  $\mathbf{x}_3 = \begin{bmatrix} 1 & -1 & 1 \end{bmatrix}$
- ▶ Assume following 1-hot vectors for output:
  - ▶  $\mathbf{y}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$
  - ▶  $\mathbf{y}_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}$
  - ▶  $\mathbf{y}_3 = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$

# Example

- ▶ The sequence prediction task can be solved by RNNs
- ▶ Let  $|h| = 2$ ,  $f$  be `ReLU`, and  $g$  be `softmax`
- ▶ We initialize the RNN's parameters by random values

▶  $U = \begin{bmatrix} 1 & 1 \\ 2 & 0 \\ 0.5 & 1 \end{bmatrix}$

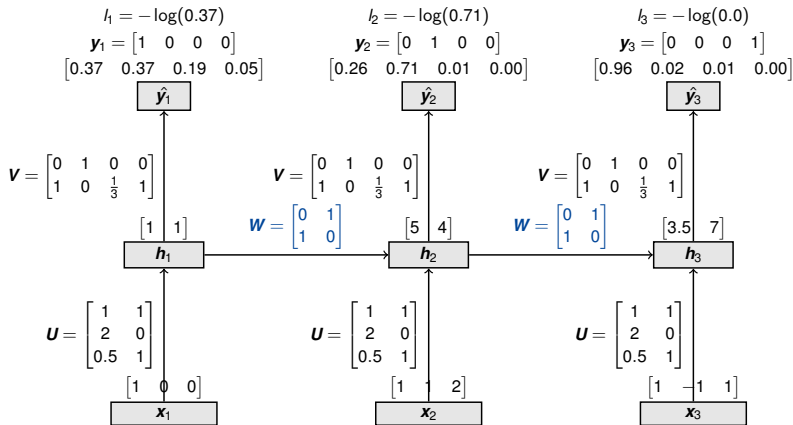
▶  $W = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

▶  $V = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & \frac{1}{3} & 1 \end{bmatrix}$

▶  $b_h, b_{\hat{y}}$  zero vectors

▶  $h_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}$

# Example



$$h_t = \text{ReLU}(x_t U + h_{t-1} W + b_h)$$

$$\hat{y}_t = \text{softmax}(h_t V + b_{\hat{y}})$$

$$\text{loss} = \frac{-1}{3}(\log(0.37) + \log(0.71) + \log(0.0))$$

# Updating Parameters

- ▶ Computation of gradients is similar as standard MLP.
- ▶ Keep in mind the parameters of RNNs are shared through time steps.
- ▶ Sometime backprop for RNNs is called backprop through time (BPTT).
- ▶ No need to go through its details as DL frameworks compute gradients.
- ▶ If you would like to compute gradients brute-force, you can do that numerically.
  - ▶ for each weight  $w$ , compute  $\frac{\text{Loss}(w+h) - \text{Loss}(w)}{h}$ ,
  - ▶ weight update after gradient computation is as in SGD
$$w \leftarrow w - \alpha \frac{\partial \text{Loss}}{\partial w}$$

# Updating Parameters

- ▶ RNNs can be seen as a very very deep neural model with sparse and skip connections
- ▶ The output of last steps are calculated based on the hidden state vectors at early steps
- ▶ Recall 1:

$$\mathbf{z}_t^{(h)} = \mathbf{x}_t U + \mathbf{h}_{t-1} W + \mathbf{b}_h$$
$$\mathbf{h}_t = f(\mathbf{z}_t^{(h)})$$

- ▶ Recall 2:

$$\mathbf{z}_t^{(\hat{y})} = \mathbf{h}_t V + \mathbf{b}_{\hat{y}}$$
$$\hat{\mathbf{y}}_t = g(\mathbf{z}_t^{(\hat{y})})$$

# Updating Parameters

- Recall 1:

$$\mathbf{z}_t^{(h)} = \mathbf{x}_t U + \mathbf{h}_{t-1} W + \mathbf{b}_h$$

$$\mathbf{h}_t = f(\mathbf{z}_t^{(h)})$$

- Recall 2:

$$\mathbf{z}_t^{(\hat{y})} = \mathbf{h}_t V + \mathbf{b}_{\hat{y}}$$

$$\hat{\mathbf{y}}_t = g(\mathbf{z}_t^{(\hat{y})})$$

- If we have only two steps:

$$\frac{\partial \text{Loss}}{\partial \mathbf{W}} = \sum_{t=1}^2 \frac{\partial \ell_t}{\partial \mathbf{W}} = \frac{\partial \ell_1}{\partial \mathbf{W}} + \frac{\partial \ell_2}{\partial \mathbf{W}}$$

$$\frac{\partial \ell_1}{\partial \mathbf{W}} = \frac{\partial \ell_1}{\partial \hat{\mathbf{y}}_1} \frac{\partial \hat{\mathbf{y}}_1}{\partial \mathbf{z}_1^{(\hat{y})}} \frac{\partial \mathbf{z}_1^{(\hat{y})}}{\partial \mathbf{h}_1} \frac{\partial \mathbf{h}_1}{\partial \mathbf{z}_1^{(h)}} \frac{\partial \mathbf{z}_1^{(h)}}{\partial \mathbf{W}} \propto \frac{\partial \mathbf{h}_1}{\partial \mathbf{z}_1^{(h)}}$$

$$\frac{\partial \ell_2}{\partial \mathbf{W}} = \frac{\partial \ell_2}{\partial \hat{\mathbf{y}}_2} \frac{\partial \hat{\mathbf{y}}_2}{\partial \mathbf{z}_2^{(\hat{y})}} \frac{\partial \mathbf{z}_2^{(\hat{y})}}{\partial \mathbf{h}_2} \frac{\partial \mathbf{h}_2}{\partial \mathbf{z}_2^{(h)}} \mathbf{h}_1 \frac{\partial \mathbf{h}_1}{\partial \mathbf{z}_1^{(h)}} \frac{\partial \mathbf{z}_1^{(h)}}{\partial \mathbf{W}} \propto \frac{\partial \mathbf{h}_2}{\partial \mathbf{z}_2^{(h)}} \frac{\partial \mathbf{h}_1}{\partial \mathbf{z}_1^{(h)}}$$



# Updating Parameters

- If we have three steps:

$$\frac{\partial \text{Loss}}{\partial \mathbf{W}} = \sum_{t=1}^3 \frac{\partial \ell_t}{\partial \mathbf{W}} = \frac{\partial \ell_1}{\partial \mathbf{W}} + \frac{\partial \ell_2}{\partial \mathbf{W}} + \frac{\partial \ell_3}{\partial \mathbf{W}}$$

$$\frac{\partial \ell_1}{\partial \mathbf{W}} \propto \frac{\partial \mathbf{h}_1}{\partial \mathbf{z}_1^{(h)}}$$

$$\frac{\partial \ell_2}{\partial \mathbf{W}} \propto \frac{\partial \mathbf{h}_2}{\partial \mathbf{z}_2^{(h)}} \frac{\partial \mathbf{h}_1}{\partial \mathbf{z}_1^{(h)}}$$

$$\frac{\partial \ell_3}{\partial \mathbf{W}} \propto \frac{\partial \mathbf{h}_3}{\partial \mathbf{z}_3^{(h)}} \frac{\partial \mathbf{h}_2}{\partial \mathbf{z}_2^{(h)}} \frac{\partial \mathbf{h}_1}{\partial \mathbf{z}_1^{(h)}}$$

# Updating Parameters

- If we have  $n$  steps:

$$\frac{\partial \text{Loss}}{\partial \mathbf{W}} = \sum_{t=1}^3 \frac{\partial \ell_t}{\partial \mathbf{W}} = \frac{\partial \ell_1}{\partial \mathbf{W}} + \frac{\partial \ell_2}{\partial \mathbf{W}} + \frac{\partial \ell_3}{\partial \mathbf{W}}$$

$$\frac{\partial \ell_1}{\partial \mathbf{W}} \propto \frac{\partial \mathbf{h}_1}{\partial \mathbf{z}_1^{(h)}}$$

$$\frac{\partial \ell_2}{\partial \mathbf{W}} \propto \frac{\partial \mathbf{h}_2}{\partial \mathbf{z}_2^{(h)}} \frac{\partial \mathbf{h}_1}{\partial \mathbf{z}_1^{(h)}}$$

$$\frac{\partial \ell_3}{\partial \mathbf{W}} \propto \frac{\partial \mathbf{h}_3}{\partial \mathbf{z}_3^{(h)}} \frac{\partial \mathbf{h}_2}{\partial \mathbf{z}_2^{(h)}} \frac{\partial \mathbf{h}_1}{\partial \mathbf{z}_1^{(h)}}$$

...

$$\frac{\partial \ell_n}{\partial \mathbf{W}} \propto \frac{\partial \mathbf{h}_n}{\partial \mathbf{z}_n^{(h)}} \cdots \frac{\partial \mathbf{h}_3}{\partial \mathbf{z}_3^{(h)}} \frac{\partial \mathbf{h}_2}{\partial \mathbf{z}_2^{(h)}} \frac{\partial \mathbf{h}_1}{\partial \mathbf{z}_1^{(h)}}$$

# Exploding Gradients

► Given:

$$\frac{\partial \ell_n}{\partial \mathbf{W}} \propto \frac{\partial \mathbf{h}_n}{\partial \mathbf{z}_n^{(h)}} \cdots \frac{\partial \mathbf{h}_3}{\partial \mathbf{z}_3^{(h)}} \frac{\partial \mathbf{h}_2}{\partial \mathbf{z}_2^{(h)}} \frac{\partial \mathbf{h}_1}{\partial \mathbf{z}_1^{(h)}}$$

if all the gradients in the chain are greater than one then their multiplications explodes

$$\frac{\partial \ell_n}{\partial \mathbf{W}} = \text{NaN}$$

# Vanishing Gradients

► Given:

$$\frac{\partial \ell_n}{\partial \mathbf{W}} \propto \frac{\partial \mathbf{h}_n}{\partial \mathbf{z}_n^{(h)}} \cdots \frac{\partial \mathbf{h}_3}{\partial \mathbf{z}_3^{(h)}} \frac{\partial \mathbf{h}_2}{\partial \mathbf{z}_2^{(h)}} \frac{\partial \mathbf{h}_1}{\partial \mathbf{z}_1^{(h)}}$$

if one of the gradients in the chain is close to zero or all gradients are less than one then their multiplications vanishes

$$\frac{\partial \ell_n}{\partial \mathbf{W}} = 0.0$$

# Vanishing and Exploding Gradients

- ▶ Why are such gradients a problem?
- ▶ In case of exploding gradients, the learning is very unstable
- ▶ The last steps become independent from the early steps
- ▶ The prediction at each step is conditioned only on a few previous steps
- ▶ These problems also happen in MLPs with many hidden layer where we use the Sigmoid activation function

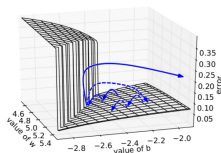
# Vanishing and Exploding Gradients

Some signs that show a model and its training may suffer from these problems:

- ▶ a very high loss on training set or no learning
- ▶ large changes in loss on each update due to the models instability
- ▶ loss becomes NaN during training
- ▶ model weights grow exponentially during training (explosion)
- ▶ the model does not learn during training
- ▶ training stops very early and any further training does not decrease the loss
- ▶ the weights closer to the last steps would change more than those at early steps
- ▶ weights shrink exponentially and become very small
- ▶ the model weights become 0 in the training phase.

# Simple Remedies for Vanishing/Exploding Gradients

- ▶ For activation of hidden layers use ReLU, and initialize  $W$  with the identity matrix (Le et al., 2015)
- ▶ Gradient clipping (Pascanu et al., 2013)



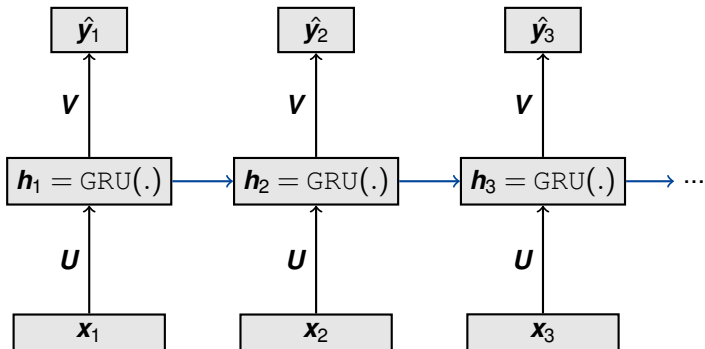
- ▶ GRUs and LSTMs

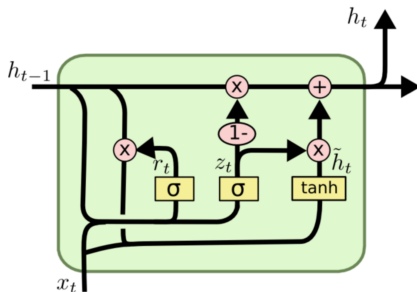
# Gated Recurrent Units (GRUs)

- ▶ GRUs are introduced by Cho et al., (2014)
- ▶ more advanced method for hidden state representation
- ▶ The key idea behind GRUs is to enable hidden states to capture long distance dependencies



# GRUs





- ▶ reset gate  $r_t = \sigma(\mathbf{x}_t \mathbf{U}^{(r)} + \mathbf{h}_{t-1} \mathbf{W}^{(r)})$
- ▶ new memory content could be  $\tilde{\mathbf{h}}_t = \tanh(\mathbf{x}_t \mathbf{U} + \mathbf{h}_{t-1} \mathbf{W} \odot r_t)$
- ▶ update gate  $z_t = \sigma(\mathbf{x}_t \mathbf{U}^{(z)} + \mathbf{h}_{t-1} \mathbf{W}^{(z)})$
- ▶ final memory encodes a combination of current content and its content in the previous time step  $\mathbf{h}_t = (1 - z_t) \odot \mathbf{h}_{t-1} + z_t \odot \tilde{\mathbf{h}}_t$

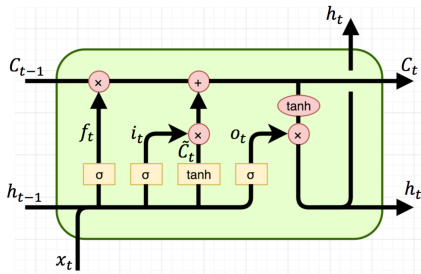
# GRU: Extreme Cases

- ▶ reset gate  $r_t \in 0, 1$
- ▶ update gate  $z_t \in 0, 1$
- ▶ If  $z_t = 0$ 
  - ▶  $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$
  - ▶  $h_t = h_{t-1}$
  - ▶ zero gradients over different time steps
  - ▶ no vanishing gradients
- ▶ If  $z_t = 1$ 
  - ▶  $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$
  - ▶  $h_t = \tilde{h}_t$
  - ▶  $\tilde{h}_t = \tanh(x_t U + h_{t-1} W \odot r_t)$
  - ▶ If  $r_t = 0$ 
    - ▶  $h_t = \tanh(x_t U)$
    - ▶ forget past
  - ▶ If  $r_t = 1$ 
    - ▶  $h_t = \tanh(x_t U + h_{t-1} W)$
    - ▶ a standard RNN gate

# Long Short-Term Memory (LSTMs)

- ▶ LSTMs were introduced by Hochreiter and Schmidhuber (1997)
- ▶ LSTMs contains more parameters than what GRU has

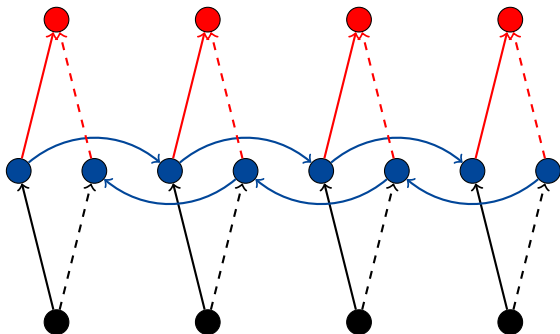
# LSTM Unit



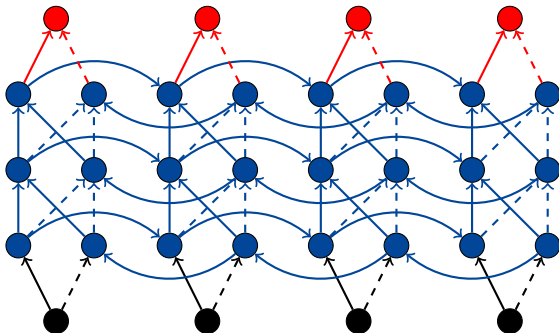
- ▶ Input gate (=write gate)  $i_t = \sigma(x_t U^{(i)} + h_{t-1} W^{(i)})$
- ▶ Forget gate (=reset gate)  $f_t = \sigma(x_t U^{(f)} + h_{t-1} W^{(f)})$
- ▶ Output gate (=read gate)  $o_t = \sigma(x_t U^{(o)} + h_{t-1} W^{(o)})$
- ▶ New memory cell is  $\tilde{c}_t = \tanh(x_t U + h_{t-1} W)$
- ▶ Final memory cell is  $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$
- ▶ Final hidden state is  $h_t = o_t \odot \tanh(c_t)$

# Bidirectional RNNs (BiRNNs)

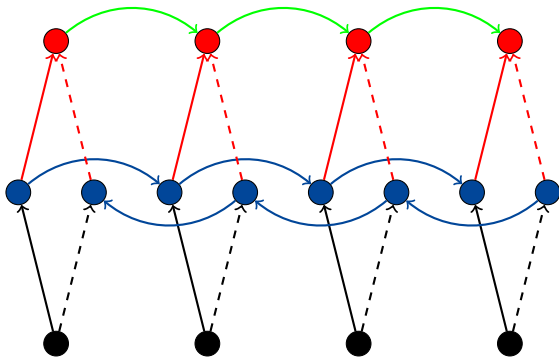
- ▶ We use two RNNs with two different sets of parameters
  - ▶ one RNN for processing input symbols from left-to-right
  - ▶ one RNN for processing input symbols from right-to-left
- ▶ Final representations of each step is the concatenation of the outputs of these RNNs.
  - ▶  $\mathbf{h}_t = [\vec{h}_t; \overleftarrow{h}_t]$



# Deep BiRNNs



# RNNs With Output Connections





# Applications of RNNs in NLP

- ▶ Part-of-Speech (POS) tagging
  - ▶ input: a sequence of words
  - ▶ output: a sequence of POS tags (NOUN, VERB, ...)
- ▶ Named Entity Recognition (NER)
  - ▶ input: a sequence of words
  - ▶ output: a sequence of NER tags (B-PER, I-PER, O, B-LOC, I-LOC,...)
- ▶ Language Modeling (LM)
  - ▶ input: a sequence of words
  - ▶ output: a sequence of words  $y_t = x_{t-1}$
- ▶ Sentence Classification
  - ▶ input: a sequence of words
  - ▶ output: one label for the whole sequence
  - ▶ trick: use the hidden state of the last step to represent the whole sentence

# Summary

- ▶ RNNs are used mostly for sequence prediction.
- ▶ RNNs face with vanishing and exploding gradient issues.
- ▶ Vanishing and Exploding Gradients in RNNs
- ▶ LSTMs and GRUs
- ▶ Applications of RNNs

Thank You!