Deep Learning for NLP
Lecture 7: Recurrent Neural Networks

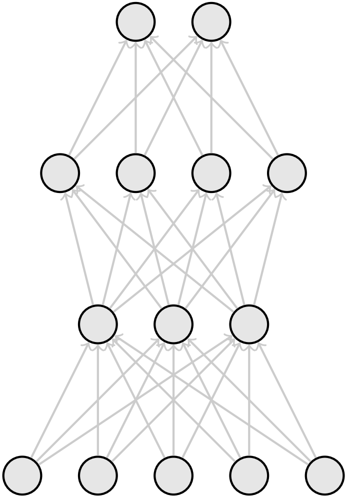
Dr. Mohsen Mesgar

Ubiquitous Knowledge Processing Lab (UKP Lab)

This lecture

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

Recall: MultiLayer Perceptron (MLP)



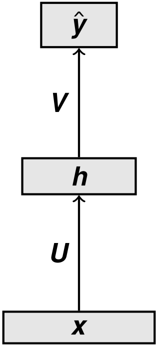
$$\hat{\boldsymbol{y}} = f\left(\boldsymbol{h}_2 \boldsymbol{W}^{(3)} + \boldsymbol{b}^{(3)}\right) = [\hat{y}_1, \hat{y}_2]$$
$$\boldsymbol{W}^{(3)} \in \mathbb{R}^{|\boldsymbol{h}_2| \times |\hat{\boldsymbol{y}}|}, \boldsymbol{b}^{(3)} \in \mathbb{R}^{1 \times |\boldsymbol{h}_3|}$$

$$\boldsymbol{h}_2 = f\left(\boldsymbol{h}_1 \boldsymbol{W}^{(2)} + \boldsymbol{b}^{(2)}\right)$$
$$\boldsymbol{W}^{(2)} \in \mathbb{R}^{|\boldsymbol{h}_1| \times |\boldsymbol{h}_2|}, \boldsymbol{b}^{(2)} \in \mathbb{R}^{1 \times |\boldsymbol{h}_2|}$$

$$\boldsymbol{h}_1 = f\left(\boldsymbol{x} \boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)}\right),$$
$$\boldsymbol{W}^{(1)} \in \mathbb{R}^{|\boldsymbol{x}| \times |\boldsymbol{h}_1|}, \boldsymbol{b}^{(1)} \in \mathbb{R}^{1 \times |\boldsymbol{h}_1|}$$

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

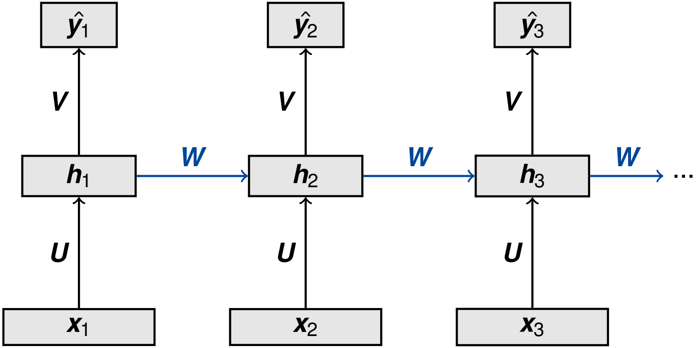
Recall: Feedforward



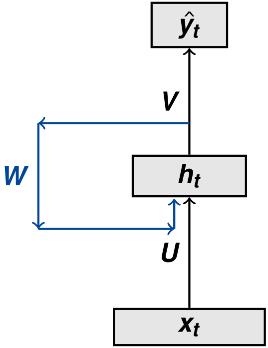
Motivation

- Text is a **sequence** of symbols.
- Symbols show characters, words etc.
- input = “a persian cat sat on the mat .” $\longrightarrow$
  input = $\big(x_1 =$a, $x_2 =$persian, $x_3 =$cat, $x_4 =$sat, $x_5 =$on, $x_6 =$the, $x_7 =$mat$\big)$
- 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 = $[\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, ..., \boldsymbol{x}_n]$, $\boldsymbol{x_t} \in \mathbb{R}^{1\times|x|}$
- $$\boldsymbol{h_t} = f\left(\boldsymbol{x_t} U + \boldsymbol{h}_{t-1} W + \boldsymbol{b_h}\right)$$
    - $\boldsymbol{U} \in \mathbb{R}^{|x|\times|h|}$, $|h|$ is the dimensionality of hidden states
    - $\boldsymbol{W} \in \mathbb{R}^{|h|\times|h|}$
    - $\boldsymbol{b_h} \in \mathbb{R}^{1\times|h|}$
- $$\hat{\boldsymbol{y}}_{\boldsymbol{t}} = g\left(\boldsymbol{h_t} V + \boldsymbol{b}_{\hat{y}}\right)$$
    - $\boldsymbol{V} \in \mathbb{R}^{|h|\times|\hat{y}|}$
    - $\boldsymbol{b}_{\hat{y}} \in \mathbb{R}^{1\times|\hat{y}|}$

Training

- input = $[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\boldsymbol{n}}]$, $\longrightarrow$ output = $[\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{\boldsymbol{n}}]$
- 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

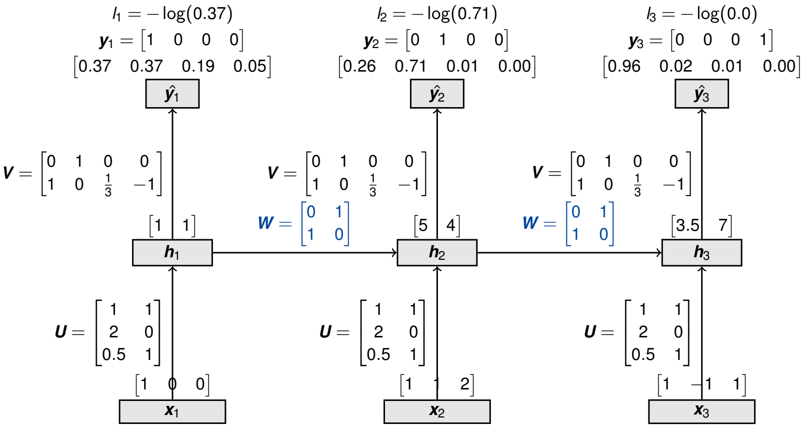
Example

- input = $\left(x1 = \text{a}, x2 = \text{persian}, x3 = \text{cat}\right) \longrightarrow$
  output = $\left(y_1 = \text{DET}, y_2 = \text{ADJ}, y_3 = \text{NOUN}\right)$
- Assume following embeddings for input:
  - $\boldsymbol{x}_1 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$
  - $\boldsymbol{x}_2 = \begin{bmatrix} 1 & 1 & 2 \end{bmatrix}$
  - $\boldsymbol{x}_3 = \begin{bmatrix} 1 & -1 & 1 \end{bmatrix}$
- Assume following 1-hot vectors for output:
  - $\boldsymbol{y}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$
  - $\boldsymbol{y}_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}$
  - $\boldsymbol{y}_3 = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$

Example

- The sequence prediction task can be solved by RNNs
- Let $|h| = 2$, f be $\texttt{ReLU}$, and g be $\texttt{softmax}$
- We initialize the RNN's parameters by random values
  - $\boldsymbol{U} = \begin{bmatrix} 1 & 1 \\ 2 & 0 \\ 0.5 & 1 \end{bmatrix}$
  - $\boldsymbol{W} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
  - $\boldsymbol{V} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & \frac{1}{3} & -1 \end{bmatrix}$
  - $\boldsymbol{b_h}, \boldsymbol{b}_{\hat{y}}$ zero vectors
  - $\boldsymbol{h}_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}$

Example



$$\boldsymbol{h_t} = \mathtt{ReLU}\left(\boldsymbol{x_t}\boldsymbol{U} + \boldsymbol{h}_{t-1}\boldsymbol{W} + \boldsymbol{b_h}\right)$$

$$\hat{\boldsymbol{y}}_{\boldsymbol{t}} = \mathtt{softmax}\left(\boldsymbol{h_t}\boldsymbol{V} + \boldsymbol{b}_{\hat{y}}\right)$$

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

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 $w \longleftarrow w - \alpha \frac{\partial \text{Loss}}{\partial w}$

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:

$$\boldsymbol{z_t}^{(h)} = \boldsymbol{x_t} U + \boldsymbol{h}_{t-1} W + \boldsymbol{b_h}$$
$$\boldsymbol{h_t} = f\left(\boldsymbol{z_t}^{(h)}\right)$$

- Recall 2:

$$\boldsymbol{z_t}^{(\hat{y})} = \boldsymbol{h_t} V + \boldsymbol{b}_{\hat{y}}$$
$$\boldsymbol{\hat{y}_t} = g\left(\boldsymbol{z_t}^{(\hat{y})}\right)$$

Updating Parameters

- Recall 1:

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

$$\boldsymbol{h}_{\boldsymbol{t}} = f\left(\boldsymbol{z}_{\boldsymbol{t}}^{(h)}\right)$$

- Recall 2:

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

$$\hat{\boldsymbol{y}}_{\boldsymbol{t}} = g\left(\boldsymbol{z}_{\boldsymbol{t}}^{(\hat{y})}\right)$$

- If we have only two steps:

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

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

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

Updating Parameters

- If we have three steps:

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

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

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

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

Updating Parameters

- If we have n steps:

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

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

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

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

$$\ldots$$

$$\frac{\partial \ell_n}{\partial \boldsymbol{W}} \propto \frac{\partial \boldsymbol{h}_{\boldsymbol{n}}}{\partial \boldsymbol{z}_{\boldsymbol{n}}^{(h)}} \cdots \frac{\partial \boldsymbol{h}_3}{\partial \boldsymbol{z}_3^{(h)}} \frac{\partial \boldsymbol{h}_2}{\partial \boldsymbol{z}_2^{(h)}} \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{z}_1^{(h)}}$$

Exploding Gradients

- Given:

$$\frac{\partial \ell_n}{\partial \boldsymbol{W}} \propto \frac{\partial \boldsymbol{h_n}}{\partial \boldsymbol{z_n}^{(h)}} \cdots \frac{\partial \boldsymbol{h}_3}{\partial \boldsymbol{z}_3^{(h)}} \frac{\partial \boldsymbol{h}_2}{\partial \boldsymbol{z}_2^{(h)}} \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{z}_1^{(h)}}$$

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

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

Vanishing Gradients

- Given:

$$\frac{\partial \ell_n}{\partial \boldsymbol{W}} \propto \frac{\partial \boldsymbol{h_n}}{\partial \boldsymbol{z_n}^{(h)}} \cdots \frac{\partial \boldsymbol{h}_3}{\partial \boldsymbol{z}_3^{(h)}} \frac{\partial \boldsymbol{h}_2}{\partial \boldsymbol{z}_2^{(h)}} \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{z}_1^{(h)}}$$

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

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

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

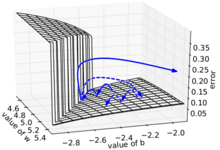
Vanishing and Exploding Gradients

Some sings 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)

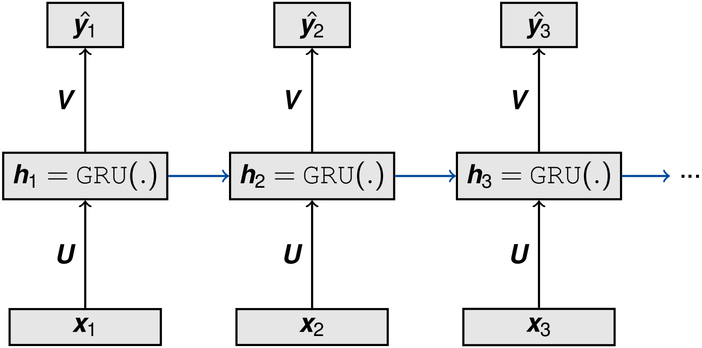


- GRUs and LSTMs

Gated Recurrent Units (GRUs)

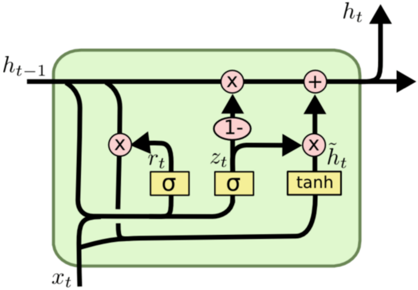
- GRUs are introduced by Cho et al., (2014)
- more advanced method for hidden state representation
- The key idea behind GRUs is to enable hidden states to capture long distance dependencies

GRUs



$\hat{\boldsymbol{y}}_1$
$\hat{\boldsymbol{y}}_2$
$\hat{\boldsymbol{y}}_3$
$\boldsymbol{V}$
$\boldsymbol{V}$
$\boldsymbol{V}$
$\boldsymbol{h}_1 = \mathrm{GRU}(.)$
$\boldsymbol{h}_2 = \mathrm{GRU}(.)$
$\boldsymbol{h}_3 = \mathrm{GRU}(.)$
...
$\boldsymbol{U}$
$\boldsymbol{U}$
$\boldsymbol{U}$
$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\boldsymbol{x}_3$

GRU



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

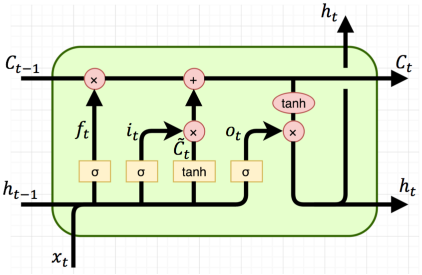
GRU: Extreme Cases

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

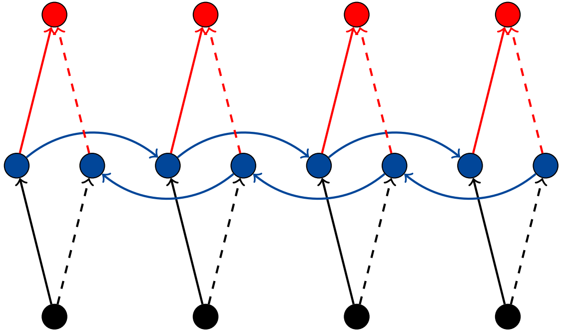
LSTM Unit



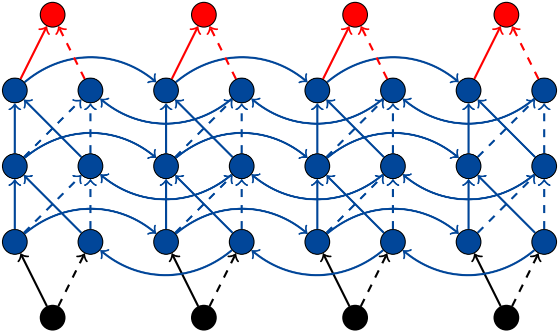
- Input gate (=write gate) $i_t = \sigma\left(\boldsymbol{x_t} \boldsymbol{U}^{(i)} + \boldsymbol{h}_{t-1} \boldsymbol{W}^{(i)}\right)$
- Forget gate (=reset gate) $f_t = \sigma\left(\boldsymbol{x_t} \boldsymbol{U}^{(f)} + \boldsymbol{h}_{t-1} \boldsymbol{W}^{(f)}\right)$
- Output gate (=read gate) $o_t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

Bidirectional RNNs (BiRNNs)

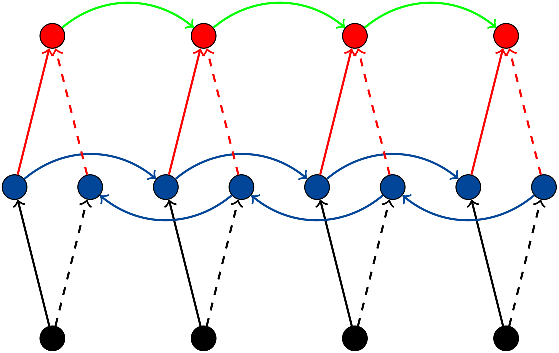
- We use two RNNs with two different sets of parameters
  - one RNN for processing input symbols from left-to-right
  - one RNN for processing input symbols from right-to-left
- Final representations of each step is the concatenation of the outputs of these RNNs.
  - $\boldsymbol{h_t} = [\overrightarrow{h_t};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



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!