Deep Learning for Natural Language Processing

Lecture 7 – Recurrent Neural Networks

Dr. Ivan Habernal

May 31, 2022

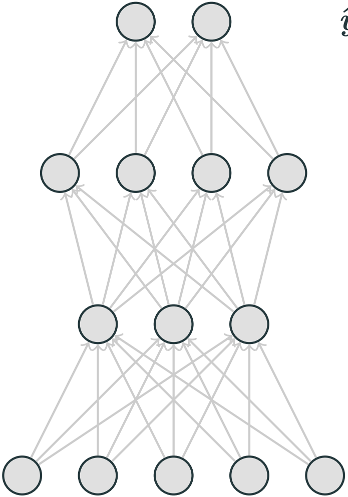
Trustworthy Human Language Technologies
Department of Computer Science
Technical University of Darmstadt

www.trusthlt.org

This lecture

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

Recall: MultiLayer Perceptron (MLP)



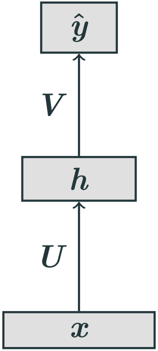
$$\hat{\boldsymbol{y}} = f\left(\boldsymbol{h_2}\,\boldsymbol{W}^{(3)} + \boldsymbol{b}^{(3)}\right) = [\hat{y}_1, \hat{y}_2]$$
$$\boldsymbol{W}^{(3)} \in \mathbb{R}^{|\boldsymbol{h_2}|\times|\hat{\boldsymbol{y}}|}, \boldsymbol{b}^{(3)} \in \mathbb{R}^{1\times|\boldsymbol{h_3}|}$$

$$\boldsymbol{h_2} = f\left(\boldsymbol{h_1}\,\boldsymbol{W}^{(2)} + \boldsymbol{b}^{(2)}\right)$$
$$\boldsymbol{W}^{(2)} \in \mathbb{R}^{|\boldsymbol{h_1}|\times|\boldsymbol{h_2}|}, \boldsymbol{b}^{(2)} \in \mathbb{R}^{1\times|\boldsymbol{h_2}|}$$

$$\boldsymbol{h_1} = f\left(\boldsymbol{x}\,\boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)}\right),$$
$$\boldsymbol{W}^{(1)} \in \mathbb{R}^{|\boldsymbol{x}|\times|\boldsymbol{h_1}|}, \boldsymbol{b}^{(1)} \in \mathbb{R}^{1\times|\boldsymbol{h_1}|}$$

$$\boldsymbol{x} = [x_1, x_2, x_3, x_4, x_5], \boldsymbol{x} \in \mathbb{R}^{1\times|\boldsymbol{x}|}$$

Recall: Feedforward



Motivation

Text is a **sequence** of symbols

Symbols = characters, words, etc.

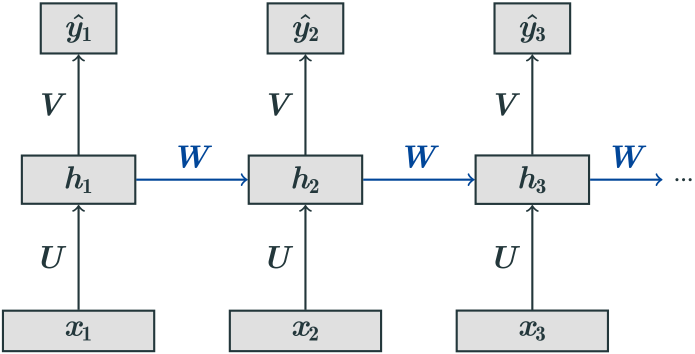
Example

input = “a persian cat on the mat” $\longrightarrow$

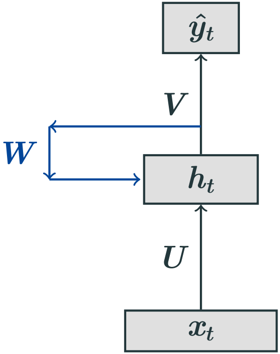
input = $\big(x_1 =$a, $x_2 =$persian, $x_3 =$cat, $x_4 =$sat, $x_5 =$on, $x_6 =$the, $x_7 =$mat$\big)$

Symbols are related to each other in text, so should symbols’ representations

Recurrent



Recurrent



$\hat{\boldsymbol{y}}_t$
$\boldsymbol{V}$
$\boldsymbol{W}$
$\boldsymbol{h}_t$
$\boldsymbol{U}$
$\boldsymbol{x}_t$

Recurrent

Input = a sequence of vectors = $[\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}, \ldots, \boldsymbol{x_n}]$, $\boldsymbol{x_t} \in \mathbb{R}^{1\times|x|}$

$$\boldsymbol{h_t} = f\Big(\boldsymbol{x_t}\, U + \boldsymbol{h_{t-1}}\, W + \boldsymbol{b_h}\Big)$$

$|h|$ is the dimensionality of hidden states

$$\boldsymbol{U} \in \mathbb{R}^{|x|\times|h|} \qquad \boldsymbol{W} \in \mathbb{R}^{|h|\times|h|} \qquad \boldsymbol{b_h} \in \mathbb{R}^{1\times|h|}$$

$$\boldsymbol{\hat{y}_t} = g\Big(\boldsymbol{h_t}\, V + \boldsymbol{b_{\hat{y}}}\Big)$$

$$\boldsymbol{V} \in \mathbb{R}^{|h|\times|\hat{y}|} \qquad \boldsymbol{b_{\hat{y}}} \in \mathbb{R}^{1\times|\hat{y}|}$$

Training

Input = $[\boldsymbol{x_1}, ..., \boldsymbol{x_n}]$, $\longrightarrow$ output = $[\boldsymbol{y_1}, ..., \boldsymbol{y_n}]$

The loss of sequence prediction is the mean of step losses

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

Backpropagation to compute gradients

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

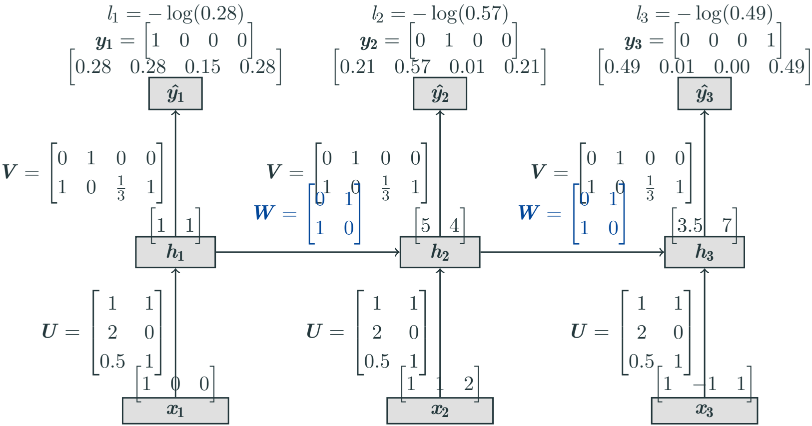
So RNNs can capture left-to-right order of input symbols

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 `ReLU`, and g be `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
    - $\boldsymbol{h_0} = \begin{bmatrix} 0 & 0 \end{bmatrix}$



$$\boldsymbol{h}_t = \mathrm{ReLU}\left(\boldsymbol{x}_t \boldsymbol{U} + \boldsymbol{h}_{t-1} \boldsymbol{W} + \boldsymbol{b}_h\right) \qquad \hat{\boldsymbol{y}}_t = \mathrm{softmax}\left(\boldsymbol{h}_t \boldsymbol{V} + \boldsymbol{b}_{\hat{y}}\right)$$

$$\ell = -\frac{1}{3}(\log(0.28) + \log(0.57) + \log(0.49))$$

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:

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

- Recall 2:

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

Updating Parameters

- Recall 1:

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

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

- Recall 2:

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

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

Updating Parameters

- 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:

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

$$\frac{\partial \ell_1}{\partial \boldsymbol{W}} \propto \frac{\partial \boldsymbol{h_1}}{\partial \boldsymbol{z_1^{(h)}}}$$

$$\frac{\partial \ell_2}{\partial \boldsymbol{W}} \propto \frac{\partial \boldsymbol{h_2}}{\partial \boldsymbol{z_2^{(h)}}} \frac{\partial \boldsymbol{h_1}}{\partial \boldsymbol{z_1^{(h)}}}$$

$$\frac{\partial \ell_3}{\partial \boldsymbol{W}} \propto \frac{\partial \boldsymbol{h_3}}{\partial \boldsymbol{z_3^{(h)}}} \frac{\partial \boldsymbol{h_2}}{\partial \boldsymbol{z_2^{(h)}}} \frac{\partial \boldsymbol{h_1}}{\partial \boldsymbol{z_1^{(h)}}}$$

$$\cdots$$

$$\frac{\partial \ell_n}{\partial \boldsymbol{W}} \propto \frac{\partial \boldsymbol{h_n}}{\partial \boldsymbol{z_n^{(h)}}} \cdots \frac{\partial \boldsymbol{h_3}}{\partial \boldsymbol{z_3^{(h)}}} \frac{\partial \boldsymbol{h_2}}{\partial \boldsymbol{z_2^{(h)}}} \frac{\partial \boldsymbol{h_1}}{\partial \boldsymbol{z_1^{(h)}}}$$

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

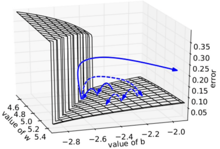
Vanishing and Exploding Gradients

When model and training suffer:

- 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

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)

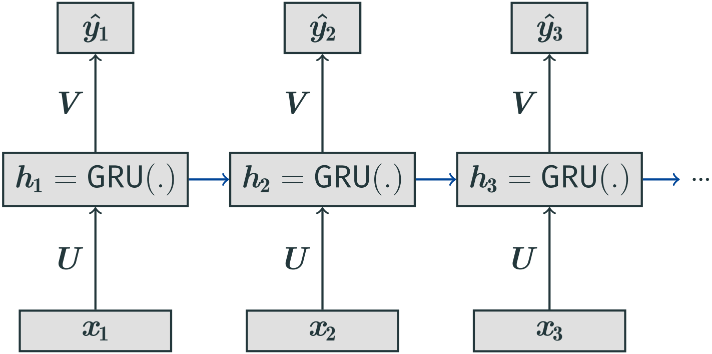


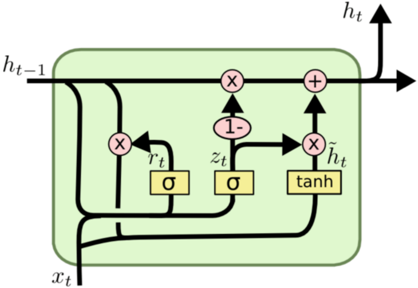
- GRUs and LSTMs

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


$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
V
V
V
$h_1 = \text{GRU}(.)$
$h_2 = \text{GRU}(.)$
$h_3 = \text{GRU}(.)$
...
U
U
U
x_1
x_2
x_3



- reset gate $r_t = \sigma(\boldsymbol{x_t}\boldsymbol{U^{(r)}} + \boldsymbol{h_{t-1}}\boldsymbol{W^{(r)}})$
- new memory content could be $\tilde{\boldsymbol{h}}_t = \texttt{tanh}(\boldsymbol{x_t}\boldsymbol{U} + \boldsymbol{h_{t-1}}\boldsymbol{W} \odot r_t)$
- 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 $\boldsymbol{h_t} = (1 - z_t) \odot \boldsymbol{h_{t-1}} + z_t \odot \tilde{\boldsymbol{h}}_t$

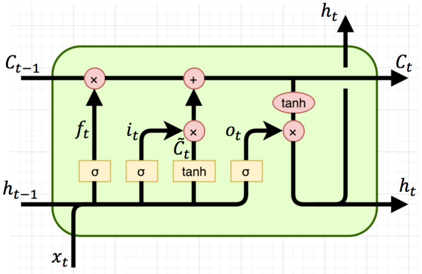
GRU: Extreme Cases

- 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 $z_t = 1$
  - $\boldsymbol{h_t} = (1 - z_t) \odot \boldsymbol{h_{t-1}} + z_t \odot \tilde{\boldsymbol{h}}_{\boldsymbol{t}}$
  - $\boldsymbol{h_t} = \tilde{\boldsymbol{h}}_{\boldsymbol{t}}$
  - $\tilde{\boldsymbol{h}}_{\boldsymbol{t}} = \mathtt{tanh}(\boldsymbol{x_t U} + \boldsymbol{h_{t-1} W} \odot r_t)$
  - If $r_t = 0$
    - $\boldsymbol{h_t} = \mathtt{tanh}(\boldsymbol{x_t U})$
    - forget past
  - If $r_t = 1$

Long Short-Term Memory (LSTMs)

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

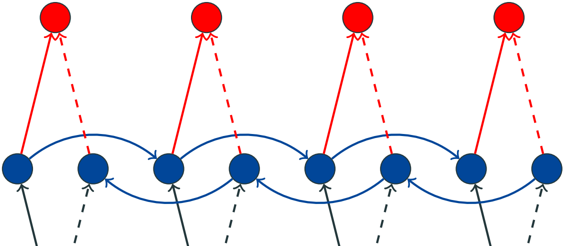
LSTM Unit



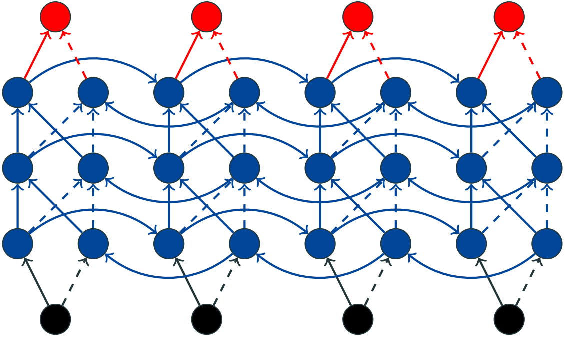
- Input gate (=write gate) $i_t = \sigma\left(\boldsymbol{x}_t \boldsymbol{U}^{(i)} + \boldsymbol{h}_{t-1} \boldsymbol{W}^{(i)}\right)$
- Forget gate (=reset gate) $f_t = \sigma\left(\boldsymbol{x}_t \boldsymbol{U}^{(f)} + \boldsymbol{h}_{t-1} \boldsymbol{W}^{(f)}\right)$
- Output gate (=read gate) $o_t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

Bidirectional RNNs (BiRNNs)

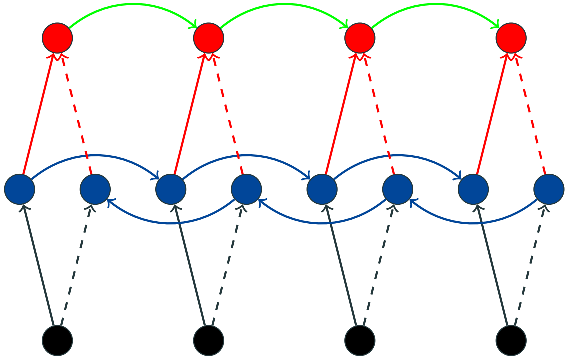
- We use two RNNs with two different sets of parameters
  - one RNN for processing input symbols from left-to-right
  - one RNN for processing input symbols from right-to-left
- Final representations of each step is the concatenation of the outputs of these RNNs.
  - $\boldsymbol{h_t} = [\overrightarrow{\boldsymbol{h_t}}\,;\overleftarrow{\boldsymbol{h_t}}]$



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

License and credits

Credits

Ivan Habernal, Mohsen Mesgar, Steffen Eger

References i