NLP course

Lecture 03: Vanishing gradient, Memory in RNNs

Radoslav Neychev

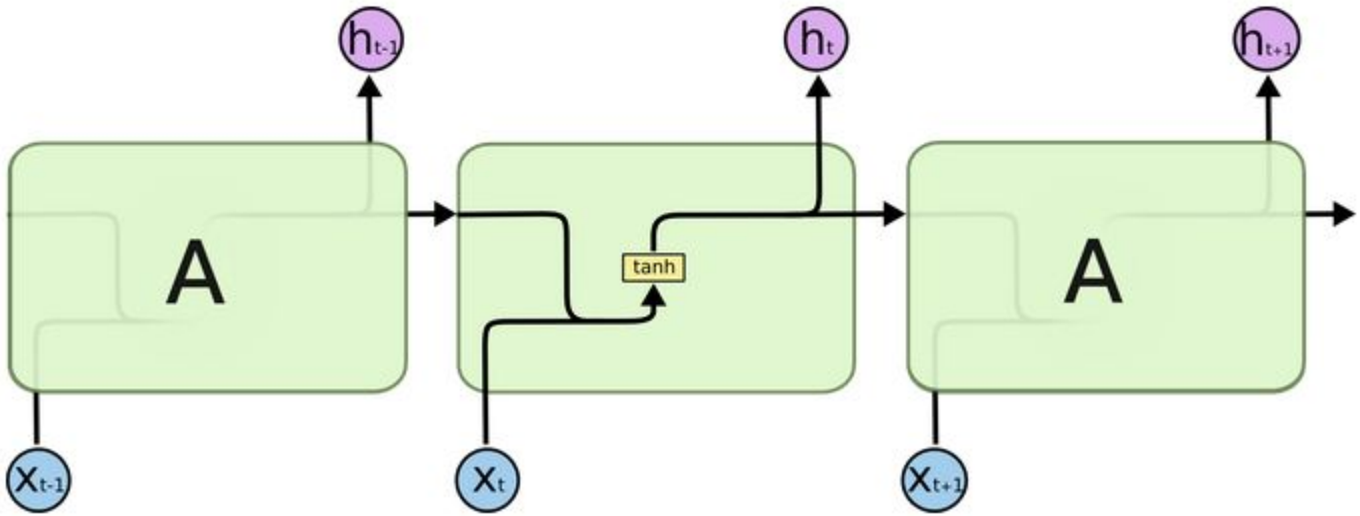
MADE, Moscow

24.03.2021

Outline

1. Recap: RNNs
2. LSTM overview
    - Gates in LSTM
3. RNNs as encoders for sequential data
4. Vanishing gradient problem
5. Exploding gradient problem
6. Q & A.

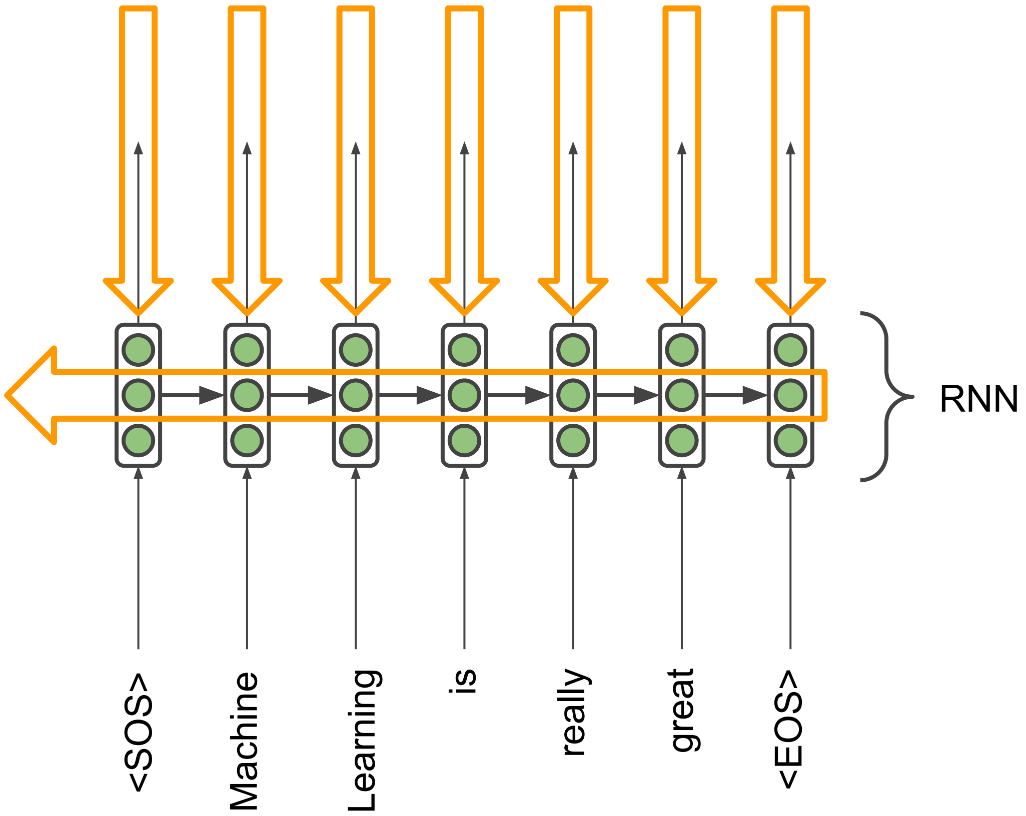
Vanilla RNN



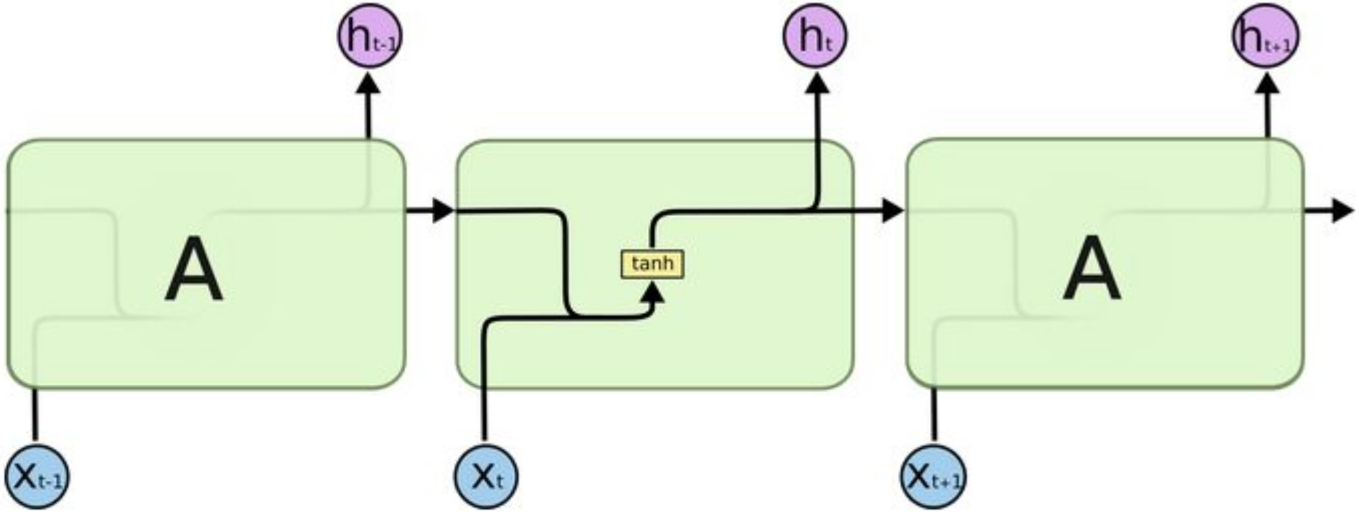
Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

How to train it?

Loss
(e.g. Negative
log-likelihood)

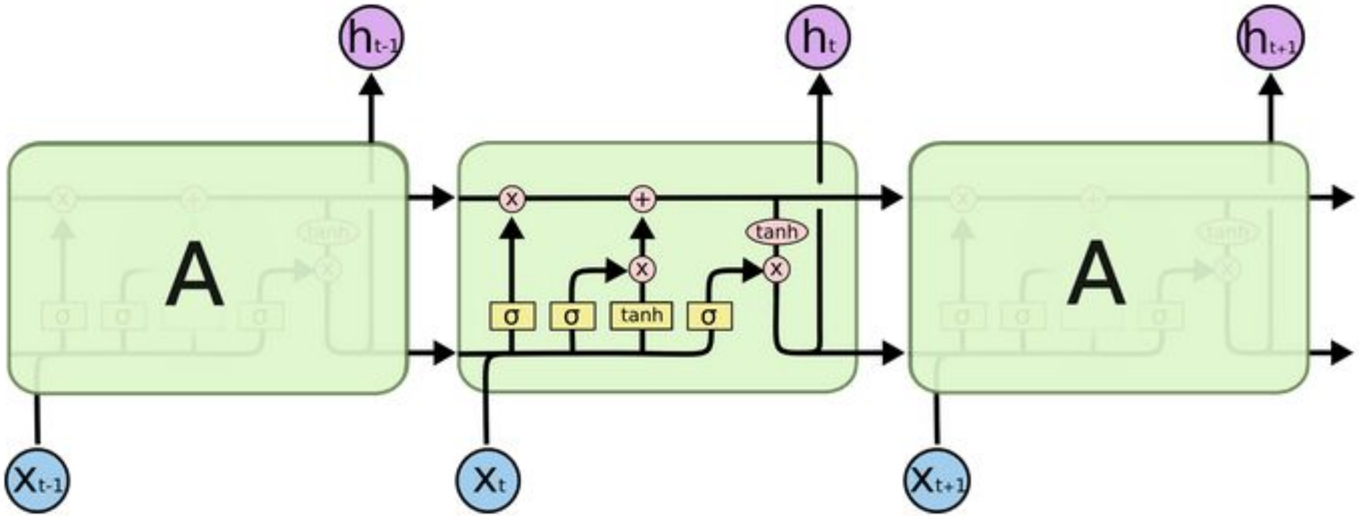


Vanilla RNN



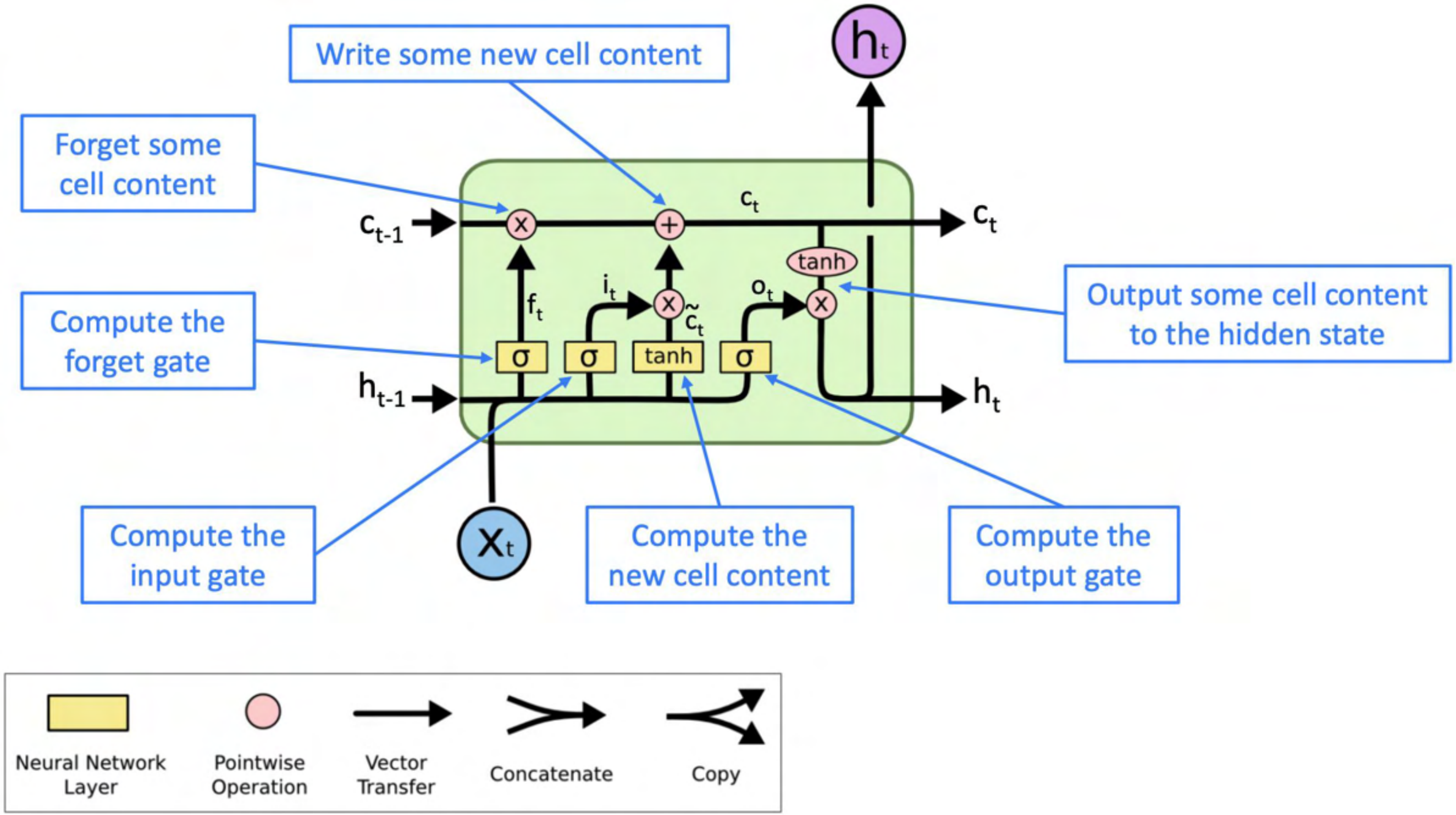
Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM



Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM: quick overview



Source: Lecture by Abigail See, [CS224n Lecture 7]

LSTM: quick overview

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

Sigmoid function: all gate values are between 0 and 1

$$\boldsymbol{f}^{(t)} = \boxed{\sigma}\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \boxed{\sigma}\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \boxed{\sigma}\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

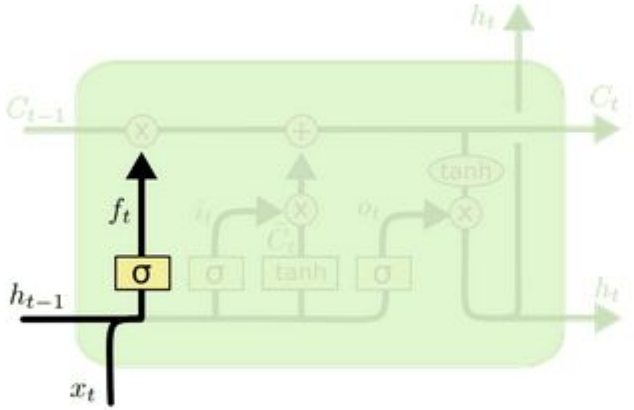
$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

All these are vectors of same length *n*

Gates are applied using element-wise product

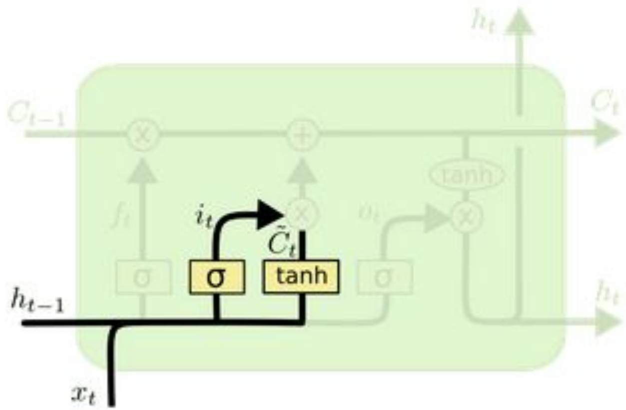
LSTM: quick overview



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM: quick overview

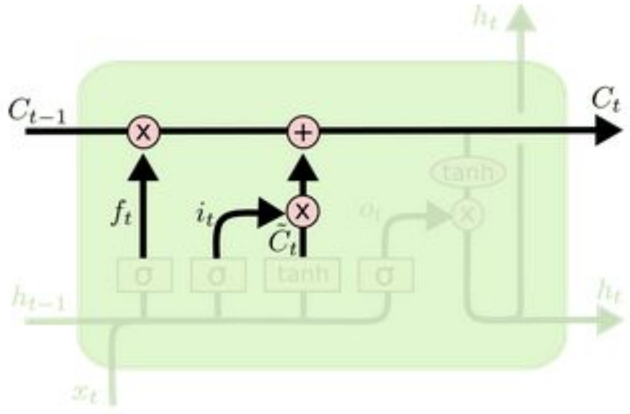


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

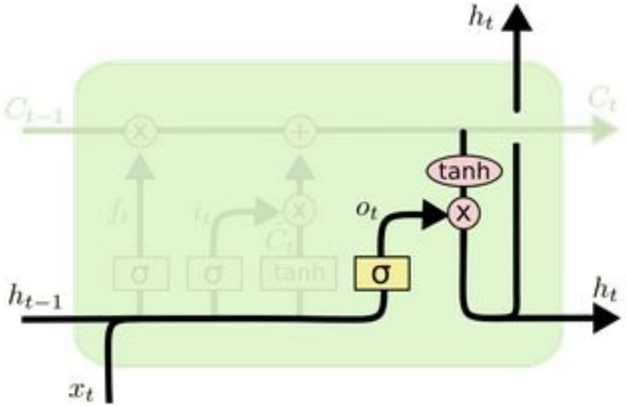
LSTM: quick overview



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM: quick overview



$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM: with formulas

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

Sigmoid function: all gate values are between 0 and 1

$$
\begin{aligned}
\boldsymbol{f}^{(t)} &= \boxed{\sigma}\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right) \\
\boldsymbol{i}^{(t)} &= \boxed{\sigma}\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right) \\
\boldsymbol{o}^{(t)} &= \boxed{\sigma}\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)
\end{aligned}
$$

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

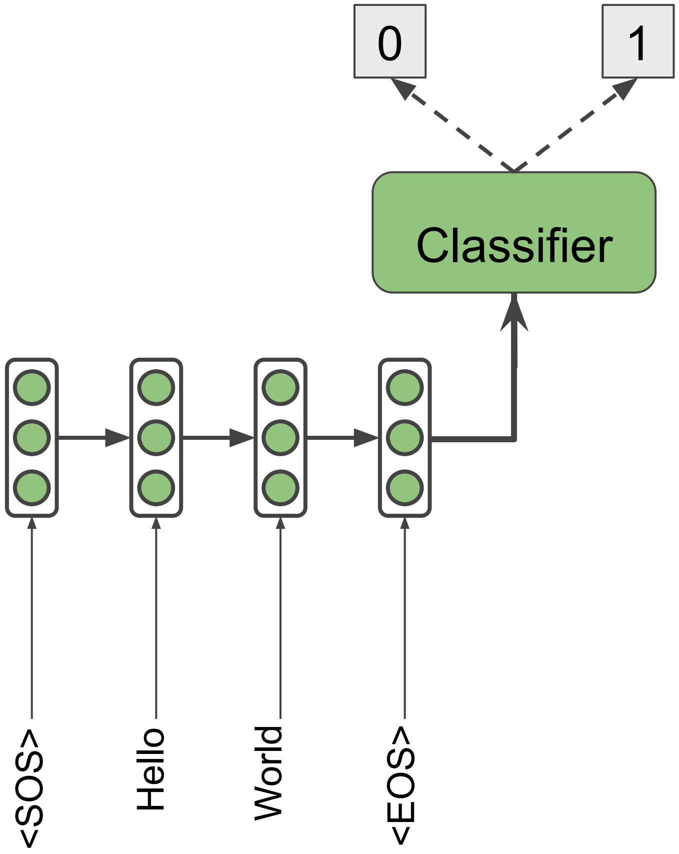
Hidden state: read ("output") some content from the cell

$$
\begin{aligned}
\tilde{\boldsymbol{c}}^{(t)} &= \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right) \\
\boldsymbol{c}^{(t)} &= \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)} \\
\boldsymbol{h}^{(t)} &= \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}
\end{aligned}
$$

All these are vectors of same length n

Gates are applied using element-wise product

RNN as encoder for sequential data



RNNs can be used to encode an input sequence in a fixed size vector.

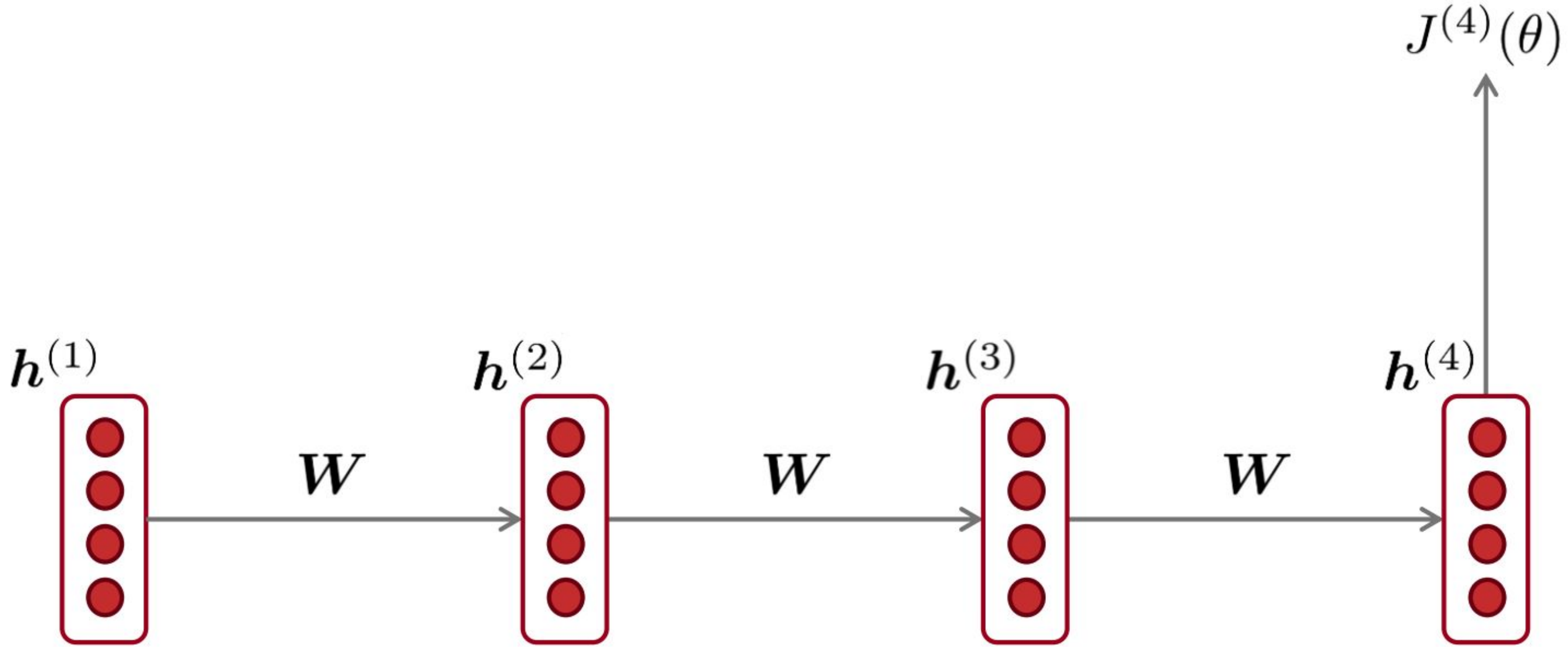
This vector can be treated as a representation of input sequence.

Example problem: PoS tagging

Pred. Tag	Actual Tag	Correct?	Token
PUNCT	PUNCT	✔	[
DET	DET	✔	this
NOUN	NOUN	✔	killing
ADP	ADP	✔	of
DET	DET	✔	a
ADJ	ADJ	✔	respected
NOUN	NOUN	✔	cleric
AUX	AUX	✔	will
AUX	AUX	✔	be
VERB	VERB	✔	causing
PRON	PRON	✔	us
NOUN	NOUN	✔	trouble
ADP	ADP	✔	for
NOUN	NOUN	✔	years
PART	PART	✔	to
VERB	VERB	✔	come
PUNCT	PUNCT	✔	.
PUNCT	PUNCT	✔	]

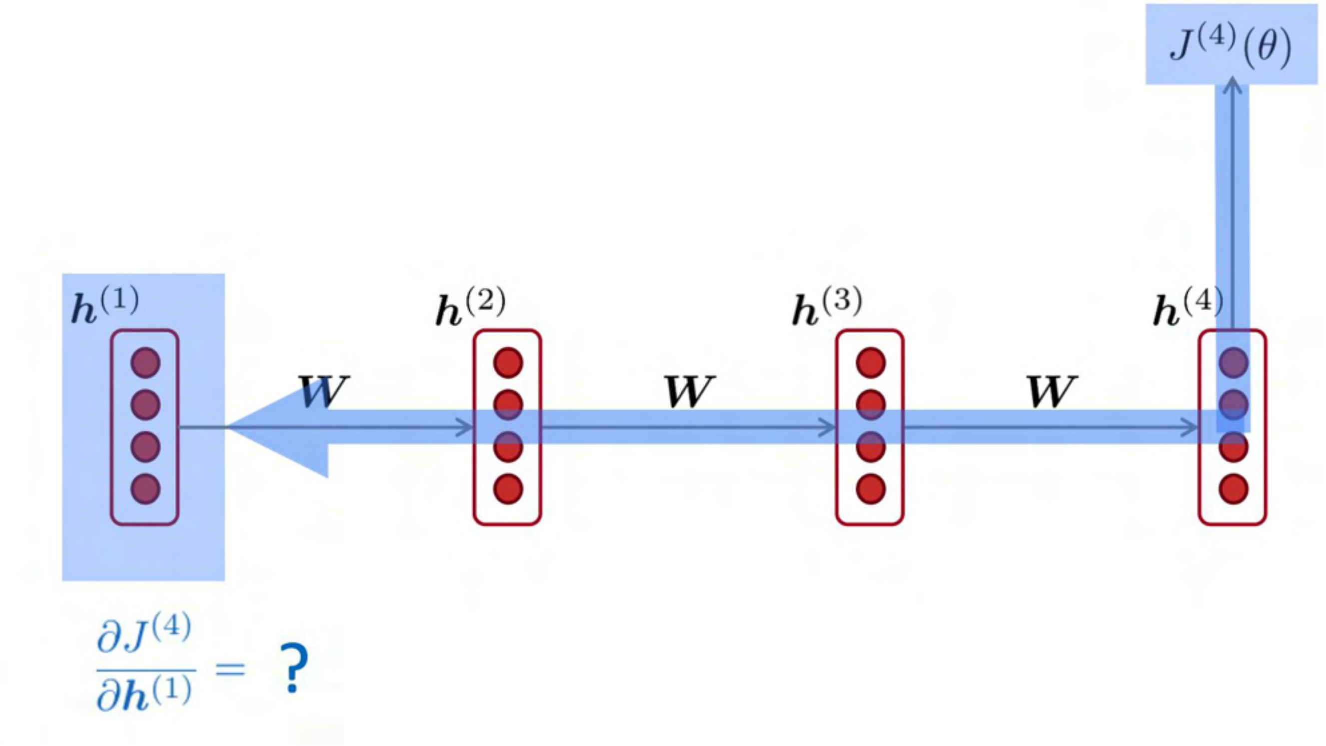
- PoS tagging can be performed using
  - Rule-based taggers
  - Dynamic programming
  - Models based on CRF (Conditional Random Field)
  - Neural Networks
  - etc.

Vanishing gradient problem



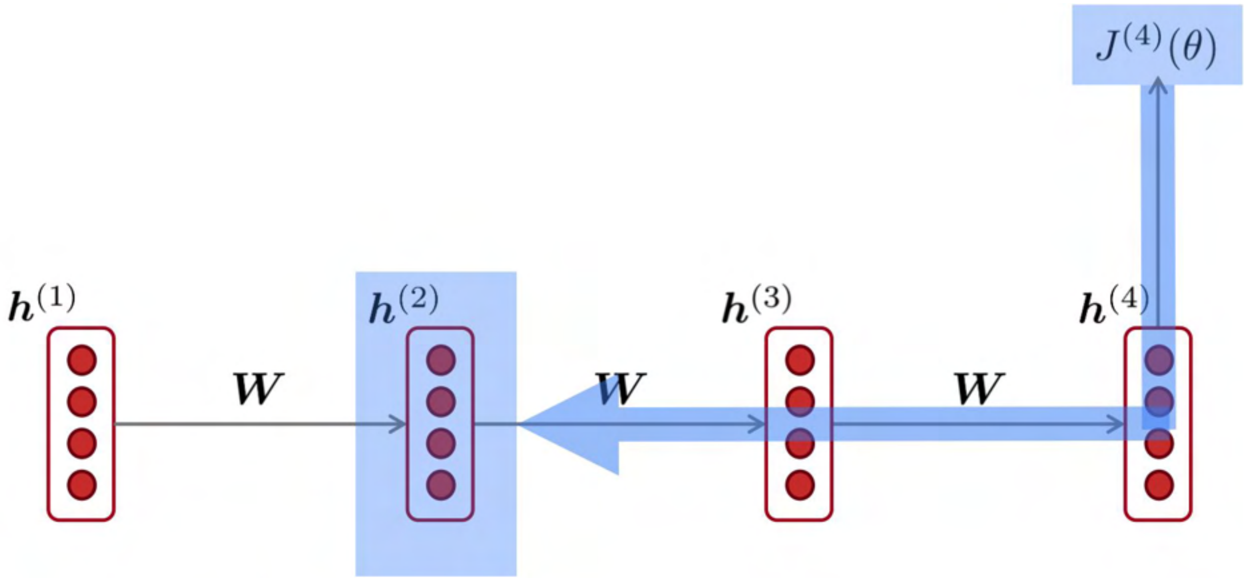
Based on: Lecture by Abigail See, [CS224n Lecture 7]

Vanishing gradient problem



Based on: Lecture by Abigail See, [CS224n Lecture 7]

Vanishing gradient problem

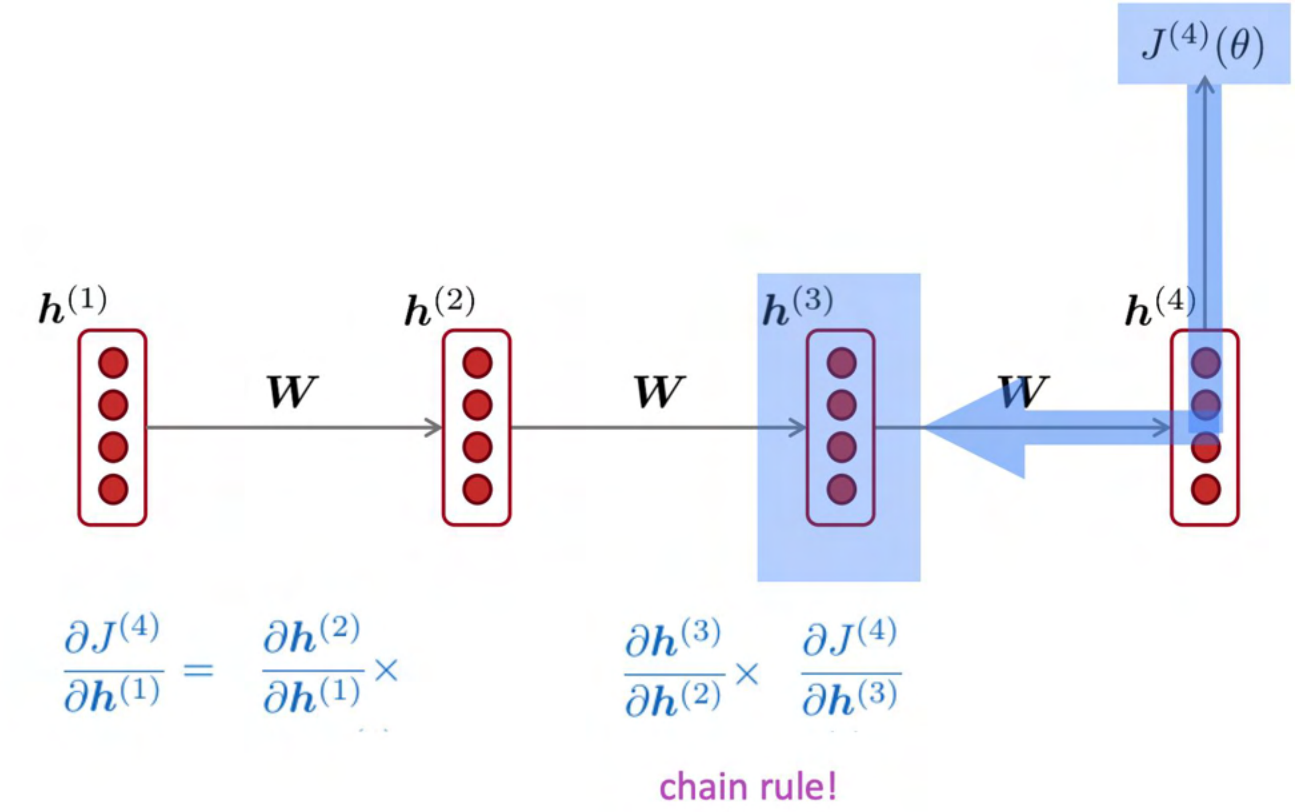


$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial J^{(4)}}{\partial h^{(2)}}$$

chain rule!

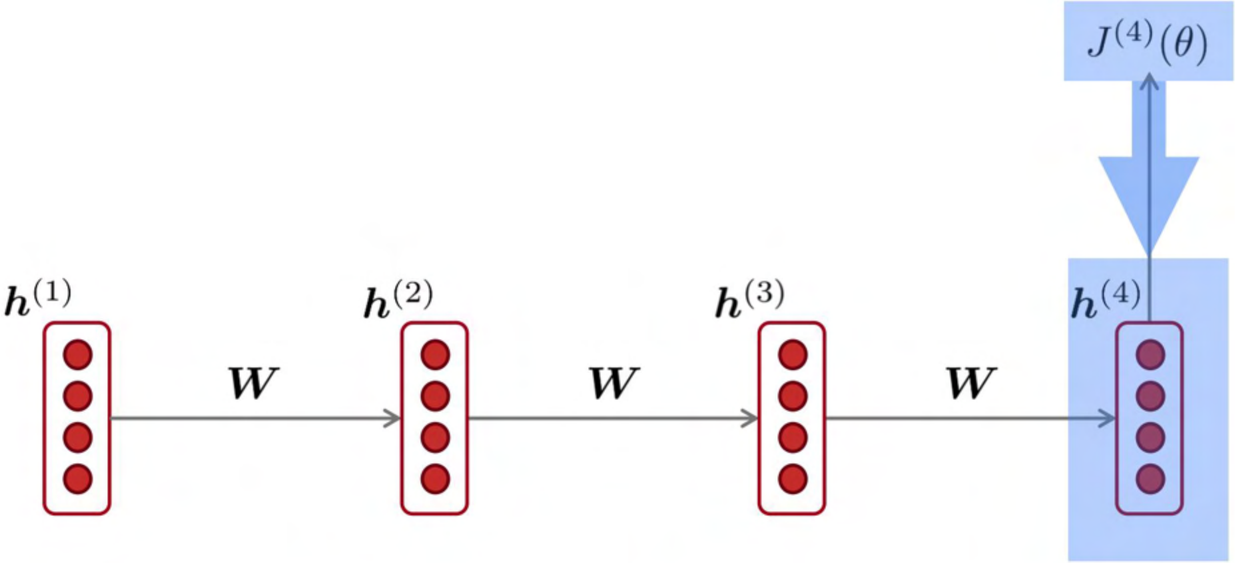
Based on: Lecture by Abigail See, [CS224n Lecture 7]

Vanishing gradient problem



Based on: Lecture by Abigail See, [CS224n Lecture 7]

Vanishing gradient problem

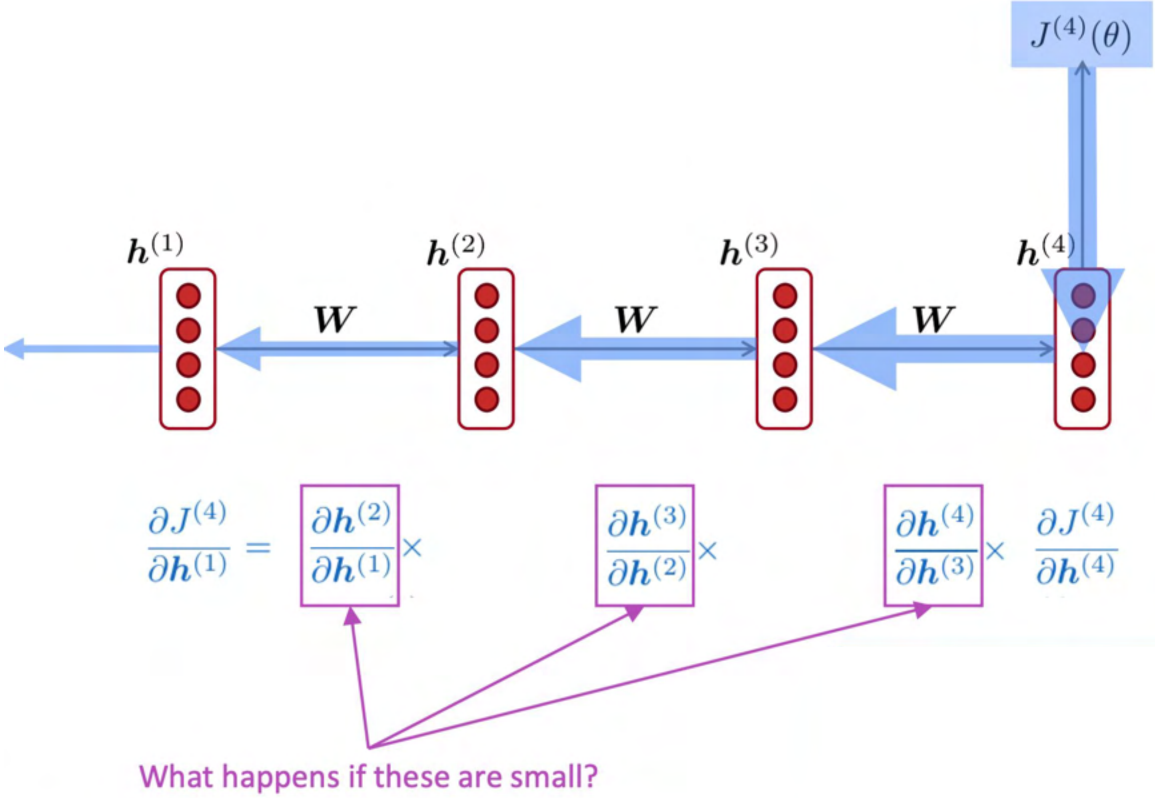


$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \times \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(4)}}$$

chain rule!

Based on: Lecture by Abigail See, CS224n Lecture 7

Vanishing gradient problem



Vanishing gradient problem:

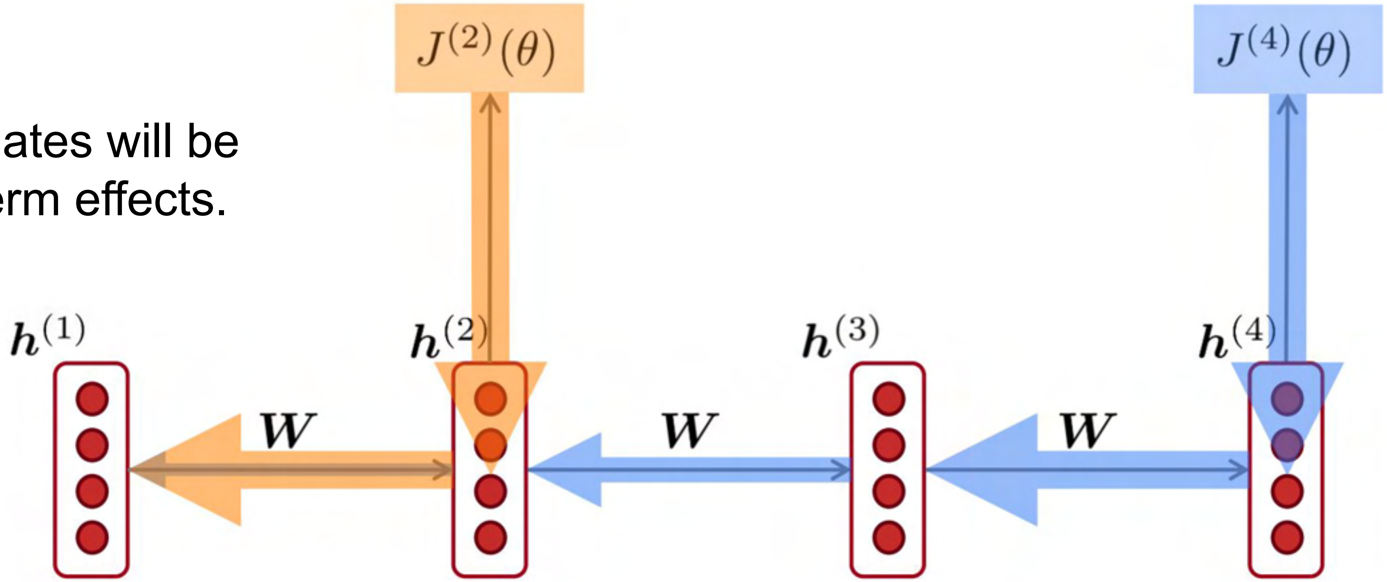
When the derivatives are small, the gradient signal gets smaller and smaller as it backpropagates further

More info: "On the difficulty of training recurrent neural networks", Pascanu et al, 2013
http://proceedings.mlr.press/v28/pascanu13.pdf

Vanishing gradient problem

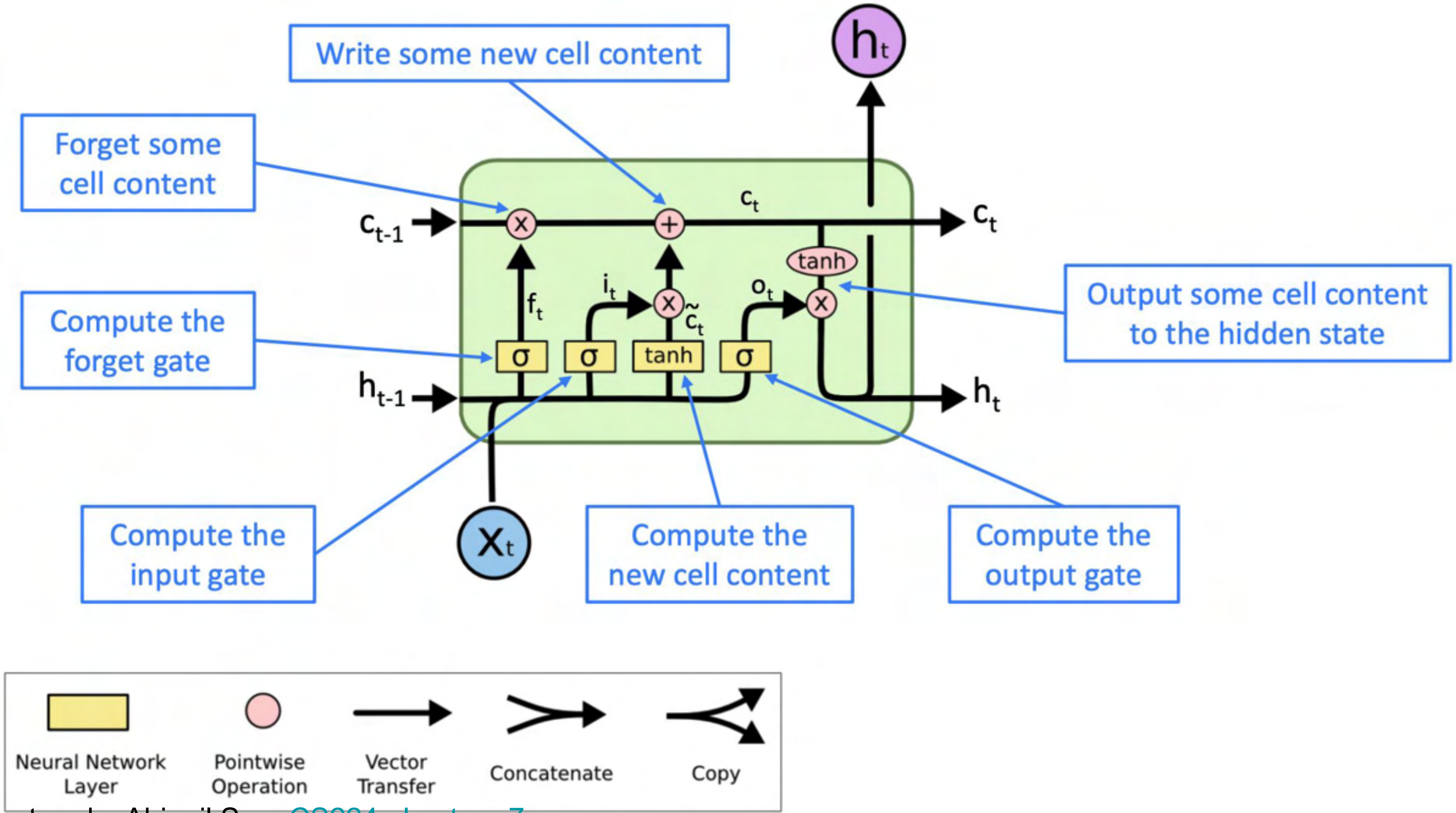
Gradient signal from far away is lost because it's much smaller than from close-by.

So model weights updates will be based only on short-term effects.



Based on: Lecture by Abigail See, CS224n Lecture 7

Vanishing gradient: LSTM



Based on: Lecture by Abigail See, [CS224n Lecture 7]

Vanishing gradient: LSTM

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

Sigmoid function: all gate values are between 0 and 1

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

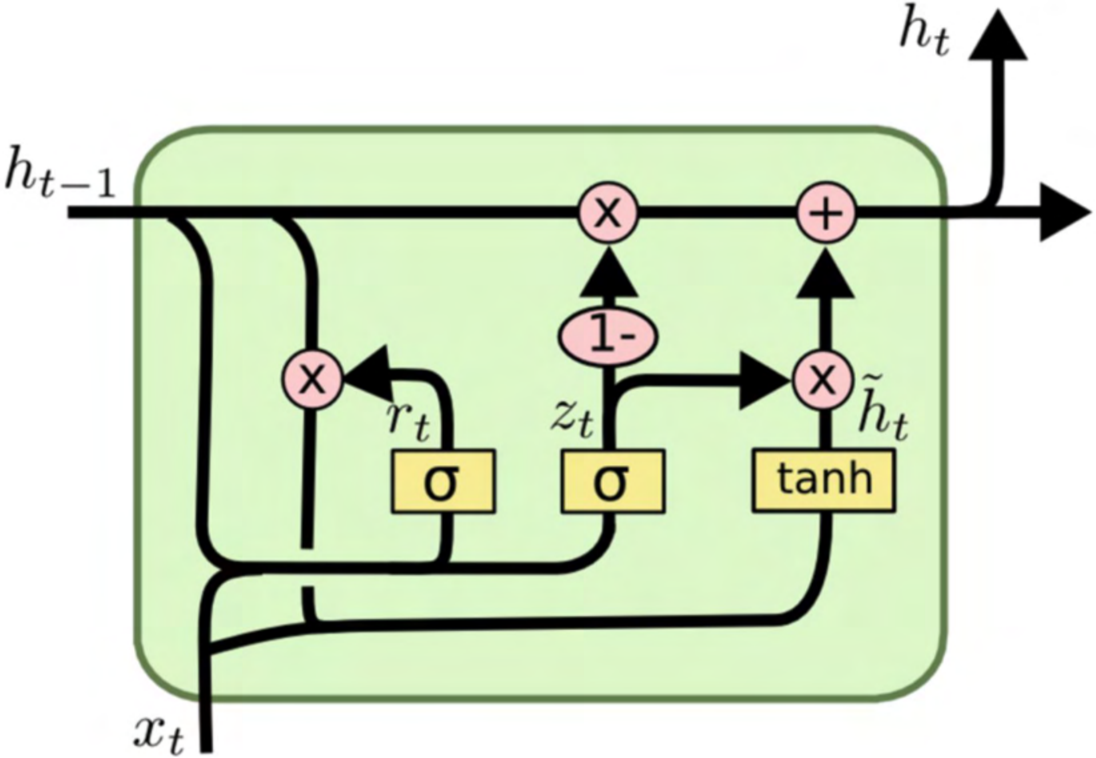
$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

All these are vectors of same length *n*

Gates are applied using element-wise product

Based on: Lecture by Abigail See, CS224n Lecture 7

Vanishing gradient: GRU



Vanishing gradient: GRU

Update gate: controls what parts of hidden state are updated vs preserved

Reset gate: controls what parts of previous hidden state are used to compute new content

$$\boldsymbol{u}^{(t)} = \sigma\left(\boldsymbol{W}_u \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_u \boldsymbol{x}^{(t)} + \boldsymbol{b}_u\right)$$

$$\boldsymbol{r}^{(t)} = \sigma\left(\boldsymbol{W}_r \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_r \boldsymbol{x}^{(t)} + \boldsymbol{b}_r\right)$$

New hidden state content: reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

Hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh\left(\boldsymbol{W}_h(\boldsymbol{r}^{(t)} \circ \boldsymbol{h}^{(t-1)}) + \boldsymbol{U}_h \boldsymbol{x}^{(t)} + \boldsymbol{b}_h\right)$$

$$\boldsymbol{h}^{(t)} = (1 - \boldsymbol{u}^{(t)}) \circ \boldsymbol{h}^{(t-1)} + \boldsymbol{u}^{(t)} \circ \tilde{\boldsymbol{h}}^{(t)}$$

How does this solve vanishing gradient?
Like LSTM, GRU makes it easier to retain info long-term (e.g. by setting update gate to 0)

Vanishing gradient: LSTM vs GRU

- LSTM and GRU are both great
  - GRU is quicker to compute and has fewer parameters than LSTM
  - There is no conclusive evidence that one consistently performs better than the other
  - LSTM is a good default choice (especially if your data has particularly long dependencies, or you have lots of training data)

Rule of thumb: start with LSTM, but switch to GRU if you want something more efficient

Vanishing gradient in non-RNN

Vanishing gradient is present in **all** deep neural networks

- Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small during backpropagation
- Lower levels are hard to train and are trained slower
- **Potential solution**:
  direct (or skip-) connections
  (just like in ResNet)

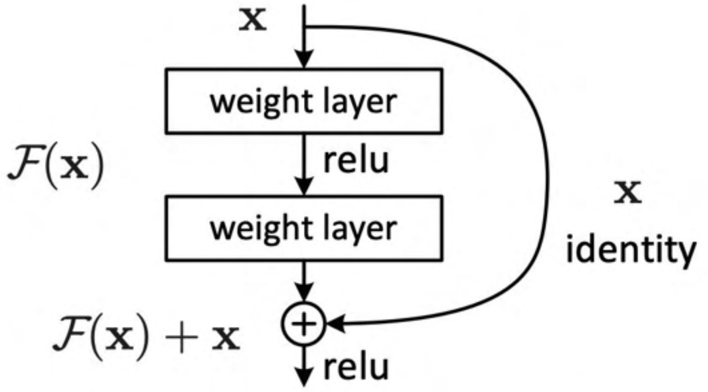


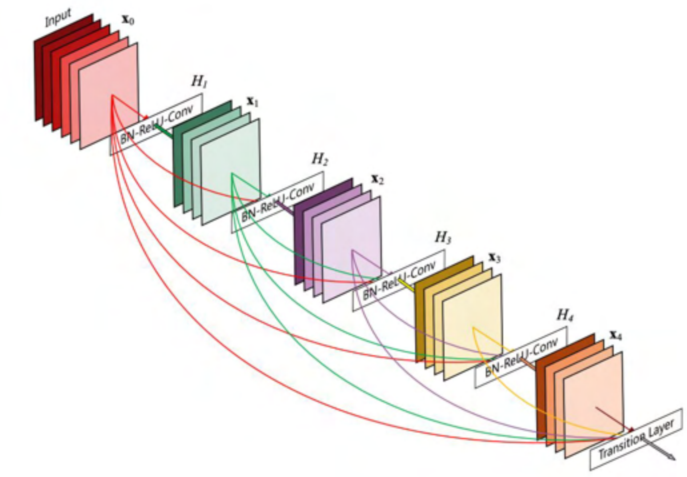
Figure 2. Residual learning: a building block.

Source: [Deep Residual Learning for Image Recognition, He et al, 2015]

Vanishing gradient in non-RNN

Vanishing gradient is present in **all** deep neural networks

- Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small during backpropagation
- Lower levels are hard to train and are trained slower
- **Potential solution**:
  dense connections
  (just like in DenseNet)



Source: [Densely Connected Convolutional Networks, Huang et al, 2017]

Vanishing gradient in non-RNN

Vanishing gradient is present in **all** deep neural networks

- Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small during backpropagation
- Lower levels are hard to train and are trained slower

Conclusion:

Though vanishing/exploding gradients are a general problem, RNNs are particularly unstable due to the repeated multiplication by the same weight matrix [Bengio et al, 1994]

Source: [Learning Long-Term Dependencies with Gradient Descent is Difficult, Bengio et al. 1994]

Exploding gradient problem

- If the gradient becomes too big, then the SGD update step becomes too big: $\theta^{new} = \theta^{old} - \overbrace{\alpha}^{\text{learning rate}} \underbrace{\nabla_\theta J(\theta)}_{\text{gradient}}$

- This can cause bad updates: we take too large a step and reach a bad parameter configuration (with large loss)

- In the worst case, this will result in Inf or NaN in your network (then you have to restart training from an earlier checkpoint)

Based on: Lecture by Abigail See, CS224n Lecture 7

Exploding gradient solution

- Gradient clipping: if the norm of the gradient is greater than some threshold, scale it down before applying SGD update

---

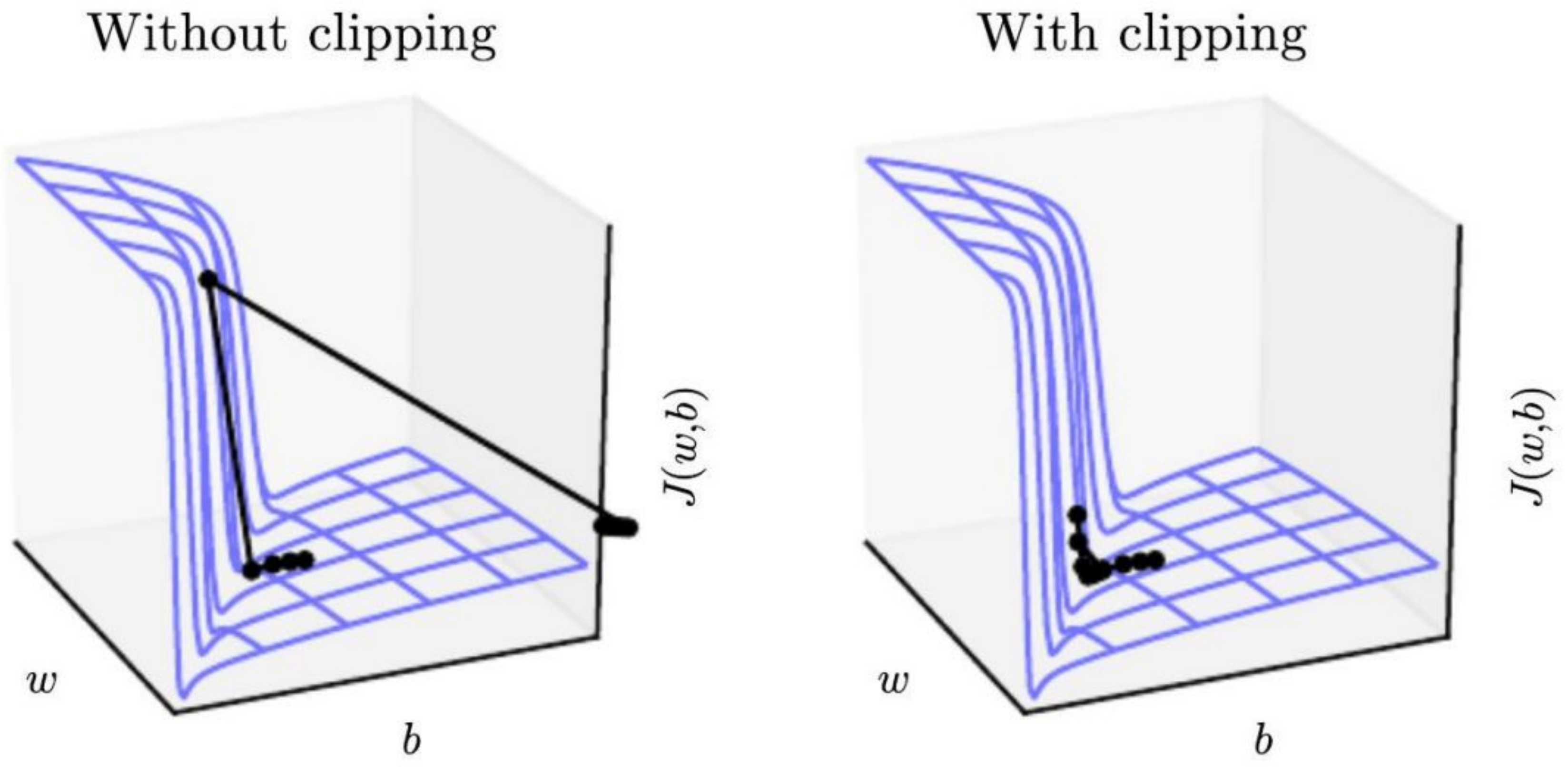
Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
    $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if

- Intuition: take a step in the same direction, but a smaller step

Based on: Lecture by Abigail See, [CS224n Lecture 7]

Exploding gradient solution

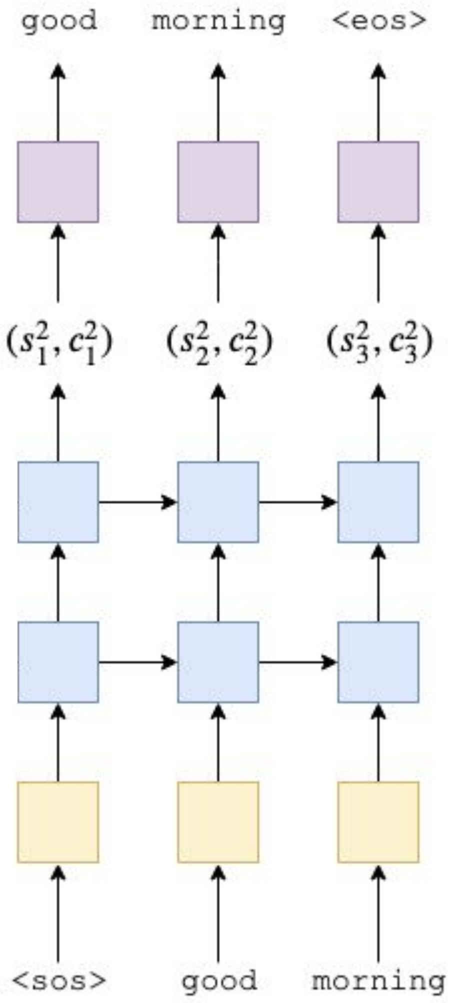


Based on: Lecture by Abigail See, [CS224n Lecture 7]

Tips

- RNN is a great choice for data with sequential structure
- Multi-layer RNN can also be of great use

- **Rule of thumb**: start with LSTM, but switch to GRU if you want something more efficient



Q & A