

# Lecture 7

## Vanishing gradient

## Advanced RNN layers

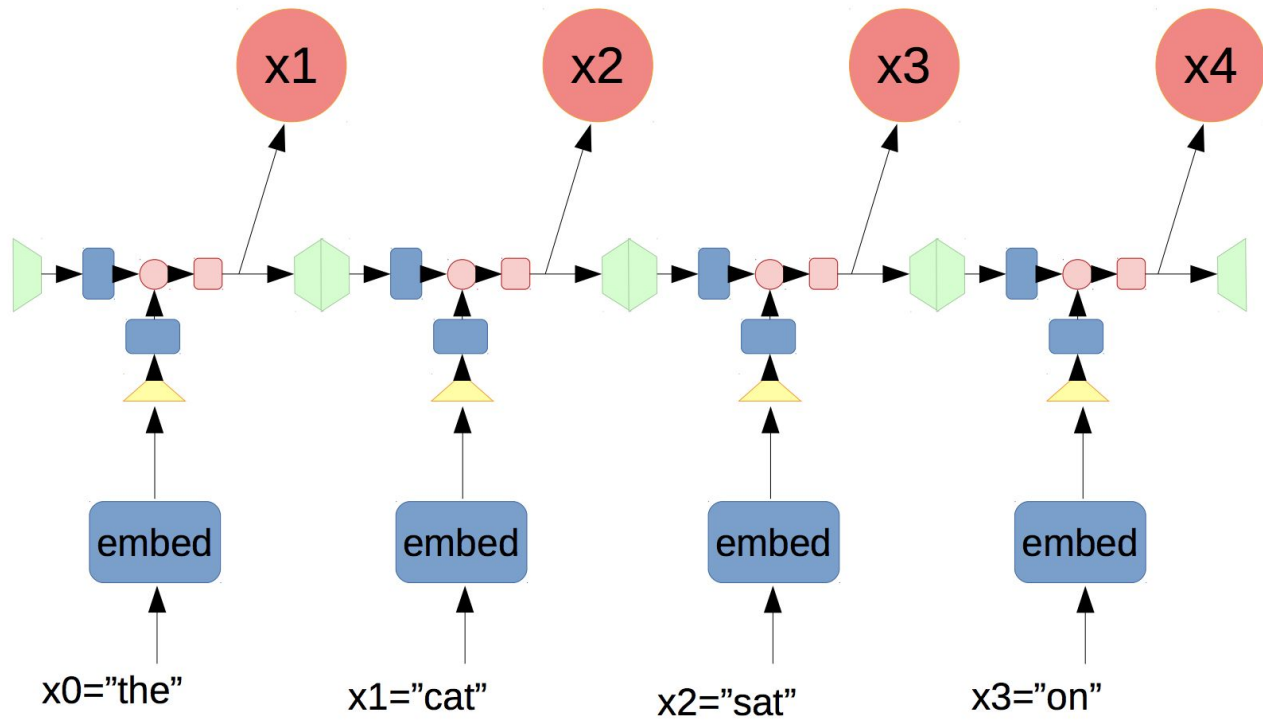
## CNNs for texts

**Vladislav Goncharenko**

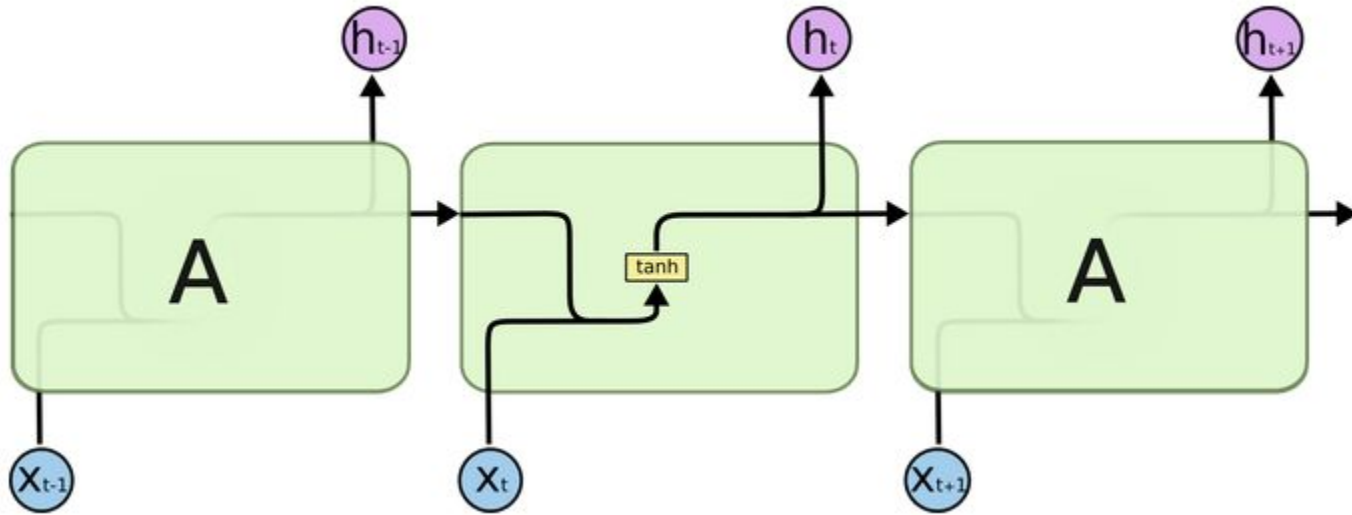
Moscow, 2021

- Simple RNN recap
- Complex RNN:
  - Vanishing gradient
  - Exploding gradient
  - LSTM/GRU
  - Gradient clipping
  - Skip connections
  - Residual networks as ensembles
- CNNs for text

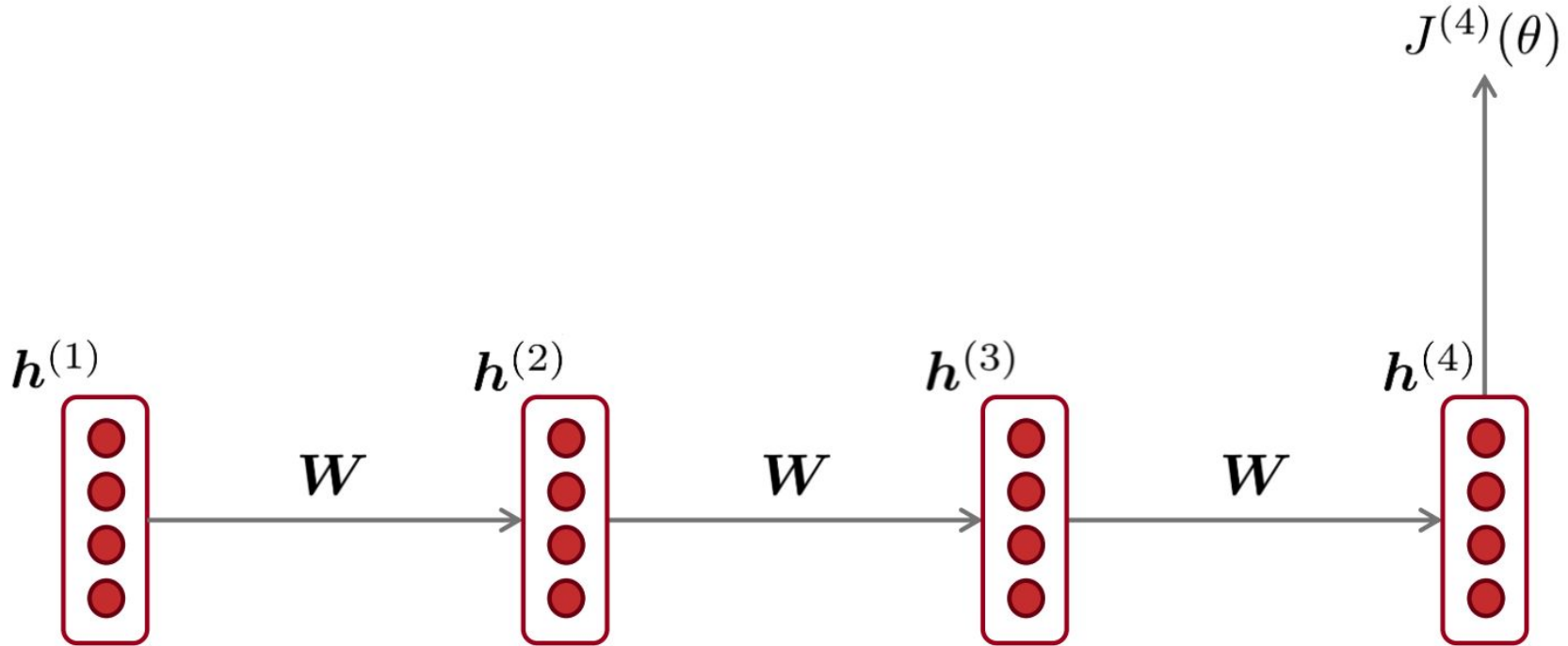
# Recap: RNN



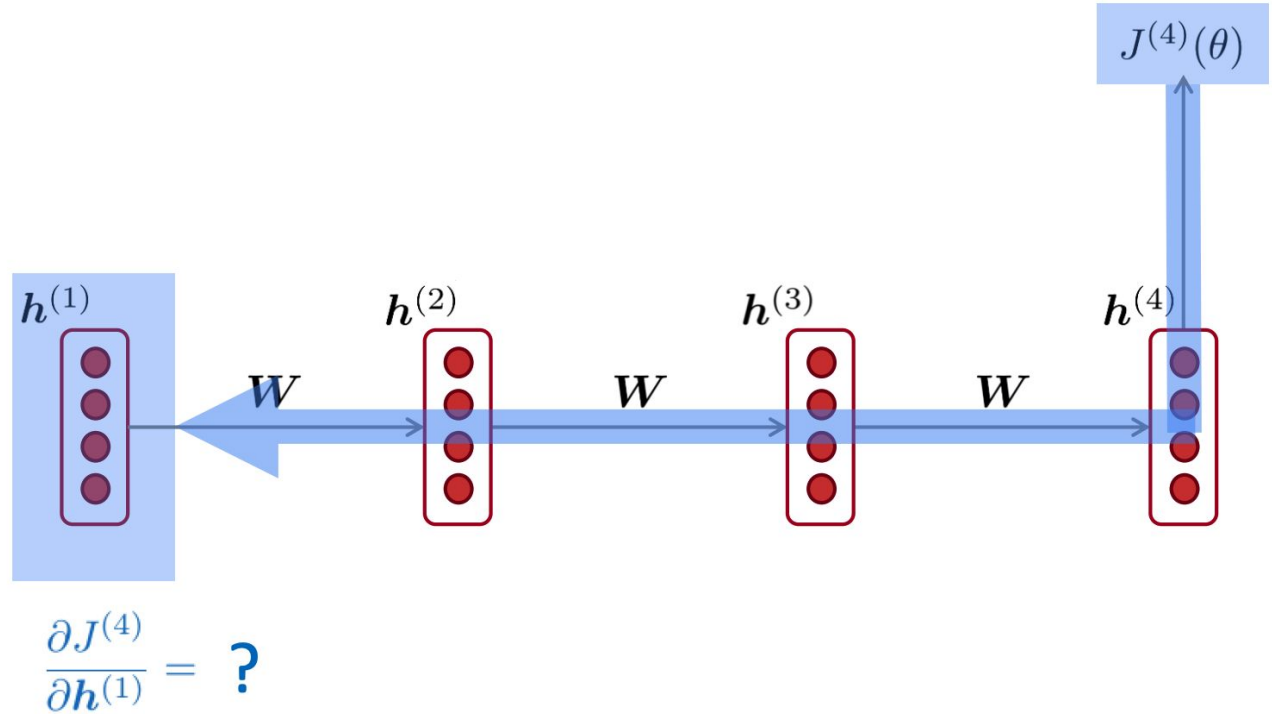
# Recap: Vanilla RNN



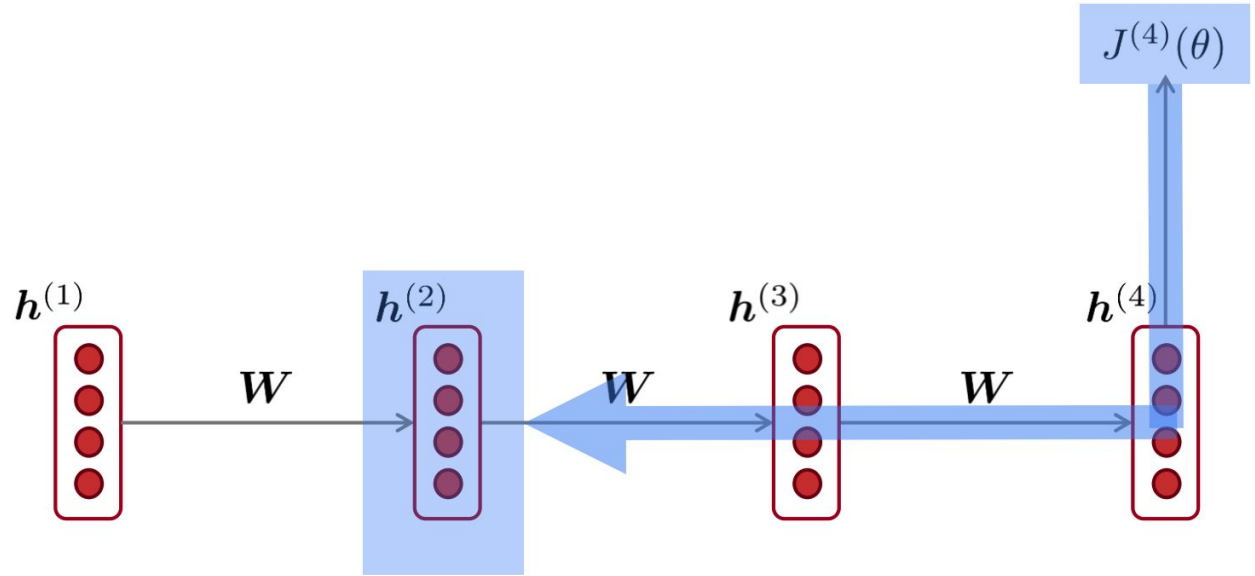
# Vanishing gradient problem



# Vanishing gradient problem



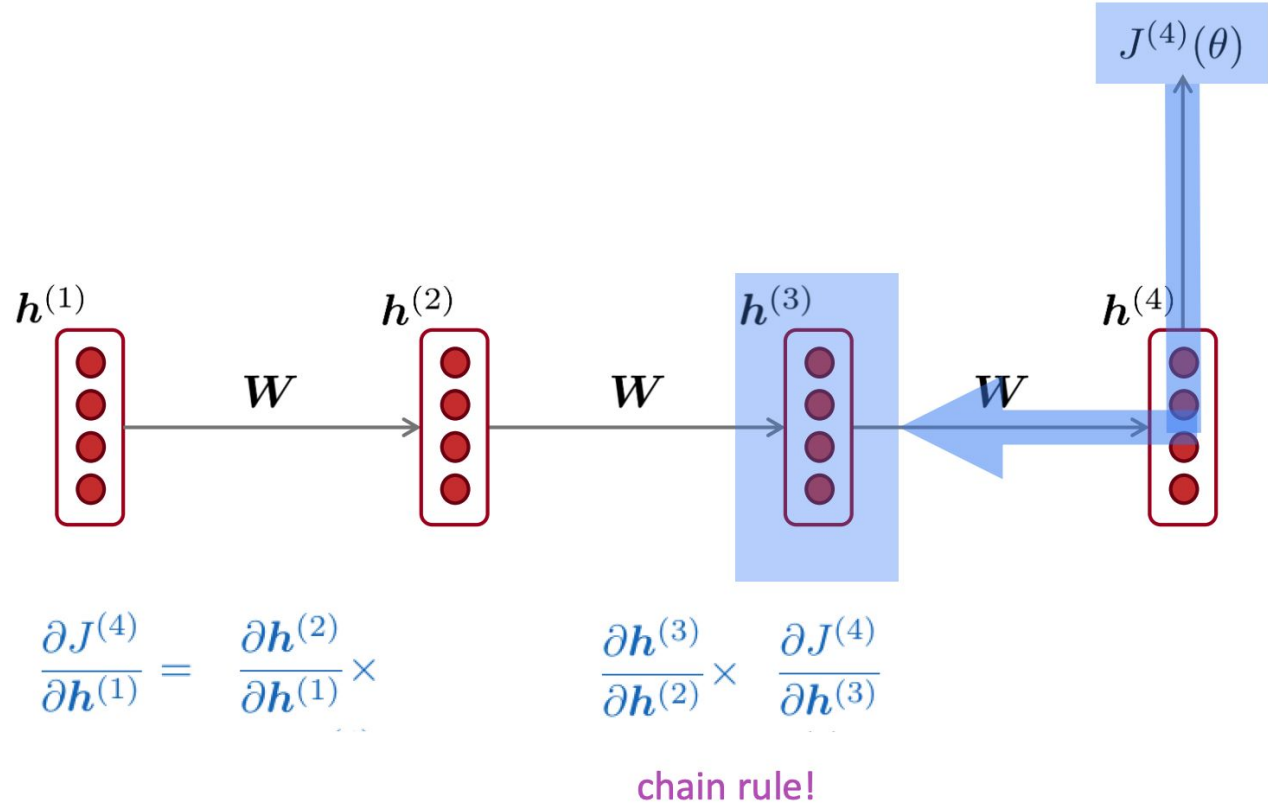
# Vanishing gradient problem



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

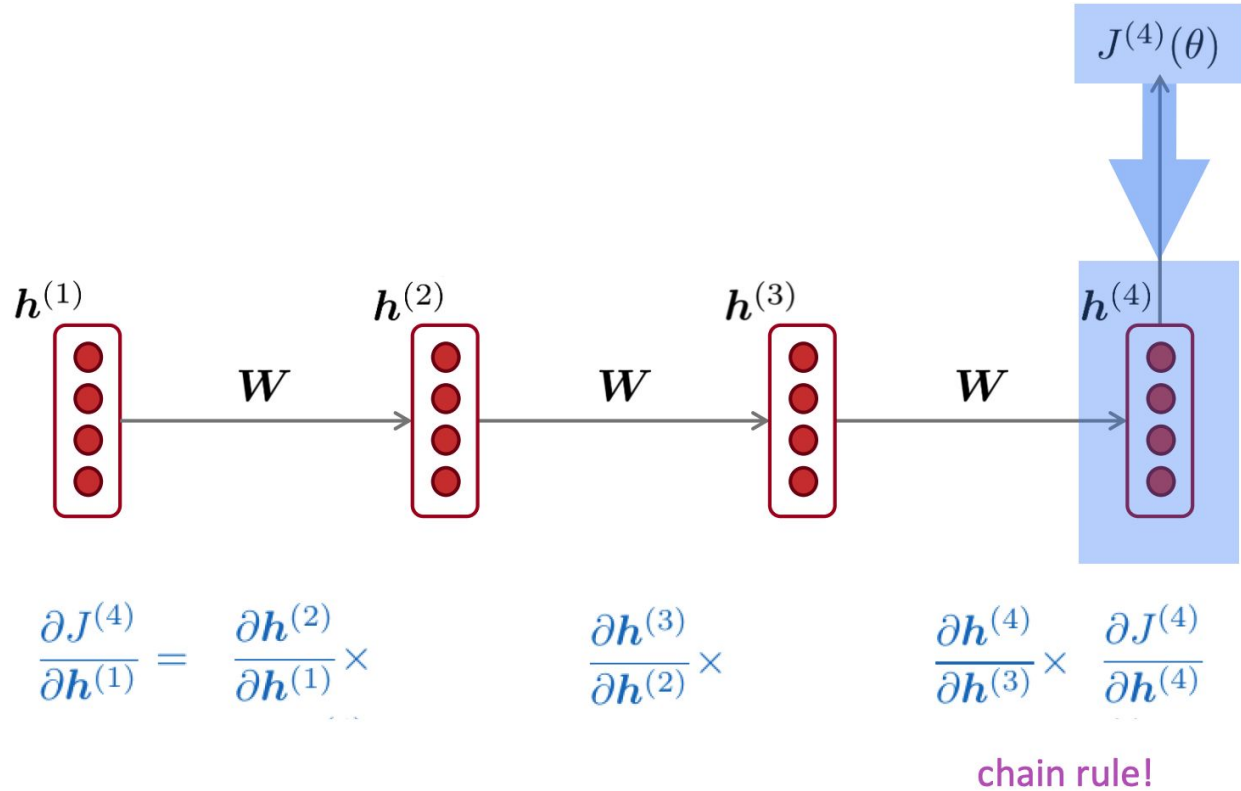
chain rule!

# Vanishing gradient problem

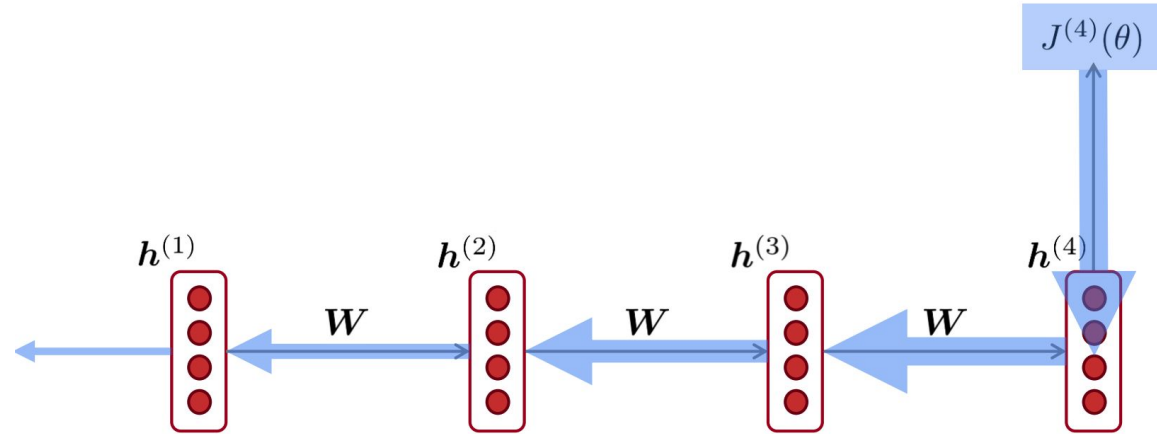




# Vanishing gradient problem



# Vanishing gradient problem



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

What happens if these are small?

Vanishing gradient problem:

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

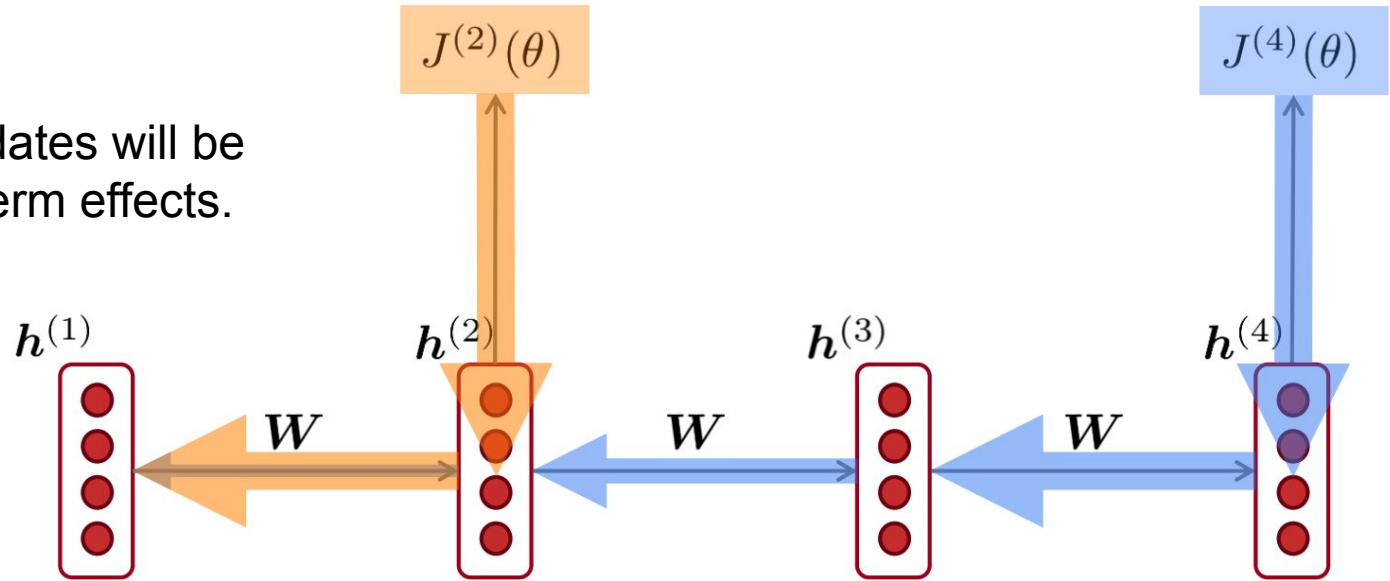
More info: "On the difficulty of training recurrent neural networks", Pascanu et al, 2013

<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.



# 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)

# 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

---

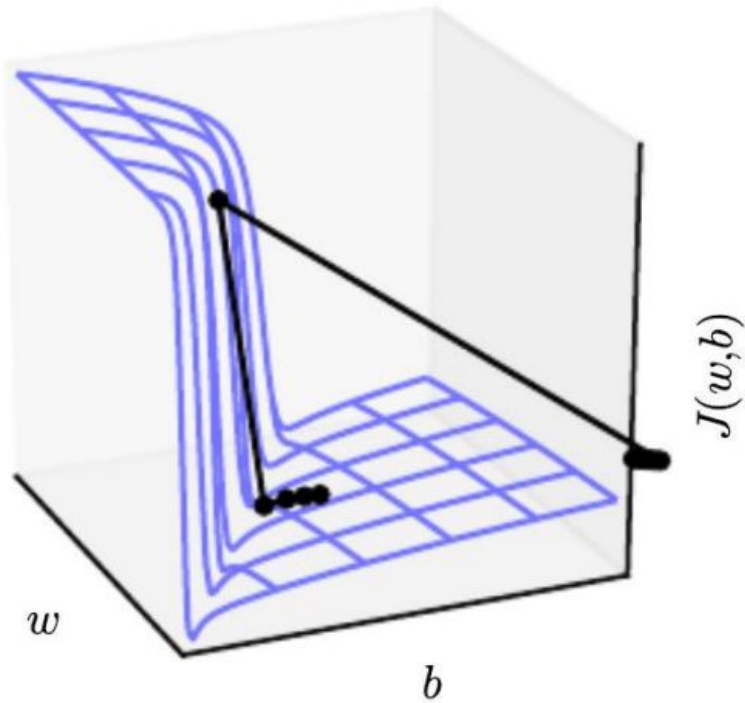
```
 $\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$   
if  $\|\hat{\mathbf{g}}\| \geq threshold$  then  
     $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$   
end if
```

---

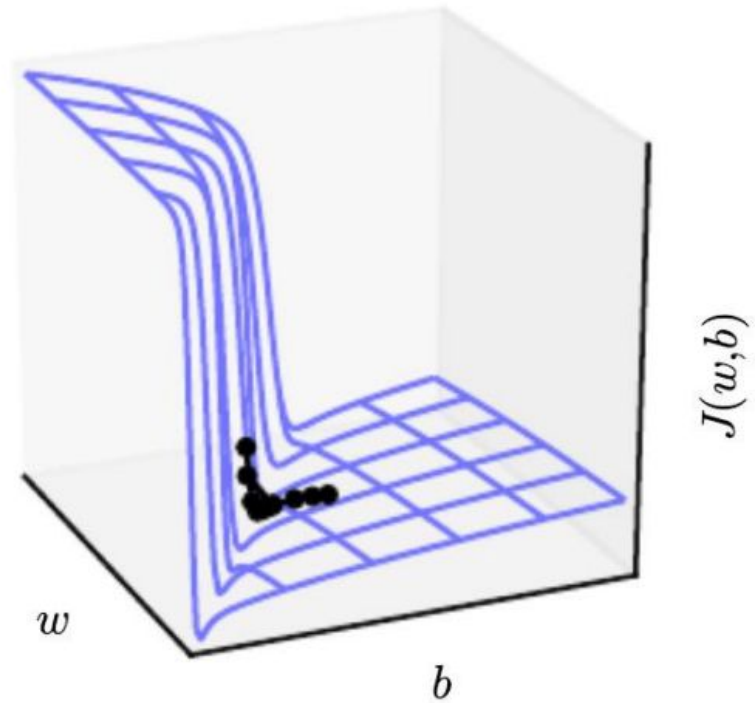
- Intuition: take a step in the same direction, but a smaller step

# Exploding gradient solution

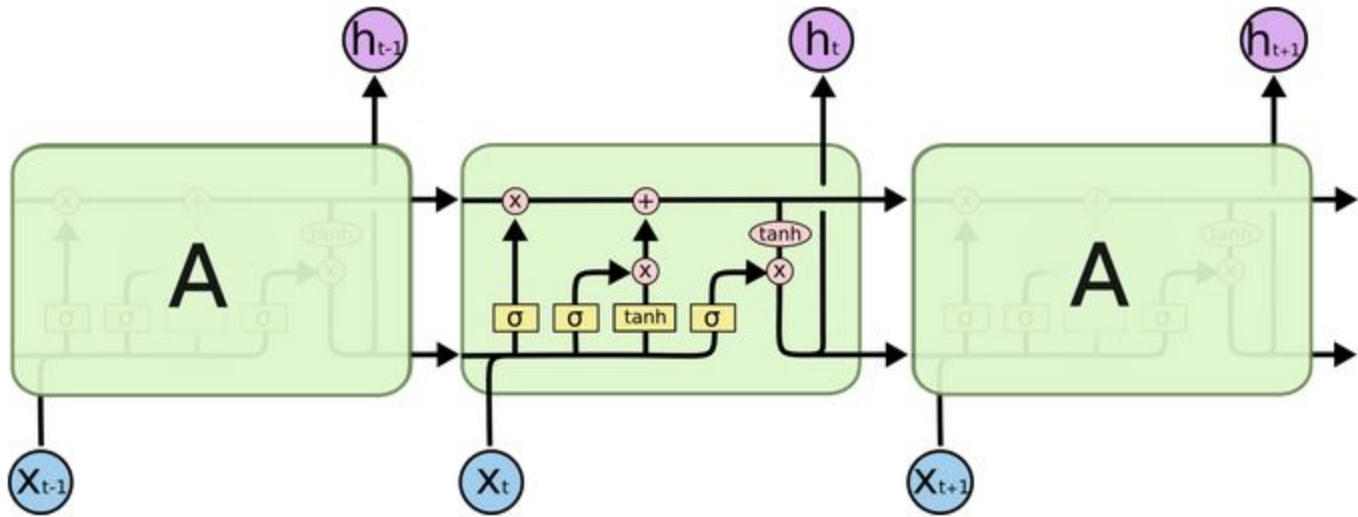
Without clipping



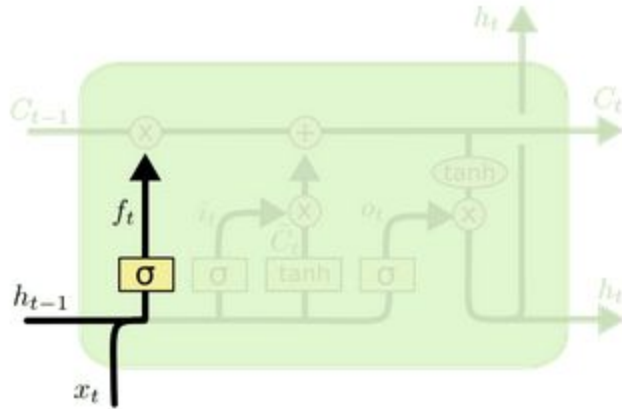
With clipping



# LSTM



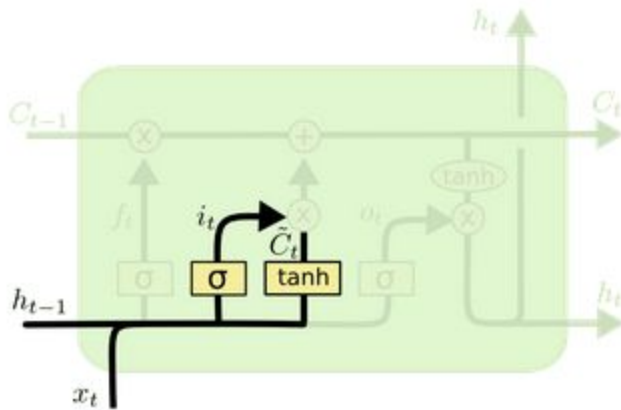
# LSTM: quick overview



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



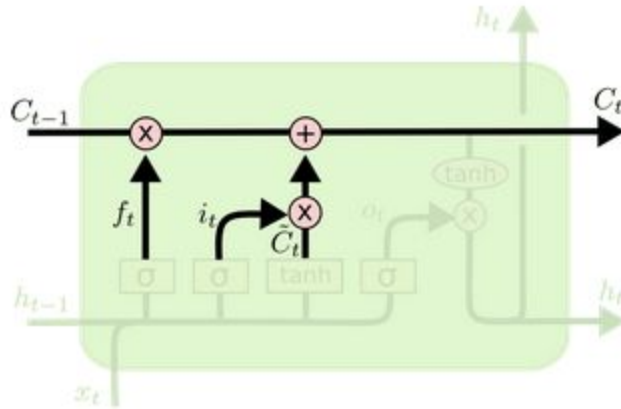
# LSTM: quick overview



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

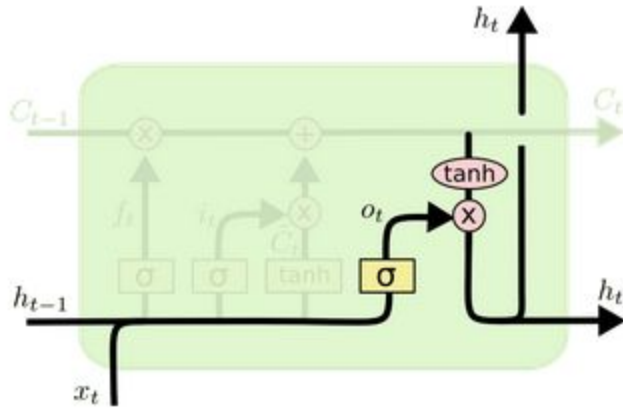
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

# LSTM: quick overview



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

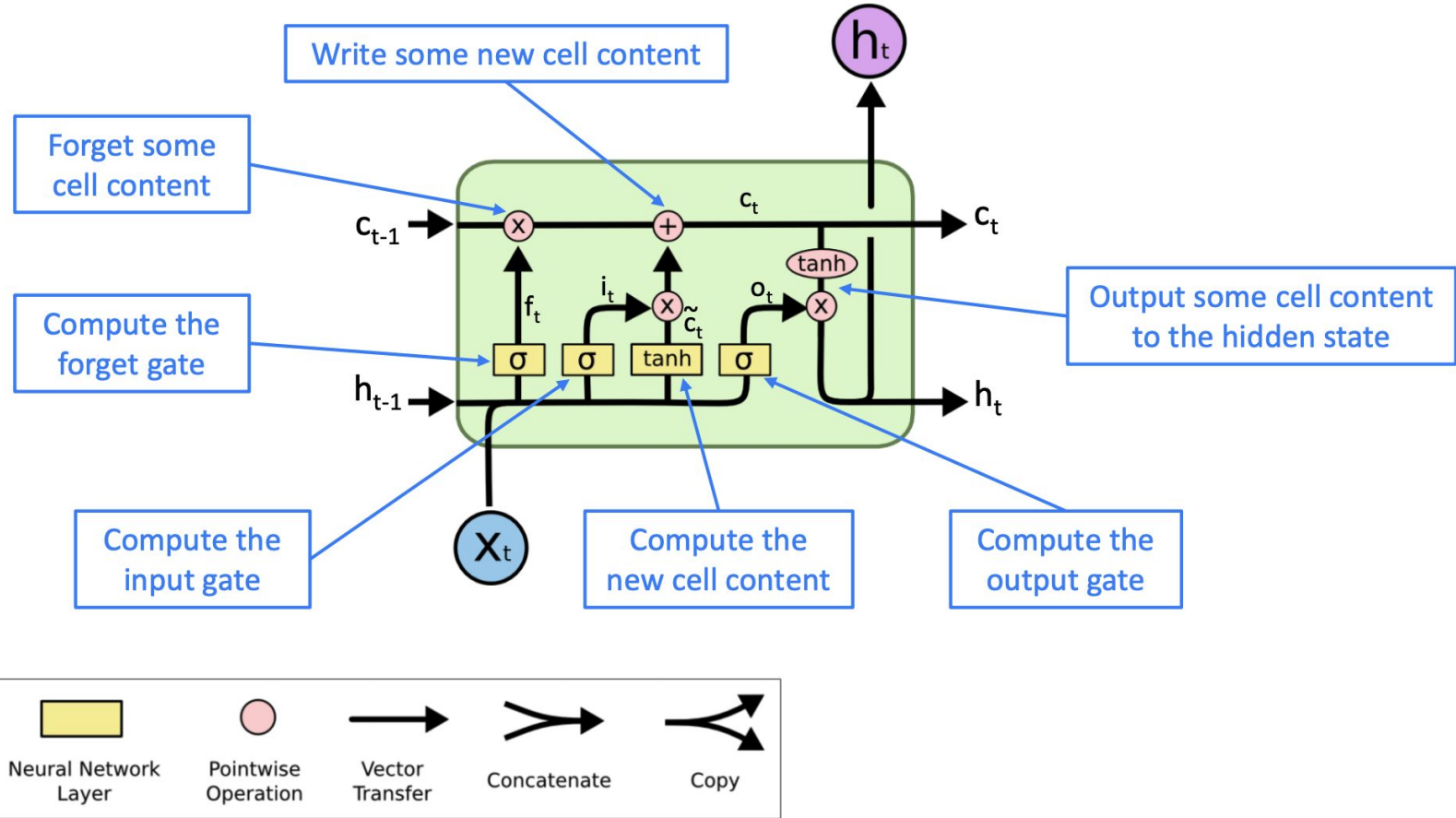
# LSTM: quick overview



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

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

# Vanishing gradient: LSTM



# 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

**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

**Sigmoid function:** all gate values are between 0 and 1

$$f^{(t)} = \sigma \left( W_f h^{(t-1)} + U_f x^{(t)} + b_f \right)$$

$$i^{(t)} = \sigma \left( W_i h^{(t-1)} + U_i x^{(t)} + b_i \right)$$

$$o^{(t)} = \sigma \left( W_o h^{(t-1)} + U_o x^{(t)} + b_o \right)$$

$$\tilde{c}^{(t)} = \tanh \left( W_c h^{(t-1)} + U_c x^{(t)} + b_c \right)$$

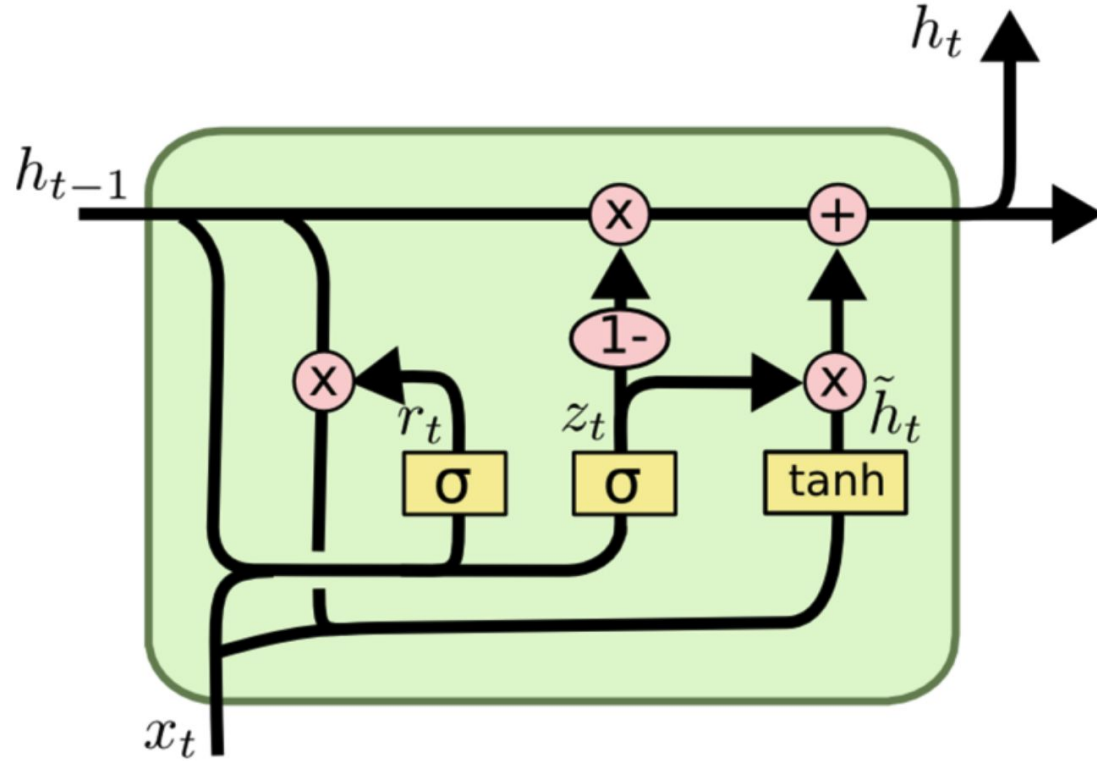
$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$$

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

All these are vectors of same length  $n$

Gates are applied using element-wise product

# Vanishing gradient: GRU



# Vanishing gradient: GRU

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

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

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

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

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

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

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

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

**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 network architectures.

- 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)

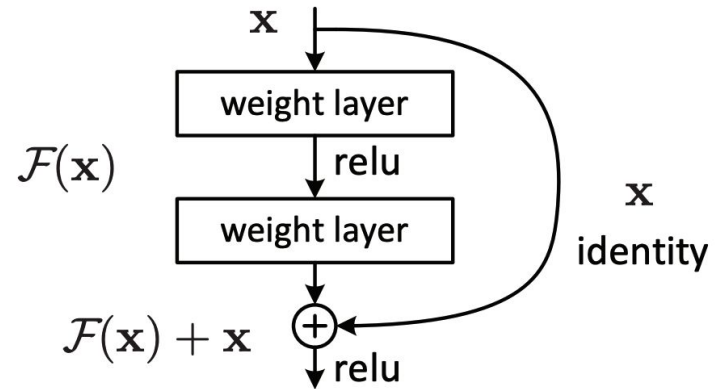
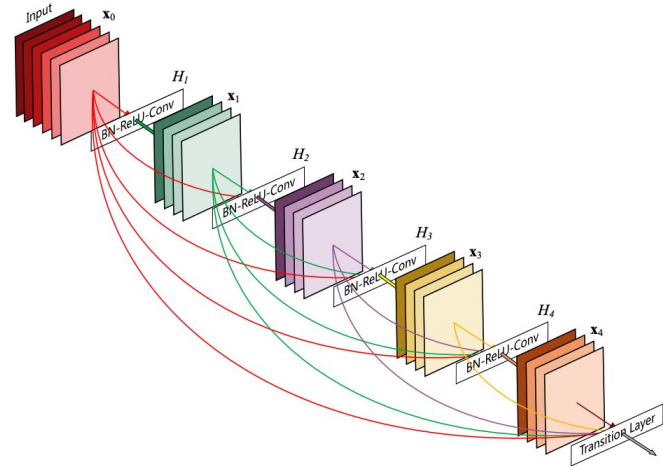


Figure 2. Residual learning: a building block.

# Vanishing gradient in non-RNN

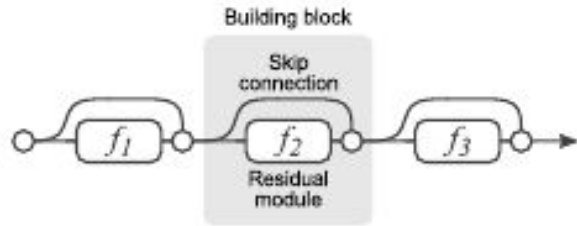
Vanishing gradient is present in **all** deep neural network architectures.

- 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)



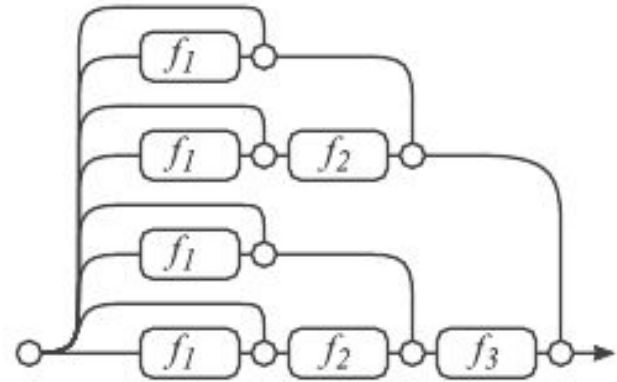
# Another view on ResNets and vanishing gradient

**“Residual Networks Behave Like Ensembles of Relatively Shallow Networks”**



(a) Conventional 3-block residual network

=



(b) Unraveled view of (a)

# Vanishing gradient in non-RNN

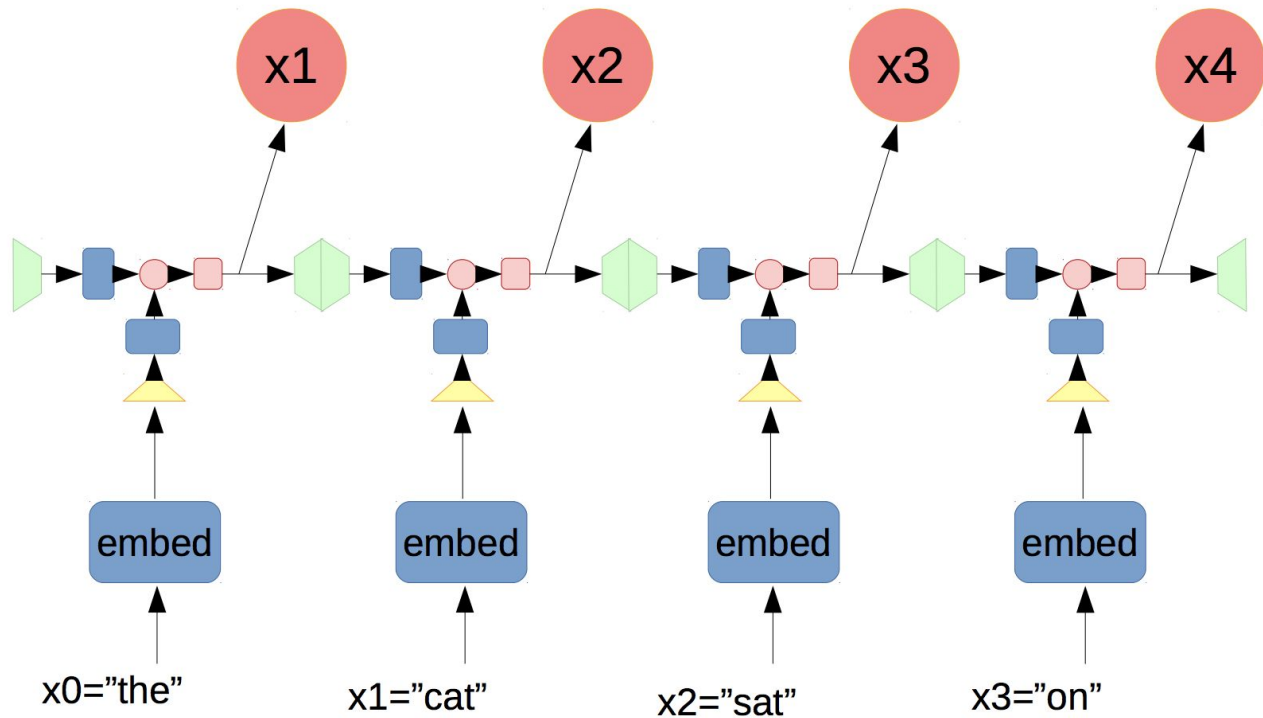
Vanishing gradient is present in **all** deep neural network architectures.

- 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(but not actually for that problem)**: dense connections (just like in DenseNet)

## 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]. Gradients magnitude drops exponentially with connection length.

# Recap: RNN

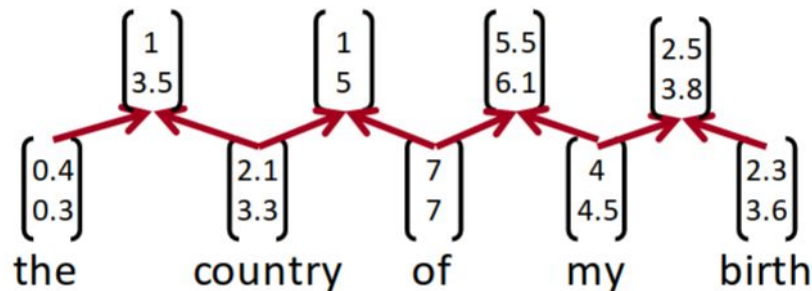


# From RNN to CNN

- RNN: Get compositional vectors for grammatical phrases only
- CNN: What if we compute vectors for every possible phrase?
  - Example: “*the country of my birth*” computes vectors for:
    - *the country, country of, of my, my birth, the country of, country of my, of my birth, the country of my, country of my birth*
- Regardless of whether it is grammatical
- Wouldn't need parser
- Not very linguistically or cognitively plausible

# From RNN to CNN

- Imagine using only bigrams



- Same operation as in RNN, but for every pair

$$p = \tanh \left( W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b \right)$$

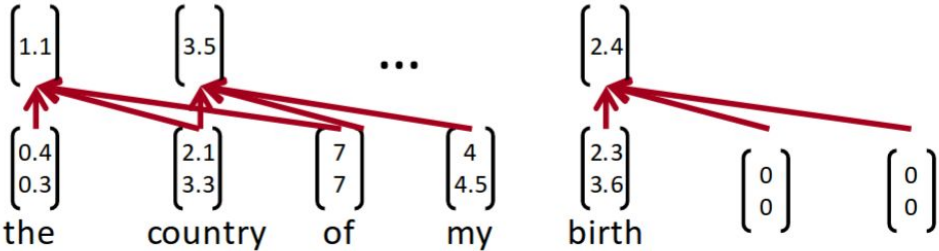
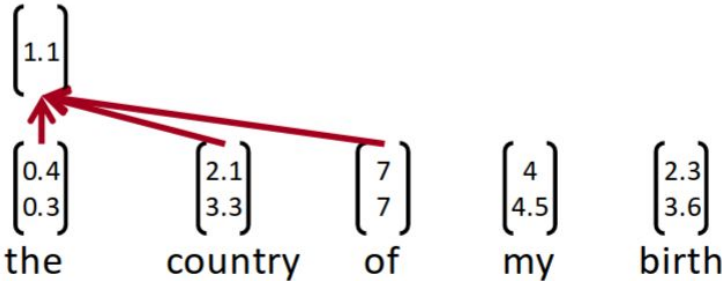
- Can be interpreted as convolution over the word vectors

# One layer CNN

- Simple convolution + pooling
- Window size may be different (2 or more)
- The feature map based on bigrams:

$$\mathbf{c} = [c_1, c_2, \dots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$

$$c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$$





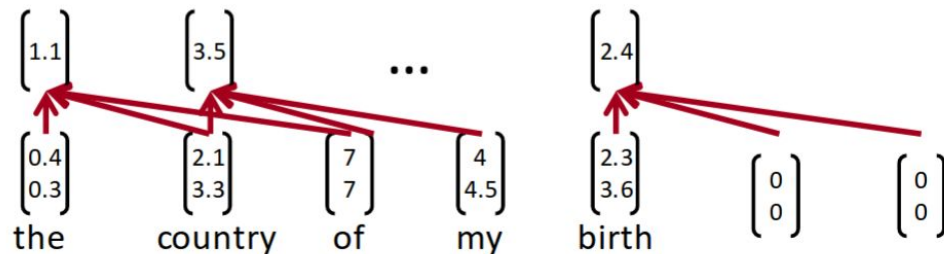
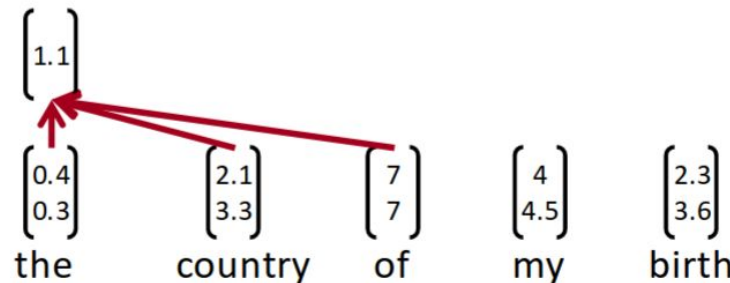
# One layer CNN

- Simple convolution + pooling
- Window size may be different (2 or more)
- The feature map based on bigrams:

$$\mathbf{c} = [c_1, c_2, \dots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$

What's next?

$$c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$$



# One layer CNN

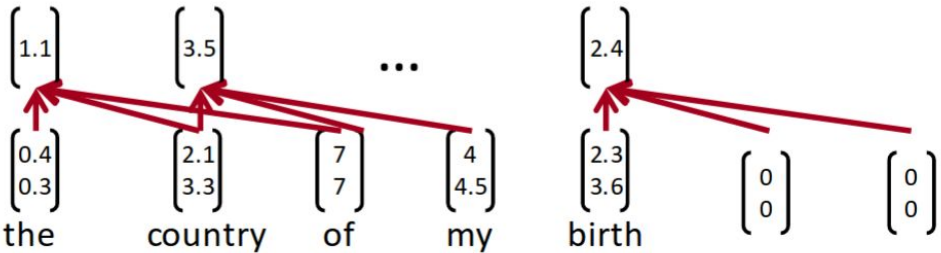
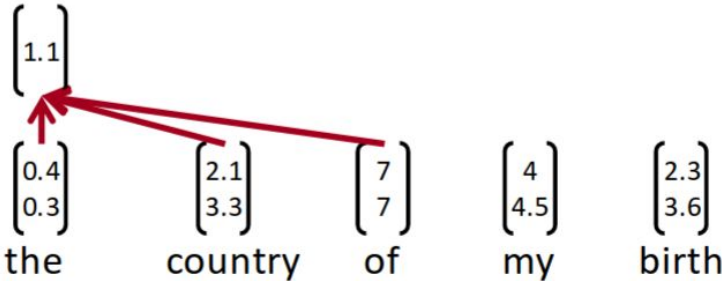
- Simple convolution + pooling
- Window size may be different (2 or more)
- The feature map based on bigrams:

$$\mathbf{c} = [c_1, c_2, \dots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$

$$c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$$

What's next?

We need more features!



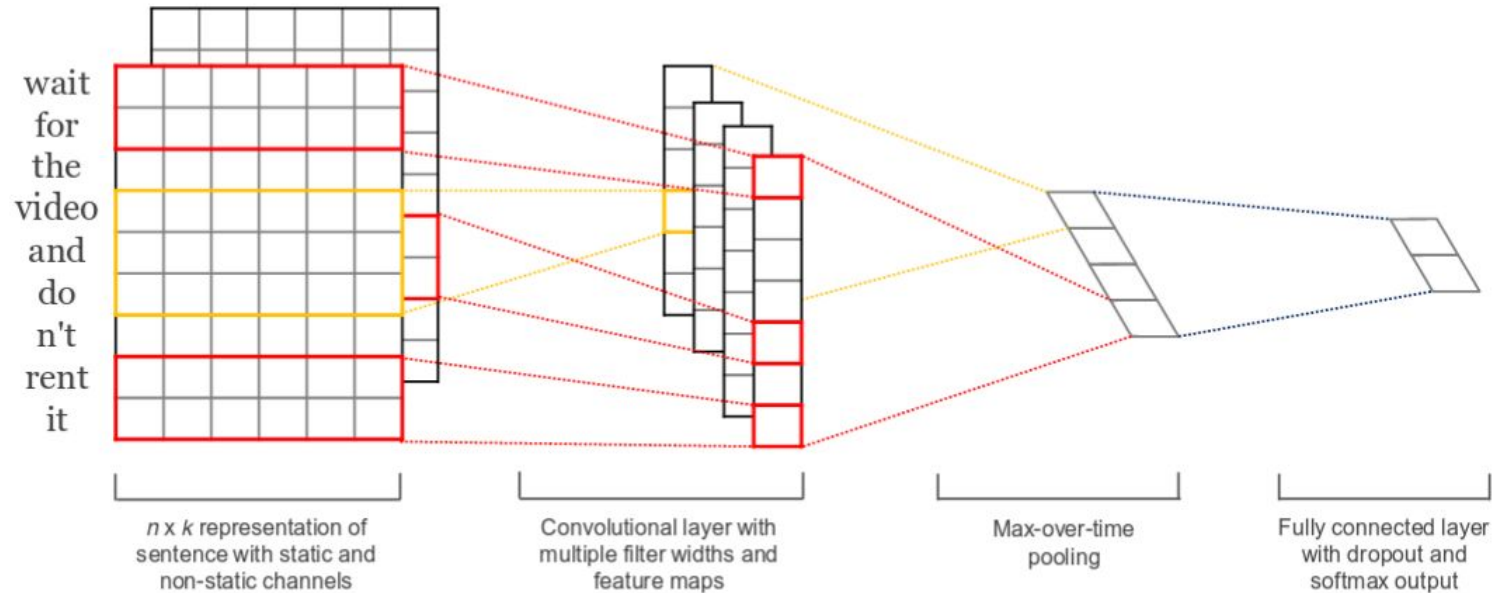
# One layer CNN

- Feature representation is based on some applied filter:

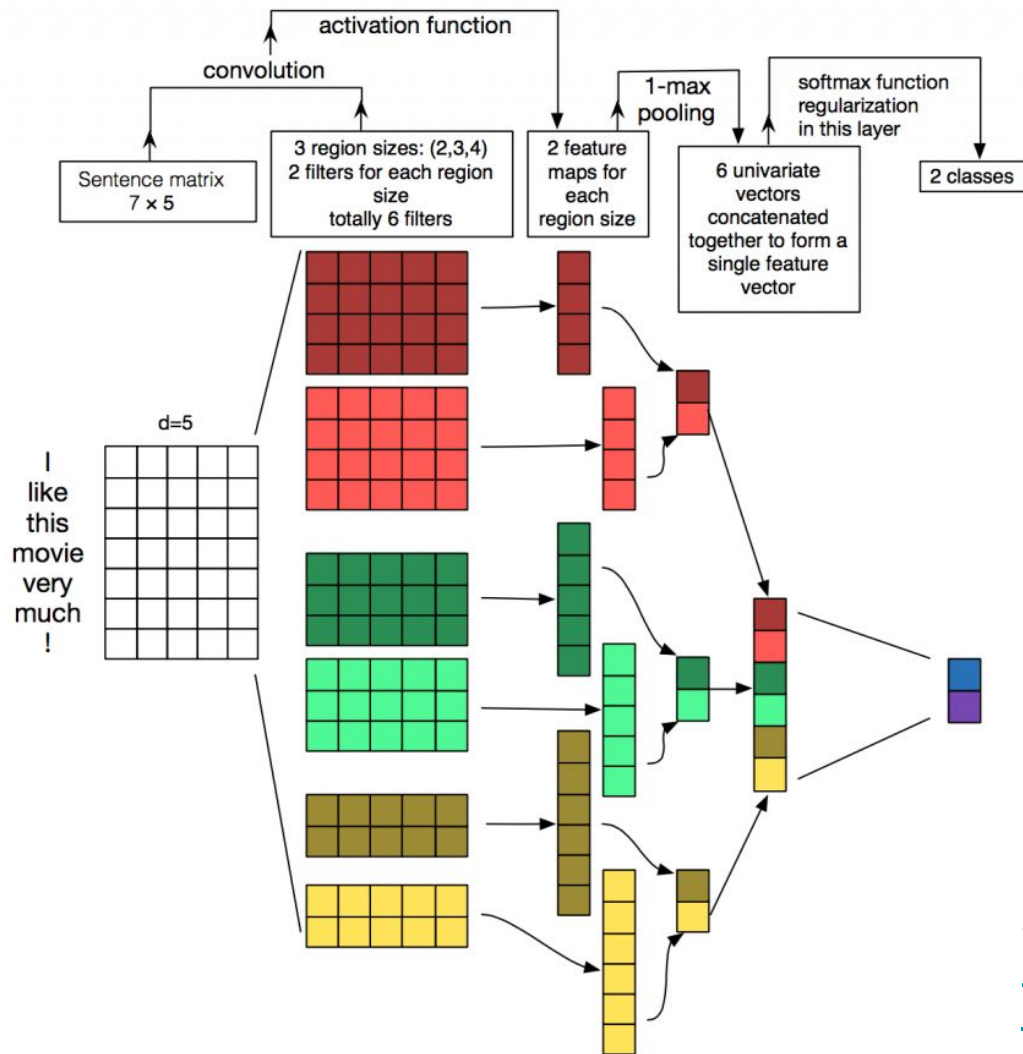
$$\mathbf{c} = [c_1, c_2, \dots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$

- Let's use pooling over the time axis:  $\hat{c} = \max\{\mathbf{c}\}$
- Now the length of  $\mathbf{c}$  is irrelevant!
- So we can use filters based on unigrams, bigrams, tri-grams, 4-grams, etc.

# Another example from Kim (2014) paper



# Example CNN structure

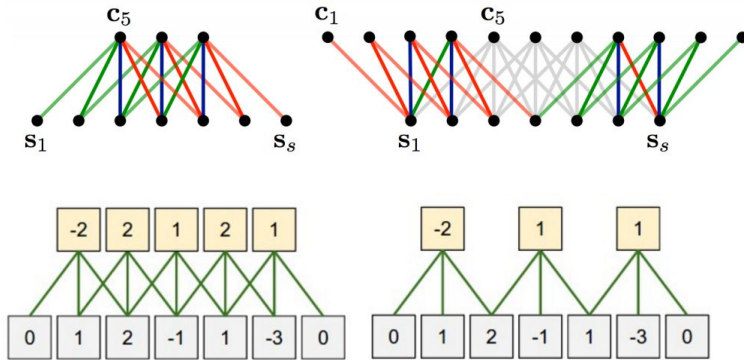


Source:

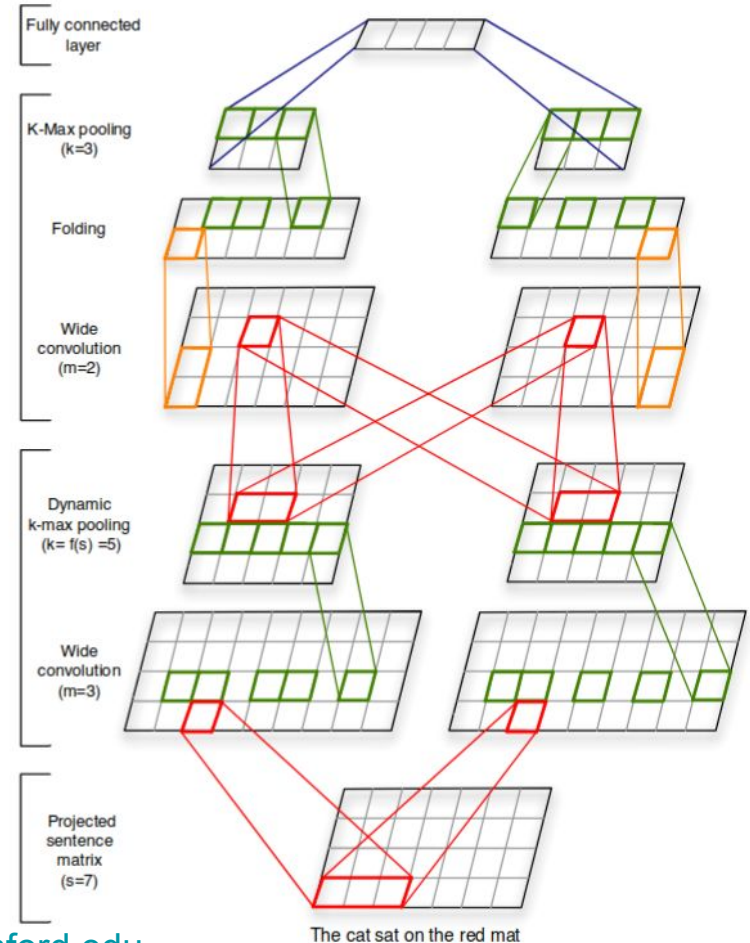
<http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/>

# More about CNN

- Narrow vs wide convolution (stride and zero-padding)

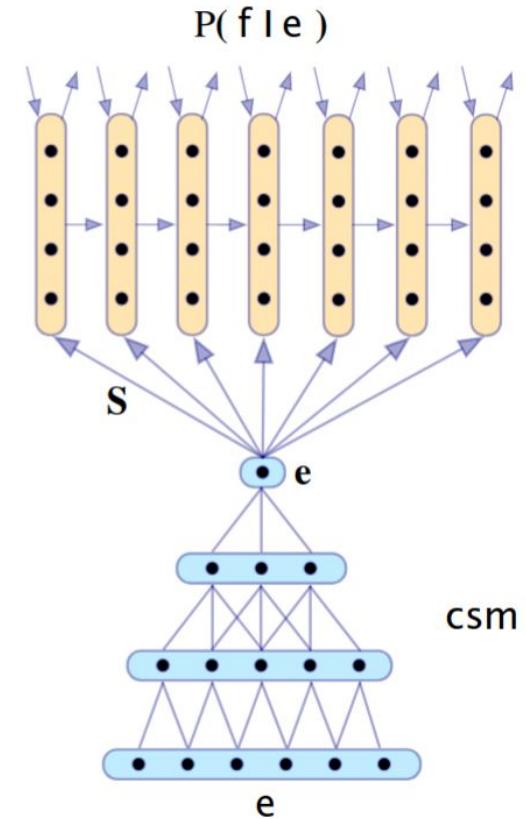


- Complex pooling schemes over sequences
- Great readings (e.g. Kalchbrenner et. al. 2014)



# CNN applications

- Neural machine translation: CNN as encoder, RNN as decoder
- Kalchbrenner and Blunsom (2013)  
“Recurrent Continuous Translation Models”
- One of the first neural machine translation efforts



# Outro and Q & A

- Vanishing gradient is present not only in RNNs
  - Use some kind of memory or skip-connections
- LSTM and GRU are both great
  - GRU is quicker, LSTM catch more complex dependencies
- Rule of thumb: start with LSTM, but switch to GRU if you want something more computationally efficient
- Clip your gradients
- Combining RNN and CNN worlds? Why not ;)

That's all. Feel free to ask any questions.