

# Lecture 02: Convolutional Neural Networks, CNN for texts

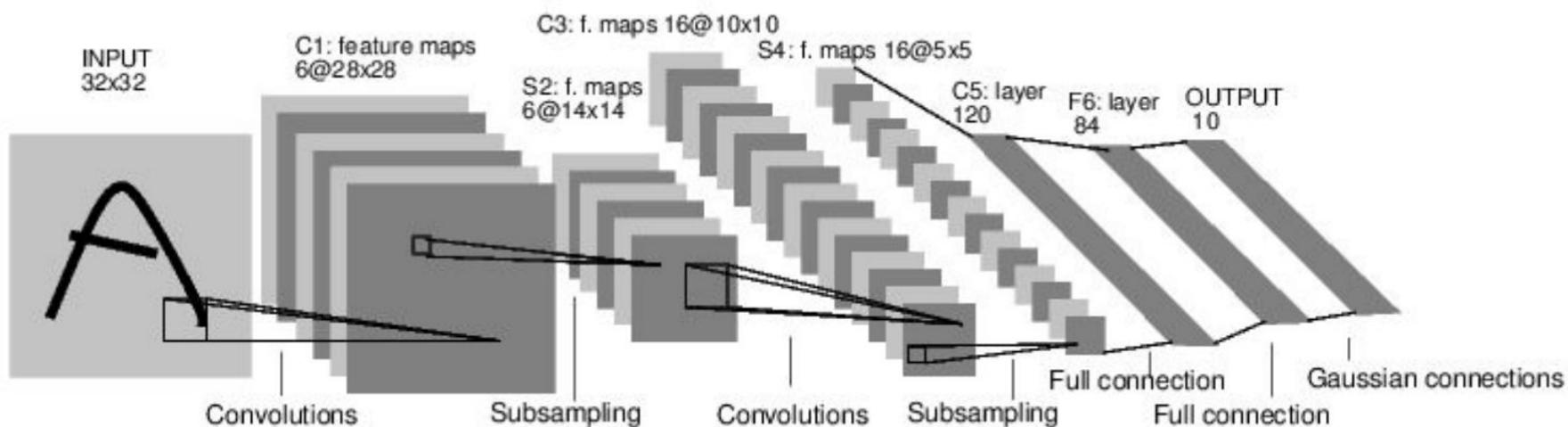
**Radoslav Neychev  
Anastasia Ianina**

Harbour.Space University  
09.06.2020, Barcelona, Spain

# Outline

- Convolutional Neural Networks
  - Intro/recap
  - Main definitions
- RNN problems recap:
  - Vanishing gradient
  - Exploding gradient
- Potential solutions:
  - LSTM/GRU
  - Gradient clipping
  - Skip connections
- CNNs for text processing

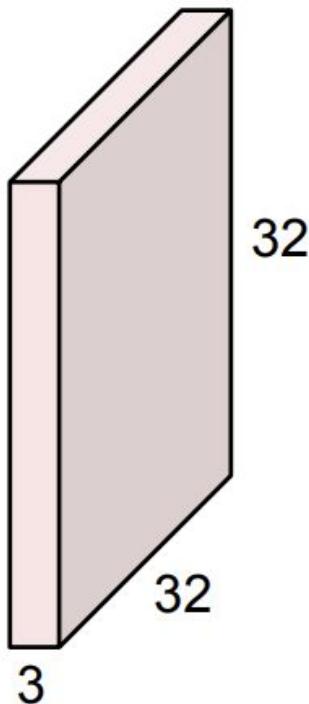
# Convolutional Neural Networks: Intro or recap



[LeNet-5, LeCun 1998]

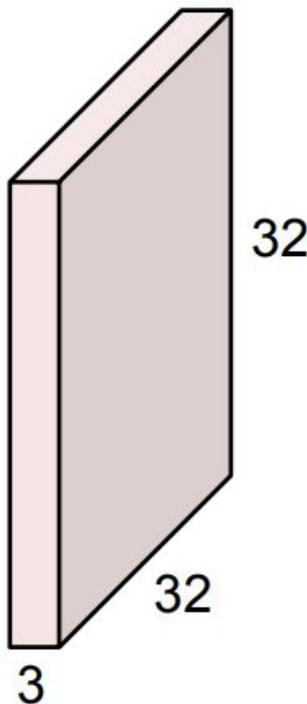
# Convolutional layer

32x32x3 image

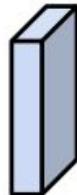


# Convolutional layer

32x32x3 image



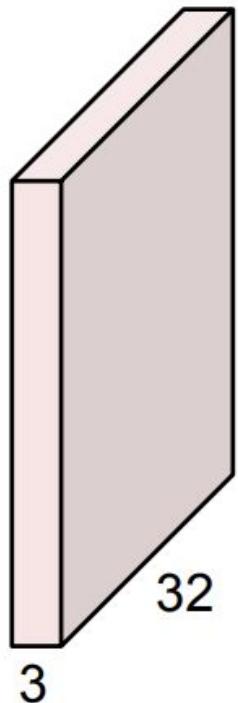
5x5x3 filter



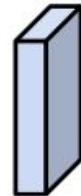
**Convolve** the filter with the image  
i.e. “slide over the image spatially,  
computing dot products”

# Convolutional layer

32x32x3 image



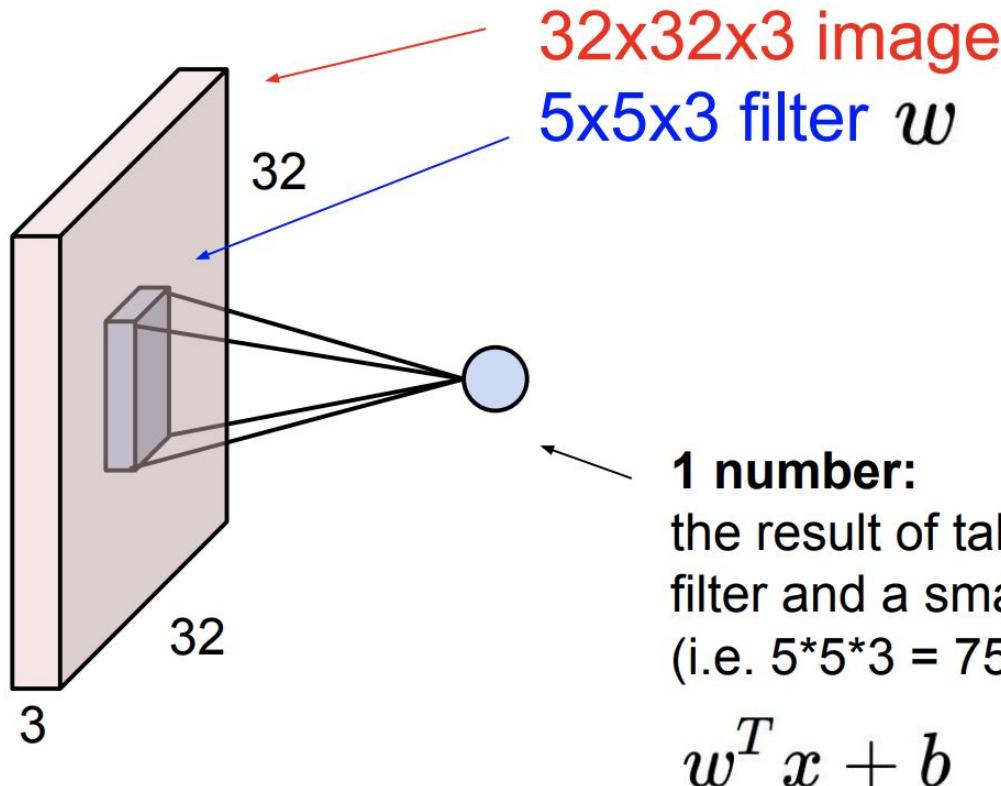
5x5x3 filter



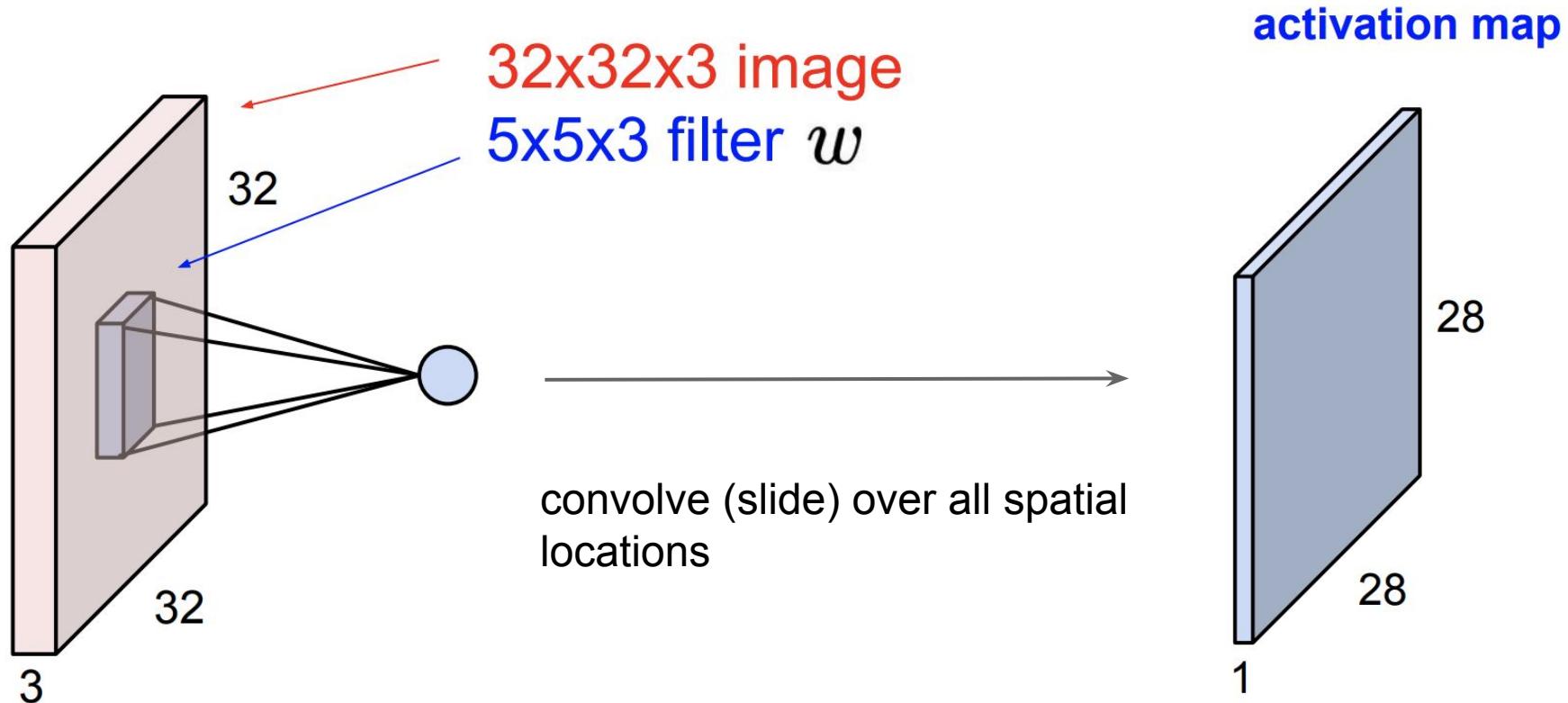
Filters extend the *depth* of the original image

**Convolve** the filter with the image  
i.e. “slide over the image spatially,  
computing dot products”

# Convolutional layer

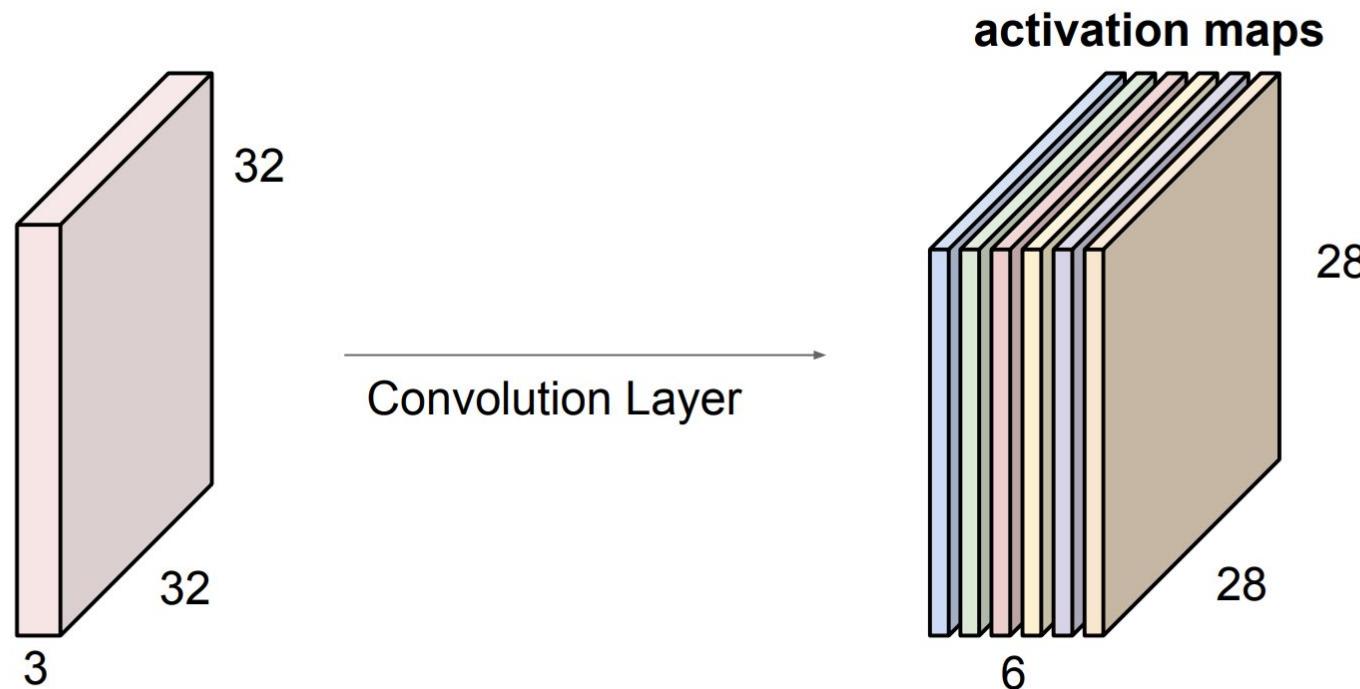


# Convolutional layer



# Convolutional layer

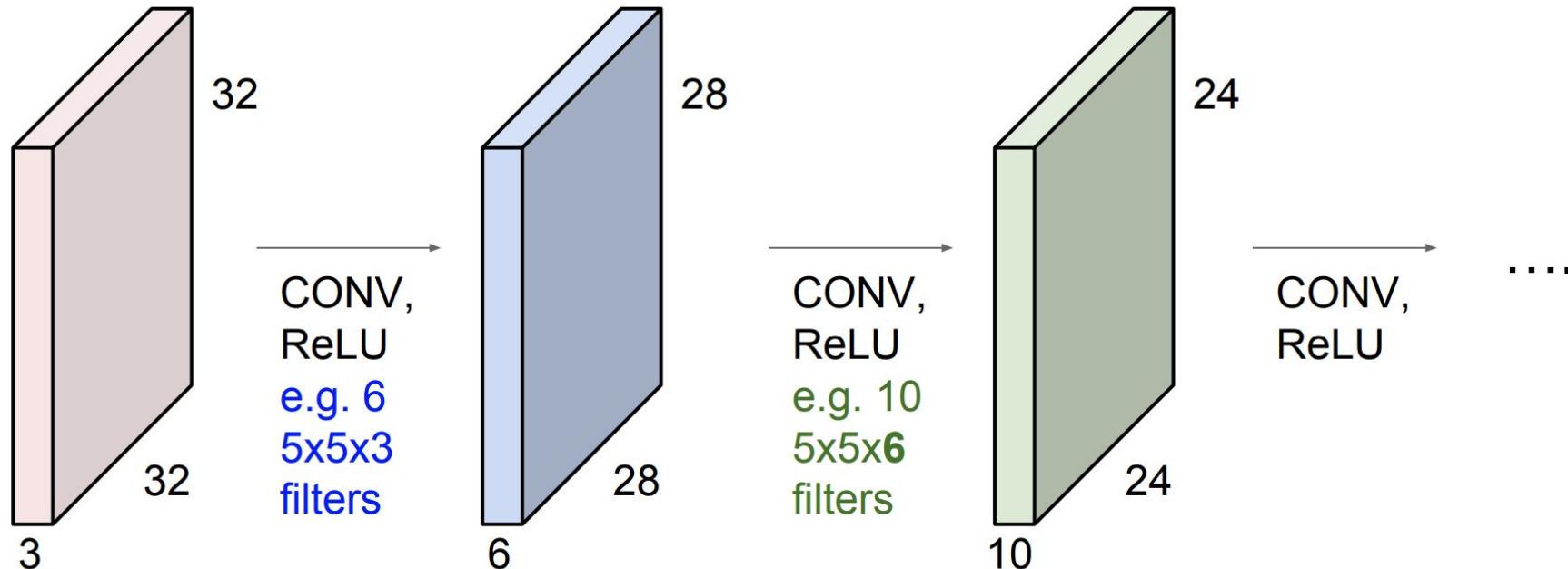
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



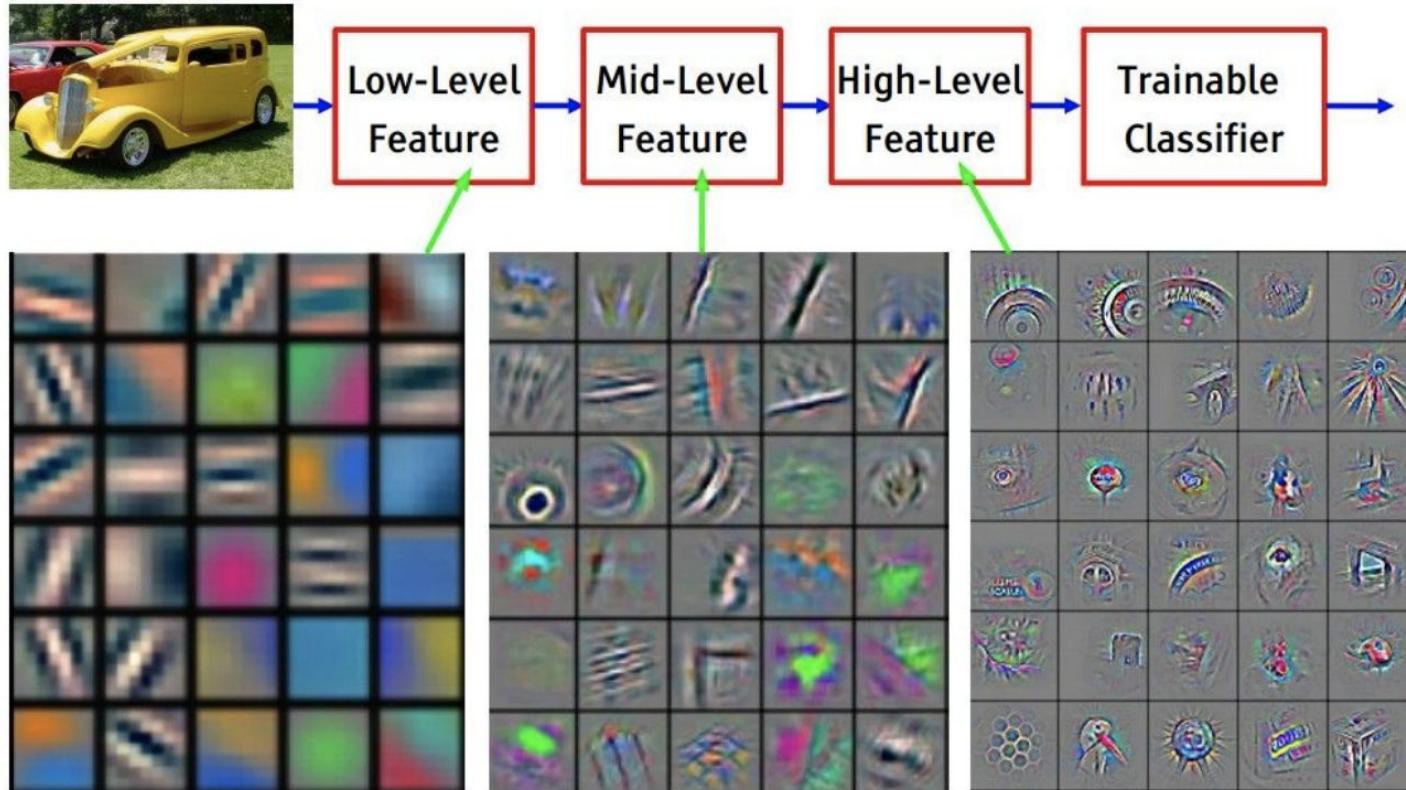
We stack these up to get a “new image” of size 28x28x6!

# Convolutional layer

**Preview:** ConvNet is a sequence of Convolutional Layers, interspersed with activation functions

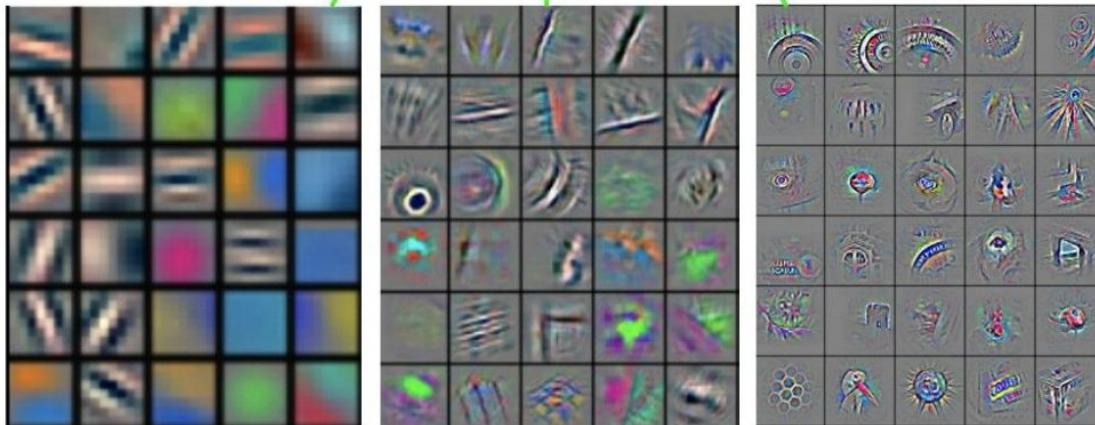


# Convolutional layer

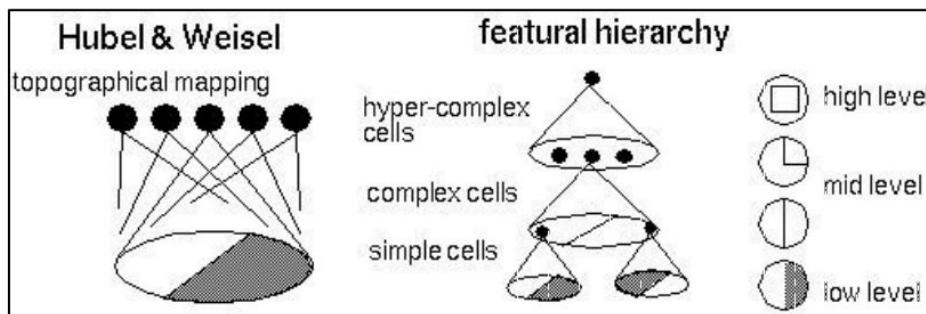


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

[From Yann LeCun slides]



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]



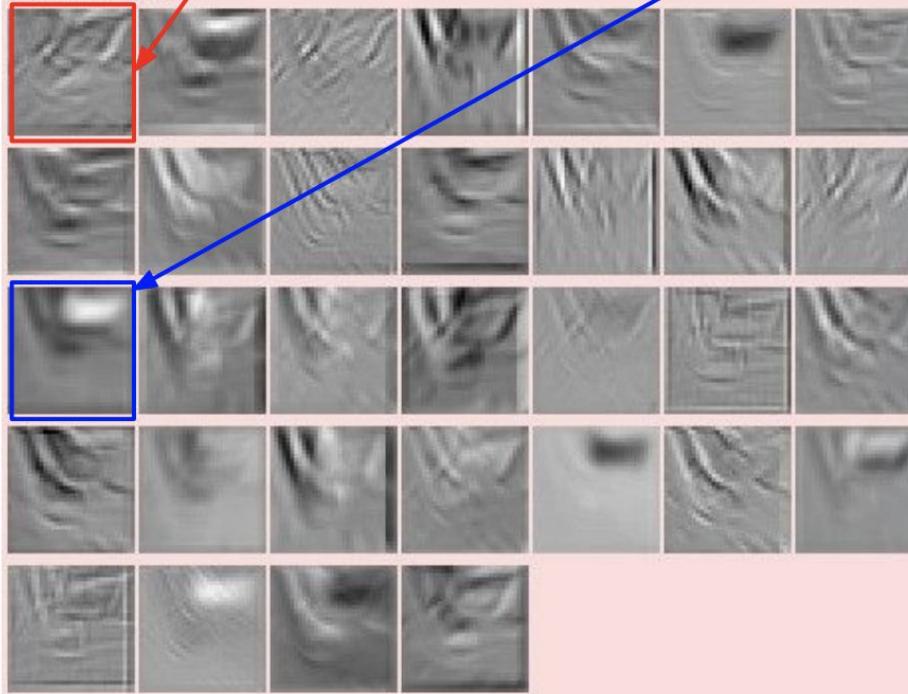
# Convolutional layer and visual cortex

[From Yann LeCun slides]



one filter =>  
one activation map

Activations:



example 5x5 filters  
(32 total)

We call the layer convolutional  
because it is related to convolution  
of two signals:

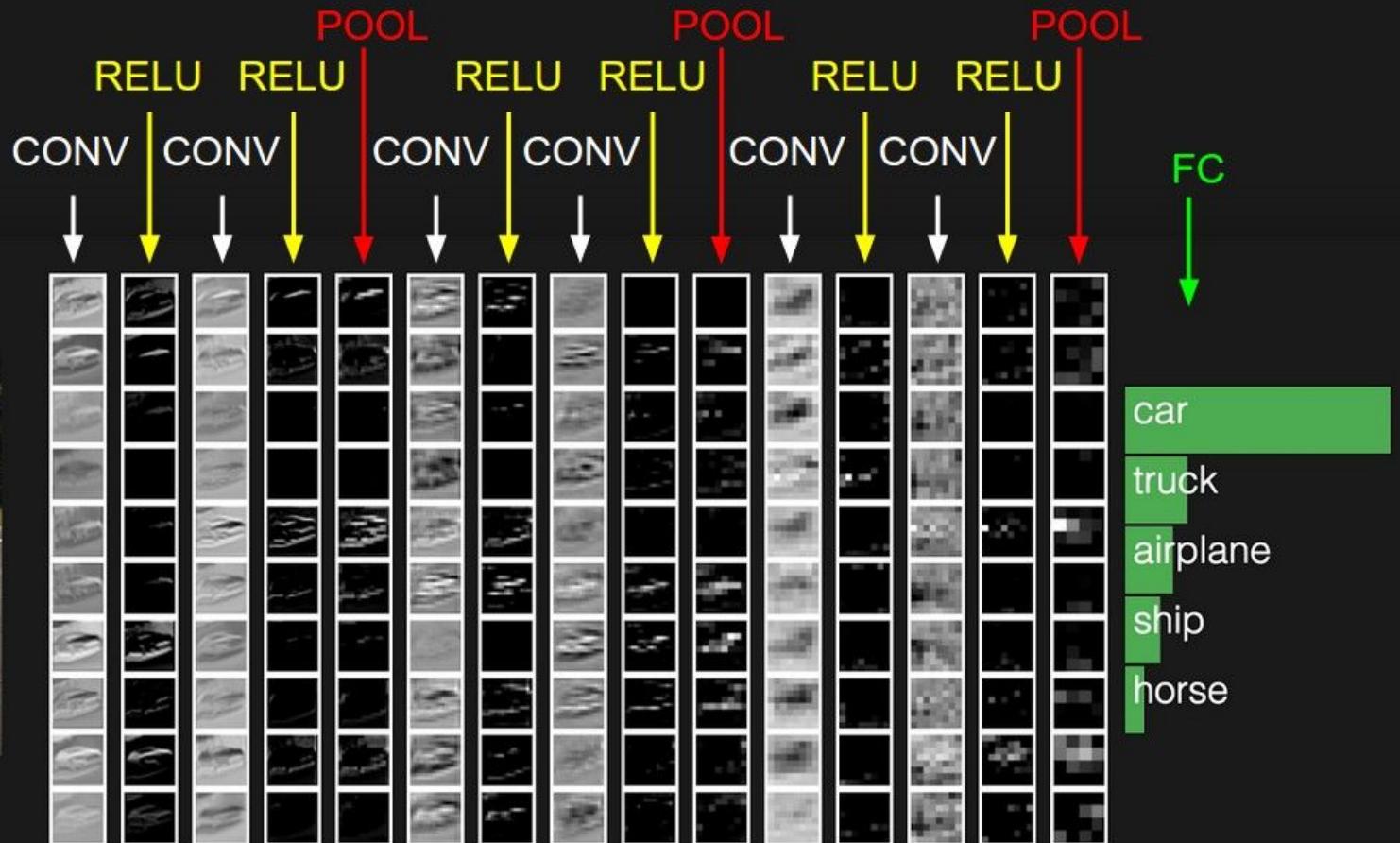
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1, n_2] \cdot g[x - n_1, y - n_2]$$



elementwise multiplication and sum of  
a filter and the signal (image)

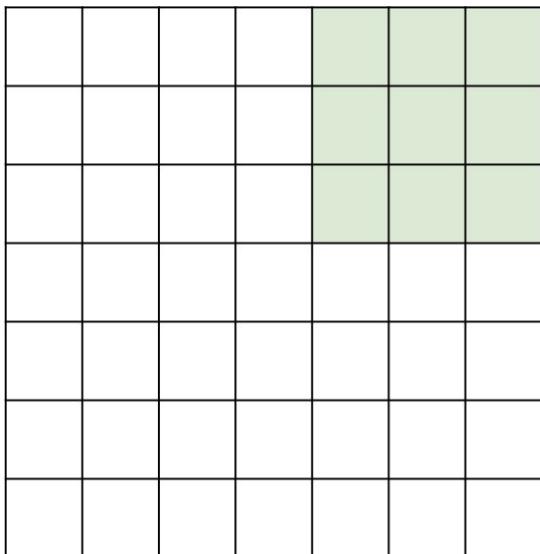
CIFAR-10 online demo:

<https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>



## A closer look at spatial dimensions:

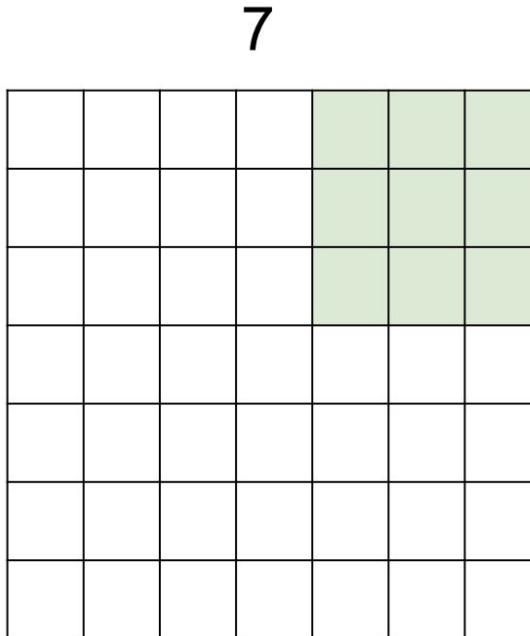
7



7x7 input (spatially)  
assume 3x3 filter

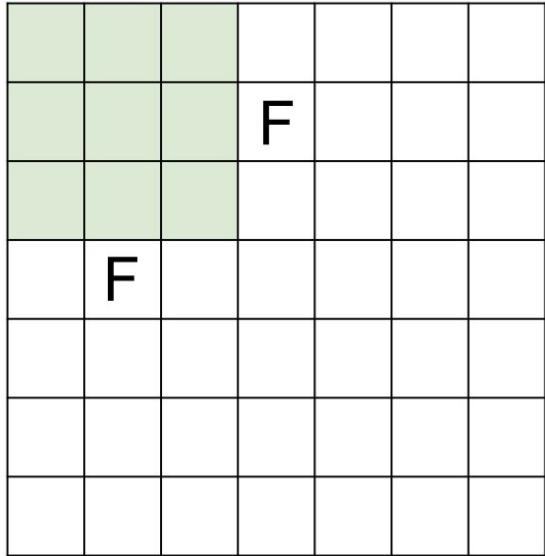
**=> 5x5 output**

## A closer look at spatial dimensions:



7x7 input (spatially)  
assume 3x3 filter  
applied **with stride 2**  
**=> 3x3 output!**

N



N

Output size:  
**(N - F) / stride + 1**

e.g.  $N = 7, F = 3$ :

$$\text{stride } 1 \Rightarrow (7 - 3)/1 + 1 = 5$$

$$\text{stride } 2 \Rightarrow (7 - 3)/2 + 1 = 3$$

$$\text{stride } 3 \Rightarrow (7 - 3)/3 + 1 = 2.33 : \backslash$$

# In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

**3x3 filter, applied with stride 1**

**pad with 1 pixel border => what is the output?**

**7x7 output!**

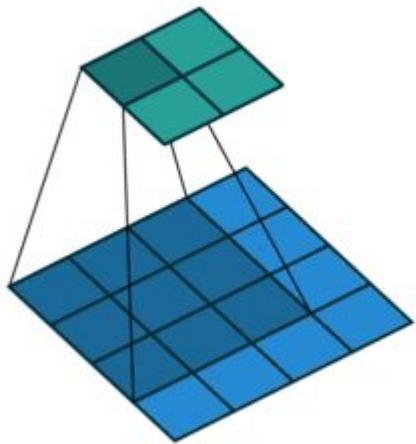
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with  $(F-1)/2$ . (will preserve size spatially)

e.g.  $F = 3 \Rightarrow$  zero pad with 1

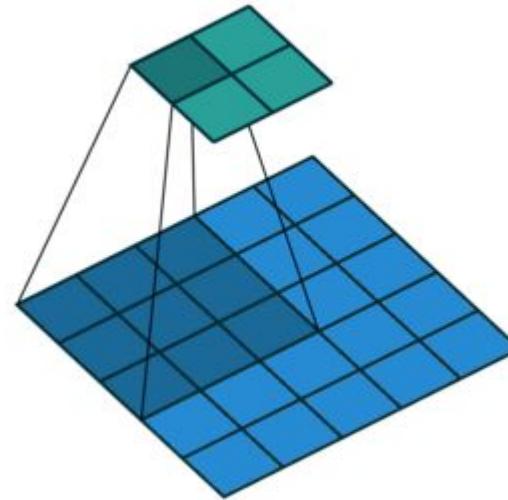
$F = 5 \Rightarrow$  zero pad with 2

$F = 7 \Rightarrow$  zero pad with 3

# Strides, padding in convolutional layer

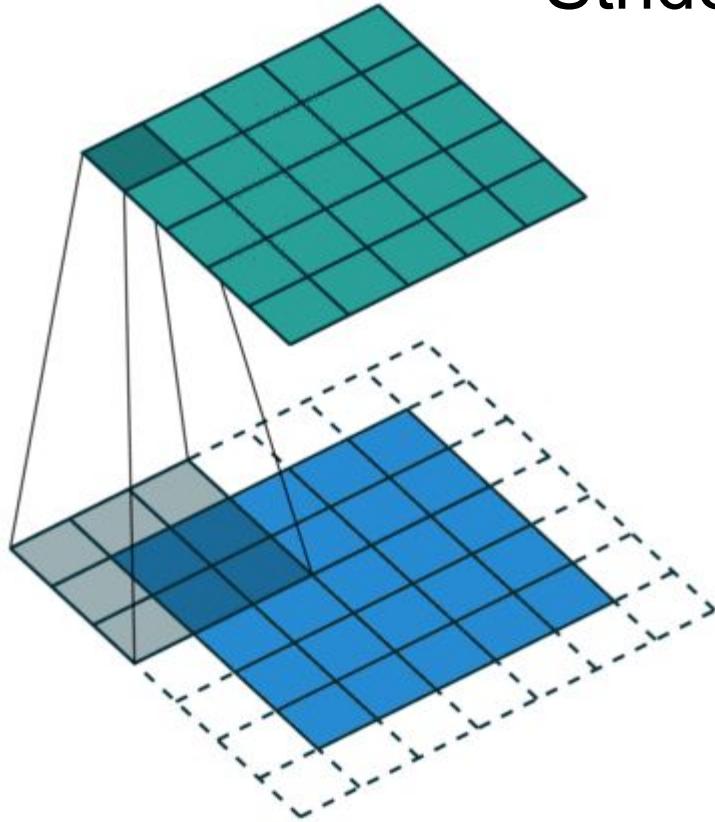


No padding,  
no strides

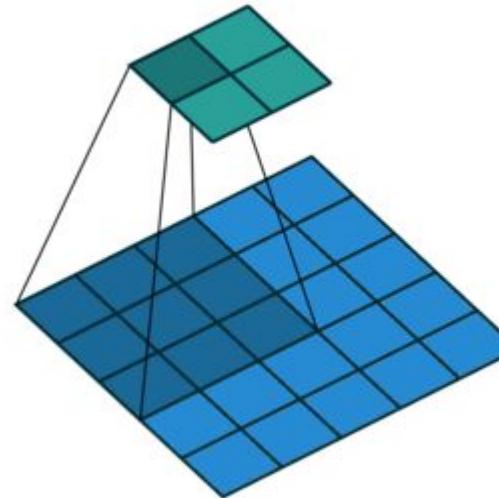


No padding,  
with strides

# Strides, padding in convolutional layer

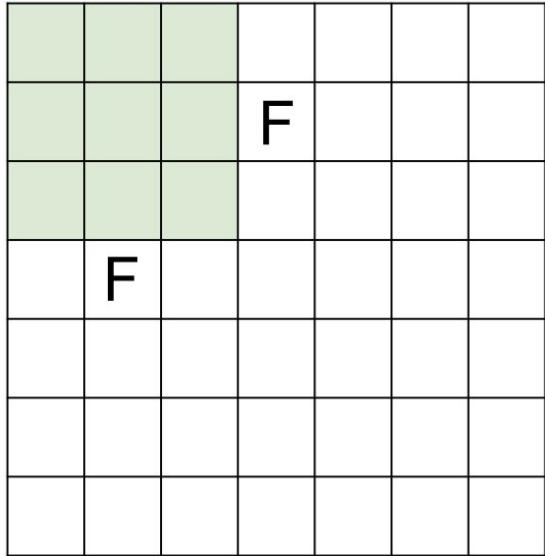


With padding,  
no strides



No padding,  
with strides

N



N

Output size:  
**(N - F) / stride + 1**

e.g.  $N = 7, F = 3$ :

$$\text{stride } 1 \Rightarrow (7 - 3)/1 + 1 = 5$$

$$\text{stride } 2 \Rightarrow (7 - 3)/2 + 1 = 3$$

$$\text{stride } 3 \Rightarrow (7 - 3)/3 + 1 = 2.33 :\backslash$$

# In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

**3x3 filter, applied with stride 1**

**pad with 1 pixel border => what is the output?**

**7x7 output!**

in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with  $(F-1)/2$ . (will preserve size spatially)

e.g.  $F = 3 \Rightarrow$  zero pad with 1

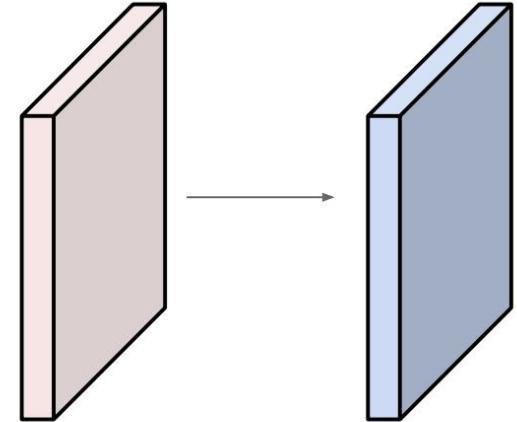
$F = 5 \Rightarrow$  zero pad with 2

$F = 7 \Rightarrow$  zero pad with 3

# Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

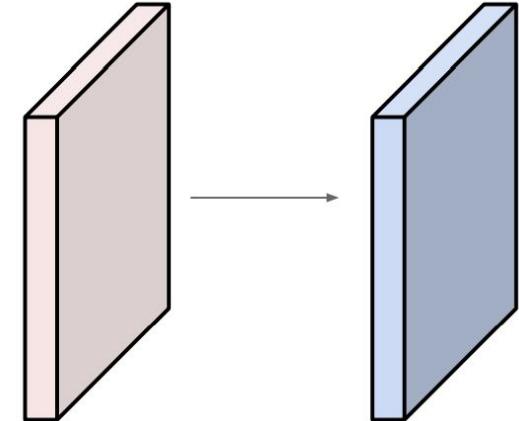


Output volume size: ?

Examples time:

Input volume: **32x32x3**

10 **5x5** filters with stride 1, pad 2



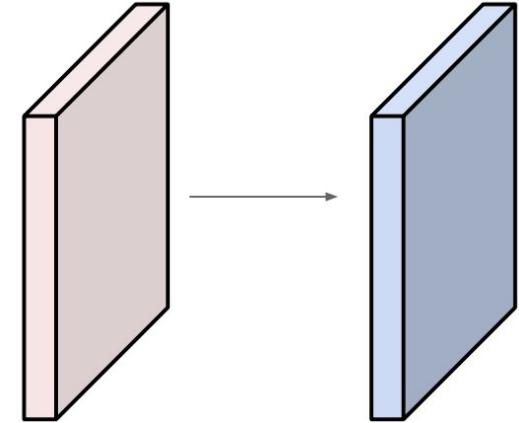
Output volume size:

$(32+2*2-5)/1+1 = 32$  spatially, so  
**32x32x10**

Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

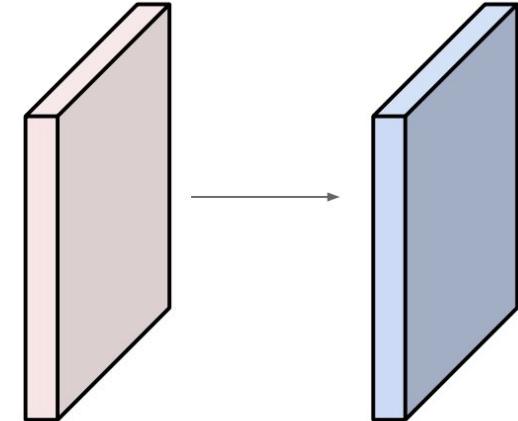


Number of parameters in this layer?

Examples time:

Input volume: **32x32x3**

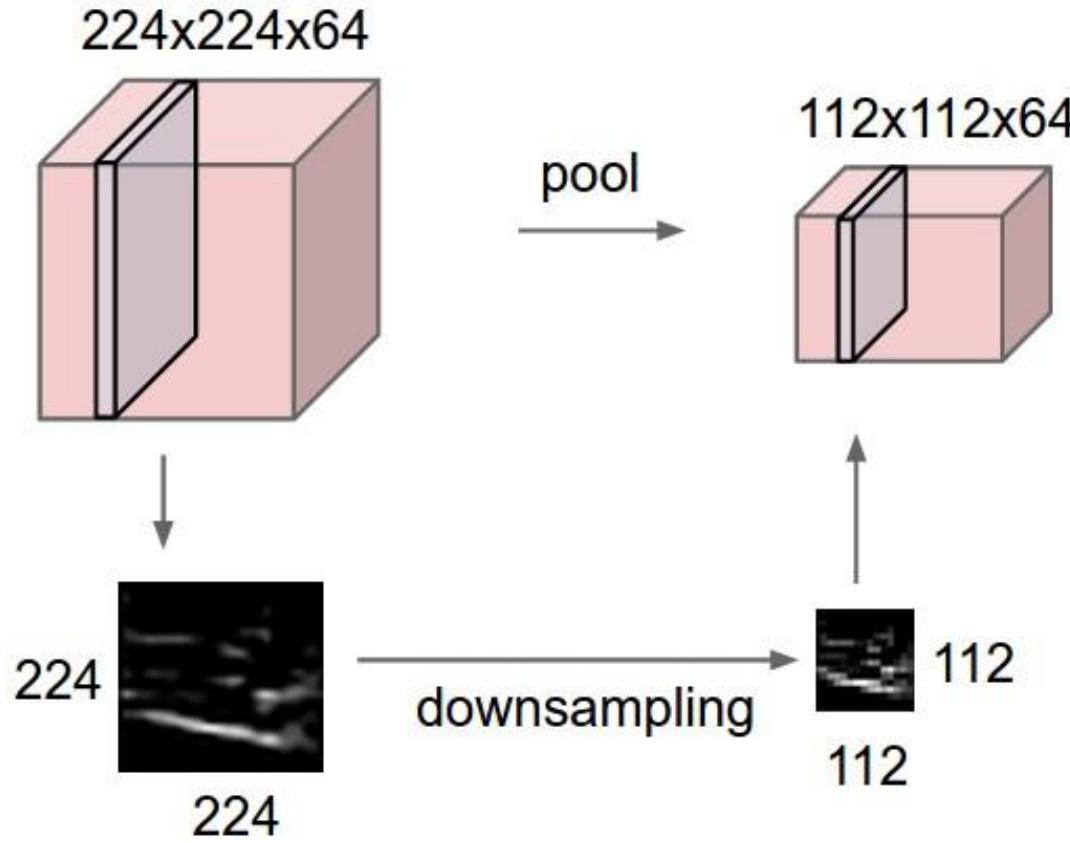
**10 5x5** filters with stride 1, pad 2



Number of parameters in this layer?

each filter has  $5*5*3 + 1 = 76$  params (+1 for bias)  
 $\Rightarrow 76*10 = 760$

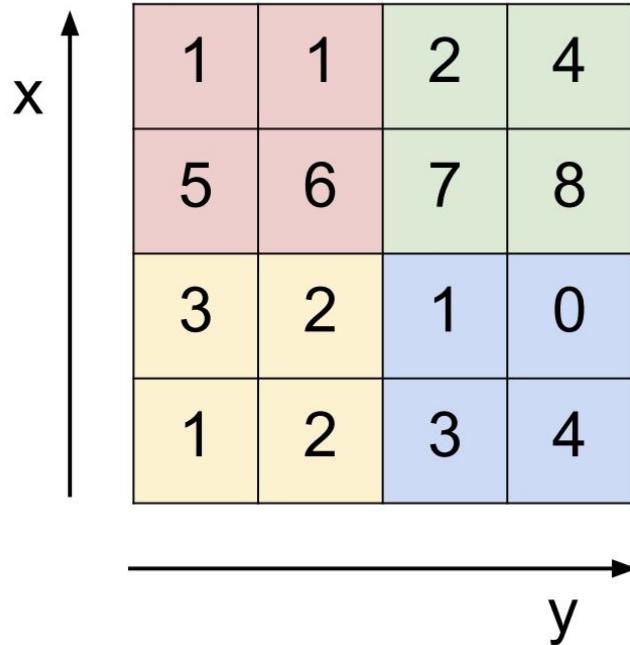
# Pooling layer



- Makes the representations smaller and more manageable
- Operates over each activation map independently

# Max pooling

Single depth slice



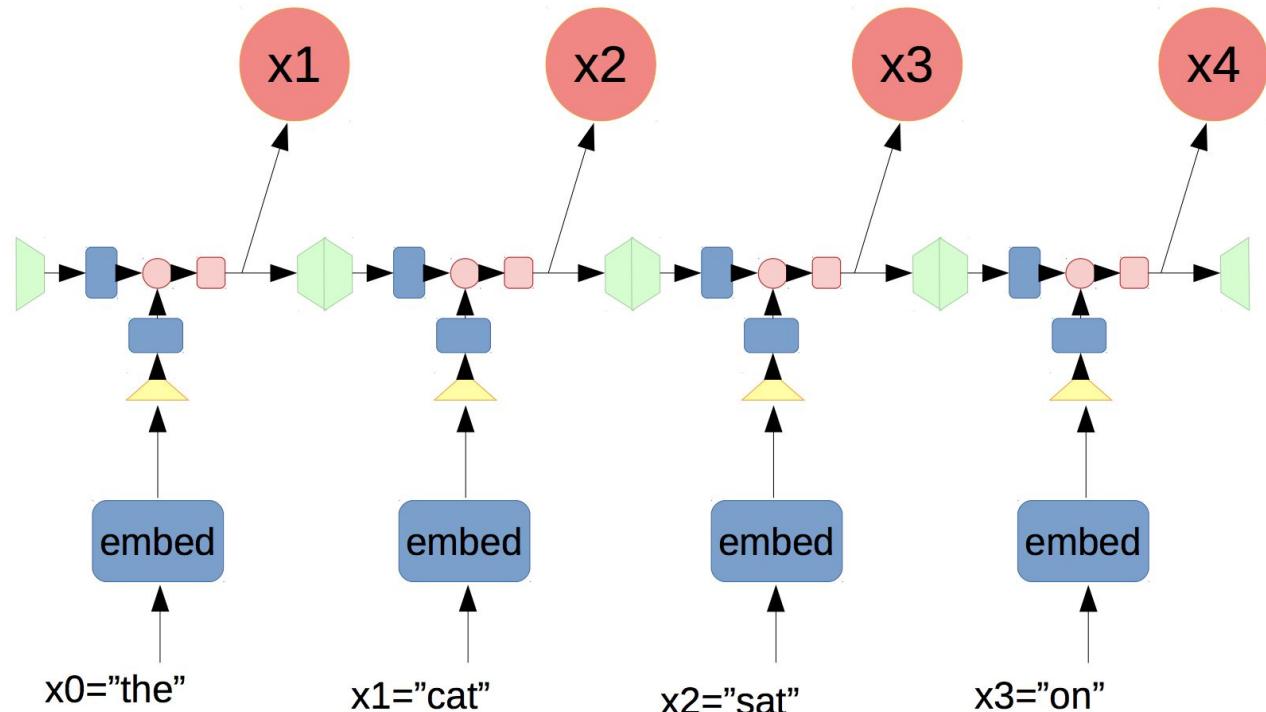
max pool with 2x2 filters  
and stride 2



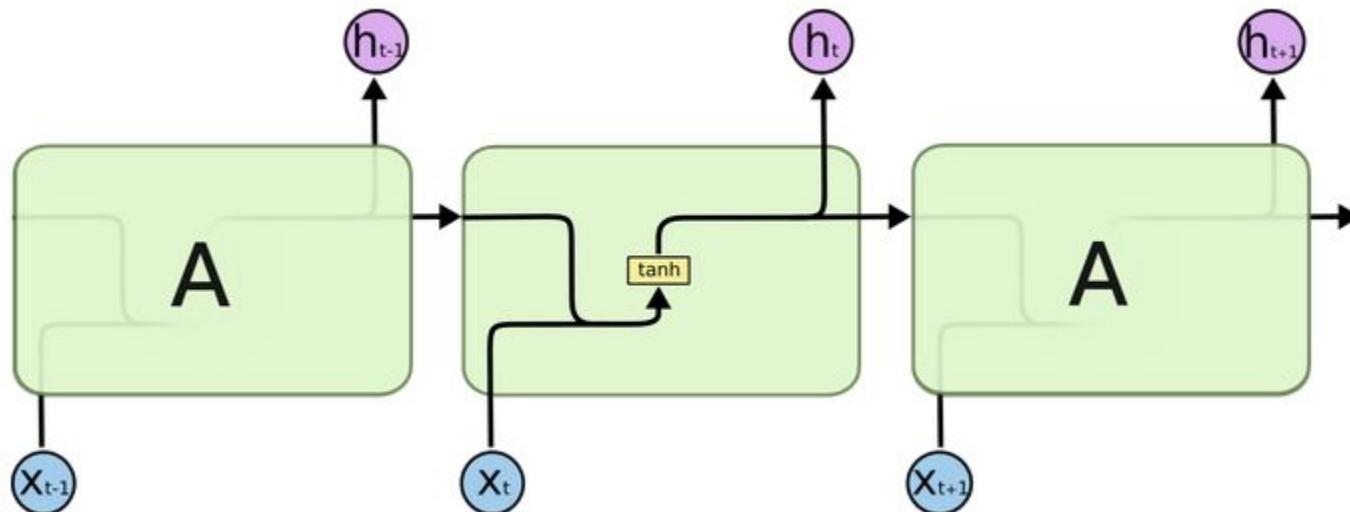
6	8
3	4

# Back to texts

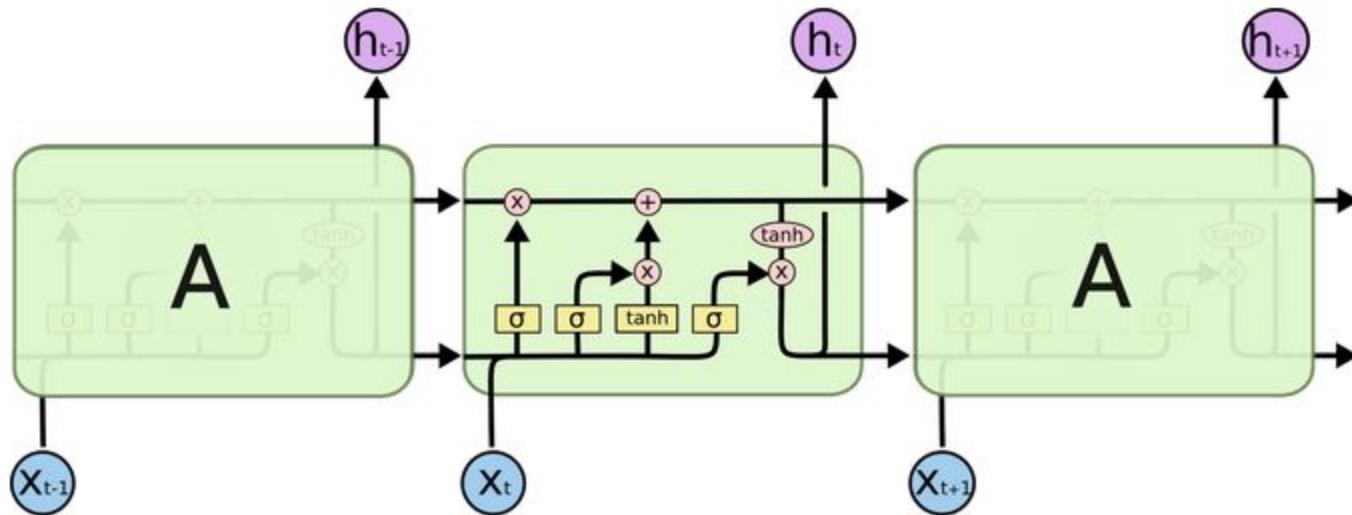
# Recap: RNN

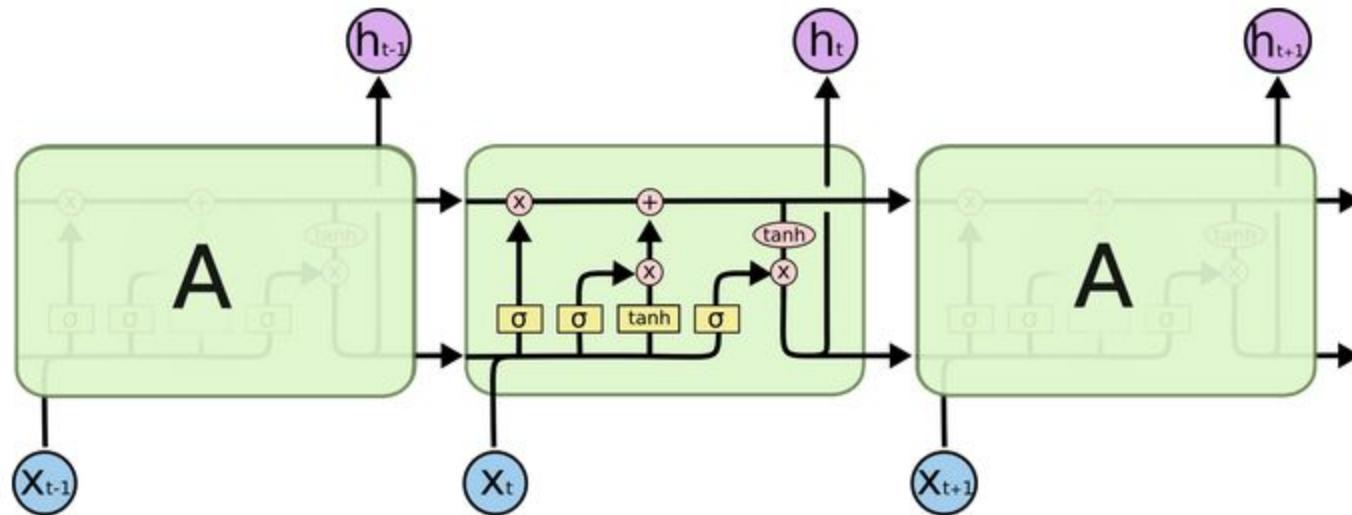


# Recap: Vanilla RNN

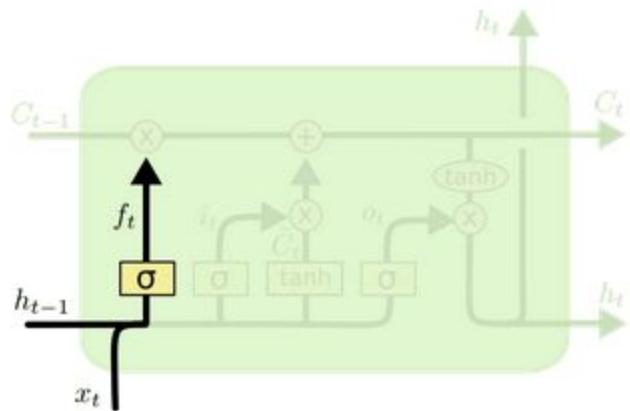


# Recap: LSTM



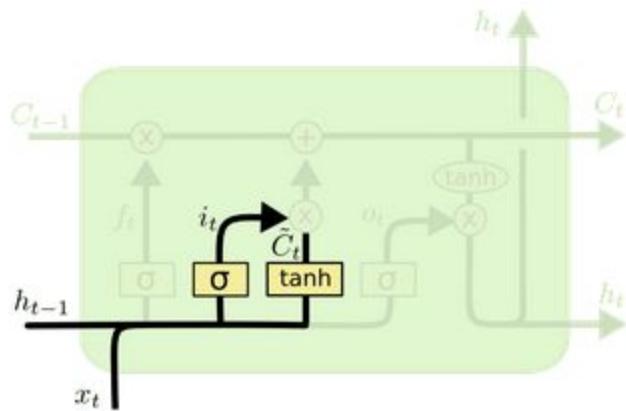


# LSTM: quick overview



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

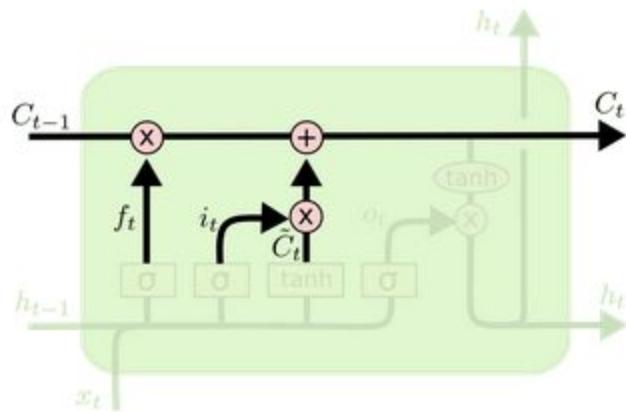
# LSTM: quick overview



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

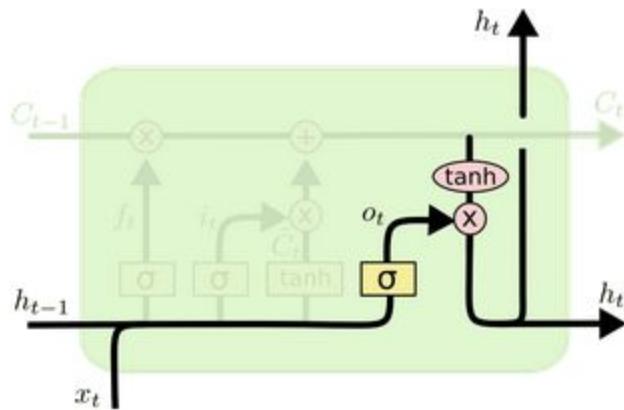
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

# LSTM: quick overview



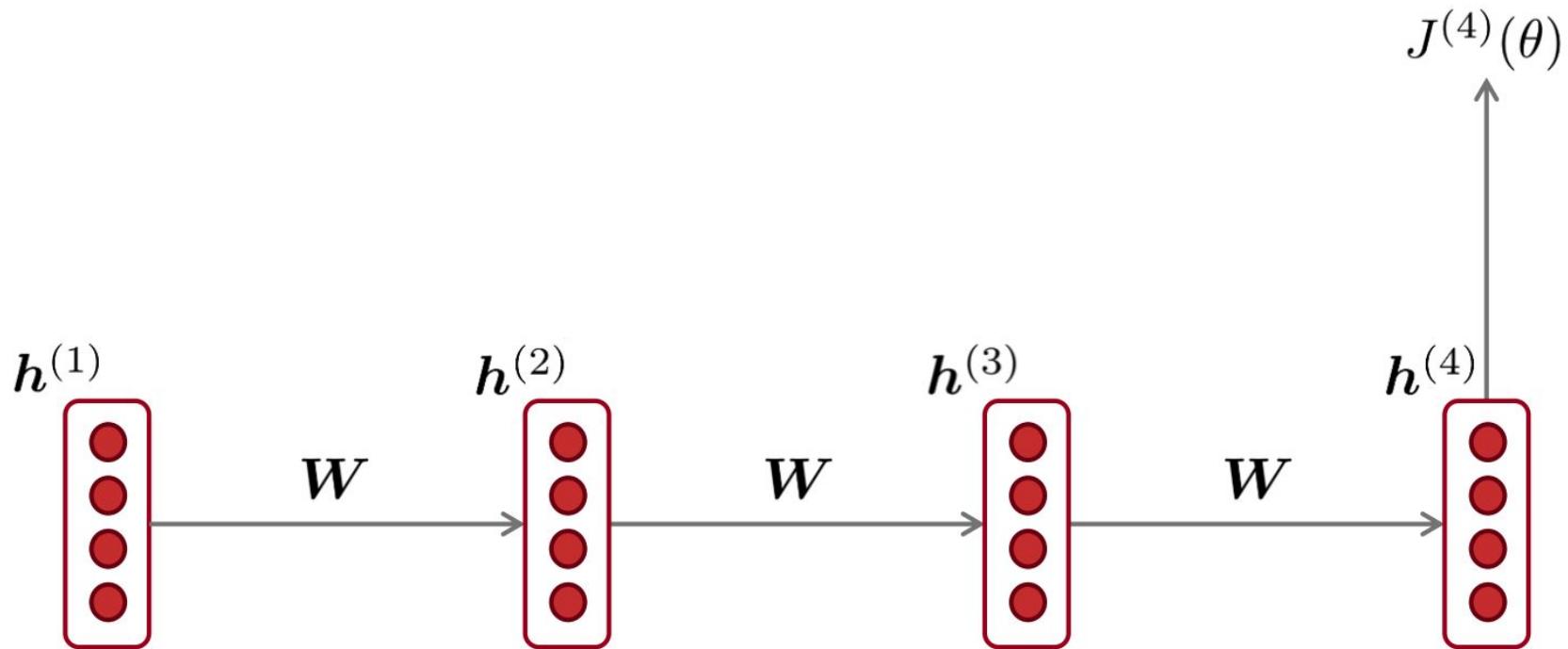
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

# LSTM: quick overview

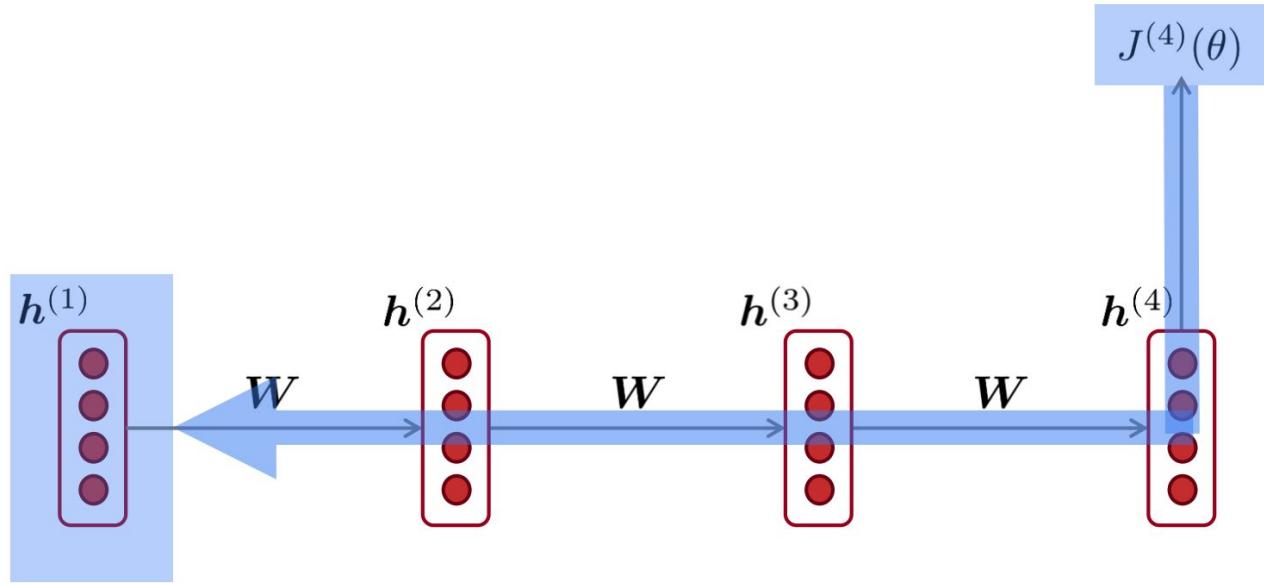


$$o_t = \sigma (W_o [ h_{t-1}, x_t ] + b_o)$$
$$h_t = o_t * \tanh (C_t)$$

# Vanishing gradient problem

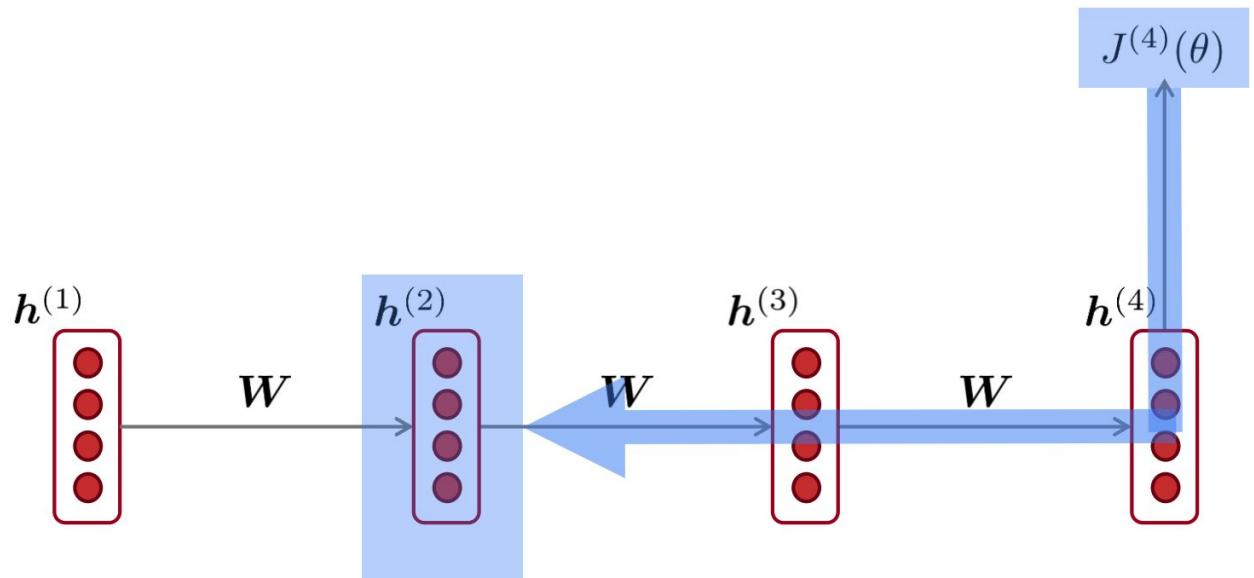


# Vanishing gradient problem



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

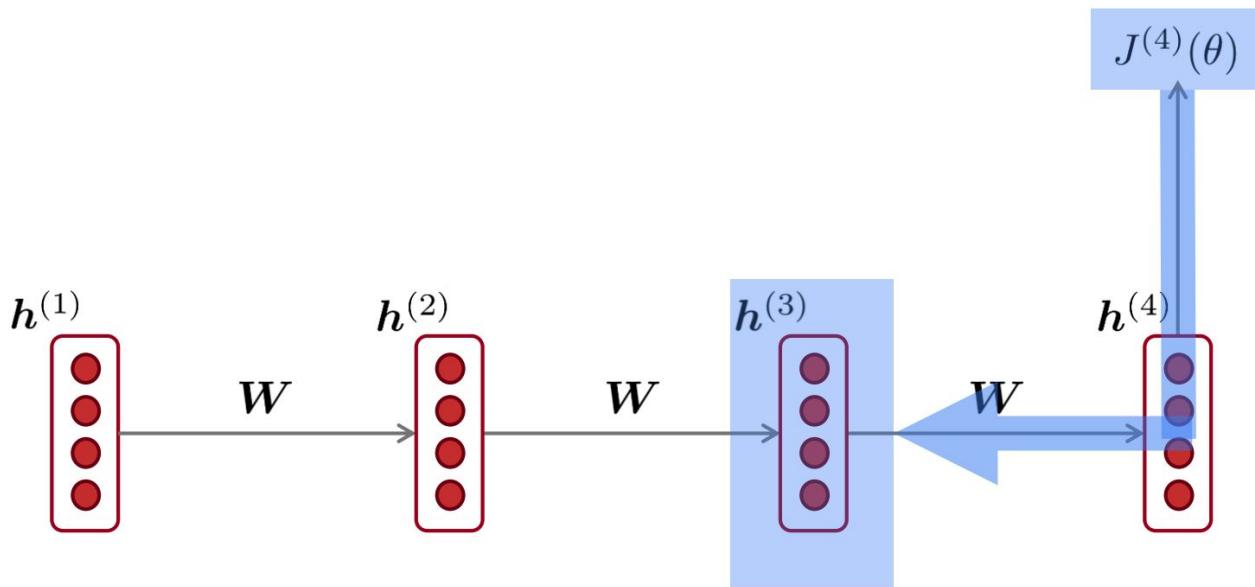
# Vanishing gradient problem



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

chain rule!

# Vanishing gradient problem

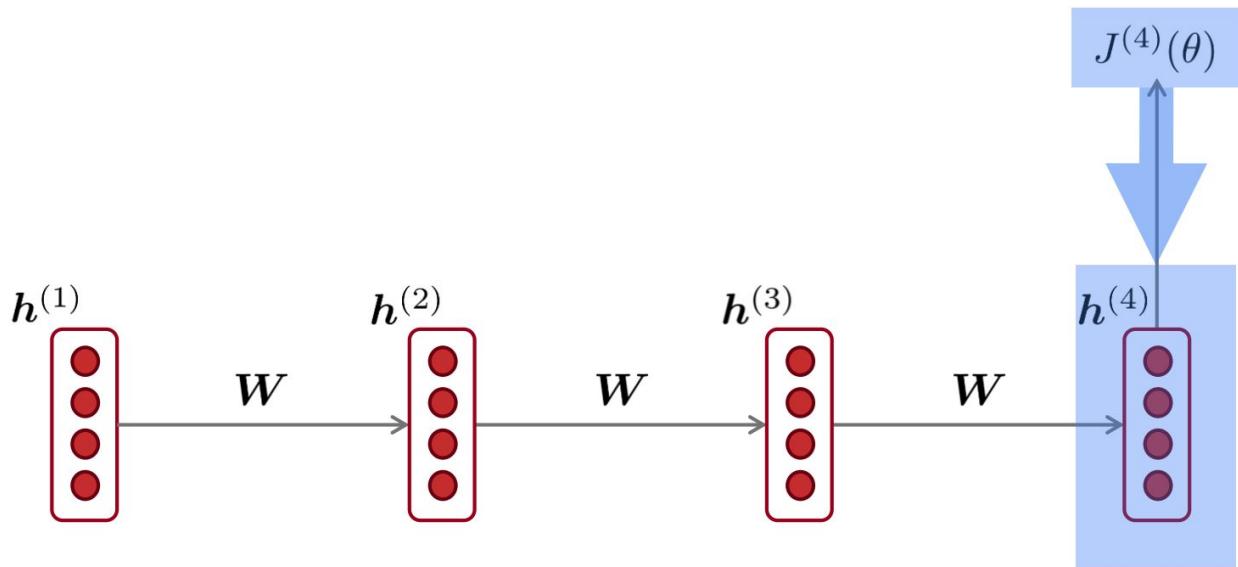


$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times$$

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

chain rule!

# Vanishing gradient problem



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

$$\frac{\partial h^{(3)}}{\partial h^{(2)}} \times$$

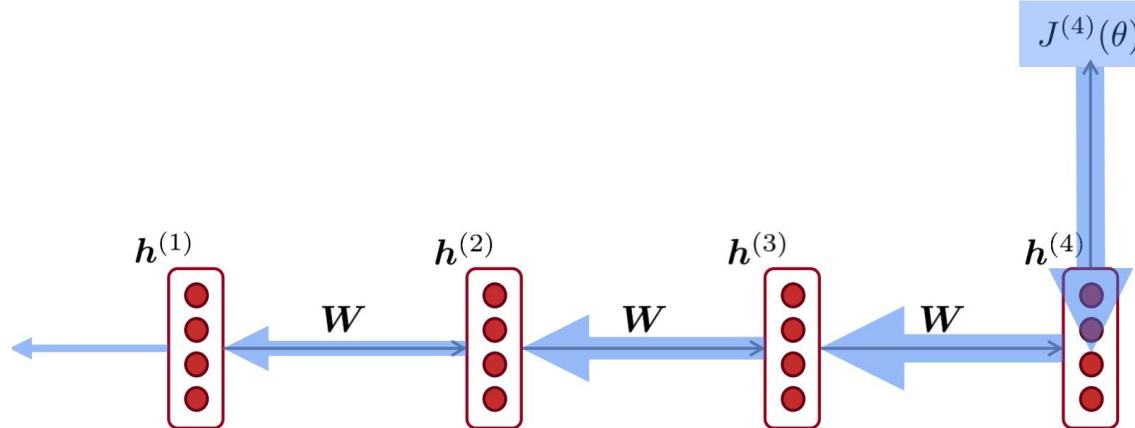
$$\frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

chain rule!

# Vanishing gradient problem

Vanishing gradient  
problem:

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



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

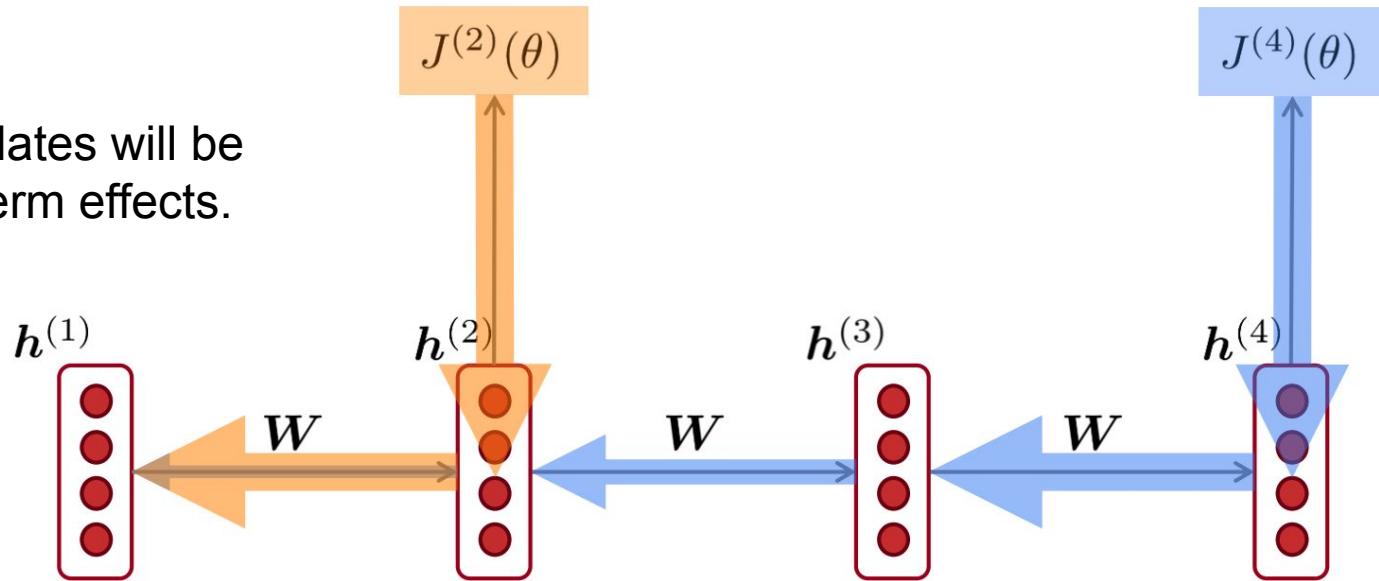
What happens if these are small?

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:
- 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)

$$\theta^{new} = \theta^{old} - \overbrace{\alpha \nabla_{\theta} J(\theta)}^{\text{gradient}}$$

learning rate

# 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

---

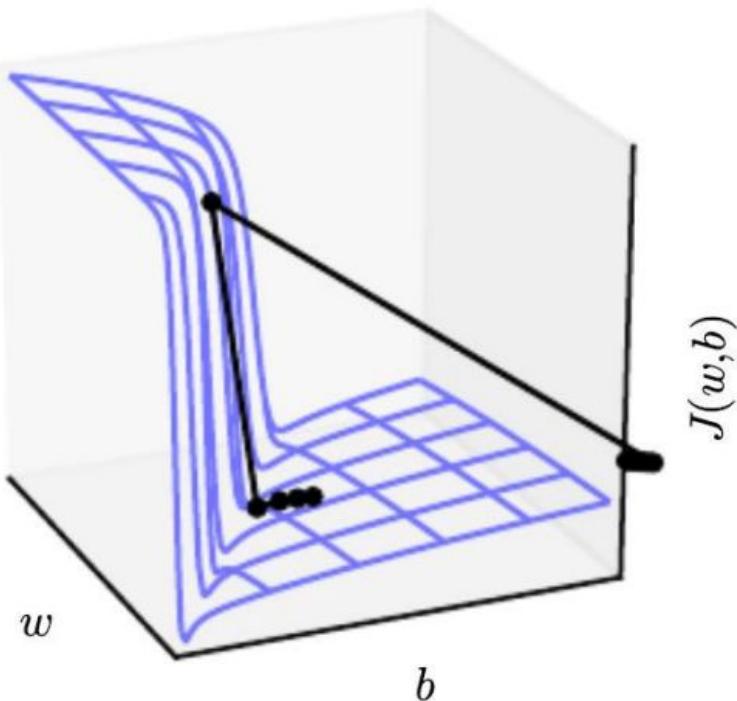
```
 $\hat{g} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$ 
if  $\|\hat{g}\| \geq threshold$  then
     $\hat{g} \leftarrow \frac{threshold}{\|\hat{g}\|} \hat{g}$ 
end if
```

---

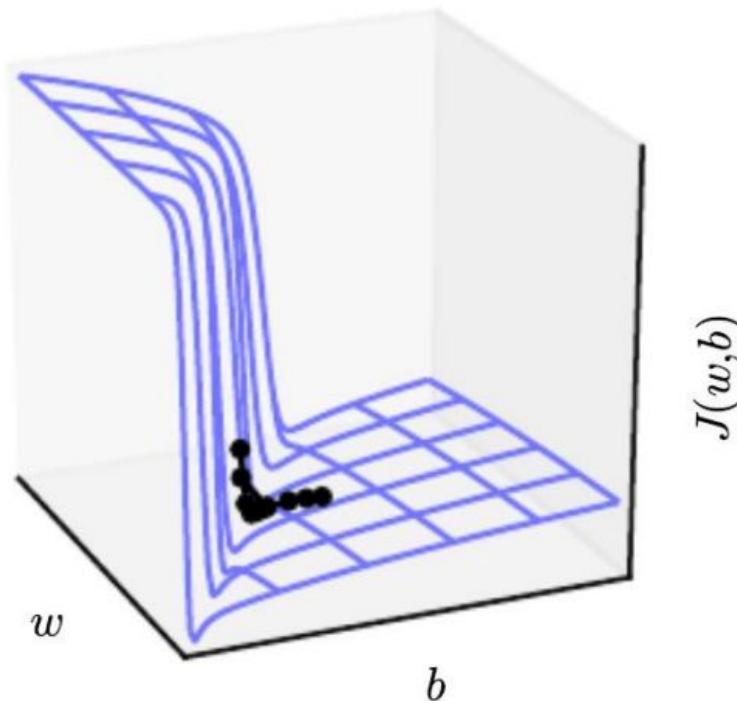
- Intuition: take a step in the same direction, but a smaller step

# Exploding gradient solution

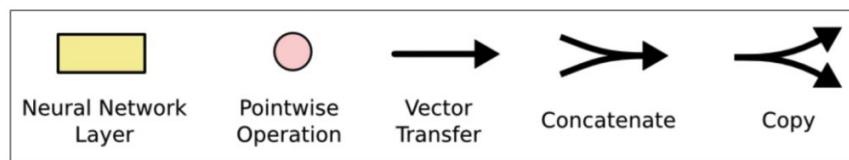
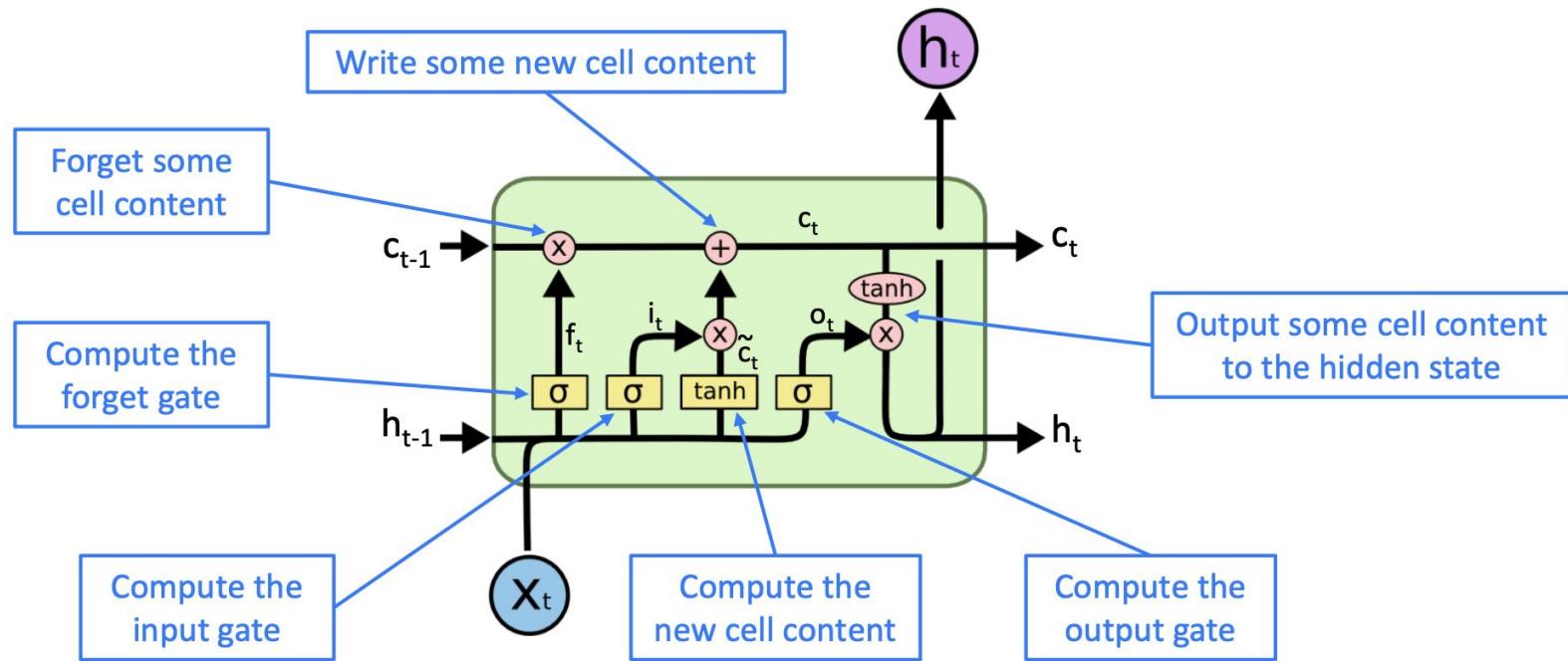
Without clipping



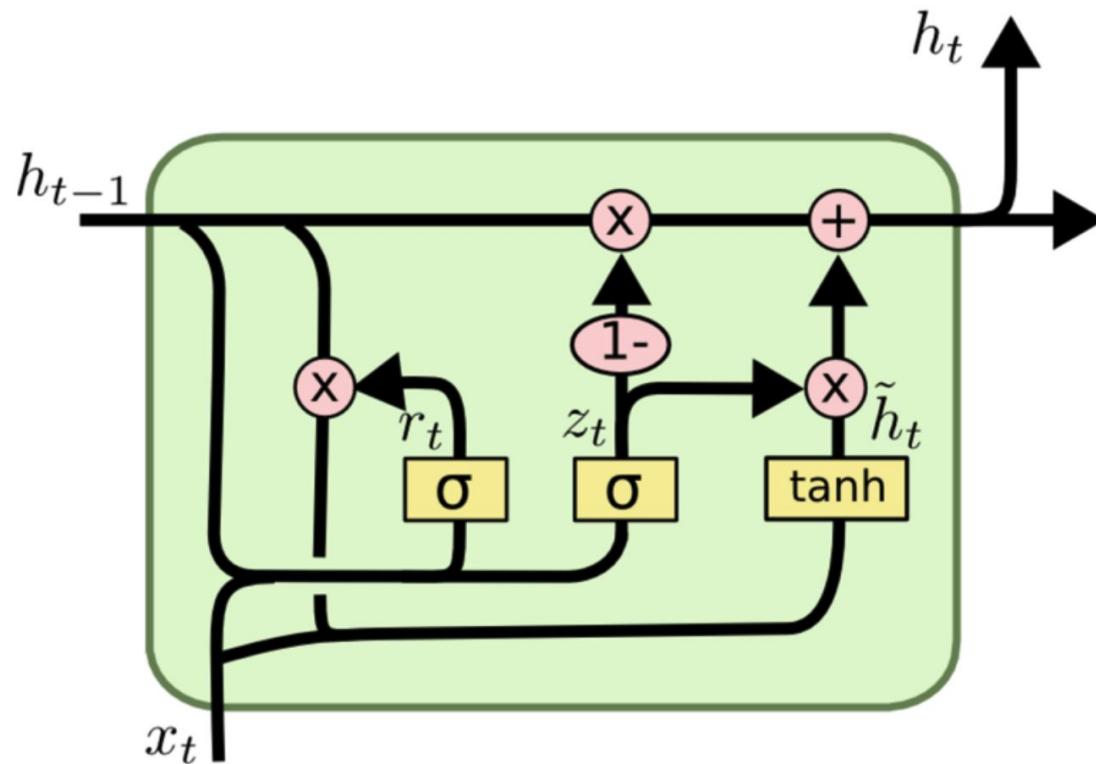
With clipping



# Vanishing gradient: LSTM



# Vanishing gradient: GRU



# 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)

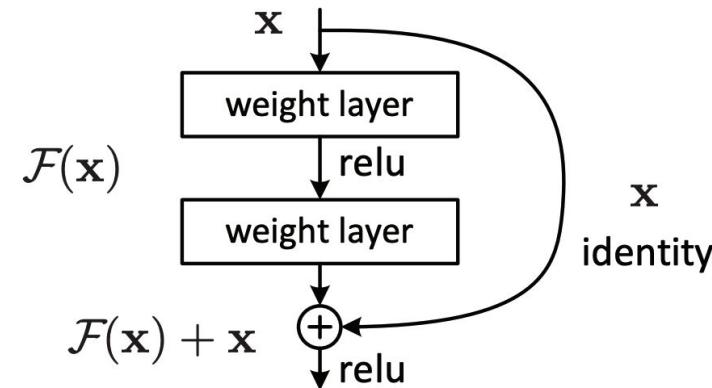
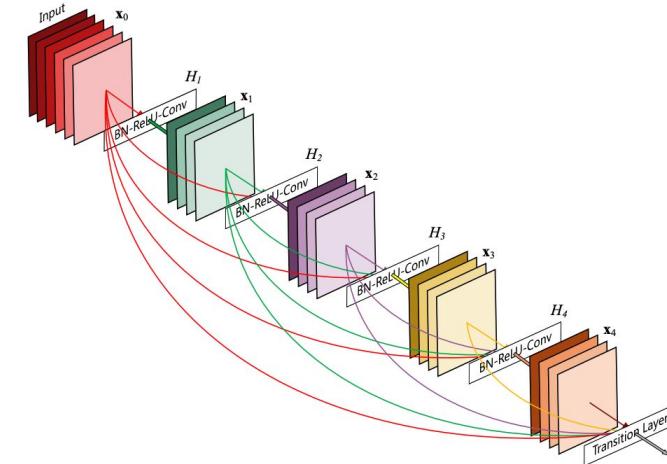


Figure 2. Residual learning: a building block.

# 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)



# Vanishing gradient in non-RNN

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

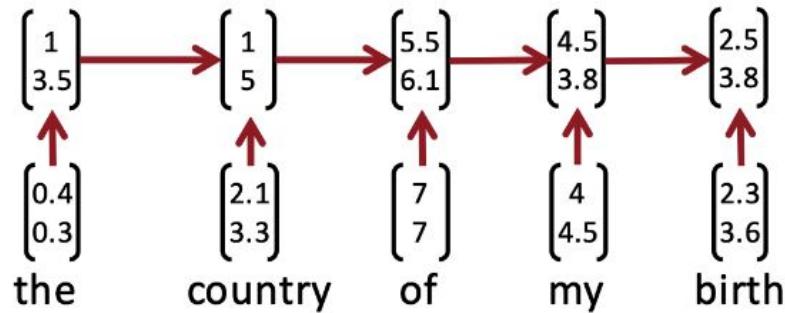
- Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small during backpropagation
- Lower levels are hard to train and are trained slower
- **Potential solution:** dense connections (just like in DenseNet)

## 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]

# Applying CNNs to texts

# From RNN to CNN

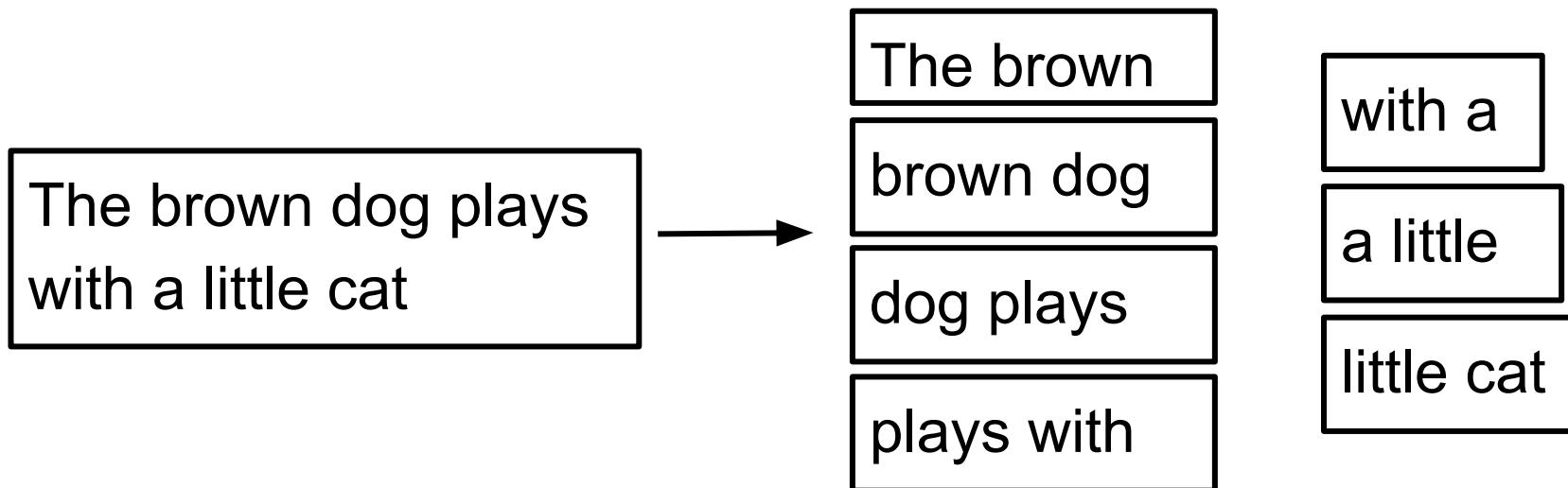


- Recurrent neural nets can not capture phrases without prefix context and often capture too much of last words in final vector

# From RNN to CNN

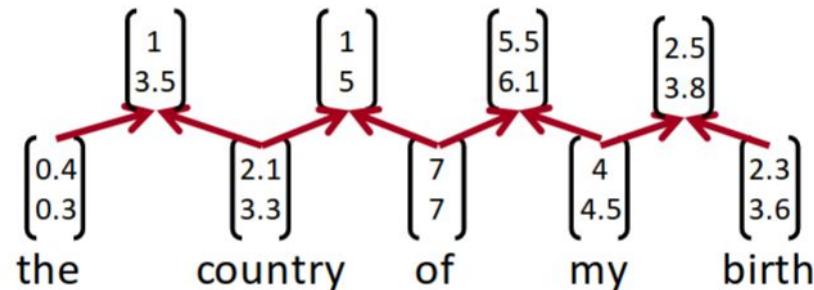
- RNN: Get compositional vectors for grammatical phrases only
- CNN: What if we compute vectors for every possible phrase?
  - Example: “*the country of my birth*” computes vectors for:
    - *the country, country of, of my, my birth, the country of, country of my, of my birth, the country of my, country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

# Recap: n-gramms



# From RNN to CNN

- Imagine using only bigrams



- Same operation as in RNN, but for every pair

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

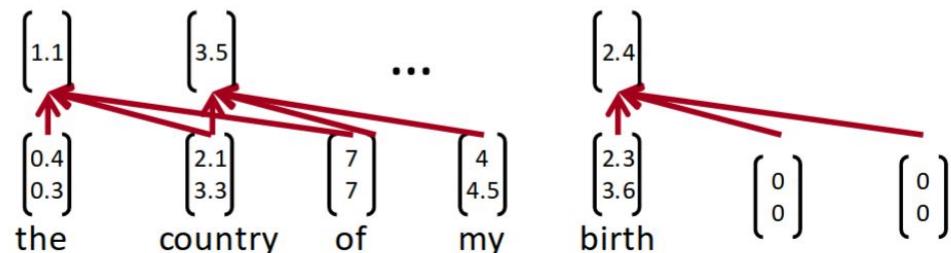
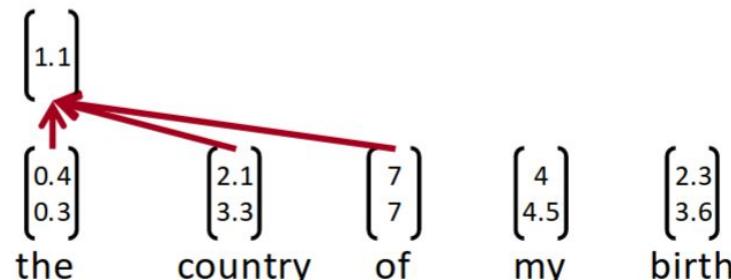
- Can be interpreted as convolution over the word vectors

# 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

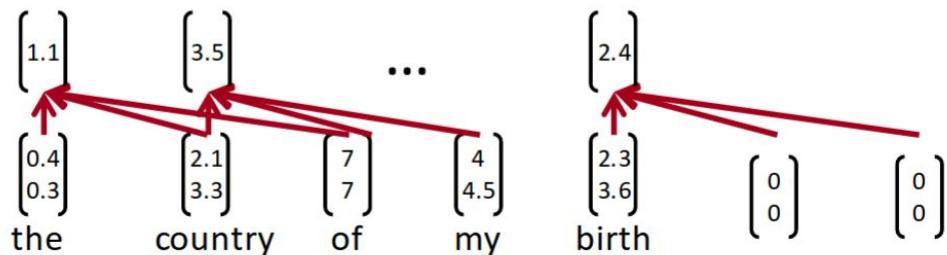
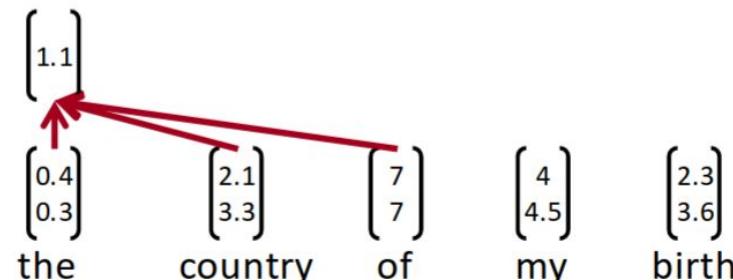
- Simple convolution + pooling
- Window size may be different (2 or more)
- The feature map based on bigrams:

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

What's next?

We need more features!

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

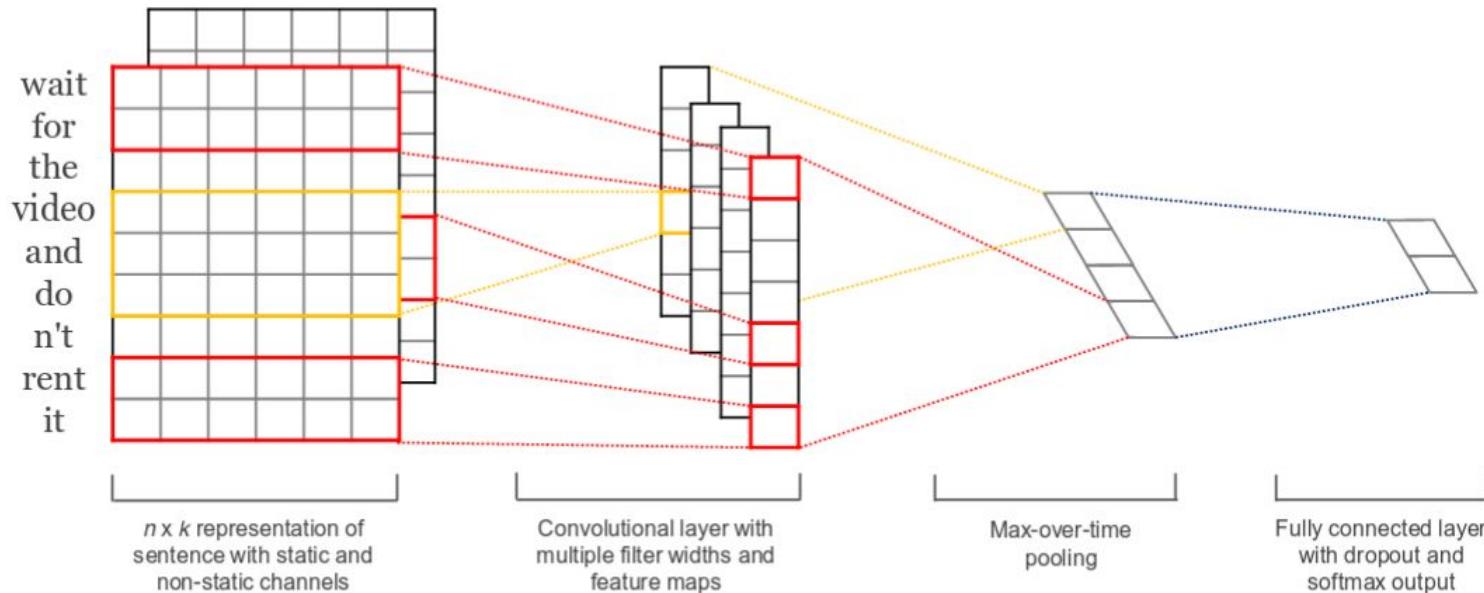


- Feature representation is based on some applied filter:

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

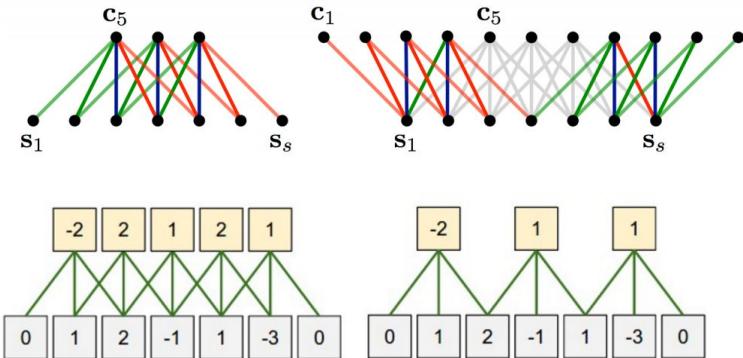
- Let's use pooling:  $\hat{c} = \max\{\mathbf{c}\}$
- Now the length of  $\mathbf{c}$  is irrelevant!
- So we can use filters based on unigrams, bigrams, tri-grams, 4-grams, etc.

# Another example from Kim (2014) paper

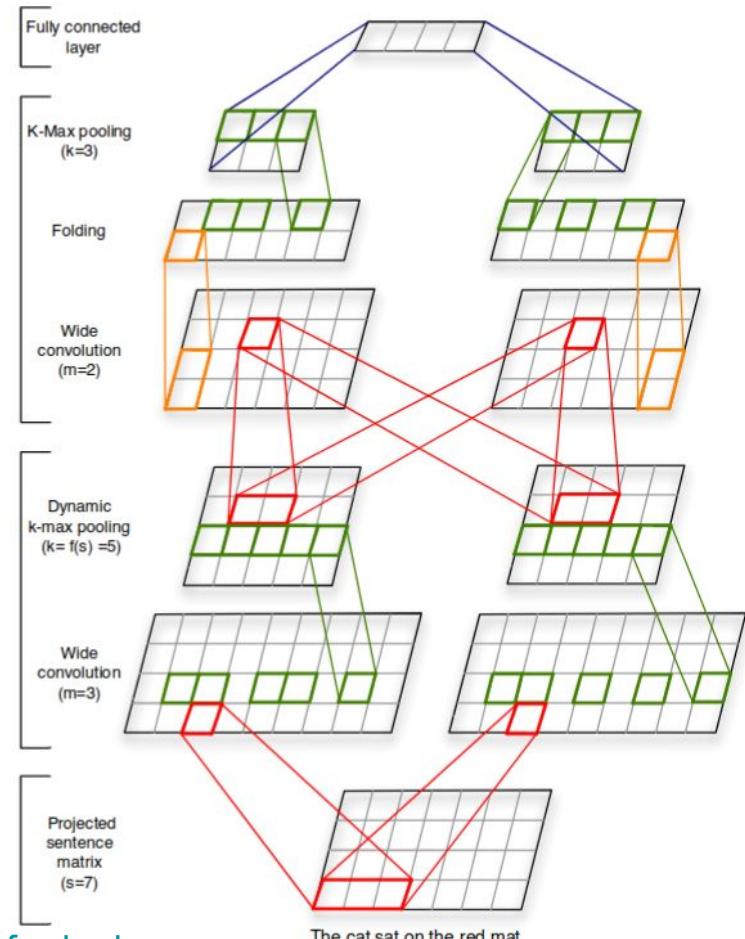


# More about CNN

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

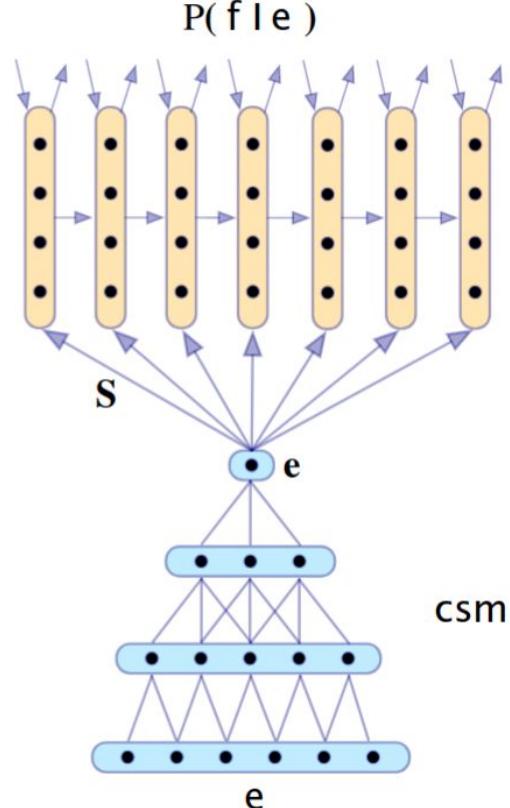


- Complex pooling schemes over sequences
- Great readings (e.g. Kalchbrenner et. al. 2014)



# 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



# Approaches comparison

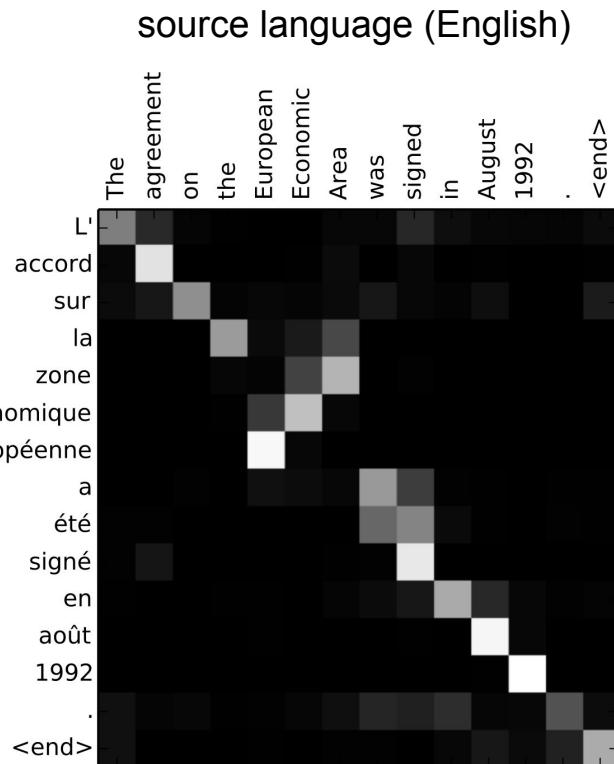
Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	<b>89.6</b>
CNN-non-static	<b>81.5</b>	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	<b>88.1</b>	93.2	92.2	<b>85.0</b>	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	—	—	—	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	—	—	—	—
RNTN (Socher et al., 2013)	—	45.7	85.4	—	—	—	—
DCNN (Kalchbrenner et al., 2014)	—	48.5	86.8	—	93.0	—	—
Paragraph-Vec (Le and Mikolov, 2014)	—	<b>48.7</b>	87.8	—	—	—	—
CCAE (Hermann and Blunsom, 2013)	77.8	—	—	—	—	—	87.2
Sent-Parser (Dong et al., 2014)	79.5	—	—	—	—	—	86.3
NBSVM (Wang and Manning, 2012)	79.4	—	—	93.2	—	81.8	86.3
MNB (Wang and Manning, 2012)	79.0	—	—	<b>93.6</b>	—	80.0	86.3
G-Dropout (Wang and Manning, 2013)	79.0	—	—	93.4	—	82.1	86.1
F-Dropout (Wang and Manning, 2013)	79.1	—	—	<b>93.6</b>	—	81.9	86.3
Tree-CRF (Nakagawa et al., 2010)	77.3	—	—	—	—	81.4	86.1
CRF-PR (Yang and Cardie, 2014)	—	—	—	—	—	82.7	—
SVM <sub>S</sub> (Silva et al., 2011)	—	—	—	—	<b>95.0</b>	—	—

- 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 efficient
- Clip your gradients
- Combining RNN and CNN worlds? Why not ;)

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

# Attention outro

target language (French)



target language (French)

