

# CS 559 Machine Learning

## Lecture 12: Neural Networks II

Ping Wang  
Department of Computer Science  
Stevens Institute of Technology



# Today's Lecture

- Convolutional Neural Networks (CNN)
- Recurrent Neural Network (RNN)
- Word2Vec

# Why CNN for Image

- Some patterns are much smaller than the whole image.

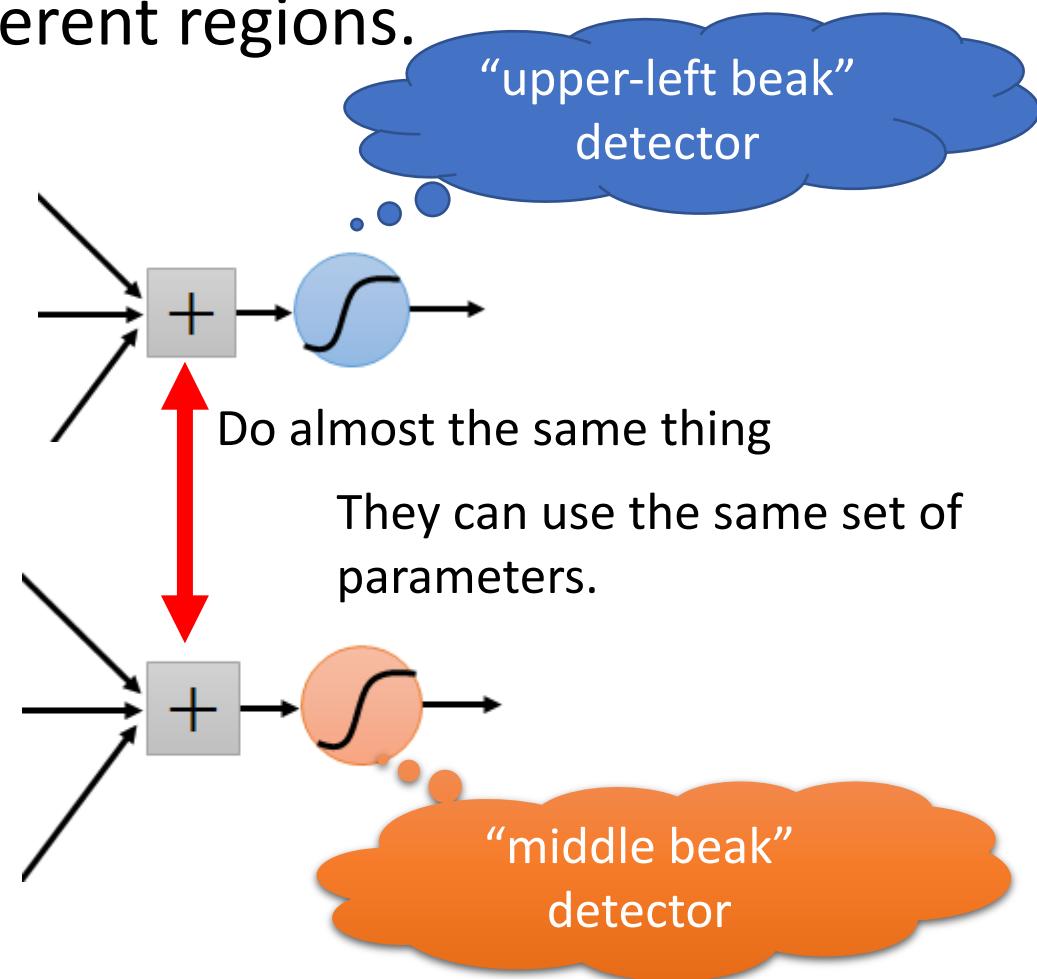
A neuron does not have to see the whole image to discover the pattern.

Connecting to small region with less parameters



# Why CNN for Image

- The same patterns appear in different regions.



# Why CNN for Image

- Subsampling the pixels will not change the object

bird



subsampling

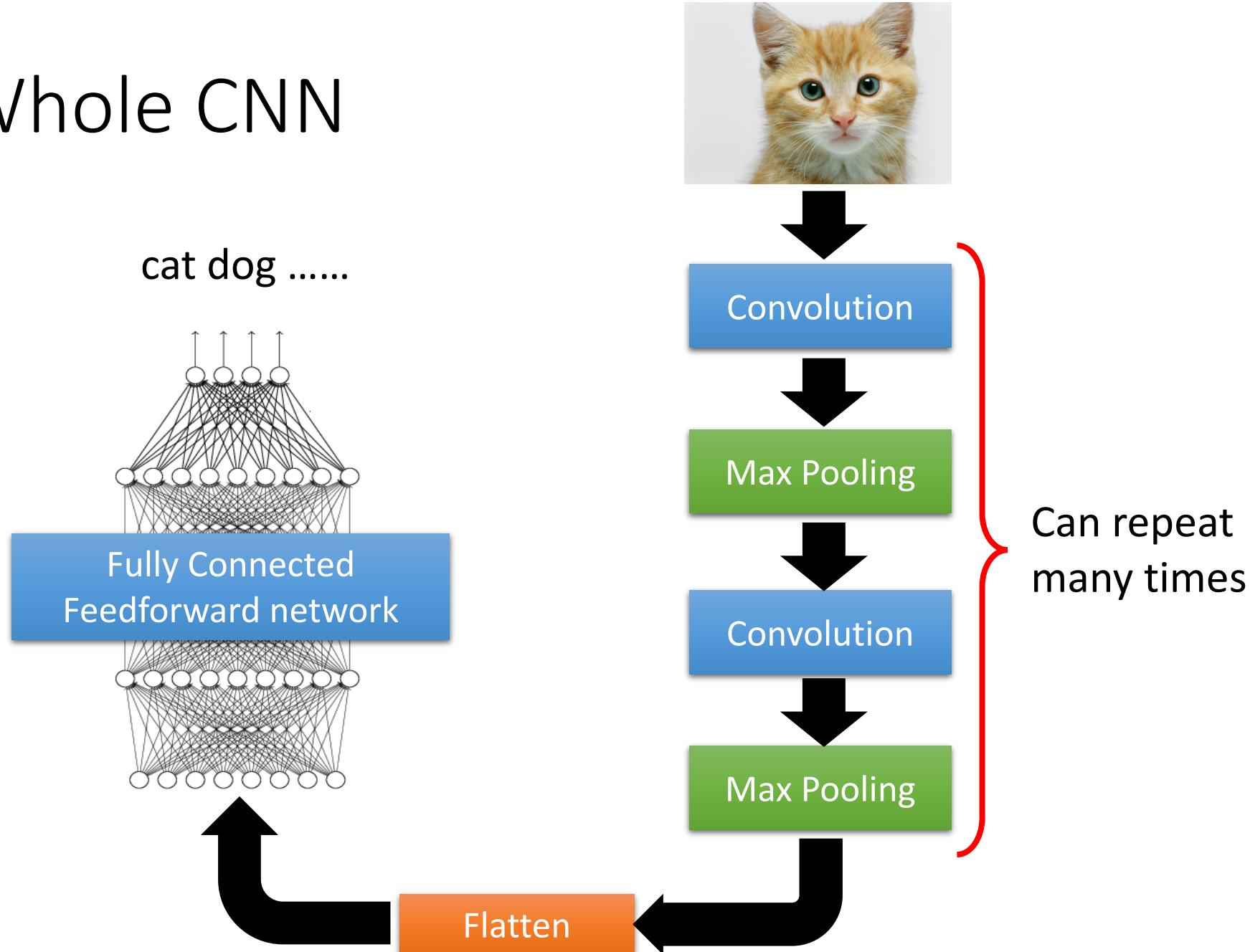
bird



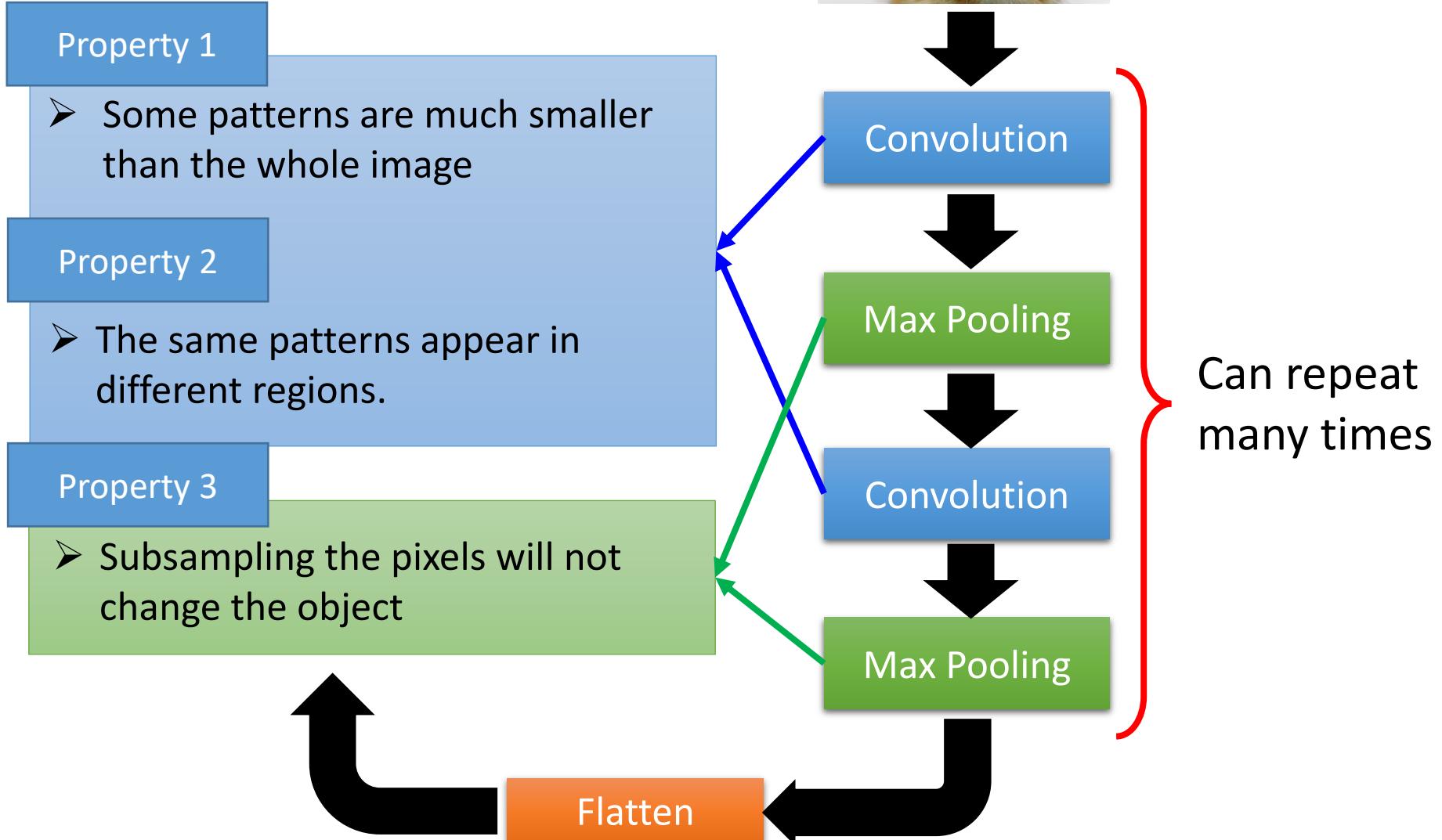
We can subsample the pixels to make image smaller

→ Less parameters for the network to process the image

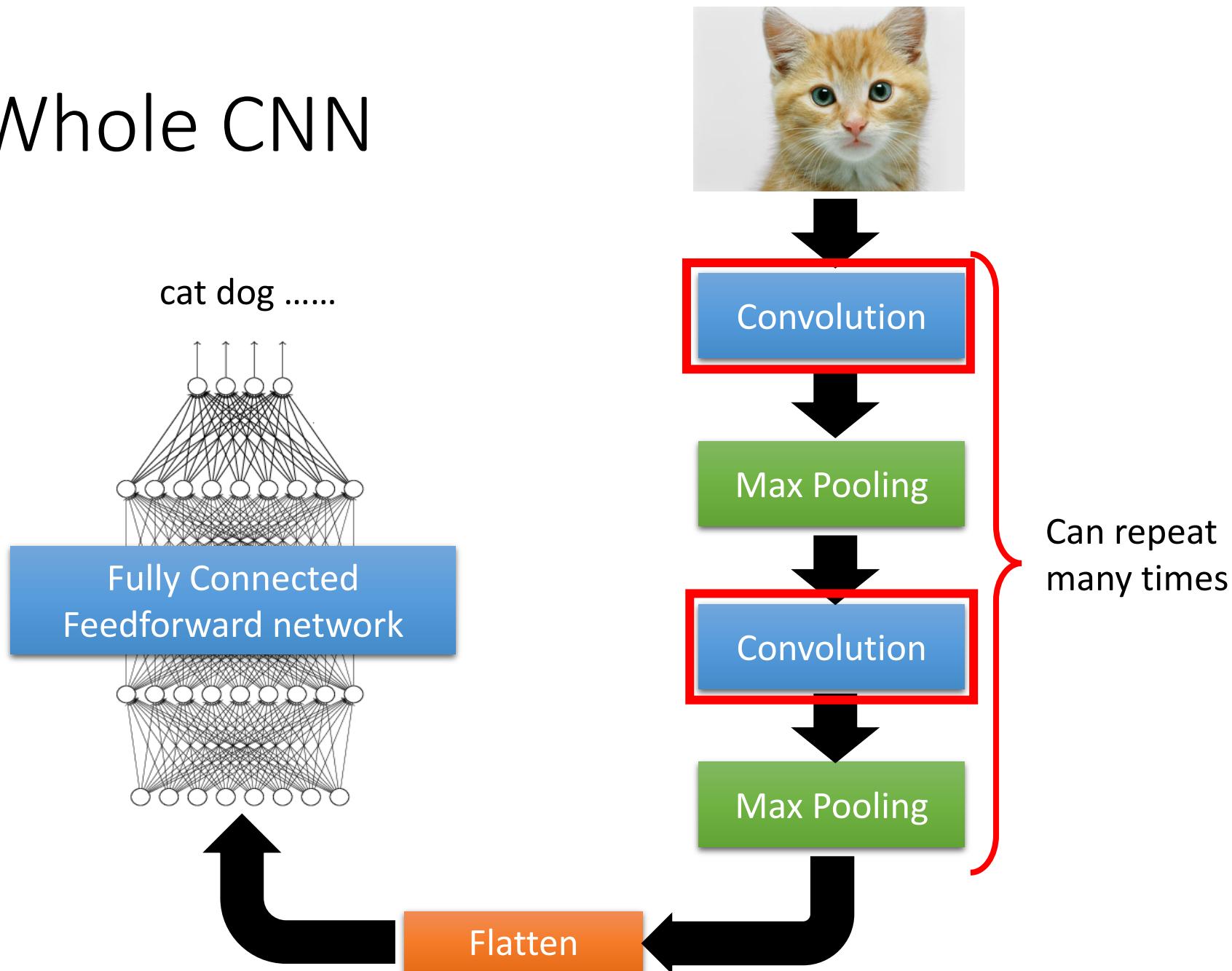
# The Whole CNN



# The Whole CNN



# The Whole CNN



# CNN – Convolution

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Property 1: Some patterns are much smaller than the whole image

Those are the network parameters to be learned.

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1  
Matrix

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2  
Matrix

⋮

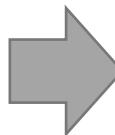
Each filter detects a small pattern (3 x 3).

# CNN – Convolution

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

3



1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

3 -1

6 x 6 image

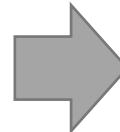
6 x 6 image

# CNN – Convolution

stride=2

1	0	0	0	0	0	1
0	1	0	0	1	0	0
0	0	1	1	0	0	0
1	0	0	0	1	0	0
0	1	0	0	1	0	0
0	0	1	0	1	0	0

3



1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

3 -3

6 x 6 image

6 x 6 image

We set stride=1 below

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

# CNN – Convolution

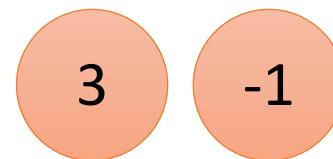
stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1



# CNN – Convolution

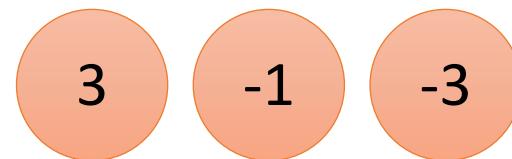
stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1



# CNN – Convolution

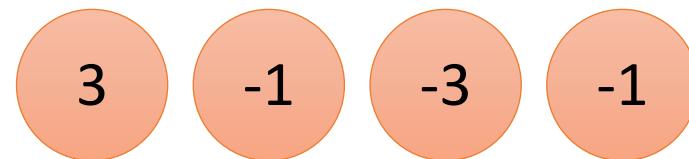
stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1



# CNN – Convolution

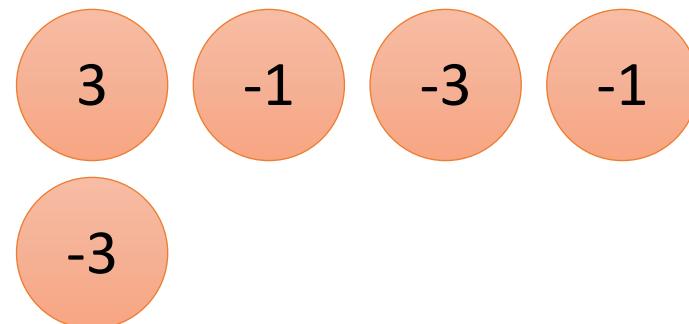
stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1



# CNN – Convolution

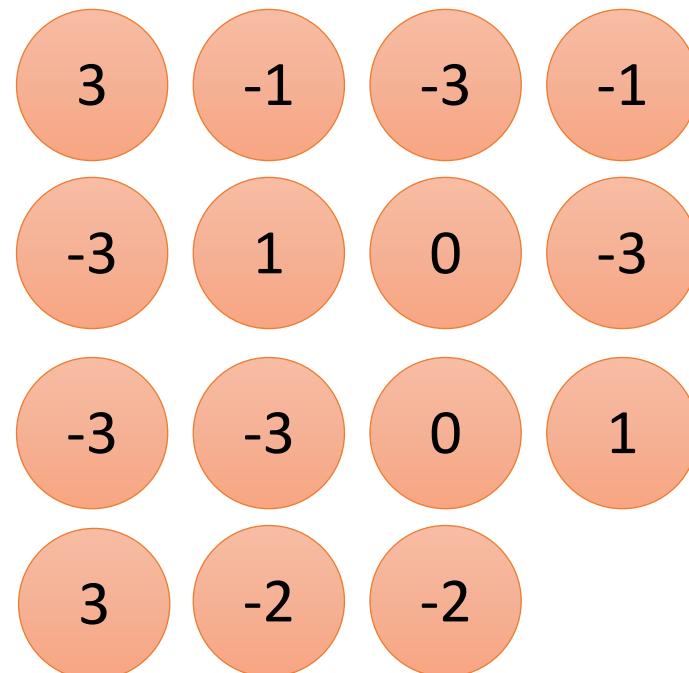
stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1



# CNN – Convolution

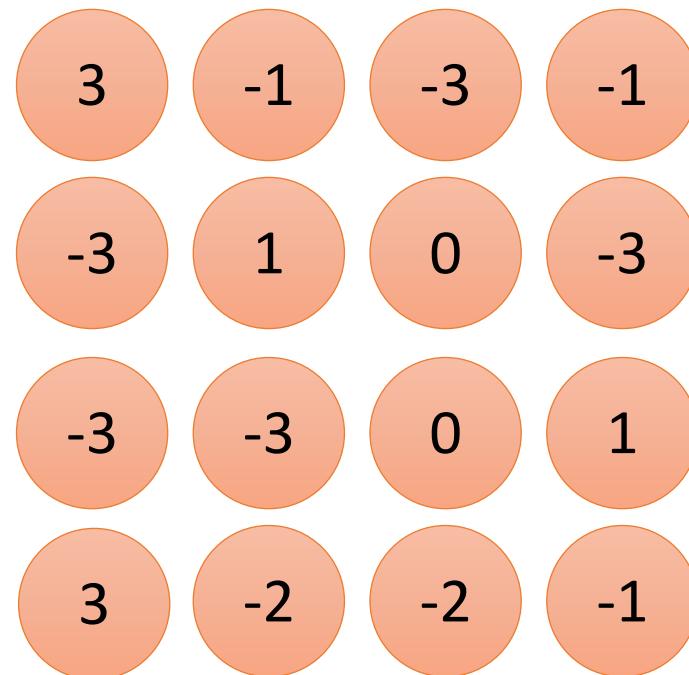
stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

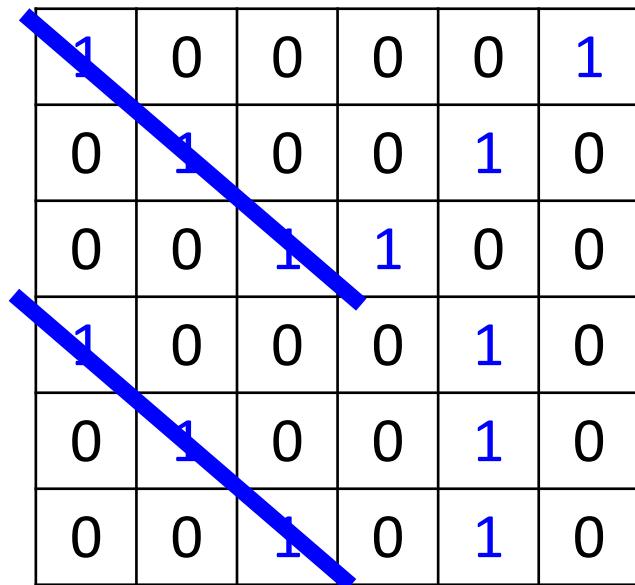
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1



# CNN – Convolution

stride=1

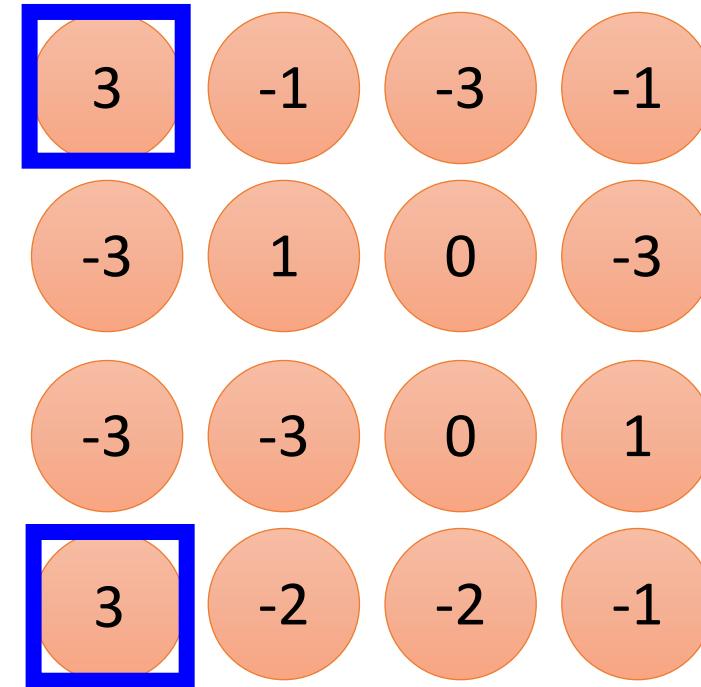


6 x 6 image

Property 2: The same patterns appear in different regions.

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1



# CNN – Convolution

stride=1

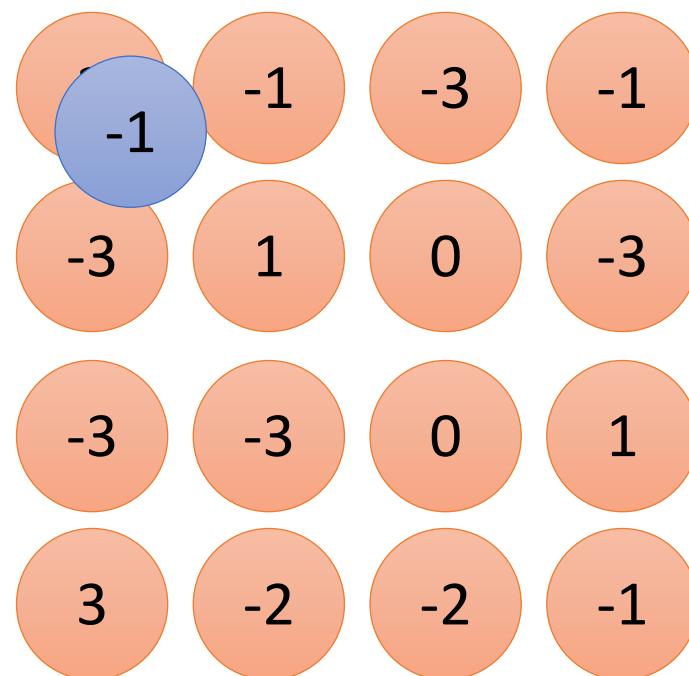
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

Do the same process for every filter



4 x 4 image

# CNN – Convolution

stride=1

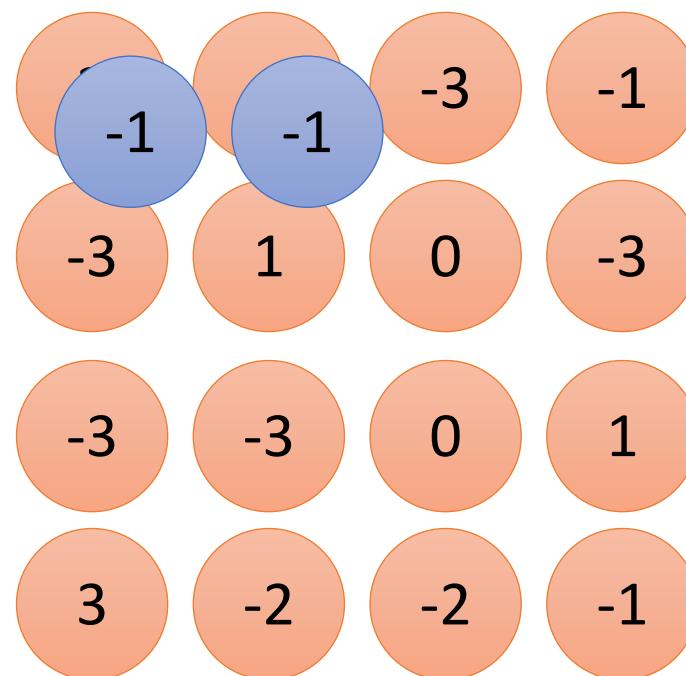
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

Do the same process for every filter



4 x 4 image

# CNN – Convolution

stride=1

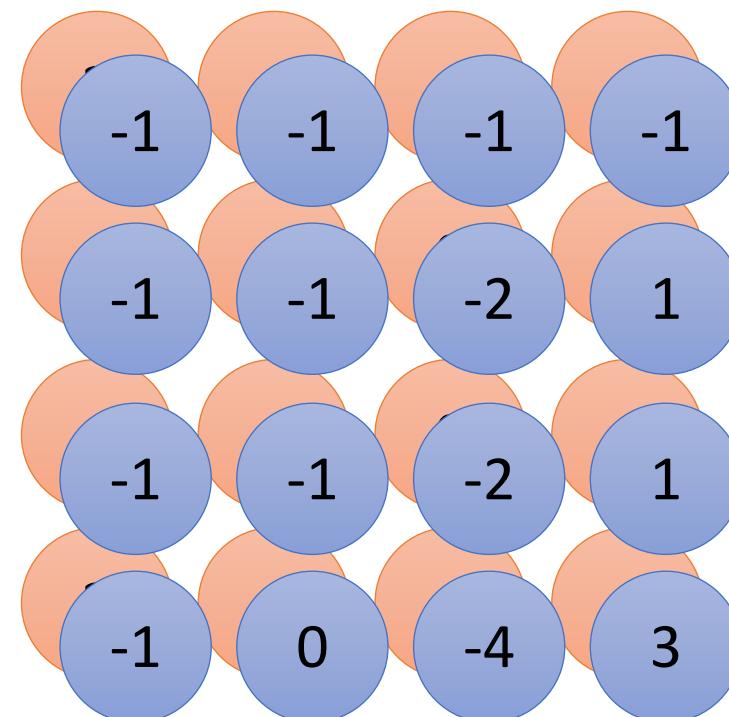
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

Do the same process for every filter



4 x 4 image

# CNN – Convolution

stride=1

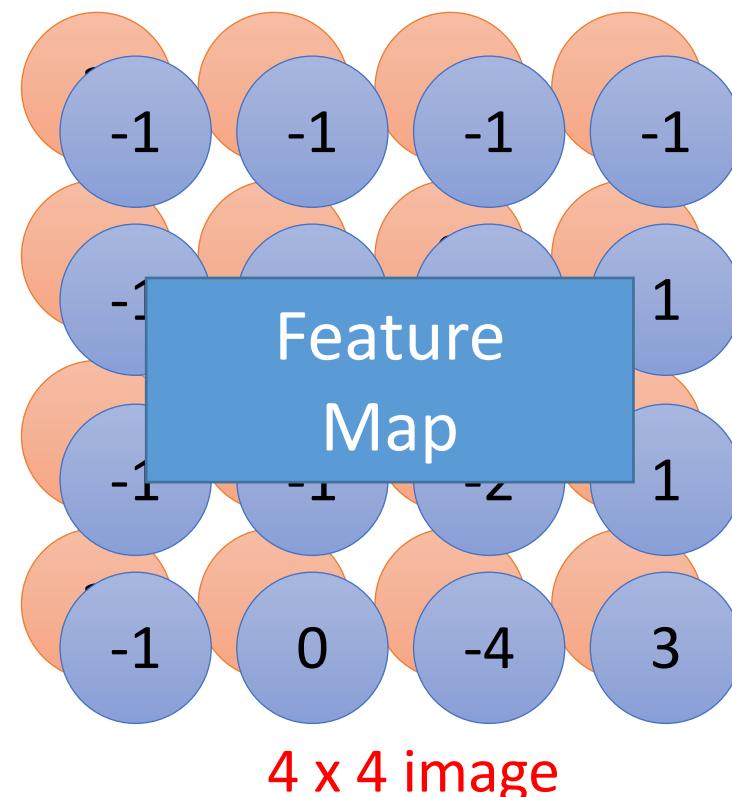
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

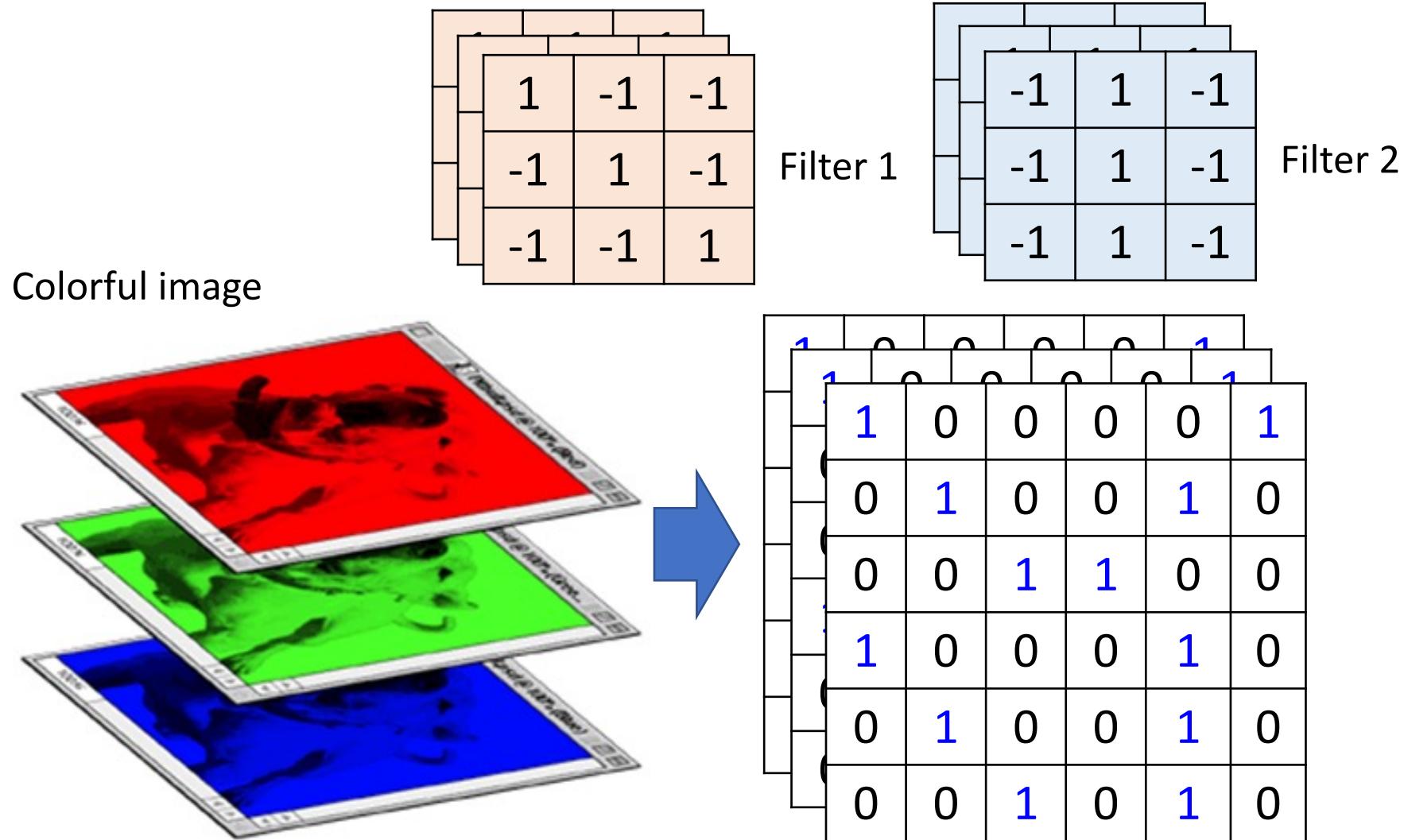
-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

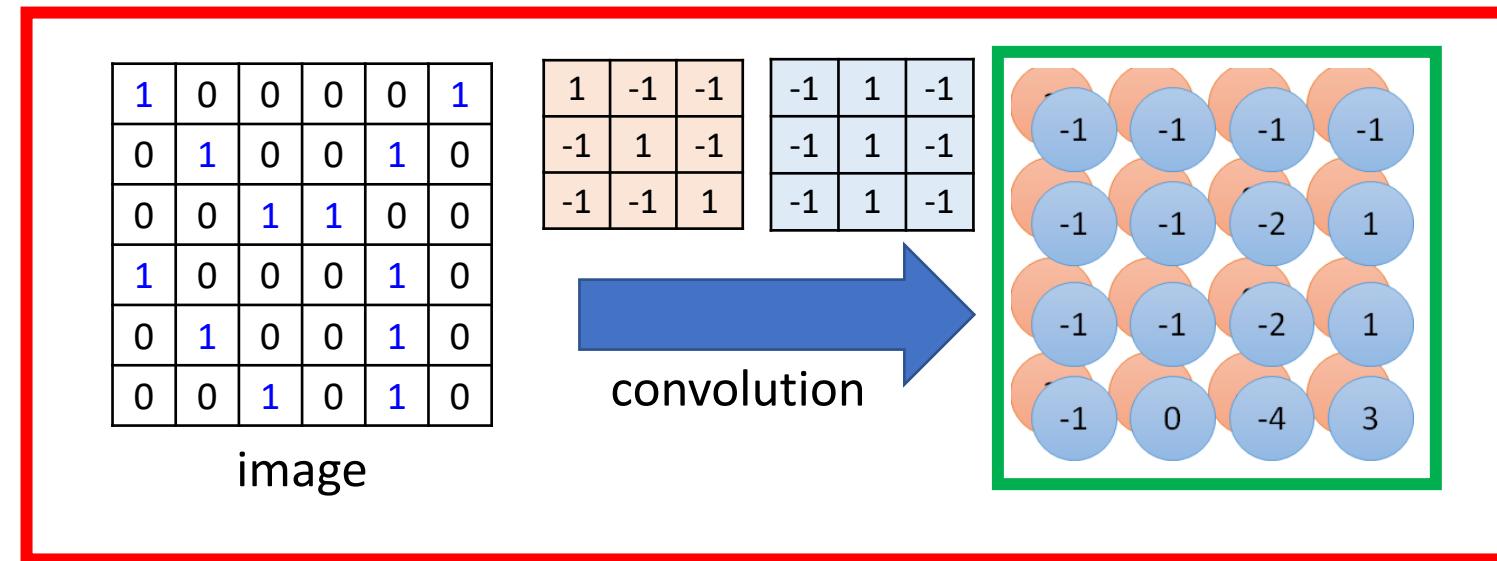
Do the same process for every filter



# CNN – Colorful Image

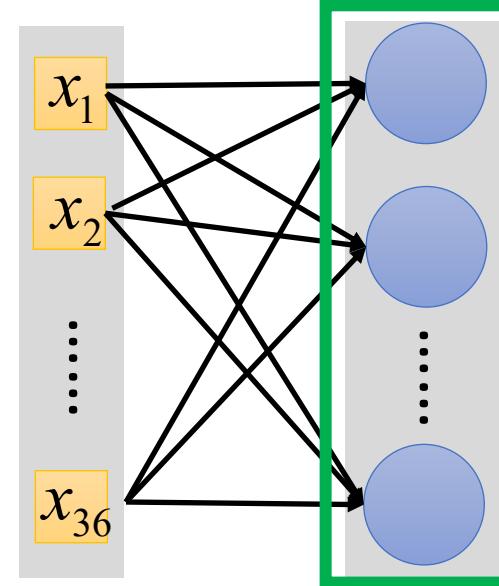


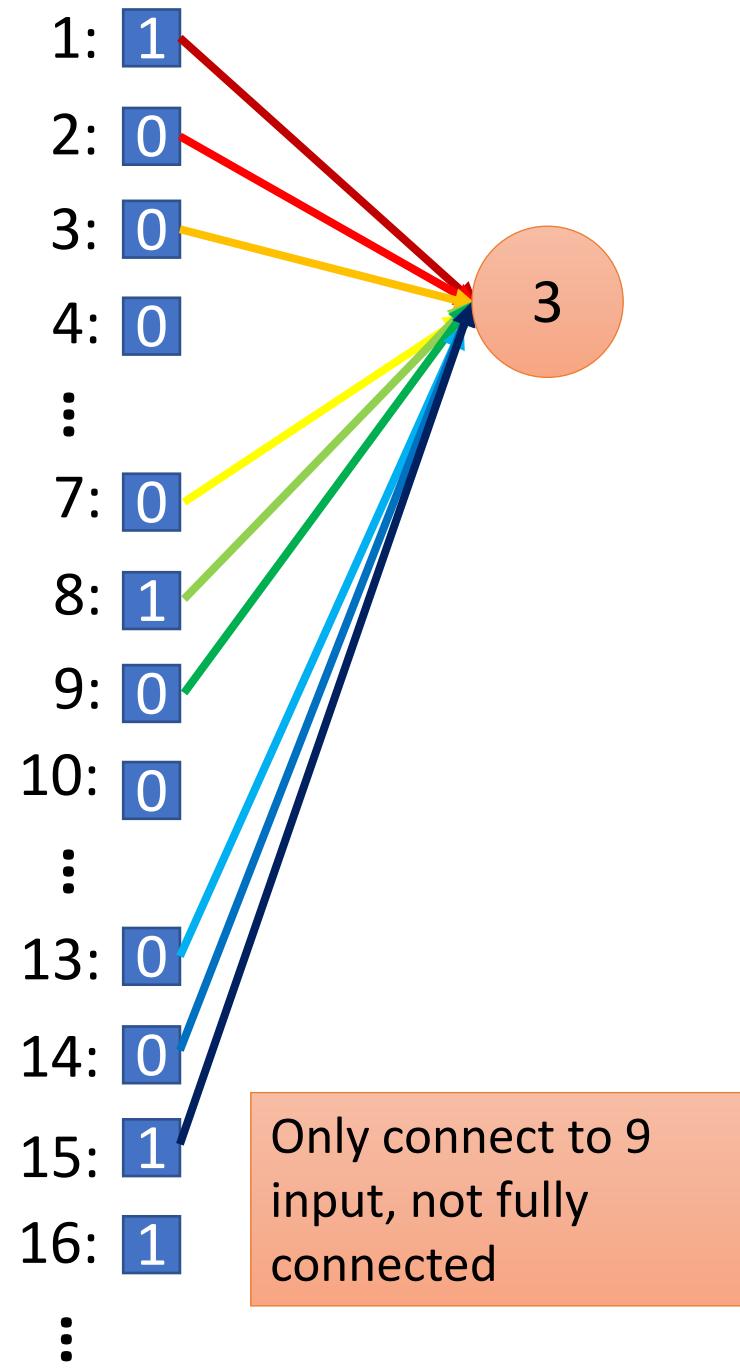
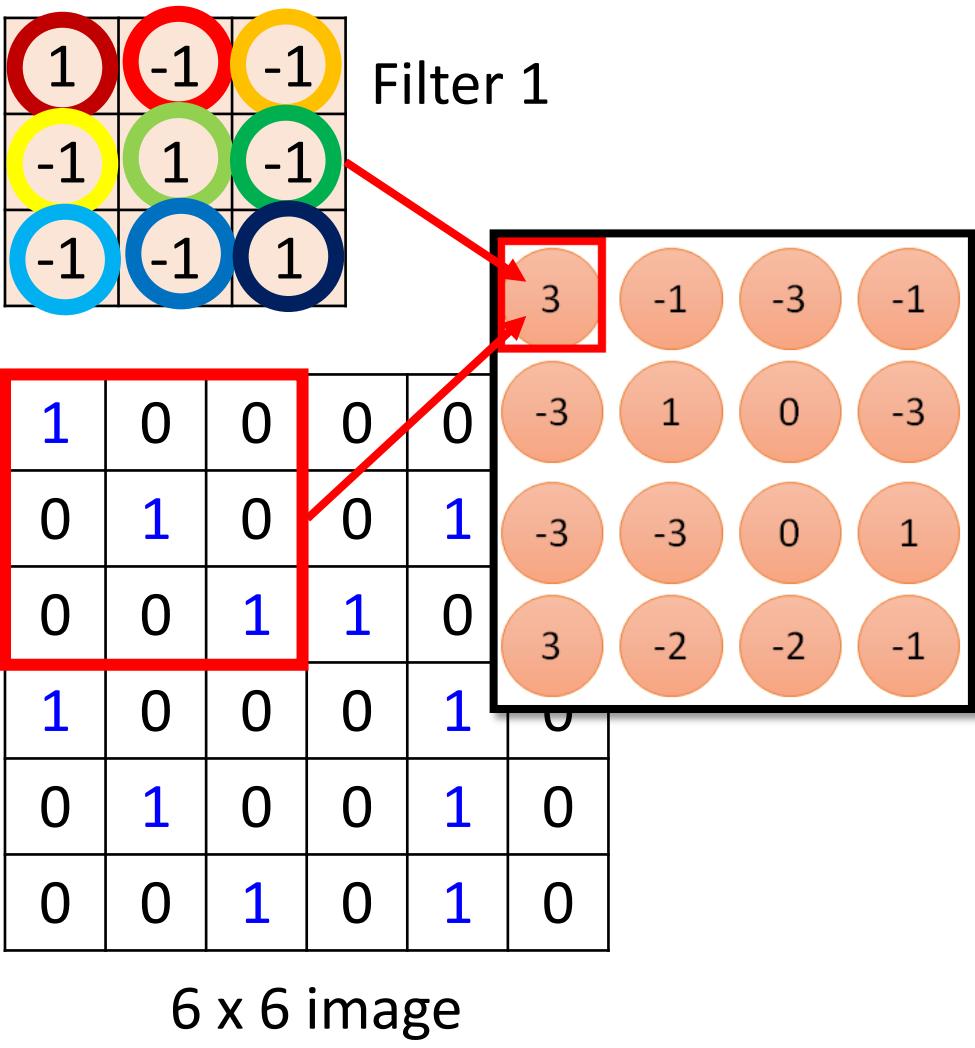
# Convolution v.s. Fully Connected

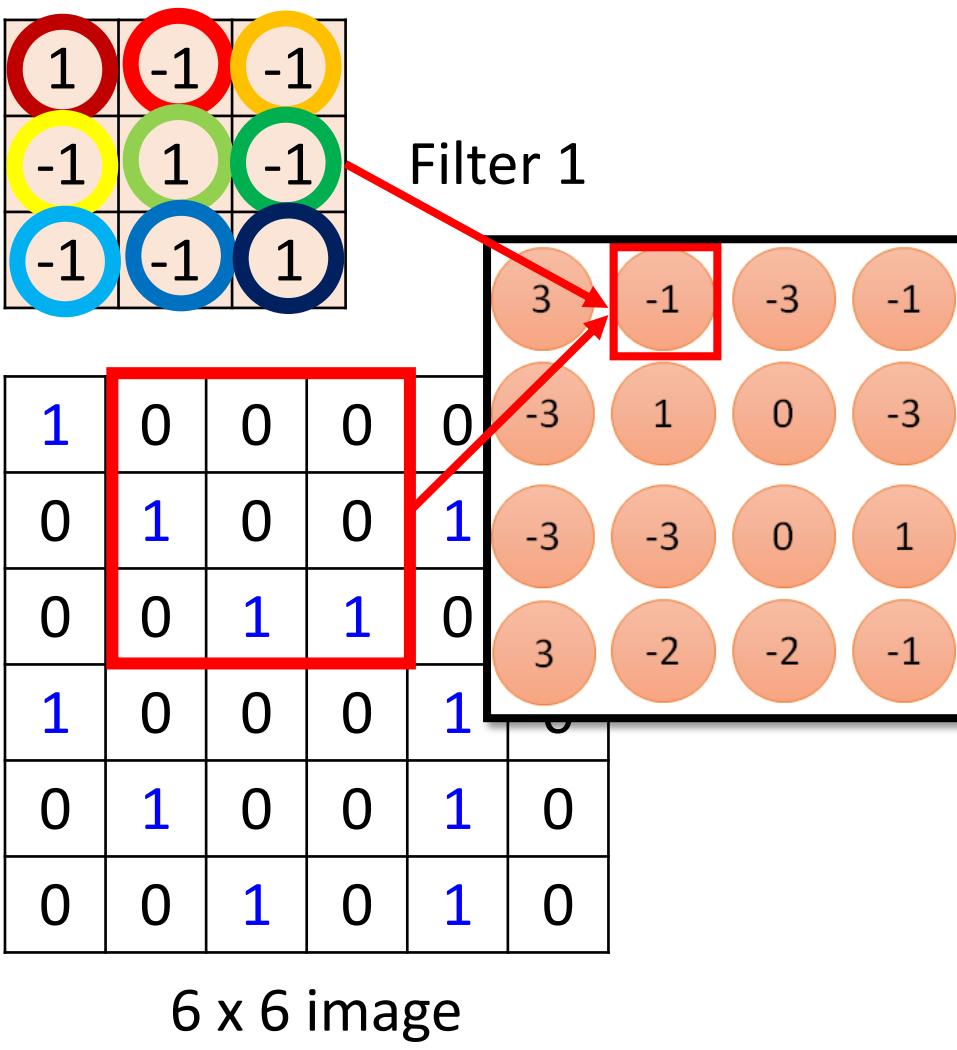


Fully-connected

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

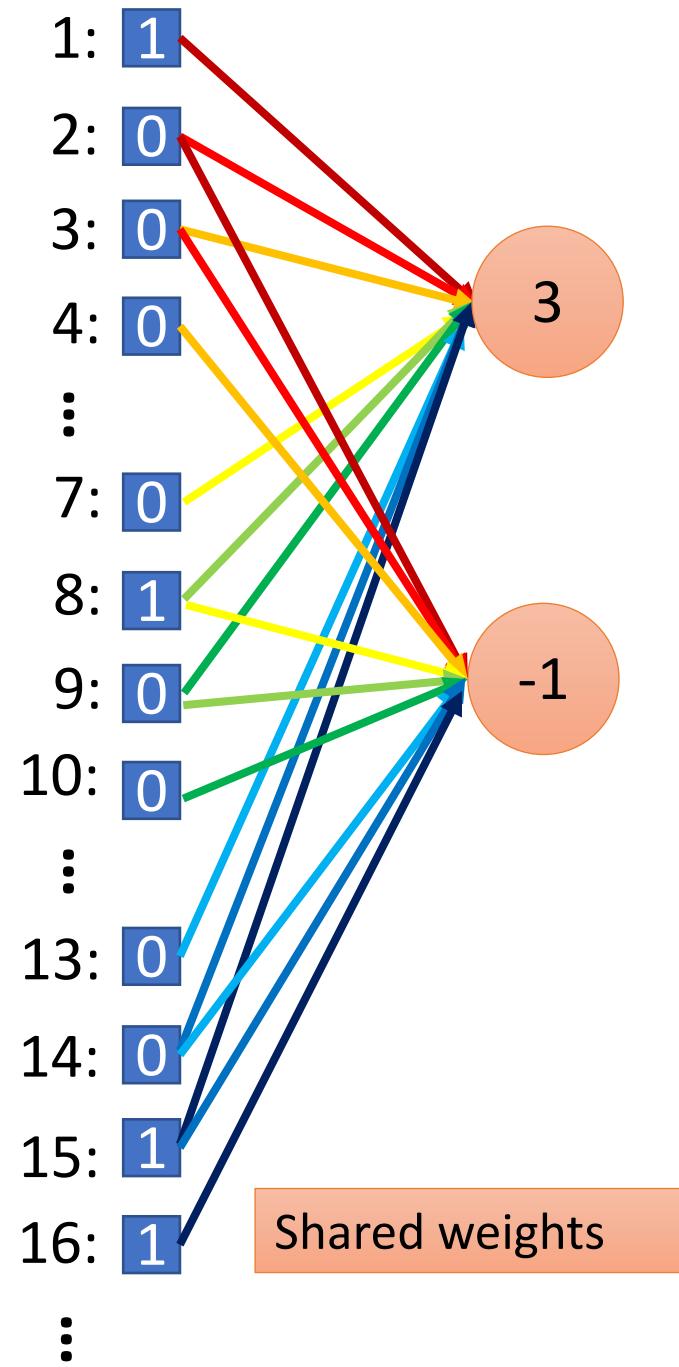




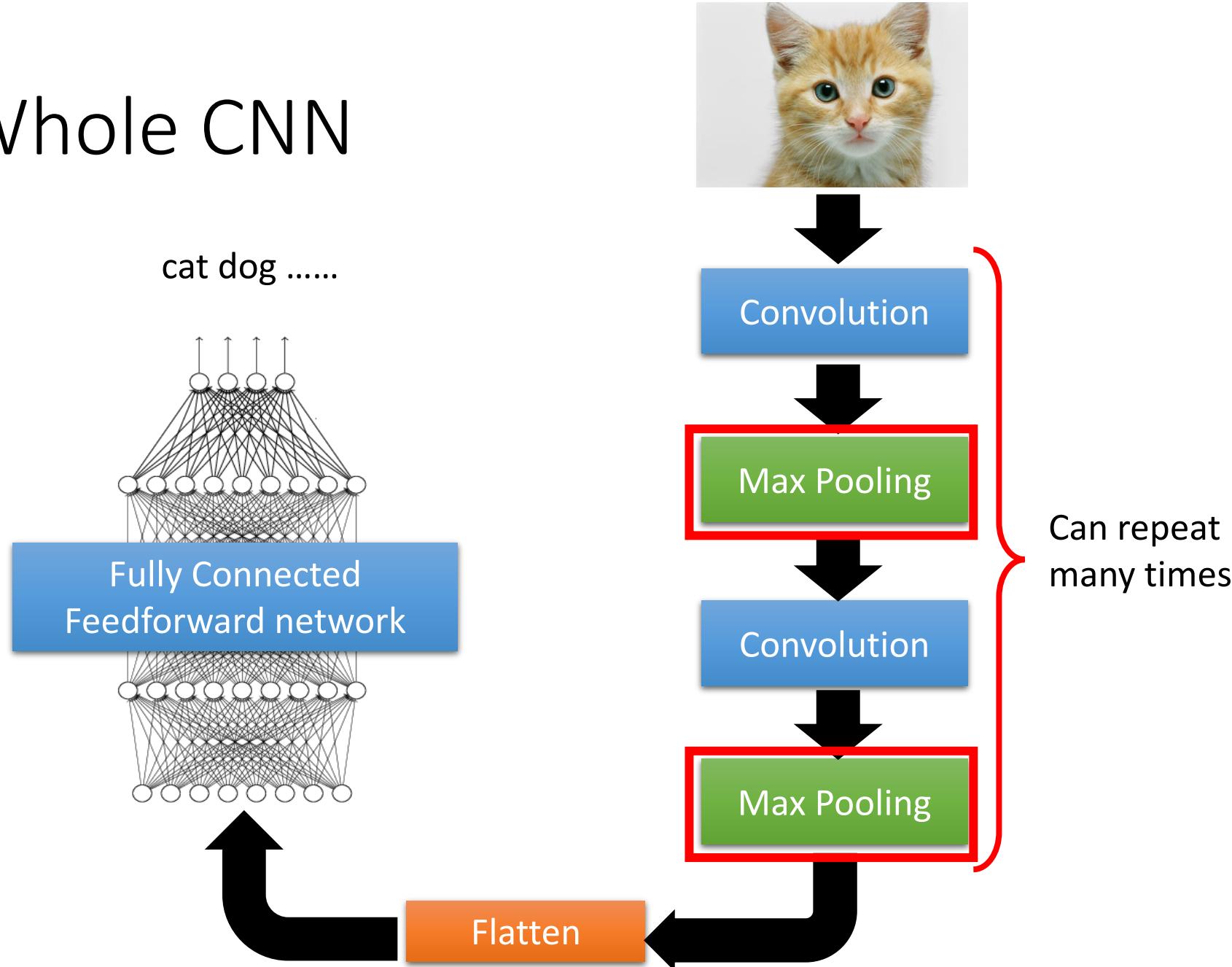


Less parameters!

Even less parameters!



# The Whole CNN



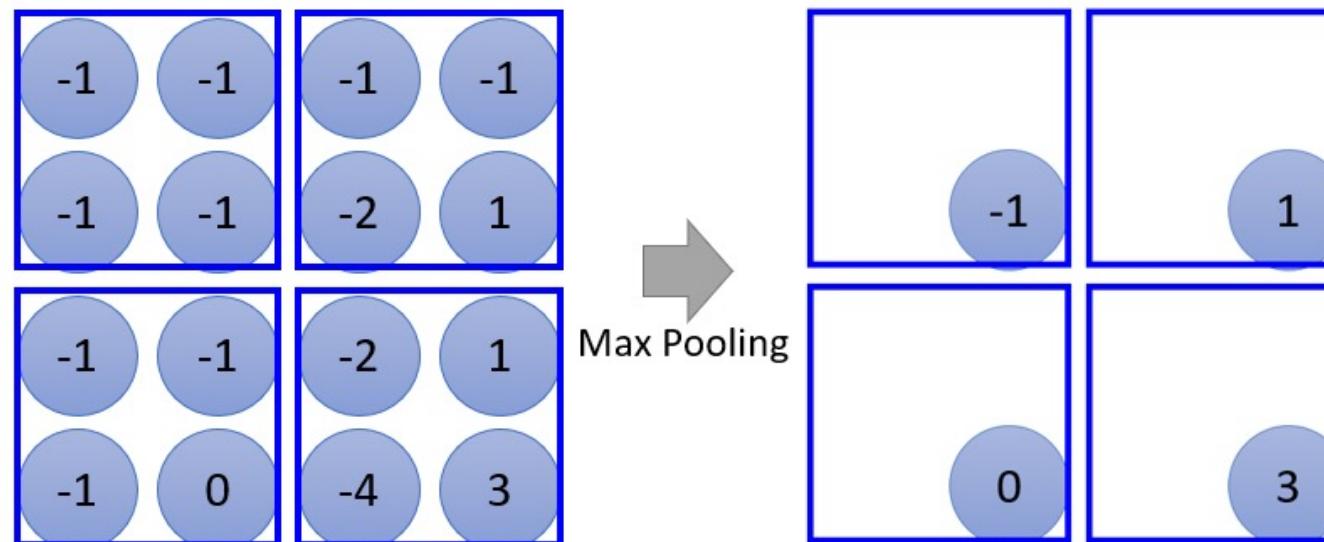
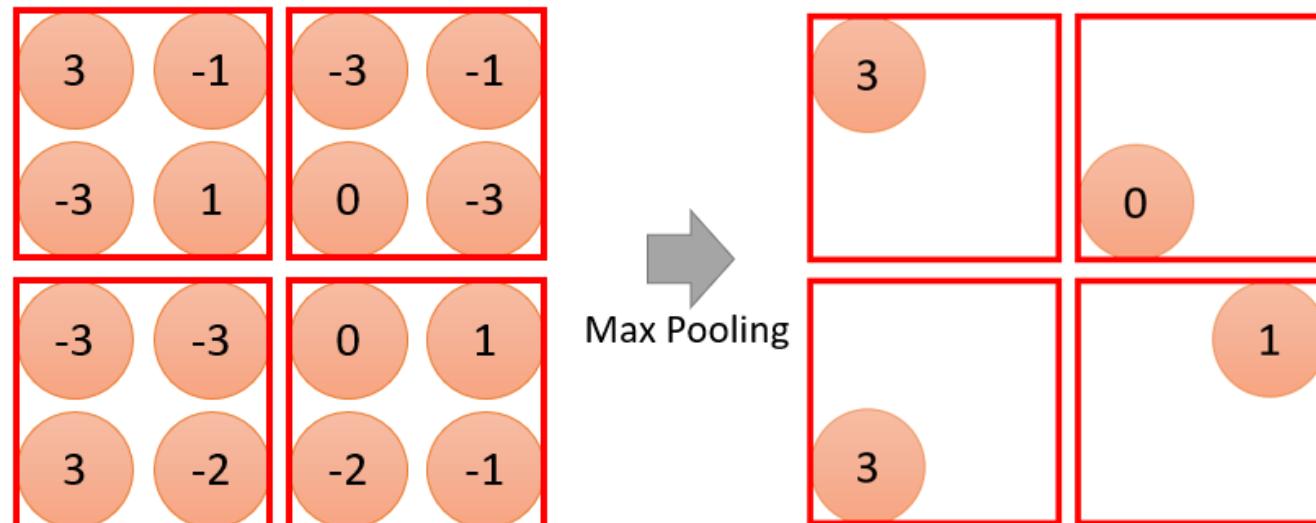
# CNN – Max Pooling

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

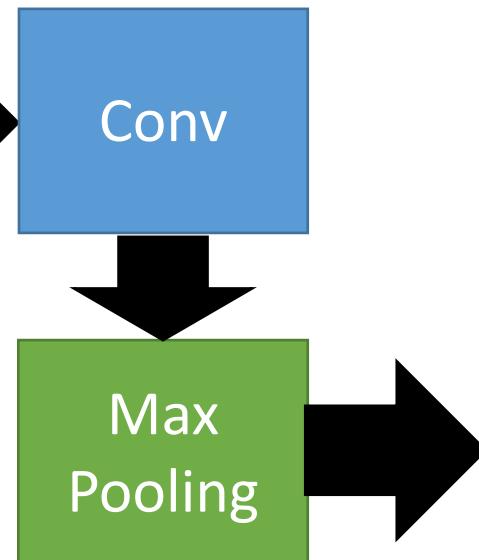
Filter 2



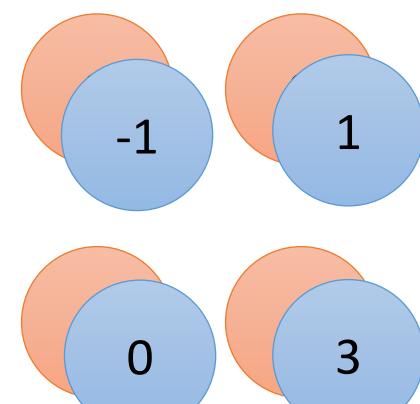
# CNN – Max Pooling

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image



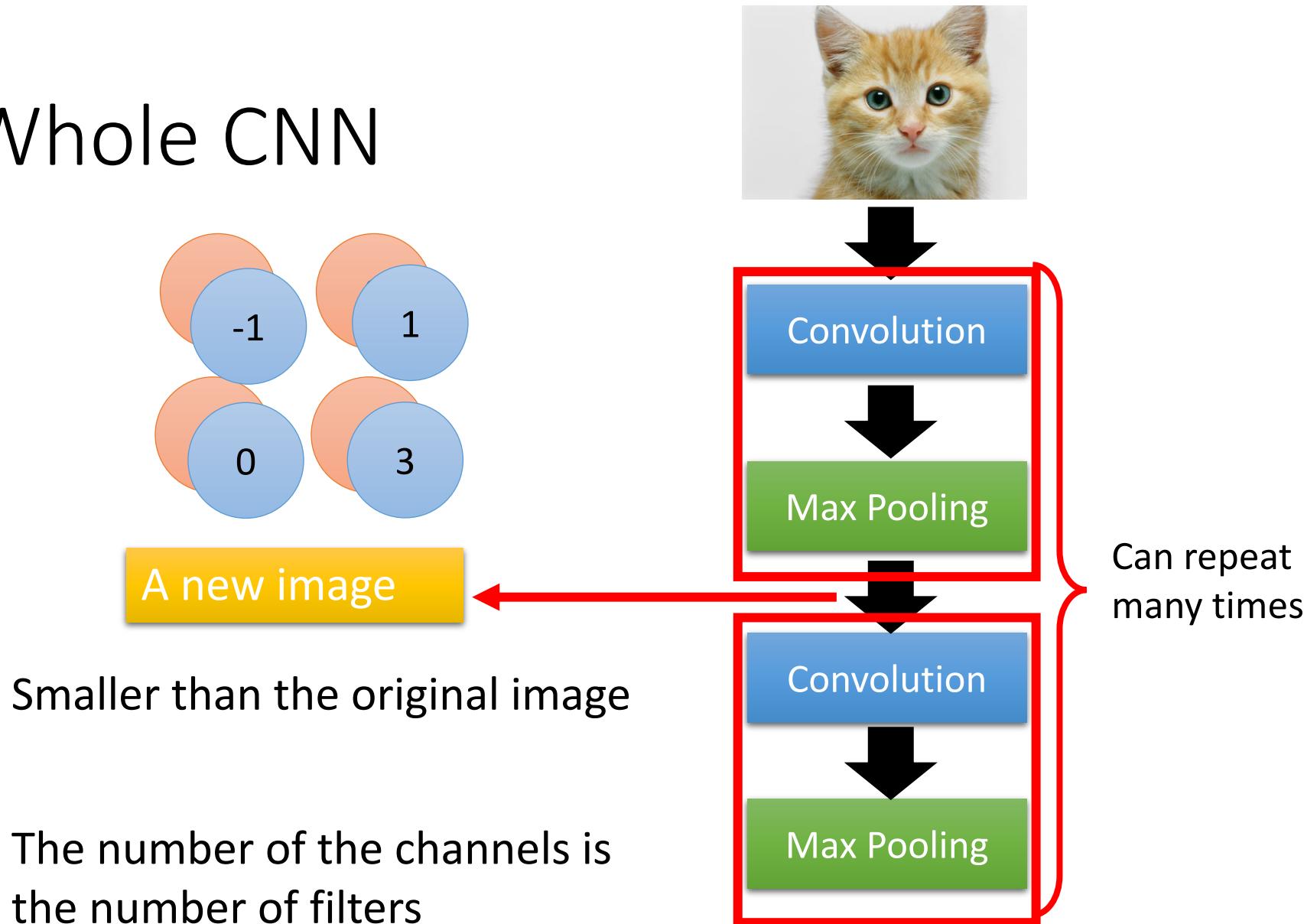
New image  
but smaller



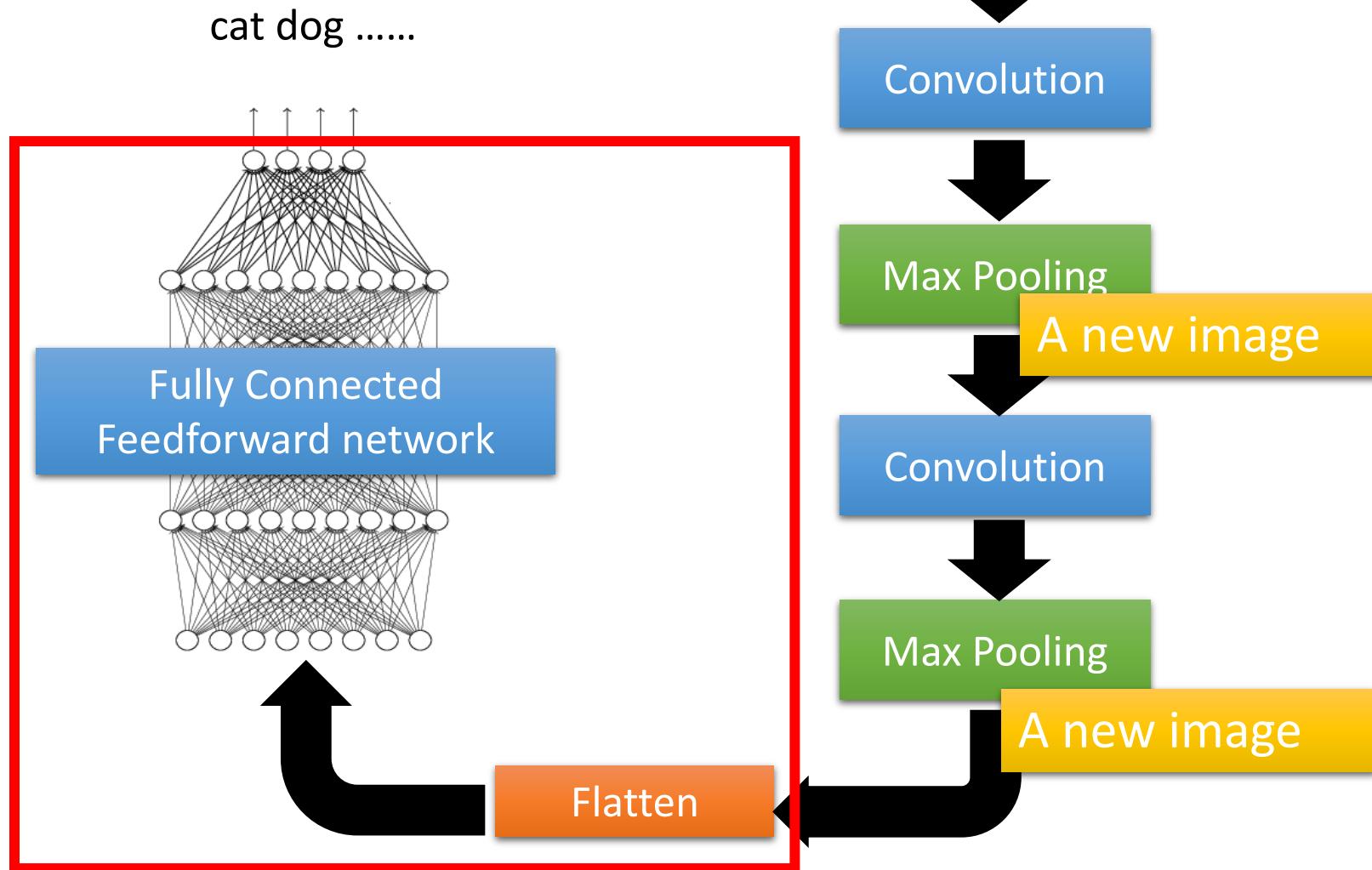
2 x 2 image

Each filter  
is a channel

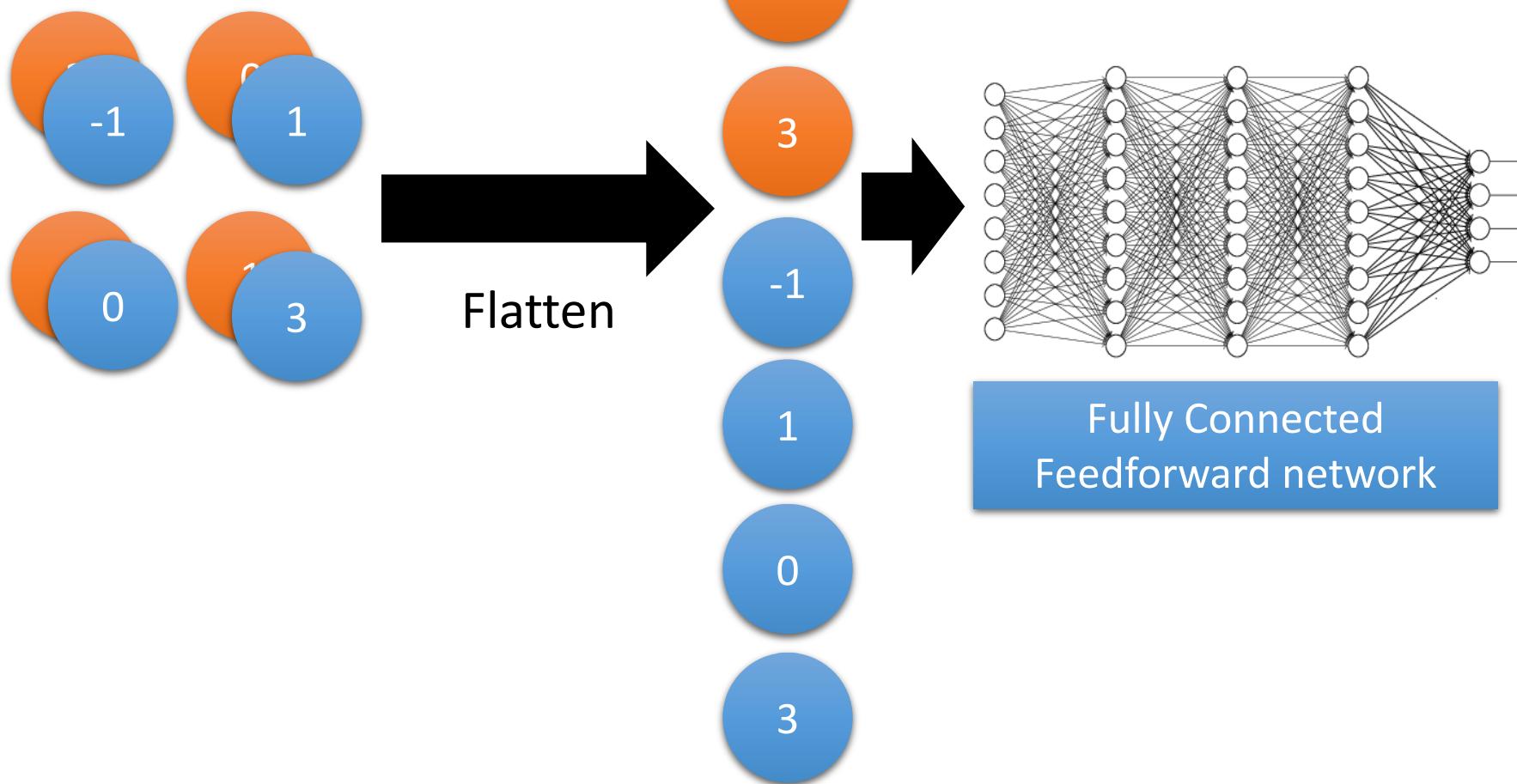
# The Whole CNN



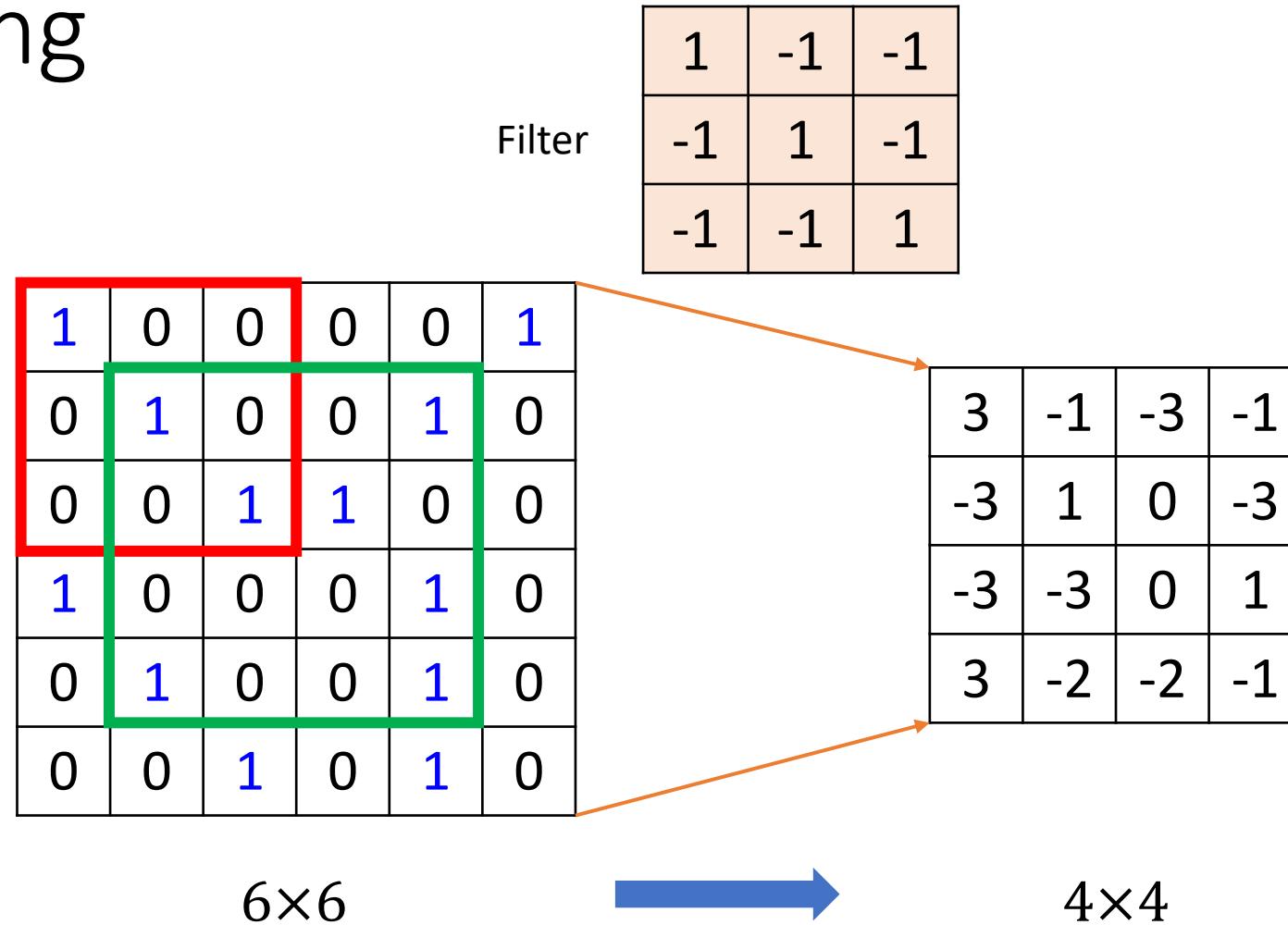
# The Whole CNN



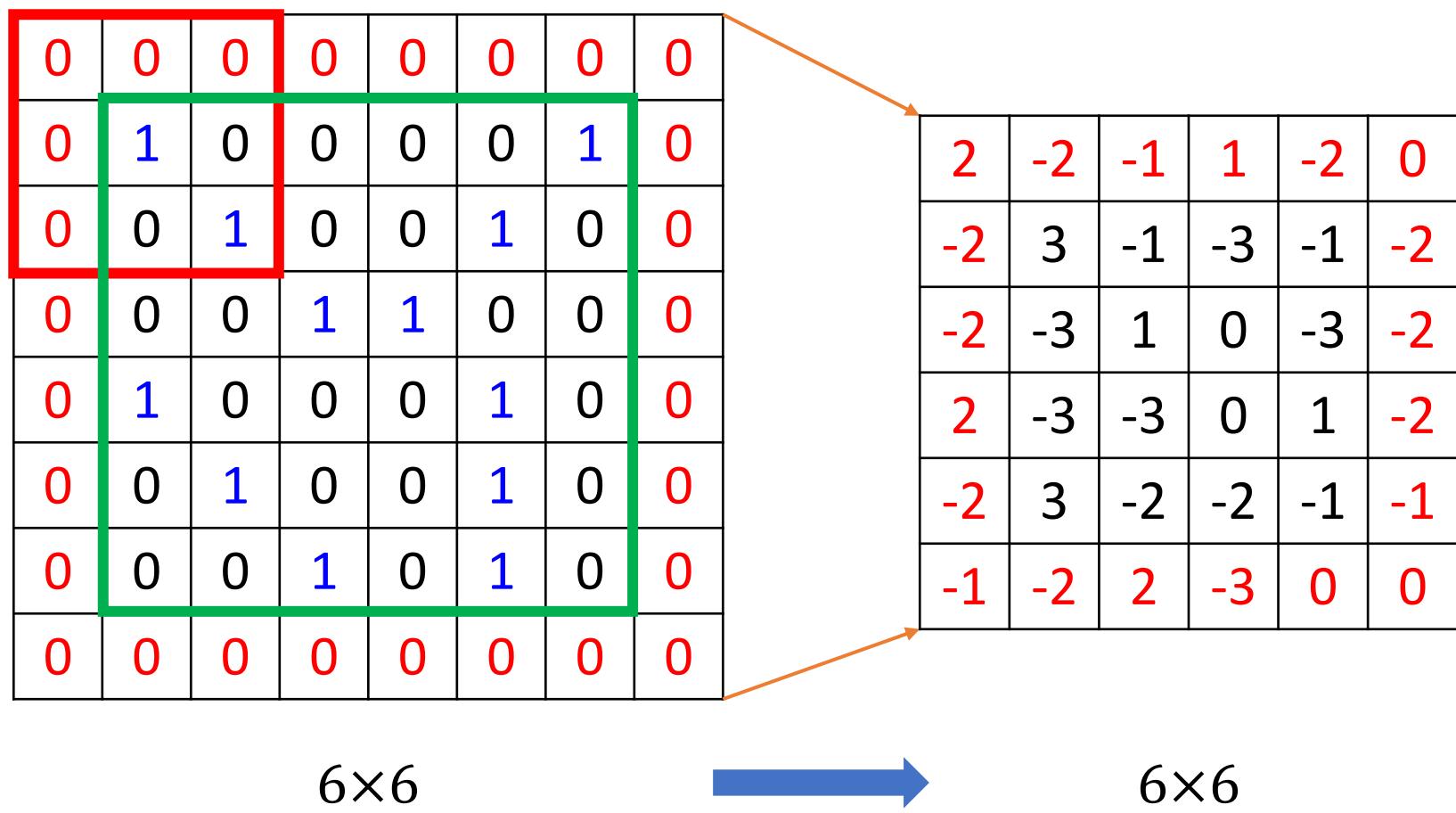
# Flatten



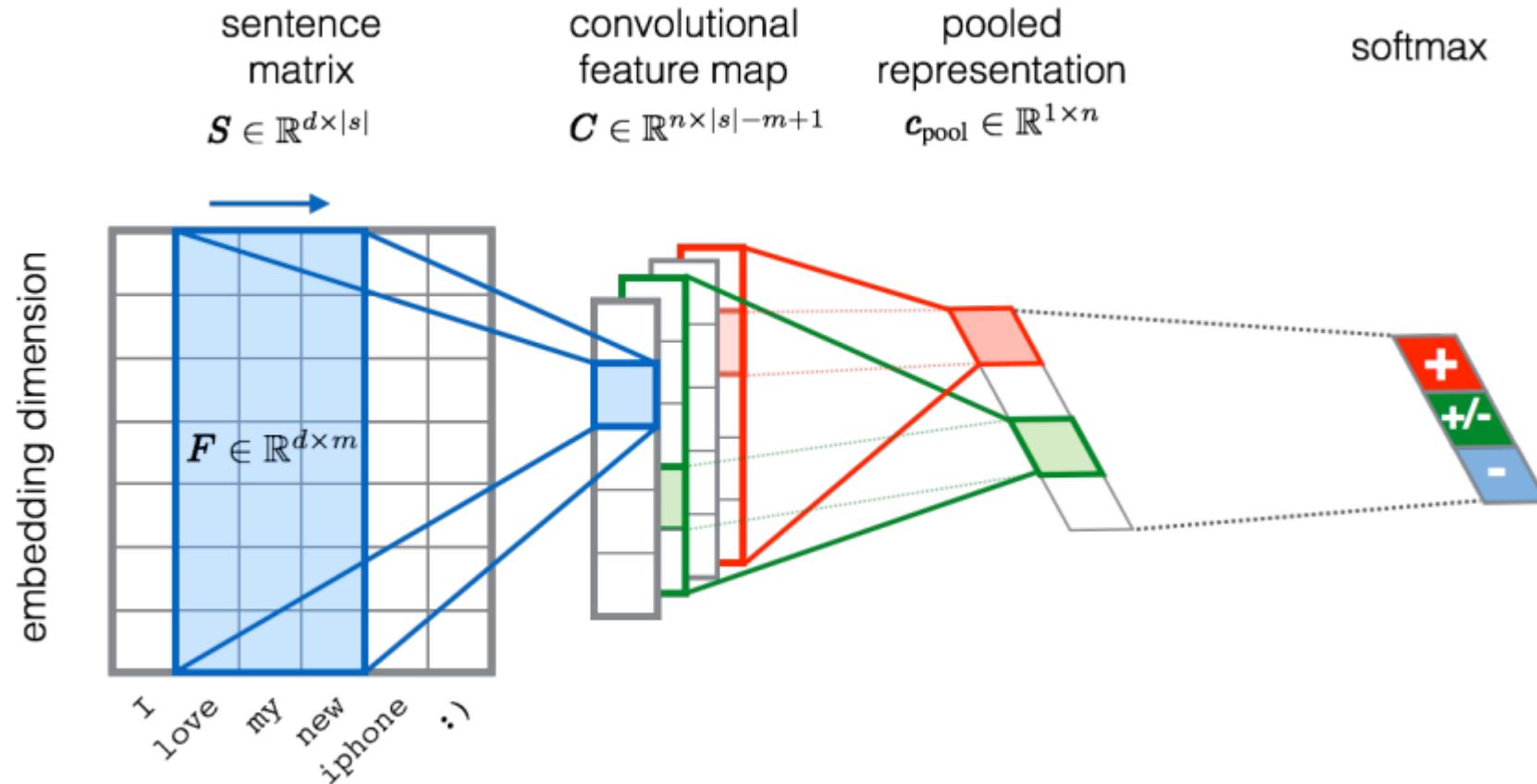
# Padding



# Padding



# Application in Text Classification



Severyn, Aliaksei, and Alessandro Moschitti. "Twitter sentiment analysis with deep convolutional neural networks." *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2015.

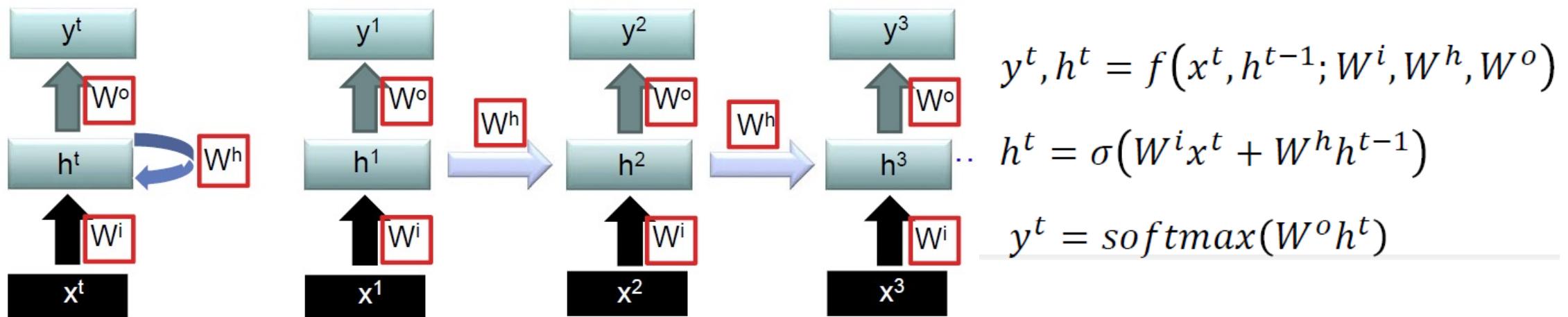
# Recurrent Neural Network

# Network Architecture

- Assumptions of FNN
  - Fixed input vector size
  - Independence
- Drawbacks of FNN
  - Cannot work with variable input sizes
  - Cannot handle temporal dependencies
  - Insensitive to the order of input
- Applications of Sequential Data
  - Text, Speech, Video, Genomes, Handwriting, Time series

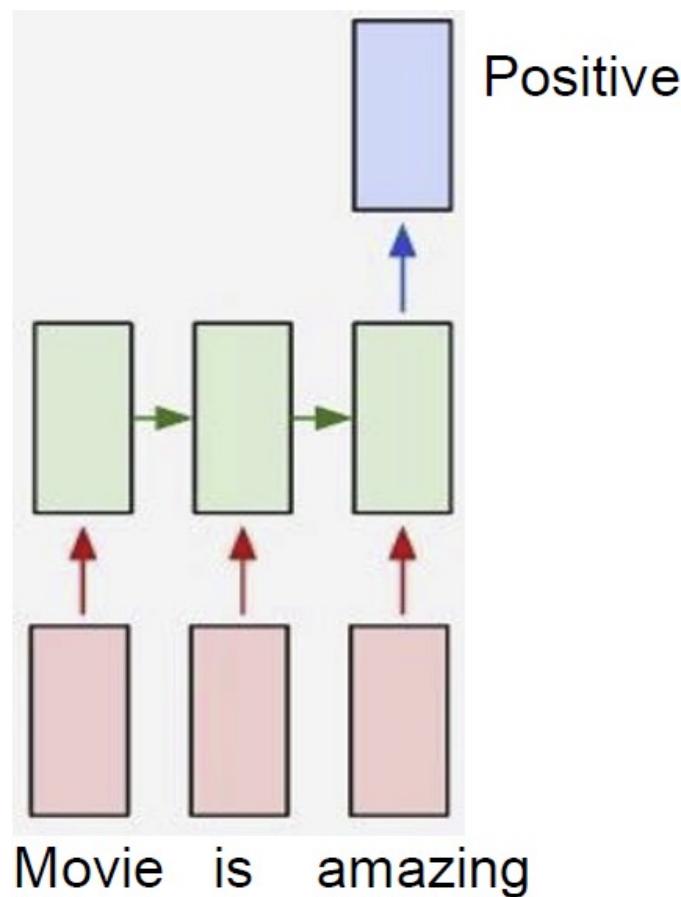
# Recurrent Neural Networks (RNN)

- We can process a sequence of vectors  $x$  by applying a recurrence formula at every time step.
- The same function and the same set of parameters are used at every time step.

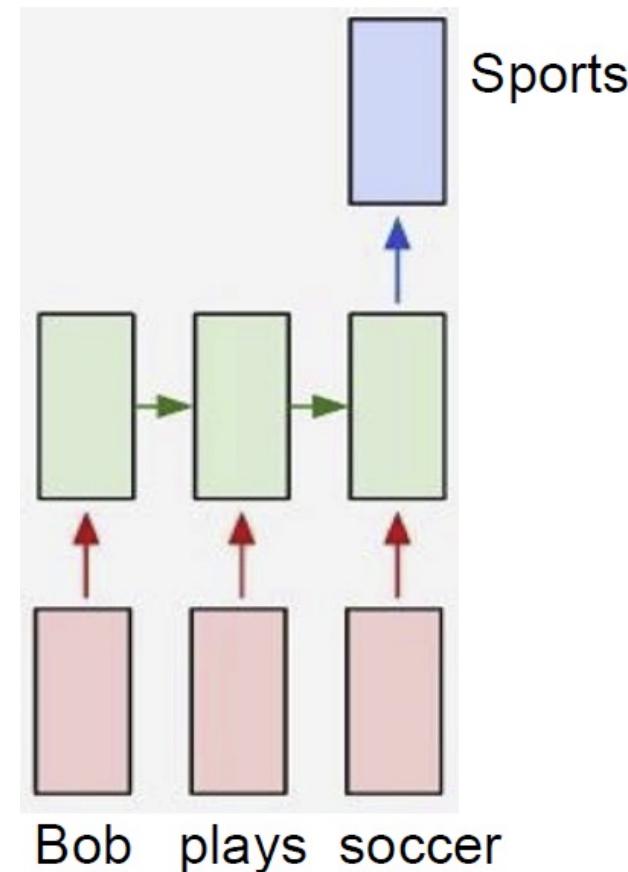


# RNN Types: Many to One

- Sentiment Classification

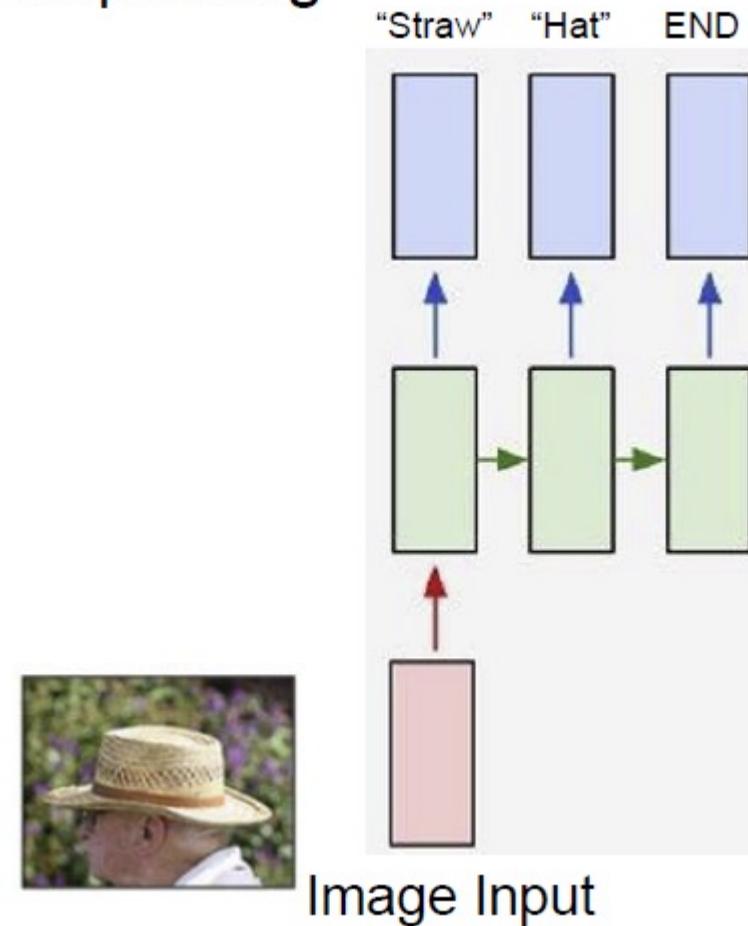


- Text Categorization



# RNN Types: One to Many

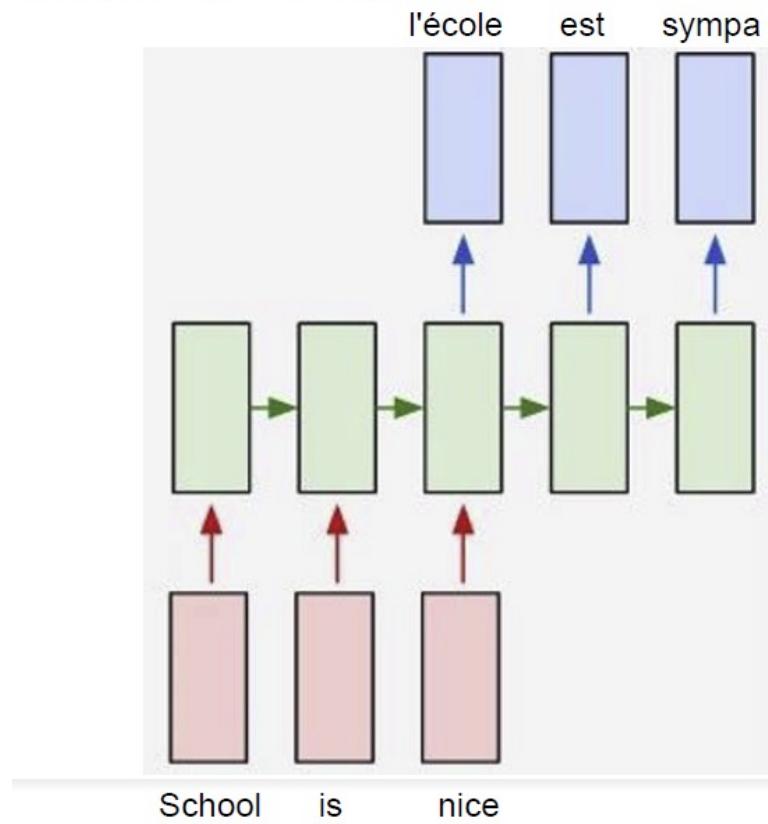
## Image Captioning



# RNN Types: Many to Many

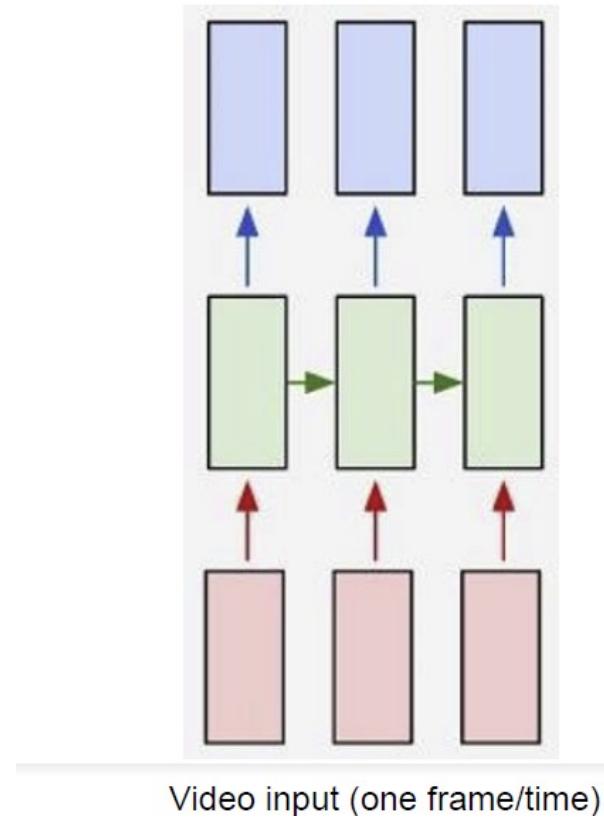
Machine Translation

(English to French)



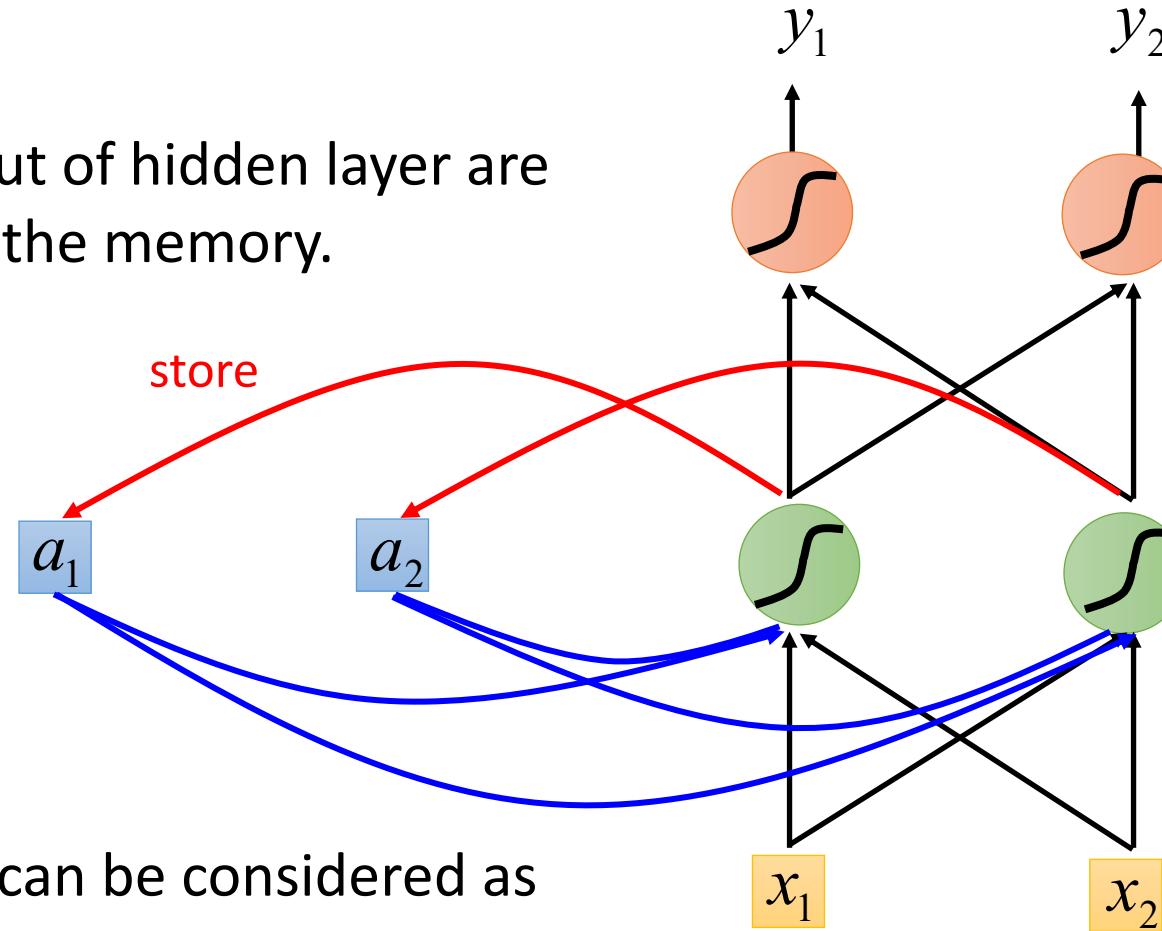
Video Classification

Frame-level classification



# Recurrent Neural Network (RNN)

The output of hidden layer are stored in the memory.

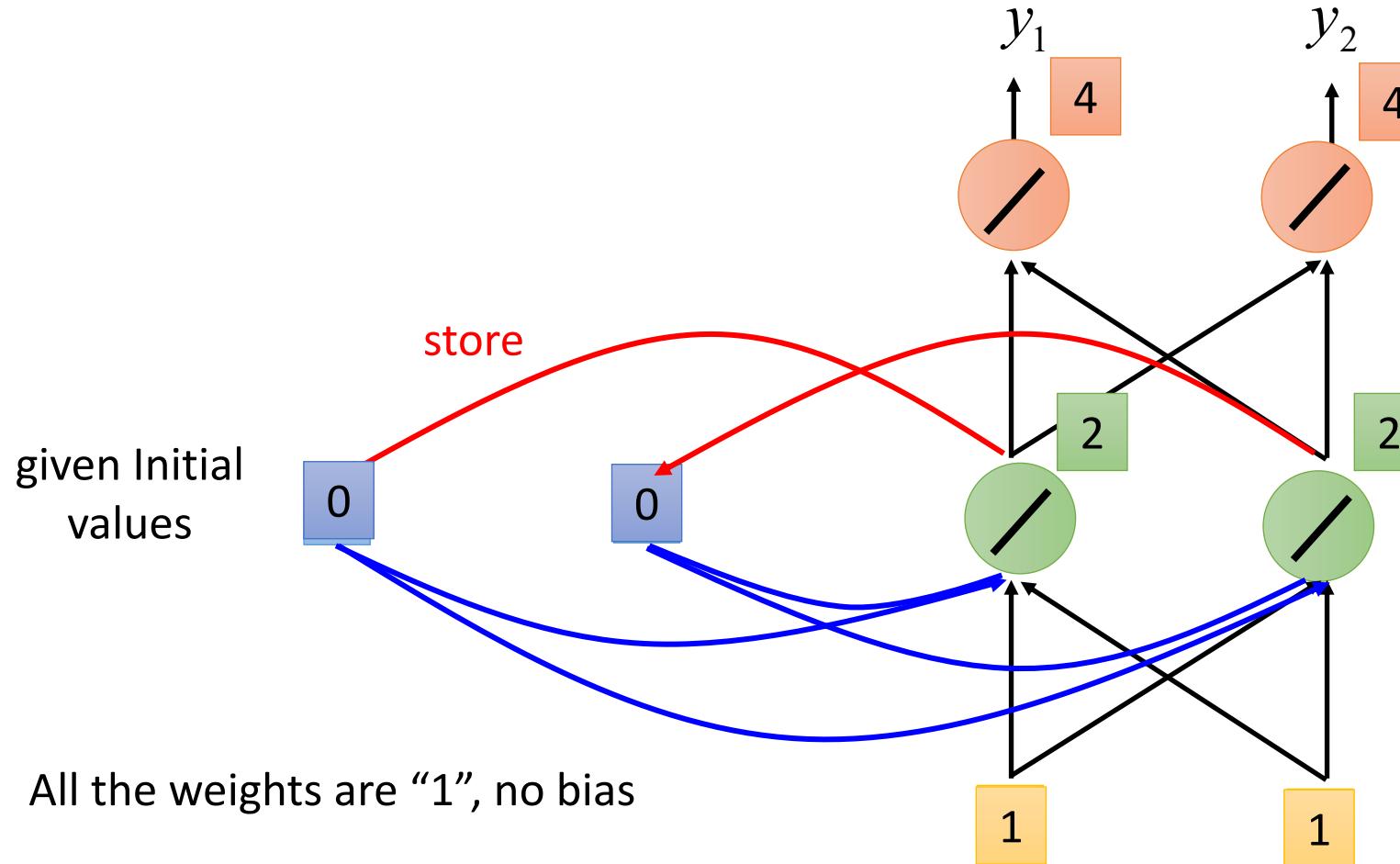


Memory can be considered as another input.

# Example

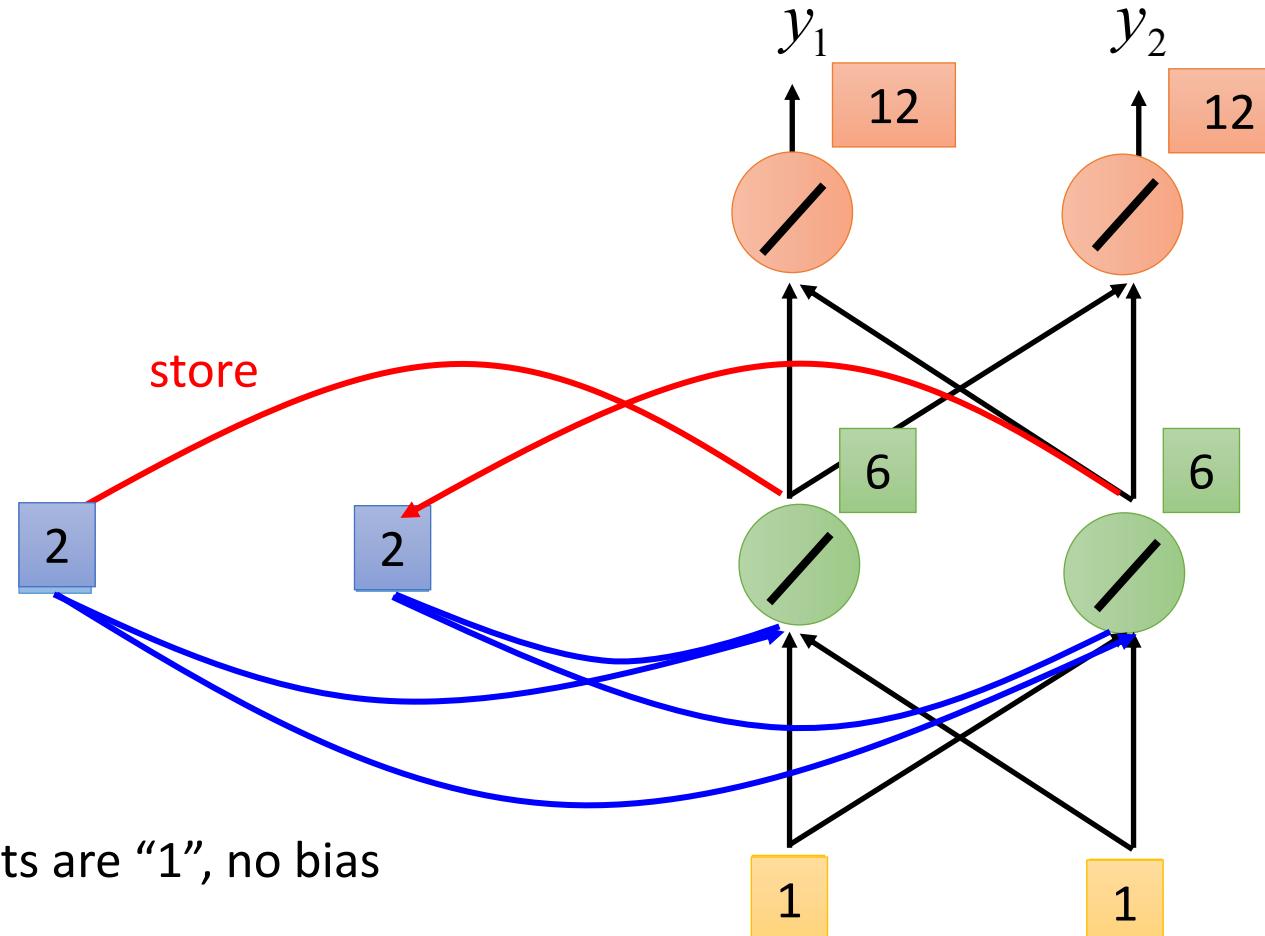
Input sequence:  $\begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 2 \\ 2 \end{bmatrix} \quad \dots \dots$

output sequence:  $\begin{bmatrix} 4 \\ 4 \end{bmatrix}$



# Example

Input sequence:  $\begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \end{bmatrix} \dots \dots$   
output sequence:  $\begin{bmatrix} 4 \\ 4 \end{bmatrix} \begin{bmatrix} 12 \\ 12 \end{bmatrix}$



All the weights are “1”, no bias

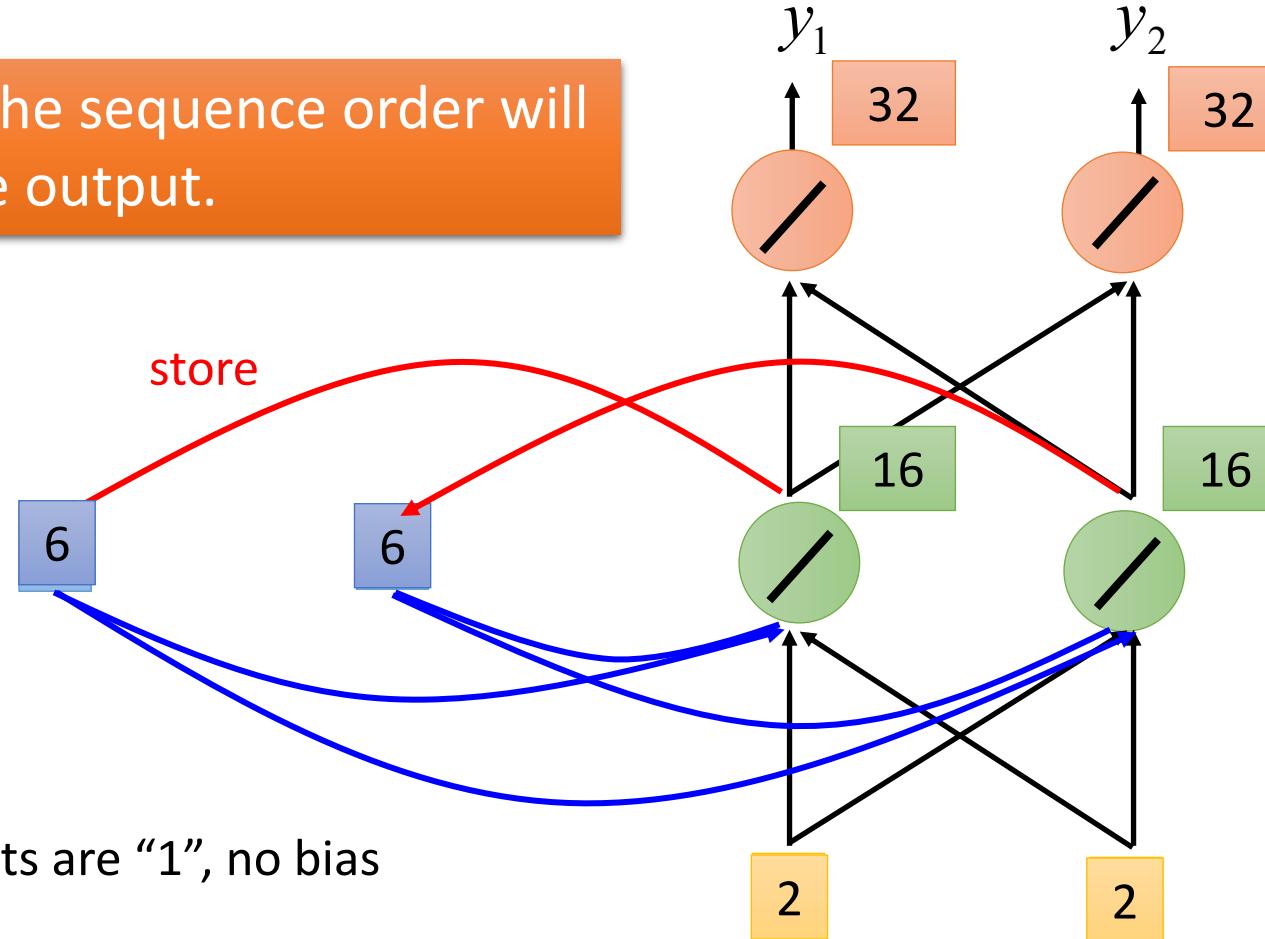
All activation functions are linear

# Example

Input sequence:  
 $\begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 2 \\ 2 \end{bmatrix} \quad \dots \dots$

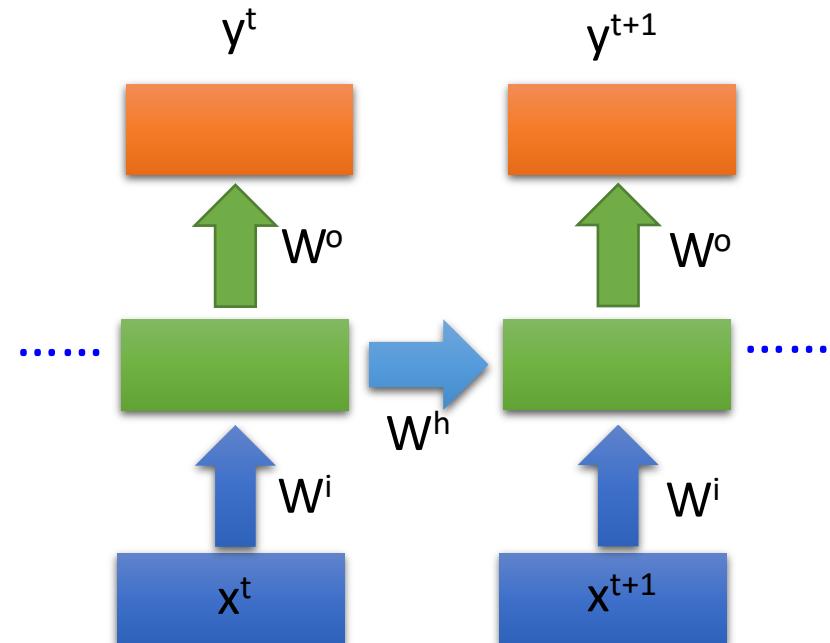
output sequence:  
 $\begin{bmatrix} 4 \\ 4 \end{bmatrix} \quad \begin{bmatrix} 12 \\ 12 \end{bmatrix} \quad \begin{bmatrix} 32 \\ 32 \end{bmatrix}$

Changing the sequence order will change the output.

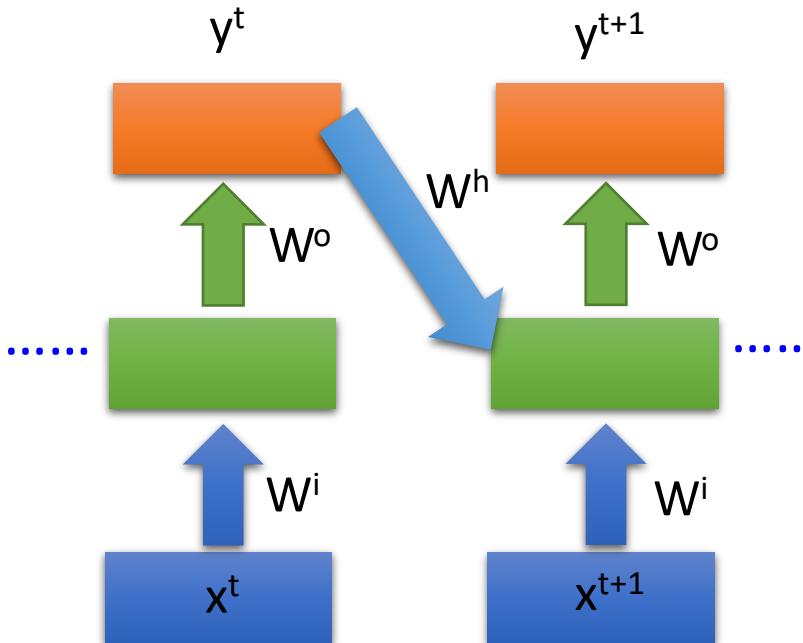


# Elman Network & Jordan Network

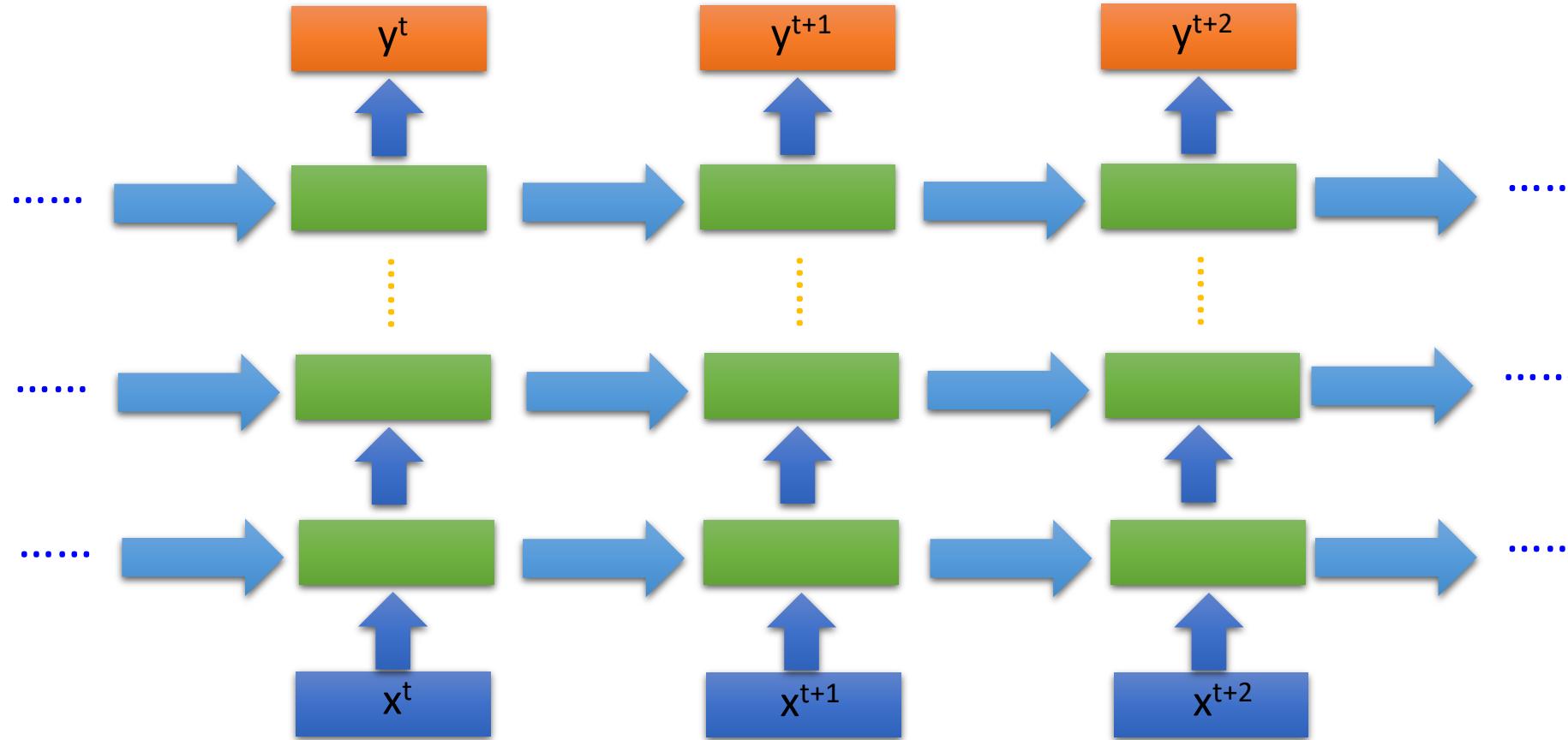
***Elman Network***



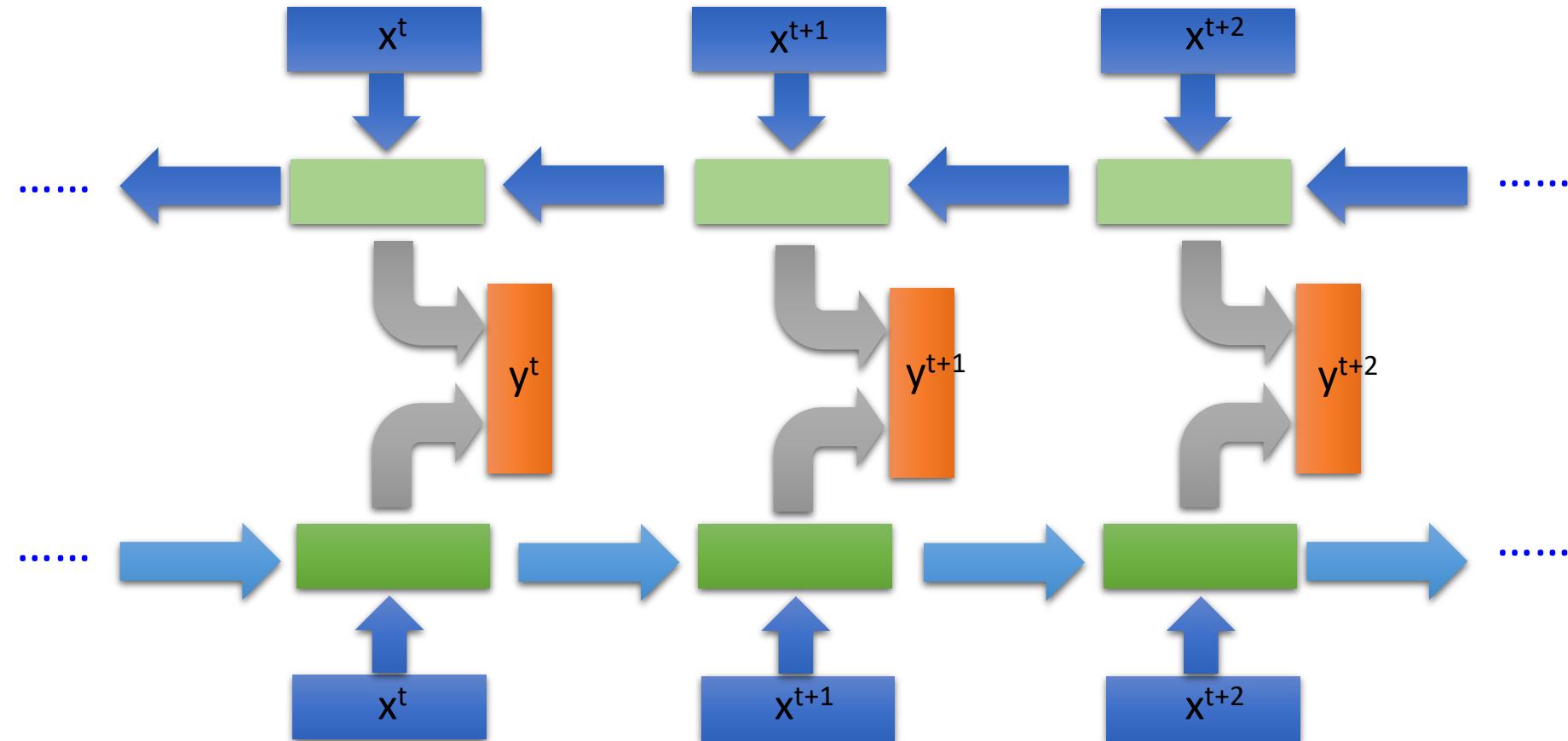
***Jordan Network***



# Of Course It Can Be Deep ...



# Bidirectional RNN



# RNN

- Training of RNN:
  - Backpropagation through time
  - Suffer from gradient vanishing/exploding
- Commonly used RNN architectures:
  - Long Short-term Memory (LSTM)
  - Gated Recurrent Units (GRU)

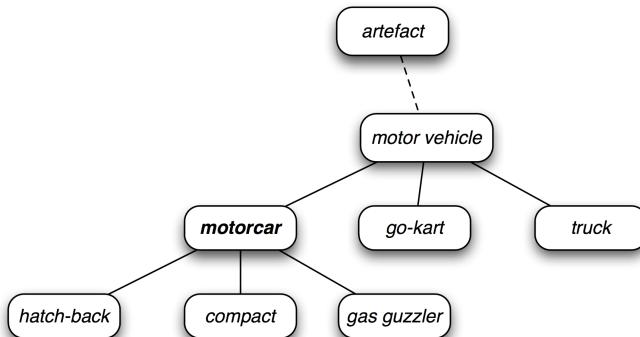
# Summary of RNN

- RNNs allow a lot of flexibility in architecture design and are learned using Backpropagation through time algorithm.
- Vanilla RNNs are simple but don't work very well.
- Common to use LSTM or GRU: their additive interactions improve gradient flow.
- Better/simpler architectures are a hot research topic.

# Word Embeddings

# Representation for Text

## Knowledge-based representation



## One-hot representation

Issues: difficult to compute the similarity  
(i.e. comparing "car" and "motorcycle")

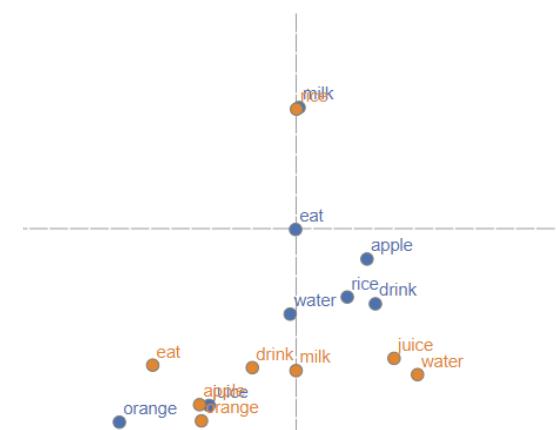
$$[0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \dots \ 0] \text{ AND } [0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \dots \ 0] = 0$$

car                                      motorcycle

## Co-occurrence Matrix

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

**Low-dimensional dense word vector**  
(by dimension reduction on the co-occurrence matrix)



# Word Embeddings

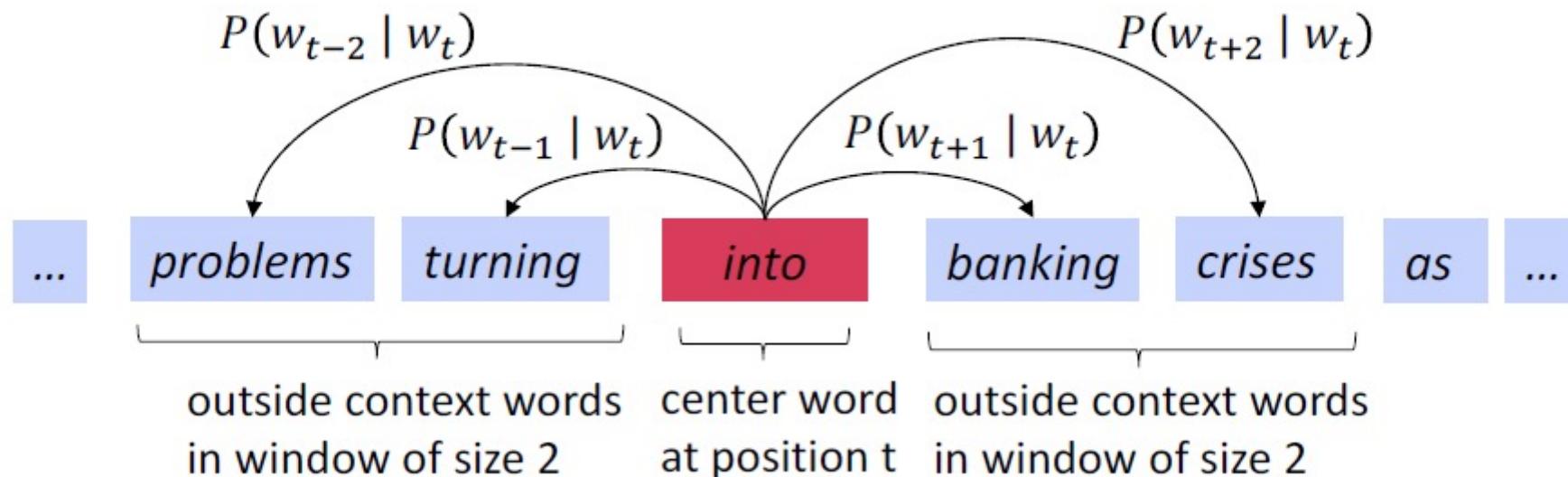
- Given an unlabeled training corpus, produce a vector for each word that encodes its semantic information. These vectors are useful because:
  - Semantic similarity between two words can be calculated as the **cosine similarity** between their corresponding word vectors.
  - **Propagate** any information into them via neural networks and **update during training**.
  - Word vectors as powerful features for various downstream **NLP tasks** since the vectors contain semantic information. Such as word classification, sentiment analysis.

# Word2Vec

- We have a large corpus of text.
- Every word in a fixed vocabulary is represented by a vector.
- Go through each position  $t$  in the text, which has a center word  $c$  and context (“outside”) words  $o$
- **Basic Idea:** predict surrounding words within a window of each word
  - Use the similarity of the word vectors for  $c$  and  $o$  to calculate the probability of  $o$  given  $c$  (or vice versa)
  - Keep adjusting the word vectors to maximize this probability
- **Advantage:** fast, easy to incorporate a new sentence/document or add a word to vocab

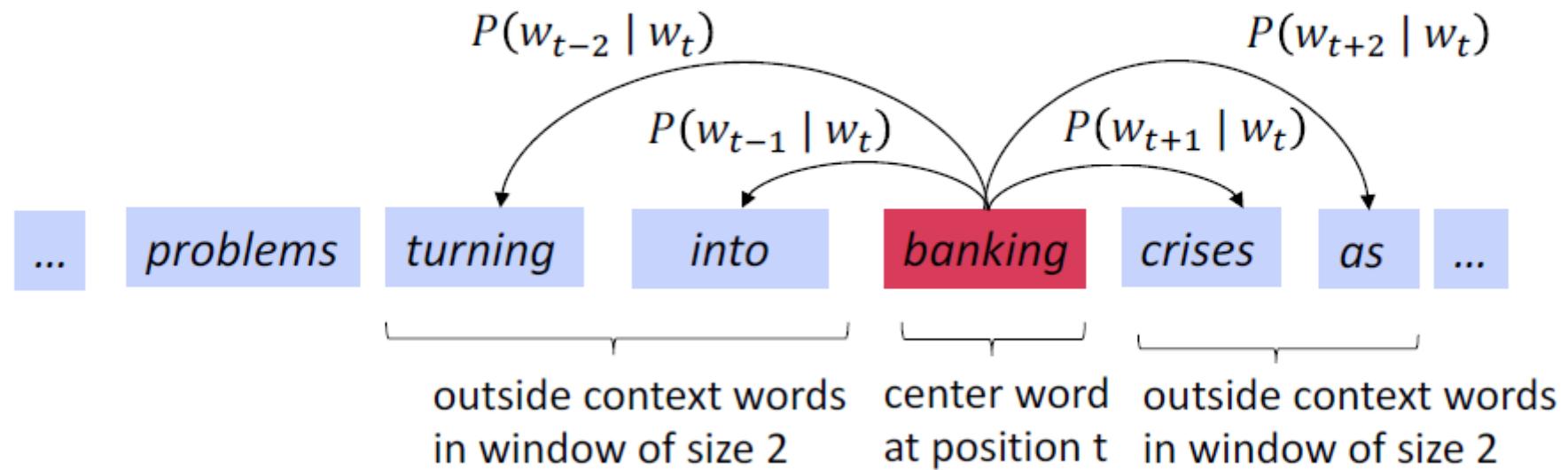
# Word2Vec: Skip-Gram Model

- Example windows and process for computing  $P(w_{t+j} | w_t)$



# Word2Vec: Skip-Gram Model

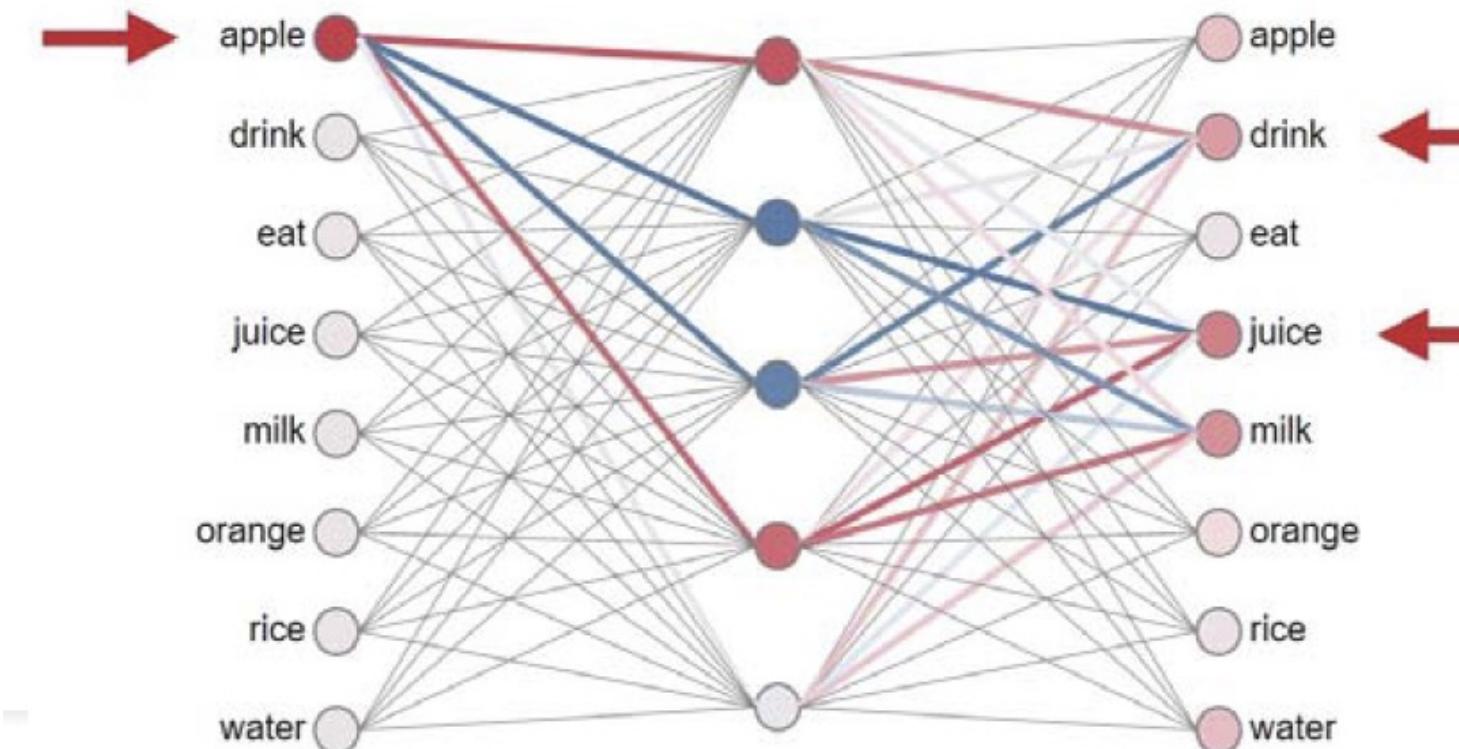
- Example windows and process for computing  $P(w_{t+j} | w_t)$



# Word2Vec Skip-Gram Visualization

## Training data:

apple|drink, juice, orange|eat, apple, rice|drink, juice, juice|drink, milk, milk|drink, rice, water|drink,  
milk, juice|orange, apple, juice|apple, drink, milk|rice, drink, drink|milk, water, drink|water, juice, drink|juice,  
water



# Word2Vec: Objective Function

For each position  $t = 1, \dots, T$ , predict context words within a window of fixed size  $m$ , given center word  $w_t$

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^T \prod_{(-m \leq j \leq m, j \neq 0)} P(w_{t+j} | w_t; \theta)$$

The **objective function**  $J(\theta)$  is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{(-m \leq j \leq m, j \neq 0)} \log P(w_{t+j} | w_t; \theta)$$

Minimizing objective function  $\Leftrightarrow$  Maximizing predictive accuracy

# Word2Vec: Objective Function

We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{(-m \leq j \leq m, j \neq 0)} \log P(w_{t+j}|w_t; \theta)$$

Question: how to calculate  $P(w_{t+j}|w_t; \theta)$ ?

Answer: we will use two vectors per word  $w$ :

- $v_w$  when  $w$  is a center word
- $u_w$  when  $w$  is a context word

Then for a center word  $c$  and a context word  $o$ :

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

# Word2Vec: Prediction Function

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Top: dot product compares similarity of  $o$  and  $c$ . Larger dot product: larger probability

Bottom: After taking exponent, normalize over entire vocabulary.

This is an example of the **softmax function**  $\mathbb{R}^n \rightarrow \mathbb{R}^n$

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)} = p_i$$

The softmax function maps arbitrary values  $x_i$  to a probability distribution  $p_i$

# Compute All Vector Gradients

Recall:  $\theta$  represents all model parameters, in one long vector

In our case with  $d$ -dimensional vectors and  $V$ -many words:

$$\theta = \begin{pmatrix} v_{\text{aardvark}} \\ \dots \\ v_{\text{zebra}} \\ u_{\text{aardvark}} \\ \dots \\ u_{\text{zebra}} \end{pmatrix} \in \mathbb{R}^{2dV}$$

Use the vectors for center words as word embeddings.

Remember: every word has two vectors

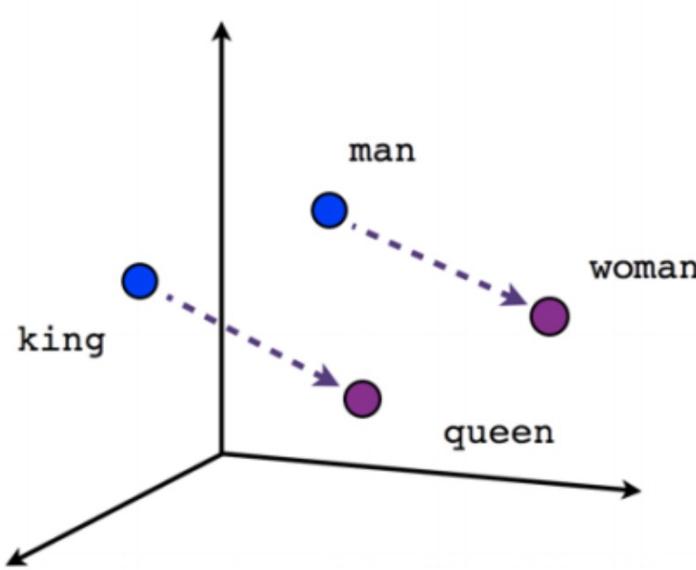
We then optimize these parameters

# Skip-gram With Negative Sampling

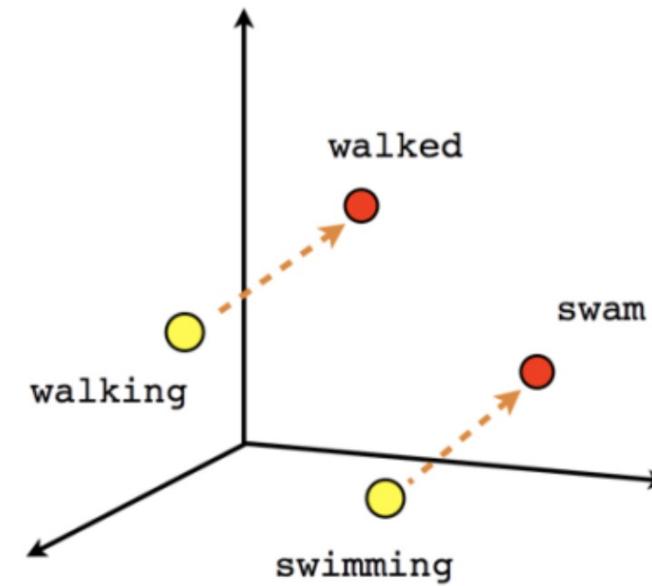
- Problem: the gradient of  $J(\theta)$  is expensive to compute. Why?
- Solution: **Contrastive learning by negative sampling**
  - Create **positive pairs** (center word and word in its context window) versus **negative pairs** (the center word paired with a random words)
  - Maximize probability that positive pair appears, minimize probability that negative pairs appear.

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J_t(\theta)$$
$$J_t(\theta) = \log \sigma(u_o^T v_c) + \sum_{i=1}^k \mathbb{E}_{i \sim P(w)} (\log \sigma(-u_i^T v_c))$$

# Word2Vec Examples



Male-Female



Verb tense

[https://radimrehurek.com/gensim/auto\\_examples/tutorials/run\\_word2vec.html](https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html)

<https://towardsdatascience.com/word-embedding-with-word2vec-and-fasttext-a209c1d3e12c>

# Implementation

- [Github repository](#) for Natural language processing (NLP) using Pytorch
  - [CNN for binary sentiment classification on text](#)
  - [RNN for predicting next word](#)
  - [Word2Vec](#)

# Summary of Today's Lecture

- Convolutional Neural Networks (CNN)
- Recurrent Neural Network (RNN)
- Word2Vec