CS 559 Machine Learning

Lecture 14: 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

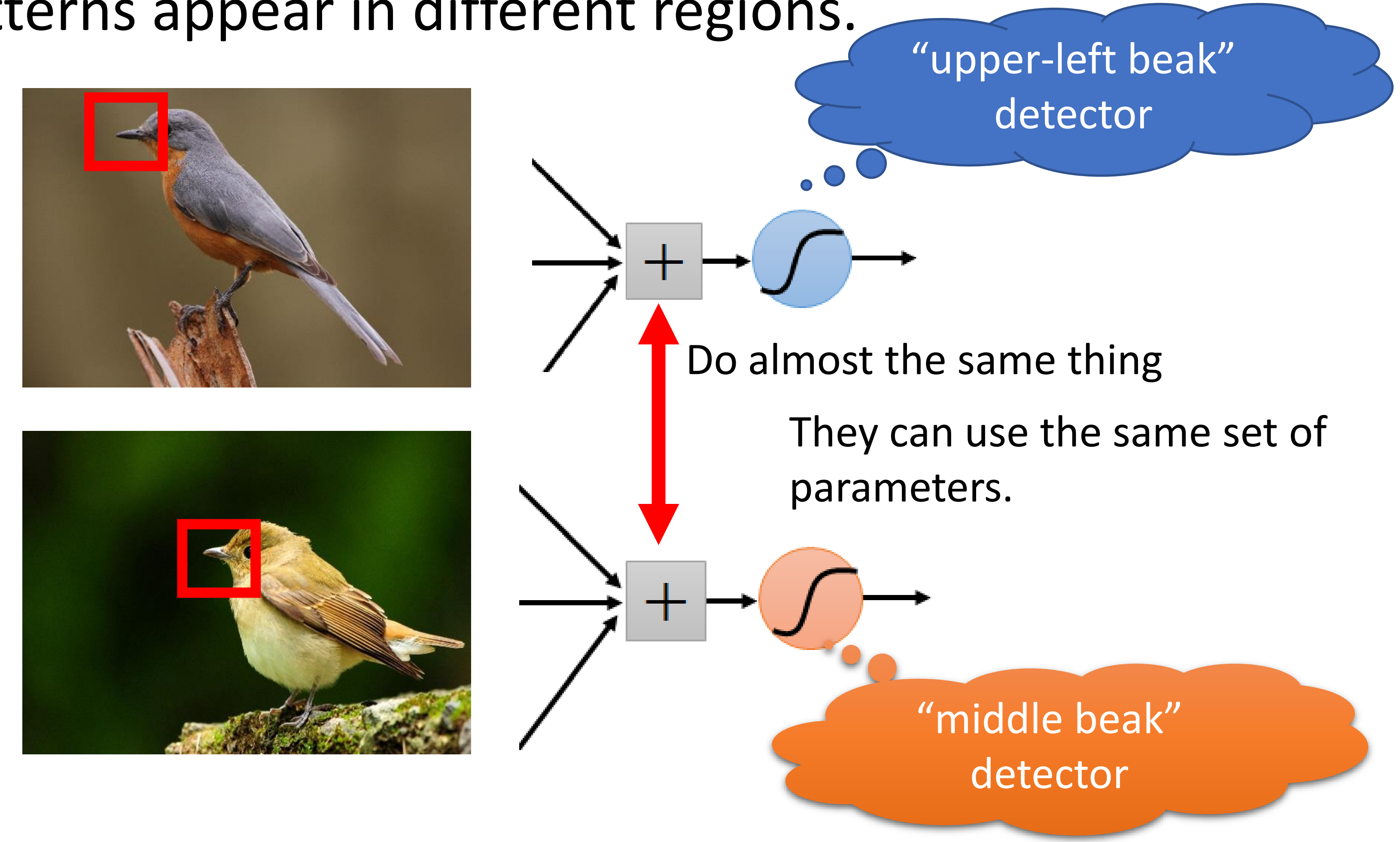
- Some patterns are much smaller than the whole image.

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

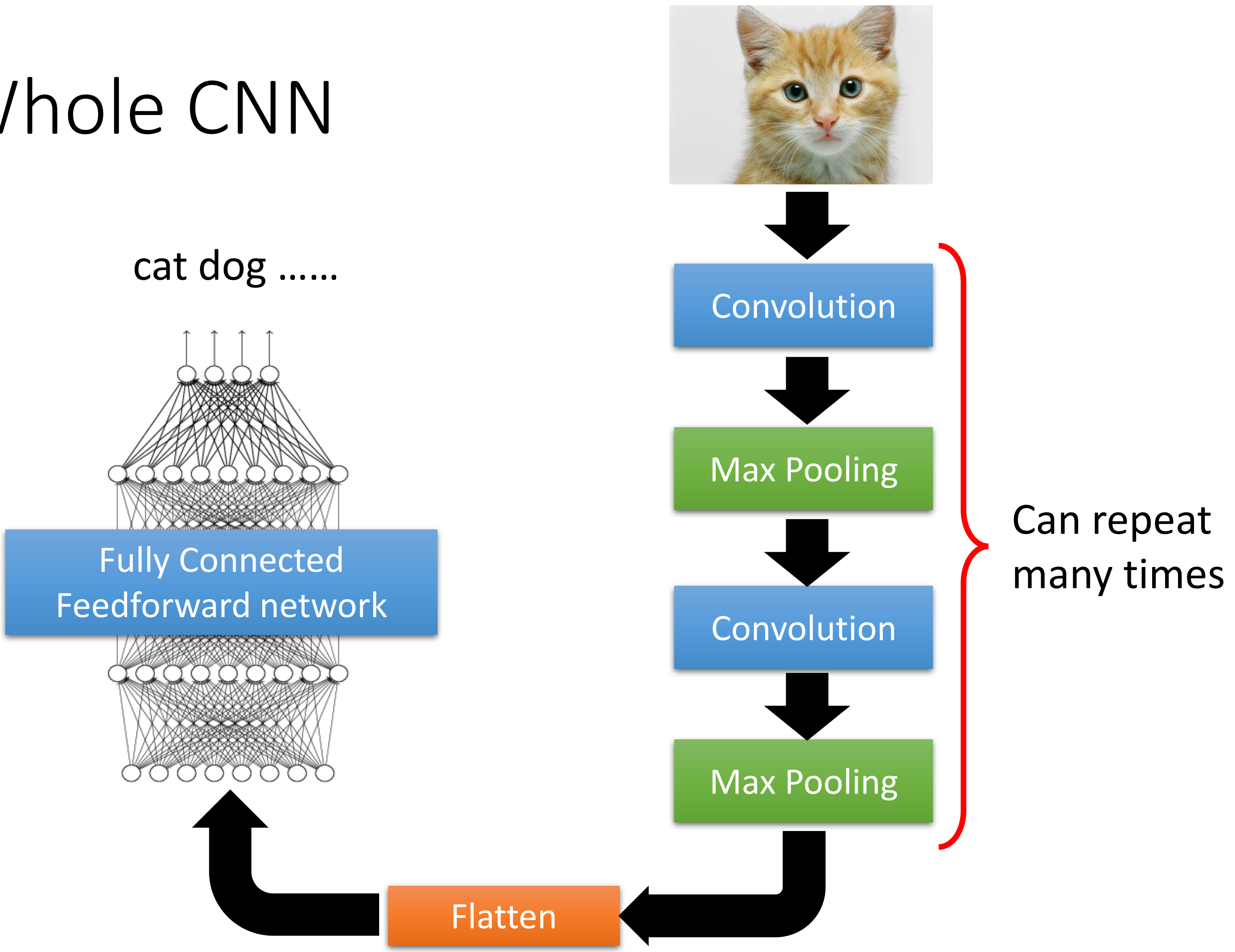
subsampling

bird

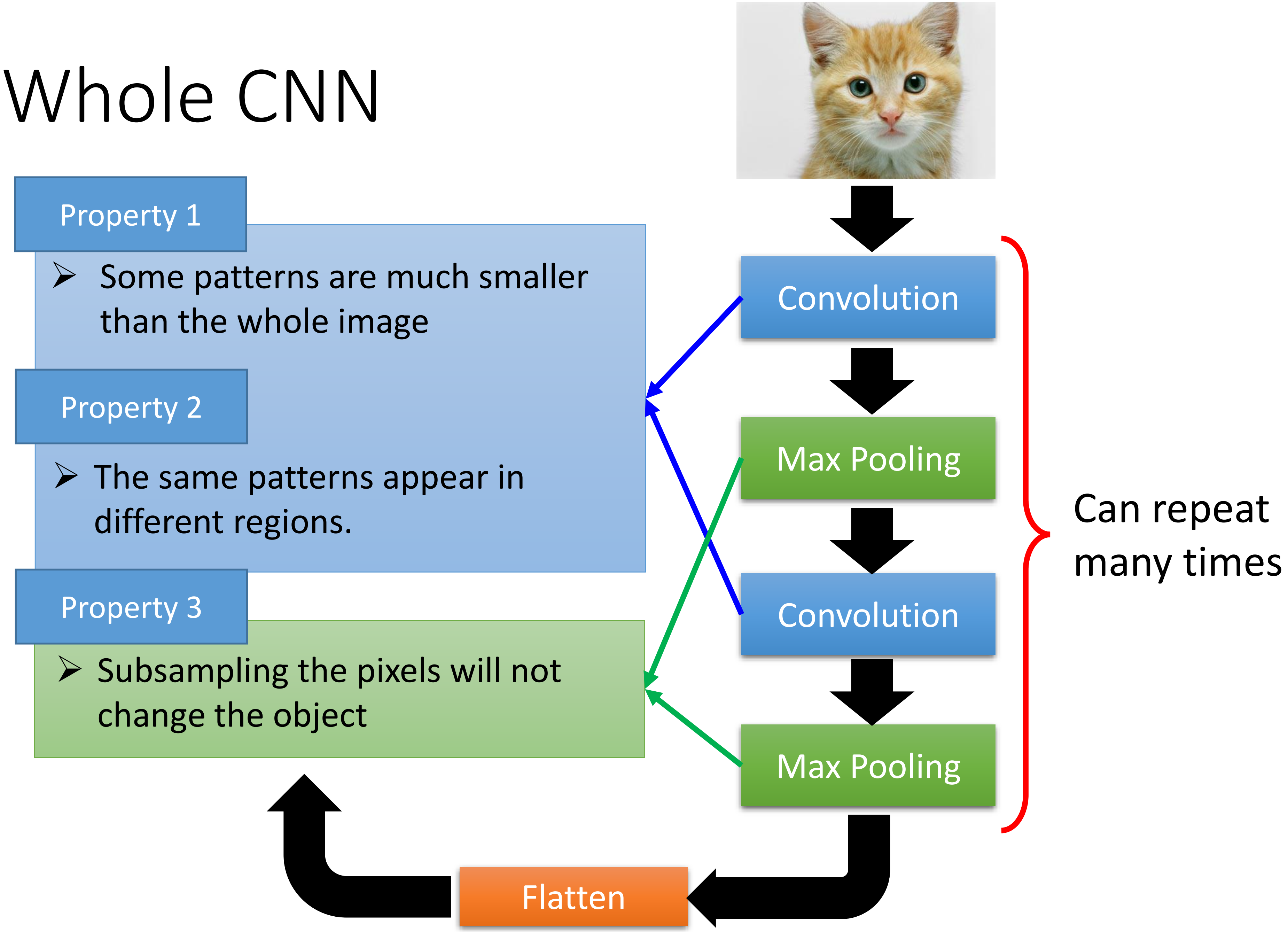
We can subsample the pixels to make image smaller

→ Less parameters for the network to process the image

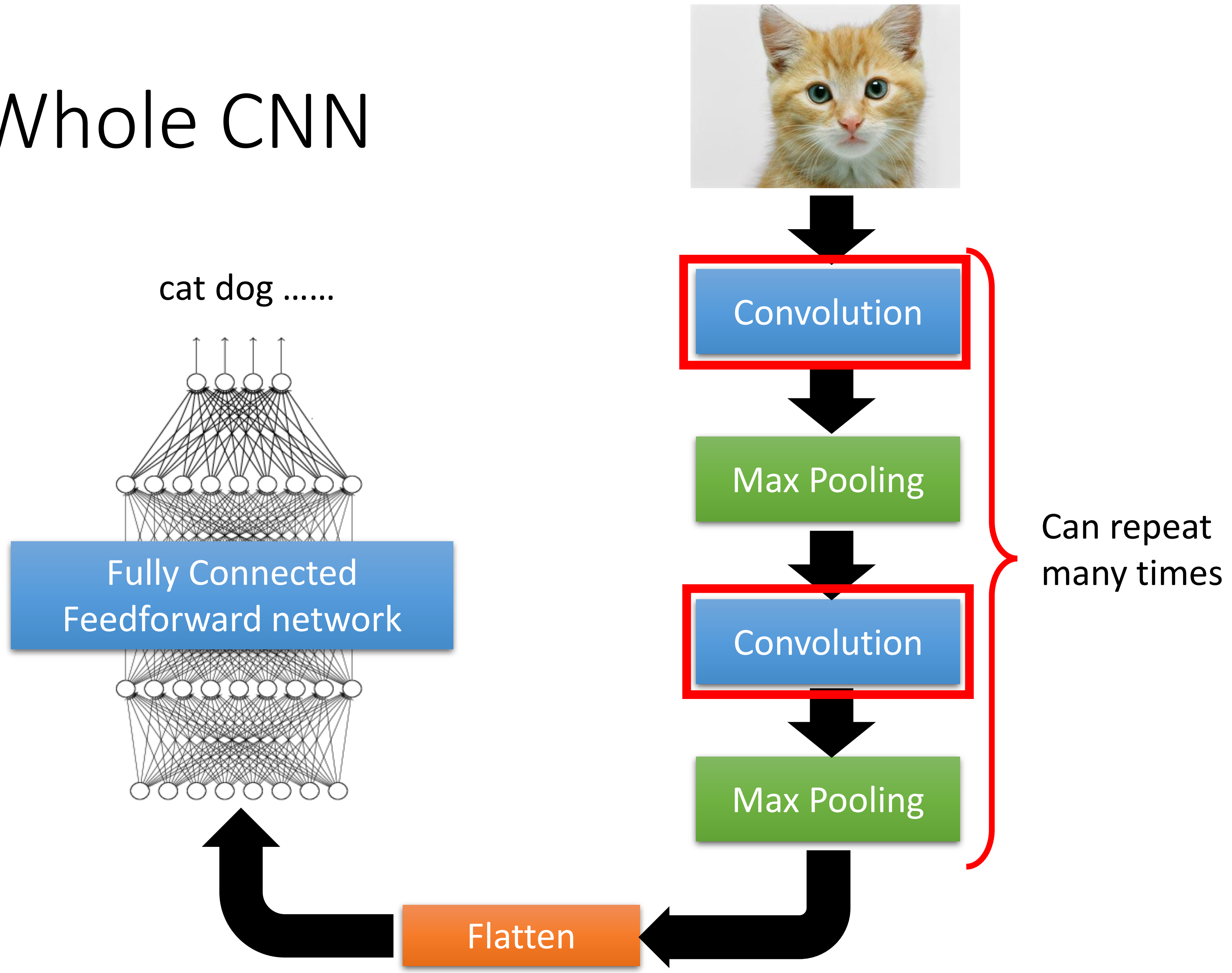
The Whole CNN



The Whole CNN



The Whole CNN



CNN – Convolution

Those are the network parameters to be learned.

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

Matrix

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

Filter 2

Matrix

⋮

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

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

CNN – Convolution

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

Filter 1

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

6 x 6 image

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

CNN – Convolution

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

Filter 1

stride=2

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

6 x 6 image

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

We set stride=1 below

CNN – Convolution

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

Filter 1

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

3 -1

CNN – Convolution

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

Filter 1

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

3 -1 -3

CNN – Convolution

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

Filter 1

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

3 -1 -3 -1

CNN – Convolution

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

Filter 1

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

3 -1 -3 -1

-3

CNN – Convolution

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

Filter 1

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

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	

CNN – Convolution

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

Filter 1

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

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

CNN – Convolution

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

Filter 1

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

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

Property 2: The same patterns appear in different regions.

CNN – Convolution

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

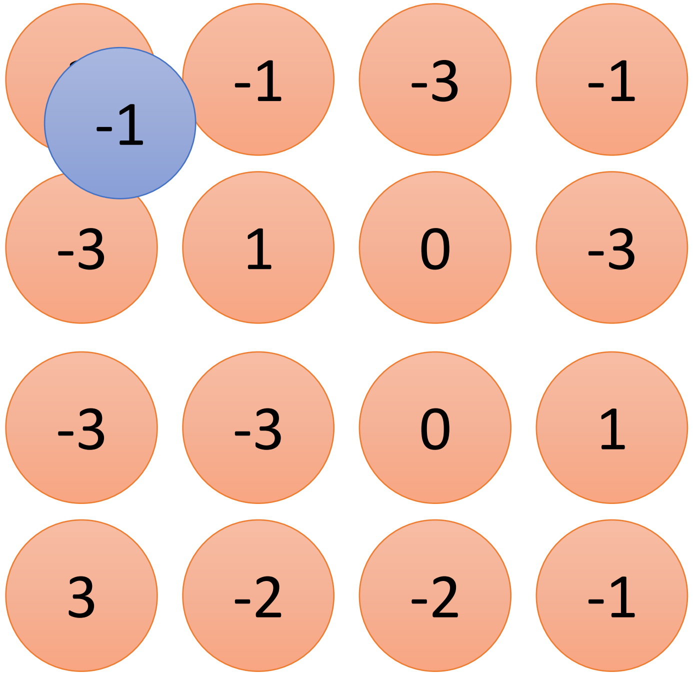
Filter 2

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

Do the same process for every filter



4 x 4 image

CNN – Convolution

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

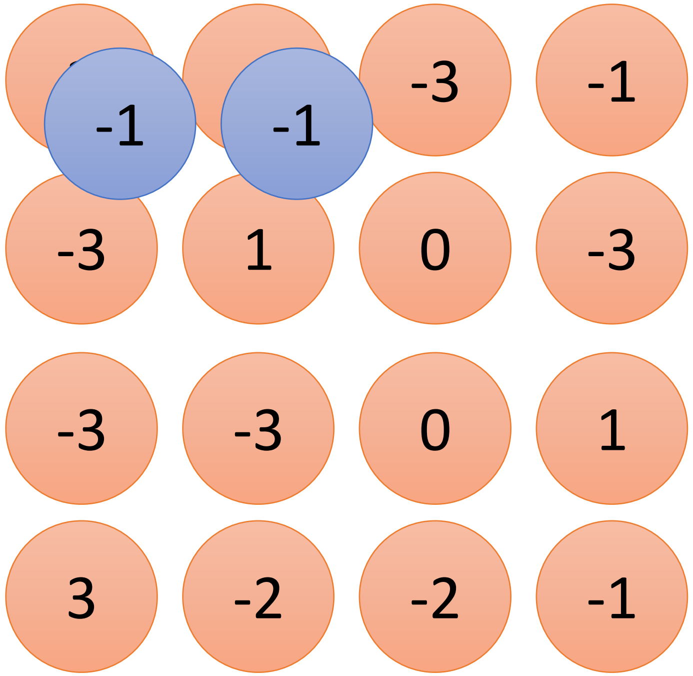
Filter 2

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

Do the same process for every filter



4 x 4 image

CNN – Convolution

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

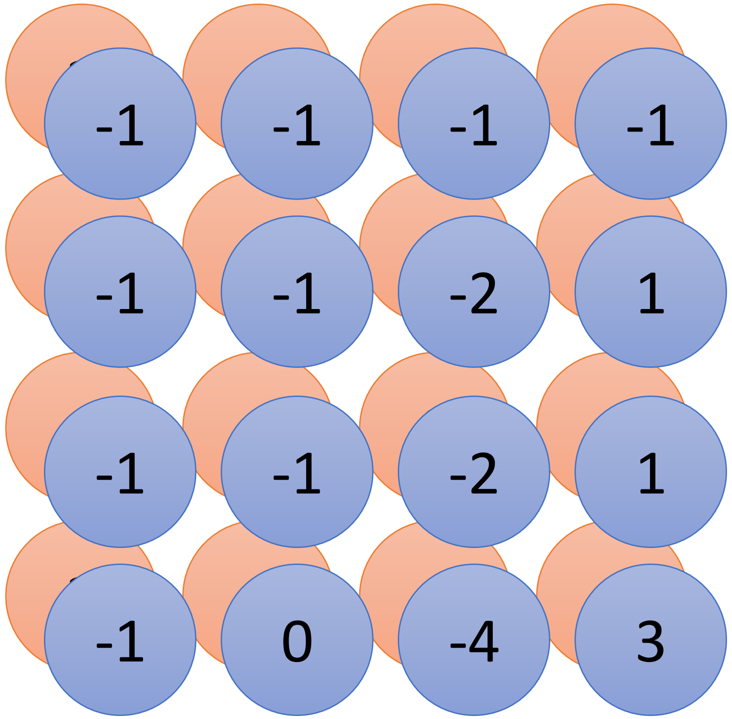
Filter 2

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

Do the same process for every filter



4 x 4 image

CNN – Convolution

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

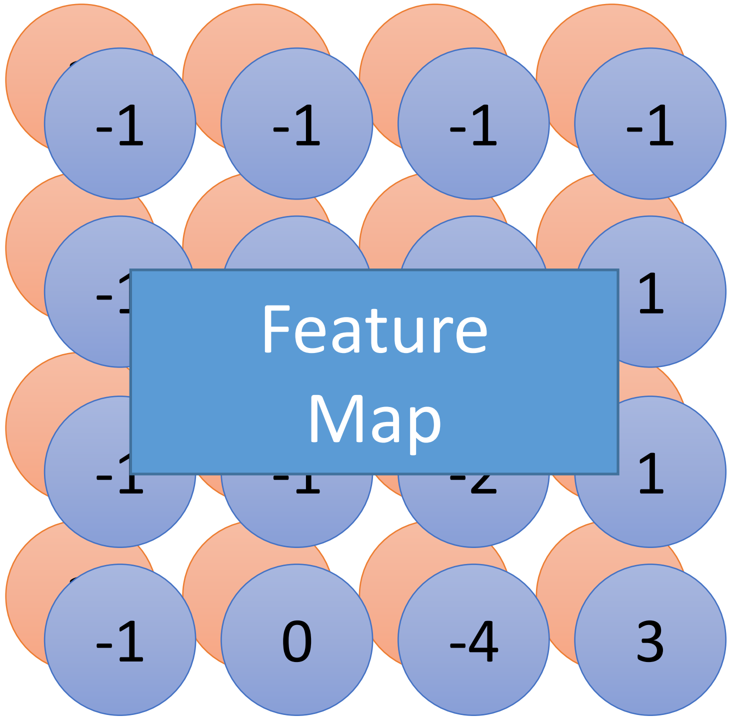
Filter 2

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

Do the same process for every filter



4 x 4 image

CNN – Colorful Image

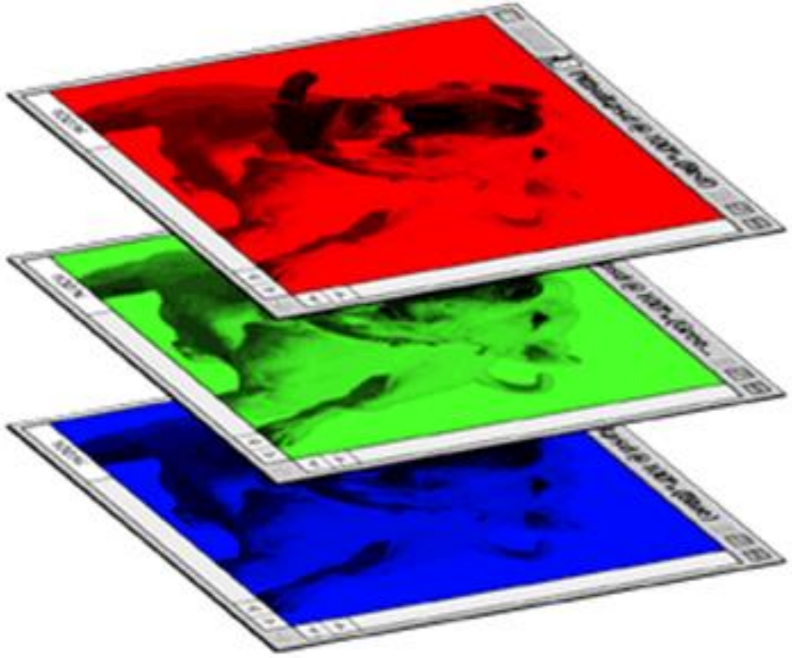
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

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

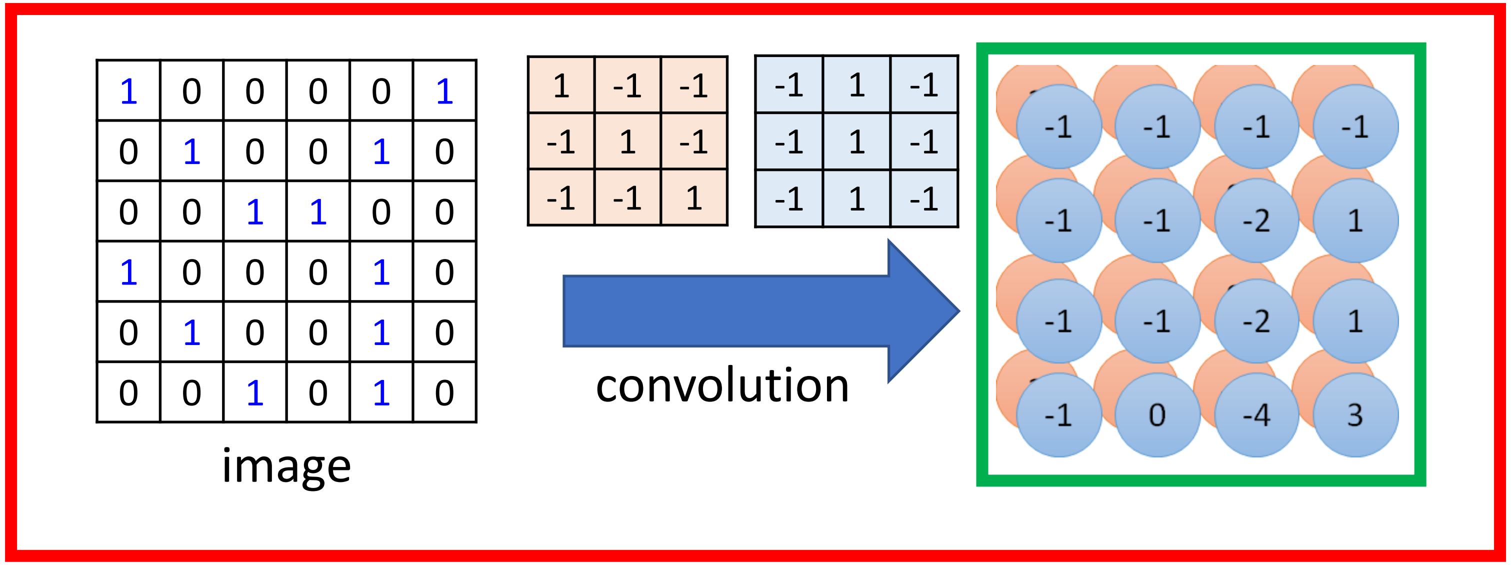
Filter 2

Colorful image

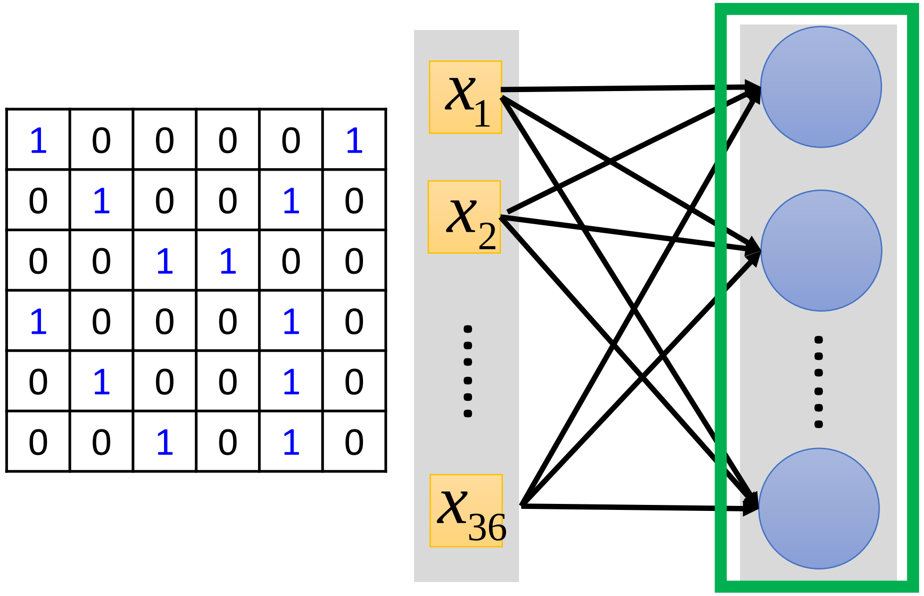


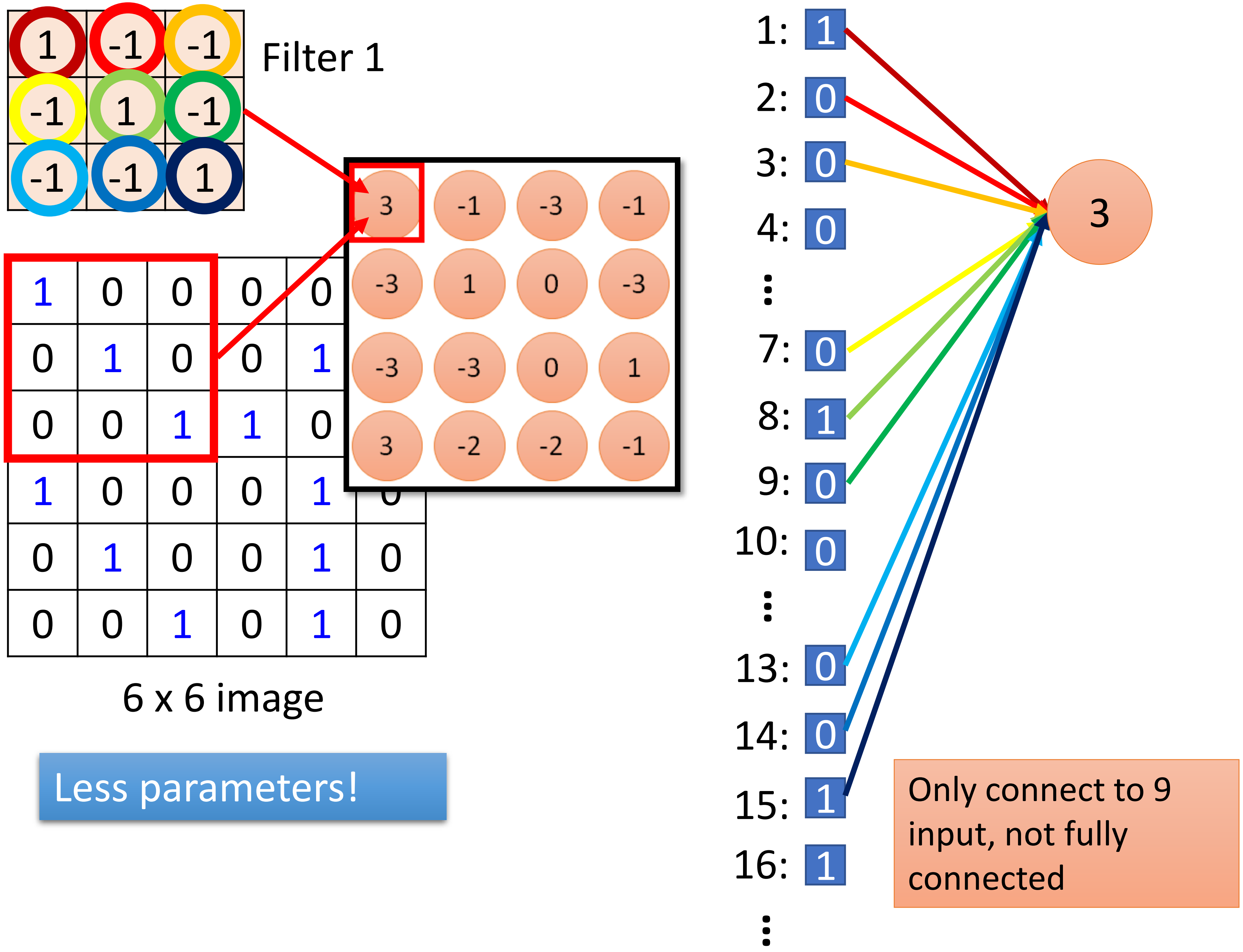
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

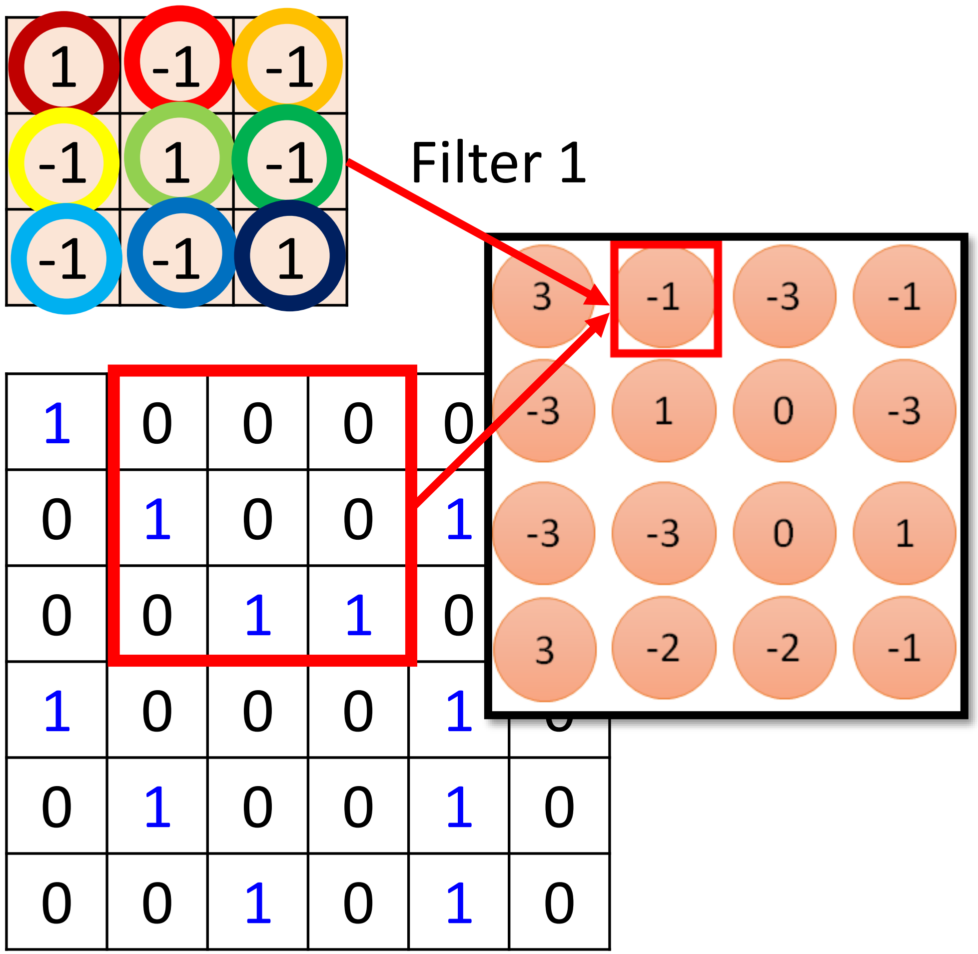
Convolution v.s. Fully Connected



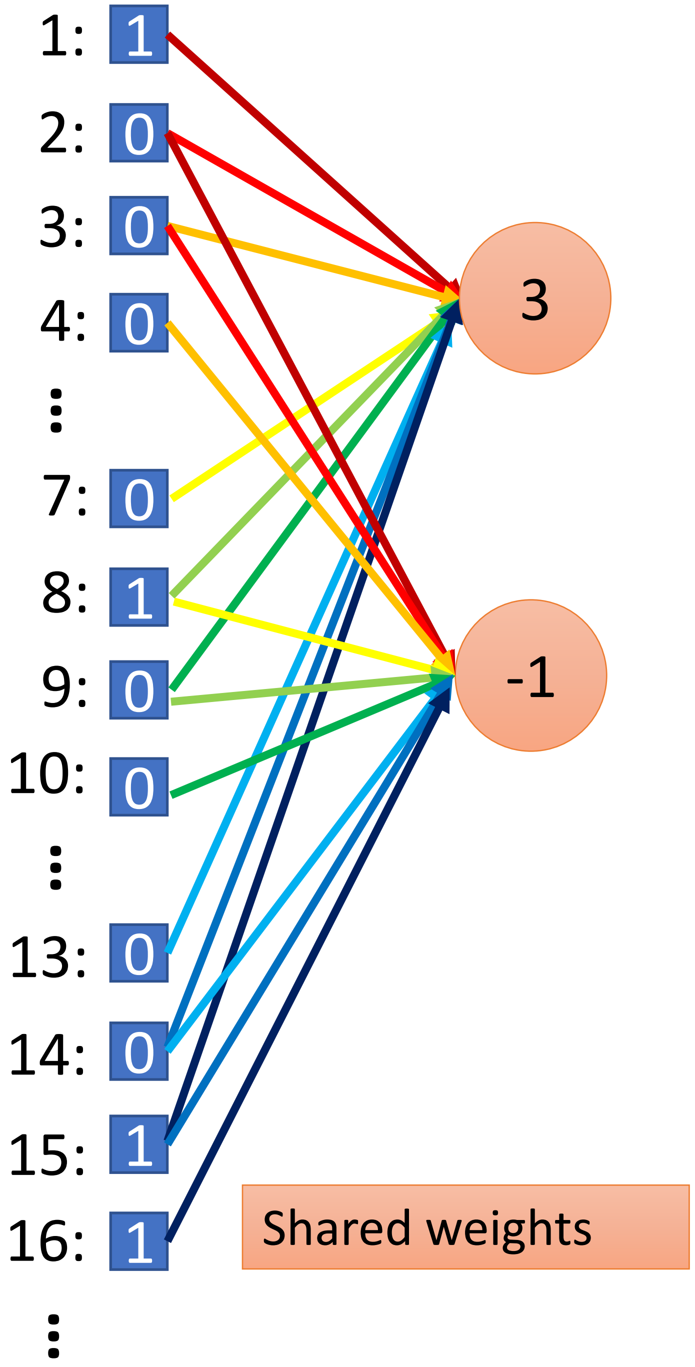
Fully-connected




Filter 1
1 -1 -1
-1 1 -1
-1 -1 1
6 x 6 image
1 0 0 0 0
0 1 0 0 1
0 0 1 1 0
1 0 0 0 1 0
0 1 0 0 1 0
0 0 1 0 1 0
3 -1 -3 -1
-3 1 0 -3
-3 -3 0 1
3 -2 -2 -1
1: 1
2: 0
3: 0
4: 0
7: 0
8: 1
9: 0
10: 0
13: 0
14: 0
15: 1
16: 1
3
Less parameters!
Only connect to 9 input, not fully connected



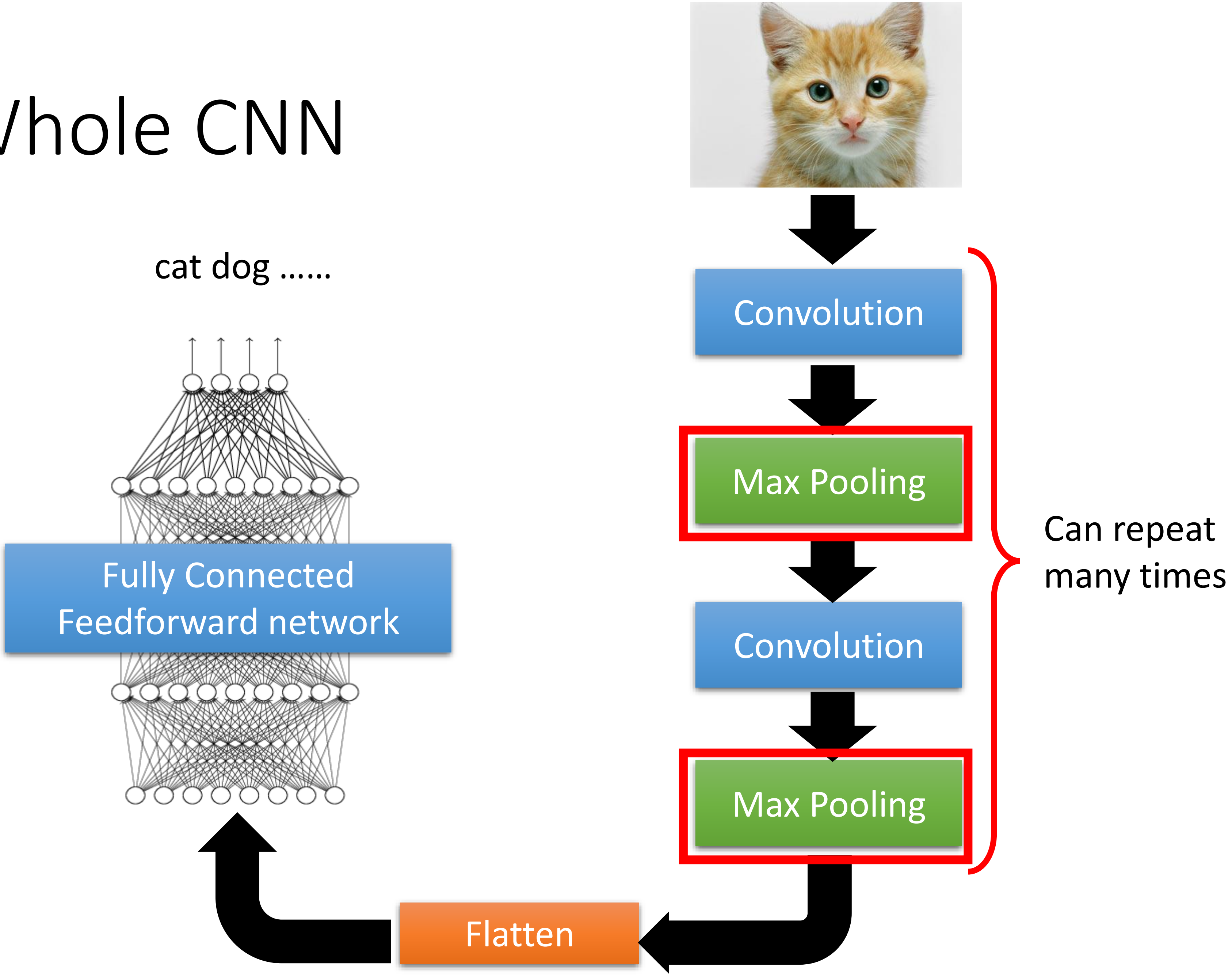
6 x 6 image



Less parameters!

Even less parameters!

The Whole CNN



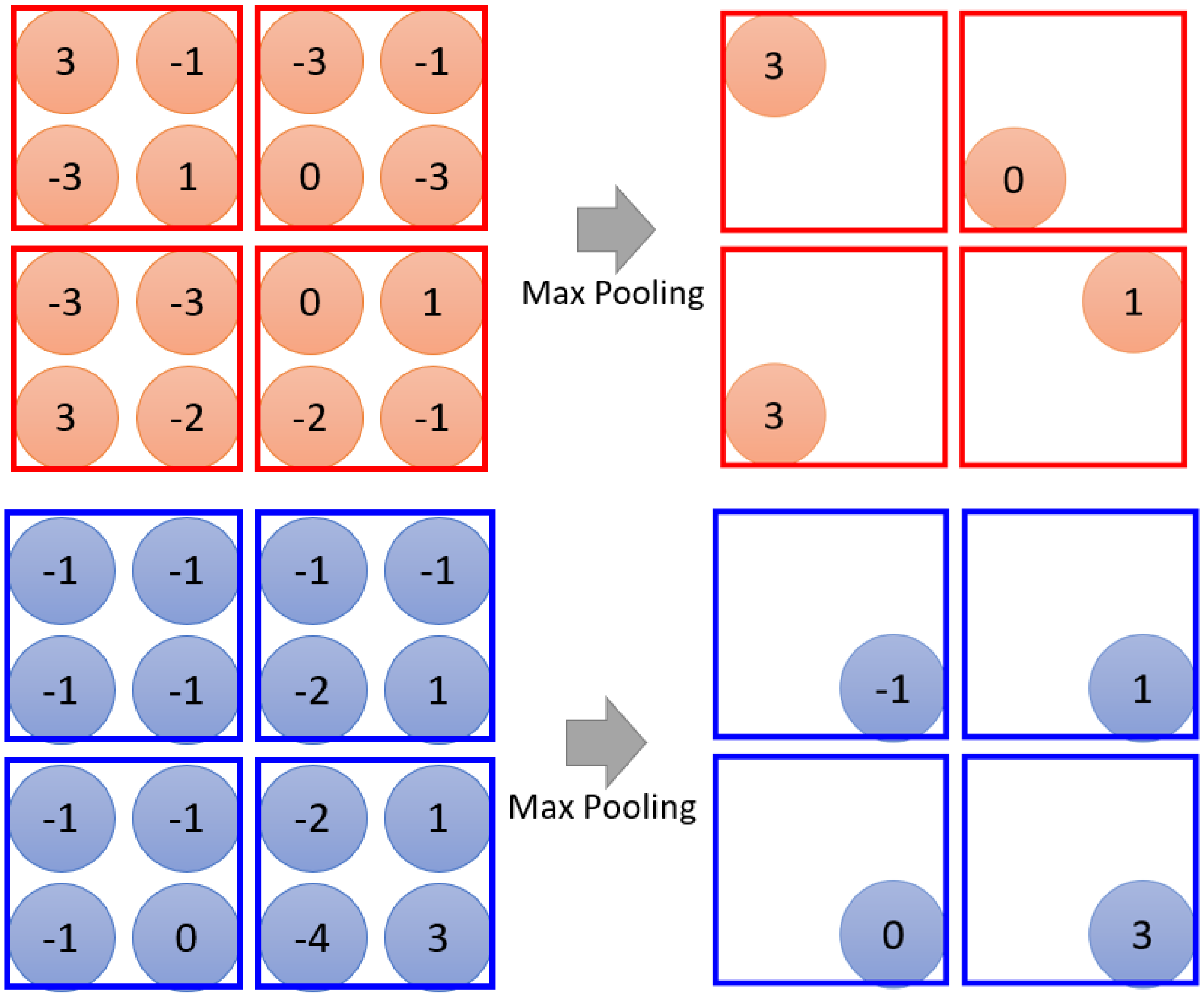
CNN – Max Pooling

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

Filter 1

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

Filter 2

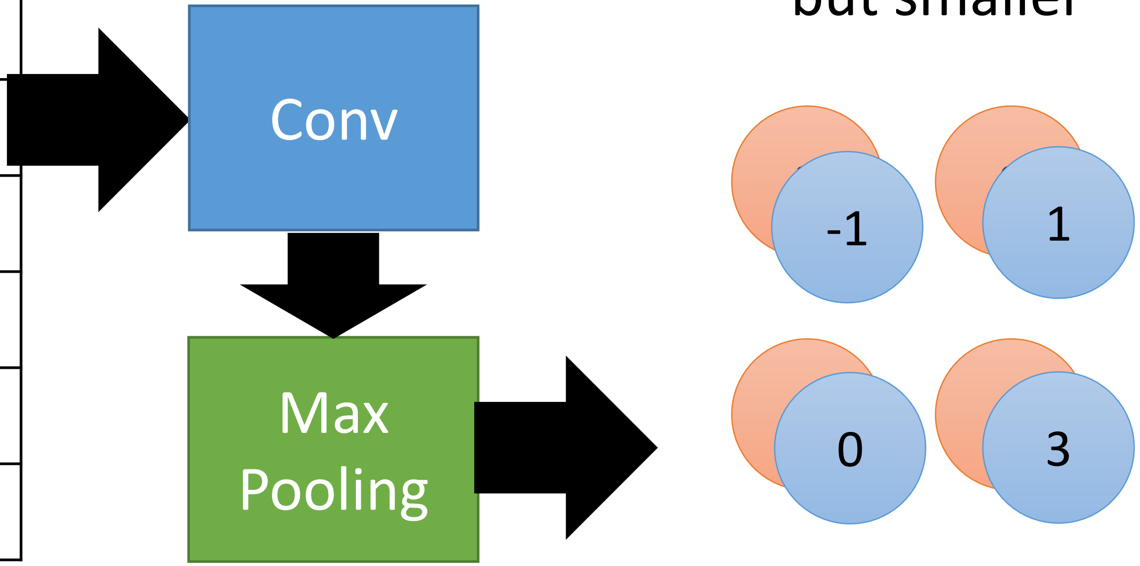


CNN – Max Pooling

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

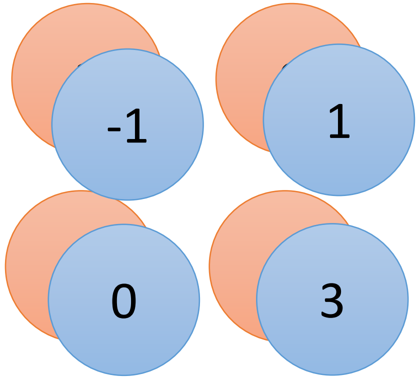
New image
but smaller



2 x 2 image

Each filter
is a channel

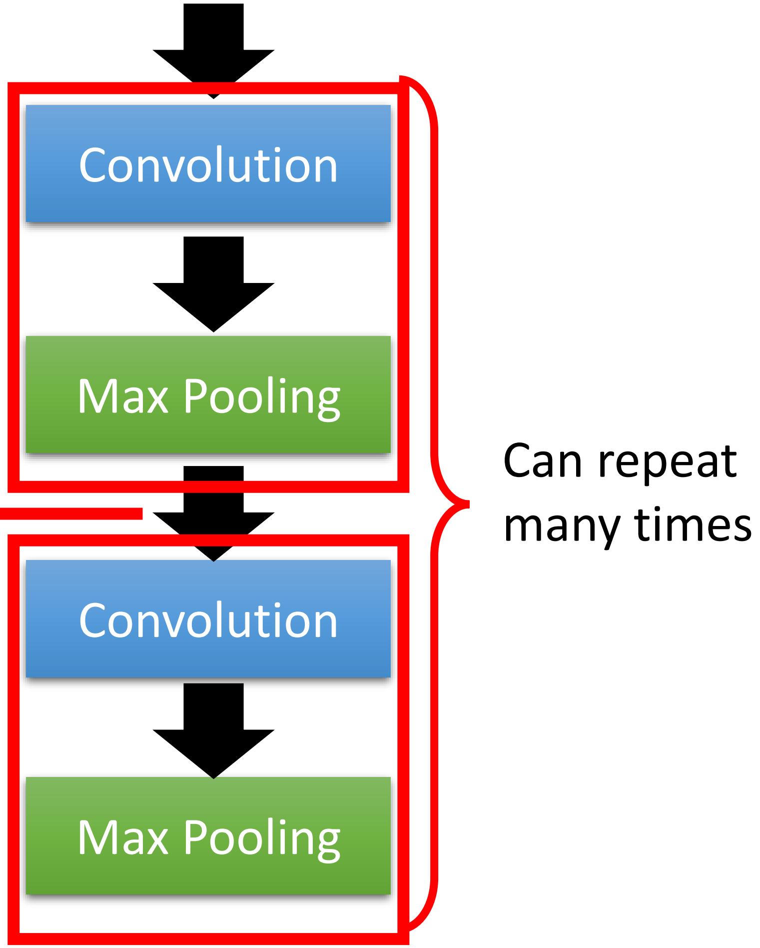
The Whole CNN



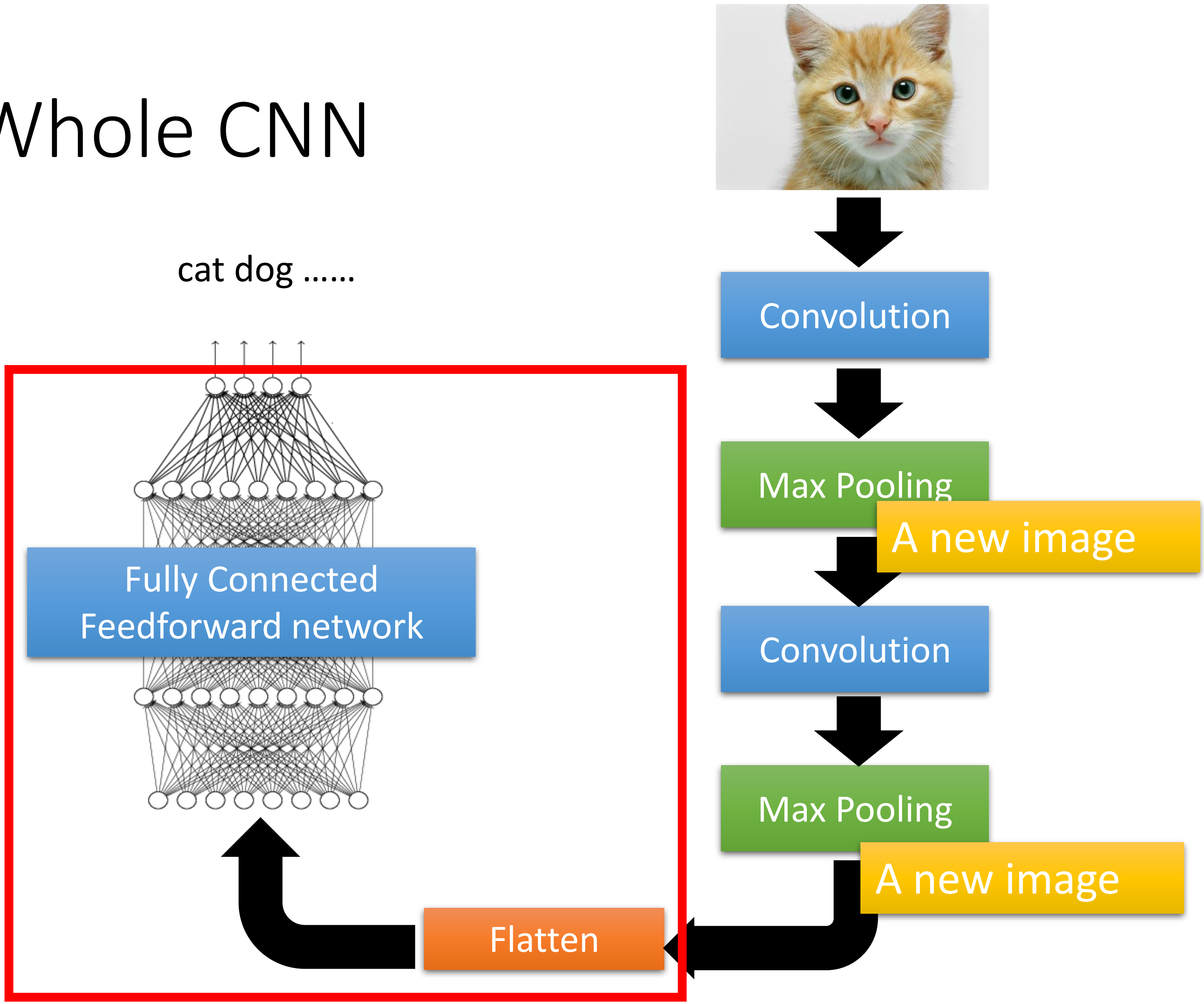
A new image

Smaller than the original image

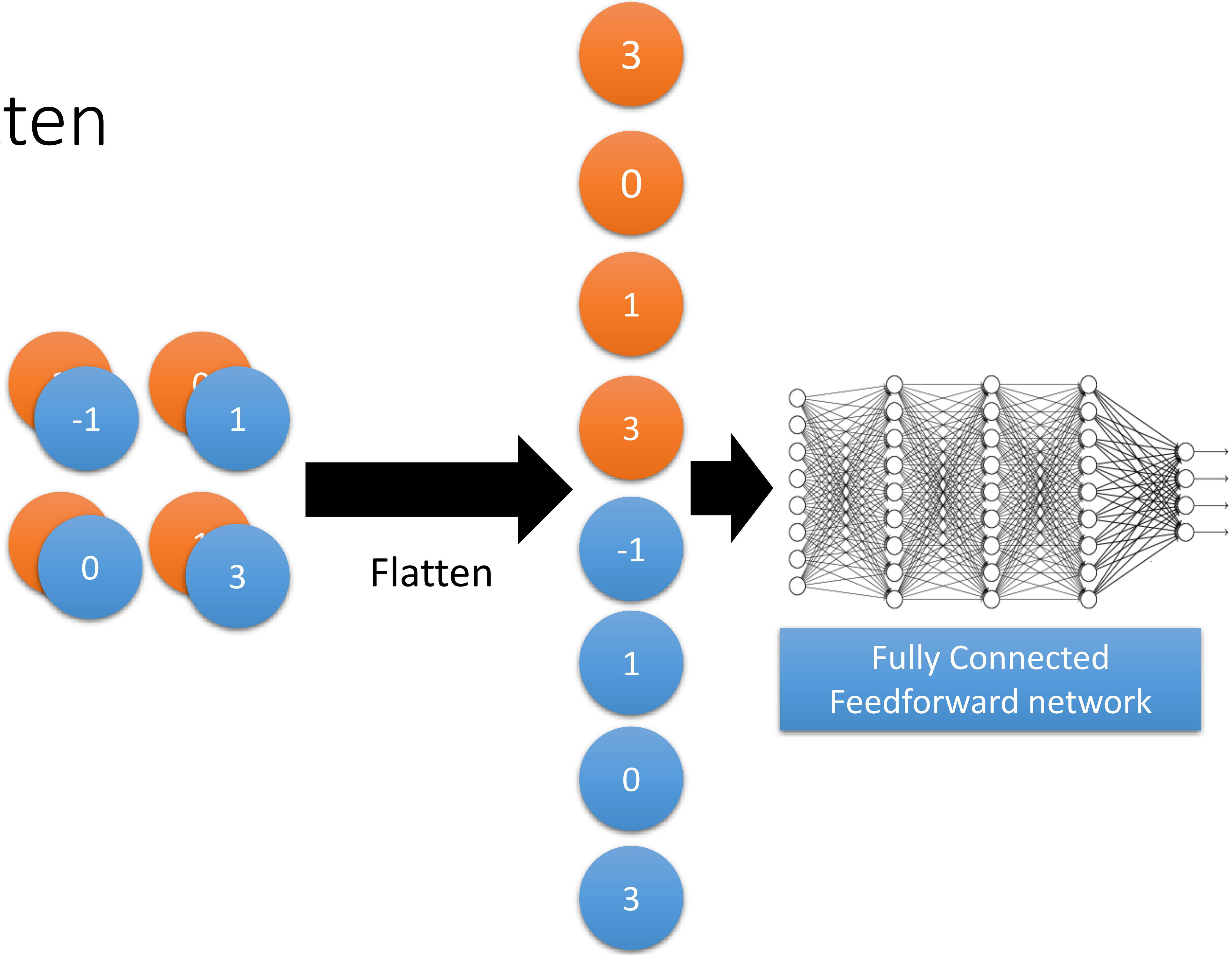
The number of the channels is the number of filters



The Whole CNN



Flatten


-1
1
0
3
Flatten
3
0
1
3
-1
1
0
3
Fully Connected
Feedforward network

Padding

Filter

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

6×6

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

4×4

Padding

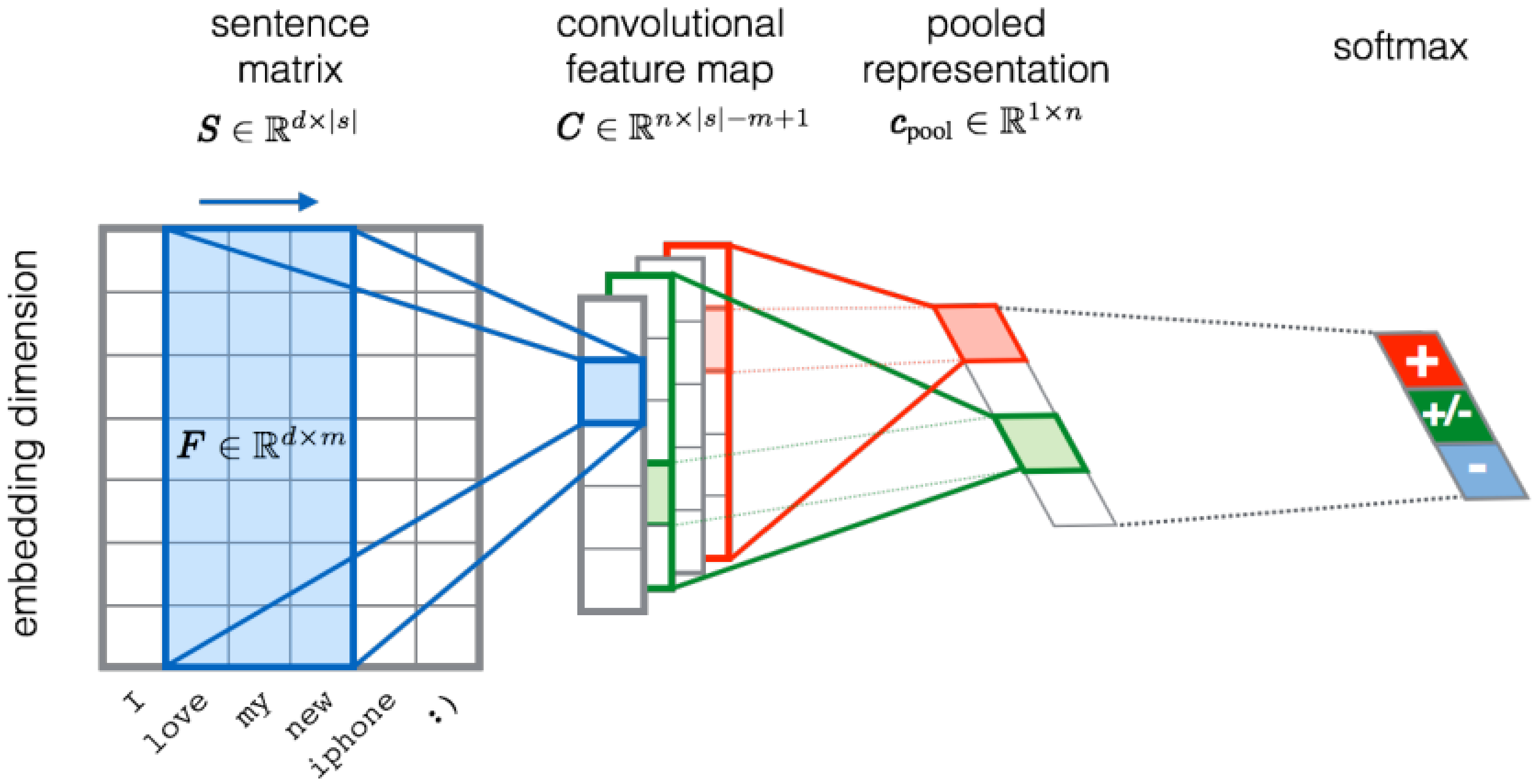
0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	0
0	0	1	0	0	1	0	0
0	0	0	1	1	0	0	0
0	1	0	0	0	1	0	0
0	0	1	0	0	1	0	0
0	0	0	1	0	1	0	0
0	0	0	0	0	0	0	0

6×6

2	-2	-1	1	-2	0
-2	3	-1	-3	-1	-2
-2	-3	1	0	-3	-2
2	-3	-3	0	1	-2
-2	3	-2	-2	-1	-1
-1	-2	2	-3	0	0

6×6

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.

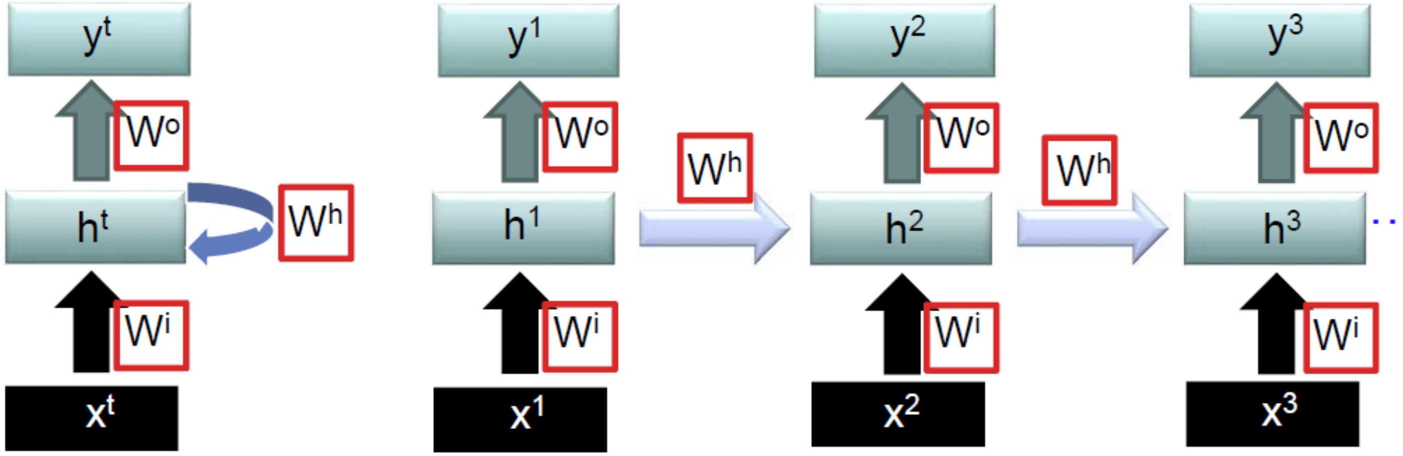
Recurrent Neural Network

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.



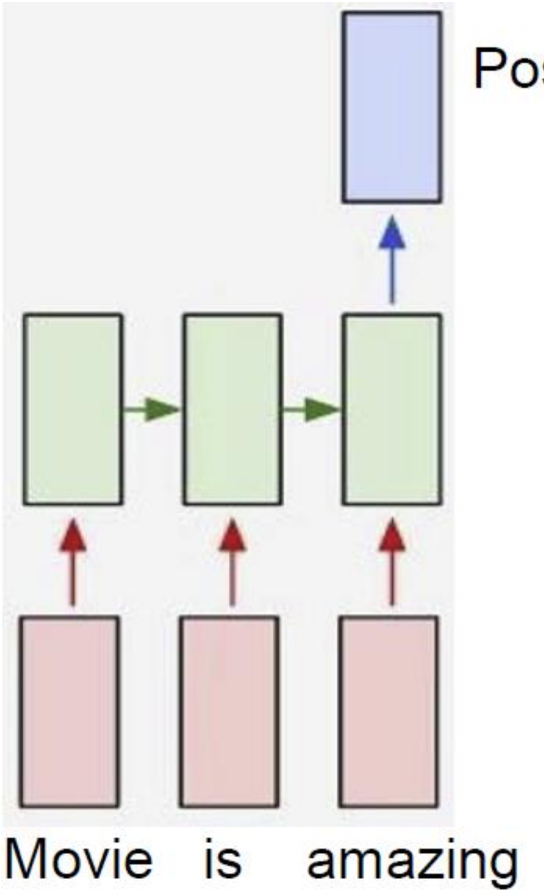
$$y^t, h^t = f(x^t, h^{t-1}; W^i, W^h, W^o)$$

$$h^t = \sigma(W^i x^t + W^h h^{t-1})$$

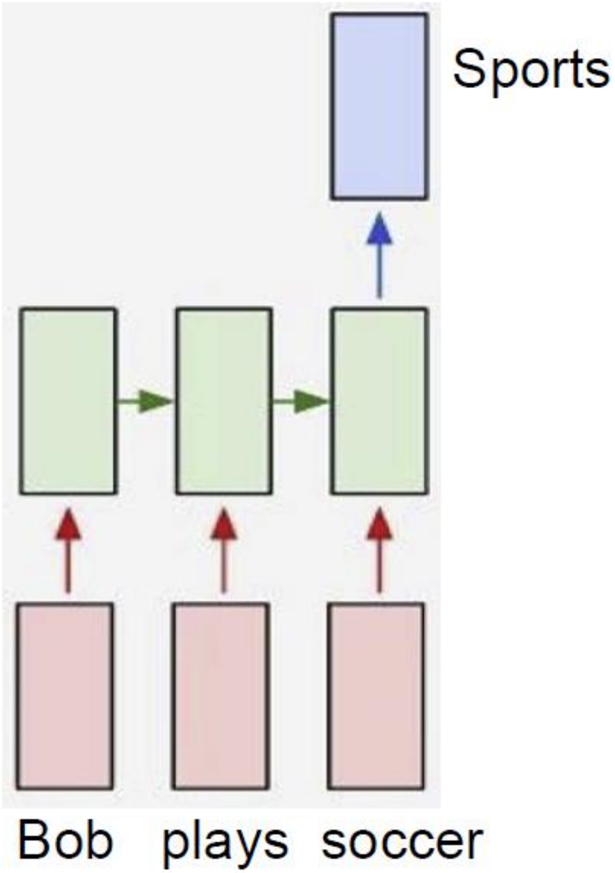
$$y^t = softmax(W^o h^t)$$

RNN Types: Many to One

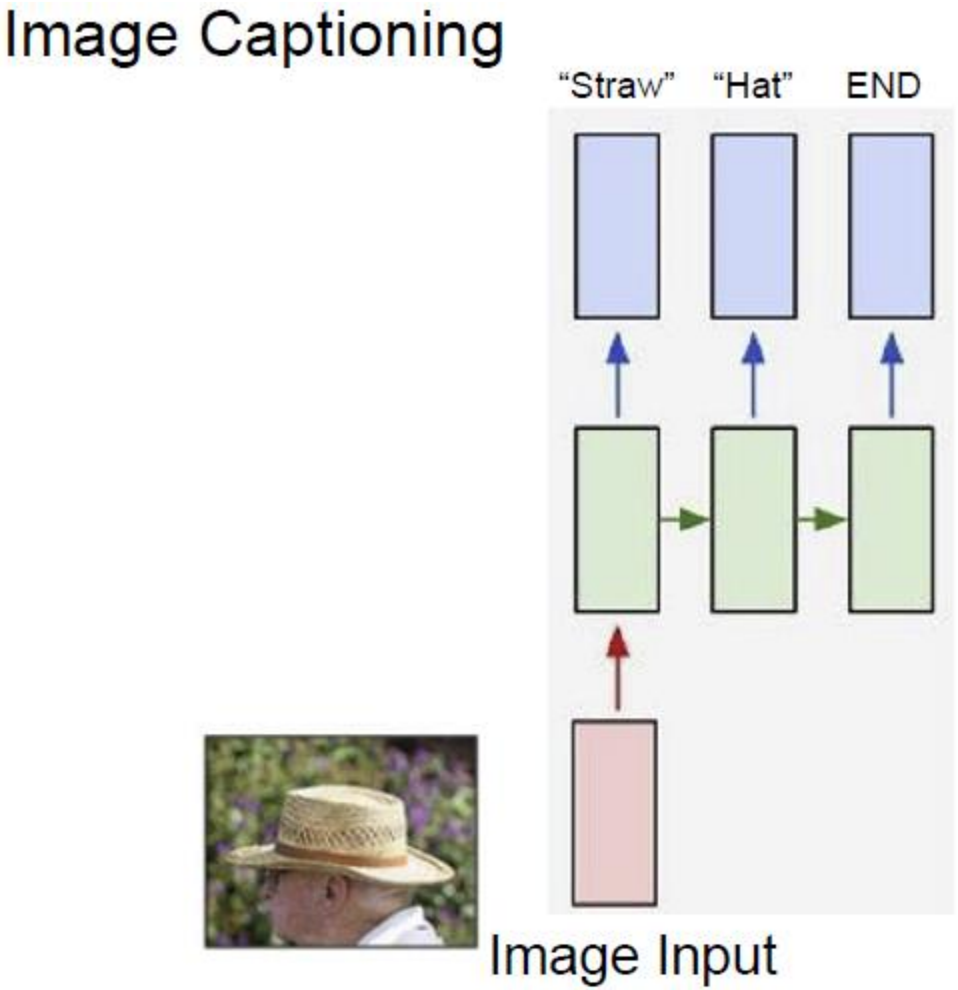
- Sentiment Classification



Text Categorization

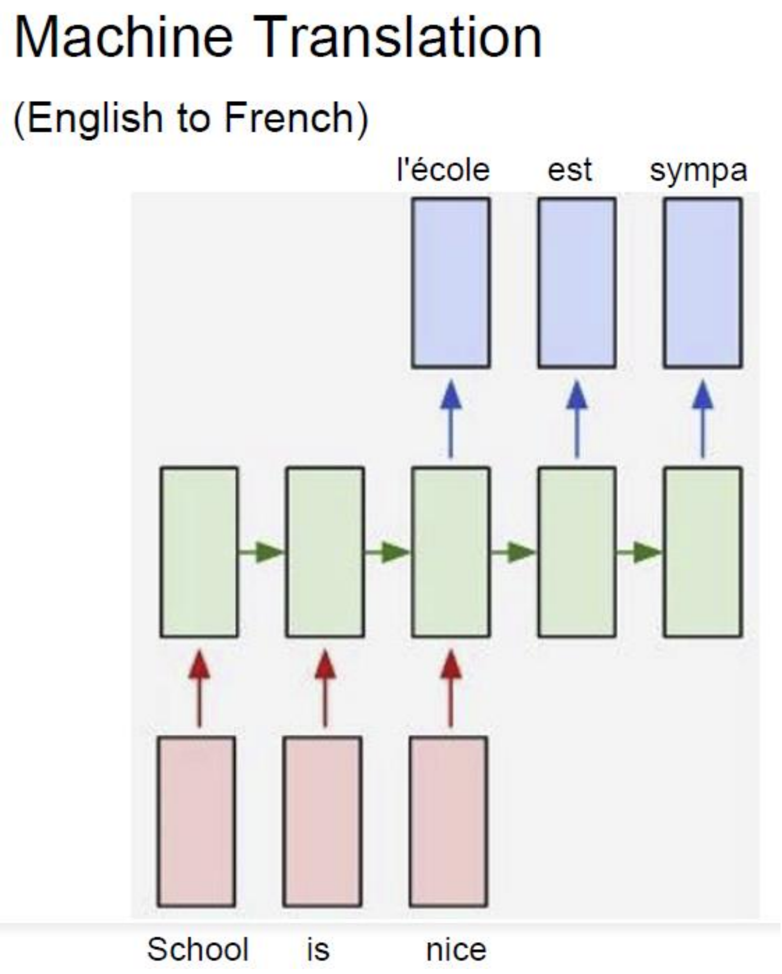


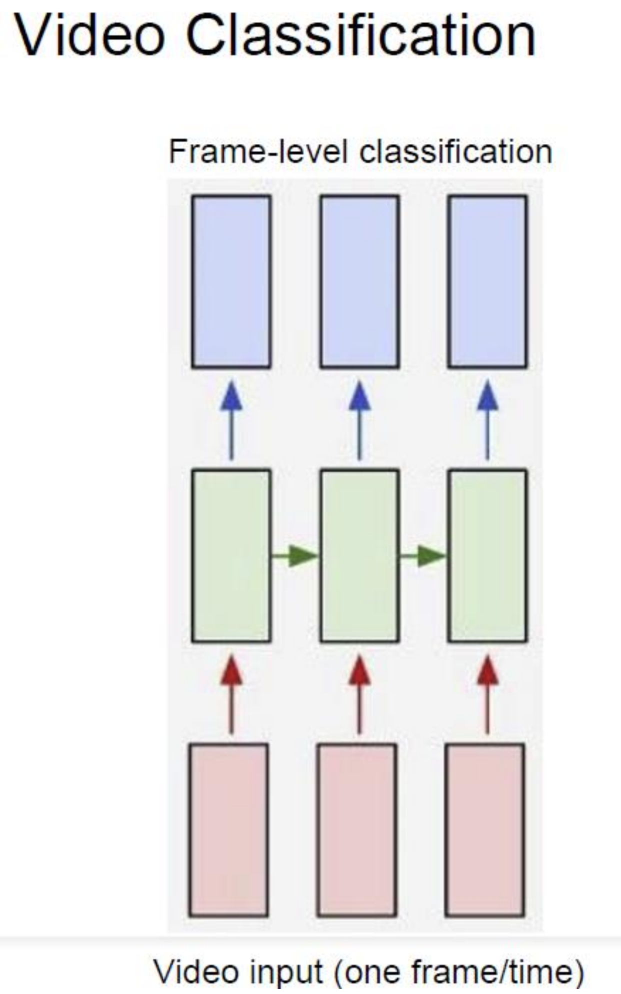
RNN Types: One to Many



http://www-cs-faculty.stanford.edu/people/karpathy/cvpr2015.pdf

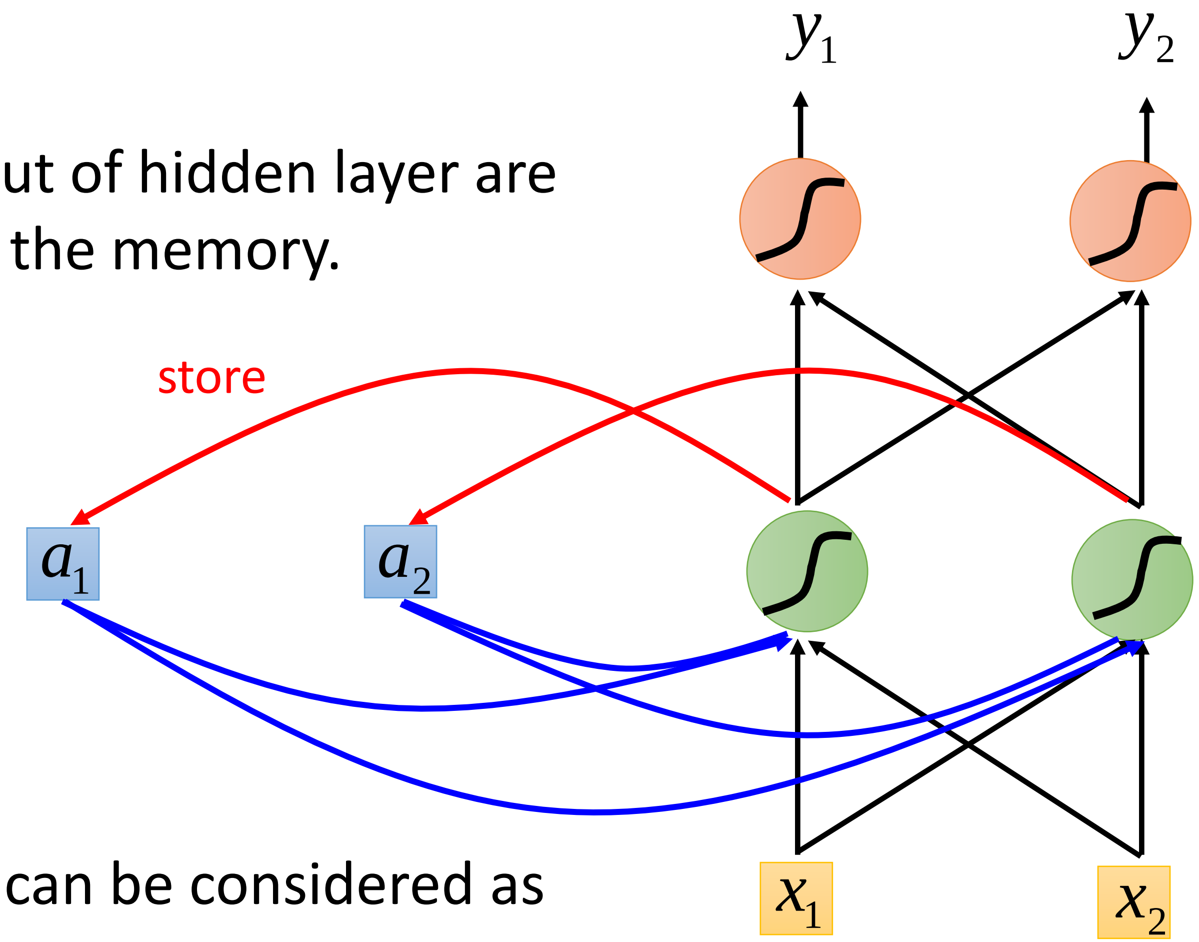
RNN Types: Many to Many





Recurrent Neural Network (RNN)

The output of hidden layer are stored in the memory.

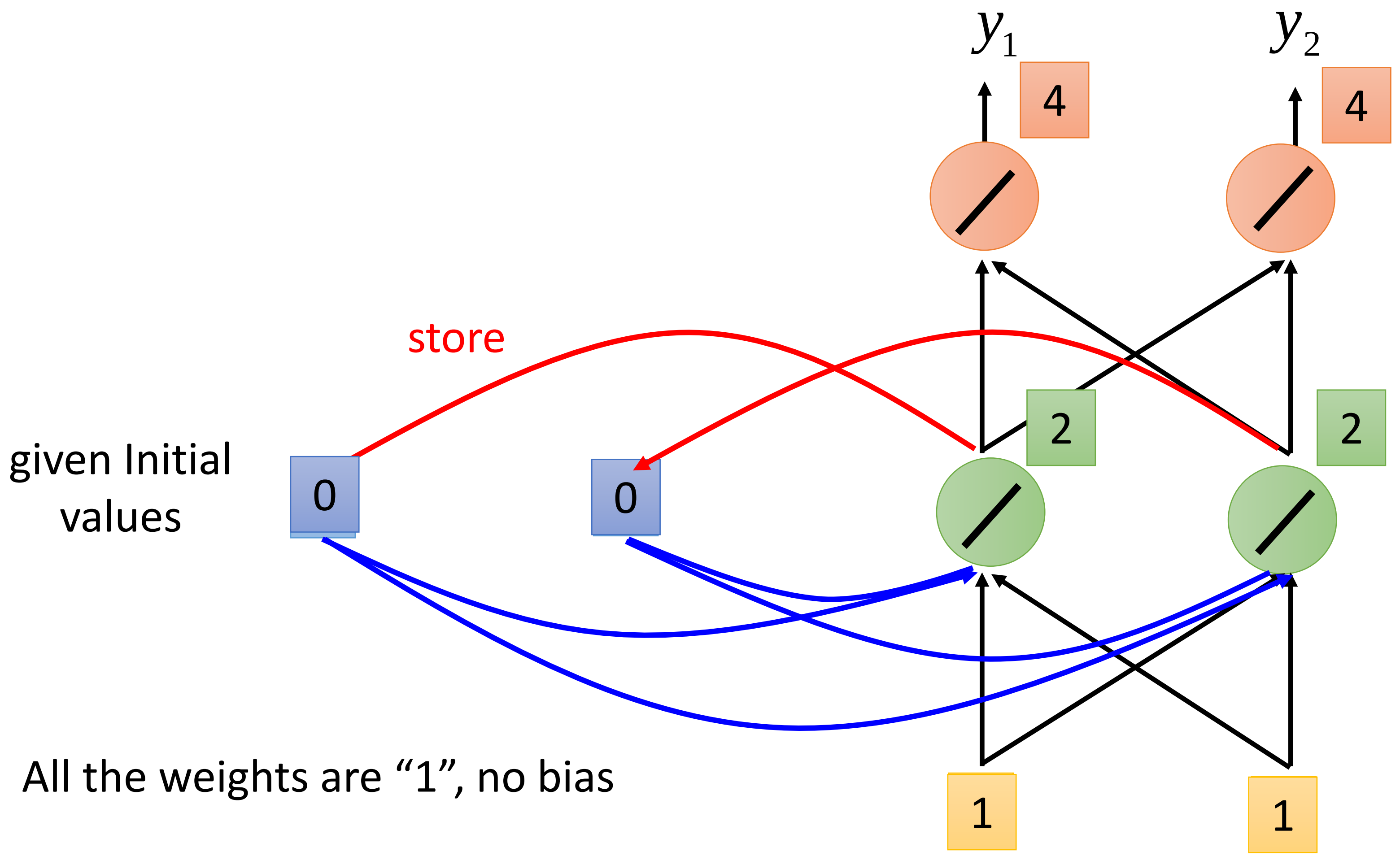


Memory can be considered as another input.

Example

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

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



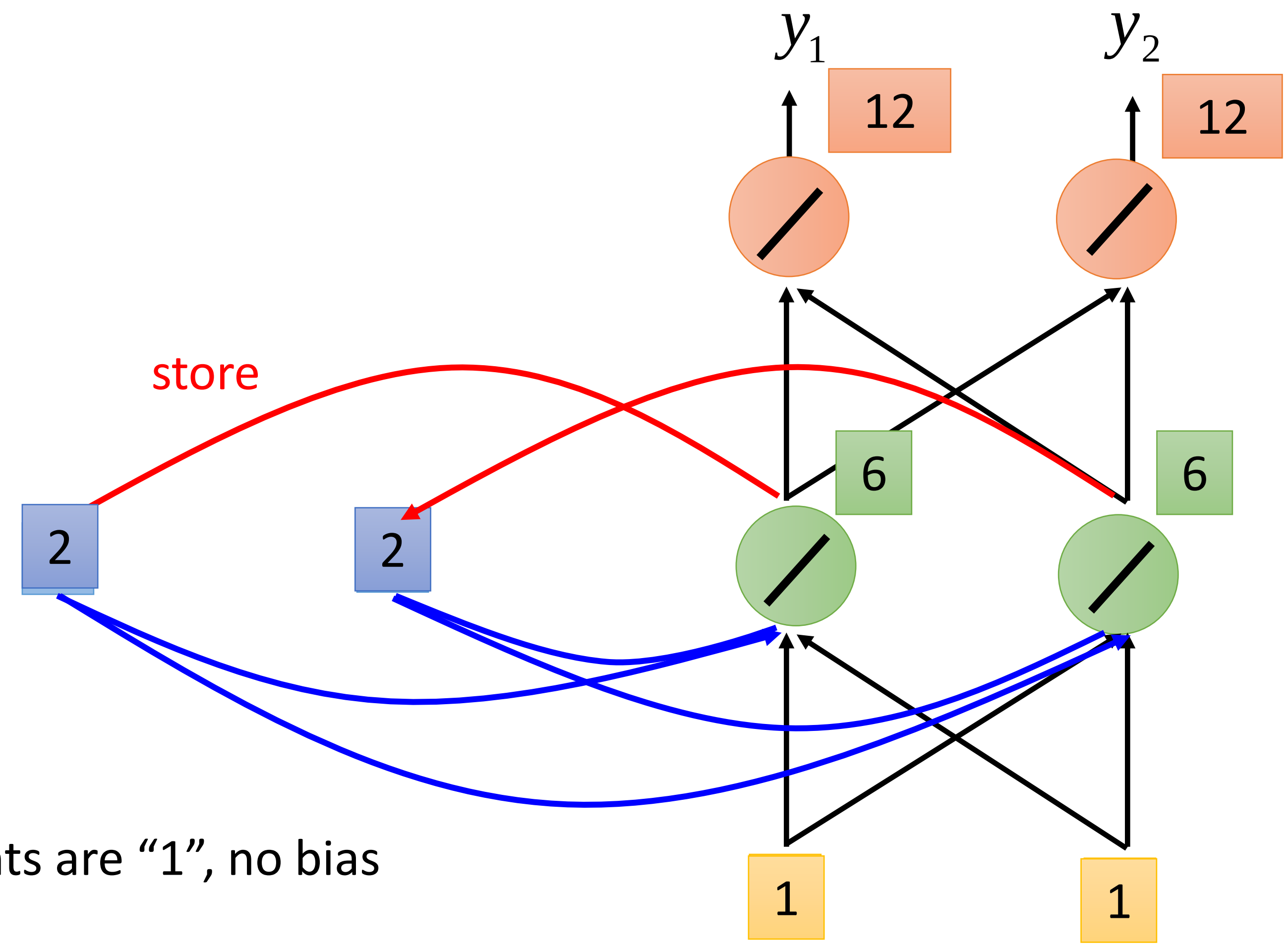
All the weights are "1", no bias

All activation functions are linear

Example

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

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



All the weights are "1", no bias

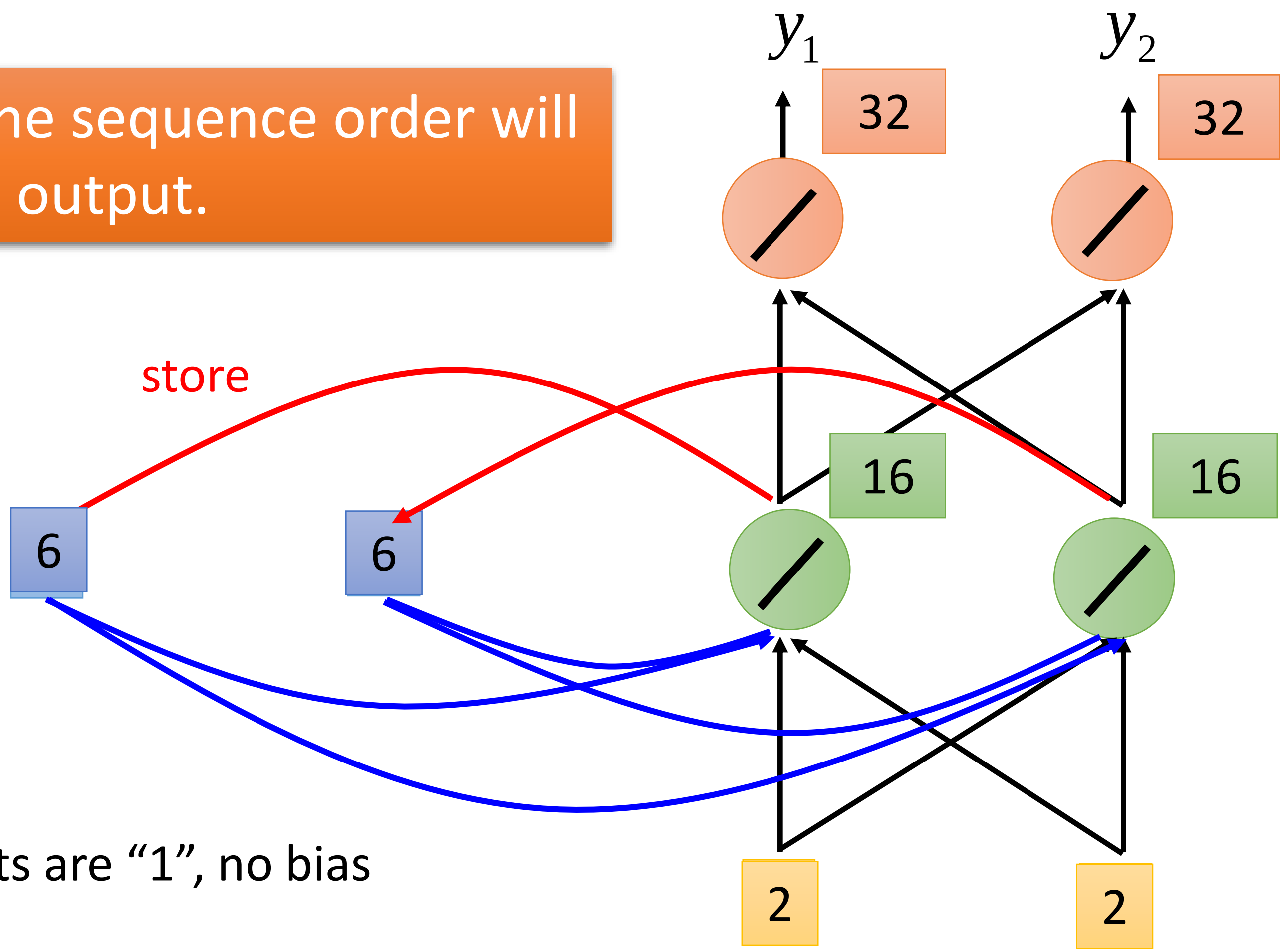
All activation functions are linear

Example

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

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

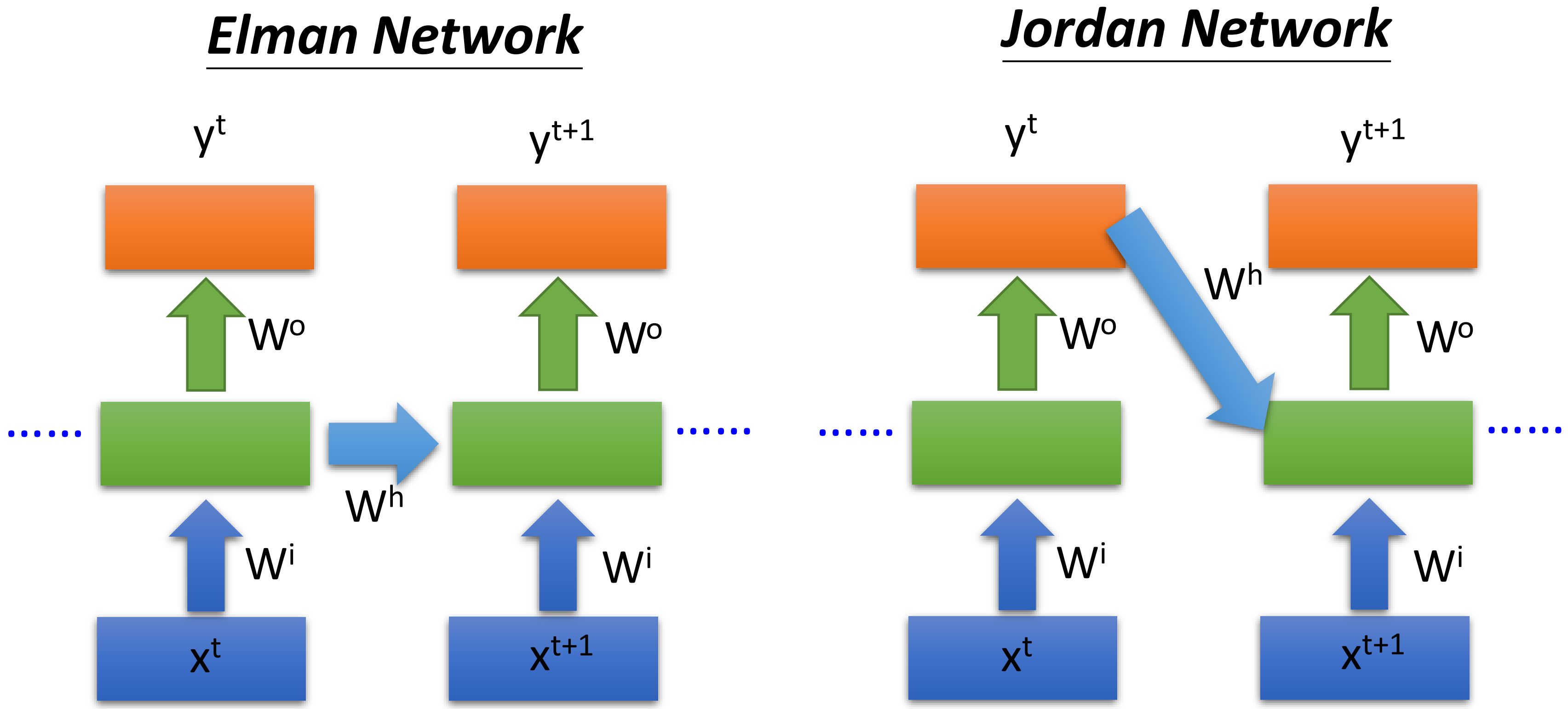
Changing the sequence order will change the output.



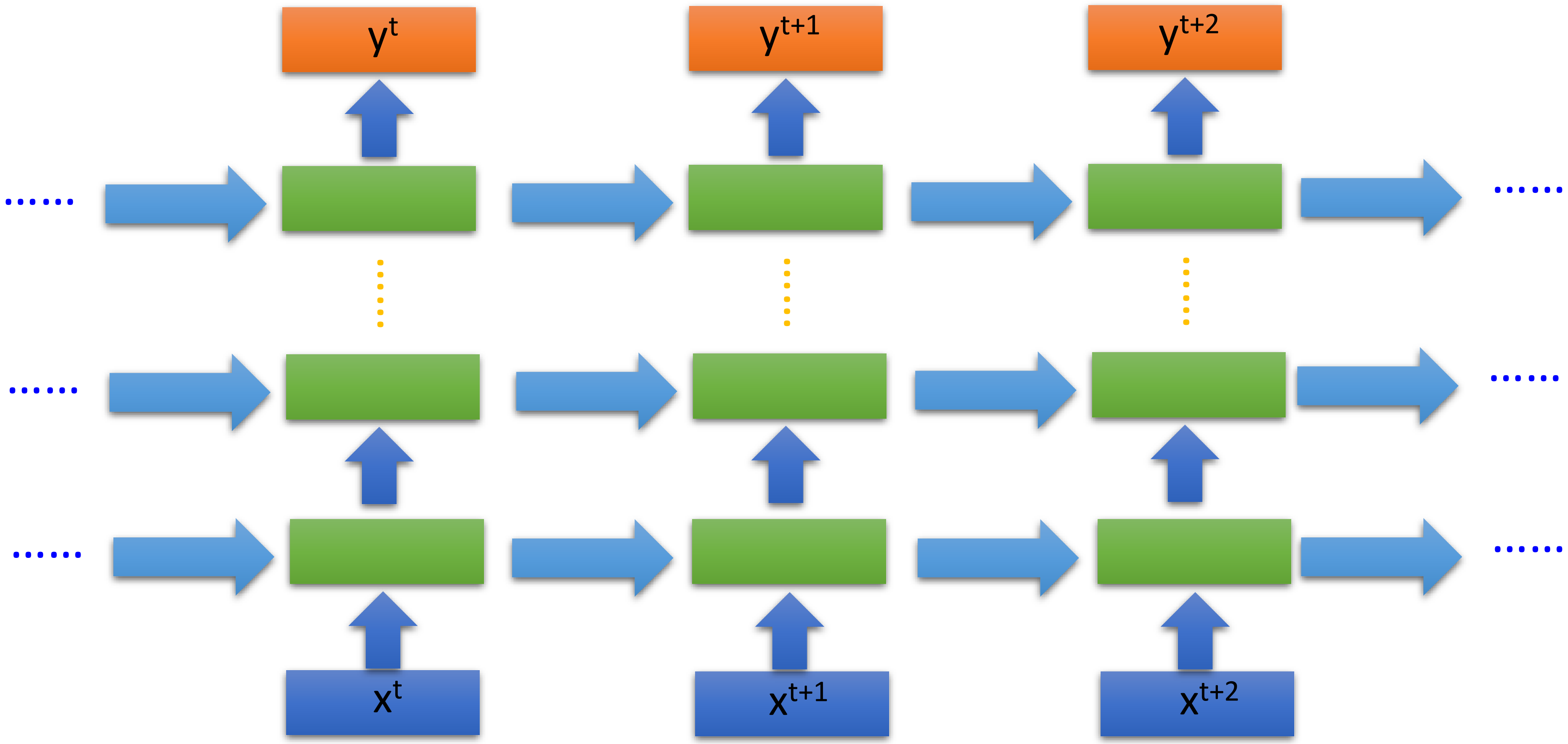
All the weights are “1”, no bias

All activation functions are linear

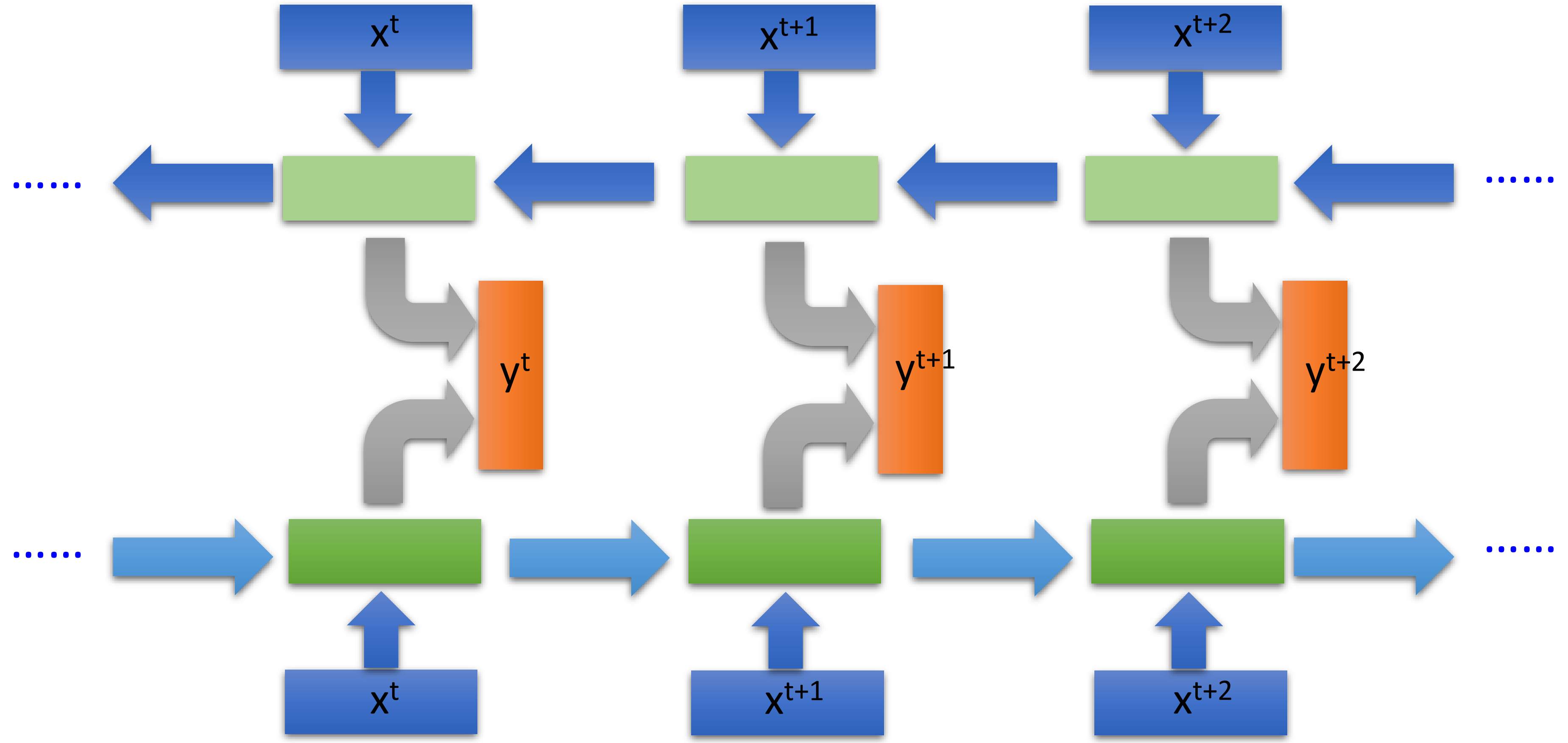
Elman Network & Jordan Network



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)

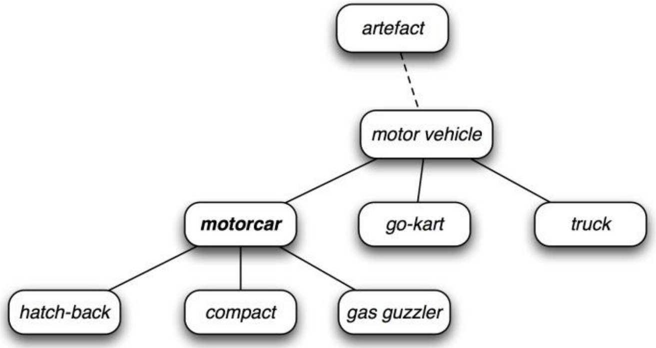
Summary of RNN

- 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")

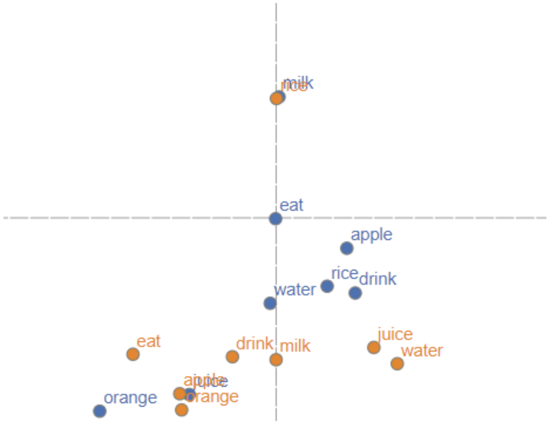
$$\underset{\text{car}}{[0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ \ldots\ 0]}\ \text{AND}\ \underset{\text{motorcycle}}{[0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ \ldots\ 0]} = 0$$

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.** They form the basis for all language-related tasks.
  - Word vectors as powerful features for various downstream **NLP tasks** since the vectors contain semantic information. Such as word classification, sentiment analysis.

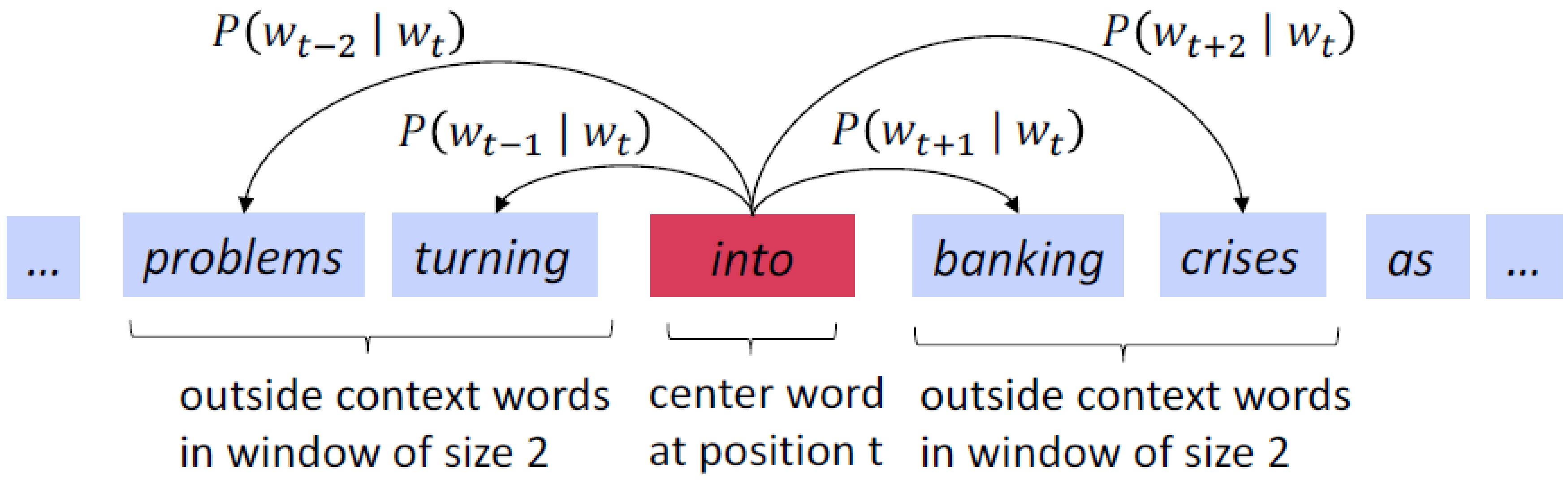
Word2Vec

- 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:** faster, easily incorporate a new sentence/document or add a word to vocab

Mikolov et al. 2013. Distributed representations of words and phrases and their compositionality

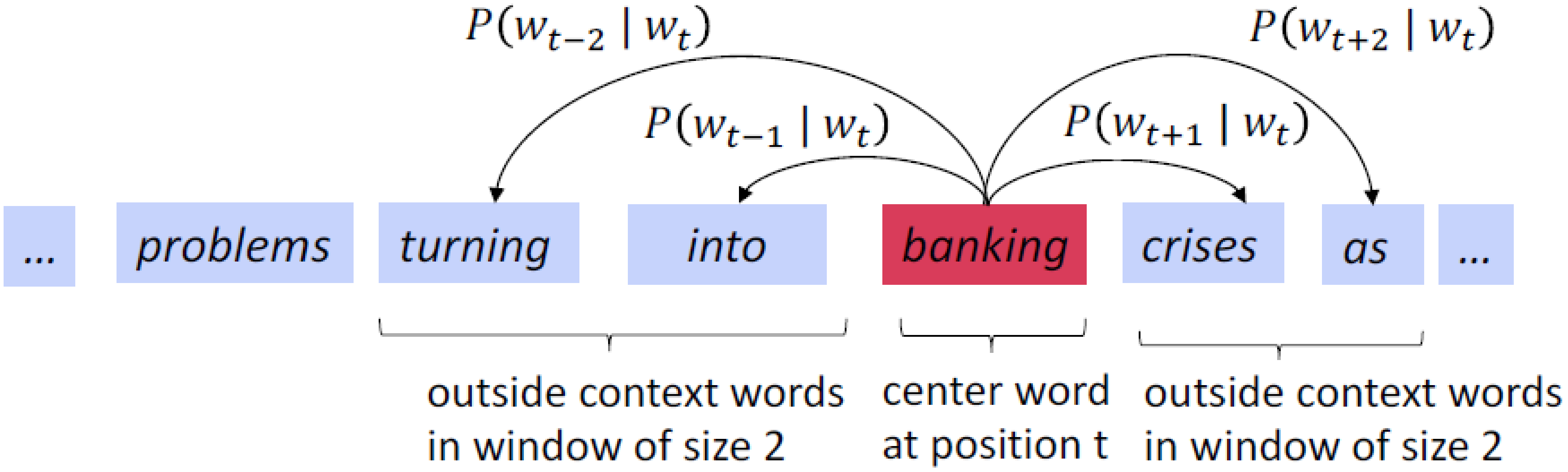
Word2Vec: Skip-Gram Model

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



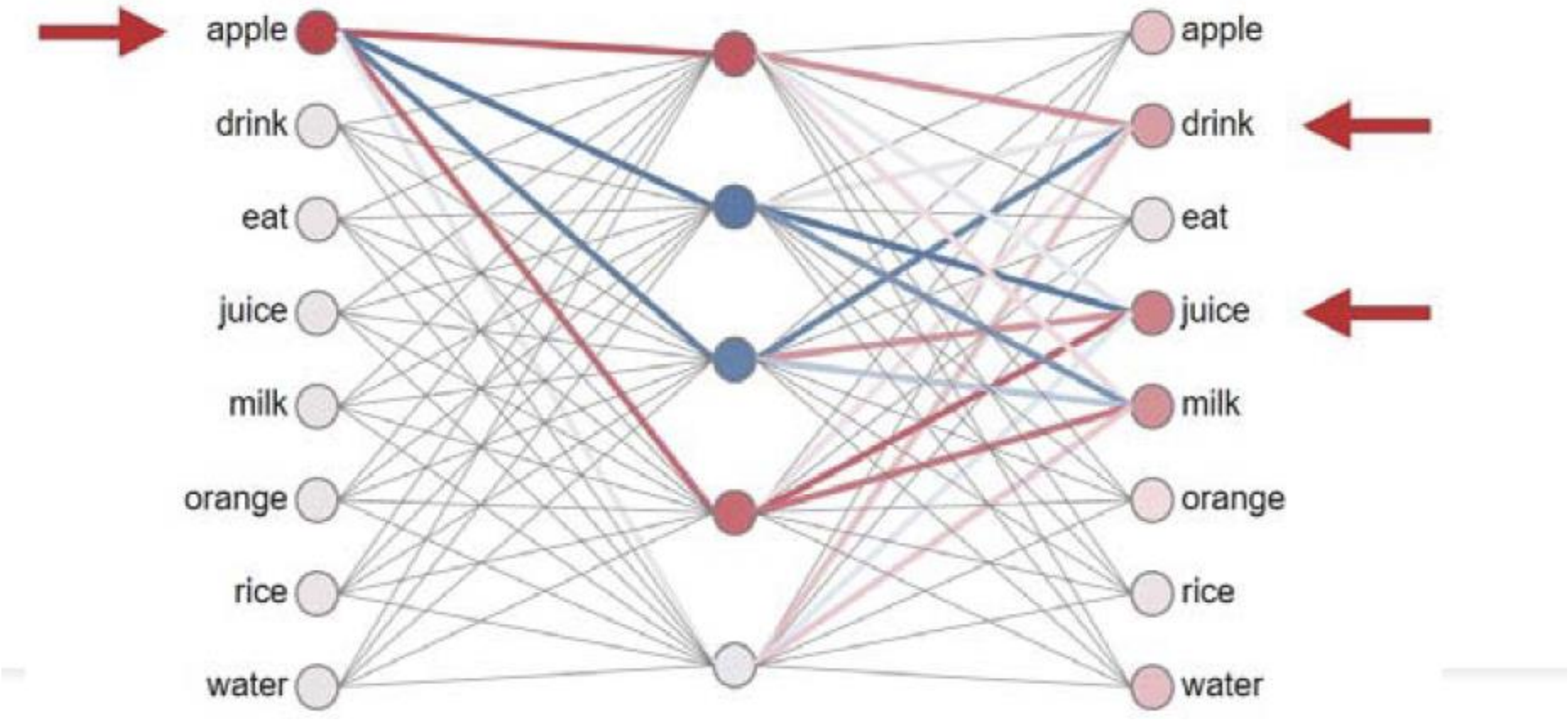
Word2Vec: Skip-Gram Model

- 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



https://ronxin.github.io/wevi/

Word2Vec: Objective Function

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

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

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:

$$p(o|c) = \frac{exp(\mathrm{u}_o^T \mathrm{v}_c)}{\sum_{w \in V} exp(\mathrm{u}_w^T \mathrm{v}_c)}$$

Word2Vec: Prediction Function

$$P(o|c) = \frac{\exp(\mathrm{u}_o^T \mathrm{v}_c)}{\sum_{w \in V} \exp(\mathrm{u}_w^T \mathrm{v}_c)}$$

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 $\mathbb{R}^n \to \mathbb{R}^n$

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

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

Compute All Vector Gradients

Recall: θ 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 vector for center words as word embeddings.

Remember: every word has two vectors

We then optimize these parameters

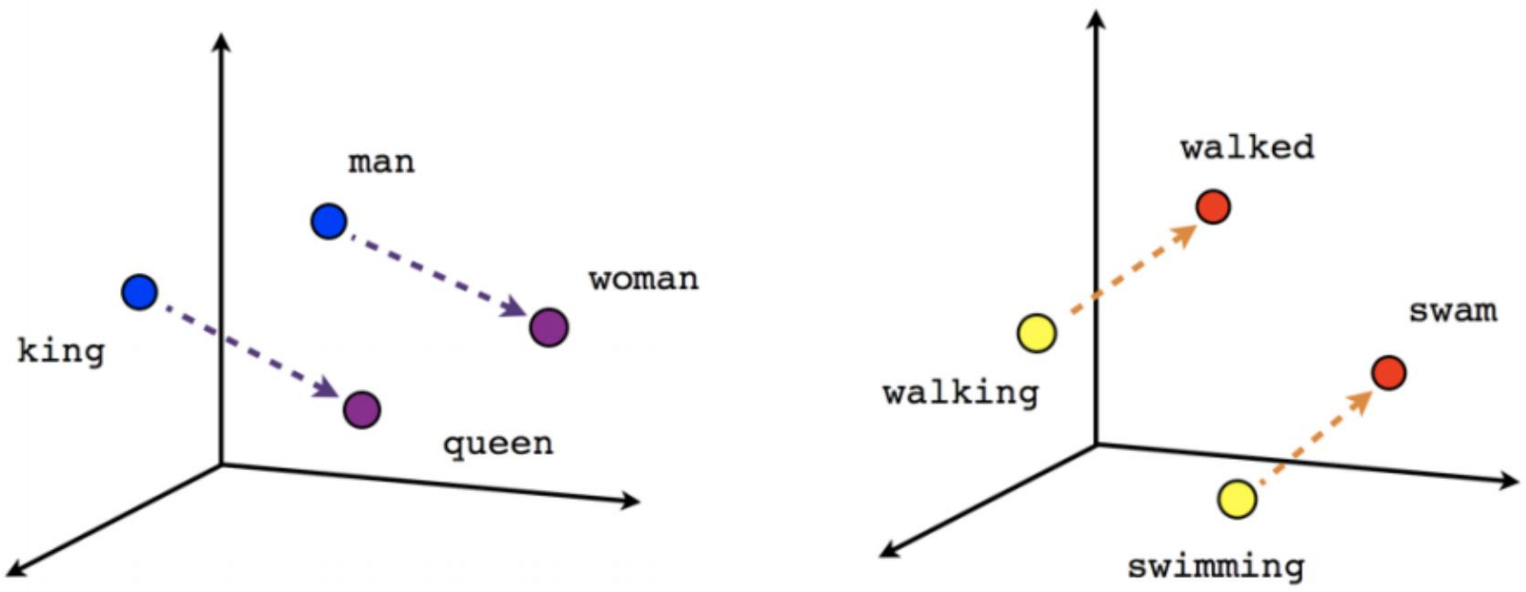
Skip-gram With Negative Sampling

- 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 prob. that negative pairs appear.

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J_t(\theta)$$

$$J_t(\theta) = \log \sigma(\mathrm{u}_o^T \mathrm{v}_c) + \sum_{i=1}^{k} \mathbb{E}_{i \sim P(w)} (\log \sigma(-\mathrm{u}_i^T \mathrm{v}_c))$$

Word2Vec Examples



Male-Female Verb tense

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