CS826 Deep Learning Theory and Practice

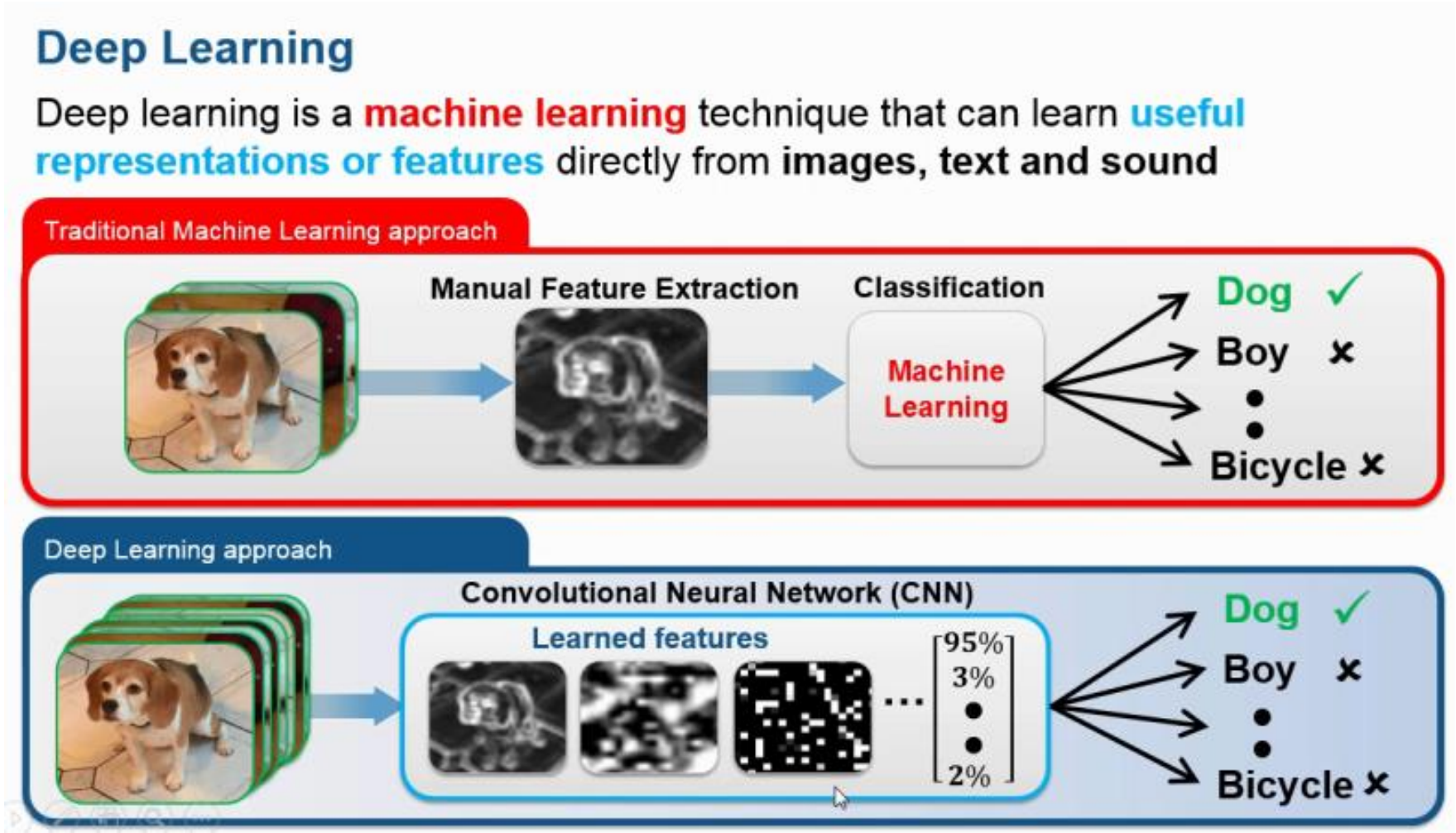
University of Strathclyde Glasgow

Deep Learning Algorithms

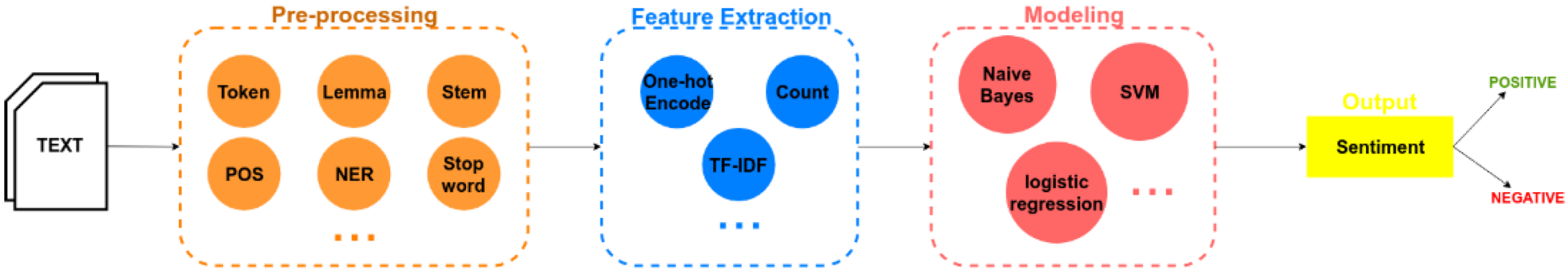
Learning objectives

- Revise: Machine Learning and MLP
- Different types of deep learning algorithms.
- The concept and theoretical of top 3 deep learning algorithms will be explained
    - CNNs
    - RNNs
    - LSTMs

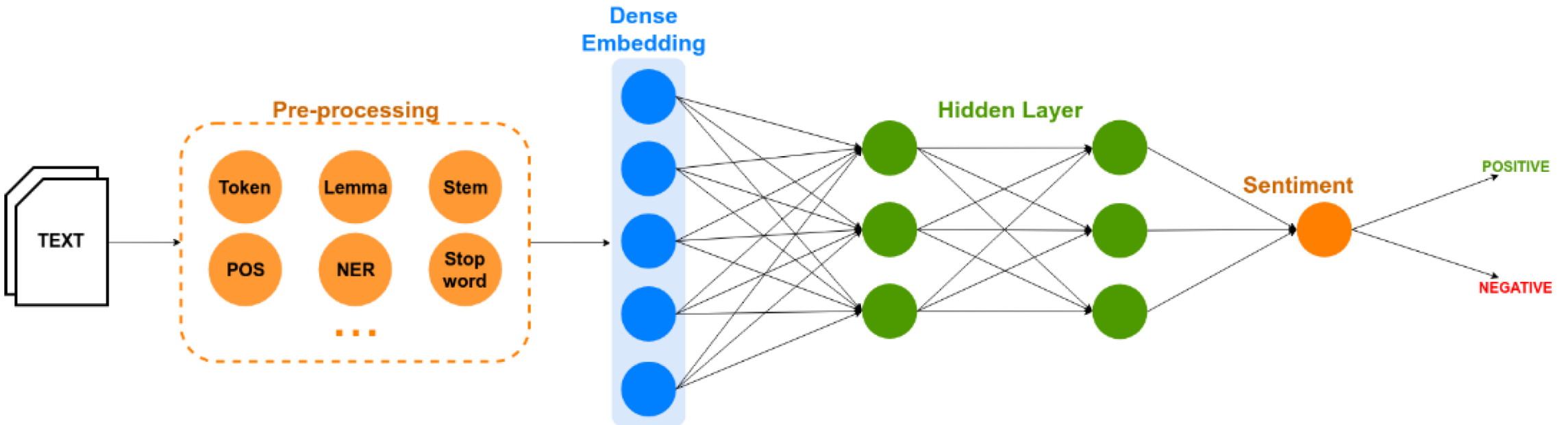
Machine learning (ML) and Deep Learning

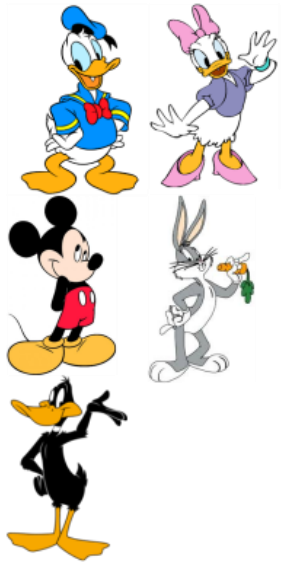


Machine Learning

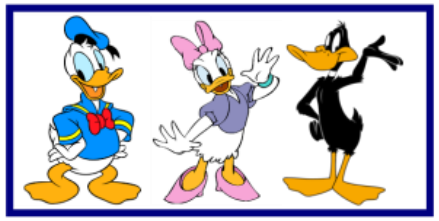


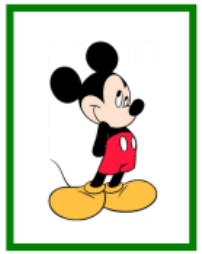
Deep Learning





Unsupervised Learning





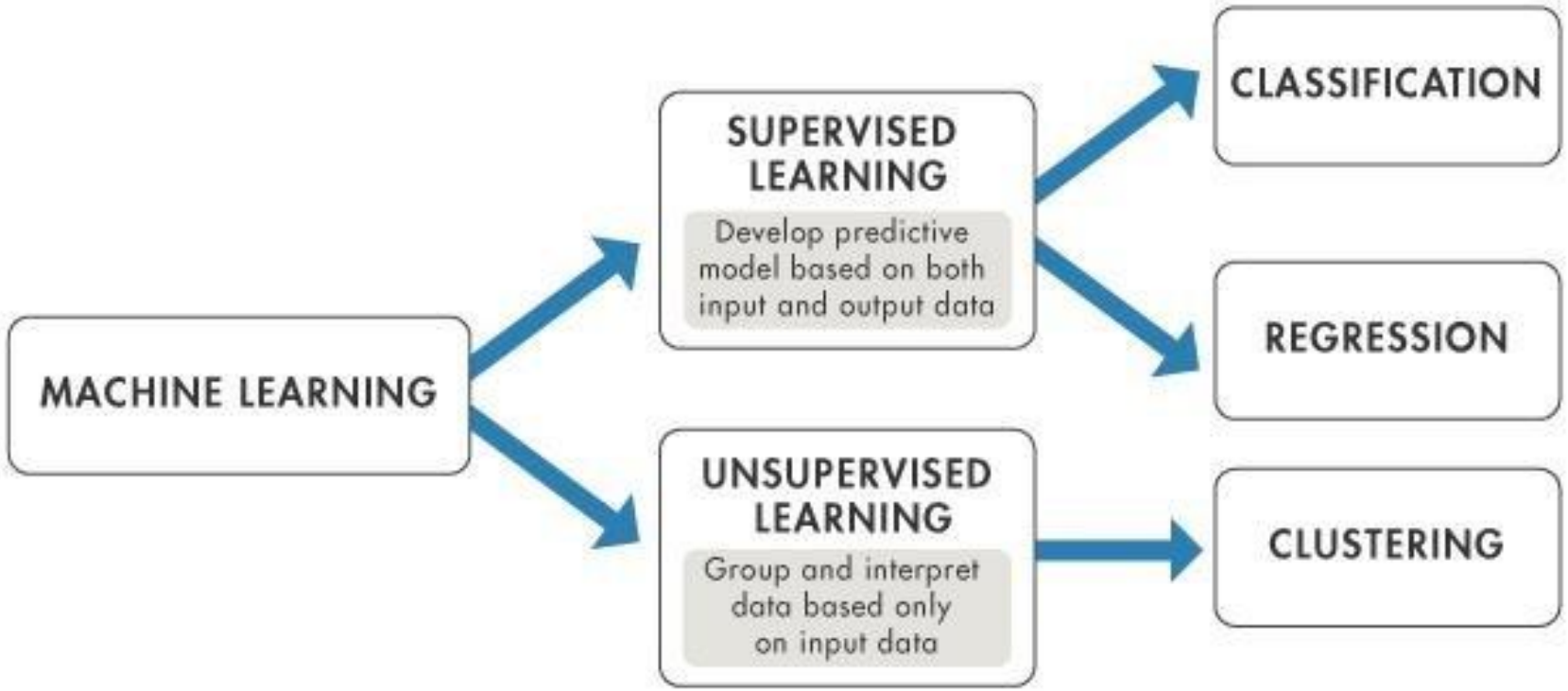
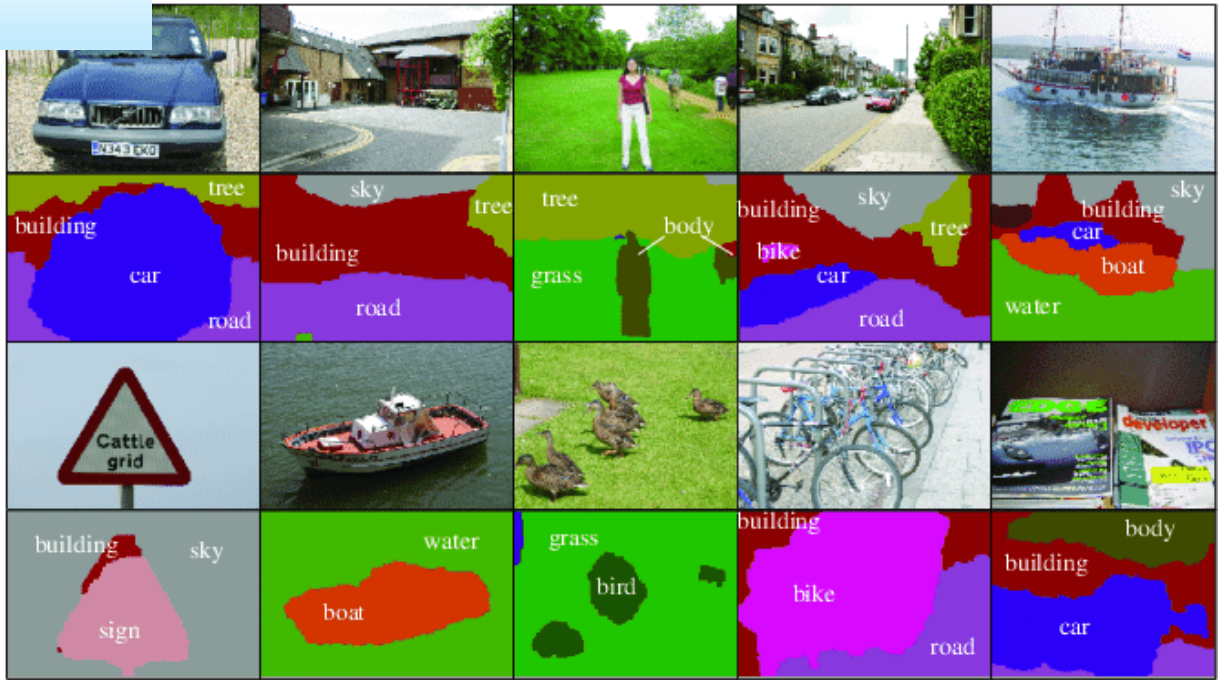
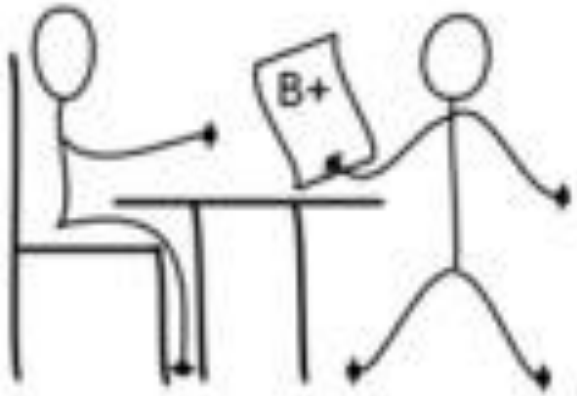


Image Labelling

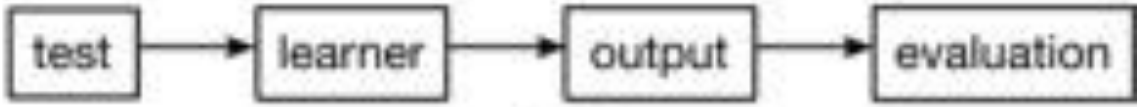


studying



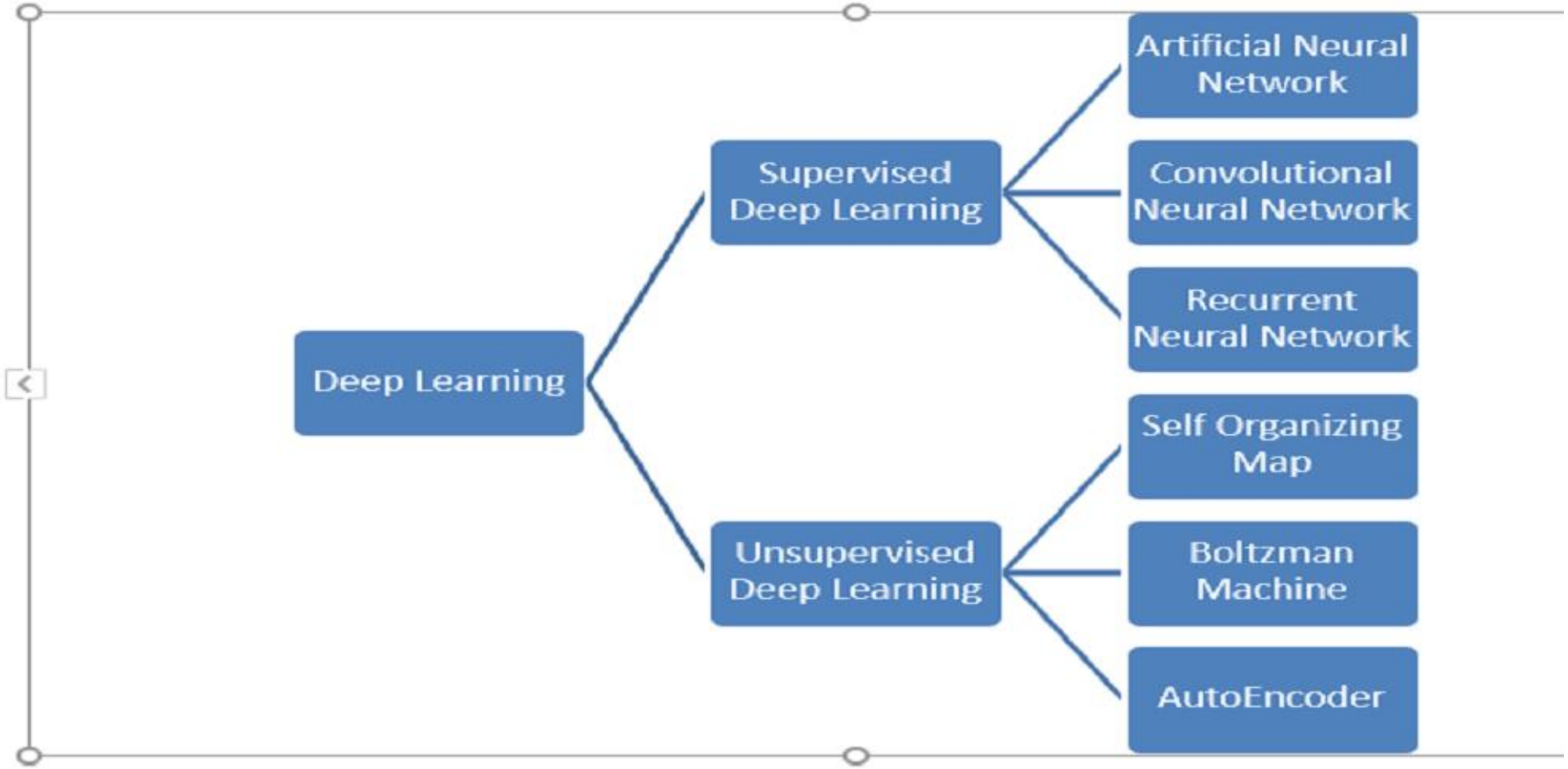
evaluation

training

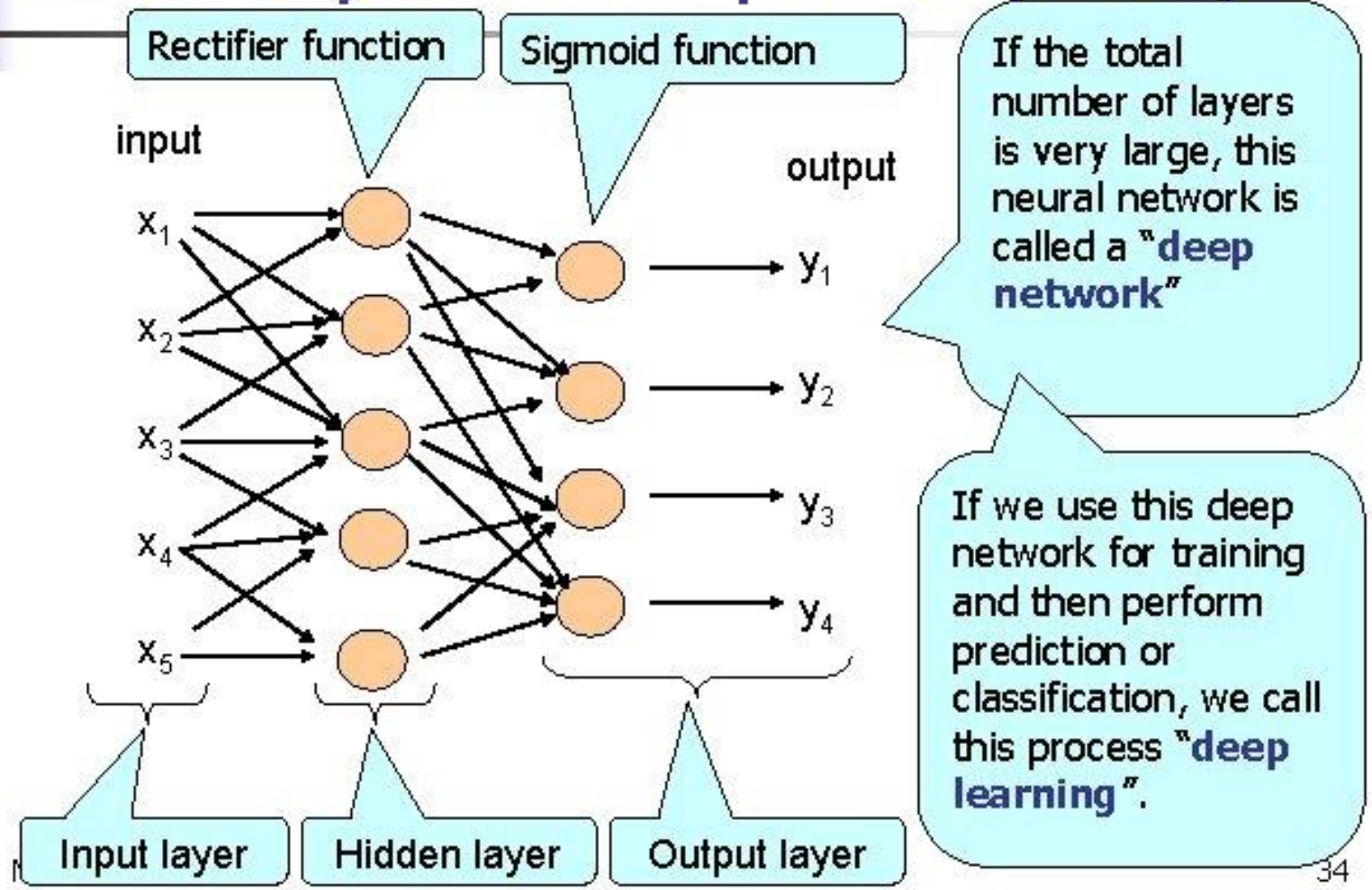


training

Deep Learning Algorithms



Multi-layer Perceptron (MLP)

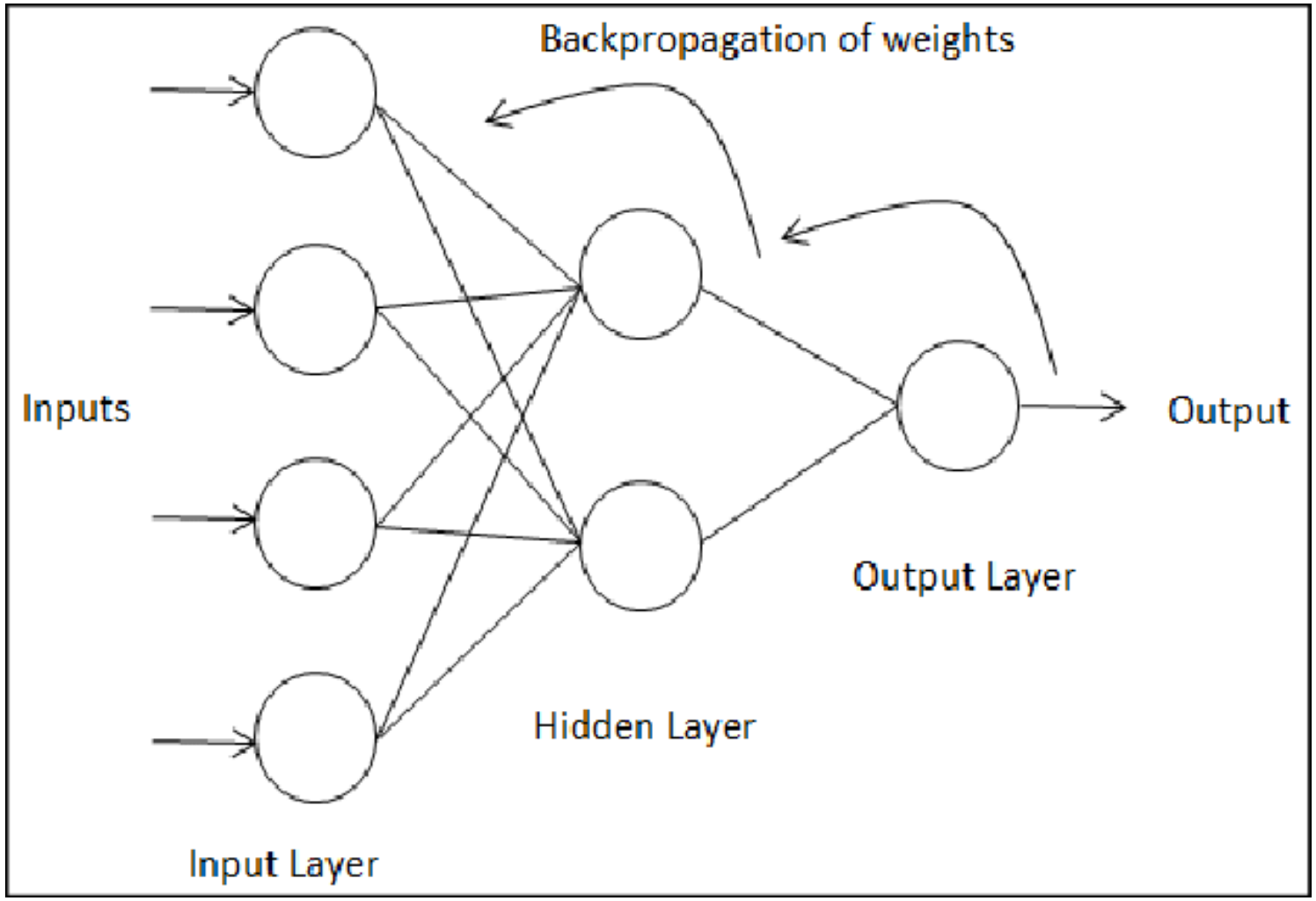


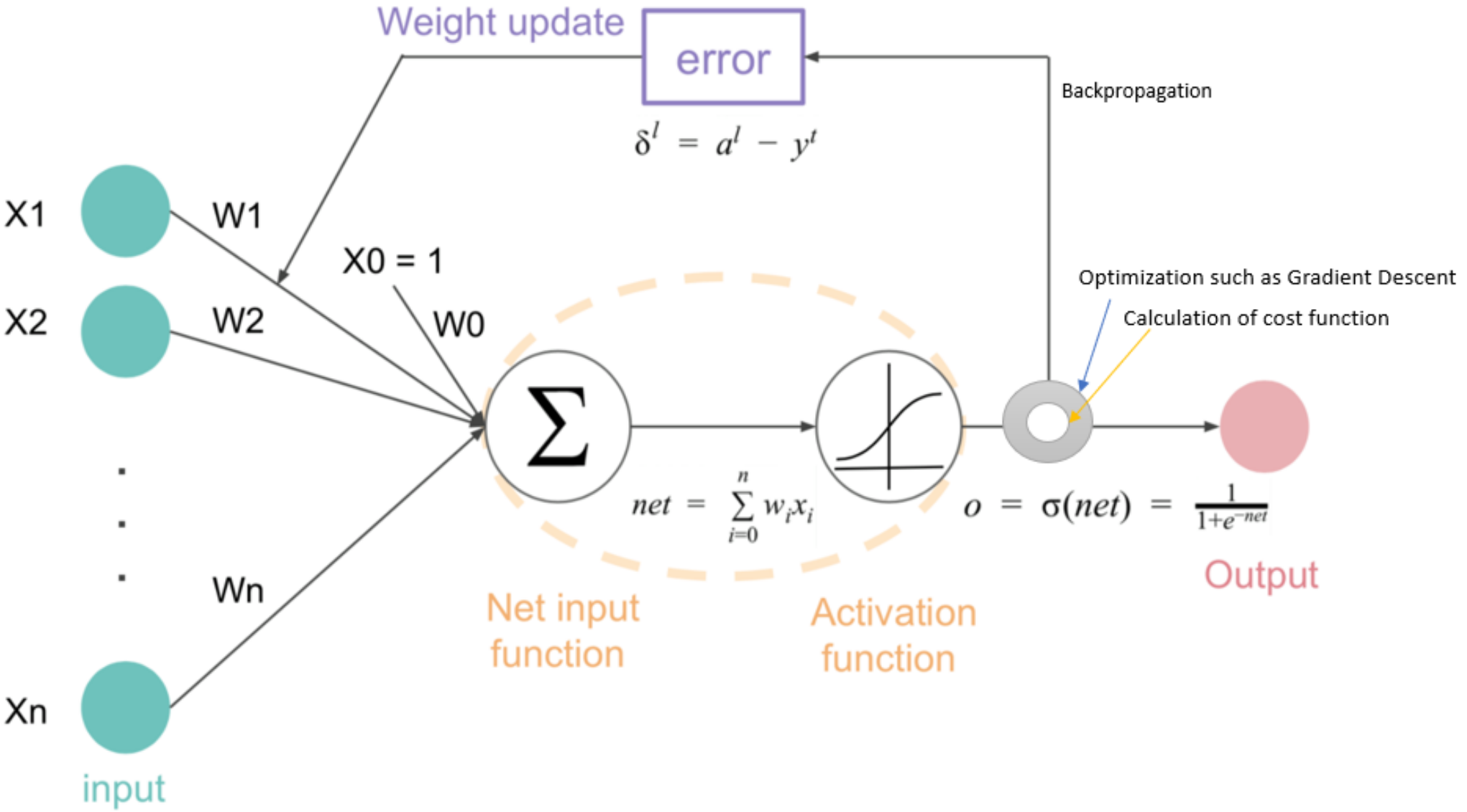
If the total number of layers is very large, this neural network is called a "**deep network**"

If we use this deep network for training and then perform prediction or classification, we call this process "**deep learning**".

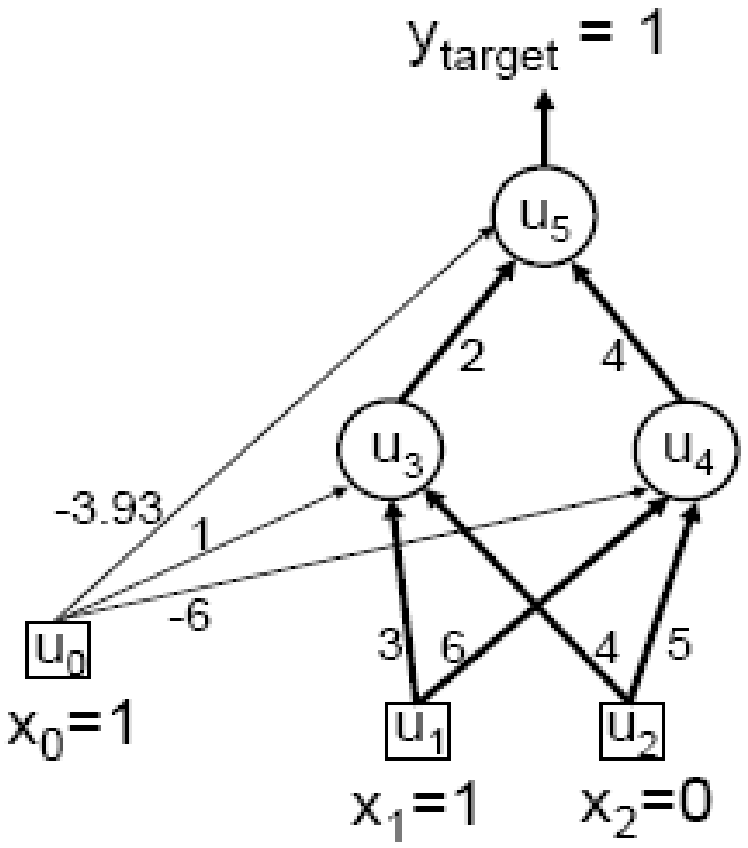
Backpropagation

- Back-propagation is just a way of propagating the total loss back into the neural network to know **how much of the loss every node is responsible for**, and
- Subsequently **updating the weights** in such a way that minimizes the loss by giving the nodes with higher error rates lower weights and vice versa.
- In simple terms, after each feed-forward passes through a network, this algorithm does the **backward pass to adjust the model's parameters based on weights and biases.**


Backpropagation of weights
Inputs
Output
Output Layer
Hidden Layer
Input Layer


Weight update
error
$\delta^l = a^l - y^t$
Backpropagation
X1
X2
Xn
W1
W2
Wn
X0 = 1
W0
Optimization such as Gradient Descent
Calculation of cost function
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1+e^{-net}}$
Net input function
Activation function
Output
input

MLP/BP: A worked example



Current state:

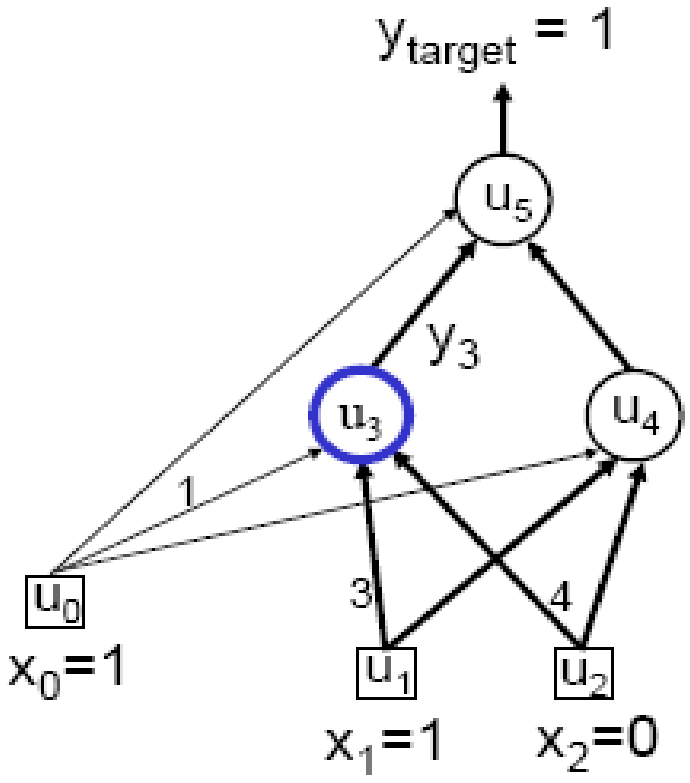
- Weights on arrows e.g. $w_{13} = 3$, $w_{35} = 2$, $w_{24} = 5$
- Bias weights, e.g.

bias for unit 4 (u_4) is $w_{04} = -6$

Training example (e.g. for logical OR problem):

- Input pattern is $x_1=1$, $x_2=0$
- Target output is $y_{target}=1$

Worked example: Forward Pass



Output for any neuron/unit j can be calculated from:

$$a_j = \sum_i w_{ij} x_i$$

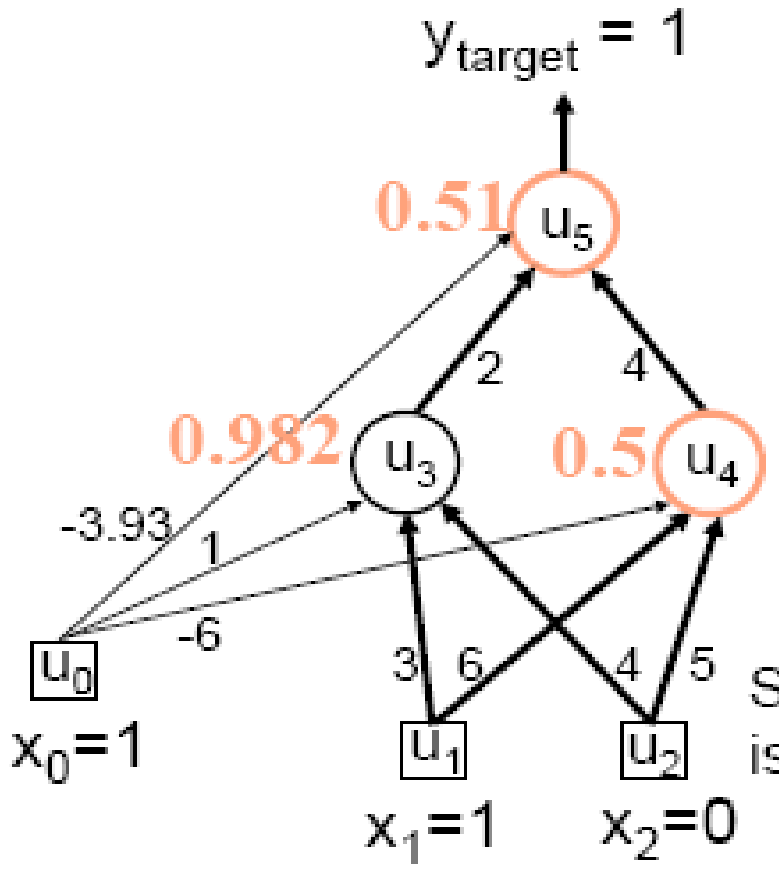
$$y_j = f(a_j) = \frac{1}{1+e^{-a_j}}$$

e.g Calculating output for Neuron/unit 3 in hidden layer:

$a_3 = 1*1 + 3*1 + 4*0 = 4$

$y_3 = f(4) = \frac{1}{1+e^{-4}} = 0.982$

Worked example: Forward Pass

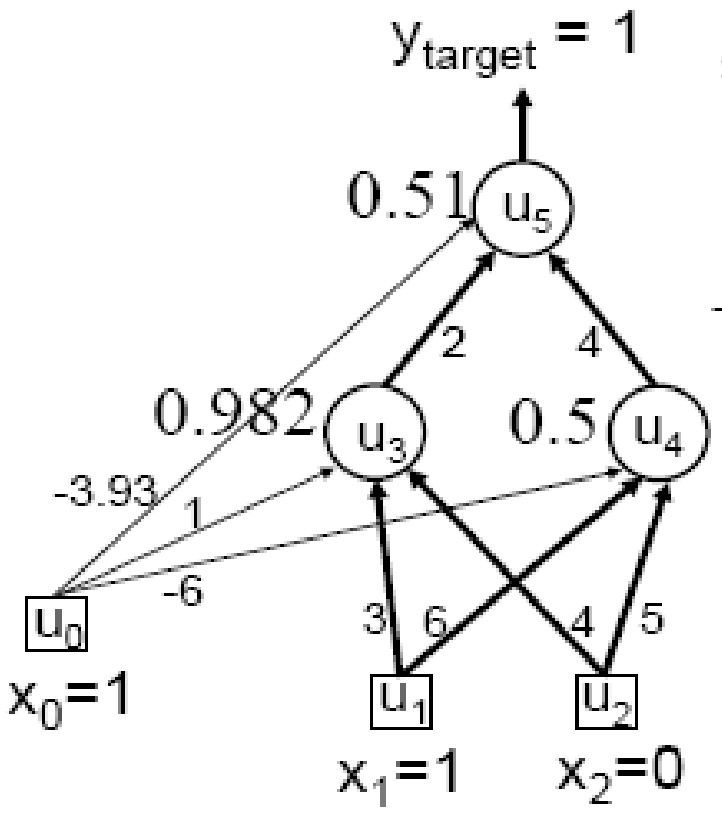


Unit	activation	output
	a_j	y_j
u_3	4.00	0.982
u_4	0.00	0.500
u_5	0.04	**0.510**

(network output)

So the error for this training example is: $(y_{target} - y_5) = (1 - 0.510) = 0.490$

Worked example: Backward Pass



Now compute delta values starting at the output:

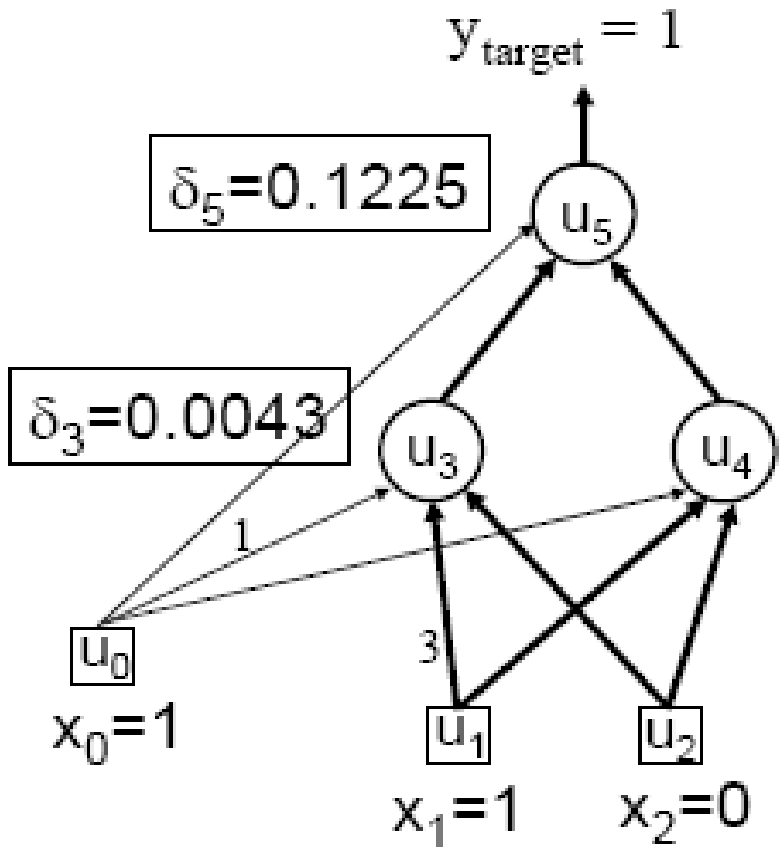
δ_5 = $y_5(1 - y_5)$ $(y_{target} - y_5)$
= 0.51(1 - 0.51) x 0.49
= **0.1225**

Then for hidden units:

δ_4 = $y_4(1 - y_4)$ w_{45} δ_5
= 0.5(1 - 0.5) x 4 x 0.1225
= **0.1225**

δ_3 = $y_3(1 - y_3)$ w_{35} δ_5
= 0.982(1-0.982) x 2 x 0.1225
= **0.0043**

Worked example: Update Weights Using Generalized Delta Rule (BP)



- Set learning rate η = 0.1

  Change weights by:

  $$\Delta w_{ij} = \eta \delta_j y_i$$

- e.g.bias weight on u_3:

$\Delta w_{03} = \eta \delta_3 x_0$

=0.1*0.0043*1

= 0.0004

So, new w_{03} ⌫

w_{03}(old)+Δw_{03}

=1+0.0004=1.0004

- and likewise:

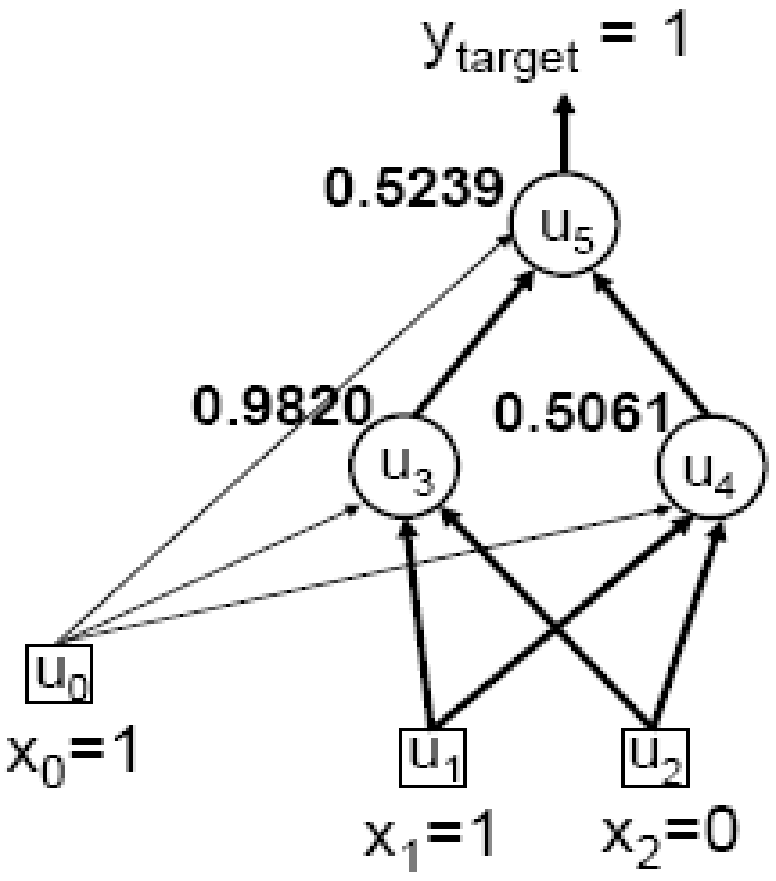
w_{13} ⌫ 3 + 0.0004

= 3.0004

Similarly for the all weights wij:

i	j	w_{ij}	δ_j	y_i	Updated w_{ij}
0	3	**1**	0.0043	1.0	**1.0004**
1	3	**3**	0.0043	1.0	**3.0004**
2	3	**4**	0.0043	0.0	**4.0000**
0	4	**-6**	0.1225	1.0	**-5.9878**
1	4	**6**	0.1225	1.0	**6.0123**
2	4	**5**	0.1225	0.0	**5.0000**
0	5	**-3.92**	0.1225	1.0	**-3.9078**
3	5	**2**	0.1225	0.9820	**2.0120**
4	5	**4**	0.1225	0.5	**4.0061**

Verification that it works



On next forward pass:

The new activations are:

$y_3 = f(4.0008) = 0.9820$

$y_4 = f(0.0245) = 0.5061$

$y_5 = f(0.0955) = \mathbf{0.5239}$

Thus the new error

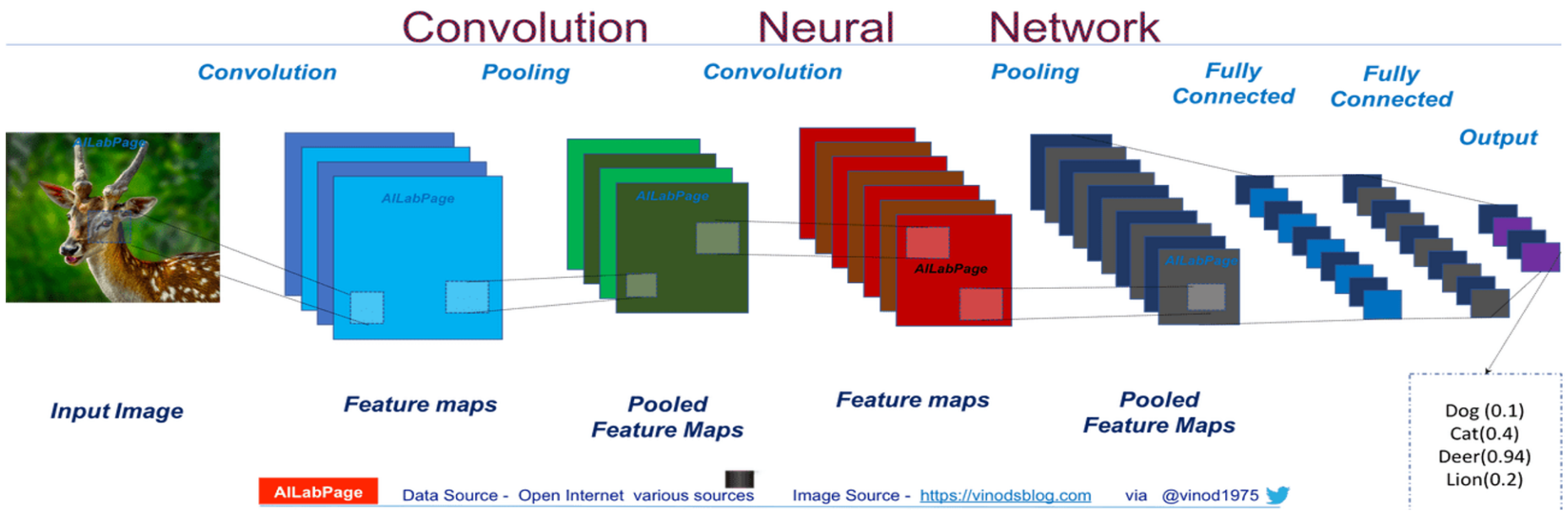
$(y_{target} - y_5)=(1 - 0.5239)=0.476$

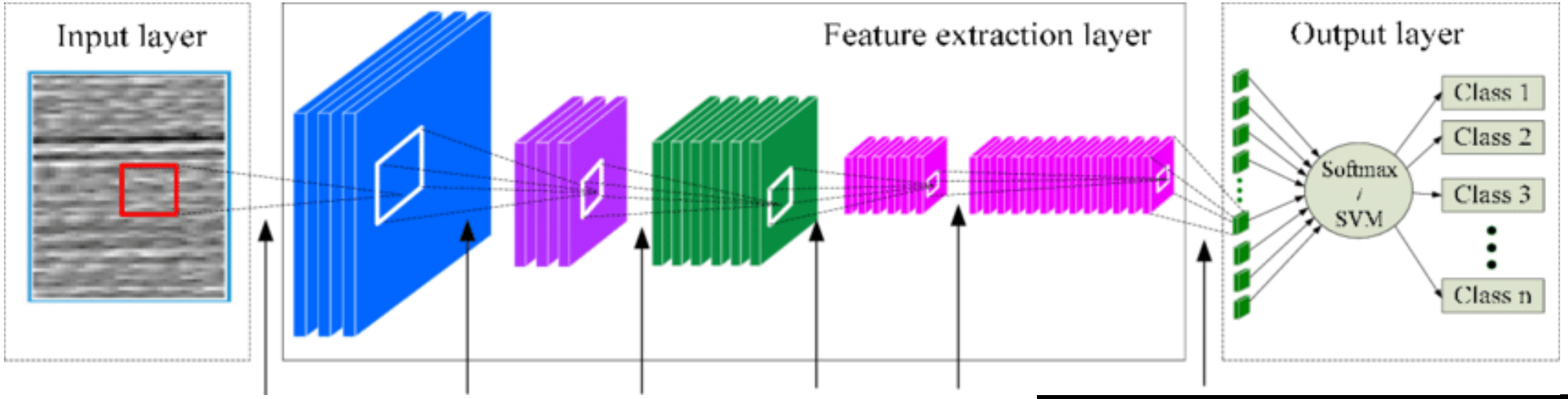
has been reduced by **0.014**

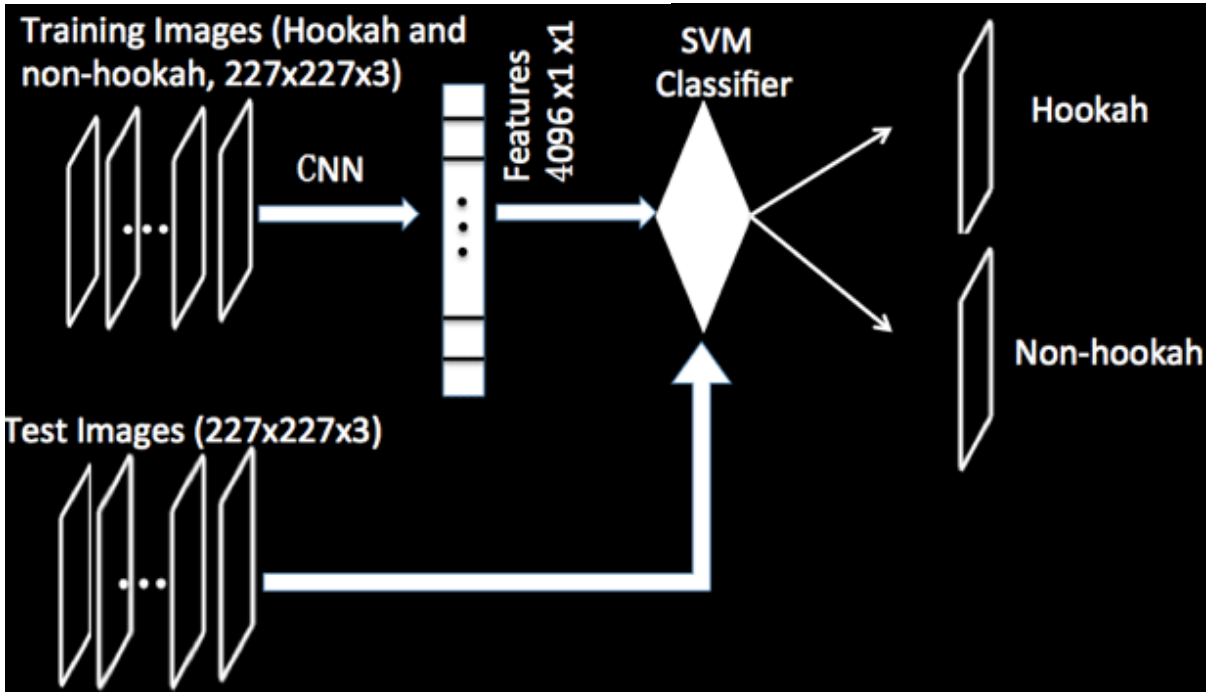
(from **0.490** to **0.476**)

Ref: "Neural Network Learning & Expert Systems" by Stephen Gallant

Convolutional NN (CNNs)



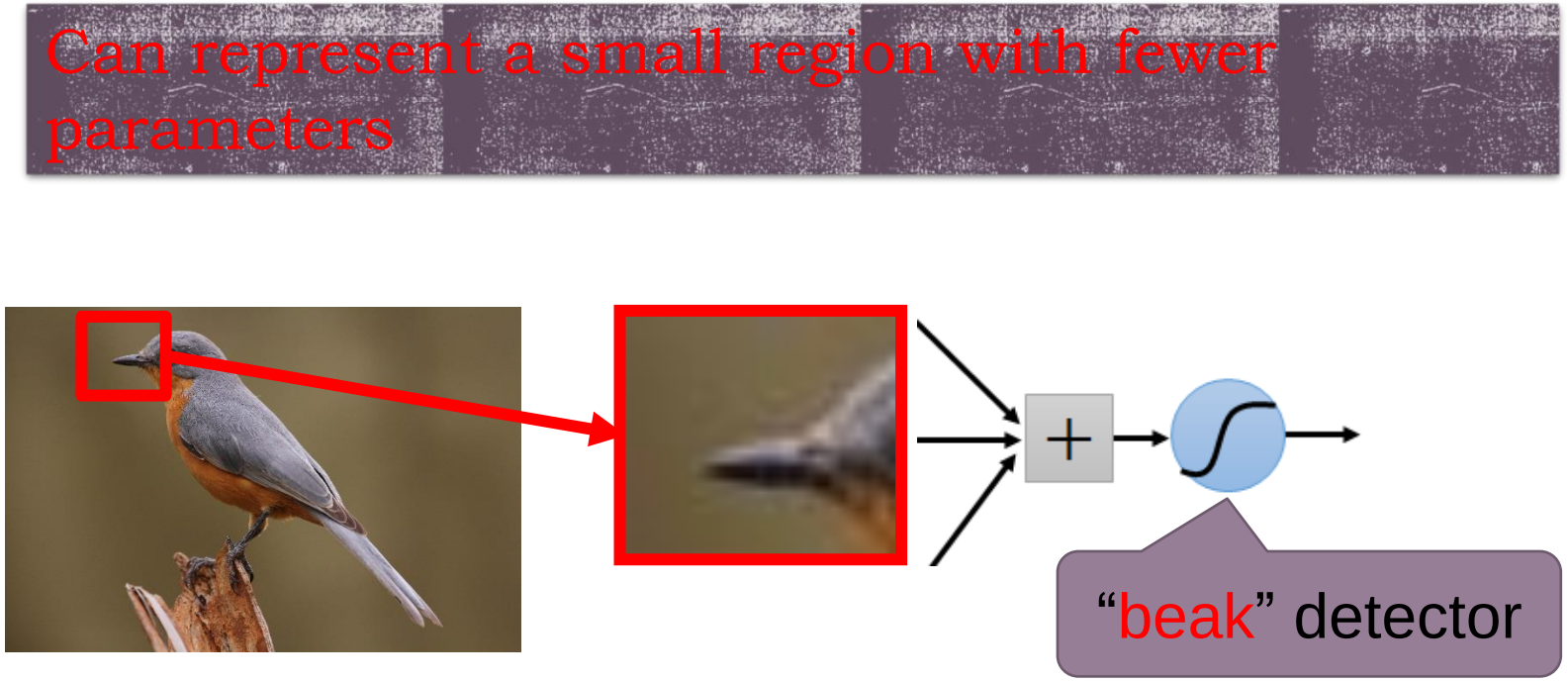




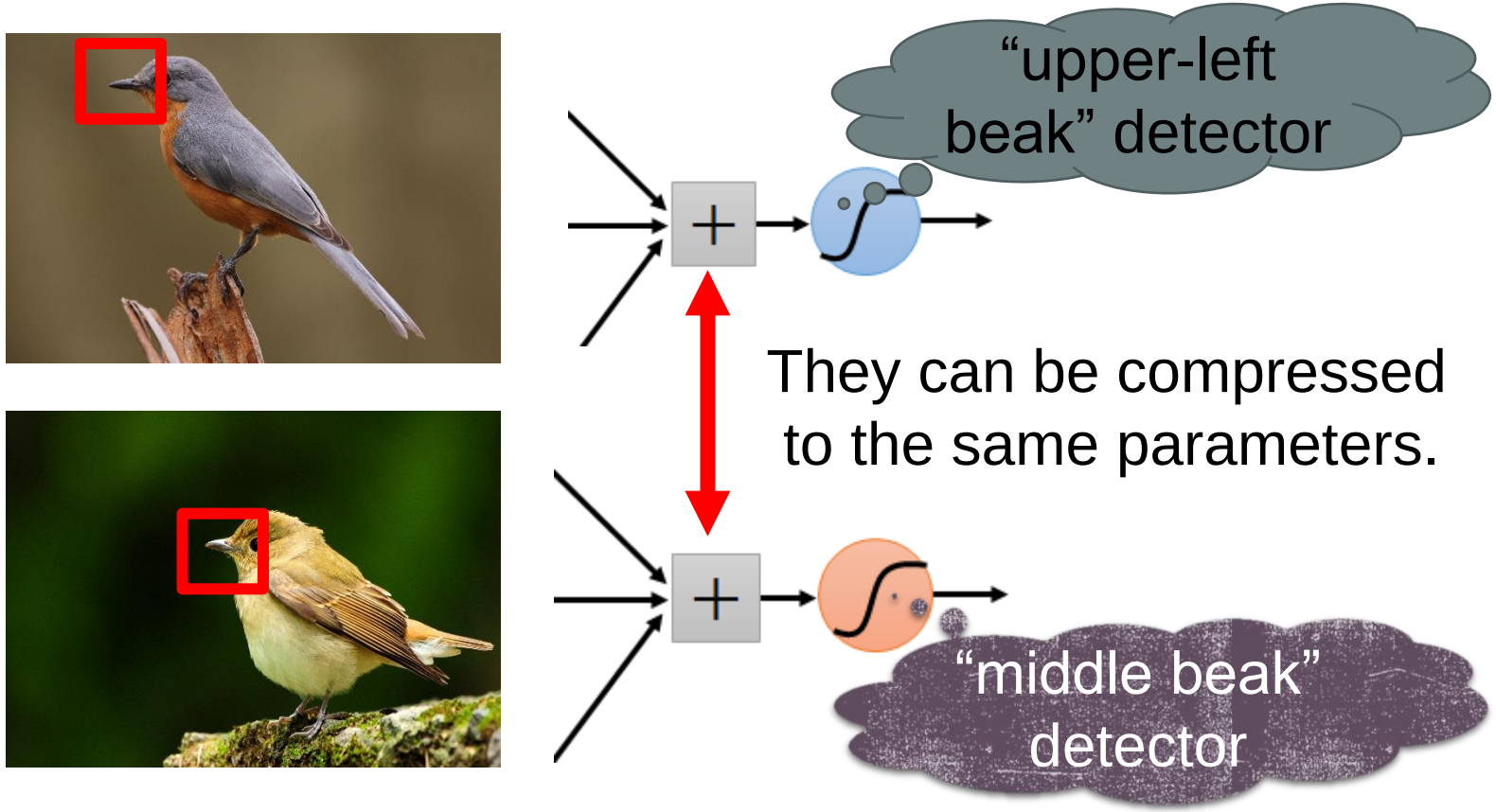
https://www.researchgate.net/figure/The-scheme-of-our-method-SVM-support-vector-machine-CNN-convolutional-neural-network_fig9_326891877
https://www.researchgate.net/figure/The-model-structure-of-the-improved-CNN-support-vector-machine-SVM-method_fig2_332321843

Consider learning an image:

- Some patterns are much smaller than the whole image

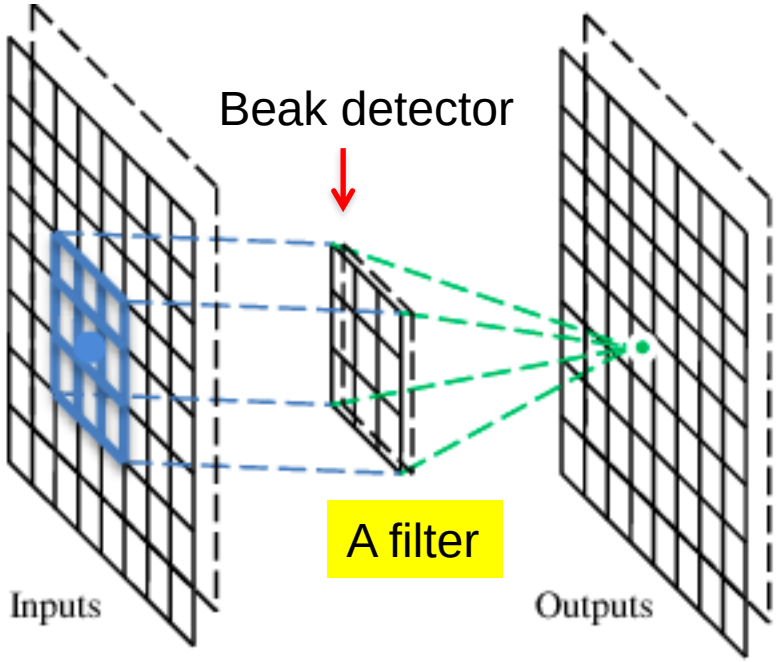


Same pattern appears in different places:
They can be compressed!
What about training a lot of such "small" detectors and each detector must "move around".



A convolutional layer

A CNN is a neural network with some convolutional layers (and some other layers). A convolutional layer has a number of filters that does convolutional operation.



Convolution

These 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

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

Filter 2

⋮ ⋮

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

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

Dot product → 3 -1

6 x 6 image

Convolution

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

Filter 1

If 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

6 x 6 image

3 -3

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

Convolution

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

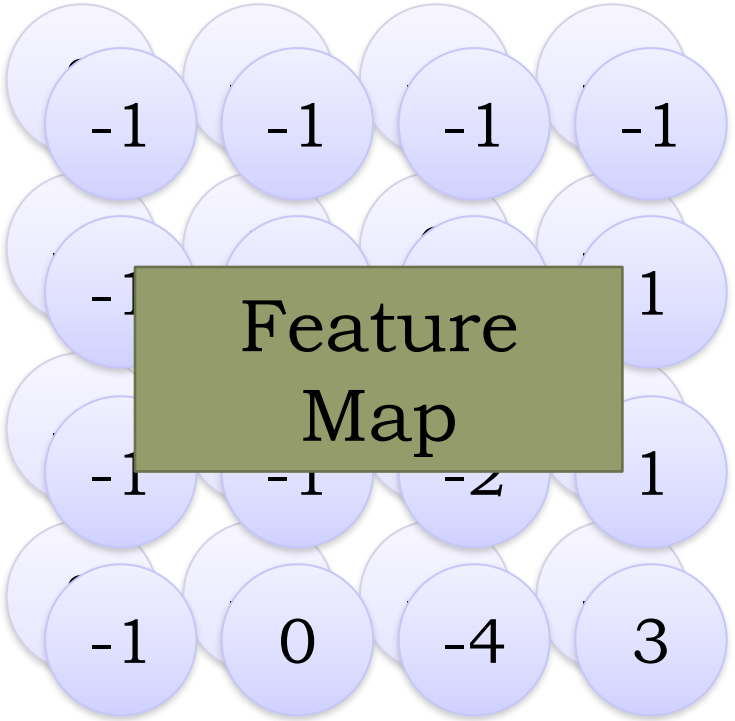
Filter 2

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

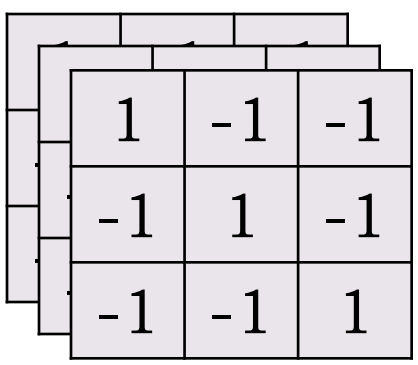
6 x 6 image

Repeat this for each filter

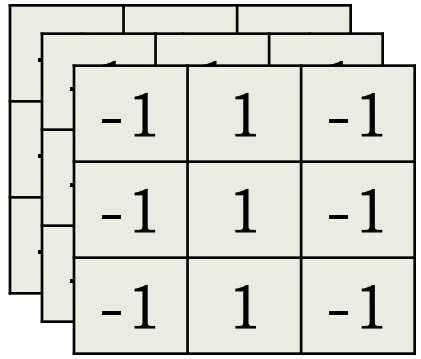


Two 4 x 4 images
Forming 2 x 4 x 4 matrix

Color image: RGB 3 channels

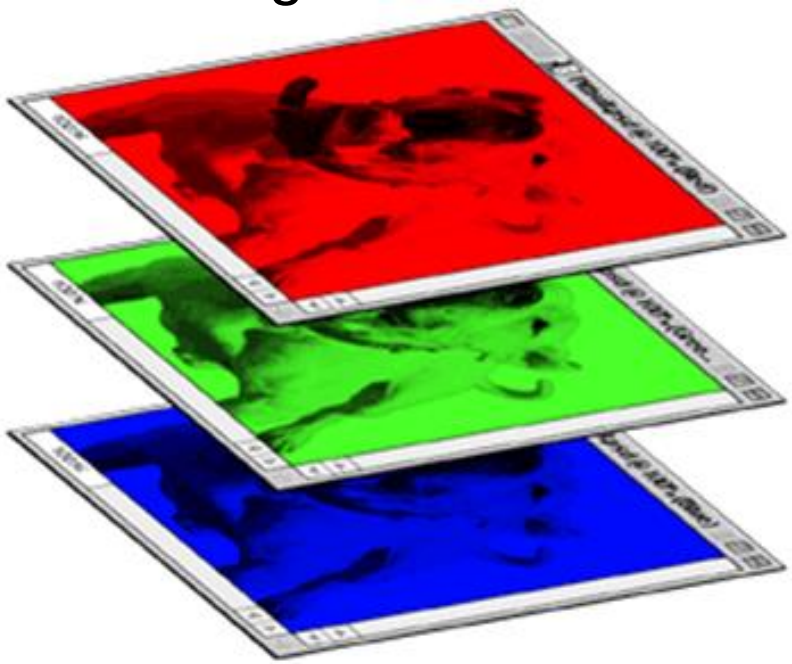


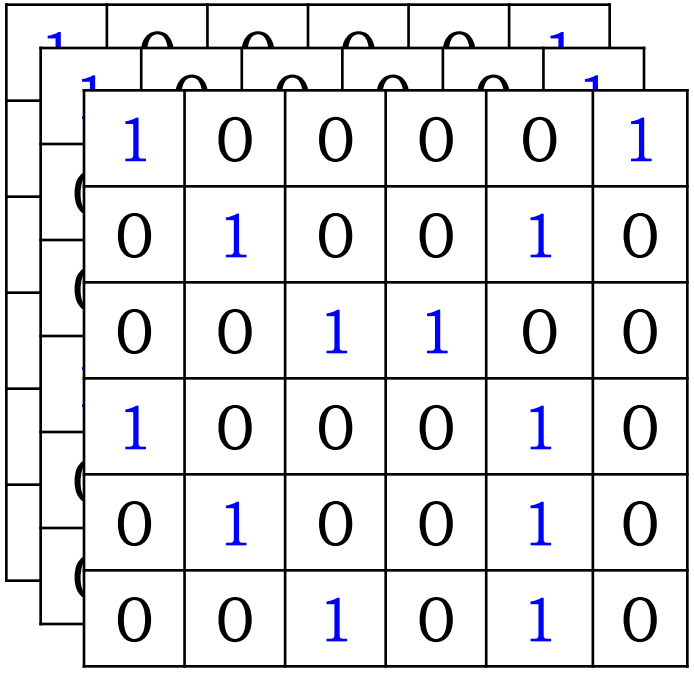
Filter 1



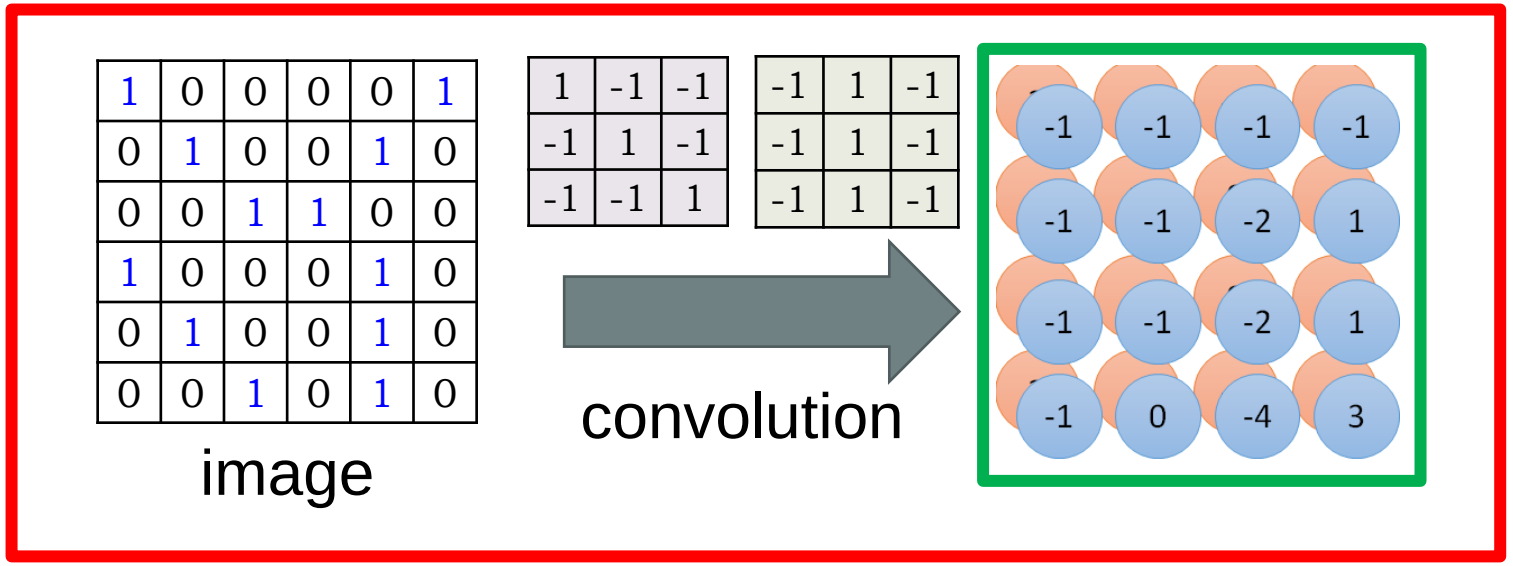
Filter 2

Color image

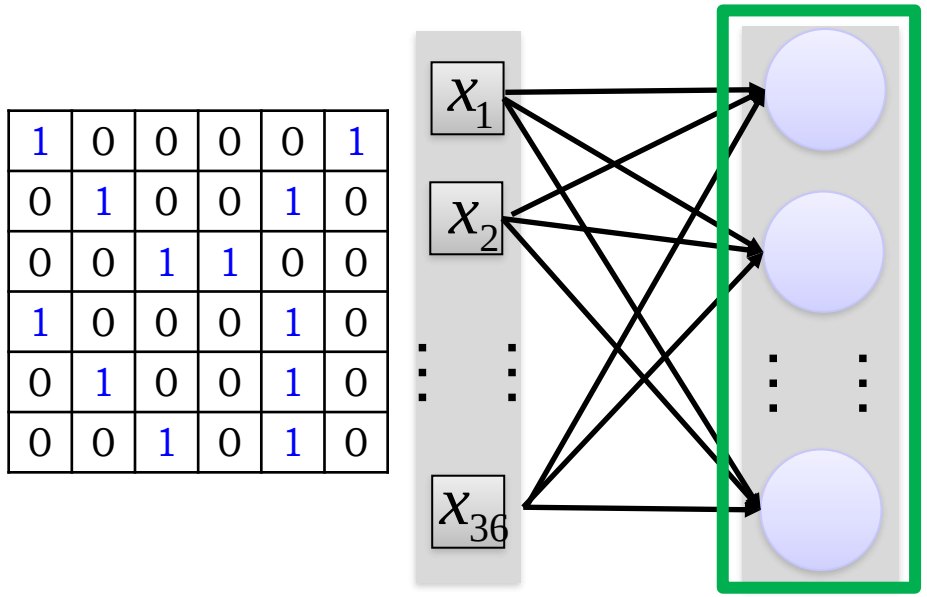


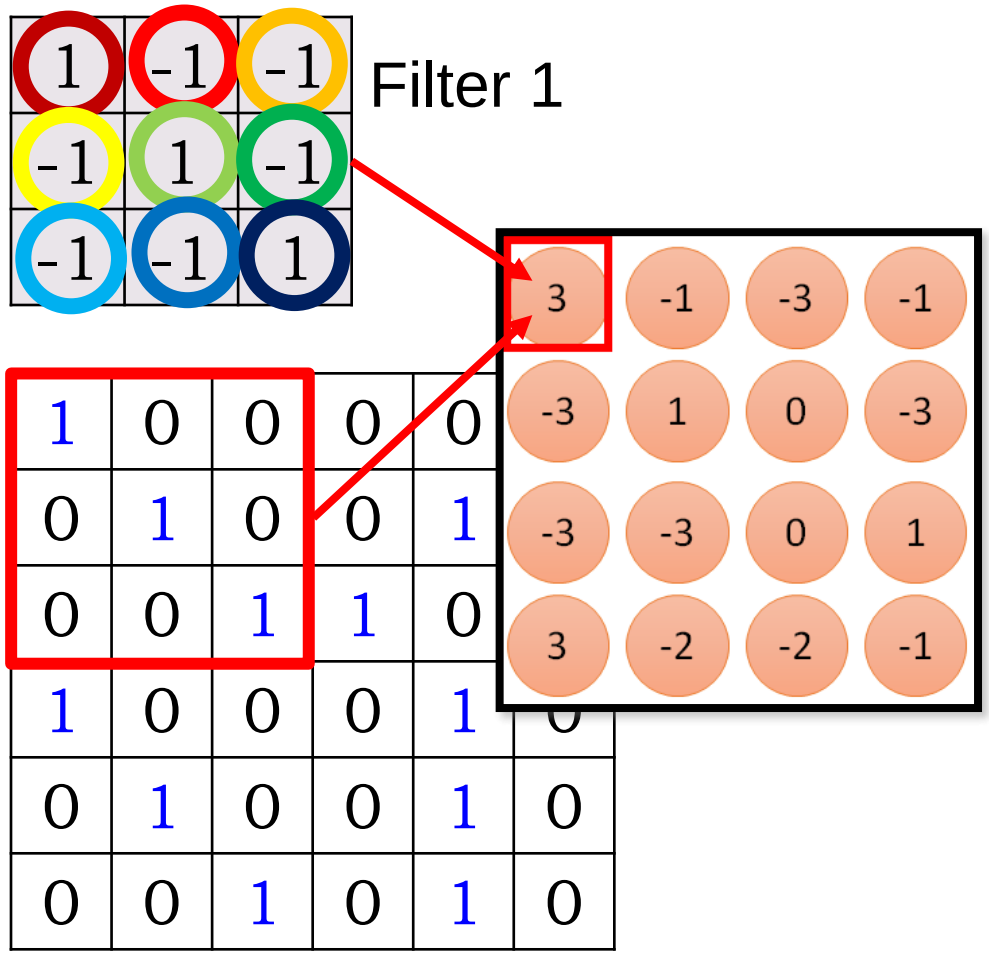


Convolution v.s. Fully Connected

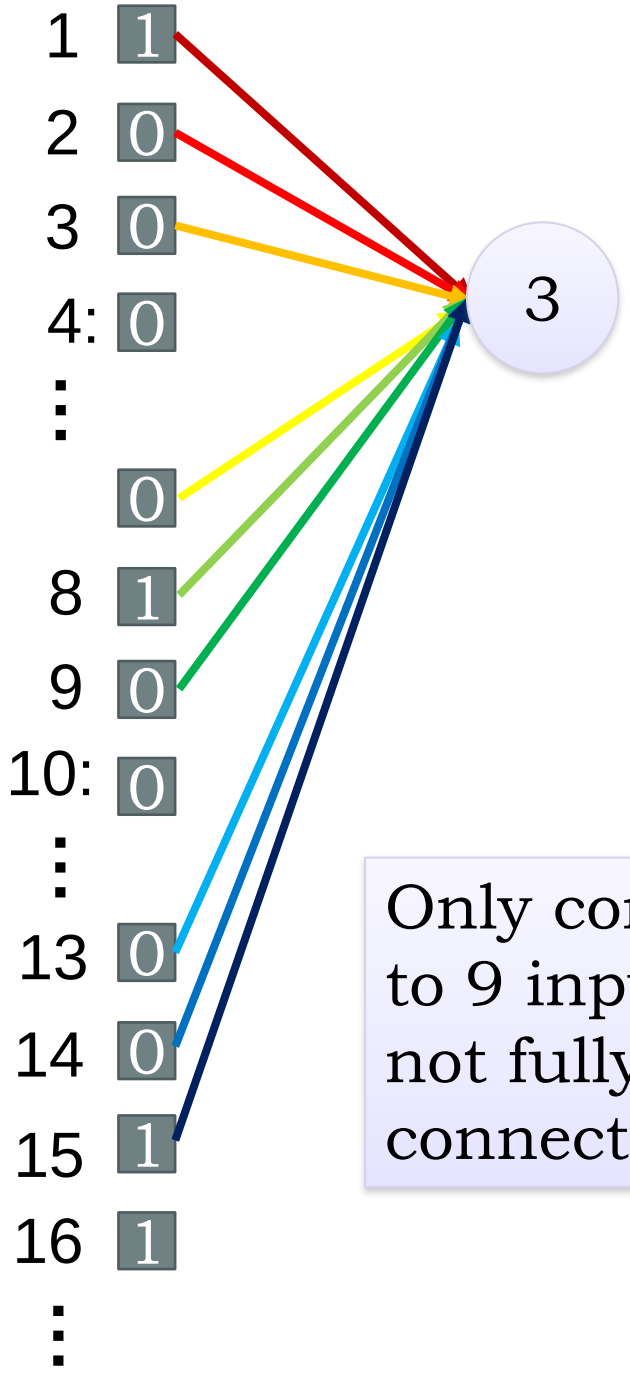


Fully-connected





fewer parameters!



Only connect to 9 inputs, not fully connected

The whole CNN

cat dog ……

Convolution

Max Pooling

Convolution

Max Pooling

Can repeat many times

Flattened

Fully Connected Feedforward network

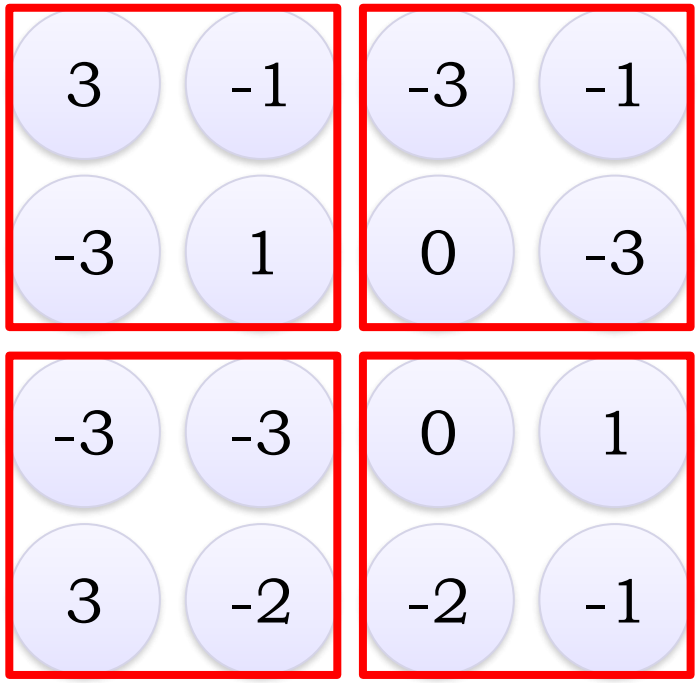
Max Pooling

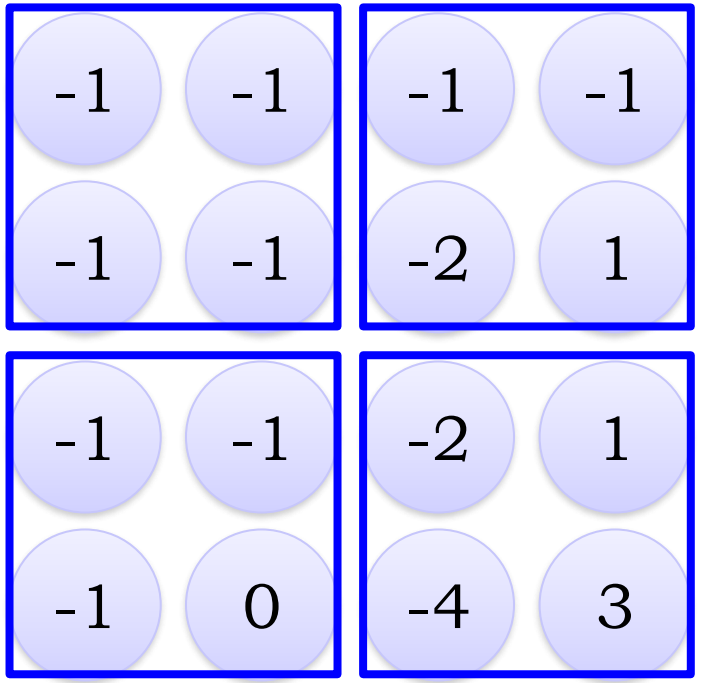
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

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

Filter 2





Why Pooling

- Subsampling pixels will not change the object

bird

bird

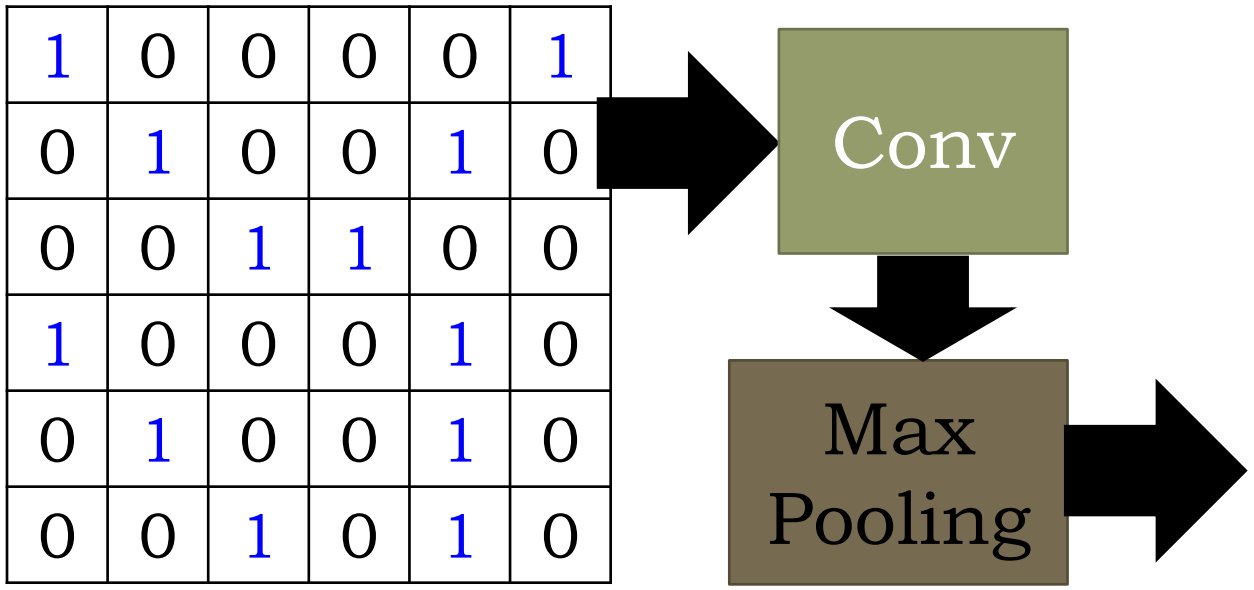
Subsampling

We can subsample the pixels to make image smaller → fewer parameters to characterize the image

A CNN compresses a fully connected network in two ways:

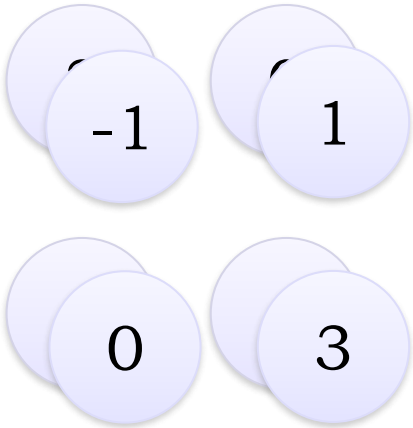
- Reducing number of connections
- Shared weights on the edges
- Max pooling further reduces the complexity

Max Pooling



6 x 6 image

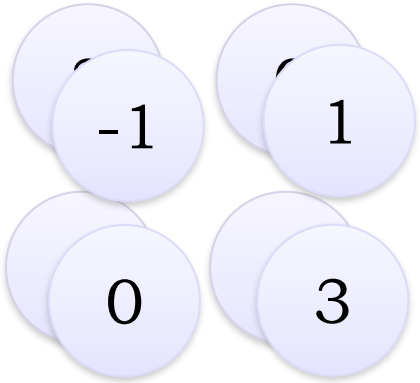
New image but smaller



2 x 2 image

Each filter is a channel

The whole CNN



A new image

Convolution

Max Pooling

Convolution

Max Pooling

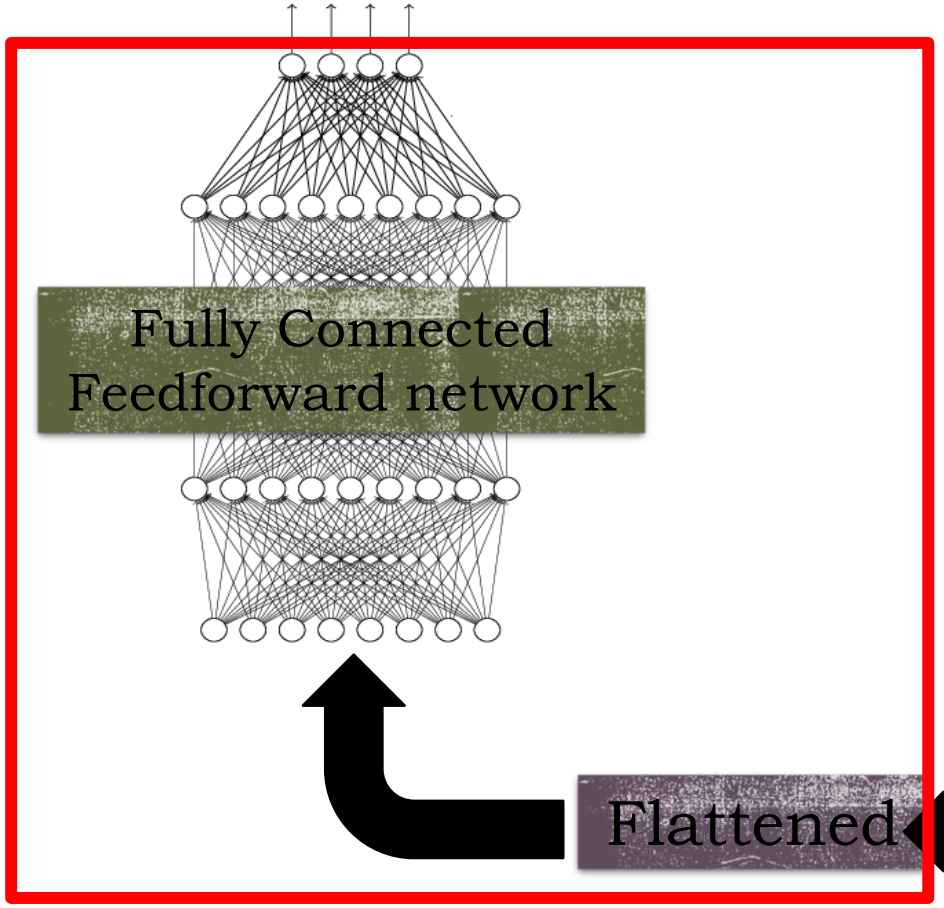
Can repeat many times

Smaller than the original image

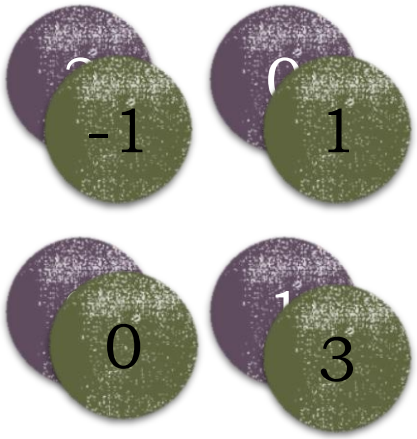
The number of channels is the number of filters

The whole CNN

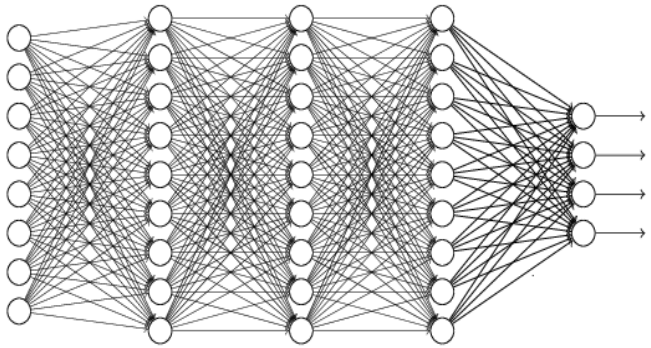
cat dog ……



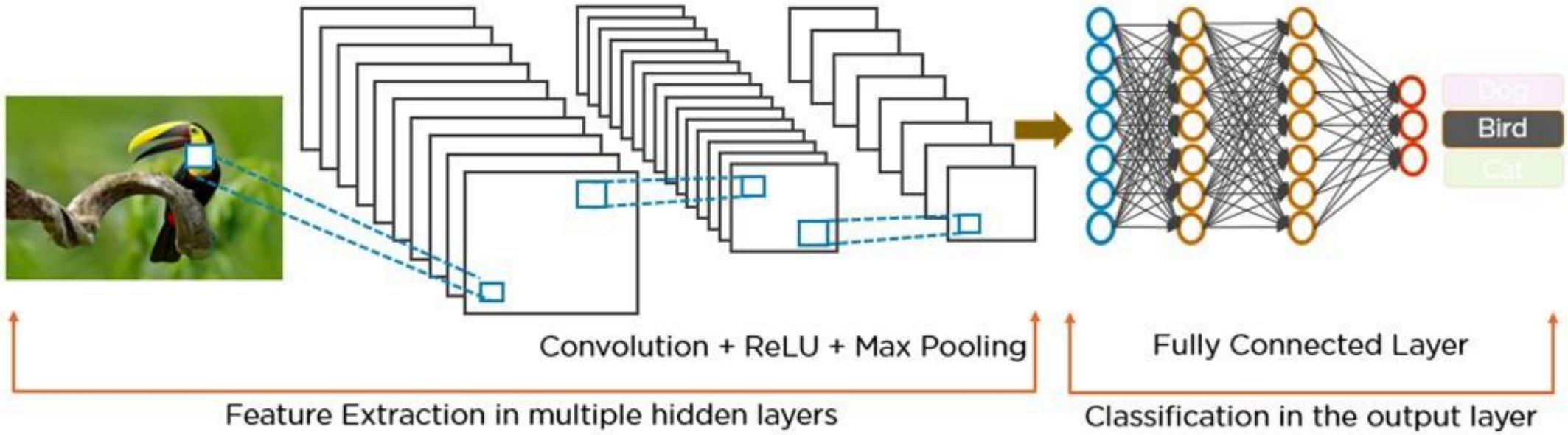
Flattening



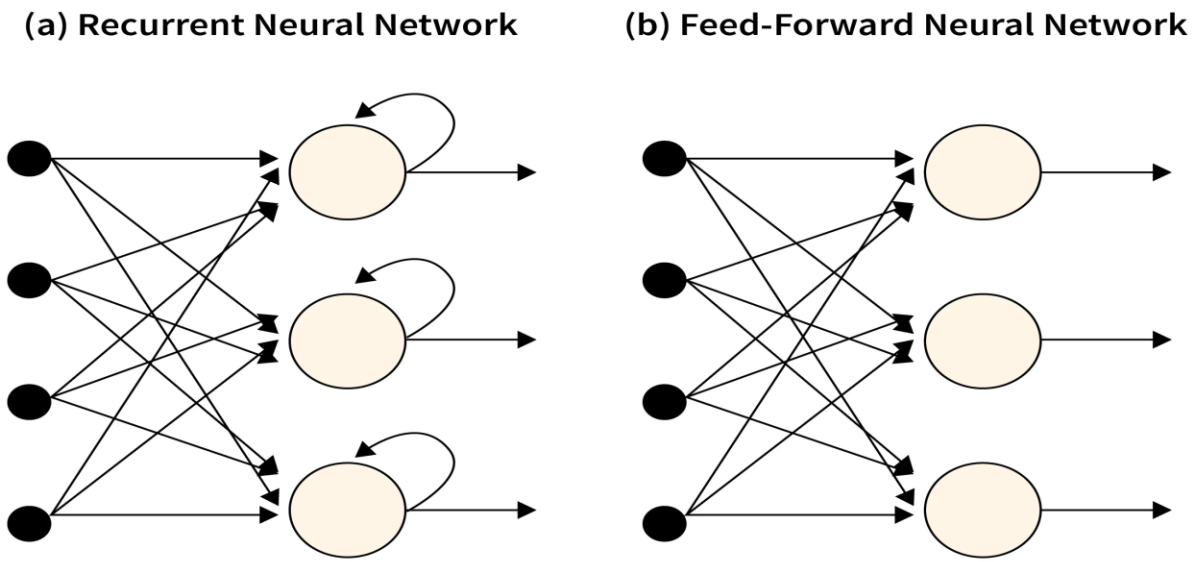
Flattened



Fully Connected Feedforward network


Dog
Bird
Cat
Convolution + ReLU + Max Pooling
Fully Connected Layer
Feature Extraction in multiple hidden layers
Classification in the output layer

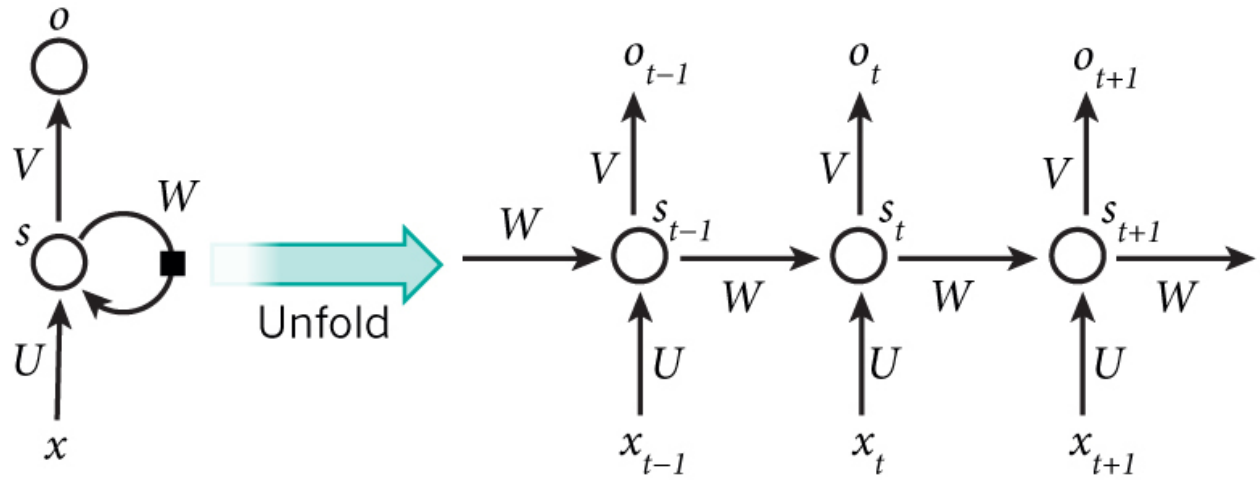
Recurrent NN

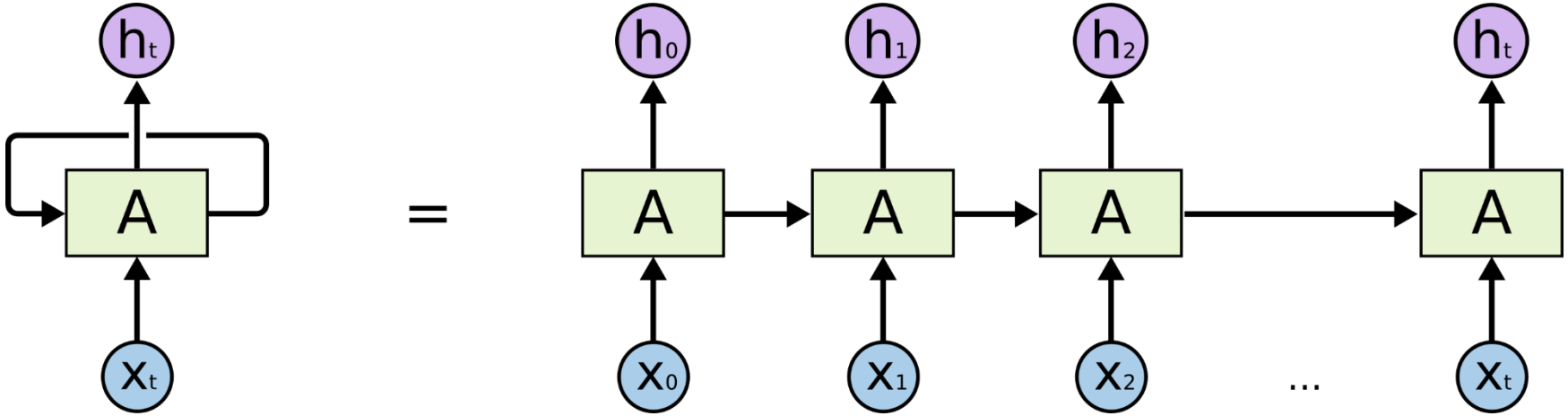


SCALER Topics

This is our fully connected network. If x_1 .... x_n, n is very large and growing, this network would become too large. We now will input one x_i at a time, and re-use the same edge weights.

Recurrent Neural Network

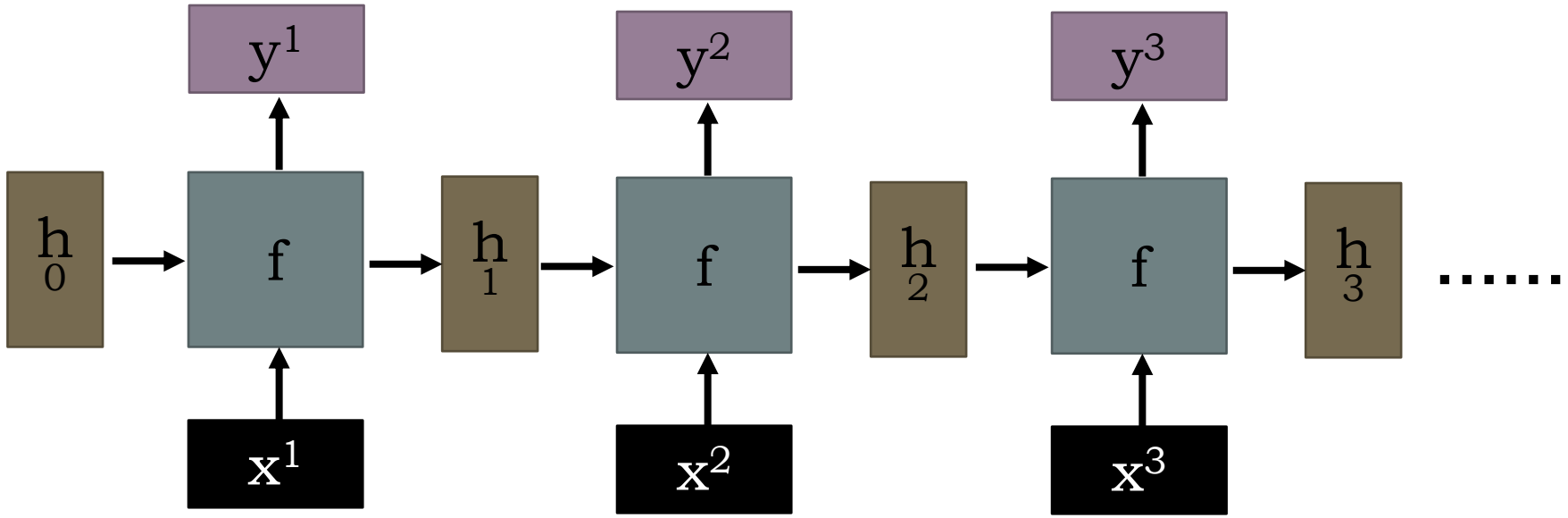




How does RNN reduce complexity?

- Given function f: h', y=f(h,x)

h and h' are vectors with the same dimension



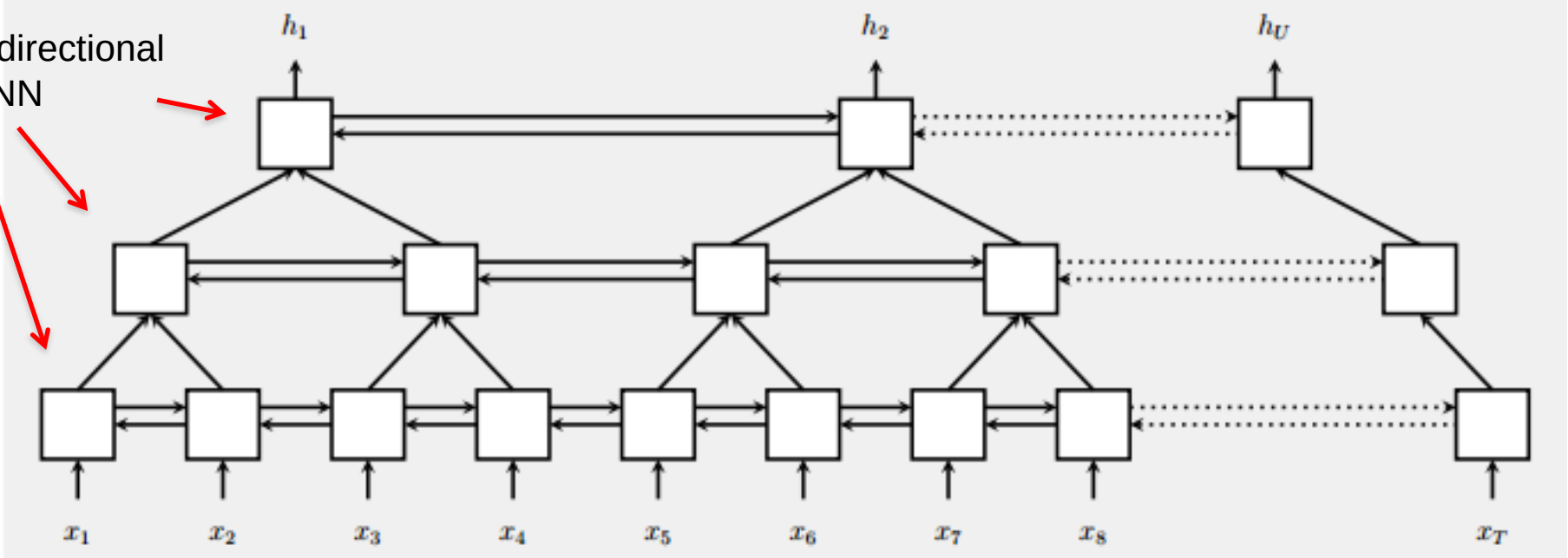
No matter how long the input/output sequence is, we only need one function f. If f's are different, then it becomes a feedforward NN. This may be treated as another compression from fully connected network.

Pyramid RNN

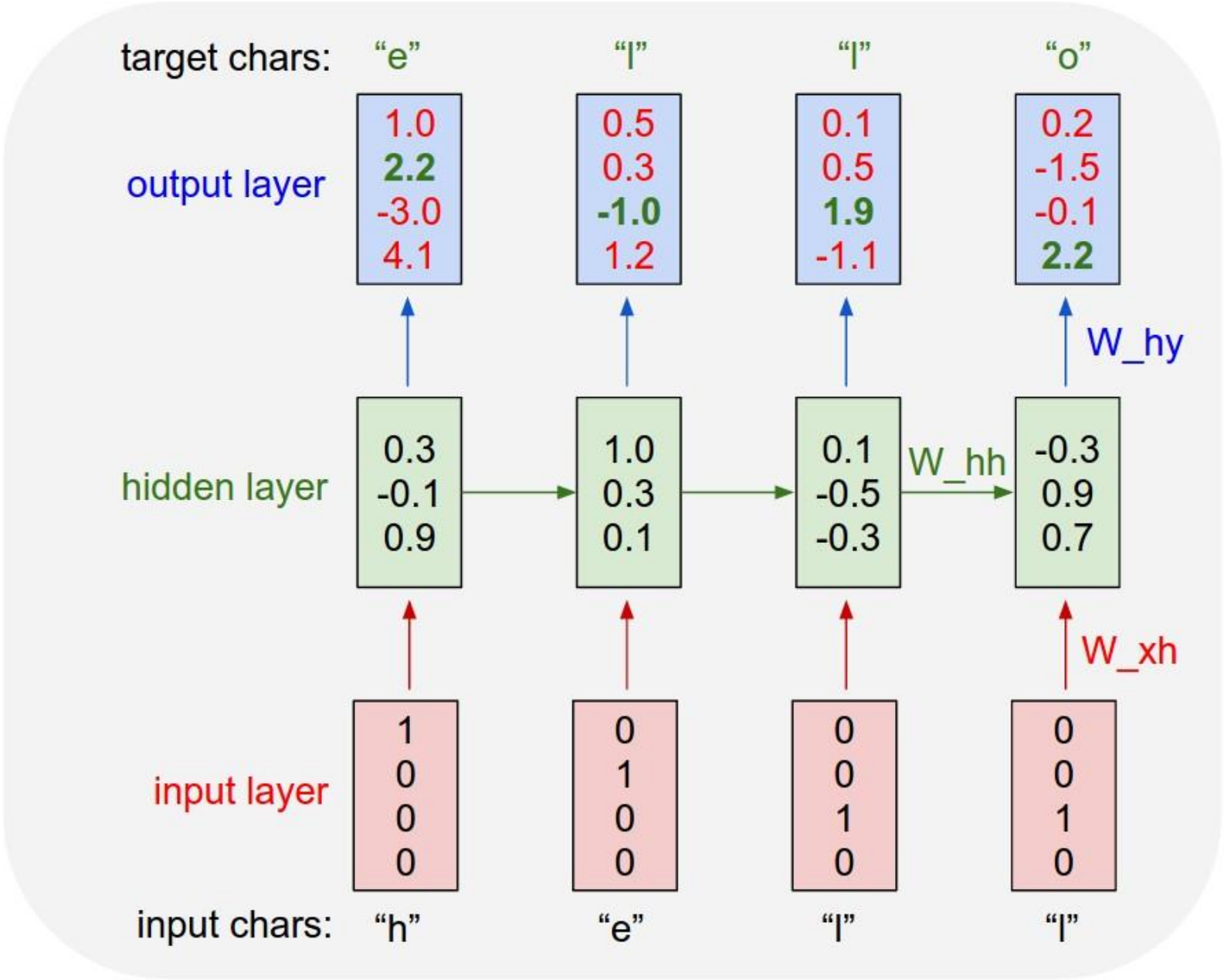
Significantly speed up training

- Reducing the number of time steps

Bidirectional RNN



W. Chan, N. Jaitly, Q. Le and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," ICASSP, 2016

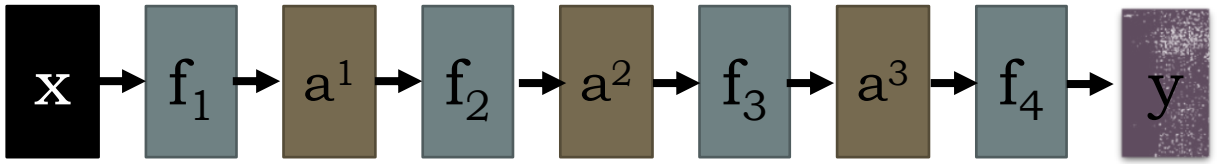

target chars:
“e”
“l”
“l”
“o”
output layer
1.0
2.2
-3.0
4.1
0.5
0.3
-1.0
1.2
0.1
0.5
1.9
-1.1
0.2
-1.5
-0.1
2.2
W_hy
hidden layer
0.3
-0.1
0.9
1.0
0.3
0.1
0.1
-0.5
-0.3
W_hh
-0.3
0.9
0.7
W_xh
input layer
1
0
0
0
0
1
0
0
0
0
1
0
0
0
1
0
input chars:
“h”
“e”
“l”
“l”

CNN vs. RNN: What are they and how do they differ?

	Convolutional neural network (CNN)	Recurrent neural network (RNN)
ARCHITECTURE	Feed-forward neural networks using filters and pooling	Recurring network that feeds the results back into the network
INPUT/OUTPUT	The size of the input and the resulting output are fixed (i.e., receives images of fixed size and outputs them to the appropriate category along with the confidence level of its prediction)	The size of the input and the resulting output may vary (i.e., receives different text and output translations—the resulting sentences can have more or fewer words)
IDEAL USAGE SCENARIO	Spatial data (such as images)	Temporal/sequential data (such as text or video)
USE CASES	Image recognition and classification, face detection, medical analysis, drug discovery and image analysis	Text translation, natural language processing, language translation, entity extraction, conversational intelligence, sentiment analysis, speech analysis

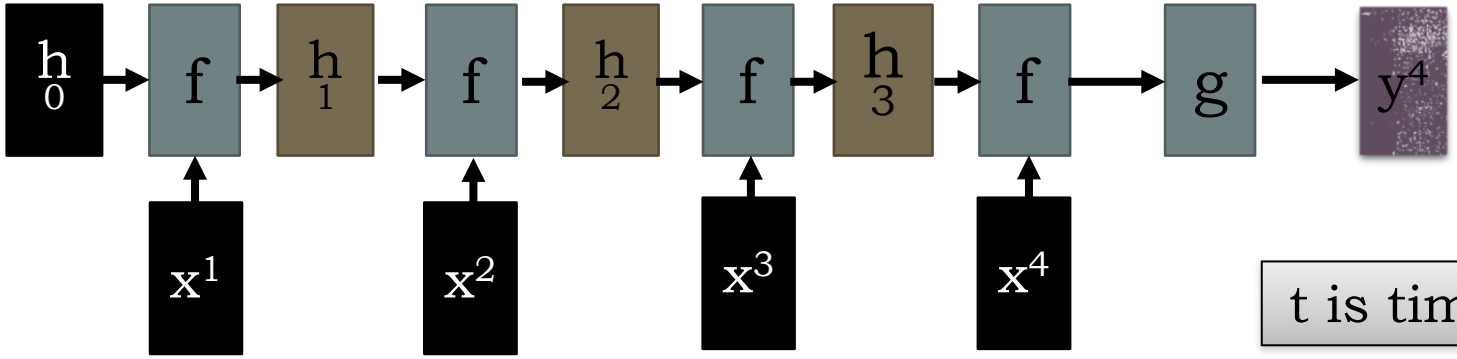
Feed-forward vs Recurrent Network

1. Feedforward network does not have input at each step
2. Feedforward network has different parameters for each layer



t is layer

$a^t = f_t(a^{t-1}) = \sigma(W^t a^{t-1} + b^t)$



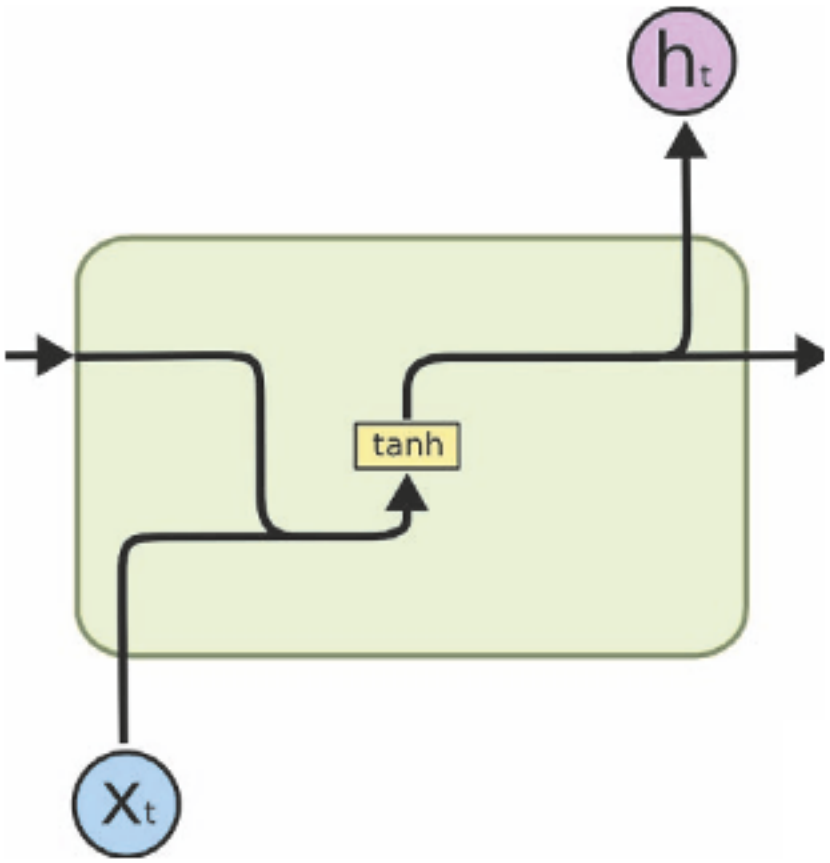
t is time step

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

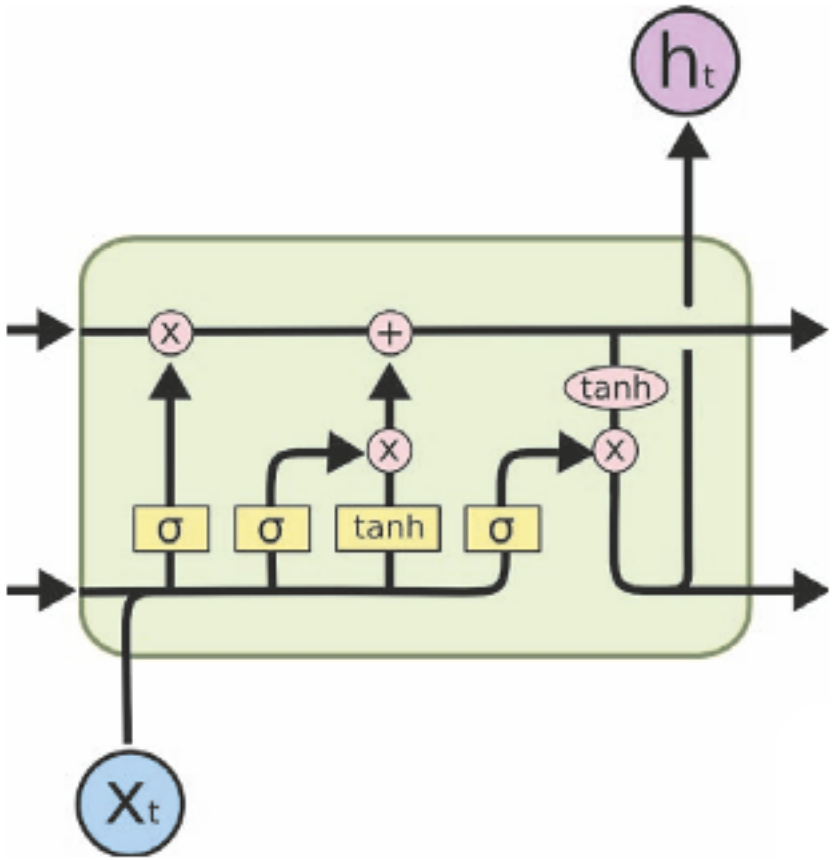
Long Short Term Memory (LSTM)

- Hochreiter & Schmidhuber (1997) solved the problem of getting an RNN to remember things for a long time (like hundreds of time steps).
- They designed a memory cell using logistic and linear units with multiplicative interactions.
- Information gets into the cell whenever its “write” gate is on.
- The information stays in the cell so long as its “keep” gate is on.
- Information can be read from the cell by turning on its “read” gate.

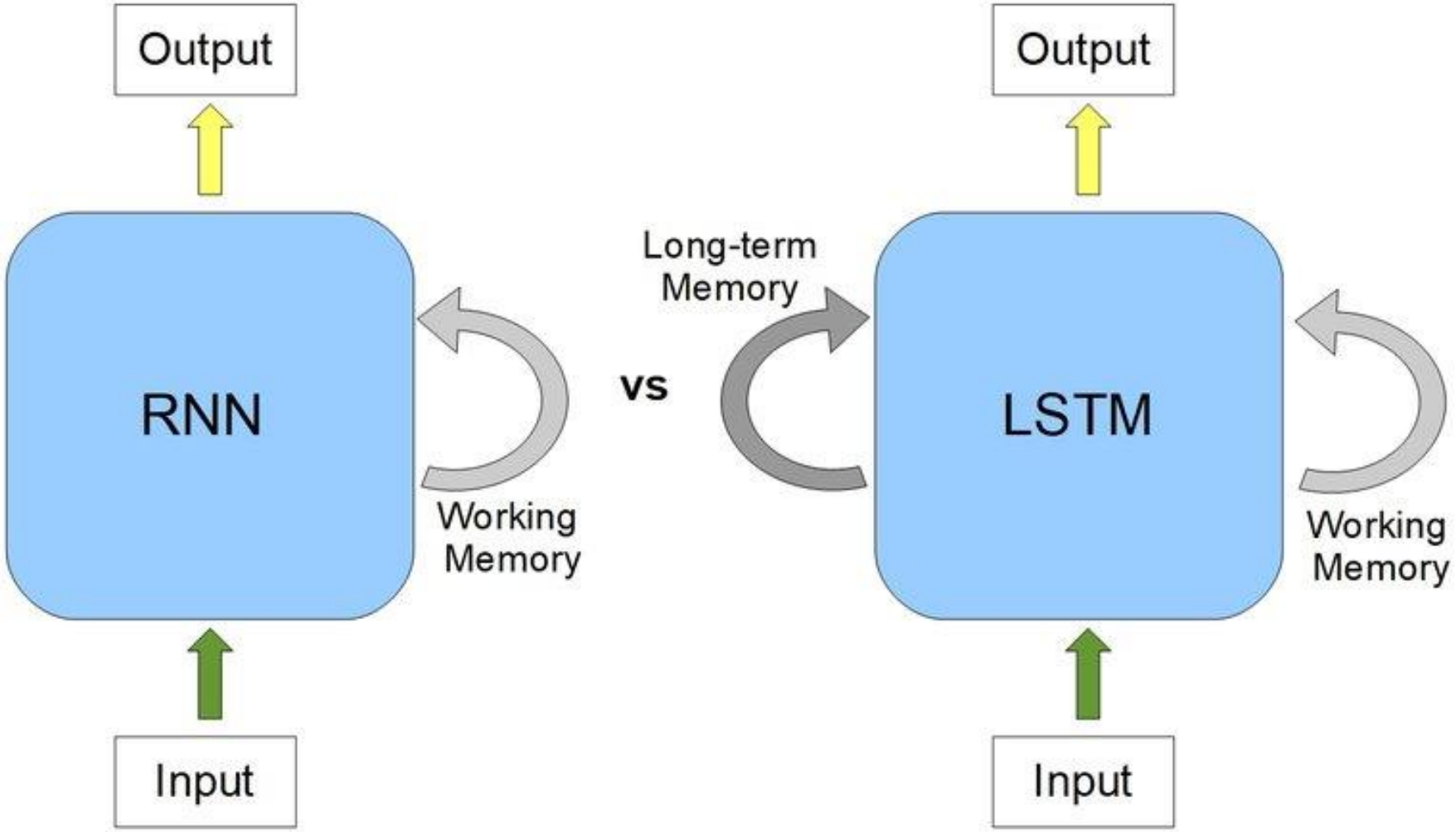
RNN vs LSTM

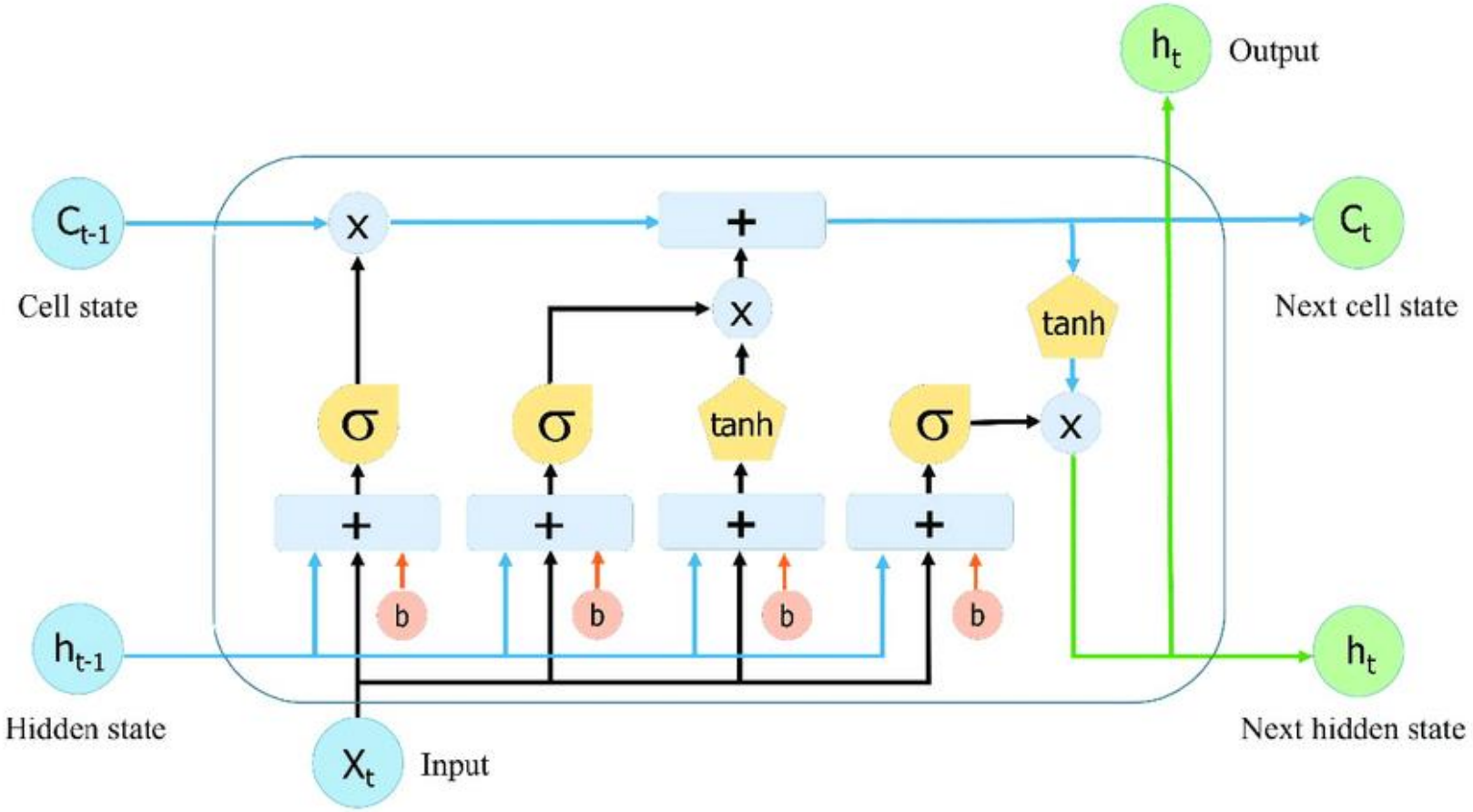


(a) RNN



(b) LSTM


Output
RNN
Working
Memory
Input
vs
Long-term
Memory
Output
LSTM
Working
Memory
Input



Inputs:

X_t Current input

C_{t-1} Memory from last LSTM unit

h_{t-1} Output of last LSTM unit

Outputs:

C_t New updated memory

h_t Current output

Nonlinearities:

σ Sigmoid layer

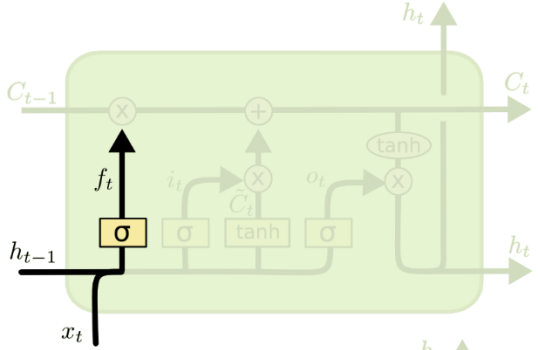
tanh Tanh layer

b Bias

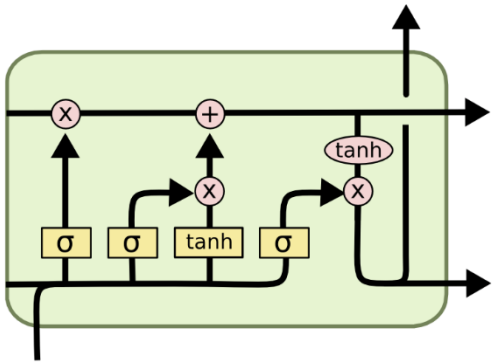
Vector operations:

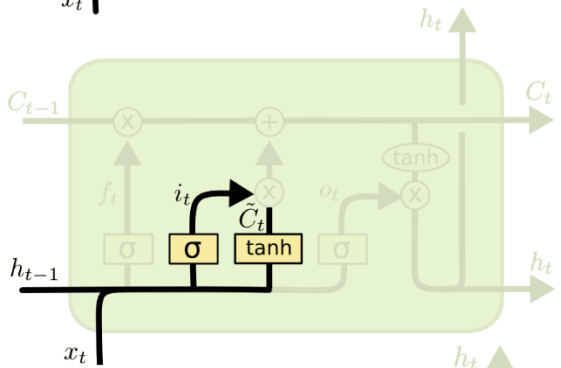
x Scaling of information

\+ Adding information



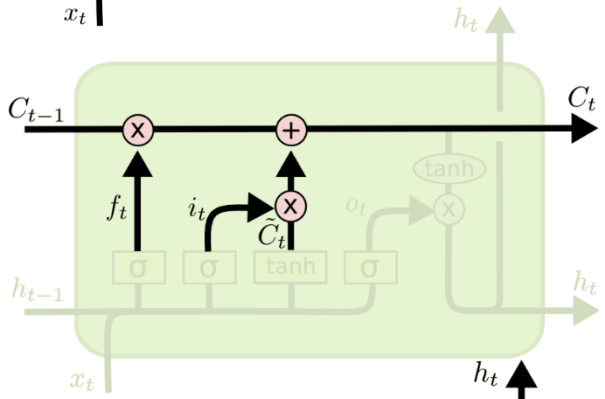
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$





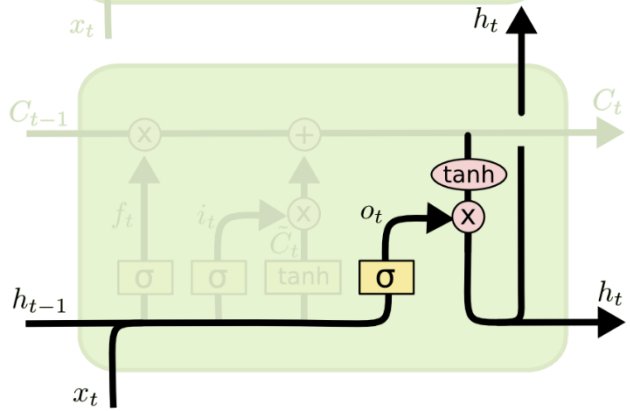
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

i_t decides what component is to be updated.
C'_t provides change contents



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

Updating the cell state



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

Decide what part of the cell state to output

References

- https://cs.uwaterloo.ca/~mli/Deep-Learning-2017-Lecture5CNN.ppt
- https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.ppt
- https://www.cs.utexas.edu/~swadhin/reading_group/slides/rnn.pptx
- Google (images) – deep learning algorithms, cnn, rnn, lstm