Goal Detection in Football by using SVM for Classification

Goal detection in football by using Support Vector Machines for classification[1]

N. Ancona, G. Cicirelli, A. Branca, A. Distante

Istituto Elaborazione Segnali ed Immagini - C.N.R.
Via Amendola 166/5 - 70126 Bari - Italy
e-mail: ancona@iesi.ba.cnr.it

Abstract

In this paper we present a technique for detecting goals during a football match by using images acquired by a single camera placed externally to the field. The method does not require the modification neither of the ball nor of the goalmouth. Due to the attitude of the camera with respect to football ground, the system can be thought of as an electronic linesman which helps the referee in establishing the occurrence of a goal during a football match. The occurrence of the event is established detecting the ball and comparing its position with respect to the location of the goalpost in image. The ball detection technique relies on a supervised learning scheme called Support Vector Machines for classification. The examples used for training are appropriately filtered version of views of the object to be detected, previously stored in form of image patterns. We have extensively tested the technique on real images in which the ball is both fully visible and partially occluded. The performance of the proposed detection scheme are measured in terms of detection rate, false positive rate and precision in the ball localization in image.

1 Introduction

In this paper we focus on the problem of detecting the occurrence of a goal during a football match, by using methods and devices which does not require the modification neither of the ball nor of the goalmouth. Automatic goal detection in football is an open problem which is getting particular attention from referee associations, sport press and supporters. In fact, there are not rare situations in which a goal occurres[1], but the referee and his collaboratores (linesmen) are not able to detecting the goal and, more important, do not award any point to neither teams correctly. Such situations occur for example when, after a shooting, the ball touches the internal side of the crossbar, bounces off the field having crossed completely the goal line and goes back, without touching the net. One of the most significative evidences of this event, named *ghost goal*, occurred during the World Cup final match on 1966 between England and West Germany. In that case, an english player struck a shot towards the German goal and the ball cannoned down from the crossbar, hit the ground and bounced back out into the play. English players claimed a goal, that the ball had passed completely over the line, but the referee, after consultation with his linesman, did not awarded the point to the England.

Optical sensors, like standard TV cameras, seem to be appropriate for approaching the problem at hand for several reasons. First of all they satisfy the main constraint of the problem, because their exploitation does not require modifications of neither the ball nor the goalmouth. They can be placed externally and also very far from the field and, if equipped with appropriate zoom lens, they provide images of the goalmouth area usefull for solving the goal detection problem. Moreover, they permit to have a direct evidence of the occurrence of a goal because the perceived images can be recorded on an analog or digital support for a successive analysis, for example by an external referee.

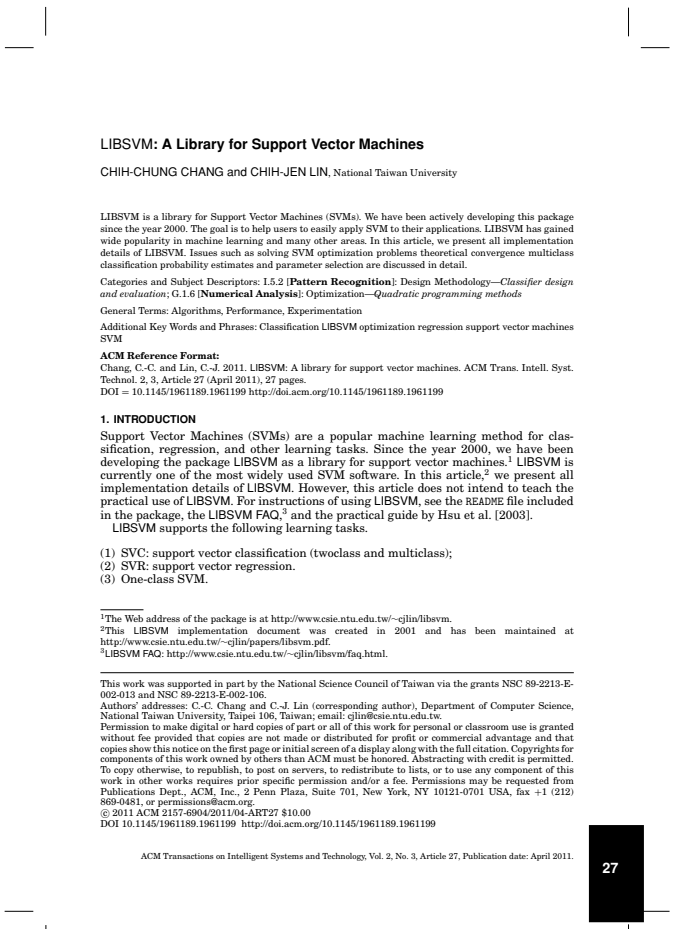
Reid and Zisserman [1] proposed an uncalibrated binocular vision system for solving the problem of goal detection in football. Their method exploits two images of the field acquired simultaneously from two different viewpoints. In both images both the goal area and the

[1] **Acknowledgements**: this paper describes research done at the Istituto Elaborazioni Segnali ed Immagini, C.N.R. in Bari. Partial support was also provided by the italian football association Federazione Italiana Gioco Calcio FIGC.

[1] A goal occurres in football when the ball completely crosses the goal line.

0-7803-7044-9/01/$10.00 ©2001 IEEE

LIBSVM: A Library for SVM



LIBSVM: **A Library for Support Vector Machines**

CHIH-CHUNG CHANG and CHIH-JEN LIN, National Taiwan University

LIBSVM is a library for Support Vector Machines (SVMs). We have been actively developing this package since the year 2000. The goal is to help users to easily apply SVM to their applications. LIBSVM has gained wide popularity in machine learning and many other areas. In this article, we present all implementation details of LIBSVM. Issues such as solving SVM optimization problems theoretical convergence multiclass classification probability estimates and parameter selection are discussed in detail.

Categories and Subject Descriptors: I.5.2 [**Pattern Recognition**]: Design Methodology—*Classifier design and evaluation*; G.1.6 [**Numerical Analysis**]: Optimization—*Quadratic programming methods*

General Terms: Algorithms, Performance, Experimentation

Additional Key Words and Phrases: Classification LIBSVM optimization regression support vector machines SVM

ACM Reference Format:
Chang, C.-C. and Lin, C.-J. 2011. LIBSVM: A library for support vector machines. ACM Trans. Intell. Syst. Technol. 2, 3, Article 27 (April 2011), 27 pages.
DOI = 10.1145/1961189.1961199 http://doi.acm.org/10.1145/1961189.1961199

1. INTRODUCTION

Support Vector Machines (SVMs) are a popular machine learning method for classification, regression, and other learning tasks. Since the year 2000, we have been developing the package LIBSVM as a library for support vector machines.[1] LIBSVM is currently one of the most widely used SVM software. In this article,[2] we present all implementation details of LIBSVM. However, this article does not intend to teach the practical use of LIBSVM. For instructions of using LIBSVM, see the `README` file included in the package, the LIBSVM FAQ,[3] and the practical guide by Hsu et al. [2003].

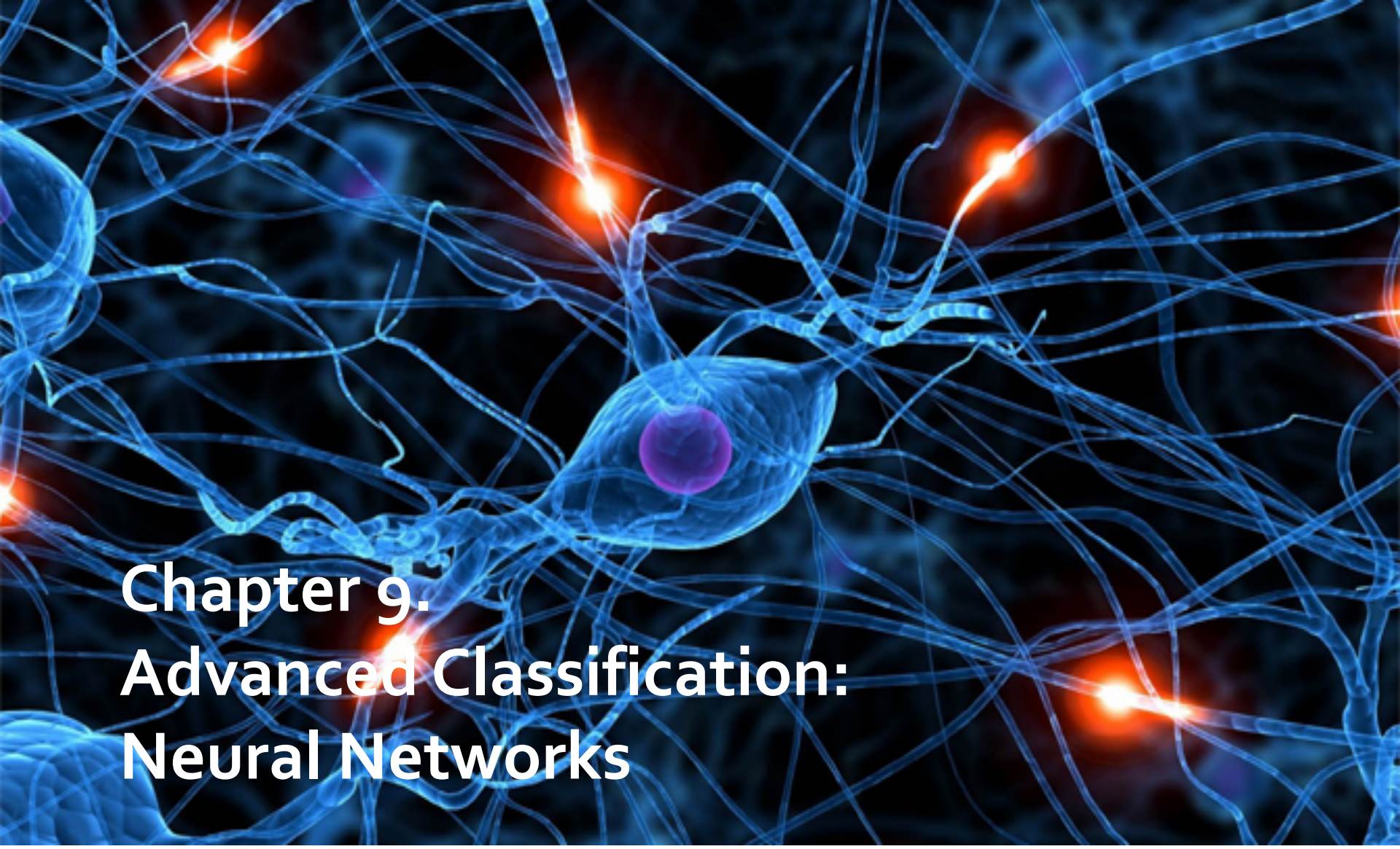
LIBSVM supports the following learning tasks.

(1) SVC: support vector classification (twoclass and multiclass);
(2) SVR: support vector regression.
(3) One-class SVM.

[1]The Web address of the package is at http://www.csie.ntu.edu.tw/~cjlin/libsvm.
[2]This LIBSVM implementation document was created in 2001 and has been maintained at http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf.
[3]LIBSVM FAQ: http://www.csie.ntu.edu.tw/~cjlin/libsvm/faq.html.

This work was supported in part by the National Science Council of Taiwan via the grants NSC 89-2213-E-002-013 and NSC 89-2213-E-002-106.
Authors' addresses: C.-C. Chang and C.-J. Lin (corresponding author), Department of Computer Science, National Taiwan University, Taipei 106, Taiwan; email: cjlin@csie.ntu.edu.tw.

DOI 10.1145/1961189.1961199 http://doi.acm.org/10.1145/1961189.1961199



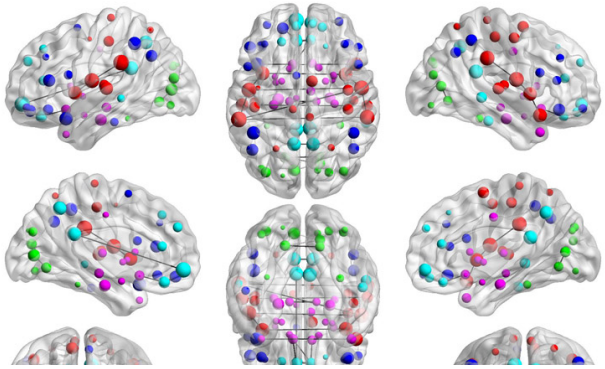
Chapter 9. Advanced Classification: Neural Networks

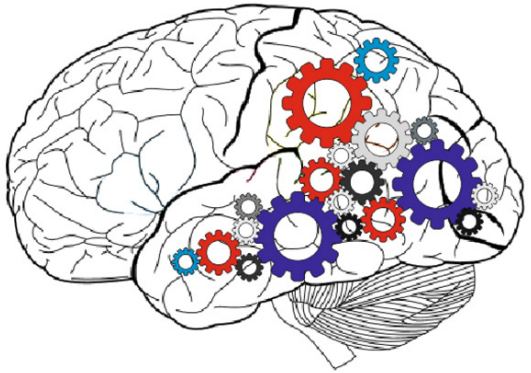
Meng Jiang

Data Science

Artificial Neural Networks (ANNs)

- **Computing systems** inspired by the *biological neural networks* that constitute *animal brains*



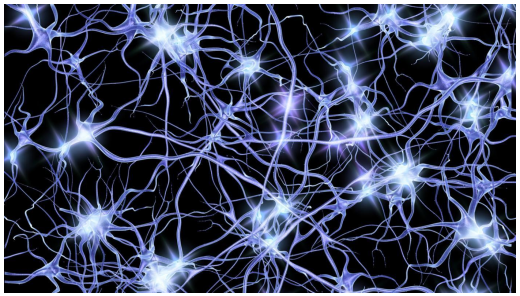
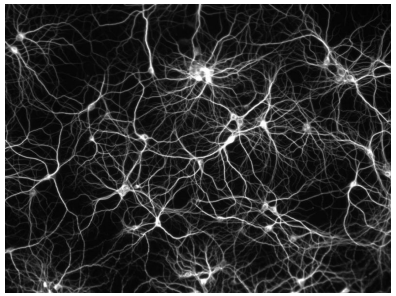
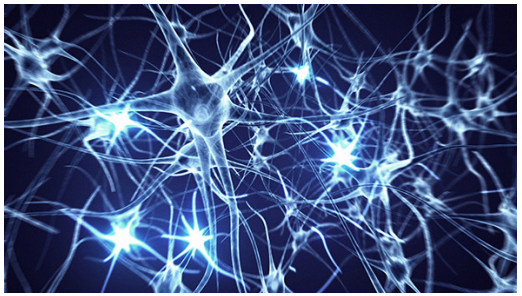


Biological Inspirations

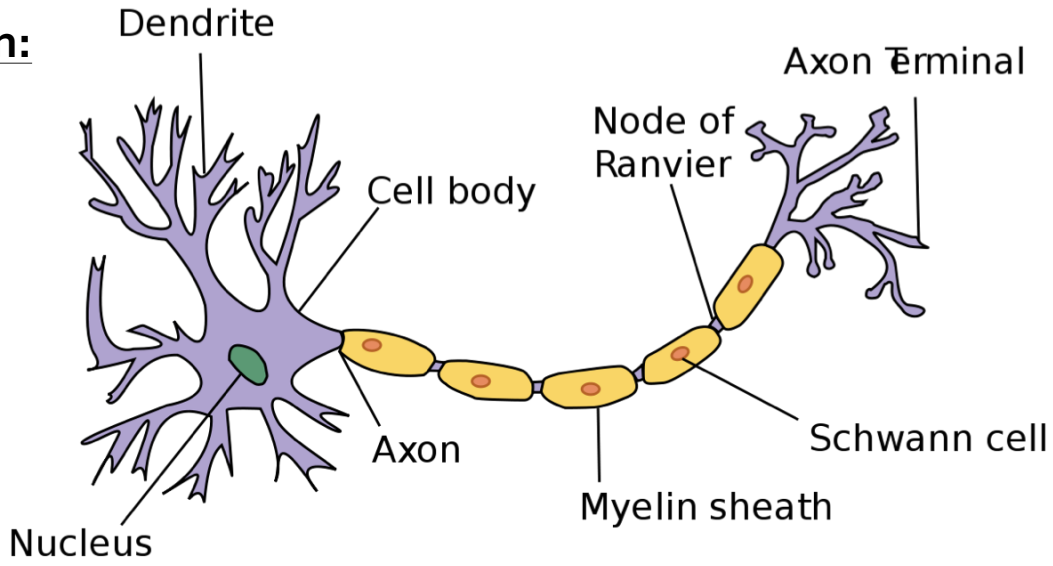
- Some numbers
  - The human brain contains about **10,000,000,000** (10B) nerve cells (neurons)
  - Each neuron is connected to the others through **10,000** synapses
- Properties of the brain
  - It can learn, reorganize itself from experience
  - It adapts to the environment
  - It is robust and fault tolerant

Neurons

- Started by psychologists and neurobiologists to develop and test computational analogues of ***neurons***

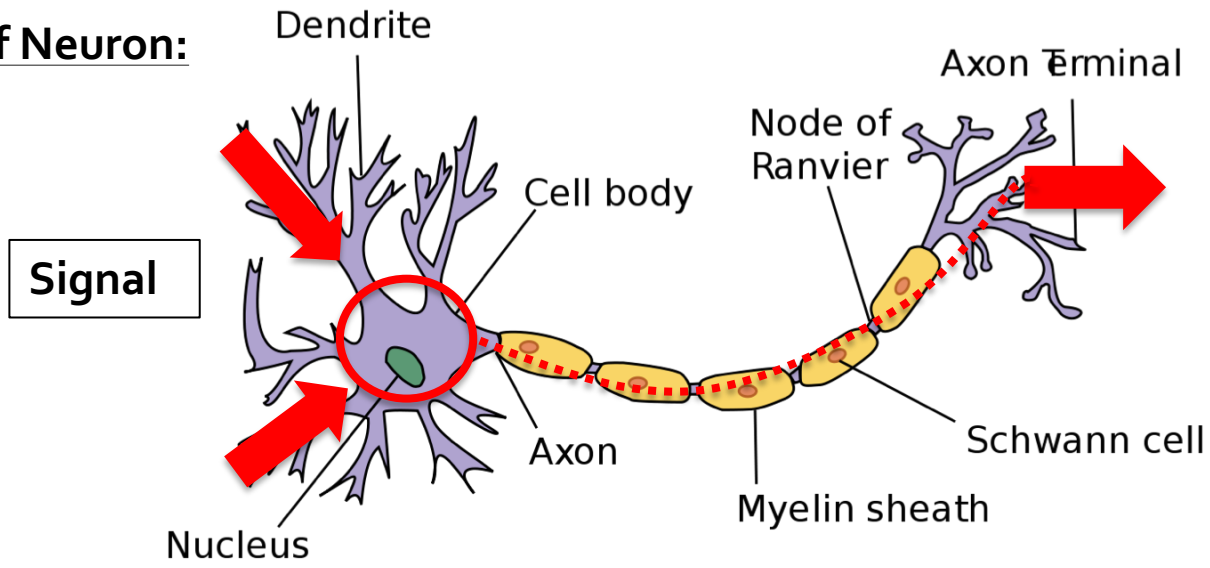
Structure of Neuron:



Neurons (cont.)

- A neuron has
  - A branching input (**dendrites**)
  - A branching output (the **axon**)
- The **information/signal** circulates from the dendrites to the axon via the **cell body**

Structure of Neuron:

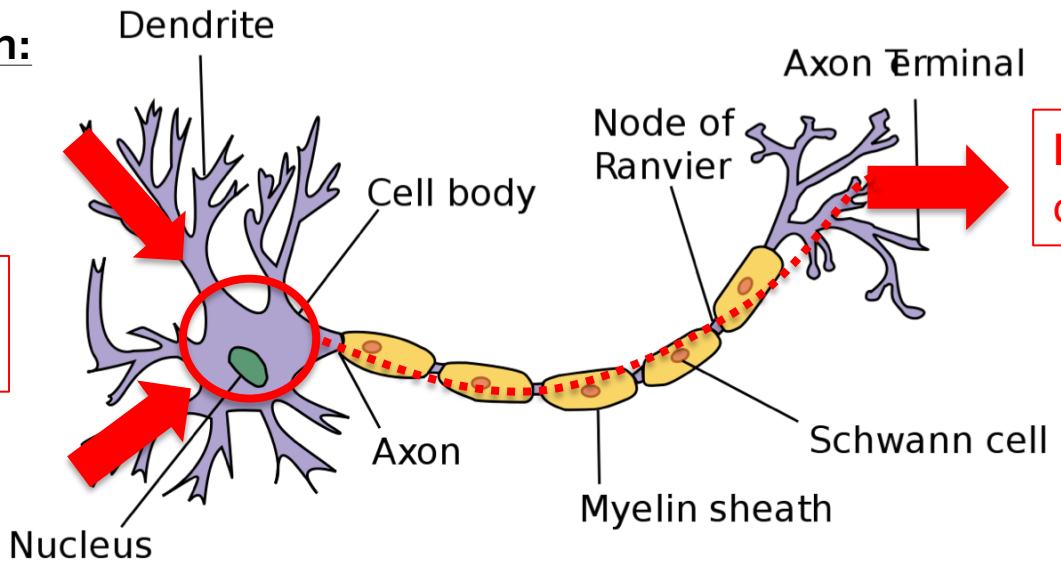


Neurons for Classification

- A neuron has
  - A branching input (**dendrites**)
  - A branching output (the **axon**)
- The ~~**information/signal**~~ circulates from the dendrites to the axon via the **cell body**

feature-label relationship

Structure of Neuron:

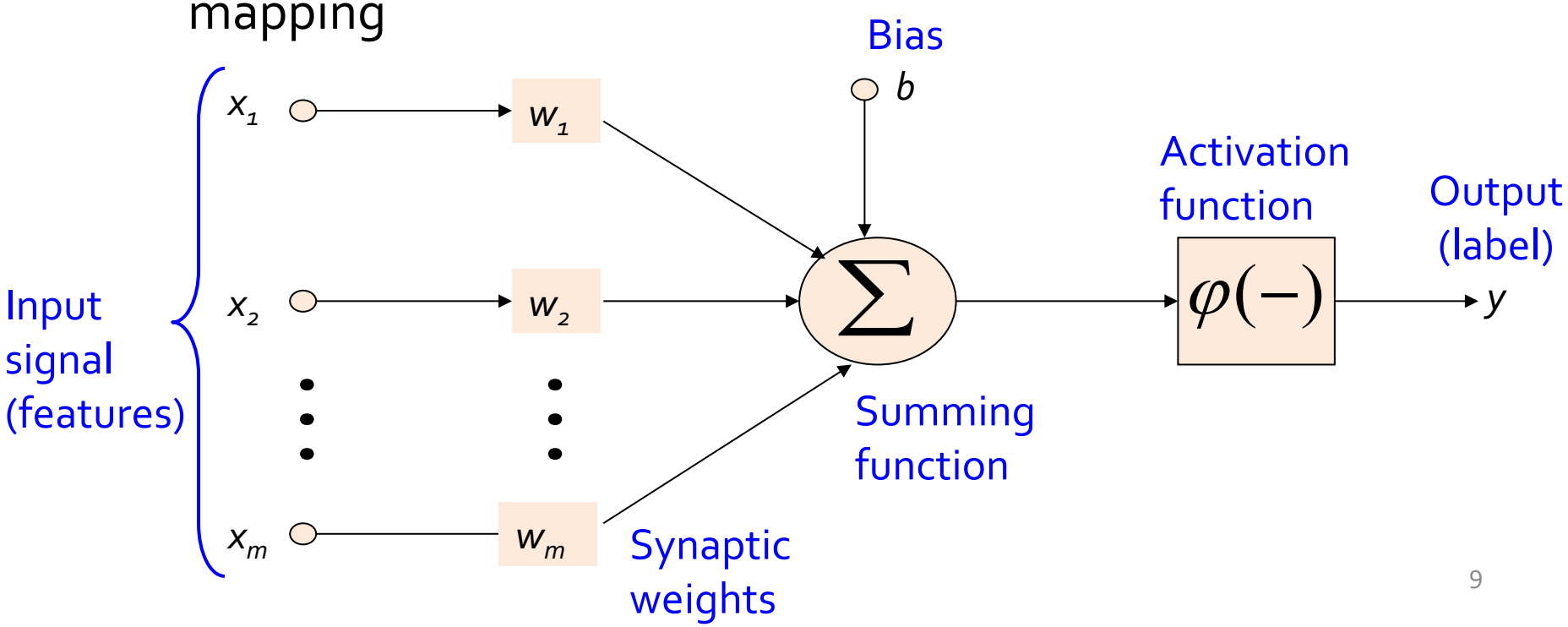


Feature values of a data point

Label of a data point

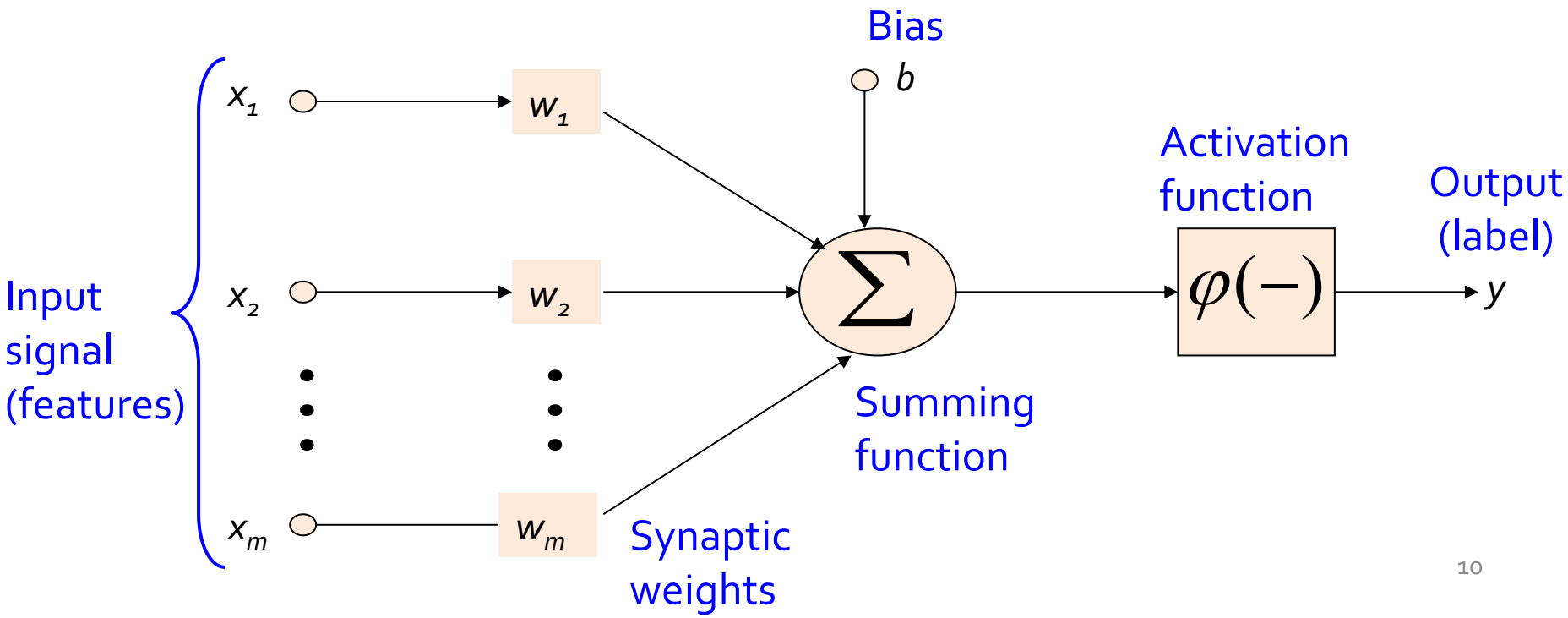
Artificial Neuron

- Definition: **Non-linear, parameterized** function with restricted output range
  - An *n*-dimensional input vector **x** is mapped into variable y by means of the scalar product and a nonlinear function mapping



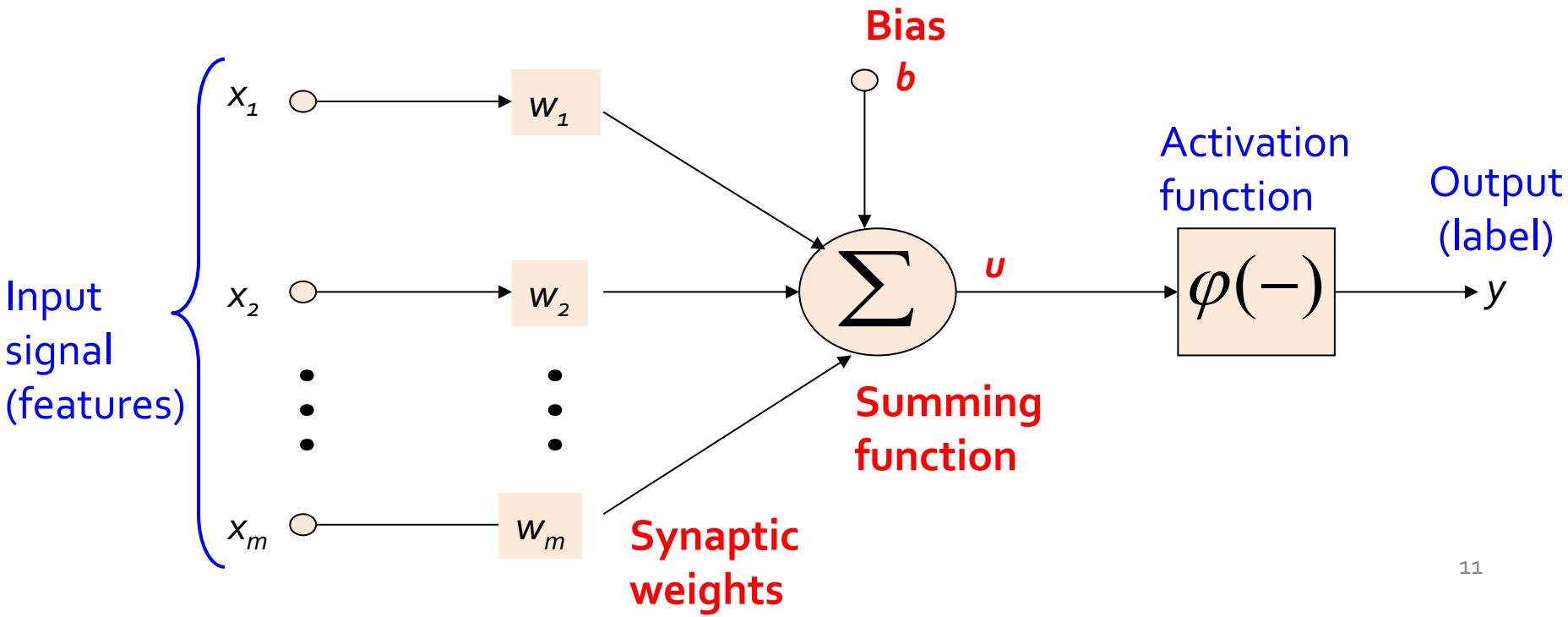
Artificial Neuron (cont.)

- It is the basic information processing **unit** of an ANN.
- The inputs are multiplied by their corresponding weights to form a *weighted sum*, which is added to the *bias* associated with unit. Then a *nonlinear activation function* is applied to it.



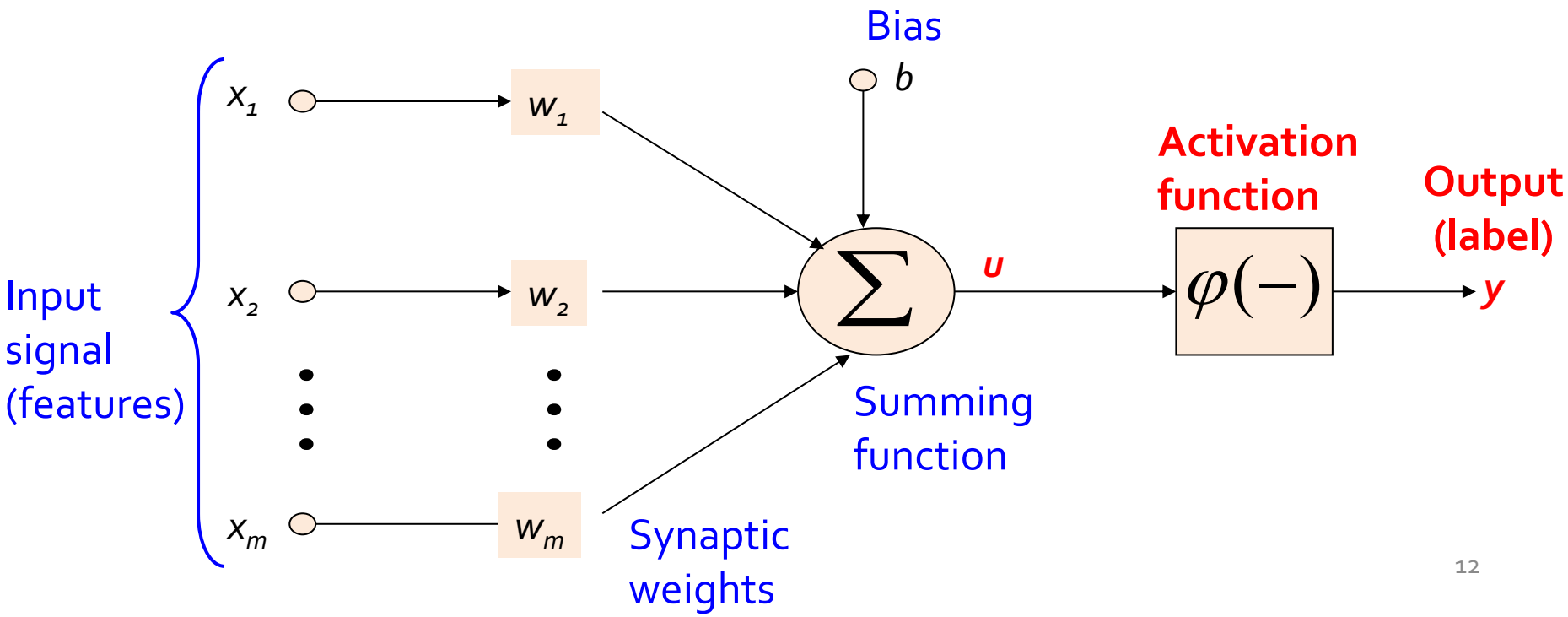
Artificial Neuron (cont.)

- A set of synapses or connecting links, each link characterized by a **weight**: $w_1, w_2, \ldots, w_m$
- A summing function (linear combiner) which computes the **weighted sum of the inputs** plus **bias**: $u = \sum_{i=1}^{m} w_i x_i + b$



Artificial Neuron (cont.)

- **Activation function** (squashing function) φ: $u \rightarrow y$ for limiting the amplitude of the output of the neuron
- E.g., **Logistic** functions (including **Sigmoid** function), **Hyperbolic tangent** function

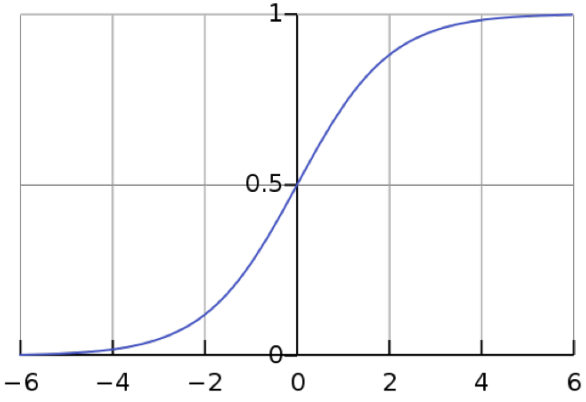


Activation Function: an "S" shape

- **Logistic functions**

$$f(x) = \frac{L}{1+e^{-k(x-x_0)}}$$

  - e = the natural logarithm base (Euler's number)
  - x_0 = the x-value of the sigmoid's midpoint
  - L = the curve's maximum value
  - K = the steepness of the curve



L = 1, k = 1, x_0 = 0

- **Sigmoid functions**

$$f(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x} = \frac{1}{2} + \frac{1}{2}\tanh\left(\frac{x}{2}\right)$$

  if x ∈ (-∞, +∞), then f(x) ∈ (0,1)

- **Hyperbolic tangent function**

$$f(x) = \tanh\left(\frac{x}{2}\right) = \frac{1-e^{-x}}{1+e^{-x}}$$

  if x ∈ (-∞, +∞), then f(x) ∈ (-1,1)

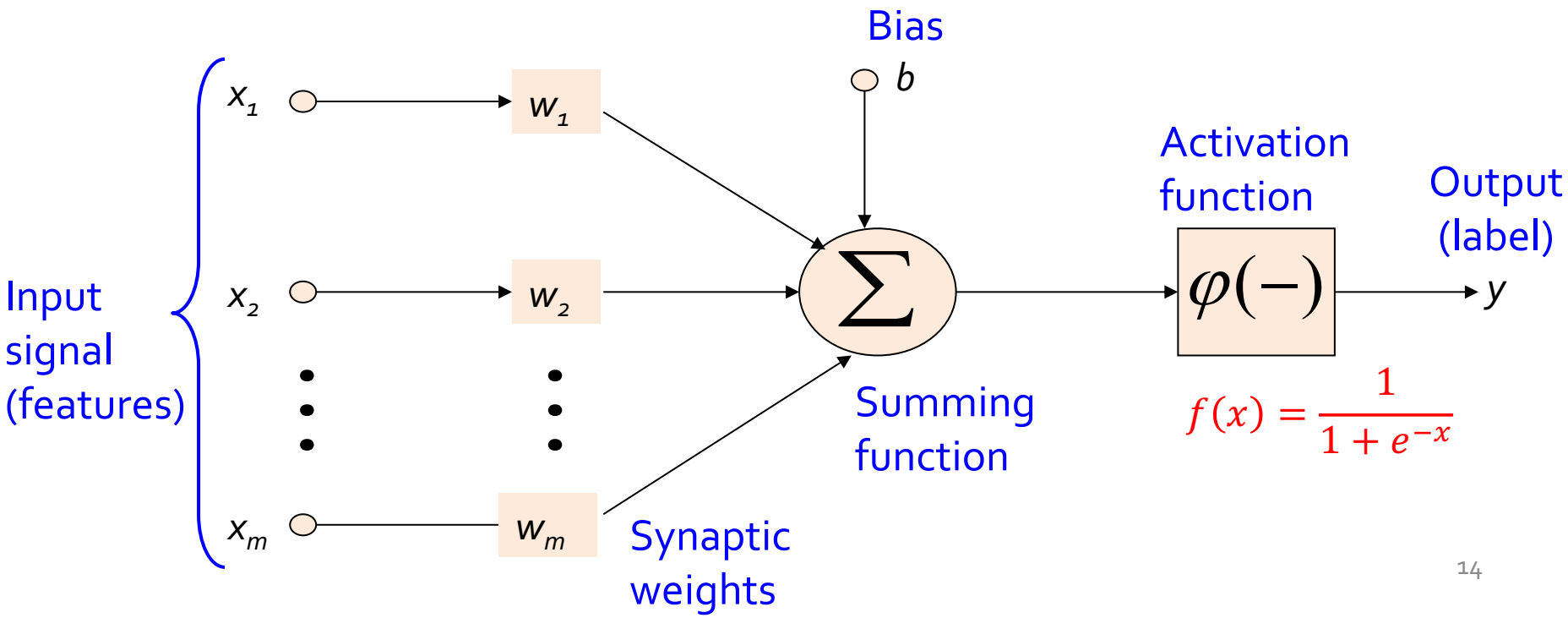
  if x ∈ [0, +∞), then f(x) ∈ [0,1)

Artificial Neuron: An Example

weights = [3.0, -0.5, -2.5, 1.5], b = -0.1

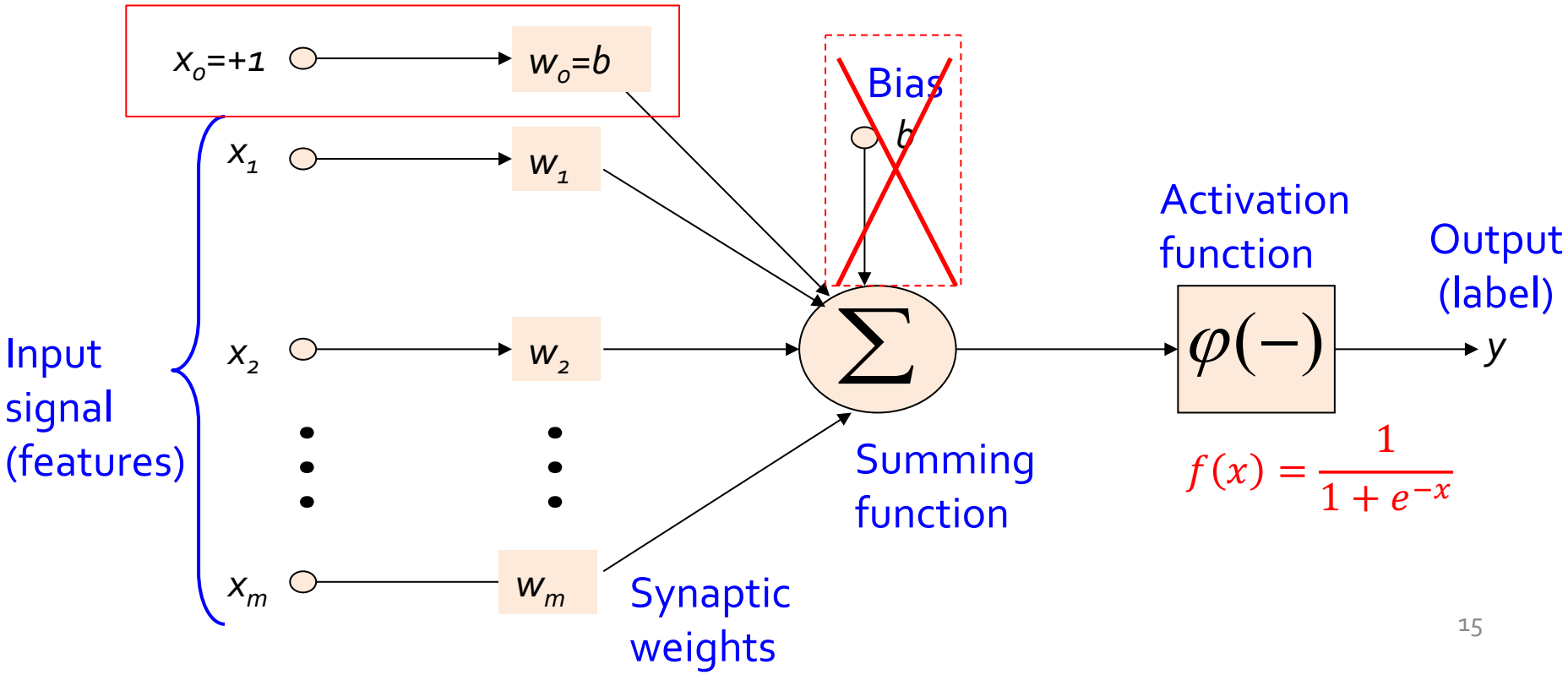
Is KDD?

	"data mining"	"web search"	"click through rate"	"pattern"	Label	**Output**
PID1	1	0	0	1	1	f(4.4)
PID2	0	1	1	0	0	f(-3.1)
PID3	1	1	0	1	1	f(3.9)



Bias

- $u = \sum_{i=1}^{m} w_i x_i + b \rightarrow u = \sum_{i=0}^{m} w_i x_i$
- $y = \varphi(u) = \varphi(\sum_{i=0}^{m} w_i x_i)$



Learning Process

- During the learning process the **weights** are modified in order to model the particular learning task correctly on the **training examples**.
- Given a data point (x_0=+1, x_1, …, x_m; y), use **gradient descent** to learn the weights (w_0, w_1, …, w_m) so that

$$\hat{y} = \varphi(u) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+\cdots+w_mx_m)}} \rightarrow y$$

This is actually Logistic Regression!

- Q: What is gradient descent? What is gradient?

Gradient

- What is gradient of a function *f(x)*?
  - Rate of change of function *f(x)* w.r.t. input x $\nabla f(x) = \frac{df}{dx}$
- What is gradient of a multiple variable function?
  - $\nabla f(x_1, x_2, \ldots, x_n) = \left[\frac{\delta f}{\delta x_1}, \frac{\delta f}{\delta x_2}, \frac{\delta f}{\delta x_3}, \ldots, \frac{\delta f}{\delta x_n}\right]$ $f(x_1, x_2, x_3, \ldots, x_n)$
- Example:

(1) $f(w_o, w_1, w_2, \ldots, w_m) = w_o x_o + w_1 x_1 + \ldots + w_m x_m$

$\nabla f(w_o, w_1, w_2, \ldots, w_m) = [x_o, x_1, x_2, \ldots, x_m]$

(2) $f(x) = \frac{1}{1+e^{-x}}$

$$\nabla f(x) = \frac{1}{1+e^{-x}} - \left(\frac{1}{1+e^{-x}}\right)^2 = f(x)(1 - f(x))$$

Gradient Descent

- Gradient descent is a **first-order iterative optimization** algorithm for finding the minimum of a function.
- To find a **local minimum** of a function using gradient descent, one *takes steps proportional to the negative of the gradient* (or of the approximate gradient) of the function at the current point.
- If instead one *takes steps proportional to the positive of the gradient*, one approaches a **local maximum** of that function; the procedure is then known as gradient ascent.

Gradient Descent (cont.)

$$\nabla f(x_1, x_2, \ldots, x_n) = \left[\frac{\delta f}{\delta x_1}, \frac{\delta f}{\delta x_2}, \frac{\delta f}{\delta x_3}, \ldots, \frac{\delta f}{\delta x_n}\right]$$

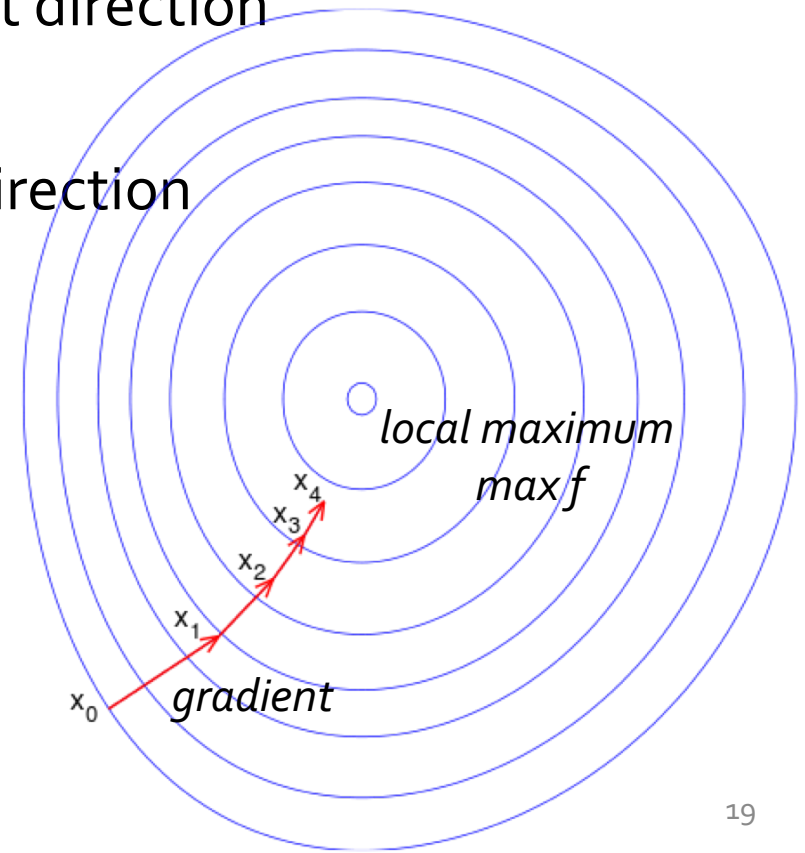
- Gradient: direction of steepest increase
- Magnitude: rate of change in that direction

Local minimum: go in the reverse direction

$$\mathbf{x} \leftarrow \mathbf{x} - R\nabla f(\mathbf{x})$$

where $\mathbf{x} = [x_1, x_2, \ldots, x_n]$

$$x_i \leftarrow x_i - R\frac{\delta f}{\delta x_i}$$



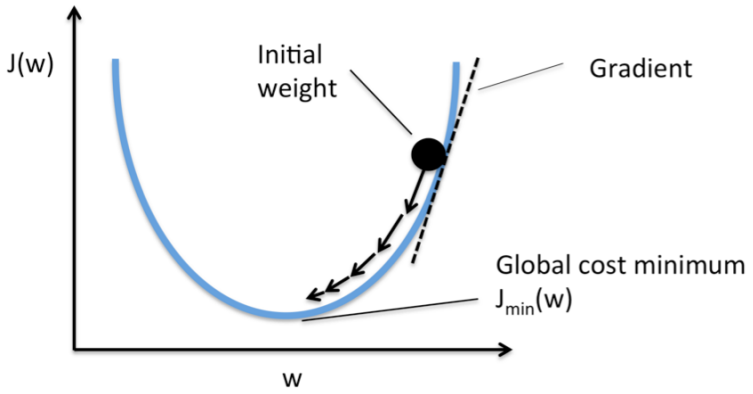
Gradient Descent for a Single Neuron

- Given a data point (x_0=+1, x_1, …, x_m; y), use **gradient descent** to learn the weights (w_0, w_1, …, w_m)

$$\hat{y} = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + \cdots + w_m x_m)}}$$

We want to minimize the loss function:

$$\min J(\boldsymbol{w}) = \frac{1}{D} \sum_{d=1}^{D} (y_d - \hat{y}_d)^2$$



Q: $\nabla J(\boldsymbol{w}) = \frac{\delta J(\boldsymbol{w})}{\delta \boldsymbol{w}} = [\frac{\delta J}{\delta w_0}, \frac{\delta J}{\delta w_1}, \ldots, \frac{\delta J}{\delta w_m}]$?


ARE YOU READY
TO COMPUTE SOME PARTIAL DERIVATIVES ?
memegenerator.net

Partial Derivatives

Given $\min J(\boldsymbol{w}) = \frac{1}{D}\sum_{d=1}^{D}(y_d - \hat{y}_d)^2$,

where $\hat{y} = \frac{1}{1+e^{-(w_0x_0+w_1x_1+\cdots+w_mx_m)}}$

$$\frac{\delta J}{\delta w_i} = \frac{2}{D}\sum_{d=1}^{D}(y_d - \hat{y}_d)\cdot(-1)\cdot\hat{y}_d(1-\hat{y}_d)\cdot x_{d,i}$$

for $i = 0, 1, \ldots, m$

Example

Initialized weights = [0.1, 0.1, 0.1, 0.1, 0.1]

	x_0	"data mining"	"web search"	"click through rate"	"pattern"	Label y_d
PID1	+1	1	0	0	1	1
PID2	+1	0	1	1	0	0
PID3	+1	1	1	0	1	1

$$\frac{\delta J}{\delta w_i} = \frac{2}{D}\sum_{d=1}^{D}(y_d - \hat{y}_d)\cdot(-1)\cdot\hat{y}_d(1-\hat{y}_d)\cdot x_{d,i}$$

$$\hat{y}_1 = \varphi(0.1 + 0.1 + 0 + 0 + 0.1) = \varphi(0.3) = 0.574$$

$$\hat{y}_2 = \varphi(0.3) = 0.574$$

$$\hat{y}_3 = \varphi(0.4) = 0.599$$

$$(y_1 - \hat{y}_1)\cdot(-1)\cdot\hat{y}_1(1-\hat{y}_1) = (1-0.574)\cdot(-1)\cdot 0.574\cdot(1-0.574) = -0.104$$

$$(y_2 - \hat{y}_2)\cdot(-1)\cdot\hat{y}_2(1-\hat{y}_2) = (0-0.574)\cdot(-1)\cdot 0.574\cdot(1-0.574) = 0.140$$

$$(y_3 - \hat{y}_3)\cdot(-1)\cdot\hat{y}_3(1-\hat{y}_3) = (1-0.599)\cdot(-1)\cdot 0.599\cdot(1-0.599) = -0.096$$

Example (cont.)

Initialized weights = [0.1, 0.1, 0.1, 0.1, 0.1]

	x_0	"data mining"	"web search"	"click through rate"	"pattern"	Label y_d
PID1	+1	1	0	0	1	1
PID2	+1	0	1	1	0	0
PID3	+1	1	1	0	1	1

$$\frac{\delta J}{\delta w_i} = \frac{2}{D}\sum_{d=1}^{D}(y_d - \hat{y}_d)\cdot(-1)\cdot\hat{y}_d(1-\hat{y}_d)\cdot x_{d,i}$$

$\frac{\delta J}{\delta w_0}$ = 2/3 * (-0.104*1+0.140*1-0.096*1) = -0.04 < 0

$\frac{\delta J}{\delta w_1}$ = 2/3 * (-0.104*1+0.140*0-0.096*1) = -0.133 < 0

$\frac{\delta J}{\delta w_2}$ = 2/3 * (-0.104*0+0.140*1-0.096*1) = 0.029 > 0

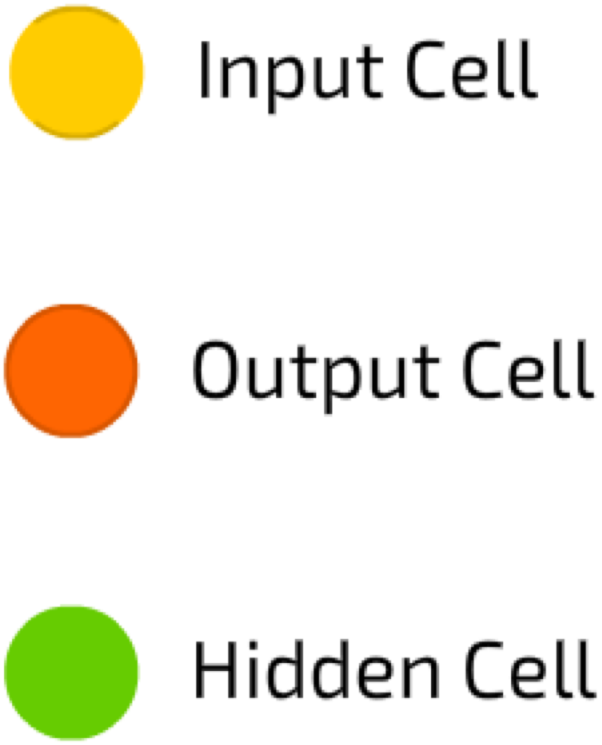
$\frac{\delta J}{\delta w_3}$ = 2/3 * (-0.104*0+0.140*1-0.096*0) = 0.093 > 0

$\frac{\delta J}{\delta w_4}$ = 2/3 * (-0.104*1+0.140*0-0.096*1) = -0.133 < 0

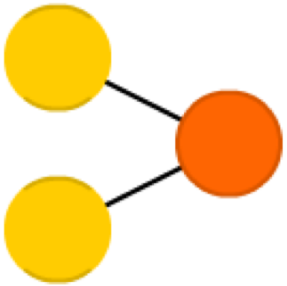
If the learning rate (magnitude) is 1.0, new weights =
[0.14, 0.233, 0.071, 0.007, 0.233]

Hidden Layer of Neurons

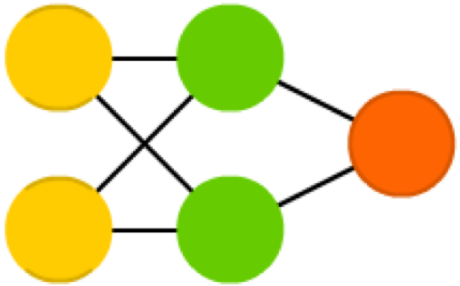
Input: data points of n=2 features
Output: one label



Perceptron (P)



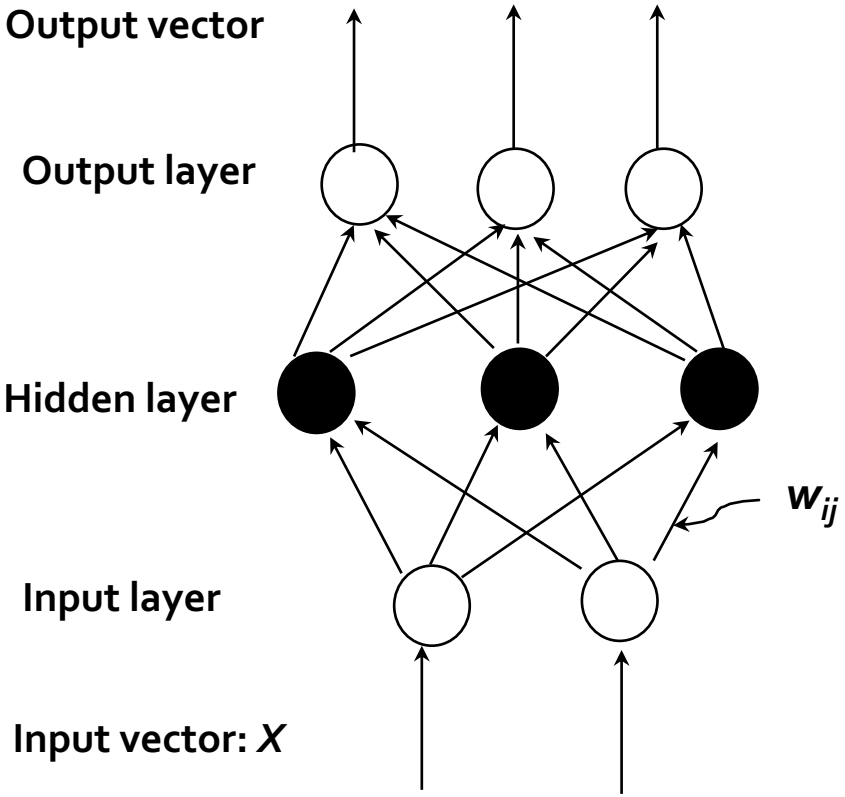
Feed Forward (FF)

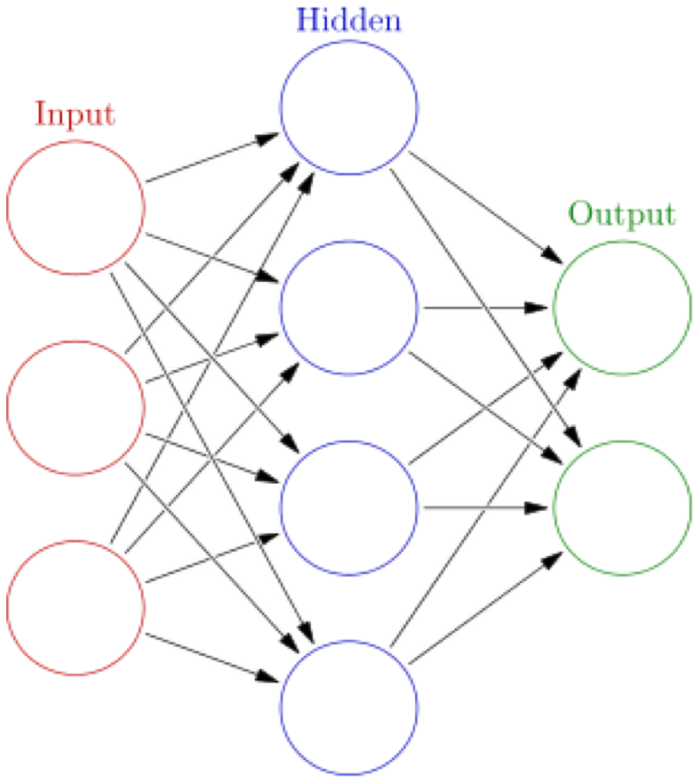


Each hidden cell (neuron)'s output is a new feature (that is a non-linear function result of old features) as input for the output cell.

Can we have multiple output cells? Yes!

A Three-Layer Feed-Forward Neural Network





Q: How many weight parameters?

This is a 2-3-3 ANN.

2*3+3*3 = 15

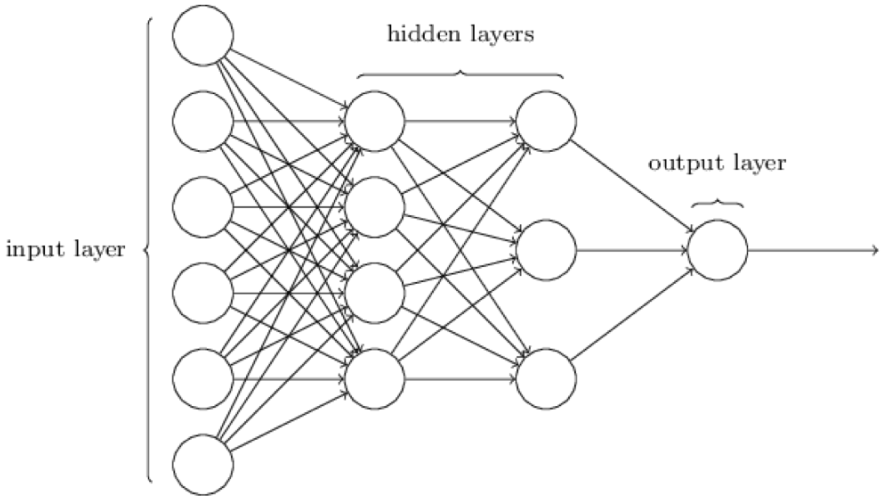
This is a 3-4-2 ANN.

3*4+4*2 = 20

Q': How many weight parameters did we have in an SVM?

Can we have multiple hidden layers? Yes!

A Two-Hidden-Layer Feed-Forward Neural Network



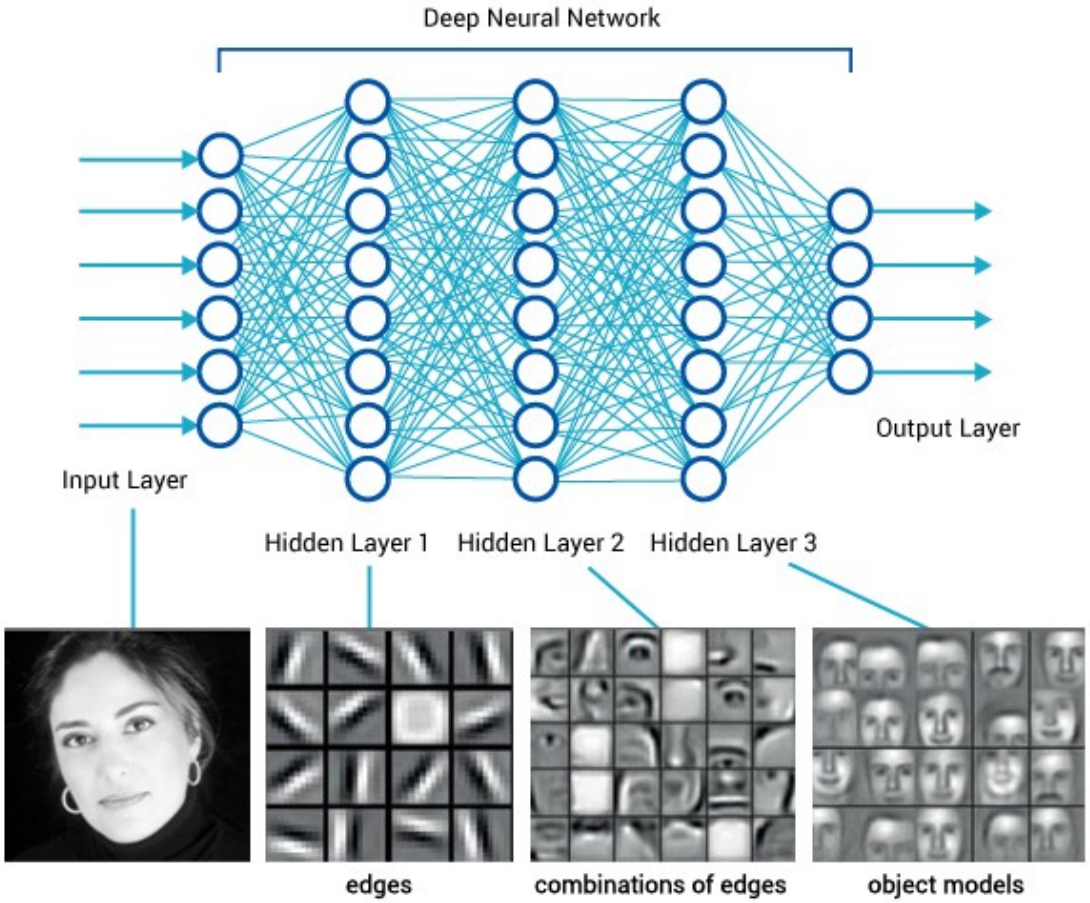
Suppose we have one hidden layer of 7 neurons. It is a 6-7-1 network.
We have 6*7+7*1 = 49 weight parameters to learn.

Here we have two hidden layers (4+3). It is a 6-4-3-1 network.
We have 6*4+4*3+3*1 = 39 weight parameters to learn.

When the network's architecture becomes more complicated, the number of parameters may **NOT** become more, but the performance may become better:
It can model more ***complicated non-linear distributions***.

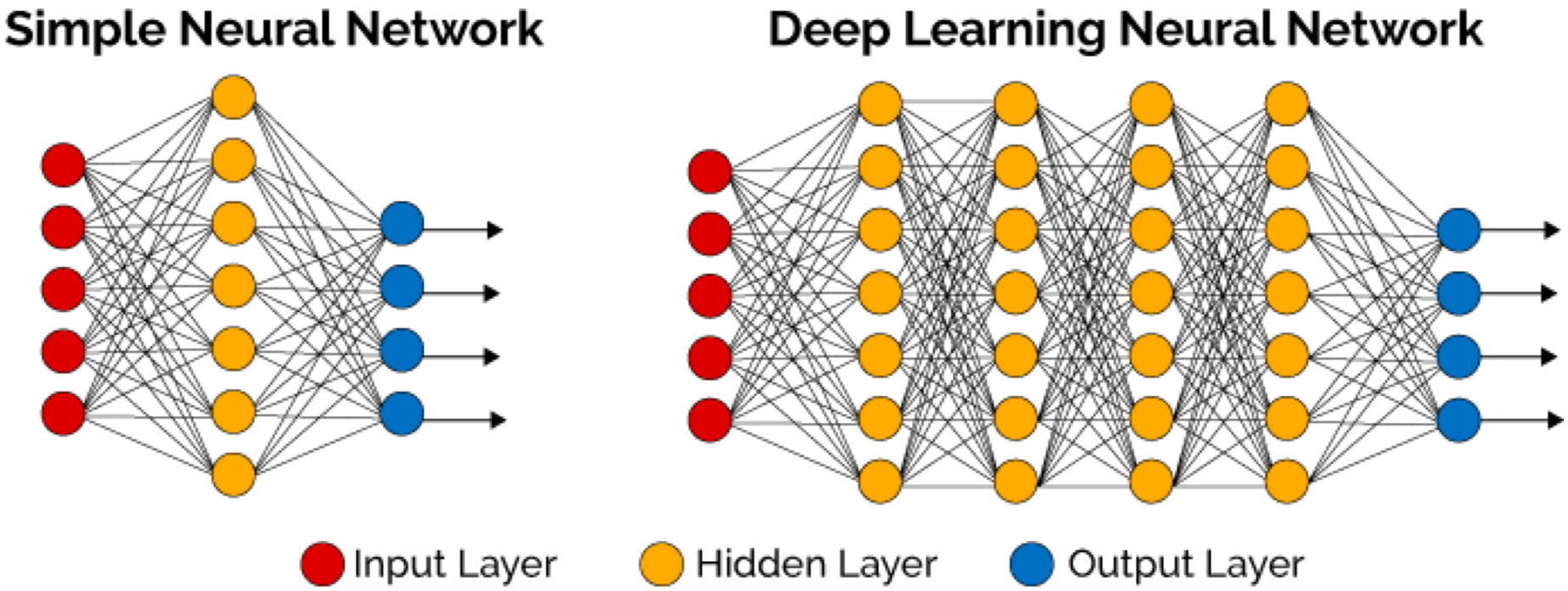
Deep Neural Networks

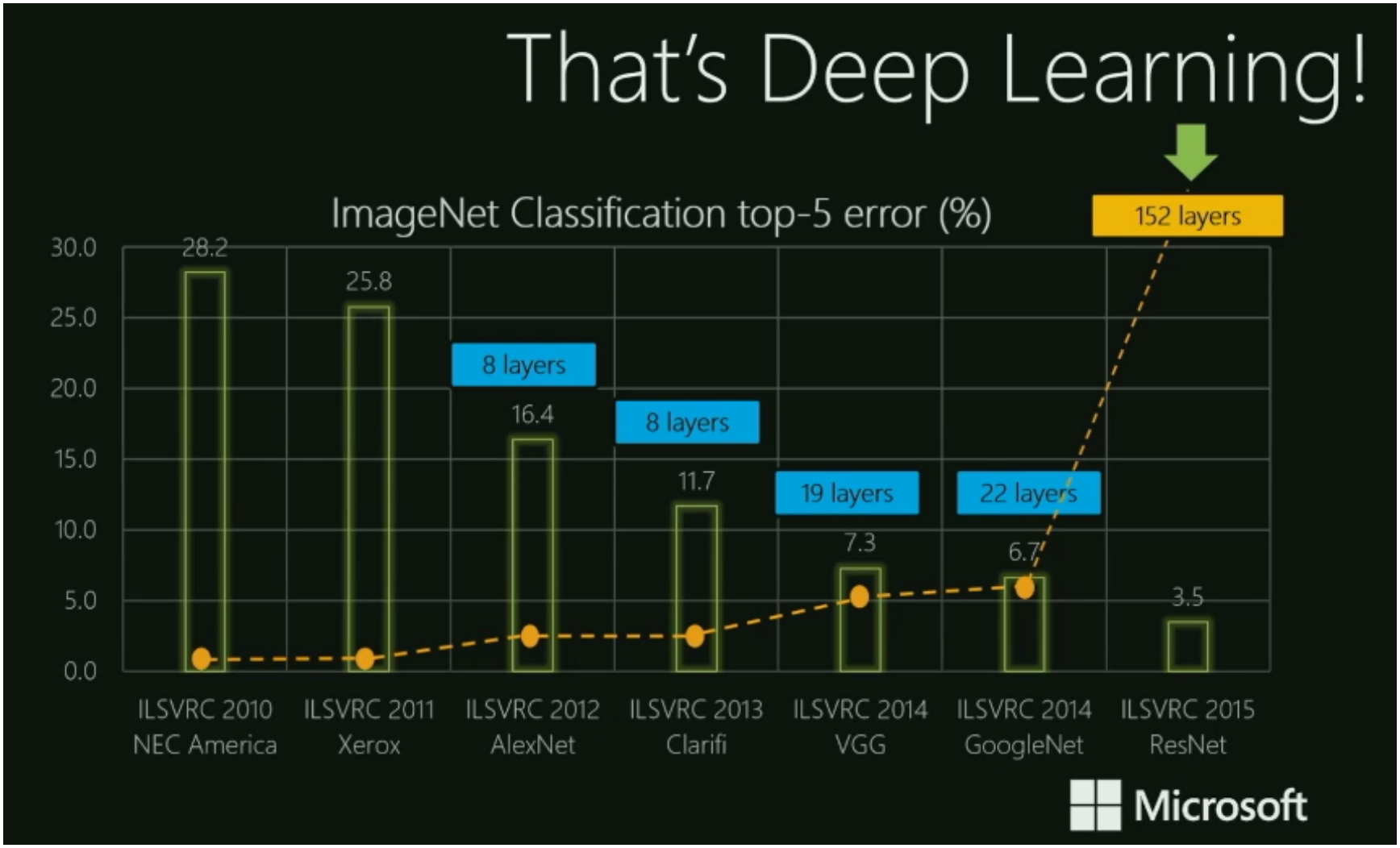
- One of the main deep learning architectures.



Deep Neural Networks (cont.)

- *How many layers is a DEEP network architecture?*




That's Deep Learning!
ImageNet Classification top-5 error (%)
152 layers
30.0
25.0
20.0
15.0
10.0
5.0
0.0
28.2
25.8
8 layers
16.4
8 layers
11.7
19 layers
7.3
22 layers
6.7
3.5
ILSVRC 2010
NEC America
ILSVRC 2011
Xerox
ILSVRC 2012
AlexNet
ILSVRC 2013
Clarifi
ILSVRC 2014
VGG
ILSVRC 2014
GoogleNet
ILSVRC 2015
ResNet
Microsoft

AN ANALYSIS OF DEEP NEURAL NETWORK MODELS FOR PRACTICAL APPLICATIONS

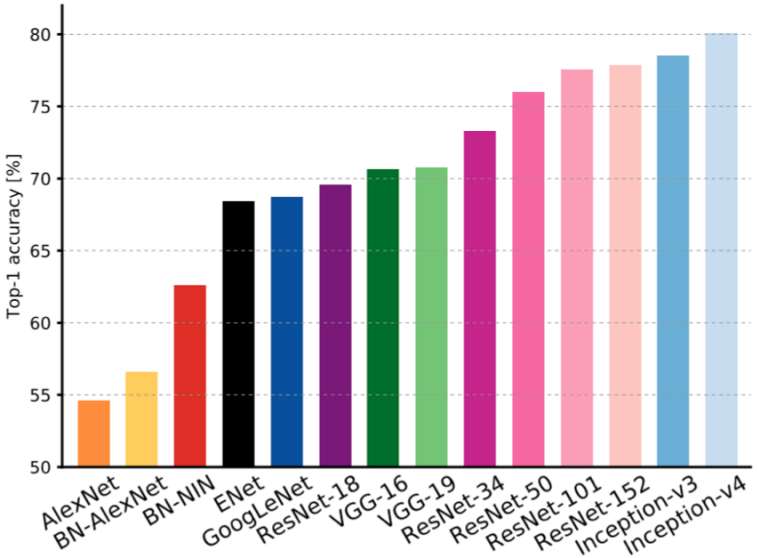


Figure 1: **Top1 *vs.* network.** Single-crop top-1 validation accuracies for top scoring single-model architectures. We introduce with this chart our choice of colour scheme, which will be used throughout this publication to distinguish effectively different architectures and their correspondent authors. Notice that networks of the same group share the same hue, for example ResNet are all variations of pink.

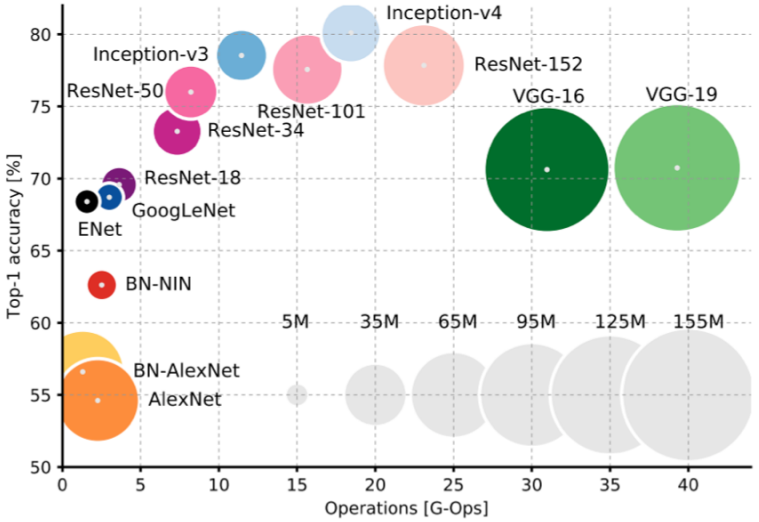


Figure 2: **Top1 *vs.* operations, size $\propto$ parameters.** Top-1 one-crop accuracy versus amount of operations required for a single forward pass. The size of the blobs is proportional to the number of network parameters; a legend is reported in the bottom right corner, spanning from 5×10^6 to 155×10^6 params. Both these figures share the same y-axis, and the grey dots highlight the centre of the blobs.

https://arxiv.org/pdf/1605.07678.pdf

Summary: Components of a Feed-Forward Neural Network

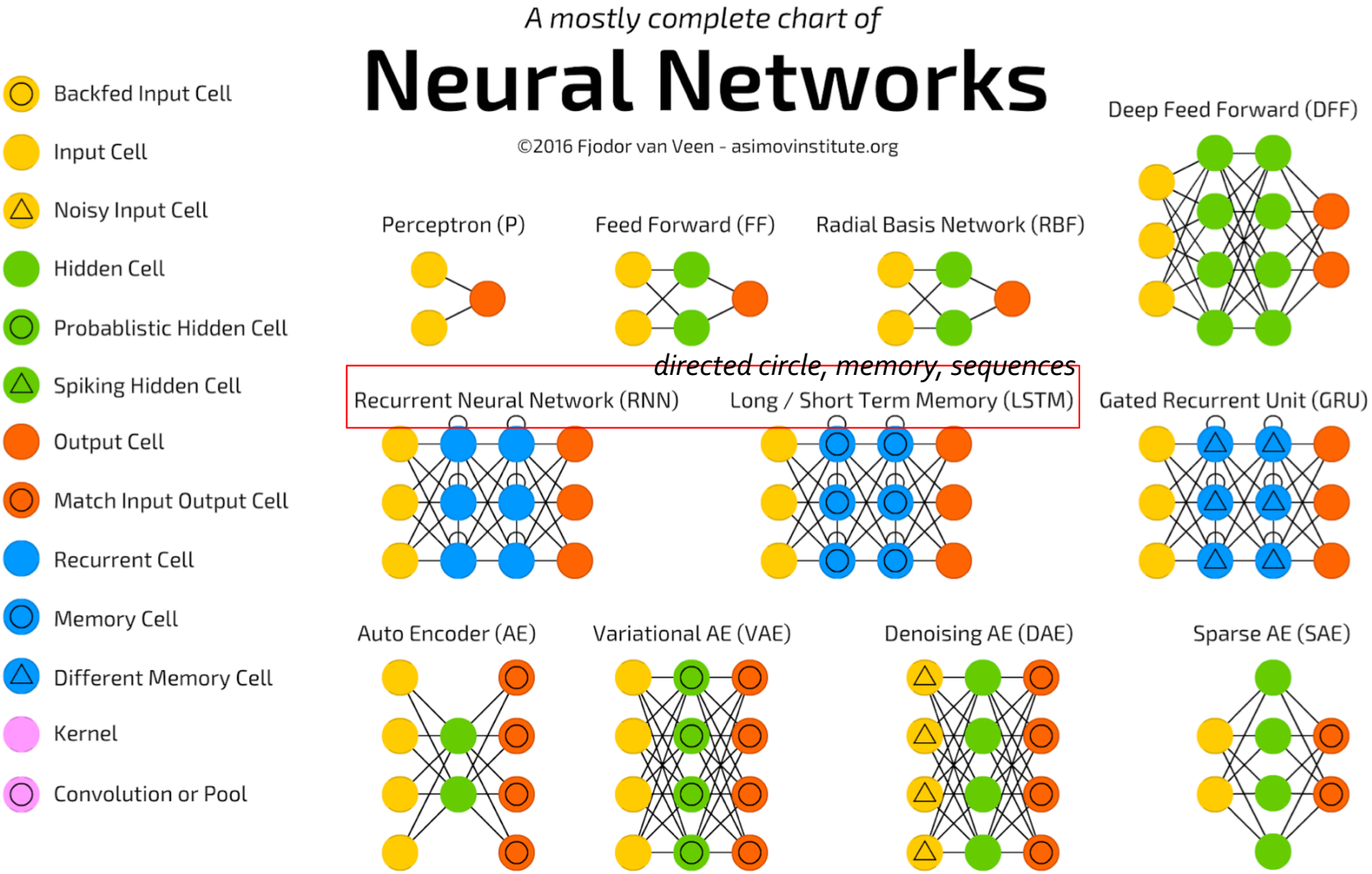
- The **inputs** to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**
- The number of **hidden layers** is arbitrary
- The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction
- The network is **feed-forward**: None of the weights cycles back to an input unit or to an output unit of a previous layer
- From a statistical point of view, networks perform **nonlinear regression**
  - Given **enough** hidden units and **enough** training samples (**and enough what?**), they can closely approximate ***any*** function

Limitations of Neural Networks

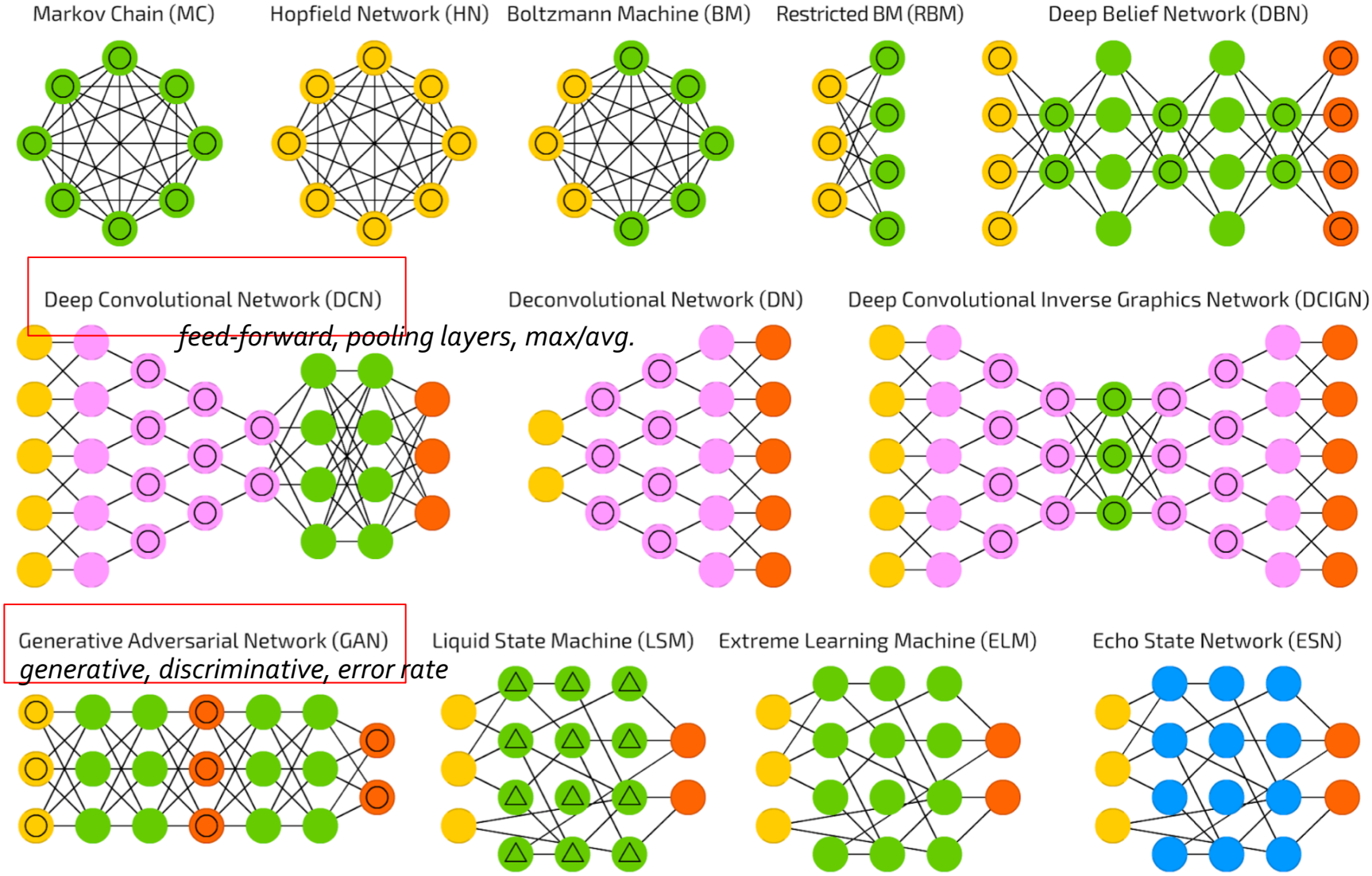
Random initialization + **densely connected networks** lead to:

- High cost
  - Training the entire neural network is to train **all the interconnected logistic regressions**.
- Difficult to train as the number of hidden layers increases
  - Recall that logistic regression is trained by **gradient descent** that is looking for **local optimum**.
- Stuck in local optima
  - The objective function of the neural network is **usually not convex**.
  - The random initialization does not guarantee starting from the proximity of global optima.
- Difficult to determine the network topology
  - Once a network has been trained and its accuracy is **unacceptable**, repeat the training process with a **different network topology** or **a different set of initial weights.**

Beyond Input/Hidden/Output Cells and Beyond Feed-Forward Network Architectures



Continue...


Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
feed-forward, pooling layers, max/avg.
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
generative, discriminative, error rate
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)