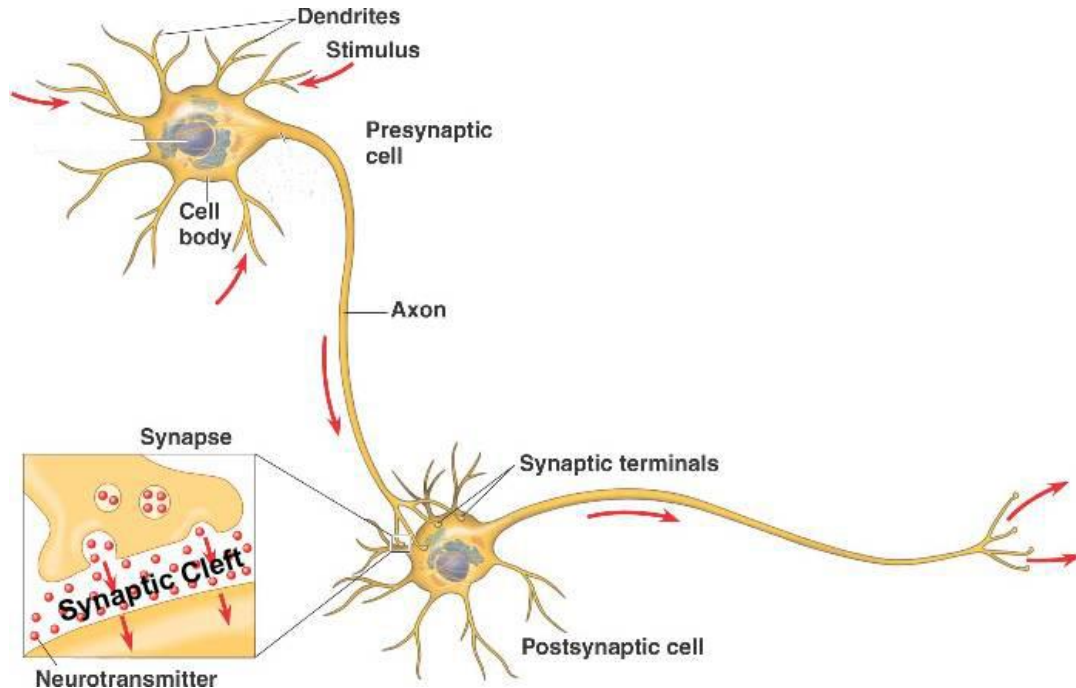


Neural Network ? Neuron?



Artificial neural networks were proposed to mimic human neural networks

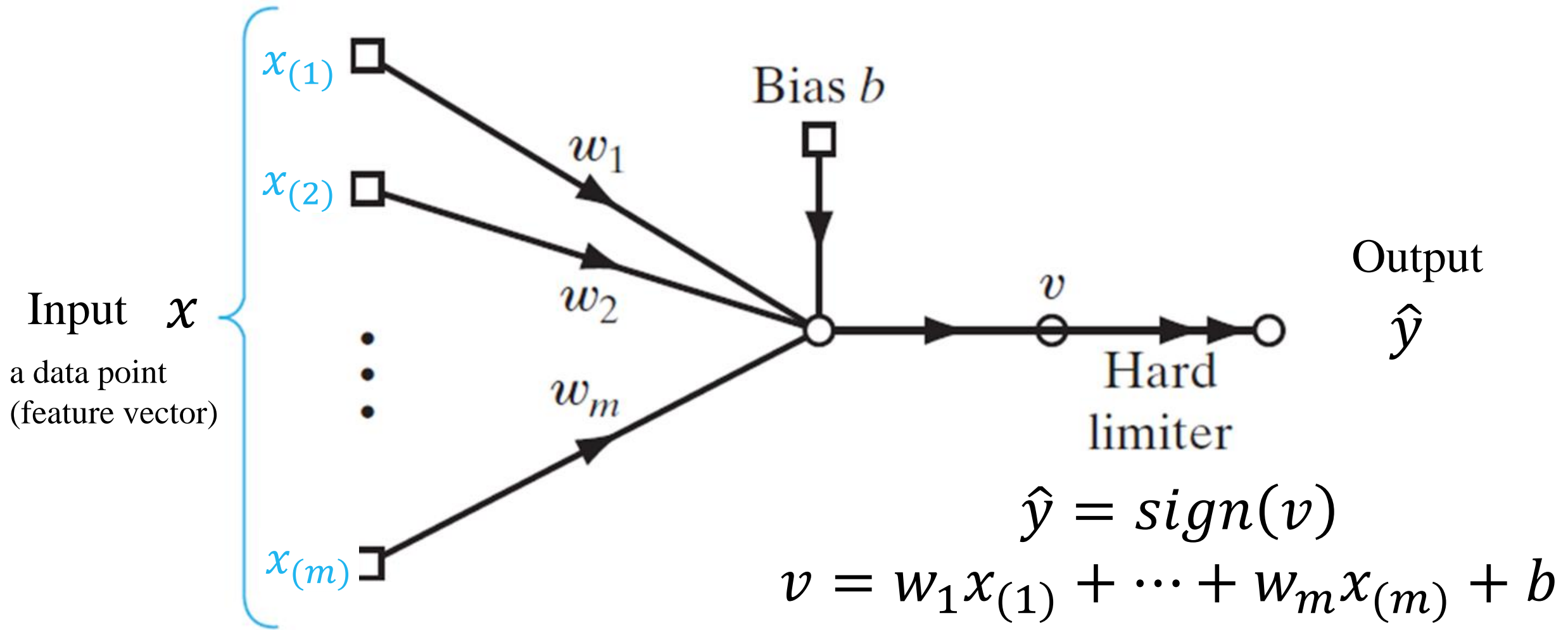
They could be similar, but NOT the same

In the field of Machine Learning,
Network refers to Artificial Neural Network
Unit refers to Neuron in an artificial neural network

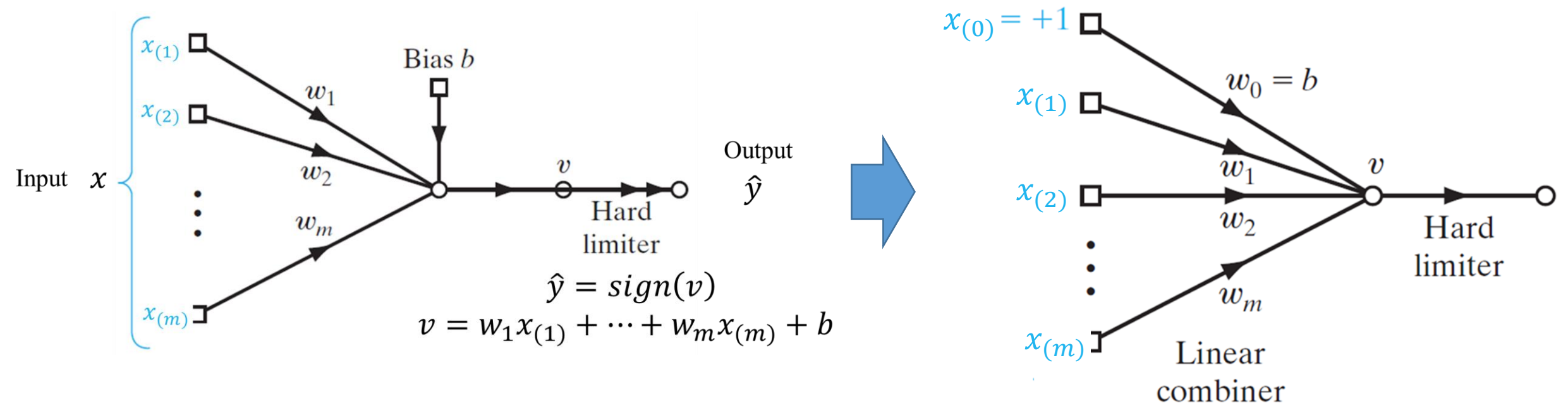
Neural Networks were 'invented' ~ 60 years ago

- McCulloch and Pitts (1943), introducing the idea of neural networks as computing machines.
- Hebb (1949), postulating the first rule for self-organized learning.
- Rosenblatt (1958), proposing the perceptron as the first model for learning with a teacher (i.e., supervised learning).
 - Rosenblatt's perceptron is the simplest form of a neural network used for the classification of patterns that are linearly separable

Perceptron



$$\text{sign}(100) = 1, \text{sign}(-10) = -1, \text{sign}(0) = 0$$



Let's define some vectors:

feature vector (input)

$$x = [1, x_{(1)}, x_{(2)}, \dots, x_{(m)}]^T$$

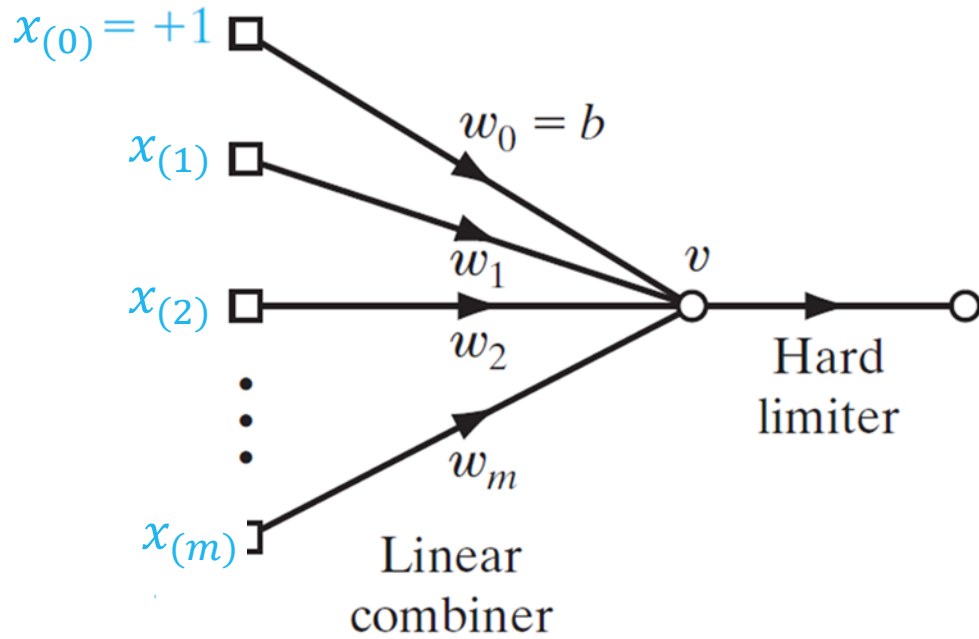
weight vector (parameters)

$$w = [b, w_1, w_2, \dots, w_m]^T$$

Then:

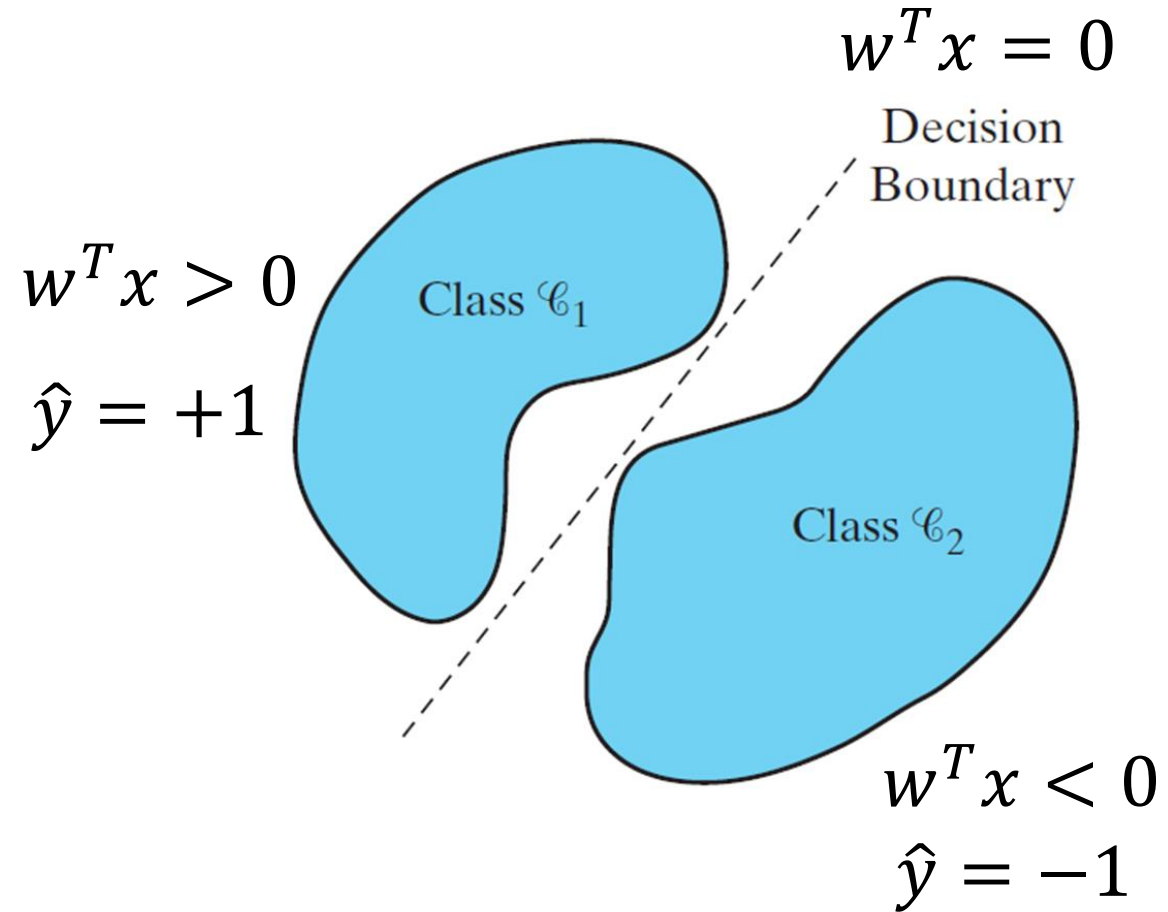
$$\hat{y} = \text{sign}(v), \text{ and } v = w^T x$$

Perceptron



$$\hat{y} = \text{sign}(v)$$
$$v = w^T x$$

It is a linear classifier with a nonlinear activation function $\text{sign}(v)$



Classification Rule:

if $w^T x > 0$, ($\hat{y} = +1$) then assign x to class-1
if $w^T x < 0$, ($\hat{y} = -1$) then assign x to class-2

Algorithm for training a perceptron

- A set of training data points $\{(x_n, y_n), n = 1, \dots, N\}$, $y_n = 1$ or -1
- Initialization: set $w = 0$ (not random)
- for n in $[1, 2, \dots, N]$:
 - compute the output: $\hat{y}_n = \text{sign}(w^T x_n)$
 - update the weight: $w \leftarrow w + \eta(y_n - \hat{y}_n)x_n$, $0 < \eta < 1$
 - if classification is wrong ($y_n \neq \hat{y}_n$) and $x_n \neq 0$ then w will be modified
- Repeat the above operations until convergence
- Perceptron Convergence Theorem: w will become stable after a finite number of iterations if the data points are linearly separable.
- The algorithm is not based on the gradient of any loss function.

Algorithm for training a perceptron

- The algorithm is not based on gradient of any loss function.

compute the output: $\hat{y}_n = \text{sign}(w^T x_n)$

update the weight: $w \leftarrow w + \eta(y_n - \hat{y}_n)x_n, \quad 0 < \eta < 1$

- Rough analysis, assuming classification is wrong for the data point x_n

$w^T x_n$ is the projection of w on x_n (assume norm of x_n is 1)

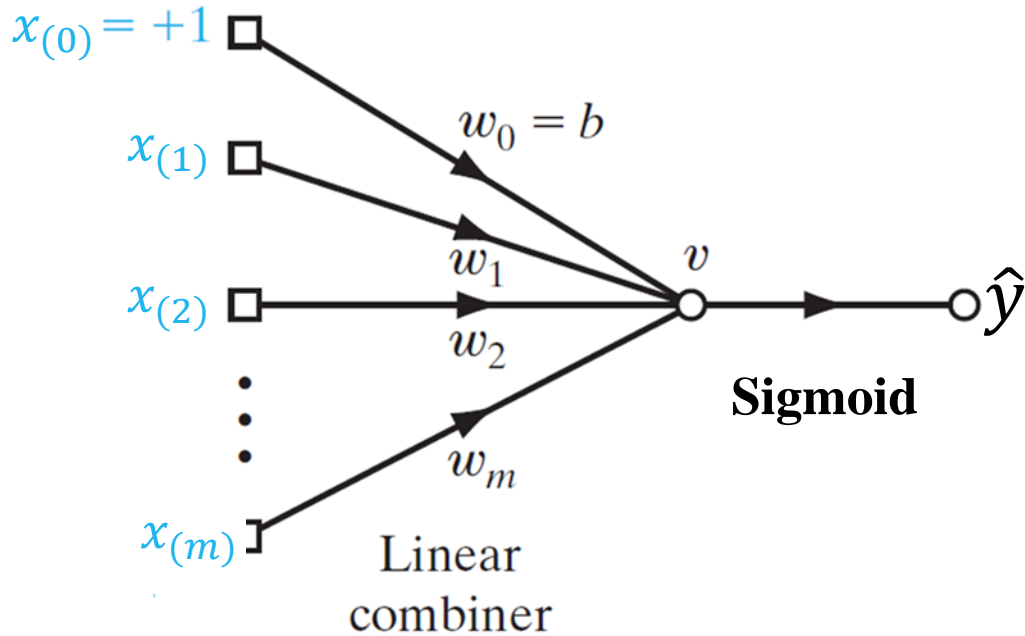
- if $y_n = 1$, then $\hat{y}_n = -1$ and $w \leftarrow w + 2\eta x_n$

w moves closer to x_n , then $w^T x_n$ becomes larger

- if $y_n = -1$ then $\hat{y}_n = 1$ and $w \leftarrow w - 2\eta x_n$

w moves away from x_n , then $w^T x_n$ becomes smaller

Logistic Regression Classifier: an enhanced perceptron



$$x = [1, x_{(1)}, x_{(2)}, \dots, x_{(m)}]^T$$

$$w = [b, w_1, w_2, \dots, w_m]^T$$

- Replace the sign function with Sigmoid

$$\hat{y} = s(v) = \frac{1}{1 + e^{-v}}$$

$$v = w^T x$$

- Loss function: binary cross entropy

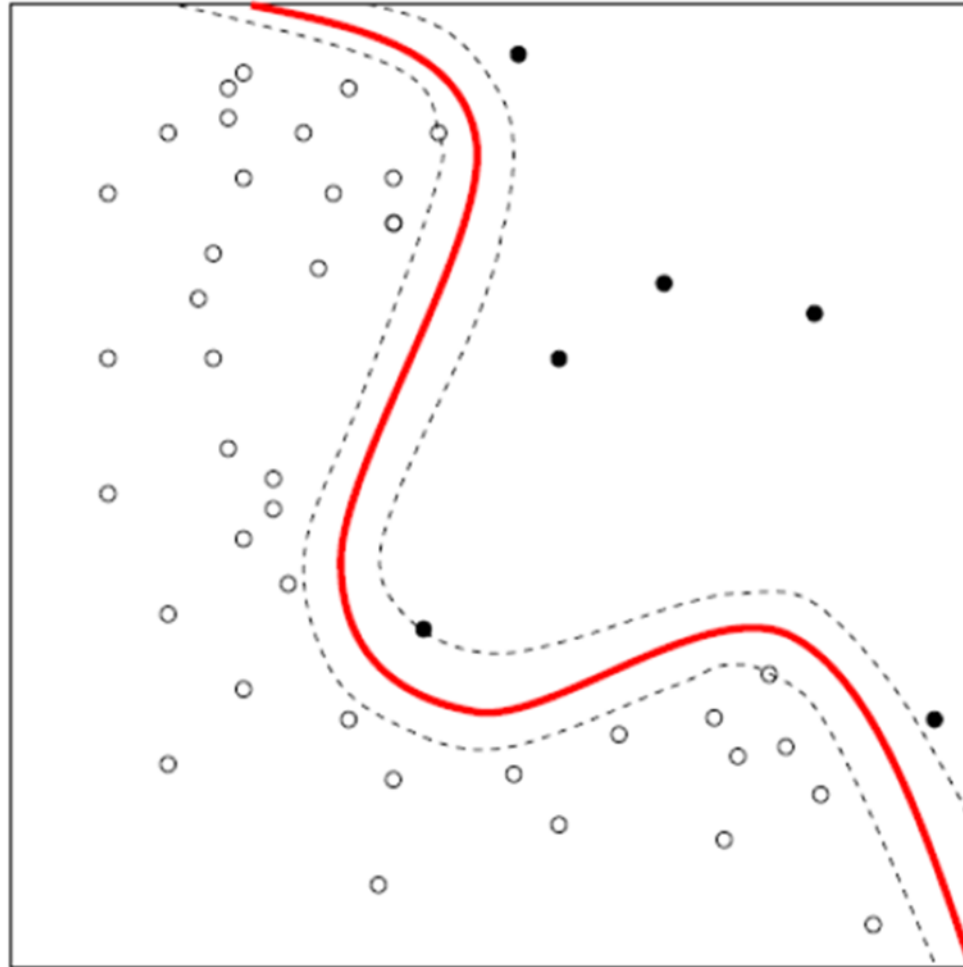
$$L(w) = -\frac{1}{N} \sum_{n=1}^N (y_n \log(\hat{y}_n) + (1 - y_n) \log(1 - \hat{y}_n))$$

- Train the classifier on a dataset using gradient descent

Initialization: $w \leftarrow \text{random}$

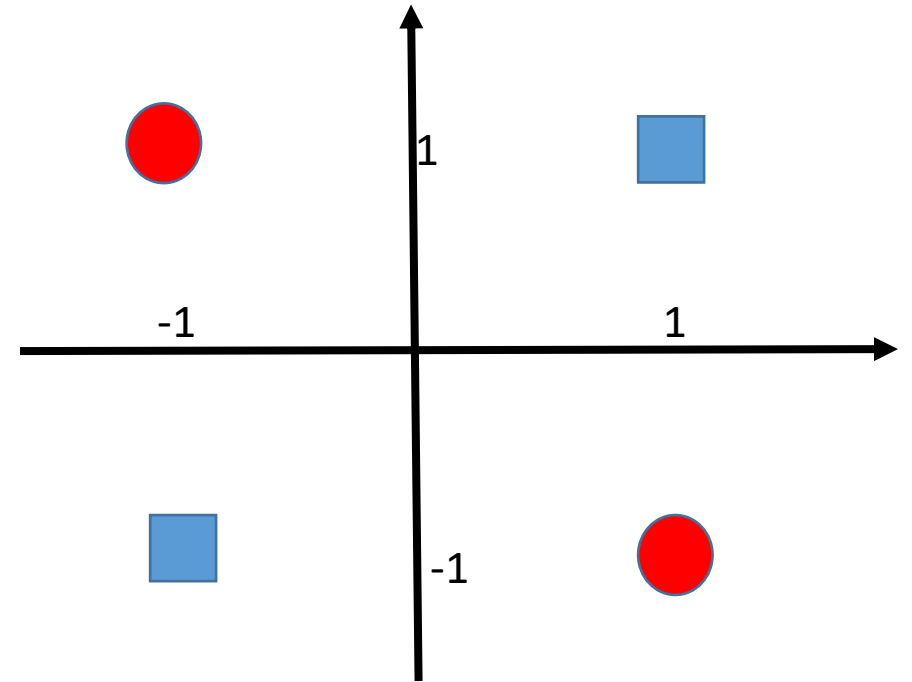
$$\text{Update: } w \leftarrow w - \eta \frac{\partial L}{\partial w}$$

Limitation of Perceptron: it is a linear classifier
but, we may need a **nonlinear decision boundary**



Limitation of Perceptron: it is a linear classifier but, data points may NOT be linearly separable

Input vector \mathbf{x}	Desired response
$(-1, -1)$ ■	-1
$(-1, +1)$ ●	+1
$(+1, -1)$ ●	+1
$(+1, +1)$ ■	-1

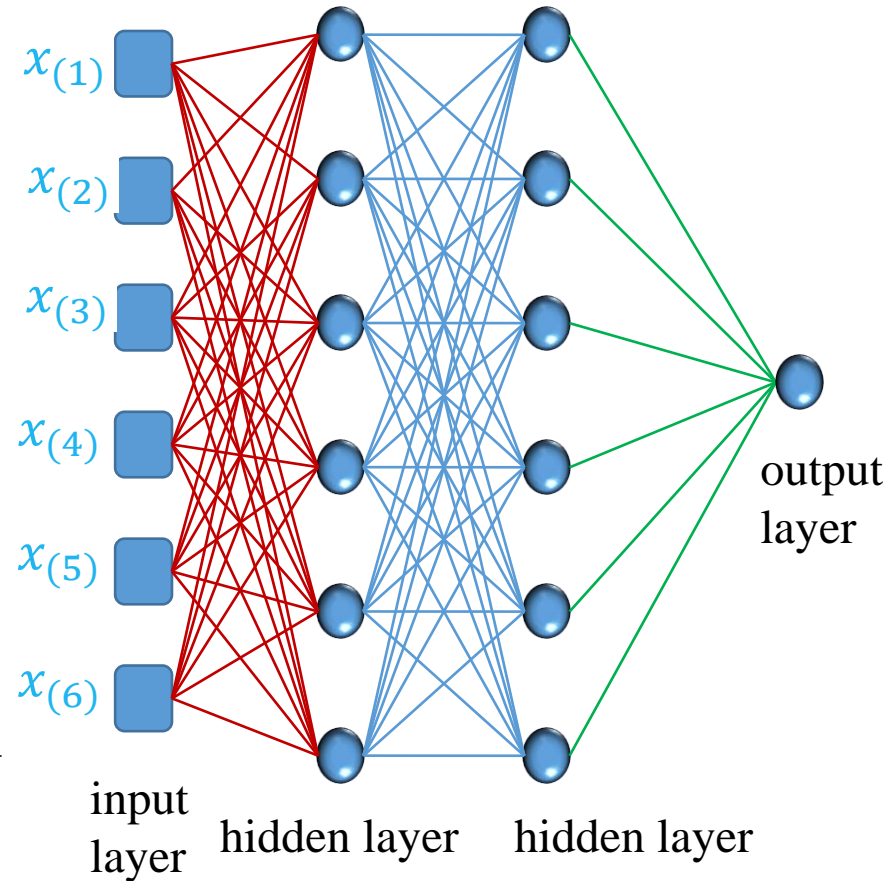


The four points are Not linearly separable

A set of connected perceptrons is a neural network

medical diagnosis

Input:
feature vector
of a patient

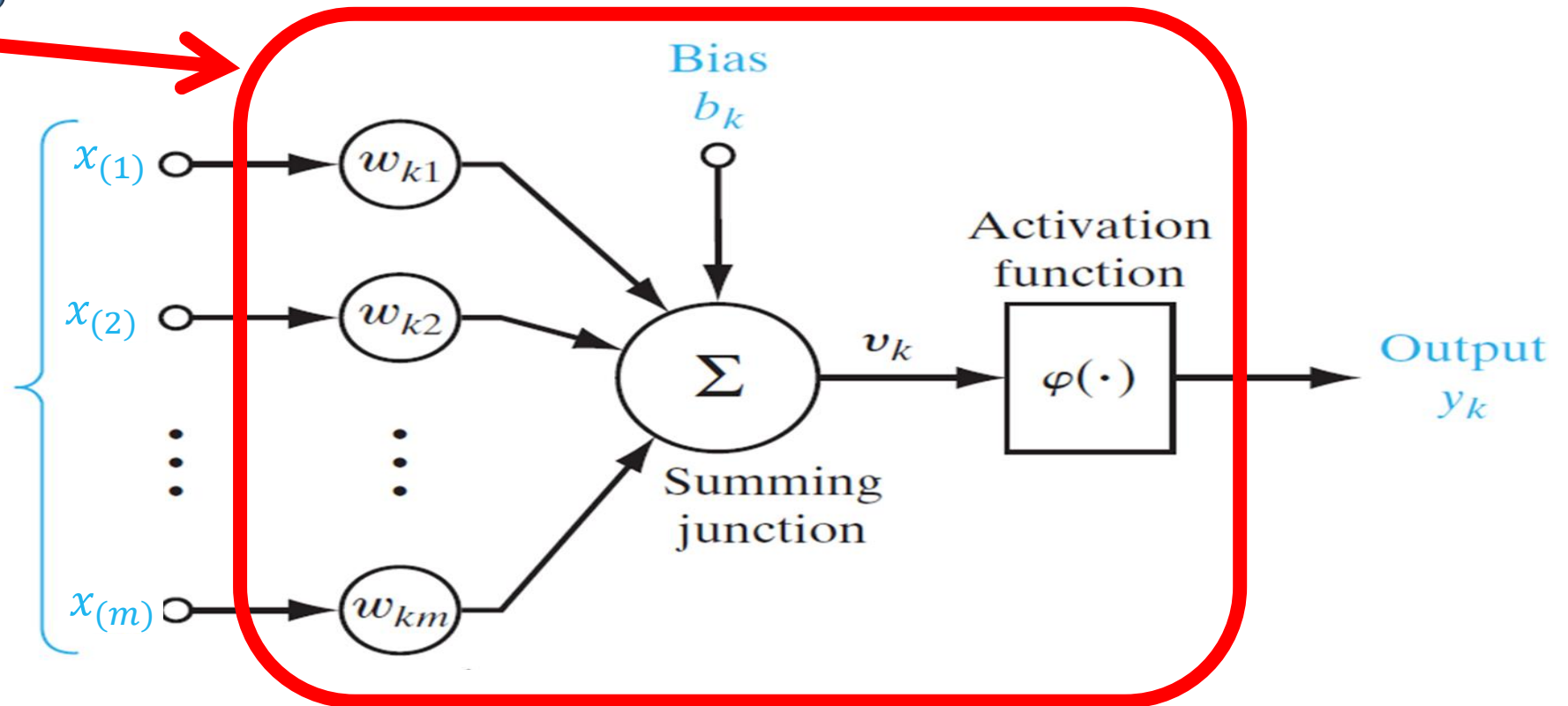


1: disease
0: normal

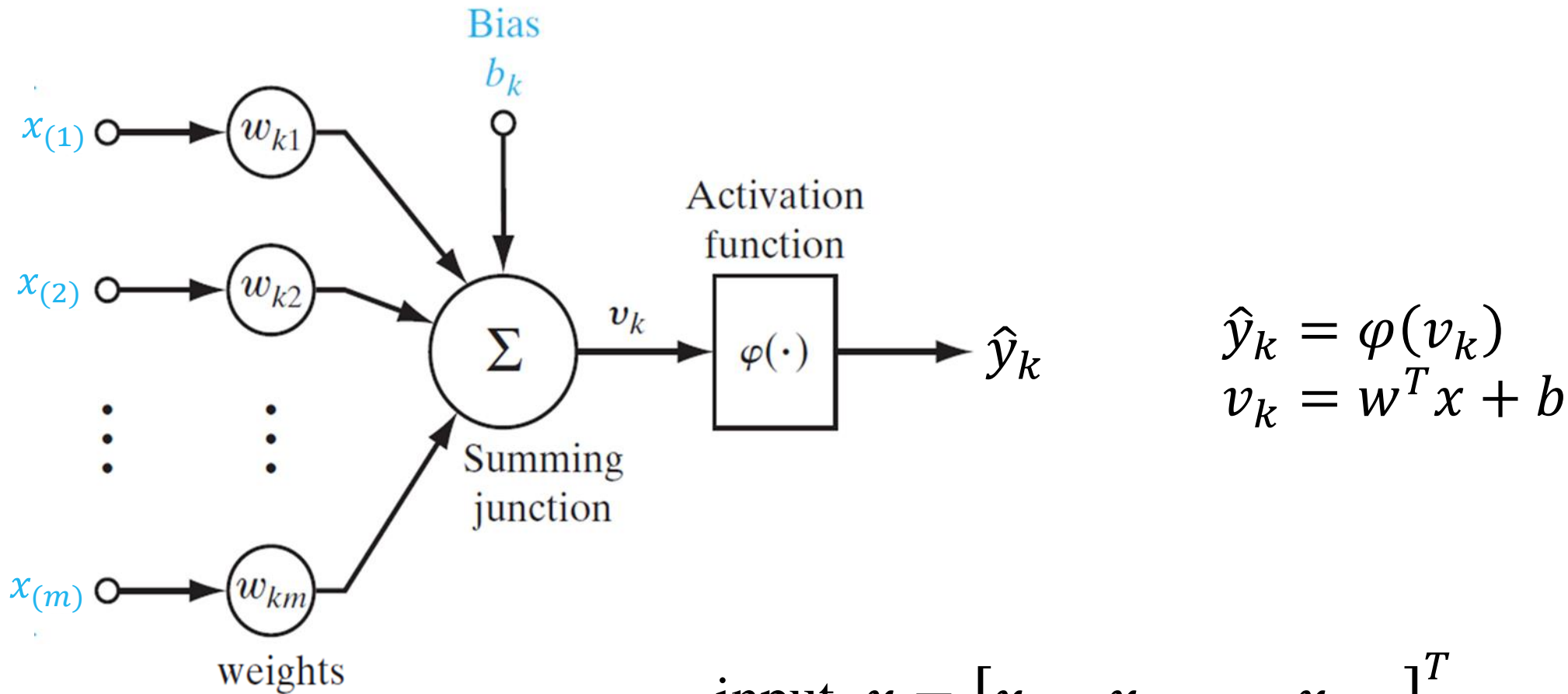
This network is also called
Multi-Layer Perceptron
(MLP)



One Unit (Perceptron)



Unit in a Network: input, weight, bias, activation, output



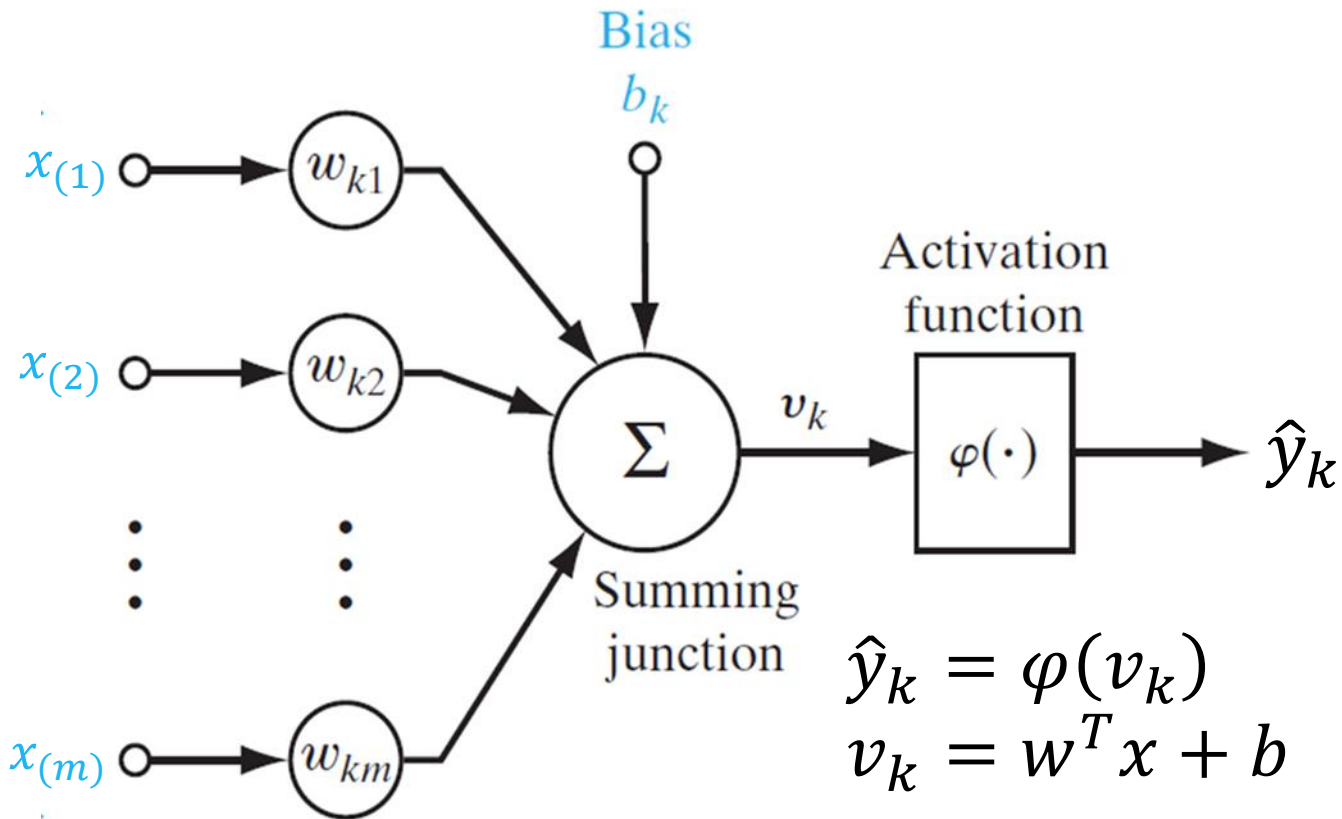
$$\hat{y}_k = \varphi(v_k)$$
$$v_k = w^T x + b$$

$$\text{input } x = [x_{(1)}, x_{(2)}, \dots, x_{(m)}]^T$$

$$\text{weight vector } w = [w_{k1}, w_{k2}, \dots, w_{km}]^T$$

k is the index of the unit

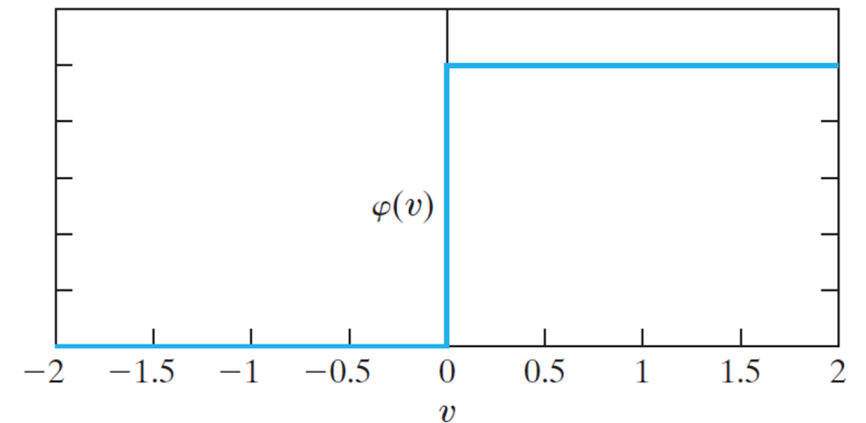
Unit in a Network: the activation function



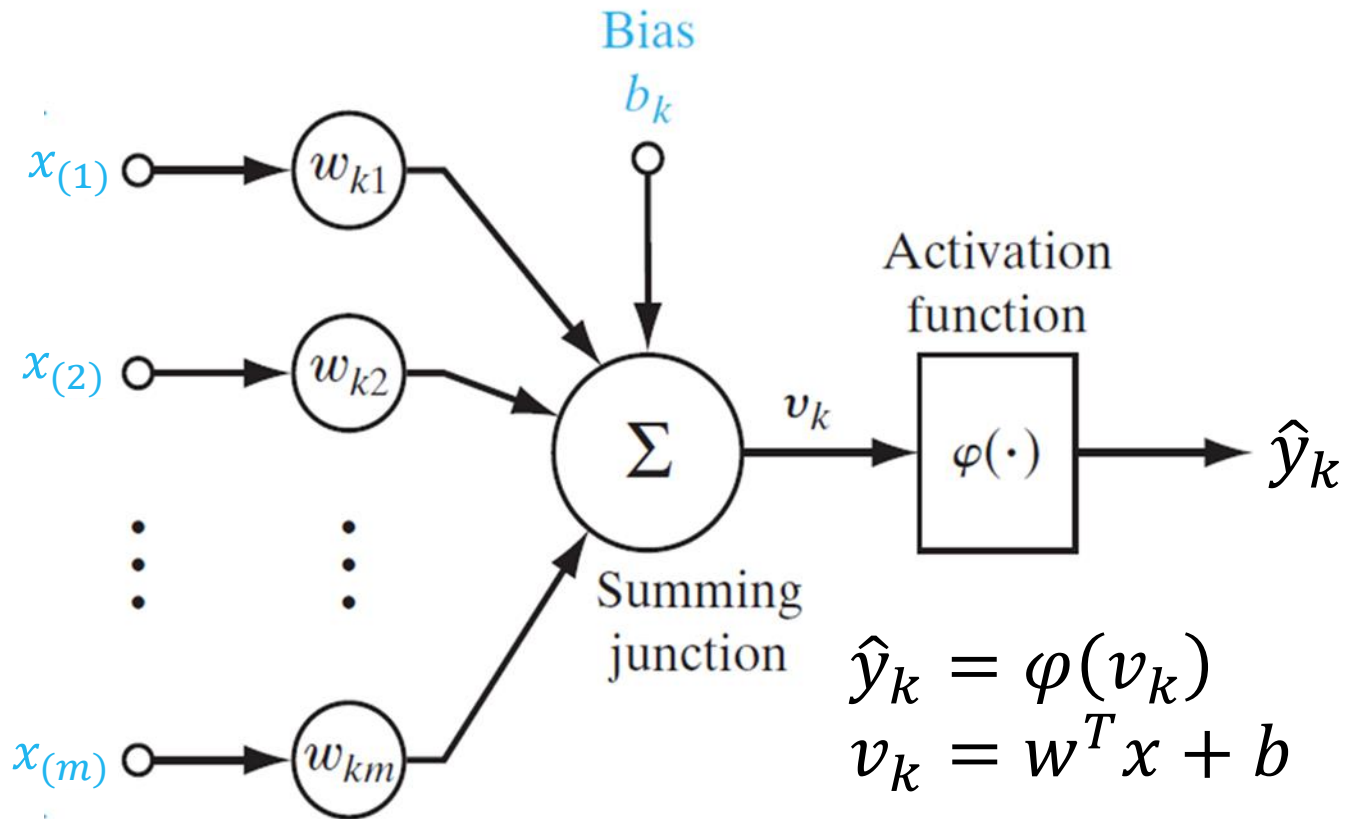
k is the index of the unit in the layer

Threshold function

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$



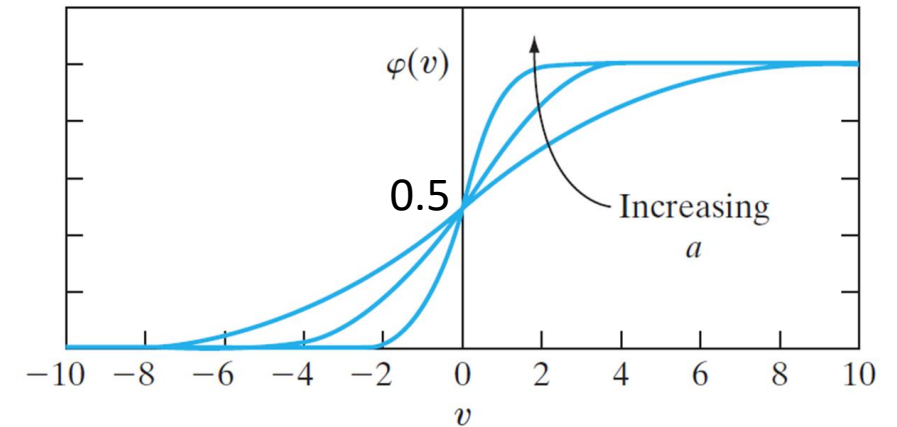
Unit in a Network: the activation function



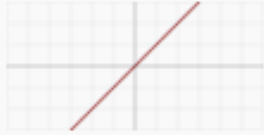

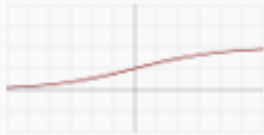
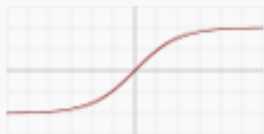
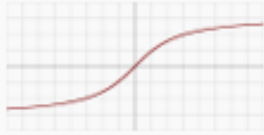
k is the index of the unit in the layer

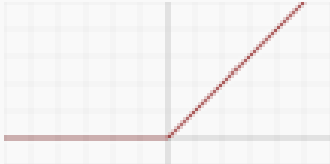
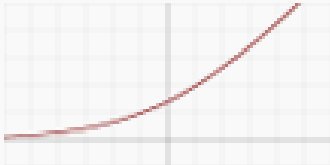
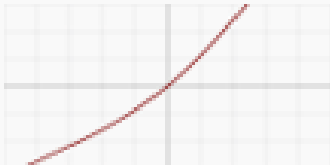
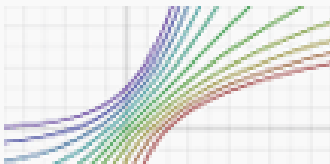
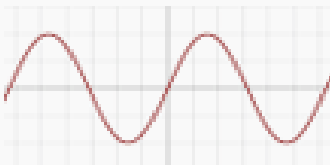
Sigmoid function

$$\varphi(v) = \frac{1}{1 + \exp(-av)}$$

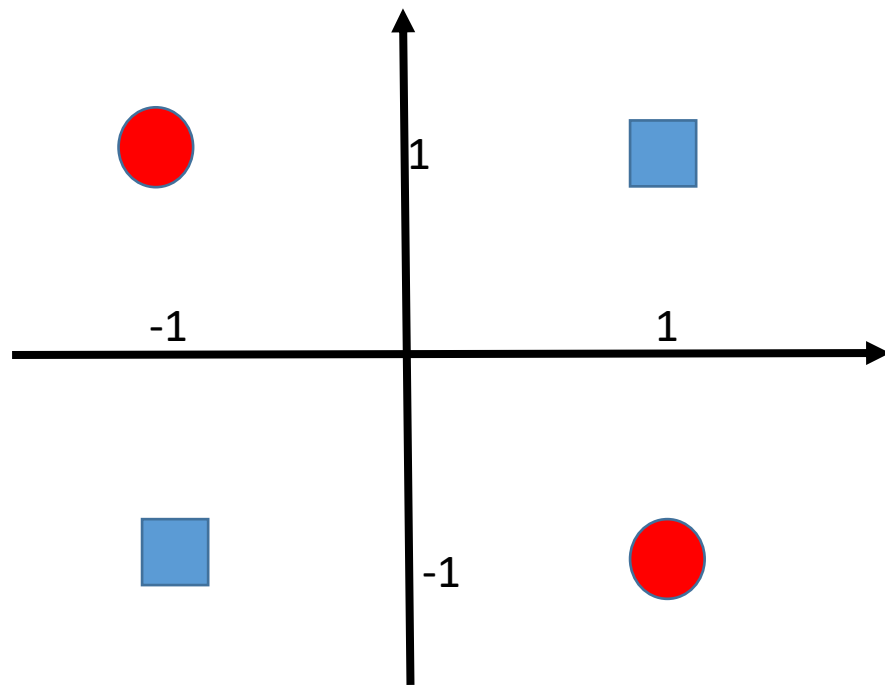






More Activation Functions, Notation: $f(x) \sim \varphi(v)$

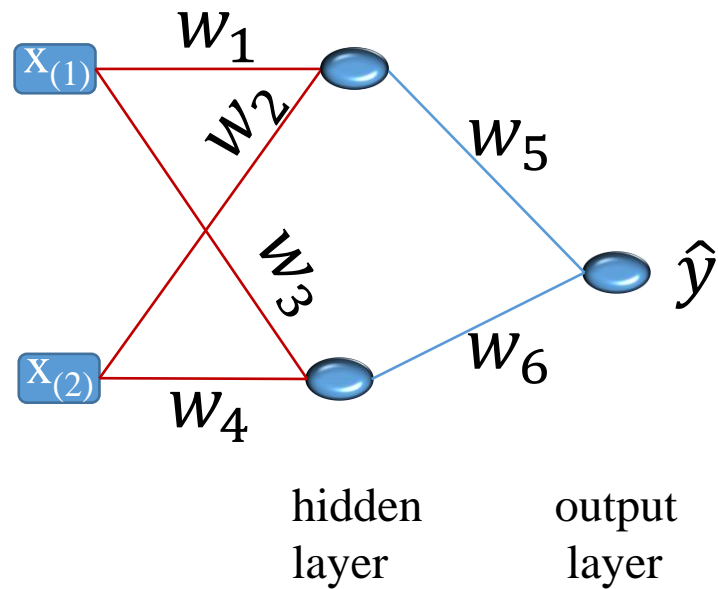
	Name	Plot	Equation	Derivative
Linear	Identity		$f(x) = x$	$f'(x) = 1$
	Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Sigmoid	Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
	<u>TanH</u>		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
	<u>ArcTan</u>		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$

Rectified Linear (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
<u>SoftPlus</u>		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$
Bent identity		$f(x) = \frac{\sqrt{x^2 + 1} - 1}{2} + x$	$f'(x) = \frac{x}{2\sqrt{x^2 + 1}} + 1$
<u>SoftExponential</u>		$f(\alpha, x) = \begin{cases} -\frac{\log_e(1 - \alpha(x + \alpha))}{\alpha} & \text{for } \alpha < 0 \\ x & \text{for } \alpha = 0 \\ \frac{e^{\alpha x} - 1}{\alpha} + \alpha & \text{for } \alpha > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \frac{1}{1 - \alpha(\alpha + x)} & \text{for } \alpha < 0 \\ e^{\alpha x} & \text{for } \alpha \geq 0 \end{cases}$
Sinusoid		$f(x) = \sin(x)$	$f'(x) = \cos(x)$

the XOR problem



Input vector x	Desired response y
$(-1, -1)$ 	-1
$(-1, +1)$ 	+1
$(+1, -1)$ 	+1
$(+1, +1)$ 	-1



This neural network can solve the XOR problem
activation function is sign

Expressive Capabilities of Neural Networks $\hat{y} = f(x)$

- Boolean functions:
 - Every Boolean function can be represented by a network with a single hidden layer
 - It might require a very large number of units in the hidden layer
- Continuous functions: (**Universal Approximation Theorem**)
 - Every bounded continuous function can be approximated with arbitrary small error, by a network with one hidden layer
 - Activation functions need to be locally bounded and piecewise continuous
- Deep network vs shallow network

A continuous function can be approximated by a deep network or a shallow network. The deep network usually uses less number of units.

Convolutional Neural Networks

- The most successful and powerful deep neural networks are convolutional neural networks - CNNs.
- CNNs are mainly used for signal and image related applications (anything that can be considered as signal/image, e.g., a chess board)
- The state-art-of CNNs have approached human performance
 - ECG signal analysis
 - Face recognition
 - Pedestrian detection and tracking
 - Image segmentation (a.k.a semantic segmentation)
 - Image-based medical diagnosis
- The basic operation in CNNs is convolution (cross-correlation)

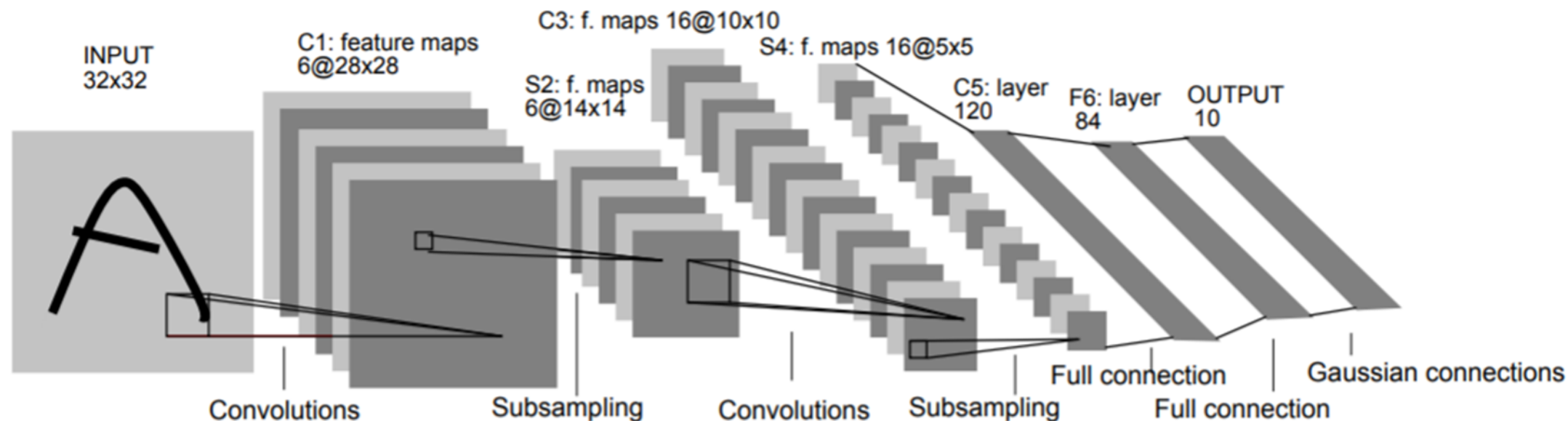
Applications of CNNs

PROC. OF THE IEEE, NOVEMBER 1998

1

Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner



2012

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

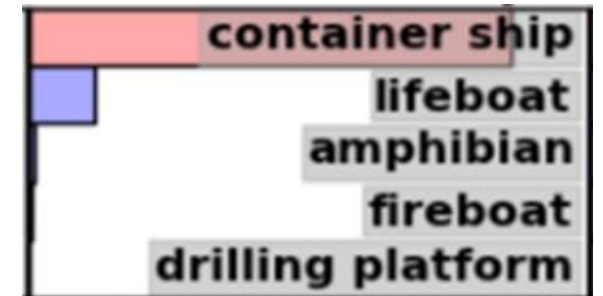
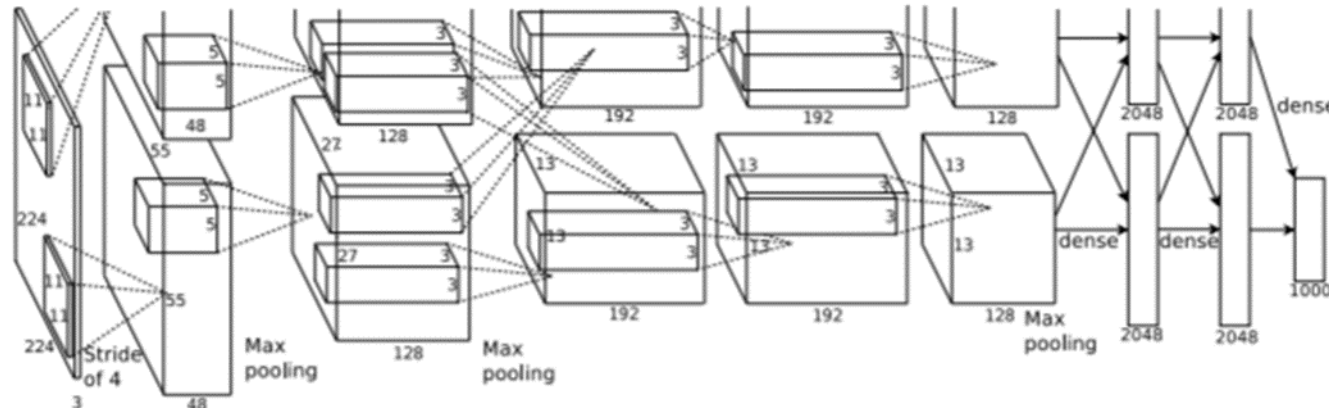
Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

AlexNet

A probability distribution



container ship



Model	Top-1	Top-5
<i>Sparse coding</i> [2]	47.1%	28.2%
<i>SIFT + FVs</i> [24]	45.7%	25.7%
CNN	37.5%	17.0%

ImageNet is the name of a very large image dataset (~14 million images)

2016

You Only Look Once: Unified, Real-Time Object Detection

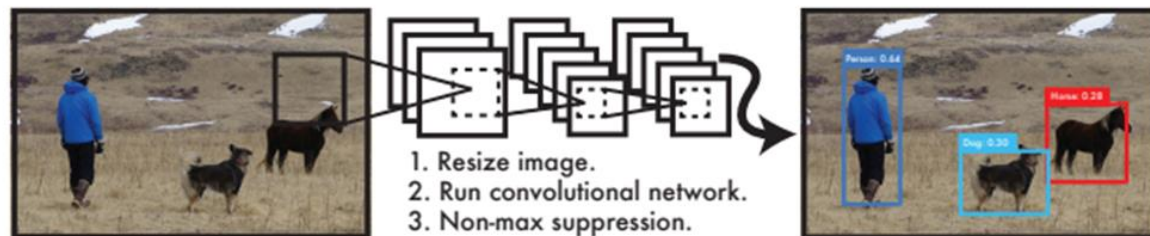
Joseph Redmon^{*}, Santosh Divvala^{*†}, Ross Girshick[¶], Ali Farhadi^{*†}

University of Washington^{*}, Allen Institute for AI[†], Facebook AI Research[¶]

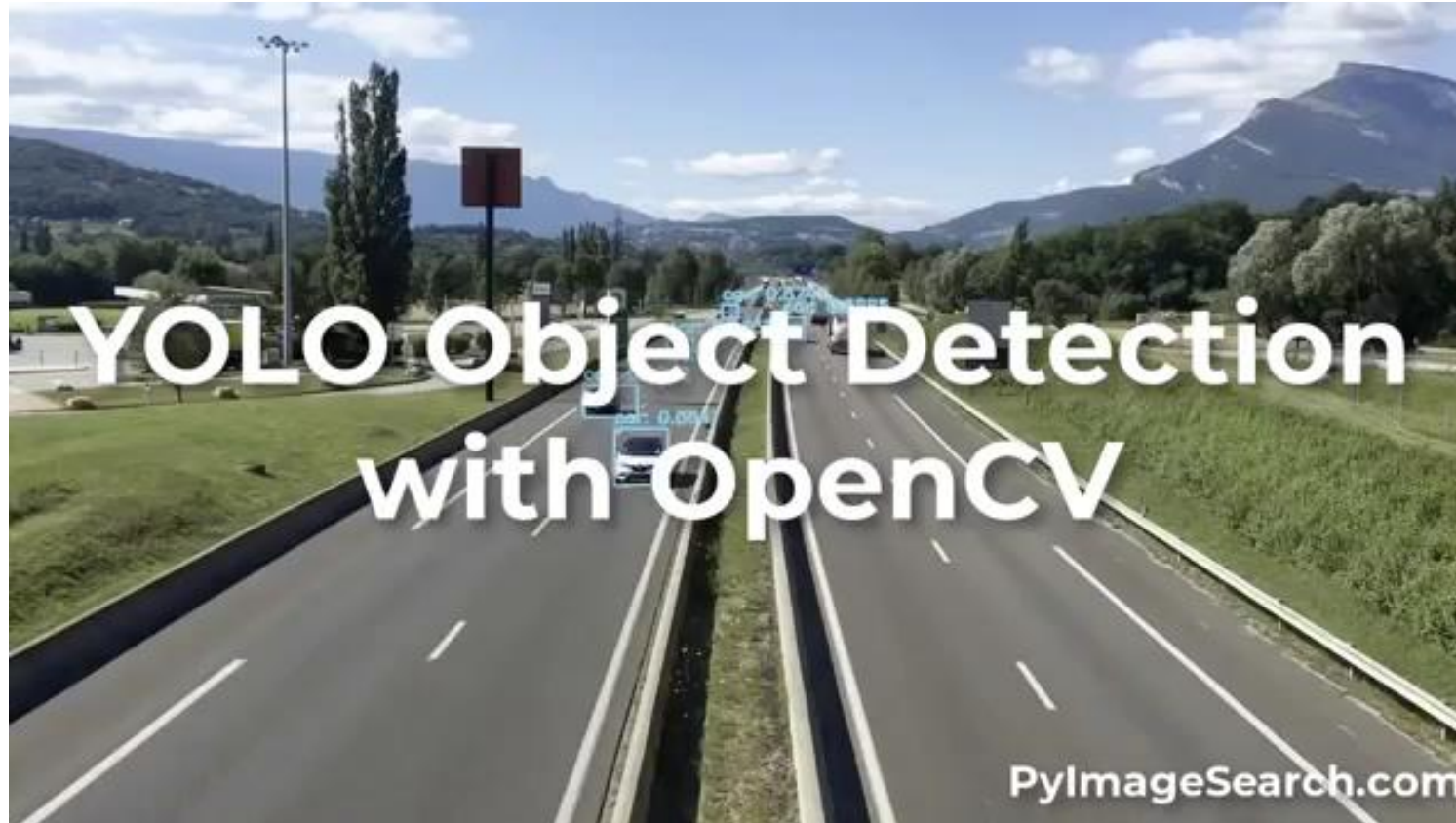
<http://pjreddie.com/yolo/>

Abstract

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a re-



<https://www.youtube.com/watch?v=eelEH2wjvhg&feature=youtu.be>



June 24, 2014

DeepFace: Closing the Gap to Human-Level Performance in Face Verification

Yaniv Taigman

Ming Yang

Marc'Aurelio Ranzato

Lior Wolf

Facebook AI Research
Menlo Park, CA, USA

{yaniv, mingyang, ranzato}@fb.com

Tel Aviv University
Tel Aviv, Israel

wolf@cs.tau.ac.il

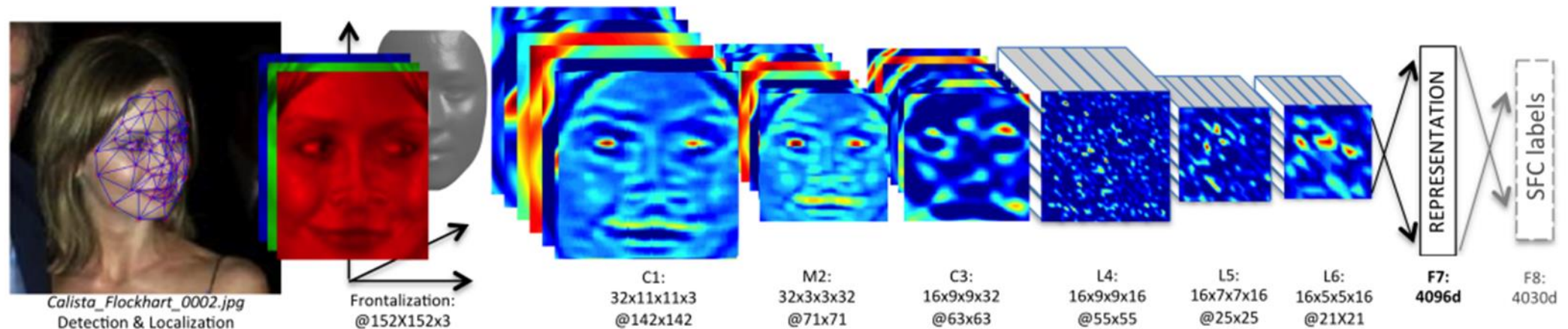


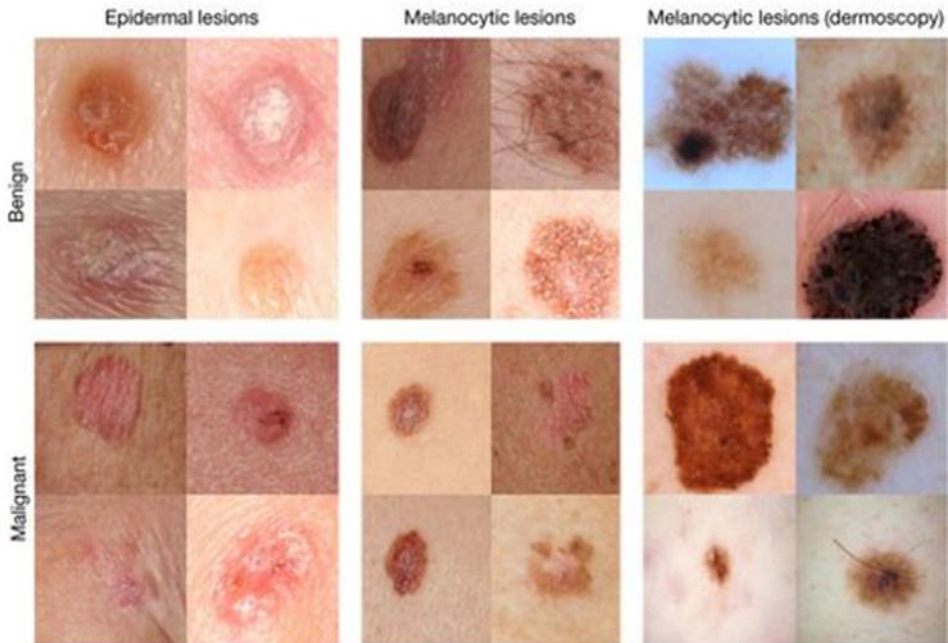
Figure 2. **Outline of the *DeepFace* architecture.** A front-end of a single convolution-pooling-convolution filtering on the rectified input, followed by three locally-connected layers and two fully-connected layers. Colors illustrate feature maps produced at each layer. The net includes more than 120 million parameters, where more than 95% come from the local and fully connected layers.

2017

Dermatologist-level classification of skin cancer with deep neural networks

Andre Esteva ✉, Brett Kuprel ✉, Roberto A. Novoa ✉, Justin Ko, Susan M. Swetter, Helen M. Blau & Sebastian Thrun ✉

Stanford University



CNN

the type of skin lesion:
cancer or benign

Google's Inception v3 CNN (48 layers)

Cardiologist-Level Arrhythmia Detection with Convolutional Neural Networks (34 layers)

Pranav Rajpurkar*

Awni Y. Hannun*

Masoumeh Haghpanahi

Codie Bourn

Andrew Y. Ng

PRANAVSR@CS.STANFORD.EDU

AWNI@CS.STANFORD.EDU

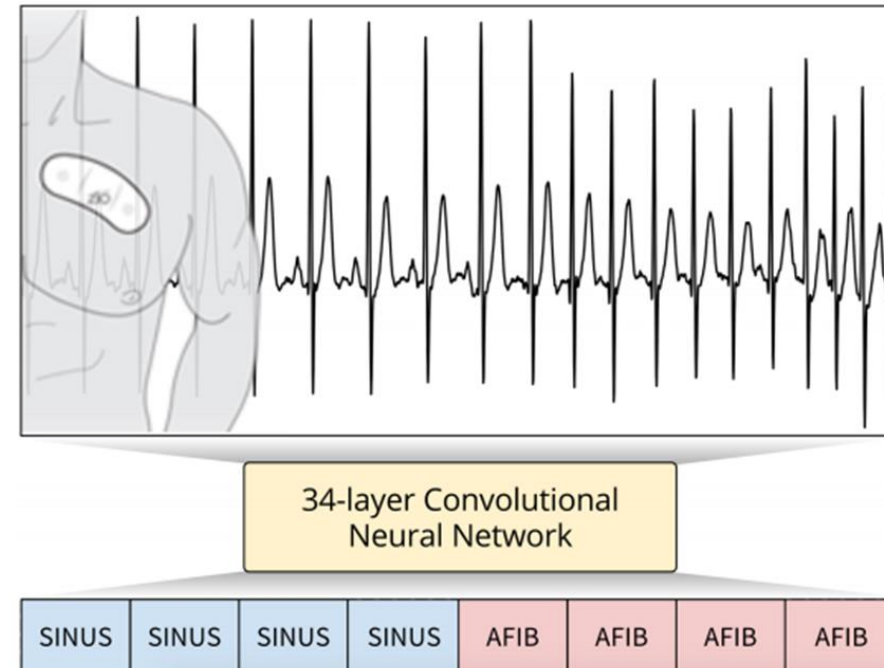
MHAGHPANAHI@IRHYTHMTECH.COM

CBOURN@IRHYTHMTECH.COM

ANG@CS.STANFORD.EDU

Abstract

We develop an algorithm which exceeds the performance of board certified cardiologists in detecting a wide range of heart arrhythmias from electrocardiograms recorded with a single-lead wearable monitor. We build a dataset with more than 500 times the number of unique patients than previously studied corpora. On this dataset, we train a 34-layer convolutional neural network which maps a sequence of ECG samples to a sequence of rhythm classes. Committees of board-certified cardiologists annotate a gold standard test set on which we compare the performance of our model to that of 6 other individual cardiologists.



Google, 2016

JAMA | **Original Investigation** | INNOVATIONS IN HEALTH CARE DELIVERY

Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs



<https://ai.googleblog.com/2016/11/deep-learning-for-detection-of-diabetic.html>

Google's Inception v3 CNN (48 layers)

Google Deepmind, 2018

<https://deepmind.com/blog/moorfields-major-milestone/>

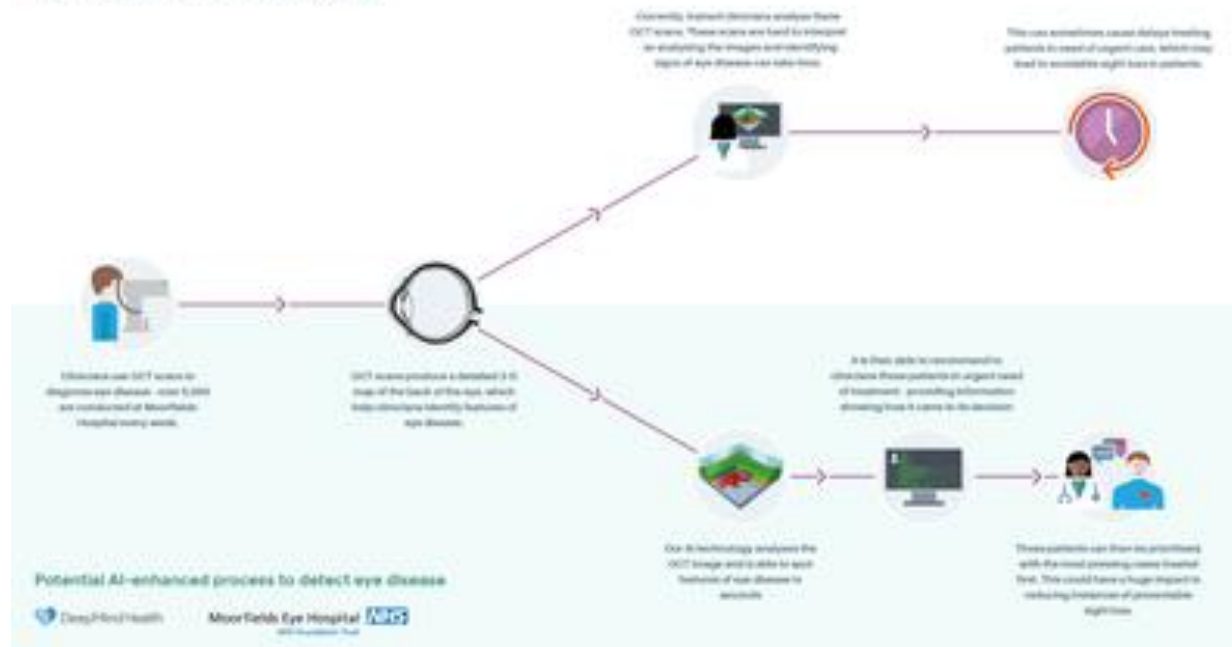
ARTICLES

<https://doi.org/10.1038/s41591-018-0107-6>

nature
medicine

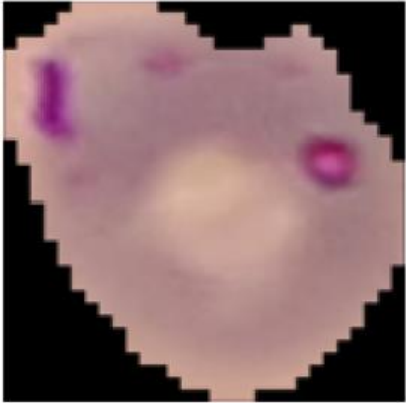
Clinically applicable deep learning for diagnosis and referral in retinal disease

Current process used to detect eye disease

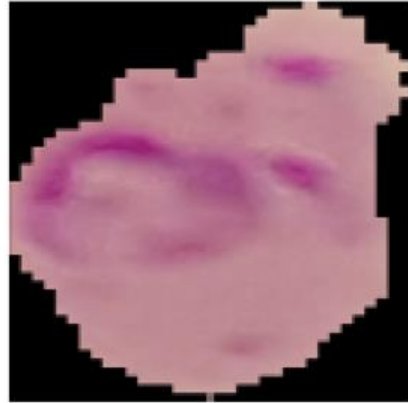


Malaria Parasite Identification in Cell Images

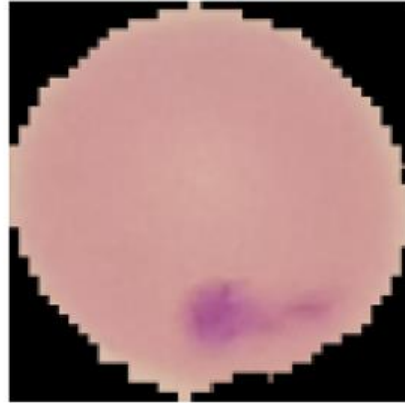
Infected



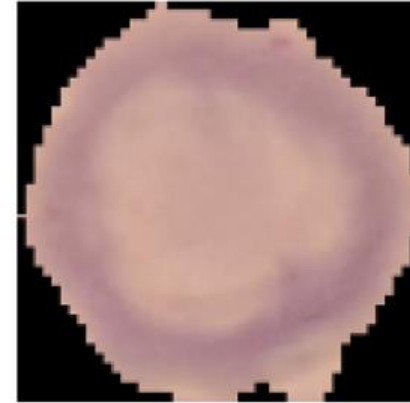
Infected



Infected



Uninfected



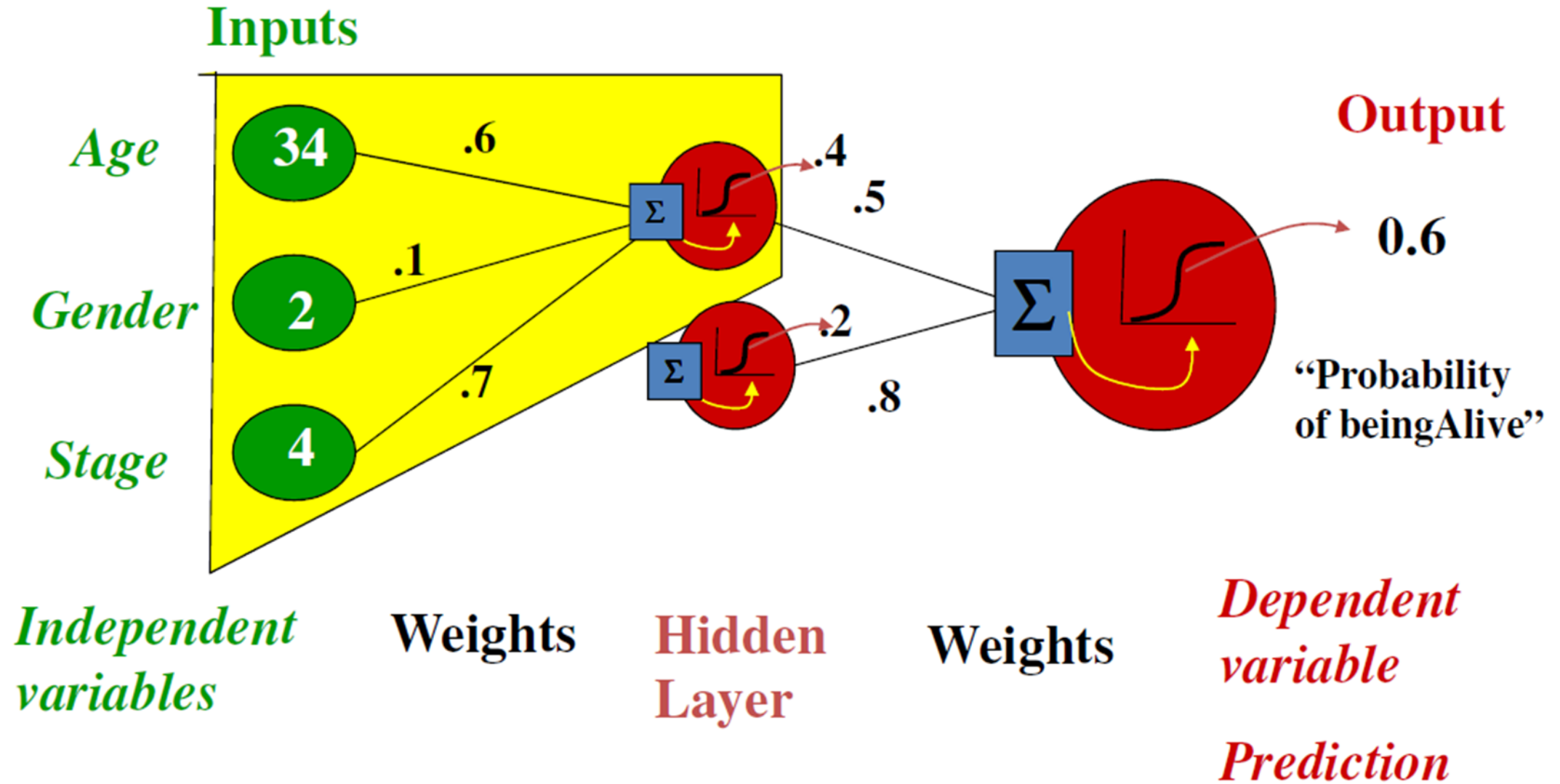
Uninfected



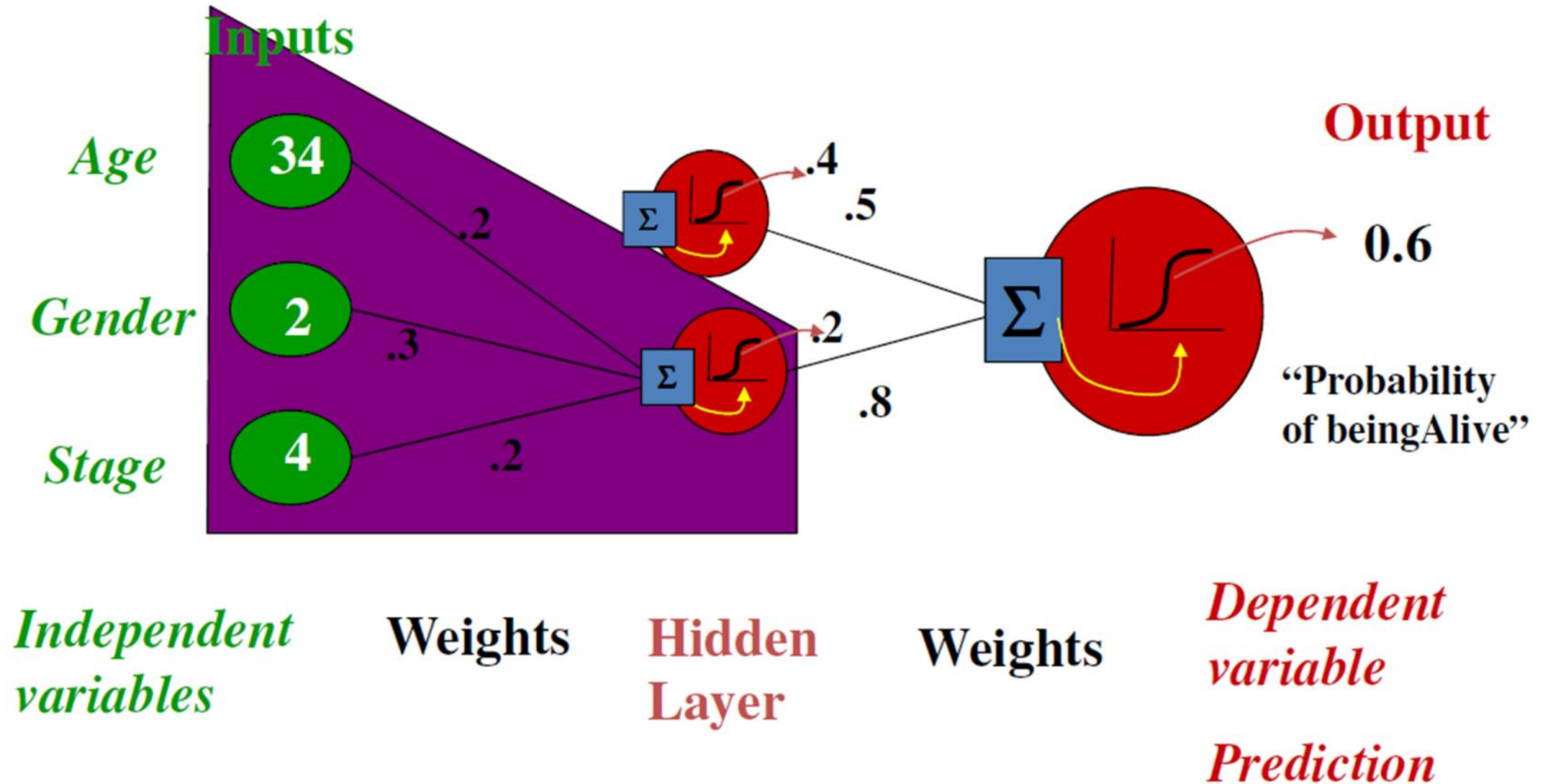
The infected cells look very different from the uninfected cells

The task is to classify the cell images into two classes:
infected vs uninfected

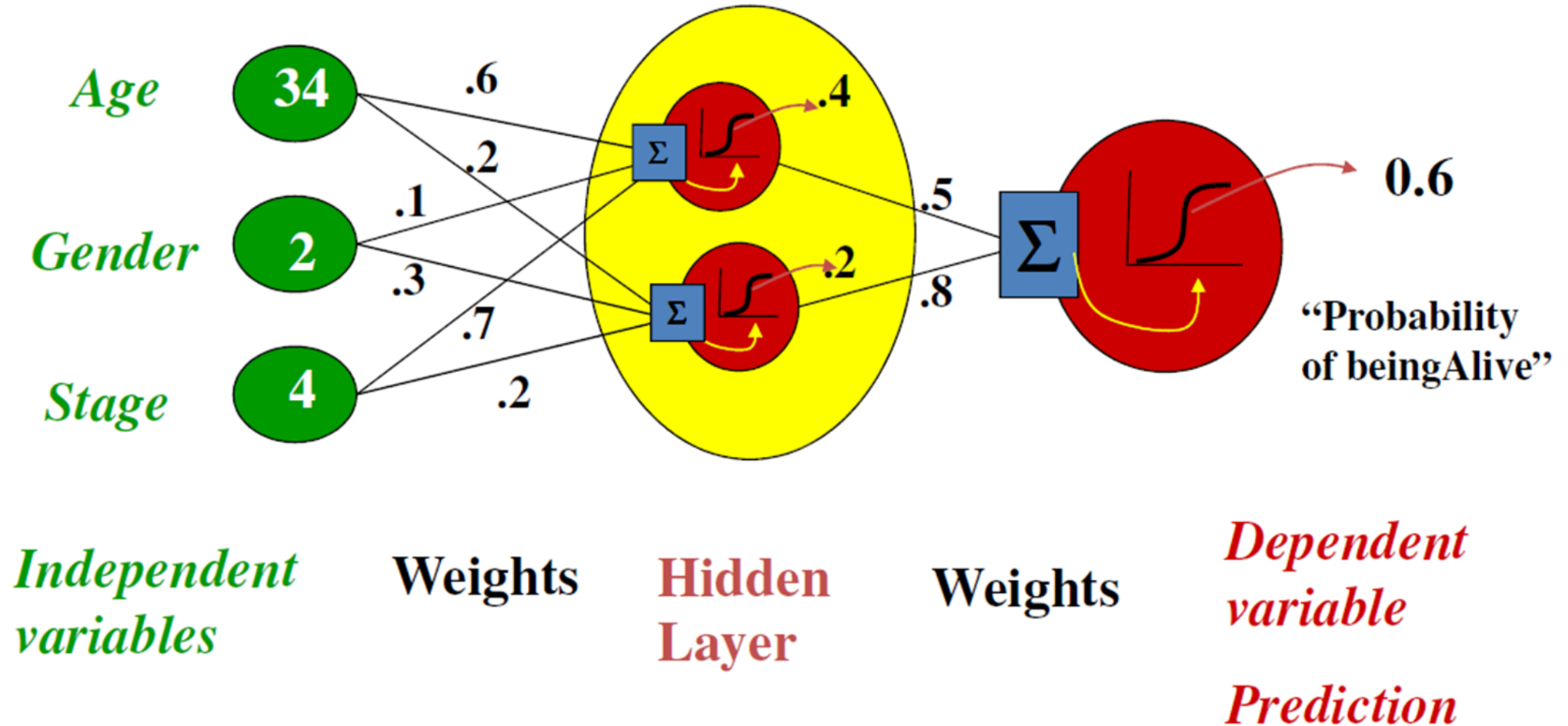
Can we train the units in a network independently ?



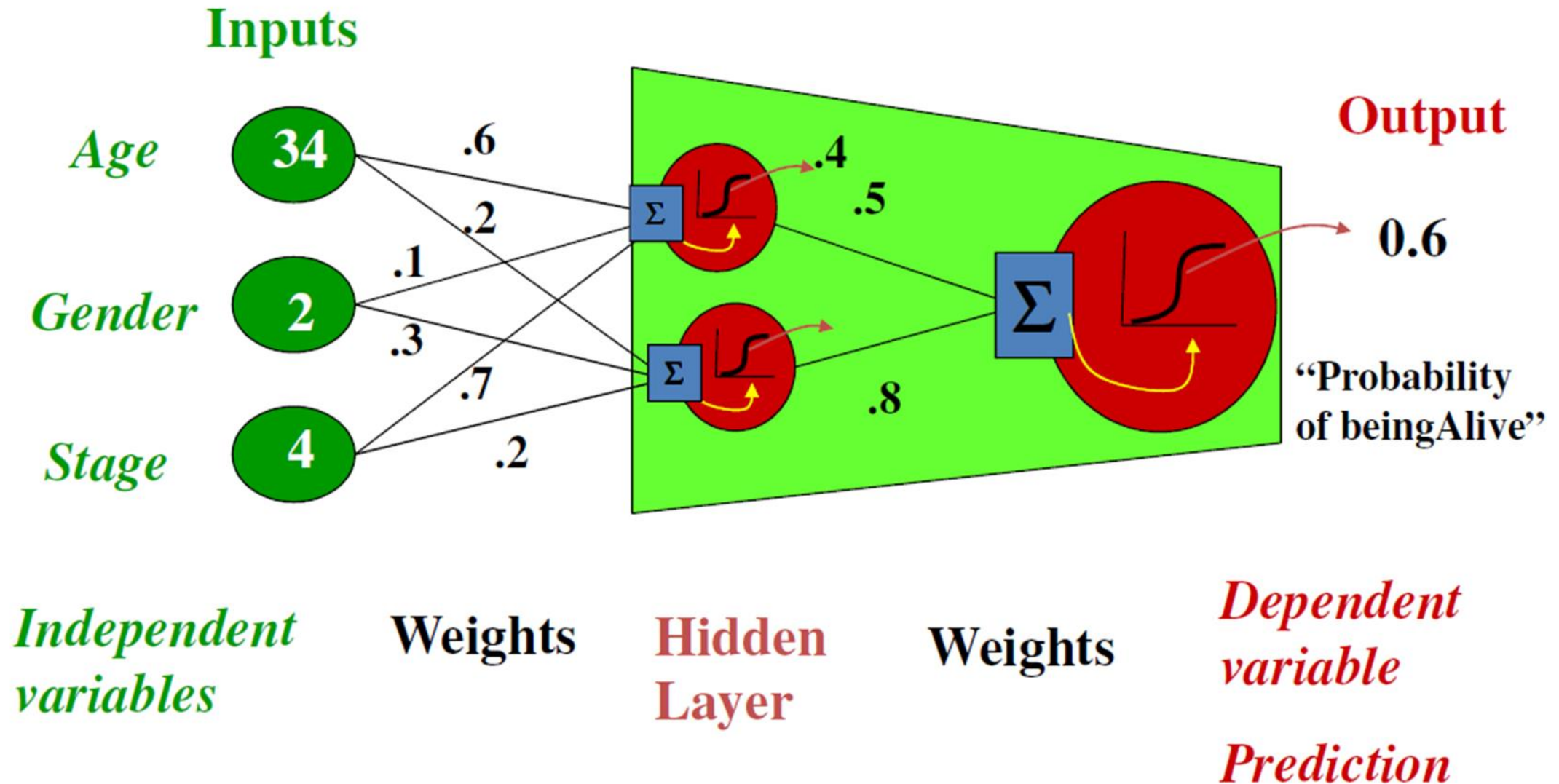
Train the units independently ?



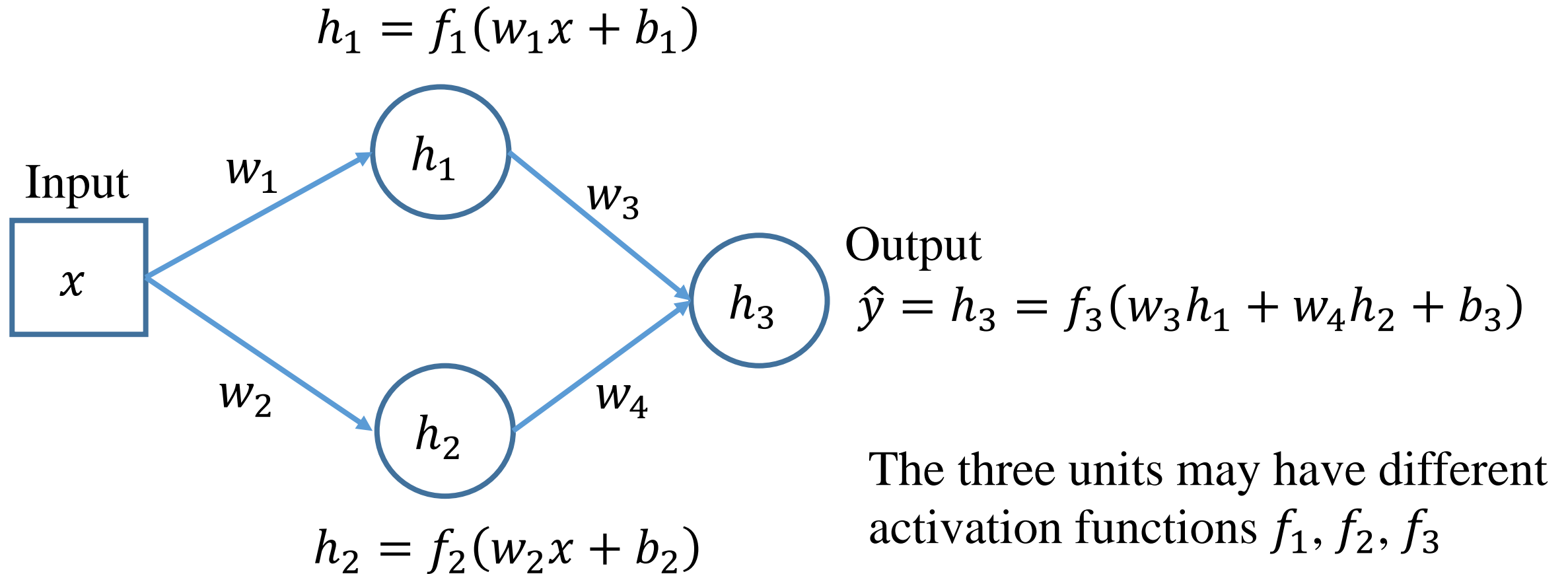
Train the units independently ?



Train the units independently ? Maybe not...
Usually, we do not have target values for hidden units



Let's train a network using gradient descent



$x, w_1, w_2, w_3, w_4, b_1, b_2, b_3, h_1, h_2, h_3$ are scalars

- The task is regression.
- x is a data point
- y is the true target value
- \hat{y} is the prediction/output
- L is the loss of x

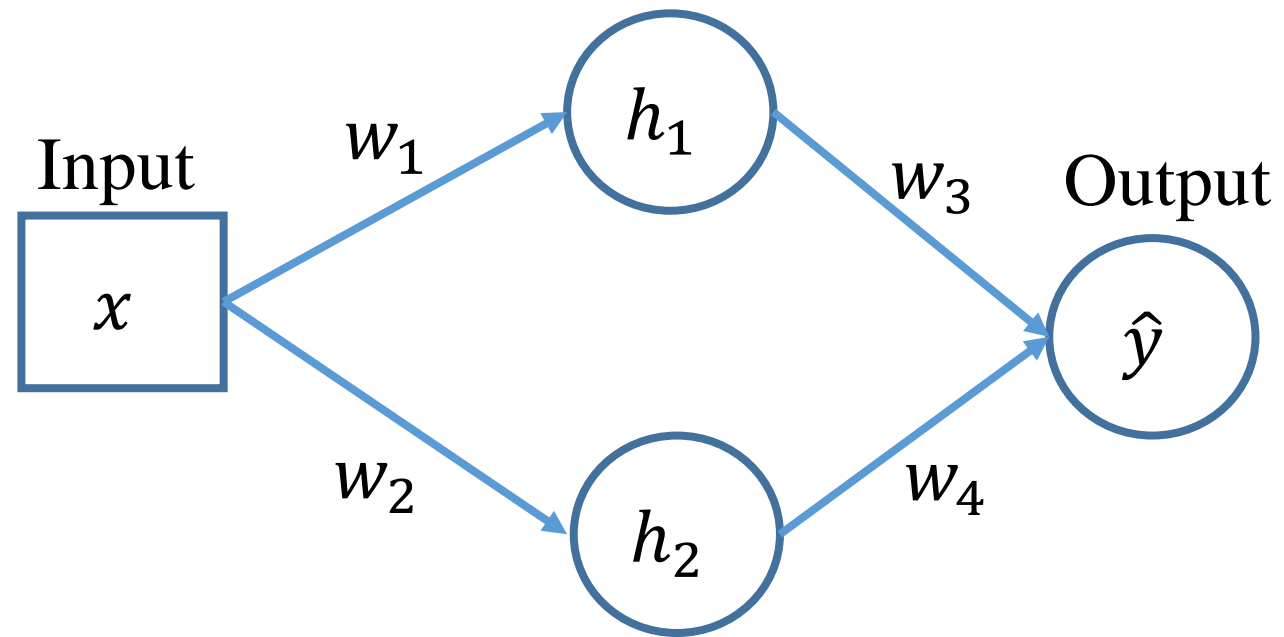
$$L = (\hat{y} - y)^2$$

$$\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$$

Using chain rule of calculus

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_3} = \frac{\partial L}{\partial \hat{y}} f'_3 h_1$$

Similarly, we can get $\frac{\partial L}{\partial w_4}$ and $\frac{\partial L}{\partial b_3}$



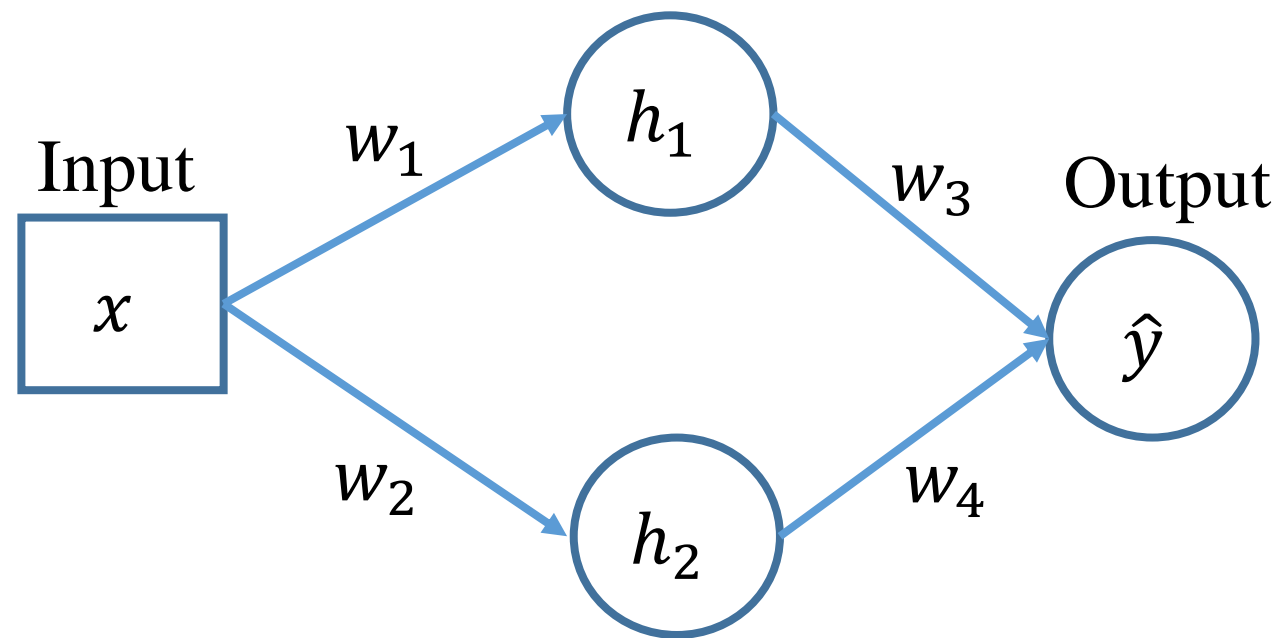
$$h_1 = f_1(w_1x + b_1)$$

$$h_2 = f_2(w_2x + b_2)$$

$$\hat{y} = f_3(w_3h_1 + w_4h_2 + b_3)$$

$$f'_n = \frac{\partial f_n(v)}{\partial v}, n = 1, 2, 3$$

- The task is regression.
- x is a data point
- y is the true target value
- \hat{y} is the prediction/output
- L is the loss of x



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial h_1} \frac{\partial h_1}{\partial w_1} = \frac{\partial L}{\partial h_1} f'_1 x$$

$$\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_1} = \frac{\partial L}{\partial \hat{y}} f'_3 w_3$$

Similarly, we can get $\frac{\partial L}{\partial b_1}$, $\frac{\partial L}{\partial w_2}$ and $\frac{\partial L}{\partial b_2}$

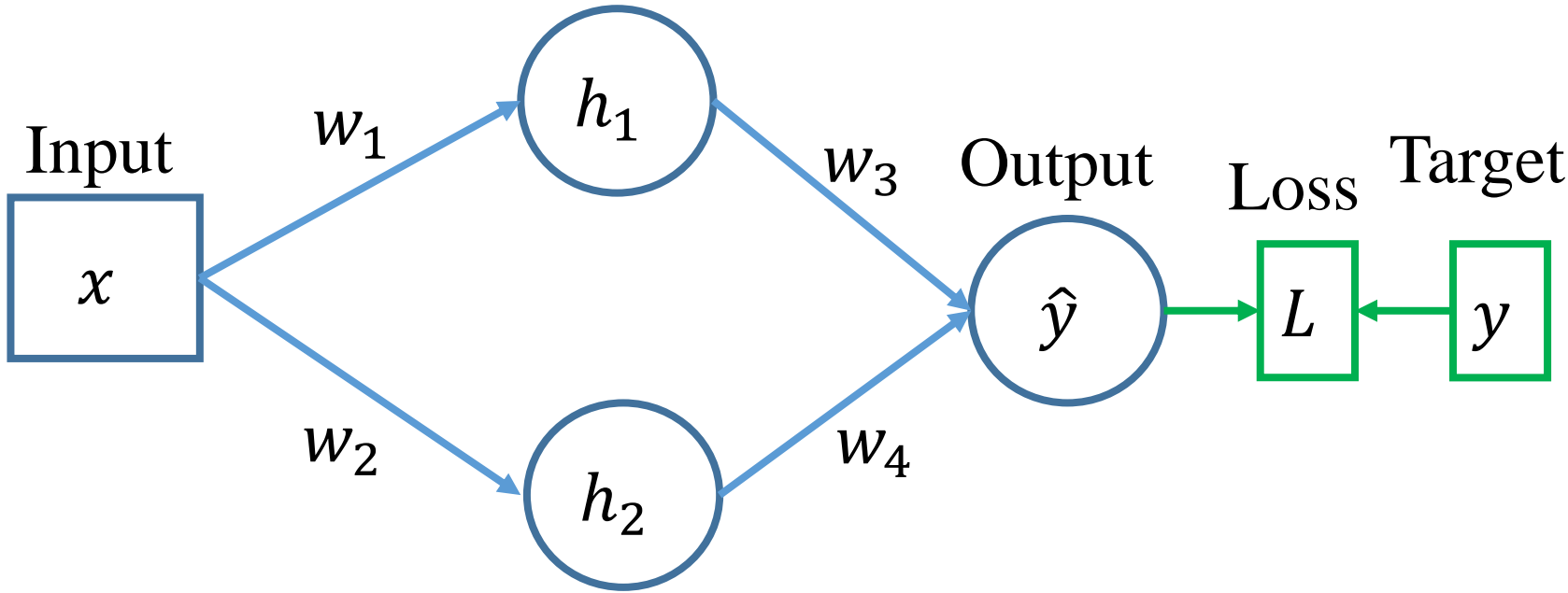
$$h_1 = f_1(w_1 x + b_1)$$

$$h_2 = f_2(w_2 x + b_2)$$

$$\hat{y} = f_3(w_3 h_1 + w_4 h_2 + b_3)$$

$$f'_n = \frac{\partial f_n(v)}{\partial v}, n = 1, 2, 3$$

Backpropagation to get $\frac{\partial L}{\partial w_1}$



Backpropagation: gradient flows from loss towards input

Diagram showing the flow of gradients during backpropagation. The gradient $\frac{\partial L}{\partial \hat{y}}$ flows from the Output node (\hat{y}) to the hidden nodes h_1 and h_2 . The gradient $\frac{\partial L}{\partial h_1}$ flows from h_1 to the input node (x), and the gradient $\frac{\partial L}{\partial w_1}$ flows from h_1 to the weight w_1 .

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial h_1} f'_1 x$$

$$\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial \hat{y}} f'_3 w_3$$

$$\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$$

$$h_1 = f_1(w_1 x + b_1)$$

$$h_2 = f_2(w_2 x + b_2)$$

$$\hat{y} = f_3(w_3 h_1 + w_4 h_2 + b_3)$$

$$f'_n = \frac{\partial f_n(v)}{\partial v}, n = 1, 2, 3$$

A Question in Homework

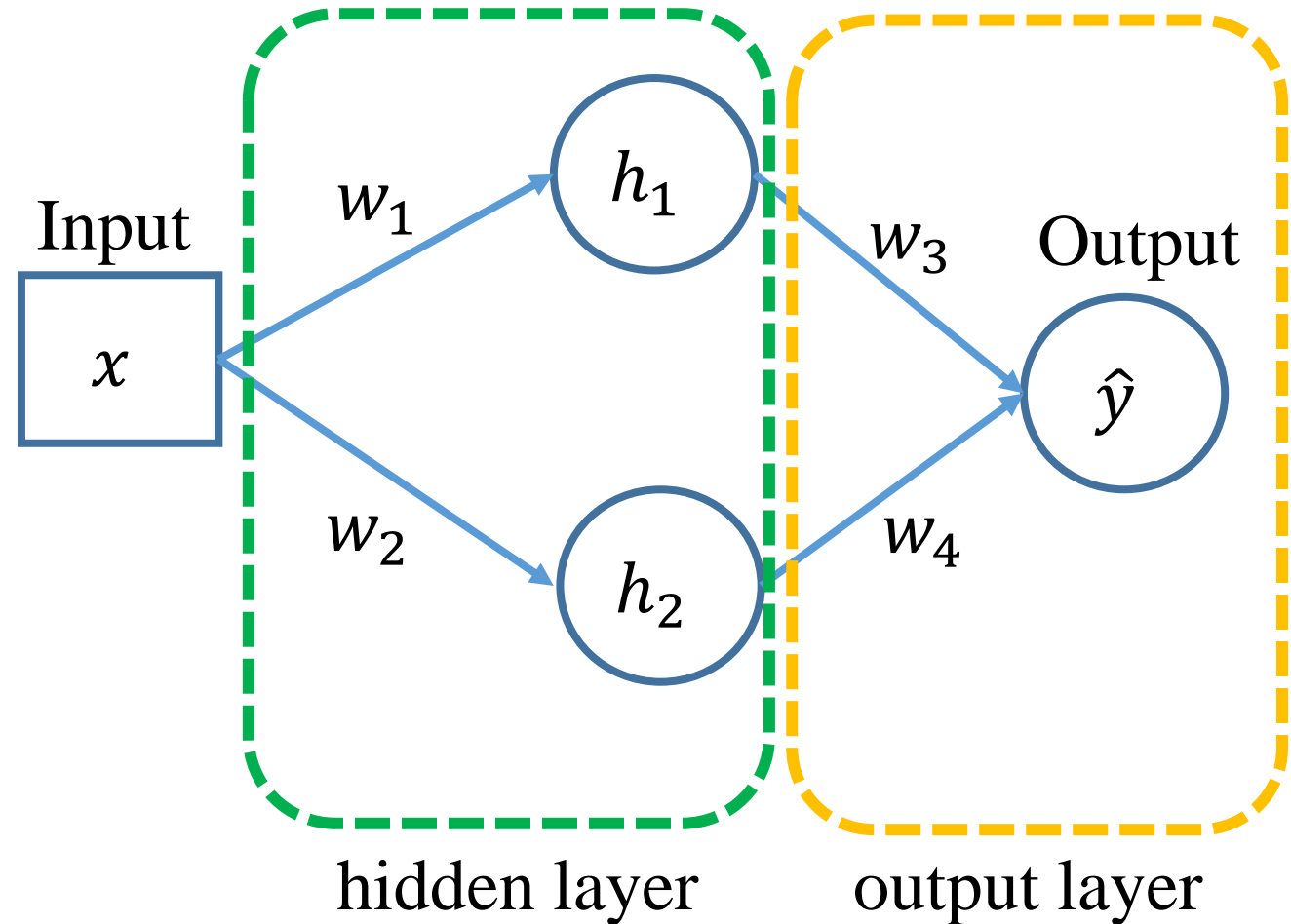
Compute the following derivatives

$$\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \frac{\partial L}{\partial w_3}, \frac{\partial L}{\partial w_4}$$

$$\frac{\partial L}{\partial b_1}, \frac{\partial L}{\partial b_2}, \frac{\partial L}{\partial b_3}$$

$$\frac{\partial L}{\partial x}$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial h_1} \frac{\partial h_1}{\partial x} + \frac{\partial L}{\partial h_2} \frac{\partial h_2}{\partial x}$$



Train a network using **Batch** Gradient Descent

- a training set $\{(x_n, y_n), n = 1, \dots, N\}$, x_n is a data sample, y_n is the class label (for classification) or target value (for regression)
- \hat{y}_n is the output of the network given input x_n
- $L(\hat{y}_n, y_n)$ is the loss of a single data sample x_n (e.g. MSE, Cross-Entropy)
- Initialization: initialize the network parameters using random numbers
- Forward Pass: compute \hat{y}_n for every data point x_n (n is from 1 to N)
- Backward Pass: compute derivative of the loss with respect to each parameter
- Update parameters: $w_k \leftarrow w_k - \eta \frac{\partial L}{\partial w_k}$, **total loss**: $L = \frac{1}{N} \sum_{n=1}^N L(\hat{y}_n, y_n)$
- Repeat the process (forward-backward-update) many times until convergence

Train a network using **Stochastic** Gradient Descent

- Initialization: initialize the network parameters using random numbers
- Forward Pass: **randomly** select **one** data point x and compute \hat{y}
- Backward Pass: compute derivative of the loss with respect to each parameter
- Update parameters: $w_k \leftarrow w_k - \eta \frac{\partial L}{\partial w_k}, \quad L = L(\hat{y}, y)$
- Repeat many times until convergence

Mini-batch Stochastic Gradient Descent (SGD)

- Initialization: initialize the network parameters using random numbers
one epoch (the network sees all of the training data points)
- Randomly shuffle data points in training set and divide them into mini-batches
a mini-batch is a subset of the training set. Assume there are **M** mini-batches
- for **m** in [1, 2, ..., M]:
 - assume data points in current mini-batch are $\{(x_i, y_i), i = 1, \dots, N_m\}$ N_m is called batch_size
 - Forward Pass: compute output \hat{y} for every data point in this mini-batch
 - Backward Pass: compute the derivatives
 - Update parameters: $w_k \leftarrow w_k - \eta \frac{\partial L}{\partial w_k}, L = \frac{1}{N_m} \sum_{i=1}^{N_m} L(\hat{y}_i, y_i)$
- Run many epochs until convergence