Shallow to Deep Neural Networks

Computer Vision and Artificial Intelligence

Dr Varun Ojha

varun.ojha@ncl.ac.uk

School of Computing

Newcastle University

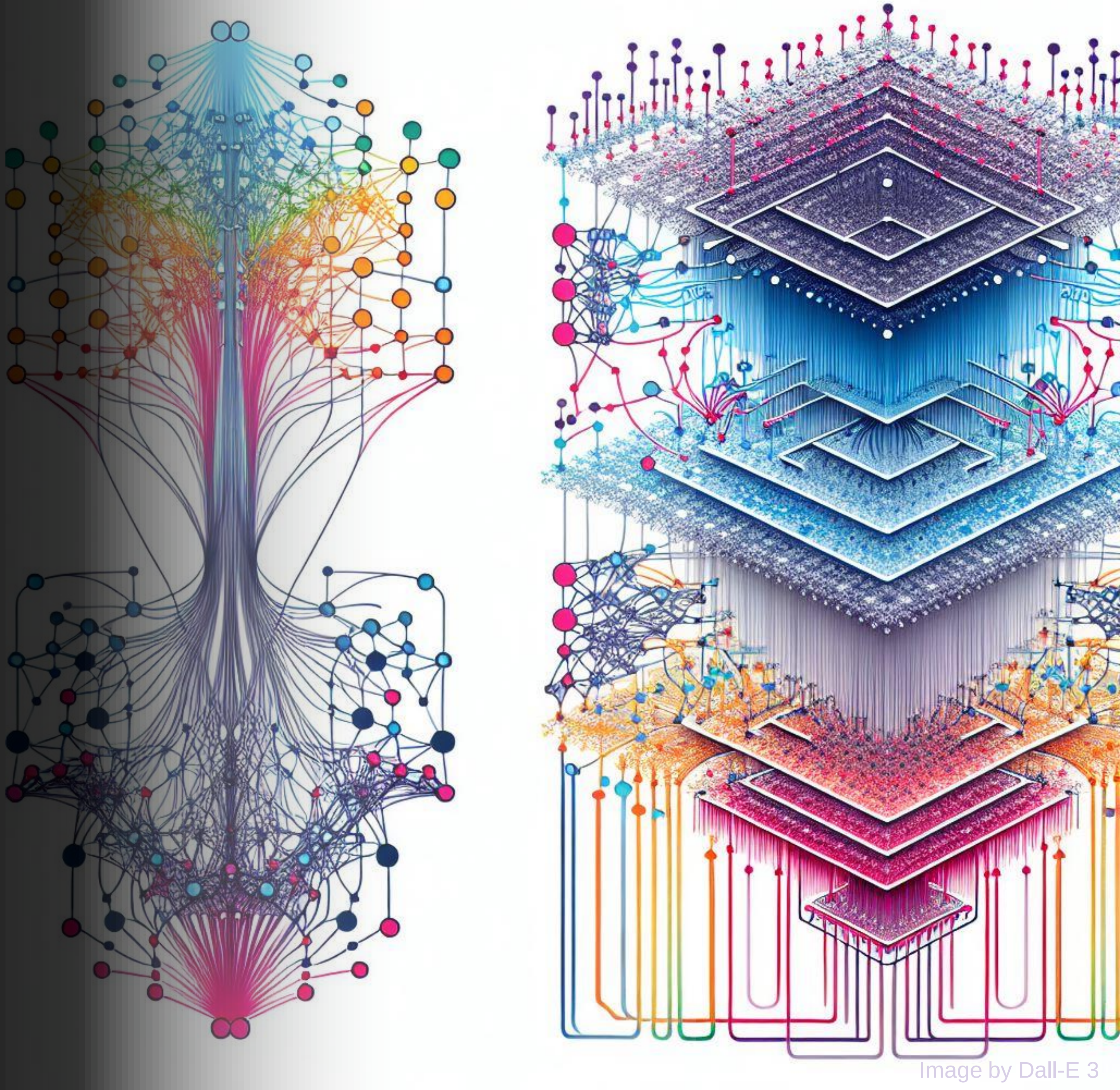


Image by Dall-E 3

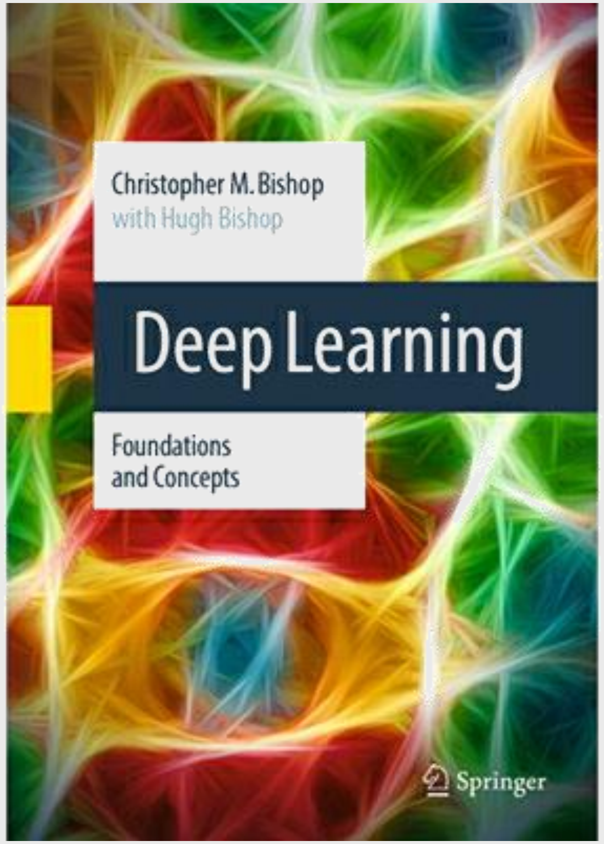
Learning objectives (neural networks)

By the end of this week, you will be able to

- Learn ‘concepts of learning’ in Neural Networks
- Understand gradient descent and backpropagation algorithms
- Distinguish shallow and deep neural network architectures
- Apply and evaluate neural networks for a pattern recognition (image classification) problem – *in practical session*

Content (neural networks)

- **Part 1: Introduction**
- **Part 2: Fundamental Theory**
    - Supervised Learning problem
    - Design of learning process
    - Gradient descent optimization
- **Part 3: Neural Network Architectures**
    - Neural Network Components
    - The Backpropagation algorithm
    - Deep Neural Networks
- **Part 4: Practical Exercise**

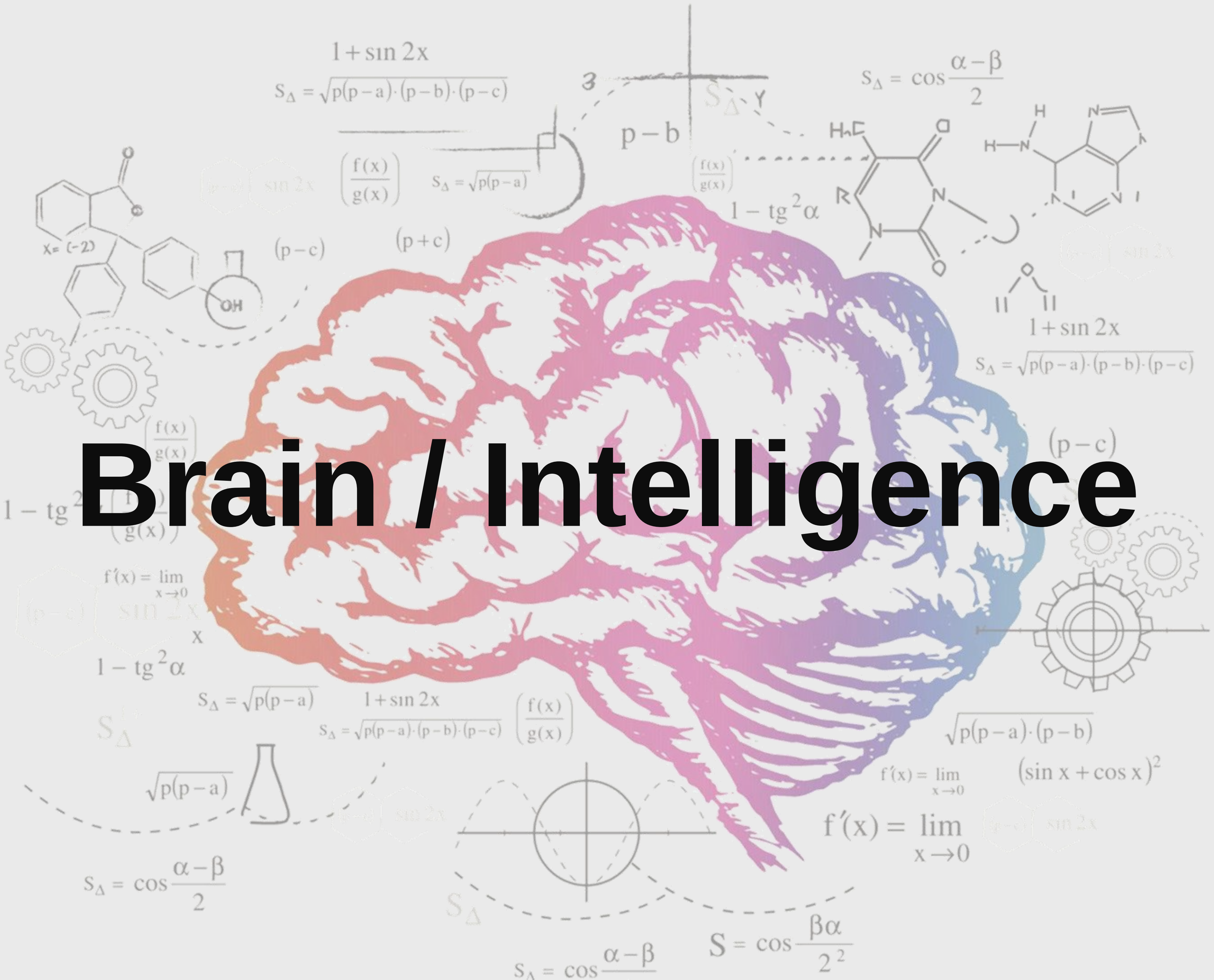


I recommend *Deep Learning* by C Bishop
https://www.bishopbook.com/

Image by Dall-E 3

Part 1
Introduction

Brain / Intelligence



esenkartal / iStock Vectors / Getty Images

Intrinsic Intelligence?

Inside a baby's mind

Experiment:
Warneken &Tomasello (2006)

Video Source:
https://www.youtube.com/watch?v=cUWIIxpUfM0
(Accessed on 21 Feb 2021)

Causal understanding of water displacement by a crow

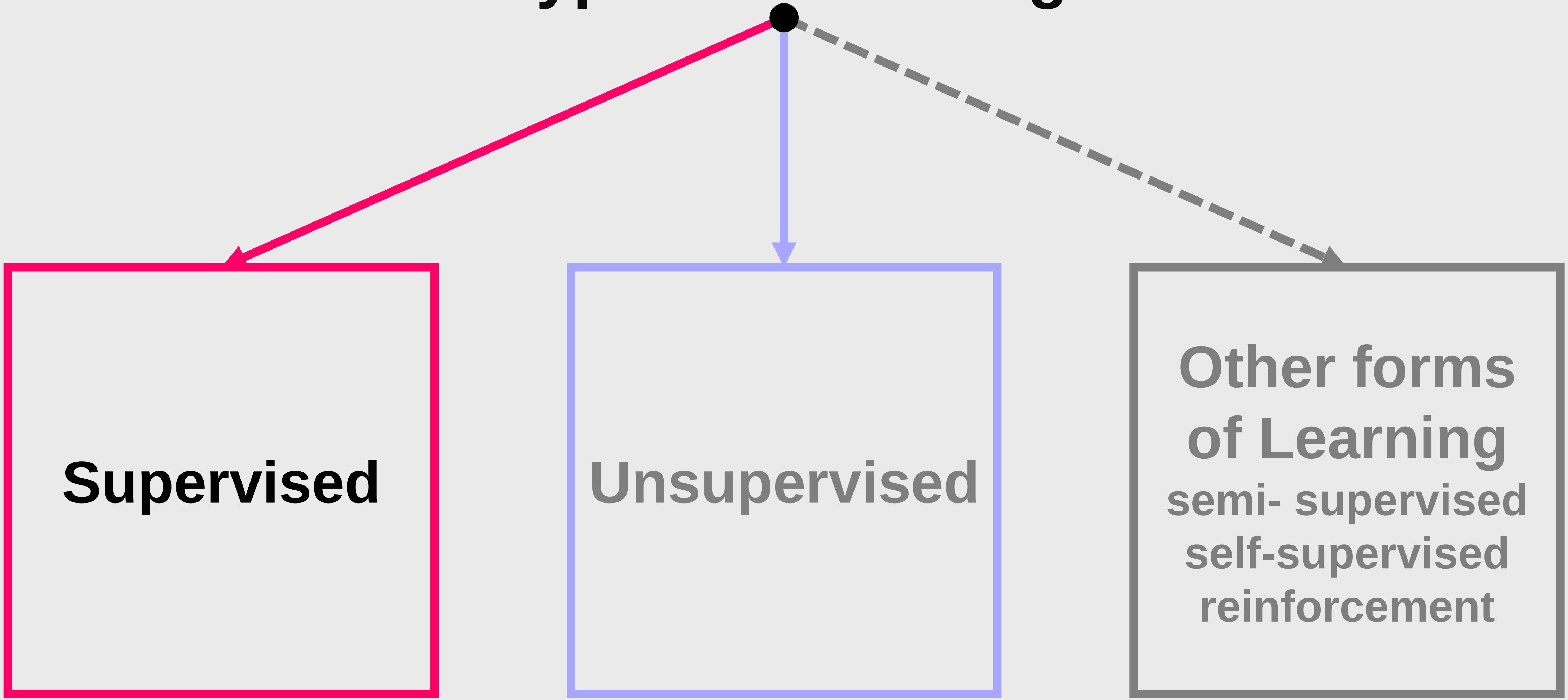
Experiment: Sarah et al. (2014), Auckland and Cambridge
Video Source: https://www.youtube.com/watch?v=ZerUbHmuY04

Learning by example

Learning / Training

Video Source:
https://www.youtube.com/watch?v=Ak7bPuR2rDw
(Accessed on 21 Feb 2021)

Types of Learning



Supervised

Unsupervised

Other forms of Learning

semi- supervised
self-supervised
reinforcement

(this lecture)

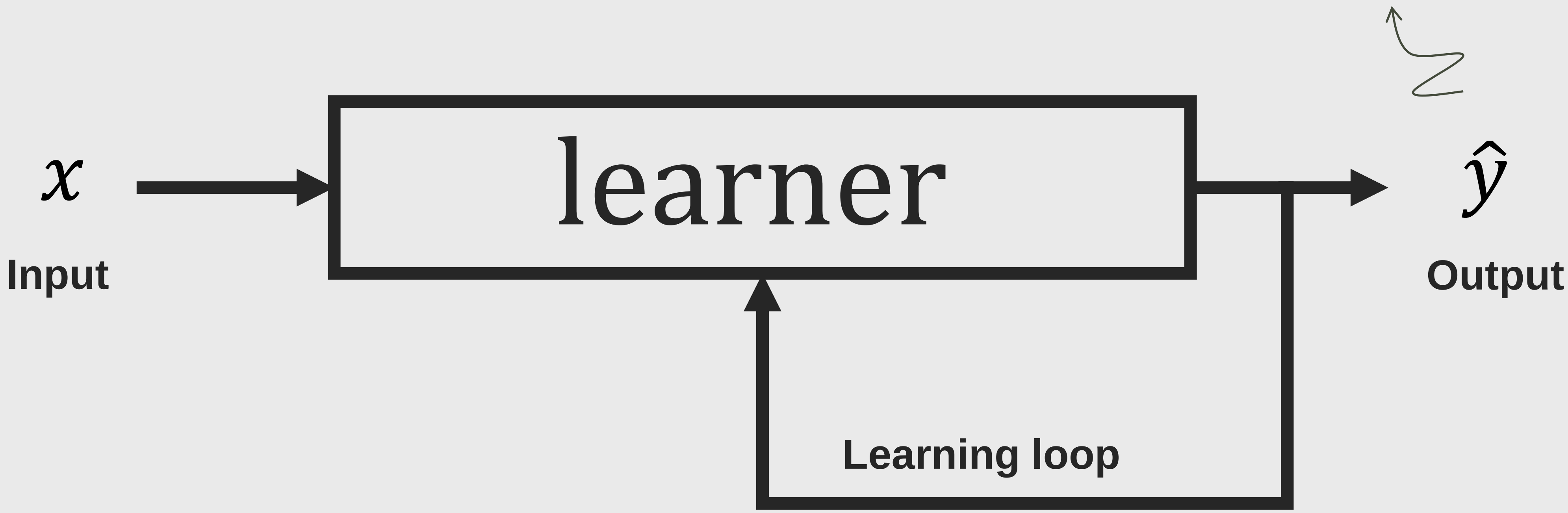
(re-visit previous lectures)

(not in scope of this module)

9:37 AM

Unsupervised

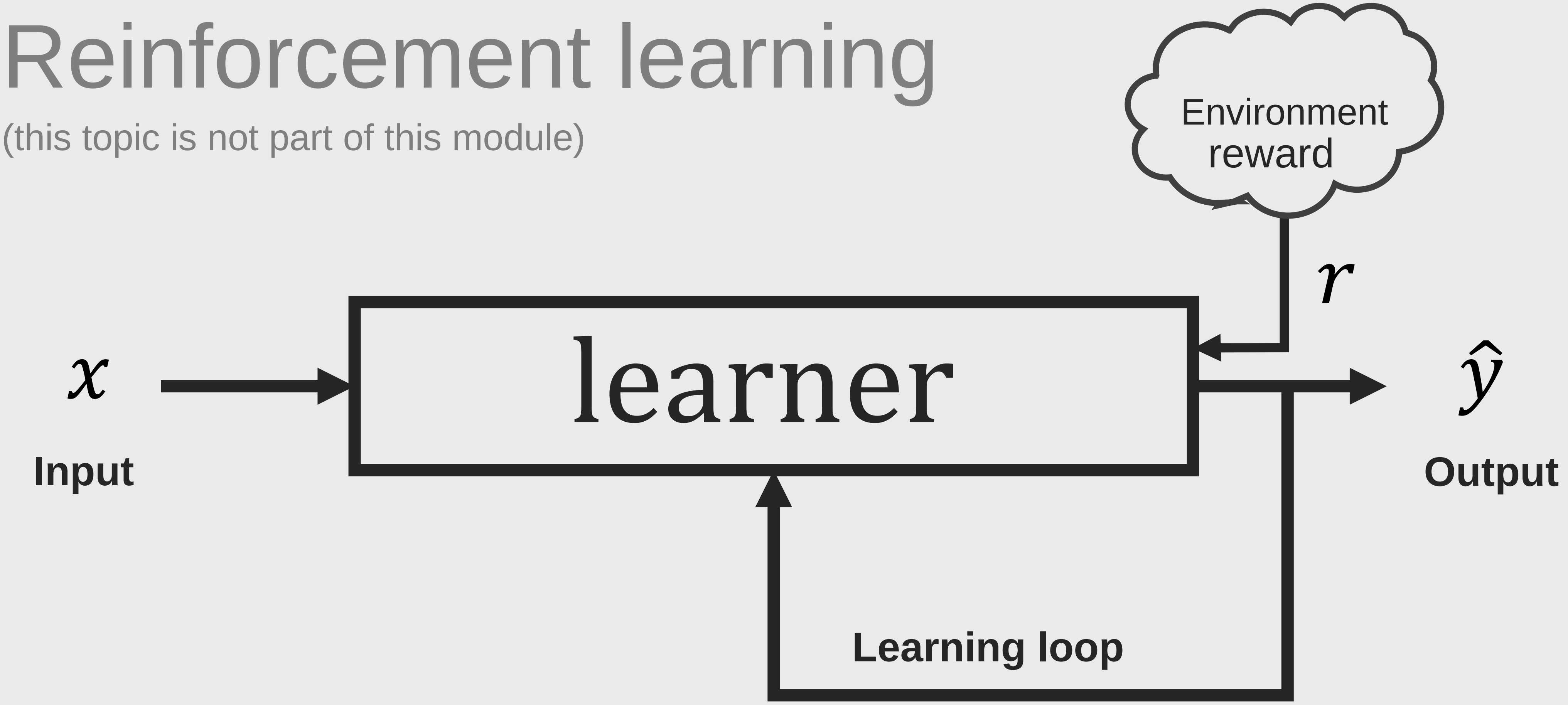
Categorise input features or learns the input features representation or learns the structure of the input



Re-visit previous lectures: Clustering (K-Means, DB Scan); Dimensionality reduction; Anomaly detection, Variational Autoencoder (VAE; *is not a part of this module*):

Reinforcement learning

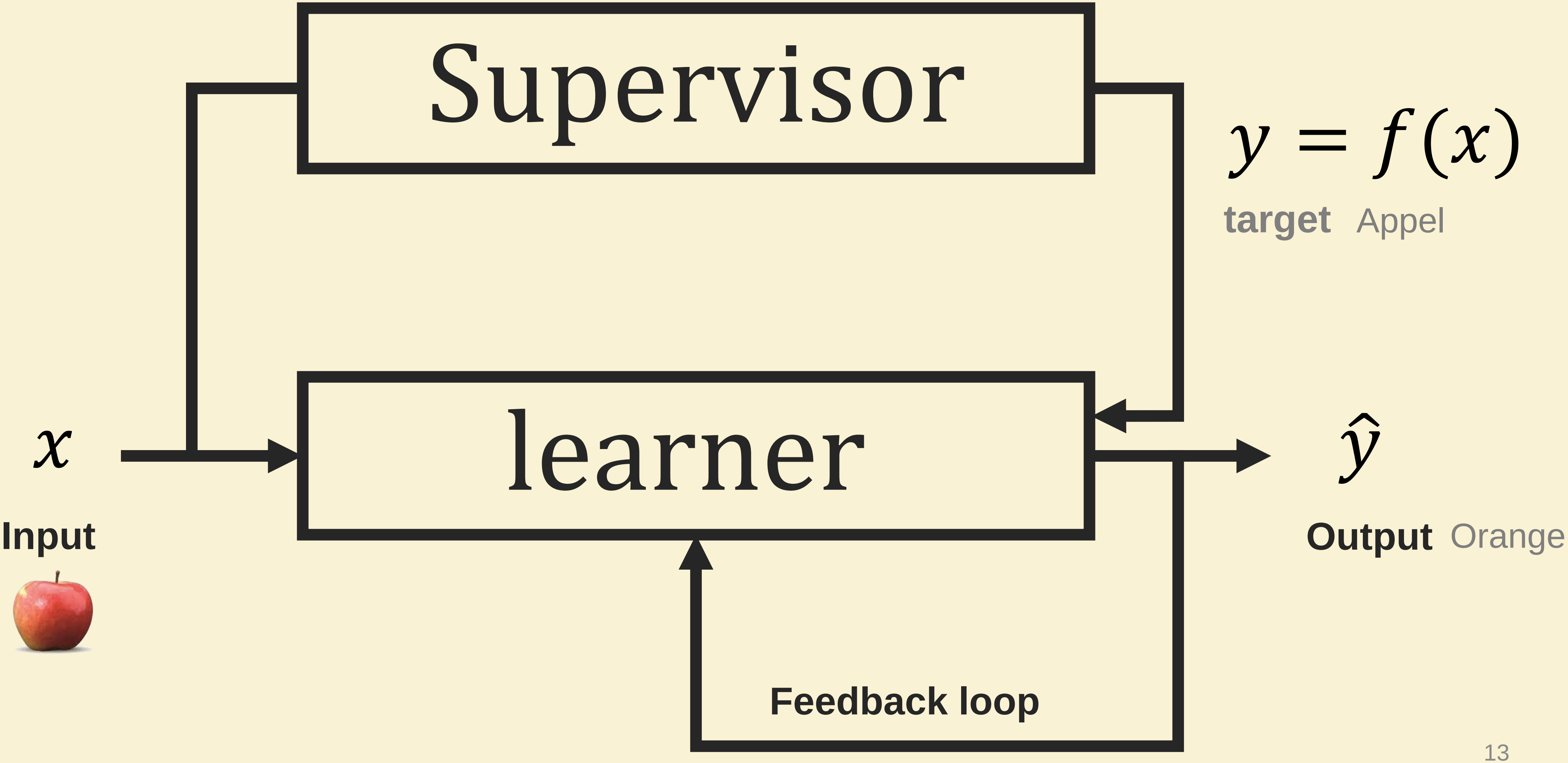
(this topic is not part of this module)



Example are game playing (most popular games are ‘Atari’ and ‘Hide and Seek’ where reinforcement learning (RL) is used, or RL is heavily used in robotics for learning control and actions)

Explore (<u>not part of this module</u>)**:** Reinforcement Learning: An Introduction by Richard S Sutton

Supervised



Supervisor

$y = f(x)$

target Appel

x

Input

learner

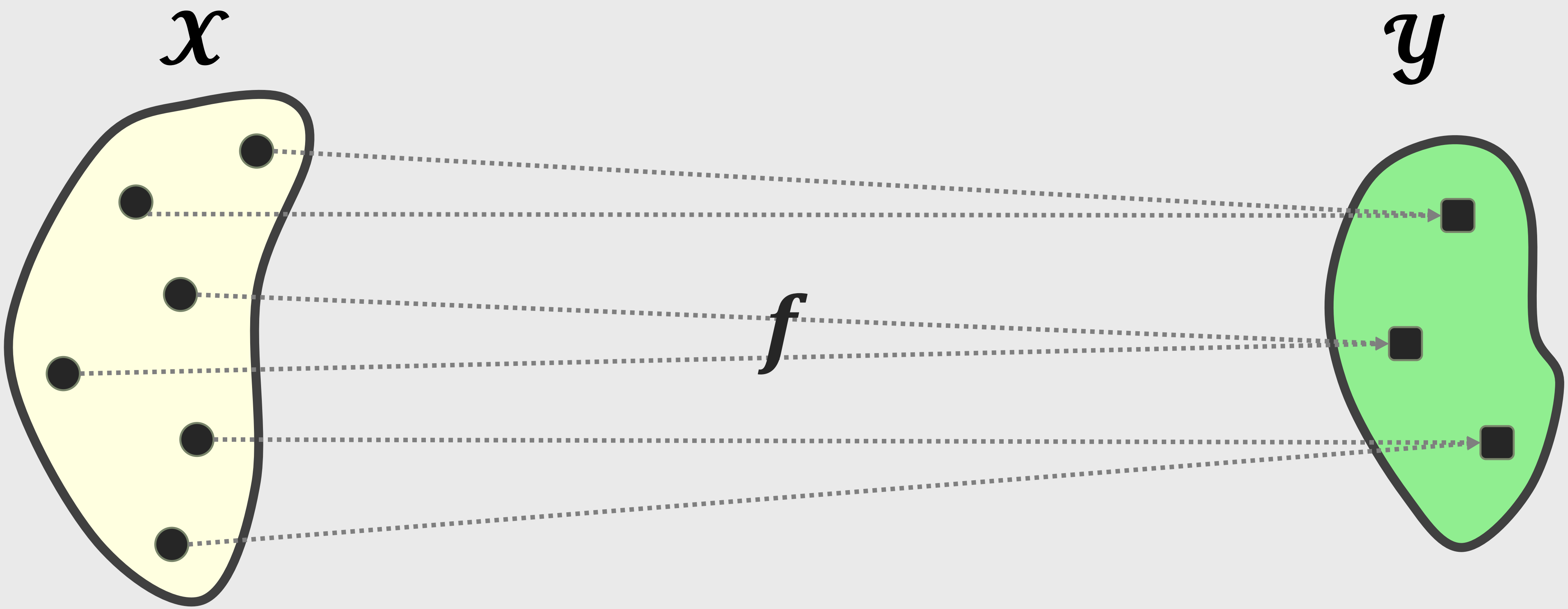
$\hat{y}$

Output Orange

Feedback loop

Learning $\boldsymbol{f} : \boldsymbol{X} \rightarrow \boldsymbol{y}$

Supervised learning is a mapping f of inputs $\mathcal{X}$ to outputs $\mathcal{Y}$

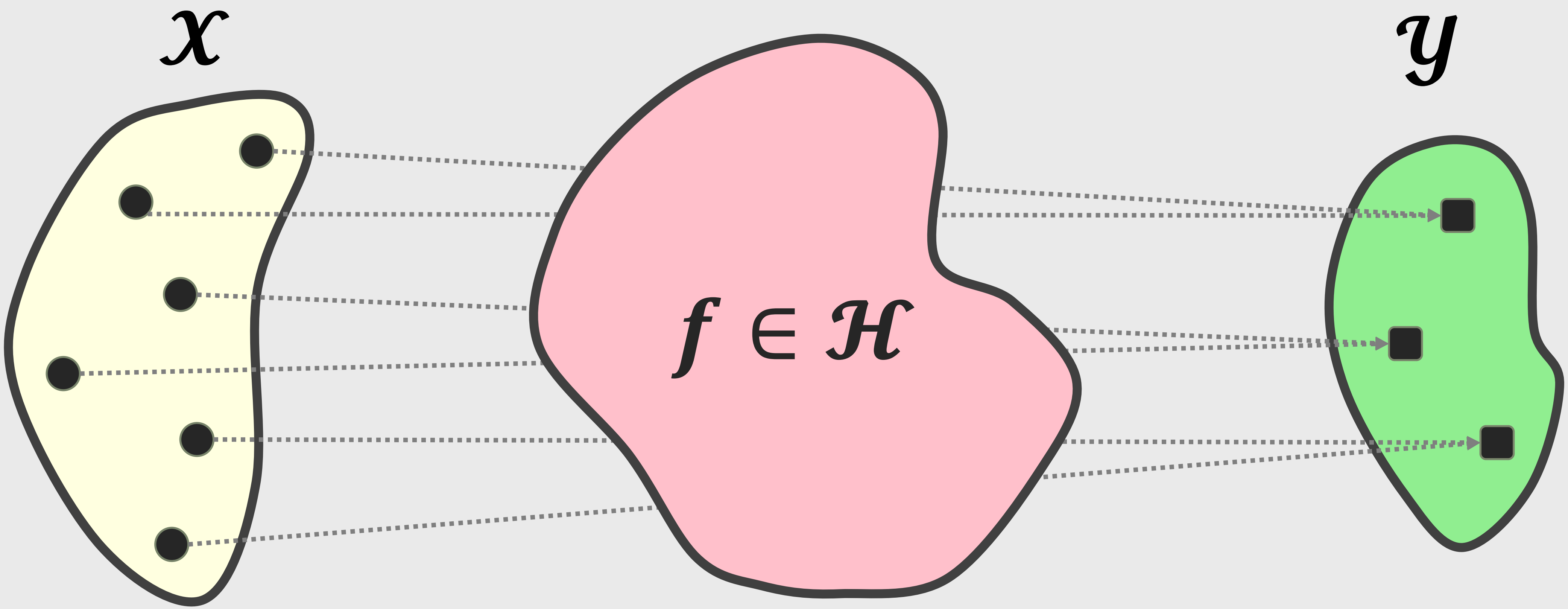


Inputs $\mathbf{X} \in$ Input space $\mathcal{X}$

outputs $\boldsymbol{y} \in$ output space $\mathcal{Y}$

Learning $\boldsymbol{f} : \boldsymbol{X} \rightarrow \boldsymbol{y}$

We need to find the unknown target function $\boldsymbol{f}$ that maps x to y



Inputs $\mathbf{X} \in$ Input space $\mathcal{X}$

model space $\mathcal{H}$

outputs $\boldsymbol{y} \in$ output space $\mathcal{Y}$

Learning: $g(\mathbf{X}) \sim f(\mathbf{X})$

We need to search a function $g(X)$ that can approximate $f(X)$

Example Training Task: AND Logic Problem

	Colour	Shape	Fruit Name
	x_1	x_2	y
Fruit $\mathbf{x_1}$:	0	0	0
$\mathbf{x_2}$:	0	1	0
$\mathbf{x_3}$:	1	0	0
$\mathbf{x_4}$:	1	1	1

Input $\mathbf{X} = (\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, \mathbf{x_4})^T$, $\mathbf{x}_i = (x_{i1}, x_{i2})$,

Output $y = \{0,1\}$

Number of Inputs $d = 2$

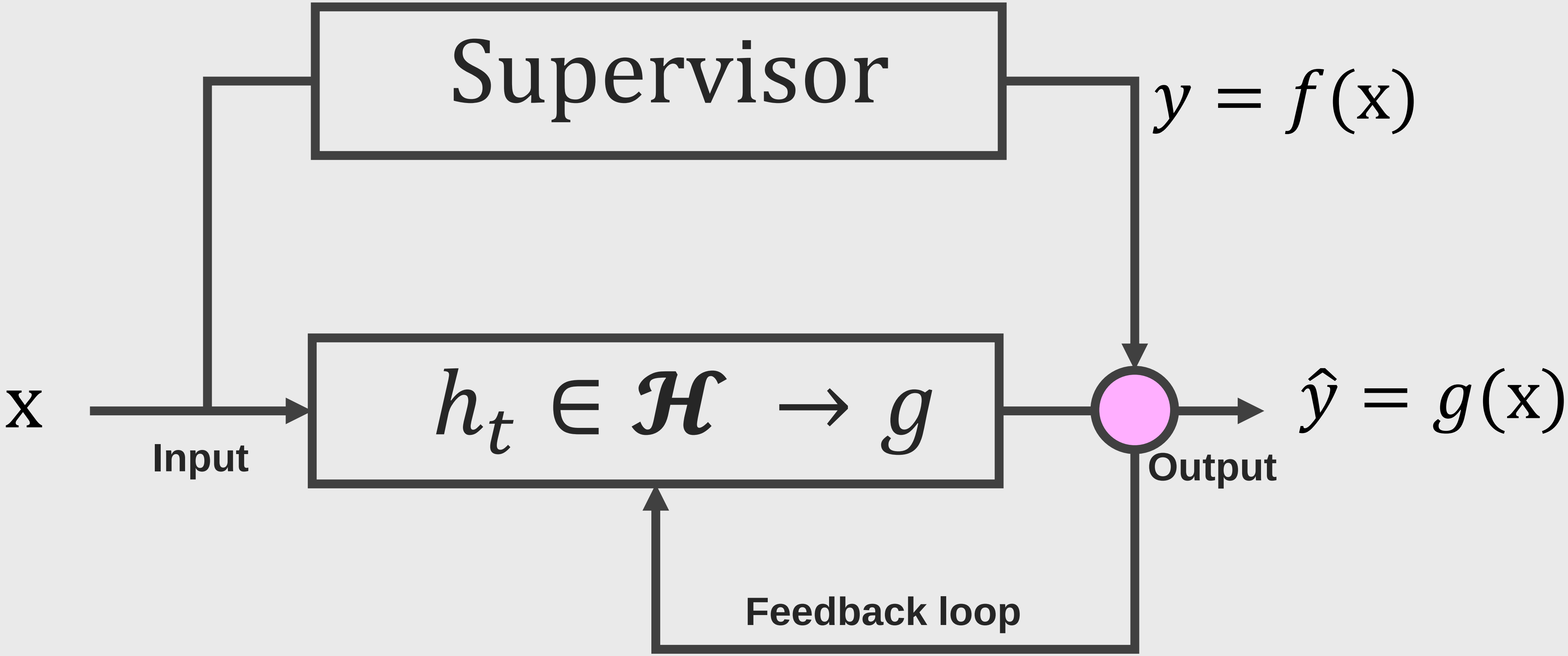
Each input x takes values either 0 or 1

Input-space $\mathcal{X} = 2^d = 2^2 = 4$

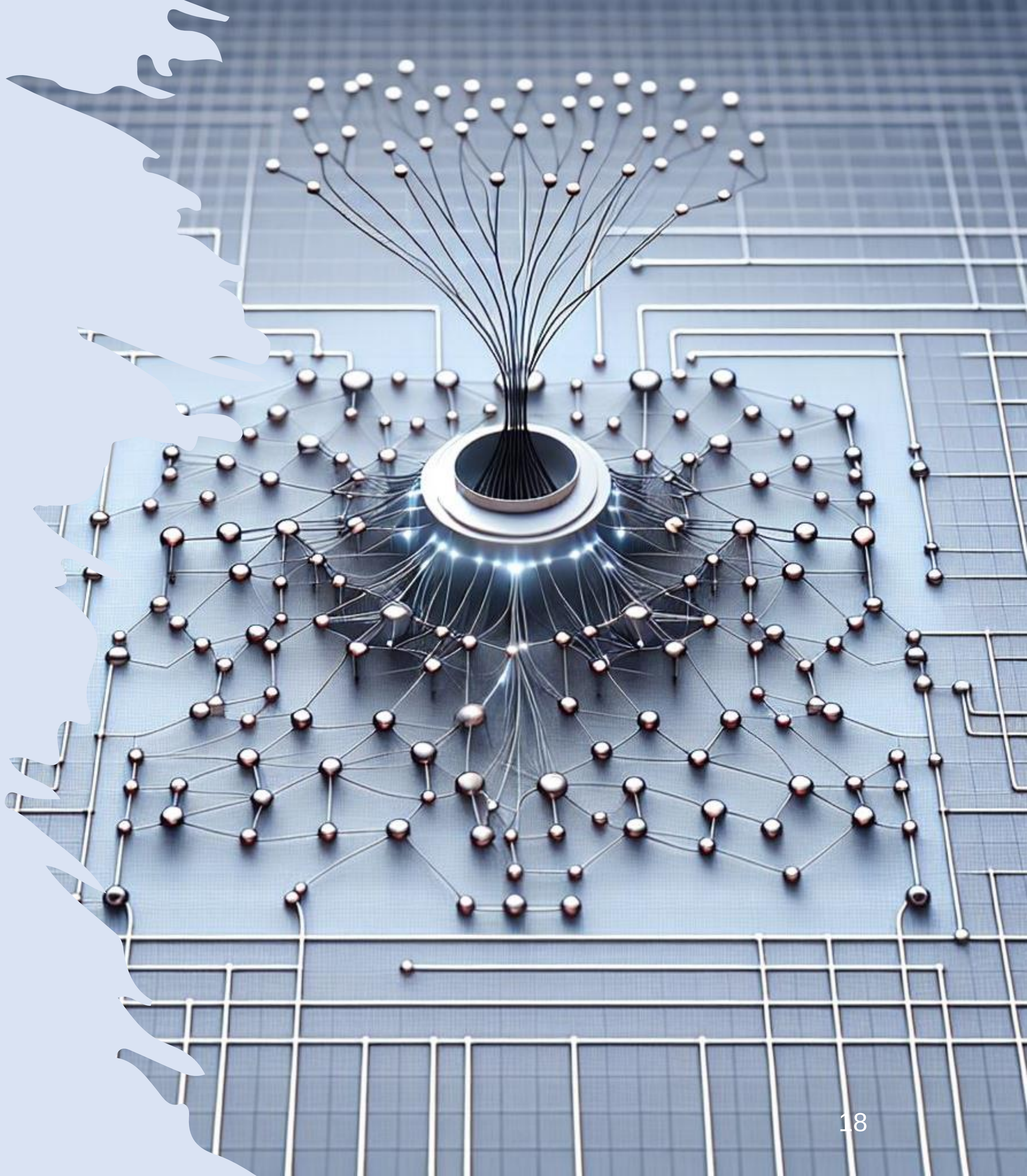
Number of outputs 1

Output y takes 2 options from $\{0,1\}$

Search a function $g: \mathbf{X} \rightarrow y$ that approximates $f(x)$



Part 2
Learning Theory



Requirements of Learning

Learning needs to

Represent a model (use a neural network architecture, deep neural networks)

Evaluate the model (use a loss/cost function, e.g., Cross Entropy or MSE)

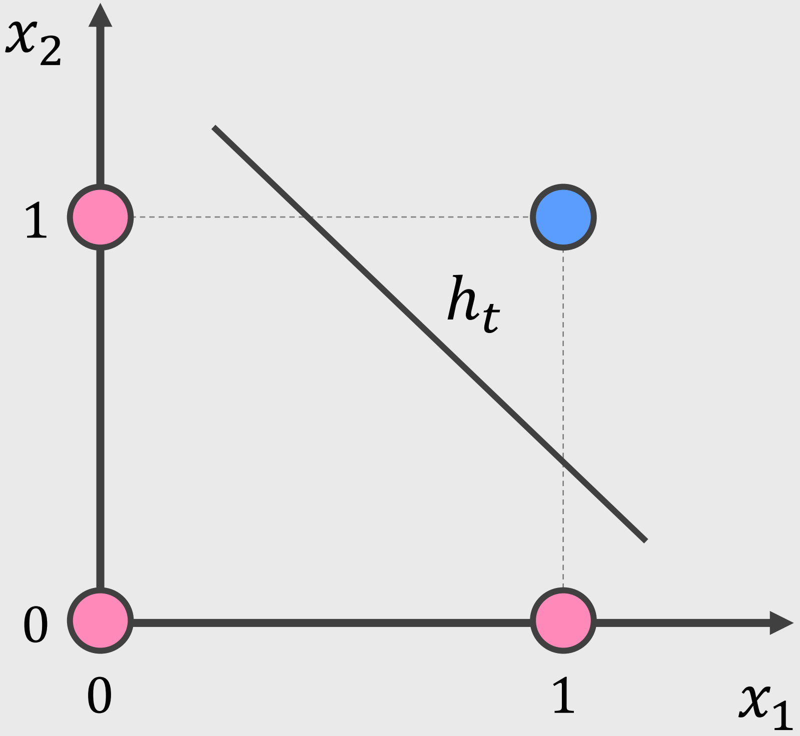
Optimize the model (use an optimizer, e.g., backpropagation – Adam or SGD)

Represent a model

A line separating data can be considered as a model

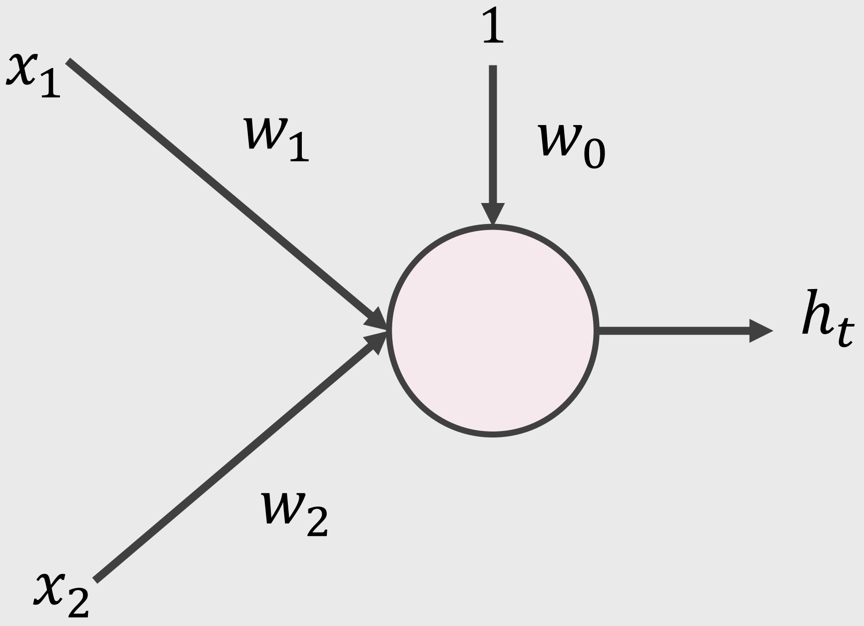
$\mathcal{D}$

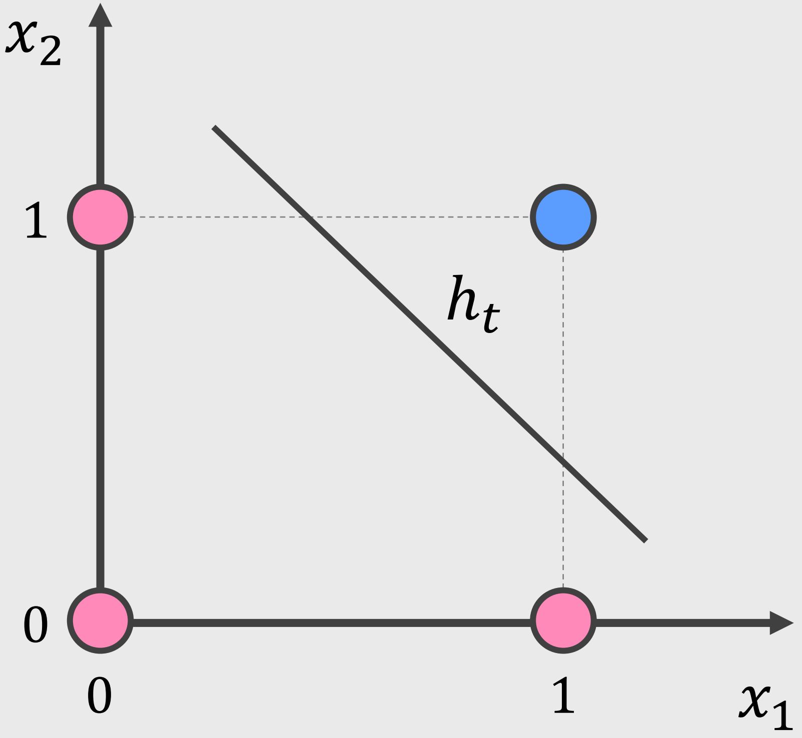
	x_1	x_2	y
1:	0	0	0
2:	0	1	0
3:	1	0	0
4:	1	1	1



Represent a model

A line separating data can be considered a model
which equivalent to a single neuron or a perceptron





9:37 AM

Represent a model $h_t \in H$

A line separating data can be considered a model which equivalent to a single neuron or a perceptron

Perceptron is a simple linear combination of inputs, which is written as:

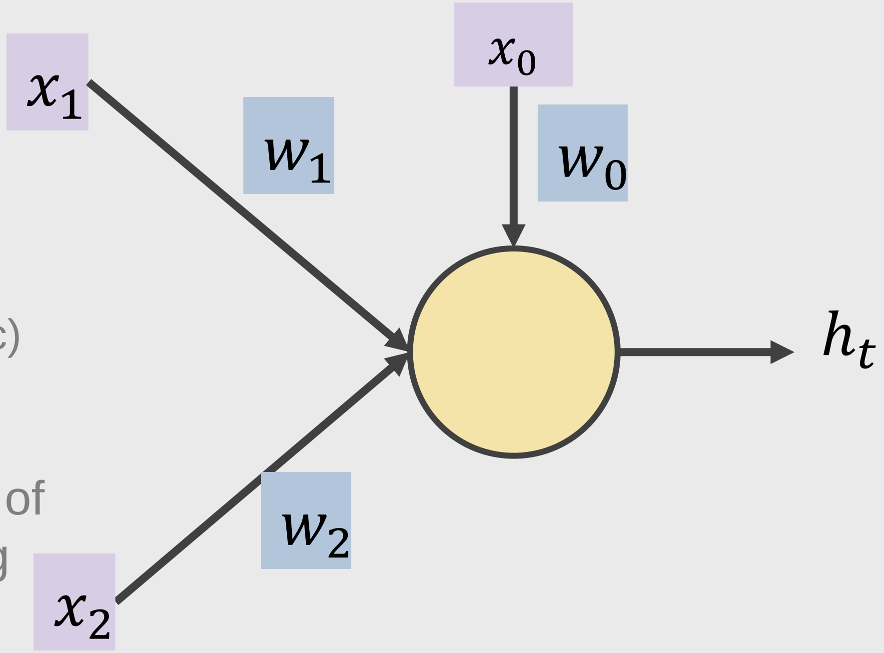
$$h_t = g(x) = \sum_{i=1}^{d} w_i \, x_i \geq x_0 w_0 ,$$

This equation is also equivalent to linear regression (y = mx + c)

where w_0 is a threshold.

Real bottleneck of Deep Learning

The model h_t has the weights w_i and the threshold w_0 as its **trainable parameters**.



Model h_t as a **perceptron / single neuron**.

Read: Sec 4.1 and Sec 5.1, Deep Learning by C Bishop

Represent a model $h_t \in H$

A line separating data can be considered as a model

A model h_t as a perceptron.

$$\sum_{i=1}^{d} w_i x_i \geq x_0 w_0$$

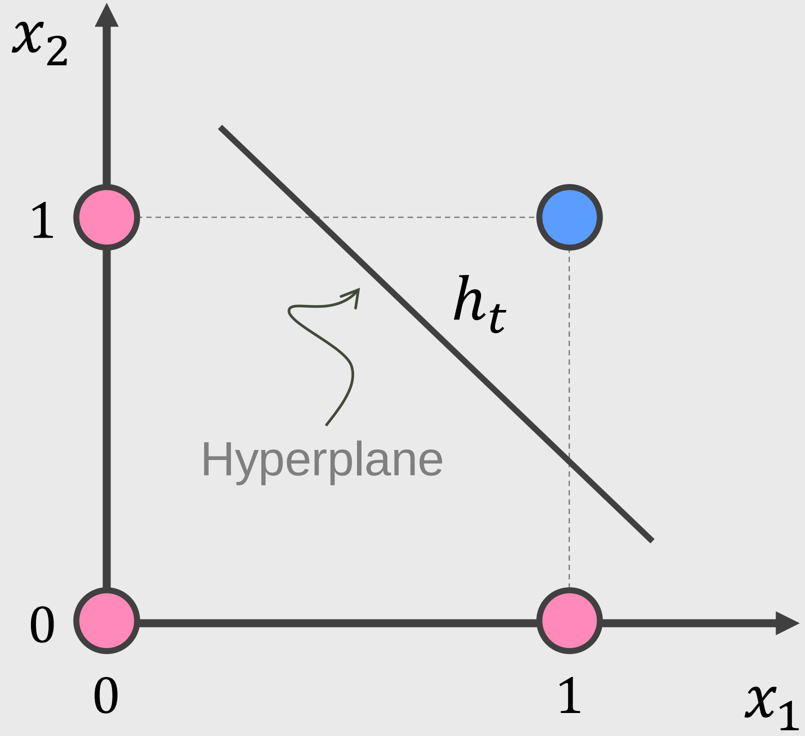
$$\sum_{i=1}^{d} w_i x_i - x_0 w_0 = 0$$

This equation is also called a hyperplane

For an artificial input (also called bias) $x_0 = 1$ we have:

$$\sum_{i=0}^{d} w_i x_i = 0$$

This is an equation of a single neuron



hyperplane as decision boundary

9:37 AM

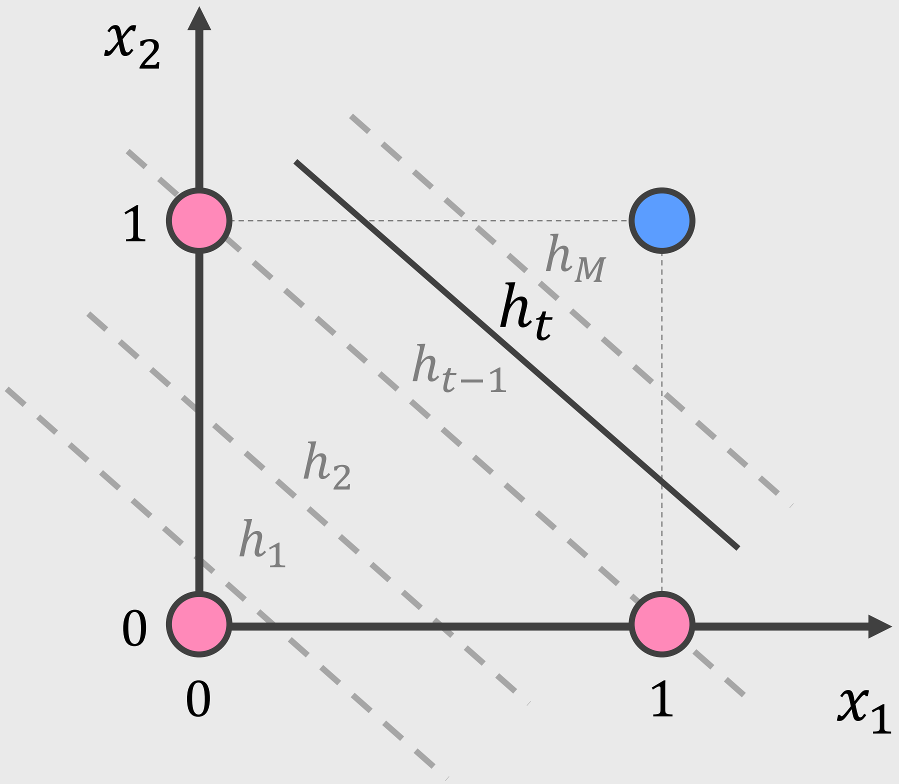
Which model $h_t \in H$ to pick?

How to **evaluate** a model: compute cost of choosing a model

		x_1	x_2	$y = f(\mathbf{x})$
$\mathcal{D}$	1:	0	0	0
	2:	0	1	0
	3:	1	0	0
	4:	1	1	1

Cost function such as the error rate:

$$E(h_t(\mathcal{D})) = \frac{1}{N}\sum_{j=1}^{N}\left(g_{\mathbf{w}}(\mathbf{x}_j) \neq f(\mathbf{x}_j)\right)$$



Optimise model $h_t \in H$ by minimizing error

How to optimize a model: compute cost of adjust the model weights

Function g of the model has parameter $\mathbf{w}$:

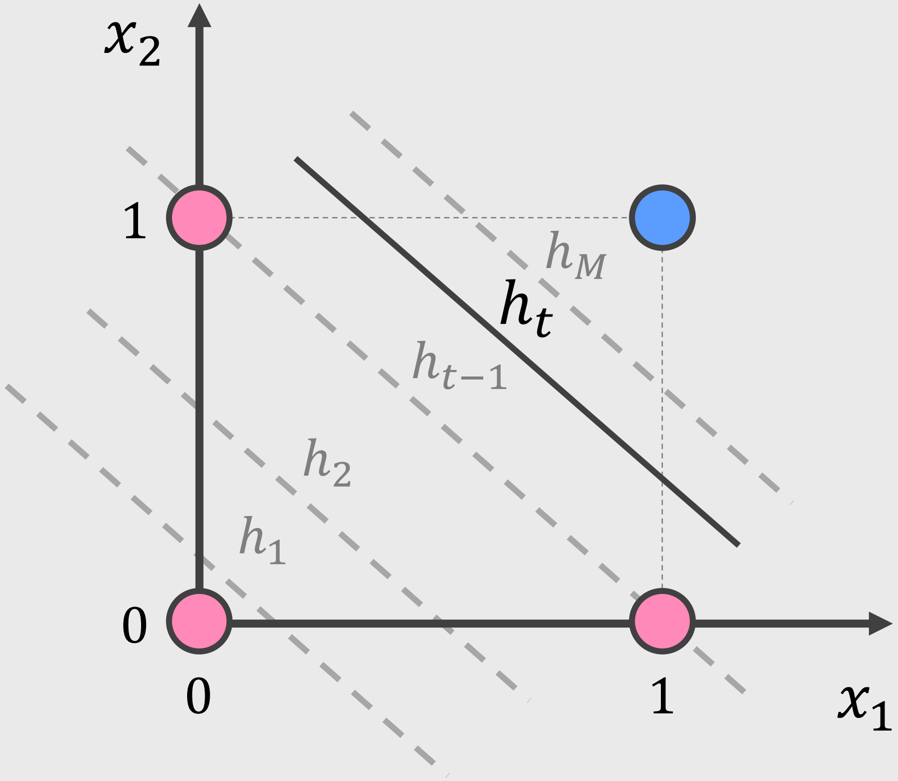
$$\hat{y} = g_{\mathbf{w}}(\mathbf{x}) = \sum_{i=0}^{d} w_i\, x_i = 0$$

Simple algorithm:

Repeat parameter $\mathbf{w}$ update for $t = 2, 3, \ldots, M$ as:

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \hat{y}\, \mathbf{x},$$

Until the error rate $E(h_t(\mathcal{D}))$ is acceptable or close to zero.



Does error $E(h_t(\mathcal{D}))$ minimization work?

9:37 AM

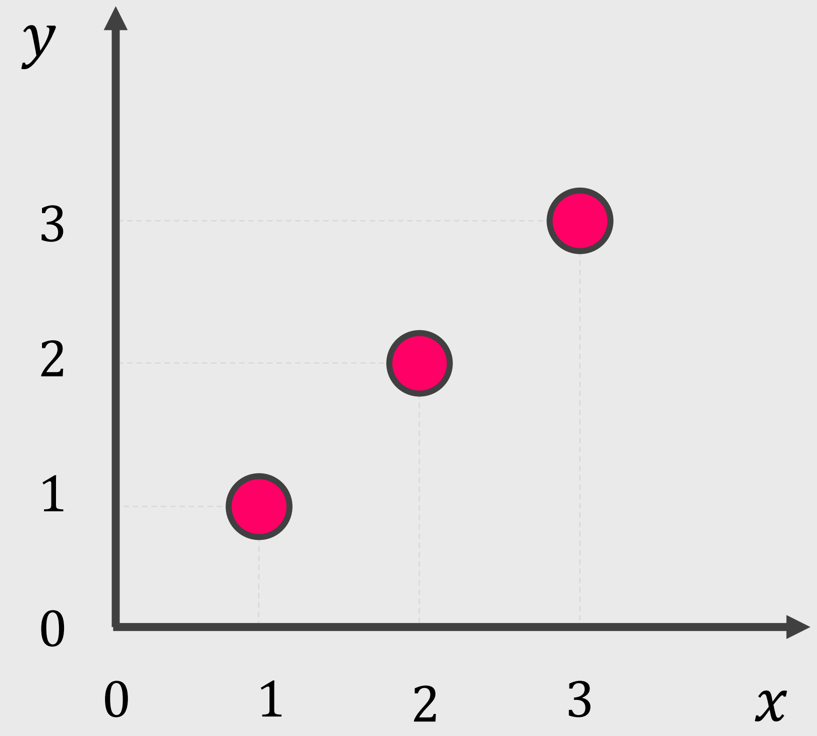
Let’s see an example (house price):

	$x = area(m^2)$	$y = price\ (in\ £)$
1:	1000	100K
2:	2000	200K
3:	3000	300K

Now, the **cost function** is a squared error:

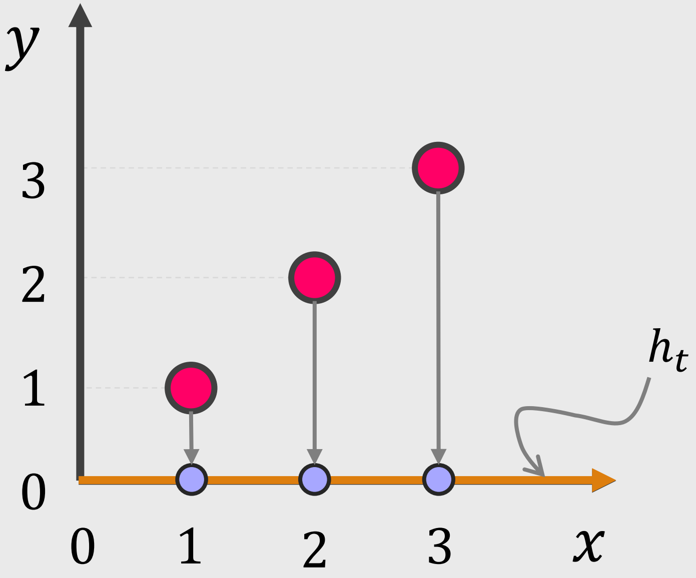
$$E(h_t(\mathbf{x}) = \frac{1}{2N}\sum_{j=1}^{N}\left(g_{\mathbf{w}}(\mathbf{x}_j) - f(\mathbf{x}_j)\right)^2$$

Note that y and x values are simplified to 1, 2 and 3



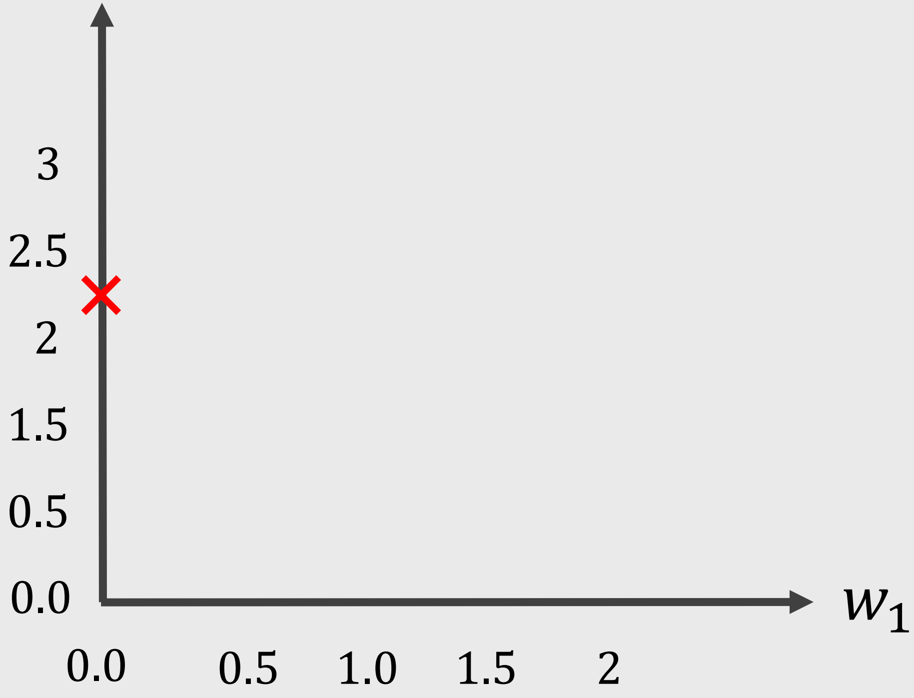
Does error $E(h_t(\mathcal{D}))$ minimization work?

Note that y and x values are simplified to 1, 2 and 3



Model / line is close to x-axis

$$E(g_{w_1}(x)) = \frac{1}{2N}\sum_{j=1}^{N}\left(f(\mathbf{x}_j) - g_{\mathbf{w}}(\mathbf{x}_j)\right)^2$$



Model h_t for $w_0 = 0$ and $w_1 = 0.0$:

$g_{\mathbf{w}}(x_i) = w_0 + w_1 x_i$ for $i = 1, 2, 3$

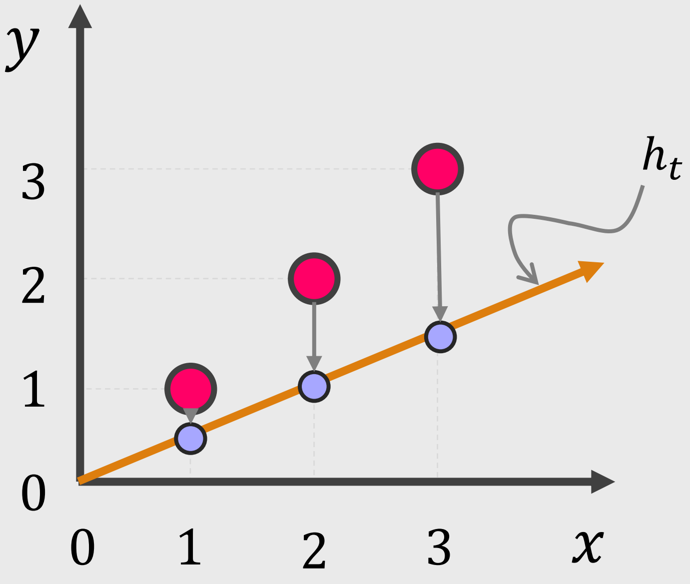
Error $E(w_1)$ for $w_0 = 0$ and $w_1 = 0.0$:

$$E(g_{\mathbf{w}}(\mathbf{x})) = \frac{(1-0)^2+(2-0)^2+(3-0)^2}{2*3} = 2.33$$

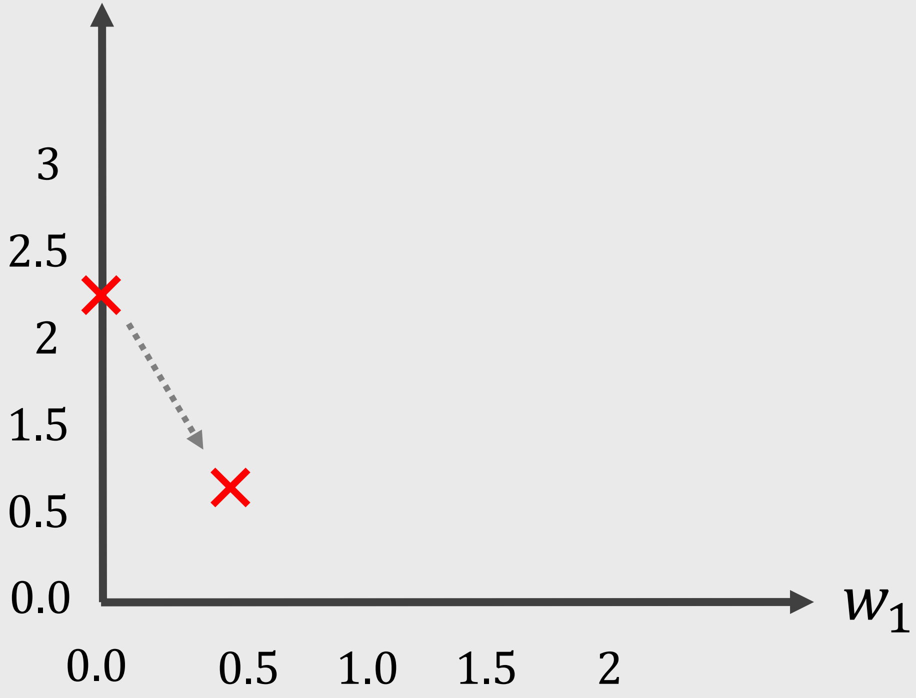
Does error $E(h_t(\mathcal{D}))$ minimization work?

9:37 AM

Note that y and x values are simplified to 1, 2 and 3



$E(g_{w_1}(x)) = \frac{1}{2N}\sum_{j=1}^{N}\left(f(\mathbf{x}_j) - g_{\mathbf{w}}(\mathbf{x}_j)\right)^2$



Model h_t for $w_0 = 0$ and $w_1 = 0.5$:

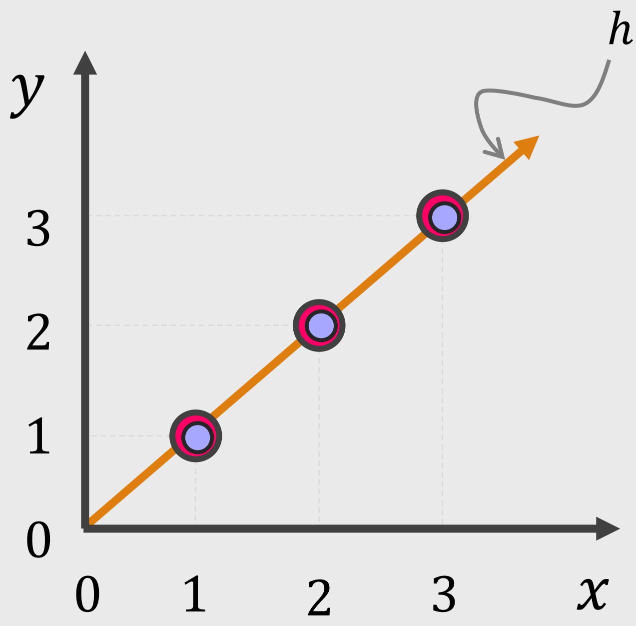
$g_{\mathbf{w}}(x_i) = w_0 + w_1 x_i$ for $i = 1, 2, 3$

Error $E(w_1)$ for $w_0 = 0$ and $w_1 = 0.5$:

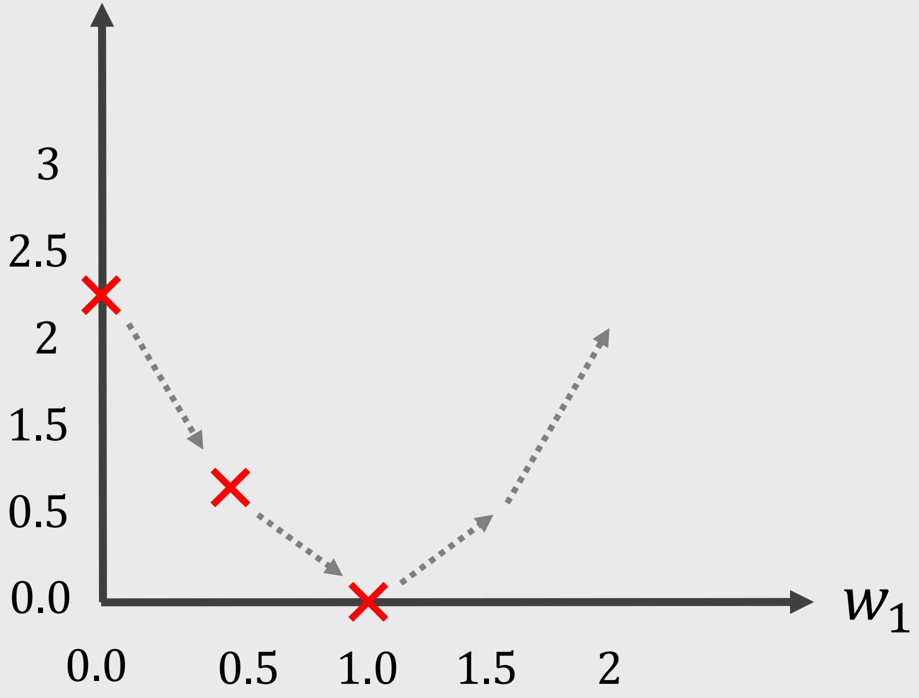
$$E(g_{\mathbf{w}}(\mathbf{x})) = \frac{(1-0.5)^2+(2-1)^2+(3-1.5)^2}{2*3} = 0.58$$

Does error $E(h_t(\mathcal{D}))$ minimization work?

Note that y and x values are simplified to 1, 2 and 3



$E(g_{w_1}(x)) = \frac{1}{2N}\sum_{j=1}^{N}\left(f(\mathbf{x}_j) - g_{\mathbf{w}}(\mathbf{x}_j)\right)^2$



Model h_t for $w_0 = 0$ and $w_1 = 1$:

$g_{\mathbf{w}}(x_i) = w_0 + w_1 x_i$ for $i = 1, 2, 3$

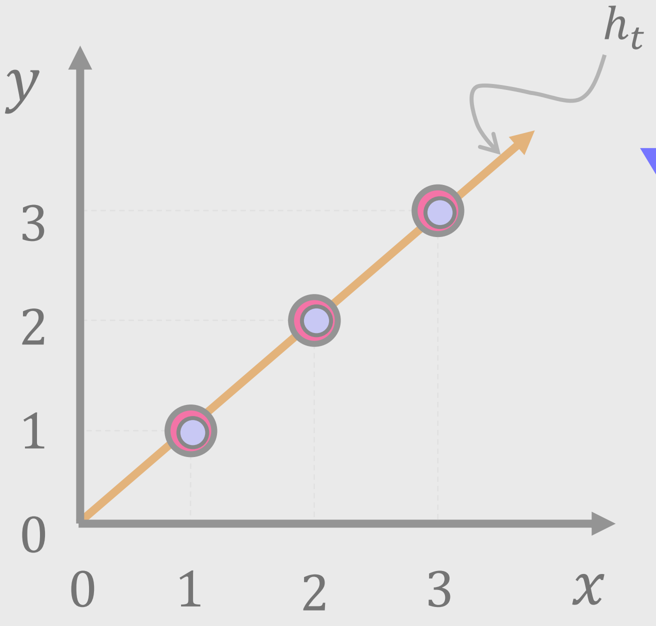
Error $E(w_1)$ for $w_0 = 0$ and $w_1 = 1$:

$$E(g_{\mathbf{w}}(\mathbf{x})) = \frac{(1-1)^2+(2-2)^2+(3-3)^2}{2*3} = 0.0$$

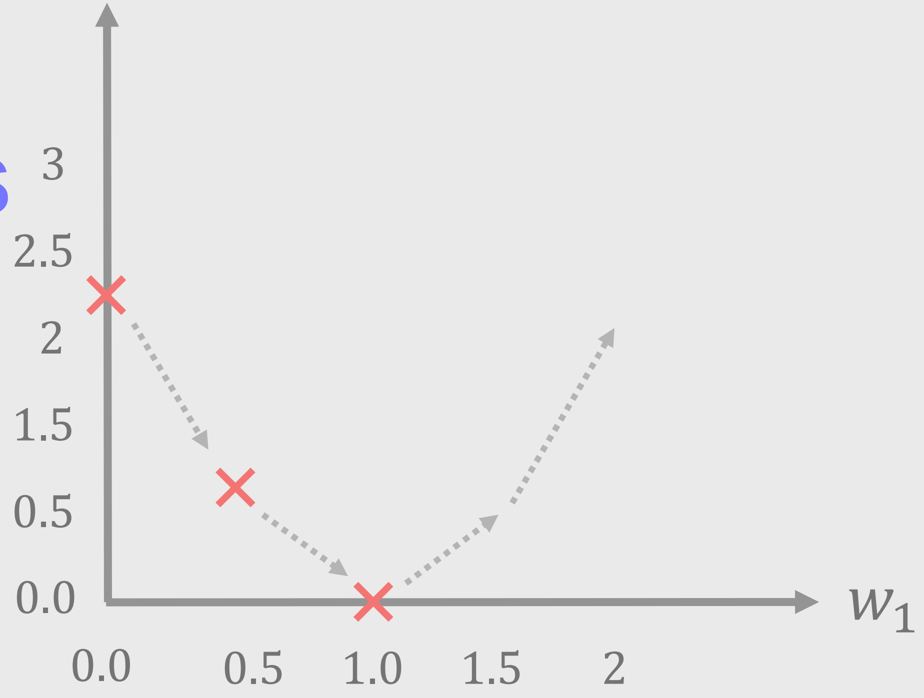
Does error $E(h_t(\mathcal{D}))$ minimization work?

9:37 AM

Note that y and x values are simplified to 1, 2 and 3



$$E(g_{w_1}(x)) = \frac{1}{2N}\sum_{j=1}^{N}\left(f(\mathbf{x}_j) - g_{\mathbf{w}}(\mathbf{x}_j)\right)^2$$



Yes, it does

But how much weight to change in each step?

Model h_t for $w_0 = 0$ and $w_1 = 1$:

$g_{\mathbf{w}}(x_i) = w_0 + w_1 x_i$ for $i = 1, 2, 3$

Error $E(w_1)$ for $w_0 = 0$ and $w_1 = 1$:

$$E(g_{\mathbf{w}}(\mathbf{x})) = \frac{(1-1)^2+(2-2)^2+(3-3)^2}{2*3} = 0.0$$

Optimizer: Gradient Descent

9:37 AM

Function g of the model has parameter $\mathbf{w}$:

$$g_{\mathbf{w}}(\mathbf{x}) = \sum_{i=0}^{d} w_i\, x_i = 0$$

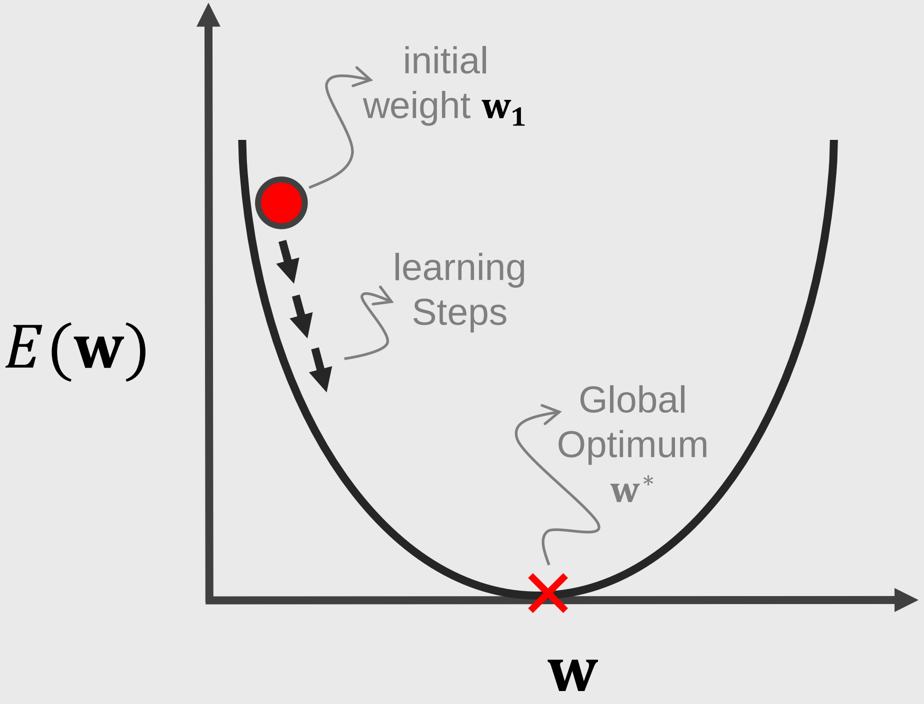
Gradient Descent Algorithm:

Repeat parameter $\mathbf{w}$ update for $t = 2, 3, \ldots, M$.

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \boxed{\hat{y}\ \mathbf{x},}$$

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \eta \frac{\partial E(g_w(x))}{\partial\, \mathbf{w}_t}\mathbf{x}$$ for learning rate η

Until error rate $E\left(g_{\mathbf{w}}(\mathbf{x})\right)$ is acceptable or goes to zero.



Sec 7.2, Deep Learning by C Bishop

Optimizer: Gradient Descent

9:37 AM

Function g of the model has parameter $\mathbf{w}$:

$$g_{\mathbf{w}}(\mathbf{x}) = \sum_{i=0}^{d} w_i \, x_i = 0$$

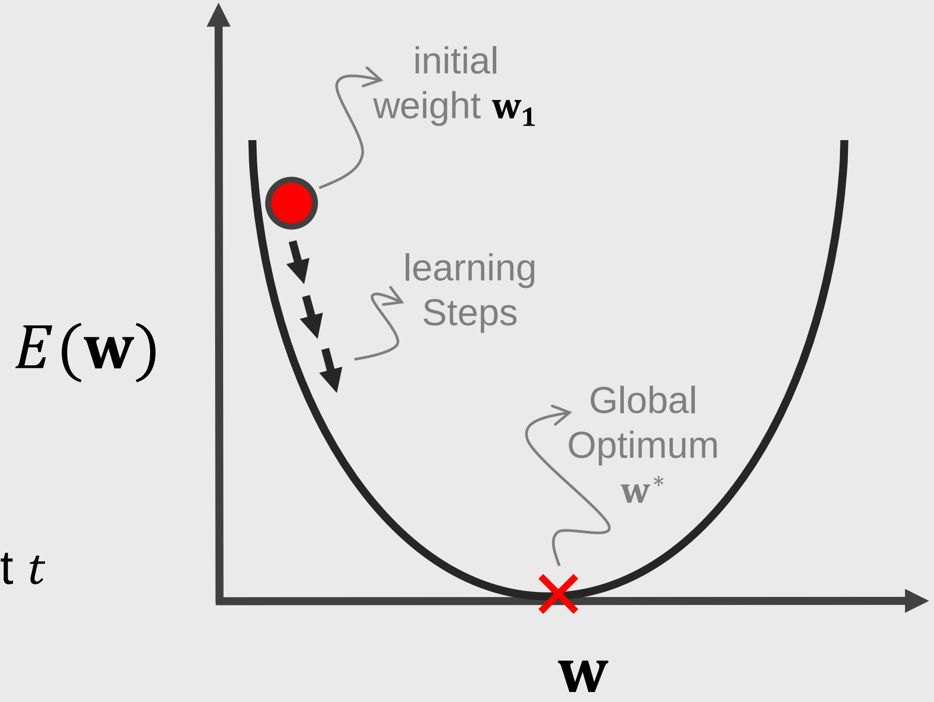
Gradient Descent Algorithm:

Repeat parameter $\mathbf{w}$ update for $t = 2, 3, \dots, M$.

$\eta \frac{\partial E(g_w(x))}{\partial \, \mathrm{w}_t}$

$\mathbf{w}_t \; = \; \mathbf{w}_{t-1} + \Delta \, \mathbf{w}_t$, where Δ is weight change (step) at t

Until error rate $E\left(g_{\mathbf{w}}(\mathbf{x})\right)$ is acceptable or goes to zero



Sec 7.2, Deep Learning by C Bishop

Versions of Gradient Descent

Stochastic Gradient Descent

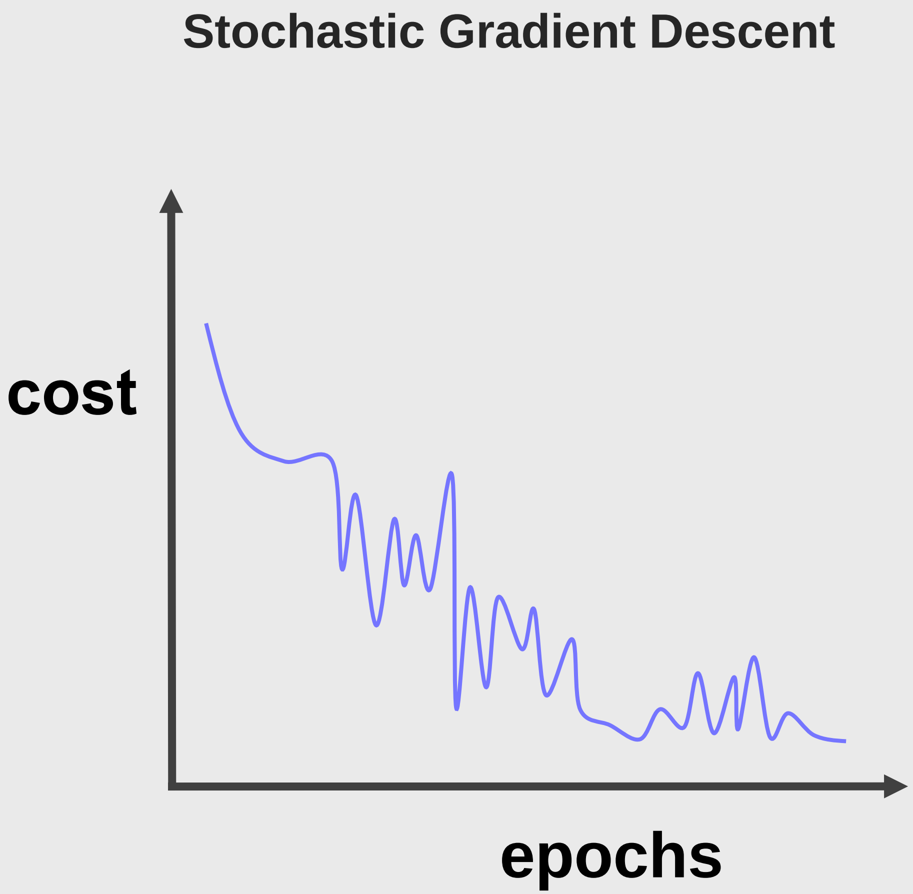
```
$t = 0$
$\mathbf{w}$ initial weights
for $t$ in epochs do
    $\mathcal{D} \leftarrow shuffle(\mathcal{D})$
    for $\mathbf{x}_j \in \mathcal{D}$ do  // for each sample
        $\nabla \mathbf{w}_j = \partial E(g_{\mathbf{w}_t}(\mathbf{x}_j))/(\partial \mathbf{w}_t)$ // gradient of error with respect to weight $\mathbf{w}_j$
        $\mathbf{w}_j = \mathbf{w}_{j-1} + \boldsymbol{\eta} \nabla \mathbf{w}_j \mathbf{x}_j$
    $t = t + 1$
```

Batch Gradient Descent

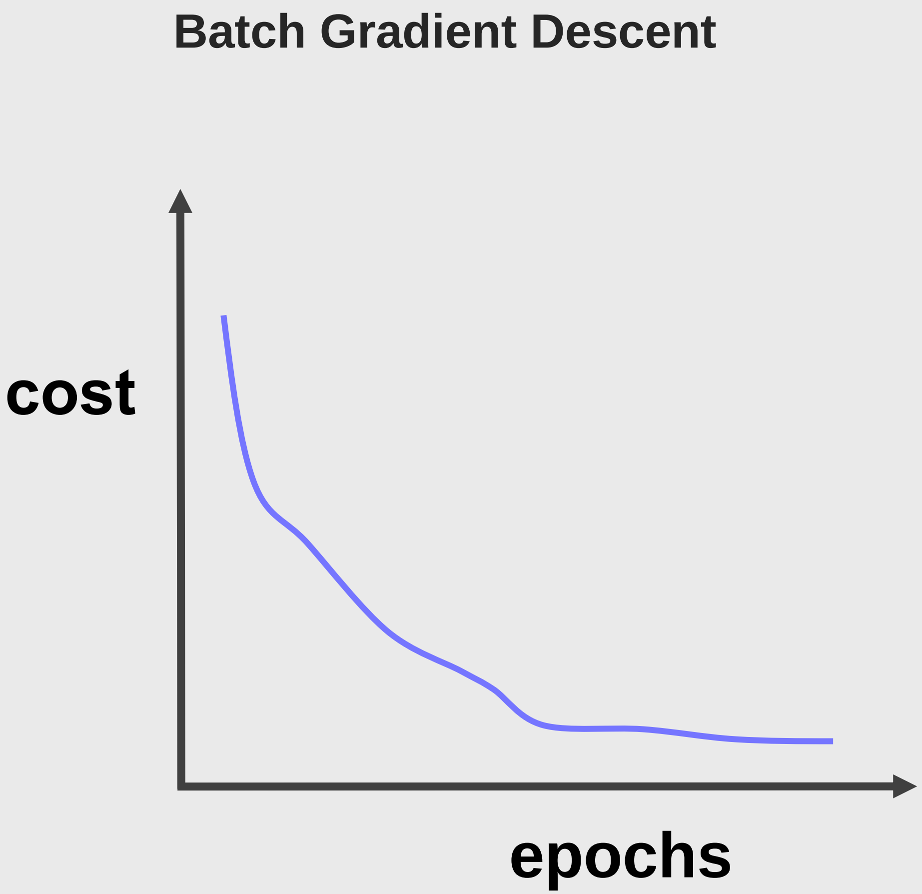
```
$t = 0$
$\mathbf{w}$ initial weights
for $t$ in epochs do
    $\mathcal{D} \leftarrow shuffle(\mathcal{D})$
    for $\mathbf{x}_j \in \mathcal{D}$ do  // for each sample
        $\nabla \mathbf{w} = \nabla \mathbf{w} + \partial E(g_{\mathbf{w}}(\mathbf{x}_j))/(\partial \mathbf{w})\mathbf{x}_j$ // gradient of error with respect to weight $\mathbf{w}_j$
    $\mathbf{w}_t = \mathbf{w}_{t-1} + \boldsymbol{\eta} \frac{\nabla \mathbf{w}}{|\mathcal{D}|}$
    $t = t + 1$
```

Sec 7.2, Deep Learning by C Bishop

Gradient Descent: Versions



Stochastic Gradient Descent



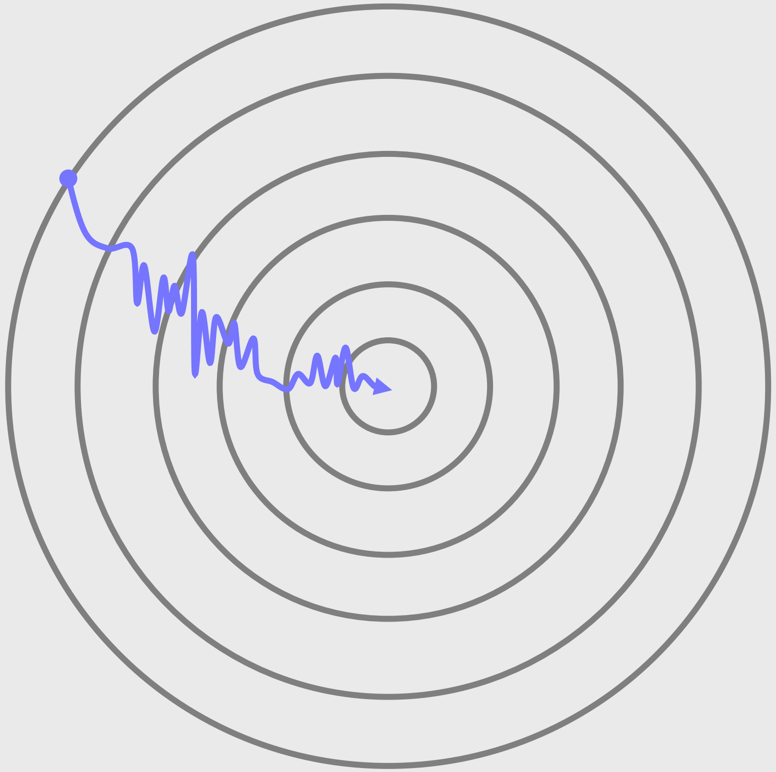
Batch Gradient Descent

Sec 7.2, Deep Learning by C Bishop

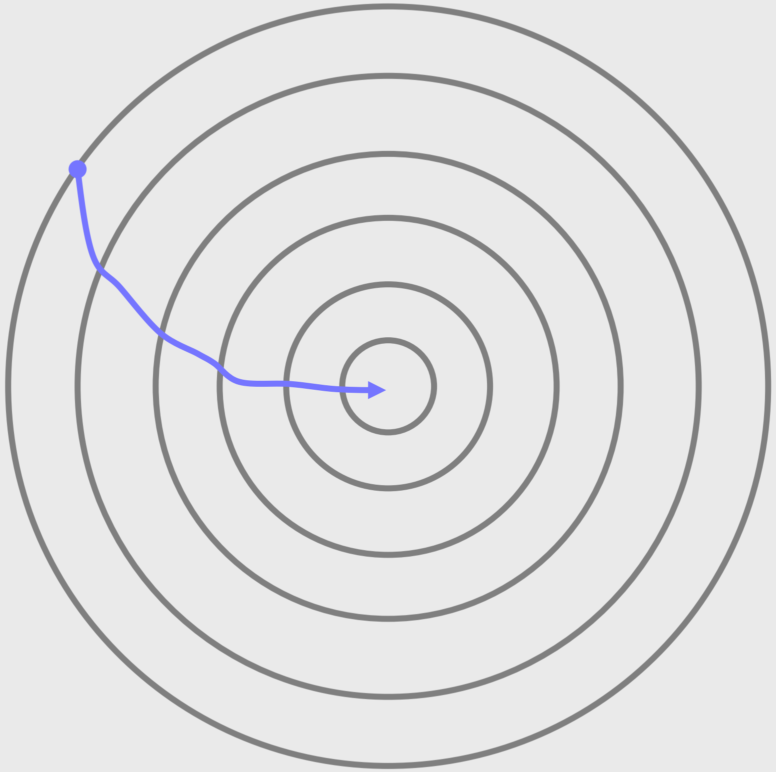
Gradient Descent: Versions

9:37 AM

Stochastic Gradient Descent



Batch Gradient Descent



Sec 7.2, Deep Learning by C Bishop

Training Method

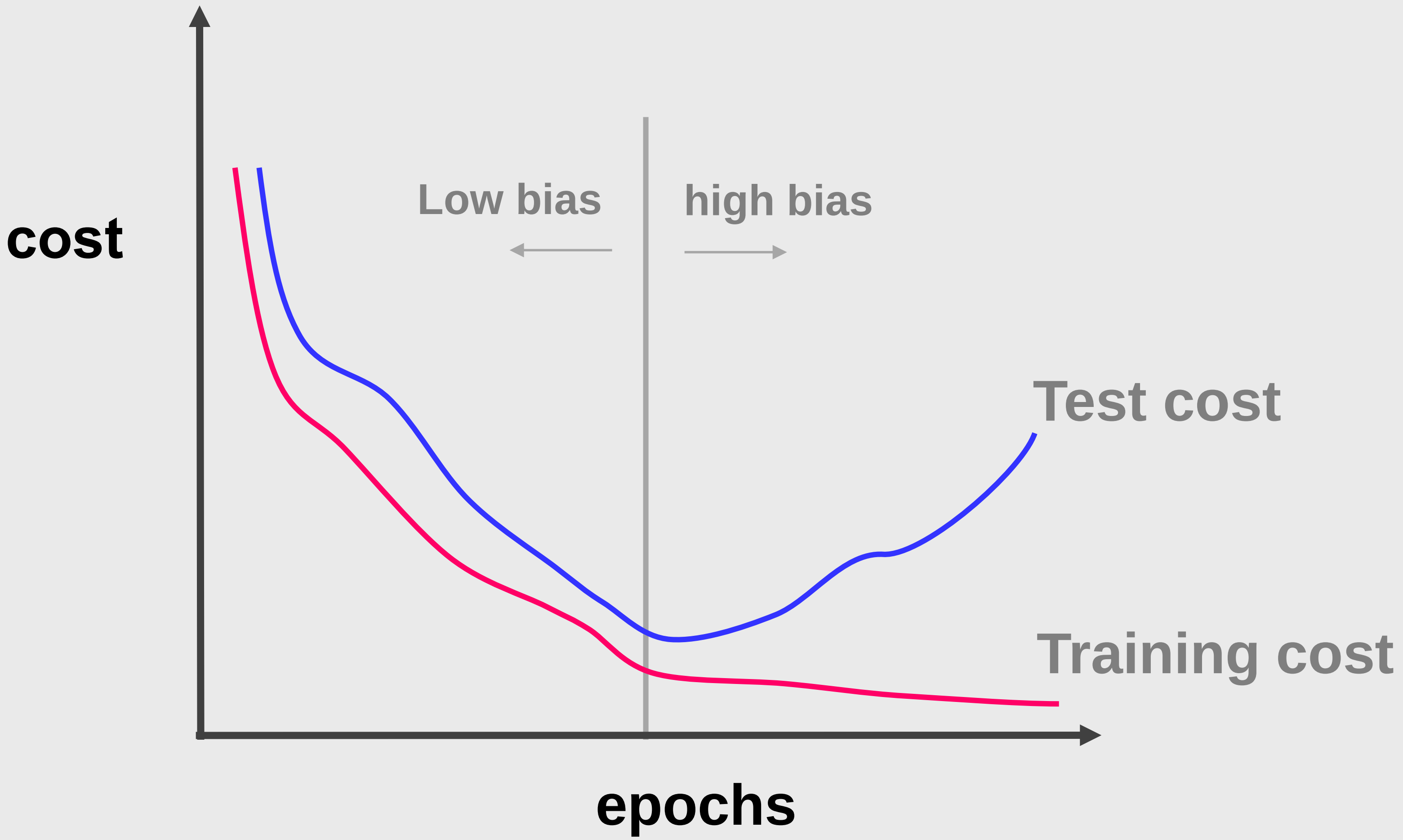
(from previous lecturers)

Training Set

Test Set

Training Method

(from previous lecturers)



Bias-Variance Issue

(from previous lecturers)

9:37 AM

Accuracy vary marginally on trials

Accuracy vary a lot on trials

low variance

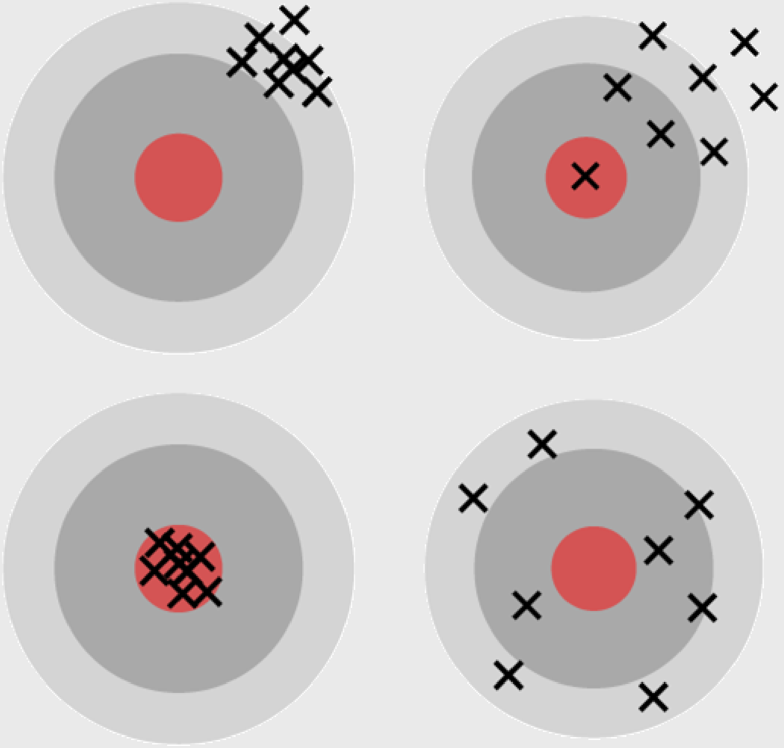
high variance

Accuracy is off by large margin

high bias

Accuracy is off by small margin

low bias

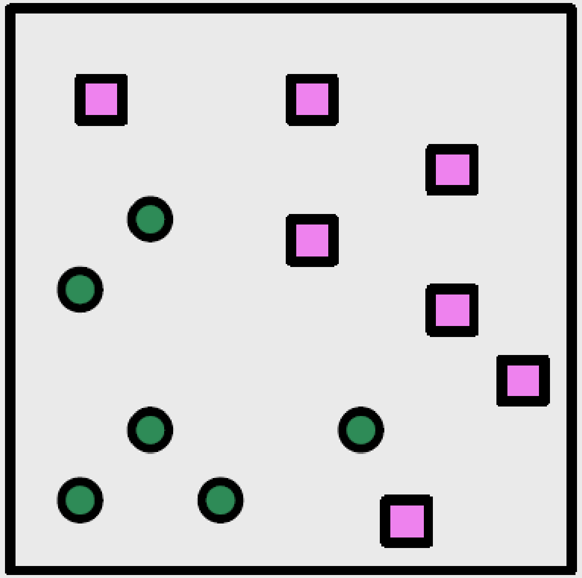


Sec 4.3, Deep Learning by C Bishop

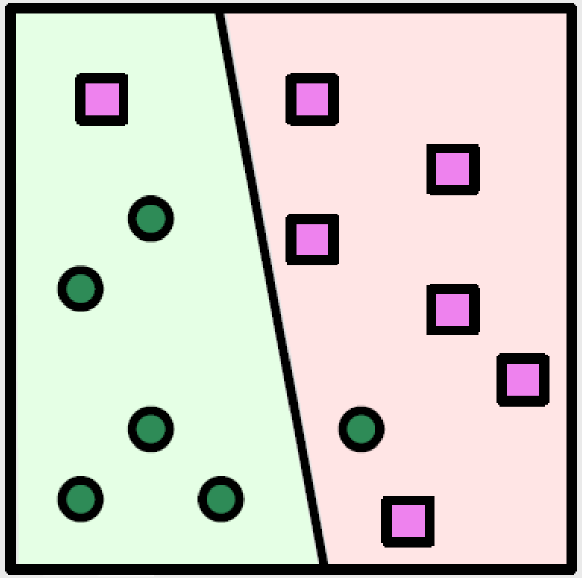
Is the chosen model good?

(from previous lecturers)

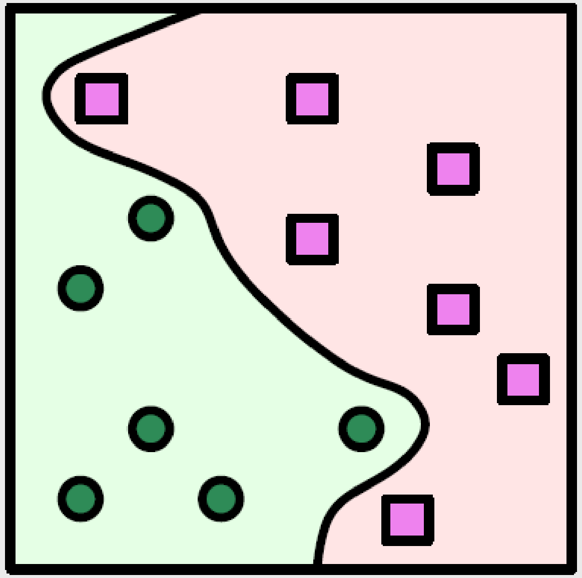
9:37 AM



Training data



Underfit



Overfit

Avoid Overfitting

(from previous lecturers)

9:37 AM

Training Set

Validation Set

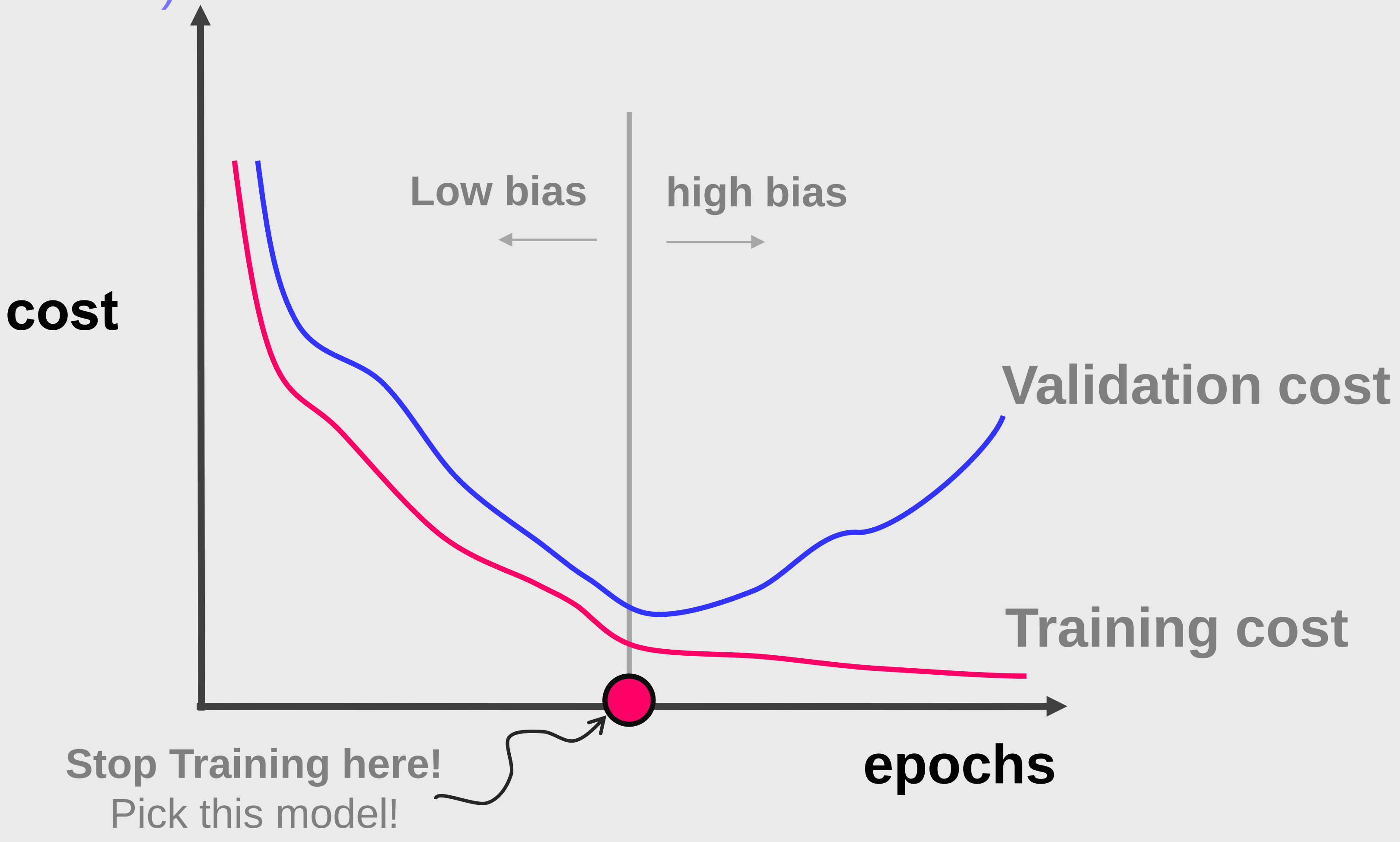
Test Set

Figure 1.6, Deep Learning by C Bishop

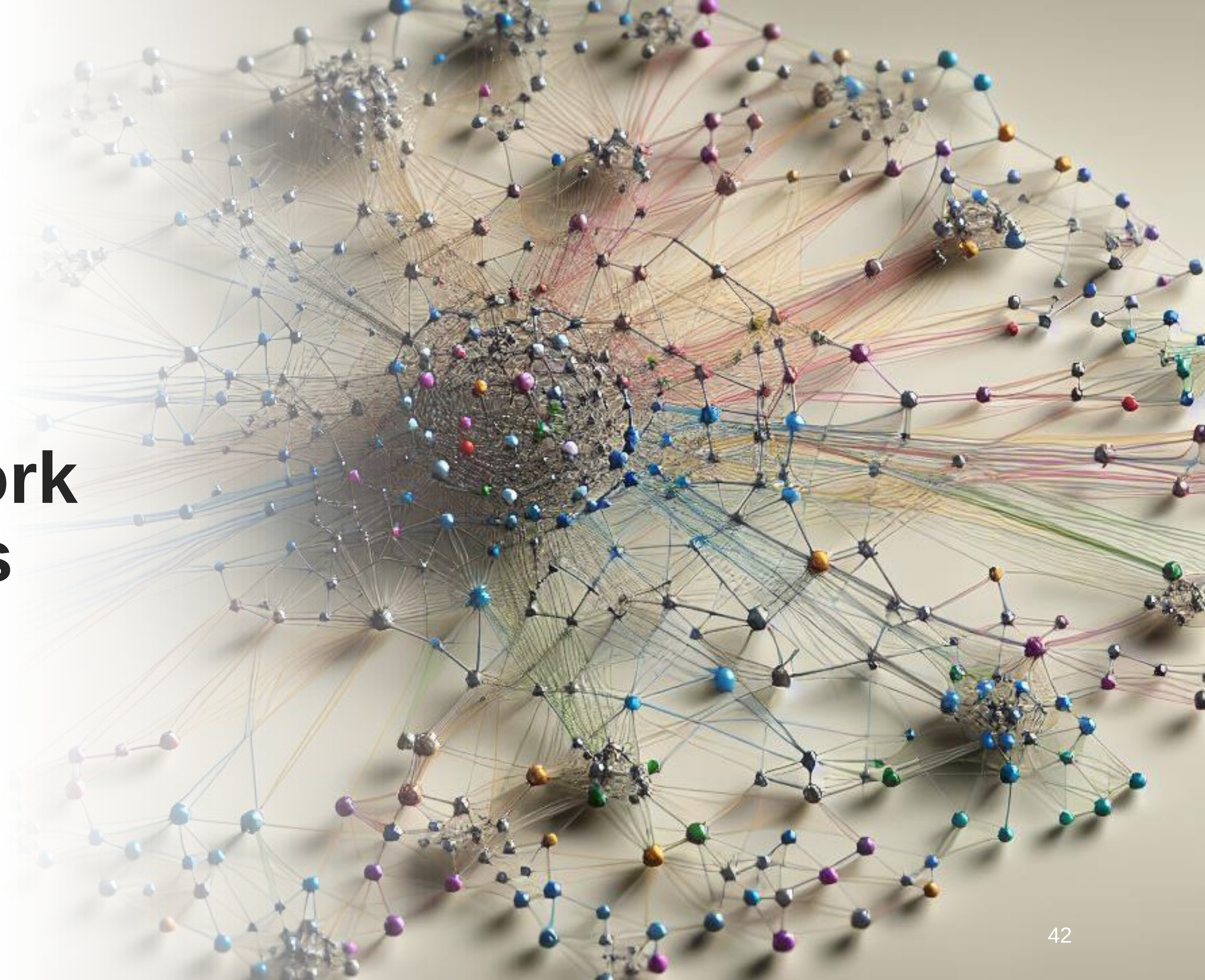
Avoid Overfitting

(from previous lecturers)

9:37 AM

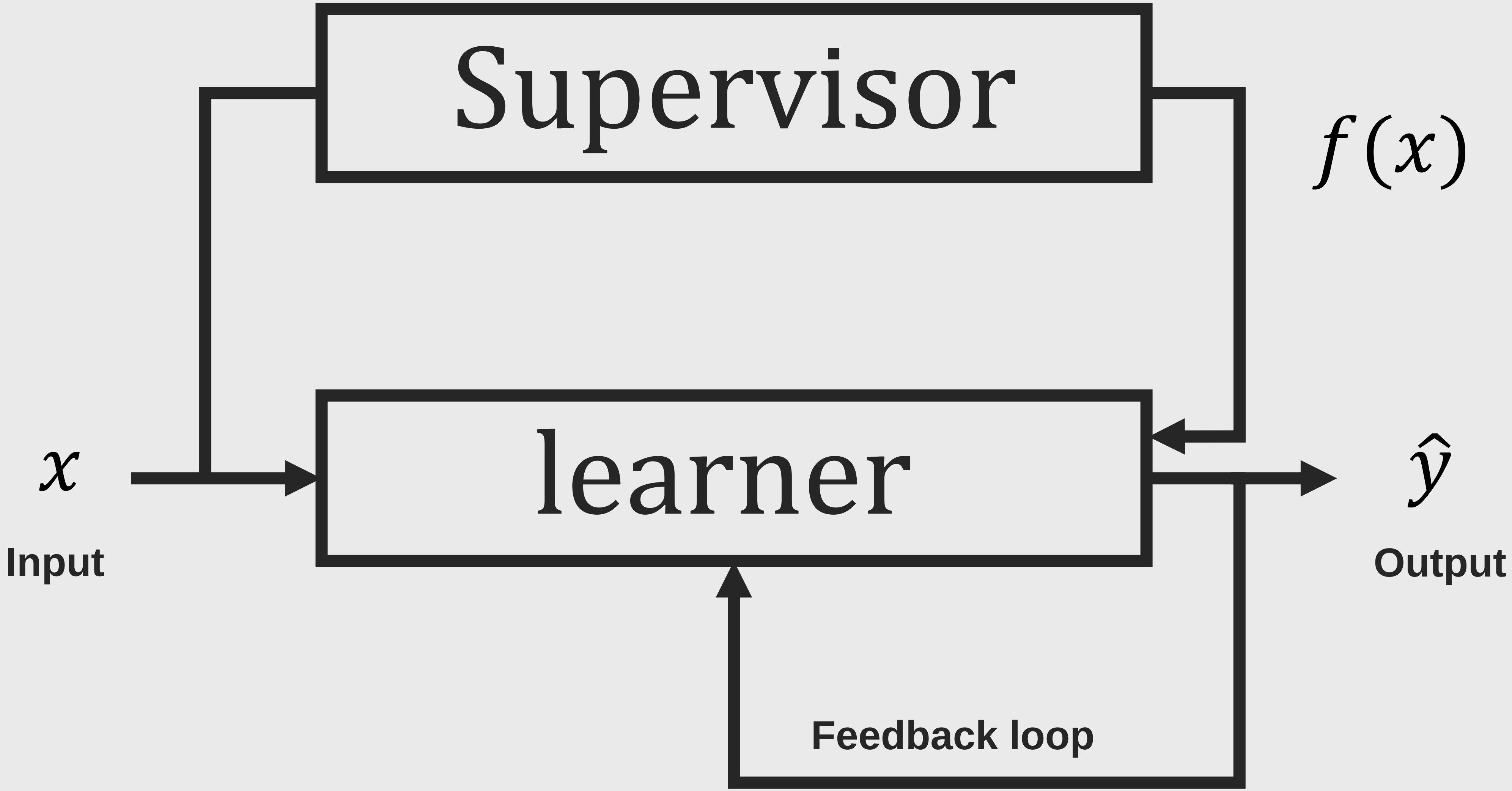


Sec 9.3.1 Deep Learning by C Bishop



Part 3

Neural Network Architectures



Supervisor

$f(x)$

x

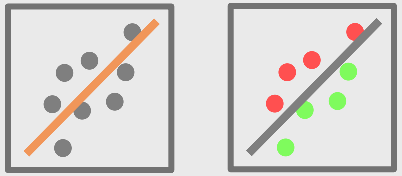
Input

learner

$\hat{y}$

Output

Feedback loop



Regression and Classification

Class/Traget attribute

	#	Inputs Attributes (Independent)		Target/Class/Output Attributes (Dependent)
		A1	**A2**	**A3**
Records	**Ex. 0**	$A1_0$	$A2_0$	$A3_0$
	Ex. 1	$A1_1$	$A2_1$	$A3_1$
	Ex. 2	$A1_2$	$A2_2$	$A3_2$
	Ex. 3	$A1_3$	$A2_3$	$A3_3$
	Ex. 4	$A1_4$	$A2_4$	$A3_4$
	Ex. 5	$A1_5$	$A2_5$	$A3_5$
	Ex. 6	$A1_6$	$A2_6$	$A3_6$
	Ex. 7	$A1_7$	$A2_7$	$A3_7$
	Ex. 8	$A1_8$	$A2_8$	$A3_8$
	Ex. 9	$A1_9$	$A2_9$	$A3_9$

Target (Class) Attributes (A3)

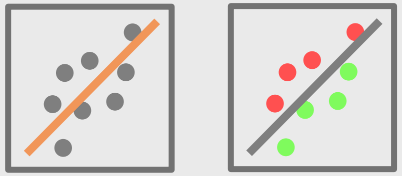
Regression

Continuous **(Numerical)** labeled data

Classification

Discrete **(Categorical)** labeled data

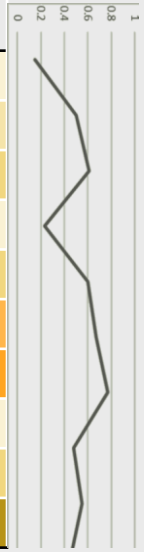
(from previous lecturers)



Tasks: Regression and Classification

Continuous
labeled data

#	Inputs (X)		Target (Y)	
	Area (m^2)	Distance(mile)	Price (£Bn)	
Ex. 0	76.85	17.27	0.15	🟨
Ex. 1	76.97	19.54	0.5	🟨
Ex. 2	77.10	18.51	0.76	🟨
Ex. 3	85.28	46.09	0.23	🟨
Ex. 4	85.42	35.83	0.6	🟨
Ex. 5	88.02	2.59	0.67	🟧
Ex. 6	77.25	6.34	0.89	🟧
Ex. 7	77.49	6.98	0.2	🟨
Ex. 8	85.81	12.18	0.55	🟨
Ex. 9	98.81	2.18	9.45	🟫



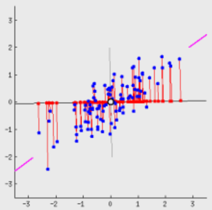
Discrete
labeled data

#	Inputs (X)		Class (Y)	
	Length (cm)	Weight (kg)	Sales	
Ex. 0	23.2	3.2	Good	🟥
Ex. 1	70.9	19.5	Bad	🟩
Ex. 2	60.5	18.51	Bad	🟩
Ex. 3	24.5	4.6	Good	🟥
Ex. 4	110.0	35.83	Bad	🟩
Ex. 5	23.8	3.7	Good	🟥
Ex. 6	25.8	4.5	Good	🟥
Ex. 7	24.7	4.9	Good	🟥
Ex. 8	85.8	25.6	Bad	🟩
Ex. 9	78.8	20.33	Bad	🟩

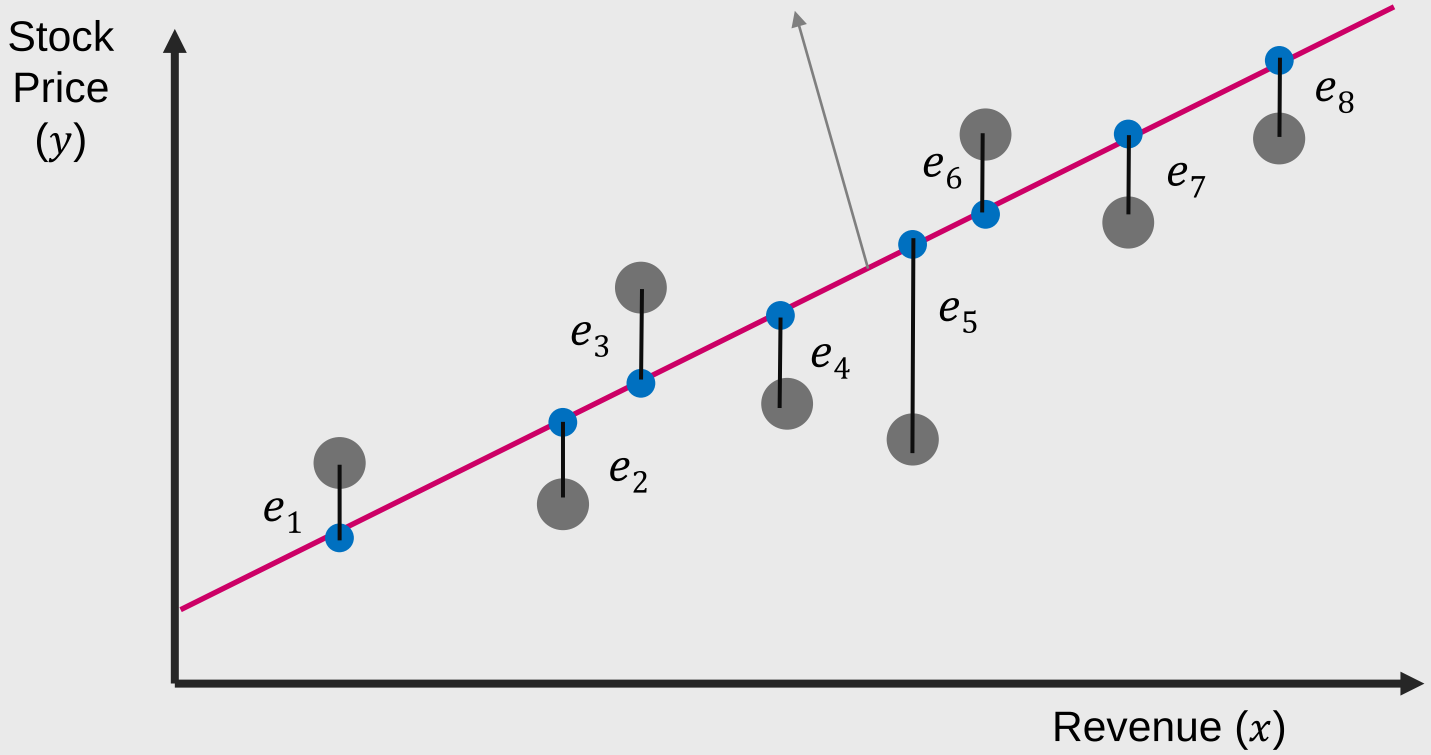
(from previous lecturers)

9:37 AM

Regression



$$\hat{y} = f(x) = w_1 x + w_0$$



- ✓ Best Fit
- ✓ Find the line (parameters of a line equation) that minimize the norm of the *y* errors
- ✓ (sum of the squares)

Error

$$e_i = \hat{y}_i - y_i$$

$$\boldsymbol{e} = \sum_{i=1}^{8} (\hat{y}_i - y_i)^2$$

(from previous lecturers)

Loss function: Mean Squared Error, $\boldsymbol{E}$

$$\boldsymbol{E} = \frac{1}{n}\sum_{i=1}^{n}(\widehat{y}_i - y_i)^2$$

$\widehat{y}_i$ - predicted output
y_i - target output
n - number of examples in training/test set

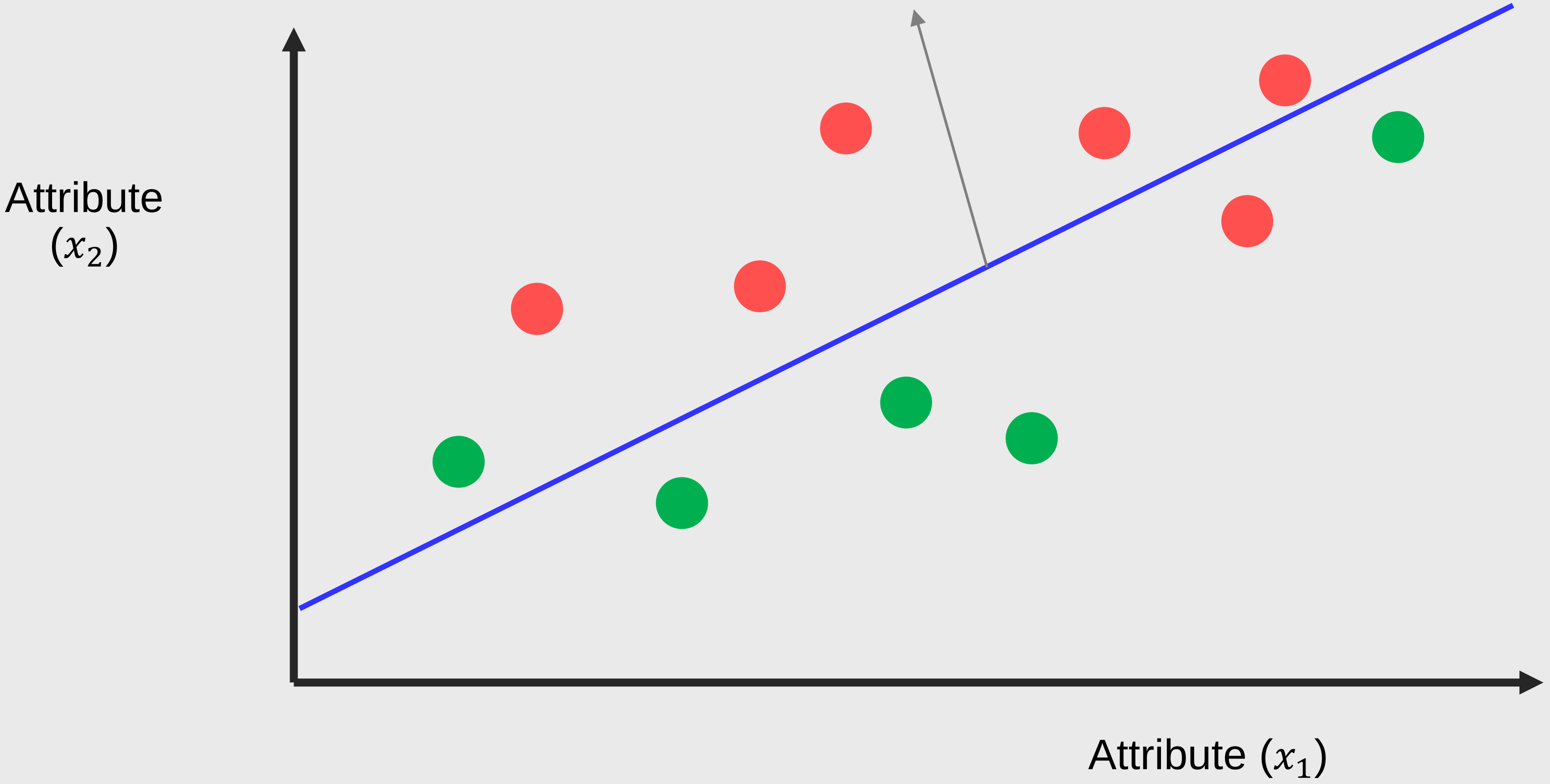
Loss function: Mean Absolute Error, $\boldsymbol{E}$

$$\boldsymbol{E} = \frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i|$$

$\hat{y}_i$ - predicted output
y_i - target output
n - number of examples in training/test set

Classification

$$\hat{y} = f(x) = w_1 x + w_0$$

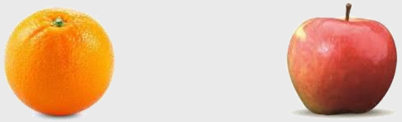


- ✓ Best Fit
- ✓ Find the line (parameters of a line equation) that minimize the error (misclassification) rate

$$\mathbf{e} = \frac{1}{n}\sum_{i=1}^{n} \hat{y}_i \neq y_i$$

(from previous lecturers)

Loss function: Misclassification rate, $\boldsymbol{E}$



$$\boldsymbol{E} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i \neq y_i)$$

$\hat{y}_i$ - predicted output
y_i - target output
n - number of examples in training/test set

Loss function: Log loss

It effectively works by updating network weights on correct classification and penalizing models for a misclassification

$$-\log \mathrm{P}(y_i \mid \hat{y}_i) = -\left(\overbrace{\boxed{(\boldsymbol{y_i})\mathbf{log}(\hat{y}_i)}}^{\text{This part will be zero if } y_i = 0} + \overbrace{\boxed{(\mathbf{1} - \boldsymbol{y_i})\mathbf{log}(\mathbf{1} - \hat{y}_i)}}^{\text{This part will be zero if } y_i = 1} \right)$$

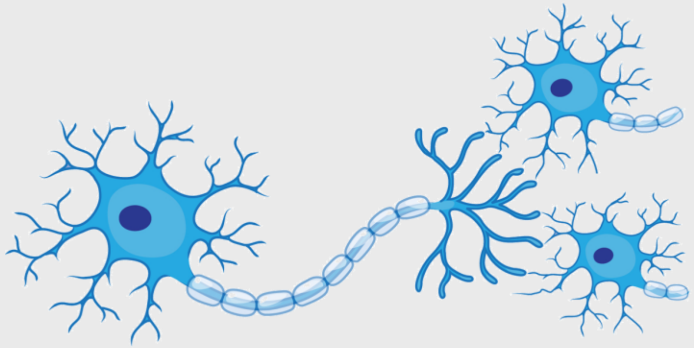
$\hat{y}_i$ - predicted output
y_i - target output
n - number of examples in training/test set

Cross Entropy, $\boldsymbol{E}$

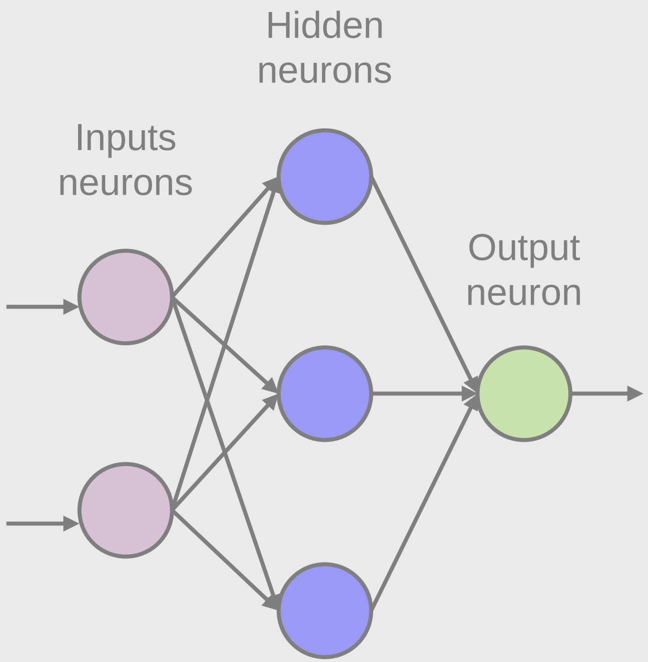
$$E = -\sum_{i=1}^{C} \boldsymbol{y_i} \mathbf{log}(\boldsymbol{\hat{y}_i})$$

$\hat{y}_i$ - predicted output distribution (SoftMax output)
y_i - target output (target out distribution , one-hot encoding)
C - number of classes

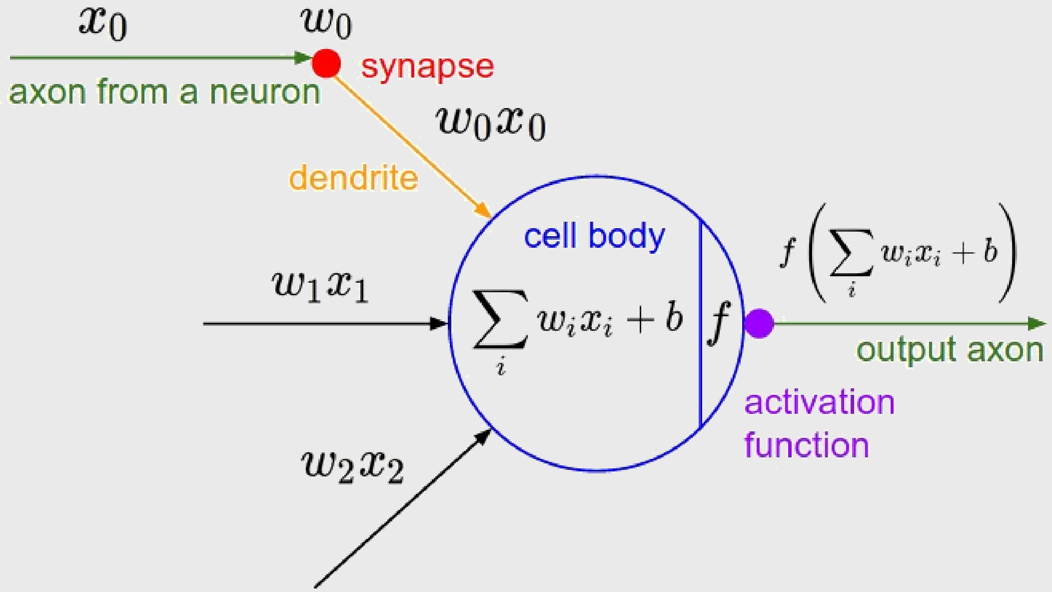
Neural Networks



Biological networks of
neurons in human brains

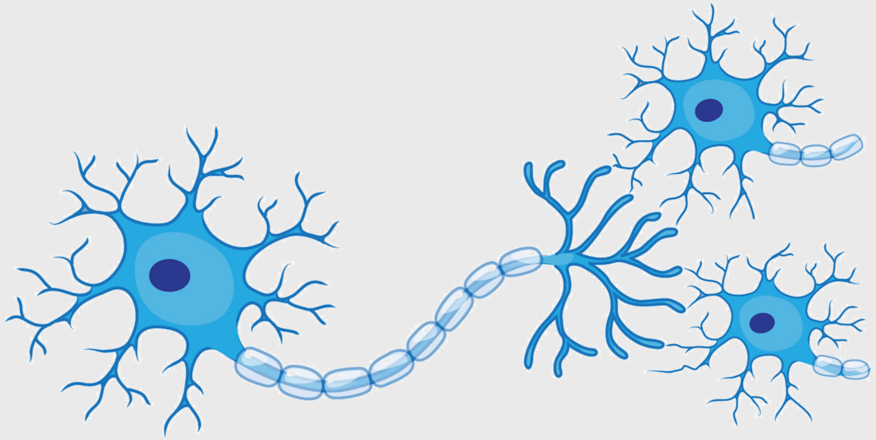


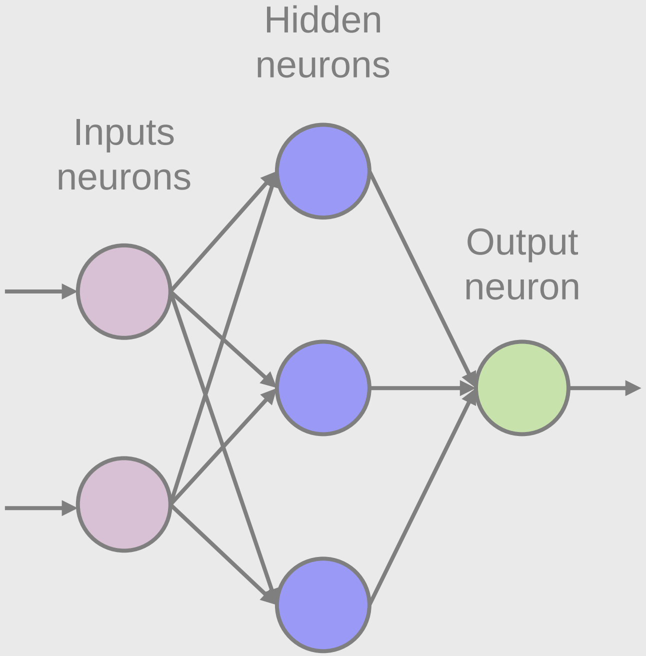
AI representation
of biological neural networks

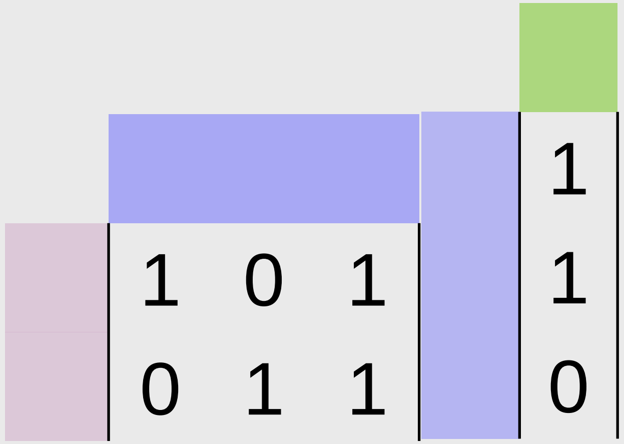


9:37 AM

Neural Networks







$$\begin{bmatrix}1 & 0 & 1\\0 & 1 & 1\end{bmatrix}\begin{bmatrix}1\\1\\0\end{bmatrix}$$

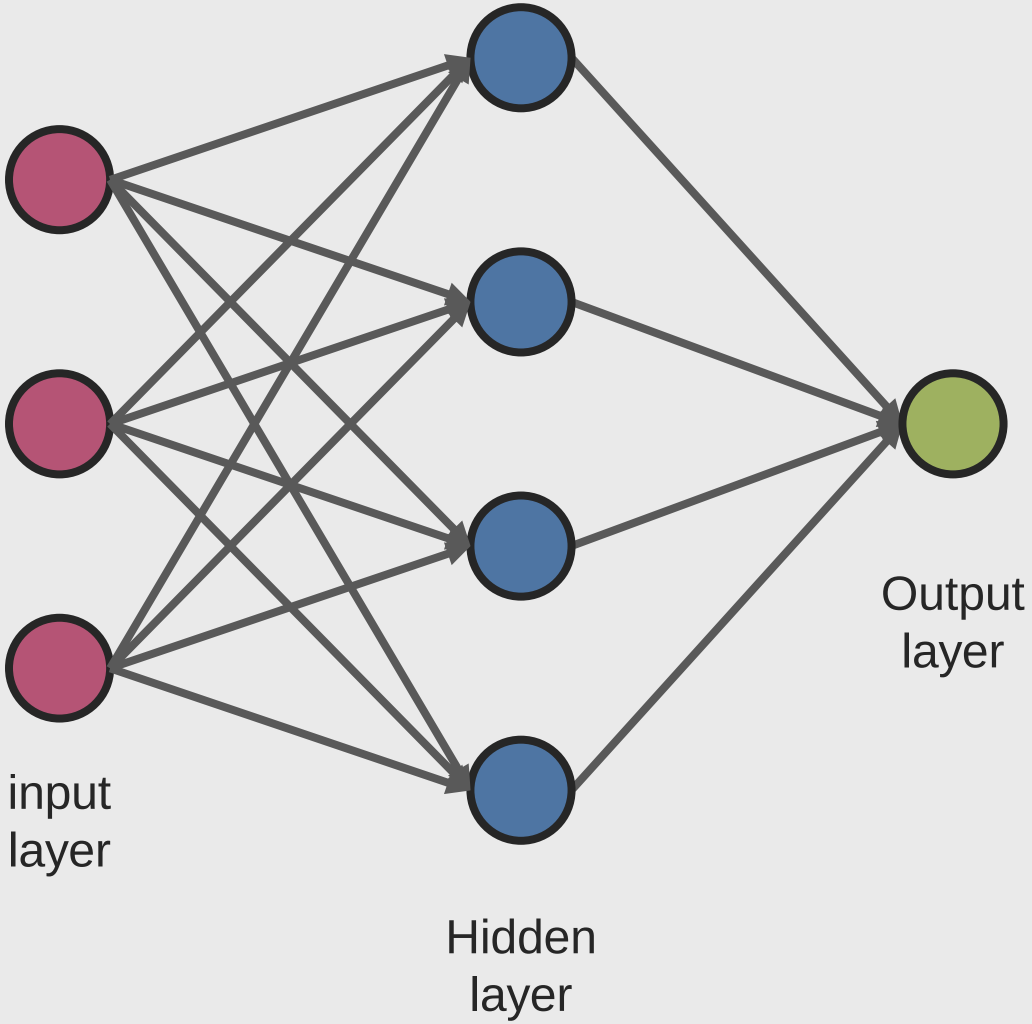
1 Biological networks of neurons in human brains

2 AI representation of biological neural networks

3 Mathematical representation of the neural networks

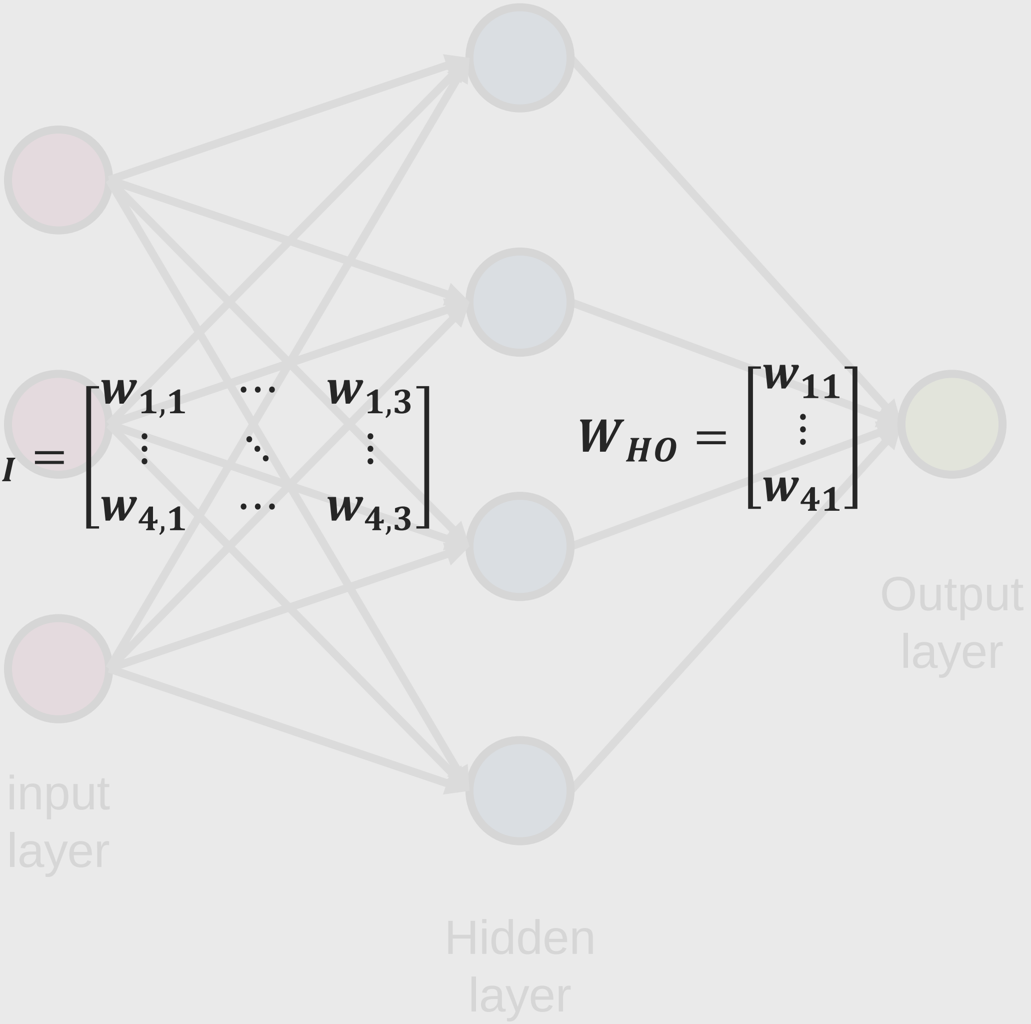
NEURAL NETWORK

Architecture



NEURAL NETWORK

Weights (parameters)

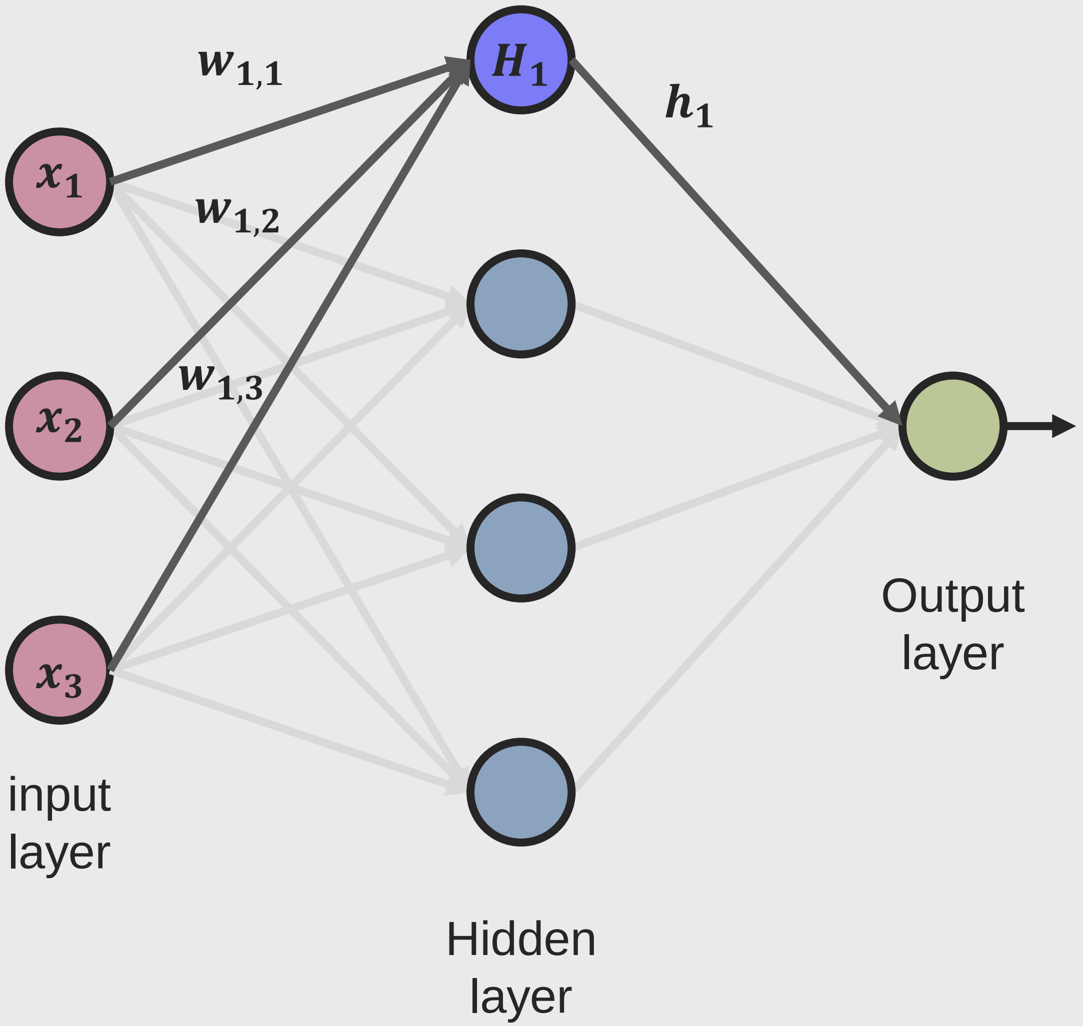


$$W_{HI} = \begin{bmatrix} w_{1,1} & \cdots & w_{1,3} \\ \vdots & \ddots & \vdots \\ w_{4,1} & \cdots & w_{4,3} \end{bmatrix}$$

$$W_{HO} = \begin{bmatrix} w_{11} \\ \vdots \\ w_{41} \end{bmatrix}$$

NEURAL NETWORK

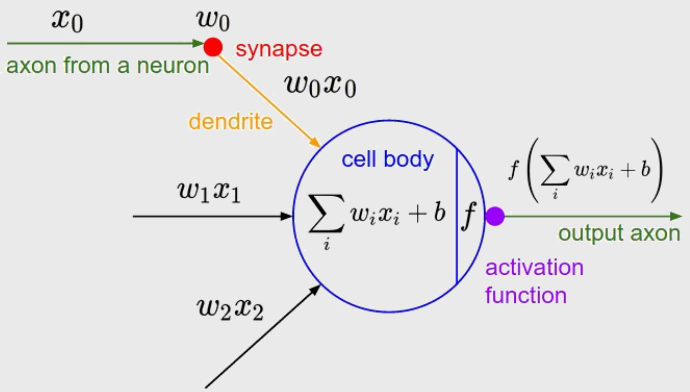
Computation: Hidden layer



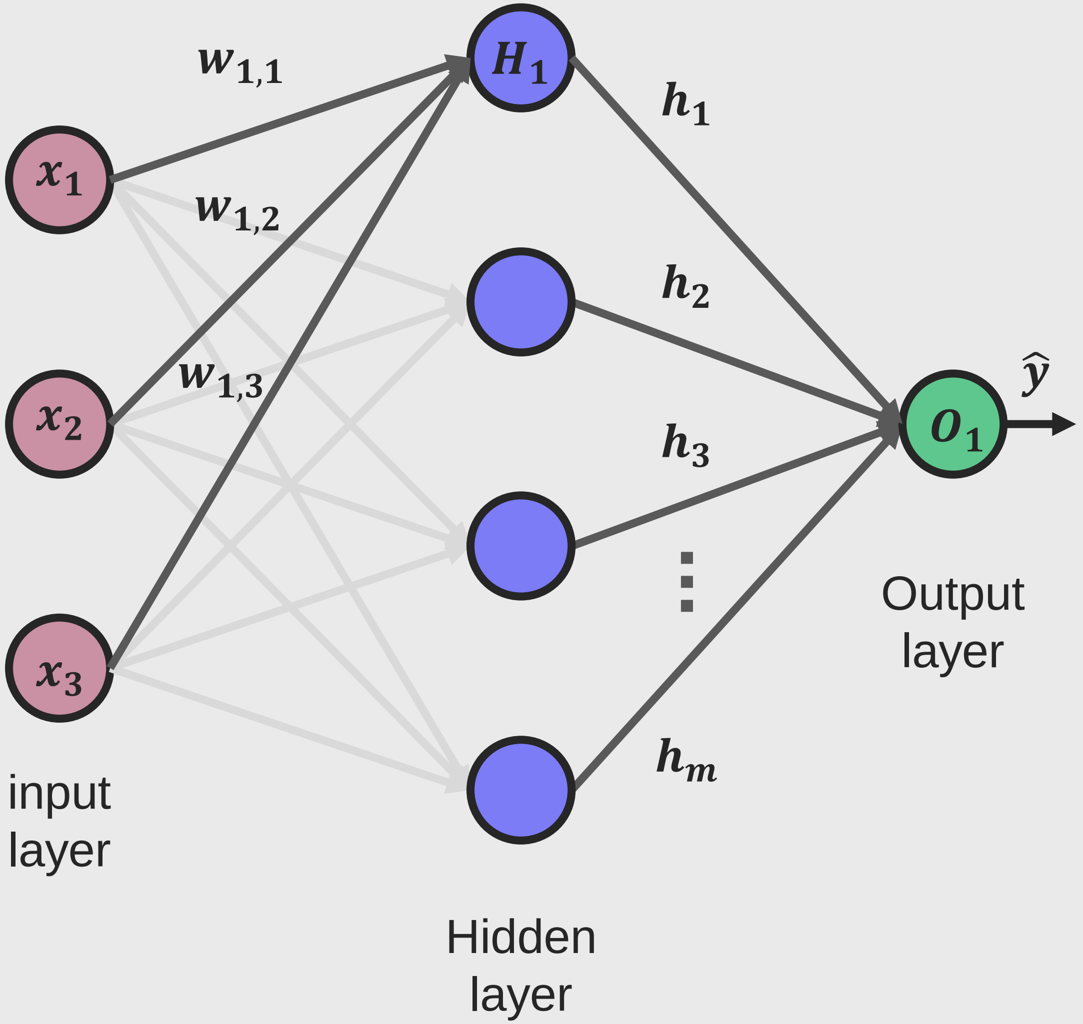
For $\boldsymbol{n}$ inputs, a hidden layer node's $\boldsymbol{h_j}$ output is expressed as:

$$h_j = \varphi_h\left(\sum_{i=1}^{n} w_{ji}.x_i\right)$$

Where $\boldsymbol{\varphi_h}$ is an activation function:



9:37 AM



For n inputs, a hidden layer node's h_j output is expressed as:

$$h_j = \varphi_h\left(\sum_{i=1}^{n} w_{ji}.\, x_i\right)$$

Where φ_h is an activation function:

For m hidden nodes and an output node, the output nodes output is expressed as:

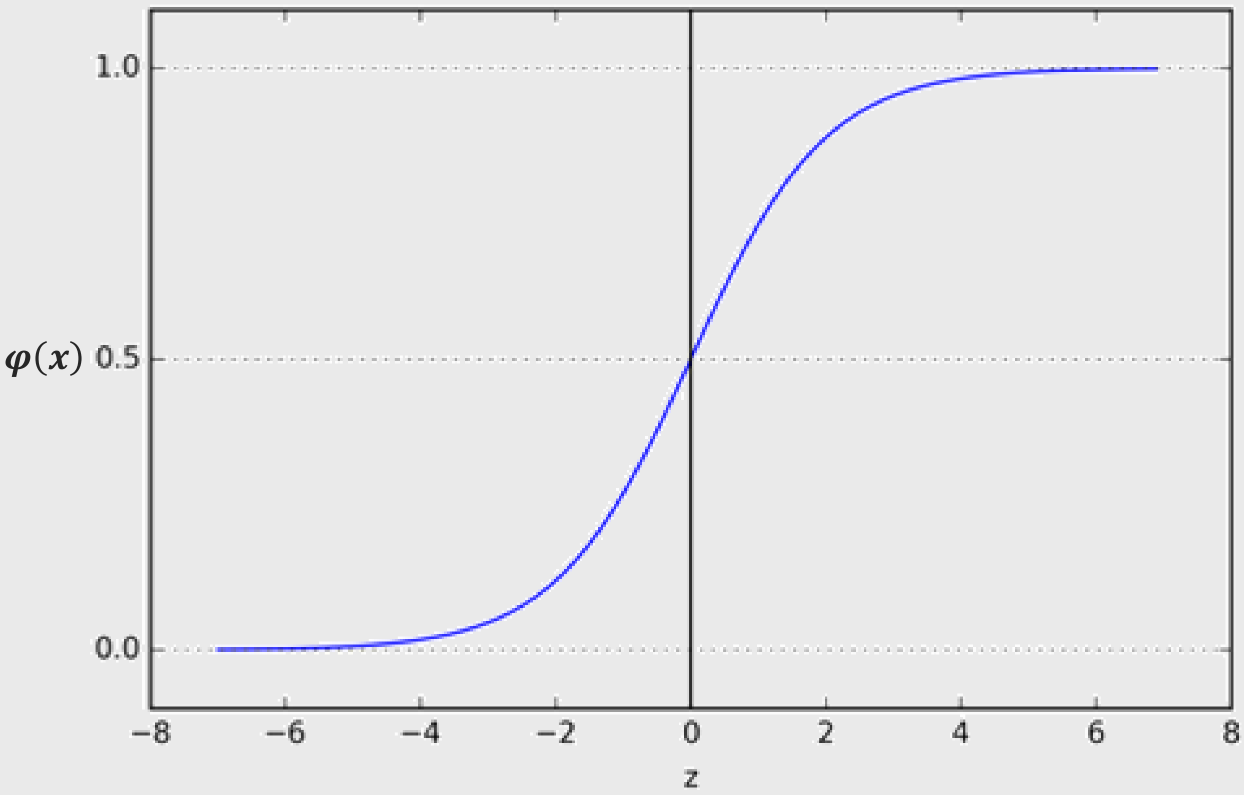
$$\hat{y} = \varphi_O\left(\sum_{j=1}^{m} w_{jk}.\, h_j\right)$$

NEURAL NETWORK

Computation: Output layer

Sigmoid activation

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

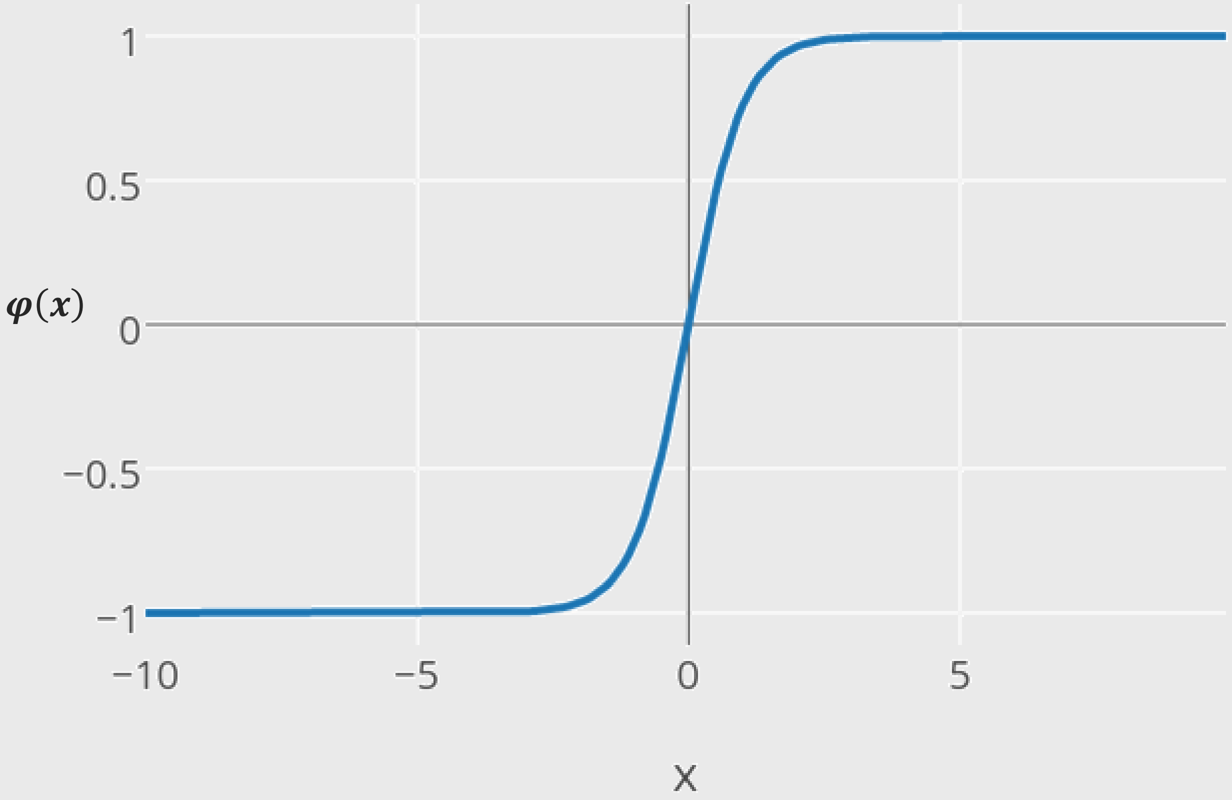


NEURAL NETWORK

Activation function

Tangent hyperbolic activation

$$\varphi(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

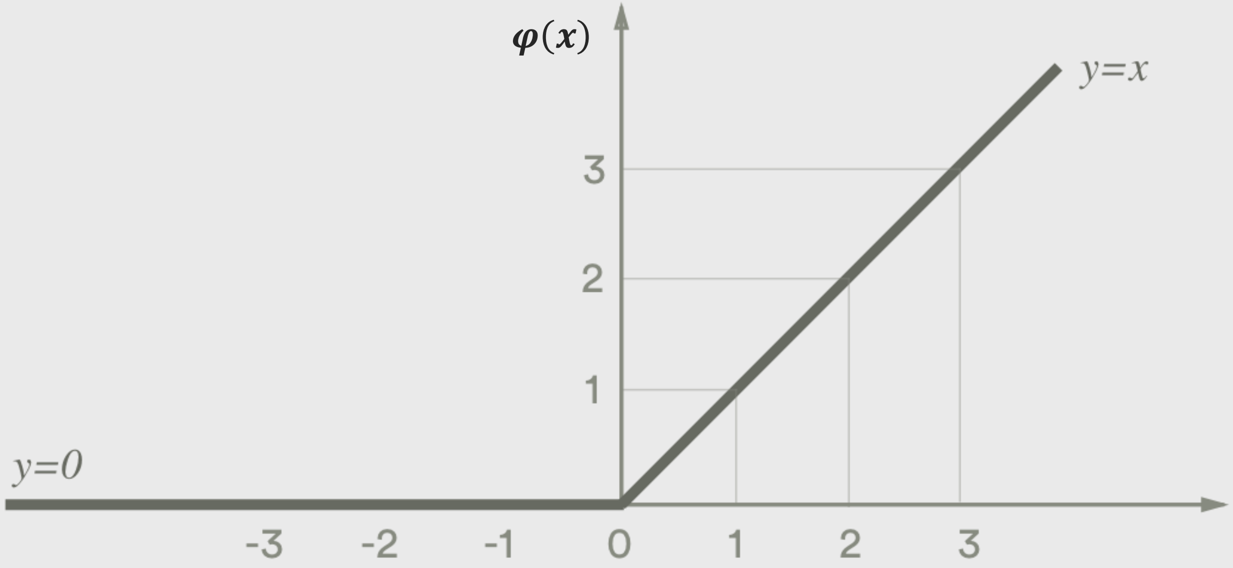


NEURAL NETWORK

Activation function

Rectified Linear Unit (ReLU)

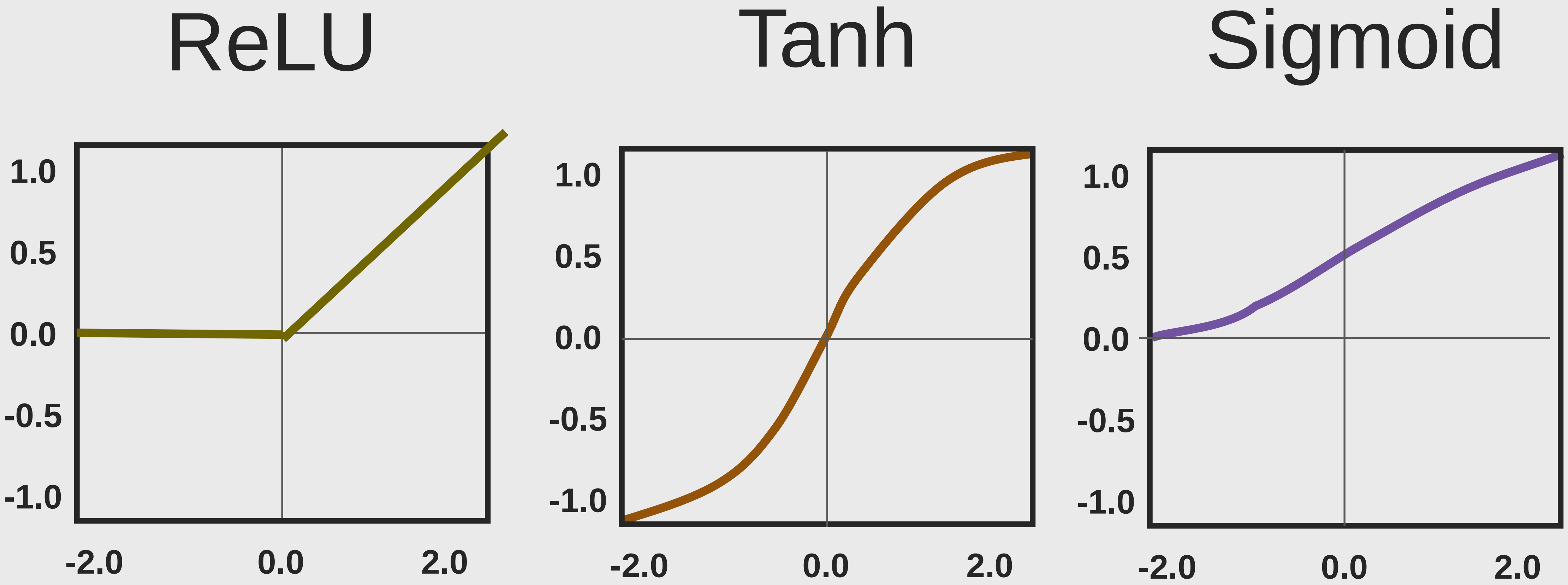
$$\boldsymbol{\varphi(x) = max\,(0, x)}$$



Source: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7

NEURAL NETWORK

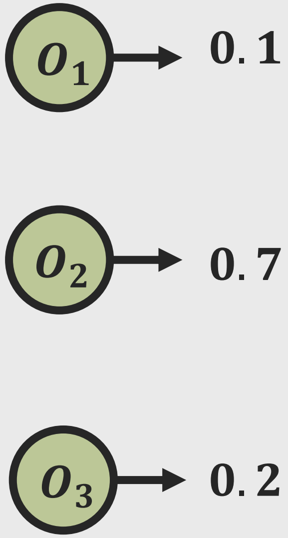
Activation functions



Sec 6.2.3, Deep Learning by C Bishop

SoftMax Activation

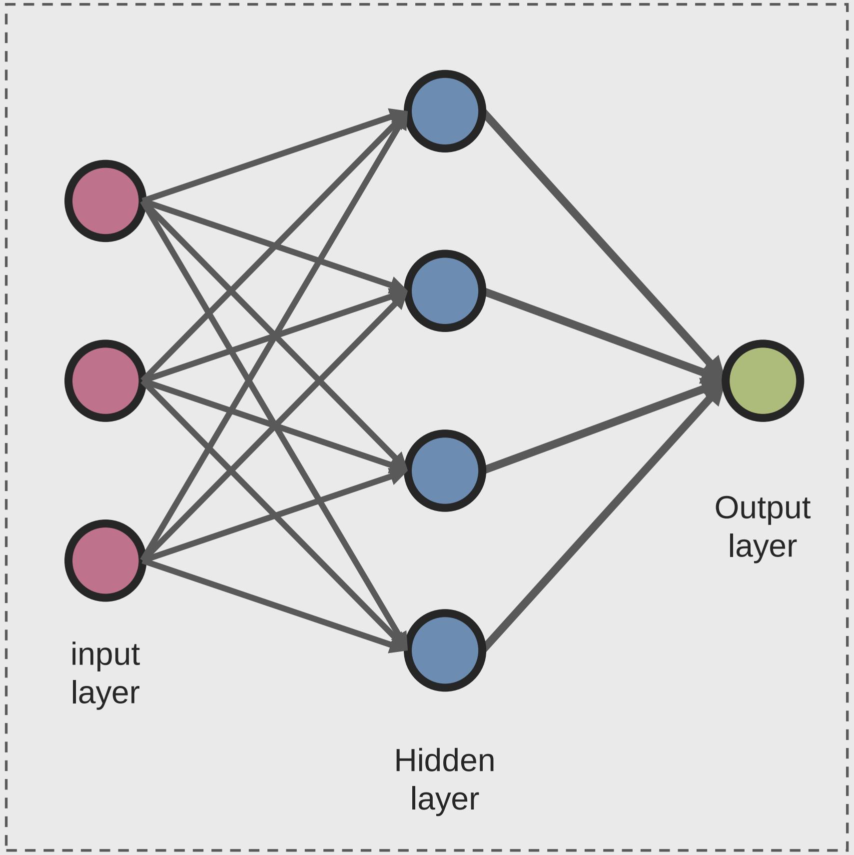
$$\boldsymbol{\varphi(x_i) = \frac{e^{x_i}}{\sum_i^k e^{x_j}}}$$ for k units



PROBABILITIES DISTRIBUTION OF ALL LABELS

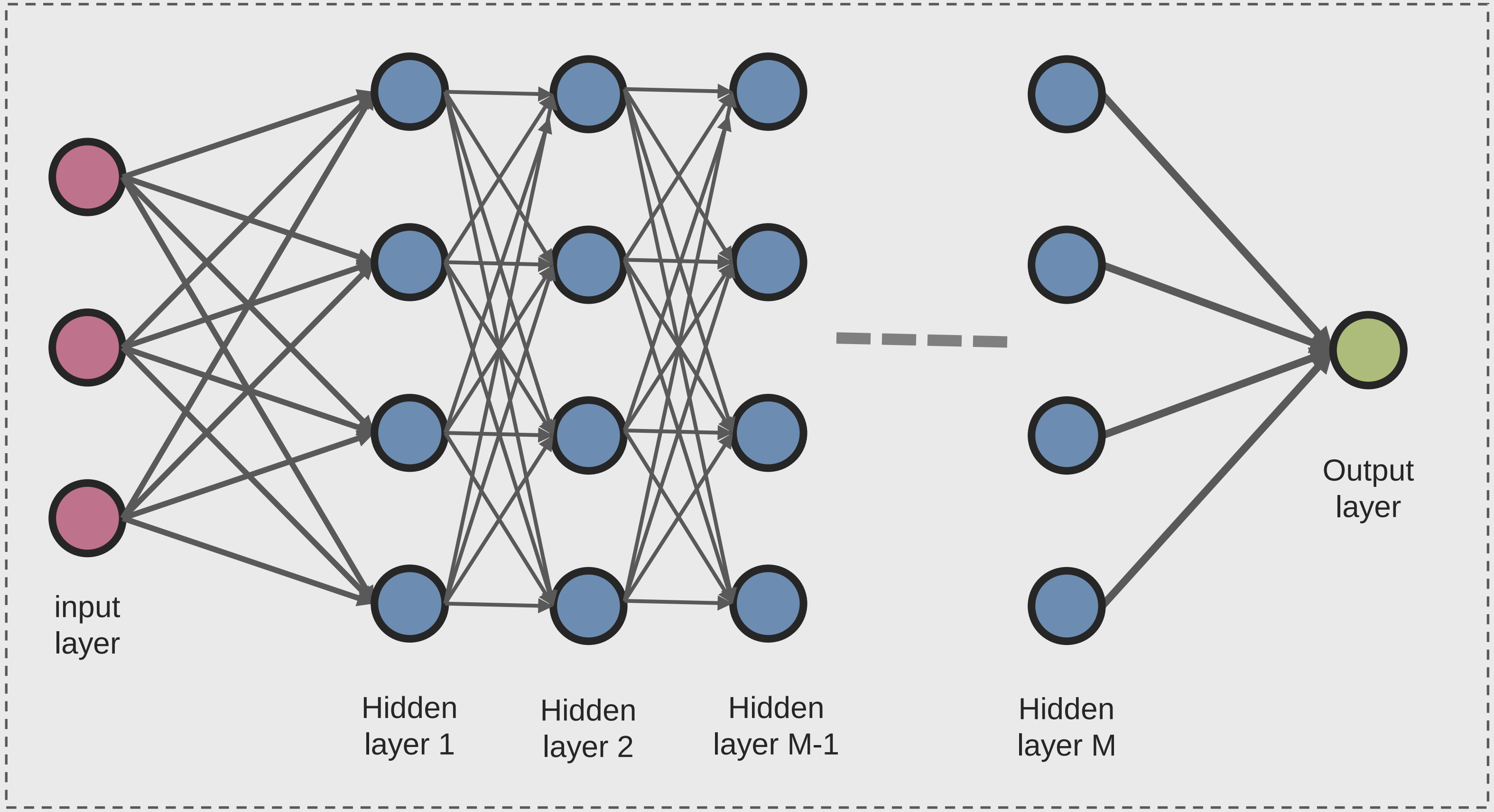
NEURAL NETWORK: Architecture

A regular neural network architecture



SHALLOW LEARNING

A deep neural network architecture



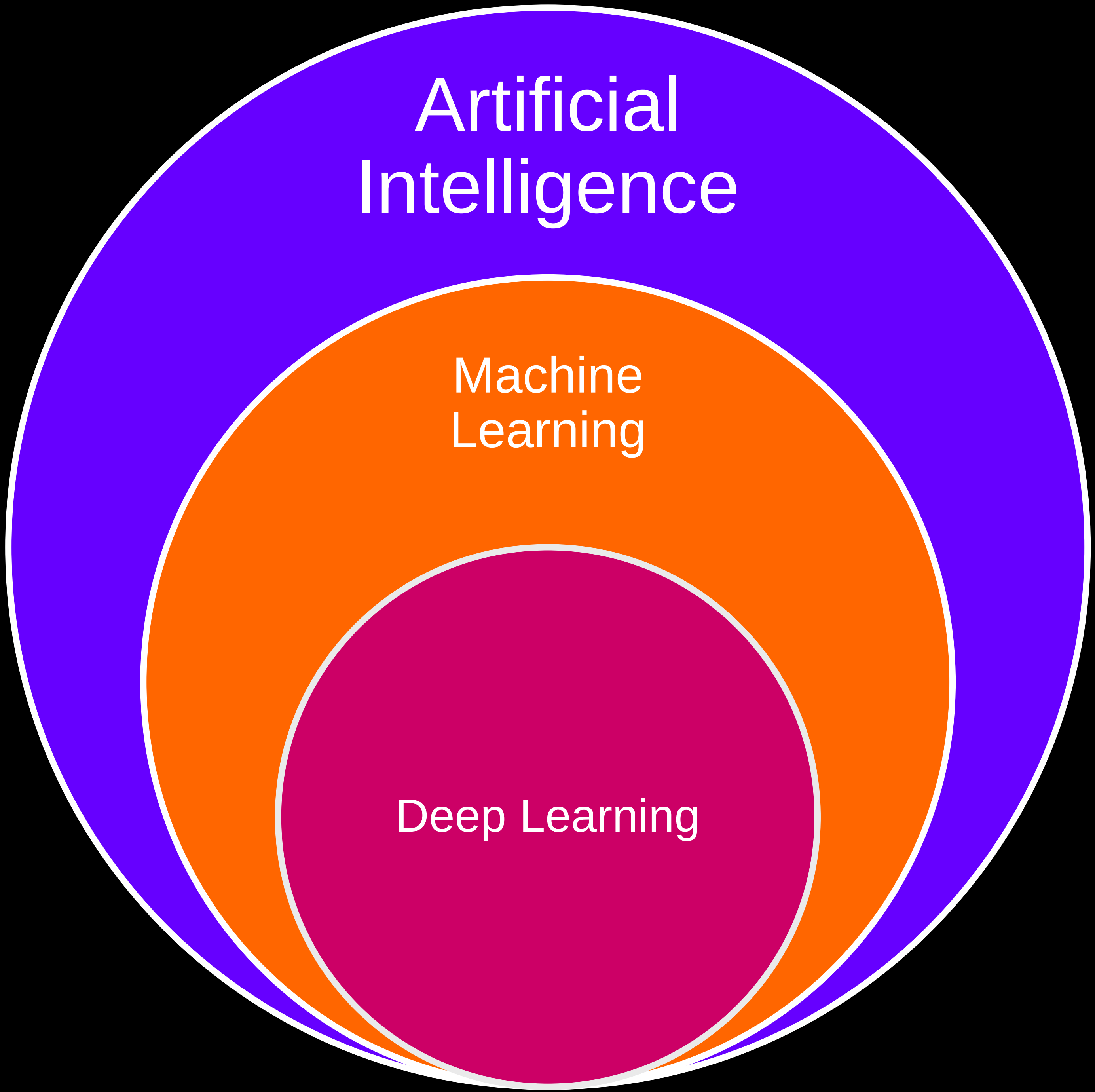
DEEP LEARNING

Deep Learning

Deep learning is an artificial intelligence function that imitates the workings of the human brain in processing data and creating patterns for use in decision making.

Deep learning is a **subset of machine learning in artificial intelligence** that has networks capable of learning supervised/unsupervised from data that is structured/unstructured or labelled/unlabelled.

Source: https://www.investopedia.com/terms/d/deep-learning.asp



Artificial Intelligence

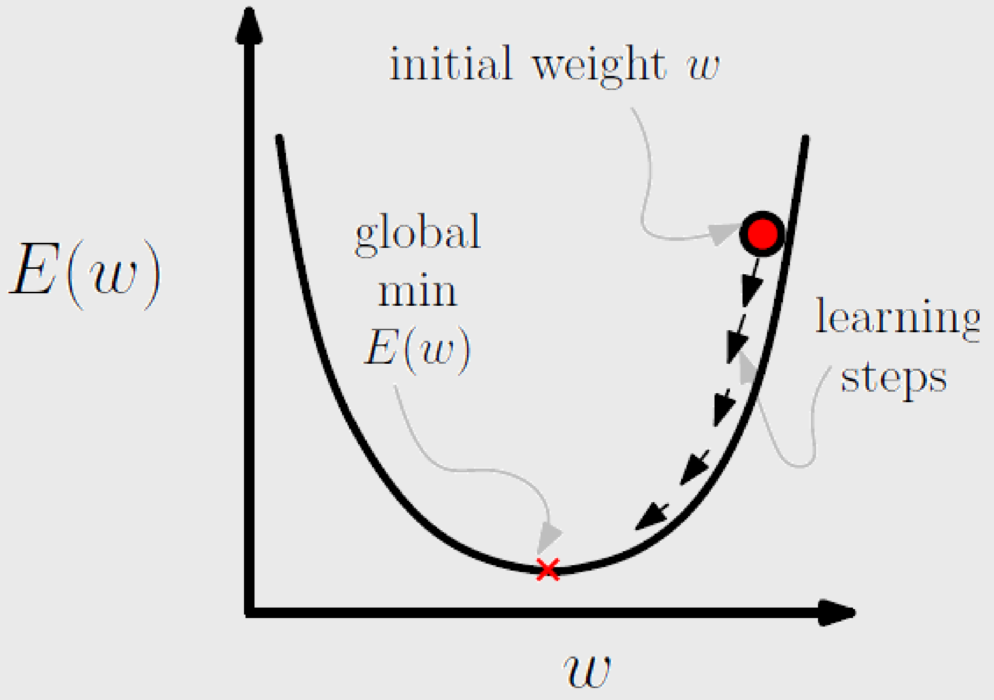
Machine Learning

Deep Learning

(DEEP) NEURAL NETWORK

Optimisation

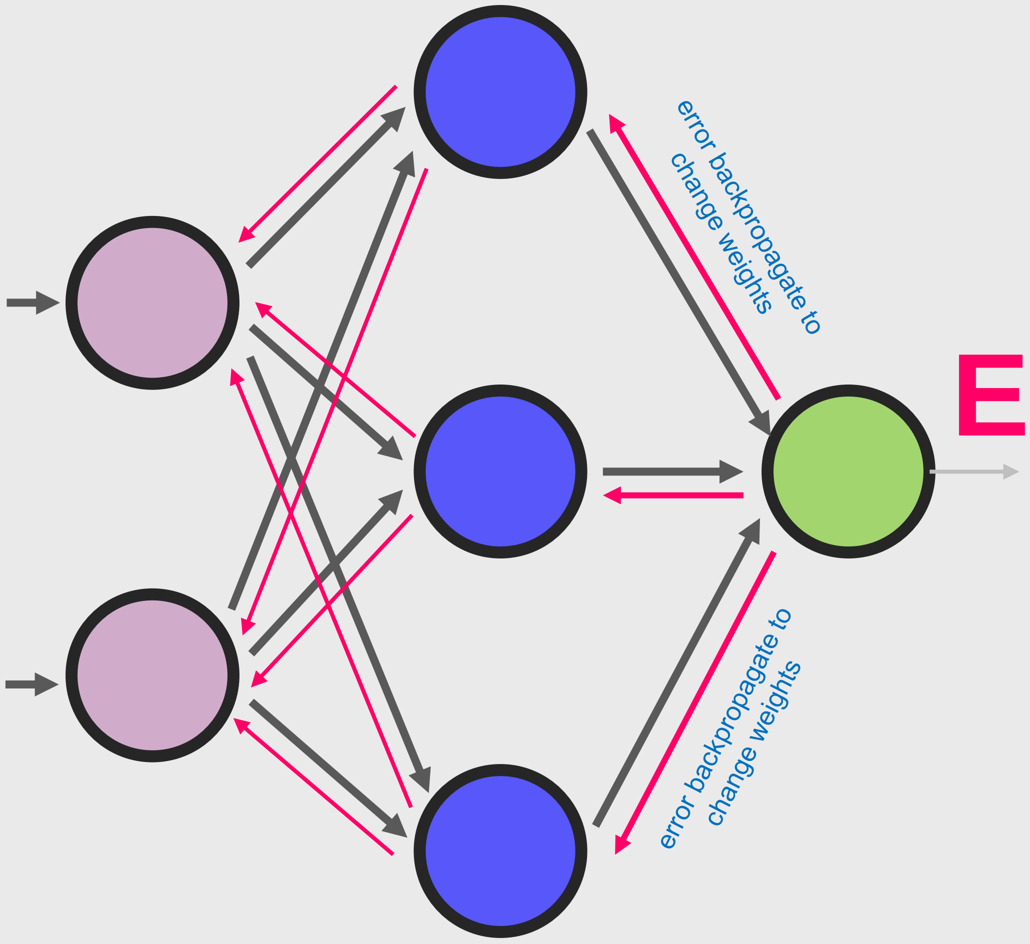
- Stochastic gradient descent
- Mini-batch gradient descent
- Batch gradient descent
- Backpropagation



9:37 AM

BACKPROPAGATION

Algorithm for Updating Learning Systems



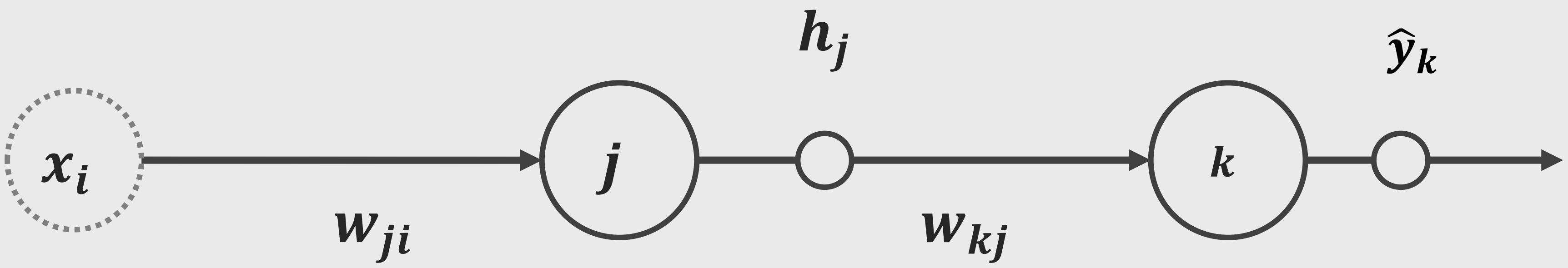
E = Output - Target

Learning Systems Update

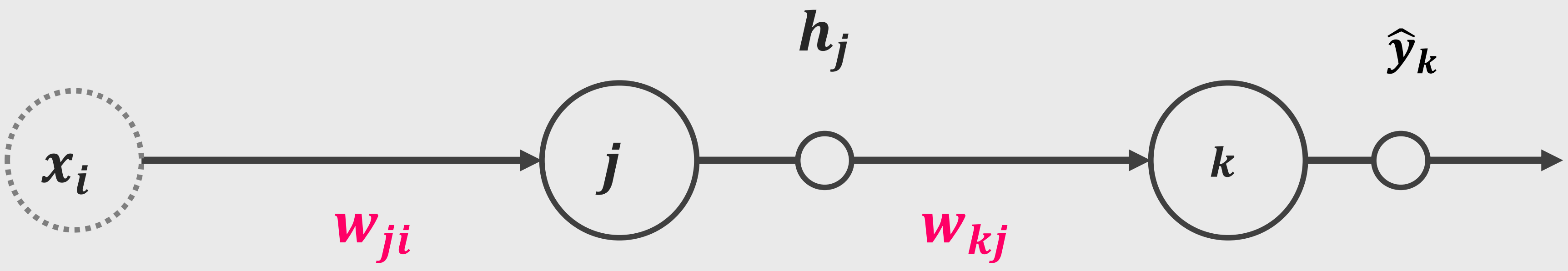
$$W_{NEW} \leftarrow W_{OLD} + W_{CHANGE}$$

Sec 8.1.1 to 8.1.3, Deep Learning by C Bishop

Backpropagation Algorithm

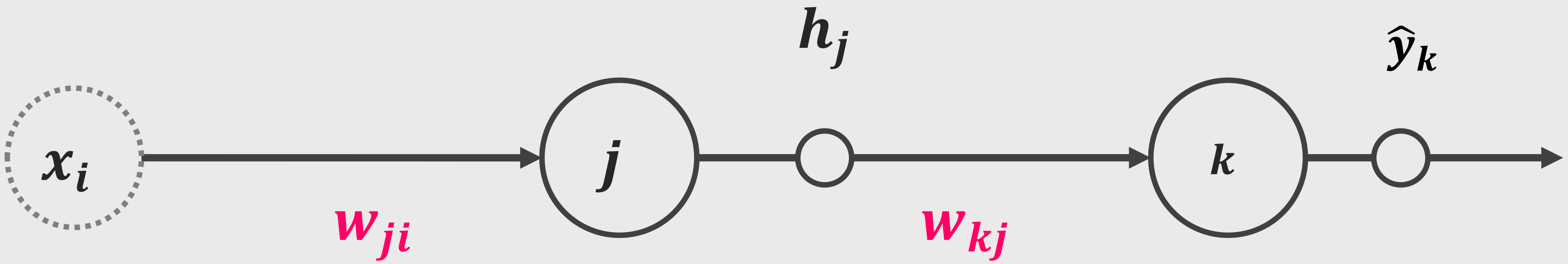


Backpropagation: **Forward Pass**



Sec 8.1.1 to 8.1.3, Deep Learning by C Bishop

Backpropagation: **Error at Output layer**

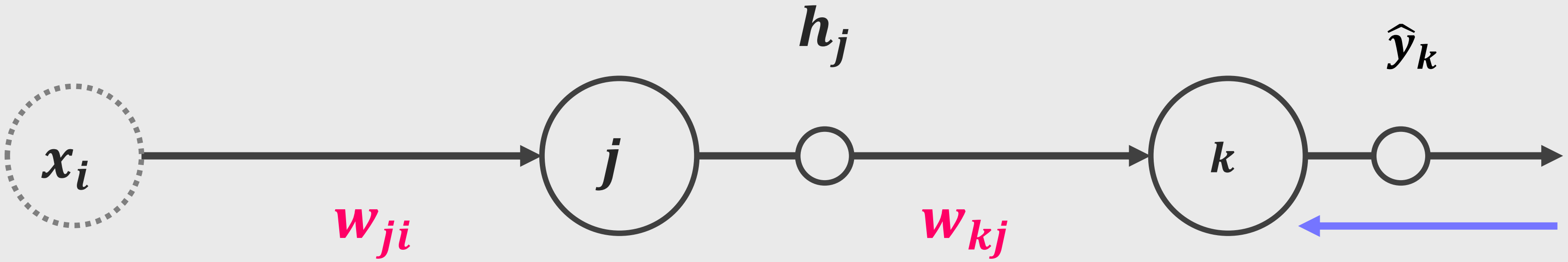


$$e_k = y_k - \hat{y}_k$$

Sec 8.1.1 to 8.1.3, Deep Learning by C Bishop

Backpropagation: **Backward pass**

Output layer delta (δ_k) considering sigmoidal output node(s)



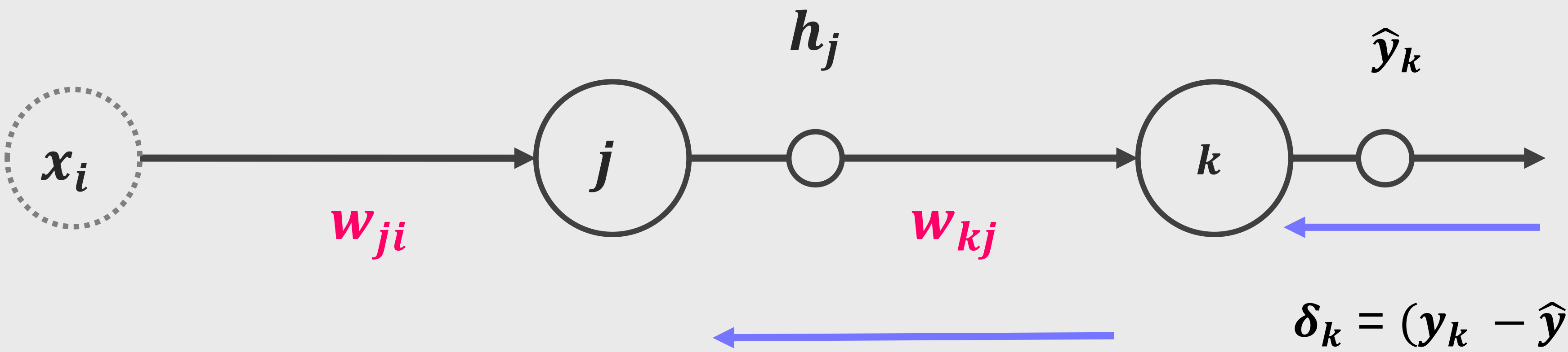
$e_k = y_k - \hat{y}_k$

$\delta_k = (y_k - \hat{y}_k)\,\hat{y}_k\,(1 - \hat{y}_k)$

Sec 8.1.1 to 8.1.3, *Deep Learning* by C Bishop

Backpropagation: **Backward pass**

Hidden layer delta (δ_j) considering sigmoidal hidden node(s)



$$e_k = y_k - \hat{y}_k$$

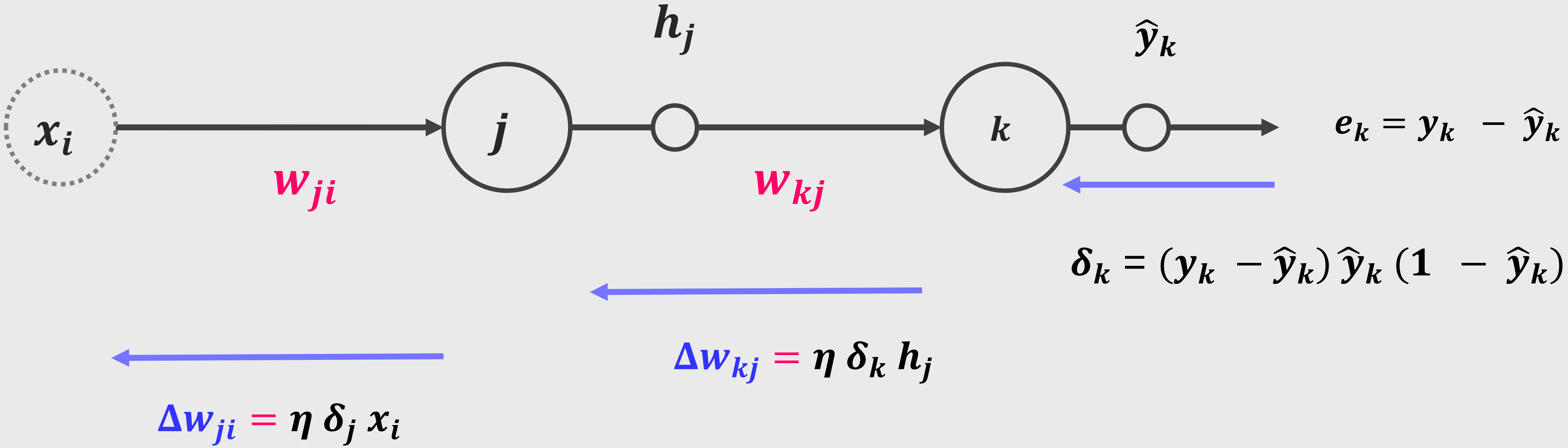
$$\delta_k = (y_k - \hat{y}_k)\,\hat{y}_k\,(1 - \hat{y}_k)$$

$$\delta_j = h_j\,(1 - h_j)\,\textstyle\sum_k \delta_k\, w_{kj}$$

Sec 8.1.1 to 8.1.3, Deep Learning by C Bishop

Backpropagation: **Backward pass**

Hidden layer delta (δ_j) considering sigmoidal hidden node(s)



$$e_k = y_k - \hat{y}_k$$

$$\delta_k = (y_k - \hat{y}_k)\,\hat{y}_k\,(1 - \hat{y}_k)$$

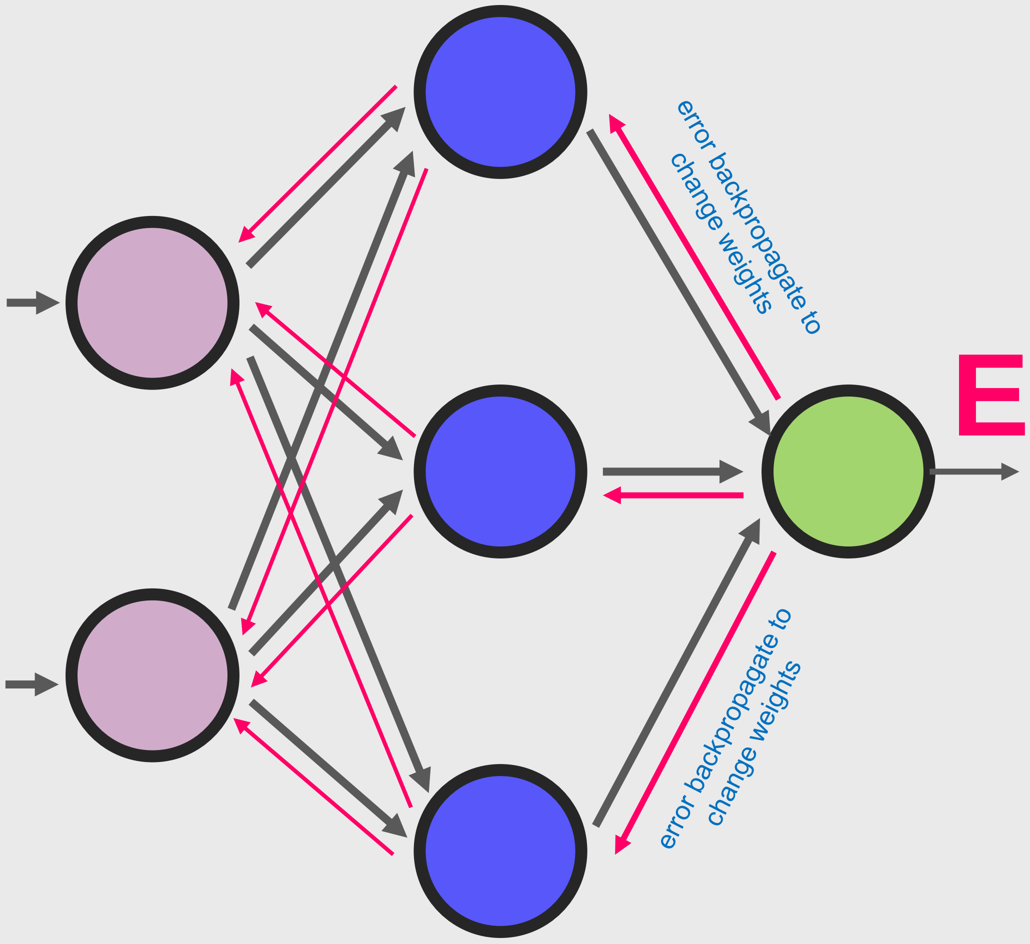
$$\Delta w_{kj} = \eta\,\delta_k\,h_j$$

$$\Delta w_{ji} = \eta\,\delta_j\,x_i$$

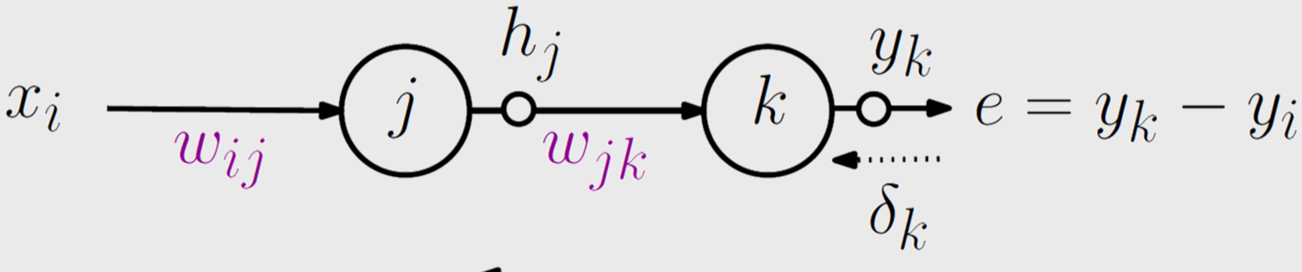
Sec 8.1.1 to 8.1.3, Deep Learning by C Bishop

BACKPROPAGATION

Algorithm for Updating Learning Systems



E = Output - Target



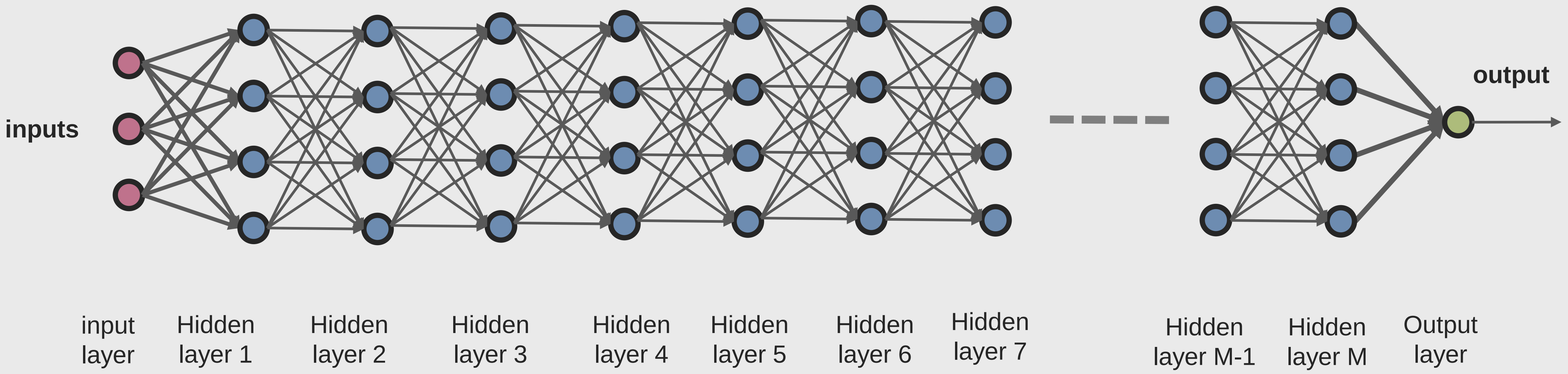
$$\Delta w_{jk} = \delta_j h_j$$

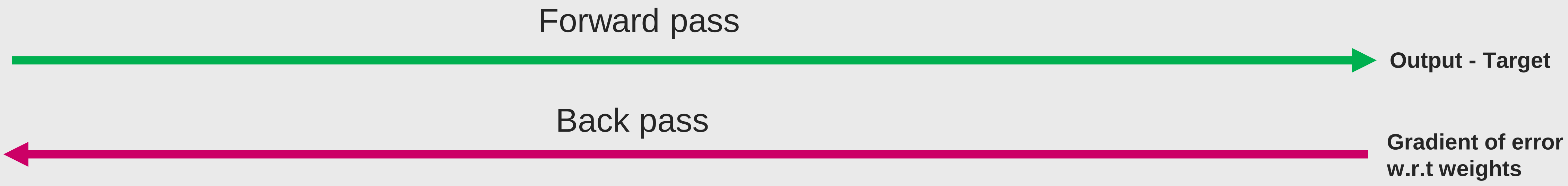
$$\Delta w_{ij} = \delta_i x_i$$

Sec 8.1.1 to 8.1.3, Deep Learning by C Bishop

BACKPROPAGATION

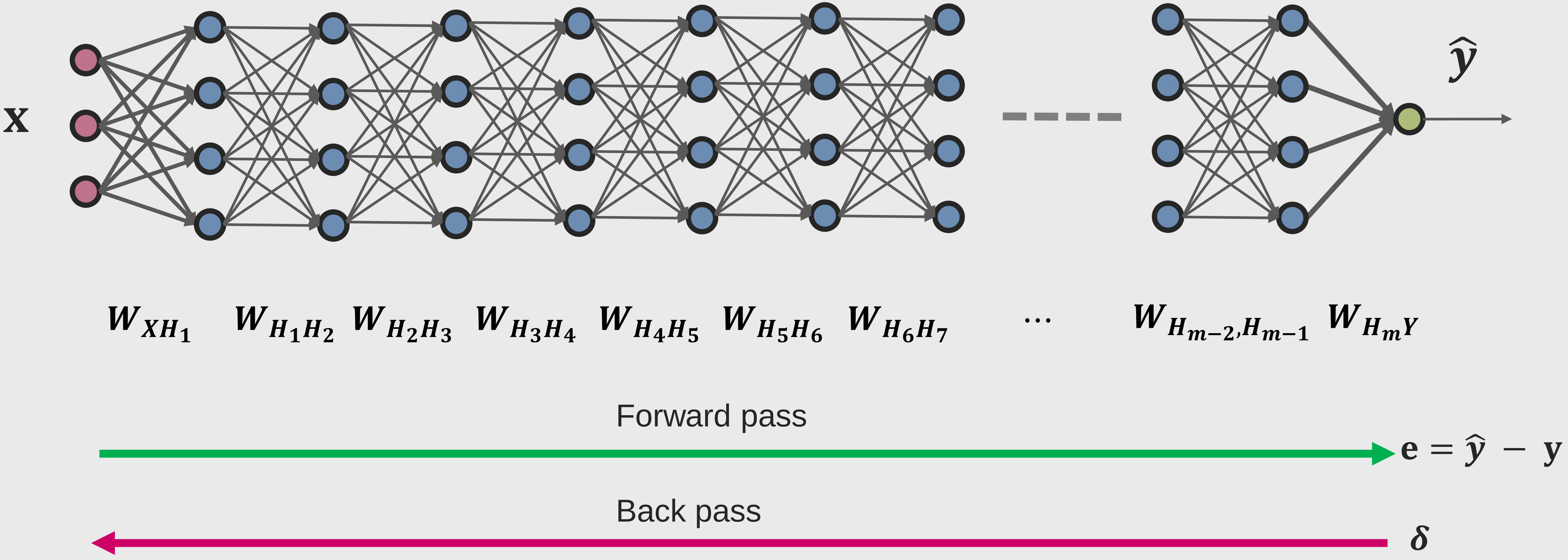
Deep Learning





BACKPROPAGATION

Deep Learning



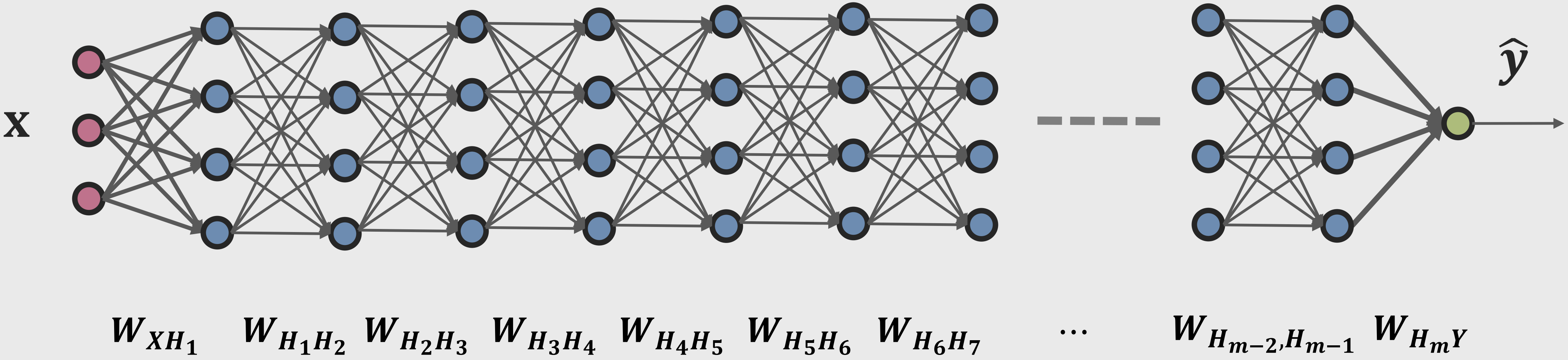
Forward pass

$\mathbf{e} = \hat{\boldsymbol{y}} - \mathbf{y}$

Back pass

$\boldsymbol{\delta}$

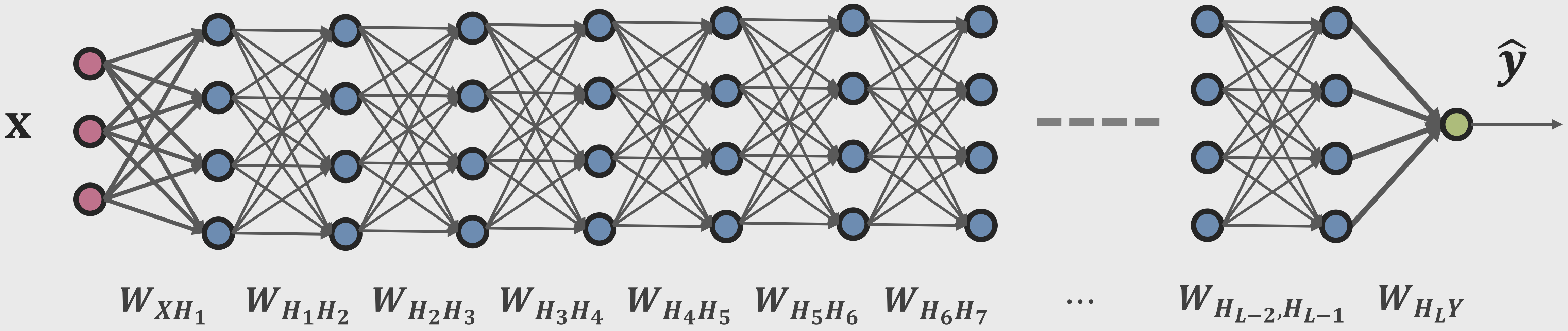
Vanishing Gradient



Gradient of error back propagation

Extremely small gradient ← Initial Gradient

Vanishing Gradient



Forward pass: $\hat{\mathbf{y}} = \varphi_L(W_L\varphi_{L-1}(W_{L-1}\cdots\varphi_3(W_3\varphi_2(W_2\varphi_1(W_1\mathbf{x})))\cdots))$

$\xrightarrow{\qquad\qquad} \mathbf{e} = \hat{\mathbf{y}} - \mathbf{y}$

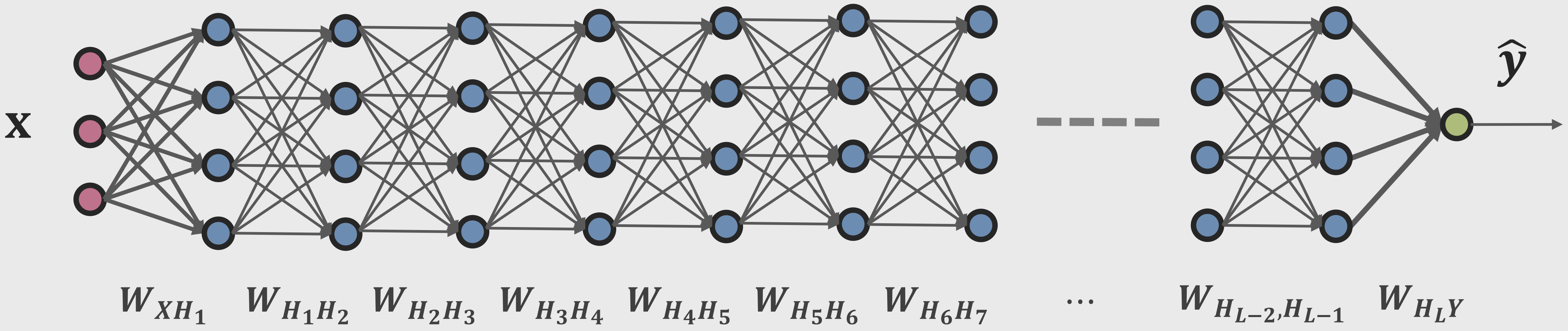
$$\hat{\mathbf{y}} = W_L \begin{bmatrix} 0.5 & \cdots & 0.0 \\ \vdots & \ddots & \vdots \\ 0.0 & \cdots & 0.5 \end{bmatrix}^{L-1}$$

$w = 0.5^{L-1}$ for a large L this will be **extremely small**. That is, weight w is a an exponentially **decreasing** function of L

This is caused by **sigmoid function** because its derivative lies between 0.0 and 0.25

9:37 AM

Vanishing Gradient



Forward pass: $\hat{y} = \varphi_L(W_L\varphi_{L-1}(W_{L-1}\cdots\varphi_3(W_3\varphi_2(W_2\varphi_1(W_1\mathbf{x})))\cdots))$

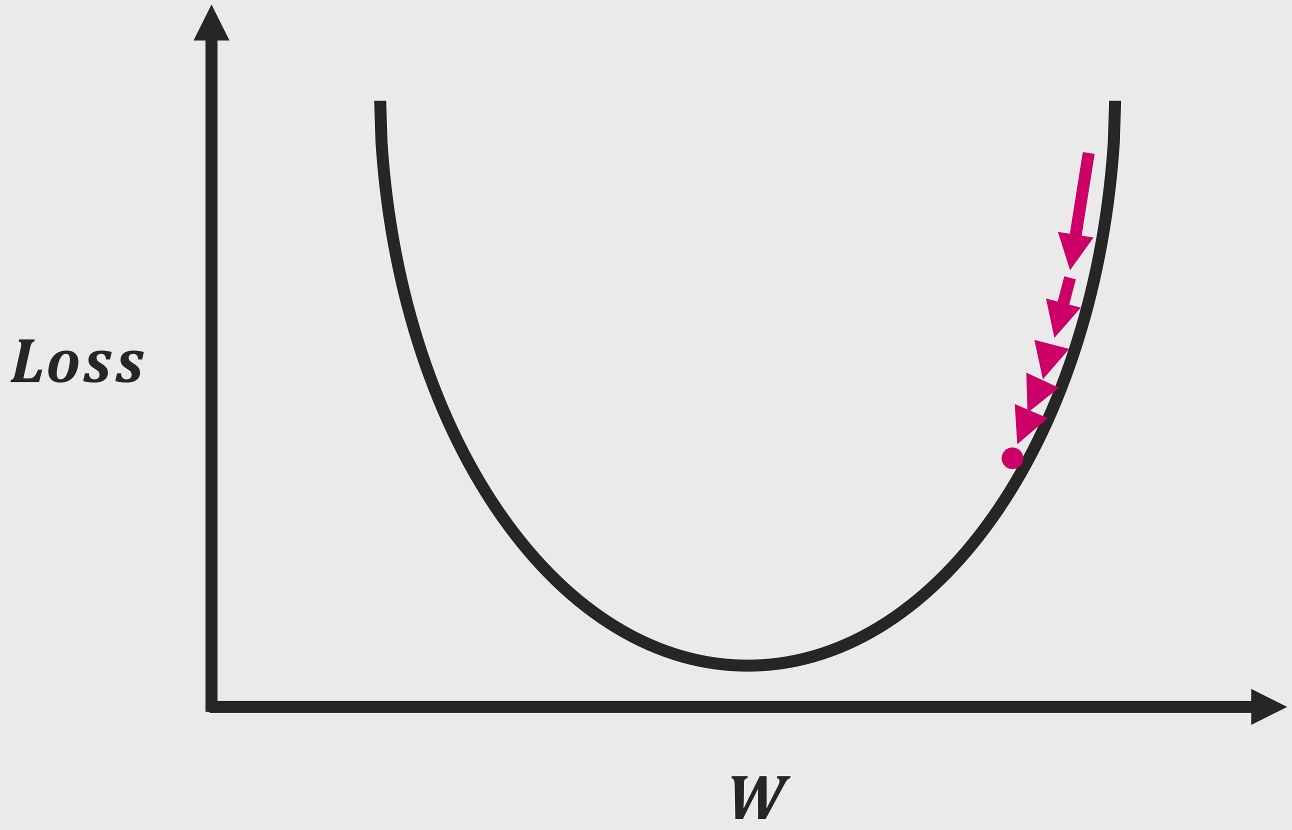
$\mathbf{e} = \hat{y} - \mathbf{y}$

$$\hat{y} = W_L\begin{bmatrix}0.5 & \cdots & 0.0\\ \vdots & \ddots & \vdots\\ 0.0 & \cdots & 0.5\end{bmatrix}^{L-1}$$

$w = 0.5^{L-1}$ for a large L this will be **extremely small**. That is, weight w is a an exponentially **decreasing** function of L

Solution:
Use of **ReLU** function
$\varphi_1(x) = max(0, x)$

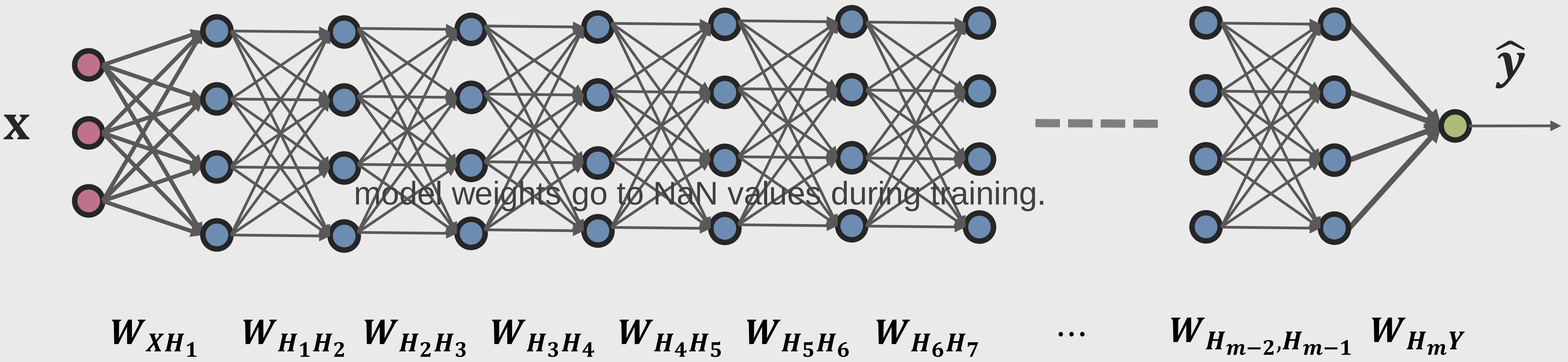
Vanishing Gradient



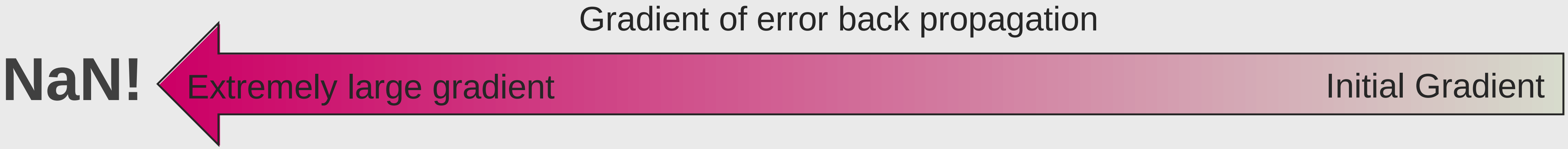
gradient become very small

Convergence virtually stops because weights do not change any more

Exploding Gradient



model weights go to NaN values during training.



Gradient of error back propagation

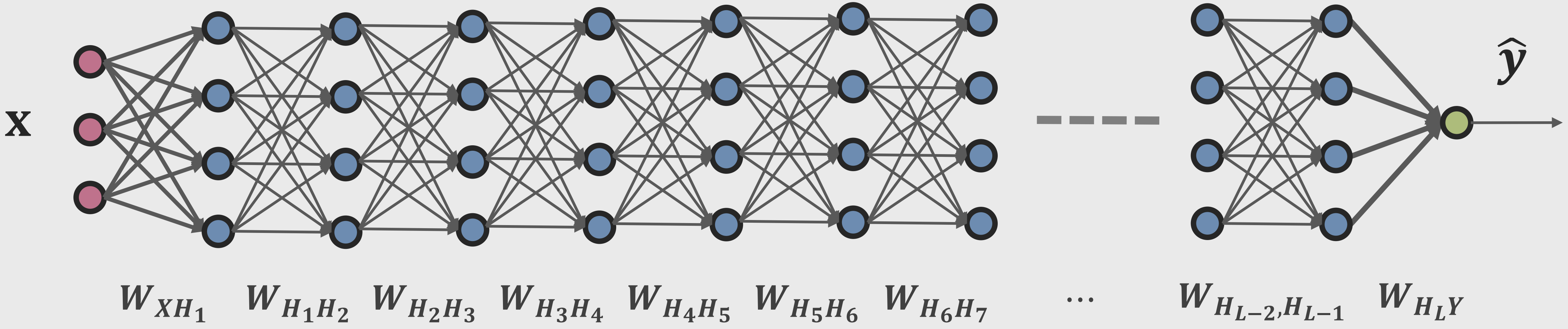
NaN!

Extremely large gradient

Initial Gradient

9:37 AM

Exploding Gradient



Forward pass: $\hat{y} = \varphi_L(W_L\varphi_{L-1}(W_{L-1}\cdots\varphi_3(W_3\varphi_2(W_2\varphi_1(W_1\mathbf{x})))\cdots))$

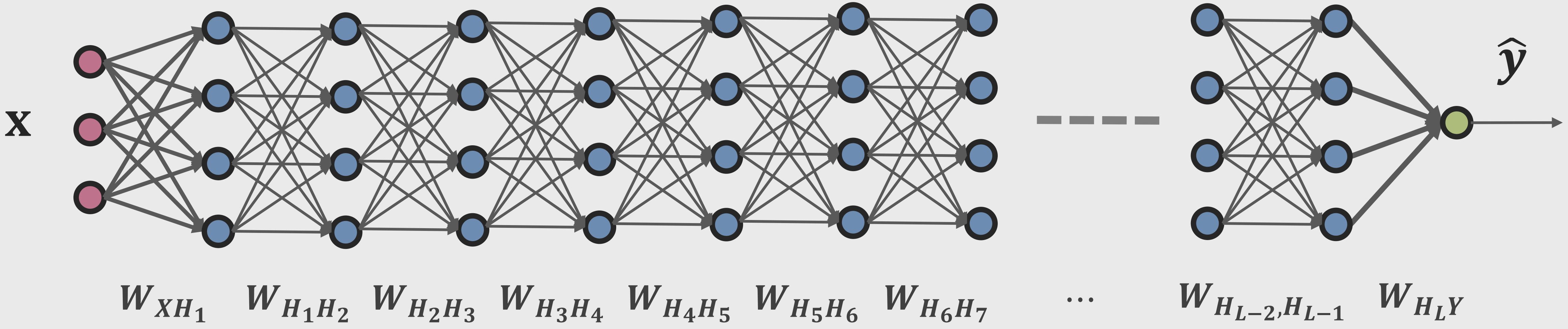
$\mathbf{e} = \hat{y} - \mathbf{y}$

$$\hat{y} = W_L \begin{bmatrix}1.5 & \cdots & 0.0\\ \vdots & \ddots & \vdots \\ 0.0 & \cdots & 1.5\end{bmatrix}^{L-1}$$

$w = 1.5^{L-1}$ for a large L this will be **extremely larger**. That is, weight w is a an exponentially **increase** function of L

This is caused by **initialization of weights with large values.**

Exploding Gradient



Forward pass: $\hat{y} = \varphi_L(W_L\varphi_{L-1}(W_{L-1}\cdots\varphi_3(W_3\varphi_2(W_2\varphi_1(W_1\mathbf{x})))\cdots))$

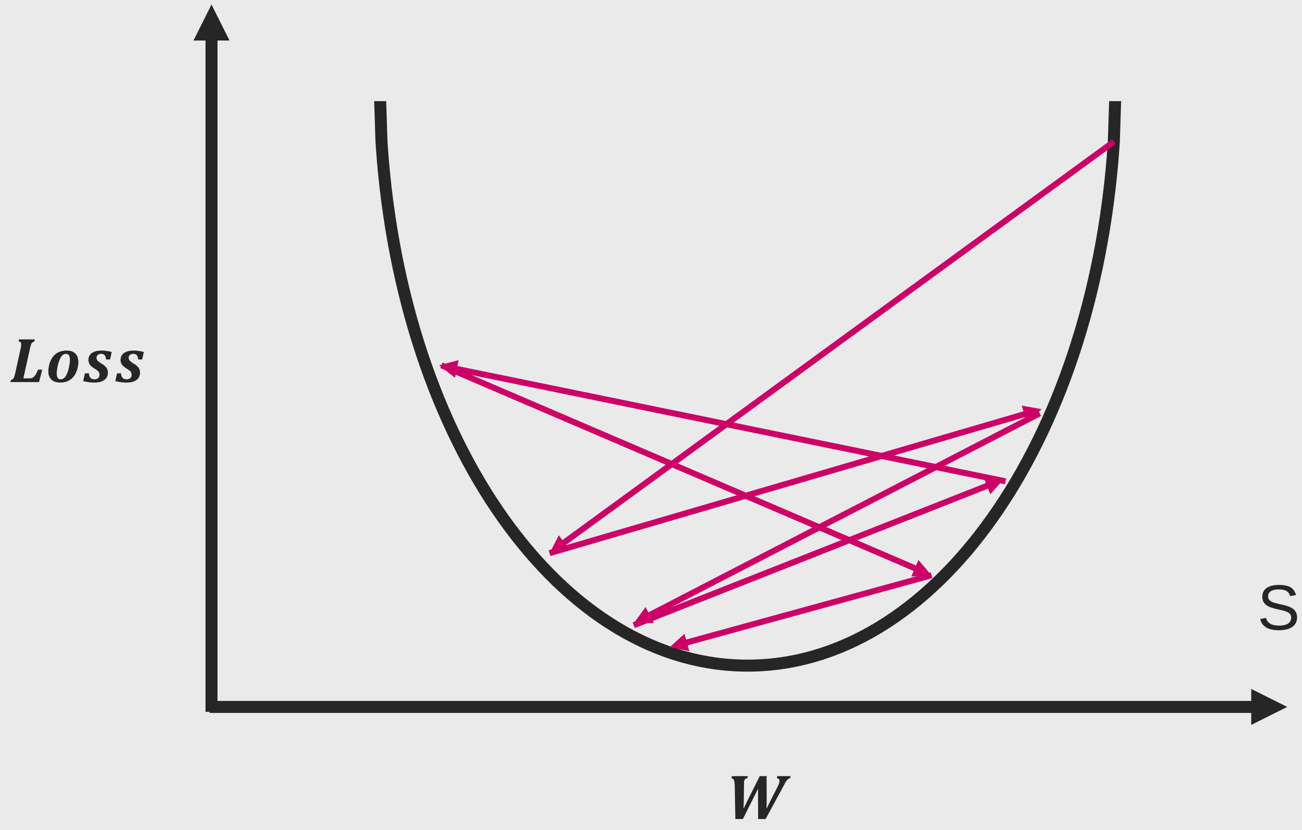
$\longrightarrow \mathbf{e} = \hat{y} - \mathbf{y}$

$$\hat{y} = W_L \begin{bmatrix} 1.5 & \cdots & 0.0 \\ \vdots & \ddots & \vdots \\ 0.0 & \cdots & 1.5 \end{bmatrix}^{L-1}$$

$w = 1.5^{L-1}$ for a large L this will be extremely larger. That is, weight w is a an exponentially **increase** function of L

Solution:
Gradient clipping and/or better weight initialization

Exploding Gradient



Highly fluctuating gradient descent

Weight abruptly changes.

Skips Global minima

Part 4

Practical Session

See attached document in Canvas

python™

TensorFlow

PyTorch

Google Colaboratory