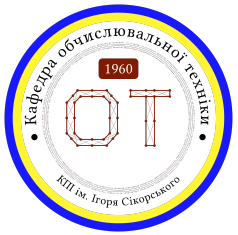


Системи штучного інтелекту

Лекція 1-2: Вступ до штучного інтелекту

Кочура Юрій Петрович
iuriy.kochura@gmail.com
@y_kochura

Сьогодні

- Інтелект vs штучний інтелект
- Визначення штучного інтелекту та парадигма
- Типи машинного навчання
- Концепція глибинного навчання
- Приклади застосування глибинного навчання
- Перцептрон: пряме та зворотне поширення
- Загальні функції активації

Штучний інтелект

Чи може машина думати?

I.—COMPUTING MACHINERY AND INTELLIGENCE

BY A. M. TURING

1. *The Imitation Game.*

I PROPOSE to consider the question, 'Can machines think?'

Image source: biography

— Alan Turing, 1950

—

Credits: Alan Turing, 1950.

In the process of trying to imitate an adult human mind we are bound to think a good deal about the process which has brought it to the state that it is in. We may notice three components,

a. The initial state of the mind, say at birth,

b. The education to which it has been subjected,

c. Other experience, not to be described as education, to which it has been subjected.

Instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education one would obtain the adult brain. Presumably the child-brain is something like a note-book as one buys it from the stationers. Rather little mechanism, and lots of blank sheets. (Mechanism and writing are from our point of view almost synonymous.) Our hope is that there is so little mechanism in the child-brain that something like it can be easily programmed.

— Alan Turing, 1950

—

Credits: Alan Turing, 1950.

Що таке інтелект?

- Інтелект — це про здатність

  навчатися приймати рішення для досягнення цілей

- Навчання, прийняття рішення, та цілі є ключовими

Що таке штучний інтелект?

- У широкому сенсі

Будь-яка техніка, яка дозволяє комп'ютерам імітувати поведінку людини

Що таке штучний інтелект?

- У вузькому сенсі

> Штучний інтелект — здатність інженерної системи обробляти, застосовувати та вдосконалювати здобуті знання та вміння.

- Знання — це факти, інформація та навички, набуті через досвід або навчання.

—
Credits: ISO/IEC TR 24028:2020(en), 2020.

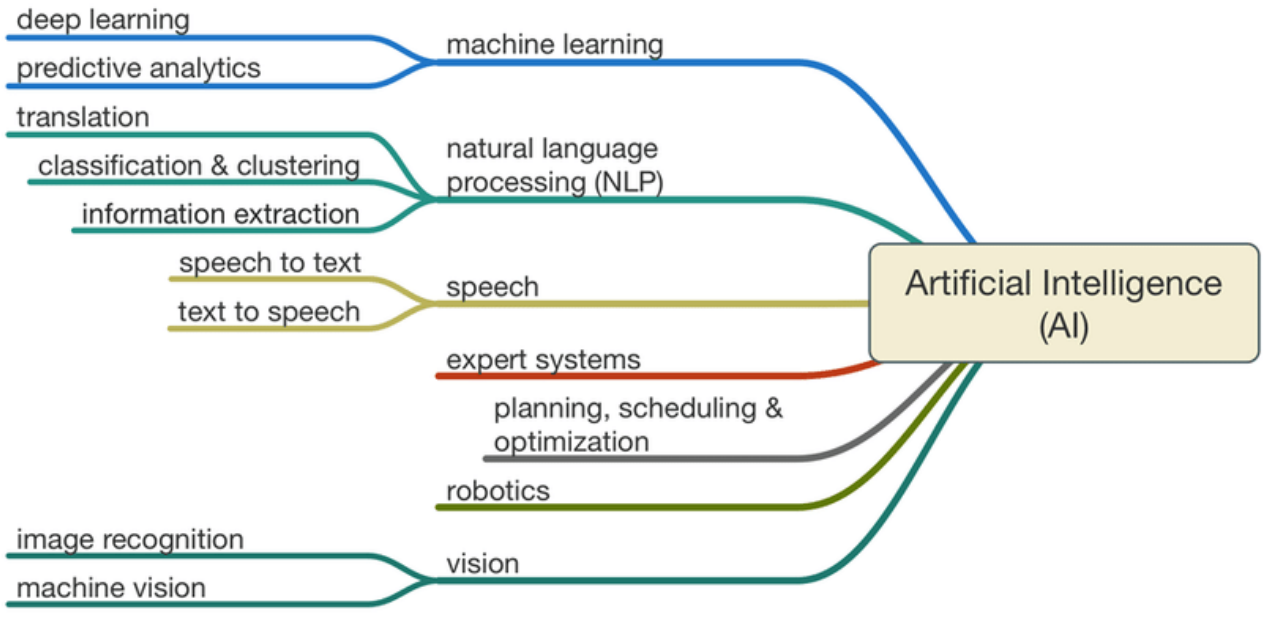
Коротка історія

- 1940—1952: Early days
  - 1943: McCulloch & Pitts: Boolean circuit model of brain
  - 1950: Turing's ''Computing Machinery and Intelligence''
- 1952–1956: The birth of AI
  - 1950s: Early AI programs, including Samuel's checkers program, Newell & Simon's Logic Theorist, Gelernter's Geometry Engine
  - 1956: Dartmouth meeting: ''Artificial Intelligence'' adopted
- 1956–1974: The golden years
  - 1958: Frank Rosenblatt invented perceptron (simple neural network)
  - 1964: Bobrow's program that solves algebra word problems
  - 1965: Robinson's complete algorithm for logical reasoning
- 1974–1980: The first AI winter
- 1980–1987: Expert systems industry boom
- 1987—1993: Expert systems industry busts: the second AI winter
- 1993–2011: Statistical approaches
  - Resurgence of probability, focus on uncertainty
  - General increase in technical depth
  - Intelligent agents
- 2011–present: Deep Learning, Big Data and AI
  - Big data, big compute, neural networks
  - AI used in many industries

—

Credits: Wikipedia - History of artificial intelligence

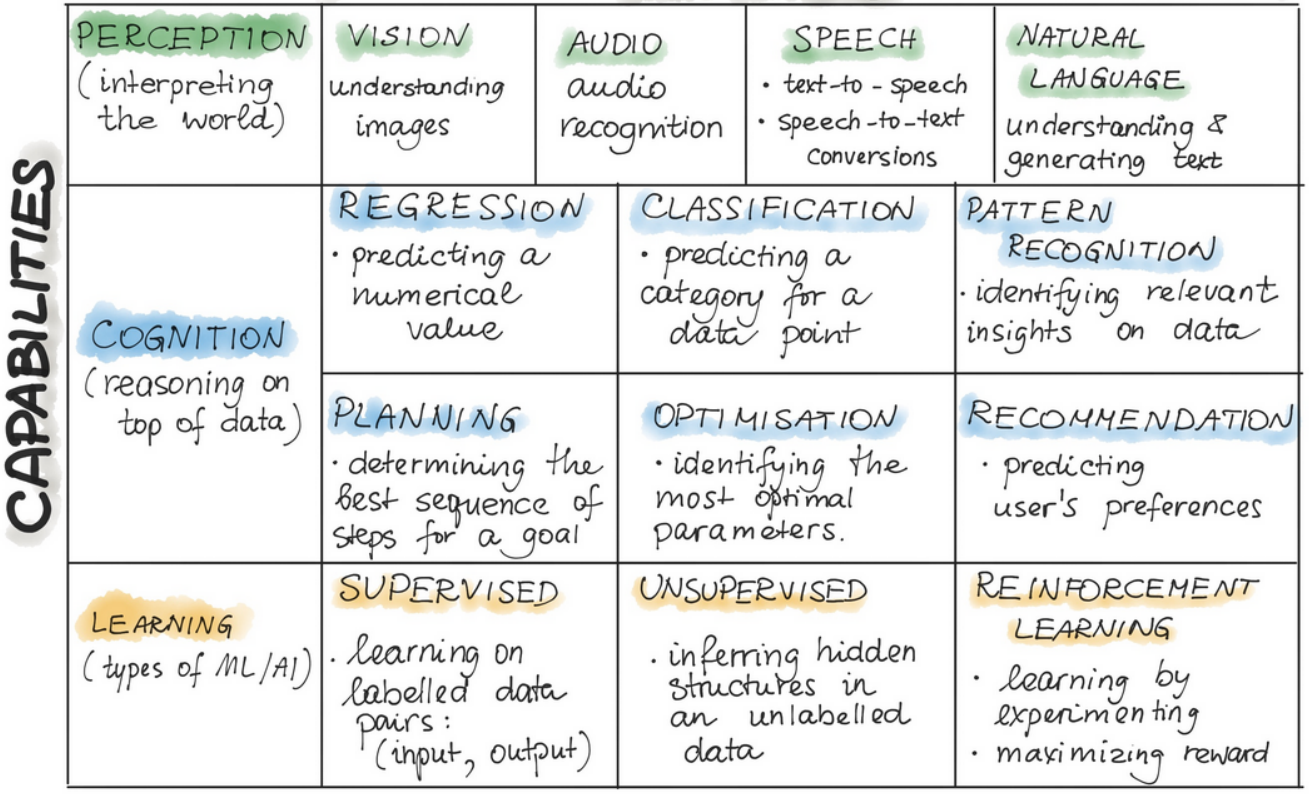
AI — багата галузь



—

Image Source: Marizel B. and Ma. Louella Salenga, 2018.

SUMMARY OF ML/AI CAPABILITIES



—

Image Source: Why you Might Want to use Machine Learning.

"Just as electricity transformed almost everything 100 years ago, today I actually have a hard time thinking of an industry that I don't think AI will transform in the next several years."

— Andrew Ng

—

Credits: Andrew Ng: Artificial Intelligence is the New Electricity, 2017.

Машинне навчання

Що таке машинне навчання?

A. L. Samuel*

Some Studies in Machine Learning
Using the Game of Checkers. II—Recent Progress

Field of study that gives computers the ability to learn without being explicitly programmed.

— Arthur Samuel, 1959

Image source: wikipedia

—
Credits: Arthur Samuel, 1959.

Що таке машинне навчання?

Machine Learning is the study of computer algorithms that improve automatically through experience.

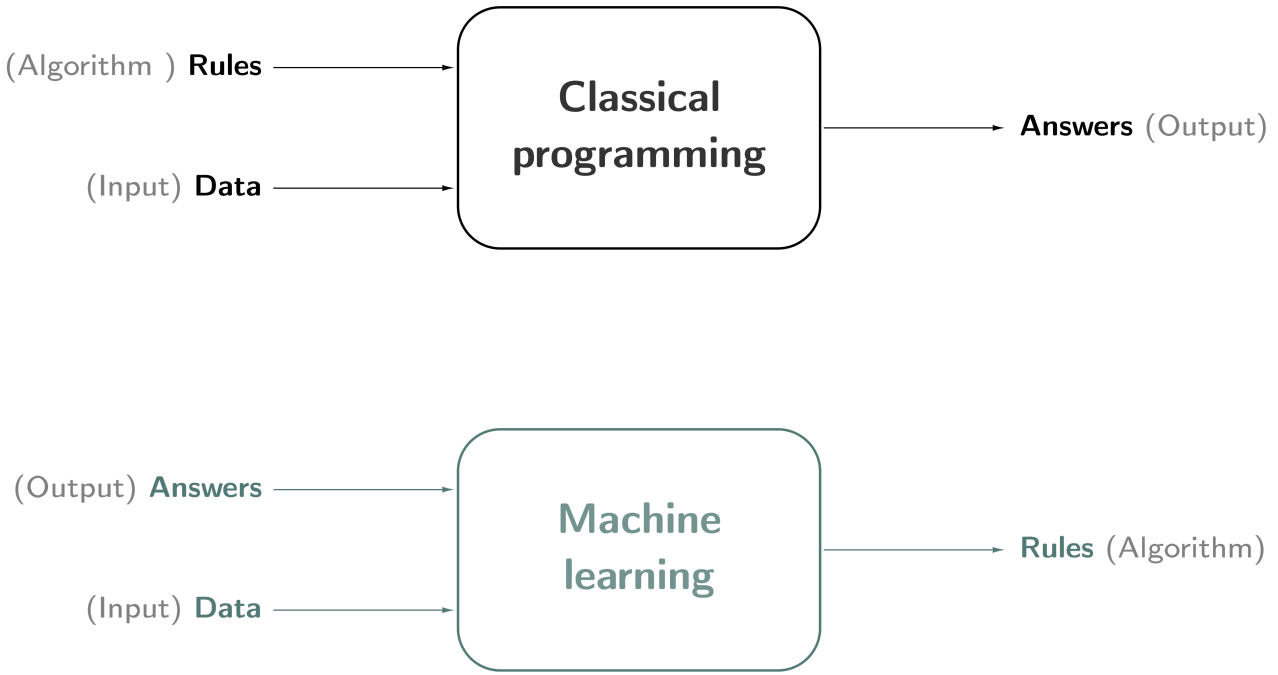
— Tom Mitchell, 1997

Image source: Tom Mitchell's Home Page

—

Credits: Tom Mitchell, 1997.

Машинне навчання: нова парадигма програмування



Типи машинного навчання

Контрольоване навчання **Supervised learning**	**Напівконтрольоване навчання** **Semi-supervised learning**	**Неконтрольоване навчання** **Unsupervised learning**	**Навчання з підкріплення** **Reinforcement learning**
Дані: (x, y) x – приклад, y – мітка	**Дані:** (x, y) та $x, \lvert(x, y)\rvert < \lvert x\rvert$ x – приклад, y – мітка	**Дані:** x x – приклад, немає міток!	**Дані:** пари **стан-дія**
Мета – знайти функцію відображення $x \longrightarrow y$	**Мета** – знайти функцію відображення або категорію $x \longrightarrow y$	**Мета** – знайти правильну категорію.	**Мета** – максимізація загальної винагороди, отриманої агентом при взаємодії з навколишнім середовищем.
Приклад  Це є яблуко.	**Приклад**  Це є яблуко.	**Приклад**  Цей об'єкт схожий на інший.	**Приклад**  Їжте це, бо це зробить вас сильнішим.

What is Deep Learning?

Artificial Intelligence

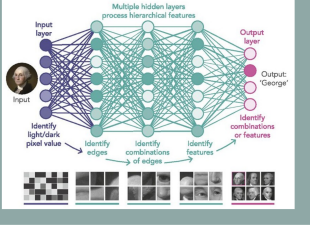
Any technique that enable computers to mimic human behavior

Machine Learning

Ability to learn without explicitly being programmed

Deep Learning

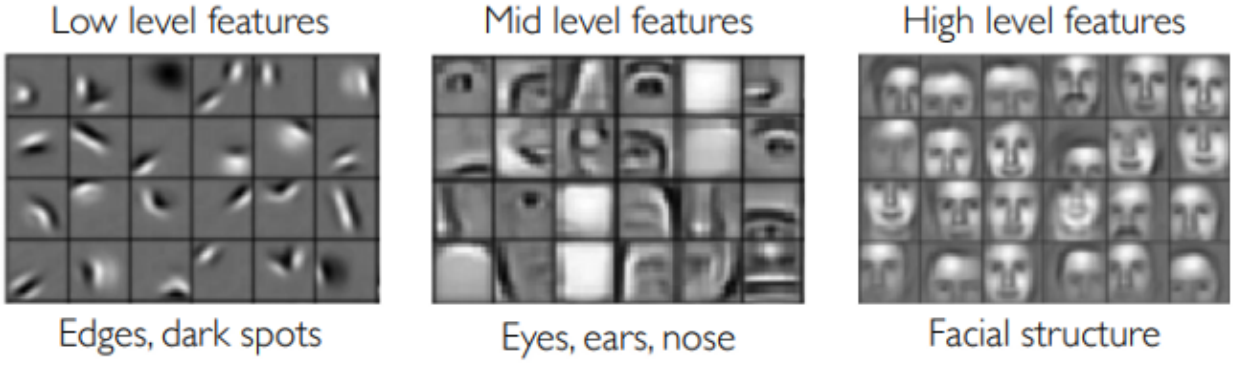
Extract patterns from data using neural networks



Чому глибинне навчання (DL)?

Ознаки, розроблені вручну, займають багато часу, є крихкими та не підлягають масштабуванню на практиці

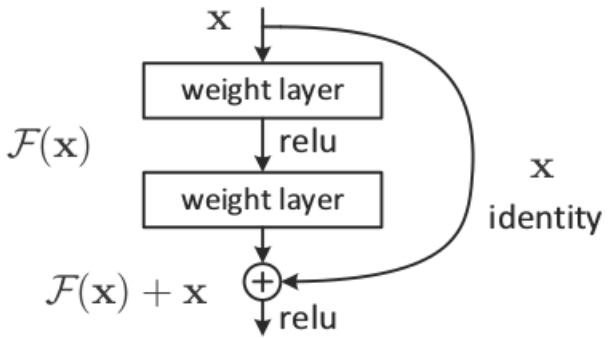
Чи можемо ми вивчити основні ознаки безпосередньо з даних?



Slide source: MIT 6.S191

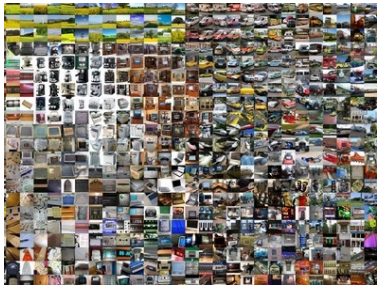
Чому **DL** працює зараз?

Algorithms (old and new)



More data

- Larger Datasets
- Easier Collection & Storage

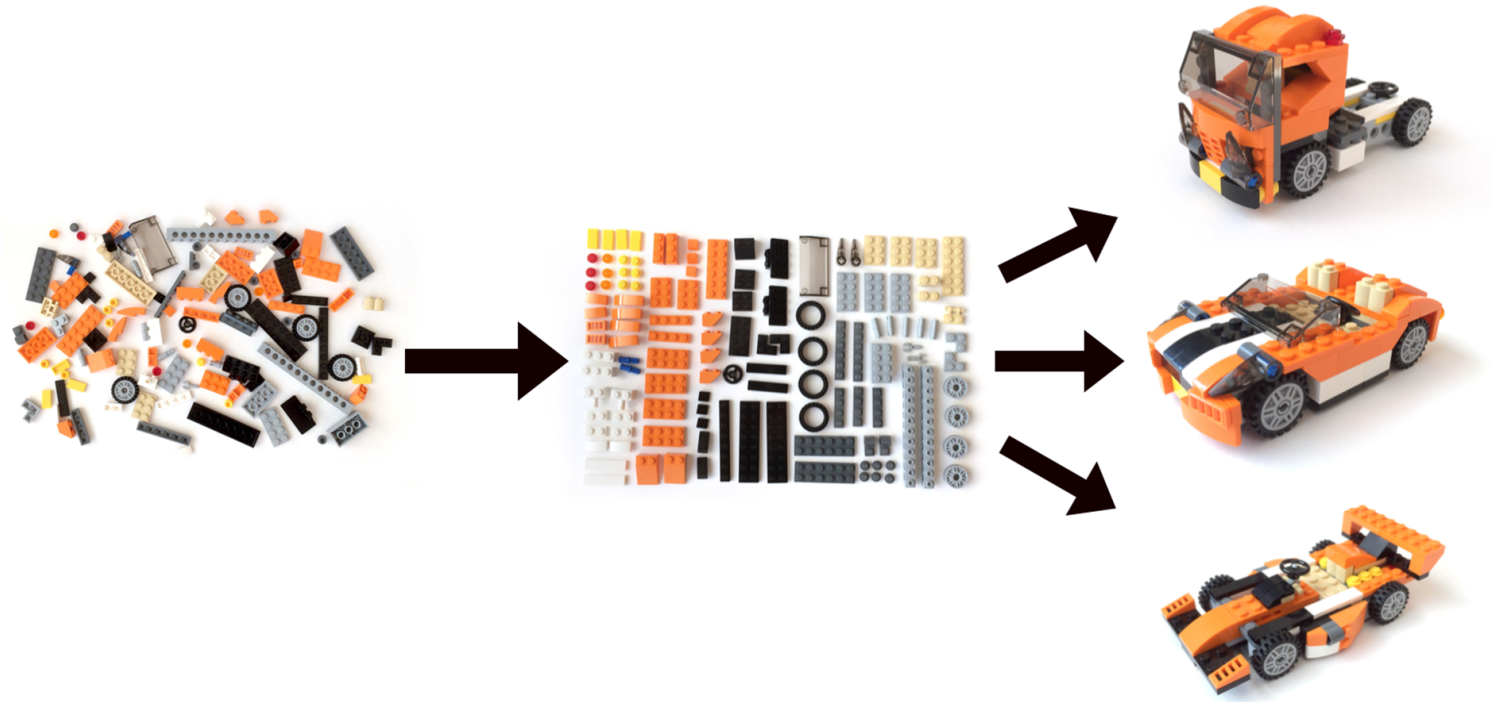


Software (New models and improved techniques)

Caffe2 · Microsoft CNTK · TensorFlow · theano · Chainer · PYTORCH · dmlc mxnet · K

Faster compute engines

DL як архітектурна мова



—

Image source: http://chelseamarzean.com/post-the-atomic-workflow/, 2016.

Image source: Christopher Bishop

For the last forty years we have programmed computers; for the next forty years we will train them.

— Chris Bishop, 2020.

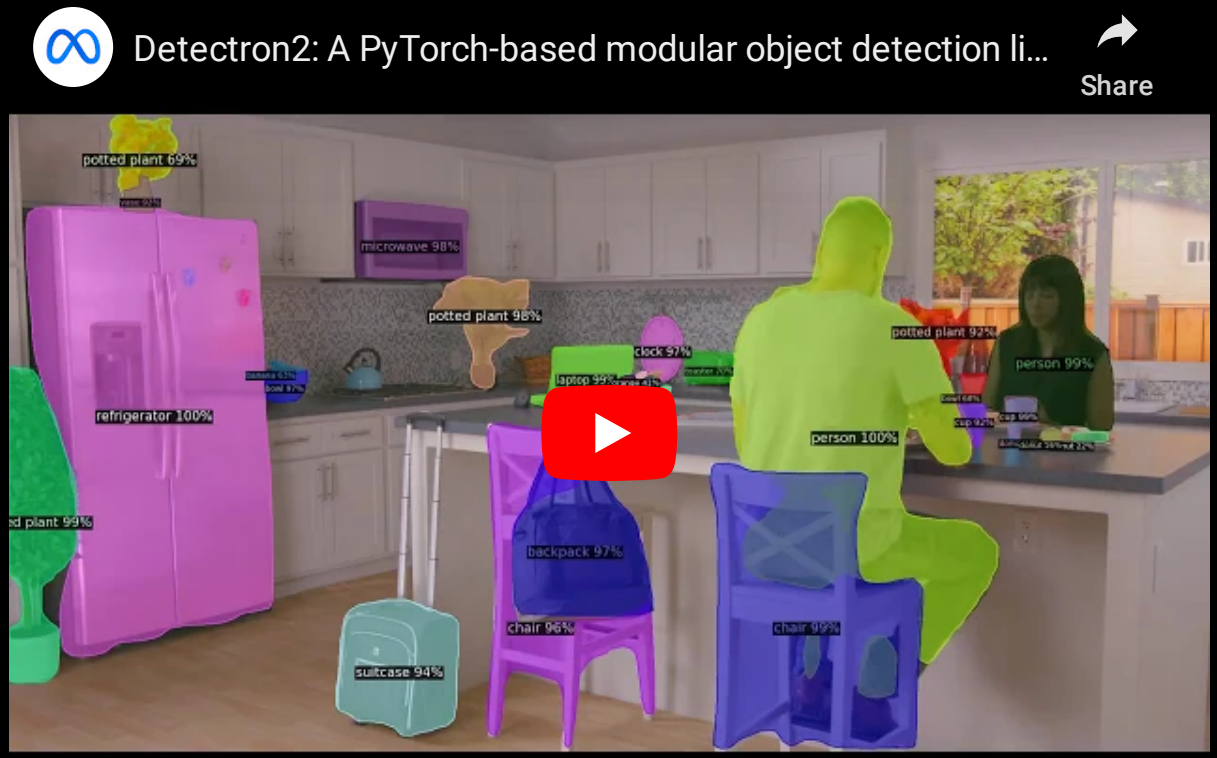
—

Credits: Chris Bishop on "The Real AI Revolution" NeurIPS 2020

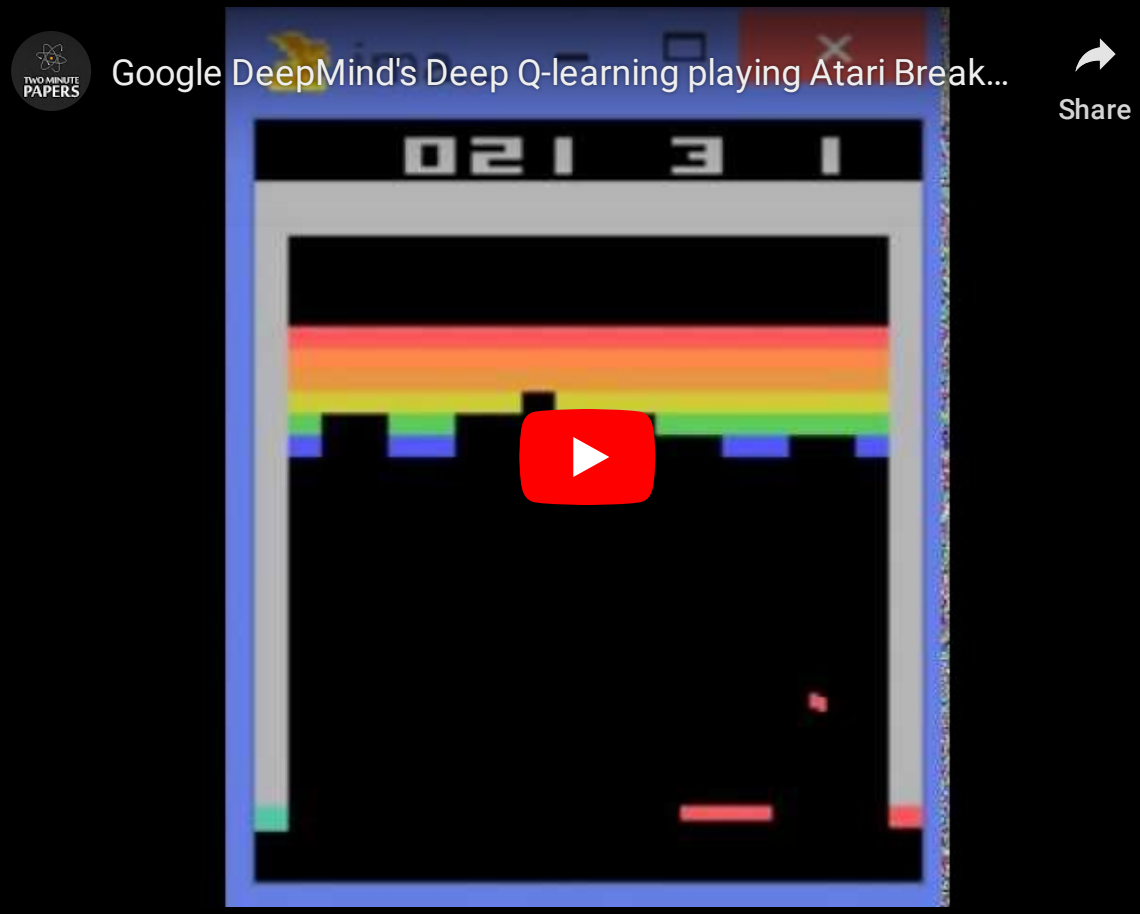
*"ACM named **Yann LeCun**, **Geoffrey Hinton**, and **Yoshua Bengio** recipients of the **2018 ACM A.M. Turing Award** for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing."*

—
Credits: Wikipedia

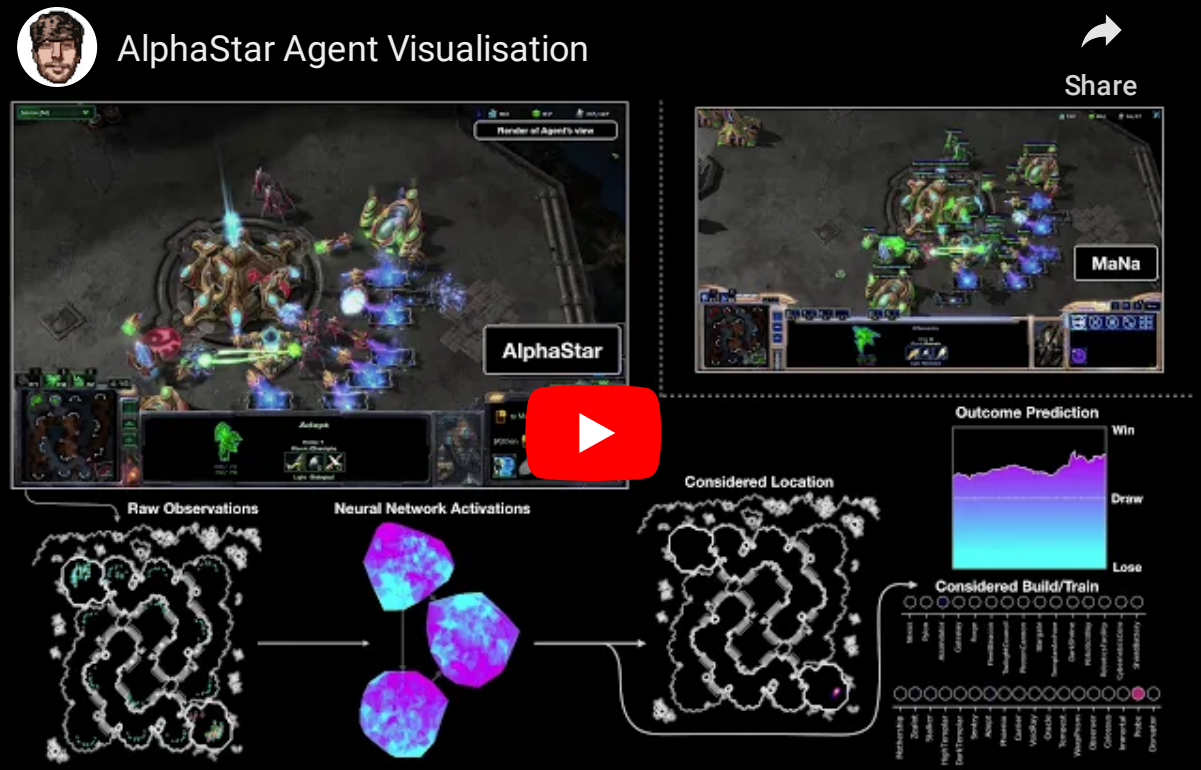
Застосування та успіхи



Object detection, pose estimation, segmentation (2019)



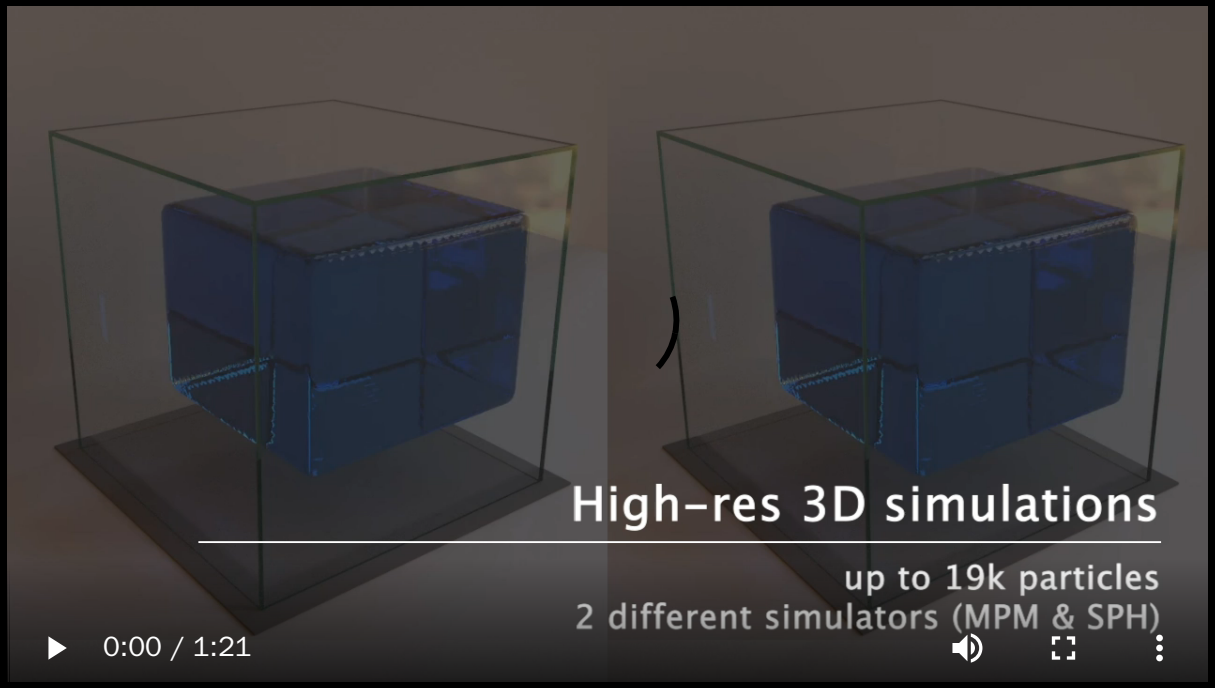
Reinforcement learning (Mnih et al, 2014)



Strategy games (Deepmind, 2016-2018)

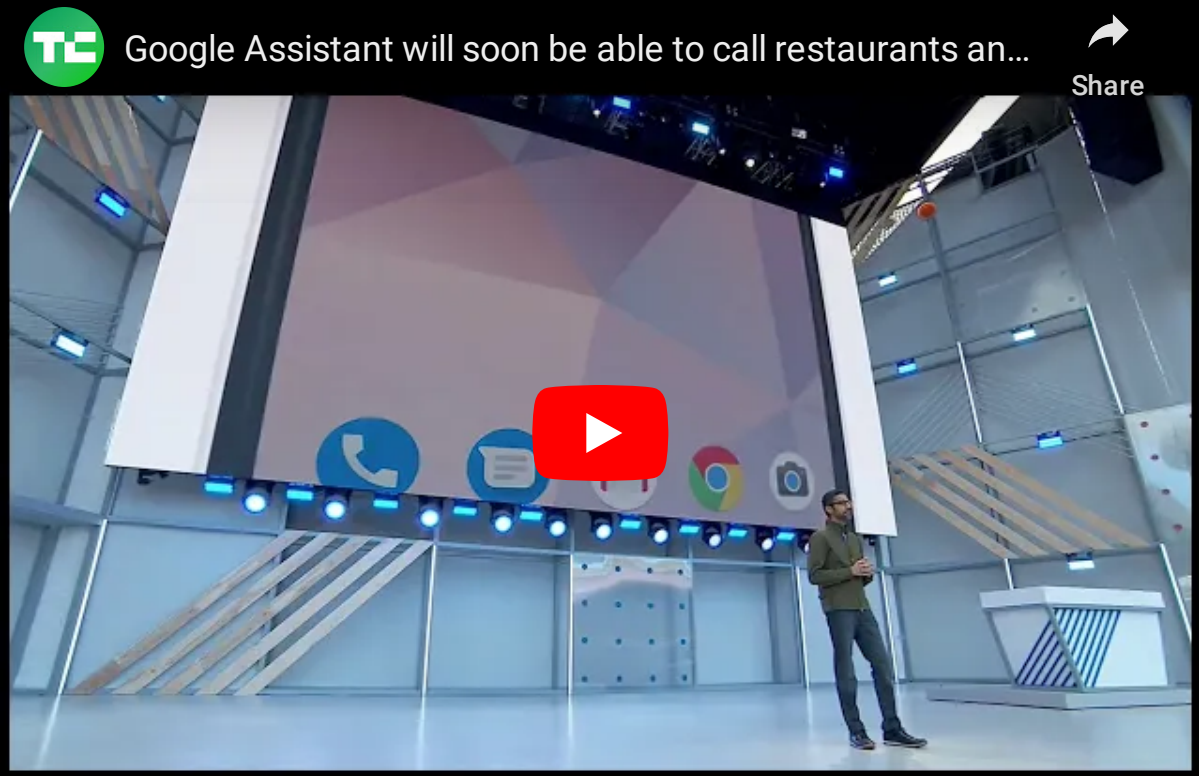
Autonomous cars (NVIDIA, 2016)

Autopilot (Tesla, 2019)

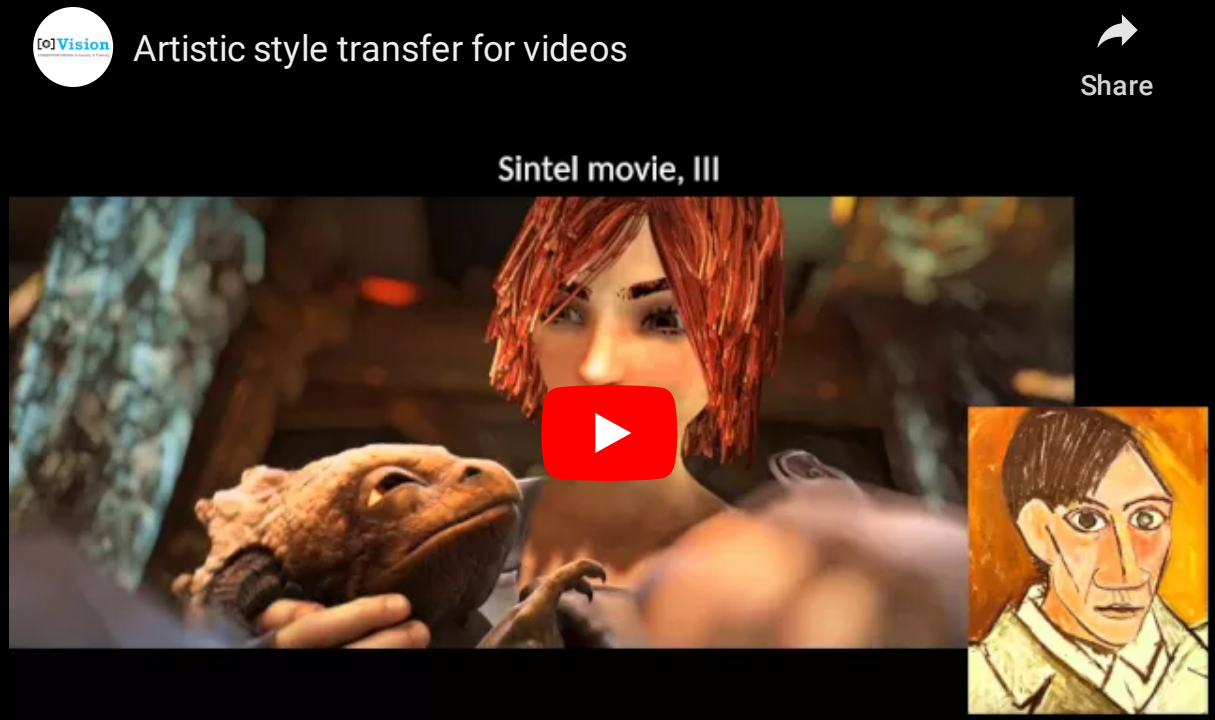


Physics simulation (Sanchez-Gonzalez et al, 2020)

AI for Science (Deepmind, AlphaFold, 2020)



Speech synthesis and question answering (Google, 2018)



Artistic style transfer (Ruder et al, 2016)

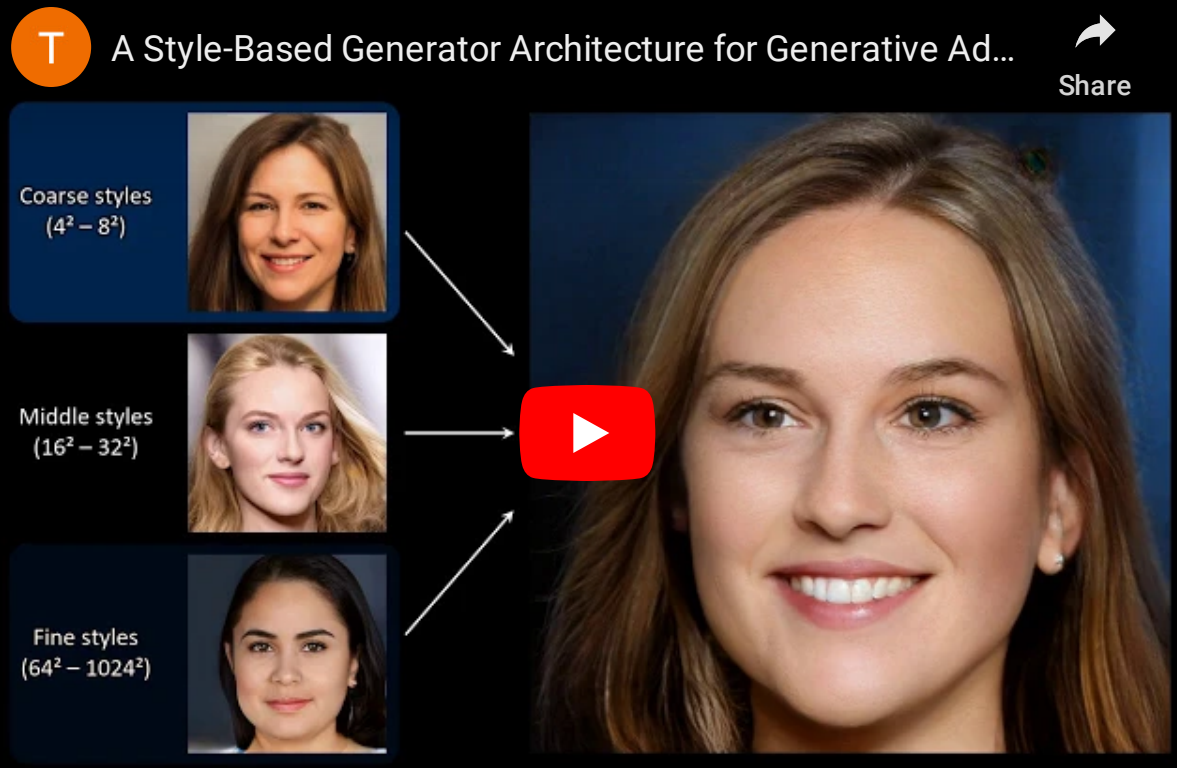
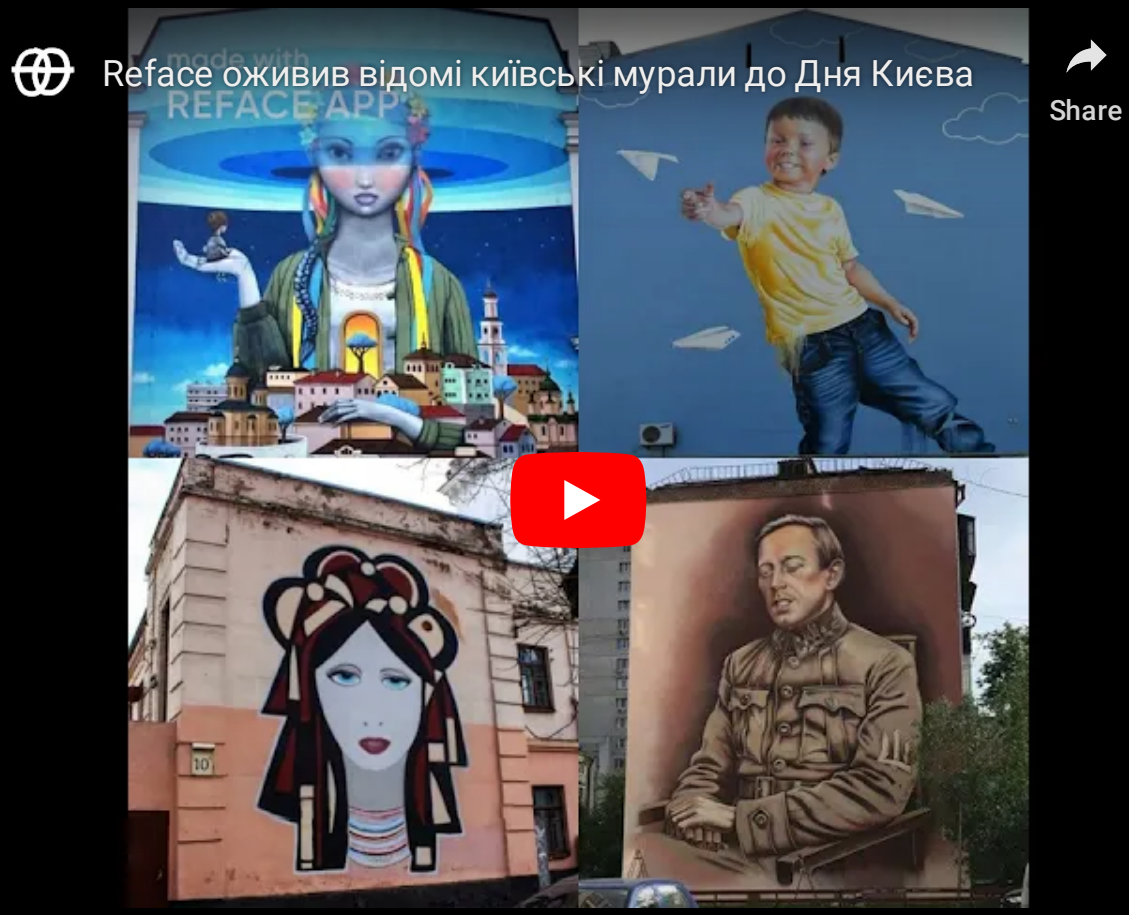


Image generation (Karras et al, 2018)

Music composition (NVIDIA, 2017)

Dali Lives (2019)



Reface revived the famous Kyiv murals for Kyiv Day (2021)

Перцептрон

Структурний будівельний блок глибокого навчання

Перцептрон

Модель перцептрона (Rosenblatt, 1958)

$$g(z) = \begin{cases} 1 & \text{if } z = \sum_i w_i x_i + b \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Ця модель спочатку була мотивована біологією, де w_i — синаптичні ваги для вхідних сигналів x_i та g — активація нейрона.

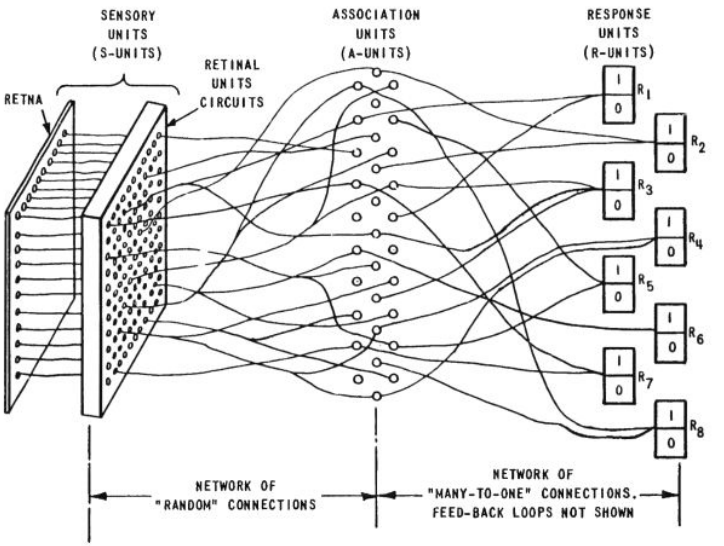
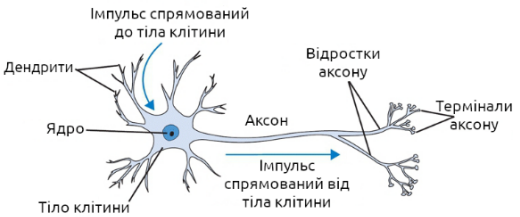
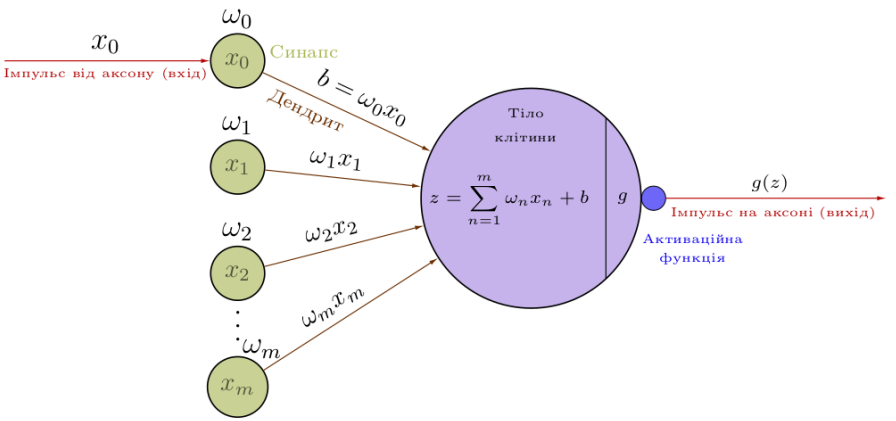


Figure 1 ORGANIZATION OF THE MARK I PERCEPTRON

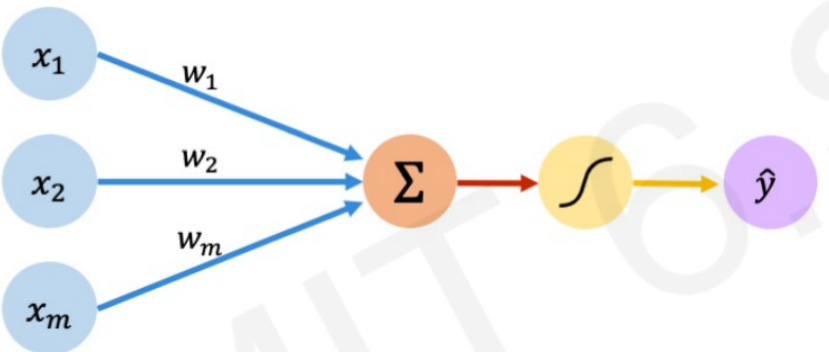
—

Image source: Frank Rosenblatt, Mark I Perceptron operators' manual, 1960.

Біологічний **vs** штучний нейрон

Біологічний нейрон	Штучний нейрон
	

The Perceptron: Forward Propagation



Inputs Weights Sum Non-Linearity Output

Output

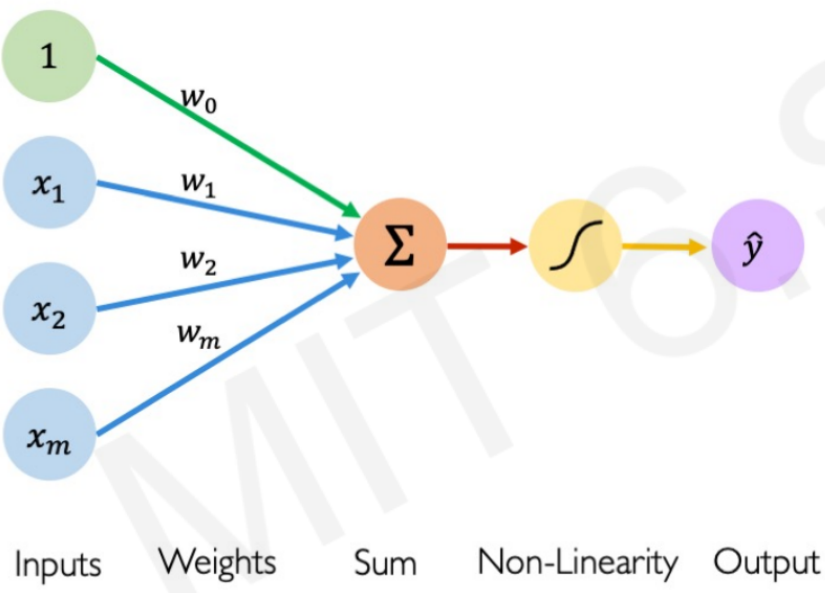
Linear combination of inputs

$$\hat{y} = g\left(\sum_{i=1}^{m} x_i \, w_i\right)$$

Non-linear activation function

Slide source: MIT 6.S191

The Perceptron: Forward Propagation



Output

Linear combination of inputs

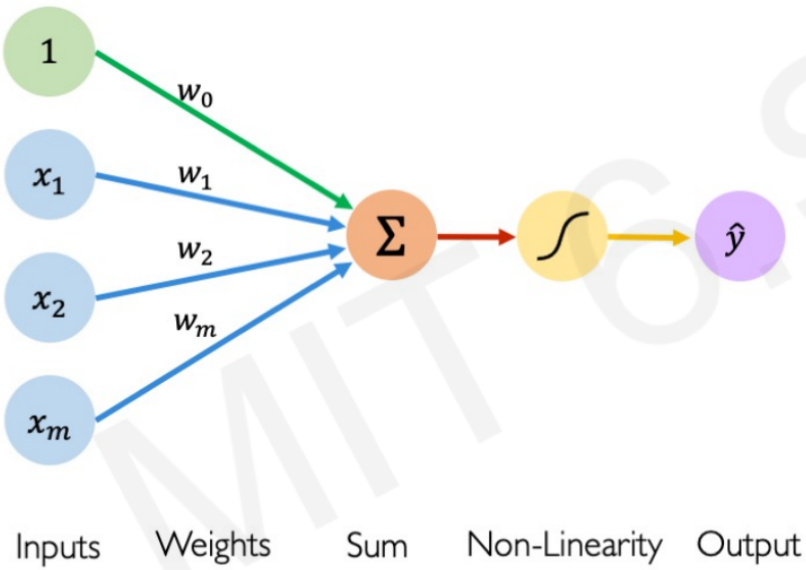
$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i \, w_i\right)$$

Non-linear activation function

Bias

Slide source: MIT 6.S191

The Perceptron: Forward Propagation



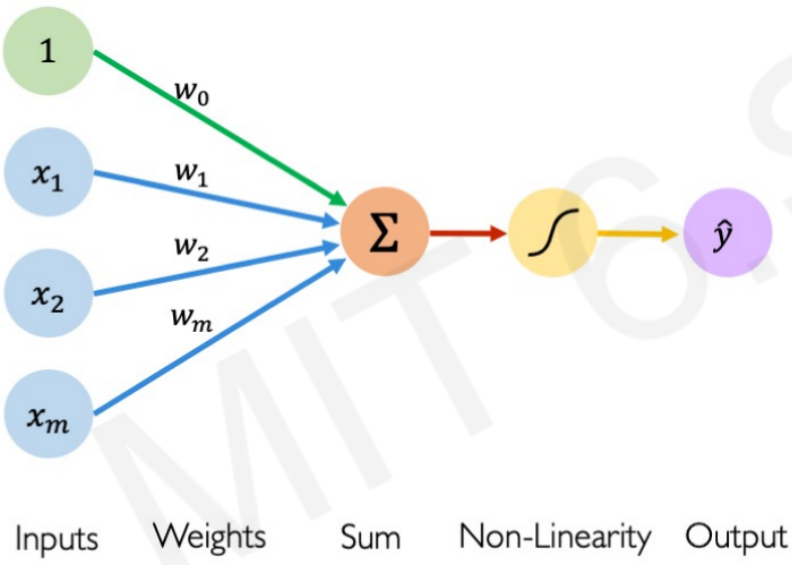
$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i\, w_i\right)$$

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W}\right)$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

Slide source: MIT 6.S191

The Perceptron: Forward Propagation

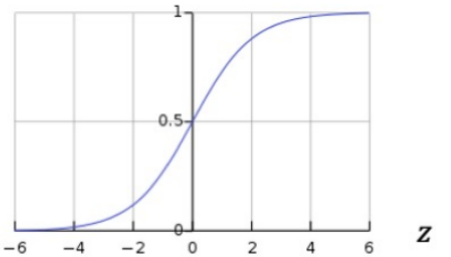


Activation Functions

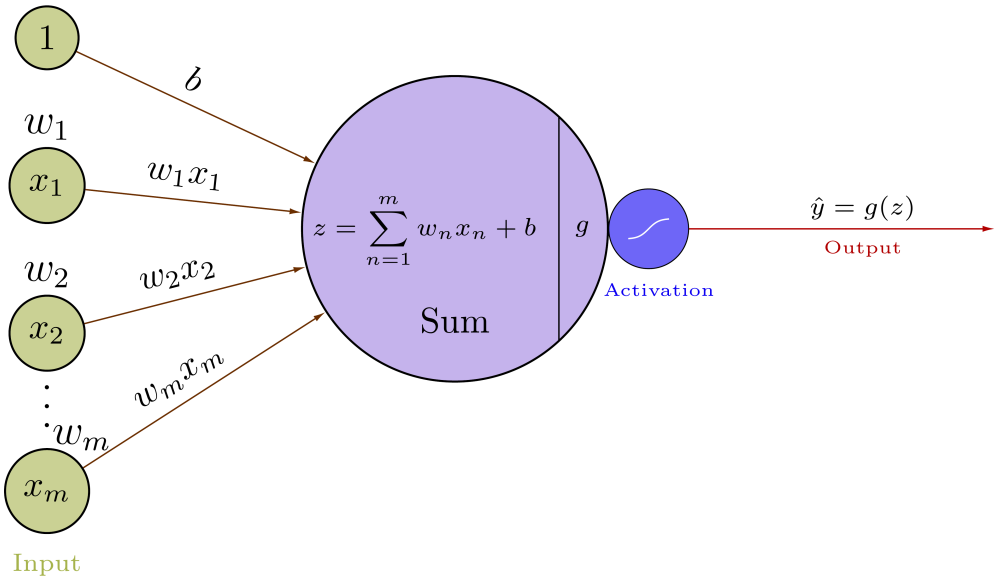
$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$

- Example: sigmoid function

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$



Slide source: MIT 6.S191

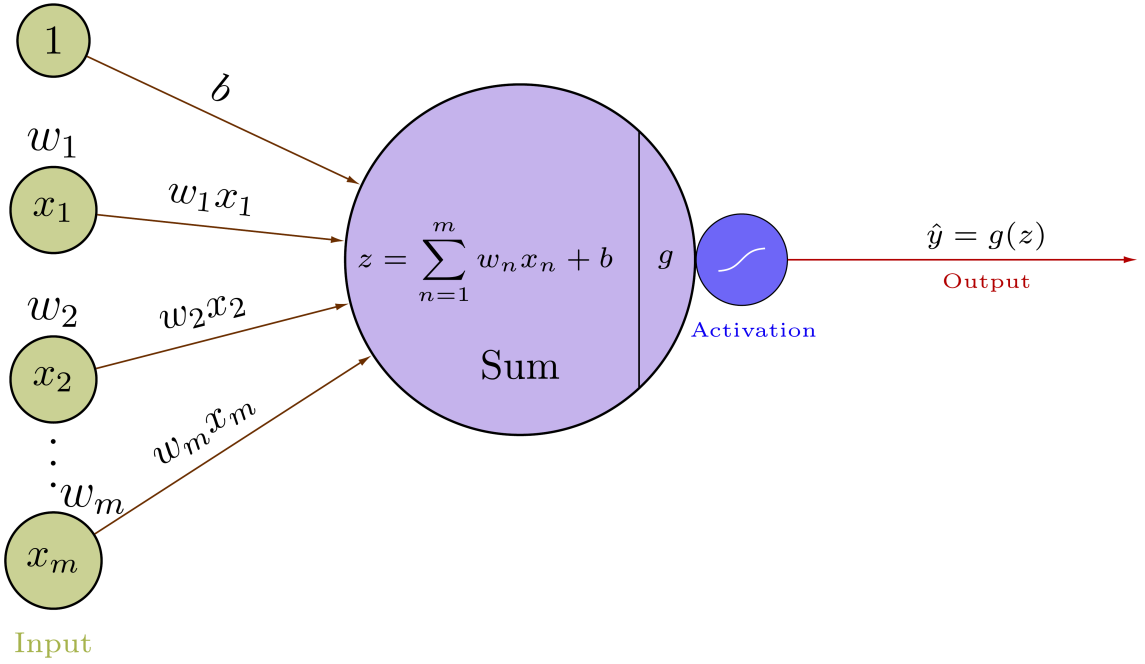


$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \qquad \mathbf{W} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} \qquad \mathbf{X}^T = \begin{bmatrix} x_1 & x_2 & \cdots & x_m \end{bmatrix}$$

$$z = \sum_{n=1}^{m} w_n x_n + b = \mathbf{X}^T \cdot \mathbf{W} + b = \mathbf{W}^T \cdot \mathbf{X} + b$$

$$\hat{y} = g(z)$$

$$\mathcal{L}(\hat{y}, y) = -\frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right)$$



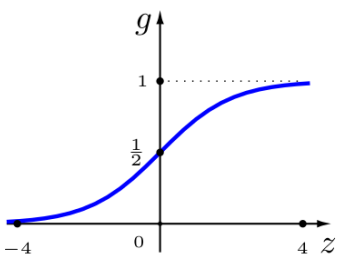
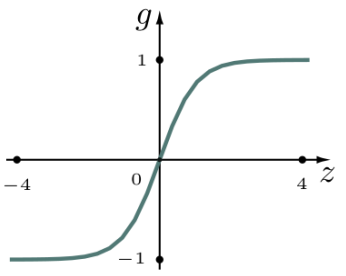
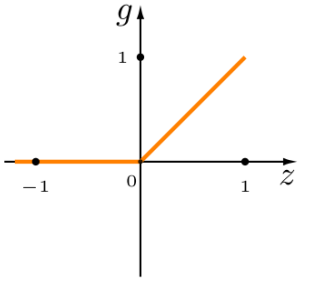
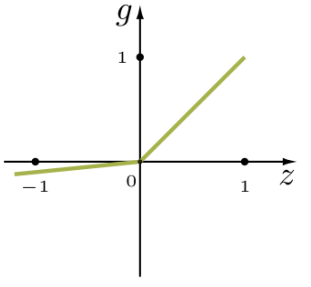
Пряме поширення

$$z = \sum_{n=1}^{m} w_n x_n + b = \mathbf{X}^T \cdot \mathbf{W} + b = \mathbf{W}^T \cdot \mathbf{X} + b$$

$$\hat{y} = g(z)$$

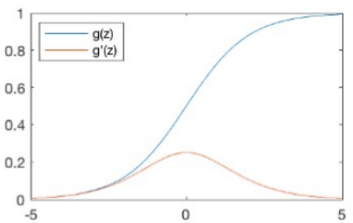
$$\mathcal{L}(\hat{y}, y) = -\frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right)$$

Загальні функції активації

Sigmoid	**Tanh**	**ReLU**	**Leaky ReLU**
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = max(0, z)$	$g(z) = max(\epsilon z, z)$ $\epsilon \ll 1$
			

Common Activation Functions

Sigmoid Function

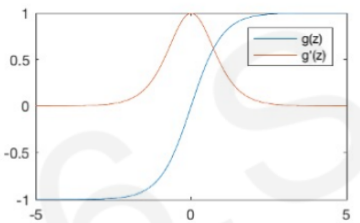


$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

`tf.math.sigmoid(z)`

Hyperbolic Tangent

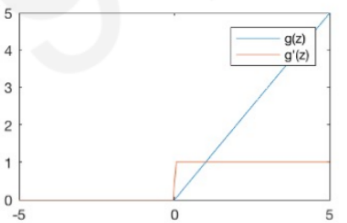


$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

`tf.math.tanh(z)`

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

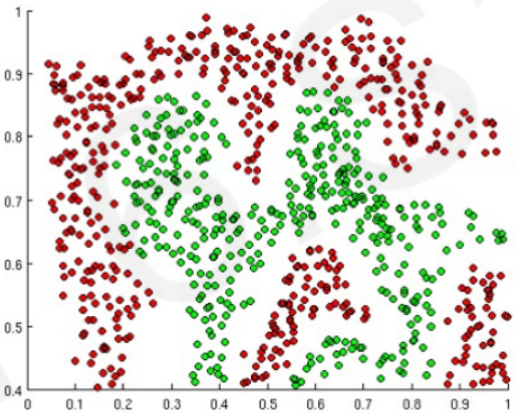
`tf.nn.relu(z)`

TensorFlow code blocks

NOTE: All activation functions are non-linear

Importance of Activation Functions

*The purpose of activation functions is to **introduce non-linearities** into the network*

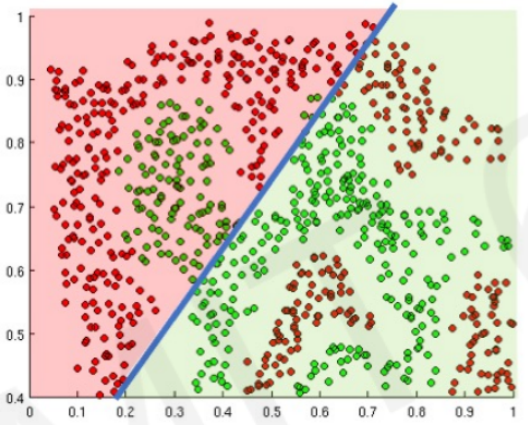


What if we wanted to build a neural network to distinguish green vs red points?

Slide source: MIT 6.S191

Importance of Activation Functions

*The purpose of activation functions is to **introduce non-linearities** into the network*

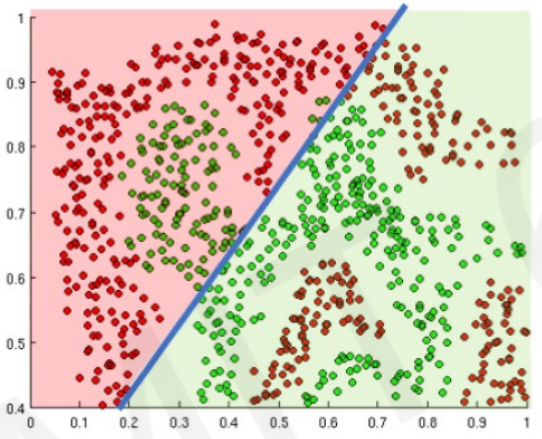


Linear activation functions produce linear decisions no matter the network size

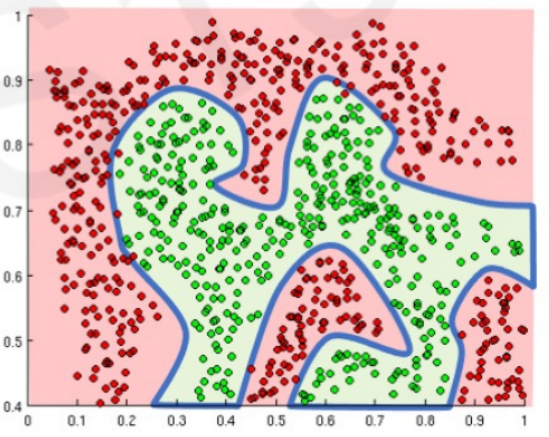
Slide source: MIT 6.S191

Importance of Activation Functions

*The purpose of activation functions is to **introduce non-linearities** into the network*



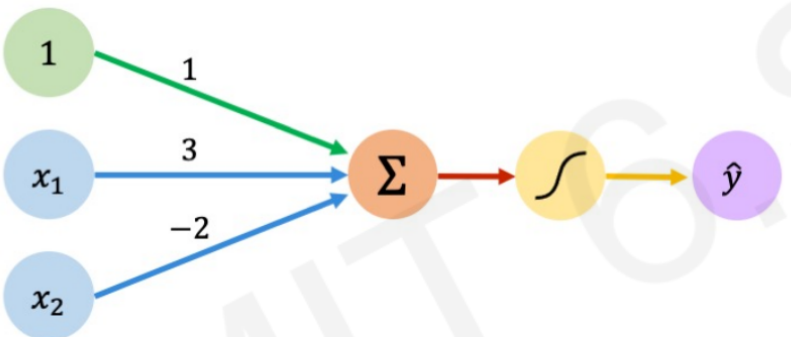
Linear activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

Slide source: MIT 6.S191

The Perceptron: Example



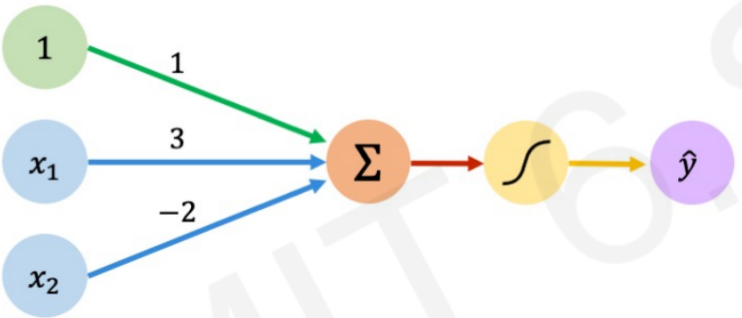
We have: $w_0 = 1$ and $\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

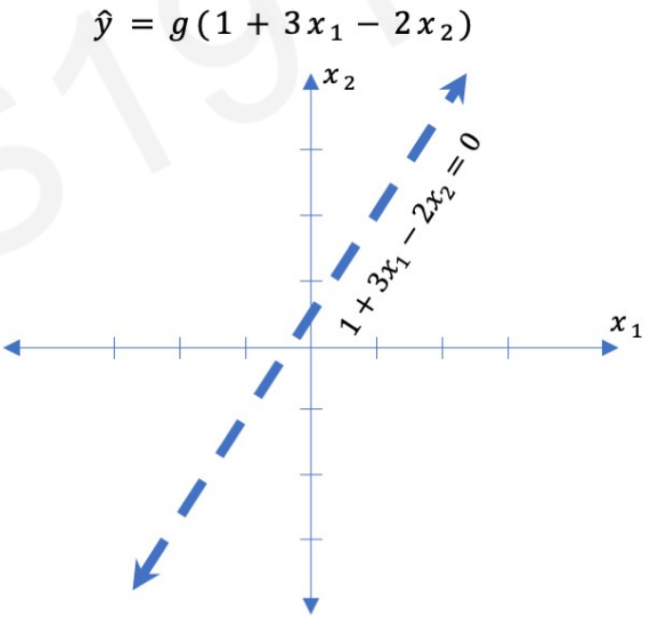
$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$

$$\hat{y} = g(\underbrace{1 + 3x_1 - 2x_2}_{\text{This is just a line in 2D!}})$$

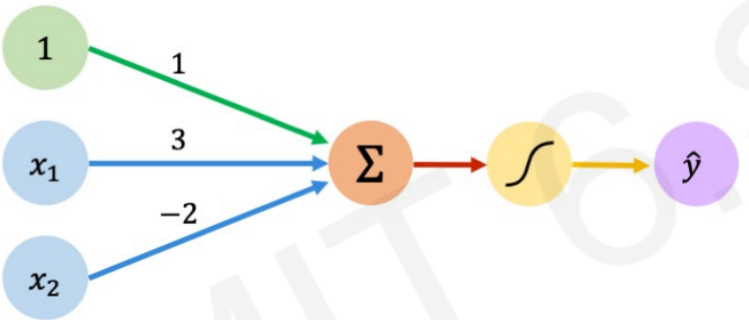
The Perceptron: Example





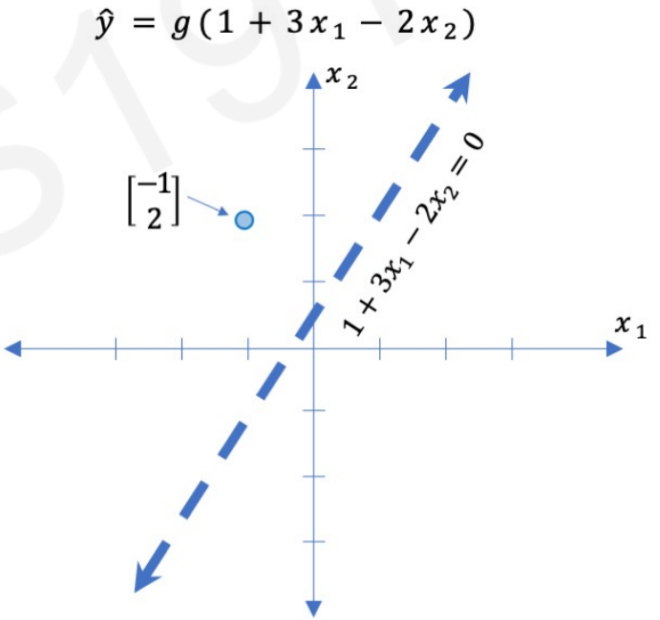
Slide source: MIT 6.S191

The Perceptron: Example

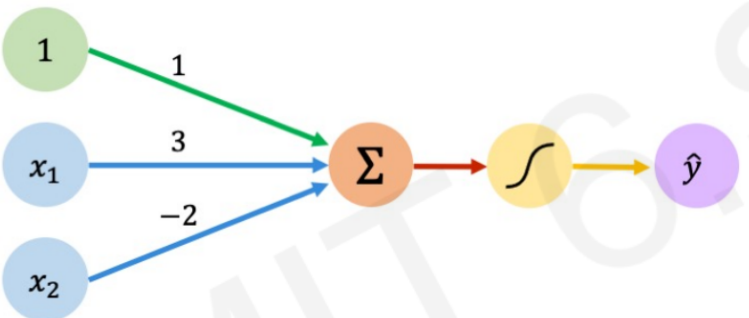


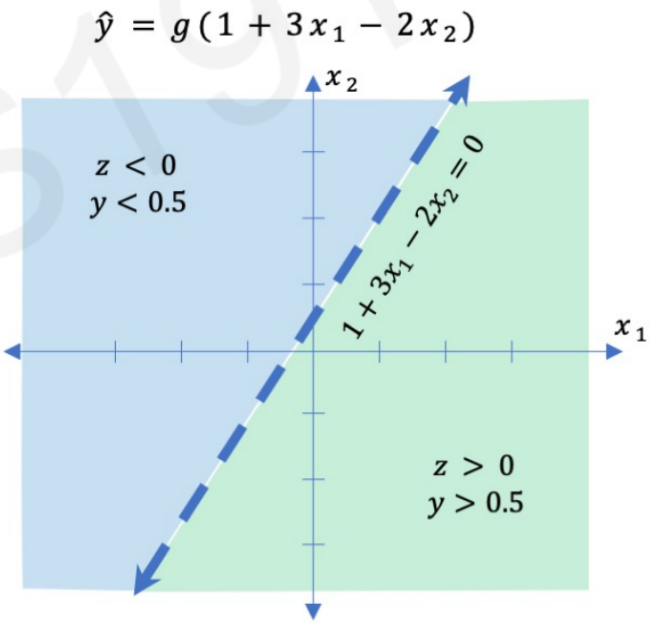
Assume we have input: $\boldsymbol{X} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$

$$\hat{y} = g(1 + (3 * -1) - (2 * 2))$$
$$= g(-6) \approx 0.002$$



The Perceptron: Example





Slide source: MIT 6.S191

Приклад **2**

Припустимо кількість вхідних ознак $m = 3$

$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -0.1 \\ 0.7 \\ 0.5 \end{bmatrix} \qquad \mathbf{W} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 2 \end{bmatrix} \qquad b = 0.8$$

$$z = \sum_{n=1}^{3} w_n x_n + b = w_1 x_1 + w_2 x_2 + w_3 x_3 + b =$$
$$= 1 \cdot -0.1 + -2 \cdot 0.7 + 2 \cdot 0.5 + 0.8 = 0.3$$

$$z = \mathbf{X}^T \cdot \mathbf{W} + b = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} + b =$$
$$= w_1 x_1 + w_2 x_2 + w_3 x_3 + b = 0.3$$

$$\hat{y} = g(z) = g(\mathbf{X}^T \cdot \mathbf{W} + b) = \frac{1}{1 + \exp(-z)} = \frac{1}{1 + \exp(-0.3)} \approx 0.57$$

Одновимірний градієнтний спуск

Одновимірний градієнтний спуск

Розглянемо деяку неперервну диференційовану дійсну функцію $f : \mathbb{R} \to \mathbb{R}$. Розкладаючи у ряд Тейлора, ми отримуємо::

$$f(x + \varepsilon) = f(x) + \varepsilon f^{'}(x) + \mathcal{O}(\varepsilon^2)$$

Щоб усе було просто, давайте виберемо фіксований розмір кроку $\alpha > 0$ та оберемо $\varepsilon = -\alpha f^{'}(x)$. Підставляючи це у ряд Тейлора, отримаємо:

$$f(x - \alpha f^{'}(x)) = f(x) - \alpha f^{'2}(x) + \mathcal{O}(\alpha^2 f^{'2}(x))$$

Якщо похідна $f^{'}(x) \neq 0$ не зникає ми робимо прогрес так як $\alpha f^{'2}(x) > 0$. Крім того, ми завжди можемо вибрати α досить малим, щоб вирази вищих порядків стали нерелевантними. Тому ми приходимо до

$$f(x - \alpha f^{'}(x)) \lessapprox f(x)$$

Це означає, якщо ми використовуємо

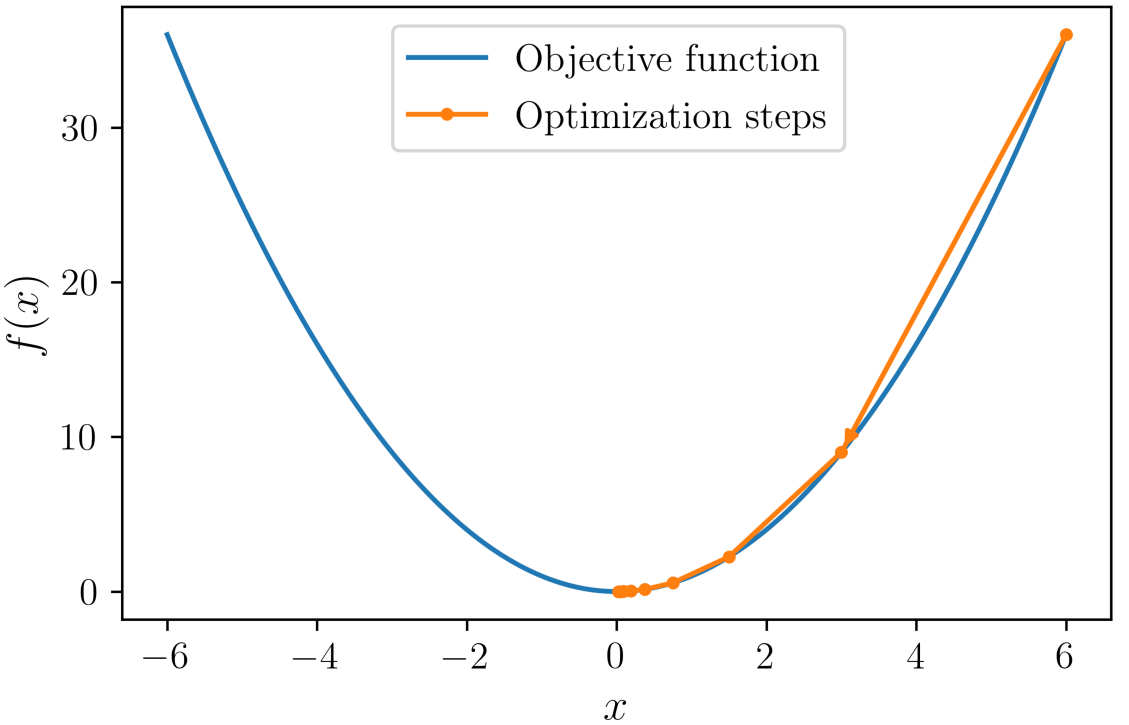
$$x \leftarrow x - \alpha f^{'}(x)$$

для ітерації по x, значення функції $f(x)$ може зменшитись.

```python
import numpy as np

def f(x):  # Objective function
    return x**2

def f_grad(x):  # Gradient (derivative) of the objective function
    return 2 * x

def bgd(alpha, f_grad):
    x = 6.0         # Initial value of x
    results = [x]
    epoch = 8       # Number of iterations
    for i in range(epoch):
        x -= alpha * f_grad(x)
        results.append(float("%.6f" % x))
    print(f'epoch {epoch}, x: {x:.6f}')
    return results

results = bgd(0.25, f_grad)
print(results)
```

```
epoch 8, x: 0.023438
[6.0, 3.0, 1.5, 0.75, 0.375, 0.1875, 0.09375, 0.046875, 0.023438]
```

The progress of optimizing over x

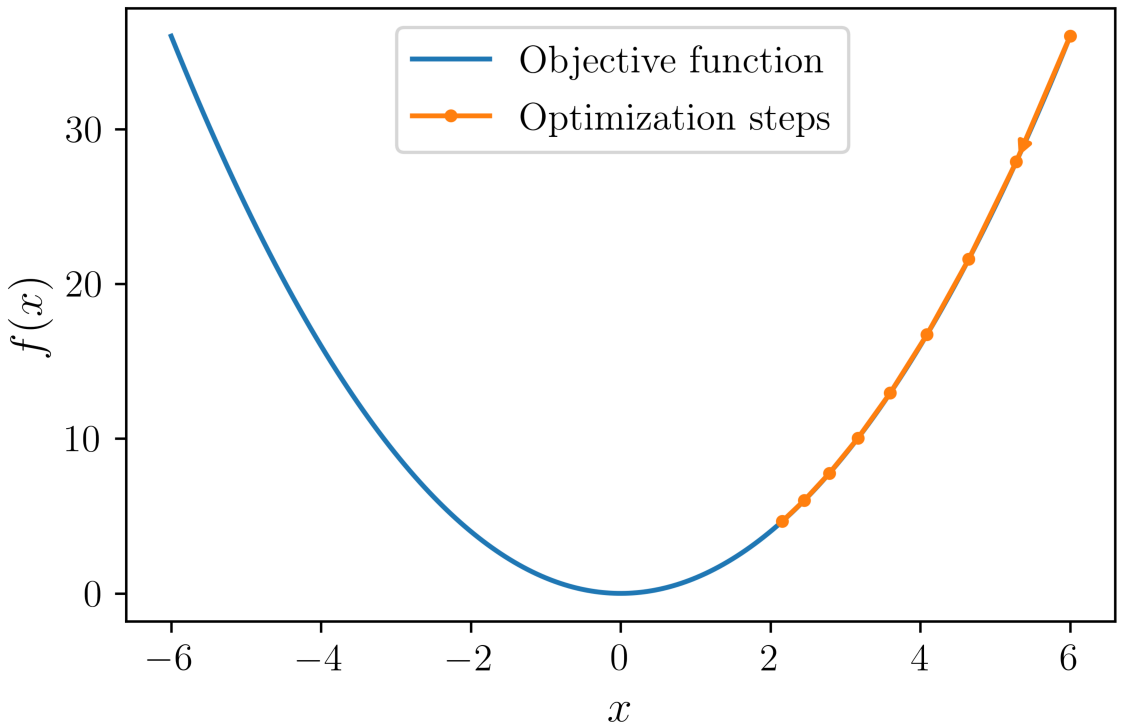


One-Dimensional Gradient Descent ($\alpha = 0.25$)

The progress of optimizing over x

```
results = bgd(0.06, f_grad)
```

```
epoch 8, x: 2.157807
```

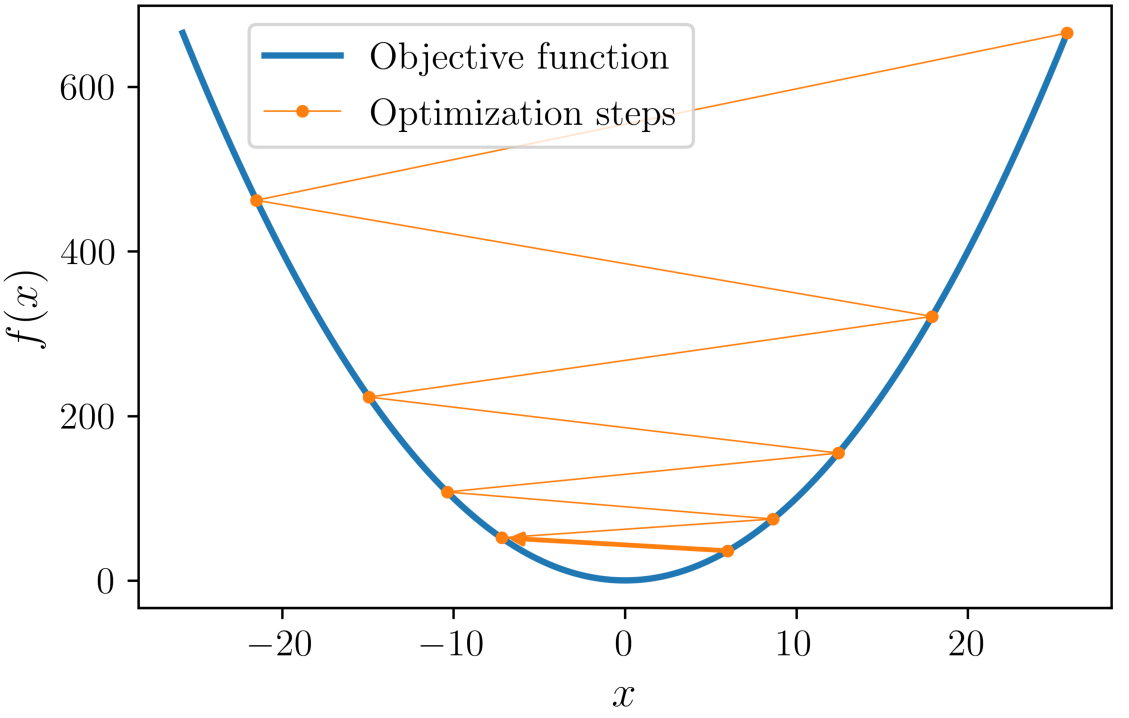


One-Dimensional Gradient Descent ($\alpha = 0.06$)

The progress of optimizing over x

```
results = bgd(1.1, f_grad)
```

```
epoch 8, x: 25.798902
```



One-Dimensional Gradient Descent ($\alpha = 1.1$)

Перцептрон: Зворотне поширення

In Leibniz notations, the chain rule states that

$$\frac{\partial \ell}{\partial \theta_i} = \sum_{k \in \text{parents}(\ell)} \frac{\partial \ell}{\partial u_k} \underbrace{\frac{\partial u_k}{\partial \theta_i}}_{\text{recursive case}}$$

Backpropagation

- Since a neural network is a composition of differentiable functions, the total derivatives of the loss can be evaluated backward, by applying the chain rule recursively over its computational graph.
- The implementation of this procedure is called reverse automatic differentiation or backpropagation.

Forward propagation

$$z = \sum_{n=1}^{m} w_n x_n + b = \mathbf{X}^T \cdot \mathbf{W} + b = \mathbf{W}^T \cdot \mathbf{X} + b$$

$$\hat{y} = g(z) = \sigma(z) = \frac{1}{1 + \exp(-z)}$$

$$\mathcal{L}(\hat{y}, y) = -\frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right)$$

Backward propagation

$$\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}}$$

$$\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial z} = \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} = \hat{y} - y$$

$$\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial \mathbf{W}} = \mathbf{X}^T \cdot (\hat{y} - y)$$

$$\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial b} = \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial b} = \hat{y} - y$$

Update the weights and bias

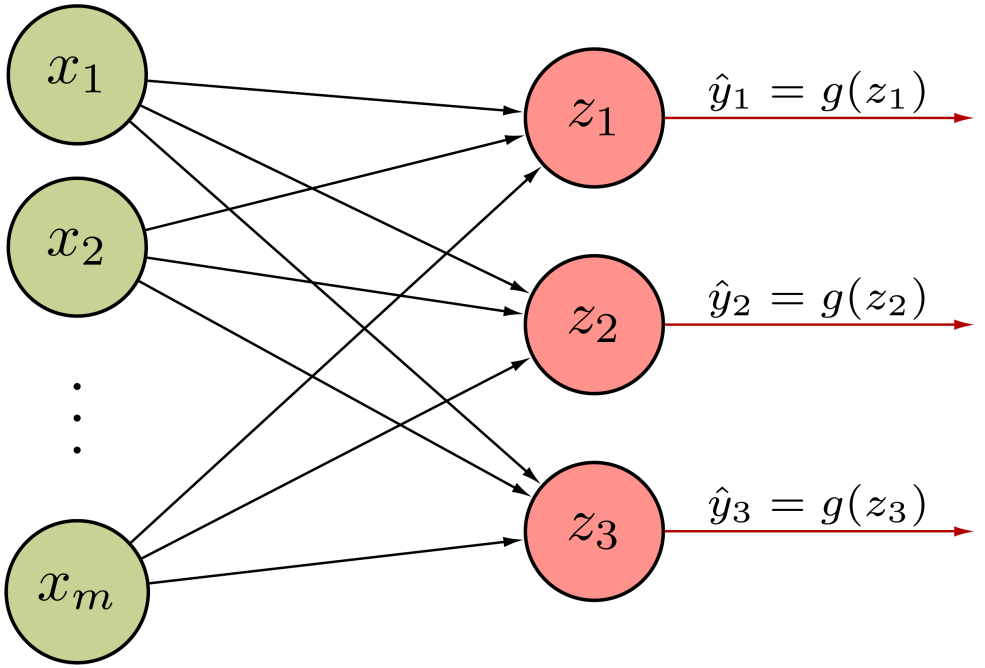
$$\mathbf{W} = \mathbf{W} - \alpha \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \mathbf{W}}$$

$$b = b - \alpha \frac{\partial \mathcal{L}(\hat{y}, y)}{\partial b}$$

Multi Output Perceptron

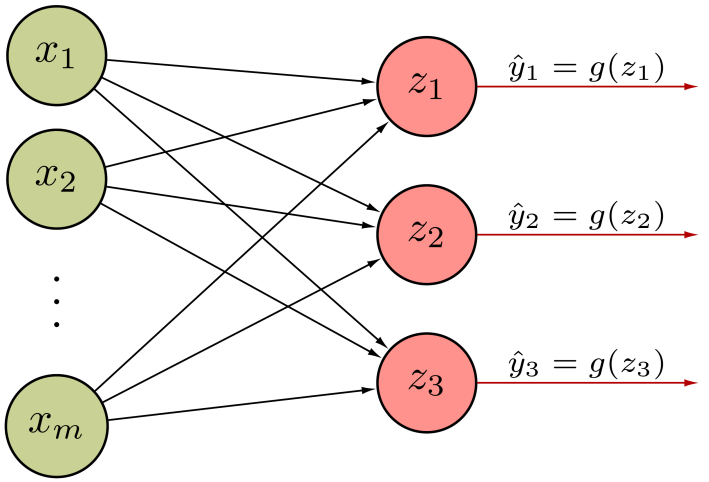
Multi Output Perceptron

Because all inputs are densely connected to all outputs, these layers are called Dense layers



$$z_j = \sum_{n=1}^{m} w_{j,n} x_n + b_j$$

Example

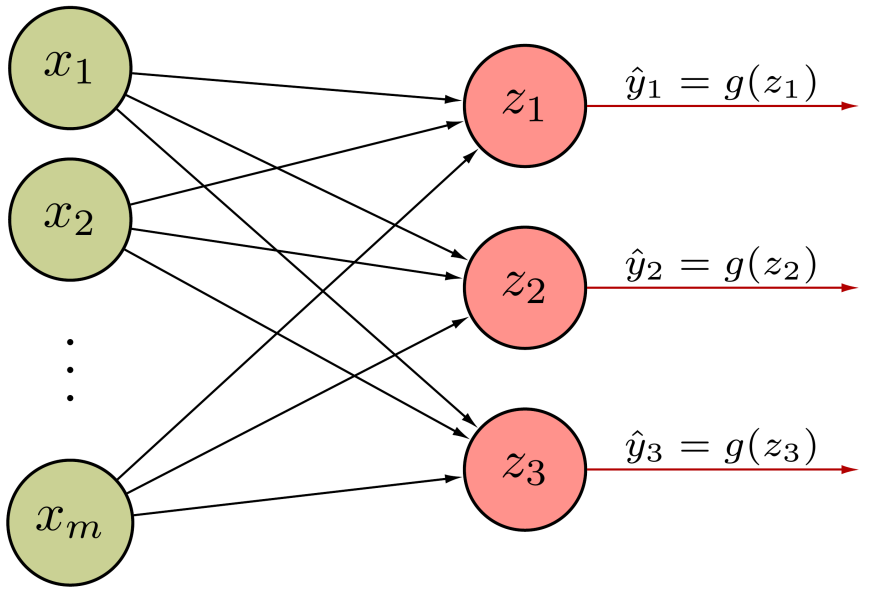


$$\mathbf{X}^{m\times 1} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \qquad \mathbf{W}^{3\times m} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ w_{31} & w_{32} & \cdots & w_{3m} \end{bmatrix} \qquad \mathbf{b}^{3\times 1} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$\mathbf{z} = \mathbf{W}\cdot\mathbf{X} + \mathbf{b} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ w_{31} & w_{32} & \cdots & w_{3m} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} =$$

$$= \begin{bmatrix} w_{11}x_1 + w_{12}x_2 + \cdots + w_{1m}x_m + b_1 \\ w_{21}x_1 + w_{22}x_2 + \cdots + w_{2m}x_m + b_2 \\ w_{31}x_1 + w_{32}x_2 + \cdots + w_{3m}x_m + b_3 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

Because all inputs are densely connected to all outputs, these layers are called Dense layers

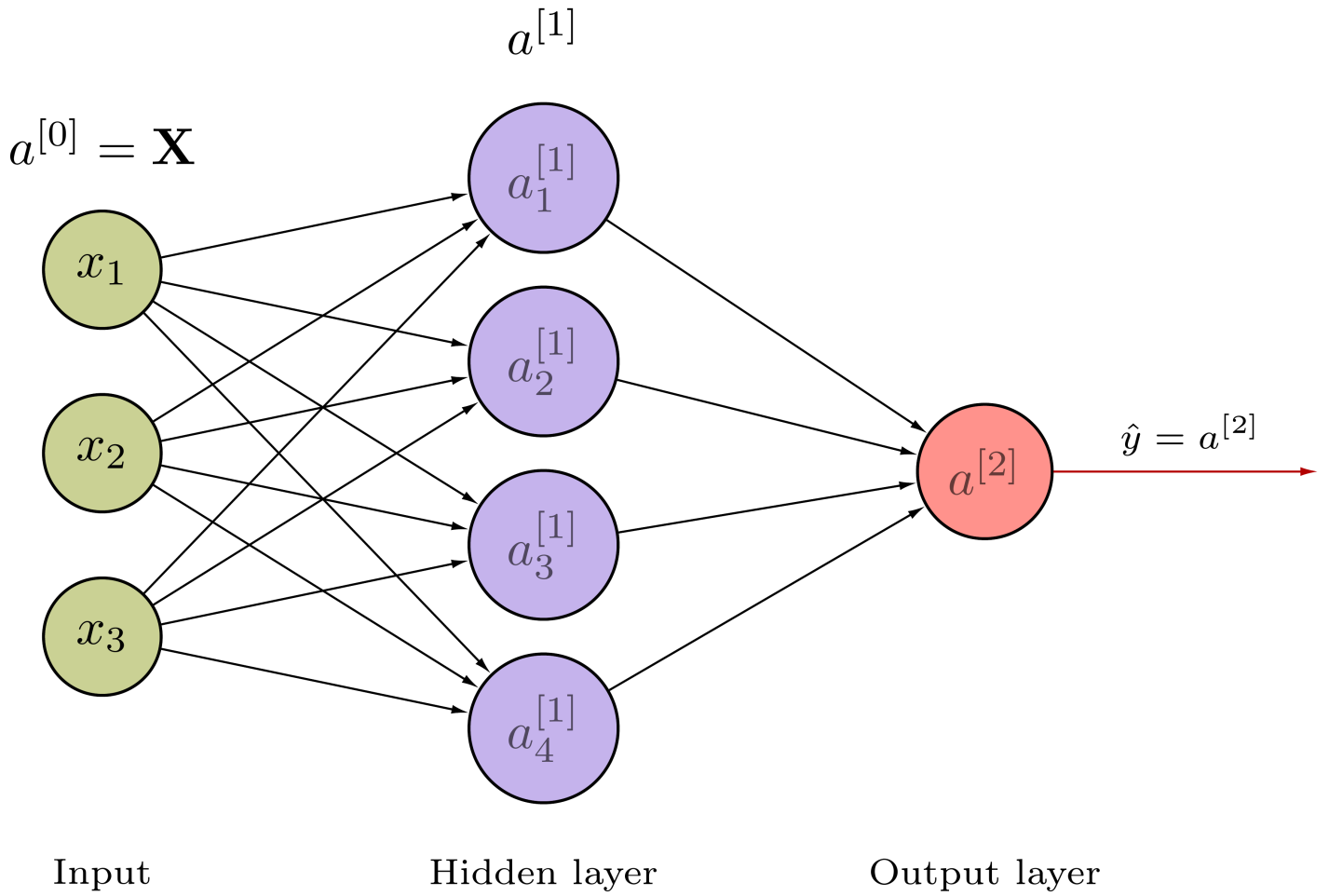



```python
import tensorflow as tf

layer = tf.keras.layers.Dense(
    units=3)
```

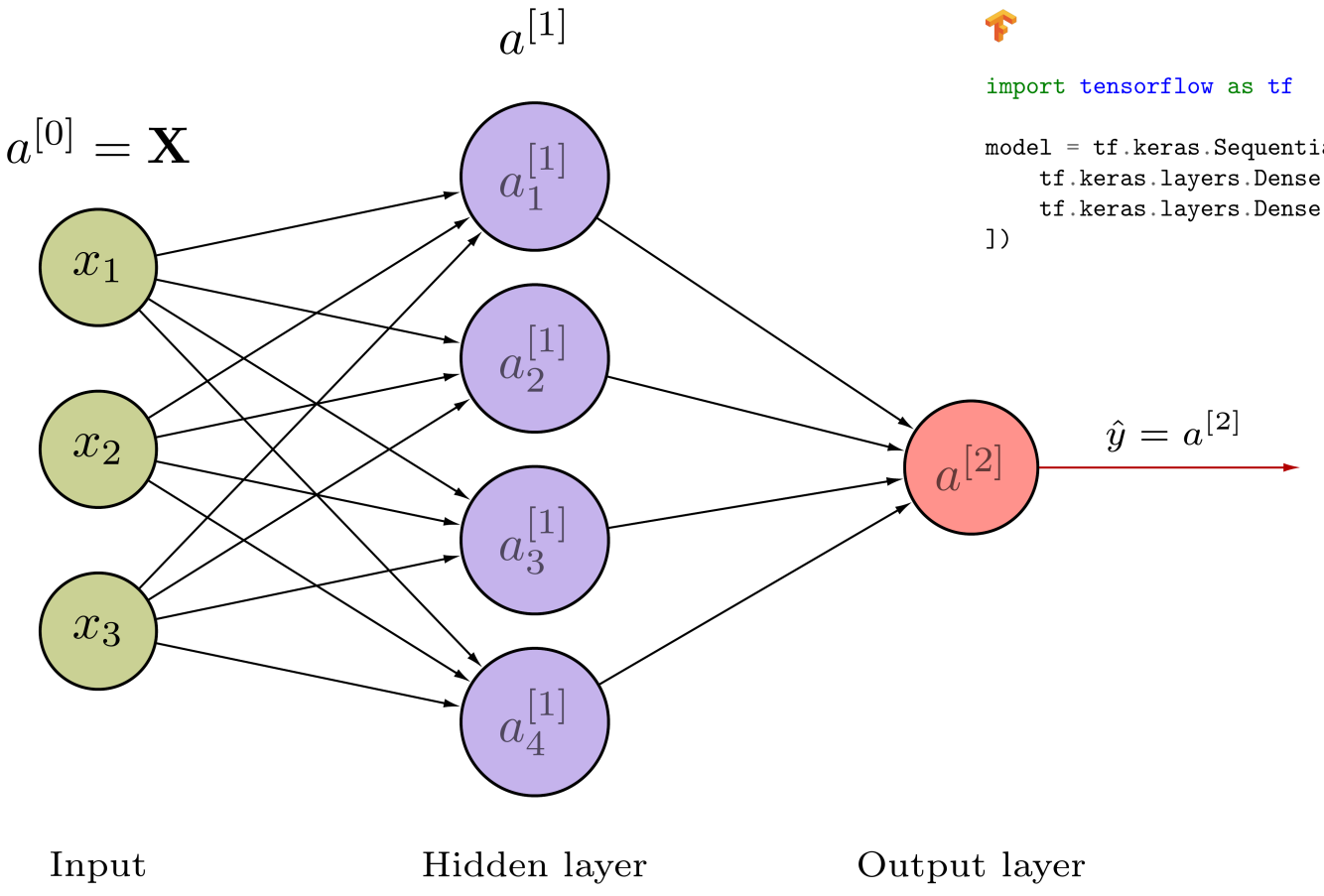
$$z_j = \sum_{n=1}^{m} w_{j,n} x_n + b_j$$

Multilayer Perceptron

One hidden layer Neural Network

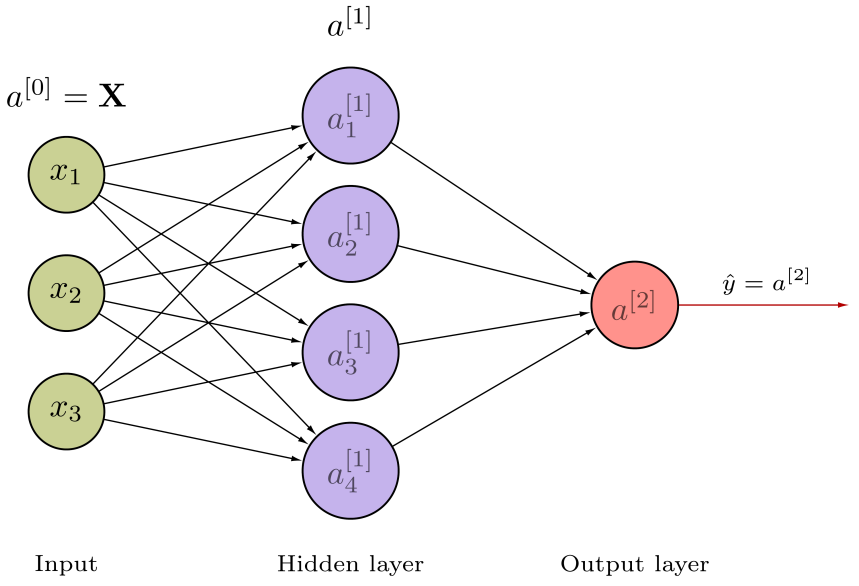


One hidden layer Neural Network




```python
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Dense(4),
    tf.keras.layers.Dense(1)
])
```

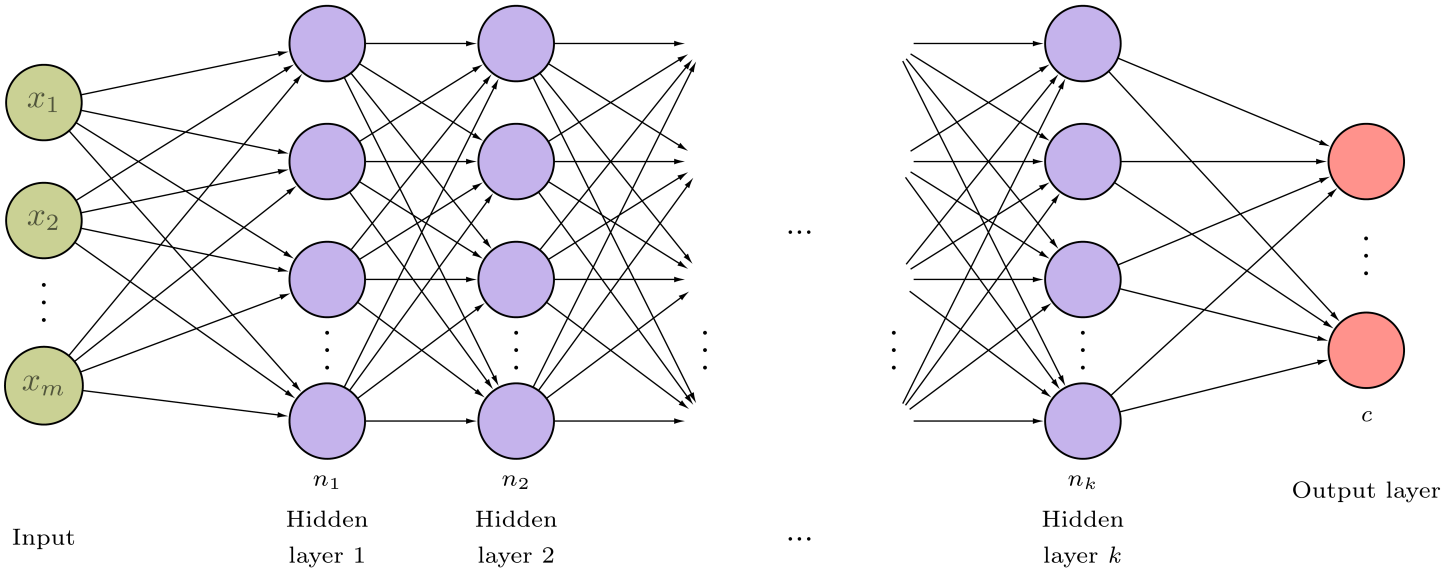
One hidden layer Neural Network



$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \mathbf{W}^{[1]} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix} \qquad \mathbf{b}^{[1]} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \qquad \mathbf{W}^{[2]} = \begin{bmatrix} w_1 & w_2 & w_3 & w_4 \end{bmatrix} \qquad b^{[2]} = b$$

$$\begin{aligned} \mathbf{z}^{[1]} &= \mathbf{W}^{[1]} \cdot \mathbf{X} + \mathbf{b}^{[1]} \\ \mathbf{a}^{[1]} &= g^{[1]}(\mathbf{z}^{[1]}) \\ z^{[2]} &= \mathbf{W}^{[2]} \cdot \mathbf{a}^{[1]} + b^{[2]} \\ \hat{y} &= a^{[2]} = g^{[2]}(z^{[2]}) \end{aligned}$$

Deep Neural Network




```python
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Dense(n1),
    tf.keras.layers.Dense(n2),
    .
    .
    .

    tf.keras.layers.Dense(nk),
    tf.keras.layers.Dense(c)
])
```

Applying Neural Networks

Example Problem

Will I pass this class?

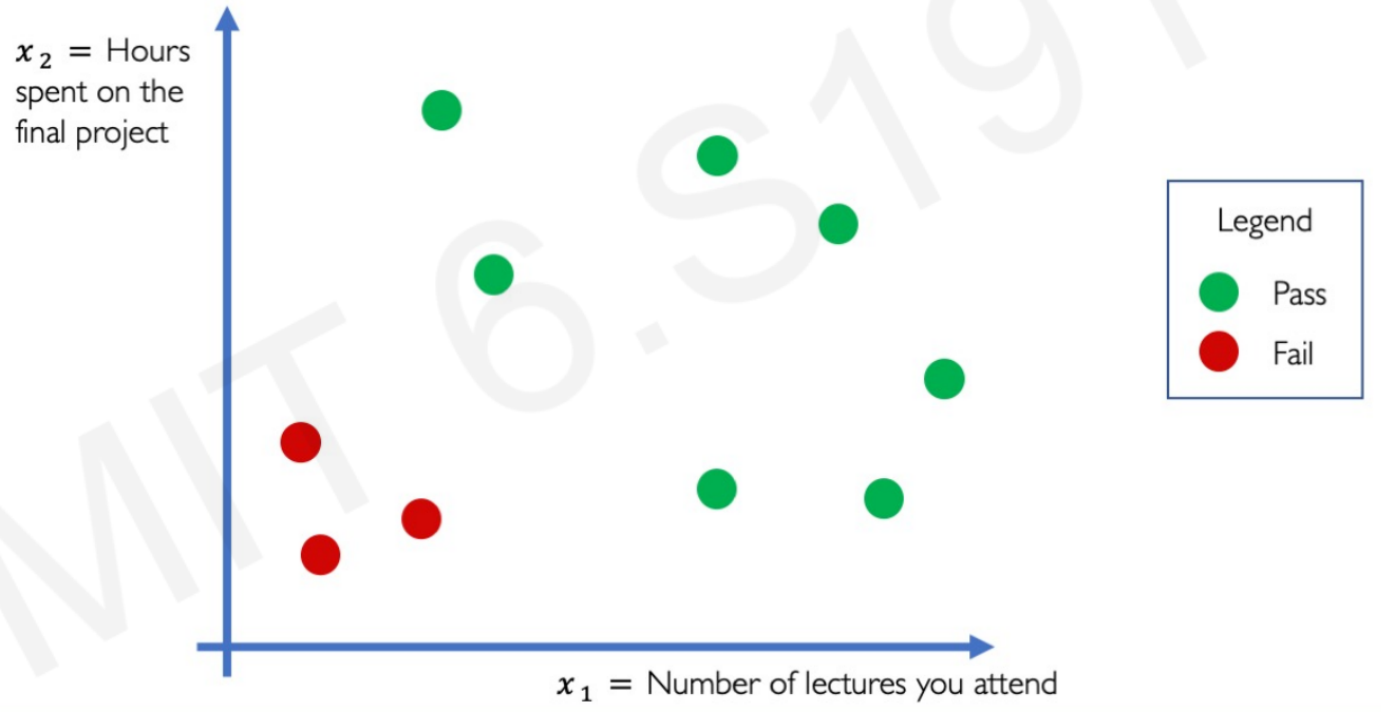
Let's start with a simple two feature model

x_1 = Number of lectures you attend

x_2 = Hours spent on the final project

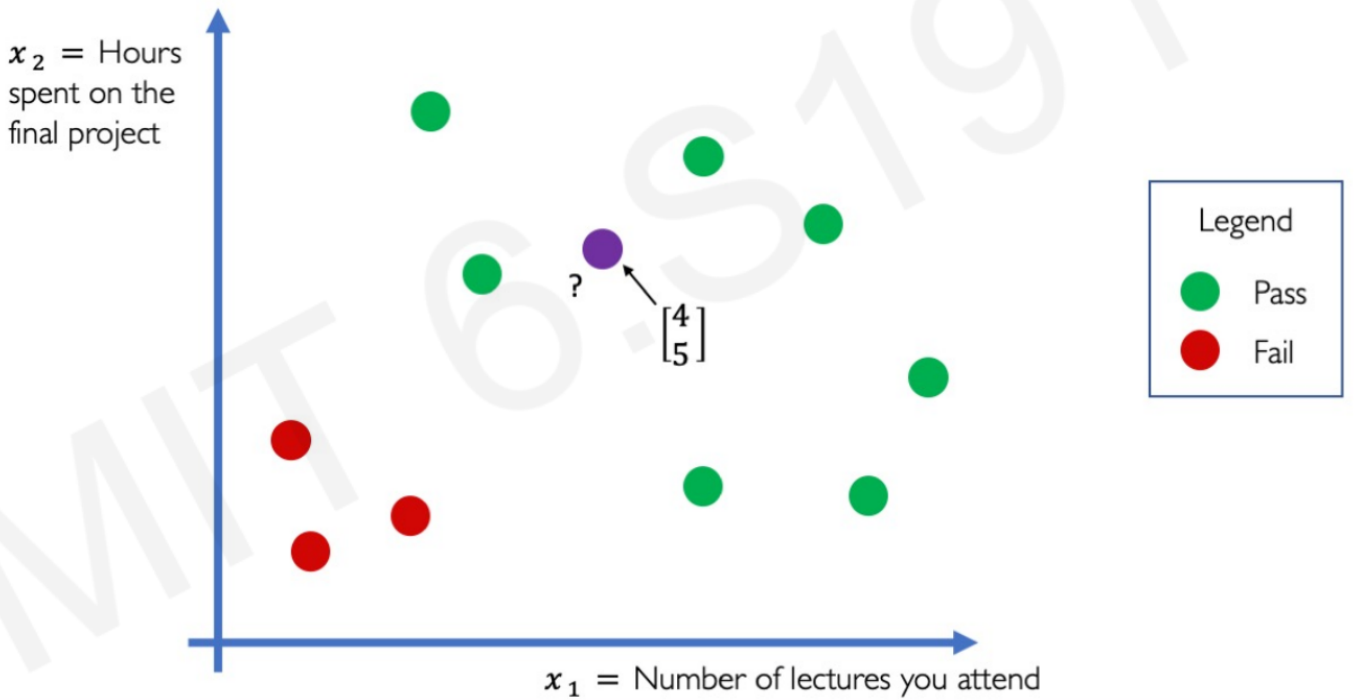
Slide source: MIT 6.S191

Example Problem: Will I pass this class?



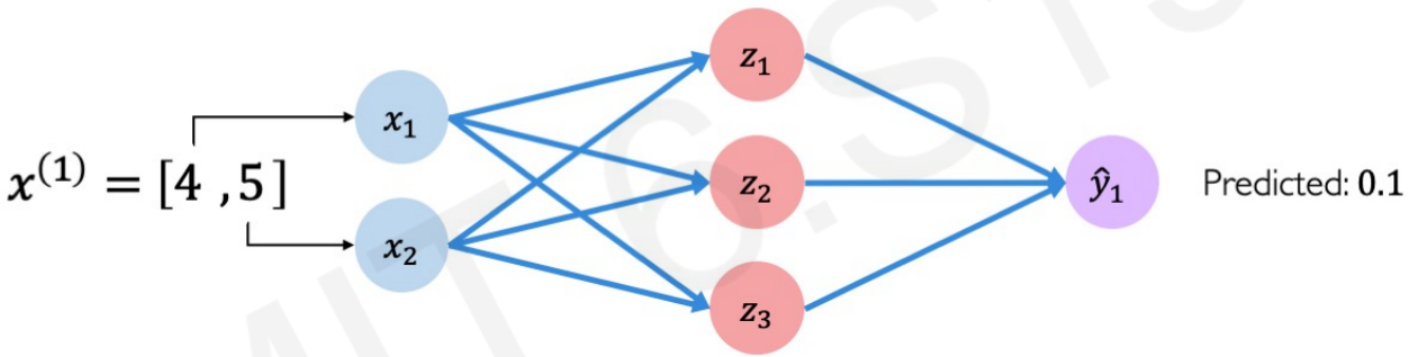
Slide source: MIT 6.S191

Example Problem: Will I pass this class?



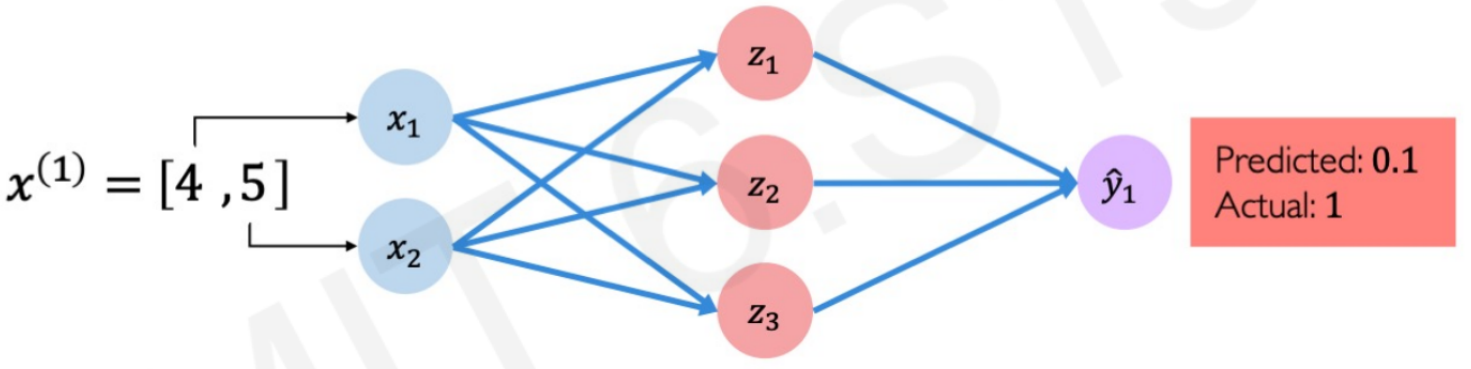
Slide source: MIT 6.S191

Example Problem: Will I pass this class?



Slide source: MIT 6.S191

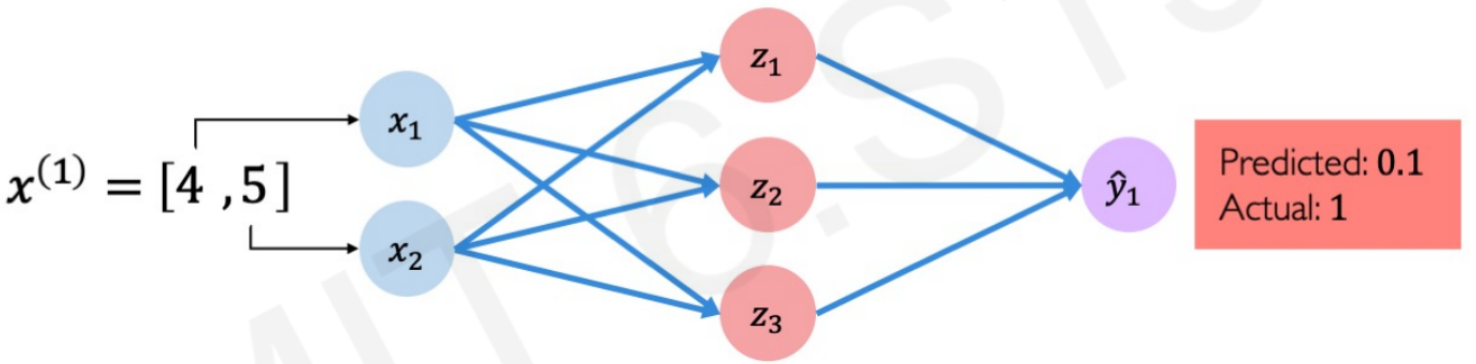
Example Problem: Will I pass this class?



Slide source: MIT 6.S191

Quantifying Loss

*The **loss** of our network measures the cost incurred from incorrect predictions*

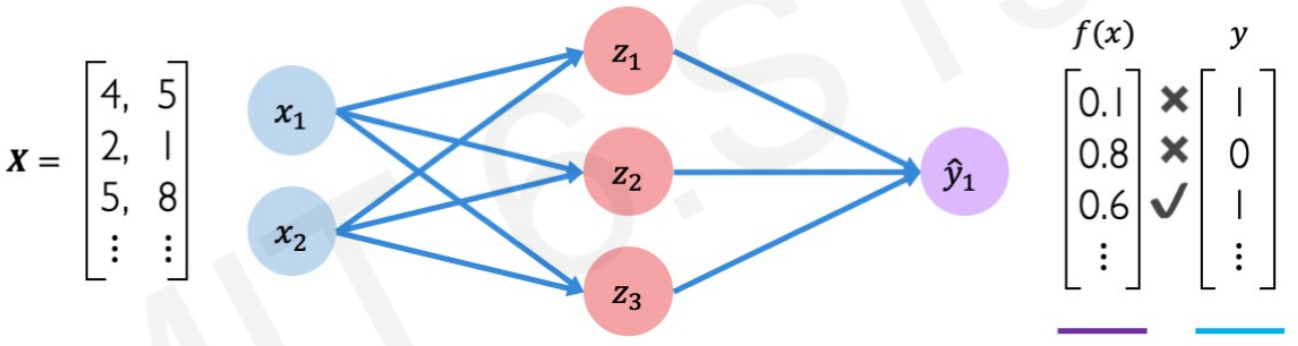


$$\mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

Predicted Actual

Empirical Loss

*The **empirical loss** measures the total loss over our entire dataset*



$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

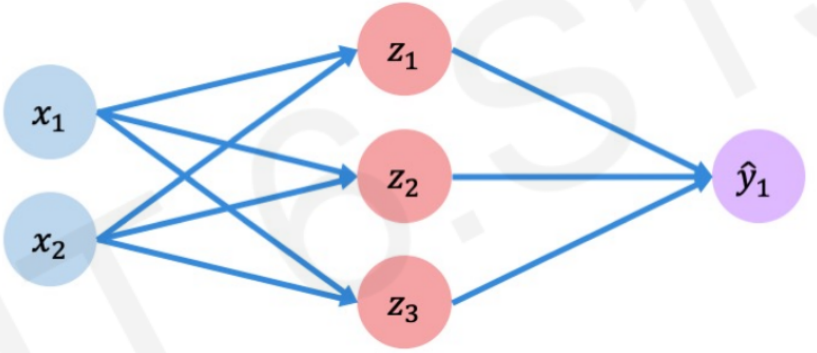
Predicted Actual

Also known as:
- Objective function
- Cost function
- Empirical Risk

Binary Cross Entropy Loss

Cross entropy loss *can be used with models that output a probability between 0 and 1*

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$f(x)$ y

$$\begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

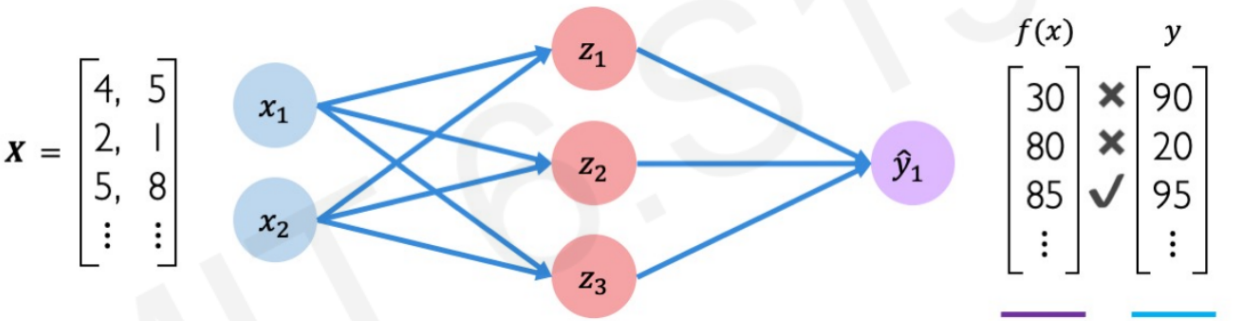
$$J(\boldsymbol{W}) = -\frac{1}{n}\sum_{i=1}^{n} \underbrace{y^{(i)}}_{\text{Actual}} \log \underbrace{\left(f\left(x^{(i)}; \boldsymbol{W}\right)\right)}_{\text{Predicted}} + \underbrace{(1 - y^{(i)})}_{\text{Actual}} \log \underbrace{\left(1 - f\left(x^{(i)}; \boldsymbol{W}\right)\right)}_{\text{Predicted}}$$

```
loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(y, predicted) )
```


Slide source: MIT 6.S191

Mean Squared Error Loss

Mean squared error loss *can be used with regression models that output continuous real numbers*



$$J(W) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - f\left(x^{(i)}; W\right)\right)^2$$

Actual Predicted

```
loss = tf.reduce_mean( tf.square(tf.subtract(y, predicted)) )
loss = tf.keras.losses.MSE( y, predicted )
```

Кінець