

Штучна нейронна мережа

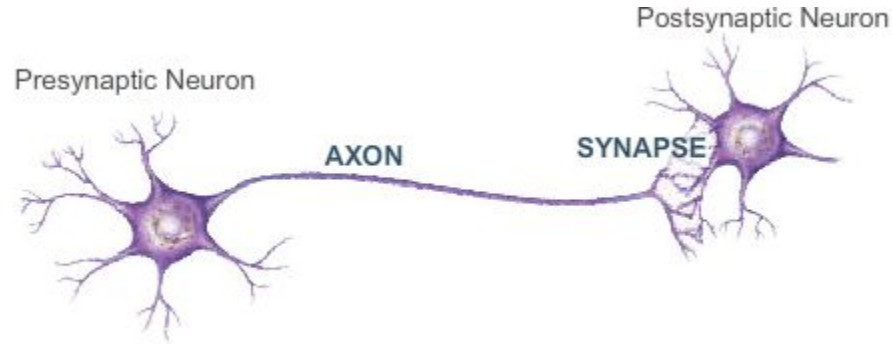
Кізло Тарас



Біологічна нейронна мережа

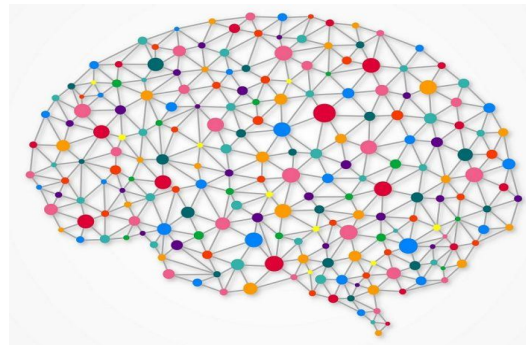
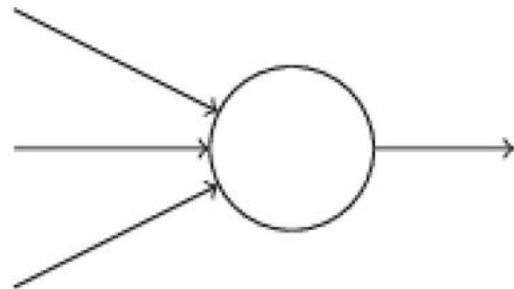
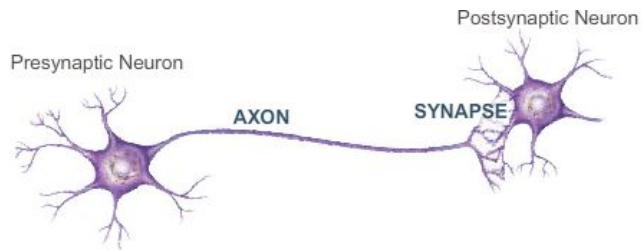


Біологічна нейронна мережа



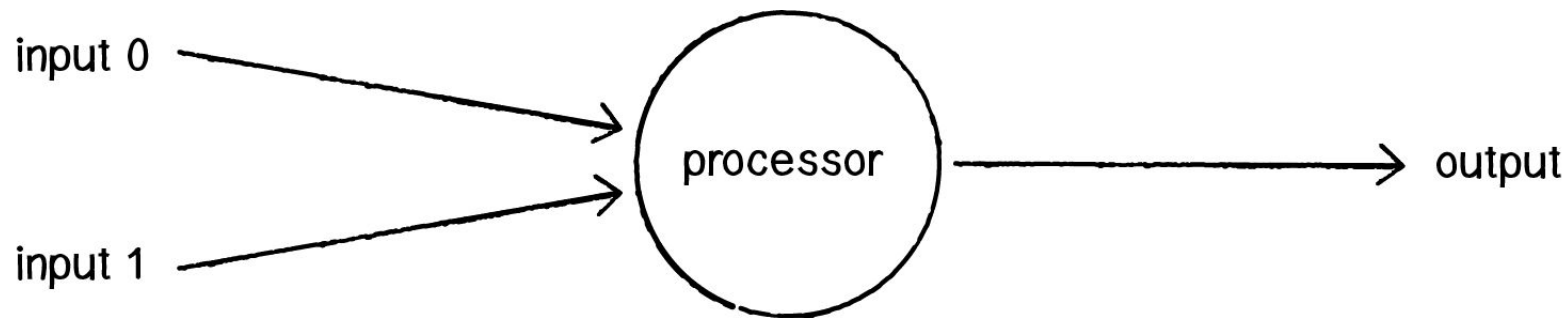
- Нейрон — це клітина яка що **обробляє** та **передає** інформацію у вигляді електричного або хімічного **сигналу**
- Аксон — відросток нервової клітини, що **проводить** імпульси іншим нервовим клітинам
- Передача хімічних сигналів відбувається через **синапси** - контакти між нейронами та іншими клітинами
- Деякі зв'язки між нейронами **сильніші** за інші. Сильніше з'єднання передає **більше** сигналу
- Якщо нейрон отримує **достатньо** енергії він **передає** її далі

Спрощена модель





Парцептрон

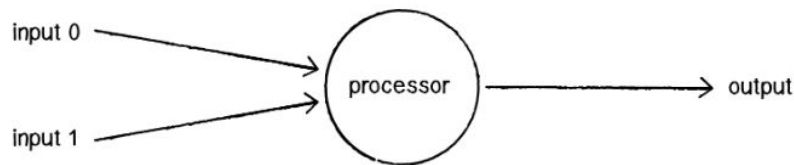




Пряме поширення

Feed forward

Propagation



1. Отримати на вхід аргументи

Input 0: $x_1 = 12$

Input 1: $x_2 = 4$

2. Зважуємо вхідні аргументи

Weight 0: 0.5

Weight 1: -1

Input 0 * Weight 0 $\Rightarrow 12 * 0.5 = 6$

Input 1 * Weight 1 $\Rightarrow 4 * -1 = -4$

3. Сумуємо

Sum = 6 + -4 = 2

4. Генеруємо результат

Output = sign(sum) \Rightarrow sign(2) \Rightarrow +1



Алгоритм парцептрона

1. Кожний вхідний параметр помножити на зв'язок
2. Просумувати результати

$$p_j(t) = \sum_i o_i(t) w_{ij}$$






3. Повернути результат залежно від функції активації

$$o_j(t) = f_{out}(a_j(t))$$

```
public int Output(float[] inputs)
{
    return activationFunction(sumUp(inputs));
}
```

```
private float SumUp(float[] inputs)
{
    float sum = 0;
    for(int i = 0; i < inputs.length; ++i)
    {
        sum += weights[i] * inputs[i];
    }
    return sum;
}
```

```
private int Step(float x)
{
    if (x > 0) return 1;
    else return -1;
}
```

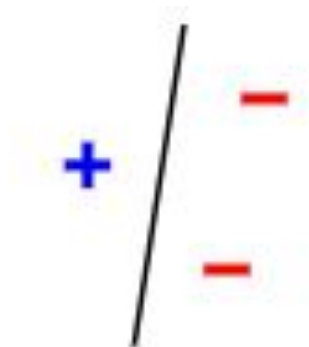
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Relu		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Sigmoid		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Binary Step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Tanh		$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - f(x)^2$



Вміння

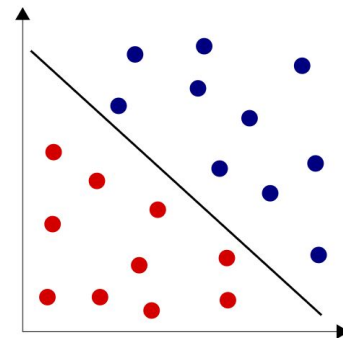
AND

	T	F
T	T	F
F	F	F



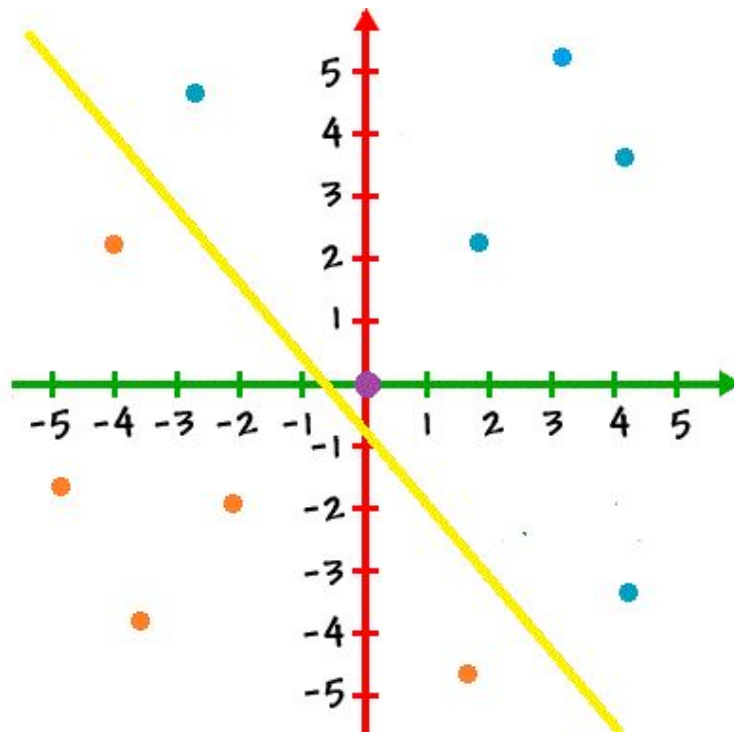
OR

	T	F
T	T	T
F	T	F

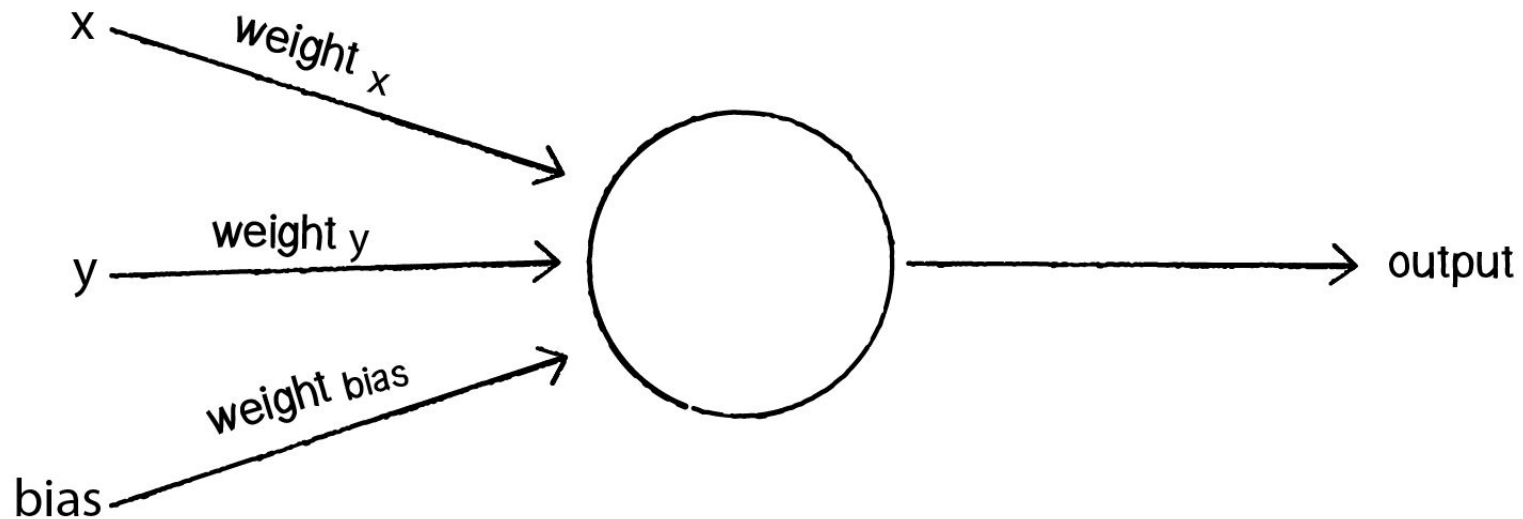


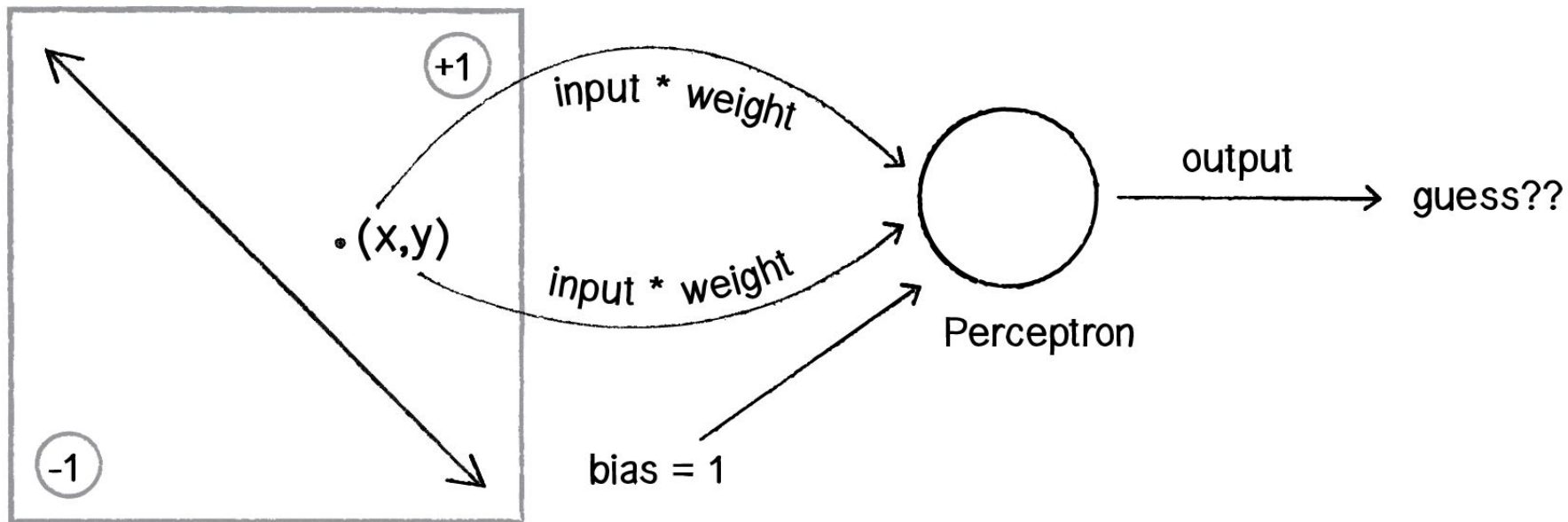


Класифікація



Bias



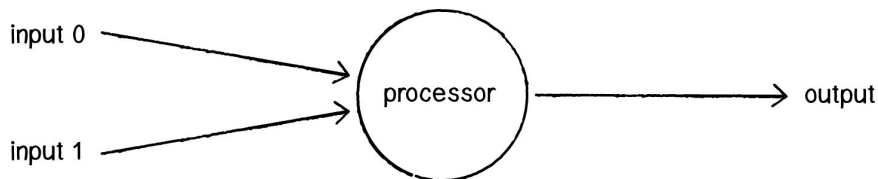




Зворотнє поширення Backpropagation

$$\text{ERROR} = \text{DESIRED OUTPUT} - \text{GUESS OUTPUT}$$

1. Дати парцептрону вхідні параметри на які нам відомий результат
2. Отримати результат від парцептрона
3. Обрахувати помилку
4. **Поправити ваги** згідно помилки
5. Повторити



Desired	Guess	Error
- 1	- 1	0
- 1	+ 1	- 2
+ 1	- 1	+ 2
+ 1	+ 1	0

ERROR = DESIRED OUTPUT - GUESS OUTPUT

$\Delta\text{WEIGHT} = \text{ERROR} * \text{INPUT}$

NEW WEIGHT = WEIGHT + ΔWEIGHT

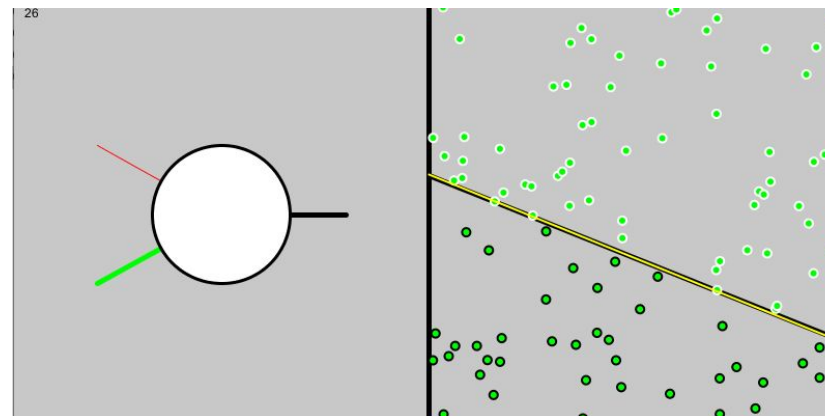
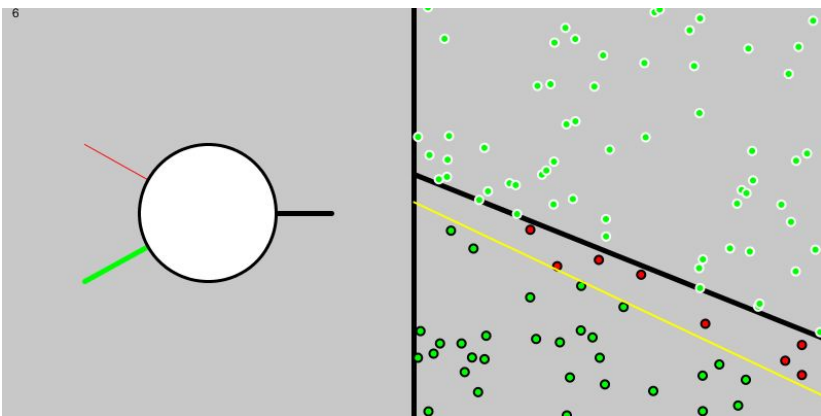
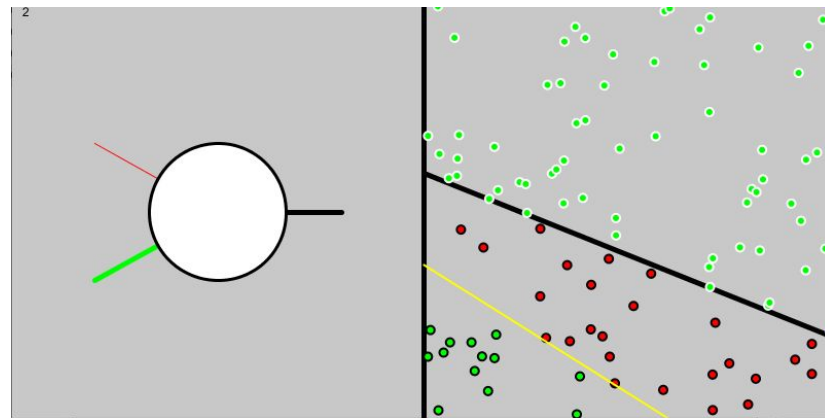
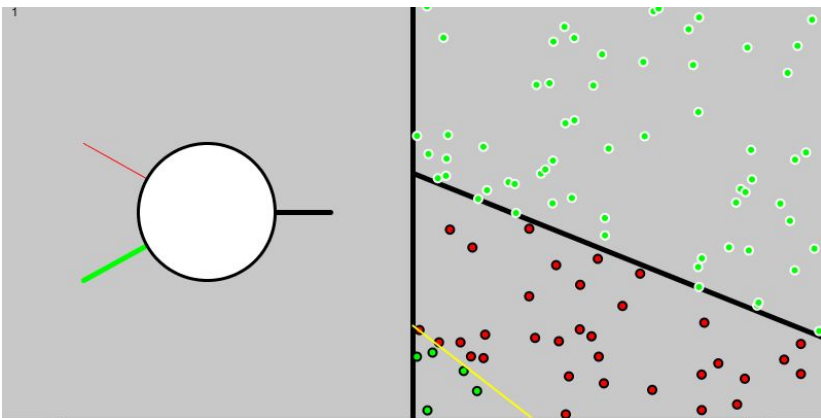
NEW WEIGHT = WEIGHT + ERROR * INPUT

NEW WEIGHT = WEIGHT + ERROR * INPUT * LEARNING CONSTANT

```
private const float learningRate = 0.01;

public void train(float[] inputs, int target)
{
    int error = target - this.output(inputs);

    for(int i = 0; i < weights.length; ++i)
    {
        weights[i] += error * inputs[i] * learningRate;
    }
}
```





AND

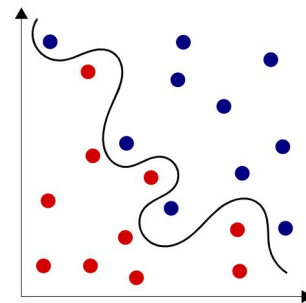
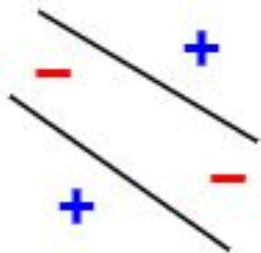
	T	F
T	T	F
F	F	F

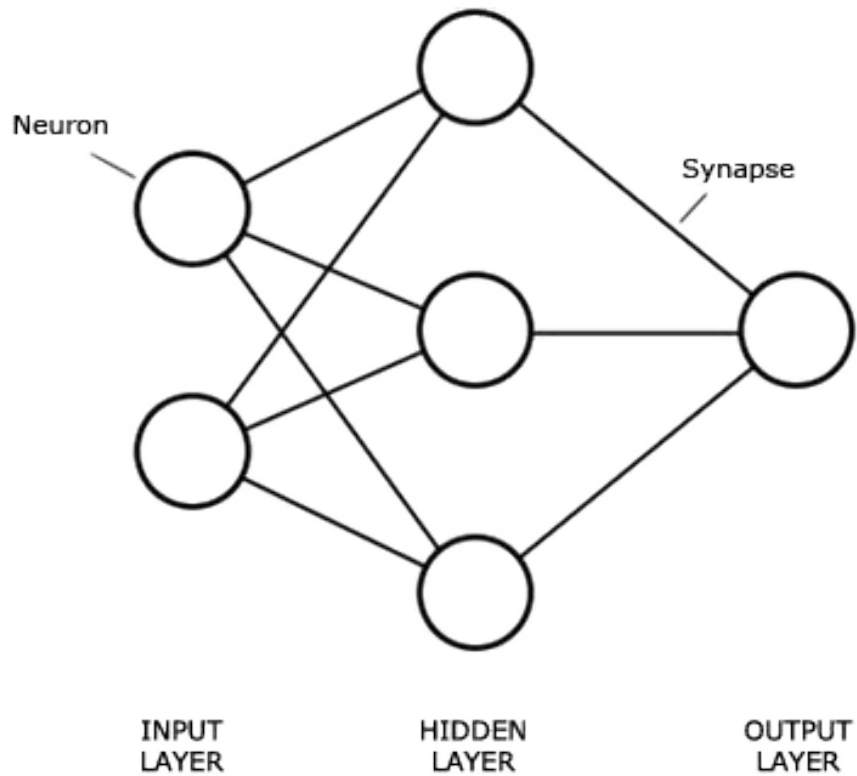
OR

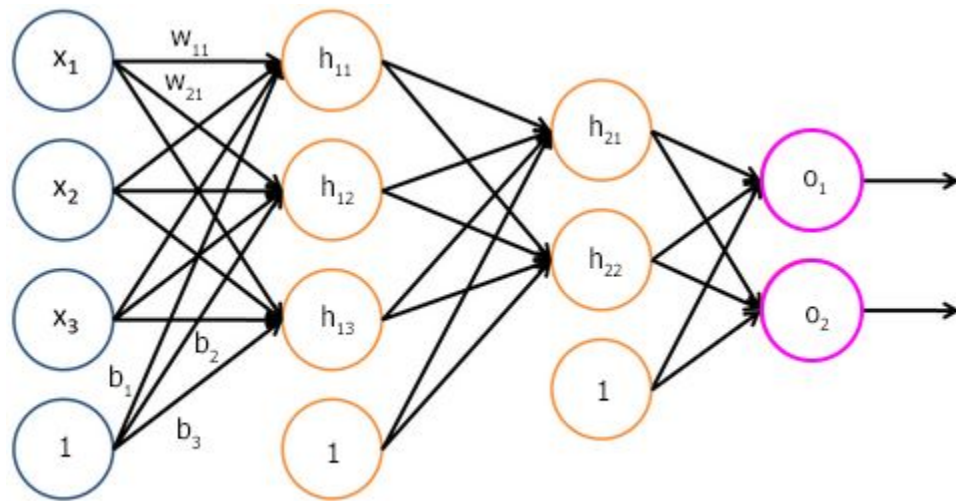
	T	F
T	T	T
F	T	F

XOR

	T	F
T	F	T
F	T	F








$$H_i = a(W_i * I_i + B_i)$$

$$\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & b_1 \\ w_{21} & w_{22} & w_{23} & b_2 \\ w_{31} & w_{32} & w_{33} & b_3 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{bmatrix} = \begin{bmatrix} w_{11} * x_1 + w_{12} * x_2 + w_{13} * x_3 \\ w_{21} * x_1 + w_{22} * x_2 + w_{23} * x_3 \\ w_{31} * x_1 + w_{32} * x_2 + w_{33} * x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$



```
private float output(float[] inputs)
{
    float sum = 0;
    for(int i = 0; i < inputs.length; ++i)
    {
        sum += weights[i] * inputs[i];
    }
    return activationFunction(sum);
}
```



```
private double[] output(double[] input)
{
    Vector input = new Vector(input);

    for (int i = 0; i < weights.Length; ++i)
    {
        input = activationFunction(weights[i] * input + biases[i]);
    }

    return input.ToArray();
}
```

Trainer

×

Train Neural Network

Info

Input Neuron Amount

Hidden Neuron Amount

Output Neuron Amount

Setting

Training Amount

Activation Function

Learning Rate

Activation

Train With Logical Operators

AND

OR

XNOR

XOR

randomize weight

show result

Result

F F

F T

T F

T T

Neural Network

Trainer

×

Train Neural Network

Info

Input Neuron Amount

Hidden Neuron Amount

Output Neuron Amount

Setting

Training Amount

Activation Function

Learning Rate

Activation

Train With Logical Operators

AND

OR

XNOR

XOR

randomize weight

show result

Result

F F

F T

T F

T T

Neural Network



Зворотнє поширення Backpropagation

ERROR = DESIRED OUTPUT - GUESS OUTPUT

$$\begin{bmatrix} E_{h1} \\ E_{h2} \end{bmatrix} = \begin{bmatrix} w_{11} w_{12} w_{13} \\ w_{21} w_{22} w_{23} \end{bmatrix}^T * \begin{bmatrix} E_{o1} \\ E_{o2} \end{bmatrix}$$

$$E_{h_i} = W_i^T * E_{h_{i+1}}$$

```
int error = target - this.output(inputs);
```



```
Vector[] errors = new Vector[Layers.Length - 1];  
// output error, the last position  
errors[errors.Length - 1] = Target - Result;  
// errors  
for (int i = errors.Length - 2; i >= 0; --i)  
{  
    errors[i] = Matrix.Transpose(weights[i + 1]) * errors[i + 1];  
}
```

$$G_i = f'(I_{i+1})$$

$$\Delta \text{WEIGHT} = \text{ERROR} * \text{INPUT} * \text{LEARNING CONSTANT}$$

$$\Delta W_i = \text{LEARNING CONSTANT} * E_i \circ G_i * I_i^T$$

$$W = W + \Delta W$$

$$W_i = W_i + \Delta W_i$$

$$\Delta B_i = \text{LEARNING CONSTANT} * E_i \circ G_i$$

$$B_i = B_i + \Delta B_i$$

```
for(int i = 0; i < weights.length; ++i)
{
    weights[i] += error * inputs[i] * learningRate;
}
```

```
Vector Gradient = derivative(input.Last());
for (int i = weights.Length - 1; i >= 0; --i)
{
    // neuron in input to output layer
    Vector changes = learningRate * errors[i] * Gradient;
    biases[i] += changes;
    weights[i] += changes * Vector.Transpose(input[i]);

    Gradient = derivative(input[i]);
}
```



Правило навчання

Правило навчання — це алгоритм, який змінює параметри нейронної мережі так щоб вхідні аргументи давали необхідний результат

Часто відбуваються зміни:

- ваг
- функції активації
- додавання нейрона в шар
- додавання нового прихованого шару
- зміна зв'язку між нейронами



Розробка нейронної мережі

Collect Data

Organize Data

Encode Data

Neural Network Design

Feed Forward

Recurrent

LSTM

NEAT

Neural Network Training

Supervised

Unsupervised

Reinforcement

Gradient Descent
backpropagation

K-Means

Generative
Adversarial

Q Learning

Brute
Force

Genetic
Algorithm

Deploy Neural Network

Software

Hardware

Video Game



Архітектура нейронної мережі

Perceptron (P)



Feed Forward (FF)



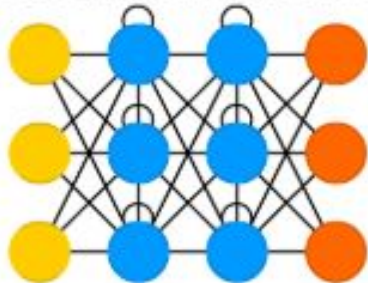
Radial Basis Network (RBF)



Deep Feed Forward (DFF)



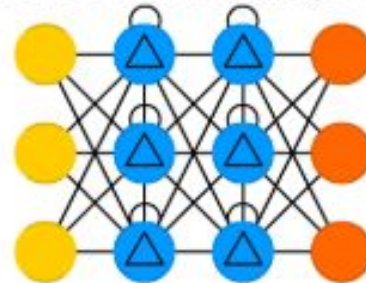
Recurrent Neural Network (RNN)



Long / Short Term Memory (LSTM)

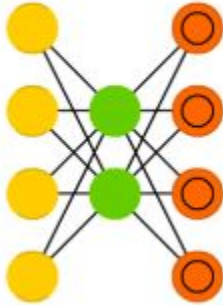


Gated Recurrent Unit (GRU)

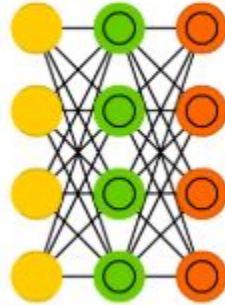


Архітектура нейронної мережі

Auto Encoder (AE)



Variational AE (VAE)



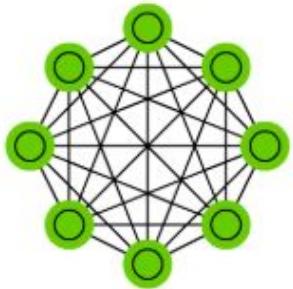
Denoising AE (DAE)



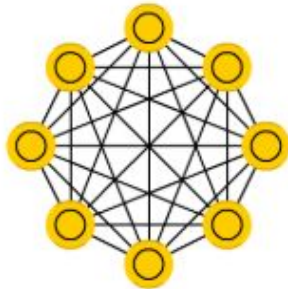
Sparse AE (SAE)



Markov Chain (MC)



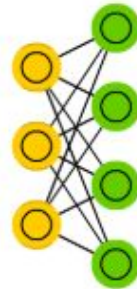
Hopfield Network (HN)



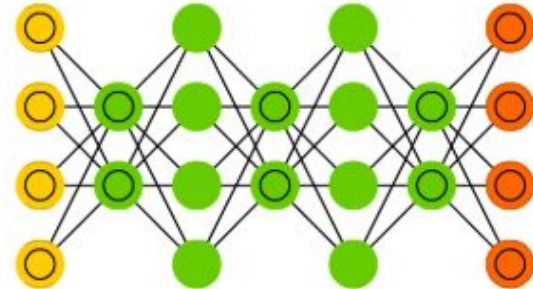
Boltzmann Machine (BM)



Restricted BM (RBM)



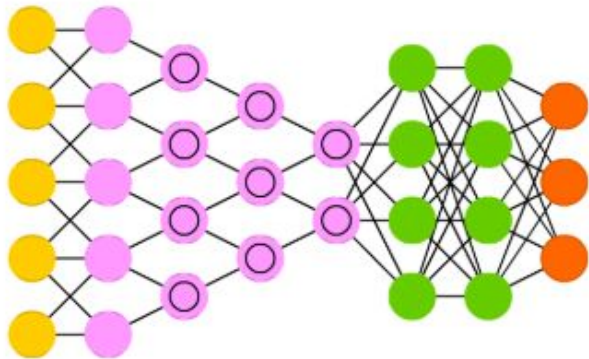
Deep Belief Network (DBN)



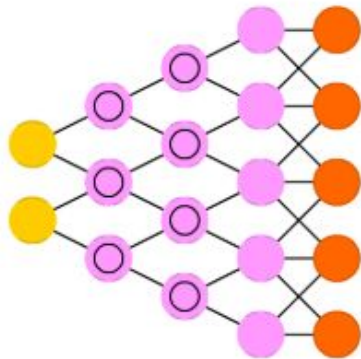


Архітектура нейронної мережі

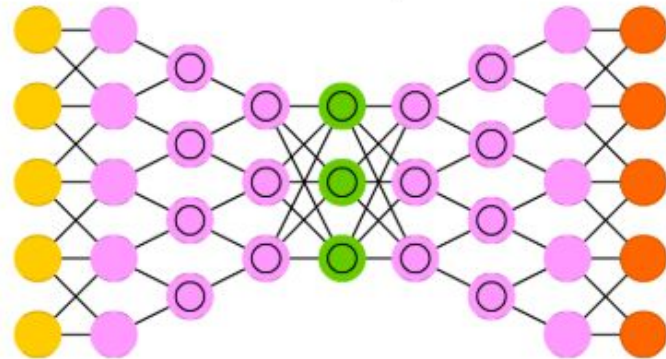
Deep Convolutional Network (DCN)



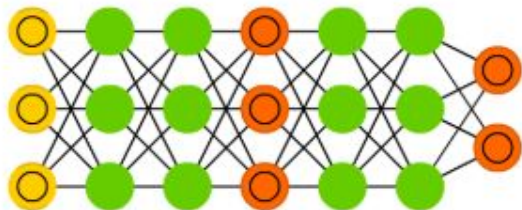
Deconvolutional Network (DN)



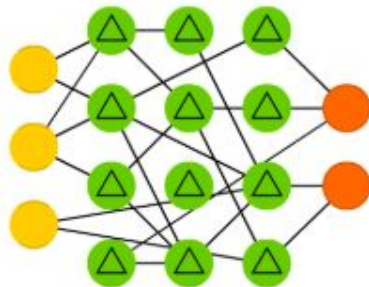
Deep Convolutional Inverse Graphics Network (DCIGN)



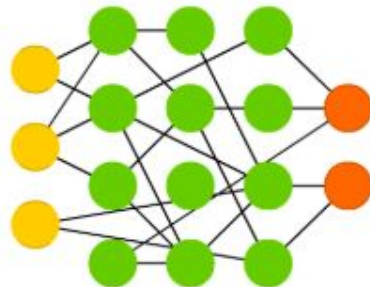
Generative Adversarial Network (GAN)



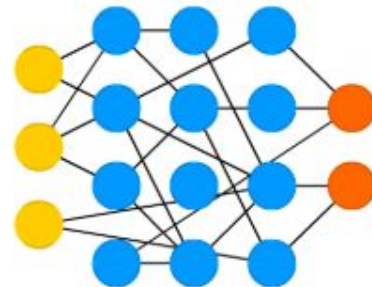
Liquid State Machine (LSM)



Extreme Learning Machine (ELM)

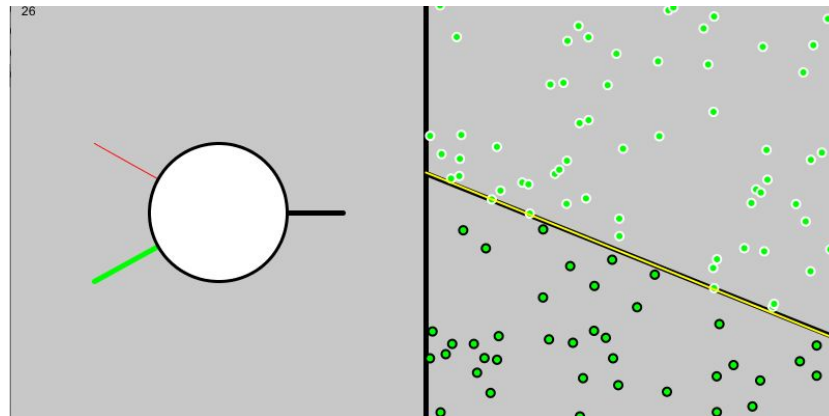
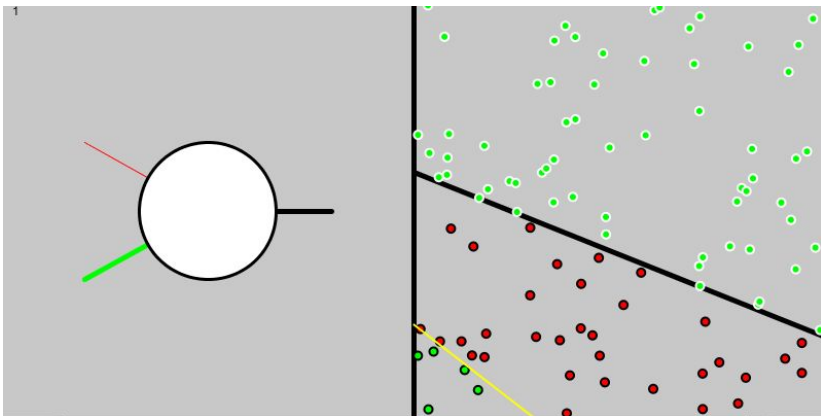


Echo State Network (ESN)





ПРИКЛАДИ



Trainer

×

Train Neural Network

Info

Input Neuron Amount

Hidden Neuron Amount

Output Neuron Amount

Setting

Training Amount

Activation Function

Learning Rate

Activation

Train With Logical Operators

ANDOR


XNORXOR

randomize weight

show result

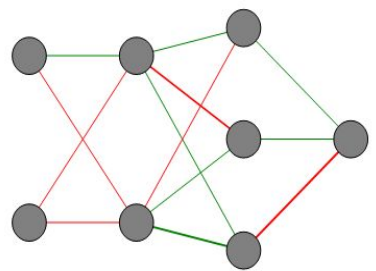
Result

F F F T



T F T T

Neural Network



Trainer

×

Train Neural Network

Info

Input Neuron Amount

Hidden Neuron Amount

Output Neuron Amount

Setting

Training Amount

Activation Function

Learning Rate

Activation

Train With Logical Operators

ANDOR


XNORXOR

randomize weight

show result

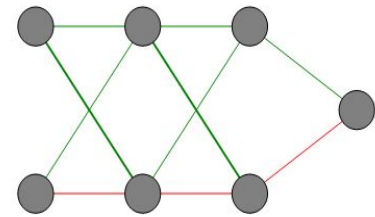
Result

F F F T



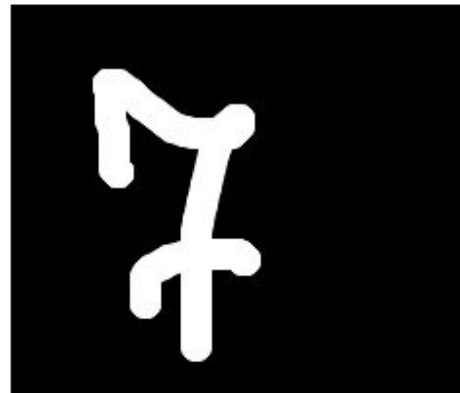
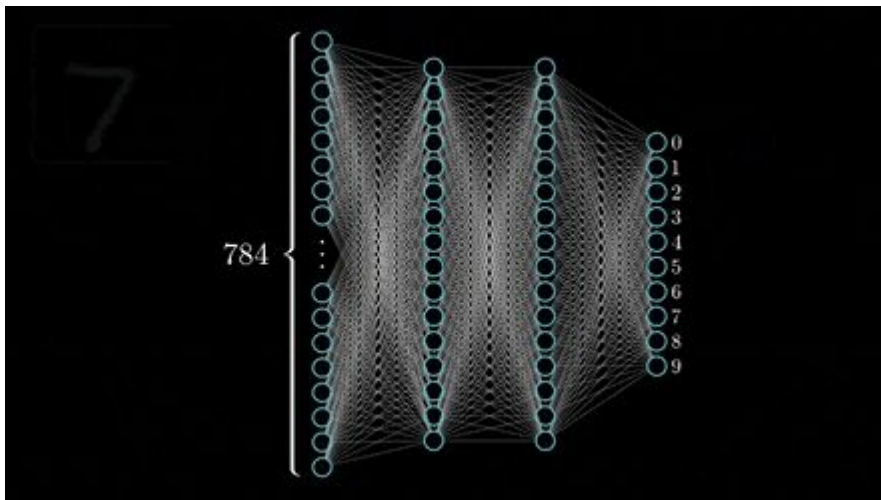
T F T T

Neural Network

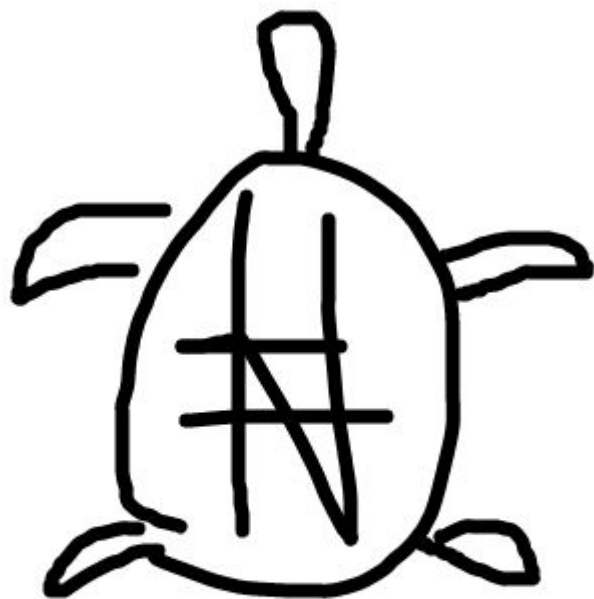




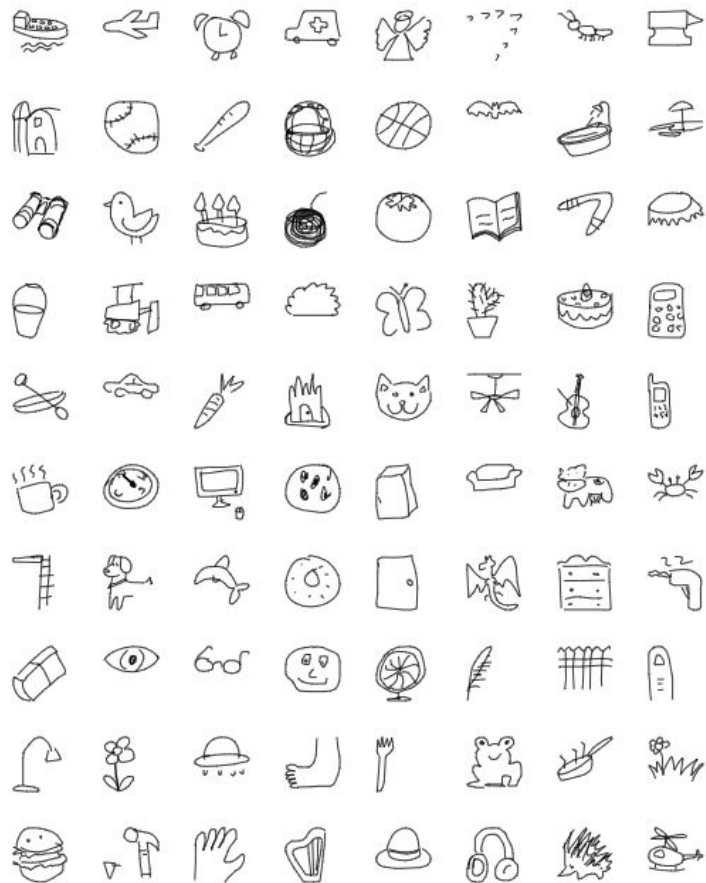
User Guess: 6



User Guess: 7



Oh I know, it's sea turtle!



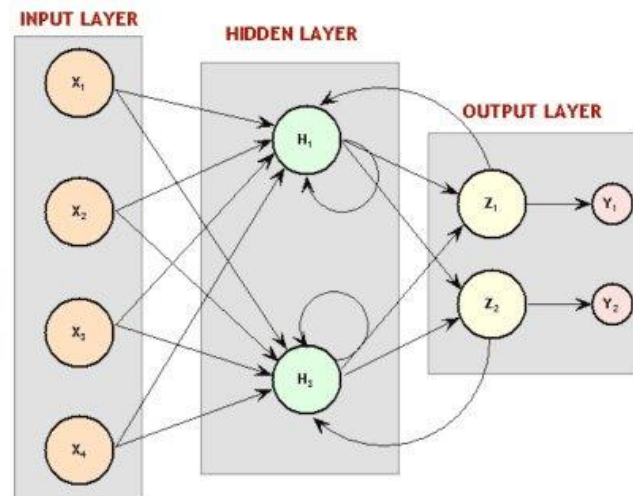
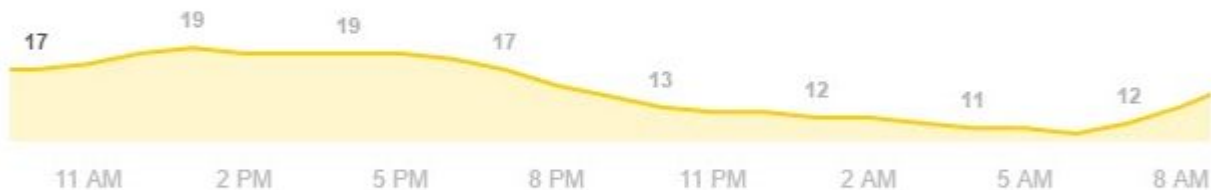


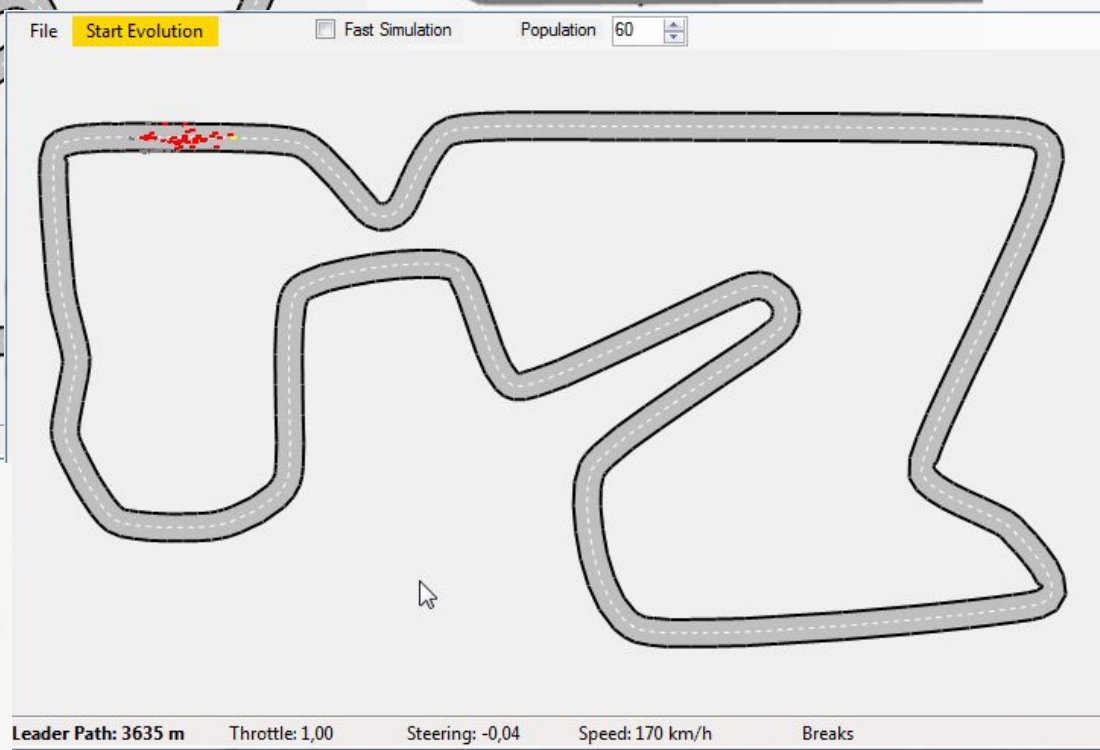
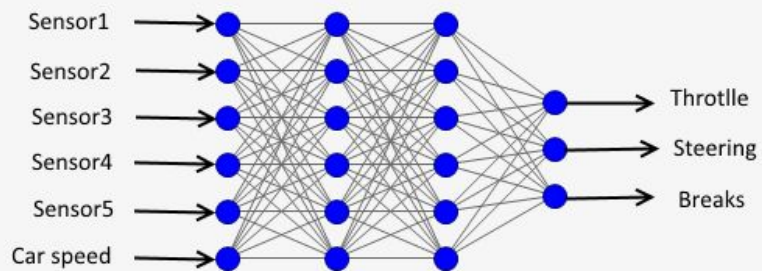
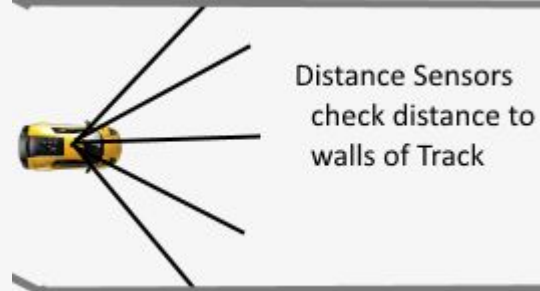
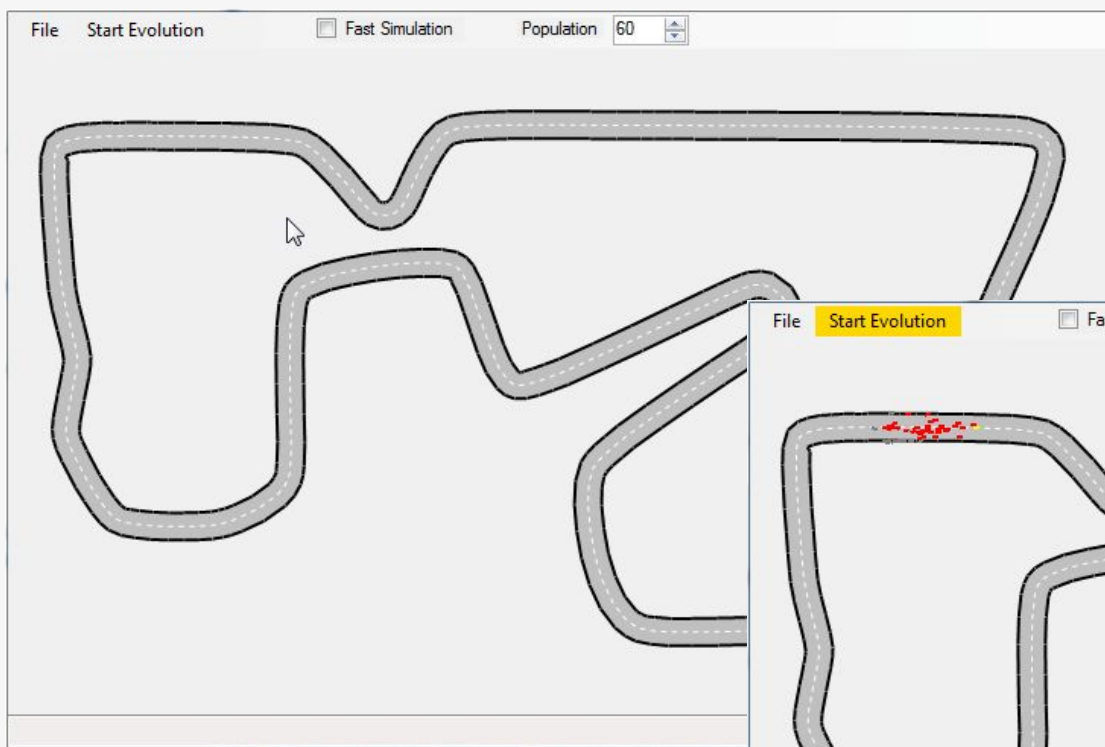


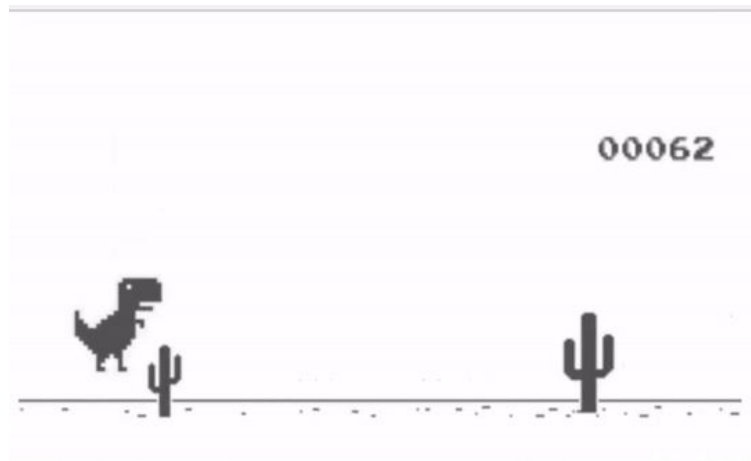
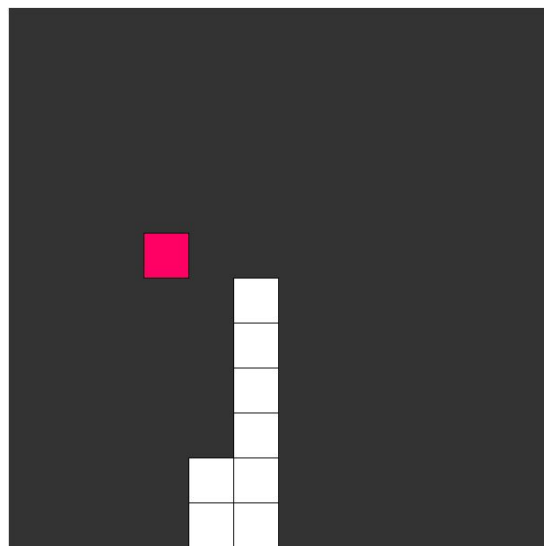
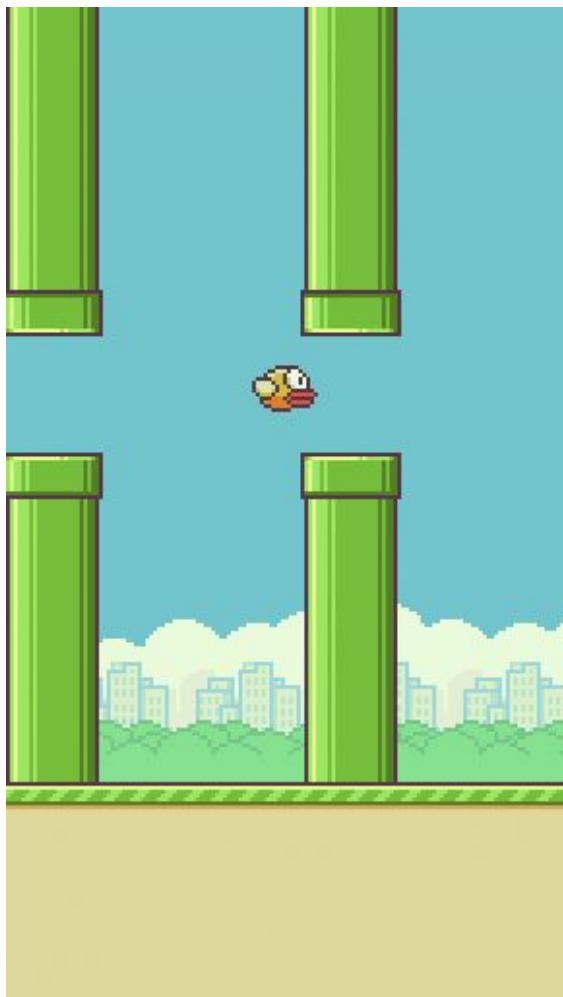
17°C | °F

Precipitation: 0%
Humidity: 57%
Wind: 6 km/h

Temperature Precipitation Wind









Приклади

1. [Point classifier](#)
2. [Logic operator classifier](#)
3. [Number classifier \(mnist\)](#)
4. [Quickdraw](#)
5. [How old do I look?](#)
6. [Deep art](#)
7. [Racing](#)
8. [Flappy bird](#)
9. [Snake](#)
10. [Dino game](#)



Література

1. [Guide for beginners](#)
2. [The mostly complete chart of Neural Networks](#)
3. [Neural Networks \(MMF Article\)](#)
4. [A friendly introduction to Recurrent Neural Networks](#)
5. [Beginner Intro to Neural Networks](#)
6. [Feedforward neural network](#)