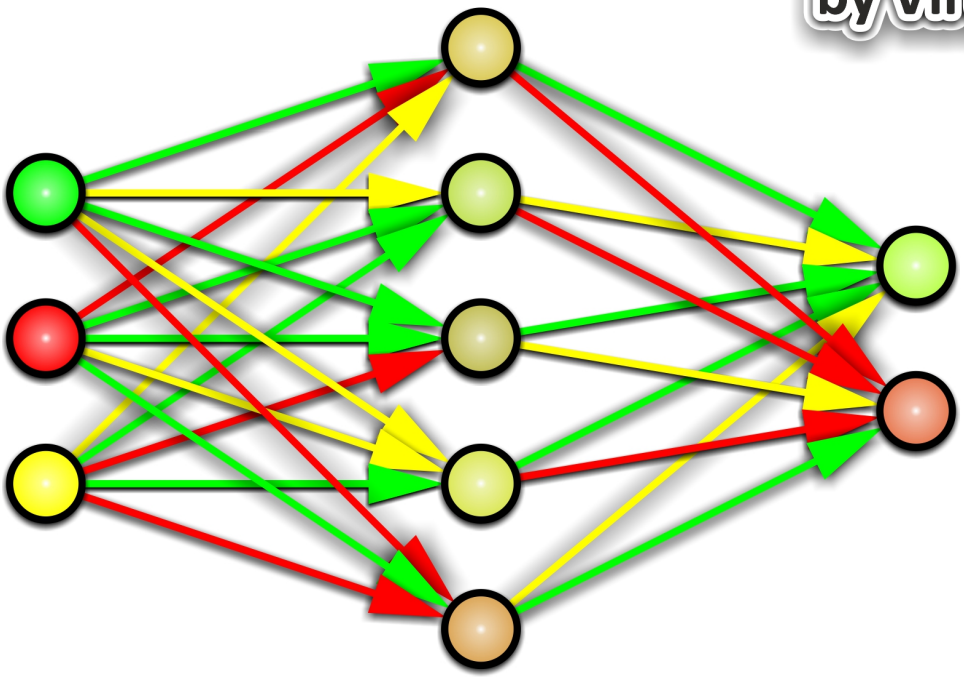
ARTIFICIAL NEURAL NETWORK
PERCEPTRON
by VirtualSUN



Набір скриптів для створення та навчання власної штучної нейронної мережі.

Створення власної штучної нейронної мережі (ШНМ) не є великою проблемою. А от навчити ШНМ виконувати ті чи інші дії та завдання – це вже справжній виклик. Для того, щоб полегшити Вам завдання створення та навчання ШНМ Я написав ряд скриптів, які все зроблять за вас. Єдине, що від вас потребується - це правильно «пояснити» ШНМ, що вона має вивчити.

2018

ЗМІСТ

Загальний опис скриптів.

Perceptron.cs – не MonoBehaviour скрипт штучної нейронної мережі (ШНМ) типу персептрон. Цей скрипт автоматично створить персептрон за вашими параметрами, вирішить поставлене завдання (якщо пройде «навчання»), збереже та завантажить вашу ШНМ.

PerceptronInterface.cs – цей скрипт можна використовувати для полегшення створення, збереження та завантаження ШНМ під час періоду навчання. Це інтерфейс для персептрону у ігровому режимі.

PerceptronVisualization.cs – не MonoBehaviour скрипт для візуалізації персептрону. Показує всі шари, нейрони та зв’язки між нейронуми ШНМ. Використовується скриптом PerceptronInterface.cs, але при потребі можна використовувати і у власних скриптах.

PerceptronLernByBackPropagation.cs – не MonoBehaviour скрипт для навчання персептрону методом зворотного поширення помилки. Вам достатню ввести вибірку завдань з відповідями або створити «вчителя», і ШНМ почне своє навчання. А за допомогою гнучких налаштувань Ви можете швидно і легко завершити навчання ШНМ.

PerceptronBackPropagationInterface.cs – це скрипт інтерфейса для навчання персептрону методом поширення помилки у ігровому режимі.

PerceptronLernByRandomGeneration.cs – не MonoBehaviour скрипт для навчання персептрону методом випадкової генерації певної кількості «клонів» об’єкту який потрібно навчити. Усі створені «клони» отримують видозмінений випадковим чином персептрон від об’єкту навчання на першій генерації, а на всіх наступних від кращого з попередньої генерації. Велика кількість гнучких налаштувань навчання дає можливість отримати само навчений персептрон.

PerceptronRandomGenerationInterface.cs – це скрипт інтерфейса для навчання персептрону методом випадкової генерації у ігровому режимі.

InterfaceGUI.cs – не MonoBehaviour скрипт для скриптів інтерфейсів.

Formulas.cs – не MonoBehaviour скрипт для полегшення вирішення циклічних чи подібних завдань.

Детальний опис основних скриптів.

public class PerceptronLernByBackPropagation – не MonoBehaviour скрипт штучної нейронної мережі (ШНМ) типу персептрон.

Основні змінні скрипту Perceptron.cs	
public float AFS	Розмір функції активації.
public bool B	Якщо «вірно», то створюється додатковий нейрон зміщення в кожному шарі, крім вихідного шару. Цей нейрон завжди дорівнює одиниці. Його вагові зв'язки впливають на всі нейрони наступного шару, крім нейрону зміщення.
public bool AFWM	Якщо «вірно», то всі нейрони сприймають значення від -1 до 1. Це стосується також вхідних та вихідних нейронів. Якщо «невірно» - то від 0 до 1.
public float[] Input	Вхідний шар ШНМ . Розмір масиву вказує кількість вхідних значень, з нейроном зміщення (якщо B == true, останній нейрон завжди = 1).
public int[] NIHL	Кількість нейронів у прихованих шарах, з нейроном зміщення. Розмір масиву вказує кількість прихованих шарів, з нейроном зміщення (якщо B == true, останній нейрон кожного шару завжди = 1).
public float[] Output	Вихідний шар ШНМ . Розмір масиву вказує кількість вихідних значень. Ніколи не містить нейрон зміщення.
public float[][] Neuron	Значення нейронів. Перша частина масиву відповідає шару персептрону. Друга частина масиву - це число нейрону в шарі.
public float[][][] NeuronWeight	Значення зв'язків між нейронуми. Перша частина масиву відповідає шарові ШНМ. Друга частина масиву - це число нейрону наступного шару. Третя частина масиву - це число нейрону поточного шару.

Команда скрипту	**Змінні команди**	**Пояснення**
public void CreatePerceptron		Створення персептрону за допомогою параметрів.
	float ActivationFunctionScale	Розмір функції активації.
	bool Bias	Якщо «вірно», то створюється додатковий нейрон зміщення в кожному шарі, крім вихідного шару. Цей нейрон завжди дорівнює одиниці. Його вагові зв'язки впливають на всі нейрони наступного шару.
	bool ActivationFunctionWithMinus	Якщо «вірно», то всі нейрони сприймають значення від -1 до 1. Це стосується також вхідних та вихідних нейронів. Якщо «невірно» - то від 0 до 1.
	int NumberOfInputs	Кількість вхідних нейронів, без нейрону зміщення.
	int[] NumberOfNeuronInHidenLayers	Масив який вказує кількість нейронів (без нейрону зміщення) у кожному прихованому шарі ШНМ. Розмір масиву вказує на кількість прихованих шарів.
	int NumbersOfOutputs	Кількість вихідних нейронів.
public void Load		Завантаження персептрону з файлу.
	string PerceptronFile	Ім'я файлу для завантаження.
public void PerceptronSolution		Рішення персептрону.
private float Sumator		Сума всіх значень нейронів помножених на їх вагу.
	float[] Neuron	Нейрони певного шару.
	float[] NeuronWeight	Ваги того ж шару, що і нейрони.
private float ActivationFunction		Функція активації нейронів.
	float Sum	Сума всіх значень нейронів помножених на їх вагу.
private void CreatingNeurons		Створення нейронів та їх зв'язків між собою.
	StreamReader SR	Вказаний потік з файлу завантаження. Використовуйте "null", якщо файл не використовується.
public void Save		Збереження параметрів персептрону у файл.
	string PerceptronFile	Ім'я файлу для завантаження.

public class PerceptronLernByBackPropagation – не MonoBehaviour скрипт для навчання персептрону методом зворотного поширення помилки.

Основні змінні скрипту PerceptronLernByBackPropagation.cs	
public int LearningSpeed	Швидкість навчання. Кількість кроків навчання за один кадр ігрового режиму.
public int LearnIteration	Лічильник кількості кроків навчання.
public float LearningRate	Сила зміни вагових зв'язків при навчанні. Впливає на швидкість та якість навчання.
public float DesiredMaxError	Вказує якою має бути максимальна різниця між вірною відповіддю та відповіддю ШНМ.
public float MaxError	Максимальна різницю між вірною відповіддю та відповіддю ШНМ.
public bool Learned	Якщо різниця між вірною відповіддю та відповіддю ШНМ менша за DesiredMaxError, то персептрон рахується як навчений.
public bool ShuffleSamples	Якщо «вірно», то зразки завдань та відповідей будуть перемішуватися при навчанні.

Команда скрипту	**Змінні команди**	**Пояснення**
public void Learn		Перший варіант використання команди. Навчання персептрону методом зворотного поширення помилки з вмістом певної кількості завдань з відповідями. Може використовувати ShuffleSamples.
	Perceptron PCT	Персептрон який треба вчити.
	float[][] Task	Масив завдань. Перша частина масиву відповідає за номер завдання. Друга частина масиву – номер вхідного нейрону.
	float[][] Answer	Масив відповідей. Перша частина масиву відповідає за номер відповіді. Друга частина масиву – номер вихідного нейрону.
public void Learn		Другий варіант використання команди. Навчання персептрону методом зворотного поширення помилки з вмістом одного завдання з відповіддю. Не використовує LearningSpeed, DesiredMaxError та ShuffleSamples. Рекомендується використовувати DesiredMaxError = 0.
	Perceptron PCT	Персептрон який треба вчити.
	float[] Task	Масив одного завдання для кожного вхідного нейрону персептрону.
	float[] Answer	Масив однієї відповіді для кожного вихідного нейрону персептрону.
public void Learn		Третій варіант використання команди. Навчання персептрону методом зворотного поширення помилки з вмістом однієї відповідді. Використовувати якщо завдання напряму вводиться у вхідний шар персептрону. Не використовує LearningSpeed, DesiredMaxError та ShuffleSamples. Рекомендується використовувати DesiredMaxError = 0.
	Perceptron PCT	Персептрон який треба вчити.
	float[] Answer	Масив однієї відповіді для кожного вихідного нейрону персептрону.
private void ShufflingSamples		Перемішування зразків завдань та відповідей.
	float[][] Task	Масив завдань. Перша частина масиву відповідає за номер завдання. Друга частина масиву – номер вхідного нейрону.
	float[][] Answer	Масив відповідей. Перша частина масиву відповідає за номер відповіді. Друга частина масиву – номер вихідного нейрону.
public void ModificateStartWeights		Модифікація вагових зв'язків персептрону перед навчанням.
	Perceptron PCT	Персептрон який треба вчити.
	bool MSW	Якщо «вірно», то вагові зв'язки модифікуються по спеціальному алгоритму. Якщо «невірно», то всі вігові зв'язки генеруються випадково в діапазоні від -0,5 до 0,5.
private float DX		Пошук похідної від значення певного нейрону.
	float X	Значення нейрону.
	float ActivationFunctionScale	Розмір функції активації персептрону.
	bool ActivationFunctionWithMinus	Чи використовує персептрон мінусове значення.

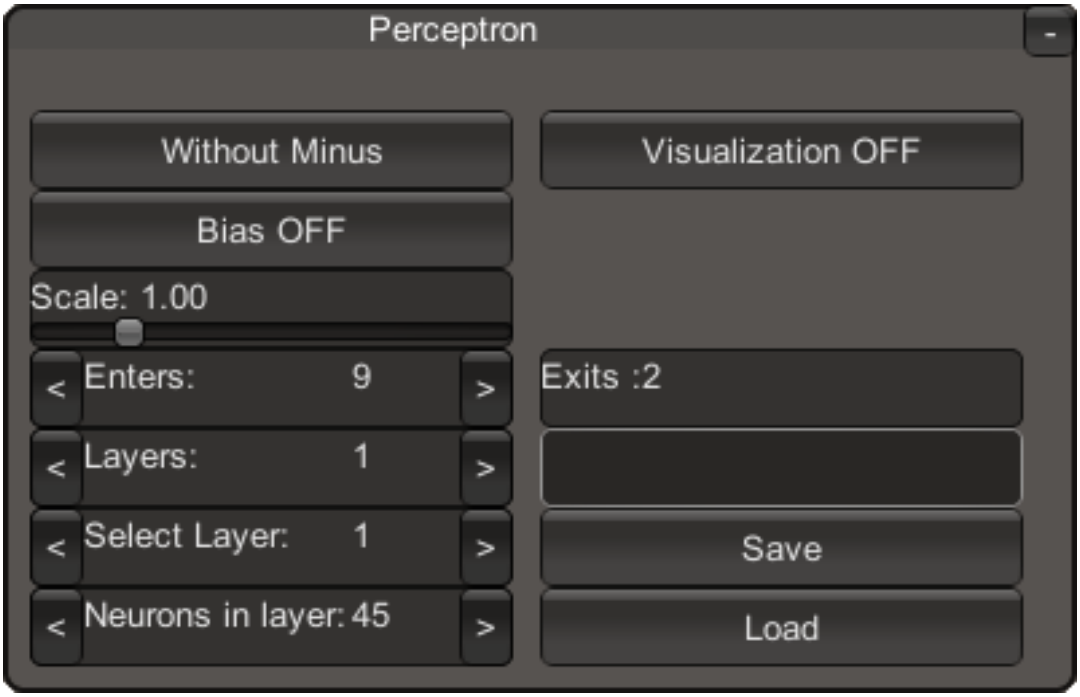
public class PerceptronLernByRandomGeneration – не MonoBehaviour скрипт для навчання персептрону методом випадкової генерації певної кількості «клонів» об’єкту який потрібно навчити.

Основні змінні скрипту PerceptronLernByRandomGeneration.cs	
public int AmountOfChildren	Кількість «дітей» у кожному поколінні.
public int BestGeneration	Краще покоління на даний момент.
public int Generation	Загальна кількість поколінь.
public int ChildrenInGeneration	Кількість «дітей» у останньому поколінні.
public float BestLongevity	Краще «довголіття» на даний момент.
public float ChildrenDifference	Різниця вагових зв’язків між поколіннями.
public float ChildrenDifferenceAfterEffects	Різниця вагових зв’язків з коефіцієнтом впливу між поколіннями.
public bool ChildrenGradient	Якщо «вірно», то лінійно згладжує різницю вагових зв’язків між «дітьми» у поколінні.
public float GenerationEffect	Зменшувальний коефіцієнт впливу на різницю вагових зв’язків між поколіннями. Впливає при умові, що не дорівнює нулю та теперішнє покоління було кращим за попереднє.
public float GenerationSplashEffect	Збільшувальний коефіцієнт впливу на різницю вагових зв’язків між поколіннями. Впливає при умові, що не дорівнює нулю та теперішнє покоління було гіршим за попереднє.
public float Chance	Шанс вибрати поточне гірше покоління. Він змінюється за рахунок параметру public float ChanceCoefficient з кожним новим поколінням
public float ChanceCoefficient	Коефіцієнт впливу на шанс випадковості вибору теперішнього гіршого покоління. Впливає при умові, що не дорівнює нулю.

Команда скрипту	**Змінні команди**	**Пояснення**
public void StudentData		Збір даних для навчання.
	GameObject Student	Головний ігровий об’єкт (GameObject) який має вчитися.
	Object HereIsANN	Скрипт який містить персептрон.
	string PerceptronName	Ім’я змінної (Perceptron) яким названо персептрон у скрипті який містить персептрон (HereIsANN).
	Object StudentControls	Скрипт керування головного ігрового об’єкту.
	string StudentCrash	Ім’я змінної (bool) яким названо причина «аварії» ігрового об’єкту у скрипті керування (StudentControls).
	string StudentLife	Ім’я змінної (float) яким названо «довголіття» ігрового об’єкта у скрипті керування (StudentControls).
public void Learn		Навчання персептрону методом випадкової генерації.
	Perceptron PCT	Персептрон який треба вчити.
public void StopLearn		Негайна зупинка навчання з передачею інформації вагових зв’язків кращого персептрону з кращого покоління на персептрон який навчається.
	Perceptron PCT	Персептрон який треба вчити.
public void Reset		Скинути інформацію про навчання.

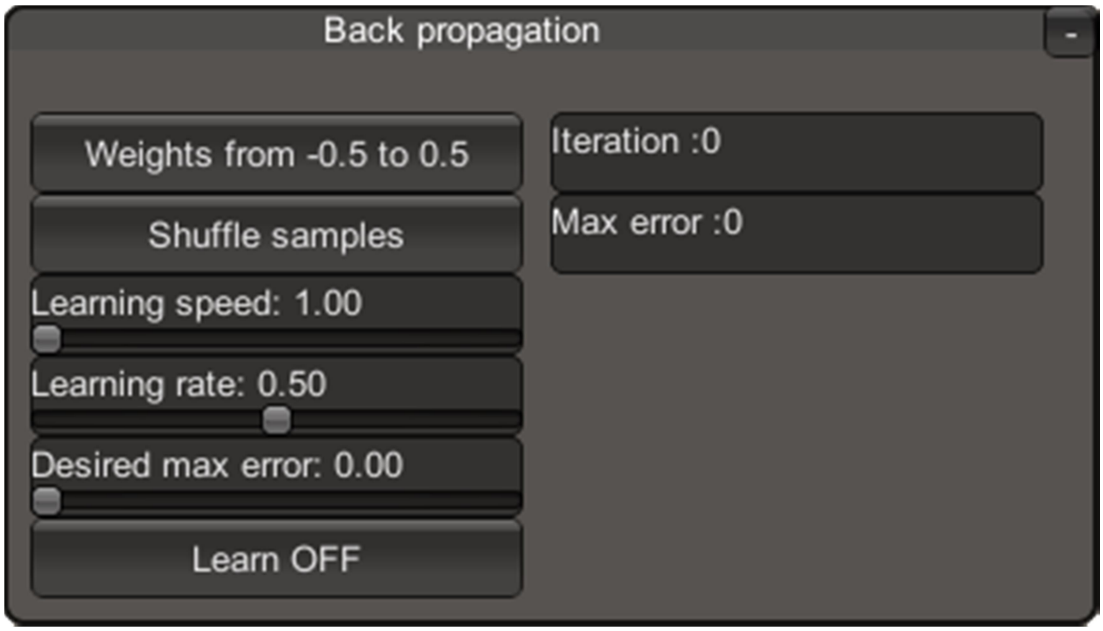
Опис допоміжних інтерфейсів.

PerceptronInterface.cs - цей скрипт можна використовувати для полегшення створення, збереження та завантаження ШНМ під час періоду навчання. Це інтерфейс для персептрону у ігровому режимі. Керує параметрами скрипту **Perceptron.cs**.



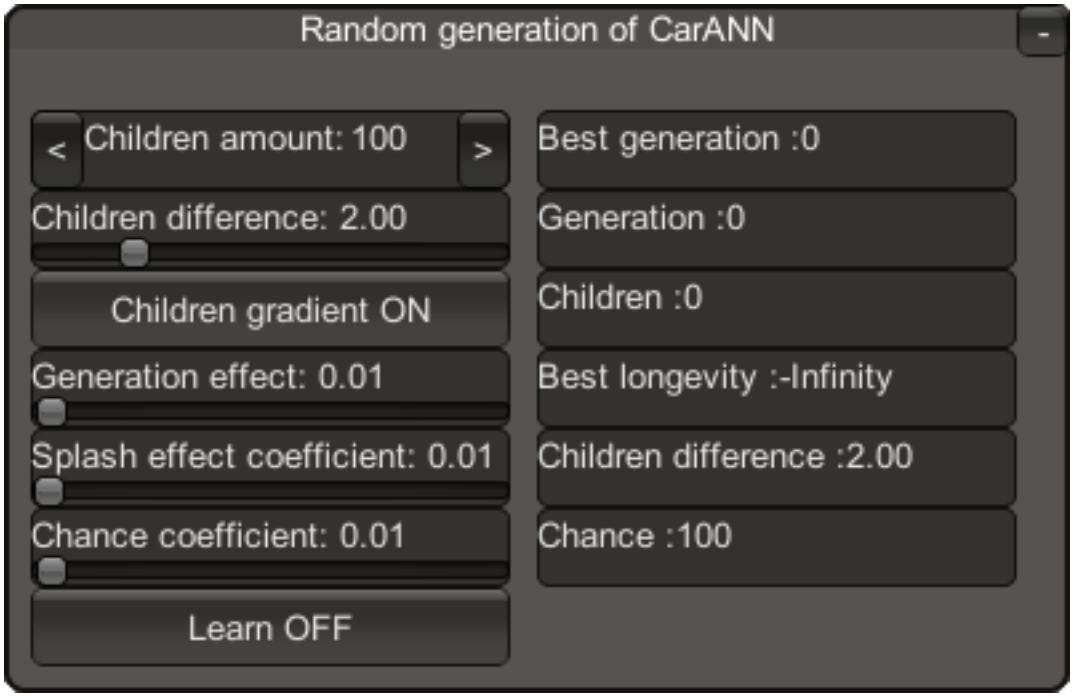
Without Minus With Minus	Якщо «With Minus», то всі нейрони сприймають значення від -1 до 1. Це стосується також вхідних та вихідних нейронів. Якщо «Without Minus» - то від 0 до 1. Керує параметром public bool AFWM.
Bias OFF Bias ON	Якщо «Bias ON», то створюється додатковий нейрон зміщення в кожному шарі, крім вихідного шару. Цей нейрон завжди дорівнює одиниці. Його вагові зв'язки впливають на всі нейрони наступного шару, крім нейрону зміщення. Якщо «Bias OFF», то нейрону зміщення не буде. Керує параметром public bool B.
Scale	Розмір функції активації. Керує параметром public float AFS.
Enters	Вказує кількість нейронів у вхідному шарі, без врахування нейрону зміщення. Впливає на масиви float[] Input, public float[][] Neuron та public float[][][] NeuronWeight.
Layers	Вказує кількість прихованих шарів. Впливає на масиви public int[] NIHL, public float[][] Neuron та public float[][][] NeuronWeight.
Select Layer	Вибір прихованого шару який потребує змін.
Neurons in layer	Вказує кількість нейронів у вибраному прихованому шарі. Впливає на масиви public int[] NIHL, public float[][] Neuron та public float[][][] NeuronWeight.
Visualization OFF Visualization ON	Якщо «Visualization ON» - демонструє вигляд персептрону з заданими параметрами. Якщо «Visualization OFF» - не демонструє. Використовує скрипт **PerceptronVisualization.cs**.
Exits	Показує кількість вихідних нейронів. Їх кількість треба вказувати при створенні персептрону.
GUI.TextField	Ім'я файлу для збереження або завантаження персептрону.
Save	Зберігає параметри та всі вагові зв'язки персептрону у файл, якщо вказано ім'я файлу. Керує командою public void Save.
Load	Завантажує параметри та всі вагові зв'язки персептрону у файл, якщо вказано ім'я файлу. Керує командою public void Load.

PerceptronBackPropagationInterface.cs – це скрипт інтерфейса для навчання персептрону методом поширення помилки у ігровому режимі. Керує параметрами скрипту **PerceptronLernByBackPropagation.cs**.



Weights from -0.5 to 0.5 Mod Weights	Модифікація вагових зв'язків персептрону перед навчанням. Якщо «Mod Weights», то вагові зв'язки модифікуються по спеціальному алгоритму. Якщо «Weights from -0.5 to 0.5», то всі вігові зв'язки генеруються випадково в діапазоні від -0,5 до 0,5. Керує командою public void ModificateStartWeights.
Samples one by one Shuffle samples	Перемішування зразків завдань та відповідей. Якщо «Shuffle samples», то зразки завдань та відповідей будуть перемішуватися при навчанні. Якщо «Samples one by one» - зразки будуть йти у заданому порядку. Керує параметром public bool ShuffleSamples.
Learning speed	Швидкість навчання. Вказує кількість кроків навчання за один кадр ігрового режиму. Керує параметром public int LearningSpeed.
Learning rate	Вказує силу зміни вагових зв'язків при навчанні. Впливає на швидкість та якість навчання. Керує параметром public float LearningRate.
Desired max error	Вказує якою має бути максимальна різниця між вірною відповіддю та відповіддю ШНМ серед усіх зразків завдань та відповідей. Керує параметром public float DesiredMaxError.
Learn OFF Learn ON	Якщо «Learn ON», то починає навчання персептрону методом зворотного поширення помилки. Керує першим варіантом команди public void Learn.
Iteration	Показує кількість кроків навчання. Отримує данні з параметру public int LearnIteration.
Max error	Показує максимальну помилку відповіді вихідних нейронів та зразку відповіді серед усіх зразків відповідей. Отримує данні з параметру public float MaxError.

PerceptronRandomGenerationInterface.cs - це скрипт інтерфейса для навчання персептрону методом випадкової генерації у ігровому режимі. Керує параметрами скрипту **PerceptronLernByRandomGeneration.cs**.



Children amount	Кількість «дітей» у кожному поколінні. Керує параметром public int AmountOfChildren.
Children difference	Різниця вагових зв'язків між поколіннями. Керує параметром public float ChildrenDifference.
Children gradient OFF Children gradient ON	Якщо «Children gradient ON», то лінійно згладжує різницю вагових зв'язків між «дітьми» у поколінні.
Generation effect	Зменшувальний коефіцієнт впливу на різницю вагових зв'язків між поколіннями. Впливає при умові, що не дорівнює нулю та теперішнє покоління було кращим за попереднє. Керує параметром public float GenerationEffect.
Splash effect coefficient	Збільшувальний коефіцієнт впливу на різницю вагових зв'язків між поколіннями. Впливає при умові, що не дорівнює нулю та теперішнє покоління було гіршим за попереднє. Керує параметром public float GenerationSplashEffect.
Chance coefficient	Коефіцієнт впливу на шанс випадковості вибору теперішнього гіршого покоління. Впливає при умові, що не дорівнює. Керує параметром public float ChanceCoefficient.
Learn OFF Learn ON	Якщо «Learn ON», то починає навчання персептрону методом випадкової генерації. Керує командою public void Learn. Якщо «Learn OFF» і проводилось навчання, то зупиняє навчання. Використовує команду public void StopLearn.
Best Generation	Показує номер кращого покоління на даний момент. Отримує данні з параметра public int BestGeneration.
Generation	Показує яке покоління на даний момент. Отримує данні з параметру public int Generation.
Children	Показує кількість «дітей» у поточному поколінні. Отримує данні з параметру public int ChildrenInGeneration.
Best longevity	Показує краще «довголіття» на даний момент. Отримує данні з параметру public float BestLongevity.
Children difference	Різниця вагових зв'язків між поколіннями з коефіцієнтами впливу. Отримує данні з параметру public float ChildrenDifferenceAfterEffects.
Chance	Показує шанс вибору поточне гірше покоління. Він змінюється з кожним новим поколінням. Отримує данні з параметру public float Chance.

Як користуватися.

Для початку треба створити певне завдання яка має виконувати ШНМ. Треба пам’ятати, що ШНМ має отримувати певні вхідні данні, і треба їх правильно підготувати. Персептрон сприймає вхідні данні від 0 до 1, або від -1 до 1 (див. ст. 1 «public bool AFWM»). Також треба визначитися в кількості вхідних даних.

Те саме стосується і вихідних даних. Отже їх треба вірно конвертувати для коректної відповіді на поставлене завдання.

Кількість прихованих шарів та нейронів в них - вже залежить від Вас. Іноді бувають завдання де не має змісту використовувати прихований шар, а іноді навпаки - треба більше шарів та більше нейронів. В будь якому разі, їх кількість по різному впливає на якість та швидкість навчання ШНМ.

Далі треба вирішити яким методом навчати ШНМ. Якщо вже є підготовлені зразки завдань та відповідей, або завдання може вирішувати якийсь сторонній «вчитель», то можна використовувати метод зворотного поширення помилки. Якщо ж не має підготовлених зразків чи «вчителя» - можна використати метод випадкової генерації.

Для покращення розуміння, використання даного пересептрона, мною підготовлено декілька уроків.

Завдання «Місія - не здохнути».

Розглянемо вже підготовлене завдання: Є «корова» яка з часом хоче їсти. Є «їжа» яку «корова» може з’їсти. «Корова помре» якщо не буде «їсти», або «з’їсть» забагато. «Корова» може «їсти» тільки передом, інакше «помре». Треба навчити «корову» правильно та вчасно «їсти».

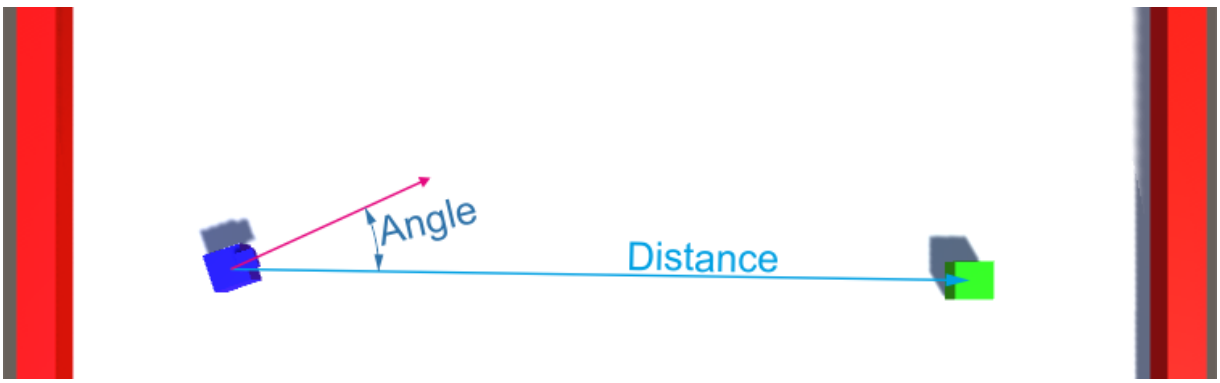
У папці «Tutorial» є вже заготовлені скрипти та сцена для завдання.

«Корова» отримує три значення:

1. Дистанція до «їжі». Діагональ рівня має біля 41.
2. Кут повороту зі знаком відносно переду «корови» до «їжі». Від -180 до 180.
3. «Ситість корови» яка зменшується з часом. 50 – максимальне значення для «виживання».

«Корова» керується двома значення:

1. Повернути до «їжі». Від -1 до 1.
2. Рухатися до «їжі». Від -1 до 1.



Також у «корови» є додаткові значення для отримання вище перерахованих.

Ось скрипт «корови» **TutorialCowControl.cs:**

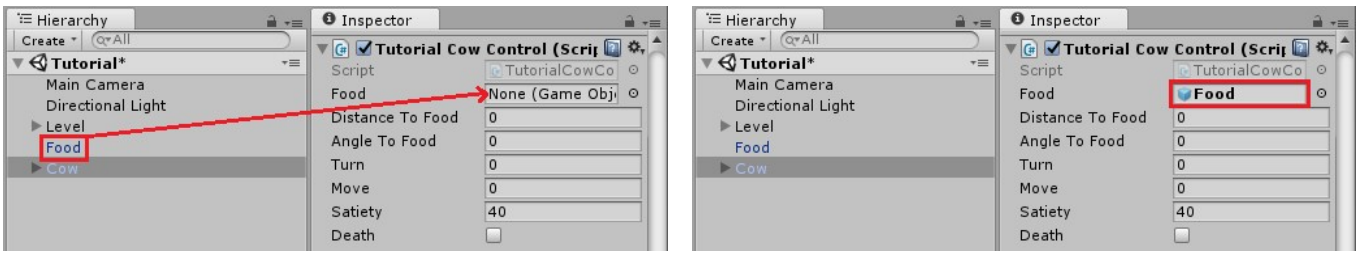
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TutorialCowControl : MonoBehaviour
{
    public GameObject Food;             //Food's GameObject
    public float DistanceToFood = 0;    //Distance to food
    public float AngleToFood = 0;       //Angle to food

    public float Turn = 0;              //Turn of cow
    public float Move = 0;              //Move of cow
    public float Satiety = 40;          //Satiety of cow
    public bool Death = false;          //If true - cow will die (reset position)

    void Update()
    {
        //Max & min turn
        if (Turn > 1)
            Turn = 1;
        else if (Turn < -1)
            Turn = -1;

        //Max & min move
        if (Move > 1)
            Move = 1;
        else if (Move < -1F)
            Move = -1F;

        //Cow reset
        if (Death)
        {
            Satiety = 40;
            transform.position = new Vector3(0, 0.5F, 0);
            transform.eulerAngles = new Vector3(0, transform.eulerAngles.y, 0);
            Death = false;
        }

        //Controls of cow
        transform.Rotate(0, Turn * 10F, 0);
        transform.Translate(0, 0, Move / 10F);

        //Food info
        DistanceToFood = Vector3.Distance(transform.position, Food.transform.position);
        AngleToFood = Vector3.Angle(transform.forward, Food.transform.position - transform.position)
* Mathf.Sign(transform.InverseTransformPoint(Food.transform.position).x);

        //The satiety of the cow decreases with time
        Satiety -= Time.deltaTime;
        if (Satiety < 0 || Satiety > 50)
            Death = true;
    }
}
```

«Їжа» впливає на «корову» при дотику. Якщо «корова» вірно «з'їсть їжу» (кут повороту не перевищує 5 градусів), то збільшує «ситість» (+15), а «їжа» міняє місце положення. Інакше «помре».

Ось скрипт «їжі» **TutorialFood.cs:**

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TutorialFood : MonoBehaviour
{
    private bool Moving = false;

    void Start ()
    {
        MoveFood();         //Move food
    }

    void Update()
    {
        if (Moving)
            Moving = false;
    }

    void OnCollisionEnter(Collision col)
    {
        TutorialCowControl TPC = col.gameObject.GetComponent<TutorialCowControl>();
        if (TPC != null && !Moving)        {
            //The cow must eat at a certain angle
            if (Mathf.Abs(TPC.AngleToFood) > 5)
                TPC.Death = true;
            else
            {
                TPC.Satiety += 15;
                MoveFood();
            }
        }
    }

    void MoveFood()          //Move food
    {
        //Random position
        transform.position = new Vector3(Random.Range(-14F, 14F), 0.5F, Random.Range(-14F, 14F));
        Moving = true;
    }
}
```

Не забудьте вказати «корові» де «їжа»:



Керування «коровою» є, «їжа» є. А тепер треба створити «корові мозок».

Як створити персептрон.

Щоб створити пересептрон треба створити MonoBehaviour скрипт та вписати в нього:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class NameOfScript : MonoBehaviour
{
    public Perceptron PerceptronName = new Perceptron();
    . . .
    void Start()
    {
        PerceptronName.CreatePerceptron(1, false, true, 3, null, 2);
        . . .
    }

    void Update()
    {
        . . .
        PerceptronName.Input[0] = . . . ;
        PerceptronName.Input[1] = . . . ;
        . . .
        PerceptronName.Solution();
        . . .
        . . . = PerceptronName.Output[0];
        . . . = PerceptronName.Output[1];
        . . .
    }
    . . .
}
```

Пояснення дивіться на сторінці 4.

Перед PerceptronName.Solution() треба ввести конвертовані вхідні данні у вхідні нейрони.

Після PerceptronName.Solution() – вивести вихідні данні вихідних нейронів та конвертувати їх.

Для завдання «Місія - не здохнути» створимо мозок «корови» у файлі під назвою **TutorialCowPerceptron.cs**. Його додайте до об’єкту «корови».

У «корови» є три основних значення які вона має сприймати та два значення керування (див. ст. 10). Отже потрібен персептрон з трьома вхідними та двома вихідними нейронами. Щоб менше гратися з конвертаціїю значень створимо персептрон «з мінусом» (див. ст. 4, bool ActivationFunctionWithMinus). А вже кількість прихованих шарів та нейронів в них буде залежить від Вас та навчання ШНМ (раджу просто пробувати міняти кількість прихованих шарів, кількість нейронів в них та дивитися на результат). Але в прикладі вони будуть вказані.

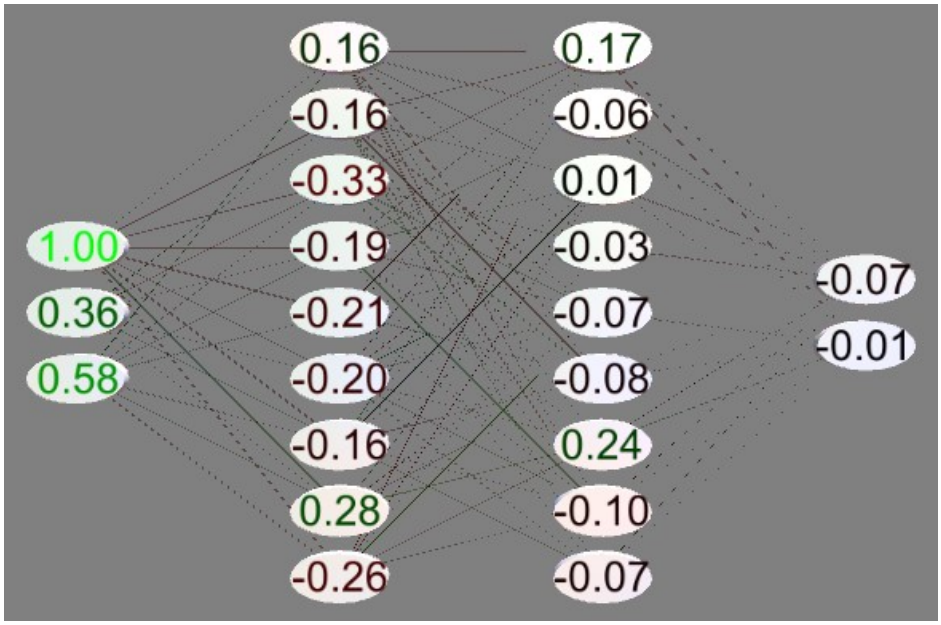
А на останок додамо інтерфейс персептрону для зручності зміни параметрів у ігровому режимі.

Не забуваємо, що для подачі даних персептрону у вхідний шар потрібно ці данні конвертувати так щоб вони вірно сприймалися ШНМ. А вихідні значення конвертувати так щоб отримувати потрібні дані.

Ось скрипт «мозку корови» **TutorialCowPerceptron.cs:**

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TutorialCowPerceptron : MonoBehaviour
{
    private TutorialCowControl THC;             //Cow control
    public Perceptron PCT = new Perceptron();   //Perceptron
    private PerceptronInterface PI;             //Perceptron interface

    void Start()
    {
        //Find cow control
        THC = gameObject.GetComponent<TutorialCowControl>();

        //Hiden layers and neurons
        int[] Layers = new int[2];
        Layers[0] = 9;
        Layers[1] = 9;

        //Create perceptron
        PCT.CreatePerceptron(1, false, true, 3, Layers, 2);

        //Add perceptron interface to game object & add perceptron to interface
        PI = gameObject.AddComponent<PerceptronInterface>();
        PI.PCT = PCT;
    }

    // Update is called once per frame
    void Update()
    {
        //Convert vaule
        PCT.Input[0] = THC.AngleToFood / 180F;      //Work with angles. Min vaule = -180, max vaule = 180
        PCT.Input[1] = THC.DistanceToFood / 41F;    //Work with distance. Max vaule = 41
        PCT.Input[2] = THC.Satiety / 50F;           //Work with satiety. Min vaule = 0, max vaule = 50
        PCT.Solution();                             //Perceptron solution

        //For this tutorial not need to convert vaule
        THC.Turn = PCT.Output[0];
        THC.Move = PCT.Output[1];
    }
}
```

А ось як «мозок» буде виглядати по заданим параметрам зі скрипта вище:



Урок №1. Навчання вибіркою завдань та відповідей.

Для навчання персептрона вирішувати завдання «Місія - не здохнути» (див. ст. 10 та 12) потрібна вибірка завдань. Можна довго сидіти і створювати цю вибірку самому, а можна просто зробити генератор. Знаючи, що «корова» помре коли «ситість» буде менша за 0 або більша за 50, а їжа дає +15 до «ситості» - треба вказати, що «їсти» при «ситості» більше 35 «корові» не можна. Також їй не можна «їсти» якщо кут повороту передньої частини «корови» та дистанції до їжі більше за 5 градусів.

Можна зробити певний висновок:

При будь яких умовах «корова» має зміст розвертатися до «їжі». Якщо «корова» далеко від «їжі», то є зміст їй рухатися до «їжі» до певної дистанції, а потім зачекати певного рівня «ситості», при умові, що вона не достатньо «голодна». Коли вона «зголодніла» - накинутися на «їжу», коли дивиться на неї.

Таке згодиться:

```
Turn = AngleToFood / 180F;
Move = 0F;
if (DistanceToFood > 3.5F && Mathf.Abs(AngleToFood) < 45)
    Move = 1F;
else if (Satiety < 25 && Mathf.Abs(AngleToFood) < 5)
    Move = 1F;
```

Створимо скрипт **TutorialCowPerceptron. cs** (див. ст. 14) і додамо його до «корови».

Створимо скрипт **CowLearning.cs** і додамо його до «корови».

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CowLearning : MonoBehaviour
{
    void Start ()
    {
        . . .
```

Потрібно створити вибірку завдань та відповідей (чим більше, тим краще. Але будьте обережні – вся вибірка при навчанні проходить за один кадр ігрового режиму). Створимо два двомірних масиви. Перший вимір для номеру завдання/відповіді, другий - для кожного нейрону вхідного/вихідного шару.

Варіант 1. Це може бути випадковий генератор завдань/відповідей на основі попередніх висновків (не забуваємо конвертувати завдання/відповіді):

```
. . .
float[][] Answers = new float[50][];
float[][] Tasks = new float[50][];
int i = 0;
while (i < Answers.Length)
{
    Tasks[i] = new float[3];
    if (i % 3 == 0)
    {
        Tasks[i][0] = Random.Range(-180F, 180F) / 180F;
        Tasks[i][1] = Random.Range(0F, 41F) / 41F;
        Tasks[i][2] = Random.Range(0F, 50F) / 50F;
    }
    else
    {
        Tasks[i][0] = Random.Range(-5F, 5F) / 180F;
        Tasks[i][1] = Random.Range(0F, 4F) / 41F;
        Tasks[i][2] = Random.Range(0F, 40F) / 50F;
    }
    Answers[i] = new float[2];
    Answers[i][0] = Tasks[i][0];
    Answers[i][1] = 0;
    if (Tasks[i][1] > 3.5F / 41F && Mathf.Abs(Tasks[i][0]) < 45F / 180F)
        Answers[i][1] = 1;
    else if (Tasks[i][2] < 25F / 50F && Mathf.Abs(Tasks[i][0]) < 2.5F / 180F)
        Answers[i][1] = 1;
    i++;
}
. . .
```

Варіант 2. Це може бути упорядкований генератор завдань/відповідей на основі попередніх висновків (не забуваємо конвертувати завдання/відповіді):

```
. . .
int i = 0;
int c = -1;
int p = -1;
int a = 9;  //number of angular variations
int d = 4;  //number of distance variations
int s = 4;  //number of variations of hunger
float[][] Answers = new float[a * d * s][];
float[][] Tasks = new float[a * d * s][];
while (i < Answers.Length)
{
    if (i % (a * s) == 0)
        c++;
    if (i % a == 0)
        p++;
    Tasks[i] = new float[3];
    Tasks[i][0] = ((-90F + 180F / (a - 1) * (i % a)) / (c + p + 1)) / 180F;
    Tasks[i][1] = ((41F - 41F / d * (c % d)) / (c + 1)) / 41F;
    Tasks[i][2] = (50F - 50F / s * (p % s)) / 50F;
    Answers[i] = new float[2];
    Answers[i][0] = Tasks[i][0];
    Answers[i][1] = 0;
    if (Tasks[i][1] > 3.5F / 41F && Mathf.Abs(Tasks[i][0]) < 45F / 180F)
        Answers[i][1] = 1;
    if (Tasks[i][2] < 25F / 50F && Mathf.Abs(Tasks[i][0]) < 2.5F / 180F)
        Answers[i][1] = 1;
    i++;
}
. . .
```

Також для, зручності, додамо інтерфейс навчання методом зворотного поширення помилки:

```
        . . .
        PerceptronBackPropagationInterface PLBBPI =
gameObject.AddComponent<PerceptronBackPropagationInterface>();
        . . .
```

Створені масиви завдань/відповідей заносимо до інтерфейсу навчання:

```
        . . .
        PLBBPI.Task = Tasks;
        PLBBPI.Answer = Answers;
        . . .
```

Вказуємо інтерфейсу «мозок» (персептрон):

```
        . . .
        PLBBPI.PCT = gameObject.GetComponent<TutorialCowPerceptron>().PCT;
    }
}
```

Тепер можна запускати ігровий режим. При бажанні або потребі настроїти персептрон як захочеться (тільки для донної вправи не міняйте Enters та не використовуйте Without Minus (див. ст. 7), без зміни конвертування змінних у персептроні та його навчанні) та змінити настройки навчання (не дуже грайтеся з Learning speed, при великих об’ємах вибірки завдання/відповіді може почати «підвисати». Раджу залишати = 1). Натискаємо на Learn OFF (див. ст. 8).

Тепер Ваш персептрон буде навчатися. Процес навчання буде рахуватися закінченим коли Max error буде меншим за Desired max error (див. ст. 8).

Іноді Max error буде більшим за Desired max error навіть після тривалого навчання. Зазвичай це обумовлено тим, що не правильно конвертовані вхідні/вихідні значення, або сама вибірка містить помилки/неточності, або недостатньо прихованих шарів(або нейронів у в них). Тут вже треба шукати помилки та виправляти їх, або пробувати змінювати приховані шари.

Зберегти персептрон, а потім завантажити його ви зможете завдяки інтерфейсу персептрону (див. ст. 7).

Урок №2. Навчання з «викладачем».

Тут все дуже схоже на урок №1, але замість вибірки треба створити «викладача». Для цього використаємо алгоритм з попереднього уроку:

```
Turn = AngleToFood / 180F;
Move = 0F;
if (DistanceToFood > 3.5F && Mathf.Abs(AngleToFood) < 90)
    Move = 1F;
else if (Satiety< 35 && Mathf.Abs(AngleToFood) < 5)
    Move = 1F;
```

Створимо скрипт **CowLearning.cs** (або переписуємо його якщо проходили урок №1) і додамо його до «корови»:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CowLearning : MonoBehaviour
{
    . . .
```

«Викладачу» потрібен буде доступ до змінних «корови»:

```
    . . .
    private TutorialCowControl TCC;
    . . .
```

«Викладач» буде давати тільки одну вибірку відповідей. Адже завдання буде йти від керування «корови». Створюємо двомірний масив вибірки відповідей. Перший вимір буде = 1 (лише одна вибірка), другий - для кожного нейрону вихідного шару.

```
    . . .
    public float[][] Answer = new float[1][];
    . . .
```

Також для, зручності, додамо інтерфейс навчання методом зворотного поширення помилки.

```
    . . .
    private PerceptronBackPropagationInterface PLBBPI;
    void Start ()
    {
        PLBBPI = gameObject.AddComponent<PerceptronBackPropagationInterface>();
        . . .
```

Отримуємо доступ до змінних «корови» та вказуємо розмір масиву вибірки у другому вимірі:

```
        . . .
        TCC = gameObject.GetComponent<TutorialCowControl>();
        Answer[0] = new float[2];
        . . .
```

Вказуємо інтерфейсу «мозок» (персептрон):

```
        . . .
        PLBBPI.PCT = gameObject.GetComponent<TutorialCowPerceptron>().PCT;
    }
    . . .
```

Нехай «вчитель» працює лише тоді, коли увімкнемо навчання у інтерфейсі:

```
    . . .
    void Update()
    {
        if (PLBBPI.Learn)
        {
        . . .
```

Тепер «вчитель» вказуватиме «мозку», що вчити:

```
        . . .
            Answer[0][0] = TCC.AngleToFood / 180F;

            if (TCC.DistanceToFood > 3.5F && Mathf.Abs(TCC.AngleToFood) < 90)
                Answer[0][1] = 1;
            else if (TCC.Satiety < 35 && TCC.DistanceToFood > 0 && Mathf.Abs(TCC.AngleToFood) < 5)
                Answer[0][1] = 1;
            else if (TCC.Move > 0)
                Answer[0][1] = 0;

            PLBBPI.Answer = Answer;
        }
    }
}
```

Тепер можна запускати ігровий режим. При бажанні або потребі настроїти персептрон як захочеться (тільки для донної вправи не міняйте Enters та не використовуйте Without Minus (див. ст. 7), без зміни конвертування змінних у персептроні та його навчанні) та змінити настройки навчання (Learning speed, та Desired max error не будуть впливати на навчання). Натискаємо на Learn OFF (див. ст. 8).

Рекомендація: Використовуйте низькі значення Learning rate (0.01 - 0.1).

Тепер Ваш персептрон буде навчатися. Але процес навчання може виявитися досить тривалим. Коли навчання закінчено будете вирішувати лише ви, або можете додати параметр який буде вказувати при яких умовах персептон навчений (наприклад: час життя, довжина пройденої дистанції, або кількість виконаних дій і т.п.).

Зберегти налаштування персептрону, а потім завантажити їх ви зможете завдяки інтерфейсу персептрону (див. ст. 7).

Урок №3. Навчання методом випадкової генерації.

Для цього уроку знадобиться ввести «довголіття» до скрипта «корови» (див. ст. 11)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TutorialCowControl : MonoBehaviour
{
    . . .
    public float LifeTime = 0;
    . . .
```

Це обумовлено тим, що при навчанні методом випадкової генерації треба відслідковувати кращого «клона» у поколінні.

Крім того, бажано збільшувати «довголіття» при виконанні вірних дій, або/та зменшувати «довголіття» при виконанні не вірних дій.

Нехай «довголіття» збільшується з часом. І нехай зменшується «довголіття» коли «корова» невірно дивиться на «їжу».

```
    . . .
    void Update()
    {
        . . .
        LifeTime += Time.deltaTime - (Mathf.Abs(AngleToFood) / 180F) * Time.deltaTime;
    }
}
```

Також будемо скидати «довголіття» коли «корова помирає».

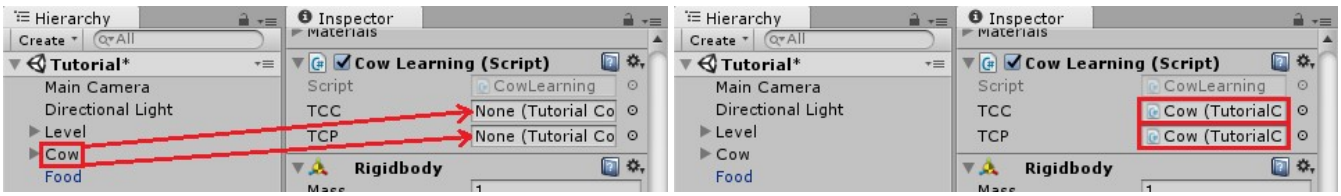
```
    . . .
    void Update()
    {
        . . .
        if (Death)
        {
            . . .
            LifeTime = 0;
        }
        . . .
```

Створимо скрипт **CowLearning.cs** (або переписуємо його якщо проходили урок №1 або урок №2) і додамо його до будь-якого об’єкту:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CowLearning : MonoBehaviour
{
    . . .
```

Далі треба вказати скрипти керування «корови» та «мозок корови»:

```
    . . .
    public TutorialCowControl TCC;
    public TutorialCowPerceptron TCP;
    . . .
```



Додамо інтерфейс методу випадкової генерації:

```
    . . .
    private PerceptronRandomGenerationInterface PRGI;

    void Start()
    {
        PRGI = gameObject.AddComponent<PerceptronRandomGenerationInterface>();
        . . .
```

Вкажемо інтерфейсу персептрон який треба навчити:

```
        . . .
        PRGI.PCT = TCP.PCT;
        . . .
```

Та додамо до методу випадкової генерації потрібну інформацію щодо «корови»:

```
        . . .
        PRGI.PLBRG.StudentData(TCP.gameObject, TCP, "PCT", TCC, "Death", "LifeTime");
    }
}
```

Пояснення дивіться на сторінці 6.

Тепер можна запускати ігровий режим. При бажанні або потребі настроїти персептрон як захочеться (тільки для донної вправи не міняйте Enters та не використовуйте Without Minus (див. ст. 7), без зміни конвертування змінних у персептроні та його навчанні) та змінити настройки навчання (див. ст. 9). Натискаємо на Learn OFF (див. ст. 9).

Тепер Ваш персептрон буде навчатися. Ефект навчання зазвичай видно вже з перших поколінь. Але процес навчання може виявитися досить тривалим. Процес навчання можна рахувати завершаним, коли є суттєва різниця між Best Generation та Generation (див. ст. 9). Щоб зупинити процес навчання, та передати вагові зв'язки навчання до персептрону який навчається натисніть Learn ON.

Пограйтеся з налаштуванням для кращого розуміння інтерфейсу методу випадкової генерації.

Зберегти налаштування персептрону, а потім завантажити їх ви зможете завдяки інтерфейсу персептрону (див. ст. 7).

Як зберегти та завантажити налаштування персептону.

Як створити персептрон ми вже розглядали (див. ст. 13). Як зберегти/завантажити налаштування та вагові зв'язки за допомогою інтерфейсу дивіться на сторінці 7. А от як зберегти та завантажити його налаштування та вагові зв'язки у скриптах розглянемо далі.

Щоб зберегти налаштування та вагові зв'язки персептрону використовуйте команду:

```
. . .
PerceptronName.Save("SaveName");
. . .
```

Де SaveName – ім'я файлу для збереження.

Файл налаштуваня та вагови зв'язків буде збережено за адресою:

```
Application.dataPath + "/ANN/PerceptronStatic/" + PerceptronFile + ".ann"
```

PerceptronFile - ім'я файлу для збереження/завантаження (див. ст. 4).

Щоб завантажити налаштування та вагові зв'язки персептрону замість використання команди «PerceptronName.CreatePerceptron(…)» використовуйте команду «PerceptronName.Load("LoadName");»:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class NameOfScript : MonoBehaviour
{
    public Perceptron PerceptronName = new Perceptron();
    . . .
    void Start()
    {
        PerceptronName.Load("LoadName");
        . . .
```

Де LoadName – ім'я файлу для завантаження.

Заключне слово.

Сподіваюся, що Вам сподобається моя робота. Я намагався якомога полегшити завдання створення, навчання, збереження та завантаження налаштувань персептрону та його вагових зв’язків.

Якщо у Вас виникнуть питання, або з’являться пропозицій щодо поліпшення даної роботи – пишіть на VirtualSUN13@gmail.com. З радістю відповім.

Особлива подяка Леониду Терешонкову за моральну підтриму та Юлі Павлович за переклад. ☺