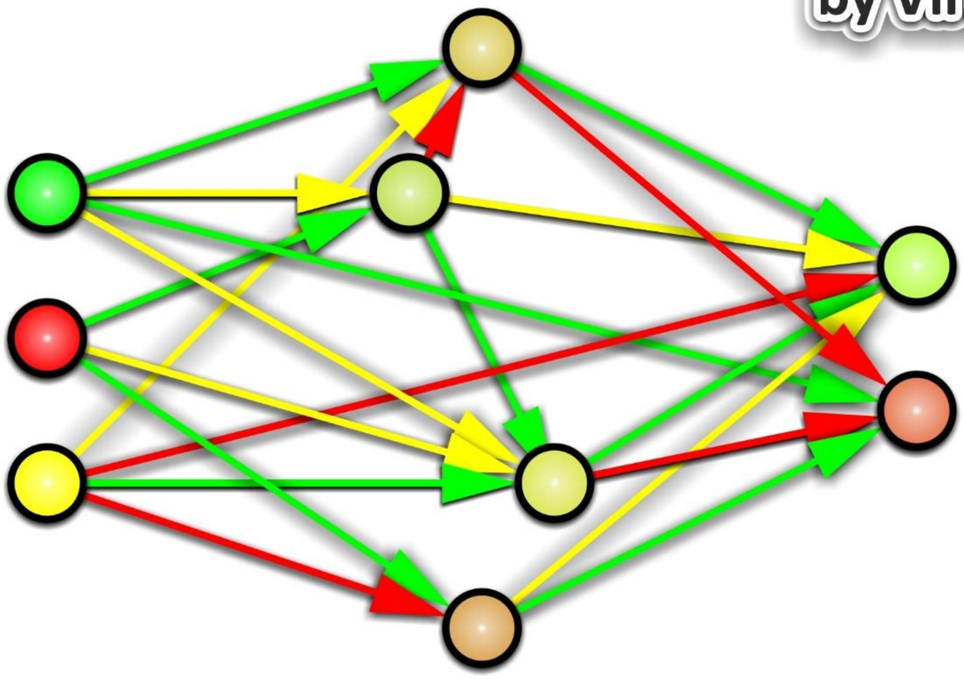
ARTIFICIAL NEURAL NETWORK ANN & NEAT

by VirtualSUN



A set of scripts for creating and learning your own artificial neural network.

Creating your own artificial neural network (ANN) is not a big issue. But to teach the ANN to do certain tasks is a real challenge. In order to ease the task of creating and teaching the ANN, I wrote a set of scripts, that will do everything for you. The only thing that is needed from you is to “explain” the ANN properly, what information is needed to be learned.

2019

CONTENT

A general scripts description.

ANN.cs is a non MonoBehaviour script of an artificial neural network (ANN). This script will automatically create the ANN based on your parameters, fulfill a task (if the training takes place), save and load your ANN.

ANNNode.cs is a non MonoBehaviour script that contains the core information of the ANN's node.

ANNConnection.cs is a non MonoBehaviour script that contains the core information of the ANN's connection between nodes.

ANNInterface.cs script can be used to facilitate the creation, visualization, saving and loading of ANNs during the learning process. This is the interface for the ANN in the game mode.

ANNLearnByNEAT.cs is a non MonoBehaviour script for teaching the ANN by NEAT (NeuroEvolution of Augmenting Topologies). With the help of flexible settings, you can quickly and easily complete the training of the ANN.

ANNLearnByNEATInterface.cs is an interface script for teaching the ANN by the NEAT in the game mode.

InterfaceGUI.cs is a non MonoBehaviour script (static) for interfaces' scripts.

DrawANNWeight.cs is a non MonoBehaviour script (static) for the ANN's weights visualization. It is used by the ANNInterface.cs script.

Formulas.cs is a non MonoBehaviour script (static) for fulfilling cyclic and alike tasks.

A detailed description of the main scripts.

public class ANN – is a non MonoBehaviour script for an artificial neural network (ANN).

The main variables of the ANN.cs script	
public float AFS	A size of the activation function.
public bool B	If it is “true”, then in each node (neuron) the additional weight of the bias is activated. Does not work on input neurons.
public bool AFWM	If it is “true”, then all the neurons take the values from -1 to 1. This also applies to incoming and outgoing neurons. If it is “false” then all the neurons take values from 0 to 1.
public float[] Input	The input layer of the ANN. The size of the array indicates the number of the input values.
public float[] Output	Output layer of the ANN. The size of the array indicates the number of the output values.
public Dictionary<int, ANNNode> Node	Dictionary of the nodes (neurons) of ANN. Contains each node of the ANN.
public Dictionary<int, ANNConnection> Connection	Dictionary of the connections (neurons) of ANN. Contains each node of the ANN.

The Command of the Script	**The variables of the Commands**	**Description**
public void Create		Creation of ANN with the help of parameters.
	float AFS	A Size of an activation function.
	bool B	If it is “true”, then in each node (neuron) the additional weight of the bias is activated. Does not work on input neurons.
	bool AFWM	If it is "true", then all the neurons’ values are from -1 to 1. It also applies to the input and output neurons. If it is "false", the neurons take values from 0 to 1.
	int NumberOfInputs	The amount of the input neurons.
	int NumbersOfOutputs	The number of output neurons.
public void Load		Load the ANN from the file. Supports “ANN Perceptron” storage files.
	string ANNFile	Filename for loading.
public void Solution		ANN’s solution.
private float ActivationFunction		The neurons activation function.
	float Sum	The total sum of all the values of neurons multiplied by their weight.
private void CreatingNeurons		Creation of the neurons and their units with each other.
	StreamReader SR	The specified stream from the load file. Use "null" if the file is not used.
public void Save		Save ANN’s parameters to a file.
	string ANNFile	The name of the file is to be loaded.
public void FixConnections()		Arrange the numbering of nodes in the connections.
public void NumberingCorrection()		Sort node numbering in order.

public class ANNNode is a non MonoBehaviour script that contains the core information of the ANN's node.

The main variables of the ANNNode.cs script	
public float Neuron	The value of neuron.
public float Bias	The value of neuron's bias.
public ArrayList ConnectionIn	List of input connections numbers.
public int Fullness	The fullness of the neuron connections when solving.
public Vector2 Position	Position at visualization.

The Command of the Script	**The variables of the Commands**	**Description**
public ANNNode		Input information about the neuron.
	float Neuron	The value of neuron (node).
	float Bias	The value of neuron's bias.
	ArrayList ConnectionIn	List of input connections numbers.
	int Fullness	The fullness of the neuron connections when solving.
	Vector2 Position	Position at visualization.
	string Info	Text view of a neuron (node).
public override string ToString		Transfers information of the node to a text view.

public class ANNConnection is a non MonoBehaviour script that contains the core information of the ANN's connection.

The main variables of the ANNConnection.cs script	
public int In	Input neuron number.
public int Out	Output neuron number.
public float Weight	Connection weight.
public bool Enable	Connection activity.

The Command of the Script	**The Variables of the Command**	**Description**
public void ANNConnection		Input information about the connection.
	public int In	Input neuron number.
	public int Out	Output neuron number.
	public float Weight	Connection weight.
	public bool Enable	Connection activity.
	string Info	Text view of a connection.
public override string ToString()		Transfers information of the connection to a text view.

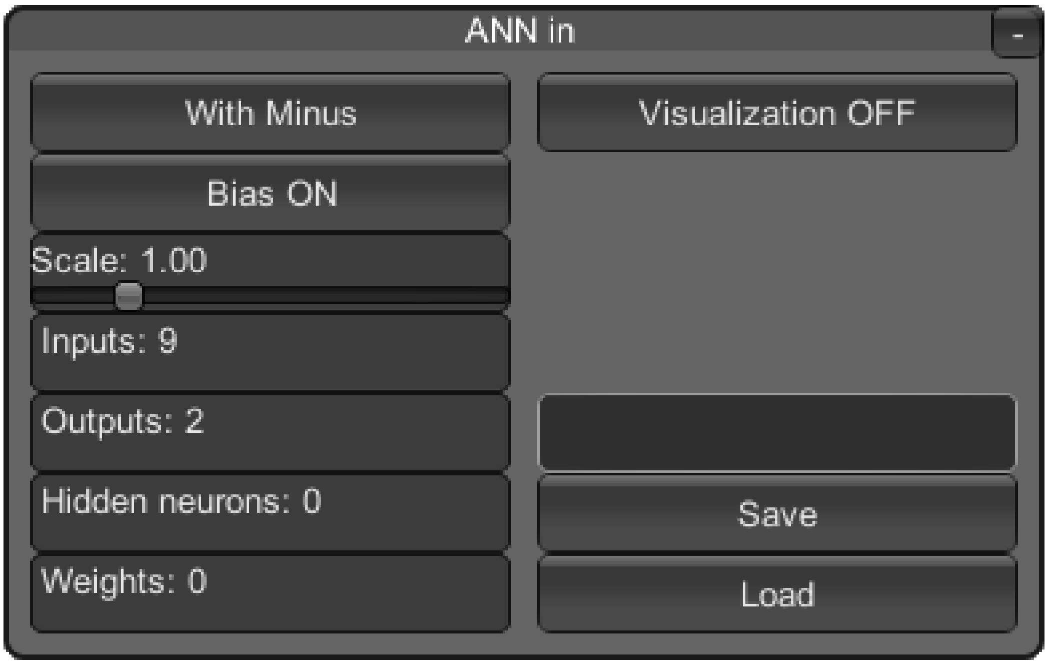
public class ANNLearnByNEAT is a non MonoBehaviour script for teaching ANN.

The main variables of the ANNLearnByNEAT.cs script	
public ANN Ann	ANN what is need to be learning.
public int AmountOfChildren	The number of "children" in an each generation.
public int ChildrenInWav	Amount of "children" in each generation's wave.
public bool ChildrenByWave	If "true" - by waves. If "false" - use maximum in one time.
public int BestGeneration	The best generation at the moment.
public int Generation	The total number of generations.
public int ChildrenInGeneration	The number of "children" in this generation.
public bool Cross	Permission to "cross" the two best "children".
public bool PerceptronStart	If "true", then in the first generation of children all input neurons will be connected with the output neurons.
public bool Autosave	Auto-save when learning.
public int AutosaveStep	Step of auto-save.
public string AutosaveName	The initial file name for auto-save.
public bool IgnoreCollision	If "true", it will be ignored the collision of children.
public float BestLongevity	The best "longevity" at the moment.
public float ChildrenDifference	The difference of the weight units among the generations.
public float Chance	A chance to choose the better from the worse, current generation. It changes because of the parameter public float ChanceCoefficient with each new generation.
public float ChanceCoefficient	The affect coefficient on the chance of random choice of the worse, current generation. The command works under the condition that it is not equal to zero.
public float MutationAddNeuron	Proportion of using mutation "add neuron".
public float MutationAddWeight	Proportion of use of mutation "add weight".
public float MutationChangeOneWeight	Proportion of use of mutation "change one weight".
public float MutationChangeWeights	Proportion of using mutation "change all weights".
public float MutationChangeOneBias	Proportion of using mutation "change one bias".
public float MutationChangeBias	Proportion of using mutation "change all bias".
public float MutationSum	The sum of the proportional values of the mutation.
public int AddingWeightsCount	The number of simultaneous adding of neuronal relationships to each "child".
public float ChangeWeightSign	The chance to change the sign of the weight.

The Command of the Script	**The Variables of the Command**	**Description**
public void StudentData		A data collection for the learning.
	GameObject Student	The main game object (GameObject), that has to learn.
	Object HereIsANN	A script, that contains ANN.
	string ANNName	The name of the variable (ANN), that was given to the ANN in the script, which contains ANN (HereIsANN).
	Object StudentControls	The control script of main game object.
	string StudentCrash	The name of the variable (bool), that was given to the reason for the "crash" of the game object in the control script (StudentControls).
	string StudentLife	The name of the variable (float), that was given to the "longevity" of the game object in the control script (StudentControls).
public void Learn		Teaching the ANN.
public void StopLearn		The immediate stop of learning with the transfer of weight units information of the better ANN from a better generation to ANN, that is being trained.
public void Reset		Reset learning information.

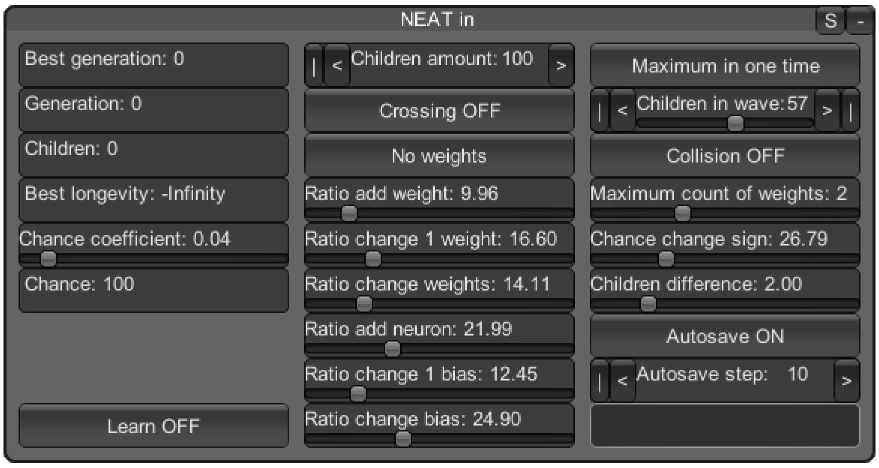
Description of the auxiliary interfaces.

ANNInterface.cs - this script can be used to facilitate the creation, storage, and loading of ANN during the learning process. This is the interface for the ANN in the game mode. It controls the parameters of the **ANN.cs** script.



Without Minus With Minus	If "With Minus" is shown, then all the neurons' values are from -1 to 1. This also applies to the input and output neurons. If "Without Minus" is shown, then all the neurons' values are from 0 to 1. It controls the parameter public bool AFWM.
Bias OFF Bias ON	If "Bias ON" is shown, then in each node (neuron) the additional weight of the bias is activated. Does not work on input neurons.. If "Bias OFF" is shown, then there is no bias in neurons. The controls the parameter public bool B.
Scale	The scale of the activation function. The controls the parameter public float AFS.
Inputs	It indicates the number of neurons in the input layer.
Outputs	It indicates the number of neurons in the output layer.
Hidden neurons	Total number of hidden neurons.
Weights	Total number of weights.
Visualization OFF Visualization ON	If "Visualization ON" is shown, then it shows the structure of ANN with the given parameters. If "Visualization OFF" is shown, then it does not show the structure of ANN with the given parameters.
GUI.TextField	The filename for the saving or loading ANN.
Save	It saves the parameters and all the weight units of ANN into the file, if the filename is specified. It controls the command public void Save.
Load	It loads the parameters and all the weight units of the ANN from the file, if the filename is specified. It controls the command public void Load.

ANNLearnByNEATInterface.cs - is an interface script for teaching the ANN by NEAT (NeuroEvolution of Augmenting Topologies). It controls the parameters of the **ANNLearnByNEAT.cs** script.



NEAT in

S -

Best generation: 0

Generation: 0

Children: 0

Best longevity: -Infinity

Chance coefficient: 0.04

Chance: 100

Learn OFF

| < Children amount: 100 >

Crossing OFF

No weights

Ratio add weight: 9.96

Ratio change 1 weight: 16.60

Ratio change weights: 14.11

Ratio add neuron: 21.99

Ratio change 1 bias: 12.45

Ratio change bias: 24.90

Maximum in one time

| < Children in wave: 57 > |

Collision OFF

Maximum count of weights: 2

Chance change sign: 26.79

Children difference: 2.00

Autosave ON

| < Autosave step: 10 >

Best Generation	It shows the best generation number at the very moment. It gets the data from the parameter public int BestGeneration.
Generation	It shows what generation is like at the moment. It gets the data from the parameter public int Generation.
Children	It shows the number of "children" in the current generation. It gets the data from parameter public int ChildrenInGeneration.
Best longevity	It shows a better longevity at the moment. It gets the data from the parameter public float BestLongevity.
Chance coefficient	If !=0 than the chance of changing the weight sign to the opposite after the generation impact on it. Manipulates the parameter public float ChanceCoeefficient.
Chance	It shows the chance of choosing the current worse generation. It changes with each new generation. It gets the data from the parameter public float Chance.
Learn OFF Learn ON	If "Learn ON" is used, then it starts the training of the ANN. It controls the command public void Learn. If "Learn OFF" is used and the training took place, then it stops the training. It uses the command public void StopLearn.
Children amount	The amount of "children" in each generation. It controls the parameter public int AmountOfChildren.
Crossing OFF Crossing ON	Enable / Disable "cross" the two best "children". It controls the parameter public bool Cross.
No weights Perceptron	If the "Perceptron" shows, then in the first generation of children all the input neurons will be connected to the output neurons. It controls the parameter public bool PerceptronStart.
Ratio add weight	Proportion to use the "add weight" mutation. It controls the parameter public float MutationAddWeight.
Ratio change 1 weight	Proportion of use of mutation "change one weight". It controls the parameter public float MutationChangeOneWeight.
Ratio change weights	Proportion of using mutation "change all weights". It controls the parameter public float MutationChangeWeights.
Ratio add neuron	Proportion of using mutation to "add neuron". It controls the parameter public float MutationAddNeuron.
Ratio change 1 bias	Proportion of using mutation "change one bias". It controls the parameter public float MutationChangeOneBias.
Ratio change bias	Proportion of mutation use "change all bias". It controls the parameter public float MutationChangeBias.
Maximum in one time By waves	Ability to study by waves in a generation. A constant maximum number or a certain amount in a single wave. It controls the parameter public bool ChildrenByWave.
Children in wave	The number of "children" in one wave. It controls the parameter public int ChildrenInWave.
Collision ON Collision OFF	Enable / disable collision between "children". It controls the parameter public bool IgnoreCollision.
Maximum count of weights	The maximum number of simultaneous connections between neurons in each "child". It controls the parameter public int AddingWeightsCount.
Chance change sign	The chance to change the sign of weight. It controls the parameter public float ChangeWeightSign.
Children difference	The difference of weight units among the generations. It controls the parameter public float ChildrenDifference.
Autosave OFF Autosave ON	Enable/disable auto-save of ANN during training. It controls the parameter public bool Autosave.
Autosave step	Save the ANN at each specified step. It controls the parameter public int AutosaveStep.
GUI.TextField	The filename for the saving or loading ANN.

How to use.

First of all, a specific task for ANN has to be created. It is to be remembered, that the ANN must receive a certain input data and it has to be properly prepared. ANN accepts input data from 0 to 1, or from -1 to 1 (see Page 4, "public bool AFWM"). It is also necessary to determine the number of the input data.

The same applies to the output data. Consequently, they must be correctly converted for the correct response to the task.

The amount of the hidden layers and the neurons in them depends on learning. In any case, their amount of the neurons and connections has a different influence on the quality and speed of learning of the ANN.

In general, the study of ANN does not differ much from the training in the project "ANN Perceptron".

The task "Mission is "Not to Die".

Let's look into the prepared task: There is a "cow" which eventually wants to eat. There is "food", that the "cow" can eat. The "cow" will die, if it does not eat or eat too much. The "cow" can only eat with the front side of the body, otherwise it will die. The "cow" must be taught to eat properly and in time.

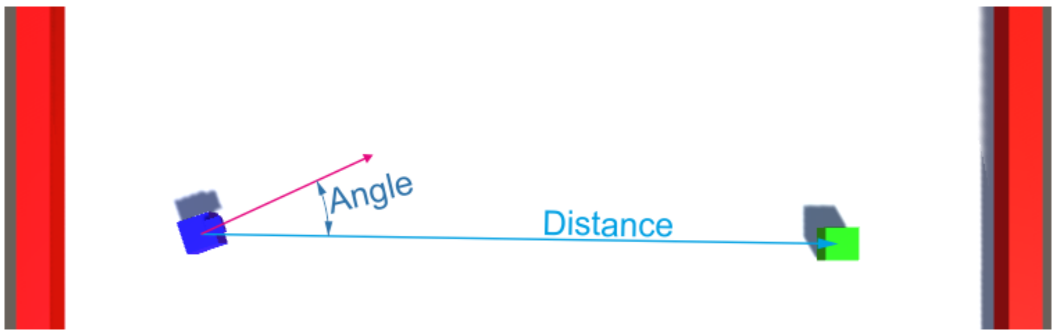
The Tutorial folder has already prepared scripts and a scene for the task.

The "cow" gets three meanings:

1. The distance to "food". The level diagonal is about 41.
2. The angle of rotation with the sign relative to the front of the "cow" to "food". From -180 to 180.
3. The "cow's fullness" that decreases over time. 50 is the maximum value for the "survival".

The "cow" is guided by two values:

1. Turn to food. From -1 to 1.
2. Move to food. From -1 to 1.



Also, the "cow" has the additional values for getting the above mentioned.

The "Cow" script is the following **TutorialCowControl.cs:**

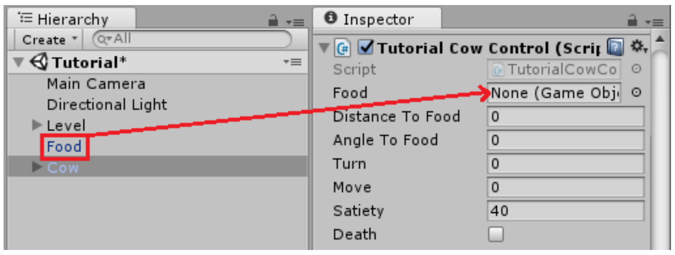
```
using UnityEngine;

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;
    }
}
```

"Food" affects the cow, when it is touched. If the "cow" eats "food" correctly (the angle of rotation does not exceed 5 degrees), then it increases its "fullness" (+15), and "food" changes its position. Otherwise it will die.

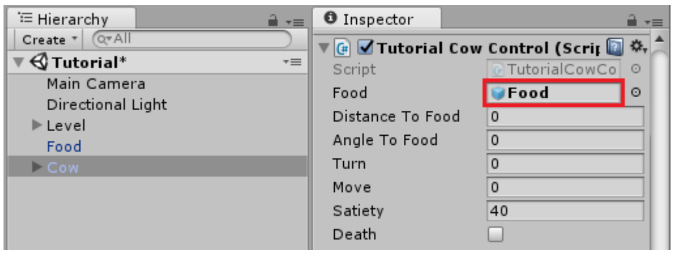
The "food" script is the following **TutorialFood.cs:**

```
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();
            }
        }
    }

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

Do not forget to indicate the "cow" where the "food" is:



Script	TutorialCowCo
Food	None (Game Obj
Distance To Food	0
Angle To Food	0
Turn	0
Move	0
Satiety	40
Death	☐



Script	TutorialCowCo
Food	Food
Distance To Food	0
Angle To Food	0
Turn	0
Move	0
Satiety	40
Death	☐

There is the "cow" and the "food" control. Now the "cow's brain" has to be created.

How to create the ANN.

To create ANN, you need to create a MonoBehaviour script and type in the following:

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

public class NameOfScript : MonoBehaviour
{
    public ANN AnnName = new ANN();
    . . .
    void Start()
    {
        AnnName.Create(3, 2);
        . . .
    }

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

See the explanation on Page 4.

Before using AnnName.Solution () the converted input data has to be entered into the input neurons.

After using AnnName.Solution () – the output data of the output neurons has to be output and then converted.

For the task "Mission is not to die", we will create a "cow's brain" in a file with the following name **Lesson4TutorialCow.cs**. It has to be added to the "cow" object.

The "cow" has three main values that it has to take and two control values (see Page 9). Consequently, the ANN with three input and two output neurons is needed. To make it easier, the ANN with the negative (a minus) value is created (see Page 4, bool AFWM). And now the amount of hidden neurons and connections with them will depend on the training of ANN.

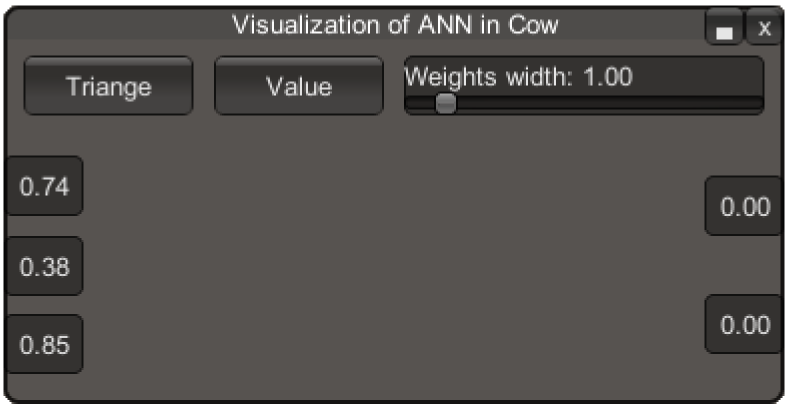
And finally, it will be convenient to change the parameters in the game mode, if we add the ANN interface.

Do not forget, that to input data of ANN into the input layer, it is necessary to convert these data the way it is correctly taken by ANN. And to convert the output values the way to get the data you need.

The "cow’s brain" script is the following **Lesson4TutorialCow.cs:**

```
using UnityEngine;
public class Lesson4TutorialCow : MonoBehaviour
{
    private TutorialCowControl THC;     //Cow control
    public ANN Ann = new ANN();         //ANN
    private ANNInterface NI;            //ANN interface

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

        //Create ANN
        Ann.Create(3, 2);

        //Add ANN interface to game object & add ANN to interface
        NI = gameObject.AddComponent<ANNInterface>();
        NI.Ann = Ann;
    }

    void Update()
    {
        //Convert vaule
        Ann.Input[0] = THC.AngleToFood / 180F;      //Work with angles. Min vaule = -180, max vaule = 
180
        Ann.Input[1] = THC.DistanceToFood / 41F;    //Work with distance. Max vaule = 41
        Ann.Input[2] = THC.Satiety / 50F;           //Work with satiety. Min vaule = 0, max vaule = 50
        Ann.Solution();                             //ANN solution
        //For this tutorial not need to convert vaule
        THC.Turn = Ann.Output[0];
        THC.Move = Ann.Output[1];
    }
}
```

Here's how the "brain" will look like with the specified parameters from the script above:



And YES - it's empty here.

Lesson # 4. Training.

For this lesson, you will need to enter "longevity" into the "cow" script (see Page 10)

```
using UnityEngine;

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

This is due to the fact that when learning needs to track a better "clone" in the generation.

In addition, it is better to increase "longevity", when the correct actions are done, and/or to reduce the "longevity", when the actions are done incorrectly.

Let the "longevity" increase over time. And let the "longevity" reduce, when the "cow" looks at "food" incorrectly.

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

We will also reset the "longevity", when the "cow dies". This is not a mandatory action (the script itself cancels "longevity").

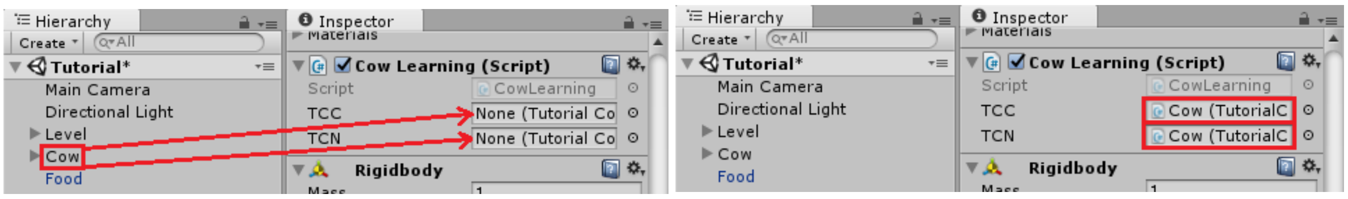
```
    . . .
    void Update()
    {
        . . .
        if (Death)
        {
            . . .
            LifeTime = 0;
        }
        . . .
```

Create a **Lesson4CowLearning.cs** script and add it to any object:

```
using UnityEngine;

public class Lesson4CowLearning: MonoBehaviour
{
    . . .
```

Then, you need to specify the "cow" and the "cow's brain" control scripts:

```
    . . .
    public TutorialCowControl TCC;
    public Lesson4TutorialCow TCN;
    . . .
```



Script CowLearning
TCC None (Tutorial Co
TCN None (Tutorial Co

Script CowLearning
TCC Cow (TutorialC
TCN Cow (TutorialC

Add a learning interface:

```
    . . .
    private ANNLearnByNEATInterface NLI;

    void Start()
    {
        NLI = gameObject.AddComponent<ANNLearnByNEATInterface>();
        . . .
```

Specify the interface ANN that is needed to be taught:

```
        . . .
        NLI.PCT = TCP.Ann;
        . . .
```

To the NEAT method add the proper information about the "cow":

```
        . . .
        NLI.NL.StudentData(TCP.gameObject, TCP, "Ann", TCC, "Death", "LifeTime");
    }
}
```

See the description on Page 6.

Also, for more effective training, to the **TutorialFood.cs** add bonuses for "children" when they are right "eating food":

```
. . .
    void OnCollisionEnter(Collision col)
    {
        . . .
        if (TCC != null && !Moving)
        {
            . . .
            else
            {
                . . .
                TCC.LifeTime += 15;
            }
        }
    }
. . .
```

Now you can run the game mode. Do not use Without Minus (see Page 7) for this task. Change the training settings according to your desire (see Page 8). Click Learn OFF (see Page 8).

Now your ANN is starting learning. You can notice the effect of learning in the first generation. But the learning process can take a quite long time. The learning process can be considered completed if there is a significant difference between Best Generation and Generation (see Page 8). To stop the learning process, and to transfer the weight units of the learning to the ANN, that is being learned, click Learn ON. The ANN configuration can be saved, and then loaded through the interface of the ANN (see Page 7).

How to save and load the ANN's settings.

We have already reviewed how to create the ANN (see Page 12). See Page 7 to find out how to save/load settings and weight units with the help of the interface. Now we will consider how to save and load its settings and weight units in the scripts.

To save the settings and weight units of the ANN, use the following command:

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

SaveName is the filename for saving.

The configuration file and the weight units will be saved to the following address:

```
Application.dataPath + "/ANN/ANN/" + ANNFile + ".ann"
```

ANNFile is the filename for saving/loading (see Page 4).

Use the AnnName.Load ("LoadName") command instead of using the command AnnName.Create (...) to load the ANN configuration and weight units:

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

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

LoadName is the filename for the loading.

The conclusion.

I hope you have enjoyed my work. I tried to ease the task of creating, learning, saving and loading the ANN setting and its weights.

If you have any questions, or there are any suggestions for improving this work, write to VirtualSUN13@gmail.com. I'll be happy to answer.

Special thanks to Leonid Tereshonkov for a moral support and Yulia Pavlovych for translation the document. ☺

PS: And do not forget to leave your feedback about the project (http://u3d.as/1rTD). I would be very grateful to you.

You can also update "ANN&NEAT" to "ANN&TOOLS".

Contacts.

YouTube: https://www.youtube.com/channel/UClblqhEzoATg-Jvk0AviVlw

Unity Connect: https://connect.unity.com/u/sergey-voroshilov-vladimirovich

Twitter: https://twitter.com/sun_virtual

Instagram: https://www.instagram.com/virtualsun13/

Forum: https://forum.unity.com/threads/released-ann-perceptron.558868/

E-mail: VirtualSUN13@gmail.com