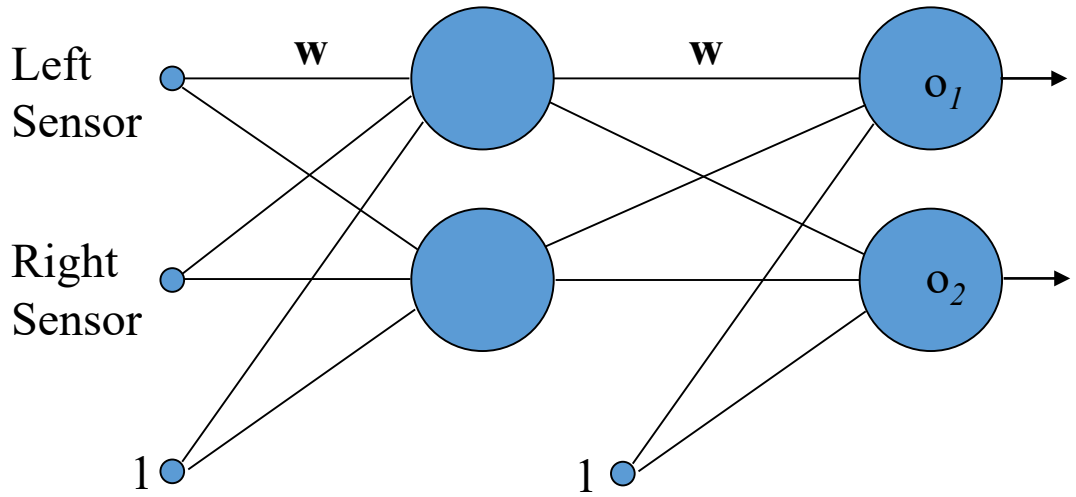
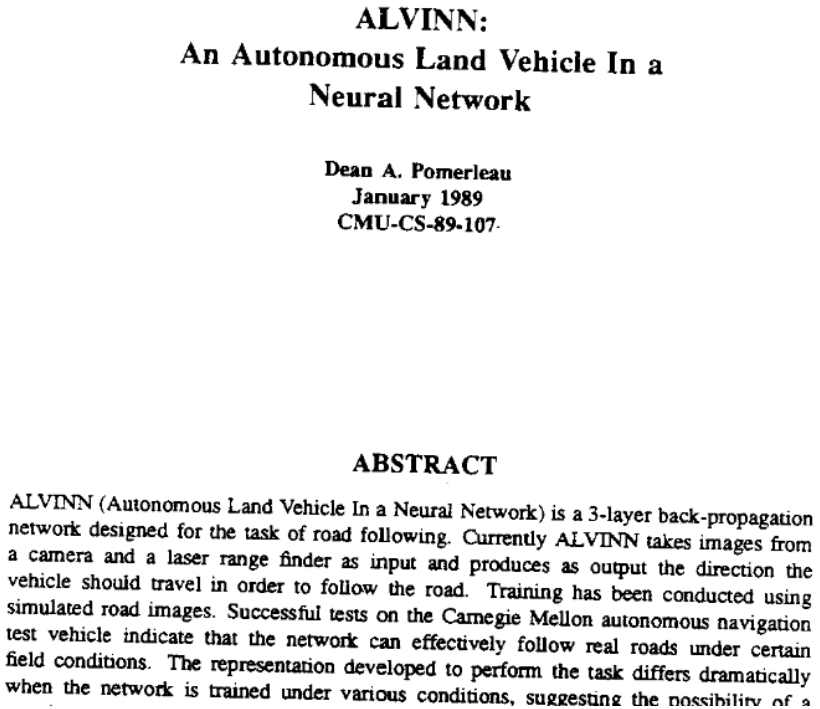
Evolving Neural Networks to Optimize Connection Weights



CJ Chung
LTU Computer Science

ALVINN: believe to be the first vehicle road following project using NN in 1989 at CMU

https://kilthub.cmu.edu/articles/journal_contribution/ALVINN_an_autonomous_land_vehicle_in_a_neural_network/6603146



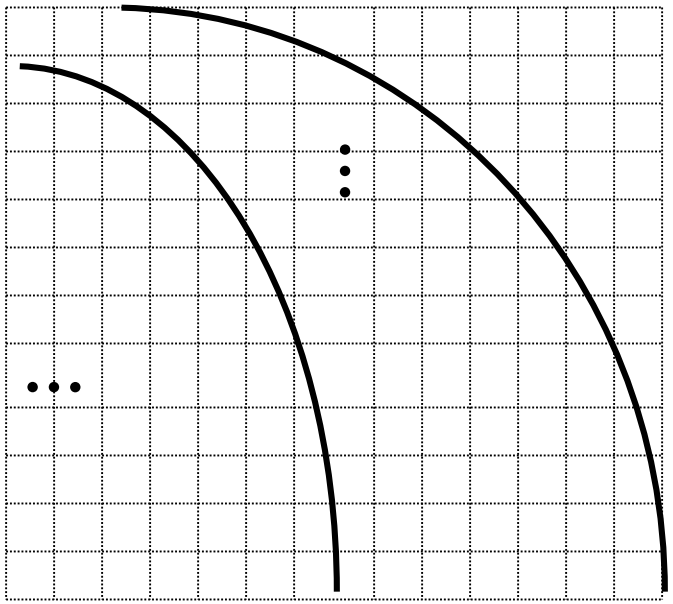
ALVINN:
An Autonomous Land Vehicle In a Neural Network

Dean A. Pomerleau
January 1989
CMU-CS-89-107

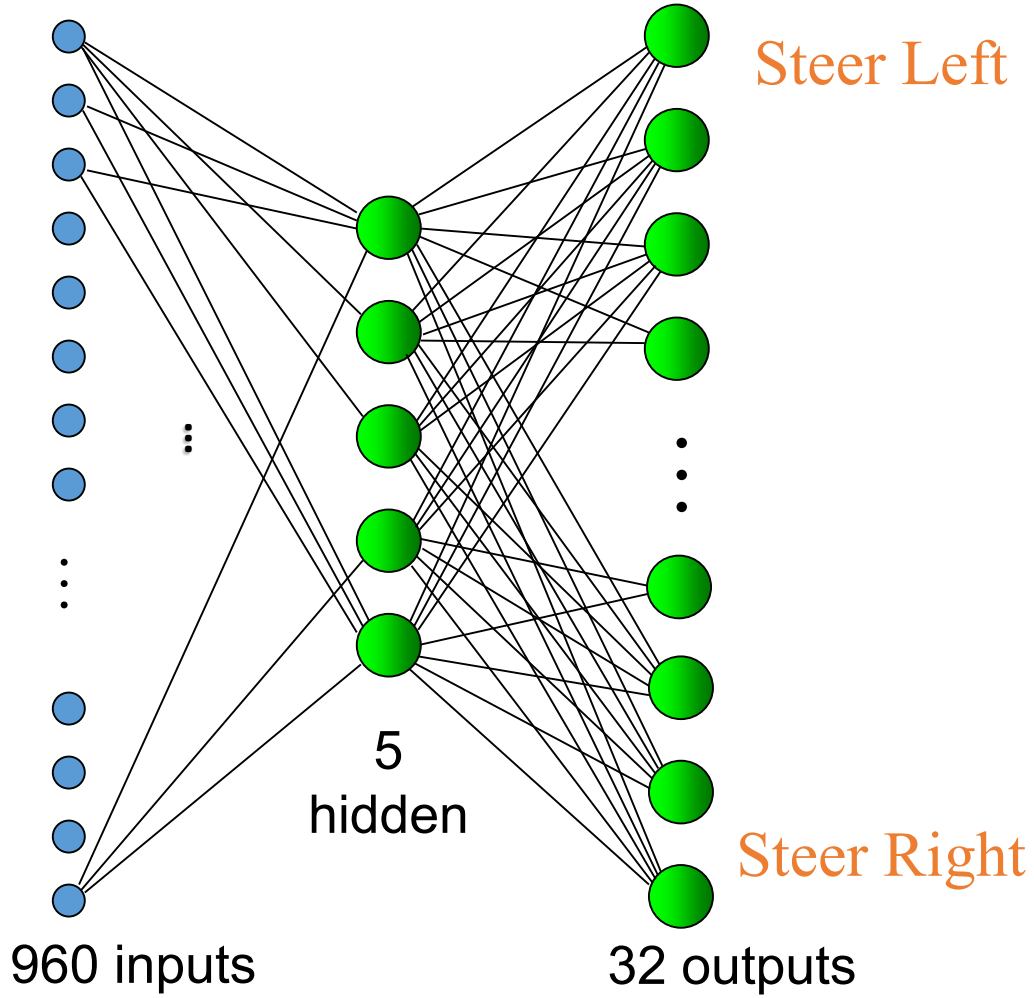
ABSTRACT

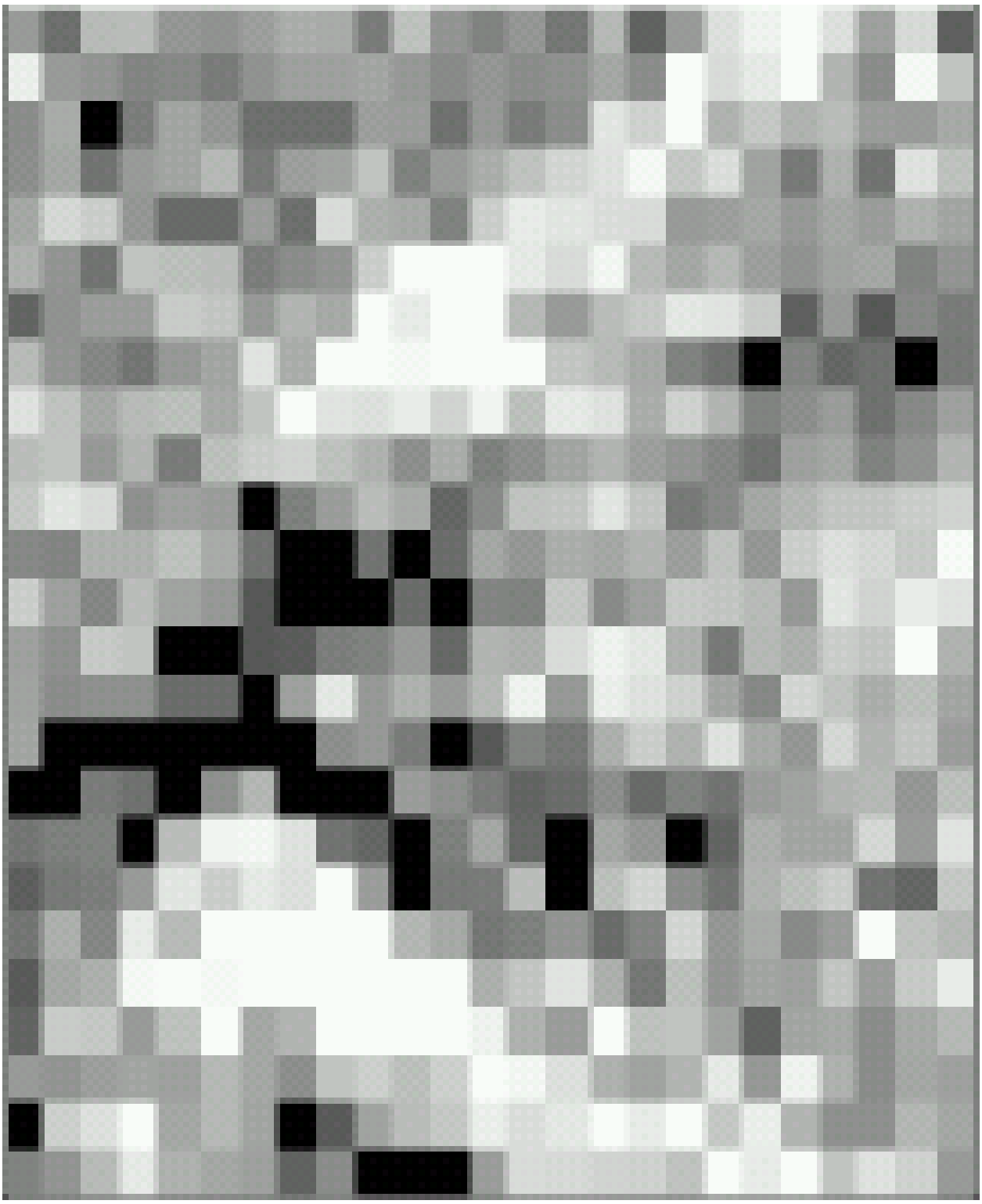
ALVINN (Autonomous Land Vehicle In a Neural Network) is a 3-layer back-propagation network designed for the task of road following. Currently ALVINN takes images from a camera and a laser range finder as input and produces as output the direction the vehicle should travel in order to follow the road. Training has been conducted using simulated road images. Successful tests on the Carnegie Mellon autonomous navigation test vehicle indicate that the network can effectively follow real roads under certain field conditions. The representation developed to perform the task differs dramatically when the network is trained under various conditions, suggesting the possibility of a

ALVINN:
Autonomous Land Vehicle In a Neural Net



30x32





CJ Chung

Why ANN as a controller model?

NN may be good at

- noisy
- unstructured
- dynamic
- partially observable
- uncertain environments

If trained well…

ALVIN

- **Capabilities:**
  - Learns to drive a van using a camera image
  - can drive around 40 mph once trained
  - Person provides training input in the form of an example of correct driving
- **Network:**
  - 1 hidden layer (5 nodes) , 4997 trainable weights
  - input layer = intensity from 30x32 photosensitive array (960 inputs)
  - output layer = 32 nodes on continuum from steer left to steer right

Calculating number of weights to train with one hidden layer

- i: number of inputs
- h: Number of hidden neurons
- m: number of output neurons

- $(i+1) \cdot h + (h+1) \cdot m$

Problems at that time in 1989

- Optimizing a function with 5,000 variables was hard
- CMU developed a “fast BP algorithm”
- Still gets stuck at local optimas

An ANN training can be considered as an minimizing objective function with multiple local minimums?

- Yes, training an Artificial NN can indeed be seen as minimizing an objective function (or loss function) that often has multiple local minima. This is because the optimization landscape of ANNs is typically non-convex due to their complex architectures, such as multiple layers, activation functions, and interconnections.
- https://www.adventuresinwhy.com/post/local-minima-distribution

Key points to consider in training NN (optimizing the NN)

- Non-Convex Landscape: The loss function's surface has many valleys and peaks due to the high-dimensional parameter space.
- Optimization Algorithms: Techniques like gradient descent or its variants (e.g., Adam, RMSProp) are used to navigate this landscape. They attempt to find a local or global minimum that reduces the error. EDL might be an alternative technique.
- Role of Initialization: The starting point (initial weights) can affect whether the optimizer converges to a local minimum or a better solution.
- Global Minimum: While multiple local minima exist, many of them may be "good enough" in terms of performance, and stochastic training methods (like mini-batch gradient descent) can help avoid poor local minima.
- Regularization and Techniques: Methods such as dropout, batch normalization, and learning rate scheduling can influence the minimization process, helping the network generalize better.

Why EDL?

- The surface is infinitely *large*
- The surface is *non-differentiable (depends on the loss and activation functions)*
- The surface is *multi-modal*
- The surface is *complex and noisy*
- The surface is *deceptive*

- *Training dataset is small*

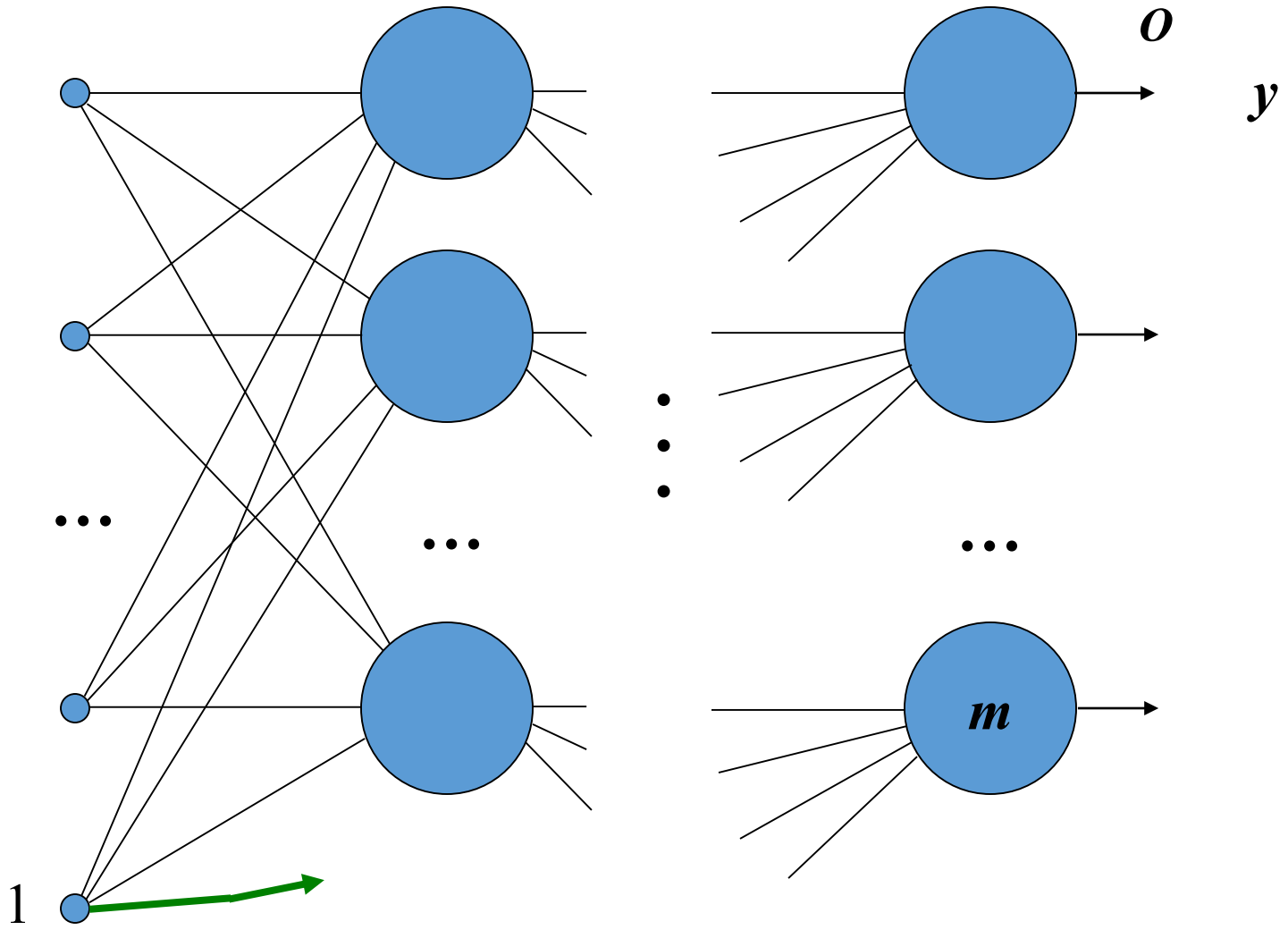
Review: EDL (Evolutionary Deep Learning)

EDL is a broad term encompassing the combination of evolutionary methods with DL. Examples include:

1. Evolutionary Connection Weight (parameter) Optimization (instead of Backpropagation)
2. Evolutionary Hyper-parameter / Architecture Optimization
   (HP, Size/Structure of neural networks)
3. Neuro Evolution of Augmenting Topologies (NEAT)
   Evolving both weights and architecture (1. + 2.). Also topologies
4. AutoML – to automate ML/DL model dev. workflow

Evolving NN Connection Weights

n: # of samples (training patterns)



Objective function for NN Connection Weights when RMS error function is used

$$min\ F(\mathbf{W}) = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{m}\left(d_{ij} - o_{ij}\right)^2}{nm}}, where$$

$\mathbf{W}$: weight vector for the whole NN

d_{ij}: desired output

o_{ij}: network output

n: number of training data

m: number of output neurons of the NN

Performance Metrics

- Percent correct – problem: 0% usually in the beginning of the learning
- Sum-Squared error
- Average Sum-Squared Error
- Mean Absolute Error
- RMS(Root Mean Square) Error, Normalized Error

Objective function for NN Connection Weights, when sum-squared error used

$$min\ F(\mathbf{W}) = \sum_{i=1}^{n}\sum_{j=1}^{m}\left(d_{ij} - o_{ij}\right)^2, where$$

$\mathbf{W}$: weight vector for the whole NN

d_{ij}: desired output

o_{ij}: network output

n: number of training data

m: number of output neurons of the NN

Evolving Complex Behaviors

- Bootstrapping problem: *to initialize the population is difficult*
- **Easy Evolution**: specify primitive behaviors in detail. Leave it to the evolution to optimize how to use them and parameters of them. *For example, HW3. The NN architecture and activation function is fixed.*
- Free Evolution: Do not restrict the types of behaviors that evolution can find

Summary

NN training is an optimization problem to minimize the overall error between desired output and model output for all training samples

Training dataset for a NN with **a single output neuron**

Index i	X_0	X_1	...	X_{n-1}	d (desired)
0					
1					
⋮					
$m-1$					

$$\text{Minimize } F(w_0, w_1, \cdots w_{n-1}, b) = \sum_{i=0}^{m-1} (d_i - y_i)^2, \text{ where}$$

sum squared error

m: # training examples

$$y = f\left(\sum_{i=0}^{n-1} (x_i w_i + b)\right), \text{ where}$$

n: # of variables (features)

NN training is an optimization problem

Training dataset for OR function

i	X_0	X_1	d
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

$$\text{Minimize } F(w_0, w_1, b) = (0 - y_0)^2 + (1 - y_1)^2 + (1 - y_2)^2 + (1 - y_3)^3,$$

where

$$y_0 = f(0 \cdot w_0 + 0 \cdot w_1 + b)$$

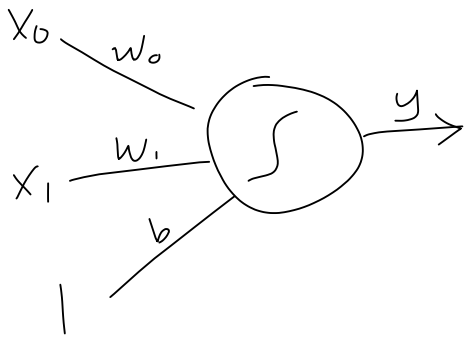
$$y_1 = f(0 \cdot w_0 + 1 \cdot w_1 + b)$$

$$y_2 = f(1 \cdot w_0 + 0 \cdot w_1 + b)$$

$$y_3 = f(1 \cdot w_0 + 1 \cdot w_1 + b)$$

Activation function, sigmoid

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

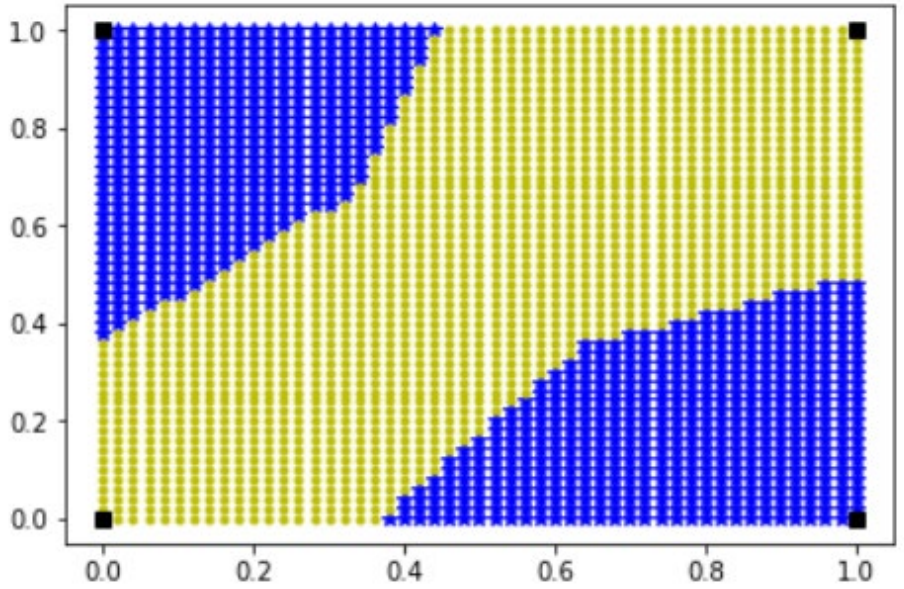


```
static double[,] input = new double[,] { { 0.0, 0.0, 1.0 }, { 0.0, 1.0, 1.0 },
                                         { 1.0, 0.0, 1.0 }, { 1.0, 1.0, 1.0 } };
static double[] desired_out = new double[] { 0.0, 1.0, 1.0, 0.0 };
static double[] hidden = new double[] { 0.0, 0.0 }; //used for storing act_fn(wsum)
                                                    //of each hidden neuron
static double ffp(double[] w, int i) // Feed Forward, i = sample pattern index
{
    int m = 0; // index for 9 weights
    for (int j = 0; j < hidden.Length; j++) // for each hidden neuron
    {
        hidden[j] = 0;
        for (int k = 0; k < 3; k++) // for each weight
            hidden[j] += input[i, k] * w[m++];
        hidden[j] = sigmoid(hidden[j]);
    }
    double ANN_output = 0.0;
    for (int n = 0; n < 2; n++) // for each hidden neuron output
        ANN_output += hidden[n] * w[m++];
    ANN_output += w[m]; // 1.0 * w for the threshold of the output neuron
    ANN_output = sigmoid(ANN_output);
    return ANN_output;
}
```

```
static double objectiveFunc(double[] x)
{
    double errSum = 0;
    double y;
    for (int i = 0; i < 4; i++) // for each input pattern
    {
        y = ffp(x, i);
        errSum += (desired_out[i] - y) * (desired_out[i] - y); // sum squared error
    }
    return errSum;
}
```

HW3 Assignment

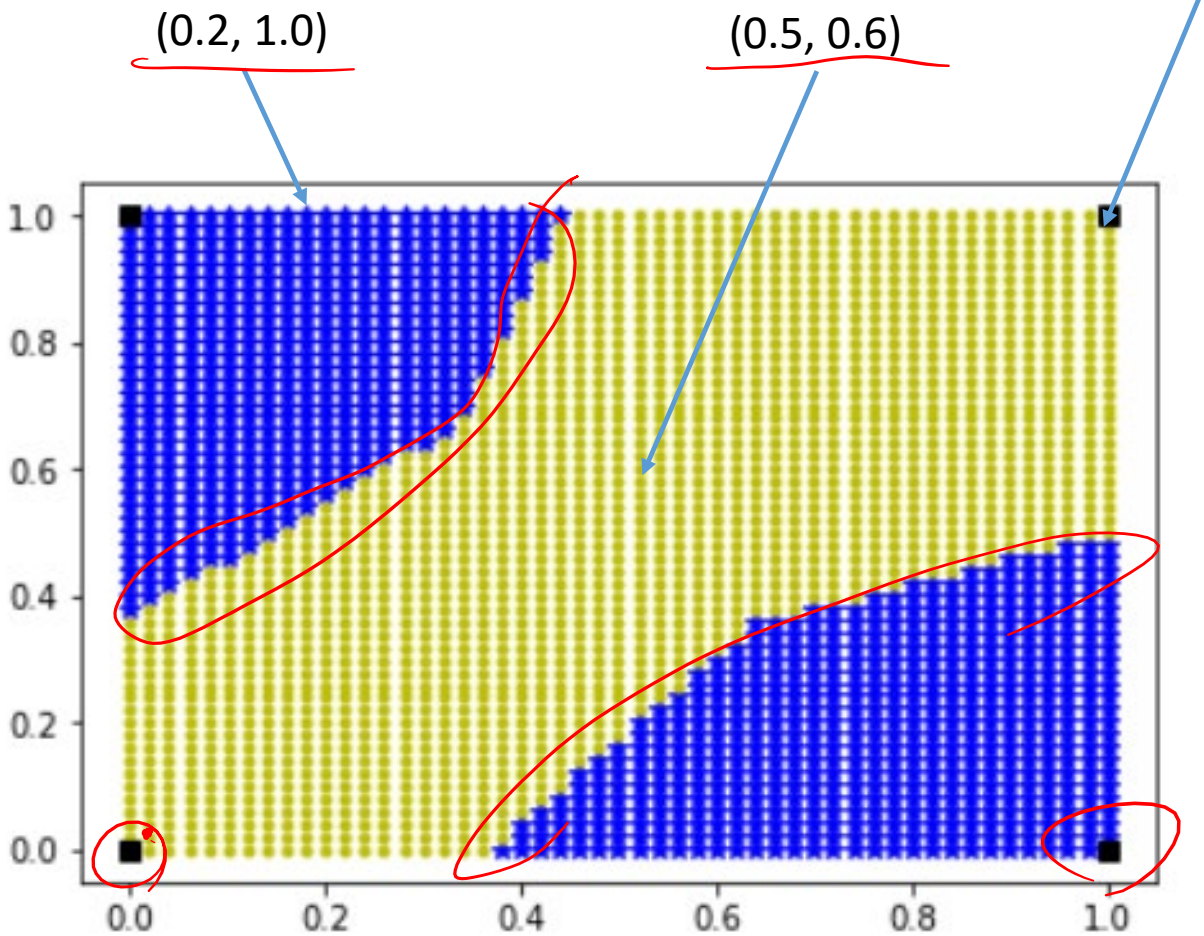
- Write a Python program to train a XOR NN using ES(1+1) with 1/5 rule
- Overfitting/underfitting/right-fit* analysis: Plot the output of the evolved model for each mesh point to show the NN learning as you did in the Deep Learning class



* good-fit, appropriate-fit, optimal-fit, balanced-fit

Self-Ex: Old HW, Plotting XOR learning (1 /2)

- Plot the output of the trained model for each mesh point to show the NN learning
- Results are not consistent mainly due to (1) random weight initialization* and (2) the very small sample dataset size
- Read the rubric on Canvas!



(*) https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks

Old HW, Plotting XOR learning (2 /2)

Code hint. Simplest way. You may directly call `plt.plot()` for each mesh point, though it is slow.

```
for x in np.arange(0.0, 1.02, 0.02):
    for y in np.arange(0.0, 1.02, 0.02):

      if “model prediction with [x,y]” > 0.5: # Use verbose=0 in m.predict()
          plt.plot(x, y, 'b*') # to plot a fired point with blue *
      else:
          plt.plot(x, y, 'y.') # to plot a non-fired point with yellow dot
plt.show()
```

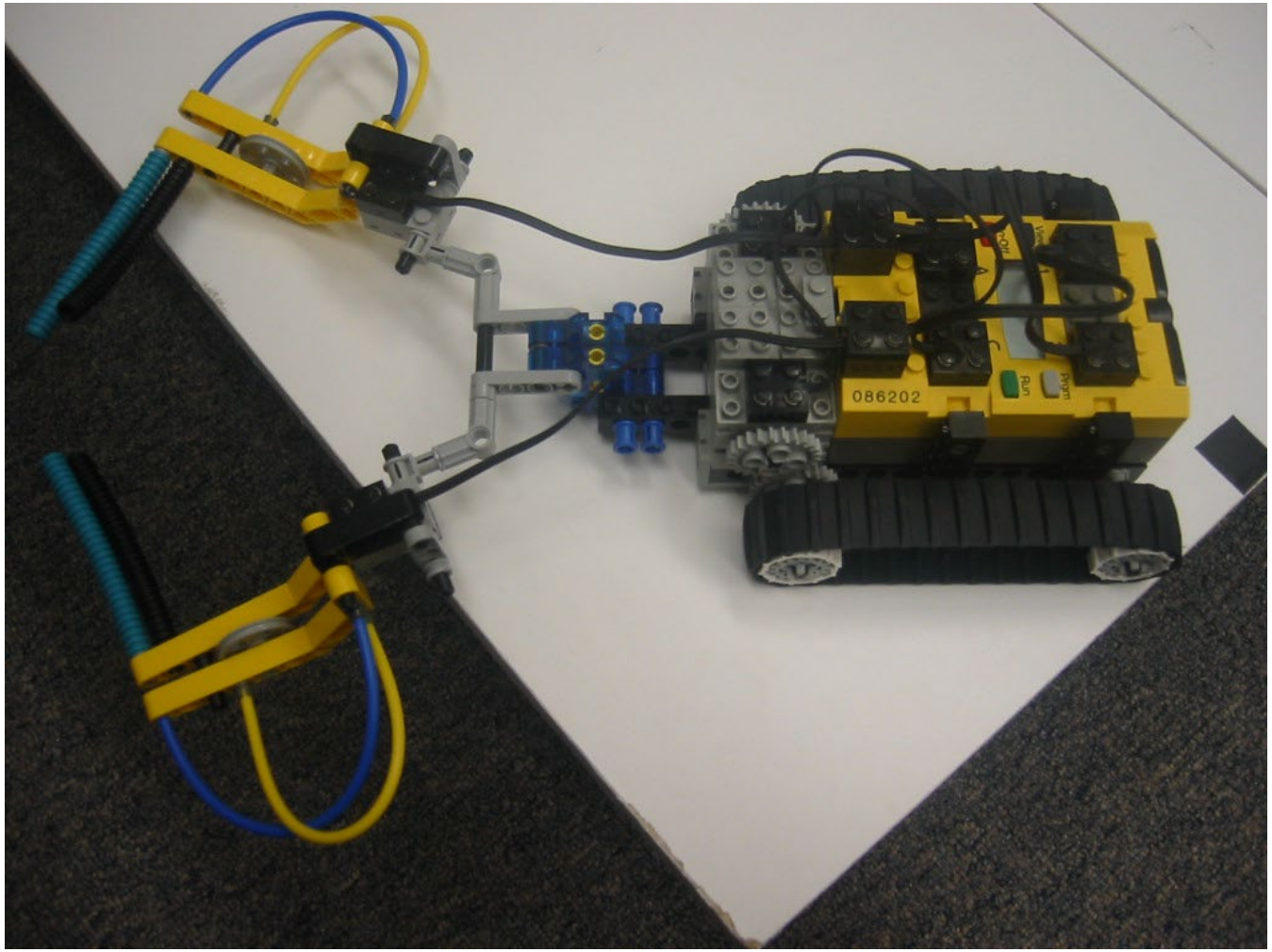
How to plot faster?

Please test and improve the code, **XOR_graph.ipynb** on Canvas. A part of future assignment

Evolutionary Connection Weights Optim Applications

CJ Chung

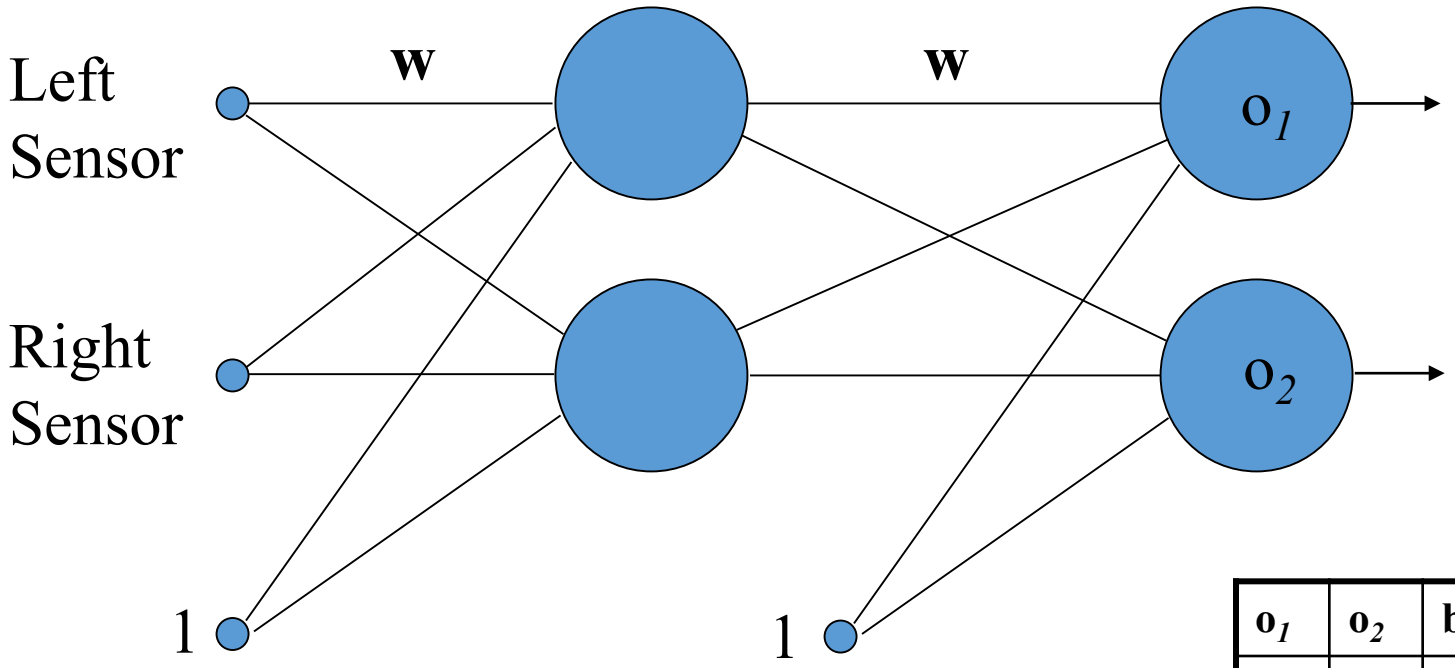
A Lego Robot with two touch sensors
Obstacle avoidance and no-fall (Chung, 2001)



I was one of the pioneers, but no paper…

Left touch sensor is pressed, then backward slightly and turn right

Evolved NN for the Lego Robot



o_1	o_2	**behavior**
1	1	forward
-1	1	Backward and Left
1	-1	Backward and Right
-1	-1	Stop

Training Data and Found Weight Values

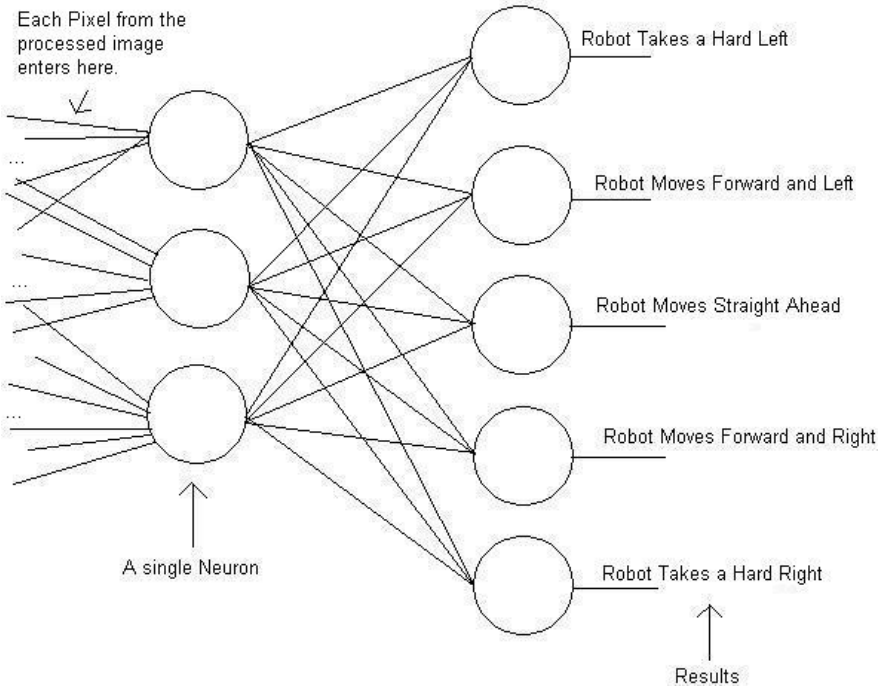
L	R	o_1	o_2	behavior
-1	-1	1	1	Forward
-1	1	-1	1	Backward and Left
1	-1	1	-1	Backward and Right
1	1	-1	-1	Stop

Found 12 weights by ES(1+1) with 1/5 rule:

-22.158, 15.563, 1.409, -7.026, -14.418, -6.110,

-2.782, 45.487, 11.314, 12.196, 1.535, 1.422

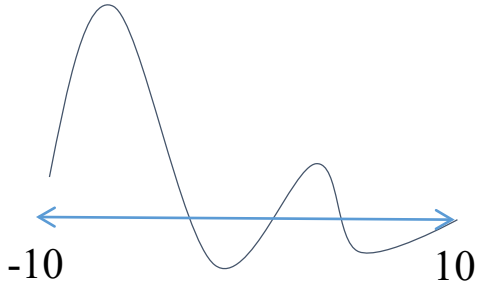
LTU IGVC 2003

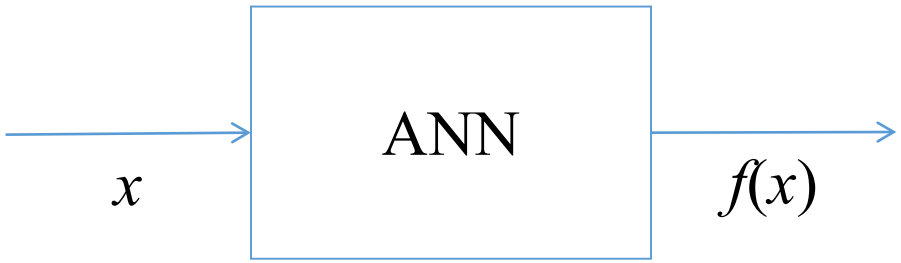


https://www.robofest.net/igvc/DR/CogitoBot2.pdf

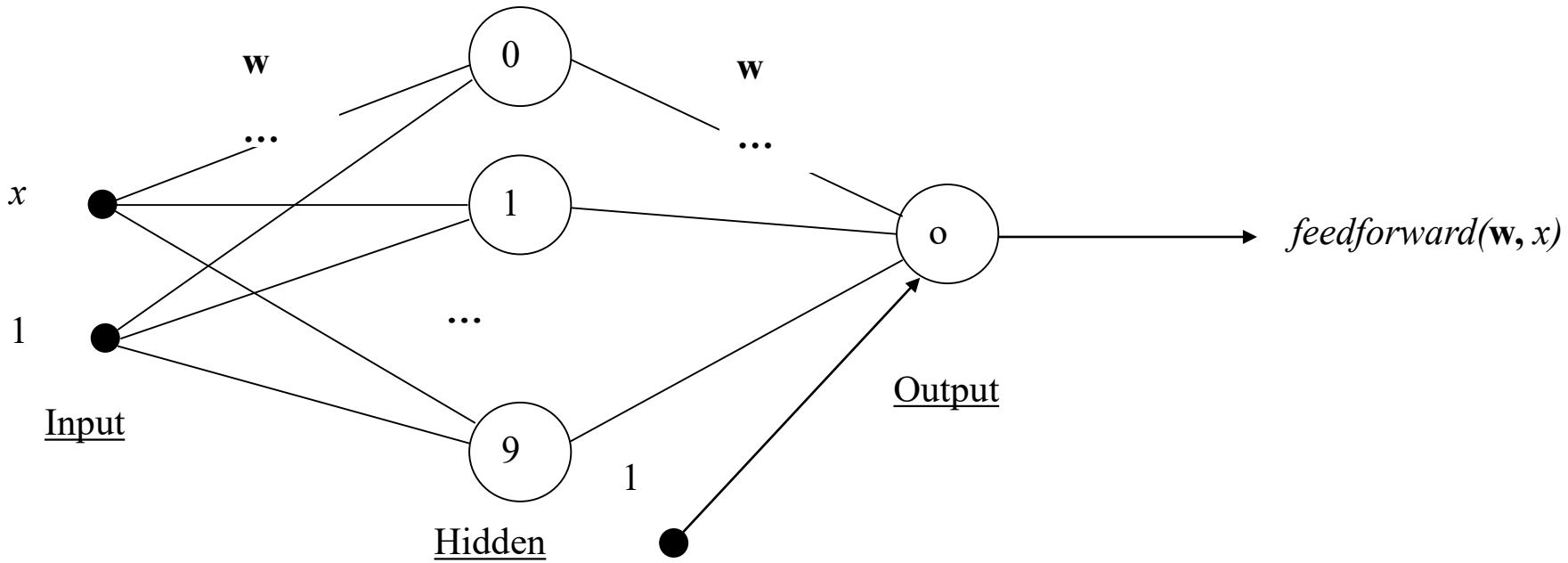
Maurice Tedder, David Chamulak, Li-Ping Chen, Santosh Nair, Andrey Shvartsman, I Tseng, and Chan-Jin Chung, "An Affordable Modular Mobile Robotic Platform with Fuzzy Logic Control and Evolutionary Artificial Neural Networks", The Journal of Robotic Systems, Vol 2, Number 8, Aug. 2004, Editors: Gerardo Beni and Susan Hackwood, pp.419-428

Function Approximation (regression)

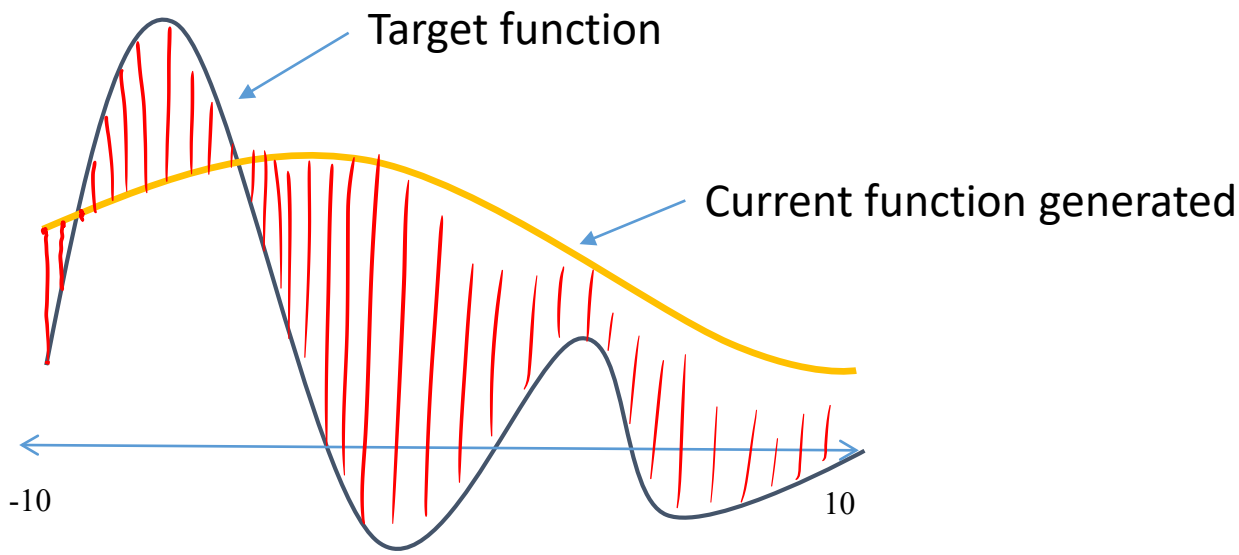




ANN design for the function approximator



Design and implement an evaluation function to be plugged into ES(1+1) with 1/5 rule optimizer – sum squared error



Design an evaluation function (objective function) to be plugged into ES(1+1) with 1/5 rule optimizer

```
double eval(double[] w) // w is a reference to the weight vector
{  double i;
    double sumSqrd = 0.0;
    for(i = -10.0; i < 10.0; i=i+0.01)
        sumSqrd += (f(i) – feedforward(w,i))*(f(i)-feedforward(w,i));
    return sumSqrd;
}
```

Previous assignment as an example (1/3)

- Train an ANN using ES(1+1) with 1/5 rule
- Training Data Set
  - The 9 sample patterns for V (**1**) or H (**0**)
  - Each sample has 6 values

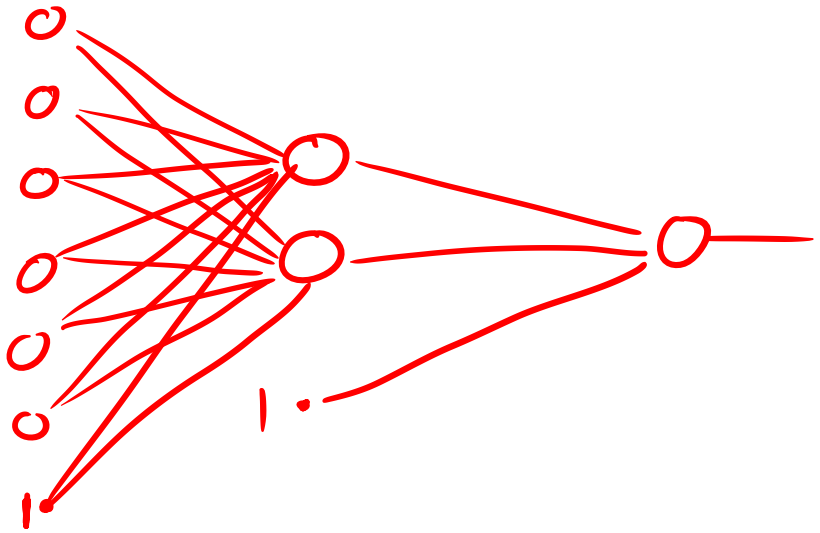
2.5	2.5	2.5	1.5	2.5	0.5	**1**
1.5	2.5	1.5	1.5	1.5	0.5	**1**
0.5	2.5	0.5	1.5	0.5	0.5	**1**
2.5	2.5	1.5	2.5	0.5	2.5	**0**
2.5	1.5	1.5	1.5	0.5	1.5	**0**
2.5	0.5	1.5	0.5	0.5	0.5	**0**
0.5	2.5	1.5	2.5	2.5	2.5	**0**
0.5	1.5	1.5	1.5	2.5	1.5	**0**
0.5	0.5	1.5	0.5	2.5	0.5	**0**

Previous Assignment as an example (2/3)

- When an acceptable solution vector is found, terminate the evolution
- Read another file to test
- Display the output of trained ANN for each pattern in the test data file
- Display the percent success of the system

Previous assignment as an example (3/3)

- Draw the NN architecture. How many output neurons? See next slides
- How many weight variables to optimize?
- What if no acceptable solution is found…
  - Adjust number of hidden neurons
  - Change the low..high for the uniform random number generator
  - Increase # of max generation (?)
  - If there is no hope in the beginning of evolution, restart



$(6+1)\cdot 2 + (2+1)\cdot 1 = 14+3 = 17$