CS 678 Machine Learning
Programming Assignment #3

Evolving “Smarter” Game Characters using Neural Networks

Specification

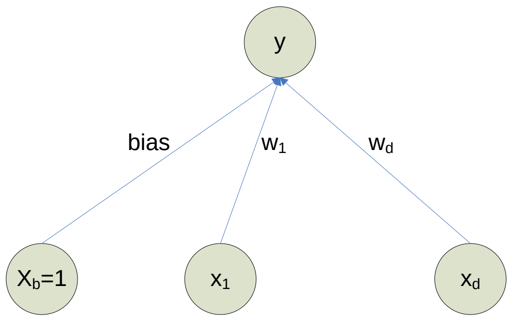
The goal is to create a neural network, i.e. a multilayered perceptron that provides basic intelligence and decision-making ability to a game character. A collection of training data (consisting of environment/action pairs) is used to train the network to respond appropriately to sets of environmental stimuli. The network should then be able to generalize its behavior such that it can direct a game character to respond appropriately, even to situations it has not encountered before.

Perceptron

The basic processing element of a neural network is the perceptron. It is composed of inputs: $x_b=1$, $x_1..x_d$; connection weights: bias, $w_1..w_d$; and the output function σ.

$$\sigma = \sum_{j=1}^{d} w_j x_j + bias$$

A multiple-input perceptron is represented as in the figure below:



The perceptron can be used as a regression or classification operator by using it to implement a linear discriminant (threshold) function, as in:

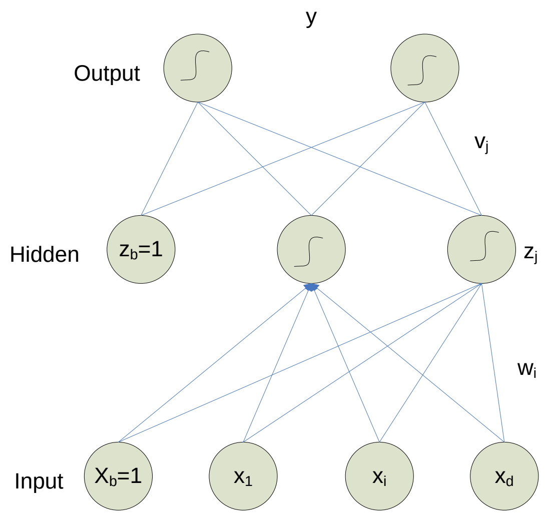
$$y = \begin{cases} 1, if \sigma > 0 \\ 0, otherwise \end{cases}$$

This assumes that the classes are linearly separable, representing linear functions. In cases where a non-linear function is required, one whose outputs are a smoothly differentiable function of its inputs, the *sigmoid* (squashing) function can be used:

$$y = sigmoid(\sigma) = \frac{1}{1+e^{-\sigma}}$$

Multi-layer Perceptrons

The multi-layer feedforward network includes an additional intermediate or *hidden* layer between the input and output layers (see figure below). It is used in conjunction with the sigmoid function to allow the network to approximate non-linear functions. There may be multiple hidden layers, but often there is simply one hidden layer with multiple perceptrons (the number of which may have to be experimentally determined). Generally, each perceptron is connected to every perceptron in the "next" layer. Note that the network may include a *bias* input at the hidden layer.



Training the Network

Whether a single perceptron, or a multi-layer network, the training process consists of presenting examples to the system (as in all forms of Supervised learning) and adjusting the weights until the desired outcome is obtained. A process called gradient descent is used to implement the learning, converging iteratively via the form:

$$\text{Update} = \text{LearningFactor} \cdot (\text{DesiredOutput} - \text{ActualOutput}) \cdot \text{Input}$$

The notion of *training error* (distance from the target value *t*) is used to determine the direction of steepest descent. Training example (*d*) error is commonly defined as:

$$E \equiv \tfrac{1}{2} \sum_{d \in D} (t_d - y_d)^2$$

The error *E* is computed after all input values have been fed forward. Then each weight is altered proportionally for each edge, as in:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

where η is the learning rate (a convergence factor).

Error Backpropagation

Since there is no target value for perceptrons in the hidden layer error is propagated backwards from the output layer and used for training:

For each training example do:

1. Propagate the input forward through the network
    a. Input the instance and calculate the output of each unit

2. Propagate the errors backward through the network
    a. For each output unit *j*, calculate the error

        $E_j = (t_j - y_j) y_j (1 - y_j)$ *// note derivative of sigmoid func*

    b. For each hidden unit *i*, calculate the error

        $E_i = h_i (1 - h_i) \sum_k w_{ik} E_k$

    c. Update each network weight proportionally

        $w_j = w_j + \eta E_j z_j$ *// connecting hidden to output*

        $w_i = w_i + \eta E_i x_i$ *// connecting input to hidden*

Datasets

Two dataset spreadsheets are posted on the course info page:

- Windsurf: a classic; the same dataset used in the decision tree and naïve Bayes tutorials.
- Game: a small set of actions that a game AI should take, given a particular set of environmental conditions.

The data can be preprocessed, modified and normalized as desired.

Requirements:

Write modular code that implements perceptrons and their connections in a network, calculates the (sigmoid) output function, calculates error, backpropagates error, and adjusts weights appropriately. Train your neural network using the provided training data, and test it using the provided test data.

As usual, your program can be written in any language. Submit a written analysis of your project and be prepared to present your solution to the class:

- □ Describe any interesting experiments, configuration details, problems, etc.
- □ Demonstrate the effectiveness of your neural network.
- □ Include a discussion/analysis of your results.