Investigation into the inner workings of Concept Learning and Decision Trees

Adriaan Louw (53031377)

May 30, 2018

Contents

1 Background

2 Perceptrons

2.1 What is a Perceptron

A perceptrons is a unit that takes in inputs x_i and weights ω_i and returns either -1 or 1 depending on whether the dot product of x_i and ω_i is larger than some k.

$$o(x_i, ..., x_n) = \begin{cases} 1 & \omega_1 x_1 + ... + \omega_n x_n > k \\ -1 & otherwise \end{cases} \tag{1}$$

Neural networks can be built up our of perceptrons. These form a layer of nodes in the neural network and can be used to map various boolean functions

2.2 Perceptron learning algorithm

Gradient decent algorithm can be used to train the neural networks based on perceptrons. There needs to be an error function that defines the training error. The error can be fined as

$$E(\vec{\omega}) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2 \tag{2}$$

where d is a training example in the training set D, t_d is the target output for training example d and o_d is the output of the neural network for training example d.

Training neural networks consist of updating the weights of each of the nodes in the network based upon some training data. As in

$$\vec{\omega_i} \leftarrow \vec{\omega_i} + -\eta \bigtriangledown E(\vec{\omega}) \tag{3}$$

where $\vec{\omega}$ is the set of weights of the neural network, η is the learning rate and $\bigtriangledown E(\vec{\omega})$ is the partial derivative of the Error function. The negative of the derivative is used to minimise the size of the error. In other words, the weights are updated each iteration such that the error becomes smaller until some minimum value is reached.

$$\begin{aligned}
\bigtriangledown E(\vec{\omega}) &= \frac{\delta}{\delta\omega_i}\frac{1}{2}\sum_{d \in D}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d \in D}\frac{\delta}{\delta\omega_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d \in D}2(t_d - o_d)\frac{\delta}{\delta\omega_i}(t_d - o_d) \\
&= \sum_{d \in D}(t_d - o_d)\frac{\delta}{\delta\omega_i}(t_d - o_d)
\end{aligned} \tag{4}$$

Then using Equation 1 as a definition for o_d we get

$$\begin{aligned}
\bigtriangledown E(\vec{\omega}) &= \sum_{d \in D}(t_d - o_d)\frac{\delta}{\delta\omega_i}(t_d - o_d) \\
&= \sum_{d \in D}(t_d - o_d)\frac{\delta}{\delta\omega_i}(t_d - \vec{\omega}.\vec{x_d}) \\
&= -\sum_{d \in D}(t_d - o_d)x_{id}
\end{aligned} \tag{5}$$

Therefore combining equations 3 and 5

$$\vec{\omega_i} \leftarrow \vec{\omega_i} + \sum_{d \in D}(t_d - o_d)x_{id} \tag{6}$$

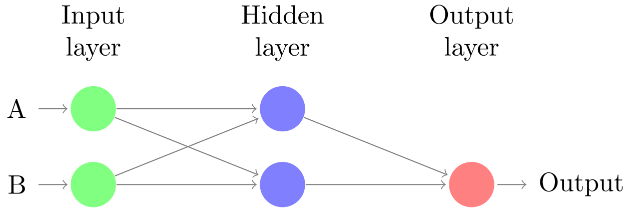
Each weight is updated by using the previous value of the weight plus the learning rate η times the error for each weight and data point.

2.3 Examples

2.3.1 Example 1

Given boolean function $A \wedge B$. This gives can be represented by the truth table

A	B	$A \wedge B$
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1



We will need a network with 2 input nodes and 1 output node.
The treshold

2.4 Limitations of Perceptrons

A 2 layer percepron network can implement any boolean function. But no network of perceptrons can implement continuous functions. This is because of the nature of the perceptron, returning -1 or 1 depending on some threshold.

(Mitchell, 1997)

3 Backpropagation

3.1 Original Paper On Error Propagation

Rumelhart, Hinton, and Williams (1986) is the first paper on error propagation.

3.2 Most common form of Backpropagation

1. Create network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units
2. Initialise all weights to a small random number
3. Until termination condition is reached
   (a) For each $\langle \vec{x}, \vec{t} \rangle$ training example
      i. Input instance $\vec{x}$ into network
      ii. Compute output o_u of every unit u in network.
      iii. For each output unit k calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t - o_k) \tag{7}$$

      iv. For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} \omega_{kh} \delta_k \tag{8}$$

v. Update each network weight ω_{ji}

$$\omega_{ij} \leftarrow \omega_{ij} + \eta \delta_j x_{ji} \tag{9}$$

From (Mitchell, 1997, p98)

3.3 Variants and extentions to Backpropagation

3.3.1 Adding Momentum

It is possible to add a momentum term when updating the weights. For example from equation 9 we get

$$\omega_{ij}(n) \leftarrow \omega_{ij}(n-1) + \eta \delta_j x_{ji} + \alpha \bigtriangledown \omega_{ij}(n-1) \tag{10}$$

Here each update is dependant on the previous update. This can help the network to go over local minima or to continue moving over flat regions. (Mitchell, 1997, p100)

3.3.2 Simulated Annealing

In this variation, the learning rate is gradually decreased over time. (Nils J Nilsson, 1998)

3.4 Question 3.4

3.5 Question 3.5

For the play tennis example. We will use 6 input nodes $(x_1, ..., x_6)$. x_1 = Sky, x_2 = AirTemp, x_3 = Humidity, x_4 = Wind, x_5 = Water and x_6 = Forecast. There will be 6 hidden nodes $(h_1, ..., h_6)$ and one output node. We additionally have a training rate $\eta = 0.1$.

We need to represent these boolean states ad real numbers. Therefore the first value will be 0.1 and the second 0.9.

Thus for Sky: Sunny = 0.1 and Rainy = 0.9

For AirTemp: Warm = 0.1 and Cold = 0.9

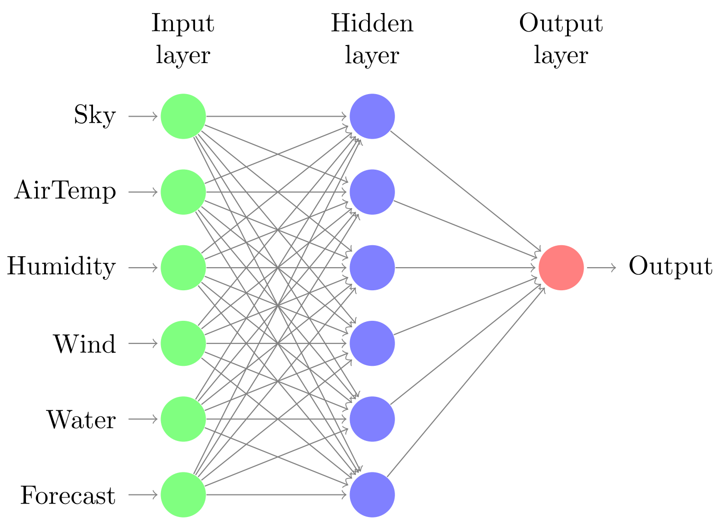
For Humidity: Normal = 0.1 and High = 0.9

For Wind: Strong = 0.1 and Weak = 0.9

For Water: Warm = 0.1 and Cool = 0.9

For Forecast: Same = 0.1 and Change = 0.9

Output layer will return 0.1 for No and 0.9 for Yes.



From (Mitchell, 1997, p101) we have the error

$$E(\vec{\omega}) = \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2 \tag{11}$$

and from (Mitchell, 1997, p97)

$$o = \frac{1}{1 + e^{-\vec{\omega}.\vec{x}}} \tag{12}$$

References

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Nils J Nilsson. (1998). *Introduction to Machine Learning*. Retrieved from `http://robotics.stanford.edu/people/nilsson/MLBOOK.pdf`

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. Retrieved from `http://www.cs.toronto.edu/ hinton/absps/naturebp.pdf` doi: 10.1038/323533a0