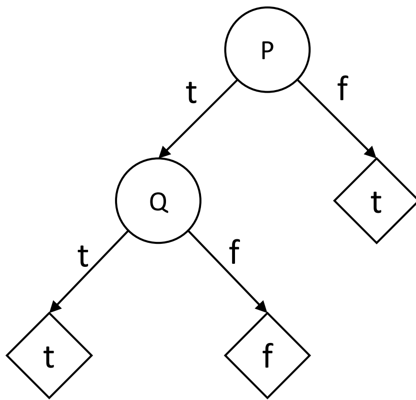
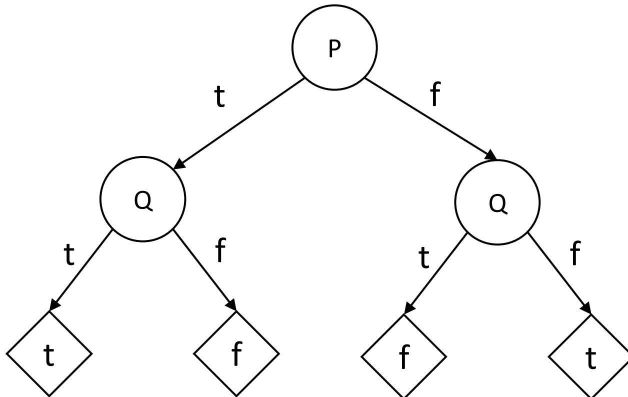
Neural Network Learning– Solution

1) Can a decision tree represent the Boolean function $f(P,Q) \equiv P \Rightarrow Q$? What about a single perceptron with a step function (whose step is at zero) as threshold function? What about a network of perceptrons with step functions (whose steps are at zero) as threshold functions? Now answer the same three questions for the Boolean function $f(P,Q) \equiv P \Leftrightarrow Q$.

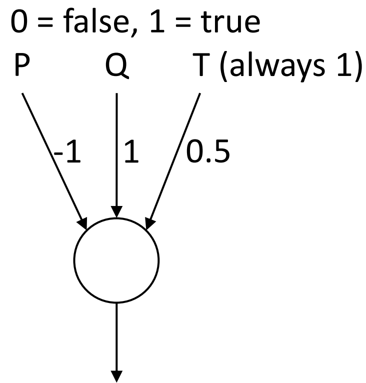
Answer:
A decision tree to represent the Boolean function $f(P,Q) \equiv P \Rightarrow Q$:



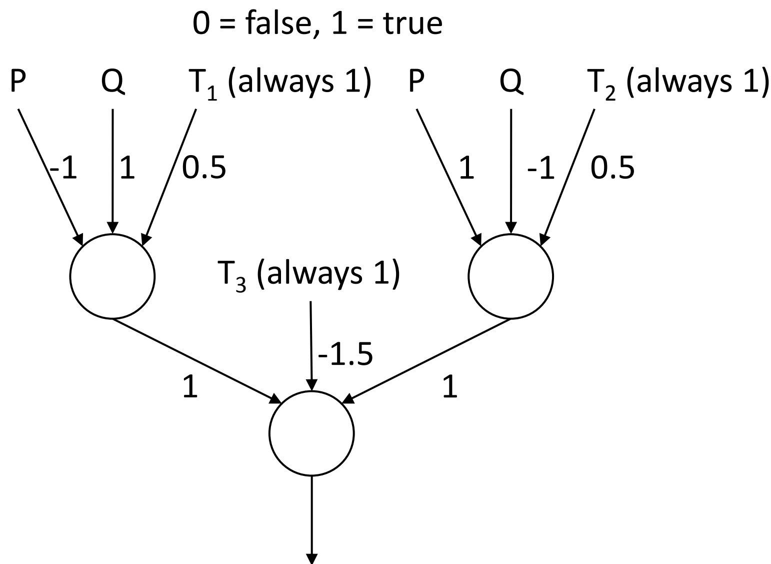
A decision tree that represents the Boolean function $f(P,Q) \equiv P \Leftrightarrow Q$:



A single perceptron (which is trivially a network of perceptrons) with a step function (whose step is at zero) that represents the Boolean function $f(P,Q) \equiv P \Rightarrow Q$:



A network of perceptrons with step functions that represents the Boolean function $f(P, Q) \equiv P \Leftrightarrow Q \equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P)$:



A single perceptron cannot be used to represent the Boolean function $f(P, Q) \equiv P \Leftrightarrow Q$ because the four cases (where P and Q are true or false) is not linearly separable (similar to the XOR example in the Perceptron Learning slides).

2) Develop the training rule for a perceptron with a sigmoid function as threshold function and the sum of errors to the power of four (instead of the sum of squared errors) as error function.

Answer:
Terminology:

- g is the sigmoid function with $g(x) = 1/(1+e^{-x})$ and its derivative $g'(x) = g(x)(1 - (g(x))$.
- c_i is the desired value for the ith training example.
- o_i is the value computed by the perceptron as the value of the ith training example.
- w_j is the weight given to the jth feature.

- f_{ij} is the value of the jth feature of the ith training example.

We use the error function $E := 1/4 \sum_i (c_i - o_i)^4 = 1/4 \sum_i (c_i - g(\sum_j w_j f_{ij}))^4$

$dE/dw_j = \sum_i (c_i - o_i)^3 d(c_i - o_i)/dw_j$

$= \sum_i (c_i - o_i)^3 d(c_i - g(\sum_j w_j f_{ij}))/dw_j$

$= \sum_i (c_i - o_i)^3 (-g(\sum_j w_j f_{ij}))d(\sum_j w_j f_{ij})/dw_j$

$= \sum_i (c_i - o_i)^3 (-f_{ij} g'(\sum_j w_j f_{ij}))$

Thus, weight w_j gets updated by gradient descent as follows (batch update), where α is a small learning rate:

$w_j := w_j - \alpha dE/dw_j = w_j + \alpha \sum_i f_{ij}(c_i - o_i)^3 g'(\sum_j w_j f_{ij})$

If we want to update the weights one training example at a time, we use the following (for the ith example):

$w_j := w_j + \alpha f_{ij}(c_i - o_i)^3 g'(\sum_j w_j f_{ij})$

In this case, we need to cycle through all examples.

3) (Courtesy of Russell and Norvig) Suppose that a training set contains only a single example, repeated 100 times. In 80 of the 100 cases, the single output value is 1; in the other 20, it is 0. What will a neural network predict for this example, assuming that it has been trained on all training examples and reaches a global optimum? (Hint: To find the global optimum, differentiate the error function and set the resulting expression to zero.)

Answer:
We determine the weights that minimize the error function. The error function is

$E = 1/2 \sum_i (c_i - o_i)^2$

$= 1/2(80(1 - o)^2 + 20(0 - o)^2)$

$= 50 Output^2 - 80 Output + 40$

The derivative of the error with respect to the output is

$dE/dOutput = 100 Output - 80$

Setting the derivative to zero, we find that $Output = 0.8$, the probabilistic prediction $P(Output = 1)$.

4) If we train a neural network for 1,000 epochs (one training example at a time), does it make a difference whether we present all training examples in turn for 1000 times or whether we first present the first training example 1000 times, then the second training example for 1000 times, and so on? Why?

Answer:
The resulting networks would be different. The error used to adjust the weights of the network during training is based on the training examples given to it during that epoch. In the case of all training examples presented in turn 1000

times, the error is based on the error over all examples because they are all present during each epoch. In the case where each training example is presented individually for 1000 times, one first reduces the error only with respect to the first training example, then only with respect to the second training example, and so on. The result is that the network will "forget" the earlier training examples. As later training examples are presented, the weights will move away from the weights necessary to classify the first training examples correctly.

5) You are given n numbers and have to determine their mean. Develop a gradient descent rule for this purpose.

Answer:
Let $x_1, \ldots, x_n$ denote the given numbers, $\bar{x}$ denote their mean and $\tilde{x}$ denote our current estimate of their mean. A possible error function is:

$Error(\tilde{x}) = 1/2(\sum_i (\tilde{x} - x_i))^2$.

This error function is non-negative and zero exactly at the mean (that is, when $\tilde{x} = \bar{x}$). It has exactly one global minimum, namely at the mean.

Update equation:

$$\tilde{x} := \tilde{x} - \alpha \frac{dError(\tilde{x})}{d\tilde{x}} = \tilde{x} - \alpha \frac{1}{2} 2n \sum_i (\tilde{x} - x_i) = \tilde{x} + \alpha n \sum_i (x_i - \tilde{x})$$

We can use this update equation in two different ways.

Schema 1 (batch update):

$\tilde{x} :=$ random number

Repeatedly do:

$\tilde{x} := \tilde{x} + \alpha n \sum_i (x_i - \tilde{x})$

Below is sample code to determine the mean of 2, 3 and 7 using Schema 1:

```
#include <stdio.h>

main()
{
    int i;
    double mean = 10.0;
    double alpha = 0.0001;
    double alpha_n = alpha * 3;

    for (i = 0; i < 100000; ++i)
    {
        mean = mean + alpha_n * (
            (2.0 - mean) +
            (3.0 - mean) +
```

```
            (7.0 - mean));
        printf("%f\n", mean);
    }
}
```

Schema 2 (one training example at a time):

$\tilde{x} :=$ random number

Repeatedly do:

$\tilde{x} := \tilde{x} + \alpha(x_1 - \tilde{x})$

$\tilde{x} := \tilde{x} + \alpha(x_2 - \tilde{x})$

$\tilde{x} := \tilde{x} + \alpha(x_3 - \tilde{x})$

...

Below is a sample code to determine the mean of 2, 3 and 7, using Schema 2:

```
#include <stdio.h>

main()
{
    int i;
    double mean = 10.0;
    double alpha = 0.0001;

    for (i = 0; i < 100000; ++i)
    {
        mean = mean + alpha * (2.0 - mean);
        printf("%f\n", mean);
        mean = mean + alpha * (3.0 - mean);
        printf("%f\n", mean);
        mean = mean + alpha * (7.0 - mean);
        printf("%f\n", mean);
    }
}
```

6) Explain exactly why networks of perceptrons with linear activation functions are uninteresting (that is, networks of perceptrons where, for each perceptron, the output is some constant times the weighted sum of the inputs). Use equations if necessary.

Answer:
Networks of perceptrons with linear activation functions are uninteresting because a single perceptron can be used to represent a whole network of perceptrons with linear activation functions (since a weighted sum of weighted sums of feature values can be calculated simply as a weighted sum of feature values).

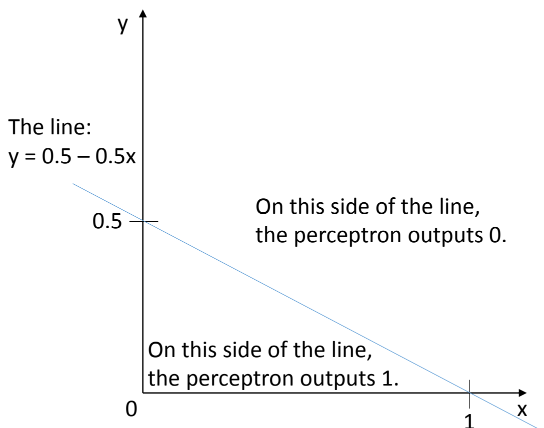
7) Does it make sense to use an inductive machine learning method if it cannot even represent all training examples correctly?

Answer:
Yes. Machine learning methods can have small errors on both training and test sets even if they cannot represent all training examples correctly and thus cannot have zero errors. In fact, it is often the case that naive Bayesian classifiers, perceptrons and other machine learning formalisms can represent the function to be learned only "approximately".

8) Is overfitting more or less likely when the training set is small or large? Is overfitting more or less likely when the number of parameters to learn (such as the number of weights in a neural network) is small or large?

Answer:
Overfitting is more likely when the training set is small and the number of parameters is large. The larger the number of parameters to learn, the more training examples are needed to estimate them well.

9) Design a perceptron with inputs x and y and a threshold function as activation function whose output for a given input pair (x, y) is given by:



Clearly specify the weights and threshold of your perceptron. Show your reasoning.

Answer:
We start by rearranging the terms in the formula of the line:

$y = 0.5 - 0.5x$

$0.5x + y = 0.5$

$x + 2y = 1$ (only for cosmetic reasons)

We then find the inequality for the half-plane that contains exactly the points for which the perceptron outputs 1. There are two possibilities:

$x + 2y > 1$

$x + 2y < 1$

We can easily determine which one is the correct one by selecting a point for which the perceptron outputs 1, and see which inequality it satisfies. In this instance, we can pick (0,0), which satisfies the second inequality. Therefore, we want our perceptron to output 1 when $x+2y < 1$. To determine the weights and the thresholds, we simply rearrange the inequality to be similar to the threshold function:

$x + 2y < 1$

$-x - 2y > -1$

So, the weight for x is -1, the weight for y is -2, and the threshold is -1. Note that, one can multiply both weights and the threshold with the same positive constant, and the perceptron continues to represent the same function.

10) You are given the function $f(x, y) = x^2 + xy + y^2$ and are trying to find a local **minimum** using gradient descent. You randomly start with $x = 1.3$ and $y = 5.4$. Perform the first step of gradient descent with learning rate $\alpha = 0.01$. Show the resulting values for x and y as well as all of your calculations.

Answer:
We first take the derivative of f with respect to x, and with respect to y:

$df(x, y)/dx = 2x + y$

$df(x, y)/dy = x + 2y$

We then write down the update rule for gradient descent:

$x := x - \alpha f(x, y)/dx = x - \alpha(2x + y)$

$y := y - \alpha f(x, y)/dy = y - \alpha(x + 2y)$

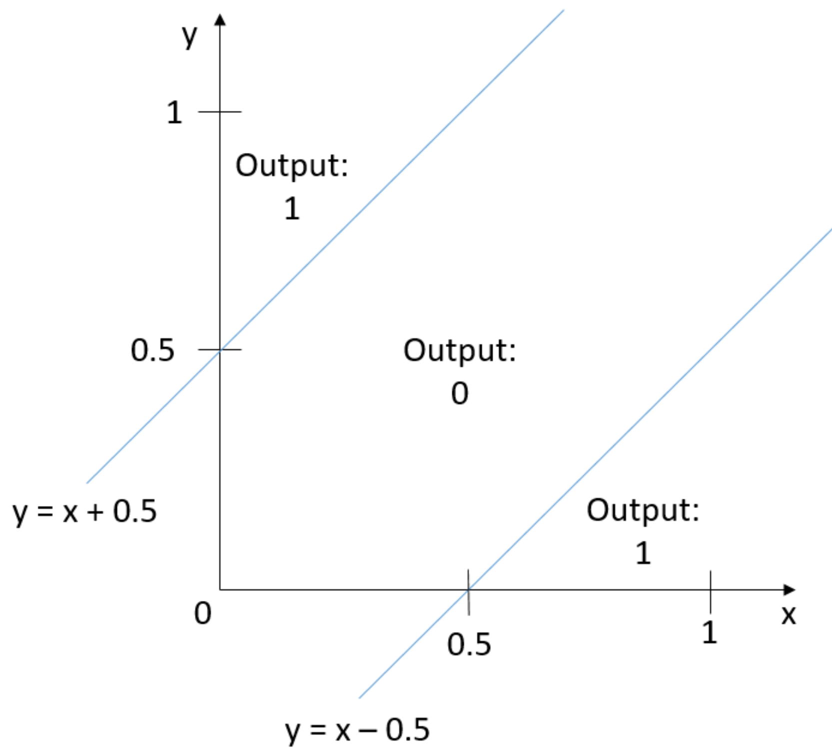
Plugging in the values for x, y, and α, we get the new values for x and y after one iteration of gradient descent:

$x := 1.3 - 0.01(1.3 \times 2 + 5.4) = 1.22$

$y := 5.4 - 0.01(1.3 + 5.4 \times 2) = 5.279$

11) The slides on "Neural Networks" show that a neural network consisting of three perceptrons can represent x XOR y for two inputs x and y. Show in an x-y coordinate system for which real-valued(!) inputs x and y this particular neural network outputs 1 and for which real-valued inputs x and y it outputs 0, assuming that it uses a threshold function as activation function (rather than a sigmoid function).

Answer:


y
1
Output:
1
0.5
Output:
0
y = x + 0.5
Output:
1
0
0.5
1
x
y = x – 0.5