

DM534 - Introduction to Computer Science

Training Session, Week 41-43, Autumn 2016

Solution

Included.

Exercise 1. k -Nearest Neighbors: Prediction

(Based on slide 21)

Suppose you are trying to predict a response y to an input x and that you are given the set of training data $D = [(x_1, y_1), \dots, (x_{11}, y_{11})]$ reported and plotted in Figure 1.

$$D = \begin{bmatrix} (8, & 8.31) \\ (14, & 5.56) \\ (0, & 12.1) \\ (6, & 7.94) \\ (3, & 10.09) \\ (2, & 9.89) \\ (4, & 9.52) \\ (7, & 7.77) \\ (8, & 7.51) \\ (11, & 8.0) \\ (8, & 10.59) \end{bmatrix}$$

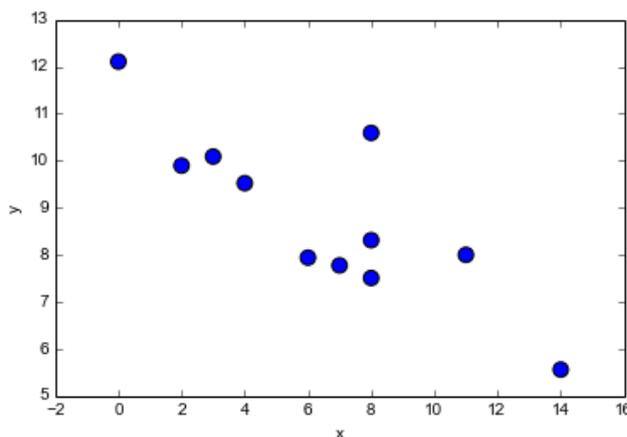


Figure 1: The data for Exercise 1.

Using 5-nearest neighbors, what would be the prediction on a new input $x = 8$?

Solution

First, we need to determine the set $N_5(x)$ of points from D with the 5 shortest distances from x . Hence, we calculate the distance from x to each point in D . For example, the distance between x and \vec{x}_1 is:

$$d(\vec{x}, \vec{x}_1) = \sqrt{(8 - 8)^2} = 0$$

Then, once we have calculated all Euclidean distances for x we rank them in increasing order and take the 5 data points whose corresponding distances are the shortest. Here, we can carry out this process more easily by inspection of the given plot and conclude that:

$$N_5(x) = \begin{bmatrix} (8, & 8.31) \\ (6, & 7.94) \\ (7, & 7.77) \\ (8, & 7.51) \\ (8, & 10.59) \end{bmatrix}$$

Then the prediction \hat{y} can be calculated as:

$$\hat{y}(x) = \frac{1}{k} \sum_{i|x_i \in N_k(x)} y_i = \frac{1}{5} (8.31 + 7.94 + 7.77 + 7.51 + 10.59) = 8.424$$

What form of learning is this exercise about?

- Supervised learning, regression
- Supervised learning, classification
- Unsupervised learning
- Reinforcement learning

Solution

Supervised learning, regression

Exercise 2. k -Nearest Neighbors: Prediction

(Based on slide 21)

Suppose you are trying to predict the class $y \in \{0, 1\}$ of an input (x_1, x_2) and that you are given the set of training data $D = [((x_1, x_2), y_1), \dots, ((x_{11,1}, x_{11,2}), y_{11})]$ reported and plotted in Figure 2.

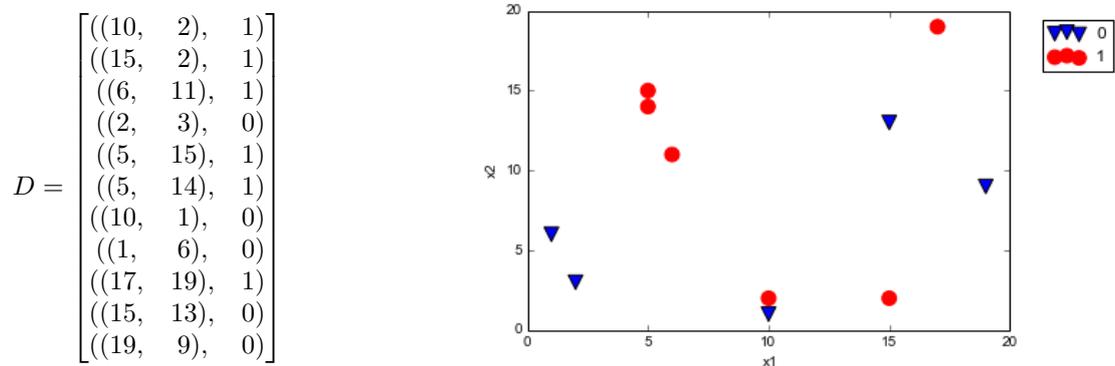


Figure 2: The data for Exercise 2.

Using the 5-nearest neighbors method, what would be the prediction on the new input $\vec{x} = (5, 10)$?

Solution

First, we need to determine the set $N_5(\vec{x})$ of points from D with the 5 shortest distances from \vec{x} . We choose to use the Euclidean distance. Hence, we calculate the Euclidean distance from \vec{x} to each point in D . For example the distance between \vec{x} and \vec{x}_1 is:

$$d(\vec{x}, \vec{x}_1) = \sqrt{(5 - 10)^2 + (10 - 2)^2} \approx 9.44$$

Then, once we have calculated all Euclidean distances for x we rank them in increasing order and take the 5 data points whose corresponding distances are the shortest. Here, we can carry out this process more easily by inspection of the given plot and conclude that:

$$N_5(\vec{x}) = \begin{bmatrix} ((6, 11), 1) \\ ((2, 3), 0) \\ ((5, 15), 1) \\ ((5, 14), 1) \\ ((1, 6), 0) \end{bmatrix}$$

Then, the prediction \hat{y} is given by majority vote. Since 3 points in $N_5(x)$ have $y = 1$ and 2 points have $y = 0$ then 1 wins and $\hat{y} = 1$.

What form of learning is this exercise about?

- Supervised learning, regression
- Supervised learning, classification
- Unsupervised learning
- Reinforcement learning

Solution

Supervised learning, classification

Exercise 3. k -Nearest Neighbors: Loss

(Based on slide 21)

Suppose a 7-nearest neighbors regression search returns $\{4, 2, 8, 4, 9, 11, 100\}$ as the 7-nearest y values for a given x value. Consider an hypothesis set made by two functions only: median and average. Let \hat{y} be their prediction for x .

- If the evaluation is done by means of the *absolute value loss*,

$$L_1 = |y - \hat{y}|$$

which of the two functions minimizes the sum of absolute value loss function:

$$\hat{L}_1 = \sum_{j=1}^k |y_j - \hat{y}|$$

on the given data?

- If the evaluation is done by means of the *squared loss error*,

$$L_2 = (y - \hat{y})^2$$

which of the two functions minimizes the sum of squares loss function

$$\hat{L}_2 = \sum_{j=1}^k (y_j - \hat{y})^2$$

on the given data?

- [Advanced] Show that the average is the function that minimizes the loss function L_2 in a k -nearest neighbor method for regression on any data. [Hint: In \hat{L}_2 , the only variable is \hat{y} (y_i 's are given by the input). The value for \hat{y} that minimizes \hat{L}_2 can be found by differentiation in \hat{y} .]

[Note: You can carry out the calculations by hand or you can use any program of your choice.]

Solution

Note: The calculations below are done in Python and Numpy, however this is not a requirement. Calculations can be done hand or any other program. The important thing here is that the numerical results are the same. We calculate \hat{y} for the average and for the median. The median is determined by sorting the y values and selecting the value with half of the data above and half of the data below it. In this case it is the 4th value in the ordered sequence, that is 8. We can carry out the operations in Python. In Python, Numpy is a module that implements arrays. Vectors and matrices can be implemented as objects made of lists and implementations of specific operations. However, Numpy makes us available a new data type, “array”, and the implementation of vectorized operations for this data type.

```
In [1]: import numpy as np
        from __future__ import division

        y = np.array([2,4,4,8,9,11,100])
        y_hat = sum(y)/len(y) # = np.average(y)
        print(y_hat)
```

19.7142857143

```
In [2]: np.average(y)
```

```
Out [2]: 19.714285714285715
```

```
In [3]: np.median(y)
```

```
Out [3]: 8.0
```

```
In [4]: sum( (y-np.average(y))**2 )
```

Out [4]: 7581.4285714285706

In [5]: `sum((y-np.median(y))**2)`

Out [5]: 8542.0

In [6]: `sum(np.absolute(y-np.average(y)))`

Out [6]: 160.57142857142858

In [7]: `sum(np.absolute(y-np.median(y)))`

Out [7]: 110.0

Hence, on this data the average minimizes L_2 and the median minimizes L_1 .

For the general case we need to find the value \hat{y} that minimizes

$$\hat{L}_2 = \sum_{j=1}^k (y_j - \hat{y})^2$$

For a given set of k data, this is a function in one variable, \hat{y} . Since the function is derivable we calculate the derivative in \hat{y} and find the minimum by setting it to zero:

$$\frac{d}{d\hat{y}} \hat{L}_2 = \frac{d}{d\hat{y}} \sum_{j=1}^k (y_j - \hat{y})^2 = 0$$

$$2 \sum_{j=1}^k -(y_j - \hat{y}) = 0$$

$$-2 \sum_{j=1}^k y_j + 2k\hat{y} = 0$$

$$\hat{y} = \frac{1}{k} \sum_{j=1}^k y_j$$

which indeed is the formula of the average value.

Exercise 4. Linear Regression: Prediction

(Based on slides 24-26)

As in [Exercise 1](#), you are trying to predict a response y to an input x and you are given the same set of training data $D = [(x_1, y_1), \dots, (x_{11}, y_{11})]$, also reported and plotted in [Figure 3](#). However, now you want to use a linear regression model to make your prediction. After training, your model looks as follows:

$$g(x) = -0.37x + 11.22$$

The corresponding function is depicted in red in [Figure 3](#). What is your prediction \hat{y} for the new input $x = 8$?

Solution

$$\hat{y} = g(8) = -0.37 \times 8 + 11.22 = 8.26$$

What is the squared error loss if later you find out that the actual response for x is 9?

Solution

From the definition of square error loss:

$$L_2 = (y - \hat{y})^2$$

it follows that L_2 for $(x, y) = (8, 9)$ is $(8.26 - 9)^2 = 0.548$.

$$D = \begin{bmatrix} (8, & 8.31) \\ (14, & 5.56) \\ (0, & 12.1) \\ (6, & 7.94) \\ (3, & 10.09) \\ (2, & 9.89) \\ (4, & 9.52) \\ (7, & 7.77) \\ (8, & 7.51) \\ (11, & 8.0) \\ (8, & 10.59) \end{bmatrix}$$

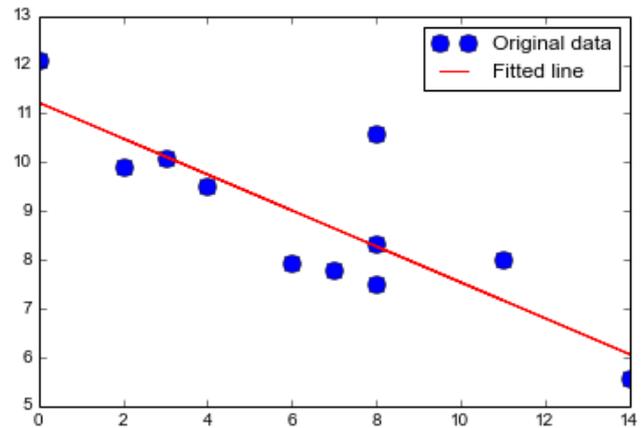


Figure 3: The data for Exercise 4.

Exercise 5. Linear Regression: Training

(Based on Slides 24-26)

Calculate the linear regression line for the set of points:

$$D = \begin{bmatrix} (2, 2) \\ (3, 4) \\ (4, 5) \\ (5, 9) \end{bmatrix}$$

Calculate also the *training error* defined as the sum of squared errors on all data from D .

Plot the points and the regression line on the Cartesian coordinate system.

[You can carry out the calculations by hand or you can use any program of your choice. Similarly, you can draw the plot by hand or get aid from a computer program.]

Solution

We use the equations of slide 26:

$$\bar{x} = \frac{2+3+4+5}{4} = 3.5 \quad \bar{y} = \frac{2+4+5+9}{4} = 5.0$$

and

$$\hat{b} = -2.7 \quad \hat{a} = 2.2$$

We carry out these calculations in Python using the module Numpy.

In [2]: `import numpy as np`

```
m = 4
input = np.array([
    [2, 2],
    [3, 4],
    [4, 5],
    [5, 9]
])
```

Then we can slice the matrix to extract only the x or y coordinates:

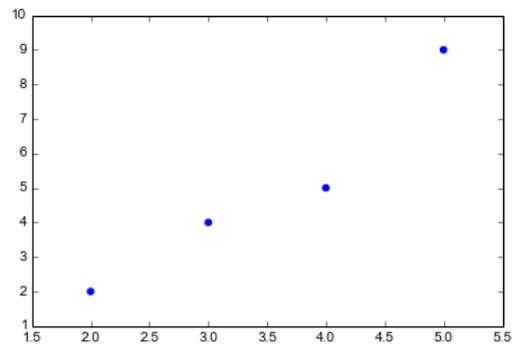
```
In [3]: xx = input[:,0]
        yy = input[:,1]
        xx
```

```
Out[3]: array([2, 3, 4, 5])
```

To plot we import the module pyplot from the 2D plotting library matplotlib

```
In [4]: import matplotlib.pyplot as plt
```

```
plt.figure(1)
plt.scatter(xx, yy, color='b', marker='o')
```



We are then ready to calculate the parameters of the linear regression using the formulas in the slides. We must be careful that the division in Python 2.7 by default returns the integral part. To allow float numbers we import another module:

```
In [5]: from __future__ import division
```

```
x_bar = sum(xx)/m
y_bar = sum(yy)/m

print(x_bar)
print(y_bar)
```

```
3.5
5.0
```

The calculations can be performed in vectorized form, that is, working with arrays. `sum()` sums over the elements of an array and `**` makes the elements wise square of the elements of an array. Hence:

```
In [6]: a_hat = sum((xx-x_bar)*(yy-y_bar))/sum((xx-x_bar)**2)
b_hat = y_bar - a_hat * x_bar

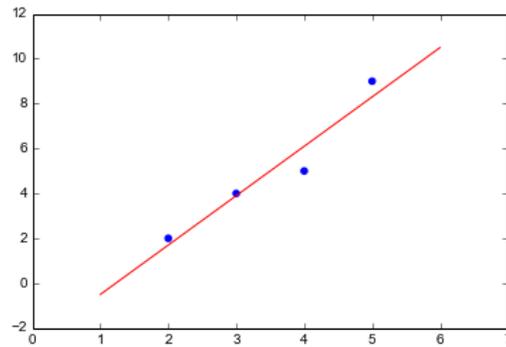
print(a_hat)
print(b_hat)
```

```
2.2
-2.7
```

Finally, we can plot the line. In `pytplot` the easiest way to plot a line is by giving two points and plotting the segment between them. We generate the x coordinates and then calculate the corresponding y value:

```
In [7]: x12 = np.linspace(1, 6, 2)
y12 = np.array(b_hat + a_hat * x12)

plt.figure(1)
plt.scatter(xx, yy, color='b', marker='o')
plt.plot(x12, y12.T, color='r')
plt.show()
```



To calculate the sum of squared errors for the training data we need to first calculate the predictions \hat{y} of the linear model on each point of the training set. This can be done as follows:

```
In [8]: def g(var):
        return (b_hat + a_hat * var)
        vec_g = np.vectorize(g)
        y_hat = g(xx)
        L_hat = sum((yy-y_hat)**2)
        print L_hat
```

1.8

Exercise 6. Multilayer Perceptrons

(Based on Slides 66, 67.)

Determine the Boolean Function represented by the perceptron in Figure 4:

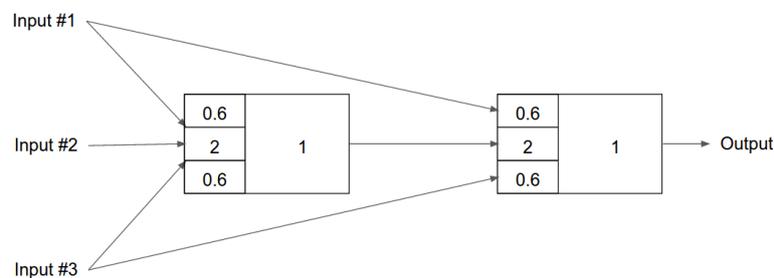


Figure 4: . The multilayer perceptron of Exercise 6.

Solution

x_1	x_2	x_3	y
0	0	0	0
0	0	1	0
0	1	0	1
1	0	0	0
0	1	1	1
1	1	0	1
1	0	1	1
1	1	1	1

Exercise 7. Feed-Forward Neural Networks: Single Layer Perceptron

(Based on Slides 49-54)

Determine the parameters of a single perceptron (that is, a neuron with step function) that implements the majority function: for n binary inputs the function outputs a 1 only if more than half of its inputs are 1.

Solution

Set all input weights to 1 and the threshold (bias) to $n/2$.

Exercise 8. Single Layer Neural Networks: Prediction

(Based on Slides 49-63.)

In [Exercise 2](#), we predicted the class $y \in \{0, 1\}$ of an input (x_1, x_2) with the 5-nearest neighbors method using the data from set D . We used those data to train a single layer neural network for the same task. The result is depicted in [Figure 5](#).

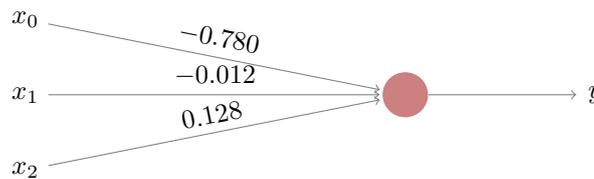


Figure 5: A single layer neural network for the task of [Exercise 8](#).

- Calculate the prediction of the neural network for the new input $\vec{x} = (5, 10)$. Assume a step function as activation function in the unit, which is therefore a perceptron.

Solution

The step function implemented by the neuron returns $\vec{\theta} \cdot \vec{x} > 0$. In our case:

$$\vec{\theta} \cdot \vec{x} = -0.780 - 0.012 \cdot x_1 + 0.128 \cdot x_2 = 0.44$$

Since the value is > 0 then $\hat{y} = 1$.

- Calculate the prediction of the neural network for the new input $\vec{x} = (5, 10)$. Assume a logistic function as activation function in the unit, which is therefore a sigmoid neuron.

Solution

$$g(\vec{x}) = \frac{1}{1 + \exp(0.780 + 0.012 \cdot x_1 - 0.128 \cdot x_2)} = 0.608$$

and since $g(\vec{x}) > 0.5$ then $\hat{y} = 1$.

- Compare the results at the previous two points against the result in [Exercise 2](#). Are they all consistent? Is this expected to be always the case? Which one is right?

Solution

The three methods return the same result in this case but they could return different results, in particular the k -nearest neighbors can be different from the single neuron cases. It is impossible to say who is right, because we do not know the actual response to x .

- In binary classification, the training error can be defined as the number of mispredicted cases. Calculate the training error for the network under the two different activation functions. Which one performs better according to the training error?

Solution

We need to repeat the operations at the previous point for all points in D . We can use Python, or any other program (eg, R), for that. The results are reported in the table, where we used \hat{y}_p and \hat{y}_s to indicate the predictions of the perceptron and sigmoid neuron, respectively:

x_1	x_2	y	$\vec{\theta} \cdot \vec{x}$	\hat{y}_p	$g(\vec{\theta} \cdot \vec{x})$	\hat{y}_s
10.	2.	1.	-0.64	0	0.34	0
15.	2.	1.	-0.7	0	0.33	0
6.	11.	1.	0.56	1	0.64	1
2.	3.	0.	-0.42	0	0.4	0
5.	15.	1.	1.08	1	0.75	1
5.	14.	1.	0.95	1	0.72	1
10.	1.	0.	-0.77	0	0.32	0
1.	6.	0.	-0.02	0	0.49	0
17.	19.	1.	1.45	1	0.81	1
15.	13.	0.	0.7	1	0.67	1
19.	9.	0.	0.14	1	0.54	1

We observe that both two types of neurons perform in the same way: they predict 4 cases wrong and 7 right. The training error defined as the number of wrong predictions is 4.

- Derive and draw in the plot of [Exercise 2](#). the decision boundaries between 0s and 1s that is implied by the perceptron and the sigmoid neuron. Are the points linearly separable? [See page 4 of Lecture Notes.]

Solution

As explained in the answer to [Exercise 10](#). we can derive the decision boundaries as follows. The decision boundary for the perceptron is $\vec{\theta} \cdot \vec{x} = 0$:

$$-0.012 * x_1 + 0.128 * x_2 - 0.780 = 0$$

which is a line indeed.

The decision boundary for the sigmoid neuron is $g(\vec{\theta} \cdot \vec{x}) = 0.5$. That is:

$$\frac{1}{1 + \exp(0.780 + 0.012x_1 - 0.128x_2)} = 0.5$$

Simplifying:

$$\begin{aligned} 1 &= 0.5(1 + \exp(0.780 + 0.012x_1 - 0.128x_2)) \\ 0.5 &= 0.5 \exp(0.780 + 0.012x_1 - 0.128x_2) \\ \log_e 1 &= (0.780 + 0.012x_1 - 0.128x_2) \\ \log_e 1 &= (0.780 + 0.012x_1 - 0.128x_2) \\ 0.012x_1 - 0.128x_2 + 0.780 &= 0 \end{aligned}$$

which is also a line. The two neurons lead to the same separator function, which is depicted in [Figure 6](#)!

Note, however, that in the traing phase, that is, when the values of the weights have to be decided, using one or the other activation function can lead to different values for $\vec{\theta}$. This is because the output of the logistic function is a real value while the one of the step function is either 0 or 1. Hence, the loss function to optimize is different and may have minima in different points of the space of parameters $\vec{\theta}$.

In [Figure 6](#) we can recognise the 4 points that are misclassified (the four red dots below the line). We can also see that the separator found is perhaps not the best one. It seems that a separator with only two points mispredicted should be possible. How can better parameters be found?

Finally, we can observe that the data points are not linearly separable and that hence a training error equal to zero on this training set is not possible with the single layer neurons analysed.

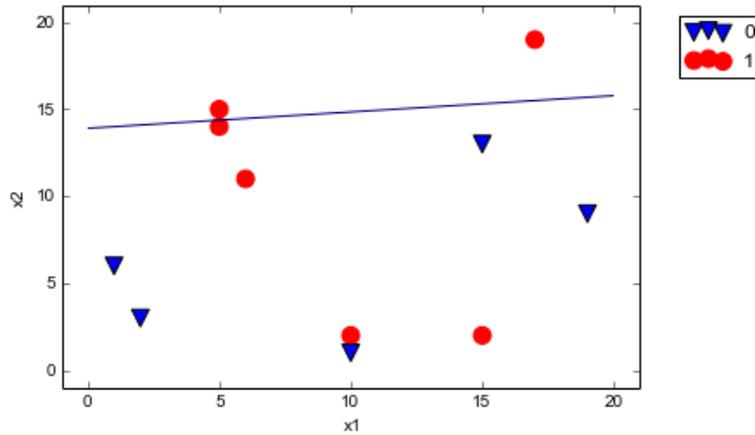
Below I report part of the Python code behind these calculations. It assumes that the module numpy is imported and that the data set D is put in the array C .

In [10]: `import math`

```
def stepfunc(x):
    return -0.780 -0.012*x[0] + 0.128*x[1]

def logistic(x):
    return 1/(1+math.exp(0.780 +0.012*x[0] - 0.128*x[1] ))

print stepfunc([5,10]) # > 0
print logistic([5,10]) # > 0.5
```

Figure 6: The linear separator for the Exercise [Exercise 8](#).

0.44
0.608259030747

Training error:

```
In [11]: l = np.apply_along_axis(logistic,1,C[:,0:2])
s = np.apply_along_axis(stepfunc,1,C[:,0:2])
lb = np.where(l>0.5,1,0)
sb = np.where(s>0,1,0)
L = sum(sb != C[:,2])

In [13]: C0 = C[C[:,2]==0]
C1 = C[C[:,2]==1]
l1=plt.scatter(C1[:,0], C1[:,1], s=100, marker='o',color="red")
l0=plt.scatter(C0[:,0], C0[:,1], s=100, marker="v")
#plt.axis.xlabel=('x')
#plt.axis.Axis.ylabel=('x')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend((l0,l1),('0','1'),bbox_to_anchor=(1.05,1),loc=2)
x0 = np.linspace(0, 20)
x1 = np.linspace(0, 20)[: , None]
plt.contour(x0, x1.ravel(), -0.780 -0.012*x0 + 0.128*x1, [1])
plt.show()
```

Exercise 9. Single Layer Perceptrons

(Based on Slides 49-67)

Can you represent the two layer perceptron of Figure 7 as a single perceptron that implements the same function? If yes, then draw the perceptron.

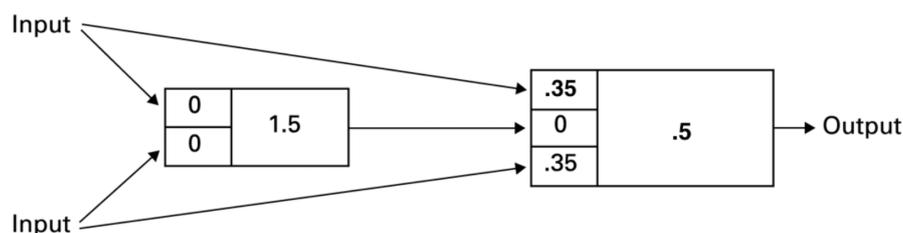


Figure 7: A two layer neural network

Solution

It corresponds to a single perceptron with two inputs of weight .35 each. Indeed, the output of the first neuron is multiplied by 0 in the second, hence it has no influence whatever its output is. This can also be shown by the equivalent outputs of the two networks on all 4 possible combinations of inputs.

Exercise 10. Expressiveness of Single Layer Perceptrons

(Based on slide 63)

Is there a Boolean (logical) function in two inputs that cannot be implemented by a single perceptron? Does the answer change for a single sigmoid neuron?

Solution

Yes. We saw, for example, in the lecture notes that the algebraic expression of a perceptron is:

$$\text{output} := \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

Then the decision boundary is

$$\sum_j w_j x_j = \text{threshold}$$

In the case of two inputs, x_1 and x_2 , this becomes: $w_1 x_1 + w_2 x_2 = \text{threshold}$, which corresponds to the equation of a line in the Cartesian plane:

$$x_2 = -\frac{w_1}{w_2} x_1 + \frac{1}{w_2} \text{threshold}$$

(you might have seen this with y in place of x_2 and x in place of x_1 .) Figure 8 taken from the slides gives an example of a non separable case:

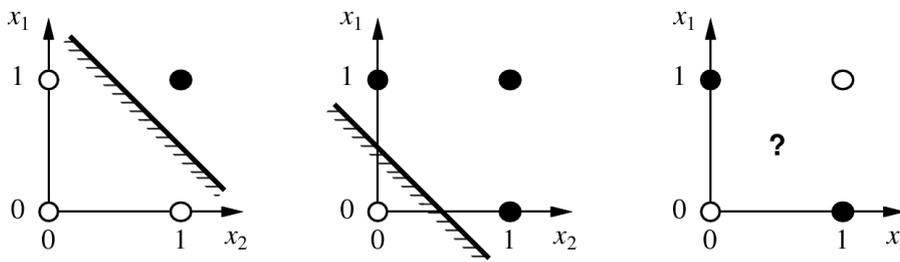


Figure 8: The figure is part of the solution only.

A sigmoid neuron would have the same problem. Indeed, if we use the value 0.5 as the discriminant on the output of a sigmoid neuron to answer 0 or 1 then the decision boundary corresponds to:

$$\frac{1}{1 + \exp(-\sum_j w_j x_j - b)} = 0.5$$

Solving in \vec{x} we obtain an equation of the form:

$$\sum_j w_j x_j - b = \text{constant}$$

which therefore is also a line.

Exercise 11. Logical Functions and Neural Networks

(Based on Slide 49-63)

The NAND gate is universal for computation, that is, we can build any computation up out of NAND gates. We saw in the Study Group session that a single perceptron can model a NAND gate. From here, it follows that using networks of perceptrons we can compute any logical function.

For example, we can use NAND gates to build a circuit which adds two bits, x_1 and x_2 . This requires computing the bitwise sum, $x_1 \text{ XOR } x_2$, as well as a carry bit which is set to 1 when both x_1 and x_2 are 1, i.e., the carry bit is just the bitwise product $x_1 x_2$. The circuit is depicted in Figure 9.

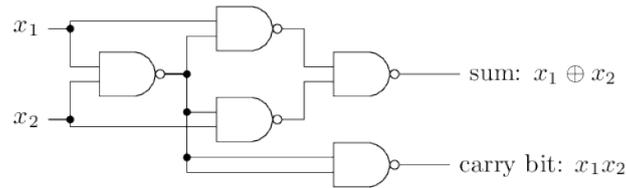
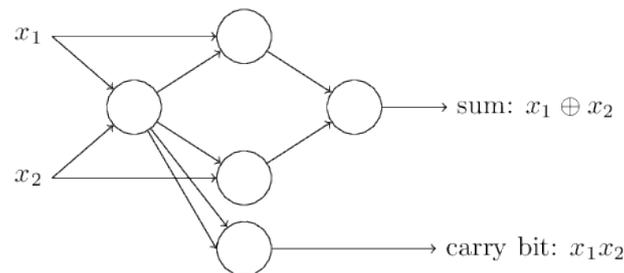


Figure 9: The adder circuit of [Exercise 11](#).

Draw a neural network of NAND perceptrons that would simulate the adder circuit from the figure. [You do not need to decide the weights. You have already discovered which weights for a single perceptron would implement a NAND function in one of the exercises for the Study Group.]

What is the advantage of neural networks with respect to logical circuits?

Solution



The computational universality of perceptrons is reassuring because it tells us that networks of perceptrons can be as powerful as any other computing device. NN are however not merely a new type of NAND gate. It turns out that we can devise learning algorithms which can automatically tune the weights and biases of a network of artificial neurons. This tuning happens in response to external stimuli, without direct intervention by a programmer. These learning algorithms enable us to use artificial neurons in a way which is radically different to conventional logic gates. Instead of explicitly laying out a circuit of NAND and other gates, our neural networks can simply learn to solve problems, sometimes problems where it would be extremely difficult to directly design a conventional circuit.

Exercise 12. Computer Performance Prediction

(Based on Slides 17-69)

You want to predict the running time of a computer program on any architecture. To achieve this task you collect the running time of the program on all machines you have access to. At the end you have a spreadsheet with the following columns of data:

- (1) vendor name: 30 different brands
- (2) MYCT: machine cycle time in nanoseconds (integer)
- (3) MMIN: minimum main memory in kilobytes (integer)
- (4) MMAX: maximum main memory in kilobytes (integer)
- (5) CACH: cache memory in kilobytes (integer)
- (6) CHMIN: minimum channels in units (integer)
- (7) CHMAX: maximum channels in units (integer)
- (8) Running time in seconds (integer)

Indicate which of the following is correct:

- a. It is a supervised learning, regression task. Therefore, we can apply 5-nearest neighbors. For a new machine, the predicted running time in seconds is the average of the running time of the 5 closest machines. The distance between machines is calculated as the sum of the squares of the differences for each of the attributes from (1) to (7).

- b. It is a supervised learning, regression task. Therefore, we can apply a linear model that takes attributes (1)-(7) as independent variables and attribute (8) as response variable.
- c. It is a supervised learning, classification task. Therefore, we can train a multilayer neural network that has an input layer made by input nodes one for each of the attributes (1)-(7); an output layer made by one single sigmoid node that outputs the predicted running time in seconds; an hidden layer of say 10 nodes made by sigmoid nodes.
- d. the same as in the previous point but where the output layer is now made by one single node whose activation function is a linear function. The task is then supervised learning, regression.
- e. It is an unsupervised learning task.
- f. It is a reinforcement learning task.

Solution

a. b. d. are correct. c. would not be correct because the output would always be in $[0, 1]$ while the running time of a program is not restricted to that interval. Thus, this is an example of neural network applied for regression rather than classification, as we saw so far.

Exercise 13. The Recursion Formula in Multilayer Neural Networks

(Based on Slide 66-67)

[Advanced] Suppose you have a single hidden layer neural network with *linear activation functions*. That is, for each unit the output is some constant c times the weighted sum of the inputs plus a constant d .

For a given assignment of the weights \vec{w} , write down equations for the value of the units in the output layer as a function of \vec{w} and the input layer \vec{x} , without any explicit mention of the output of the hidden layer. Show that there is a network with no hidden units that computes the same function.

Solution

Let $g(x) = cx + d$ be the linear function in all units. Let x_i be the inputs to the network and hence to the hidden layers and let y_k be the output of the hidden layer and the input to the output nodes.

The output of a hidden layer is

$$y_k = g\left(\sum_i w_{ik}x_i\right) = c \sum_i w_{ik}x_i + d$$

and the output z_j of the output node is:

$$z_j = g\left(\sum_k w_{kj}y_k\right) = c \sum_k w_{kj}y_k + d = c \left(\sum_k w_{kj}c \left(\sum_i w_{ik}x_i + d\right)\right) + d$$

Hence the net corresponds to a network with no hidden layer, linear activation function $g(x) = c^2x + d(1 + \sum_k w_{kj})$ in the output layer and weights: $\sum_k w_{kj} \sum_i w_{ik}$.