Machine Learning - Exercise 4

Robin Perälä 175910

21.4.2022

Task 1

(Nielsen, p. 10, part I)

Sigmoid neurons simulating perceptrons. Suppose we take all the weights and biases in a network of perceptrons, and multiply them by a positive constant, $c > 0$. Show that the behavior of the network doesn't change.

Here is the perceptron rule:

$$output = \begin{cases} 0, & \sum_j w_j x_j + b \leq 0 \\ 1, & \sum_j w_j x_j + b > 0 \end{cases}$$

The step is

$$\sum_j w_j x_j + b = 0,$$

where W are the weights and B are the biases.

We multiply the weights and biases by the constant c.

$$\sum_j c w_j x_j x + cb = 0$$

.

c is a contant and can be put in front

$$c(\sum_j w_j x_j + b) = 0$$

.

For that to be satisfied, either: $c = 0$ or $\sum_j w_j x_j + b = 0$ or both must hold.

By definition $c > 0$, and therefore we know that $\sum_j w_j x_j + b = 0$ holds. We are back at the original equation we started with before multiplying by the constant c. From that we can conclude that the constant is not affecting the behavior of the network.

Task 2

(ISLR, 10.10.1, p. 458)

Consider a neural network with two hidden layers: $p = 4$ input units, 2 units in the first hidden layer, 3 units in the second hidden layer, and a single output.

(a) *Draw a picture of the network, similar to Figures 10.1 or 10.4.*

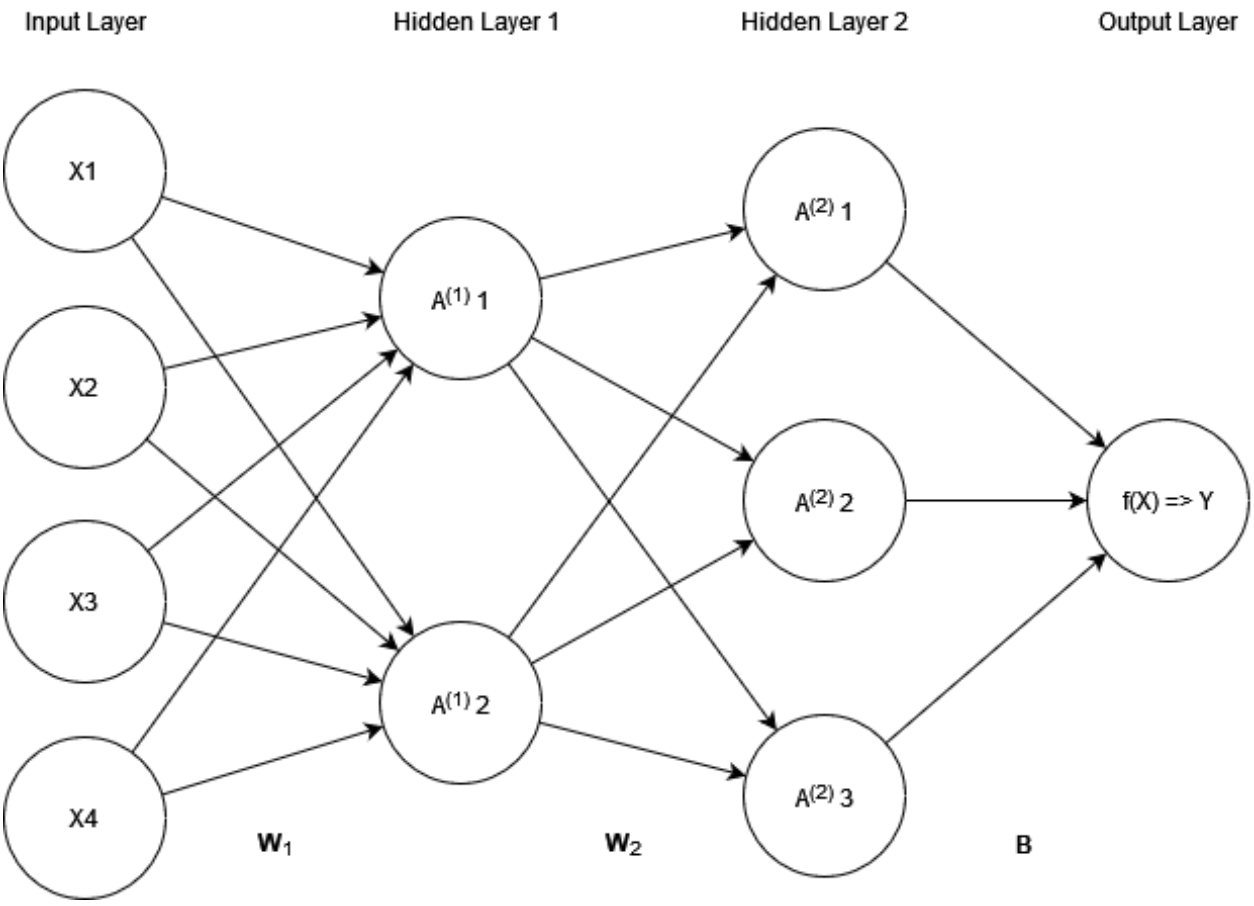


Figure 1: Picture drawing of neural network (tool used: draw.io)

(b) *Write out an expression for $f(X)$, assuming ReLU activation functions. Be as explicit as you can!*

ReLU activation function is

$$g(z) = \begin{cases} z, & z > 0 \\ 0, & z <= 0 \end{cases}$$

Let's build our expression step by step. We try to do it in a logical way by using the picture drawn in 2a. We start from the inputs at the left-hand-side of the picture and proceed towards the right, until we end up at the final output.

X is the input $X = \{X_1, X_2, X_3, X_4\}$. It is the input into the first hidden layer through a linear model inside the activation function $g()$.

The linear model is of the form

$$w_0 + \sum_{j=1}^{p} w_j X_j,$$

where w_j are weights, w_0 is the bias. and p is the number of inputs. The activation function is applied for all nodes on the first layer. We notate a node in the first hidden layer with k. Therefore the result of a node in the first hidden layer is

$$A_k = g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j)$$

After using the activation function on the first layer, those results become inputs into the second layer. Let's define some more notation. We notate the first layer with subscript $^{(1)}$ and the second layer with subscript $^{(2)}$. $A_k^{(1)}$ is the result for node k on the first layer and $A_n^{(2)}$ the result for node n on the second layer. The result from the first layer is notated

$$A_k^{(1)} = g(w_{k0}^{(1)} + \sum_{j=1}^{p} w_{kj}^{(1)} X_j)$$

The second layer also uses the same activation function but with different weights and bias. All nodes in the first hidden layer become inputs into the second hidden layer, i.e. we need to sum them up (with different weights and biases)

$$A_n^{(2)} = g(w_{n0}^{(2)} + \sum_{k=1}^{K} w_n^{(2)} A_k^{(1)})$$

Which is the same thing as

$$A_n^{(2)} = g(w_{n0}^{(2)} + \sum_{k=1}^{K} w_n^{(2)} g(w_{k0}^{(1)} + \sum_{j=1}^{p} w_{kj}^{(1)} X_j))$$

Where K is the number of nodes in the first hidden layer. All the results from the second layer are then summed up with different weights and a bias to obtain the output.

$$f(X) = \beta_0 + \sum_{n=1}^{N} \beta_n A_n^{(2)}$$

Where N is the number of nodes in the second hidden layer

$$f(X) = \beta_0 + \sum_{n=1}^{N} \beta_n g(w_0^{(2)} + \sum_{k=1}^{K} w_k^{(2)} A_k^{(1)})$$

$$f(X) = \beta_0 + \sum_{n=1}^{N} \beta_n g(w_{n0}^{(2)} + \sum_{k=1}^{K} w_{nk}^{(2)} g(w_{k0}^{(1)} + \sum_{j=1}^{p} w_{kj}^{(1)} X_j))$$

More specifically, in our example the expression for $f(x)$ is

$$f(X) = \beta_0 + \sum_{n=1}^{3} \beta_n g(w_{n0}^{(2)} + \sum_{k=1}^{2} w_{nk}^{(2)} g(w_{k0}^{(1)} + \sum_{j=1}^{4} w_{kj}^{(1)} X_j))$$

Which can also be written as

$$f(X) = \beta_0 + \beta_1 A_1^{(2)} + \beta_2 A_2^{(2)} + \beta_3 A_3^{(2)}$$

(c) *Now plug in some values for the coefficients and write out the value of* $f(X)$.

$$\begin{aligned} W_{1j} &= \{0, 1, 2, 3, 4\} \\ W_{2j} &= \{-5, -6, -7, -8, -9\} \\ W_{1k} &= \{0, 1, 2\} \\ W_{2k} &= \{3, 4, 5\} \\ W_{3k} &= \{-6, -7, -8\} \\ \beta_n &= \{0, 1, 2, 3\} \end{aligned}$$

$$\begin{aligned} A_1^{(1)} &= g(0 + X_1 + 2X_2 + 3X_3 + 4X_4) \\ A_2^{(1)} &= g(-5 - 6X_1 - 7X_2 - 8X_3 - 9X_4) \end{aligned}$$

$$\begin{aligned} A_1^{(2)} &= g(0 + A_1^{(1)} + 2A_2^{(1)}) \\ A_2^{(2)} &= g(3 + 4A_1^{(1)} + 5A_2^{(1)}) \\ A_3^{(2)} &= g(-6 + -7A_1^{(1)} + -8A_2^{(1)}) \end{aligned}$$

$$f(X) = 0 + 1A_1^{(2)} + 2A_2^{(2)} + 3A_3^{(2)}$$

The final expression can be achieved by combining all previous terms. The combination is very long and messy, and therefore decide not to write it explicitly (it is left as an exercise for the interested reader)

Extra: We also test the function with one set of values for X. Specifically $X_1 = 3, X_2 = 4, X_3 = 5, X_4 = 6$

```
W_1j = 0:4
W_2j = -(5:9)
W_1k = 0:2
W_2k = 3:5
W_3k = -(6:8)
beta_n = 0:3

X = 3:6

v11 = W_1j %*% c(1, X)
A11 = ifelse(v11>0, v11, 0)

v12 = W_2j %*% c(1, X)
A12 = ifelse(v12>0, v12, 0)

v21 = W_1k %*% rbind(1, A11, A12)
A21 = ifelse(v21>0, v21, 0)
```

```
v22 = W_2k %*% rbind(1, A11, A12)
A22 = ifelse(v22>0, v22, 0)

v23 = W_3k %*% rbind(1, A11, A12)
A23 = ifelse(v23>0, v23, 0)

fx = beta_n %*% c(1, A21, A22, A23)
fx
```

```
##       [,1]
## [1,]  456
```

(d) *How many parameters are there?*

We have 4 inputs and therefore 4 weights into each of the two perceptrons in the first hidden layer. Each of the perceptron also has a bias. This results in 10 parameters for that layer.

The two perceptrons in the first hidden layer have two weights each to the second hidden layer with 3 perceptrons. Each perceptron also has a bias and therefore there are 9 parameters at that layer.

The 3 perceptrons in the hidden layer provide one weight each to the final output. The final output also has the bias, and therefore there are 4 parameters at that layer.

All in all there are

$$\begin{aligned} & W_1 + W_2 + B \\ =& (4+1)\cdot 2 + (2+1)\cdot 3 + (3+1) \\ =& 10 + 9 + 4 \\ =& 23 \text{ parameters} \end{aligned}$$

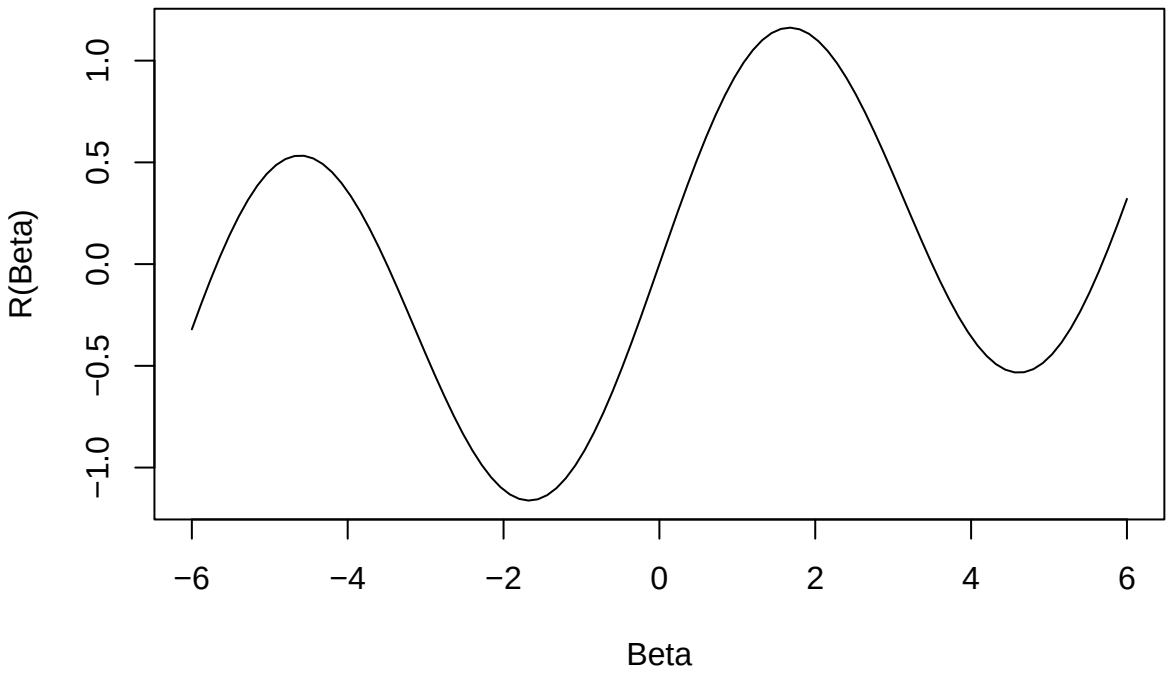
Task 3

(ISLR, 10.10.6, p. 459)

Consider the simple function $R(\beta) = sin(\beta) + \beta/10$.

(a) *Draw a graph of this function over the range* $\beta \in [-6, 6]$.

```
curve(sin(x)+x/10, from=-6, to=6,
      xlab="Beta", ylab="R(Beta)")
```



(b) *What is the derivative of this function?*

$\frac{dR}{d\beta} = \cos(\beta) + \frac{1}{10}$

(c) *Given* $\beta^0 = 2.3$*, run gradient descent to find a local minimum of* $R(\beta)$ *using a learning rate of* $p = 0.1$*. Show each of* $\beta^0, \beta^1, \ldots$ *in your plot, as well as the final answer.*

For 2.3, the derivative becomes $\cos(2.3) + \frac{1}{10} = -0.566276$

To get the next beta we use $\beta_1 = \beta_0 - \frac{dR}{d\beta} \cdot L$ Where L is the learning rate

The learning rate is 0.1 and therefore the new β_1 is

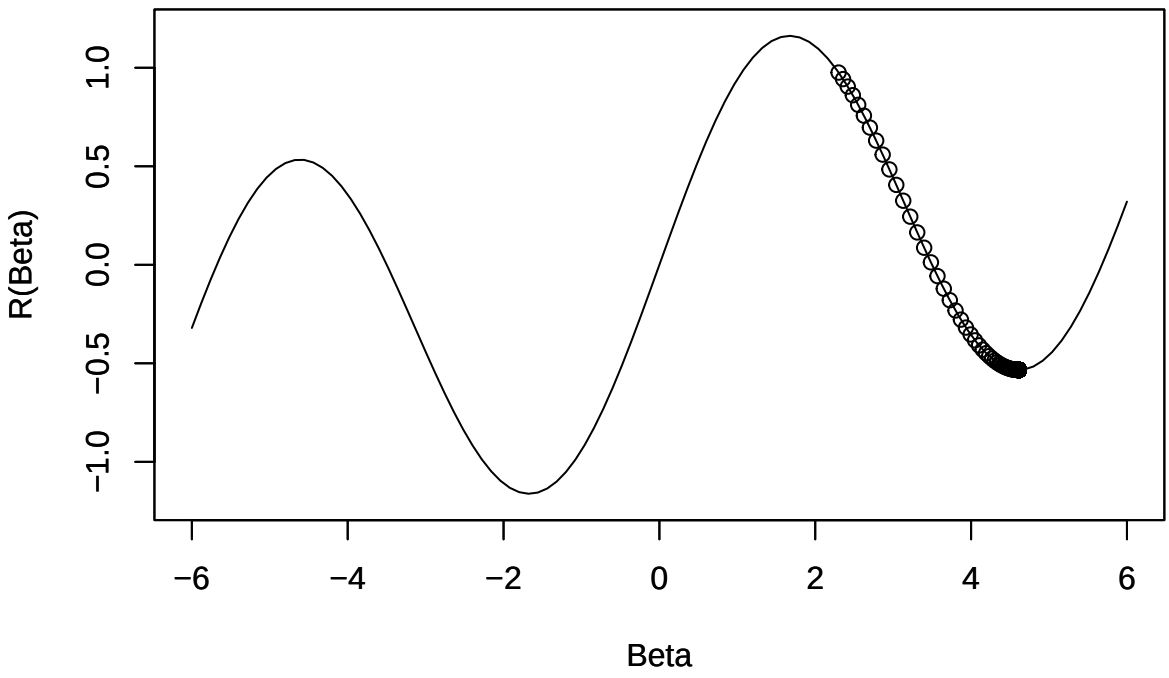
$$= 2.3 + 0.1 \cdot 0.566 = 2.3 + 0.0566 = 2.3566$$

We repeat the process until we find a local minimum: R(4.6122) = -0.533765

```r
R  = function(beta){sin(beta) + beta/10}
dR = function(beta){cos(beta) + 1/10}
newBeta = function(beta){beta - L*dR(beta)}

descent = function(beta0){
  betaArray = c(beta0)
  for(n in 1:1000){
    betaArray = c(betaArray, newBeta(betaArray[n]))
  }
  betaArray
}

plotDescent = function(betas){
  curve(sin(x)+x/10, from=-6, to=6,
        xlab="Beta", ylab="R(Beta)",
        xlim = c(-6,6), ylim=c(-1.2, 1.2))
  par(new=TRUE)
  plot(x=betas, y=R(betas),
       xlab="Beta", ylab="R(Beta)",
       xlim = c(-6,6), ylim=c(-1.2, 1.2))
}


#Let's find the minimum when beta0 = 2.3
L = 0.1
beta0 = 2.3
allBetas = descent(beta0)
cat("Beta: ", tail(allBetas, 1),
    "\tR: ", tail(R(allBetas), 1))
```

```
## Beta:  4.612222  R:  -0.5337653
```

```r
plotDescent(allBetas)
```

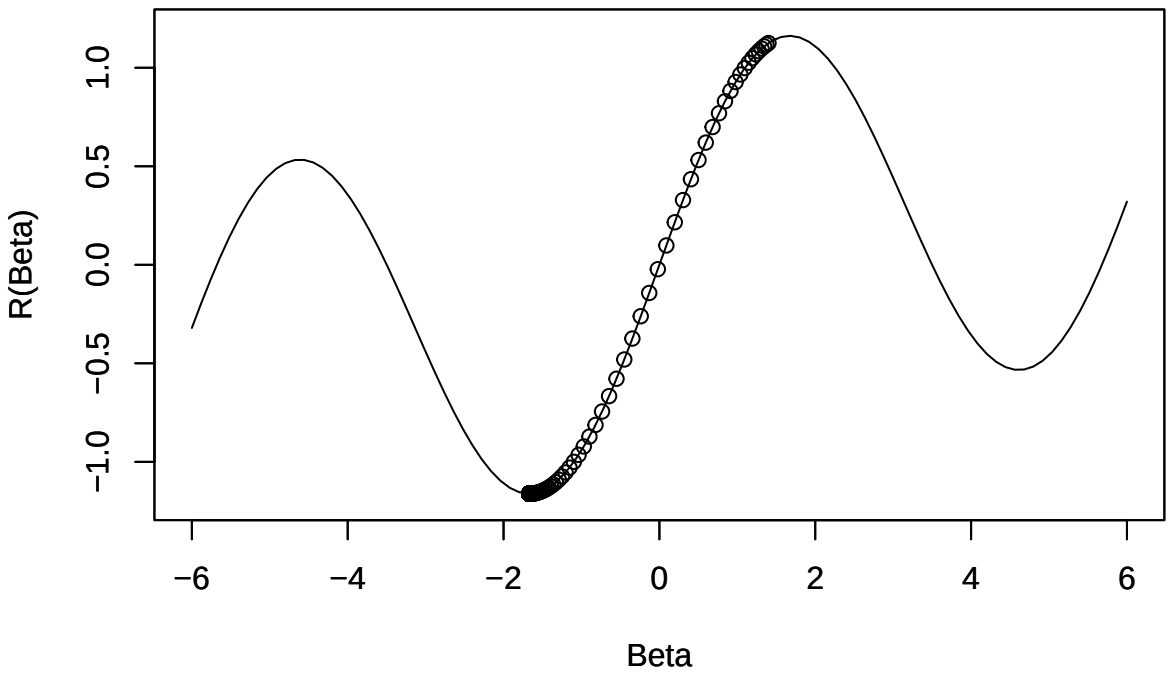


(d) *Repeat with* $\beta^0 = 1.4$.

```
#Let's find the minimum when beta0 = 1.4
beta0 = 1.4
allBetas = descent(beta0)
cat("Beta: ", tail(allBetas, 1),
    "\tR: ", tail(R(allBetas), 1))
```

```
## Beta:  -1.670964     R:  -1.162084
```

```
plotDescent(allBetas)
```


R(Beta)
1.0
0.5
0.0
−0.5
−1.0
−6
−4
−2
0
2
4
6
Beta

Task 4

(Nielsen, p. 21)

Indeed, there's even a sense in which gradient descent is the optimal strategy for searching for a minimum. Let's suppose that we're trying to make a move Δv in position so as to decrease C as much as possible. This is equivalent to minimizing $\Delta C \approx \nabla C \cdot \Delta v$. We'll constrain the size of the move so that $||\Delta v|| = \epsilon$ for some small fixed $\epsilon > 0$. In other words, we want a move that is a small step of a fixed size, and we're trying to find the movement direction which decreases C as much as possible. It can be proved that the choice of Δv which minimizes $\nabla C \cdot \Delta v$ is $\Delta v = -\eta \nabla C$, where $\eta = \epsilon / ||\nabla C||$ is determined by the size constraint $||\Delta v|| = \epsilon$. So gradient descent can be viewed as a way of taking small steps in the direction which does the most to immediately decrease C.

Exercises

- *Prove the assertion of the last paragraph. Hint: If you're not already familiar with the Cauchy-Schwarz inequality, you may find it helpful to familiarize yourself with it.*

$$|\nabla C \cdot \Delta v| \leq ||\nabla C||\ ||\Delta v||$$

is equivalent to

$$\nabla C \cdot \Delta v \leq ||\nabla C||\ ||\Delta v||$$
$$\text{or}$$
$$\nabla C \cdot \Delta v \geq -\ ||\nabla C||\ ||\Delta v||$$

Because we want to minimize $\nabla C \cdot \Delta v$, we choose

$$min(\nabla C \cdot \Delta v) = -||\nabla C||\ ||\Delta v||$$

which is

$$min(\nabla C \cdot \Delta v) = -||\nabla C||\ \epsilon$$

We call this expression **(1)**. We also know that

$$\sqrt{(\nabla C \cdot \nabla C)} = ||\nabla C||$$
$$\nabla C \cdot \nabla C = ||\nabla C||^2$$
$$\frac{\nabla C \cdot \nabla C}{||\nabla C||} = ||\nabla C||$$

Multiply both sides by ϵ

$$\frac{\epsilon \nabla C \cdot \nabla C}{||\nabla C||} = \epsilon ||\nabla C||$$

We call this expression **(2)**. Now combine expression **(1)** and **(2)**

$$min(\nabla C \cdot \Delta v) = -||\nabla C||\ \epsilon = -\frac{\epsilon \nabla C \cdot \nabla C}{||\nabla C||} = -\frac{\epsilon \nabla C}{||\nabla C||} \nabla C = -\eta \nabla C \nabla C$$

which means that

$$= \Delta v \nabla C => \Delta v = -\eta \nabla C$$

- *I explained gradient descent when C is a function of two variables, and when it's a function of more than two variables. What happens when C is a function of just one variable? Can you provide a geometric interpretation of what gradient descent is doing in the one-dimensional case?*

Geometrically it is moving along the tangent line of the function, i.e. the derivative. When the derivative is negative, $-\eta \nabla C$ tells us to increase the value of v (move to the right along the v-axis) and when the derivative is positive we decrease the value of v (move to the left). Mathematically: $v_{n+1} = v_n - \eta \nabla C$. This algorithm iterates until the derivative of C is sufficiently close to zero. This means we have reached the minimum.