Lab 3

Emilio Dorigatti

2020-11-20

Welcome to the third lab. The first exercise is an implementation of gradient descent on a bivariate function. The second exercise is about computing derivatives of the weights of a neural network, and the third exercise combines the previous two.

Exercise 1

This exercise is about gradient descent. We will use the function $f(x_1, x_2) = (x_1 - 6)^2 + x_2^2 - x_1 x_2$ as a running example:

1. Use pen and paper to do three iterations of gradient descent:
   - Find the gradient of f;
   - Start from the point $x_1 = x_2 = 6$ and use a step size of $1/2$ for the first step, $1/3$ for the second step and $1/4$ for the third step;
   - What will happen if you keep going?
2. Write a function that performs gradient descent:
   - For simplicity, we use a constant learning rate.
   - Can you find a way to prematurely stop the optimization when you are close to the optimum?

```
func.value = function(x) {
  (x[1] - 6)^2+x[2]^2-x[1]*x[2]
}

func.gradient = function(x) {
  c(
    2*x[1]-x[2]-12,
    -x[1]+2*x[2]
  )
}

func.value(c(6, 6))
```

```
## [1] 0
```

```
func.gradient(c(6, 6))
```

```
## [1] -6  6
```

Does it match what you computed?

```
gradient_descent_optimizer = function(x0, func, grad, max_steps, alpha) {
  # x0 is the initial point
  # func computes the value of the function at a given point
  # grad computes the gradient of the function at a given point
  # max_steps is the maximum number of gradient descent steps
  # alpha is the learning rate

  x = x0
  v = func(x0)
  for(i in 1:max_steps) {
    x = x - alpha * grad(x)
```

```
    vnew = func(x)
    if(v - vnew < 1e-4) {
      break
    }
    v = vnew
  }
  x
}

gradient_descent_optimizer(c(6, 6), func.value, func.gradient, 10, 0.1)
```

```
## [1] 7.943505 4.056495
```

Play a bit with the starting point and learning rate to get a feel for its behavior; how close can you get to the minimum?

Solution

The gradient of f is:

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \begin{vmatrix} \partial f/\partial x_1 \\ \partial f/\partial x_2 \end{vmatrix} = \begin{vmatrix} 2(x_1 - 6) - x_2 \\ 2x_2 - x_1 \end{vmatrix}$$

For $\mathbf{x} = |6, 6|^T$ we have $f(\mathbf{x}) = 0$ and $\nabla_{\mathbf{x}} f(\mathbf{x}) = |-6, 6|^T$.

Let $\mathbf{x}^{(t)}$ denote the point at the t-th iteration. Then:

t	$\mathbf{x}^{(t)}$	$f(\mathbf{x}^{(t)})$	$\nabla_{\mathbf{x}} f(\mathbf{x})$	$\mathbf{x}^{(t+1)}$
1	$\|6, 6\|$	0	$\|-6, 6\|$	$\|6, 6\| - (1/2) \cdot \|-6, 6\| = \|9, 3\|$
2	$\|9, 3\|$	-9	$\|3, -3\|$	$\|9, 3\| - (1/3) \cdot \|3, -3\| = \|8, 4\|$
3	$\|8, 4\|$	-12	$\|0, 0\|$	$\|8, 4\| - (1/4) \cdot \|0, 0\| = \|8, 4\|$

Where all vectors are intended to be vertical. As the gradient at the last point is zero, nothing will change if we continue to apply this procedure.

Exercise 2

This exercise is about computing gradients with the chain rule, with pen and paper. We will work with a neural network with a single hidden layer with two neurons and an output layer with one neuron (see Figure 1).

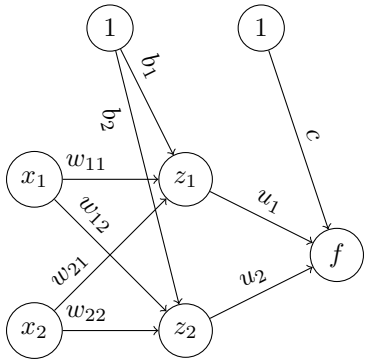


Figure 1: Neural network used in Exercise 2.

The neurons in the hidden layer use the tanh activation, while the output neuron uses the sigmoid. The loss used in binary classification is the *binary cross-entropy*:

$$\mathcal{L}(y, f_{out}) = -y \log f_{out} - (1-y)\log(1-f_{out})$$

where $y \in \{0, 1\}$ is the true label and $f_{out} \in (0, 1)$ is the predicted probability that $y = 1$.

1. Compute $\partial \mathcal{L}(y, f_{out})/\partial f_{out}$
2. Compute $\partial f_{out}/\partial f_{in}$
3. Show that $\partial \sigma(x)/\partial x = \sigma(x)(1-\sigma(x))$
4. Show that $\partial \tanh(x)/\partial x = 1 - \tanh(x)^2$ (Hint: $\tanh(x) = (e^x - e^{-x})(e^x + e^{-x})^{-1}$)
5. Compute $\partial f_{in}/\partial c$
6. Compute $\partial f_{in}/\partial u_1$
7. Compute $\partial \mathcal{L}(y, f_{out})/\partial c$
8. Compute $\partial \mathcal{L}(y, f_{out})/\partial u_1$
9. Compute $\partial f_{in}/\partial z_{2,out}$
10. Compute $\partial z_{2,out}/\partial z_{2,in}$
11. Compute $\partial z_{2,in}/\partial b_2$
12. Compute $\partial z_{2,in}/\partial w_{12}$
13. Compute $\partial z_{2,in}/\partial x_1$
14. Compute $\partial \mathcal{L}(y, f_{out})/\partial b_2$
15. Compute $\partial \mathcal{L}(y, f_{out})/\partial w_{12}$
16. Compute $\partial \mathcal{L}(y, f_{out})/\partial x_1$

You will notice that there are lots of redundancies. We will see how to improve these computations in the lecture and in the next lab. Luckily, modern deep learning software computes gradients automatically for you.

Solution

Question 1

$$\begin{aligned}\frac{\partial \mathcal{L}(y, f_{out})}{\partial f_{out}} &= \frac{\partial}{\partial f_{out}}\Big(y \log f_{out} + (1-y)\log(1-f_{out})\Big) \\ &= -\frac{y}{f_{out}} + \frac{1-y}{1-f_{out}}\end{aligned}$$

Question 2

$$\begin{aligned}\frac{\partial f_{out}}{\partial f_{in}} &= \frac{\partial}{\partial f_{in}} \frac{1}{1+e^{-f_{in}}} \\ &= -(1+e^{-f_{in}})^{-2} \cdot -e^{-f_{in}} \\ &= \frac{e^{-f_{in}}}{(1+e^{-f_{in}})^2}\end{aligned}$$

Question 3

$$\begin{aligned}\frac{\partial}{\partial x}\frac{1}{1+e^{-x}} &= \frac{e^{-x}}{(1+e^{-x})^2} \\ &= \frac{1}{1+e^{-x}} \cdot \frac{(1+e^{-x})-1}{1+e^{-x}} \\ &= \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right) \\ &= \sigma(x)(1-\sigma(x))\end{aligned}$$

Question 4

$$\begin{aligned}\frac{\partial}{\partial x}\tanh(x) &= \frac{\partial}{\partial x}\frac{e^x - e^{-x}}{e^x + e^{-x}} \\ &= \frac{(e^x + e^{-x})(e^x + e^{-x}) - (e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2} \\ &= 1 - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2}\end{aligned}$$

Question 5

$$\begin{aligned}\frac{\partial f_{in}}{c} &= \frac{\partial}{c}\left(c + u_1 \cdot z_{1,out} + u_2 \cdot z_{2,out}\right) \\ &= 1\end{aligned}$$

Question 6

$$\begin{aligned}\frac{\partial f_{in}}{u_1} &= \frac{\partial}{u_1}\left(c + u_1 \cdot z_{1,out} + u_2 \cdot z_{2,out}\right) \\ &= z_{1,out}\end{aligned}$$

Question 7

$$\begin{aligned}\frac{\partial \mathcal{L}(y, f_{out})}{\partial c} &= \frac{\partial \mathcal{L}(y, f_{out})}{\partial f_{out}} \cdot \frac{\partial f_{out}}{\partial f_{in}} \cdot \frac{\partial f_{in}}{c} \\ &= \left(-\frac{y}{f_{out}} + \frac{1-y}{1-f_{out}}\right) \cdot \sigma(f_{in})(1 - \sigma(f_{in})) \cdot 1\end{aligned}$$

Question 8

$$\begin{aligned}\frac{\partial \mathcal{L}(y, f_{out})}{\partial u_1} &= \frac{\partial \mathcal{L}(y, f_{out})}{\partial f_{out}} \cdot \frac{\partial f_{out}}{\partial f_{in}} \cdot \frac{\partial f_{in}}{u_1} \\ &= \left(-\frac{y}{f_{out}} + \frac{1-y}{1-f_{out}}\right) \cdot \sigma(f_{in})(1 - \sigma(f_{in})) \cdot z_{1,out}\end{aligned}$$

Question 9

$$\begin{aligned}\frac{\partial f_{in}}{\partial z_{2,out}} &= \frac{\partial}{\partial z_{2,out}}\left(c + u_1 \cdot z_{1,out} + u_2 \cdot z_{2,out}\right) \\ &= u_2\end{aligned}$$

Question 10

$$\begin{aligned}\frac{\partial z_{2,out}}{\partial z_{2,in}} &= \frac{\partial}{\partial z_{2,in}}\sigma(z_{2,in}) \\ &= 1 - \tanh(z_{2,in})^2\end{aligned}$$

Question 11

$$\begin{aligned}\frac{\partial z_{2,in}}{\partial b_2} &= \frac{\partial}{\partial b_2}\left(b_2 + w_{12} \cdot x_1 + w_{22} \cdot x_2\right) \\ &= 1\end{aligned}$$

Question 12

$$\begin{aligned}\frac{\partial z_{2,in}}{\partial w_{12}} &= \frac{\partial}{\partial b_2}\left(b_2 + w_{12} \cdot x_1 + w_{22} \cdot x_2\right) \\ &= x_1\end{aligned}$$

Question 13

$$\begin{aligned}\frac{\partial z_{2,in}}{\partial x_1} &= \frac{\partial}{\partial x_1}\left(b_2 + w_{12}\cdot x_1 + w_{22}\cdot x_2\right)\\ &= w_{12}\end{aligned}$$

Question 14

$$\begin{aligned}\frac{\partial \mathcal{L}(y, f_{out})}{\partial b_2} &= \frac{\partial \mathcal{L}(y, f_{out})}{\partial f_{out}}\cdot\frac{\partial f_{out}}{\partial f_{in}}\cdot\frac{\partial f_{in}}{\partial z_{2,out}}\cdot\frac{\partial z_{2,out}}{\partial z_{2,in}}\cdot\frac{\partial z_{2,in}}{\partial b_2}\\ &= \left(-\frac{y}{f_{out}} + \frac{1-y}{1-f_{out}}\right)\cdot\sigma(f_{in})(1-\sigma(f_{in}))\cdot u_2\cdot(1-\tanh(z_{2,in})^2)\cdot 1\end{aligned}$$

Question 15

$$\begin{aligned}\frac{\partial \mathcal{L}(y, f_{out})}{\partial w_{12}} &= \frac{\partial \mathcal{L}(y, f_{out})}{\partial f_{out}}\cdot\frac{\partial f_{out}}{\partial f_{in}}\cdot\frac{\partial f_{in}}{\partial z_{2,out}}\cdot\frac{\partial z_{2,out}}{\partial z_{2,in}}\cdot\frac{\partial z_{2,in}}{\partial w_{12}}\\ &= \left(-\frac{y}{f_{out}} + \frac{1-y}{1-f_{out}}\right)\cdot\sigma(f_{in})(1-\sigma(f_{in}))\cdot u_2\cdot(1-\tanh(z_{2,in})^2)\cdot x_1\end{aligned}$$

Question 16

$$\begin{aligned}\frac{\partial \mathcal{L}(y, f_{out})}{\partial x_1} &= \frac{\partial \mathcal{L}(y, f_{out})}{\partial f_{out}}\cdot\frac{\partial f_{out}}{\partial f_{in}}\cdot\frac{\partial f_{in}}{\partial z_{2,out}}\cdot\frac{\partial z_{2,out}}{\partial z_{2,in}}\cdot\frac{\partial z_{2,in}}{\partial x_1}\\ &\quad + \frac{\partial \mathcal{L}(y, f_{out})}{\partial f_{out}}\cdot\frac{\partial f_{out}}{\partial f_{in}}\cdot\frac{\partial f_{in}}{\partial z_{1,out}}\cdot\frac{\partial z_{1,out}}{\partial z_{1,in}}\cdot\frac{\partial z_{1,in}}{\partial x_1}\\ &= \left(-\frac{y}{f_{out}} + \frac{1-y}{1-f_{out}}\right)\cdot\sigma(f_{in})(1-\sigma(f_{in}))\cdot u_2\cdot(1-\tanh(z_{2,in})^2)\cdot w_{12}\\ &\quad + \left(-\frac{y}{f_{out}} + \frac{1-y}{1-f_{out}}\right)\cdot\sigma(f_{in})(1-\sigma(f_{in}))\cdot u_1\cdot(1-\tanh(z_{1,in})^2)\cdot w_{11}\end{aligned}$$

Exercise 3

Now that we know how to do gradient descent and how to compute the derivatives of the weights of a simple network, we can try to do these steps together and train our first neural network! We will use the small dataset with five points we studied in the first lab.

First, let's define the dataset:

```
data.x1 = c(0, 1, 0, -1, 0);
data.x2 = c(0, 0, -1, 0, 1);
data.y = c(1, 0, 0, 0, 0);
```

Next, a function to compute the output of the network:

```
sigmoid = function(x) {
  1 / (1 + exp(-x))
}

nnet.predict = function(params) {
  # params is a vector, we first unpack for clarity
  b1 = params[1]; b2 = params[2];
  w11 = params[3]; w12 = params[4];
  w21 = params[5]; w22 = params[6];
  c = params[7]; u1 = params[8]; u2 = params[9];

  z1 = tanh(b1 + data.x1 * w11 + data.x2 * w21)
  z2 = tanh(b2 + data.x1 * w12 + data.x2 * w22)
```

```
  f = sigmoid(c + u1 * z1 + u2 * z2)
  f
}

# this should return the predictions for the five points in the datasets
params = rnorm(9)
nnet.predict(params)
```

```
## [1] 0.4491225 0.4965392 0.5436179 0.3752571 0.4099662
```

Since gradient descent is done on the loss function, we need a function to compute it:

```
nnet.loss = function(params) {
  preds = nnet.predict(params)
  -mean(data.y * log(preds + 1e-15) + (1 - data.y) * log(1 - preds + 1e-15))
}

nnet.loss(params)
```

```
## [1] 0.6538249
```

Now, we need to compute the gradient of each parameter:

```
nnet.gradient = function(params) {
  # params is a vector, we first unpack for clarity
  b1 = params[1]; b2 = params[2];
  w11 = params[3]; w12 = params[4];
  w21 = params[5]; w22 = params[6];
  c = params[7]; u1 = params[8]; u2 = params[9];

  # first, we perform the forward pass
  z1in = b1 + data.x1 * w11 + data.x2 * w21
  z1out = tanh(z1in)

  z2in = b2 + data.x1 * w12 + data.x2 * w22
  z2out = tanh(z2in)

  fin = c + u1 * z1out + u2 * z2out
  fout = sigmoid(fin)

  # now we start back-propagation through the loss and the output neuron
  dL_dfout = -data.y / (fout + 1e-15) + (1 - data.y) / (1 - fout + 1e-15)
  dfout_dfin = sigmoid(fin) * (1 - sigmoid(fin))

  # compute the gradients for the parameters of the output layer
  dfin_dc = 1
  dfin_du1 = z1out
  dfin_du2 = z2out
  # take the mean gradient across data points
  dL_dc = mean(dL_dfout * dfout_dfin * dfin_dc)
  dL_du1 = mean(dL_dfout * dfout_dfin * dfin_du1)
  dL_du2 = mean(dL_dfout * dfout_dfin * dfin_du2)

  # back-propagate through the neurons in the first hidden layer
  dfin_dz1out = u1
  dfin_dz2out = u2

  dz1out_dz1in = 1 - tanh(z1in)^2
  dz2out_dz2in = 1 - tanh(z2in)^2
```

```
  # and compute the derivatives of the parameters of the hidden layer
  dz1in_db1 = dz2in_db2 = 1
  dL_db1 = mean(dL_dfout * dfout_dfin * dfin_dz1out * dz1out_dz1in * dz1in_db1)
  dL_db2 = mean(dL_dfout * dfout_dfin * dfin_dz2out * dz2out_dz2in * dz2in_db2)

  dz1in_dw11 = dz2in_dw12 = data.x1
  dL_dw11 = mean(dL_dfout * dfout_dfin * dfin_dz1out * dz1out_dz1in * dz1in_dw11)
  dL_dw12 = mean(dL_dfout * dfout_dfin * dfin_dz2out * dz2out_dz2in * dz2in_dw12)

  dz1in_dw21 = dz2in_dw22 = data.x2
  dL_dw21 = mean(dL_dfout * dfout_dfin * dfin_dz1out * dz1out_dz1in * dz1in_dw21)
  dL_dw22 = mean(dL_dfout * dfout_dfin * dfin_dz2out * dz2out_dz2in * dz2in_dw22)

  # return the derivatives in the same order as the parameters vector
  c(
    dL_db1, dL_db2,
    dL_dw11, dL_dw12,
    dL_dw21, dL_dw22,
    dL_dc, dL_du1, dL_du2
  )
}

nnet.gradient(params)
```

```
## [1]  0.10535249 -0.16134603 -0.03237562  0.03281629 -0.06440284  0.05575347
## [7]  0.25490058 -0.04360869 -0.06480158
```

Finite differences are a useful way to check that the gradients are computed correctly:

```
# first, compute the analytical gradient of the parameters
gradient = nnet.gradient(params);

eps = 1e-9
for(i in 1:9) {
  # compute loss when subtracting eps to parameter i
  neg_params = c(params);
  neg_params[i] = neg_params[i] - eps;
  neg_value = nnet.loss(neg_params);

  # compute loss when adding eps to parameter i
  pos_params = c(params);
  pos_params[i] = pos_params[i] + eps;
  pos_value = nnet.loss(pos_params);

  # compute the "empirical" gradient of parameter i
  fdiff_gradient = mean((pos_value - neg_value) / (2 * eps));

  # error if difference is too large
  stopifnot(abs(gradient[i] - fdiff_gradient) < 1e-5);
}

print("Gradients are correct!")
```

```
## [1] "Gradients are correct!"
```

We can finally train our network. Since the network is so small compared to the dataset, the training procedure is very sensitive to the way the weights are initialized and the step size used in gradient descent.

Try to play around with the learning rate and the random initialization of the weights and find reliable values that make training successful in most cases.

```
min_loss = 10
best_params = NULL

for(i in 1:10) {
  params = rnorm(9, sd = 1)
  optimized_params = gradient_descent_optimizer(
    params, nnet.loss, nnet.gradient, max_steps = 100, alpha = 1
  )
  final_loss = nnet.loss(optimized_params)
  cat("Loss", final_loss, ifelse(final_loss < 0.1, "*", ""), "\n")

  if(final_loss < min_loss) {
    min_loss = final_loss
    best_params = optimized_params
  }
}
```

```
## Loss 0.4603099
## Loss 0.09263696 *
## Loss 0.162037
## Loss 0.3783302
## Loss 0.4366499
## Loss 0.04427358 *
## Loss 0.4067504
## Loss 0.06471511 *
## Loss 0.1449693
## Loss 0.05334381 *
```

We can use the function in the previous lab to visualize the decision boundary of the best network:

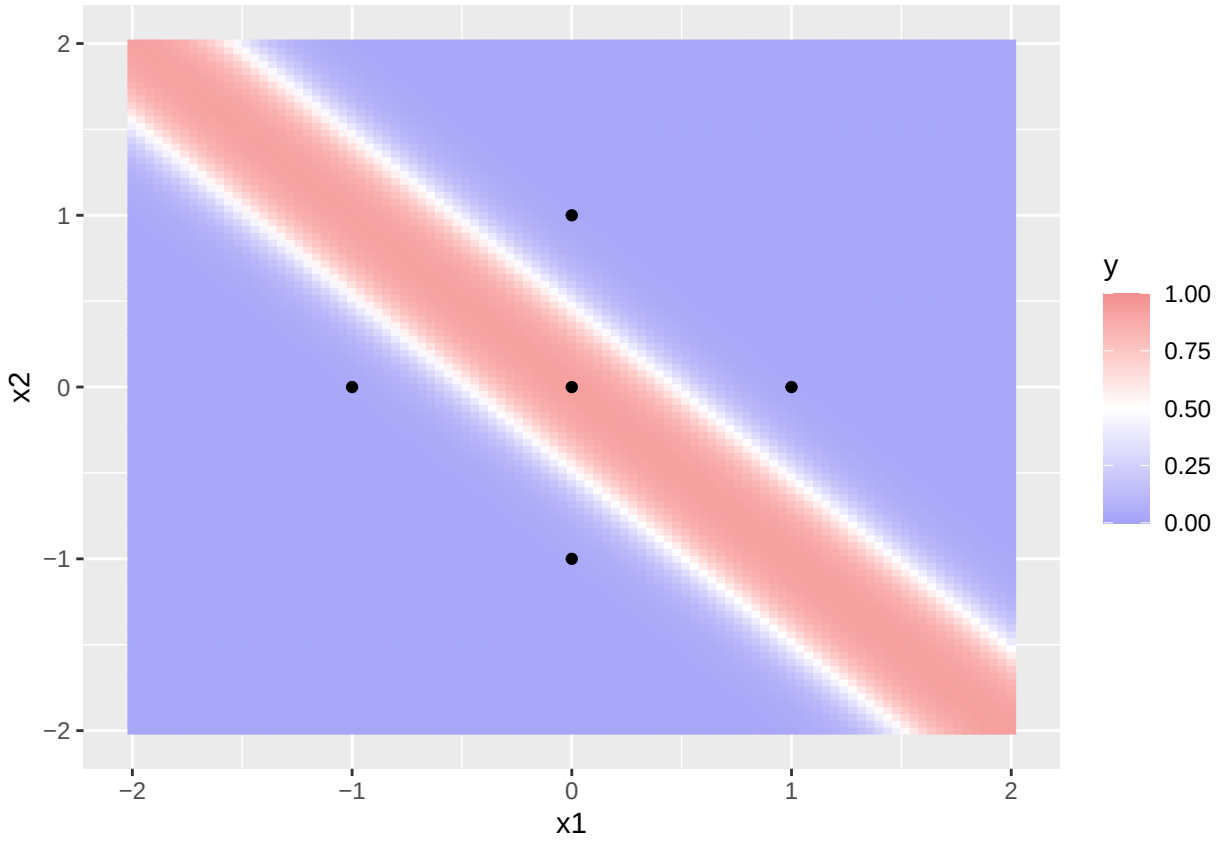
```
library(scales)
library(ggplot2)

data = as.matrix(expand.grid(
  x0 = 1:1,
  x1 = seq(-2, 2, 1 / 25),
  x2 = seq(-2, 2, 1 / 25)
))

data.x1 = data[,2]
data.x2 = data[,3]

plot_grid = function(predictions) {
  # plots the predicted value for each point on the grid;
  # the predictions should have one column and
  # the same number of rows (10,201) as the data
  df = cbind(as.data.frame(data), y = predictions)
  ggplot() +
    geom_tile(aes(x = x1, y = x2, fill = y, color = y), df) +
    scale_color_gradient2(low = muted("blue", 70), mid = "white",
                          high = muted("red", 70), limits = c(0, 1),
                          midpoint = 0.5) +
    scale_fill_gradient2(low = muted("blue", 70), mid = "white",
                         high = muted("red", 70), limits = c(0, 1),
                         midpoint = 0.5) +
    geom_point(aes(x=c(0, 1, 0, -1, 0), y=c(0, 0, -1, 0, 1)))
}

plot_grid(nnet.predict(best_params))
```



Also try to visualize the decision boundary of network with random parameters:

```
plot_grid(nnet.predict(rnorm(9, sd=5)))
```