Tutorial + Homework Business Analytics

Week 12 – Gradient Descent and Neural Networks

Part I: Gradient Descent

Exercise 12.1 Gradient Descent

Let $f: \mathbb{R}^2 \to \mathbb{R}$ be the convex function given by

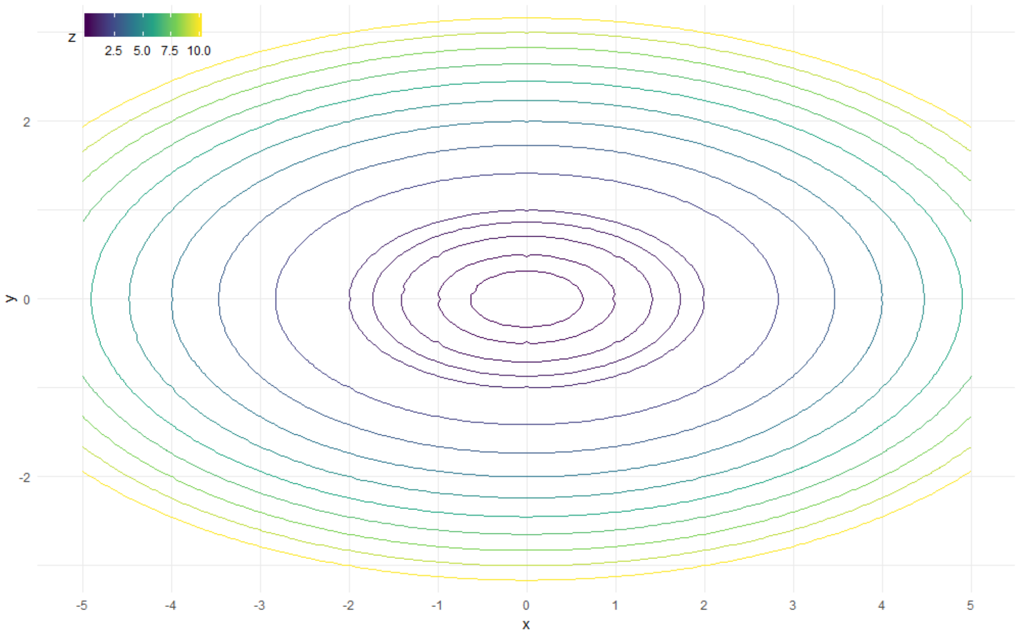
$$f(x, y) = \frac{x^2}{4} + y^2$$

a) **[Tutorial]** Find $\nabla f(x, y)$.

Solution 12.1a:

$$\nabla f(x, y) = \begin{pmatrix} \partial f(x,y)/\partial x \\ \partial f(x,y)/\partial y \end{pmatrix} = \begin{pmatrix} 0.5\, x \\ 2\, y \end{pmatrix}$$

b) **[Tutorial]** Starting from $(x_0, y_0) = (3, 2)^T$ perform 3 steps of gradient descent with a learning rate of $\alpha = 1$. Plot your gradient steps. What do you observe? Does the function value decrease in each step? Will this sequence converge to the optimum at $(0,0)$?



Solution: 12.1b

Note: all solutions to 12.1 are also contained (and vizualised) in the `R-Skript-Gradient-descent.R` file that can be found on moodle.

For later reference, our original function value is $f(x_0, y_0) = \frac{3^2}{4} + 2^2 = 6.25$

In the case of two-dimensional input vectors $(x, y)^T$, the gradient update rule is $\binom{x_n}{y_n} = \binom{x_{n-1}}{y_{n-1}} - \alpha\nabla f(x_{n-1}, y_{n-1})$, thus we calculate

$$\nabla f(x_0, y_0) = \begin{pmatrix} 0.5 \cdot 3 \\ 2 \cdot 2 \end{pmatrix} = \begin{pmatrix} 0.5 \cdot 3 \\ 2 \cdot 2 \end{pmatrix} = \begin{pmatrix} 1.5 \\ 4 \end{pmatrix},$$

$$\binom{x_1}{y_1} = \binom{x_0}{y_0} - \alpha \cdot \nabla f \binom{x_0}{y_0} = \binom{3}{2} - 1 \cdot \binom{1.5}{4} = \binom{1.5}{-2}.$$

Just for comparison, the new function value is then

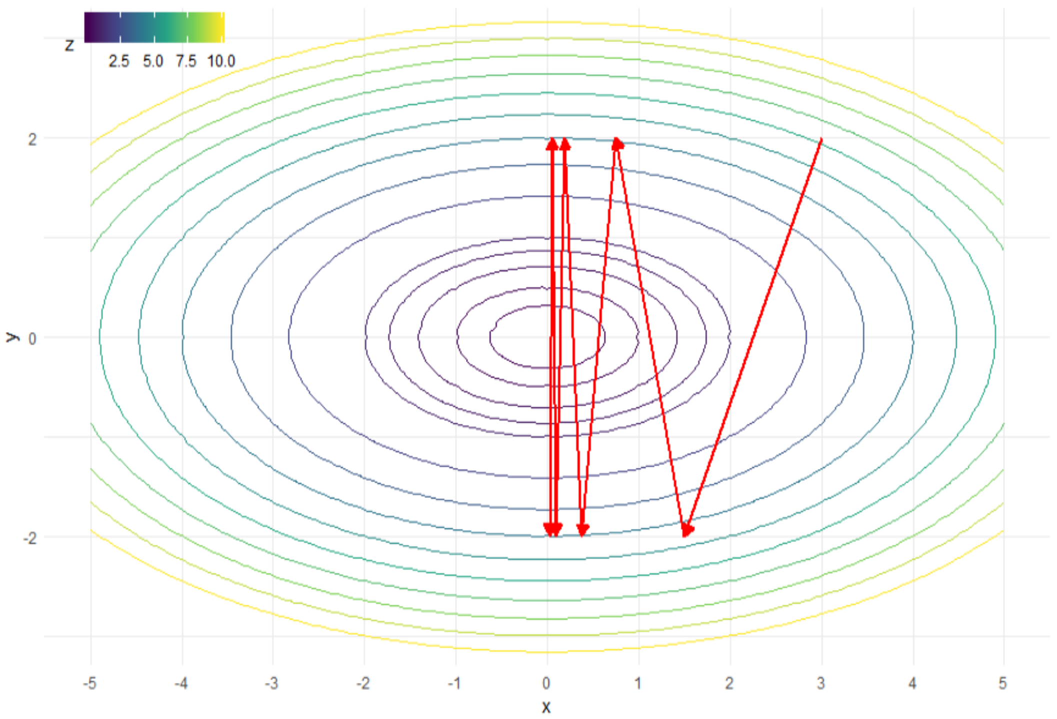
$$f\binom{x_1}{y_1} = \frac{1.5^2}{4} + (-2)^2 = 4.5625$$

Repeating these steps, using the update formula above, we get

$\binom{x_2}{y_2} = \binom{1.5}{-2} - 1 \cdot \binom{0.75}{-4} = \binom{0.75}{2}$, $\quad f\binom{x_2}{y_2} \approx 4.141$

$\binom{x_3}{y_3} = \binom{0.75}{2} - 1 \cdot \binom{0.375}{4} = \binom{0.375}{-2}$, $\quad f\binom{x_3}{y_3} \approx 4.035$

However, visualizing the first 7 steps, we see that the sequence does not converge to the minimum at $(0,0)^T$.



In fact,

$$\nabla f(0,2) = \begin{pmatrix} 0 \\ 4 \end{pmatrix} = -\nabla f(0,-2),$$

and thus, in the limit, our method will oscillate between these two points, never achieving a function value of less than 4 and never reaching the optimum.

c) **[Homework]** Repeat b) but with a learning rate rule that is guaranteed to converge:

$$\alpha_n = \frac{1}{n}$$

Solution 12.1c

The gradient update formula with dynamic step size α_n is

$$\begin{pmatrix} x_n \\ y_n \end{pmatrix} = \begin{pmatrix} x_{n-1} \\ y_{n-1} \end{pmatrix} - \alpha_n \nabla f(x_{n-1}, y_{n-1}),$$

as $\alpha_1 = \frac{1}{1} = 1$, the first step for calculating $(x_1, y_1)^T$ is identical to b). Afterwards, we get

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} 1.5 \\ -2 \end{pmatrix} - 1/2 \cdot \begin{pmatrix} 0.75 \\ -4 \end{pmatrix} = \begin{pmatrix} 1.125 \\ 0 \end{pmatrix}, \qquad f\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} \approx 0.316$$

$$\begin{pmatrix} x_3 \\ y_3 \end{pmatrix} \approx \begin{pmatrix} 1.125 \\ 0 \end{pmatrix} - 1/3 \cdot \begin{pmatrix} 0.563 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.936 \\ 0 \end{pmatrix}, \qquad f\begin{pmatrix} x_3 \\ y_3 \end{pmatrix} \approx 0.220$$

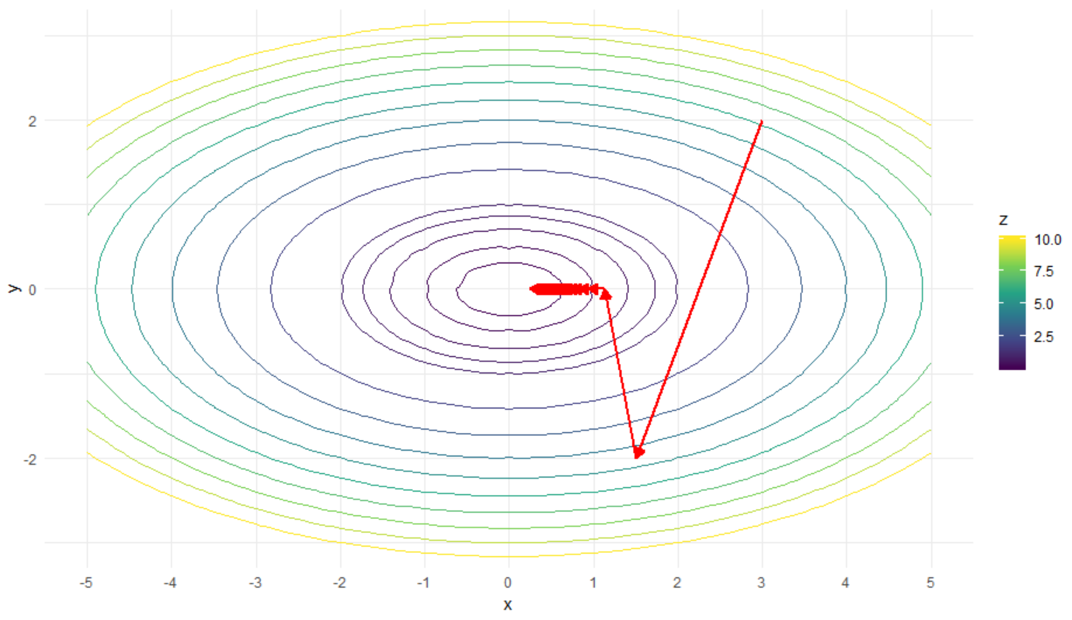
$$\begin{pmatrix} x_4 \\ y_4 \end{pmatrix} \approx \begin{pmatrix} 0.936 \\ 0 \end{pmatrix} - 1/4 \cdot \begin{pmatrix} 0.469 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.820 \\ 0 \end{pmatrix}, \qquad f\begin{pmatrix} x_4 \\ y_4 \end{pmatrix} \approx 0.168$$

We see that with this choice, we have broken the oscillation that we saw in the previous exercise. However, we also observe that the first coordinate is diminishing much slower than we saw before with a learning rate of 1.

In fact, if we continue, we will get

$$\begin{pmatrix} x_{50} \\ y_{50} \end{pmatrix} \approx \begin{pmatrix} 0.239 \\ 0 \end{pmatrix}, \quad \begin{pmatrix} x_{100} \\ y_{100} \end{pmatrix} \approx \begin{pmatrix} 0.169 \\ 0 \end{pmatrix}, \quad \begin{pmatrix} x_{100} \\ y_{100} \end{pmatrix} \approx \begin{pmatrix} 0.076 \\ 0 \end{pmatrix}$$

Depending on our requirements, this rate of convergence might be much too slow. Visualization of 50 steps:



d) **[Tutorial]** Starting from (x_1, y_1) found in b), perform 2 additional steps of the *momentum method*, with $\beta = 0.25$ and $\alpha = 1$. Assume that $d_1 = \nabla f(x_0, y_0)$

Recall from tutorial slides:

$$d_n = \beta d_{n-1} + \alpha \nabla f(x_{n-1}), \qquad x_n = x_{n-1} - d_n.$$

Solution 12.1d

If we had started with $d_0 = (0,0)^T$, calculating (x_1, y_1) is identical to (b) and (c), since

$$d_1 = \beta \cdot 0 + \alpha \nabla f \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \nabla f \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} 1.5 \\ 4 \end{pmatrix}$$

Continuing, with the second step, we get

$$d_2 = \beta \cdot d_1 + \alpha \, \nabla f \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = 0.25 \cdot \begin{pmatrix} 1.5 \\ 4 \end{pmatrix} + \begin{pmatrix} 0.75 \\ -4 \end{pmatrix} = \begin{pmatrix} 1.125 \\ -3 \end{pmatrix}$$

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} - d_2 = \begin{pmatrix} 1.5 \\ -2 \end{pmatrix} - \begin{pmatrix} 1.125 \\ -3 \end{pmatrix} = \begin{pmatrix} 0.375 \\ 1 \end{pmatrix}, \qquad f\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} \approx 1.035$$

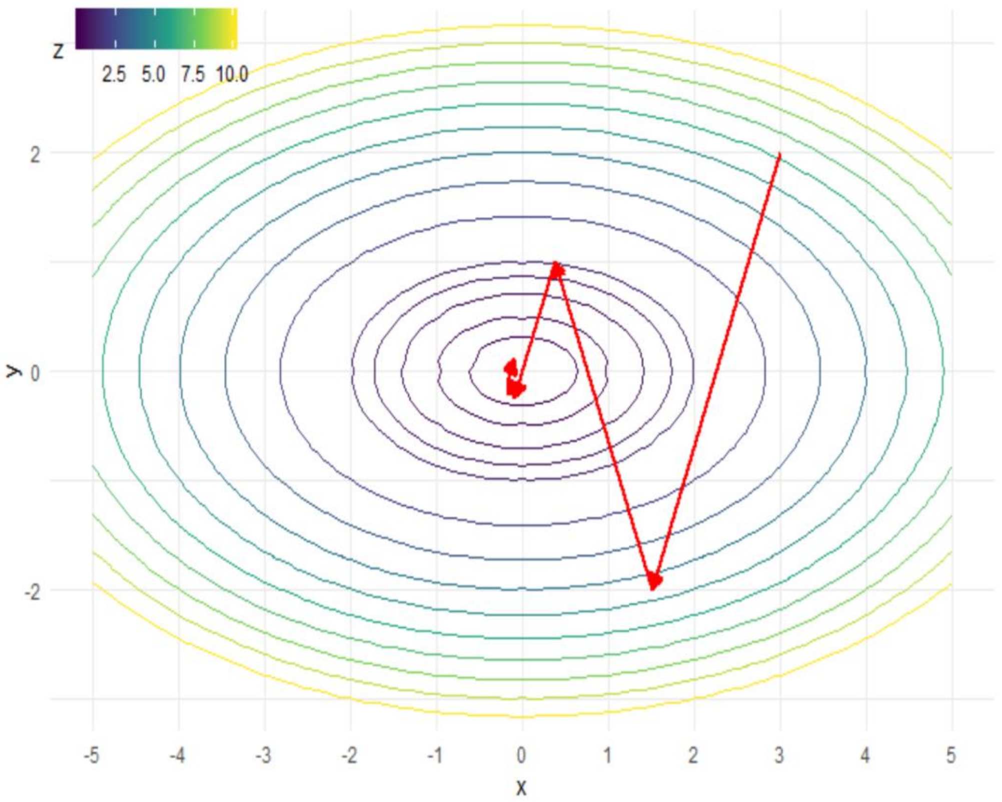
$$d_3 = 0.25 \cdot d_2 + \nabla f \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = 0.25 \cdot \begin{pmatrix} 1.125 \\ -3 \end{pmatrix} + \begin{pmatrix} 0.1875 \\ 2 \end{pmatrix} \approx \begin{pmatrix} 0.469 \\ 1.25 \end{pmatrix}$$

$$\begin{pmatrix} x_3 \\ y_3 \end{pmatrix} = \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} - d_3 \approx \begin{pmatrix} 0.375 \\ 1 \end{pmatrix} - \begin{pmatrix} 0.469 \\ 1.25 \end{pmatrix} = \begin{pmatrix} -0.094 \\ -0.25 \end{pmatrix}, \qquad f\begin{pmatrix} x_3 \\ y_3 \end{pmatrix} \approx 0.065$$

With momentum, the oscillation from b) has been broken, while also admitting larger step-sizes than those in c). In particular, when d and ∇f point into the same direction, momentum yields to larger step-length than the gradient alone, if they point into opposite directions, momentum shortens the step.

Compared with the steps taken in b), in the second and third step, the momentum term dampened the oscillation in the y component and accelerated the steps in the x component.

5 Steps visualized:



Part II: Neural Networks

The following information will be used for exercises 12.2, 12.3 and 12.4:

Consider the following feed-forward neural network that consists of

- An input layer ($l = 0$) representing two-dimensional points

$$a^{[0]} = \left(a_1^{[0]}, a_2^{[0]}\right)^T \in \mathbb{R}^2$$

- A hidden layer $l = 1$ with 2 hidden nodes and sigmoid activation function $g^{[1]}$
- An output layer $l = 2$ with one node and sigmoid activation function $g^{[2]}$.

In the following, we will train the NN for binary classification on a data set. For a given input-output pair (x, y), we will write x for an input to the NN, and model $y \approx \hat{y} = a^{[2]}$ and evaluate the model using the *cross-entropy loss*

$$\ell(y, \hat{y}) = -[y \ln \hat{y} + (1 - y) \ln(1 - \hat{y})]$$

Note: In the following exercises, w*e will write inputs as column vectors, and likewise consider a data matrix where each column is an observation, and each row is a feature. This notation differs from the other weeks of the course (where we had features in columns, observations in rows), but doing so will greatly simplify the mathematical notation.*

Exercise 12.2 Neural Networks: Forward Pass

a) **[Tutorial]** Write down the formulas for the forward pass of this neural network. How many trainable parameters does it have?

Solution:

We will consider the case of a single input observation. Let $x = \begin{pmatrix} a_1^{[0]} \\ a_2^{[0]} \end{pmatrix}$ be the input vector associated with that observation. Then the forward pass through the neural network can be written using the following four equations:

$$z^{[1]} = W^{[1]}x + b^{[1]}, \qquad a^{[1]} = g^{[1]}\left(z^{[1]}\right) = \sigma\left(z^{[1]}\right)$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}, \qquad a^{[2]} = g^{[2]}\left(z^{[2]}\right) = \sigma(z^{[2]})$$

Where $\sigma(z) = \frac{1}{1+\exp(-z)}$ is the sigmoid function. $\hat{y} = a^{[2]}$ is the output of the NN.

The free parameters of the Neural Net are the entries $W^{[1]},\ b^{[1]},\ W^{[2]},\ b^{[2]}$, so the question asks us to find their dimensions.

As both the input and hidden layer have 2 units, the shapes of both x and $a^{[1]}$ are 2x1 (i.e. column vectors with two entries, the number of rows corresponds to the number of units in the layer, the number of columns (1) corresponds to the number of observations from the data). From this we can deduct that the dimension of $W^{[1]}$ must be 2x2 because the dimensions are fully determined by the fact that the matrix multiplication must have correct dimensions, i.e.

$$\text{first dimension of } W^{[1]} = \text{first dimension of } z^{[1]} = \#\{\text{nodes in layer } 1\} = 2$$
$$\text{second dimension of } W^{[1]} = \text{first dimension } of\ x = \#\{\text{nodes in layer } 0\} = 2$$

Similarly, for the addition to be valid, $b^{[1]}$ must have the same shape as $z^{[1]}$ and $W^{[1]}x$, it its shape is therefore 2x1.

Repeating the same deliberations for the second layer we get shapes of 1x2 for $W^{[2]}$ and 1x1 for $b^{[2]}$.

With parameters

$$W^{[1]} \in \mathbb{R}^{2\times 2},\ b^{[1]} \in \mathbb{R}^{2},\ W^{[2]} \in \mathbb{R}^{1\times 2}, \text{ and } b^{[2]} \in \mathbb{R}^{1},$$

the network thus has a total of $4 + 2 + 2 + 1 = 9$ trainable parameters.

b) **[Homework]** Given the following dataset of $n = 4$ observations (in columns) and initial parameters, perform one forward pass (calculate the *empirical risk* $\mathcal{L}$).

$$X = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}, \qquad Y = \begin{pmatrix} 0 & 1 & 1 & 1 \end{pmatrix}$$

$$W^{[1]} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad b^{[1]} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \qquad W^{[2]} = \begin{pmatrix} 1 & -1 \end{pmatrix}, \qquad b^{[2]} = (0)$$

Solution

To calculate the forward pass, we apply the four formulas for $z^{[1]}, a^{[1]}, z^{[2]}, a^{[2]}$ found in (a). In fact, when applying this formula to *multiple observations at once*, we can write observations in columns as in the data matrix X and still apply the same equations. Instead of vectors $z^{[1]}, a^{[1]}, z^{[2]}, a^{[2]}$ we will then get matrices $Z^{[1]}, A^{[1]}, Z^{[2]}, A^{[2]}$ that have the individual results for each observation in their columns:

$$Z^{[1]} = W^{[1]}X + b^{[1]} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}.$$

(Here we've used the following slight abuse of notation: when adding a vector to a matrix (with matching number of rows), we mean adding it to each column of that matrix separately.)

We continue by applying the activation function to each element of $Z^{[1]}$:

$$A^{[1]} = \sigma(Z^{[1]}) = \sigma\begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{1+e^{-0}} & \frac{1}{1+e^{-0}} & \frac{1}{1+e^{-1}} & \frac{1}{1+e^{-1}} \\ \frac{1}{1+e^{-0}} & \frac{1}{1+e^{-1}} & \frac{1}{1+e^{-0}} & \frac{1}{1+e^{-1}} \end{pmatrix}$$

$$\approx \begin{pmatrix} 0.5 & 0.5 & 0.731 & 0.731 \\ 0.5 & 0.731 & 0.5 & 0.731 \end{pmatrix}.$$

Doing the same for the second layer, we get:

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]} = (1 \quad -1)\begin{pmatrix} 0.5 & 0.5 & 0.731 & 0.731 \\ 0.5 & 0.731 & 0.5 & 0.731 \end{pmatrix} + 0$$

$$= (0 \quad -0.231 \quad 0.231 \quad 0),$$

$$A^{[2]} = \sigma((0 \quad -0.231 \quad 0.231 \quad 0)) \approx (0.5 \quad 0.442 \quad 0.558 \quad 0.5).$$

Using the loss function given in exercise b), we can calculate the loss of the network output $A^{[2]}$ element wise:

$$\begin{aligned} \ell(Y, A^{[2]}) &\approx -[Y * \ln A^{[2]} + (1-Y) * \ln(1 - A^{[2]})] \\ &= -[\ (0 \quad 1 \quad 1 \quad 1) * \ln((0.5 \quad 0.442 \quad 0.558 \quad 0.5)) \\ &\quad + (1 \quad 0 \quad 0 \quad 0) * \ln((0.5 \quad 0.558 \quad 0.442 \quad 0.5))] \\ &= -[\ (0 \quad 1 \quad 1 \quad 1) * (-0.693 \quad -0.815 \quad -0.584 \quad -0.693) \\ &\quad + \ (1 \quad 0 \quad 0 \quad 0) * (-0.693 \quad -0.584 \quad -0.815 \quad -0.693)] \\ &= -[(0 \quad -0.815 \quad -0.584 \quad -0.693) + (-0.693 \quad 0 \quad 0 \quad 0)] \\ &= (0.693 \quad 0.815 \quad 0.584 \quad 0.693). \end{aligned}$$

The *risk* is given by the average loss in each observation:

$$\mathcal{L}(Y, A^{[2]}) = \frac{1}{n}\sum_{i=1}^{n} \ell\left(y_{(i)}, a_{(i)}^{[2]}\right) = \frac{1}{4} \cdot (0.693 + \ 0.815 + 0.584 + 0.693) = 0.696$$

Exercise 12.3 Neural Networks: Backpropagation

In this exercise, we will continue to work with the same neural network as in Exercise 12.2.

a) **[Tutorial]** Calculate the partial derivatives $\frac{\partial \ell}{\partial w_1^{[2]}}$ and $\frac{\partial \ell}{\partial w_2^{[2]}}$ that will be used to update $W^{[2]}$ in backpropagation.

   Hint: You may use the following derivative of the sigmoid function $\sigma(\cdot)$ without proof: $\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$

Solution 12.3a:
We apply backpropagation to find the desired derivatives with respect to $W^{[2]}$: While ℓ is not directly expressed as a function of $W^{[2]}$, we have

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}, \quad a^{[2]} = \sigma\left(z^{[2]}\right), \quad \ell\left(y, a^{[2]}\right) = -[y \ \ln a \ + (1-y) \ \ln(1-a)]$$

We can thus find the desired derivative by applying the chain rule twice:

$$\frac{\partial \ell}{\partial w_1^{[2]}} = \frac{\partial \ell}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial w_1^{[2]}} = \frac{\partial \ell}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{\,[2]}}{\partial w_1^{[2]}}$$

We calculate these derivatives separately by deriving the three functions above:

$$da^{[2]} = \frac{\partial \ell\left(a^{[2]}, y\right)}{\partial a^{[2]}} = -\left[y \cdot \frac{1}{a^{[2]}} + (1-y)\frac{-1}{1-a^{[2]}}\right] = \frac{1-y}{1-a^{[2]}} - \frac{y}{a^{[2]}}$$

$$\frac{\partial a^{[2]}}{\partial z^{[2]}} = \sigma'\left(z^{[2]}\right) = \sigma\left(z^{[2]}\right) \cdot \left(1 - \sigma\left(z^{[2]}\right)\right) = a^{[2]} \cdot (1 - a^{[2]})$$

where we used the equation given for the sigmoid derivative and the fact that the sigmoid is the activation function of the second layer. We can combine the first two derivatives we calculated to get

$$dz^{[2]} = \frac{\partial \ell}{\partial z^{[2]}} = \frac{\partial \ell}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} = \left[\frac{1-y}{1-a^{[2]}} - \frac{y}{a^{[2]}}\right] \cdot a^{[2]} \cdot \left(1 - a^{[2]}\right)$$

$$= a^{[2]}(1-y) - \left(1 - a^{[2]}\right)y = a^{[2]} - a^{[2]}y - y + a^{[2]}y = a^{[2]} - y$$

To calculate the last derivative, we use the fact that $z^{[2]}$ is scalar (only one unit in output layer) to rewrite its equation without matrix notation:

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} = w_1^{[2]}a_1^{[1]} + w_2^{[2]}a_2^{[1]} + b^{[2]}$$

Thus

$$\frac{\partial z^{[2]}}{\partial w_1^{[2]}} = \mathrm{a}_1^{[1]}, \quad \frac{\partial z^{[2]}}{\partial w_2^{[2]}} = \mathrm{a}_2^{[1]}$$

Combining the results of the individual derivatives, we get

$$\begin{aligned} dW^{[2]} &= \begin{pmatrix} \frac{\partial \ell}{\partial w_1^{[2]}} & \frac{\partial \ell}{\partial w_2^{[2]}} \end{pmatrix} = dz^{[2]} \cdot \begin{pmatrix} \frac{\partial z^{[2]}}{\partial w_1^{[2]}} & \frac{\partial z^{[2]}}{\partial w_2^{[2]}} \end{pmatrix} \\ &= \left(a^{[2]} - y\right) \cdot \left(a_1^{[1]} \quad a_2^{[1]}\right) = \left(a^{[2]} - y\right) \cdot a^{[1]^T}. \end{aligned}$$

b) **[Homework]** With the data and parameters from 12.2b, now perform a partial backward pass and update the parameter W^2 using a gradient descent with learning rate $\alpha = 1$. Perform another forward pass on the full data. Did the risk decrease?

Solution 12.2b:

We can perform the update step using the gradient found in exercise 12.2a).

$$dW^{[2]} = dz^{[2]} \cdot \begin{pmatrix} \frac{\partial z^{[2]}}{\partial w_1^{[2]}} & \frac{\partial z^{[2]}}{\partial w_2^{[2]}} \end{pmatrix} = \left(a^{[2]} - y\right) \cdot a^{[1]^T}$$

In exercise a), however, we only considered the backward pass for a single observation (x, y). When dealing with multiple observations at once, we need some minor adjustments:

1. Replace $dz^{[2]}$ with $dZ^{[2]} = A^{[2]} - Y$
2. Replace $a^{[1]^T}$ with $A^{[1]^T}$
3. We are actually interested in the derivative of the *risk*, rather than the univariate *loss* for each observation. The *risk* is $\frac{1}{n}\sum(\text{losses})$, so the calculations above would give us the sum of gradients for individual observations. However, we're interested in the averages, so we must add a factor of $1/n$.

In total we get (using the intermediate results from 12.2b for the forward pass):

$$\begin{aligned} dW^{[2]} &= \frac{\partial \mathcal{L}}{\partial W^{[2]}} = \frac{1}{n}\left(A^{[2]} - Y\right)A^{[1]^T} \\ &= \frac{1}{4}\left((0.5 \quad 0.442 \quad 0.558 \quad 0.5) - (0 \quad 1 \quad 1 \quad 1)\right) \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.731 \\ 0.731 & 0.5 \\ 0.731 & 0.731 \end{pmatrix} \\ &= \frac{1}{4}\left((0.5 \quad -0.558 \quad -0.442 \quad -0.5)\right) \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.731 \\ 0.731 & 0.5 \\ 0.731 & 0.731 \end{pmatrix} \\ &= \frac{1}{4}\,(-0.718 \quad -0.744) = (-0.179 \quad -0.186) \end{aligned}$$

We can thus perform one gradient update step:

$$W_{new}^{[2]} = W^{[2]} - \alpha \cdot dW^{[2]} = (1 \quad -1) - 1 \cdot (-0.179 \quad -0.186) = (1.18 \quad -0.814)$$

To check whether the risk decreased, let's perform another forward pass. Everything up to $A^{[1]}$ is the same as in exercise 12.2b) because we didn't change anything in the first layer. In layer 2 we get:

$$Z_{new}^{[2]} = W_{new}^{[2]} A^{[1]} + b^{[2]} = (1.18 \quad -0.814) \begin{pmatrix} 0.5 & 0.5 & 0.731 & 0.731 \\ 0.5 & 0.731 & 0.5 & 0.731 \end{pmatrix} + 0$$

$$= (0.183 \quad -0.005 \quad 0.455 \quad 0.267)$$

$$A_{new}^{[2]} = \sigma\big((0.183 \quad -0.005 \quad 0.455 \quad 0.267)\big) \approx (0.546 \quad 0.499 \quad 0.612 \quad 0.566)$$

Resulting in losses of

$$\ell_{new} = \ell\left(Y, A_{new}^{[2]}\right) = (0.789 \quad 0.696 \quad 0.491 \quad 0.568)$$

And a new risk/average loss of

$$\mathcal{L}_{new} = \frac{1}{4}(.789 + .696 + .491 + .568) = 0.635$$

As the previous risk (from 12.2b) was 0.696, we see that even updating just a single matrix of parameters improved the neural network.

Exercise 12.4 [Homework]

This is an optional, advanced programming exercise. You don't need to be able to solve it yourself, but you should nevertheless study the provided solution file to familiarize yourself with the workings of NNs in practice.

a) **[optional advanced programming exercise]**
Implement the forward and backward pass for this neural network in R.

Train the neural network on the dataset from 12.2b by performing 100 forward and backward passes and updating the parameters using gradient descent. Use learning rate $\alpha = 1$.

Hint: *For n observations, the full parameter derivatives of the risk function are given by:*

$$dW^{[1]} = \frac{1}{n} dZ^{[1]} X^T, \qquad db^1 = \frac{1}{n}\text{rowSums}(dZ^{[1]}),$$
$$dW^{[2]} = \frac{1}{n} dZ^{[2]} {A^{[1]}}^T, \qquad db^2 = \frac{1}{n}\text{rowSums}(dZ^{[2]})$$

with

$$dZ^{[1]} = {W^{[2]}}^T dZ^{[2]} * A^{[1]} * (1 - A^{[1]}), \quad dZ^{[2]} = A^{[2]} - Y$$

where $*$ is element-wise multiplication and for a given expression F we use the shorthand notation $dF = {}^{\partial \mathcal{L}}/_{\partial F}$.

b) *Now apply the same neural network to the following, more complex dataset:*

$$X = \begin{pmatrix} 0 & 0 & 0 & 0.5 & 0.5 & 0.5 & 1 & 1 & 1 \\ 0 & 0.5 & 1 & 0 & 0.5 & 1 & 0 & 0.5 & 1 \end{pmatrix},$$

$$Y = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Can the NN still be trained to fit this dataset? If no, why not? What would you change?

Solution to 12.4:

See R-script-NN.R
Note: When working with neural networks in practice, you would usually use some library that provides "autodifferentiation" features, like PyTorch, tensorflow, JAX or similar. With these libraries, when you specify the formulas for the forward pass, a computational graph will be built and the software will implement the derivatives for the backward pass automatically. In this R-script, however, we show the backward computations by hand, to show what is happening under the hood.