Lab 3

Welcome to the third lab. The first exercise is a simple implementation of gradient descent, the second exercise is a primer on backpropagation with neural networks of 1, 2 and 3 layers, and the third exercise shows the importance of a good weight initialization.

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:
   - At the t-th iteration, use a step size of $\alpha t^{-1/2}$
   - Can you find a way to prematurely stop the optimization when you are close to the optimum?

```
func.value = function(x) {
  # TODO compute the value of f at x
}

func.gradient = function(x) {
  c(
    # TODO compute the gradienf of f at x
  )
}

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

Does it match what you computed?

```
gradient_descent_optimizer = function(x0, func, grad, max_steps, lrate) {
  # TODO use a for loop to do gradient descent
}

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

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?

Exercise 2

This exercise is about computing gradients with the chain rule, with pen and paper.

1. Show that the derivative of the sigmoid $\sigma(x) = (1 + e^{-x})^{-1}$ is $\sigma(x)(1 - \sigma(x))$
2. Consider the output of a neural network with one layer, $\mathbf{y} = \phi(\mathbf{z}) = \phi(\mathbf{W}\mathbf{x} + \mathbf{b})$ with $\phi(\cdot)$ an arbitrary activation function:
   - Compute the partial derivative of z_i with respect to x_j, w_{jk} and b_j.
   - Compute the partial derivative of y_i wit respect to x_j, w_{jk} and b_j.
3. Consider two layers of a neural network: $\mathbf{y} = \phi_1(\mathbf{W}\phi_2(\mathbf{V}\mathbf{x} + \mathbf{c}) + \mathbf{b})$:
   - Compute the partial derivative of y_i with respect to x_j, v_{jk} and c_j

4. Consider three layers of a neural network $\mathbf{y} = \phi_1(\mathbf{W}\phi_2(\mathbf{V}\phi_3(\mathbf{U}\mathbf{x} + \mathbf{d}) + \mathbf{c}) + \mathbf{b})$:
   - Compute the partial derivative of y_i with respect to x_j, u_{jk} and d_j.

Exercise 3

This exercise shows that proper initialization of the weights of a neural network is crucial.

1. Show, for a feed-forward neural network with tanh activation on the hidden layers, linear output activation and mean squared error loss function, that the origin in parameters space is a stationary point of the error function.
2. Consider again a feed-forward neural network with tanh activation on the hidden layers, and consider both inputs and weights as i.i.d. random variables with zero mean and a certain variance $Var[x]$ and $Var[w^i]$ (this is the variance of each element of the weight matrix of the i-th layer). Furthermore, assume that all inputs and hidden activations are small enough, so that the tanh is approximately linear. For this exercise, forget about biases. The goal of this exercise is to find the best variance of the weights for a random initialization of a neural network.
   - Show that, for X and Y independent random variables, $Var[XY] = Var[X]E[Y]^2 + Var[Y]E[X]^2 + Var[X]Var[Y]$, where $E[\cdot]$ denotes expectation.
   - Compute the variance of z_j^i, the i-th element of the j-th hidden layer after activation. First, find a recursive formula in terms of the variance of the previous layer, then unroll it to find a closed form in terms of the variance of the inputs, of the weights, and the number of neurons in each layer.
   - Compute the variance of $\partial\mathcal{L}/\partial z_j^i$. Stop at $Var[\partial\mathcal{L}/\partial z_d]$, where d is the number of layers of the network. Again, derive a closed formula.
   - Compute the variance of $\partial\mathcal{L}/\partial w_{ij}^k$ with a closed formula, where w_{ij}^k is the weight connecting the i-th neuron of the k-th layer to the j-th neuron of the $(k+1)$-th layer.
   - As you can see, variances are multiplied together both in the forward pass and in the backward pass. Why is this a problem as the size and number of layers grows?
   - In order to avoid these troubles, we would like to keep the variances constant throughout the network, i.e. have $Var[z_i] = Var[z_j]$ and $Var[\partial\mathcal{L}/\partial z_i] = Var[\partial\mathcal{L}/\partial z_j]$ for all pairs of layers i and j. Use the formulas you derived earlier to find the variance of the weights that satisfies these constraints; as you can see, this is only possible when all layers have the same number of neurons. As a compromise between these two constraints, the variance of every weight should depend on the average number of neurons of the two layers they connect.
   - Finally, assume the weights of layer i are initialized from an uniform distribution in the interval $[-a_i, a_i]$. Find a suitable value for a_i so that $Var[w^i]$ is the same as what you found in the previous point.
   - Congratulations, you have just discovered the Glorot (aka Xavier) initialization, one of the most widely used formulas to initialize weights in deep learning!