Basic Math Facts that We Studied in Class

1 Optimization

To find the values x that maximize or minimize a function $f(x)$ of one variable, we find all the points at which the derivative $\frac{df}{dx}$ is equal to 0, compute the value of the function at all these points, and select a point at which this value is, correspondingly, the largest or the smallest.

To find the values $x_1, \ldots, x_n$ at which a function $f(x_1, \ldots, x_n)$ of several variables attains, e.g., its largest value, we take into account that at these values, a change in each of the variables will decrease the value of the function, so it attains maximum over each x_i. Thus, all the partial derivatives must be equal to 0:

$$\frac{\partial f}{\partial x_1} = 0, \ldots, \frac{\partial f}{\partial x_n} = 0.$$

Similar equations hold if we want to find the values that minimize a given function.

2 Why Linear Interpolation

Suppose that we have several equidistance points

$$x_0, x_1 = x_0 + \Delta x, x_2 = x_1 + \Delta x = x_0 + 2\Delta x, \ldots, x_i = x_0 + i \cdot \Delta x, \ldots$$

all the way to $x_n = x_0 + n \cdot \Delta x$. Let us consider the situation when:

- we know the values y_0 and y_n of a function $f(x)$ at the two endpoints points x_0 and x_n, and
- we want to estimate the values $y_i = f(x_i)$ of this function for all other points.

In many practical situations, we know that the dependence $y = f(x)$ is smooth, which means that when the values of x are close, the corresponding values of y are also close. For our problem, this means that

$$y_1 \approx y_0, y_2 \approx y_1, \ldots, \text{ and } y_n \approx y_{n-1}.$$

In other words, we want all the differences $e_i = y_i - y_{i-1}$ to be close to 0: $e_1 \approx 0$, $e_2 \approx 0$, $\ldots$, $e_n \approx 0$. Thus, we want the point $e = (e_1, \ldots, e_n)$ be close

to the point $(0, \ldots, 0)$. The square of the distance between these two points is equal to $\sum_{i=1}^{n} e_i^2$. If we want the smoothest possible interpolation, we need to minimize this sum, i.e., minimize the expression

$$e_1^2 + \ldots + e_n^2 =$$

$$(y_1 - y_0)^2 + (y_2 - y_1)^2 + \ldots + (y_i - y_{i-1})^2 + (y_{i+1} - y_i)^2 + \ldots + (y_n - y_{n-1})^2.$$

Here, the unknowns are $y_1, y_2, \ldots, y_{n-1}$.

Differentiating the above expression with respect to y_1 and equating the derivative to 0, we get

$$2(y_1 - y_0) + 2(y_2 - y_1) \cdot (-1) = 0.$$

Dividing both sides by 2 and moving the negative term to the other side, we get

$$y_1 - y_0 = y_2 - y_1.$$

Similarly, differentiating with respect to y_i and equating the derivative to 0, we get

$$2(y_i - y_{i-1}) + 2(y_{i+1} - y_i) \cdot (-1) = 0.$$

Dividing both sides by 2 and moving the negative term to the other side, we get

$$y_i - y_{i-1} = y_{i+1} - y_i.$$

Thus, we have

$$y_1 = y_0 = y_2 - y_1 = y_3 - y_2 = \ldots,$$

i.e., the difference between the two neighboring values of y_i is the same. Let us denote this different by Δy. Thus, $y_1 = y_0 + \Delta y$, $y_2 = y_1 + \Delta y = y_0 + 2\Delta y$, $\ldots$, and, in general, $y_i = y_0 + i \cdot \Delta y$.

From $x_i = x_0 + i \cdot \Delta x$, we conclude that $i = \dfrac{x_i - x_0}{\Delta x}$. Substituting this value i into the formula for y_i, we get

$$y_i = y_0 + \frac{x_i - x_0}{\Delta x} \cdot \Delta y = x_i \cdot \frac{\Delta y}{\Delta x} + \left(y_0 - x_0 \cdot \frac{\Delta y}{\Delta x} \right),$$

i.e., a linear formula $y = a \cdot x + b$, where

$$a = \frac{\Delta y}{\Delta x} \text{ and } y_0 - x_0 \cdot \frac{\Delta y}{\Delta x}.$$

3 Constraint Optimization

In many practical situations, we need to find the value that maximize a given function $f(x_1, \ldots, x_n)$ under constraint $g(x_1, \ldots, x_n) = 0$. (A more general constraint $a(x) = b(x)$ can be described in this form for $g(x) = a(a) - b(x)$.) This *constraint optimization* problem can be solved as follows:

- first, we consider an unconstrained optimization problem of optimizing the auxiliary function $f(x_1, \ldots, x_n) + \lambda \cdot g(x_1, \ldots, x_n)$, where λ is an unknown parameter; this parameter is known as *Langange multiplier*; as a result, we get a solution $x_i(\lambda)$ depending on λ;
- we plug in these solutions into the constraint $g(x_1, \ldots, x_n) =$, thus getting an equation for λ; we solve this equation, fuinding λ;
- finally, we plug the resulting value λ into the expression $x_i(\lambda)$, thus finding the solutio to the original constraint optimization problem.

If we have several constraints $g_1(x) = 0$, $\ldots$, $g_m(x) = 0$, then we need to similarly consider a function $f(x) + \lambda_1 \cdot g_1(x) + \ldots + \lambda_m \cdot g_m(x)$ with several parameters $\lambda_1, \ldots, \lambda_m$.

4 Linearization

By definition, a derivative is a limit:

$$\frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}.$$

By definition, a limit means that when Δx is small, the two terms are almost equal, i.e., that

$$\frac{df}{dx} \approx \frac{f(x + \Delta x) - f(x)}{\Delta x}.$$

Multiplying both sides by Δx and moving $f(x)$ to the right-hand side, we conclude that

$$f(x + \Delta x) \approx f(x) + \frac{df}{dx} \cdot \Delta x.$$

If we have a function $f(x_1, x_2)$ of two variables, we can use the derivative with respect to x_1 to conclude that

$$f(x_1 + \Delta x_1, x_2 + \Delta x_2) \approx f(x_1, x_2 + \Delta x_2) + \frac{\partial f}{\partial x_1} \cdot \Delta x_1.$$

Here, we can use the derivative with respect to x_2 and get

$$f(x_1, x_2 + \Delta x_2) \approx f(x_1, x_2) + \frac{\partial f}{\partial x_2} \cdot \Delta x_2.$$

Substituting this expression into the previous formula, we conclude that

$$f(x_1 + \Delta x_1, x_2 + \Delta x_2) \approx f(x_1, x_2) + \frac{\partial f}{\partial x_1} \cdot \Delta x_1 + \frac{\partial f}{\partial x_2} \cdot \Delta x_2.$$

In general, we get

$$f(x_1 + \Delta x_1, \ldots, x_n + \Delta x_n) \approx f(x_1, \ldots, x_n) + \frac{\partial f}{\partial x_1} \cdot \Delta x_1 + \ldots + \frac{\partial f}{\partial x_n} \cdot \Delta x_n. \quad (1)$$

This expression is known as *linearization* (or the first terms in Taylor series).

5 Gradient Descent

If we can solve the system of equations $\frac{\partial f}{\partial x_i} = 0$, we can find the minimum of a function. However, often, these equations are too complicated and not easy to solve. In this case, a reasonable idea is to minimized a linearized version (1) of the objective function. We start with some point $(x_1, \ldots, x_n)$ and we want to find a point $x_i' = x_i + \Delta x_i$ in some small δ-vicinity of this point for which the linearized expression (1) is the smallest possible. Thus, we minimize the expression (1) under the constraint

$$(\Delta x_1)^2 + \ldots + (\Delta x_n)^2 = \delta^2.$$

For this problem, the Lagrange multiplier method leads to the unconstrained optimization problem:

$$f(x_1 + \Delta x_1, \ldots, x_n + \Delta x_n) \approx f(x_1, \ldots, x_n) + \frac{\partial f}{\partial x_1} \cdot \Delta x_1 + \ldots + \frac{\partial f}{\partial x_n} \cdot \Delta x_n +$$

$$\lambda \cdot \left((\Delta x_1)^2 + \ldots + (\Delta x_n)^2 - \delta^2\right) \to \min.$$

Differentiation this expression with respect to Δx_i and equating the derivative to 0, we get

$$\frac{\partial f}{\partial x_i} + 2\lambda \cdot \Delta x_i = 0,$$

hence

$$\Delta x_i = -\alpha \cdot \frac{\partial f}{\partial x_i},$$

where we denoted $\alpha \stackrel{\text{def}}{=} \frac{1}{2\lambda}$.

Thus, based on the original point x_i, we move to a new point

$$x_i' = x_i - \alpha \cdot \frac{\partial f}{\partial x_i}.$$

Based on the point x_i', we similarly get a new point x_i'', etc. This optimization technique is known as *gradient descent.*

6 Backpropagation

How a neural network works. As we mentioned in class, the simplest 3-layer neural network works as follows:

- first, we have a layer in which each neuron k performs a linear transformation to the inputs $x_1, \ldots, x_n$, and thus, produces a value

$$z_k = \sum_{i=1}^{n} w_{ki} \cdot x_i - w_{k0};$$

- then, we have a non-linear layer in which we apply, to each output z_k of the first layer, a non-linear transformation, resulting in $y_k = s_k(z_k)$, for some function $s_k(z)$; this function is known as the *activation function*;
- finally, we have a linear layer that transforms the values y_k into their linear combination $y = \sum\limits_{k=1}^{K} W_k \cdot y_k - W_0$.

Substituting the expressions for y_k and z_k into this formula, we conclude that

$$y = \sum_{k=1}^{K} W_k \cdot s_k\left(\sum_{i=1}^{n} w_{ki} \cdot x_i - w_{k0}\right) - W_0.$$

Traditional neural networks use sigmoid activation function (borrowed from biological neurons):

$$s_k(z) = s(z) = \frac{1}{1 + \exp(-z)},$$

so

$$y = \sum_{k=1}^{K} W_k \cdot s\left(\sum_{i=1}^{n} w_{ki} \cdot x_i - w_{k0}\right) - W_0.$$

How to train a neural network: idea. We know the values $\widetilde{y}^{(p)}$, $p = 1, \ldots, P$, corresponding to given inputs $x^{(p)} = \left(x_1^{(p)}, \ldots, x_n^{(p)}\right)$. We want to find the weights w_{ki} and W_k for which the neural network, for each of given inputs, will produce the value as close to the desired output $y^{(p)}$ as possible. In other words, for each *pattern* $(x_1, \ldots, x_n, \widetilde{y})$, we want to minimize the square of the difference

$$J = (y - \widetilde{y})^2.$$

To find the weights that minimize this difference, we will use the gradient descent method. On each step of this method, we replace each original weight w with the new value $w + \Delta w$, where $\Delta w = -\alpha \cdot \dfrac{\partial J}{\partial w}$. To come up with the computational formulas, let us compute the corresponding partial derivatives.

Analysis of the problem. Due to chain rule, for each weight,

$$\frac{\partial J}{\partial w} = 2(y - \widetilde{y}) \cdot \frac{\partial y}{\partial w}.$$

Let us denote the difference $y - \widetilde{y}$ (i.e., the approximation error) by e. Then, we have

$$\frac{\partial J}{\partial w} = 2e \cdot \frac{\partial y}{\partial w},$$

and thus,

$$\Delta w = -2\alpha \cdot e \cdot \frac{\partial y}{\partial w}.$$

For W_0, we have $\dfrac{\partial y}{\partial W_0} = -1$, so

$$\Delta W_0 = 2\alpha \cdot e. \tag{1}$$

For W_k, we have $\dfrac{\partial y}{\partial W_k} = y_k$, so

$$\Delta W_k = -2\alpha \cdot e \cdot y_k.$$

From the computational viewpoint, we can simplify this formula by taking into account that we have already computed $2\alpha \cdot e$ when computing ΔW_0, so we can simply take

$$\Delta W_k = -\Delta W_0 \cdot y_k. \tag{2}$$

For w_{k0}, we have

$$\frac{\partial y}{\partial w_{k0}} = \frac{\partial y}{\partial y_k}\frac{\partial y_k}{\partial w_{k0}} = W_k \cdot \frac{\partial s\left(\sum\limits_{i=1}^{n} w_{ki} \cdot x_i - w_{k0}\right)}{\partial w_{k0}} =$$

$$W_k \cdot s'\left(\sum_{i=1}^{n} w_{ki} \cdot x_i - w_{k0}\right) \cdot (-1).$$

Here, due to the definition of $s(z)$, we have

$$s'(z) = \frac{\exp(-z)}{(1-\exp(-z))^2} = \frac{\exp(-z)}{1-\exp(-z)} \cdot \frac{1}{1-\exp(-z)}.$$

The second term is simply $s(z)$, the first can be computed as

$$1 - s(z) = 1 - \frac{1}{1-\exp(-z)} = \frac{1-(1-\exp(-z))}{1-\exp(-z)} = \frac{\exp(-z)}{1-\exp(-z)},$$

thus $s'(z) = s(z) \cdot (1 - s(z))$. For $z = z_k$, we have $s(z_k) = y_k$, so $s'(z_k) = y_k \cdot (1 - y_k)$, and the formula for Δw_{k0} takes the form

$$\Delta w_{k0} = -2\alpha \cdot e \cdot W_k \cdot y_k \cdot (1 - y_k) \cdot (-1) = 2\alpha \cdot e \cdot W_k \cdot y_k \cdot (1 - y_k).$$

We already know the product $2\alpha \cdot e \cdot y_k$, it is equal to $-\Delta W_k$, so

$$\Delta w_{k0} = -\Delta W_k \cdot W_k \cdot (1 - y_k). \tag{3}$$

Finally, for w_{ki}, we have

$$\frac{\partial y}{\partial w_{ki}} = \frac{\partial y}{\partial y_k}\frac{\partial y_k}{\partial w_{ki}} = W_k \cdot \frac{\partial s\left(\sum\limits_{i=1}^{n} w_{ki} \cdot x_i - w_{k0}\right)}{\partial w_{ki}} =$$

$$W_k \cdot s'\left(\sum_{i=1}^{n} w_{ki} \cdot x_i - w_{k0}\right) \cdot x_i.$$

So,

$$\Delta w_{ki} = -2\alpha \cdot e \cdot W_k \cdot y_k \cdot (1 - y_k) \cdot x_i.$$

We know that $2\alpha \cdot e \cdot W_k \cdot y_k \cdot (1 - y_k) = \Delta w_{k0}$, so all we need to get Δw_{ki} is to multiply it by $-x_i$:

$$\Delta w_{ki} = -\Delta w_{k0} \cdot x_i. \tag{4}$$

Thus, we arrive at the following algorithm – known as *backpropagation*.

Resulting algorithm. We start with random values of the weights. We select some α, e.g., $\alpha = 0.1$.

Then, we cycle over all the patterns: first use the 1st one, then the 2nd one, etc., then again the 1st one, etc., until all the approximation errors become small.

For each pattern $(x_1, \ldots, x_n, \widetilde{y})$, we first compute the values

$$y_k = s\left(\sum_{i=1}^{n} w_{ki} \cdot x_i - w_{k0}\right),$$

then we compute $y = \sum\limits_{k=1}^{K} W_k \cdot y_k - W_0$, and the approximation error $e = y - \widetilde{y}$.

Then, we use formulas (1)–(4) to find the changes to all the weights, and then replace the original weights with the new values $W_0 + \Delta W_0$, $W_k + \Delta W_k$, etc.

After this, we apply the new weights to the next pattern, etc.