Sebastian Bordt

Eberhard Karls Universität Tübingen

Deep Learning

Exercise 1 – Intro to EDF

Release date: October 14, 2026

1 Pen and Paper

1.1 Derivatives

a) Let $\mathbf{f}(\mathbf{x}) = (e^{\sqrt{x_3}}\sin(x_1), \sqrt{\log(x_3) + x_2^2}), \mathbf{f}: \mathbb{R}^3 \mapsto \mathbb{R}^2$. Calculate the matrix of partial derivatives $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x})$.

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \end{bmatrix} = \begin{bmatrix} e^{\sqrt{x_3}}\cos(x_1) & 0 & \frac{e^{\sqrt{x_3}}\sin(x_1)}{2\sqrt{x_3}} \\ 0 & \frac{x_2}{\sqrt{x_2^2+\log(x_3)}} & \frac{1}{2x_3\sqrt{x_2^2+\log(x_3)}}, \end{bmatrix}$$

b) Let $a: \mathbb{R}^1 \mapsto \mathbb{R}^1$ and define $\mathbf{f}(\mathbf{x}) = (a(x_1), \ldots, a(x_n)), \mathbf{f}: \mathbb{R}^n \mapsto \mathbb{R}^n$. Calculate the matrix of partial derivatives $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x})$. What property has the resulting matrix?

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}) = \begin{bmatrix} \frac{\partial a}{\partial x_1} & 0 & \ldots & 0 \\ 0 & \frac{\partial a}{\partial x_2} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \frac{\partial a}{\partial x_n} \end{bmatrix}$$

This is a diagonal matrix.

c) Let $\sigma(x) = \frac{1}{1+e^{-x}}, \sigma: \mathbb{R}^1 \mapsto \mathbb{R}^1$. Calculate $\sigma'(x)$.

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}$$
$$\frac{\partial \sigma}{\partial x} = \frac{e^x \cdot (1+e^x) - e^x \cdot e^x}{(1+e^x)^2} = \frac{e^x}{(1+e^x)^2} = \sigma(x)(1-\sigma(x))$$

d) Let $\mathbf{f}$ be like in **b)** with $a = \sigma$. Calculate the matrix of partial derivatives $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x})$.

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}) = \begin{bmatrix} \sigma'(x_1) & 0 & \ldots & 0 \\ 0 & \sigma'(x_2) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \sigma'(x_n) \end{bmatrix}$$

$$= \begin{bmatrix} \sigma(x_1)(1-\sigma(x_1)) & 0 & \ldots & 0 \\ 0 & \sigma(x_2)(1-\sigma(x_2)) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \sigma(x_n)(1-\sigma(x_n)) \end{bmatrix}$$

e) Let $\mathbf{W} \in \mathbb{R}^{m \times n}$, $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} = \mathbf{W}\mathbf{x}$.

i) Calculate the matrix of partial derivatives $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$.

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \mathbf{W}$$

ii) Calculate $\frac{\partial y_k}{\partial w_{ij}}$ for $k \in \{1, \ldots, m\}, i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$.

$$\frac{\partial y_k}{\partial w_{ij}} = \begin{cases} x_j & \text{if } k = i \\ 0 & \text{if } k \neq i \end{cases}$$

iii) Let $\mathbf{A} \in \mathbb{R}^{1 \times m}$. Simplify $\hat{w}_{ij} = \mathbf{A}\frac{\partial \mathbf{y}}{\partial w_{ij}}$.

$$\hat{w}_{ij} = \mathbf{A}\frac{\partial \mathbf{y}}{\partial w_{ij}} = \sum_{k=1}^{m} a_{1k}\frac{\partial y_k}{\partial w_{ij}} = a_{1i}x_j$$

1.2 Linear Regression

The loss function $\mathcal{L}$ of a Linear Regression model with a scalar input and scalar output is given by:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(w_0 + w_1 x_i - y_i)^2$$

In this simple 1D case, we are fitting a straight line to the dataset. This line is determined by its offset along the y axis w_0 and its slope w_1. Compute the partial derivative of the loss wrt. the y intercept $\frac{\partial \mathcal{L}}{\partial w_0}$ and the partial derivative of the loss wrt. the slope $\frac{\partial \mathcal{L}}{\partial w_1}$.

$$\frac{\partial \mathcal{L}}{\partial w_0} = \frac{2}{N}\sum_{i}^{N}(w_0 + w_1 x_i - y_i)$$
$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{2}{N}\sum_{i}^{N} x_i(w_0 + w_1 x_i - y_i)$$

1.3 Computational Graphs and Backpropagation

Complete the following exercises for each of these expressions:

$$\mathcal{A} = \frac{\sqrt{a}}{e^{a+b}}\log(a+b) + \cos(\sqrt{a})$$
$$\mathcal{B} = 42^{a+b+\sqrt{c}} + \sin(a+b+\sqrt{c}) + (a+b+\sqrt{c})b\sqrt{c}$$
$$\mathcal{C} = \sum_{i=1}^{3}(w_0 + w_1 x_i - y_i)^2$$

a) Introduce intermediate variables $(t_1, t_2, t_3, \ldots)$ such that there is a single mathematical operation per assignment. Reuse intermediate variables whenever possible.

b) Draw the corresponding computation graph based on the variables you introduced in **a)**.

c) For each node in your computation graph, write down the products of partial derivatives (e.g. $\frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial x}$) that are computed by this node during backpropagation. For each such product indicate to which node the results are sent to (e.g., by writing down the products close to the corresponding graph edges as we did it in the lecture). Note: you do not need to symbolically calculate or simplify the derivative for specific operations.

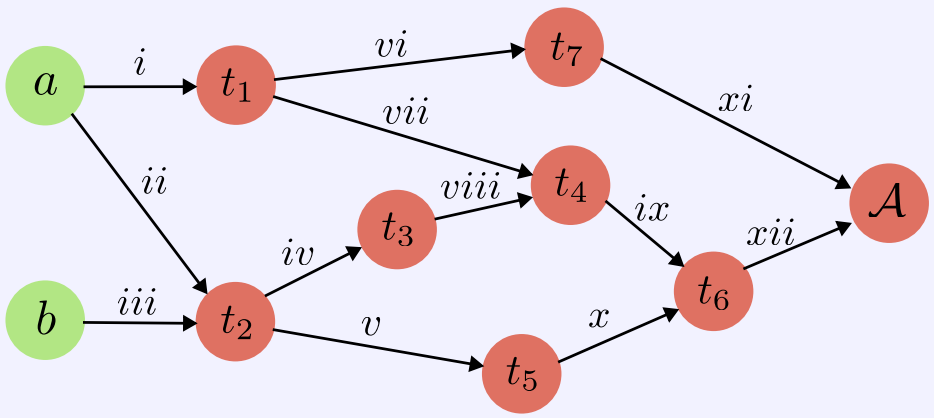
d) At each node, write down the sum of partial derivative products that calculates the derivative of the output node wrt. this node. Your expressions only need to contain the direct children for the node and again you do not need to symbolically calculate or simplify the derivatives.

$\mathcal{A} = \frac{\sqrt{a}}{e^{a+b}}\log(a+b) + \cos(\sqrt{a})$

a)

$t_1 = \sqrt{a}$
$t_2 = a + b$
$t_3 = e^{t_2}$
$t_4 = \frac{t_1}{t_3}$
$t_5 = \log t_2$
$t_6 = t_4 \cdot t_5$
$t_7 = \cos t_1$
$\mathcal{A} = t_6 + t_7$

b)



c)

$$xii := \frac{\partial \mathcal{A}}{\partial \mathcal{A}}\frac{\partial \mathcal{A}}{\partial t_6}$$
$$xi := \frac{\partial \mathcal{A}}{\partial \mathcal{A}}\frac{\partial \mathcal{A}}{\partial t_7}$$
$$x := \frac{\partial \mathcal{A}}{\partial t_6}\frac{\partial t_6}{\partial t_5}$$
$$ix := \frac{\partial \mathcal{A}}{\partial t_6}\frac{\partial t_6}{\partial t_4}$$
$$viii := \frac{\partial \mathcal{A}}{\partial t_4}\frac{\partial t_4}{\partial t_3}$$
$$vii := \frac{\partial \mathcal{A}}{\partial t_4}\frac{\partial t_4}{\partial t_1}$$
$$vi := \frac{\partial \mathcal{A}}{\partial t_7}\frac{\partial t_7}{\partial t_1}$$
$$v := \frac{\partial \mathcal{A}}{\partial t_5}\frac{\partial t_5}{\partial t_2}$$
$$iv := \frac{\partial \mathcal{A}}{\partial t_3}\frac{\partial t_3}{\partial t_2}$$
$$iii := \frac{\partial \mathcal{A}}{\partial t_2}\frac{\partial t_2}{\partial b}$$
$$ii := \frac{\partial \mathcal{A}}{\partial t_2}\frac{\partial t_2}{\partial a}$$
$$i := \frac{\partial \mathcal{A}}{\partial t_1}\frac{\partial t_1}{\partial a}$$

d)

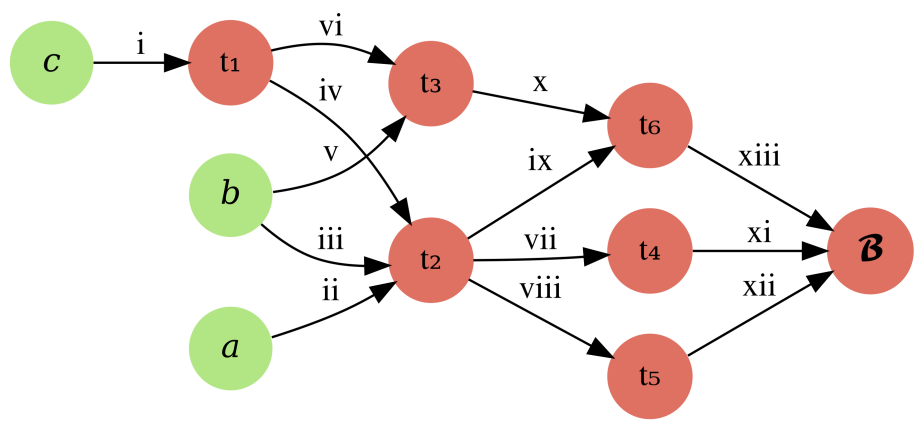
$$\mathcal{A} : \frac{\partial \mathcal{A}}{\partial \mathcal{A}}$$
$$t_7 : \frac{\partial \mathcal{A}}{\partial t_7}$$
$$t_6 : \frac{\partial \mathcal{A}}{\partial t_6}$$
$$t_5 : \frac{\partial \mathcal{A}}{\partial t_5}$$
$$t_4 : \frac{\partial \mathcal{A}}{\partial t_4}$$
$$t_3 : \frac{\partial \mathcal{A}}{\partial t_3}$$
$$t_2 : \frac{\partial \mathcal{A}}{\partial t_5}\frac{\partial t_5}{\partial t_2} + \frac{\partial \mathcal{A}}{\partial t_3}\frac{\partial t_3}{\partial t_2}$$
$$t_1 : \frac{\partial \mathcal{A}}{\partial t_7}\frac{\partial t_7}{\partial t_1} + \frac{\partial \mathcal{A}}{\partial t_4}\frac{\partial t_4}{\partial t_1}$$
$$b : \frac{\partial \mathcal{A}}{\partial b}$$
$$a : \frac{\partial \mathcal{A}}{\partial t_2}\frac{\partial t_2}{\partial a} + \frac{\partial \mathcal{A}}{\partial t_1}\frac{\partial t_1}{\partial a}$$

$\mathcal{B} = 42^{a+b+\sqrt{c}} + \sin(a+b+\sqrt{c}) + (a+b+\sqrt{c})b\sqrt{c}$

a)

$t_1 = \sqrt{c}$
$t_2 = a + b + t_1$
$t_3 = b \cdot t_1$
$t_4 = 42^{t_2}$
$t_5 = \sin t_2$
$t_6 = t_2 \cdot t_3$
$\mathcal{B} = t_4 + t_5 + t_6$

b)



c) Each edge e from a node v to a node u carries the product

$$e := \frac{\partial \mathcal{B}}{\partial u}\frac{\partial u}{\partial v},$$

which node u computes during backpropagation and sends to node v. With the edge labels of the graph:

$$i := \frac{\partial \mathcal{B}}{\partial t_1}\frac{\partial t_1}{\partial c} \qquad ii := \frac{\partial \mathcal{B}}{\partial t_2}\frac{\partial t_2}{\partial a} \qquad iii := \frac{\partial \mathcal{B}}{\partial t_2}\frac{\partial t_2}{\partial b}$$
$$iv := \frac{\partial \mathcal{B}}{\partial t_2}\frac{\partial t_2}{\partial t_1} \qquad v := \frac{\partial \mathcal{B}}{\partial t_3}\frac{\partial t_3}{\partial b} \qquad vi := \frac{\partial \mathcal{B}}{\partial t_3}\frac{\partial t_3}{\partial t_1}$$
$$vii := \frac{\partial \mathcal{B}}{\partial t_4}\frac{\partial t_4}{\partial t_2} \qquad viii := \frac{\partial \mathcal{B}}{\partial t_5}\frac{\partial t_5}{\partial t_2} \qquad ix := \frac{\partial \mathcal{B}}{\partial t_6}\frac{\partial t_6}{\partial t_2}$$
$$x := \frac{\partial \mathcal{B}}{\partial t_6}\frac{\partial t_6}{\partial t_3} \qquad xi := \frac{\partial \mathcal{B}}{\partial \mathcal{B}}\frac{\partial \mathcal{B}}{\partial t_4} \qquad xii := \frac{\partial \mathcal{B}}{\partial \mathcal{B}}\frac{\partial \mathcal{B}}{\partial t_5}$$
$$xiii := \frac{\partial \mathcal{B}}{\partial \mathcal{B}}\frac{\partial \mathcal{B}}{\partial t_6}$$

d) We omit the nodes with a fan-out of 1: their derivative is the product on their single outgoing edge (see **c)**).

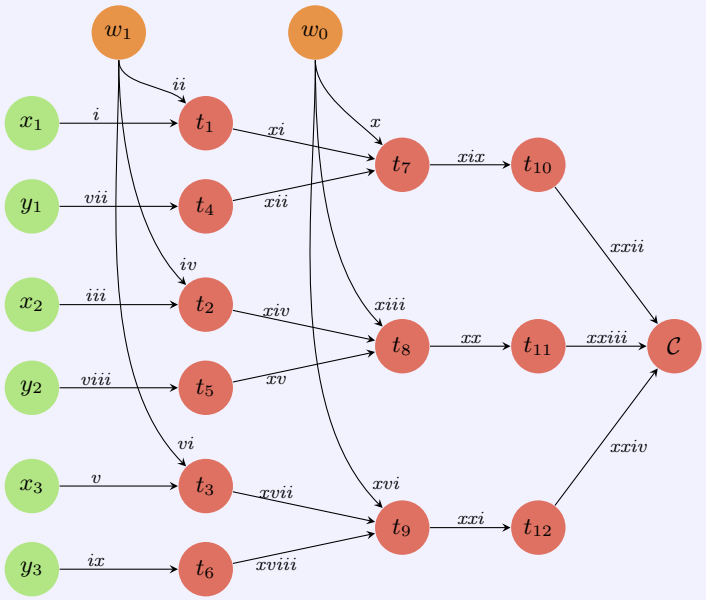
$$t_2 : \frac{\partial \mathcal{B}}{\partial t_4}\frac{\partial t_4}{\partial t_2} + \frac{\partial \mathcal{B}}{\partial t_5}\frac{\partial t_5}{\partial t_2} + \frac{\partial \mathcal{B}}{\partial t_6}\frac{\partial t_6}{\partial t_2}$$
$$t_1 : \frac{\partial \mathcal{B}}{\partial t_2}\frac{\partial t_2}{\partial t_1} + \frac{\partial \mathcal{B}}{\partial t_3}\frac{\partial t_3}{\partial t_1}$$
$$b : \frac{\partial \mathcal{B}}{\partial t_2}\frac{\partial t_2}{\partial b} + \frac{\partial \mathcal{B}}{\partial t_3}\frac{\partial t_3}{\partial b}$$

$$\mathcal{C} = \sum_{i=1}^{3} (w_0 + w_1 x_i - y_i)^2$$

a)

$$\begin{aligned}
t_1 &= w_1 \cdot x_1 \\
t_2 &= w_1 \cdot x_2 \\
t_3 &= w_1 \cdot x_3 \\
t_4 &= -1 \cdot y_1 \\
t_5 &= -1 \cdot y_2 \\
t_6 &= -1 \cdot y_3 \\
t_7 &= w_0 + t_1 + t_4 \\
t_8 &= w_0 + t_2 + t_5 \\
t_9 &= w_0 + t_3 + t_6 \\
t_{10} &= t_7^2 \\
t_{11} &= t_8^2 \\
t_{12} &= t_9^2 \\
\mathcal{C} &= t_{10} + t_{11} + t_{12}
\end{aligned}$$

b)



c) As for $\mathcal{B}$, the edge e from v to u carries $e := \frac{\partial \mathcal{C}}{\partial u}\frac{\partial u}{\partial v}$:

$$\begin{aligned}
i &:= \frac{\partial \mathcal{C}}{\partial t_1}\frac{\partial t_1}{\partial x_1} & ii &:= \frac{\partial \mathcal{C}}{\partial t_1}\frac{\partial t_1}{\partial w_1} & iii &:= \frac{\partial \mathcal{C}}{\partial t_2}\frac{\partial t_2}{\partial x_2} \\
iv &:= \frac{\partial \mathcal{C}}{\partial t_2}\frac{\partial t_2}{\partial w_1} & v &:= \frac{\partial \mathcal{C}}{\partial t_3}\frac{\partial t_3}{\partial x_3} & vi &:= \frac{\partial \mathcal{C}}{\partial t_3}\frac{\partial t_3}{\partial w_1} \\
vii &:= \frac{\partial \mathcal{C}}{\partial t_4}\frac{\partial t_4}{\partial y_1} & viii &:= \frac{\partial \mathcal{C}}{\partial t_5}\frac{\partial t_5}{\partial y_2} & ix &:= \frac{\partial \mathcal{C}}{\partial t_6}\frac{\partial t_6}{\partial y_3} \\
x &:= \frac{\partial \mathcal{C}}{\partial t_7}\frac{\partial t_7}{\partial w_0} & xi &:= \frac{\partial \mathcal{C}}{\partial t_7}\frac{\partial t_7}{\partial t_1} & xii &:= \frac{\partial \mathcal{C}}{\partial t_7}\frac{\partial t_7}{\partial t_4} \\
xiii &:= \frac{\partial \mathcal{C}}{\partial t_8}\frac{\partial t_8}{\partial w_0} & xiv &:= \frac{\partial \mathcal{C}}{\partial t_8}\frac{\partial t_8}{\partial t_2} & xv &:= \frac{\partial \mathcal{C}}{\partial t_8}\frac{\partial t_8}{\partial t_5} \\
xvi &:= \frac{\partial \mathcal{C}}{\partial t_9}\frac{\partial t_9}{\partial w_0} & xvii &:= \frac{\partial \mathcal{C}}{\partial t_9}\frac{\partial t_9}{\partial t_3} & xviii &:= \frac{\partial \mathcal{C}}{\partial t_9}\frac{\partial t_9}{\partial t_6} \\
xix &:= \frac{\partial \mathcal{C}}{\partial t_{10}}\frac{\partial t_{10}}{\partial t_7} & xx &:= \frac{\partial \mathcal{C}}{\partial t_{11}}\frac{\partial t_{11}}{\partial t_8} & xxi &:= \frac{\partial \mathcal{C}}{\partial t_{12}}\frac{\partial t_{12}}{\partial t_9} \\
xxii &:= \frac{\partial \mathcal{C}}{\partial \mathcal{C}}\frac{\partial \mathcal{C}}{\partial t_{10}} & xxiii &:= \frac{\partial \mathcal{C}}{\partial \mathcal{C}}\frac{\partial \mathcal{C}}{\partial t_{11}} & xxiv &:= \frac{\partial \mathcal{C}}{\partial \mathcal{C}}\frac{\partial \mathcal{C}}{\partial t_{12}}
\end{aligned}$$

d) Again we omit the nodes with a fan-out of 1.

$$\begin{aligned}
w_1 &: \frac{\partial \mathcal{C}}{\partial t_1}\frac{\partial t_1}{\partial w_1} + \frac{\partial \mathcal{C}}{\partial t_2}\frac{\partial t_2}{\partial w_1} + \frac{\partial \mathcal{C}}{\partial t_3}\frac{\partial t_3}{\partial w_1} \\
w_0 &: \frac{\partial \mathcal{C}}{\partial t_7}\frac{\partial t_7}{\partial w_0} + \frac{\partial \mathcal{C}}{\partial t_8}\frac{\partial t_8}{\partial w_0} + \frac{\partial \mathcal{C}}{\partial t_9}\frac{\partial t_9}{\partial w_0}
\end{aligned}$$

1.4 Logistic Regression

Using the chain rule, show that

$$\nabla_{\mathbf{w}} \frac{1}{B} \sum_{i=1}^{B} \mathcal{L}(y_i, \hat{y}_i) = \frac{1}{B} \sum_{i=1}^{B} (\hat{y}_i - y_i)\mathbf{x}_i$$

where

$$\mathcal{L}(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$
$$\hat{y}_i = \sigma(\mathbf{w}^\top \mathbf{x}_i)$$

Note: Here we use the numerator (Jacobian) layout convention for matrix calculus: the derivative $\frac{\partial \mathcal{L}}{\partial \mathbf{w}}$ of a scalar with respect to a column vector is a row vector, and the gradient $\nabla_{\mathbf{w}} \mathcal{L}$ is its transpose. Layout conventions vary in different resources as well as implementations. More on matrix calculus layout conventions can be found here.

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{w}} &= \frac{\partial \mathcal{L}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial h_i} \frac{\partial h_i}{\partial \mathbf{w}} \\
&= -\left(\frac{y_i}{\hat{y}_i} - \frac{1 - y_i}{1 - \hat{y}_i} \right) \hat{y}_i (1 - \hat{y}_i) \mathbf{x}_i^\top \\
&= (\hat{y}_i - y_i) \mathbf{x}_i^\top \\
\text{with } h_i &= \mathbf{w}^\top \mathbf{x}_i
\end{aligned}$$

By the additivity of the gradient it follows

$$\nabla_{\mathbf{w}} \frac{1}{B} \sum_{i=1}^{B} \mathcal{L}(y_i, \hat{y}_i) = \frac{1}{B} \sum_{i=1}^{B} \nabla_{\mathbf{w}} \mathcal{L}(y_i, \hat{y}_i) = \frac{1}{B} \sum_{i=1}^{B} \left(\frac{\partial \mathcal{L}}{\partial \mathbf{w}} \right)^\top = \frac{1}{B} \sum_{i=1}^{B} (\hat{y}_i - y_i) \mathbf{x}_i$$

2 Coding

2.1 Derivatives

Implement the function `vector_derivative_product` in the jupyter notebook `vector_derivative_product.ipynb` that takes $\mathbf{A}$ and $\mathbf{x}$ as numpy arrays as input and returns the matrix $\hat{\mathbf{W}}$ (with elements $\hat{w}_{ij}$) as numpy array. Find a solution based on broadcasting instead of using `for` loops.

See `vector_derivative_product_solution.ipynb`.

2.2 Linear Regression

a) In the jupyter notebook `linear_regression.ipynb` an almost complete code for loading a dataset and training a Linear Regression model is given. Merely the function `compute_derivatives(x, y, slope, offset_y)` needs to be implemented. Here, `x` and `y` are equally sized numpy arrays containing the inputs $(x_1, x_2, \ldots)$ and outputs $(y_1, y_2, \ldots)$. `slope` corresponds to w_1 and `offset_y` corresponds to w_0. The function should return a tuple of float values containing the derivatives wrt. the parameters: $(\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_0})$. Implement the function and verify it by training a Linear Regression model and checking whether the plotted line fits the dataset.

See `linear_regression_solution.ipynb`

b) In the same jupyter notebook, implement the function `compute_weights_cf(X, y)` that calculates a closed-form solution to the Linear Regression problem. The function returns the values for `slope` and `offset_y` as a tuple of floats. Verify the implementation by using the `plot` and `compute_error` functions.

See `linear_regression_solution.ipynb`

c) `linear_regression.ipynb` also contains almost complete code for training a Linear Regression model with help of the Educational Framework (EDF). Merely the computational node `L2Loss`

needs to be implemented and the computational graph needs to be defined. The node `L2Loss` computes the *elementwise* squared error between its two input nodes `y` and `y_hat`. Since EDF operates on whole batches, which means that multiple inputs are processed simultaneously, `y` and `y_hat` are $B \times 1$ matrices, where B is the batch size. For this exercise, we consider the entire dataset as one batch, thus $B = N$. Since this node computes the elementwise squared error, the node's output should also be a $B \times 1$ matrix. In contrast to exercise **b)** it is not necessary to compute the mean along the dataset/batch dimension, which is implicitly taken care of by the EDF. Implement the `L2Loss` node including the forward and the backward pass and define the computation graph for the Linear Regression model. Again you can verify your solution by training the model and checking whether the plotted line fits the dataset.

> See `linear_regression_solution.ipynb`

2.3 Logistic Regression

a) In the jupyter notebook `logistic_regression.ipynb` an almost complete code for loading the MNIST handwritten digit dataset and training a Logistic Regression model is given. Merely the function `compute_derivative(x, y, y_hat)` needs to be implemented. This function should return the gradient wrt. the parameters $\nabla_{\mathbf{w}} \frac{1}{B} \sum_{i=1}^{B} \mathcal{L}(y_i, \hat{y}_i)$ as a numpy array. The argument `x` is a $B \times (D+1)$ matrix, where row `x[i]` contains the input vector $\mathbf{x}_{i+1}$ with a constant 1 appended for the bias term, B is the batch size and D is the number of input features. `y` and `y_hat` are vectors of size B. `y[i]` and `y_hat[i]` contain the scalars y_{i+1} and $\hat{y}_{i+1}$, respectively. Verify your solution by training a Logistic Regression model and checking whether the test error drops during the course of training.

Remark: In the preprocessing part of `logistic_regression.ipynb`, four different versions of the MNIST dataset are created. Beside the original dataset, which has a resolution of 28×28 pixels, downscaled versions with a resolution of 1×1 (`x_train_mean`, `x_test_mean`), 4×4 (`x_train_4x4`, `x_test_4x4`) and 8×8 (`x_train_8x8`, `x_test_8x8`) pixels are created. After downscaling, the pixel matrices are flattened into a vector. The code for this exercise is set up to use the 1×1 pixel version of the dataset. Your implementation of the function `compute_derivative` is not affected by this as it should handle a variable number of input features D.

> See `logistic_regression_solution.ipynb`.

b) `logistic_regression.ipynb` also contains almost complete code for training a Logistic Regression model with help of the EDF. EDF already implements the required `CrossEntropyLoss` node. For binary classification this node expects as input a matrix `p` of size $B \times 2$ where each entry `p[i]` corresponds to a probability vector with entries `p[i][0]` $= P(Y = 0|X = \mathbf{x}_i)$ and `p[i][1]` $= P(Y = 1|X = \mathbf{x}_i)$. In contrast, we defined our logistic regression model to only output a single probability $P(Y = 1|X = \mathbf{x}_i)$, which is sufficient since the other probability can be deduced via the relationship $P(Y = 0|X = \mathbf{x}_i) = 1 - P(Y = 1|X = \mathbf{x}_i)$. However, to make use of EDF's `CrossEntropyLoss` we need to provide the other probability as well. To this end, we define the computation node `SingleProbToProbVector` that takes as input a $B \times 1$ matrix containing only the probabilities $P(Y = 1|X = \mathbf{x}_i)$ and returns a $B \times 2$ matrix containing both $P(Y = 1|X = \mathbf{x}_i)$ and $P(Y = 0|X = \mathbf{x}_i)$ for each $i \in \{1, \ldots, b\}$.
More concretely, let `z` be the $B \times 1$ matrix that the node receives as input and let `o` be the $B \times 2$ matrix the node outputs, then `o[i][0]` $=$ `z[i][0]` and `o[i][1]` $=$ `1 - z[i][0]` for each `i` $\in \{0, \ldots, B-1\}$.
The code for this exercise is also set up to use the 1×1 version of the dataset. Implement the forward and backward pass of the `SingleProbToProbVector` node and define the computation graph. Again you can verify your implementation by training a Logistic Regression model for the 1×1 dataset. A similar test error like in the previous exercise **b)** should be achieved.

> **Derivation of backward():** Following the recipe given in the exercise we first write down

the operation of our computational node in scalar form:

$$\begin{aligned}\texttt{y}[\texttt{b},0] &= \texttt{z}[\texttt{b},0]\\ \texttt{y}[\texttt{b},1] &= 1-\texttt{z}[\texttt{b},0]\end{aligned}$$

where $\texttt{b} \in \{0,\dots B-1\}$. We use the chain rule to derive the gradient calculation. We have to consider all ys that might depend on our input $\texttt{z}[\texttt{i},0]$ where $\texttt{i} \in \{0,\dots B-1\}$:

$$\begin{aligned}\texttt{z.grad}[\texttt{i},0] &= \frac{\partial \mathcal{L}}{\partial \texttt{z}[\texttt{i},0]}\\ &= \sum_{\texttt{b}=0}^{B-1} \frac{\partial \mathcal{L}}{\partial \texttt{y}[\texttt{b},0]}\frac{\partial \texttt{y}[\texttt{b},0]}{\partial \texttt{z}[\texttt{i},0]} + \frac{\partial \mathcal{L}}{\partial \texttt{y}[\texttt{b},1]}\frac{\partial \texttt{y}[\texttt{b},1]}{\partial \texttt{z}[\texttt{i},0]}\\ &= \frac{\partial \mathcal{L}}{\partial \texttt{y}[\texttt{i},0]}1 + \frac{\partial \mathcal{L}}{\partial \texttt{y}[\texttt{i},1]}(-1)\end{aligned}$$

The last equality follows from

$$\frac{\partial \texttt{y}[\texttt{b},0]}{\partial \texttt{z}[\texttt{i},0]} = \begin{cases}1, \text{if } \texttt{b} = \texttt{i}\\ 0, \text{if } \texttt{b} \neq \texttt{i}\end{cases}$$

and

$$\frac{\partial \texttt{y}[\texttt{b},1]}{\partial \texttt{z}[\texttt{i},0]} = \begin{cases}-1, & \text{if } \texttt{b} = \texttt{i}\\ 0, & \text{if } \texttt{b} \neq \texttt{i}\end{cases}$$

In summary we need to make sure that in the `backward` function the following operation is executed for each $\texttt{i} \in \{0,\dots B-1\}$:

$$\texttt{z.grad}[\texttt{i},0] \mathrel{+}= \frac{\partial \mathcal{L}}{\partial \texttt{y}[\texttt{i},0]} - \frac{\partial \mathcal{L}}{\partial \texttt{y}[\texttt{i},1]}$$

Remembering that the gradients w.r.t. to the outputs are already precomputed and stored in `self.grad` we get:

$$\texttt{z.grad}[\texttt{i},0] \mathrel{+}= \texttt{self.grad}[\texttt{i},0] - \texttt{self.grad}[\texttt{i},1]$$

The same operation is executed along the first dimension (the batch dimension) since `i` is always in the same place. We can therefore vectorize this as follows:

$$\texttt{z.addgrad}((\texttt{self.grad}[:,0] - \texttt{self.grad}[:,1]).\texttt{reshape}(-1,1))$$

The reshaping is necessary because selecting for example the first column of `self.grad` (i.e. `self.grad[:,0]`) returns an array of shape $(B,)$ but we need an array of shape $(B,1)$ here, because `z` has shape $(B,1)$ here.

Remark: `CrossEntropyLoss` uses the entry `p[i][y_i]`, so it treats column 0 as $P(Y=0|X=\mathbf{x}_i)$. With `o[i][0]` = `z[i][0]` as defined above, training therefore turns `z` into the probability $P(Y=0|X=\mathbf{x}_i)$, not $P(Y=1|X=\mathbf{x}_i)$. The error rates are not affected, because the predicted class is the arg max over the two columns. To keep `z` = $P(Y=1|X=\mathbf{x}_i)$, swap the columns, `o[i][0]` = `1 - z[i][0]` and `o[i][1]` = `z[i][0]`; the backward pass then becomes `z.addgrad((self.grad[:, 1] - self.grad[:, 0]).reshape(-1, 1))`.

The rest of the solution is given in `logistic_regression_solution.ipynb`.

c) Use the EDF to train Logistic Regression models with the 4×4, 8×8 and 28×28 pixel versions of the dataset. Use the cells at the end of `logistic_regression.ipynb` for these additional experiments.

See `logistic_regression_solution.ipynb`.