Deep Learning

Tutorial 3 – Exercise Sheet 2 and Solutions of Sheet 1

Naama Pearl
Theory of Machine Learning Group
University of Tübingen

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Tübingen AI Center
tuebingen.ai

ellis
European Laboratory for Learning and Intelligent Systems

Outline

- Quiz
- Exercise sheet 2
- Solutions of exercise sheet 1

Quiz, questions 1 to 5

1. **(B)** $\nabla_{\mathbf{A}} \mathcal{L}$ is $M \times D$. The gradient has the shape of the parameter
2. **(C)** An affine function, $\mathbf{y} = \mathbf{A}_2(\mathbf{A}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 = (\mathbf{A}_2\mathbf{A}_1)\,\mathbf{x} + (\mathbf{A}_2\mathbf{b}_1 + \mathbf{b}_2)$.
   Without a nonlinearity, the extra layer adds nothing
3. **(A)** The validation set. The test set stays untouched for the final estimate, and the training set favors the network that overfits most
4. **(C)** Some network with finitely many hidden units is ϵ-close everywhere.
   The theorem says nothing about finding the weights (A),
   unseen data (B), or how many units are needed (D)
5. **(B)** $\sum_d (\hat{y}_d - y_d)^2$. $-\log$ of independent Gaussians with a fixed variance is the sum of squared errors, up to constants.
   (C) would come from a Laplace likelihood, (D) is the cross-entropy



Exercise Sheet 2

Exercise sheet 2

Sheet 2

Exercise	You need	Covered in
Pen & paper		
1.1 Derivatives (ReLU, SiLU, tanh, softmax, cross-entropy)	derivatives, chain rule	tutorial 1, lecture 3
1.2 Softmax invariance	softmax	lecture 3
1.3 A gradient step of a 1-layer MLP	MLP, gradient descent	lecture 3
Coding		
2.1 Binary classification of 2D points (MLP)	EDF, your sheet 1 code (2.3)	tutorial 2
2.2 Image classification of MNIST (MLP)		
a-c) Logistic regression and MLP, 10 classes	softmax, cross-entropy	lecture 3
d) Tanh, ReLU, SiLU nodes	activation functions, 1.1	lecture 3
e) Numerical gradient	numerical differentiation	lecture 4 (next week)
f-i) Learning rate, depth, width	training and evaluating	lectures 3 and 4

- Same conda environment as sheet 1, **solutions** on **11.11.**

Solutions of Exercise Sheet 1

1. Pen and Paper

The Jacobian of a function from $\mathbb{R}^3$ to $\mathbb{R}^2$

Sheet 1 – Solution 1.1 a)

Task Calculate the matrix of partial derivatives $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ for
$\mathbf{f}(\mathbf{x}) = \big(f_1(\mathbf{x}),\ f_2(\mathbf{x})\big) = \big(e^{\sqrt{x_3}} \sin x_1,\ \sqrt{\log x_3 + x_2^2}\big), \mathbf{f}\colon \mathbb{R}^3 \to \mathbb{R}^2$

- The shape of $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ is 2×3, one row per output, one column per input

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{pmatrix} e^{\sqrt{x_3}} \cos x_1 & 0 & \frac{e^{\sqrt{x_3}} \sin x_1}{2\sqrt{x_3}} \\ 0 & \frac{x_2}{\sqrt{\log x_3 + x_2^2}} & \frac{1}{2x_3\sqrt{\log x_3 + x_2^2}} \end{pmatrix}$$

- Row 1, $f_1 = e^u \sin x_1$ with $u = \sqrt{x_3}$

$$\frac{\partial f_1}{\partial x_1} = e^{\sqrt{x_3}} \cos x_1$$
$$\frac{\partial f_1}{\partial x_2} = 0 \quad (\text{no } x_2)$$
$$\frac{\partial f_1}{\partial x_3} = \sin x_1 \cdot e^{\sqrt{x_3}} \cdot \frac{1}{2\sqrt{x_3}}$$

- Row 2, $f_2 = \sqrt{u}$ with $u = \log x_3 + x_2^2$

$$\frac{\partial f_2}{\partial x_1} = 0 \quad (\text{no } x_1)$$
$$\frac{\partial f_2}{\partial x_2} = \frac{1}{2\sqrt{u}} \cdot 2x_2$$
$$\frac{\partial f_2}{\partial x_3} = \frac{1}{2\sqrt{u}} \cdot \frac{1}{x_3}$$

The Jacobian of an elementwise function

Sheet 1 – Solution 1.1 b)

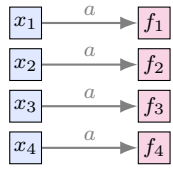
Task Calculate $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ for $\mathbf{f}(\mathbf{x}) = \big(a(x_1), \dots, a(x_n)\big)$, $a\colon \mathbb{R} \to \mathbb{R}$. What property does the matrix have?

$\mathbf{f}\colon \mathbb{R}^n \to \mathbb{R}^n$

- The shape of $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ is $n \times n$
- $f_i = a(x_i)$ depends only on x_i, so

$$\frac{\partial f_i}{\partial x_j} = \begin{cases} a'(x_i) & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

- All entries together, $\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{pmatrix} a'(x_1) & & 0 \\ & \ddots & \\ 0 & & a'(x_n) \end{pmatrix}$
- The matrix is **diagonal**



nonzero only on the diagonal

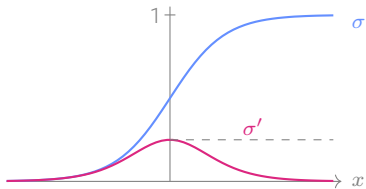
Sigmoid: the derivative

Sheet 1 – Solution 1.1 c)

Task Calculate $\sigma'(x)$ for $\sigma(x) = \frac{1}{1+e^{-x}}$

c) $\sigma(x) = \frac{1}{1+e^{-x}} = u^{-1}$ with $u = 1 + e^{-x}$

$$\begin{aligned}
\sigma'(x) &= -u^{-2} \cdot u' \qquad \text{(chain rule)} \\
&= -\frac{1}{(1+e^{-x})^2} \cdot (-e^{-x}) = \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} \\
&= \sigma(x) \cdot \frac{(1+e^{-x})-1}{1+e^{-x}} = \sigma(x) \cdot \left(1 - \frac{1}{1+e^{-x}}\right) \\
&= \sigma(x)\left(1 - \sigma(x)\right)
\end{aligned}$$



σ' is at most $\sigma(0)\,(1-\sigma(0)) = \frac{1}{4}$

The Jacobian of a sigmoid

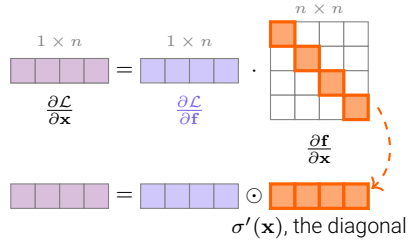
Task Calculate $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ for $\mathbf{f}(\mathbf{x}) = \big(a(x_1), \dots, a(x_n)\big)$ from b), with $a = \sigma$

- $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ is diagonal with entries $\sigma'(x_i) = \sigma(x_i)\big(1 - \sigma(x_i)\big)$
- This is how an **activation layer** works, σ is applied to every input separately

- In the backward pass, the node gets $\frac{\partial \mathcal{L}}{\partial \mathbf{f}}$
- Chain rule in matrix form $\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \frac{\partial \mathcal{L}}{\partial \mathbf{f}} \frac{\partial \mathbf{f}}{\partial \mathbf{x}}$
- Entry j is the row times column j, only $i = j$ is not zero

$$\frac{\partial \mathcal{L}}{\partial x_j} = \sum_i \frac{\partial \mathcal{L}}{\partial f_i} \frac{\partial f_i}{\partial x_j} = \frac{\partial \mathcal{L}}{\partial f_j} \sigma'(x_j)$$

- An **elementwise product**, the code never builds the $n \times n$ matrix



the row $\odot$ the diagonal

`Sigmoid.backward` in `edf.py`, `self.grad` is $\frac{\partial \mathcal{L}}{\partial \mathbf{f}}$

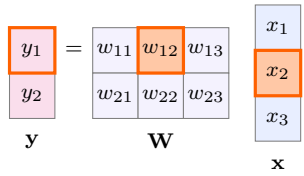
```
self.x.addgrad(self.grad
    * self.value * (1 - self.value))
```

Linear layer: the derivative for $\mathbf{x}$

Sheet 1 — Solution 1.1 e) i

Task Calculate $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ for $\mathbf{y} = \mathbf{W}\mathbf{x}$ with $\mathbf{W} \in \mathbb{R}^{m \times n}$, $\mathbf{x} \in \mathbb{R}^{n}$, $\mathbf{y} \in \mathbb{R}^{m}$

- The shape of $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ is $m \times n$, one row per output y_k, one column per input x_j



- Every output y_k is a sum

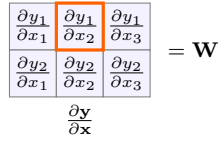
  $y_k = \sum_j w_{kj}\, x_j$ (row k of $\mathbf{W}$ times $\mathbf{x}$)

  For example

$$y_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3$$
$$y_2 = w_{21}x_1 + w_{22}x_2 + w_{23}x_3$$

- Only one term of y_k contains x_j, so $\frac{\partial y_k}{\partial x_j} = w_{kj}$

  for example $\frac{\partial y_1}{\partial x_2} = w_{12}$



- Entry (k, j) of $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ is w_{kj}, so $\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \mathbf{W}$
- Backward pass $\nabla_{\mathbf{x}} \mathcal{L} = \mathbf{W}^\top \nabla_{\mathbf{y}} \mathcal{L}$ $(n) = (n \times m)\,(m)$, the transpose of $\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}}\,\mathbf{W}$

  In EDF `grad @ A.T`, since `x @ A` uses $\mathbf{A} = \mathbf{W}^\top$

Linear layer: the derivative for w_{ij}

Sheet 1 – Solution 1.1 e) ii

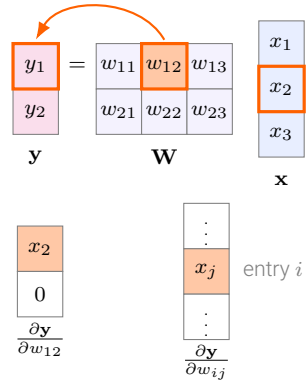
Task Calculate $\frac{\partial y_k}{\partial w_{ij}}$ for all k, i and j

$$y_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3, \qquad y_2 = w_{21}x_1 + w_{22}x_2 + w_{23}x_3$$

- w_{12} appears only in y_1, next to x_2, so
  $\frac{\partial y_1}{\partial w_{12}} = x_2, \quad \frac{\partial y_2}{\partial w_{12}} = 0$
- In general, w_{ij} appears only in output y_i, next to x_j

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

- As a vector, $\frac{\partial \mathbf{y}}{\partial w_{ij}}$ has x_j in entry i and zeros elsewhere



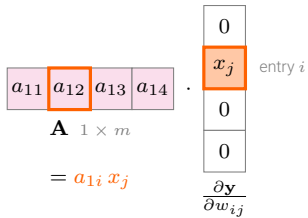
Linear layer: the loss derivative for w_{ij}

Sheet 1 — Solution 1.1 e) iii

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

$\mathbf{A} \frac{\partial \mathbf{y}}{\partial w_{ij}}$ is a row times a column, a number

$$\begin{aligned} \hat{w}_{ij} &= \textstyle\sum_{k=1}^{m} a_{1k} \frac{\partial y_k}{\partial w_{ij}} \\ &= a_{11} \cdot 0 + \ldots + a_{1i}\, x_j + \ldots + a_{1m} \cdot 0 \\ &= a_{1i}\, x_j \qquad \text{(only } k = i \text{ is not zero)} \end{aligned}$$



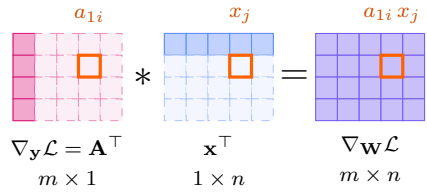
- If $\mathbf{A} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}}$ comes from the layers after $\mathbf{y}$, then $\hat{w}_{ij} = \frac{\partial \mathcal{L}}{\partial w_{ij}}$

Linear layer: the loss derivative for $\mathbf{W}$

Sheet 1 – Solution 1.1 e) iii

- The gradient puts all partial derivatives in the shape of the variable (tutorial 1)
- So $\nabla_{\mathbf{W}}\mathcal{L}$ is $m \times n$ like $\mathbf{W}$, with entry (i, j) equal to $\frac{\partial \mathcal{L}}{\partial w_{ij}} = a_{1i}\, x_j$ (slide 9, $\mathbf{A} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}}$)

$$\nabla_{\mathbf{W}}\mathcal{L} = \begin{pmatrix} a_{11}x_1 & \cdots & a_{11}x_n \\ \vdots & \ddots & \vdots \\ a_{1m}x_1 & \cdots & a_{1m}x_n \end{pmatrix}$$



copy the column right and the row down (dashed), then multiply

- This is the **outer product** $\nabla_{\mathbf{W}}\mathcal{L} = \nabla_{\mathbf{y}}\mathcal{L}\, \mathbf{x}^\top$ ($m \times 1$ times $1 \times n$)
- **Why one weight at a time?** The chain rule for all of $\mathbf{W}$ needs $\frac{\partial \mathbf{y}}{\partial \mathbf{W}}$, a vector by a matrix, which is not a matrix (three indices). For one weight, $\frac{\partial \mathbf{y}}{\partial w_{ij}}$ is a vector

As in lecture 3 and `Affine.backward` (tutorial 2). Coding task 2.1 computes it in NumPy

Linear regression: the derivatives of the loss

Task Compute $\frac{\partial \mathcal{L}}{\partial w_0}$ and $\frac{\partial \mathcal{L}}{\partial w_1}$, and implement them in `compute_derivatives` (2.2 a)

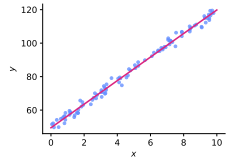
$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(w_0 + w_1 x_i - y_i)^2 = \frac{1}{N}\sum_{i=1}^{N} r_i{}^2$ with the residual $r_i = w_0 + w_1 x_i - y_i$

$$\frac{\partial \mathcal{L}}{\partial w_0} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial\, r_i{}^2}{\partial w_0} = \frac{1}{N}\sum_{i=1}^{N} 2r_i \cdot \frac{\partial r_i}{\partial w_0} = \frac{1}{N}\sum_{i=1}^{N} 2r_i \cdot 1 = \frac{2}{N}\sum_{i=1}^{N} r_i$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial\, r_i{}^2}{\partial w_1} = \frac{1}{N}\sum_{i=1}^{N} 2r_i \cdot \frac{\partial r_i}{\partial w_1} = \frac{1}{N}\sum_{i=1}^{N} 2r_i \cdot x_i = \frac{2}{N}\sum_{i=1}^{N} r_i\, x_i$$

chain rule with outer derivative $2r_i$ and inner derivative $\partial r_i/\partial w$, the sum and $\frac{1}{N}$ stay as they are

```
def compute_derivatives(x, y, slope, offset_y):
    r = offset_y + slope * x - y                # (N,)
    return 2 * np.mean(r * x), 2 * np.mean(r)
```



The code, solution 2.2 a)

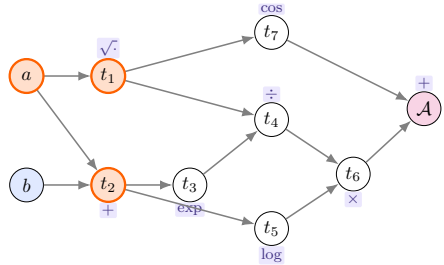
Computation graphs: the graph of $\mathcal{A}$

Sheet 1 — Solution 1.3

Task a) Introduce intermediate variables, one operation each. b) Draw the computation graph

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

$$\begin{aligned}
t_1 &= \sqrt{a} \\
t_2 &= a + b \\
t_3 &= e^{t_2} && = e^{a+b} \\
t_4 &= t_1 / t_3 && = \sqrt{a} \,/\, e^{a+b} \\
t_5 &= \log t_2 && = \log(a+b) \\
t_6 &= t_4 \cdot t_5 && = \sqrt{a} \,/\, e^{a+b} \cdot \log(a+b) \\
t_7 &= \cos t_1 && = \cos(\sqrt{a}) \\
\mathcal{A} &= t_6 + t_7 && = \sqrt{a} \,/\, e^{a+b} \cdot \log(a+b) \\
& && \quad + \cos(\sqrt{a})
\end{aligned}$$

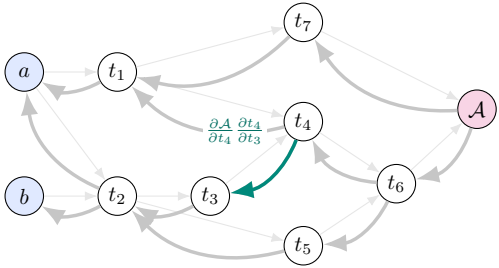


- One operation per variable, t_1 and t_2 are computed once and reused
- a, t_1 and t_2 feed two nodes each

Computation graphs: the backward pass for $\mathcal{A}$

Sheet 1 — Solution 1.3

Task c) Write the product each edge sends back. d) At each node, write the sum that gives the derivative of the output



curved arrows: the gradient flows back from $\mathcal{A}$

- **c)** every edge $u \to v$ sends $\frac{\partial \mathcal{A}}{\partial v}\frac{\partial v}{\partial u}$ back to u
- **d)** every node sums what it receives. A node with one child gets one term, for example $\frac{\partial \mathcal{A}}{\partial t_3} = \frac{\partial \mathcal{A}}{\partial t_4}\frac{\partial t_4}{\partial t_3}$
- A node with two children **sums** two terms

$$\frac{\partial \mathcal{A}}{\partial t_2} = \frac{\partial \mathcal{A}}{\partial t_5}\frac{\partial t_5}{\partial t_2} + \frac{\partial \mathcal{A}}{\partial t_3}\frac{\partial t_3}{\partial t_2}$$

$$\frac{\partial \mathcal{A}}{\partial a} = \frac{\partial \mathcal{A}}{\partial t_2}\frac{\partial t_2}{\partial a} + \frac{\partial \mathcal{A}}{\partial t_1}\frac{\partial t_1}{\partial a}$$

$$\frac{\partial \mathcal{A}}{\partial t_1} = \frac{\partial \mathcal{A}}{\partial t_7}\frac{\partial t_7}{\partial t_1} + \frac{\partial \mathcal{A}}{\partial t_4}\frac{\partial t_4}{\partial t_1}$$

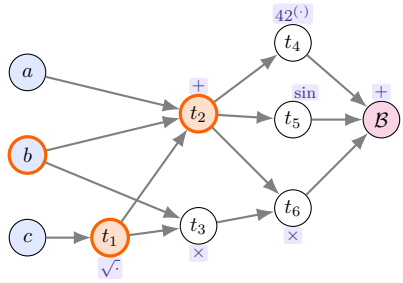
Computation graphs: the graph of $\mathcal{B}$

Sheet 1 — Solution 1.3

Task a) Introduce intermediate variables, one operation each. b) Draw the computation graph

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

$$\begin{aligned}
t_1 &= \sqrt{c} \\
t_2 &= a + b + t_1 && = a + b + \sqrt{c} \\
t_3 &= b \cdot t_1 && = b\sqrt{c} \\
t_4 &= 42^{t_2} && = 42^{a+b+\sqrt{c}} \\
t_5 &= \sin t_2 && = \sin(a+b+\sqrt{c}) \\
t_6 &= t_2 \cdot t_3 && = (a+b+\sqrt{c})\, b\sqrt{c} \\
\mathcal{B} &= t_4 + t_5 + t_6
\end{aligned}$$

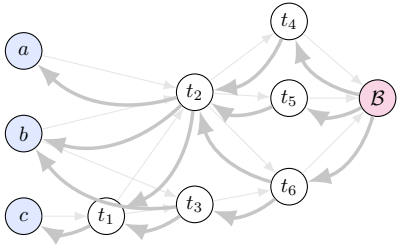


- t_2 feeds three nodes, t_1 and b feed two nodes each

Computation graphs: the backward pass for $\mathcal{B}$

Sheet 1 — Solution 1.3

Task c) Write the product each edge sends back. d) At each node, write the sum that gives the derivative of the output



- **c)** every edge $u \to v$ sends $\frac{\partial \mathcal{B}}{\partial v}\frac{\partial v}{\partial u}$ back to u
- **d)** every node sums what it receives. Nodes with one child get one term, for example $\frac{\partial \mathcal{B}}{\partial t_3} = \frac{\partial \mathcal{B}}{\partial t_6}\frac{\partial t_6}{\partial t_3}$

$$\frac{\partial \mathcal{B}}{\partial t_2} = \frac{\partial \mathcal{B}}{\partial t_4}\frac{\partial t_4}{\partial t_2} + \frac{\partial \mathcal{B}}{\partial t_5}\frac{\partial t_5}{\partial t_2} + \frac{\partial \mathcal{B}}{\partial t_6}\frac{\partial t_6}{\partial t_2}$$

$$\frac{\partial \mathcal{B}}{\partial t_1} = \frac{\partial \mathcal{B}}{\partial t_2}\frac{\partial t_2}{\partial t_1} + \frac{\partial \mathcal{B}}{\partial t_3}\frac{\partial t_3}{\partial t_1} \qquad \frac{\partial \mathcal{B}}{\partial b} = \frac{\partial \mathcal{B}}{\partial t_2}\frac{\partial t_2}{\partial b} + \frac{\partial \mathcal{B}}{\partial t_3}\frac{\partial t_3}{\partial b}$$

Computation graphs: the graph of $\mathcal{C}$

Sheet 1 — Solution 1.3

Task a) Introduce intermediate variables, one operation each. b) Draw the computation graph

$$\mathcal{C} = \sum_{i=1}^{3} (w_0 + w_1 x_i - y_i)^2$$ (the squared error of linear regression on 3 points)

For each data point $i = 1, 2, 3$

$z_i = w_1 \cdot x_i$ (weighted input)

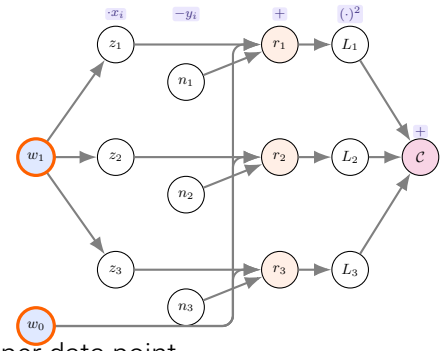
$n_i = -1 \cdot y_i$ (negative target)

$r_i = w_0 + z_i + n_i$ (residual)

$= w_0 + w_1 x_i - y_i$

$L_i = r_i^2$ (loss of point i)

$\mathcal{C} = L_1 + L_2 + L_3$

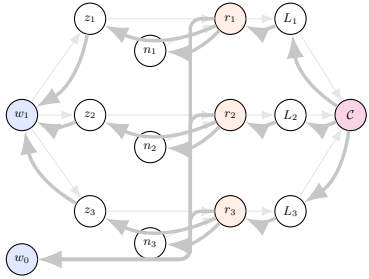


- w_0 and w_1 feed three nodes each, one per data point

Computation graphs: the backward pass for $\mathcal{C}$

Sheet 1 — Solution 1.3

Task c) Write the product each edge sends back. d) At each node, write the sum that gives the derivative of the output



- **c)** every edge $u \to v$ sends $\frac{\partial \mathcal{C}}{\partial v}\frac{\partial v}{\partial u}$ back to u
- **d)** every node sums what it receives, only w_0 and w_1 get more than one term
- In EDF, `Parameter.addgrad` sums over the batch

$$\frac{\partial \mathcal{C}}{\partial w_1} = \sum_{i=1}^{3} \frac{\partial \mathcal{C}}{\partial z_i}\frac{\partial z_i}{\partial w_1} = \sum_{i=1}^{3} 2 r_i \, x_i \qquad \frac{\partial \mathcal{C}}{\partial w_0} = \sum_{i=1}^{3} \frac{\partial \mathcal{C}}{\partial r_i}\frac{\partial r_i}{\partial w_0} = \sum_{i=1}^{3} 2 r_i$$

- The weights are used once per data point, so their gradients have **one term per point** (as in 1.2)

Logistic regression: deriving the gradient

Sheet 1 — Solution 1.4

Task Show that $\nabla_{\mathbf{w}} \frac{1}{B} \sum_i \mathcal{L}(y_i, \hat{y}_i) = \frac{1}{B} \sum_i (\hat{y}_i - y_i)\, \mathbf{x}_i$

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

$$\underset{1\times D}{\frac{\partial \mathcal{L}}{\partial \mathbf{w}}} = \underset{1\times 1}{\frac{\partial \mathcal{L}}{\partial \hat{y}_i}} \underset{1\times 1}{\frac{\partial \hat{y}_i}{\partial h_i}} \underset{1\times D}{\frac{\partial h_i}{\partial \mathbf{w}}}$$

$$= -\left(\frac{y_i}{\hat{y}_i} - \frac{1 - y_i}{1 - \hat{y}_i}\right) \underbrace{\hat{y}_i(1 - \hat{y}_i)}_{\sigma'(h_i),\ \text{1.1 c)}} \mathbf{x}_i^\top$$

$$= -\big(y_i(1 - \hat{y}_i) - (1 - y_i)\, \hat{y}_i\big)\, \mathbf{x}_i^\top = (\hat{y}_i - y_i)\, \mathbf{x}_i^\top$$

- The gradient is the transpose, $(\hat{y}_i - y_i)\, \mathbf{x}_i$, a column like $\mathbf{w}$
- The gradient of the mean is the mean of the gradients, $\frac{1}{B} \sum_i (\hat{y}_i - y_i)\, \mathbf{x}_i$

The code, solution 2.3 a)

Linear vs. logistic regression

	Linear regression (1.2)	**Logistic regression** (1.4)
Score	$h = \mathbf{w}^\top \mathbf{x}$	$h = \mathbf{w}^\top \mathbf{x}$
Target y	a real number	a label, 0 or 1
Output $\hat{y}$	$\hat{y} = h$	$\hat{y} = \sigma(h)$, a probability
Loss	$(\hat{y} - y)^2$	$-y \log \hat{y} - (1 - y) \log(1 - \hat{y})$
Likelihood of y	Gaussian	Bernoulli
Gradient $\nabla_{\mathbf{w}} \mathcal{L}$	$2(\hat{y} - y)\,\mathbf{x}$	$(\hat{y} - y)\,\mathbf{x}$
Solution	closed form (2.2 b)	gradient descent only

- The same linear score. The distribution of y decides the output function and the loss
- Both gradients are prediction minus target, times the input. In 1.4, σ' cancels against the derivative of the cross-entropy

Solutions of Exercise Sheet 1

2. Coding

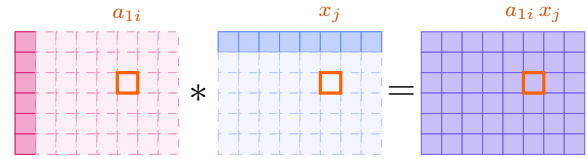
Broadcasting: an outer product

Sheet 1 — Solution 2.1

Task Implement `vector_derivative_product(A, x)`, the matrix of all $\hat{w}_{ij} = a_{1i}\, x_j$

implements 1.1 e) iii

`A (1, 64)` and `x (32,)`



- `*` copies the column to the right and the row down (dashed), both become `(64, 32)`
- Then it multiplies entry by entry, so entry (i, j) is $a_{1i}\, x_j$

```
return A.reshape(-1, 1) * x.reshape(1, -1)
```

- The same result with `A.T * x`, `A.T @ x[None, :]`, `np.outer(A, x)`

Linear regression: the derivatives in code

Sheet 1 — Solution 2.2 a)

Task Implement `compute_derivatives`, the two derivatives of the loss (implements 1.2)

With the residual $r_i = w_0 + w_1 x_i - y_i$, solution 1.2 gives

$$\frac{\partial \mathcal{L}}{\partial w_0} = \frac{2}{N} \sum_{i=1}^{N} r_i, \qquad \frac{\partial \mathcal{L}}{\partial w_1} = \frac{2}{N} \sum_{i=1}^{N} r_i \, x_i$$

```
def compute_derivatives(x, y, slope, offset_y):
    r = offset_y + slope * x - y            # (N,), all residuals at once
    return 2 * np.mean(r * x), 2 * np.mean(r)
```

- `r` holds all N residuals, so there is no loop over the points
- `np.mean` is the $\frac{1}{N} \sum_i$, `r * x` multiplies entry by entry
- The function returns the slope derivative first, then the offset derivative

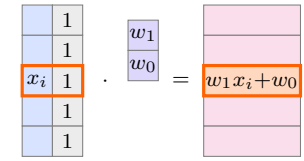
The derivation, solution 1.2

Linear regression: the gradient in matrix form

Sheet 1 — Solution 2.2 b)

Task Implement `compute_weights_cf(X, y)`, the closed form solution (the derivatives of 1.2 set to zero)

Every row of $\mathbf{X}$ is $(x_i, 1)$



Row i of $\mathbf{Xw}$ is the prediction for point i, with slope w_1 and offset w_0

- The loss in matrix form $\mathcal{L}(\mathbf{w}) = \|\mathbf{Xw} - \mathbf{y}\|^2$ (the $\frac{1}{N}$ does not move the minimum)
- Its gradient, the derivatives of

  solution 1.2

  stacked (without the $\frac{1}{N}$, r_i the residuals)
- Row j of $\mathbf{X}^\top$ times the residuals is the sum over the points, and r_i is entry i of $\mathbf{Xw} - \mathbf{y}$

$$\nabla_{\mathbf{w}} \mathcal{L} = 2 \begin{pmatrix} \sum_i r_i x_i \\ \sum_i r_i \end{pmatrix} = 2 \begin{pmatrix} x_1 & \cdots & x_N \\ 1 & \cdots & 1 \end{pmatrix} \begin{pmatrix} r_1 \\ \vdots \\ r_N \end{pmatrix} = 2\mathbf{X}^\top(\mathbf{Xw} - \mathbf{y})$$

Linear regression: the closed form

Sheet 1 — Solution 2.2 b)

Task Implement `compute_weights_cf(X, y)`, the closed form solution (the derivatives of 1.2 set to zero)

- The gradient from the last slide, $\nabla_{\mathbf{w}} \mathcal{L} = 2\mathbf{X}^\top(\mathbf{X}\mathbf{w} - \mathbf{y})$
- At the minimum it is zero, $\mathbf{X}^\top\mathbf{X}\,\mathbf{w} = \mathbf{X}^\top\mathbf{y}$
- $\mathbf{X}^\top\mathbf{X}$ is 2×2 and invertible here (the x_i are not all equal, so the two columns of $\mathbf{X}$ are independent), so

$$\underbrace{\mathbf{w}}_{2} = \underbrace{(\mathbf{X}^\top\mathbf{X})^{-1}}_{2\times 2}\,\underbrace{\mathbf{X}^\top}_{2\times 100}\,\underbrace{\mathbf{y}}_{100}$$

```
w = np.linalg.solve(X.T @ X, X.T @ y)       # solves X^T X w = X^T y
```

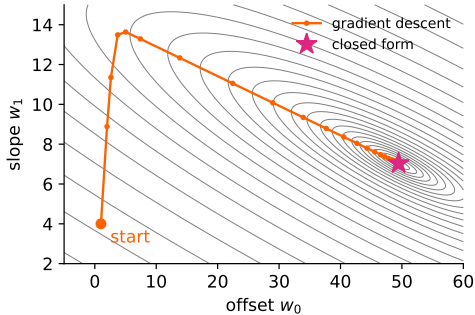
- Why not `inv(X.T @ X) @ X.T @ y`? `solve` never computes the inverse, so it is **faster** and **more accurate**

Closed form vs. gradient descent

Sheet 1 — Solution 2.2 a-b)

Task Train with the derivatives (2.2 a) and with the closed form (2.2 b), and check that the line fits (2.2 a) implements 1.2)

- Closed form, $w_1 = 7.04$ (slope) and $w_0 = 49.46$ (offset) in one step
- Gradient descent (2.2 a) needs 100 000 steps to get close (7.06, 49.27)
- The data has x between 0 and 10. In the prediction $w_1 x + w_0$, a change in w_1 is multiplied by x (up to 10), a change in w_0 only by 1
- So the loss is much steeper in w_1 than in w_0. The step size must be small for w_1, and then w_0 moves slowly
- Preprocessing fixes this (lecture 4)

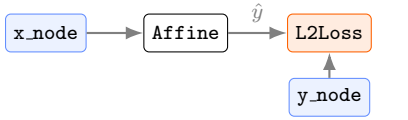


the loss as contours, and the path of gradient descent

Linear regression: a node for the squared error

Sheet 1 – Solution 2.2 c)

Task Implement the `L2Loss` node (forward and backward) and the graph of linear regression in EDF (the backward is the derivative of 1.2)



- Forward, one squared error per sample $L_i = (\hat{y}_i - y_i)^2$, `(B, 1)`
- Backward, $\frac{\partial}{\partial \hat{y}}(\hat{y} - y)^2 = 2(\hat{y} - y)$, times the incoming `grad`
  y is an `Input`, its gradient is dropped
- No mean in the node. `Backward` starts with `loss.grad` $= 1/B$
- Both must be `(B, 1)`. With `(B,)`, the difference is `(B, B)` (tutorial 2)

```
self.value = (self.y_hat.value - self.y.value) ** 2              # forward
d = 2 * (self.y_hat.value - self.y.value)                         # backward
self.y_hat.addgrad(self.grad * d);   self.y.addgrad(-self.grad * d)
```

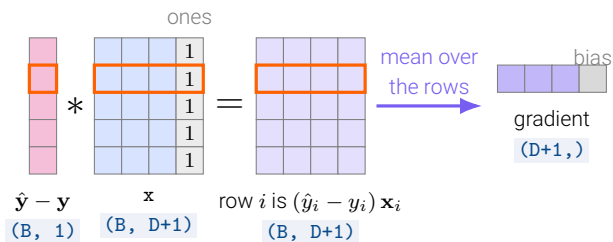
Logistic regression: the gradient in NumPy

Sheet 1 — Solution 2.3 a)

Task Implement `compute_derivative(x, y, y_hat)`, the gradient $\frac{1}{B}\sum_i(\hat{y}_i - y_i)\,\mathbf{x}_i$

implements 1.4

`x (B, D+1)`, the D pixels and a **column of ones** (the last weight is the bias), $\hat{y}_i = \sigma(\mathbf{w}^\top \mathbf{x}_i)$ (1.4)




```
return np.mean((y_hat - y).reshape(-1, 1) * x, axis=0)
```

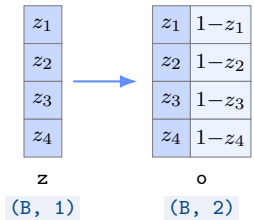
- The same as a matrix product (tutorial 2) `x.T @ (y_hat - y) / len(y)`
- Without `.reshape(-1, 1)`, `(B,) * (B, D+1)` does not broadcast
- With the ones, one gradient holds the weights and the bias, one step updates both

Logistic regression: from one probability to two

Sheet 1 — Solution 2.3 b)

Task Implement `SingleProbToProbVector`, from $P(Y{=}1 \mid \mathbf{x})$ `(B, 1)` to both probabilities `(B, 2)`, and the graph (the model of 1.4 in EDF)

- So far, **one probability per sample**, $z_i = \hat{y}_i = \sigma(\mathbf{w}^\top \mathbf{x}_i)$, `z` `(B, 1)`
- With two classes, the second probability is $1 - z_i$
- EDF's `CrossEntropyLoss` expects **one probability per class**, `(B, C)`, and takes the entry of the true class

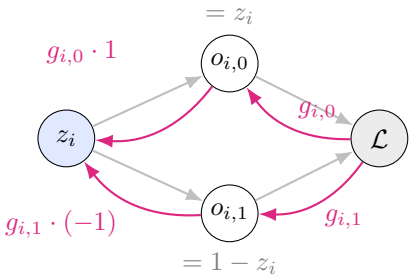


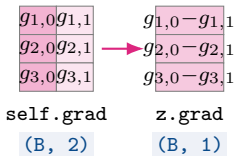
Forward, one new column

```
np.concatenate([z, 1 - z], axis=1)
```

Logistic regression: the backward pass

Sheet 1 — Solution 2.3 b)





- z_i reaches the loss on **two paths**, through both columns
- `self.grad` brings $g_{i,0} = \frac{\partial \mathcal{L}}{\partial o_{i,0}}$ and $g_{i,1} = \frac{\partial \mathcal{L}}{\partial o_{i,1}}$
- Each path multiplies by its local derivative, 1 and -1
- The paths meet at z_i, so they are **summed** (chain rule)
  $\frac{\partial \mathcal{L}}{\partial z_i} = g_{i,0} \cdot 1 + g_{i,1} \cdot (-1) = g_{i,0} - g_{i,1}$

All samples at once

```
(g[:, 0] - g[:, 1]).reshape(-1, 1)
```

`(B,)` reshaped back to `(B, 1)`

Logistic regression: test error vs. resolution

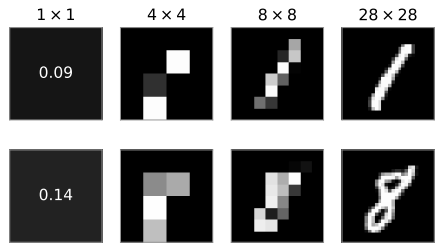
Sheet 1 — Solution 2.3 c)

Task Train the EDF model on the 4×4, 8×8 and 28×28 versions of the images (the model of 1.4)

Logistic regression in EDF, digits 1 and 8, 10 epochs, D input features

Train 12 593 images (6 742 ones, 5 851 eights), test 2 109 images (1 135 ones, 974 eights)

Input	D	Test error	Accuracy
1×1 (mean)	1	10.3 %	89.7 %
4×4	16	7.6 %	92.4 %
8×8	64	2.7 %	97.3 %
28×28	784	1.0 %	99.0 %



a 1 and an 8 from the training set, with their mean pixel

- The mean brightness already separates many 1s from 8s (an 8 has more ink)
- More pixels give the model more information, and the error drops