10907 Pattern Recognition

Lecturers
Prof. Dr. Ivan Dokmanić ⟨ivan.dokmanic@unibas.ch⟩

Tutors
Felicitas Haag ⟨felicitas.haag@unibas.ch⟩
Alexandra Spitzer ⟨alexandra.spitzer@unibas.ch⟩
Cheng Shi ⟨cheng.shi@unibas.ch⟩
Vinith Kishore ⟨vinith.kishore@unibas.ch⟩

Problem set 2 [Model Solutions]

Math

Exercise 1 (Matrix calculus basics - $\star$).

Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{x} \in \mathbb{R}^n$. Consider the scalar function

$$f(\boldsymbol{x}) \;=\; \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x}.$$

1. Show that the gradient with respect to x is

$$\nabla_x f(\boldsymbol{x}) \;=\; (\boldsymbol{A} + \boldsymbol{A}^\top)\,\boldsymbol{x}.$$

   Hint: There are multiple ways to go about this, one is to write $f(\boldsymbol{x}) = \sum_{i,j} \boldsymbol{x}_i \boldsymbol{A}_{ij} \boldsymbol{x}_j$ and differentiate w.r.t. $\boldsymbol{x}_k$; another way is to try to understand $f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} = (\boldsymbol{A}\boldsymbol{x})^\top \boldsymbol{x} = g(\boldsymbol{x})^\top \boldsymbol{x}$ with $g(\boldsymbol{x}) =$ and then apply the chain rule.

2. Show that if $\boldsymbol{A}$ is symmetric, then $\nabla_x f(\boldsymbol{x}) = 2\boldsymbol{A}\boldsymbol{x}$.

3. Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{x} \in \mathbb{R}^n$. Show that

$$\|\boldsymbol{A}\boldsymbol{x}\|^2 \;=\; \boldsymbol{x}^\top \boldsymbol{A}^\top \boldsymbol{A}\,\boldsymbol{x}.$$

4. Let $\boldsymbol{X}, \boldsymbol{Y} \in \mathbb{R}^{m\times n}$. Show that

$$\nabla_X \tfrac{1}{2}\|\boldsymbol{X} - \boldsymbol{Y}\|^2 = \boldsymbol{X} - \boldsymbol{Y}.$$

1. Component-wise derivation: $f(x) = \sum_{i=1}^n \sum_{j=1}^n x_i A_{ij} x_j$. Then for each k,

$$\frac{\partial f}{\partial x_k} = \sum_j A_{kj} x_j \;+\; \sum_i A_{ik} x_i = (Ax)_k + (A^\top x)_k.$$

   Stacking over k gives $\nabla_x f(x) = (A + A^\top)x$.

   Using chain/product rule: $\nabla_x f(\boldsymbol{x}) = d(\boldsymbol{A}\boldsymbol{x})^\top \boldsymbol{x} + (\boldsymbol{A}\boldsymbol{x})^\top d\boldsymbol{x}$.

   Since $d(\boldsymbol{A}\boldsymbol{x}) = \boldsymbol{A}\,d\boldsymbol{x}$, we have

$$(d(\boldsymbol{A}\boldsymbol{x}))^\top \boldsymbol{x} \;=\; (\boldsymbol{A}\,d\boldsymbol{x})^\top \boldsymbol{x} \;=\; (d\boldsymbol{x})^\top \boldsymbol{A}^\top \boldsymbol{x}.$$

   Hence

$$df \;=\; (d\boldsymbol{x})^\top \boldsymbol{A}^\top \boldsymbol{x} \;+\; (\boldsymbol{A}\boldsymbol{x})^\top d\boldsymbol{x} \;=\; \left(\boldsymbol{A}^\top \boldsymbol{x} + \boldsymbol{A}\boldsymbol{x}\right)^\top d\boldsymbol{x}.$$

   By the definition of the gradient (i.e., $df = (\nabla_{\boldsymbol{x}} f)^\top d\boldsymbol{x}$ for all $d\boldsymbol{x}$), it follows that

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = (\boldsymbol{A} + \boldsymbol{A}^\top)\,\boldsymbol{x}$$

2. If $A = A^\top$, then $\nabla_x f(x) = (A + A)x = 2Ax$.

3. $\|\boldsymbol{Ax}\|^2 = (\boldsymbol{Ax})^\top(\boldsymbol{Ax}) = \boldsymbol{x}^\top\boldsymbol{A}^\top\boldsymbol{A}\,\boldsymbol{x}$.
4. Let $f(\boldsymbol{X}) = \frac{1}{2}\|\boldsymbol{X}-\boldsymbol{Y}\|_F^2$ with

$$\|\boldsymbol{X}-\boldsymbol{Y}\|_F^2 = \sum_{i=1}^{m}\sum_{j=1}^{n}(x_{ij}-y_{ij})^2.$$

Then

$$f(\boldsymbol{X}) = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n}(x_{ij}-y_{ij})^2.$$

Differentiate elementwise:

$$\frac{\partial f}{\partial x_{ij}} = \frac{1}{2}\cdot 2(x_{ij}-y_{ij}) = x_{ij}-y_{ij}.$$

Hence the matrix of partial derivatives is

$$\nabla_X f(\boldsymbol{X}) = \left[\tfrac{\partial f}{\partial x_{ij}}\right]_{i,j} = \boldsymbol{X}-\boldsymbol{Y}.$$

Exercise 2 (Multivariate linear regression - ⋆⋆)**.**

In multivariate linear regression we define the residual sum of squares (RSS) as

$$\mathrm{RSS}(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y}\|_2^2, \tag{1}$$

where

- $\boldsymbol{X}$ is the $N\times D$ data matrix, containing D-dimensional features for N data points.
- $\boldsymbol{w}$ is the weight vector.
- $\boldsymbol{y}$ is the vector of target values for each training sample.

Solve the following tasks.

1. Partial derivatives: Show that

$$\frac{\partial \mathrm{RSS}(\boldsymbol{w})}{\partial w_k} = \|\boldsymbol{x}_{\cdot,k}\|_2^2 w_k - \boldsymbol{x}_{\cdot,k}^T(\boldsymbol{y}-\boldsymbol{X}_{\cdot,-k}\boldsymbol{w}_{-k})$$

where

- $\boldsymbol{x}_{\cdot,k}$ is the k-th column of $\boldsymbol{X}$;
- $\boldsymbol{X}_{\cdot,-k}$ is the data matrix $\boldsymbol{X}$ without the k-th column;
- $\boldsymbol{w}_{-k}$ is the weight vector without the k-th component.

Hint: Separate the k-th weigth component from the rest.

We consider the residual sum of squares

$$RSS(\mathbf{w}) = \tfrac{1}{2}\,\|X\mathbf{w}-\mathbf{y}\|_2^2,$$

where $X\in\mathbb{R}^{n\times d}$ is the data matrix, $\mathbf{w}\in\mathbb{R}^d$ is the weight vector, and $\mathbf{y}\in\mathbb{R}^n$ is the target vector.

—

Matrix dimensions.

$$X = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{pmatrix} \in \mathbb{R}^{n\times d}, \qquad \mathbf{x}_i^T \in \mathbb{R}^{1\times d}.$$

Equivalently,

$$X = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_d \end{bmatrix},$$

where $\mathbf{x}_j$ is the j-th column of X.

- The weight vector $\mathbf{w}$ must have dimension $d \times 1$, so that $X\mathbf{w}$ is valid ($n \times d$ times $d \times 1 = n \times 1$).
- The target vector $\mathbf{y}$ has dimension $n \times 1$.

—

Partial derivative. Let $\mathbf{x}_{:,k}$ denote the k-th column of X, and let X_{-k} be X without its k-th column. Similarly, $\mathbf{w}_{-k}$ is the weight vector without the k-th entry.

Then,

$$\frac{\partial RSS(\mathbf{w})}{\partial w_k} = \|\mathbf{x}_{:,k}\|_2^2 \, w_k \;-\; \mathbf{x}_{:,k}^T \Big(\mathbf{y} - X_{-k}\mathbf{w}_{-k}\Big).$$

—

Expansion of RSS.

$$\begin{aligned} RSS(\mathbf{w}) &= \tfrac{1}{2}\, \|X\mathbf{w} - \mathbf{y}\|_2^2 \\ &= \tfrac{1}{2} \sum_{i=1}^{n} \big(\mathbf{x}_i^T \mathbf{w} - y_i\big)^2 \\ &= \tfrac{1}{2} \sum_{i=1}^{n} \left(\sum_{j=1}^{d} x_{ij} w_j - y_i \right)^2 . \end{aligned}$$

If we pull out the k-th component explicitly:

$$RSS(\mathbf{w}) = \tfrac{1}{2} \sum_{i=1}^{n} \left(x_{ik} w_k + \sum_{\substack{j=1 \\ j\neq k}}^{d} x_{ij} w_j - y_i \right)^2 .$$

—

Note. For any vector $\mathbf{z} \in \mathbb{R}^N$,

$$\|\mathbf{z}\|_2^2 = z_1^2 + z_2^2 + \cdots + z_N^2 .$$

Now we take the partial derivative with respect to w_k:

$$\frac{\partial RSS}{\partial w_k} = \frac{1}{2} \cdot 2 \cdot \sum_{i=1}^{n} \left(x_{ik} w_k + \sum_{\substack{j=1 \\ j\neq k}}^{d} x_{ij} w_j - y_i \right) \cdot (x_{ik}).$$

University of Basel

$$= \sum_{i=1}^{n} \Big(x_{ik}^2 w_k + x_{ik} \Big(\sum_{\substack{j=1 \\ j \neq k}}^{d} x_{ij} w_j - y_i \Big) \Big).$$

Using the definition of the ℓ_2-norm, we can rewrite this as

$$\frac{\partial RSS}{\partial w_k} = \|X_{:,k}\|_2^2 \, w_k - X_{:,k}^T \Big(y - X_{-k} w_{-k} \Big),$$

where $X_{:,k}$ denotes the k-th column of X (a vector in $\mathbb{R}^n$) and X_{-k} is X without the k-th column

2. Optimal weights: We define the prediction residuals as

$$\boldsymbol{r}_k = \boldsymbol{y} - \boldsymbol{X}_{\cdot,-k} \boldsymbol{w}_{-k}$$

for $k = 1, \ldots, D$. Prove that if $\frac{\partial \text{RSS}}{\partial w_k} = 0$, then the optimal weight for the k-th feature (k-th coordinate or dimension or pixel) is given by

$$\widehat{w}_k = \frac{\boldsymbol{x}_{\cdot,k}^T \boldsymbol{r}_k}{\|\boldsymbol{x}_{\cdot,k}\|^2}. \tag{2}$$

Recall from Part 1 that

$$\frac{\partial \, \text{RSS}(w)}{\partial w_k} = \|x_{:,k}\|_2^2 \, w_k - x_{:,k}^\top (y - X_{:,-k} w_{-k}) = \|x_{:,k}\|_2^2 \, w_k - x_{:,k}^\top r_k.$$

Setting the first-order condition to zero gives

$$0 = \|x_{:,k}\|_2^2 \, w_k - x_{:,k}^\top r_k,$$

hence, provided $\|x_{:,k}\|_2 > 0$,

$$\widehat{w}_k = \frac{x_{:,k}^\top r_k}{\|x_{:,k}\|_2^2} .$$

Since $\text{RSS}(w)$ is a convex quadratic in w_k, this stationary point is the minimizer. (If $x_{:,k} = 0$, RSS does not depend on w_k and any w_k is optimal.)

Interpretation Given current coefficients $\boldsymbol{w}_{-k}$ the optimal update for coordinate k is the projection of $\boldsymbol{x}_{\cdot,k}$ onto the current residual $\boldsymbol{r}_k$. Iterating these closed-form updates over all coordinates converges to the global minimizer because RSS is a convex quadratic.

Exercise 3 (Classification: optimal decision thresholds under a reject option - $\star\star$)**.**

In many applications, the classifier is allowed to "reject" a test example rather than classifying it into one of the classes. Consider, for example, a case in which the cost of a misclassification is 10 CHF and the reward (negative cost) of correct prediction is 0.5 CHF. On the other hand, the cost of an additional human evaluation is only 3 CHF. We can summarize this by the following loss matrix:

Decision	true label Y	
$\hat{Y}$	Y=0	Y=1
predict $\hat{Y} = 0 \mid x$	-0.5	10
predict $\hat{Y} = 1 \mid x$	10	-0.5
reject	3	3

1. Let us denote $p_1 = p(Y = 1|x)$ as the posterior probability of class 1 given the observed feature vector x. Show that in general, for this loss matrix, but for any posterior distribution, there will be two thresholds θ_0 and θ_1 such that the optimal decision is to predict $\hat{Y} = 0$ if $p_1 < \theta_0$, reject if $\theta_0 < p_1 < \theta_1$ and predict $\hat{Y} = 1$ if $p_1 > \theta_1$.

2. Compute θ_0 and θ_1.
3. Now change the costs so that a correct prediction yields -1 and a wrong prediction costs 20 (the rejection cost remains 3). Recompute the thresholds.

Posterior expected loss:

$$P(a \mid x) \; = \; \sum_y \ell(y,a)\, p(y \mid x).$$

Let $p_1 = p(y=1 \mid x)$ and $1-p_1 = p(y=0 \mid x)$. Then

$$P(a \mid x) = p(y=0 \mid x)\, \ell(0,a) + p(y=1 \mid x)\, \ell(1,a) = (1-p_1)\, \ell(0,a) + p_1\, \ell(1,a).$$

Case $a=0$ (predict $\hat{y}=0$).

$$\begin{aligned} P(0 \mid x) &= (1-p_1)\, \ell(0,0) + p_1\, \ell(1,0) \\ &= (1-p_1)(-0.5) + p_1 \cdot 10 \\ &= -\tfrac{1}{2}(1-p_1) + 10p_1. \end{aligned}$$

Case $a=1$ (predict $\hat{y}=1$).

$$\begin{aligned} P(1 \mid x) &= (1-p_1)\, \ell(0,1) + p_1\, \ell(1,1) \\ &= (1-p_1) \cdot 10 + p_1(-0.5) \\ &= 10(1-p_1) - \tfrac{1}{2}p_1. \end{aligned}$$

Case $a=2$ (reject).

$$\begin{aligned} P(2 \mid x) &= (1-p_1)\, \ell(0,2) + p_1\, \ell(1,2) \\ &= (1-p_1) \cdot 3 + p_1 \cdot 3 \\ &= 3(1-p_1) + 3p_1. \end{aligned}$$

To prefer choosing $a=0$ over rejection, we must have

$$(1-p_1)\, \ell(0,0) + p_1\, \ell(1,0) \; < \; (1-p_1)\, \ell(0,2) + p_1\, \ell(1,2).$$

This can be rearranged as

$$\begin{aligned} \ell(0,0) - p_1\ell(0,0) + p_1\ell(1,0) \; &< \; \ell(0,2) - p_1\ell(0,2) + p_1\ell(1,2), \\ p_1\big(\ell(1,0) - \ell(0,0) + \ell(0,2) - \ell(1,2)\big) \; &< \; -\ell(0,0) + \ell(0,2). \end{aligned}$$

Hence,

$$p_1 \; < \; \frac{\ell(0,2) - \ell(0,0)}{\ell(1,0) - \ell(0,0) + \ell(0,2) - \ell(1,2)} \; =: \; \theta_0.$$

To prefer choosing $a=1$ over rejection we must have

$$(1-p_1)\, \ell(0,1) + p_1\ell(1,1) \; < (1-p_1)\, \ell(0,2) + p_1\ell(1,2).$$

That is,

$$\begin{aligned} \ell(0,1) - p_1\ell(0,1) + p_1\ell(1,1) \; &< \; p_1(-\ell(0,2) + \ell(1,2)) + \ell(0,2), \\ p_1\big(\ell(0,2) + \ell(1,2) + \ell(0,1) - \ell(1,1)\big) \; &> \; -\ell(0,2) + \ell(0,1). \end{aligned}$$

Thus,

$$p_1 \; > \; \frac{-\ell(0,2) + \ell(0,1)}{-\ell(0,2) + \ell(1,2) + \ell(0,1) - \ell(1,1)} \; =: \; \theta_1.$$

Therefore, there exist thresholds θ_0 and θ_1 such that it is optimal to choose

$$a^* = \begin{cases} 0, & \text{if } p_1 < \theta_0, \\ 1, & \text{if } p_1 > \theta_1, \\ \text{reject}, & \text{if } \theta_0 < p_1 < \theta_1, \end{cases}$$

with $\theta_0 < \theta_1$.

Case 1: our given loss matrix.

$$\theta_0 = \frac{3+0.5}{10+0.5+3-3} = \tfrac{1}{3}, \qquad \theta_1 = \frac{-3+10}{-3+3+10+0.5} = \tfrac{2}{3}.$$

Case 2: doubled losses.

$$\theta_2 = \frac{3+1}{20+1+3-3} \approx 0.105, \qquad \theta_3 = \frac{-3+20}{20+1} \approx 0.810.$$

Notation note. We use uppercase letters for random variables (e.g., Y) and lowercase for realized values or given observations (e.g., x). Hats mark decisions, e.g., $\hat{Y}$.

Exercise 4 (Logistic Regression - ⋆⋆)**.**

Consider data in $\mathcal{X} = \mathbb{R}$ with binary labels in $\mathcal{Y} = \{+1, -1\}$. The joint distribution of data and labels is given as

$$p(\boldsymbol{x}, y) = \frac{1}{4\pi} \exp\left(-\frac{1}{2}\|\boldsymbol{x} - y\mathbf{1}\|_2^2\right) \tag{3}$$

where $\mathbf{1} = [1, 1]^T$ is the all-one vector.

- Determine $P[y = 1|\boldsymbol{x}]$ and $P[y = -1|\boldsymbol{x}]$.

  Each datapoint $\boldsymbol{x}$ belongs to $\mathbb{R}^2$, i.e.

  $$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad x_1, x_2 \in \mathbb{R},$$

  with a binary label $y \in \{+1, -1\}$.

  The joint distribution of $\boldsymbol{x}$ and y is

  $$p(\boldsymbol{x}, y) = \frac{1}{4\pi} \exp\left(-\tfrac{1}{2}\|\boldsymbol{x} - y\mathbb{1}\|_2^2\right),$$

  with

  $$\mathbb{1} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

  We want to determine $P(y = 1 \mid \boldsymbol{x})$ and $P(y = -1 \mid \boldsymbol{x})$, i.e. the probability that the output is ± 1 given the input vector $\boldsymbol{x}$.

  Bayes theorem:

  $$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}, \qquad \text{where } A, B \text{ are events and } P(B) \neq 0.$$

  Using Bayes' theorem in our case:

  $$P(y = 1 \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid y = 1)\, P(y = 1)}{p(\boldsymbol{x})},$$

  $$P(y = -1 \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid y = -1)\, P(y = -1)}{p(\boldsymbol{x})}.$$

  Notes:

  - We assume that $y = \pm 1$ is equally likely, so

    $$P(y = 1) = P(y = -1) = \tfrac{1}{2}.$$

- We can directly determine $p(\boldsymbol{x} \mid y)$ from the joint distribution:

$$p(\boldsymbol{x} \mid y = \pm 1) = \frac{p(\boldsymbol{x}, y = \pm 1)}{P(y = \pm 1)}.$$

- The marginal likelihood $p(\boldsymbol{x})$ is given by

$$p(\boldsymbol{x}) = \sum_y p(\boldsymbol{x}, y).$$

Let us first compute $p(\boldsymbol{x} \mid y = \pm 1)$. We need $p(\boldsymbol{x}, y = \pm 1)$ and $p(y = \pm 1)$ for this and

$$p(y = \pm 1) = \tfrac{1}{2}.$$

From the joint distribution we know

$$p(\boldsymbol{x}, y = 1) = \frac{1}{4\pi} \exp\left(-\tfrac{1}{2} \left\|\boldsymbol{x} - \mathbb{1}\right\|_2^2\right),$$

$$p(\boldsymbol{x}, y = -1) = \frac{1}{4\pi} \exp\left(-\tfrac{1}{2} \left\|\boldsymbol{x} - (-\mathbb{1})\right\|_2^2\right) = \frac{1}{4\pi} \exp\left(-\tfrac{1}{2} \left\|\boldsymbol{x} + \mathbb{1}\right\|_2^2\right).$$

Putting this together:

$$p(\boldsymbol{x} \mid y = 1) = \frac{p(\boldsymbol{x}, y = 1)}{P(y = 1)} = \frac{\frac{1}{4\pi} \exp\left(-\frac{1}{2} \left\|\boldsymbol{x} - \mathbb{1}\right\|_2^2\right)}{\frac{1}{2}} = \frac{1}{2\pi} \exp\left(-\tfrac{1}{2} \left\|\boldsymbol{x} - \mathbb{1}\right\|_2^2\right),$$

$$p(\boldsymbol{x} \mid y = -1) = \frac{p(\boldsymbol{x}, y = -1)}{P(y = -1)} = \frac{\frac{1}{4\pi} \exp\left(-\frac{1}{2} \left\|\boldsymbol{x} + \mathbb{1}\right\|_2^2\right)}{\frac{1}{2}} = \frac{1}{2\pi} \exp\left(-\tfrac{1}{2} \left\|\boldsymbol{x} + \mathbb{1}\right\|_2^2\right).$$

Thus, both conditional distributions $p(\boldsymbol{x} \mid y = \pm 1)$ are Gaussian densities centered at $\pm\mathbb{1}$ with covariance I.
Next we calculate the marginal likelihood $p(\boldsymbol{x})$:

$$p(\boldsymbol{x}) = \sum_y p(\boldsymbol{x}, y) = p(\boldsymbol{x}, y = 1) + p(\boldsymbol{x}, y = -1).$$

Using the joint distribution:

$$p(\boldsymbol{x}) = \frac{1}{4\pi} \left[\exp\left(-\tfrac{1}{2}\|\boldsymbol{x} - \mathbb{1}\|_2^2\right) + \exp\left(-\tfrac{1}{2}\|\boldsymbol{x} + \mathbb{1}\|_2^2\right)\right].$$

Putting it all together, the posterior probability becomes

$$P(y = \pm 1 \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid y = \pm 1)\, P(y = \pm 1)}{p(\boldsymbol{x})}.$$

We want to compute

$$P(y = 1 \mid \boldsymbol{x}) = \frac{\frac{1}{2\pi} \exp\left(-\frac{1}{2}\|\boldsymbol{x} - \mathbb{1}\|^2\right) \cdot \frac{1}{2}}{\frac{1}{4\pi} \left(\exp\left(-\frac{1}{2}\|\boldsymbol{x} - \mathbb{1}\|^2\right) + \exp\left(-\frac{1}{2}\|\boldsymbol{x} + \mathbb{1}\|^2\right)\right)}.$$

Simplifying,

$$P(y = 1 \mid \boldsymbol{x}) = \frac{\exp\left(-\frac{1}{2}\|\boldsymbol{x} - \mathbb{1}\|^2\right)}{\exp\left(-\frac{1}{2}\|\boldsymbol{x} - \mathbb{1}\|^2\right) + \exp\left(-\frac{1}{2}\|\boldsymbol{x} + \mathbb{1}\|^2\right)}.$$

Factor out the first exponential:

$$P(y = 1 \mid \boldsymbol{x}) = \frac{1}{1 + \exp\left(-\frac{1}{2}\|\boldsymbol{x} + \mathbb{1}\|^2 + \frac{1}{2}\|\boldsymbol{x} - \mathbb{1}\|^2\right)}.$$

—

Expanding the norms. For $\boldsymbol{x} = (x_1, x_2)^T \in \mathbb{R}^2$ and $\boldsymbol{\mu} = (1, 1)^T$:

$$\|\boldsymbol{x} + \mathbb{1}\|^2 = (x_1 + 1)^2 + (x_2 + 1)^2 = x_1^2 + x_2^2 + 2x_1 + 2x_2 + 2,$$
$$\|\boldsymbol{x} - \mathbb{1}\|^2 = (x_1 - 1)^2 + (x_2 - 1)^2 = x_1^2 + x_2^2 - 2x_1 - 2x_2 + 2.$$

Thus,

$$-\tfrac{1}{2}\|\boldsymbol{x} + \mathbb{1}\|^2 + \tfrac{1}{2}\|\boldsymbol{x} - \mathbb{1}\|^2 = -2x_1 - 2x_2.$$

—

Final result. Therefore,

$$P(y = 1 \mid \boldsymbol{x}) = \frac{1}{1 + \exp(-2(x_1 + x_2))} = \sigma(2(x_1 + x_2)),$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the logistic sigmoid.
Similarly,

$$P(y = -1 \mid \boldsymbol{x}) = 1 - P(y = 1 \mid \boldsymbol{x}).$$

- Compute the weight vector for logistic regression which satisfies

$$\begin{bmatrix} P[y=-1|\boldsymbol{x}] \\ P[y=1|\boldsymbol{x}] \end{bmatrix} = f_{\boldsymbol{w}}(\boldsymbol{x}) := \begin{bmatrix} e^{\boldsymbol{w}_1^\intercal \boldsymbol{x}}/(e^{\boldsymbol{w}_1^\intercal \boldsymbol{x}} + e^{\boldsymbol{w}_2^\intercal \boldsymbol{x}}) \\ e^{\boldsymbol{w}_2^\intercal \boldsymbol{x}}/(e^{\boldsymbol{w}_1^\intercal \boldsymbol{x}} + e^{\boldsymbol{w}_2^\intercal \boldsymbol{x}}) \end{bmatrix}$$

Determine all possible values of $\boldsymbol{w} = (\boldsymbol{w}_1, \boldsymbol{w}_2)$ (Hint: Consider the shift invariance.)

We obtained

$$P(y = 1 \mid \boldsymbol{x}) = \frac{1}{1 + \exp(-2x_1 - 2x_2)}.$$

Multiplying numerator and denominator by $\exp(x_1 + x_2)$ gives

$$P(y = 1 \mid \boldsymbol{x}) = \frac{\exp(x_1 + x_2)}{\exp(x_1 + x_2) + \exp(-x_1 - x_2)}. \tag{*}$$

Similarly, for $y = -1$ we obtain

$$P(y = -1 \mid \boldsymbol{x}) = \frac{\exp(-x_1 - x_2)}{\exp(x_1 + x_2) + \exp(-x_1 - x_2)}. \tag{**}$$

—

Connection to logistic regression. For logistic regression (softmax form), the probability vector is

$$\begin{bmatrix} P(y = -1 \mid \boldsymbol{x}) \\ P(y = 1 \mid \boldsymbol{x}) \end{bmatrix} = f_{\boldsymbol{\omega}}(\boldsymbol{x}) := \begin{bmatrix} \dfrac{\exp(\boldsymbol{\omega}_1^T \boldsymbol{x})}{\exp(\boldsymbol{\omega}_1^T \boldsymbol{x}) + \exp(\boldsymbol{\omega}_2^T \boldsymbol{x})} \\ \dfrac{\exp(\boldsymbol{\omega}_2^T \boldsymbol{x})}{\exp(\boldsymbol{\omega}_1^T \boldsymbol{x}) + \exp(\boldsymbol{\omega}_2^T \boldsymbol{x})} \end{bmatrix}.$$

The first entry corresponds to

$$P(y = -1 \mid \boldsymbol{x}) = \frac{\exp(\boldsymbol{\omega}_1^T \boldsymbol{x})}{\exp(\boldsymbol{\omega}_1^T \boldsymbol{x}) + \exp(\boldsymbol{\omega}_2^T \boldsymbol{x})}, \tag{Δ}$$

and the second entry to

$$P(y = 1 \mid \boldsymbol{x}) = \frac{\exp(\boldsymbol{\omega}_2^T \boldsymbol{x})}{\exp(\boldsymbol{\omega}_1^T \boldsymbol{x}) + \exp(\boldsymbol{\omega}_2^T \boldsymbol{x})}. \tag{$\Delta\Delta$}$$

—

Matching terms. Comparing $(*)$ with (Δ), we require

$$\boldsymbol{\omega}_1^T\boldsymbol{x} = -x_1 - x_2, \qquad \boldsymbol{\omega}_2^T\boldsymbol{x} = x_1 + x_2.$$

Thus, we can choose

$$\boldsymbol{\omega}_1 = \begin{bmatrix}-1\\-1\end{bmatrix}, \qquad \boldsymbol{\omega}_2 = \begin{bmatrix}1\\1\end{bmatrix}.$$

Then

$$\boldsymbol{\omega}_1^T\boldsymbol{x} = -x_1 - x_2, \qquad \boldsymbol{\omega}_2^T\boldsymbol{x} = x_1 + x_2,$$

and both (Δ) and $(\Delta\Delta)$ match with $(*)$ and $(**)$.

—

General solution with shift invariance. Because of the shift invariance property of the softmax function, we may add a constant $k \in \mathbb{R}$ to both weight vectors without changing the probabilities:

$$\boldsymbol{\omega}_1 = \begin{bmatrix}-1\\-1\end{bmatrix} + k\mathbb{1}, \qquad \boldsymbol{\omega}_2 = \begin{bmatrix}1\\1\end{bmatrix} + k\mathbb{1}, \qquad \text{for all } k \in \mathbb{R}.$$

Exercise 5 (Properties of convolution - ⋆⋆)**.**

Consider the discrete convolution as defined in class, for simplicity in 1D,

$$(x * h)[n] = \sum_{m=-\infty}^{\infty} x[n-m]h[m].$$

Note: in practice we work with filters h of "finite support"—that is, such that $h[n] = 0$ for $|n| > L$ for some $L < \infty$, but there is no difficulty in working with infinite support filters as long as they decay sufficiently fast. In fact, the math is cleaner.

Show that

- convolution is commutative, $x * h = h * x$;

$$(x * h)[n] = \sum_{m\in\mathbb{Z}} x[n-m]\,h[m] \overset{k:=n-m}{=} \sum_{k\in\mathbb{Z}} x[k]\,h[n-k] = (h * x)[n].$$

- it is also associative, $(x * h) * g = x * (h * g)$, meaning that we do not have to write the parentheses.
Compute the left–hand side:

$$\begin{aligned}\big((x * h) * g\big)[n] &= \sum_{m\in\mathbb{Z}} (x*h)[n-m]\,g[m] = \sum_{m\in\mathbb{Z}} \left(\sum_{\ell\in\mathbb{Z}} x[n-m-\ell]\,h[\ell]\right) g[m] \\ &= \sum_{m\in\mathbb{Z}}\sum_{\ell\in\mathbb{Z}} x[n-(m+\ell)]\,h[\ell]\,g[m] \overset{k:=m+\ell}{=} \sum_{m\in\mathbb{Z}}\sum_{\ell\in\mathbb{Z}} x[n-k]\,h[\ell]\,g[k-\ell] \\ &= \sum_{k\in\mathbb{Z}} x[n-k]\left(\sum_{\ell\in\mathbb{Z}} h[\ell]\,g[k-\ell]\right) \\ &= \sum_{k\in\mathbb{Z}} x[n-k]\,(h*g)[k] = \big(x * (h*g)\big)[n].\end{aligned}$$

Consider now cross-correlation, defined rather similarly

$$(x \star h)[n] = \sum_{m=-\infty}^{\infty} x[n+m]h[m].$$

Check whether

- cross-correlation is commutative. (prove it)

  No, this does not hold in general.

$$(x \star h)[n] = \sum_{m\in\mathbb{Z}} x[n+m]\,h[m] \overset{k:=n+m}{=} \sum_{k\in\mathbb{Z}} x[k]\,h[k-n] \neq (h \star x)[n].$$

- cross-correlation is associative. (prove it)

  Define the time–reversed sequence $\tilde{h}[m] := h[-m]$. Then

$$(x \star h)[n] = \sum_m x[n+m]h[m] = \sum_m x[n-m]\tilde{h}[m] = (x * \tilde{h})[n].$$

  So correlation is just convolution with a time–reversed filter.

  For convolution we know

$$(x * h) * g \;=\; x * (h * g).$$

  But with correlation,

$$(x \star h) \star g \;=\; \big(x * \tilde{h}\big) * \tilde{g}, \qquad x \star (h \star g) \;=\; x * \big(\widetilde{h \star g}\big).$$

  The two expressions involve *different reversals*: In the first, h and g are each reversed individually, in the second, the composite $(h \star g)$ is reversed as a whole.

  In general,

$$\widetilde{h \star g} \;\neq\; \tilde{h} * \tilde{g},$$

  so correlation does not satisfy associativity.

Exercise 6 (Gradient descent for logistic regression - $\star\star\star$)**.**

Let $y^{(n)} \in \{e_1, \ldots, e_K\} \subset \mathbb{R}^K$ be one-hot labels and $W \in \mathbb{R}^{K\times d}$. For scores $Wx^{(n)} \in \mathbb{R}^K$ define the softmax probabilities

$$\hat{y}_k^{(n)} \;=\; \frac{\exp\big(Wx_k^{(n)}\big)}{\sum_{j=1}^K \exp\big(Wx_j^{(n)}\big)}, \qquad k = 1, \ldots, K,$$

and the cross-entropy

$$\ell\big(\hat{y}^{(n)}, y^{(n)}\big) \;=\; -\sum_{k=1}^K y_k^{(n)} \log \hat{y}_k^{(n)}.$$

and the empirical risk

$$R_N(W) \;=\; \frac{1}{N}\sum_{n=1}^N \ell\big(\hat{y}_k^{(n)},\, y^{(n)}\big).$$

Derive the gradient of the empirical risk $R_N(W)$ with respect to W and show that

$$\nabla_W R_N(W) \;=\; \frac{1}{N}\sum_{n=1}^N \big(\hat{y}^{(n)} - y^{(n)}\big)\,\big(x^{(n)}\big)^\top.$$

University of Basel

The empirical risk

$$R_N(W) \;=\; \frac{1}{N}\sum_{n=1}^{N} \ell\left(\hat{y}_k^{(n)},\, y^{(n)}\right).$$

is defined as a sum of loss functions. In order to derive this gradient we need to apply the chain rule mutliple times. To make the calculation easier let's define scores and reformulate the softmax

$$z^{(n)} := W\,x^{(n)} \in \mathbb{R}^K, \qquad \hat{y}_k^{(n)} := \frac{e^{z_k^{(n)}}}{\sum_{j=1}^{K} e^{z_j^{(n)}}},\; k = 1,\ldots,K.$$

Step 1: We first calculate the gradient w.r.t. the scores $z^{(n)}$. Keep in mind that the derivative we are looking for can be expressed as the sum of derivatives of the loss function. Using the chain rule,

$$\frac{\partial \ell}{\partial z_j^{(n)}} = \sum_{k=1}^{K} \frac{\partial \ell}{\partial \hat{y}_k^{(n)}} \frac{\partial \hat{y}_k^{(n)}}{\partial z_j^{(n)}}, \qquad \frac{\partial \ell}{\partial \hat{y}_k^{(n)}} = -\frac{y_k^{(n)}}{\hat{y}_k^{(n)}}.$$

For the dervative of the softmax, write $S = \sum_{r=1}^{K} e^{z_r}$. Then $\hat{y}_k = \frac{e^{z_k}}{S}$ and $\frac{\partial S}{\partial z_j} = e^{z_j}$.

$$\begin{aligned}\frac{\partial \hat{y}_k}{\partial z_j} = \frac{\partial}{\partial z_j}\left(\frac{e^{z_k}}{S}\right) &= \frac{e^{z_k}\,\delta_{kj}\,S - e^{z_k}\,\frac{\partial S}{\partial z_j}}{S^2} = \frac{e^{z_k}\,\delta_{kj}\,S - e^{z_k}\,e^{z_j}}{S^2} \\ &= \frac{e^{z_k}}{S}\left(\delta_{kj} - \frac{e^{z_j}}{S}\right) = \hat{y}_k(\delta_{kj} - \hat{y}_j),\end{aligned}$$

where

$$\delta_{kj} = \begin{cases} 1, & k = j, \\ 0, & k \neq j \end{cases} \qquad \text{(Kronecker delta).}$$

Thus

$$\frac{\partial \hat{y}_k^{(n)}}{\partial z_j^{(n)}} = \hat{y}_k^{(n)}(\delta_{kj} - \hat{y}_j^{(n)}).$$

It can be useful to think of this equation in vector form. When is $\delta_{kj} = 1$? It is exactly for the diagnonal elements of the matrix $\hat{y}$

$$\frac{\partial \hat{y}}{\partial z} = \mathrm{Diag}(\hat{y}) - \hat{y}\,\hat{y}^\top \in \mathbb{R}^{K\times K}.$$

No we plug this into the original function

$$\frac{\partial \ell}{\partial z_j^{(n)}} = -\sum_{k=1}^{K} \frac{y_k^{(n)}}{\hat{y}_k^{(n)}}\,\hat{y}_k^{(n)}(\delta_{kj} - \hat{y}_j^{(n)}) = -y_j^{(n)} + \left(\sum_{k=1}^{K} y_k^{(n)}\right)\hat{y}_j^{(n)} = \hat{y}_j^{(n)} - y_j^{(n)},$$

because $\sum_k y_k^{(n)} = 1$ for one–hot labels (all entries but one are zero). Again thinking of the vector form,

$$\nabla_{z^{(n)}}\ell = \hat{y}^{(n)} - y^{(n)} \in \mathbb{R}^K.$$

We have arrived at the leaf of our chain rule tree! At least we derive the expression with respect to W.

$$\frac{\partial \ell}{\partial W} = \sum_{k=1}^{K} \frac{\partial \ell}{\partial \hat{y}_k^{(n)}} \frac{\partial \hat{y}_k^{(n)}}{\partial z^{(n)}} \frac{\partial z^{(n)}}{\partial W}$$

Since $z^{(n)} = Wx^{(n)}$, we have

$$\frac{\partial z^{(n)}}{\partial W} = (\cdot)\,x^{(n)\top}, \quad \text{and it follows that} \quad \nabla_W \ell\big(\hat{y}^{(n)}, y^{(n)}\big) = \big(\hat{y}^{(n)} - y^{(n)}\big)\,x^{(n)\top}.$$

Linearity of the gradient yields

$$\nabla_W R_N(W) = \frac{1}{N} \sum_{n=1}^{N} \left(\hat{y}^{(n)} - y^{(n)}\right) x^{(n)\top}.$$

Exercise 7 (Translation invariance - ⋆⋆)**.**

Let $T_{a,b}$ be an operator which translates an image by a rows and b columns,

$$(T_{a,b}x)[m,n] = x[m-a, n-b].$$

Let also H be a convolutional operator so that

$$(Hx)[n] = (x * h)[n]$$

for some h. Show that

$$T_{a,b}H = HT_{a,b}.$$

Show further that $T_{a,b}$ is itself a convolution with some filter ("impulse response")—which one?

The translation operator shifts the signal (or image) $x[m,n]$ by (a,b):

$$(T_{a,b}x)[m,n] \;=\; x[m-a, n-b].$$

The convolutional operator H is defined by convolution with some kernel h:

$$(Hx)[m,n] \;=\; (x * h)[m,n] \;=\; \sum_{k,\ell} x[k,\ell]\, h[m-k, n-\ell].$$

Now apply the translation operator:

$$(T_{a,b}Hx)[m,n] = (Hx)[m-a, n-b] = \sum_{k,\ell} x[k,\ell]\, h[(m-a)-k, (n-b)-\ell].$$

From

$$(T_{a,b}x)[k,\ell] = x[k-a, \ell-b].$$

we get by applying H:

$$\begin{aligned}(HT_{a,b}x)[m,n] &= \sum_{k,\ell} (T_{a,b}x)[k,\ell]\, h[m-k, n-\ell] \\ &= \sum_{k,\ell} x[k-a, \ell-b]\, h[m-k, n-\ell] \\ &= \sum_{k',\ell'} x[k',\ell']\, h[(m-a)-k', (n-b)-\ell'] \\ &= (T_{a,b}Hx)[m,n]\end{aligned}$$

In the last step we changed variables $k' = k-a$, $\ell' = \ell - b$.
Now we just need to show that is $T_{a,b}$ itself a convolution with a shifted delta kernel: Consider the kernel

$$h[m,n] \;=\; \delta[m-a,\, n-b],$$

where δ denotes the *Kronecker delta function*, defined by

$$\delta[p,q] \;=\; \begin{cases} 1, & \text{if } p=0 \text{ and } q=0, \\ 0, & \text{otherwise.} \end{cases}$$

Then

$$(x * h)[m, n] \;=\; \sum_{k,\ell} x[k, \ell]\, \delta[m - k - a,\; n - \ell - b].$$

The Kronecker delta acts as a "selector": it is nonzero only when

$$m - k - a = 0 \quad \text{and} \quad n - \ell - b = 0,$$

which is equivalent to

$$k = m - a, \quad \ell = n - b.$$

This means only one term in the sum "survives" and the rest of the terms of the sum are zero because the delta function is zero. The surviving term is:

$$(x * h)[m, n] \;=\; x[m - a,\; n - b].$$

This is exactly the definition of the translation operator:

$$(T_{a,b}x)[m, n] \;=\; x[m - a,\; n - b].$$

Exercise 8 (Neural network and backpropagation - $\star$)**.**

Consider the one-hidden-layer neural network defined by the function:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{W}_2\sigma(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2, \tag{4}$$

where $\boldsymbol{\theta} = \{\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{b}_1, \boldsymbol{b}_2\}$ represents the set of trainable parameters, and $\sigma(\cdot)$ is a element-wise non-linear activation function. These parameters are optimized using gradient descent update,

$$\boldsymbol{\theta}(t + 1) = \boldsymbol{\theta}(t) - \gamma\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}(t)),$$

applied to a dataset $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ with the loss function.

$$L(\boldsymbol{\theta}) = \frac{1}{2N}\sum_{i=1}^{N}(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i)^2.$$

- In a deep (more than one layer) neural network, we sometimes call $\boldsymbol{h}_1 = \sigma(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1)$ as hidden (or intermediate) layer output, or hidden features. Given that input is $\boldsymbol{x} \in \mathbb{R}^K$ and the label is a scalar $y \in \mathbb{R}$, and assuming the hidden layer dimension is F, what are the dimensions of trainable parameters $\boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{W}_1, \boldsymbol{W}_2$?

  We have an input vector $x \in \mathbb{R}^K$ and a hidden layer of dimension F. The network is

  $$f_\theta(x) = W_2\,\sigma(W_1 x + b_1) + b_2,$$

  where

  - $W_1 \in \mathbb{R}^{F\times K}$ maps the input to the hidden layer,
  - $b_1 \in \mathbb{R}^F$ is the hidden bias,
  - $\sigma(\cdot)$ is applied elementwise to $\mathbb{R}^F$,
  - $W_2 \in \mathbb{R}^{1\times F}$ maps hidden activations to the scalar output,
  - $b_2 \in \mathbb{R}$ is the output bias.

- For a single training datum $\boldsymbol{x}_1 = [1, 1]^\intercal$ and $y_1 = 1$, calculate the updated parameters $\boldsymbol{\theta}(2) = \{\boldsymbol{W}_1(2), \boldsymbol{W}_2(2), \boldsymbol{b}_1(2), \boldsymbol{b}_2(2)\}$ after the second step of gradient descent iteration. Use the initial values:

  $$\boldsymbol{W}_1(0) = [1, 1], \quad \boldsymbol{W}_2(0) = \boldsymbol{b}_1(0) = \boldsymbol{b}_2(0) = 0,$$

  with the activation function $\sigma(x) = \frac{1}{1+e^{-x}}$ and learning rate as $\gamma = 0.1$.

We define the network output:

$$f_\theta(x) = W_2\,\sigma(W_1 x + b_1) + b_2.$$

The update rule is

$$\theta(t+1) = \theta(t) - \gamma \nabla_\theta L(\theta(t)).$$

The loss is

$$L(\theta) = \frac{1}{2N}\sum_{i=1}^{N}\left(f_\theta(x_i) - y_i\right)^2.$$

For our data,

$$x_1 = \begin{bmatrix}1\\1\end{bmatrix}, \quad y_1 = 1,$$

$$W_1(0) = [1,1], \quad W_2(0) = b_1(0) = b_2(0) = 0.$$

For $(t=0)$:

$$f_\theta(x) = W_2(0)\cdot\sigma(W_1(0)x + b_1(0)) + b_2(0) = 0,$$

Update rule:

$$\theta(t+1) = \theta(t) - 0.1\,\nabla_\theta L(\theta(t)).$$

The update rules takes into accound the gradient $\nabla_\theta L(\theta(t))$ with respect to the parameters $\theta = (W_1, W_2, b_1, b_2)$:

$$\nabla_\theta L(\theta(t)) = \begin{pmatrix} \frac{\partial L(\theta)}{\partial W_1(t)} \\ \frac{\partial}{\partial W_2(t)} \\ \frac{\partial}{\partial b_1(t)} \\ \frac{\partial}{\partial b_2(t)} \end{pmatrix}$$

So more explicity, the update rule $\theta(t+1) = \theta(t) - \gamma\,\nabla_\theta L(\theta(t))$ can be written as:

$$\begin{pmatrix} W_1(t+1) \\ W_2(t+1) \\ b_1(t+1) \\ b_2(t+1) \end{pmatrix} = \begin{pmatrix} W_1(t) \\ W_2(t) \\ b_1(t) \\ b_2(t) \end{pmatrix} - \gamma\cdot \begin{pmatrix} \frac{\partial L(\theta)}{\partial W_1(t)} \\ \frac{\partial L(\theta)}{\partial W_2(t)} \\ \frac{\partial L(\theta)}{\partial b_1(t)} \\ \frac{\partial L(\theta)}{\partial b_2(t)} \end{pmatrix}$$

This means in order to calculate the new weight for the parameters, we need to calculate the partial derivative of the loss function with respect of the parameter. Let's do this for each parameter:

$$\begin{aligned}
\frac{\partial L(\theta)}{\partial W_1} &= \frac{\partial}{\partial W_1}\tfrac{1}{2}\left(f_\theta(x) - 1\right)^2 \\
&= 2\tfrac{1}{2}\left(f_\theta(x) - 1\right)\frac{\partial}{\partial W_1} f_\theta(x) \\
&= (f_\theta(x) - 1)\frac{\partial}{\partial W_1} W_2\cdot\sigma(W_1 x + b_1) + b_2 \\
&= (f_\theta(x) - 1)W_2\cdot\sigma(W_1 x + b_1)\frac{\partial}{\partial W_1}\sigma(W_1 x + b_1) \\
&= (f_\theta(x) - 1)W_2\cdot\sigma(W_1 x + b_1)(1 - \sigma(W_1 x + b_1))x^T
\end{aligned}$$

$$\begin{aligned}
\frac{\partial L(\theta)}{\partial W_2} &= \frac{\partial}{\partial W_2}\tfrac{1}{2}\left(f_\theta(x)-1\right)^2 \\
&= 2\tfrac{1}{2}\left(f_\theta(x)-1\right)\frac{\partial}{\partial W_2}f_\theta(x) \\
&= \left(f_\theta(x)-1\right)\frac{\partial}{\partial W_2}W_2\cdot\sigma(W_1x+b_1)+b_2 \\
&= \left(f_\theta(x)-1\right)\cdot\sigma(W_1x+b_1) \\
\frac{\partial L(\theta)}{\partial b_1} &= \frac{\partial}{\partial b_1}\tfrac{1}{2}\left(f_\theta(x)-1\right)^2 \\
&= 2\tfrac{1}{2}\left(f_\theta(x)-1\right)\frac{\partial}{\partial b_1}f_\theta(x) \\
&= \left(f_\theta(x)-1\right)\frac{\partial}{\partial b_1}W_2\cdot\sigma(W_1x+b_1)+b_2 \\
&= \left(f_\theta(x)-1\right)W_2\cdot\sigma(W_1x+b_1)\frac{\partial}{\partial b_1}\sigma(W_1x+b_1) \\
&= \left(f_\theta(x)-1\right)W_2\cdot\sigma(W_1x+b_1)(1-\sigma(W_1x+b_1)) \\
\frac{\partial L(\theta)}{\partial b_2} &= \frac{\partial}{\partial b_2}\tfrac{1}{2}\left(f_\theta(x)-1\right)^2 \\
&= 2\tfrac{1}{2}\left(f_\theta(x)-1\right)\frac{\partial}{\partial b_2}f_\theta(x) \\
&= \left(f_\theta(x)-1\right)\frac{\partial}{\partial b_2}W_2\cdot\sigma(W_1x+b_1)+b_2 \\
&= \left(f_\theta(x)-1\right)\cdot 1
\end{aligned}$$

Now we just need to plug in the given values $\theta(0)=(W_1(0),W_2(0),b_1(0),b_2(0))=([1,1],0,0,0)$ into the weight update formula.

$$\begin{aligned}
W_1(1) &= W_1(0)-\gamma\frac{\partial L(\theta)}{\partial W_1(0)} \\
&= W_1(0)-\gamma(f_\theta(x)-1)W_2(0)\cdot\sigma(W_1(0)x+b_1(0))(1-\sigma(W_1(0)x+b_1(0)))x^T \\
&= W_1(0) \\
&= [1,1] \\
W_2(1) &= W_2(0)-\gamma\frac{\partial L(\theta)}{\partial W_2(0)} \\
&= W_2(0)-\gamma(f_\theta(x)-1)\sigma(W_1x+b_1) \\
&= 0-0.1(0-1)\sigma([1,1]\cdot[1,1]^T+0) \\
&= -0.1(-1)\sigma(2) \\
&= \frac{0.1}{1+e^{-2}} \\
b_1(1) &= b_1(0)-\gamma\frac{\partial L(\theta)}{\partial b_1(0)} \\
&= b_1(0)-\gamma(f_\theta(x)-1)W_2(0)\cdot\sigma(W_1(0)x+b_1(0))(1-\sigma(W_1(0)x+b_1(0))) \\
&= 0 \\
b_2(1) &= b_2(0)-\gamma\frac{\partial L(\theta)}{\partial b_2(0)} \\
&= b_2(0)-\gamma(f_\theta(x)-1) \\
&= 0-0.1(0-1) \\
&= 0.1
\end{aligned}$$

Note that for $\theta(1)$, $f_\theta(x) = W_2(1) \cdot \sigma(W_1(1)x + b_1(1) + b_2(1) = 0.1 \cdot \frac{0.1}{1+e^{-2}} \frac{1}{1+e^{-2}} + 0.1$. The loss is given by:

$$\begin{aligned} L(\theta) &= \tfrac{1}{2} \left(f_\theta(x) - 1\right)^2 \\ &= \tfrac{1}{2} \Big(W_2 \cdot \sigma(W_1 x + b_1) + b_2 - 1\Big)^2 \\ &= \tfrac{1}{2} \left(0.1 \cdot \left(\frac{1}{1+e^{-2}}\right)^2 + 0.1 - 1\right)^2 \end{aligned}$$

Now we essentially repeat the update step we did before, but with the values $\theta(1) = (W_1(1), W_2(1), b_1(1), b_2(1))$:

$$\begin{aligned} W_1(2) &= W_1(1) - \gamma \frac{\partial L(\theta)}{\partial W_1(1)} \\ &= [1, 1] - 0.1 \cdot \tfrac{1}{2} \cdot 2 \left(f_\theta(x) - 1\right) \cdot 0.1 \cdot \frac{1}{1+e^{-2}} \cdot \frac{1}{1-e^{2}} \cdot \left(1 - \frac{1}{1-e^{2}}\right) [1, 1] \\ &= [1.008,\ 1.008] \\ W_2(2) &= W_2(1) - \gamma \frac{\partial L(\theta)}{\partial W_2(1)} \\ &= \frac{0.1}{1+e^{-2}} - 0.1 \cdot \tfrac{1}{2} \cdot 2 \left(f_\theta(x) - 1\right) \cdot \frac{1}{1+e^{-2}} \\ &= 0.1605 \\ b_1(2) &= b_1(1) - \gamma \frac{\partial L(\theta)}{\partial b_1(1)} \\ &= 0 - 0.1 \cdot \tfrac{1}{2} \cdot 2 \left(f_\theta(x) - 1\right) \cdot 0.1 \left(\frac{1}{1+e^{-2}}\right)^2 \left(1 - \frac{1}{1+e^{-2}}\right) \\ &= 7.6056 \cdot 10^{-4} \\ b_2(2) &= b_2(1) - \gamma \frac{\partial L(\theta)}{\partial b_2(1)} \\ &= 0.1 - 0.1 \left(f_\theta(x) - 1\right) \cdot 1 \\ &= 0.1822. \end{aligned}$$

Coding

Instruction for optional coding submission We use Gradescope to give you a platform where your coding solutions are automatically evaluated. A reminder: Gradescope is optional and has no impact on your final grade–it is simply a tool to help you check whether your code works correctly.
You should have received an invitation to the Gradescope course page. Please go to the course page `10907 Pattern Recognition 2025`, where you will see the corresponding item `Coding: Problem set 2`. To check your solution, upload your completed files `xxx.py` and `xxx.py` to Gradescope and let the autograder run.

Exercise 9 (Logistic Regression - ⋆⋆).

Logistic regression is a simple and efficient classifier for two-class problems. First, you will complete the functions that are core to the logistic regression classifier and then apply the classifier to a simple dataset.

1. Implement six functions that form the logistic regression classifier in the file `logistic.py`. The functions to be completed are:

   (a) `sigmoid()`: This takes an array as input and gives the sigmoid of each value of the array. The sigmoid function for a point $z \in \mathbb{R}$ is defined as :

   $$\text{sigmoid}(z) = \frac{1}{1+e^{-z}}$$

   (b) `predict_score()`: Takes the weight vector $w \in \mathbb{R}^D$ of size D and a matrix $X \in \mathbb{R}^{N \times D}$, with N data points of size D, and outputs the logistic regression probability (score). For a single data point $x \in \mathbb{R}^D$, the score is defined as:

   $$\text{score}(x, w) = \frac{1}{1+e^{-w^T x}}$$

   Note: The above definition is for a single example. However, the function takes multiple examples as input in the form of a matrix. You can do this without using a `for` loop.

   (c) `predict()`: Predicts the class 0 or 1 using the data matrix X, weight vector w, and a threshold. This is defined as:

   $$\text{predict}(x, w, \text{threshold}) = \mathbb{1}_{\text{score}(x,w) \geq \text{threshold}}.$$

   (d) `cross_entropy_loss()`: Outputs the cross-entropy loss (CE) given the predicted probabilities and the true labels. This is defined as:

   $$CE(y_{\text{pred}}, y) = -\frac{1}{N}\left(\sum_{i=1}^{N} y[i]\log(y_{\text{pred}}[i]) + (1-y[i])\log\left(1-y_{\text{pred}}[i]\right)\right).$$

   Note: We use the natural logarithm.

   (e) `gradient()`: Computes the gradient of the cross-entropy loss with respect to w and returns it. This is defined as:

   $$\begin{aligned} y_{\text{pred}} &= \texttt{predict\_score}(X, w) \\ \texttt{gradient}(X, y) &= \frac{\partial}{\partial w} CE(y_{pred}, y) \end{aligned}$$

   (f) `train()`: Using an initial weight vector w_{init}, learn a new estimate of the weight vector using gradient descent for a predefined number of iterations (`epochs`) and a given learning rate `lr`. The pseudocode for the training algorithm is given in Algorithm 1.

Algorithm 1 Gradient Descent

```
Require: $X \in \mathbb{R}^{N \times D}$, $w\_init \in \mathbb{R}^D$, $y\_true \in \{0,1\}^N$, $epochs \geq 0$, $lr \geq 0$
  $w \leftarrow w\_init$
  $losses = [\ ]$
  $i \leftarrow 0$
  while $i <$ epochs do
      $i \leftarrow i + 1$
      $y\_pred \leftarrow predict\_score(X, w)$
      $loss \leftarrow cross\_entropy\_loss(y\_pred, y\_true)$
      $losses.append(loss)$
      $w \leftarrow w - lr * gradient(X, w, y\_true)$
  end while
```

2. Once implemented, you will apply the classifier to the rice dataset provided in the file `Rice.csv`. The dataset consists of various physical features of two types of grains known as 'Cammeo' and 'Osmancik'. The features include: Area, Perimeter, Major Axis Length, Minor Axis Length, Eccentricity, Convex Area, and Extent observed for different grains of rice from the two species. More details about the data is present in [?].

   We have provided you with a skeleton file called `classifier.py` with all the necessary libraries loaded. You will first preprocess the dataset, apply the classifier, and report any preprocessing you do and results (ROC curve and AUC). Your task is divided as follows:

   (a) Preprocess: Once the data is loaded, you need to convert the target variable `y` in to binary variable. You can choose either of the two species as class 1. If required, you can preprocess the dataset provided by the variable `X`. Since there are integers and real numbers, some of the preprocessing methods include:

      i. Standardization: makes the numerical feature zero mean and unit variance.
      ii. Min-Max Scaling: scales features to a specified range (e.g. [0, 1]).
      iii. Log Transformation: can be used for features with skewed distributions to make them more normally distributed.

      You can choose one of the above or anything else you can think of and apply it to the dataset.

   (b) Split the data into training and test sets using the `train_test_split` function from the sklearn library.

   (c) Train the classifier on the training set and evaluate it on the test set (i.e., compute test accuracy and plot the loss over epochs). Experiment with different learning rates and numbers of epochs. *Note:* A loss curve plots the number of epochs on the x-axis and the loss value on the y-axis.

Exercise 10 (DL warm up - $\star$).

Develop one-hidden-layer neural networks as described in Exercise 8 with $\sigma(\cdot) = \text{ReLU}(\cdot)$. Train this model using the following dataset, which consists of 9 data points sampled from the quadratic function $y = x^2$:

x	-1.2	-0.9	-0.6	-0.3	0	0.3	0.6	0.9	1.2
y	1.44	0.81	0.36	0.09	0.	0.09	0.36	0.81	1.44

Table 1: Training dataset sampled from $y = x^2$

Illustrate the performance of your trained network by plotting the results for $x \in [-1.5, 1.5]$. Compare the results with different hidden layer dimensions: $F = 2, 5, 10, 50, 100,$ and 500. Be aware that the loss may not always diminish to zero.

Your plots cannot be autograded by Gradescope. To help you calibrate your results, we provide two example plots at the end of the sheet for hidden sizes $F = 2$ and $F = 500$. Use these as qualitative references to compare the shape of your fitted curves and the overall trend as F increases. Small numerical or visual differences are expected due to initialization and optimization choices. What matters is that your curves exhibit similar behavior (e.g., limited capacity and visible underfitting for $F = 2$, much higher flexibility and closer fit for $F = 500$).

Exercise 11 (Convolution and Filtering - ⋆⋆).

Convolution and filtering are fundamental operations when working with signals or images. We have provided you with a skeleton code in the file `filter.py` to perform basic 2D convolutions and some simple Gaussian low-pass and high-pass filtering.

1. Convolutions can be implemented in two ways, either in the Fourier domain or the image domain. In both cases, the function takes an image of size $K \times K$ and a filter of size $k \times k$ and `mode` as input. If `mode = 'same'` the output has the same dimension as the input image. In this case, we want to compute linear rather than circular convolution, so you will have to zero-pad the input image accordingly and then crop the result to the target dimensions. If `mode='valid'`, then you should only return that part of the output image that does not depend on zero padding. Thus the output is of size $(K-k+1) \times (K-k+1)$. Your task is to implement:
   (a) `convolve2d()`: performs convolution directly in the image domain, using for loops.
   (b) `convolve2d_fft()`: performs convolution in the Fourier domain.
   Note:
      i. Use NumPy's FFT routine.
      ii. While performing convolution in the Fourier domain, you need to first perform the filtering and then crop the result.

   (Optional) Compare computation time of `convolve2d` and `convolve2d_fft` on a large image of your choice.

2. A Gaussian low-pass filter retains only smooth features of the image while removing high-frequency, potentially noisy components. For a given standard deviation η, the filter is of spatial dimension $(2m+1) \times (2m+1)$ where $m = \lceil 4\eta \rceil$. The weights of the filter are defined as:
$$W^{\text{low-pass}}[n_1, n_2] = ce^{-(n_1^2+n_2^2)/(2\eta^2)},$$
   where $n_1 \in \{-m, -m+1, \ldots, 0, \ldots, m\}$, $n_2 \in \{-m, -m+1, \ldots, 0, \ldots, m\}$ and c is chosen such that $\sum_{n_1=-m}^{m} \sum_{n_2=-m}^{m} W[n_1, n_2] = 1$. The high-pass filter is just the complement of the low-pass, $W^{\text{high-pass}}[n_1, n_2] = \delta[n_1, n_2] - W^{\text{low-pass}}[n_1, n_2]$, which allows only the high-frequency components of the image. The function $\delta[n_1, n_2]$ is an identity filter defined as
$$\delta[n_1, n_2] = \begin{cases} 1 & \text{if } n_1 = 0 \text{ and } n_2 = 0 \\ 0 & \text{otherwise.} \end{cases}$$
   Applying this filter doesn't change the image. Your task is to implement:
   (a) `gaussian_lowpass()`: Which takes the standard deviation η as input and outputs a Gaussian low-pass filter of size $(2m + 1) \times (2m + 1)$ where $m = \lceil 4\eta \rceil$.
   (b) `gaussian_highpass()`: Outputs a Gaussian high-pass filter.
   (c) Using any one of the images in the scikit-image dataset (link), show the effect of a high-pass and low-pass filter by plotting the chosen image along with the low-pass filtered and high-pass filtered images side by side. Report the variance and filter size used. Report the convolution operation used and what is the computational complexity of this convolution?
   Note: For a color image, make sure to convert it into a grayscale image first.

University of Basel

Reference plots for Exercise 8

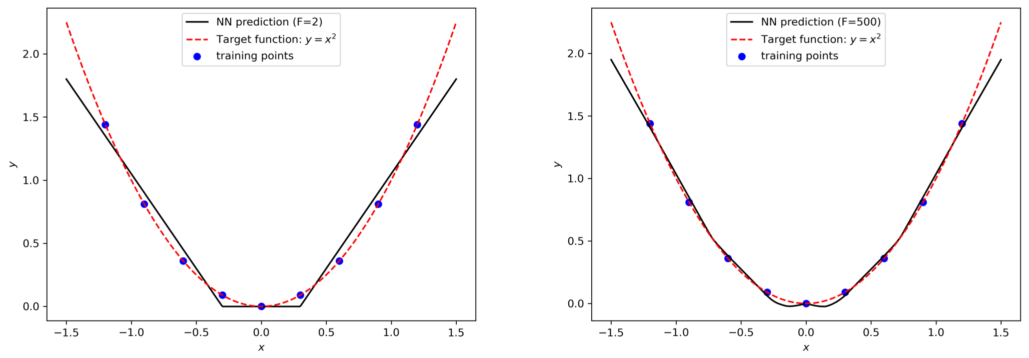

Figure 1: Sample plots showing the results of the one-hidden-layer neural network with $F = 2$ (left) and $F = 500$ (right).