Homework #7, EECS 553, F26. Due **Wed. Nov. 11**, by 11:00PM

Notes

- This is the last HW that is relevant to Exam2 on Nov. 16.

Problems

1. [10] **Alternative to optimal soft-margin hyperplane**

Recall that the **optimal soft-margin hyperplane** for supervised binary classification with labels $y_i = \pm 1$ and features $\boldsymbol{x}_i \in \mathbb{R}^d$ is found by solving the quadratic program with slack variables:

$$\begin{aligned} \underset{\boldsymbol{w}, b, \boldsymbol{\xi}}{\arg\min} \; & \frac{1}{2} \|\boldsymbol{w}\|_2^2 + \frac{C}{n} \sum_{i=1}^n \xi_i \\ \text{s.t. } & y_i \left(\boldsymbol{w}^\top \boldsymbol{x}_i + b \right) \geq 1 - \xi_i, \; \forall i \\ & \xi_i \geq 0, \; \forall i. \end{aligned}$$

Prof. Scott's course notes (`06_erm.pdf` on Canvas) show that the above constrained optimization problem is equivalent to the following *unconstrained* ERM-type optimization problem:

$$\underset{\boldsymbol{w}, b}{\arg\min} \frac{1}{n} \sum_{i=1}^n h\big(y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)\big) + \frac{\lambda}{2} \|\boldsymbol{w}\|_2^2, \quad \lambda = \frac{1}{C}, \quad h(z) = \max(0, 1 - z).$$

The function h is called the **hinge loss**. This formulation is the ERM for a surrogate loss of the form $L(y, \tilde{f}) = h(y\tilde{f})$.

(a) [3] An alternative way to introduce slack variables into the separating hyperplane problem is the following quadratic program:

$$\begin{aligned} \underset{\boldsymbol{w}, b, \boldsymbol{\xi}}{\arg\min} \; & \frac{1}{2} \|\boldsymbol{w}\|_2^2 + \frac{C}{n} \sum_{i=1}^n \xi_i^2 \\ \text{s.t. } & y_i \left(\boldsymbol{w}^\top \boldsymbol{x}_i + b \right) \geq 1 - \xi_i, \; \forall i \\ & \xi_i \geq 0, \; \forall i. \end{aligned}$$

Find an ERM-style *unconstrained* cost function $J(\boldsymbol{\theta}) = J(b, \boldsymbol{w})$ that is equivalent to this alternative formulation, and describe the surrogate loss.

(b) [3] The answer to (a) is differentiable. In fact it is **Lipschitz smooth** and convex, so we can apply **gradient descent** to it. Determine a range of **step size** values for GD that ensure convergence. Hint. Write everything in terms of $\boldsymbol{\theta} = (b, \boldsymbol{w})$.

(c) [3] Show that the GD algorithm for this problem is **kernelizable**. Hint. Draw inspiration from the solution to a previous HW problem, but use $\|\boldsymbol{w}\|_2$ instead of $\|\boldsymbol{\theta}\|_2$ in the regularizer, and initialize with $\boldsymbol{w}_0 = \boldsymbol{X}\boldsymbol{\gamma}_0$ for some $\boldsymbol{\gamma}_0 \in \mathbb{R}^n$.

(d) [1] Is the resulting kernelized method in (c) **parametric** or **nonparametric**? Explain.

(e) [0] Optional. Extend (c) to Nesterov's fast gradient method (FGM) and apply your kernelized FGM to the data in the SVM demo and examine the decision boundaries and classification errors.

2. [13] **Kernel logistic regression: practical implementation with real data**

In a previous HW problem, you derived a kernelized version of logistic regression using Newton's method. In this problem you will implement it and apply it to handwritten digit recognition.

(a) [10] Write JULIA or Python code to implement the kernelized Newton's method. Here we continue to use the simplified version where we absorb the constant 1 as the first element of each training vector $\boldsymbol{x}_i$. Your code should return the posterior probability $\eta_1(\boldsymbol{x}) = \mathsf{P}\{\mathsf{Y} = 1 \mid \mathbf{X} = \boldsymbol{x}\}$ for each test data vector.
Your submitted file should be named `klr_newton.jl` or `klr_newton.py` and should contain the following function:

```
"""
    eta = klr_newton(X, y, Xtest, λ; gamma, niter)

Kernel logistic regression binary classifier
```

```
based on kernelizing Newton's method for the ERM cost function
``J(θ) = (1/n) \\sum_{i=1}^n y_i log σ(⟨θ, x_i⟩) + (1-y_i) log(1 - σ(⟨θ, x_i⟩)) + (λ/2) ‖ θ ‖₂²``
where `θ = [b; w]`
and ``σ(t) = 1 / (1 + exp(-t))``.

Run kernelized version of Newton's method for `niter` iterations,
then use the resulting model parameters
to classify `Xtest`.

In:
- `X` : `(d+1) x n` matrix of training data (including "1" in first row!)
- `y` : `n`-vector of training labels (0,1)
- `Xtest` : `(d+1) x m` matrix of test data (including "1" in first row!)
- `lambda` : regularization parameter λ ≥ 0

Option:
- `gamma` : hyperparameter for gaussian kernel exp(-γ ‖x - x'‖₂²), γ ≥ 0;
  default 1/d
- `niter` number of iterations; default 100
- `alpha0` initial α vector; default zeros(size(X,2))

Out:
- `eta` : `m`-vector of probability that label = 1 for test data.
"""
function klr_newton(
    X::AbstractMatrix,
    y::AbstractVector,
    Xtest::AbstractMatrix,
    λ::Real;
    gamma::Real = 1 / (size(X,1)-1), # 1 / d
    niter::Int = 100,
    alpha0::AbstractVector = zeros(size(X,2)),
)
```

Submit your solution to autograder.io
Python version:

```
def klr_newton(
    X: np.ndarray,
    y: np.ndarray,
    Xtest: np.ndarray,
    lam: float,
    gamma: float = None,
    niter: int = 100,
    alpha0: np.ndarray = None,
) -> np.ndarray:
```

(b) [3] Apply your code to the 1,5 digit data in this JULIA SVM demo, or modify this Python demo to use digits 1,5. You may find it helpful to standardize the data (*e.g.*, dividing by the training feature standard deviations).

- ☐ Submit a histogram plot of the posterior probabilities similar to the one in this SVM demo.
- ☐ You have 3 hyperparameter values to select here: the regularization parameter λ, the kernel parameter γ, and the number of iterations of Newton's method to run. You may use any hyperparameter selection method; in your submission, report which method you chose and why. With suitable tuning, you should be able to get test classification errors that are at least as good as the SVM classifier results in the demo.
- ☐ A label or title in your histogram plot must show the values of λ and γ that you used, along with the test error.

(c) [0] Optional. Make a plot of the decision regions like the plot in the demo.

3. [11] **Lipschitz continuity in neural networks**

(a) [3] Let f and g denote Lipschitz continuous functions with Lipschitz constants $\mathcal{L}_f$ and $\mathcal{L}_g$ respectively. Show that $h = f \circ g$ is Lipschitz continuous and find a Lipschitz constant $\mathcal{L}_h$, where $\circ$ denotes **function composition**. (This property is important because NN models use function composition.)

(b) [3] Determine Lipschitz constants for these functions:

☐ The ReLU function $\max(x, 0)$, used in many NN models.

☐ The $\tanh$ function often used in early NN papers.

☐ Optional: The softmax function used in many classification networks. (You may use GenAI to help with this one.)

(c) [3] Consider a feedforward network model $f(\boldsymbol{x}) = \sigma_L(\boldsymbol{W}^{(L)}\sigma_{L-1}(\boldsymbol{W}^{(L-1)} \cdots \sigma_1(\boldsymbol{W}^{(1)}\boldsymbol{x})))$, where each σ_l is applied element-wise.
For the case where each σ_l is Lipschitz continuous with Lipschitz constant $\mathcal{L}_{\sigma_l}$, show that $f(\boldsymbol{x})$ is Lipschitz continuous and find a Lipschitz constant for $f(\boldsymbol{x})$.

(d) [2] Suppose we want to apply gradient descent to train the parameters $\boldsymbol{\theta}$ of this NN model with ReLU activation functions for a regression problem with ERM based on the quadratic loss with a Tikhonov regularizer. Discuss whether it is possible to find a range of (constant) step sizes so that GD that will ensure convergence of the sequence of iterates $\{\boldsymbol{\theta}_t\}$.

4. [25] **Bayesian optimization with Gaussian Process models**

This problem explores **Gaussian Process** (GP) models for finding the minimizer of a black box function (meaning we can only evaluate it, we cannot compute numerical gradients) that is expensive to evaluate. A prime example is the cross-validation risk estimate when doing model / hyperparameter selection for some machine learning algorithms.

Our goal is to find the minimizer of some function $f : \mathbb{R}^p \to \mathbb{R}$ over $\boldsymbol{x} \in \mathbb{R}^p$, assuming we initially only have access to noisy observations of the form $(\boldsymbol{x}_i, y_i)_{i=1}^n$ where

$$y_i = f(\boldsymbol{x}_i) + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma_e^2).$$

Let $\boldsymbol{y} = [y_1, \dots, y_n]^\top$, and $\boldsymbol{f} = [f(x_1), \dots, f(x_n)]^\top$. We assume a **Gaussian Process** (GP) prior on $f(\boldsymbol{x})$, namely

$$\boldsymbol{f} \sim \mathcal{N}(\mu_f \mathbf{1}_n, \boldsymbol{K}),$$

where $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for some inner product kernel k. According to our observation model, where the noise is independent of $\boldsymbol{f}$, the distribution of $\boldsymbol{y}$ conditioned on $\boldsymbol{f}$ is

$$\boldsymbol{y} \mid \boldsymbol{f} \sim \mathcal{N}(\boldsymbol{f}, \sigma_e^2 \boldsymbol{I}),$$

which marginalizes to

$$\boldsymbol{y} \sim \mathcal{N}(\mu_f \mathbf{1}_n, \boldsymbol{K} + \sigma_e^2 \boldsymbol{I}).$$

Given our n observations $\boldsymbol{y}$ so far, we would like to reason about the function value $f(\boldsymbol{x})$ for any test point $\boldsymbol{x}$. From properties of the multivariate Gaussian distribution, the joint distribution of $\boldsymbol{y}$ and $f(\boldsymbol{x})$ is

$$\begin{bmatrix} \boldsymbol{y} \\ f(\boldsymbol{x}) \end{bmatrix} \sim \mathcal{N}\left(\mu_f \mathbf{1}_{n+1}, \begin{bmatrix} \boldsymbol{K} + \sigma_e^2 \boldsymbol{I} & \boldsymbol{k}(\boldsymbol{x}) \\ \boldsymbol{k}(\boldsymbol{x})^\top & k(\boldsymbol{x}, \boldsymbol{x}) \end{bmatrix}\right), \quad \boldsymbol{k}(\boldsymbol{x}) \triangleq \begin{bmatrix} k(\boldsymbol{x}, \boldsymbol{x}_1) \\ \vdots \\ k(\boldsymbol{x}, \boldsymbol{x}_n) \end{bmatrix} \in \mathbb{R}^n.$$

It follows that the posterior distribution is

$$\begin{aligned} f(\boldsymbol{x}) \mid \boldsymbol{y} &\sim \mathcal{N}(\mu(\boldsymbol{x}), \sigma^2(\boldsymbol{x})), \\ \mu(\boldsymbol{x}) \triangleq \mu(f(\boldsymbol{x}) \mid \boldsymbol{y}) &= \mu_f + \boldsymbol{k}(\boldsymbol{x})^\top (\boldsymbol{K} + \sigma_e^2 \boldsymbol{I})^{-1} (\boldsymbol{y} - \mu_f \mathbf{1}_n) \\ \sigma^2(\boldsymbol{x}) \triangleq \sigma^2(f(\boldsymbol{x}) \mid \boldsymbol{y}) &= \boldsymbol{k}(\boldsymbol{x}, \boldsymbol{x}) - \boldsymbol{k}(\boldsymbol{x})^\top (\boldsymbol{K} + \sigma_e^2 \boldsymbol{I})^{-1} \boldsymbol{k}(\boldsymbol{x}). \end{aligned}$$

We use the plug-in estimate $\hat{\mu}_f = \mathbf{1}_n^\top \boldsymbol{y}/n$ for the mean of f.

(a) [10] **Bayesian optimization** uses the posterior mean and variance functions of $\boldsymbol{x}$ to select the next value of $\boldsymbol{x}$ to try. So computing those two functions of $\boldsymbol{x}$ is crucial.

Write JULIA or Python code to compute the posterior functions $\mu(\boldsymbol{x})$ and $\sigma(\boldsymbol{x})$ given $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n$ and $\boldsymbol{y}$, over a grid of candidate points $\boldsymbol{x}$. Use a radial basis function (Gaussian) kernel:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \exp\left(-0.5 \left\|\boldsymbol{x} - \boldsymbol{x}'\right\|_2^2 / \ell^2\right).$$

Hint. Use something like $\max(\sigma, 0)$ to ensure your standard deviations are nonnegative!
Your submitted file should be named `bo_gp.jl` or `bo_gp.py` and should contain the following function:

```
"""
    out = bo_gp(xpast, y, xgrid; ℓ = 0.05, σf = 0.9, σn = 1e-6)

Compute posterior mean and standard deviation
``μ(f(x)|y)`` and ``σ(f(x)|y)``
for a Gaussian Process with Gaussian kernel
``k(x,x') = σf² \\exp(0.5 * ‖x - x'‖₂² / ℓ²``,
as needed for Bayes optimization
when minimizing a function ``f(x)`` where ``x ∈ ℝᵖ``.

# In
- `xpast` : `p × t` matrix with `t` columns `x₁ ... xₜ`
- `y` : vector of `t` measured values `yᵢ = f(xᵢ) + εᵢ`
- `xgrid` : `p × m` array of `x ∈ ℝᵖ` values
   to evaluate the posterior mean and standard deviation.

# Option
- `ℓ` : Gaussian kernel length parameter (default `0.05`)
- `σf` : marginal standard deviation of Gaussian process (default `0.9`)
- `σn` : standard deviation of measurement noise `εᵢ` (default `1e-6`)

# Out
- `out` : `m × 2` matrix; 1st column is posterior mean
  and 2nd column is posterior standard deviation,
  both evaluated at each column of `xgrid`.
"""
function bo_gp(
    xpast::AbstractMatrix{<:Real},
    y::AbstractVector{<:Real},
    xgrid::AbstractMatrix{<:Real};
    ℓ::Real = 0.05,
    σf::Real = 0.9,
    σn::Real = 1e-6,
)
```

Submit your solution to autograder.io
Python version:

```
def bo_gp(
    xpast: np.ndarray,
    y: np.ndarray,
    xgrid: np.ndarray,
    ell: float = 0.05,
    sigma_f: float = 0.9,
    sigma_n: float = 1e-6,
) -> np.ndarray:
```

(b) [3] For the rest of this problem, we will take our function to be minimized to be

$$f(x) = 2 + \sin(2\pi x) + \sin(11\pi x), \quad x \in [0, 1].$$

(In practice you would not know the function.)

Suppose we have obtained the values of this function (without any added noise) at $x \in \{0.2, 0.4, 0.9\}$.
Using the parameter values $\sigma_f = 0.9$, $\ell = 0.05$, and $\sigma_\epsilon = 10^{-6}$ (the defaults in the code), submit a single plot showing the true function, the posterior mean function, the posterior uncertainty envelopes $\mu(\boldsymbol{x}) \pm 2\sigma(\boldsymbol{x})$ (95% confidence bands), and the five points $\{(x_i, y_i)\}$ at which the function value is known. Include a legend. Your plot should look similar to the one in the BO demo (but shading is not required). Use a plot / search grid with spacing $\Delta x = 0.01$ over $[0, 1]$.

(c) [3] The **lower confidence bound** (LCB) approach (to the acquisition function) is to select the point $\boldsymbol{x}$ that minimizes $\mu(\boldsymbol{x}) - c\sigma(\boldsymbol{x})$. where c is a tuning parameter. Using this approach with $c = 2$, determine the next value of $\boldsymbol{x}$ at which to evaluate the function.

(d) [3] The **probability of improvement** approach selects the point $\boldsymbol{x}$ that maximizes the probability that $f(\boldsymbol{x})$ is below $y_{\min}$ – the lowest function value seen so far. Because the distribution of $f(\boldsymbol{x}) \mid \boldsymbol{y}$ is $\mathcal{N}(\mu(\boldsymbol{x}), \sigma^2(\boldsymbol{x}))$, you can calculate the PI in terms of the CDF of this Gaussian distribution. Using the PI approach, determine the next value of $\boldsymbol{x}$ at which to evaluate the function. Hint. Be sure that you select a *new* point, i.e., exclude the already selected points from consideration.

(e) [3] The **expected improvement** approach is similar to PI, except it selects $\boldsymbol{x}$ to maximize $\mathsf{E}[\max(0, y_{\min} - f(\boldsymbol{x})) \mid \boldsymbol{y}]$. Once again you must determine a formula in terms of a Gaussian CDF and/or pdf. (You may use GenAI to help with the integration in this step.) Using the EI approach, determine the next value of $\boldsymbol{x}$ at which to evaluate the function.

(f) [3] Run each of the three approaches to convergence and report the estimated minimizer $x_{\min}$ found by each approach.

---

5. [5] **Make your own problem**

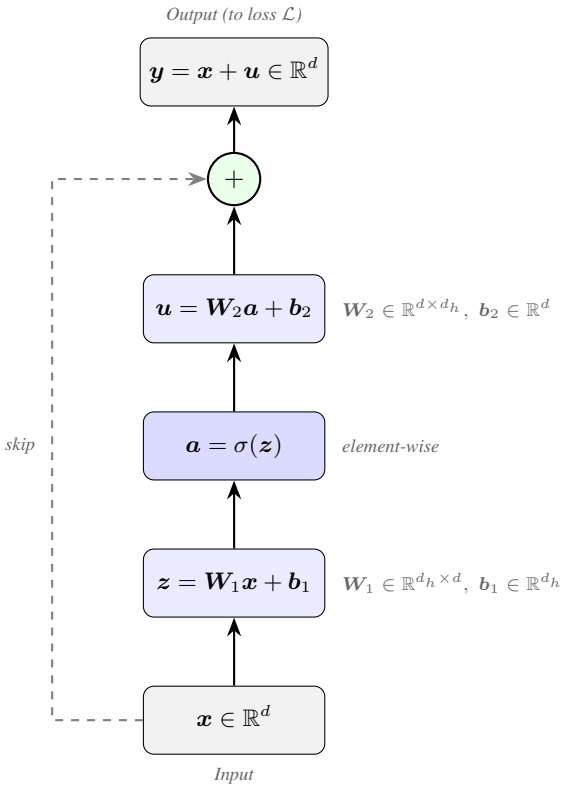
Make a clearly stated problem that could be used on an EECS 553 exam based on the topics covered by the due date of this HW set, and provide your own solution to the problem.

Follow all the instructions from HW2 except create your problem in `hw7.tex` using the Overleaf link on EECS 553 Canvas home page.

Non-graded problems

6. [17] **Backpropagation through a residual MLP block**

In this problem you will derive the gradients needed to backpropagate through a simple **residual block** consisting of two dense (fully connected) layers with an element-wise nonlinearity σ in between, plus a **residual (skip) connection** that adds the input to the output. This is the basic building block of **ResNets** and of the feed-forward part of a transformer layer. The relevant quantities for this problem are summarized below.



Symbol	Description	Size
d	Input / output dimension	scalar
d_h	Hidden layer dimension	scalar
$\boldsymbol{x}$	Input vector	d
$\boldsymbol{W}_1, \boldsymbol{b}_1$	First layer weights and bias	$d_h \times d,\ d_h$
$\boldsymbol{z}$	Pre-activation of hidden layer	d_h
$\boldsymbol{a}$	Hidden layer activation	d_h
$\boldsymbol{W}_2, \boldsymbol{b}_2$	Second layer weights and bias	$d \times d_h,\ d$
$\boldsymbol{u}$	Output of second dense layer	d
$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{u}$	Block output (after residual connection)	d
$\boldsymbol{t}$	Target vector	d

The block maps an input $\boldsymbol{x} \in \mathbb{R}^d$ to an output $\boldsymbol{y} \in \mathbb{R}^d$ as follows:

$$\boldsymbol{z} = \boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1 \in \mathbb{R}^{d_h} \quad \text{(1st dense layer)} \tag{1}$$

$$\boldsymbol{a} = \sigma(\boldsymbol{z}) \in \mathbb{R}^{d_h} \quad \text{(element-wise)} \tag{2}$$

$$\boldsymbol{u} = \boldsymbol{W}_2\boldsymbol{a} + \boldsymbol{b}_2 \in \mathbb{R}^{d} \quad \text{(2nd dense layer)} \tag{3}$$

$$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{u} \in \mathbb{R}^{d} \quad \text{(skip connection).} \tag{4}$$

The residual connection *requires* the output dimension to equal the input dimension. We train with the squared-error loss:

$$\mathcal{L} = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{t}\|_2^2. \tag{5}$$

Conventions. For any vector or matrix $\boldsymbol{Z}$, the notation $\frac{\partial \mathcal{L}}{\partial \boldsymbol{Z}}$ means the vector or matrix of the same size as $\boldsymbol{Z}$ whose (i,j)-entry is $\frac{\partial \mathcal{L}}{\partial Z_{ij}}$. We use the shorthand $\boldsymbol{g}_{\boldsymbol{z}} \equiv \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}}$, etc., and $\odot$ denotes element-wise (Hadamard) multiplication.

Useful identity. For a dense layer $\boldsymbol{v} = \boldsymbol{W}\boldsymbol{p} + \boldsymbol{c}$ where $\boldsymbol{W} \in \mathbb{R}^{m \times n}$, $\boldsymbol{p} \in \mathbb{R}^n$, and $\boldsymbol{c} \in \mathbb{R}^m$, given $\boldsymbol{g}_{\boldsymbol{v}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{v}} \in \mathbb{R}^m$:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}} = \boldsymbol{g}_{\boldsymbol{v}}\,\boldsymbol{p}^\top, \qquad \frac{\partial \mathcal{L}}{\partial \boldsymbol{c}} = \boldsymbol{g}_{\boldsymbol{v}}, \qquad \frac{\partial \mathcal{L}}{\partial \boldsymbol{p}} = \boldsymbol{W}^\top \boldsymbol{g}_{\boldsymbol{v}}. \tag{6}$$

(a) [3] Backpropagation starts at the loss.

(i) Using (5), find $\boldsymbol{g_y} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}} \in \mathbb{R}^d$.

(ii) Using (4), find $\boldsymbol{g_u} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{u}}$ and the contribution $\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}}\big|_{\text{skip}}$ to the gradient with respect to $\boldsymbol{x}$ from the skip (residual) path, in terms of $\boldsymbol{g_y}$.

(b) [3] Starting from (3) and $\boldsymbol{g_u}$, use the identity (6) to derive:

(i) $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_2} \in \mathbb{R}^{d \times d_h}$

(ii) $\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_2} \in \mathbb{R}^d$

(iii) $\frac{\partial \mathcal{L}}{\partial \boldsymbol{a}} \in \mathbb{R}^{d_h}$

(c) [4] The nonlinearity σ is applied element-wise: $a_k = \sigma(z_k)$, $k = 1, \dots, d_h$.

(i) For a general differentiable σ, use the chain rule to show that $\boldsymbol{g_z} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}} = \sigma'(\boldsymbol{z}) \odot \boldsymbol{g_a}$, where $\sigma'(\boldsymbol{z})$ denotes σ' applied element-wise.

(ii) Specialize σ' for: the ReLU $\sigma(z) = \max(0, z)$ (you may ignore the point $z = 0$), the hyperbolic tangent $\sigma(z) = \tanh(z)$, and the logistic sigmoid $\sigma(z) = 1/(1 + e^{-z})$.

(iii) For tanh and the sigmoid, express $\sigma'(\boldsymbol{z})$ in terms of the *activation* $\boldsymbol{a} = \sigma(\boldsymbol{z})$ rather than $\boldsymbol{z}$. Why is this convenient when implementing backpropagation?

(d) [3] Starting from (1) and $\boldsymbol{g_z}$, derive:

(i) $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_1} \in \mathbb{R}^{d_h \times d}$

(ii) $\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_1} \in \mathbb{R}^{d_h}$

(iii) $\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}}\big|_{\text{MLP}} \in \mathbb{R}^d$, the contribution to the gradient with respect to $\boldsymbol{x}$ from the path through the two dense layers.

(e) [4]

(i) The input $\boldsymbol{x}$ affects $\mathcal{L}$ through two paths. Combine your results from parts (a) and (d) to write the **total gradient** $\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}}$ in terms of $\boldsymbol{g_y}$, $\boldsymbol{W}_1$, $\boldsymbol{W}_2$, and $\sigma'(\boldsymbol{z})$. What multivariable calculus rule justifies combining the two paths this way?

(ii) Use your answer to (i) to show that the $d \times d$ Jacobian of the block is

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \boldsymbol{I} + \boldsymbol{W}_2 \, \mathsf{Diag}\{\sigma'(\boldsymbol{z})\} \, \boldsymbol{W}_1.$$

(iii) Suppose the weights are initialized to be very small, so that $\boldsymbol{W}_2 \approx \boldsymbol{0}$. What are $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}}$ in that case? In one or two sentences, explain why this behavior helps gradients flow backward through a deep network built by stacking many such blocks.

---

7. [0] **PCA fundamentals**

The PCA derived in lecture is for an empirical set of data. Here we consider a statistical formulation.

(a) [0] Let $\mathsf{X} \in \mathbb{R}^d$ denote a random vector with mean $\boldsymbol{\mu}_X$ and (positive semidefinite) covariance $\boldsymbol{\Sigma}$. We want to approximate X on an affine subspace, meaning

$$\hat{\mathsf{X}} \approx \boldsymbol{m} + \boldsymbol{Q}\boldsymbol{Q}^\top(\mathsf{X} - \boldsymbol{m})$$

for some $\boldsymbol{m} \in \mathbb{R}^d$ and $\boldsymbol{Q} \in \mathbb{R}^{d \times k}$ is semiunitary: $\boldsymbol{Q}^\top \boldsymbol{Q} = \boldsymbol{I}_k$. Determine the best approximation in the mean squared error (MSE) sense by solving this minimization problem:

$$\underset{\boldsymbol{m} \in \mathbb{R}^d, \, \boldsymbol{Q} \in \mathbb{R}^{d \times k} \, : \, \boldsymbol{Q}^\top \boldsymbol{Q} = \boldsymbol{I}_k}{\arg\min} \mathsf{E}\left[\left\| \left(\boldsymbol{m} + \boldsymbol{Q}\boldsymbol{Q}^\top(\mathsf{X} - \boldsymbol{m}) \right) - \mathsf{X} \right\|_2^2 \right].$$

Specifically, show that a solution is $\boldsymbol{m} = \boldsymbol{\mu}_X$ and $\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{u}_1 & \dots & \boldsymbol{u}_k \end{bmatrix}$ for some vectors $\{\boldsymbol{u}_i\}$.
Hint. Try some subtractions of $\boldsymbol{\mu}_X$ before expanding the squared norm. Also, the equality $\boldsymbol{y}'\boldsymbol{y} = \mathsf{trace}\{\boldsymbol{y}\boldsymbol{y}'\}$ may help.

(b) [0] Find a simple expression for the optimal MSE:

$$\min_{\boldsymbol{m} \in \mathbb{R}^d, \, \boldsymbol{Q} \in \mathbb{R}^{d \times k} \, : \, \boldsymbol{Q}^\top \boldsymbol{Q} = \boldsymbol{I}_k} \mathsf{E}\left[\left\| \left(\boldsymbol{m} + \boldsymbol{Q}\boldsymbol{Q}^\top(\mathsf{X} - \boldsymbol{m}) \right) - \mathsf{X} \right\|_2^2 \right]$$

in terms of properties of $\boldsymbol{\Sigma}$.

(c) Are the solutions to (a) and (b) always unique? Discuss.