Homework 4

CSCI 567 - Machine Learning - Fall 2026

Released Nov 18; due Monday Nov 30, 11:59pm PT, on Gradescope

Logistics. Homework is done in pairs; one submission per pair. Submit two things: (1) a zip file containing your code (all three `.py` files) on **Brightspace**; (2) a single PDF write-up on **Gradescope** with your answers to every question, including, for the programming parts, the printed tables and numbers (screenshots of the output are fine), the plots the code produces, and your written explanations. The write-up must be typed in LaTeX. **Total: 50 pts.**

Collaboration and AI policy. You may discuss problems at a high level with other groups, but every pair must write its own solutions and code; do not search for solutions online, share write-ups across groups, or post course material publicly. Late days: 3 total across the semester, at most 1 per assignment (a late day is charged to each partner). AI usage is not allowed for the HWs.

Setup. Use Python 3 ($\geq$3.9) with **numpy and matplotlib only**. Starter code and the dataset are provided in the `hw4_code` folder. Most of the code is provided: you write seven short functions in total, the ones marked `TODO` (two in `nn.py`, two in `gboost.py`, three in `attention_heads.py`). Do not change seeds, sizes, shrinkage values, round counts, or the reporting code at the bottom of each file. Each file prints a few sanity checks first; get those right before reading the later numbers.

Scope. Problem 1 uses Lecture 07 (neural networks, backpropagation, optimizers). Problems 2 and 3 use Lecture 08 (Part I: boosting; Part II: language modelling and attention). All notation follows the slides. Problem 3's last question connects to Lecture 10.

Problem 1: a neural network from scratch (18 pts - Medium)

You implement backpropagation and Adam for a feedforward network in numpy, then use the network to check four claims from Lecture 07 on two small datasets. Both have inputs in $\mathbb{R}^2$ and labels $y \in \{0, 1\}$, with 400 training and 1000 validation points: **rings** (HW2's concentric rings: label 0 near radius 1, label 1 near radius 2) and **xor** ($\mathbf{x}$ uniform on $[-1, 1]^2$, label 1 iff $x_1 x_2 > 0$). A linear classifier is at chance on both.

Notation. As in Lecture 07, in batch form: a batch is a matrix X_0 with one example per row, and for $l = 1, \ldots, L + 1$

$$Z_l = X_{l-1} W_l + b_l, \qquad X_l = g(Z_l) \;\; (l \leq L), \qquad \text{logits} = Z_{L+1} \in \mathbb{R}^{n \times 1}.$$

The loss is the mean logistic loss of the logits, $\frac{1}{n} \sum_i \big[-y_i \log \sigma(z_i) - (1 - y_i) \log(1 - \sigma(z_i)) \big]$. The starter (`nn.py`) provides the data, He initialization, the forward pass, SGD and momentum, the minibatch training loop, and a finite-difference gradient checker.

1.1 (6 pts) Backpropagation.

(a) **(2 pts)** Show that the output error signal of the mean logistic loss is $\Delta_{L+1} = (\sigma(Z_{L+1}) - \mathbf{y})/n$: a residual, as for the squared loss in lecture.

(b) **(4 pts)** Read the provided `forward`, then implement `backward` for any number of layers and any activation in `ACTS`, with no loop over examples. The recursion from Lecture 07 in batch form is

$$\Delta_l = \left(\Delta_{l+1} W_{l+1}^\top\right) \odot g'(Z_l), \qquad \frac{\partial \ell}{\partial W_l} = X_{l-1}^\top \Delta_l, \qquad \frac{\partial \ell}{\partial b_l} = \text{column sums of } \Delta_l.$$

Report the three gradient-check lines; each relative error must be below 10^{-6}.

1.2 (4 pts) Optimizers. The starter trains a $[2, 16, 16, 1]$ ReLU network on xor for 100 epochs with SGD, SGD with momentum ($\rho = 0.9$), and Adam, each twice: on the loss, and on the loss multiplied by 0.01.

(a) **(2 pts)** `sgd_step` and `momentum_step` are provided. Implement `adam_step` (with bias correction) exactly as on the Lecture 07 slides. Report the six printed lines, and include the figure `optimizers.pdf`.

(b) **(2 pts)** Multiplying the loss by 0.01 multiplies every gradient by 0.01 and does not change the minimizers. Which optimizers are affected? Explain from the update rules why SGD and momentum slow down and why Adam's curve is unchanged.

1.3 (3 pts) Without a nonlinearity, depth buys nothing. The starter trains the same $[2, 16, 16, 1]$ architecture with $g =$ identity, ReLU, and tanh on both datasets (Adam).

(a) **(1 pt)** Report the printed training losses and validation errors.

(b) **(2 pts)** Explain the identity row: what function class does that network represent, and why is its training loss close to $\log 2 \approx 0.693$ on both datasets?

1.4 (3 pts) Vanishing gradients. The starter builds a network with 8 hidden layers of width 16, prints $\|\partial \ell / \partial W_l\|$ for every layer at initialization, and trains it on rings with momentum, once with sigmoid units and once with ReLU.

(a) **(1 pt)** Report the printed lines.

(b) **(2 pts)** For the sigmoid network, by what factor does the gradient norm shrink from layer 9 to layer 1? Compare it with 4^8, the factor suggested by the bound $g' \leq \frac{1}{4}$ from lecture. Does the sigmoid network learn anything in 100 epochs? Does the ReLU network?

1.5 (2 pts) The learned feature map. The starter trains a $[2, 16, 2, 1]$ tanh network on rings and plots the validation points twice (`feature_map.pdf`): in input space, and in the two-dimensional last hidden layer $\mathbf{x}^{(2)}$, with the output layer's decision boundary. Include the figure and the printed validation error. What has the network done to the two rings, and what is the output layer's job after that?

Problem 2: gradient boosting for the logistic loss (12 pts - Medium)

You implement gradient boosting with stumps (Lecture 08) for binary classification and run it on HW2's dataset, with HW2's exact split. The stump itself is provided: you write the gradient and the boosting loop.

Notation. Labels are $y_i \in \{-1, +1\}$ (+1 = malignant). The model is a real-valued score $f(\mathbf{x})$, predicting $\text{sign}(f(\mathbf{x}))$, with loss $\ell(y, f) = \log(1+e^{-yf})$. Starting from the constant $f_0 = \log(n_+/n_-)$, round b computes the negative gradients $r_i = -\partial\ell(y_i, f)/\partial f$ at $f = f_{b-1}(\mathbf{x}_i)$, fits a stump h_b to the targets r_i by least squares, and sets $f_b = f_{b-1} + \nu h_b$ with shrinkage ν. A *stump* is $h(\mathbf{x}) = c_{\text{left}}$ if $x_j \le t$ and c_{right} otherwise; its least-squares fit picks the split (j, t) with the smallest squared error, and the two leaf values are the means of the r_i on the two sides.

The starter (`gboost.py`) provides `fit_stump`, `stump_predict`, and `staged_scores` (the scores f_b after every round), and first prints a sanity check of the stump that needs no code from you.

2.1 (2 pts) The targets are residuals. Show that for the logistic loss

$$r_i = \frac{y_i}{1 + e^{y_i f_i}} = y_i' - \sigma(f_i), \qquad f_i := f_{b-1}(\mathbf{x}_i), \quad y_i' := \tfrac{1+y_i}{2} \in \{0, 1\}.$$

So each round fits the next stump to the residual between the 0/1 label and the current model's predicted probability, as each round of squared-loss boosting in lecture fits the residual $y_i - f_i$.

2.2 (4 pts) Boosting. Implement `neg_gradient` (1 pt) and `boost` (3 pts), following the three numbered steps in the docstring of `boost`. Report the two tables the starter prints for $\nu = 1$ and $\nu = 0.1$ (400 rounds each).

2.3 (3 pts) Reading the runs.

(a) **(1 pt)** Include `boosting_curves.pdf`. For each ν, report the round $b^\star$ with the lowest validation loss and the test error there.

(b) **(2 pts)** Compare round b of the $\nu = 1$ table with round $10b$ of the $\nu = 0.1$ table (rounds 1 and 10, then 10 and 100). What do you see, and why is it expected from the update $f_b = f_{b-1} + \nu h_b$? At $\nu = 1$ the training loss keeps falling after round $b^\star$ while the validation loss does not: which of the two would you use to choose the number of rounds B, and why?

2.4 (3 pts) What stumps cannot do. The starter runs your code on Problem 1's rings and xor. Report the printed errors, then:

(a) **(1 pt)** Each stump reads one coordinate, so boosted stumps always produce an *additive* model, $f_B(\mathbf{x}) = f_0 + g_1(x_1) + \cdots + g_d(x_d)$. Give an additive function of (x_1, x_2) that separates the rings.

(b) **(2 pts)** Prove that no additive model can be positive at $(1, 1)$ and $(-1, -1)$ and negative at $(1, -1)$ and $(-1, 1)$. (Hint: add the first two values, then the last two.) What did Problem 1's network have that the stumps lack?

Problem 3: a two-layer attention circuit, built by hand (20 pts - Medium)

No training in this problem: the attention matrices W_Q, W_K, W_V are set by hand so that two attention layers solve a task that a counting model cannot. Layer 1 is given to you as a worked example; you build layer 2.

The task. The vocabulary has 12 letters (tokens $1, \ldots, 12$) and a start token `BOS` (token 0). A sequence is `BOS`, then $L = 6$ distinct random letters $a_1, \ldots, a_L$, then the same letters again:

$$\texttt{BOS}\ \ a_1\ a_2\ \cdots\ a_L\ \ a_1\ a_2\ \cdots\ a_L,$$

which is $T = 2L + 1 = 13$ tokens, at positions $0, \ldots, 12$. The goal is next-token prediction on the second half: at positions $t = L + 1, \ldots, 2L - 1$ the next token is determined by the first half.

The model. Attention is as in Lecture 08, with the causal mask M and a row-wise softmax:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V, \qquad P = \mathrm{softmax}\big(QK^\top/\sqrt{D_K} + M\big).$$

Two simplifications: a layer outputs $X + PV$ (the residual connection; W_V is $D \times D$), and there is no MLP and no layer norm. Each token is a row vector with four blocks,

$$\mathbf{x}_t = \big[\ \underbrace{\text{one-hot}(\text{token}_t)}_{\texttt{TOK}}\ \big|\ \underbrace{\mathbf{0}}_{\texttt{PREV}}\ \big|\ \underbrace{\mathbf{0}}_{\texttt{OUT}}\ \big|\ \underbrace{\text{one-hot}(t)}_{\texttt{POS}}\ \big] \in \mathbb{R}^{52}, \qquad \text{each block of size 13.}$$

`TOK` and `POS` are the token embedding and a (one-hot) positional encoding; `PREV` and `OUT` are empty space for the two layers to write into. After layer 2, the `OUT` block at position t is read as the prediction for token $t + 1$.

3.1 (3 pts) Attention. The starter file for this problem is `attention_heads.py`. Implement `attention_weights` and `attention_layer`. Report the output under `== 3.1 ==`; the first two matrices must match the lecture's worked example without and with the causal mask.

3.2 (3 pts) The counting baseline. The starter fits an order-1 Markov model by counting on 5000 training sequences (`bigram_fit`, provided) and prints its accuracy on the second half of 500 test sequences.

(a) **(2 pts)** Report the printed accuracy and explain it: what does row a of the fitted table look like for a letter a, and why?

(b) **(1 pt)** A Markov model of order $k = L = 6$ could predict the second half exactly, because the next token appeared exactly L positions earlier. How many distinct contexts of 6 distinct letters are there? Compare with the 5000 training sequences.

3.3 (4 pts) Layer 1: a previous-token head. The provided function `previous_token_head` sets, with all other blocks zero,

$$W_Q[\texttt{POS}, :] = \beta S, \qquad W_K[\texttt{POS}, :] = I_T, \qquad W_V[\texttt{TOK}, \texttt{PREV}] = I_{13},$$

where S is the $T \times T$ shift matrix whose row t is the one-hot vector e_{t-1} (row 0 is zero), and $\beta = 100$.

(a) **(2 pts)** Write the query q_t and the key k_s as one-hot vectors, and from them the score of (t, s). Which position does t attend to, and what does W_V copy, into which block? Report the lines printed under `== 3.3 ==`.

(b) **(2 pts)** The starter reruns layer 1 with the `POS` block zeroed. Report row 4 of P and explain it.

3.4 (6 pts) Layer 2: match and copy. After layer 1, position s holds its own token in `TOK` and the token before it in `PREV`.

(a) **(4 pts)** Fill in `match_and_copy_head`: position t attends to the positions s whose *previous* token equals the token at t, and copies the token at s into `OUT`. As in layer 1, each matrix is zero except for one identity block (times β for W_Q): you only decide which block each matrix reads and writes. Write the three matrices in block form, and explain in two or three sentences why this predicts the next token on the second half. Report the accuracy printed under `== 3.4 ==` and include `attention_patterns.pdf`.

(b) **(2 pts)** The starter reruns the circuit on sequences without `BOS`. Report the accuracy by position and the printed row of P for layer 2. Which position fails, and why? (Hint: without `BOS`, what does position 0 write into its own `PREV` block in layer 1?)

3.5 (4 pts) Sharpness and in-context learning. Both heads multiply their queries by a constant β.

(a) **(2 pts)** Report the table printed under `== 3.5 ==`. Show that the weight layer 1 puts on position $t-1$ from position t is

$$\frac{e^{\beta/\sqrt{T}}}{e^{\beta/\sqrt{T}}+t},$$

and say in one sentence why the circuit fails when β is small.

(b) **(2 pts)** The same weights are run on sequences with $L = 2$ and $L = 4$; report the accuracies. The rule "after a_i comes a_{i+1}" is different for every sequence, and the weights were fixed before any sequence was drawn. Where is that rule stored when the circuit makes a prediction? Relate this to Lecture 10's definition of in-context learning.