Homework #3

CSEP 546: Machine Learning
Professor Jamie Morgenstern
Due: **Friday, May 29, 2026, 11:59pm**
100 points

Please review all homework guidance posted on the website before submitting to Gradescope. Reminders:

- All code must be written in Python and all written work must be typeset (e.g. LaTeX).
- Make sure to read the "What to Submit" section following each question and include all items.
- Please provide succinct answers and supporting reasoning for each question. Similarly, when discussing experimental results, concisely create tables and/or figures when appropriate to organize the experimental results. All explanations, tables, and figures for any particular part of a question must be grouped together.
- For every problem involving generating plots, please include the plots as part of your PDF submission.
- When submitting to Gradescope, please link each question from the homework in Gradescope to the location of its answer in your homework PDF. Failure to do so may result in deductions of up to 10% of the value of each question not properly linked. For instructions, see this Gradescope submission guide.
- **If you used AI tools on any part of this assignment, you must turn in a complete transcript of all conversations with AI systems as part of your submission. Failure to do so is a violation of the collaboration policy.**
- Make sure to submit your code by running `inv submit` from within the `csep546` uv environment and submitting the zip folder created. Submissions that don't follow this procedure will fail the autograder on Gradescope. After the due date, we will not accept resubmissions. **We suggest you wait for the autograder on Gradescope to complete and display your score before regarding the coding portion of assignments complete.**

Not adhering to these reminders may result in point deductions.

Important: By turning in this assignment (and all that follow), you acknowledge that you have read and understood the collaboration policy with humans and AI assistants alike: `https://courses.cs.washington.edu/courses/csep546/26sp/assignments/`. Any questions about the policy should be raised at least 24 hours before the assignment is due. There are no warnings or second chances. If we suspect you have violated the collaboration policy, we will report it to the college of engineering who will complete an investigation.

Conceptual Questions

A1. The answers to these questions should be answerable without referring to external materials. Briefly justify your answers with a few words.

a. *[2 points]* True or False: Training deep neural networks requires minimizing a convex loss function, and therefore gradient descent will provide the best result.

b. *[2 points]* True or False: It is a good practice to initialize all weights to zero when training a deep neural network.

c. *[2 points]* True or False: We use non-linear activation functions in a neural network's hidden layers so that the network learns non-linear decision boundaries.

d. *[2 points]* True or False: Given a neural network, the time complexity of the backward pass step in the backpropagation algorithm can be prohibitively larger compared to the relatively low time complexity of the forward pass step.

e. *[2 points]* True or False: Neural networks are the most extensible model and therefore the best choice for any circumstance.

What to Submit:

- **Parts a-e:** 1-2 sentence explanation containing your answer.

Kernels

A2. *[5 points]* Suppose that our inputs x are one-dimensional and that our feature map is infinite-dimensional: $\phi(x)$ is a vector whose ith component is:

$$\frac{1}{\sqrt{i!}} e^{-x^2/2} x^i \ ,$$

for all nonnegative integers i. (Thus, ϕ is an infinite-dimensional vector.) Show that $K(x, x') = e^{-\frac{(x-x')^2}{2}}$ is a kernel function for this feature map, i.e.,

$$\phi(x) \cdot \phi(x') = e^{-\frac{(x-x')^2}{2}} \ .$$

Hint: Use the Taylor expansion of $z \mapsto e^z$. (This is the one-dimensional version of the Gaussian (RBF) kernel.)

What to Submit:

- Proof.

A3. This problem will get you familiar with kernel ridge regression using the polynomial and RBF kernels. First, let's generate some data. Let $n = 30$ and $f_*(x) = 6\sin(\pi x)\cos(4\pi x^2)$. For $i = 1, \ldots, n$ let each x_i be drawn uniformly at random from $[0, 1]$, and let $y_i = f_*(x_i) + \epsilon_i$ where $\epsilon_i \sim \mathcal{N}(0, 1)$. For any function f, the true error and the train error are respectively defined as:

$$\mathcal{E}_{\text{true}}(f) = \mathbb{E}_{X,Y}\left[(f(X) - Y)^2\right], \qquad \widehat{\mathcal{E}}_{\text{train}}(f) = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2 \, .$$

Our goal is, using kernel ridge regression, to construct a predictor:

$$\widehat{\alpha} = \arg\min_{\alpha} \|K\alpha - y\|_2^2 + \lambda \alpha^\top K \alpha \ , \qquad \widehat{f}(x) = \sum_{i=1}^{n} \widehat{\alpha}_i k(x_i, x)$$

where $K \in \mathbb{R}^{n \times n}$ is the kernel matrix such that $K_{i,j} = k(x_i, x_j)$, and $\lambda \geq 0$ is the regularization constant.

a. *[10 points]* Using leave-one-out cross validation, find a good λ and hyperparameter settings for the following kernels:

- $k_{\text{poly}}(x, z) = (1 + x^\top z)^d$ where $d \in \mathbb{N}$ is a hyperparameter,
- $k_{\text{rbf}}(x, z) = \exp(-\gamma \|x - z\|_2^2)$ where $\gamma > 0$ is a hyperparameter[1].

We strongly recommend implementing either grid search or random search. **Do not use sklearn**, but actually implement these algorithms. Reasonable values: $\lambda \in 10^{[-5,-1]}$ and $d \in [5, 25]$. You do **not** need to search over γ (use the heuristic in the footnote). Report the values of d, λ, and γ for both kernels.

b. *[10 points]* Let $\widehat{f}_{\text{poly}}(x)$ and $\widehat{f}_{\text{rbf}}(x)$ be the functions learned using the hyperparameters from part a. For a single plot per function $\widehat{f} \in \left\{\widehat{f}_{\text{poly}}(x), \widehat{f}_{\text{rbf}}(x)\right\}$, plot the original data $\{(x_i, y_i)\}_{i=1}^n$, the true $f(x)$, and $\widehat{f}(x)$ (define a fine grid on $[0, 1]$ to plot the functions).

What to Submit:

- **Part a:** Report the values of d, γ, and λ for both kernels.
- **Part b:** Two plots, one per function.
- **Code** on Gradescope through coding submission (hw3-A.zip).

Introduction to PyTorch

Resources

For the following two problems you will use PyTorch. Make use of PyTorch Documentation as needed. If you do not have access to a GPU, Google Colaboratory provides free cloud GPUs. To enable it: "Runtime" → "Change runtime type" → set "Hardware accelerator" to "GPU".

A4. PyTorch is a great tool for developing, deploying, and researching neural networks and other gradient-based algorithms. In this problem we will explore how this package is built and re-implement some of its core components. Start by reading the `README.md` file in the `intro_pytorch` subfolder of hw3-A.zip.

a. *[10 points]* Implement the components in folders `layers`, `losses`, and `optimizers`. Almost each file contains at least one problem function with exact directions. Lastly, implement functions in `train.py`.

b. *[5 points]* Using the above module, perform a hyper-parameter search[2]. Train (in the order provided) 6 models using each of `crossentropy_search.py` and `mean_squared_error_search.py`:

- Linear neural network (single layer, no activation)
- NN with one hidden layer (2 units), sigmoid activation after hidden layer
- NN with one hidden layer (2 units), ReLU activation after hidden layer
- NN with two hidden layers (2 units each), Sigmoid then ReLU activations
- NN with two hidden layers (2 units each), ReLU then Sigmoid activations
- NN with two hidden layers (2 units each), ReLU then ReLU activations

For each loss function, submit a plot of training and validation losses. All 6 models should appear on the same plot (12 lines per plot, 2 plots total).

c. *[5 points]* For each loss function, report the best-performing architecture (lowest validation loss at any point during training) and plot its predictions on the test set using `plot_model_guesses` from `train.py`. Report accuracy on the test set.

[1] Given a dataset $x_1, \ldots, x_n \in \mathbb{R}^d$, a heuristic for choosing a range of γ is the inverse of the median of all $\binom{n}{2}$ squared distances $\|x_i - x_j\|_2^2$.

[2] Hyper-parameters here are (1) model architectures and (2) loss functions. You do not need to search over batch size and learning rate as long as the loss curve converges.

The Softmax Function

One of the activation functions you will implement is softmax. For a prediction $\hat{y} \in \mathbb{R}^k$ corresponding to a single datapoint (in a problem with k classes):

$$\text{softmax}(\hat{y}_i) = \frac{\exp(\hat{y}_i)}{\sum_j \exp(\hat{y}_j)}$$

What to Submit:

- **Part b:** 2 plots (one per loss function), 12 lines each, showing training and validation loss. Include titles and legends.
- **Part c:** Names/descriptions of best-performing architectures and their test accuracy. 2 scatter plots (one per loss function) with predictions on the test set.
- **Code** on Gradescope through coding submission (hw3-A.zip).

Neural Networks for MNIST

A5. In Homework 1, we used ridge regression to train a classifier for the MNIST dataset. In Homework 2, we used logistic regression to distinguish between the digits 2 and 7. Now, we will use PyTorch to build a simple neural network classifier for all 10 MNIST classes.

We will implement two architectures: a shallow but wide network, and a narrow but deeper network. We use d for the number of input features (MNIST: $d = 28^2 = 784$), h_i for the dimension of the i-th hidden layer, and k for the number of target classes (MNIST: $k = 10$). For the non-linear activation, use ReLU:

$$\text{ReLU}(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}.$$

Weight Initialization

For weight matrix $W \in \mathbb{R}^{n \times m}$ and bias $b \in \mathbb{R}^n$ (where m is input dimension and n is output dimension), define $\alpha = \frac{1}{\sqrt{m}}$ and initialize according to $\text{Unif}(-\alpha, \alpha)$.

Training

Use the Adam optimizer from `torch.optim`. You may use full-batch GD, SGD, or mini-batch SGD. Use cross-entropy loss and ReLU for non-linearity.

Implementing the Neural Networks

a. *[10 points]* Let $W_0 \in \mathbb{R}^{h \times d}$, $b_0 \in \mathbb{R}^h$, $W_1 \in \mathbb{R}^{k \times h}$, $b_1 \in \mathbb{R}^k$. The forward pass of the wide, shallow network is:

$$\mathcal{F}_1(x) \coloneqq W_1 \sigma(W_0 x + b_0) + b_1$$

Use $h = 64$ hidden units. Train until 99% training accuracy and provide a training loss vs. epoch plot. Report test accuracy and loss.

b. *[10 points]* Let $W_0 \in \mathbb{R}^{h_0 \times d}$, $b_0 \in \mathbb{R}^{h_0}$, $W_1 \in \mathbb{R}^{h_1 \times h_0}$, $b_1 \in \mathbb{R}^{h_1}$, $W_2 \in \mathbb{R}^{k \times h_1}$, $b_2 \in \mathbb{R}^k$. The forward pass of the deeper network is:

$$\mathcal{F}_2(x) \coloneqq W_2 \sigma(W_1 \sigma(W_0 x + b_0) + b_1) + b_2$$

Use $h_0 = h_1 = 32$. Perform the same steps as in part a.

c. *[5 points]* Compute the total number of parameters for each network. Compare the two architectures (wide/shallow vs. narrow/deeper) in terms of parameter count and test accuracy. Is one approach better? Give an intuition for why or why not.

PyTorch restriction: You may not use any functionality from `torch.nn` except for `torch.nn.functional.relu` and `torch.nn.functional.cross_entropy`. Implement $\mathcal{F}_1$ and $\mathcal{F}_2$ from scratch.

What to Submit:

- **Parts a-b:** Training loss vs. epoch plot. Test accuracy and loss.
- **Part c:** Parameter counts for both networks. 1-2 sentence comparison.
- **Code** on Gradescope through coding submission (hw3-A.zip).

Multinomial Logistic Regression

A6. In this problem you will derive and implement logistic regression for the multiclass ($k > 2$) setting. Given a dataset $\{(x_i, y_i)\}_{i=1}^n$ with $x_i \in \mathbb{R}^d$ and $y_i \in \{1, \ldots, k\}$, multiclass logistic regression uses a weight matrix $W \in \mathbb{R}^{k \times d}$ to predict a distribution over classes via the softmax:

$$p(y = j \mid x; W) = \frac{\exp(W_j^\top x)}{\sum_{\ell=1}^k \exp(W_\ell^\top x)}$$

where W_j is the j-th row of W.
We will use MNIST ($k = 10$ classes) for this problem.

a. *[5 points]* The *cross-entropy loss* for a single example (x_i, y_i) is:

$$L_i(W) = -\log p(y_i \mid x_i; W)$$

Derive the gradient $\nabla_{W_j} L_i(W)$ for a fixed example (x_i, y_i). Your answer should be in terms of $p(y = j \mid x_i; W)$.

b. *[5 points]* Using the provided code skeleton in `multinomial_log_regression.py`, implement `J_loss` (joint log-likelihood) and `L_loss` (standard cross-entropy), as well as `accuracy`, `train`, and `main`. Run `main` to train models with each loss and generate the required plots. Report training and test accuracy for both losses.

What to Submit:

- **Part a:** Derivation of $\nabla_{W_j} L_i(W)$.
- **Part b:** Training and test accuracy for both `J_loss` and `L_loss`. Plots generated by `main`.