Homework #4

CSEP 546: Machine Learning
Professor Jamie Morgenstern
Due: **Thursday, June 11, 2026, 10:20pm**
91 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.**
- For the k-means coding problem, make sure to submit your code by running `inv submit` from within the provided HW4 code package 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.**

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. Not adhering to these reminders may result in point deductions.

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: Given a data matrix $X \in \mathbb{R}^{n \times d}$ where d is much smaller than n and $k = \text{rank}(X)$, if we project our data onto a k-dimensional subspace using PCA, our projection will have zero reconstruction error (in other words, we find a perfect representation of our data, with no information loss).

b. *[2 points]* True or False: Suppose that an $n \times n$ matrix X has a singular value decomposition of $USV^\top$, where S is a diagonal $n \times n$ matrix. Then, the rows of V are equal to the eigenvectors of $X^\top X$.

c. *[2 points]* True or False: choosing k to minimize the k-means objective (see Equation (1) below) is a good way to find meaningful clusters.

d. *[2 points]* True or False: The singular value decomposition of a matrix is unique.

e. *[2 points]* True or False: The rank of a square matrix equals the number of its unique nonzero eigenvalues.

What to Submit:

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

Think before you train

A2. **The first part of this problem (part a)** explores how you would apply machine learning theory and techniques to a real-world problem. There is one scenario detailing a setting, a dataset, and a specific result we hope to achieve. Your job is to describe how you would handle the scenario with the tools we've learned in this class. Your response should include:

(1) any pre-processing steps you would take (e.g. any data processing),

(2) the specific machine learning pipeline you would use (i.e., algorithms and techniques learned in this class),

(3) how your setup acknowledges the constraints and achieves the desired result.

You should also aim to leverage some of the theory we have covered in this class. Some things to consider may be: the nature of the data (i.e., *How hard is it to learn? Do we need more data? Are the data sources good?*), the effectiveness of the pipeline (i.e., *How strong is the model when properly trained and tuned?*), and the time needed to effectively perform the pipeline.

a. *[10 points]* **Scenario: Disease Susceptibility Predictor**

- Setting: You are tasked by a research institute to create an algorithm that learns the factors that contribute most to acquiring a specific disease.
- Dataset: A rich dataset of personal demographic information, location information, risk factors, and whether a person has the disease or not.
- Result: The company wants a system that can determine how susceptible someone is to this disease when they enter in their own personal information. The pipeline should take limited amount of personal data from a new user and infer more detailed metrics about the person. Select **two** methods and briefly discuss the tradeoffs.

The second part of this problem (parts b, c) focuses on exploring possible shortcomings of machine learning models, and what real-world implications might follow from ignoring these issues.

b. *[5 points]* Briefly describe (1) some potential shortcomings of your training process from the disease susceptibility predictor scenario above that may result in your algorithm having different accuracy on different populations, and (2) how you may modify your procedure to address these shortcomings.

c. *[5 points]* Recall in Homework 2 we trained models to predict crime rates using various features. It is important to note that **datasets describing crime have many shortcomings in describing the entire landscape of illegal behavior in a city, and that these shortcomings often fall disproportionately on minority communities**. Some of these shortcomings include that crimes are reported at different rates in different neighborhoods, that police respond differently to the same crime reported or observed in different neighborhoods, and that police spend more time patrolling in some neighborhoods than others. What real-world implications might follow from ignoring these issues?

What to Submit:

- **For part (a):** One clearly-written short paragraph (approximately 5-10 sentences).
- **For part (b):** Clearly-written and well-thought-out answers addressing (1) and (2) (as described in the problem). Two short paragraphs or one medium paragraph suffice.
- **For part (c):** One clearly-written short paragraph on real-world implications that may follow from ignoring dataset issues.

Image Classification on CIFAR-10

A3. In this problem we will explore different deep learning architectures for image classification on the CIFAR-10 dataset. Make sure that you are familiar with `torch.Tensors`, two-dimensional convolutions (`nn.Conv2d`) and fully-connected layers (`nn.Linear`), ReLU non-linearities (`F.relu`), pooling (`nn.MaxPool2d`), and tensor reshaping (`view`).

A few preliminaries:

- Make sure to read any HW4 guidance posted on EdStem for additional tips about training your models.
- Each network f maps an image $x^{\text{in}} \in \mathbb{R}^{32\times32\times3}$ (3 channels for RGB) to an output $f(x^{\text{in}}) = x^{\text{out}} \in \mathbb{R}^{10}$. The class label is predicted as $\arg\max_{i=0,1,\ldots,9} x_i^{\text{out}}$. An error occurs if the predicted label differs from the true label for a given image.
- The network is trained via multiclass cross-entropy loss.
- Create a validation dataset by appropriately partitioning the train dataset. *Hint*: look at the documentation for `torch.utils.data.random_split`. Make sure to tune hyperparameters like network architecture and step size on the validation dataset. Do **NOT** validate your hyperparameters on the test dataset.
- At the end of each epoch (one pass over the training data), compute and print the training and validation classification accuracy.
- While one could train a network for hundreds of epochs to reach convergence and maximize accuracy, this can be prohibitively time-consuming, so feel free to train for just a dozen or so epochs.

For parts (a) and (b), apply a hyperparameter tuning method (e.g. random search, grid search, etc.) using the validation set, report the hyperparameter configurations you evaluated and the best set of hyperparameters from this set, and plot the training and validation classification accuracy as a function of epochs. Produce a separate line or plot for each hyperparameter configuration evaluated (top 3 configurations is sufficient to keep the plots clean). Finally, evaluate your best set of hyperparameters on the test data and report the test accuracy.

Note 1: Please refer to the CIFAR-10 starter notebook linked on the course website. That notebook provides a complete end-to-end example of loading data, training a model using a simple network with a fully-connected output and no hidden layers (this is equivalent to logistic regression), and performing evaluation using canonical Pytorch. We recommend using this as a template for your implementations of the models below.
Note 2: If you are attempting this problem and do not have access to GPU we highly recommend using Google Colab. The provided notebook includes instructions on how to use GPU in Google Colab.

Here are the network architectures you will construct and compare.

a. *[18 points]* **Fully-connected output, 1 fully-connected hidden layer:** this network has one hidden layer denoted as $x^{\text{hidden}} \in \mathbb{R}^M$ where M will be a hyperparameter you choose (M could be in the hundreds). The nonlinearity applied to the hidden layer will be the `relu` ($\text{relu}(x) = \max\{0, x\}$. This network can be written as

$$x^{out} = W_2\text{relu}(W_1(x^{in}) + b_1) + b_2$$

where $W_1 \in \mathbb{R}^{M\times3072}$, $b_1 \in \mathbb{R}^M$, $W_2 \in \mathbb{R}^{10\times M}$, $b_2 \in \mathbb{R}^{10}$. Tune the different hyperparameters and train for a sufficient number of epochs to achieve a *validation accuracy* of at least 50%. Provide the hyperparameter configuration used to achieve this performance.

b. *[18 points]* **Convolutional layer with max-pool and fully-connected output:** for a convolutional layer W_1 with filters of size $k \times k \times 3$, and M filters (reasonable choices are $M = 100$, $k = 5$), we have that $\text{Conv2d}(x^{\text{in}}, W_1) \in \mathbb{R}^{(33-k)\times(33-k)\times M}$.

- Each convolution will have its own offset applied to each of the output pixels of the convolution; we denote this as $\text{Conv2d}(x^{\text{in}}, W) + b_1$ where b_1 is parameterized in $\mathbb{R}^M$. Apply a `relu` activation to the result of the convolutional layer.

- Next, use a max-pool of size $N \times N$ (a reasonable choice is $N = 14$ to pool to 2×2 with $k = 5$) we have that $\text{MaxPool}(\text{relu}(\text{Conv2d}(x^{\text{in}}, W_1) + b_1)) \in \mathbb{R}^{\lfloor \frac{33-k}{N} \rfloor \times \lfloor \frac{33-k}{N} \rfloor \times M}$.
- We will then apply a fully-connected layer to the output to get a final network given as

$$x^{output} = W_2(\text{MaxPool}(\text{relu}(\text{Conv2d}(x^{\text{input}}, W_1) + b_1))) + b_2$$

  where $W_2 \in \mathbb{R}^{10 \times M(\lfloor \frac{33-k}{N} \rfloor)^2}$, $b_2 \in \mathbb{R}^{10}$.

The parameters M, k, N (in addition to the step size and momentum) are all hyperparameters, but you can choose a reasonable value. Tune the different hyperparameters (number of convolutional filters, filter sizes, dimensionality of the fully-connected layers, step size, etc.) and train for a sufficient number of epochs to achieve a *validation accuracy* of at least 60%. Provide the hyperparameter configuration used to achieve this performance.

The number of hyperparameters to tune, combined with the slow training times, will hopefully give you a taste of how difficult it is to construct networks with good generalization performance. State-of-the-art networks can have dozens of layers, each with their own hyperparameters to tune. Additional hyperparameters you are welcome to play with, if you are so inclined, include: changing the activation function, replace max-pool with average-pool, adding more convolutional or fully connected layers, and experimenting with batch normalization or dropout.

What to Submit:

- **For parts (a)-(b):** A single plot of the training and validation accuracy for the top 3 hyperparameter configurations you evaluated (x-axis is training epoch; y-axis is accuracy; this plot should contain 6 lines total). If it took fewer than 3 hyperparameter configurations to pass the performance threshold, plot all hyperparameter configurations you evaluated. A horizontal line should be plotted at the targeted threshold (50% or 60%). Validation lines should be dotted, and training lines should be solid.
- **For parts (a)-(b):** List the hyperparameter values you searched over and your search method (random, grid, etc.). Provide the values of best performing hyperparameters, and accuracy of best models on test data.
- **For parts (a)-(b):** Code. You should convert your code (the .ipynb notebook) into a Python (.py) file, rename it to `hw4-cifar.py`, and submit it to the corresponding Gradescope submission. To download the file from Google Colab, you can go to File ¿ Download ¿ Download as .py.

k-means clustering

A4. Given a dataset $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^d$ and an integer $1 \leq k \leq n$, recall the following k-means objective function

$$\min_{\pi_1,...,\pi_k} \sum_{i=1}^{k} \sum_{j \in \pi_i} \|\mathbf{x}_j - \mu_i\|_2^2 \ , \quad \mu_i = \frac{1}{|\pi_i|} \sum_{j \in \pi_i} \mathbf{x}_j \ . \tag{1}$$

Above, $\{\pi_i\}_{i=1}^k$ is a partition of $\{1, 2, ..., n\}$. The objective (1) is NP-hard[1] to find a global minimizer of. Nevertheless, Lloyd's algorithm (discussed in lecture) typically works well in practice.[2]

a. *[5 points]* Implement Lloyd's algorithm for solving the k-means objective (1). Do not use any off-the-shelf implementations, such as those found in `scikit-learn`.

b. *[5 points]* Run Lloyd's algorithm on the *training* dataset of MNIST with $k = 10$. Show the image representing the center of each cluster, as a set of k 28×28 images.

Note on Time to Run — The runtime of a good implementation for this problem should be fairly fast (a few minutes); if you find it taking upwards of one hour, please check your implementation! (Hint: **For loops are costly.** Can you vectorize it or use Numpy operations to make it faster in some ways? If not, is looping through data-points or through centers faster?)

What to Submit:

- **For part (a):** Nothing required in PDF submission.
- **For part (b):** 10 images of cluster centers.
- **For parts (a)-(b):** Code. Use the HW4 code package linked on the course website. After implementing `homeworks/k_means/k_means.py` and `homeworks/k_means/main.py`, run `uv run inv submit` and submit the generated zip file to the corresponding Gradescope coding submission.

[1] To be more precise, it is both NP-hard in d when $k = 2$ and k when $d = 2$.
[2] See the references on the Wikipedia page for k-means and k-means++ for more details.

Transformers

A5.

a. *[10 points]* In this problem, you will implement the *multi-headed causally-masked scaled dot-product attention* portion of the transformer architecture for text generation. A transformer block consists of two parts: first, there is the attention block, which you will implement, and a linear layer which is implemented for you. The attention block is parametrized by four weight matrices, W_Q, W_K, W_V, and W_O. The W_Q, W_K, W_V weights are used to get the Q, K, V matrices for self-attention, where $Q = XW_Q$ (and so on), where X is the embedded text sequence. Remember that self-attention is given by the following equation from the Attention Is All You Need paper:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{2}$$

In self-attention, each Q, K, V are of shape $L \times d_{model}$, where L is the sequence length of the input. When moving from a single head of self-attention to multi-headed attention (MHA), each head of attention, Q_h, K_h, V_h, is $L \times d_k$, where d_k is the attention dimension. Since W_Q, W_K, W_V are still the same shape as before, $d_k = d_{model}/\text{num heads}$. In multi-headed attention, there is an additional W_O matrix, which is a layer after the MHA that learns to pass information between each head of attention.

For the dataset, you will be using the Shakespeare corpus, which contains all works from William Shakespeare. Important notes to remember:

- Remember to embed the input. We will use a pretrained word vectorizer called GloVE.
- Remember to apply the positional encoding to the Q, K matrices. Attention alone is position invariant.

b. *[5 points]* When sampling from the transformer, we apply softmax to the logits so that it is a valid probability distribution, where the equation for softmax is given by

$$\text{softmax}(x_i) = \frac{e^{x_i/T}}{\sum_{j=1}^{n} e^{x_j/T}}$$

where T is the sample temperature.

- What value of T makes this equivalent to greedy decoding? Justify your answer.
- What value of T makes this equivalent to vanilla softmax sampling? Justify your answer.
- What value of T makes this equivalent to sampling from a uniform distribution over all tokens? Justify your answer.

c. **Extra Credit** *[5 points]* As you have noticed, the text generations are very Shakespearean. Try downloading your email history from https://takeout.google.com/?pli=1, and training the same transformer over this data distribution! You will have to convert the data into a corpus following the same format as the Shakespeare corpus. Check the Colab directories to check the formatting. Make a fresh copy of the notebook so that you can change the training dataset, and rename it to `hw4-transformer-ec.py`.

Does the new text generations sound like you? Would you trust it to respond to your emails?

What to submit:

- **For part (a):** The plot of the training and validation loss curves and 3 example text generations that you like. Submit the Colab notebook as `hw4-transformer.py` to Gradescope.
- **For part (b):** 1-2 sentence explanation containing your answer for each question.
- **Optional for part (c):** The plot of the training and validation loss curves and 3 example text generations that you like. Submit the extra-credit Colab notebook as `hw4-transformer-ec.py` to Gradescope.