Homework #3

CSE 446/546: Machine Learning
Prof. Kevin Jamieson and Prof. Simon S. Du
Due: May 19, 2023 11:59pm
Points A: 90; B: 5

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

- 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 https://www.gradescope.com/get_started#student-submission.
- If you collaborate on this homework with others, you must indicate who you worked with on your homework by providing a complete list of collaborators on the first page of your assignment. Make sure to include the name of each collaborator, and on which problem(s) you collaborated. Failure to do so may result in accusations of plagiarism. You can review the course collaboration policy at https://courses.cs.washington.edu/courses/cse446/23sp/assignments/
- For every problem involving code, please include all code you have written for the problem as part of your PDF submission *in addition to* submitting your code to the separate assignment on Gradescope created for code. Not submitting all code files will lead to a deduction of up to 10% of the value of each question missing code.

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]* Say you trained an SVM classifier with an RBF kernel ($K(u,v) = \exp\left(-\frac{\|u-v\|_2^2}{2\sigma^2}\right)$). It seems to underfit the training set: should you increase or decrease σ?

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

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

d. *[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.

e. *[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.

f. *[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-f:** 1-2 sentence explanation containing your answer.

Support Vector Machines

A2. Recall that solving the SVM problem amounts to solving the following constrained optimization problem:

$$\text{Given data points } \mathcal{D} = \{(x_i, y_i)\}_{i=1}^n \text{ find}$$

$$\min_{w,b} ||w||_2 \text{ subject to } y_i(x_i^T w - b) \geq 1 \text{ for } i \in \{1, \dots, n\}$$

$$\text{where } x_i \in \mathbb{R}^d,\ y_i \in \{-1, 1\}, \text{ and } w \in \mathbb{R}^d.$$

Consider the following labeled data points:

$$\begin{bmatrix} 1 & 2 \\ 1 & 3 \\ 2 & 3 \\ 3 & 4 \end{bmatrix} \text{ with label } y = -1 \text{ and } \begin{bmatrix} 0 & 0.5 \\ 1 & 0 \\ 2 & 1 \\ 3 & 0 \end{bmatrix} \text{ with label } y = 1$$

a. *[2 points]* Graph the data points above. Highlight the support vectors and write their coordinates. Draw the two parallel hyperplanes separating the two classes of data such that the distance between them is as large as possible. Draw the maximum-margin hyperplane. Write the equations describing these three hyperplanes using only x, w, b(that is without using any specific values). Draw w(it doesn't have to have the exact magnitude, but it should have the correct orientation).

b. *[2 points]* For the data points above, find w and b.

Hint: Use the support vectors and the values $\{-1, 1\}$ to create a linear system of equations where the unknowns are w_1, w_2 and b.

c. *[4 points]* Show that for any solvable SVM problem, the distance between the two separating hyperplanes is $\frac{2}{||w||_2}$.

Hint 1: The distance between two hyperplanes is the distance between any point x_0 on one of the hyperplanes and its projection on the other hyperplane.
Hint 2: A direction w and an offset c define the hyperplane: $H = \{x \in \mathbb{R}^n | w^T x = c\}$. The projection of a vector y onto H is given by $P_H(y) = y - \frac{w^T y - c}{||w||_2^2} w$.

What to Submit:

- **Part a:** Write down support vectors and equations. Graph the points, hyperplanes, and w.
- **Part b:** Solution and corresponding calculations.
- **Part c:** Proof.

Kernels

A3. *[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.

A4. 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 $f_*(x) = 4\sin(\pi x)\cos(6\pi x^2)$. 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:

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

Now, 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].

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

We strongly recommend implementing either grid search or random search. **Do not use sklearn**, but actually implement of these algorithms. Reasonable values to look through in this problem are: $\lambda \in 10^{[-5,-1]}$, $d \in [5, 25]$, γ sampled from a narrow gaussian distribution centered at value described in the footnote.
Report the values of d, γ, and the λ values 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 you found in 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 $\widehat{f}(x)$ (i.e., define a fine grid on $[0, 1]$ to plot the functions).

What to Submit:

- **Part a:** Report the values of d, γ and the value of λ for both kernels as described.
- **Part b:** Two plots. One plot for each function.
- **Code** on Gradescope through coding submission.

Perceptron

B1. One of the oldest algorithms used in machine learning (from the early 60's) is an online algorithm for learning a linear threshold function called the Perceptron Algorithm. It works as follows:

1. Start with the all-zeroes weight vector $\mathbf{w}_1 = 0$, and initialize t to 1. Also let's automatically scale all examples $\mathbf{x}$ to have (Euclidean) norm 1, since this doesn't affect which side of the plane they are on.
2. Given example $\mathbf{x}$, predict positive iff $\mathbf{w}_t \cdot \mathbf{x} > 0$.
3. On a mistake, update as follows:
   - Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \mathbf{x}$.
   - Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \mathbf{x}$.
4. $t \leftarrow t + 1$.

If we make a mistake on a positive $\mathbf{x}$ we get $\mathbf{w}_{t+1} \cdot \mathbf{x} = (\mathbf{w}_t + \mathbf{x}) \cdot \mathbf{x} = \mathbf{w}_t \cdot \mathbf{x} + 1$, and similarly if we make a mistake on a negative $\mathbf{x}$ we have $\mathbf{w}_{t+1} \cdot \mathbf{x} = (\mathbf{w}_t - \mathbf{x}) \cdot \mathbf{x} = \mathbf{w}_t \cdot \mathbf{x} - 1$. So, in both cases we move closer (by 1) to the value we wanted. Here is a link if you are interested in more details.

Now consider the linear decision boundary for classification (labels in $\{-1, 1\}$) of the form $\mathbf{w} \cdot \mathbf{x} = 0$ (i.e., no offset). Now consider the following loss function evaluated at a data point $(\mathbf{x}, y)$ which is a variant on the hinge loss.

$$\ell((\mathbf{x}, y), \mathbf{w}) = \max\{0, -y(\mathbf{w} \cdot \mathbf{x})\}.$$

a. *[2 points]* Given a dataset of $(\mathbf{x}_i, y_i)$ pairs, write down a single step of subgradient descent with a step size of η if we are trying to minimize

$$\frac{1}{n}\sum_{i=1}^{n} \ell((\mathbf{x}_i, y_i), \mathbf{w})$$

for $\ell(\cdot)$ defined as above. That is, given a current iterate $\widetilde{\mathbf{w}}$ what is an expression for the next iterate?

b. *[2 points]* Use what you derived to argue that the Perceptron can be viewed as implementing SGD applied to the loss function just described (for what value of η)?

c. *[1 point]* Suppose your data was drawn i.i.d. and that there exists a $\mathbf{w}^*$ that separates the two classes perfectly. Provide an explanation for why hinge loss is generally preferred over the loss given above.

What to Submit:

- **Part a:** Expression for a single step of subgradient descent
- **Part b:** A 1-2 sentence explanation).
- **Part c:** A 1-2 sentence explanation).

Introduction to PyTorch

A5. 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. Firstly start by reading `README.md` file provided in `intro_pytorch` subfolder. A lot of problem statements will overlap between here, readme's and comments in functions.

a. *[10 points]* You will start by implementing components of our own PyTorch modules. You can find these in folders: `layers`, `losses` and `optimizers`. Almost each file there should contain at least one problem function, including exact directions for what to achieve in this problem. Lastly, you should implement functions in `train.py` file.

b. *[5 points]* Next we will use the above module to perform hyperparameter search. Here we will also treat loss function as a hyper-parameter. However, because cross-entropy and MSE require different shapes we are going to use two different files: `crossentropy_search.py` and `mean_squared_error_search.py`. For each you will need to build and train (in provided order) 5 models:

- Linear neural network (Single layer, no activation function)
- NN with one hidden layer (2 units) and sigmoid activation function after the hidden layer
- NN with one hidden layer (2 units) and ReLU activation function after the hidden layer
- NN with two hidden layer (each with 2 units) and Sigmoid, ReLU activation functions after first and second hidden layers, respectively
- NN with two hidden layer (each with 2 units) and ReLU, Sigmoid activation functions after first and second hidden layers, respectively

For each loss function, submit a plot of losses from training and validation sets. All models should be on the same plot (10 lines per plot), with two plots total (1 for MSE, 1 for cross-entropy).

c. *[5 points]* For each loss function, report the best performing architecture (best performing is defined here as achieving the lowest validation loss at any point during the training), and plot it's guesses on test set. You should use function `plot_model_guesses` from `train.py` file. Lastly, report accuracy of that model on a test set.

On Softmax function

One of the activation functions we ask you to implement is softmax. For a prediction $\hat{y} \in \mathbb{R}^k$ corresponding to 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), with 10 lines each, showing both training and validation loss of each model. Make sure plots are titled, and have proper legends.
- **Part c:** Names of best performing models (i.e. descriptions of their architectures), and their accuracy on test set.
- **Part c:** 2 scatter plots (one per loss function), with predictions of best performing models on test set.
- **Code** on Gradescope through coding submission

Neural Networks for MNIST

Resources

For questions A.4, A.5 and A.6 you will use a lot of PyTorch. In Section materials (Week 6) there is a notebook that you might find useful. Additionally make use of PyTorch Documentation, when needed.
If you do not have access to GPU, you might find Google Colaboratory useful. It allows you to use a cloud GPU for free. To enable it make sure: "Runtime" -> "Change runtime type" -> "Hardware accelerator" is set to "GPU". When submitting please download and submit a `.py` version of your notebook.

A6. In Homework 1, we used ridge regression for training a classifier for the MNIST data set. In Homework 2, we used logistic regression to distinguish between the digits 2 and 7. In this problem, we will use PyTorch to build a simple neural network classifier for MNIST to further improve our accuracy.

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

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

Weight Initialization

Consider a weight matrix $W \in \mathbb{R}^{n\times m}$ and $b \in \mathbb{R}^n$. Note that here m refers to the input dimension and n to the output dimension of the transformation $x \mapsto Wx + b$. Define $\alpha = \frac{1}{\sqrt{m}}$. Initialize all your weight matrices and biases according to $\mathrm{Unif}(-\alpha, \alpha)$.

Training

For this assignment, use the Adam optimizer from `torch.optim`. Adam is a more advanced form of gradient descent that combines momentum and learning rate scaling. It often converges faster than regular gradient descent in practice. You can use either Gradient Descent or any form of Stochastic Gradient Descent. Note that you are still using Adam, but might pass either the full data, a single datapoint or a batch of data to it. Use cross entropy for the loss function and ReLU for the non-linearity.

Implementing the Neural Networks

a. *[10 points]* Let $W_0 \in \mathbb{R}^{h\times d}$, $b_0 \in \mathbb{R}^h$, $W_1 \in \mathbb{R}^{k\times h}$, $b_1 \in \mathbb{R}^k$ and $\sigma(z)\colon \mathbb{R} \to \mathbb{R}$ some non-linear activation function applied element-wise. Given some $x \in \mathbb{R}^d$, the forward pass of the wide, shallow network can be formulated as:

$$\mathcal{F}_1(x) \coloneqq W_1\sigma(W_0x + b_0) + b_1$$

Use $h = 64$ for the number of hidden units and choose an appropriate learning rate. Train the network until it reaches 99% accuracy on the training data and provide a training plot (loss vs. epoch). Finally evaluate the model on the test data and report both the accuracy and the loss.

b. *[10 points]* Let $W_0 \in \mathbb{R}^{h_0 \times d}$, $b_0 \in \mathbb{R}^{h_0}$, $W_1 \in \mathbb{R}^{h_1 \times h_0}$, $b_1 \in \mathbb{R}^{h_1}$, $W_2 \in \mathbb{R}^{k \times h_1}$, $b_2 \in \mathbb{R}^k$ and $\sigma(z) : \mathbb{R} \to \mathbb{R}$ some non-linear activation function. Given some $x \in \mathbb{R}^d$, the forward pass of the network can be formulated as:

$$\mathcal{F}_2(x) := W_2\sigma(W_1\sigma(W_0x + b_0) + b_1) + b_2$$

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

c. *[5 points]* Compute the total number of parameters of each network and report them. Then compare the number of parameters as well as the test accuracies the networks achieved. Is one of the approaches (wide, shallow vs. narrow, deeper) better than the other? Give an intuition for why or why not.

Using PyTorch: For your solution, you may not use any functionality from the `torch.nn` module except for `torch.nn.functional.relu` and `torch.nn.functional.cross_entropy`. You must implement the networks $\mathcal{F}_1$ and $\mathcal{F}_2$ from scratch. For starter code and a tutorial on PyTorch refer to the sections 6 and 7 material.

What to Submit:

- **Parts a-b:** Provide a plot of the training loss versus epoch. In addition evaluate the model trained on the test data and report the accuracy and loss.
- **Part c:** Report the number of parameters for the network trained in part (a) and for the network trained in part (b). Provide a comparison of the two networks as described in part in 1-2 sentences.
- **Code** on Gradescope through coding submission.