Homework #3

CSE 446/546: Machine Learning
Profs. Jamie Morgenstern and Ludwig Schmidt
Due: **Wednesday** Nov 23, 2022 11:59pm
A: 101 points, **B**: 16 points

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/cse446/22au/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 non-convex loss function, and therefore gradient descent might not reach the globally-optimal solution.

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:

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

Logistic Regression

A2. Here we consider the MNIST dataset, but for binary classification. Specifically, the task is to determine whether a digit is a 2 or 7. Here, let $Y = 1$ for all the "7" digits in the dataset, and use $Y = -1$ for "2". We will use regularized logistic regression. Given a binary classification dataset $\{(x_i, y_i)\}_{i=1}^n$ for $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$ we showed in class that the regularized negative log likelihood objective function can be written as

$$J(w, b) = \frac{1}{n}\sum_{i=1}^{n} \log(1 + \exp(-y_i(b + x_i^T w))) + \lambda||w||_2^2$$

Note that the offset term b is not regularized. For all experiments, use $\lambda = 10^{-1}$. Let $\mu_i(w, b) = \frac{1}{1+\exp(-y_i(b+x_i^T w))}$.

a. *[8 points]* Derive the gradients $\nabla_w J(w, b)$, $\nabla_b J(w, b)$ and give your answers in terms of $\mu_i(w, b)$ (your answers should not contain exponentials).

b. *[8 points]* Implement gradient descent with an initial iterate of all zeros. Try several values of step sizes to find one that appears to make convergence on the training set as fast as possible. Run until you feel you are near to convergence.

   (i) For both the training set and the test, plot $J(w, b)$ as a function of the iteration number (and show both curves on the same plot).

   (ii) For both the training set and the test, classify the points according to the rule $\text{sign}(b + x_i^T w)$ and plot the misclassification error as a function of the iteration number (and show both curves on the same plot).

   Reminder: Make sure you are only using the test set for evaluation (not for training).

c. *[7 points]* Repeat (b) using stochastic gradient descent with a batch size of 1. Note, the expected gradient with respect to the random selection should be equal to the gradient found in part (a). Show both plots described in (b) when using batch size 1. Take careful note of how to scale the regularizer.

d. *[7 points]* Repeat (b) using stochastic gradient descent with batch size of 100. That is, instead of approximating the gradient with a single example, use 100. Note, the expected gradient with respect to the random selection should be equal to the gradient found in part (a).

What to Submit

- **Part a:** Proof
- **Part b:** Separate plots for b(i) and b(ii).
- **Part c:** Separate plots for c which reproduce those from b(i) and b(ii) for this case.
- **Part d:** Separate plots for c which reproduce those from b(i) and b(ii) for this case.
- **Code** on Gradescope through coding submission.

Support Vector Machines

A3. 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.

What to Submit:

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

B1.

a. *[2 points]* Let $w^\top x + b = 0$ be the hyperplane generated by a certain SVM optimization. Given some point $x_0 \in \mathbb{R}^n$, Show that the minimum *squared distance* to the hyperplane is $\frac{|x_0^\top w+b|}{\|w\|_2}$. In other words, show the following:

$$\min_x \|x_0 - x\|_2 = \frac{|x_0^\top w + b|}{\|w\|_2}$$
$$\text{subject to: } w^\top x + b = 0\,.$$

Hint: Think about projecting x_0 onto the hyperplane.

b. *[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-b:** Solution and corresponding calculations.

Kernels

A4. *[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.)

a. 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:

- **Part a** Solution and corresponding calculations.

A5. 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:

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

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].

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).

c. *[5 points]* Repeat parts a and b with $n = 300$, but use 10-fold CV instead of leave-one-out for part a.

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.
- **Part c:** Values of d, γ, and the value of λ for both kernels as described. In addition, provide two separate plots as you did for part b.
- **Code** on Gradescope through coding submission.

Neural Networks for MNIST

For questions A.6 you will use a lot of PyTorch. **This assignment will take way longer time then previous coding problems, especially for those who are not familiar with PyTorch.** We advise that students might need to start early on this problem.

Resources

In Section materials (Week 7-8) there are iPython notebooks 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 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

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

[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 $\|x_i - x_j\|_2^2$.

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:

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

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:

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

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.

Intro to Sample Complexity

B2. For $i = 1, \ldots, n$ let $(x_i, y_i) \overset{\text{i.i.d.}}{\sim} P_{X,Y}$ where $y_i \in \{-1, 1\}$ and x_i lives in some set $\mathcal{X}$ (x_i is not necessarily a vector). The 0/1 loss, or *risk*, for a deterministic classifier $f\colon \mathcal{X} \to \{-1, 1\}$ is defined as:

$$R(f) = \mathbb{E}_{X,Y}[\mathbf{1}(f(X) \neq Y)]$$

where $\mathbf{1}(\mathcal{E})$ is the indicator function for the event $\mathcal{E}$ (the function takes the value 1 if $\mathcal{E}$ occurs and 0 otherwise). The expectation is with respect to the underlying distribution $P_{X,Y}$ on (X, Y). Unfortunately, we don't know $P_{X,Y}$ exactly, but we do have our i.i.d. samples $\{(x_i, y_i)\}_{i=1}^n$ drawn from it. Define the *empirical risk* as

$$\widehat{R}_n(f) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(f(x_i) \neq y_i) \ ,$$

which is just an empirical estimate of our risk. Suppose that a learning algorithm computes the empirical risk $R_n(f)$ for all $f \in \mathcal{F}$ and outputs the prediction function $\widehat{f}$ which is the one with the smallest empirical risk. (In this problem, we are assuming that $\mathcal{F}$ is finite.) Suppose that the best-in-class function f^* (i.e., the one that minimizes the true 0/1 loss) is:

$$f^* = \arg\min_{f \in \mathcal{F}} R(f) \, .$$

a. *[2 points]* Suppose that for some $f \in \mathcal{F}$, we have $R(f) > \epsilon$. Show that

$$\mathbb{P}\left[\widehat{R}_n(f) = 0\right] \leq \mathrm{e}^{-n\epsilon} \ .$$

(You may use the fact that $1 - \epsilon \leq \mathrm{e}^{-\epsilon}$.)

b. *[2 points]* Use the *union bound* to show that

$$\mathbb{P}\left[\exists f \in \mathcal{F} \text{ s.t. } R(f) > \epsilon \text{ and } \widehat{R}_n(f) = 0\right] \leq |\mathcal{F}|\mathrm{e}^{-\epsilon n} \ .$$

Recall that the union bound says that if $A_1, \ldots, A_k$ are events in a probability space, then

$$\mathbb{P}\left[A_1 \cup A_2 \cup \ldots \cup A_k\right] \leq \sum_{1 \leq i \leq k} \mathbb{P}(A_i).$$

c. *[2 points]* Solve for the minimum ϵ such that $|\mathcal{F}|\mathrm{e}^{-\epsilon n} \leq \delta$.

d. *[4 points]* Use this to show that with probability at least $1 - \delta$

$$\widehat{R}_n(\widehat{f}) = 0 \quad \implies \quad R(\widehat{f}) - R\left(f^*\right) \leq \frac{\log(|\mathcal{F}|/\delta)}{n}$$

where $\widehat{f} = \arg\min_{f \in \mathcal{F}} \widehat{R}_n(f)$.

Context: Note that among a larger number of functions $\mathcal{F}$ there is more likely to exist an $\widehat{f}$ such that $\widehat{R}_n(\widehat{f}) = 0$. However, this increased flexibility comes at the cost of a worse guarantee on the true error reflected in the larger $|\mathcal{F}|$. This trade-off quantifies how we can choose function classes $\mathcal{F}$ that over fit. This sample complexity result is remarkable because it depends just on the number of functions in $\mathcal{F}$, not what they look like. This is among the simplest results among a rich literature known as 'Statistical Learning Theory'. Using a similar strategy, one can use Hoeffding's inequality to obtain a generalization bound when $\widehat{R}_n(\widehat{f}) \neq 0$.

What to Submit:

- **Part a:** A proof that $\mathbb{P}\left[\widehat{R}_n(f) = 0\right] \leq \mathrm{e}^{-n\epsilon}$.
- **Part b:** A proof that $\mathbb{P}\left[\exists f \in \mathcal{F} \text{ s.t. } R(f) > \epsilon \text{ and } \widehat{R}_n(f) = 0\right] \leq |\mathcal{F}|\mathrm{e}^{-\epsilon n}$.
- **Part c:** A solution finding the minimum that satisfies the equation.
- **Part d:** A proof that $\widehat{R}_n(\widehat{f}) = 0 \quad \implies \quad R(\widehat{f}) - R\left(f^*\right) \leq \frac{\log(|\mathcal{F}|/\delta)}{n}$.