Homework #3

CSE 446/546: Machine Learning
Professors Matt Golub & Pang Wei Koh
Due: Nov 18, 2024 11:59pm
Points A: 82; B: 10

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 `https://www.gradescope.com/get_started#student-submission`.
- After submitting code, please ensure the autograded results match your expectations. We will not make accommodations for code submitted improperly. Refer to the README for documentation regarding code submissions.

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/cse446/24au/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: 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:

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

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].

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: $\lambda \in 10^{[-5,-1]}$ and $d \in [5, 25]$. You do **not** need to search over γ (you can use the heuristic given in the footnote), but if you would like to, a reasonable place to start would be to sample from a narrow gaussian distribution centered at the value described 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 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.

Intro to Sample Complexity

B1. 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 $1 - \epsilon \leq \mathrm{e}^{-\epsilon}$.)

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

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 $1 - \delta$

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

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

Introduction to PyTorch

Resources

For questions A.4 and A.5, you will use PyTorch. In Section materials (Week 6) there is a notebook that you might find useful. Additionally make use of PyTorch Documentation, when needed.

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. 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 hyper-parameter search[2]. Here we will also treat

[2]In this problem, the hyper-parameters required to search are (1) model architectures and (2) loss functions. Classic hyper-parameters like batch size and learning rates are not required to be searched over as long as the loss curve converges.

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 its 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.

The 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

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, 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

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

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:

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

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:

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

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 trained model 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 (c) in 1-2 sentences.
- **Code** on Gradescope through coding submission.

Administrative

A6.

a. *[2 points]* About how many hours did you spend on this homework? There is no right or wrong answer :)