Homework #3

CSEP 546: Machine Learning
Prof. Kevin Jamieson
Due: **Friday** March 3, 2023 11:59pm
90 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/csep546/23wi/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.
- We strongly encourage you to start early on this homework, as it will likely take you a good amount longer than previous ones.

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 neural networks requires minimizing a non-convex loss function, and therefore gradient descent might not reach the globally-optimal solution.

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

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

Kernels

A2. 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 $i = 1, \dots, n$ 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.

[1]Given a dataset $x_1, \dots, x_n \in \mathbb{R}^d$, 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. *[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]* We wish to build bootstrap percentile confidence intervals for $\widehat{f}_{\text{poly}}(x)$ and $\widehat{f}_{\text{rbf}}(x)$ for all $x \in [0, 1]$ from part b.[2] Use the non-parametric bootstrap with $B = 300$ bootstrap iterations to find 5% and 95% percentiles at each point x on a fine grid over $[0, 1]$.

   Specifically, for each bootstrap sample $b \in \{1, \ldots, B\}$, draw uniformly at randomly with replacement, n samples from $\{(x_i, y_i)\}_{i=1}^n$, train an $\widehat{f}_b$ using the b-th resampled dataset, compute $\widehat{f}_b(x)$ for each x in your fine grid; let the 5% percentile at point x be the largest value ν such that $\frac{1}{B}\sum_{b=1}^{B} \mathbf{1}\{\widehat{f}_b(x) \leq \nu\} \leq .05$, define the 95% percentile analogously.

   Plot the 5 and 95 percentile curves on the plots from part b. `plt.fill_between` is a good tool to use for plotting the interval.

d. *[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-c:** Two plots. One plot for each function which also includes its 5-95 percentile curves.
- **Part d:** 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. Note that there are some local tests but no private tests or autograder for this question.

Kaggle Competition and XGBoost

A3. This problem will be of a more open-ended nature compared to previous homeworks. You will explore an ML task in a more end-to-end fashion, get practice using popular packages, and even compete in a Kaggle competition. We strongly encourage you to read through the entire spec here before starting.

Background: You are a data scientist working for a real estate company. The company wants to improve its understanding of the housing market by predicting the sale price of homes based on various features.

Data: You have been given a dataset of homes that includes various information (e.g., number of rooms, square footage, number of bathrooms) and the aim is to predict `SalePrice`. The features include a mix of many *numerical* and *categorical* features. You will also compete amongst the other members of the class in a private Kaggle competition. To do so, you will train and cross-validate on the data in `train.csv` (which contains a `SalePrice` column that you should separate), make predictions on `test.csv` (where the `SalePrice` column is removed), and then submit the results as formatted in `sample_submission.csv` at this competition link. Note that you may only make a limited number of submissions in any given day.

Tasks: Your work can be broken down into the following steps

a. Inspect the features of the dataset and make any necessary modifications or transformations to make them more useful for the prediction task. The `pandas` library is great for this purpose. For example, we'd suggest reading in the data as a `DataFrame` object with `pandas.read_csv`, which then lets you more conveniently inspect features and perform potentially important operations such as:

[2]See Hastie, Tibshirani, Friedman Ch. 8.2 for a review of the bootstrap procedure.

- Find features that have missing values in **either** the train/test sets and remove them from **both** sets. You may find `DataFrame.isna()` or `DataFrame.dropna()` useful for this purpose. If you are especially ambitious, you could even consider imputing missing values!
- Scale numerical features when appropriate and/or address skew via a log-level transform.
- Replace categorical features with a "one-hot" representation across the various unique values, such as by using `pandas.get_dummies()`
- You may even consider engineering new features altogether, such as replacing discrete neighborhood features with their latitude and longitude (obtained from an external source).

Extra Resources: You may also find it useful to perform this data exploration more interactively by writing your code up in a Google Colab notebook. Also, if you want more guidance on data transformations, Chapter 3 of *Applied Predictive Modeling* by Kuhn and Johnson is a fairly good resource. It includes discussions of the various concepts mentioned above, such as: scaling (Ch 3.2), skew (Ch 3.2), missing values (in Ch 3.4), and dummy variables for categorical features (Ch 3.6). Another resource which may be helpful is this post, which touches upon most of the above concepts (besides skew).

b. Use linear regression to build a model based on your (possibly modified) features. You may also want to add regularization.

c. Use gradient boosting trees to build a second model. Specifically, you could use XGboost by installing `xgboost` via `pip` (i.e., by running `pip install xgboost`)and then importing `xgboost.XGBRegressor`. You should then also tune the hyperparameters of this model to achieve the best performance possible.

d. Compare the performance of the linear regression model and the XGboost model and provide an explanation of why one model may perform better than the other.

Evaluation: Your models will be evaluated based on their root mean squared error (RMSE) on the holdout test set on Kaggle. Technically, this RMSE will be computed after taking the logarithms of predicted and actual prices such that similar relative errors on cheap/expensive houses are punished similarly. When you submit your results, just leave them as the raw predicted prices without taking logs. However, you may find benefit performing regressions to directly predict log of `SalePrice`, before converting your test predictions back to raw prices for the final submission to Kaggle. **You will not receive any points off based on this performance as long as you achieve a RMSE of less than 0.145. However, the top three submissions will receive 10 extra credit points.**

What to Submit:

a. *[10 points]* Code for preprocessing the data, training the linear regression model, and training the XGboost model. Note: Please make sure to include appropriate comments in your code and to include a thorough explanation of your methods in your report. Note: this will not be autograded but please include this in your code submission: the usual zip for the other problems won't include this one but you can simply attach the files for this problem after uploading the zip on the same submission. Please also attach screenshots in your written pdf.

b. *[10 points]* A report that describes the preprocessing steps you took, the results of your feature inspection and modification, the hyperparameters you tuned for the XGboost model, and the performance of both models on the test set.

c. *[5 points]* A brief explanation of how you would use the best-performing model in practice. What concerns may you have?

Neural Networks for MNIST

For question A4 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

Here are some iPython notebooks that you might find useful for getting started. Additionally, we encourage you to make use of the official PyTorch Documentation, when needed.

If you do not have access to a 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.

A4. In Homework 2, we used logistic 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

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

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:

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

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 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. Note that there are no tests or autograder for this question.