University of Illinois at Urbana-Champaign
Computer Science

CS 498 DV, Fall 2021 **Problem Set 2** Sep 30, 2021

Deadline: This problem set is due by **11:59 pm on Oct 21, 2021**. You have 5 late days combined for both the problem sets. For example, if you submitted problem set 1 late by 2 days, you can get an automatic extension of 3 days for this problem set.

Solution: Please submit a pdf file containing the answers to the text-based questions and a jupyter notebook for the coding questions. You can also program in Matlab and submit a Matlab script instead of the jupyter notebook.

Gradescope: Please submit the solution on Gradescope. You can sign up using entry code: BPGJG5.

3-credit Version: If you are taking the 3-credit version, you can skip one of the problem sets. If you attempt both problem sets, you can use the best score across the two problem sets.

1 Linear Regression

We generated some simple datasets for you to warm-up to machine learning implementations. The dataset contains the inputs ($x \in \mathbb{R}$) in the file *input.npy* and the outputs ($y \in \mathbb{R}$) in file *output.npy*. Your goal is to learn increasingly complex regression functions that mimic the relationship between inputs and outputs. You can choose to use in-built python functions (e.g. in numpy or sklearn libraries) for linear regression in this problem.

1.1 Training and Test Distributions

[10 points] As we discussed in class, the training and test set should come from the same distribution. Let's see why that is important. Split your dataset such that all data points with $x < 0$ are in the training set and all points with $x > 0$ are in the test set. Use linear regression to fit a line mapping inputs x to outputs y in the training test.

(i) Plot your training set on a 2-D plot (check base.ipynb to see how to plot). Also, plot the line that you learnt using linear regression on the same plot. Does this line fit your training data well? What's your training loss (assume least square error)?

(ii) Plot your test set on a 2-D plot. Also, plot the line that you learnt using linear regression on your training set. Does this line fit your test data well? What's the test loss (least square error)?

(iii) Now, split the dataset into training and test data *randomly*. Fit a linear regression model to the training dataset. Repeat the steps in part (ii) to create a plot for the new test set and the line. Does this line fit better? If yes, why? If not, why not?

1.2 Kernels-I

[15 points] We will use the random split of training and test data for this problem as well. In class, we discussed that if your dataset doesn't have a linear relationship between outputs and inputs, linear regression doesn't do very well. However, we can use kernels to map your input data to higher dimensional spaces. Let's say we use the function Φ to define our kernel. Φ takes in an input value x and maps it to a two-dimensional output (x^2, x).

(i) Implement the function Φ defined above that takes in your input values (x from the dataset) and maps it to a 2-D feature space.

(ii) Use linear regression to train a new model to map $\Phi(x)$ to y.

(iii) Plot the test data on a 2-D plot – with x (not $\Phi(x)$) on x-axis and y on y-axis. Also, plot the curve you learnt using the linear regression on this plot. Does this curve fit better? If yes, why? If not, why not?

(iv) **Intercept/Bias:** Linear regression models are typically defined either using $y = wx$ or $y = wx + b$. The b term is the bias or the intercept. What's the impact of using (or not using) the intercept b in your model learnt in part (*iii*)? Specifically, repeat the plots in part (iii) with and without the intercept.

1.3 Kernels-II

[5 points] In the previous problem, we looked at a two-dimensional mapping function Φ to define a kernel. However, if the dimensions of the kernel increase, it is challenging (sometimes impossible) to compute $\Phi(x)$. In that case, we need to operate directly with the kernel matrix, K, where $K_{ij} = \Phi(x_i)^T \Phi(x_j)$ defines the similarity between training points x_i and x_j. Alex believes that you can use the matrix K to compute the distance between x_i and x_j in the kernel space. Specifically, Alex wants to compute the distance between $\Phi(x_i)$ and $\Phi(x_j)$ without explicitly computing the values of $\Phi(x_i)$ and $\Phi(x_j)$ and just using values in the matrix K. Do you think Alex is correct? If so, why? If not, why not?

2 Neural Networks

2.1 Perceptron

[10 points] Let's tweak the perceptron we learnt about in class a little bit. We are interested in the classification problem. We still have the linear weights in the first layer, but we approximate the non-linear sign function (in perceptron's original model) with the sigmoid function, σ, such that $\sigma(x) = \frac{1}{1+e^{-x}}$. Furthermore, we define our loss function l as the cross entropy loss. Specifically, $l(y_p, y_g) = -y_g \log(y_p) - (1 - y_g) \log(1 - y_p)$, where y_p are the predicted values, y_g are the ground truth values (and $y_g \in \{-1, 1\}$). This architecture is also defined in Fig. 1.

Now that we have defined this new model, let's think about how we can train this. As we learnt in class, we start with a random set of weights for the perceptron and use gradient descent at each step to update the weights (w_i's). For simplicity, assume you just have one training point. Compute the gradient of the loss function with respect to a weight w_i. Then, use the computed gradient to write the update step to update w_i.

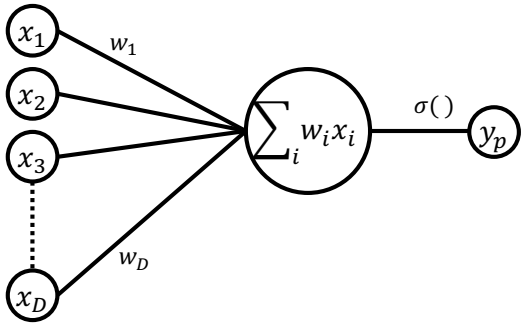

Figure 1: σ is the sigmoid function.

2.2 Backpropagation

[10 points] Neural networks are a powerful tool for modern Machine Learning. A key benefit of using neural networks is that they can learn complex decision boundaries and yet the gradient can be easily computed with respect to each weight parameter using backpropagation. We discussed what backpropagation is in class. This problem is just about making sure you practice that math. Derive the backpropagation formula (gradient update formula) assuming that all non-linear operations use the *sigmoid* operation. Specifically, $sigmoid(x) = \frac{1}{1+e^{-x}}$. Also, assume that we are interested in the regression problem, so the last layer doesn't use any non-linearity.