# CS480/680: Introduction to Machine Learning

Homework 4

Due: 11:59 pm, December 5, 2023, submit on CrowdMark/LEARN.

Submit your writeup in pdf and all source code in a zip file (with proper documentation). Write a script for each programming exercise so that the TAs can easily run and verify your results. Make sure your code runs!
[Text in square brackets are hints that can be ignored.]

## Exercise 1: VAEs and GANs (10 pts)

Code provided for this exercise is assuming a PyTorch solution, you may change it as necessary if you prefer to use TensorFlow or JAX. You may find this tutorial for GANs helpful.

a (4.5 pts) Complete the implementation of a VAE in vae.py. In your solution writeup, include the two produced graphs (corresponding to the training and test sets), with the epoch number on x-axis and the loss on the y-axis. Further include the produced samples from every 10th epoch (after epochs 10, 20, 30, 40, and 50).

Ans:

b (4.5 pts) Complete the implementation of a GAN in gan.py. In your solution writeup, include the two produced graphs (corresponding to the training and test sets), with the epoch number on x-axis and the losses (both generator and discriminator) on the y-axis. Further include the produced samples from every 10th epoch (after epochs 10, 20, 30, 40, and 50).

Ans:

c (1 pt) Describe, compare, and constrast your produced results and experiences with GANs and VAEs in this exercise.

Ans:

## Exercise 2: Robustness and Lasso (5 pts)

Consider the linear regression problem:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \|X\mathbf{w}-\mathbf{y}\|_2, \tag{1}$$

where $X \in \mathbb{R}^{n\times d}$, $\mathbf{y} \in \mathbb{R}^n$, $n$ is the number of training examples and $d$ is the number of features. For simplicity we omitted the bias. Now suppose we perturb each *feature* independently, and we are interested in solving the robust linear regression problem:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \max_{\forall j, \|\mathbf{z}_j\|_2\le\lambda} \|(X+Z)\mathbf{w}-\mathbf{y}\|_2, \tag{2}$$

where the perturbation matrix $Z = [\mathbf{z}_1, \ldots, \mathbf{z}_d] \in \mathbb{R}^{n\times d}$. Prove that robust linear regression is exactly equivalent to (square-root) Lasso (note the absence of the square on the $\ell_2$ norm):

$$\min_{\mathbf{w}\in\mathbb{R}^d} \|X\mathbf{w}-\mathbf{y}\|_2 + \lambda\|\mathbf{w}\|_1, \tag{3}$$

where recall that $\|\mathbf{w}\|_1 = \sum_j |w_j|$.

[Hint: Start from (2), apply the Cauchy-Schwarz inequality $\|\mathbf{w}\|_2 = \max_{\|\mathbf{u}\|_2\le 1} \mathbf{w}^\top\mathbf{u}$ a few times and a few other tricks, in order to convert it into (3).]

## Exercise 3: Adversarial examples (7 pts)

In this exercise we experiment with adversarial examples.

a) (1 pts) Train a neural network on MNIST. (The dataset can be downloaded with torchvision API for example.) You can build your own model or use an existing model architecture. Report the architecture

and the test accuracy (should be $> 90\%$).

[You are suggested to save this trained model for later purposes.]

[You can use any model architecture (e.g. CNN, MLP), but the test accuracy should be at least 90%.]

b) (2 pts) Use the Fast Gradient Sign Method (FGSM) to generate adversarial images for all the test images (one adversarial example per image) with $L_\infty$ perturbation budget $\epsilon = 0.2$ against your previous model, and report the model accuracy on this adversarially attacked set of test images. [The resulting value should be low.]

Display some of your adversarial images (e.g., randomly sample five) with their labels, and what do you observe?

Repeat the above with $\varepsilon = 0.1$ and $\varepsilon = 0.5$. What do you observe?

[Let $x$ denote the original image, and $\tilde{x}$ the perturbed image. $L_\infty$ perturbation with budget $\epsilon$ means $\|x - \tilde{x}\|_\infty \leq \epsilon$]

c) (2 pts) Do adversarial training: Initialize a new model with same architecture as in part a). During the training process, instead of using the original training data, generate adversarial examples (with an $L_\infty$ perturbation budget $\epsilon$ you prefer) with respect to the parameters at the current iteration and train on them instead. Use FGSM to generate these adversarial images.

After you adversarially trained the new model, repeat the instructions in part b) with budgets $\epsilon = 0.1$, 0.2, and 0.5 against your new model. Compare the results with those from part b), what do you observe by performing adversarial training?

[For adversarial training, feel free to generate and use more than one adversarial example per training image.]

d) Repeat the previous part, but perform adversarial training using Projected Gradient Descent (PGD) to generate adversarial examples. Afterwards, repeat the instructions in part b): generate FGSM adversarial examples against your new adversarially trained model with budgets $\epsilon = 0.1$, 0.2, and 0.5. Additionally, repeat, except using PGD adversarial examples (instead of FGSM) at the same budgets.

Make a table with three columns (for standard training, FGSM adversarial training, PGD adversarial training) and three rows (for standard test accuracy, robust test accuracy when attacked by FGSM, and robust test accuracy when attacked by PGD), and fill in all nine entries (including ones you haven't already computed). Comment on trends you observe.

You may either a) include three tables, one for each different $\epsilon$ budget for the attacker, or b) include just one table but pick the most interesting $\epsilon$.

e) (Optional: 0 points) Come up with a more robust model: that is, one which achieves a non-trivially higher accuracy when evaluated against PGD attacks. Do you believe that this is *truly* more robust? That is, can you come up with a better way of attacking your new method that causes the accuracy to drop lower again?

[The truth is that you are unlikely to come up with a truly and strongly robust model: in the research community, attacks are "winning" against the defenses. This is because of an asymmetry: for an attack to be good, it only needs to defeat one defense. But for a defense to be good, it must defeat all attacks.]

**Exercise 4: Membership Inference and Privacy (7 pts)**

In this problem, we will investigate membership inference attacks, and investigate whether one can defend against them.

a) (4 pts) Generate a dataset $X_1, \ldots, X_n \sim N(0, I)$, and compute the empirical mean $\bar{\mu} = \frac{1}{n}\sum_i X_i$. Suppose we have another dataset $Y_1, \ldots, Y_n \sim N(0, I)$. Our goal will be to perform membership inference: given $Z \in \cup_i \{X_i, Y_i\}$ and $\bar{\mu}$, the goal is to try to infer whether $Z$ is part of the training data (i.e., it is one of the $X_i$'s) or not (that is, it is one of the $Y_i$'s). We denote these two cases as IN and OUT. This is known as a *membership inference attack*, and being successful at this constitutes a privacy violation for the training data $X_i$'s.

For the remainder of this part, fix $n = 50$. We will repeat the same experiment for dimension $d$ ranging from 10 to 500.

Generate $X_1, \ldots, X_n, Y_1, \ldots, Y_n, \bar{\mu}$ as described above. Our algorithm will be: if $\langle Z, \bar{\mu} \rangle \geq T_d$ then predict IN, otherwise, predict OUT. $T_d$ is a threshold (which is a function of $d$). To determine $T_d$, write a function which takes in the $2n$ results of $\langle X_i, \bar{\mu} \rangle$ and $\langle Y_i, \bar{\mu} \rangle$ and chooses the threshold $T_d$ which maximizes the prediction accuracy of this test. Create a plot with dimension on the x-axis and accuracy of this membership inference attack on the y-axis.

Note that this is "cheating": we looked at the data itself and optimized this threshold $T_d$. But if we knew which elements were in and out of the set, we wouldn't need to run this algorithm. So let us see if the $T_d$'s we determined "generalize." Generate $X_1, \ldots, X_n, Y_1, \ldots, Y_n, \bar{\mu}$ freshly, and using the $T_d$ thresholds we generated previously, measure the effectiveness of this membership inference attack. Create a plot with dimension on the x-axis and accuracy of this membership inference attack on the y-axis.

We will now investigate an approach to protect against membership inference attacks. For the remainder of this part, fix $d = 50$. We will compute $X_1, \ldots, X_n, Y_1, \ldots, Y_n, \bar{\mu}$ as before, but additionally generate $\tilde{\mu} = \bar{\mu} + N(0, \sigma^2 I)$, where we will range $\sigma^2$ between 0 and 1. This will (roughly) inject differential privacy into the estimate $\tilde{\mu}$. As in the first part, for each value of $\sigma^2$, determine the threshold $U_{\sigma^2}$ which maximizes the accuracy of the following algorithm: if $\langle Z, \tilde{\mu} \rangle \geq U_{\sigma^2}$ then predict IN, otherwise, predict OUT. Create two plots. The first has $\sigma^2$ on the x-axis, and $\|\tilde{\mu}\|_2$ on the y-axis. This measures the accuracy of the estimate: the smaller, the better. The second has $\sigma^2$ on the x-axis and the membership inference attack accuracy on the y-axis (once again, the lower the better). When generating these plots, the values will have rather high variance: independently repeat the computations for each value of $\sigma^2$ several times (1000 may be reasonable) to get better estimates. [Note again that this example is cheating, but it gives an upper bound on how well this attack could work.]

Comment on all your findings as you present your results.

b) (3 pts) Now let's do membership inference on a machine learning model. We will use Fashion MNIST for this part, which can be loaded in similar ways that you have loaded MNIST in the past. We will take the first $n$ elements of the training set and train a logistic regression model on this subset. To balance things, we will also use only the first $n$ elements of the testing set. Run the following for $n = 100, 200, 400, 800, 1600, 2500, 5000, 10000$, and you may use logistic regression from sklearn.

First, train logistic regression with no regularization. Plot the training and testing accuracy on the y-axis, with $n$ on the x-axis.

Next, train logistic regression with $\ell_2$ regularization with parameter $C = 0.01$. Plot the training and testing accuracy on the y-axis, with $n$ on the x-axis.

Now, for both models, we will perform the following attack: if a point is classified correctly by the model, predict IN, otherwise, predict OUT. Measure the accuracy of this attack using the training data as the IN set, and the test data as the OUT set. Create a plot with $n$ on the x-axis, and the attack accuracy of both the regularized and unregularized models on the y-axis.

For the remainder of the problem, fix $n = 400$. Now, we will try to defend against membership inference attacks. Train two logistic regression models: one with no regularization, and one with $\ell_2$-regularization with paramger $C = 0.1$. Noise the resulting parameter vectors by adding $N(0, \sigma^2 I)$ noise, where we will vary $\sigma^2$ from 0 to 5. Create a plots for both models, with training and test accuracy on the y-axis and $\sigma^2$ on the x-axis. Run the same membership inference attack as before, and create a plot with $\sigma^2$ on the x-axis and the attack accuracy of both models on the y-axis.

Comment on all your findings as you present your results. Give the best explanation you can for why membership inference attacks happen in these settings.

c) (Optional, 0 pts) Play around with the logistic regression model. Are there other ways you can protect against this membership inference attack?