EN.601.475/675 Machine Learning

Homework 3: Due Wednesday, October 15th, 11:59pm

Submission Instructions: You will upload your solutions to the analytical portion and the programming portion separately to Gradescope.

- **Analytical Part (this file)**: You should use LaTeX to type up your solutions (e.g., via Overleaf or similar), but please ensure that **each part (e.g., a, b, etc) begins on a new page** (to facilitate grading in Gradescope). You will upload the PDF to the assignment "Homework 3: Analytical" on Gradescope.
- **Programming Part (see Canvas)**: You should upload your `.ipynb` file directly to Gradescope under the assignment "Homework 3: Programming".

Note that you must upload **both** parts to their respective assignments on Gradescope.

Collaboration Policy: You may work with other students on the homework assignments, but you must write up and submit your own work independently, and you must acknowledge your collaborators on the first page of your homework submission.

Late Submission Policy: You have up to 48 hours of "slack" for late submissions of homework assignments, across **all** homework assignments. Once you have exhausted your budget, we will deduct points for late submissions. However, **no individual assignment may be submitted more than 48 hours late**, as we will release solutions at this time.

Use of Generative AI: You are **not permitted** to use generative AI tools (e.g., ChatGPT) for problems on this problem set.

Name: _____

JHU Email: _____

Collaborators (or "None"): _____

The analytical portion (this file) has 4 questions, for a total of 65 points.

Part 1: Analytical Questions

Question 1 ..... *15 points*

In this question, we will apply the algorithm used to train AdaBoost on a binary classification problem with "decision stumps" (single split decision trees) as weak learners. The labels are $y \in \{+1, -1\}$. The three decision stumps we will use are defined as follows, using the indicator notation:

$$h_1(x) = \mathbf{1}\{\texttt{free} = 1\}, \quad h_2(x) = \mathbf{1}\{\texttt{money} = 1\}, \quad h_3(x) = \mathbf{1}\{\texttt{win} = 1\},$$

where (for the purposes of this problem) $\mathbf{1}\{\cdot\}$ returns $+1$ if the condition is true and -1 otherwise. We have the following dataset of $n = 8$ emails, where the columns represent specific words found in an email, and the $+1$ label represents "spam," -1 label represents "not spam":

i	`free`	`money`	`win`	y
1	1	0	0	+1
2	0	1	0	+1
3	0	0	1	−1
4	1	1	0	+1
5	0	0	0	−1
6	0	1	1	+1
7	1	0	1	+1
8	0	0	1	+1

Weights initialize as $D_1(i) = 1/8$. Recall that at round t we have:

$$\varepsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i], \quad \alpha_t = \tfrac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t} \quad D_{t+1}(i) \propto D_t(i) \exp(-\alpha_t y_i h_t(x_i)).$$

(a) (5 points) For each weak learner h_j, compute $\varepsilon_1(h_j) = \Pr_{i \sim D_1}[h_j(x_i) \neq y_i]$. Which of the three is the best weak learner? Using the best weak learner, compute α_1 and D_2.

Solution:

(b) (5 points) Explain how AdaBoost's reweighting helps when rare positive (+1) emails are initially misclassified by obvious word stumps like h1 (free).

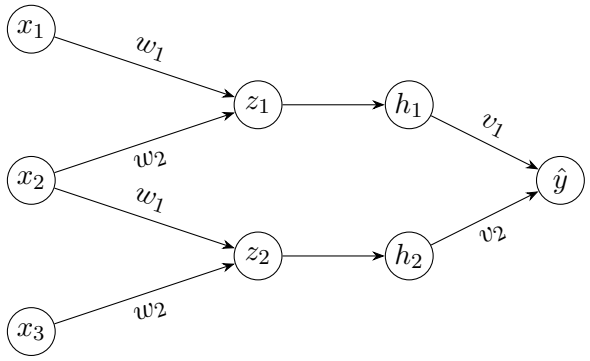
Solution:

(c) (5 points) Explain how AdaBoost can incorporate a learner that does worse than random, in other words a learner for which $\varepsilon_t > 0.5$.

Solution:

Question 2 ..... *15 points*

Consider the following mini convolutional neural net, where (x_1, x_2, x_3) is the input, followed by a convolution layer with a filter (w_1, w_2), a ReLU layer, and a fully connected layer with weight (v_1, v_2).



More concretely, the computation is specified by

$$z_1 = x_1 w_1 + x_2 w_2, \quad z_2 = x_2 w_1 + x_3 w_2,$$

$$h_1 = \max\{0, z_1\}, \quad h_2 = \max\{0, z_2\},$$

$$\hat{y} = h_1 v_1 + h_2 v_2.$$

We assume the true labels y are -1 or 1, so that $(x, y) \in \mathbb{R}^3 \times \{-1, +1\}$.

The logistic loss CNN can then be written as follows

$$\ell(y, \hat{y}) = \ln(1 + \exp(-y\hat{y})),$$

which is a function of the parameters of the network: w_1, w_2, v_1, v_2.

(a) (5 points) Write down $\frac{\partial \ell}{\partial v_1}$ and $\frac{\partial \ell}{\partial v_2}$ (show the intermediate steps that use chain rule). You can use the sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$ to simplify your notation.

Solution:

(b) (5 points) Write down $\frac{\partial \ell}{\partial w_1}$ and $\frac{\partial \ell}{\partial w_2}$ (show the intermediate steps that use chain rule). The derivative of the ReLU function is $H(a) = \mathbf{1}[a > 0]$, which you can use directly in your answer. In particular (by convention), we take the gradient of the ReLU function to be zero when $a = 0$.[1]

Solution:

[1] While the ReLU function is not differentiable at $a = 0$, zero is a *subgradient*, a generalization of the gradient to convex non-differentiable functions.

(c) (3 points) We often initialize the weights of neural networks randomly. Suppose we ignored this practice, and instead we initialized all of the parameters v_1, v_2, w_1, w_2 as zero. What is the value of $z_1, z_2, h_1, h_2, \hat{y}$ with this initialization? What happens to the gradient in this case? In conclusion, why is this a poor choice of initialization?

Solution:

(d) (2 points) Using the derivations above, fill in the missing details of the backpropagation algorithm below that is used to train this mini CNN.

Algorithm 1 Backpropagation for the above mini CNN

Input: A training set $(x_1, y_1), \ldots, (x_n, y_n)$
Initialize: set w_1, w_2, v_1, v_2 randomly
Randomly pick an example (x_i, y_i)
Forward propagation:

Solution: compute

$$z_1 = \text{[to fill in]}, \quad z_2 = \text{[to fill in]},$$
$$h_1 = \text{[to fill in]}, \quad h_2 = \text{[to fill in]}, \quad \hat{y} = \text{[to fill in]}$$

Backward propagation:

Solution:
compute

$$g_{w_1} = \text{[to fill in]},$$
$$g_{w_2} = \text{[to fill in]},$$
$$g_{v_1} = \text{[to fill in]}, \qquad g_{v_2} = \text{[to fill in]}.$$

Output: [to fill in]

Question 3 ..... *15 points*

Dropout Regularization and Variance

One approach to regularization in a neural network model is dropout. During training, dropout is applied independently to each unit (either inputs x_i, or hidden units h_{li}), setting them to zero with probability p, and otherwise scaling them by $1/(1-p)$. This is only applied during model training: during final model evaluation, the units are unchanged.

(a) (5 points) We will consider a simple linear model $\hat{y} = w^\top x$, with dropout applied to the inputs x. To start, derive the expectation and covariance matrix of the effective input vector $\tilde{x}$ when dropout is applied. Treat the inputs x as fixed, so that dropout is the only source of randomness (and hence the only source of randomness to consider in computing expectation and covariance).

Solution:

(b) (7 points) Again, treating the data y, x and the weights w as fixed, show that the expected squared loss under dropout (i.e., during training) can be written as the original squared loss plus an extra penalty on the weights.

Solution:

(c) (3 points) Interpret the penalty in (b). How is it related to L_2 (ridge) regularization? When does it differ?

Solution:

Question 4 ..... *20 points*

Bagging (Bootstrap Aggregating) is a technique designed to reduce the variance of a learning algorithm. Consider a regression setting where the true function output is $f(\mathbf{x})$, and a single base learner trained on a specific bootstrap sample predicts $\hat{f}(\mathbf{x})$.

Assume the following properties for the distribution of base learner predictions $\hat{f}(\mathbf{x})$:

- **Unbiasedness:** The expected prediction equals the true function output: $E[\hat{f}(\mathbf{x})] = f(\mathbf{x})$.
- **Variance:** The variance of the prediction is $\text{Var}[\hat{f}(\mathbf{x})] = \sigma^2$.
- **Covariance:** The covariance between the predictions of any two *different* base learners, i and j, is $\text{Cov}[\hat{f}_i(\mathbf{x}), \hat{f}_j(\mathbf{x})] = \rho\sigma^2$, where $\rho \in [0, 1]$ is the correlation coefficient.

The bagging ensemble uses M base learners, and its final prediction $\hat{F}(\mathbf{x})$ is their average:

$$\hat{F}(\mathbf{x}) = \frac{1}{M} \sum_{i=1}^{M} \hat{f}_i(\mathbf{x})$$

(a) (5 points) Calculate the expected prediction $E[\hat{F}(\mathbf{x})]$ for the Bagging ensemble. Use the expected prediction to determine the bias of the ensemble, $\text{Bias}[\hat{F}(\mathbf{x})] = E[\hat{F}(\mathbf{x})] - f(\mathbf{x})$.

Solution:

(b) (10 points) Calculate the variance of the Bagging ensemble's prediction, $\text{Var}[\hat{F}(\mathbf{x})]$. Your derivation should clearly show the sum of the M variance terms and the sum of the $M(M-1)$ covariance terms. Show that the final result can be expressed in the form:

$$\text{Var}[\hat{F}(\mathbf{x})] = \frac{\sigma^2}{M} + \rho\sigma^2\left(1 - \frac{1}{M}\right)$$

Hint: $\text{Var}\left[\sum_{i=1}^{M} X_i\right] = \sum_{i=1}^{M} \text{Var}[X_i] + \sum_{i \neq j} \text{Cov}[X_i, X_j]$.

Solution:

(c) (5 points) Analyze the ensemble's variance as the number of learners M becomes very large ($M \to \infty$).

1. If the base learners were **perfectly uncorrelated** ($\rho = 0$), what would $\text{Var}[\hat{F}(\mathbf{x})]$ converge to?
2. If the base learners were **perfectly correlated** ($\rho = 1$), what would $\text{Var}[\hat{F}(\mathbf{x})]$ converge to?

In one to two lines, explain what these limiting results reveal about *why randomness is required* for Bagging to be successful in practice.

Solution:

Part 2: Programming Questions

Please see Canvas for the link to the Programming Assignment