CS 446 / ECE 449 — Homework 3

your NetID here

Version 1.1

Instructions.

- Homework is due **Thursday, March 11, at noon CST**; no late homework accepted.
- Everyone must submit individually on Gradescope under `hw3` and `hw3code`.
- The "written" submission at `hw3` **must be typed**, and submitted in any format Gradescope accepts (to be safe, submit a PDF). You may use LaTeX, markdown, Google Docs, MS word, whatever you like; but it must be typed!
- When submitting at `hw3`, Gradescope will ask you to mark out boxes around each of your answers; please do this precisely!
- Please make sure your NetID is clear and large on the first page of the homework.
- Your solution **must** be written in your own words. Please see the course webpage for full academic integrity information. Briefly, you may have high-level discussions with at most 3 classmates, whose NetIDs you should place on the first page of your solutions, and you should cite any external reference you use; despite all this, your solution must be written in your own words.
- We reserve the right to reduce the auto-graded score for `hw3code` if we detect funny business (e.g., your solution lacks any algorithm and hard-codes answers you obtained from someone else, or simply via trial-and-error with the autograder).
- The list of library routines with the coding problems are only suggestive.
- When submitting to `hw3code`, upload `hw3.py` and `hw3_utils.py`.

Version History.

1. Initial version.
2. Corrected typo in Problem 3.

1. ResNet.

In this problem, you will implement a simplified ResNet. You do not need to change arguments which are not mentioned here (but you of course could try and see what happens).

(a) Implement a class `Block`, which is a building block of ResNet. It is described in (He et al., 2016) Figure 2.

The input to `Block` is of shape (N, C, H, W), where N denotes the batch size, C denotes the number of channels, and H and W are the height and width of each channel. For each data example $\boldsymbol{x}$ with shape (C, H, W), the output of `block` is

$$\texttt{Block}(\boldsymbol{x}) = \sigma_r\left(\boldsymbol{x} + f(\boldsymbol{x})\right),$$

where σ_r denotes the ReLU activation, and $f(\boldsymbol{x})$ also has shape (C, H, W) and thus can be added to $\boldsymbol{x}$. In detail, f contains the following layers.

i. A `Conv2d` with C input channels, C output channels, kernel size 3, stride 1, padding 1, and no bias term.
ii. A `BatchNorm2d` with C features.
iii. A ReLU layer.
iv. Another `Conv2d` with the same arguments as i above.
v. Another `BatchNorm2d` with C features.

Because 3×3 kernels and padding 1 are used, the convolutional layers do not change the shape of each channel. Moreover, the number of channels are also kept unchanged. Therefore $f(\boldsymbol{x})$ does have the same shape as $\boldsymbol{x}$.

Additional instructions are given in doscstrings in `hw3.py`.

Suggested Library routines: `torch.nn.Conv2d and torch.nn.BatchNorm2d.`

(b) Explain why a `Conv2d` layer does not need a bias term if it is followed by a `BatchNorm2d` layer.

(c) Implement a (shallow) `ResNet` consists of the following parts:

i. A `Conv2d` with 1 input channel, C output channels, kernel size 3, stride 2, padding 1, and no bias term.
ii. A `BatchNorm2d` with C features.
iii. A ReLU layer.
iv. A `MaxPool2d` with kernel size 2.
v. A `Block` with C channels.
vi. An `AdaptiveAvgPool2d` which for each channel takes the average of all elements.
vii. A `Linear` with C inputs and 10 outputs.

Additional instructions are given in doscstrings in `hw3.py`.

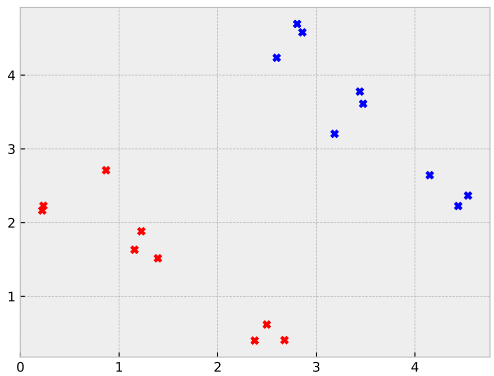
Suggested Library routines: `torch.nn.Conv2d, torch.nn.BatchNorm2d, torch.nn.MaxPool2D, torch.nn.AdaptiveAvgPool2d and torch.nn.Linear.`

Remark: The `epoch_loss` and `fit_and_evaluate` routines of `hw2` can be used to train your ResNet model. It is not required for you to do so for the purpose of this problem.

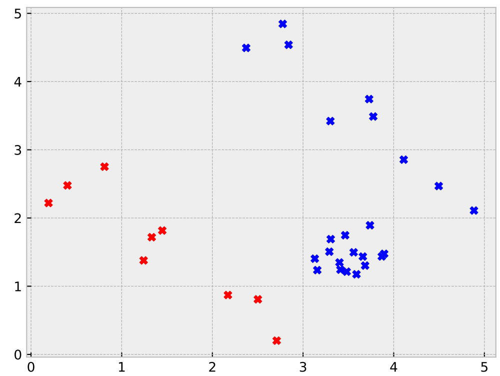
Solution.

2. Decision Trees.

Consider the training and testing data sets as given in Figures 1a and 1b for the sub-parts (a)-(c). For the sub-parts (d) & (e), refer to the training and testing data sets as given in Figures 2a and 2b.



(a) Fig 1a: Training data set 1.

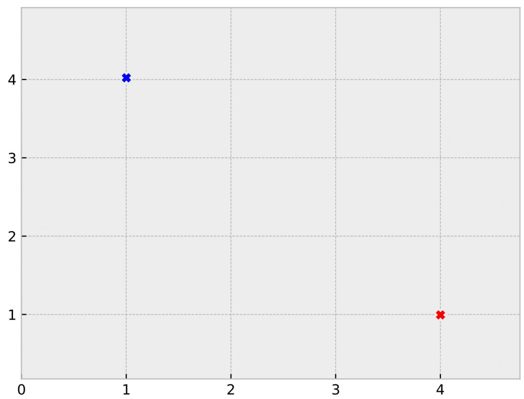


(b) Fig 1b: Testing data set 1.

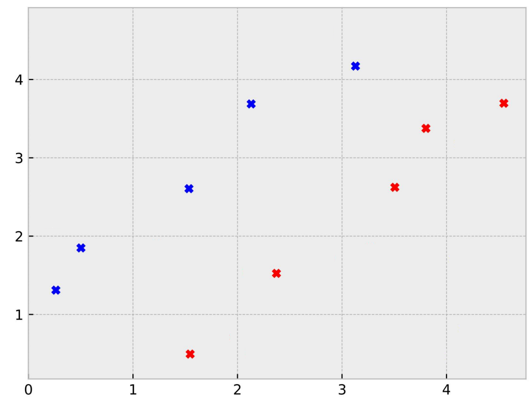
(a) Describe a decision tree of depth one with integral and axis-aligned decision boundaries which achieves error at most $\frac{1}{6}$ on training data set 1 (Figure 1a).

(b) Describe a decision tree (of any depth) with integral and axis-aligned decision boundaries which achieves zero error on training data set 1 (Figure 1a).

(c) Describe a decision tree (of any depth) with integral and axis-aligned decision boundaries which achieves zero error on training data set 1 (Figure 1a) but has error at least $\frac{1}{4}$ on testing data set 1 (Figure 1b).



(a) Fig 2a: Training data set 2.



(b) Fig 2b: Testing data set 2.

(d) Describe a decision tree with integral and axis-aligned decision boundaries with at most two splits, which achieves zero error on training data set 2 and calculate its error on testing data set 2.

(e) Construct a 1-nn classifier using training data set 2 and state its error on testing data set 2.

Remark: Note that *every* decision tree (axis aligned integral splits) with at most two splits has positive test error in this problem. Therefore one nearest neighbor is a better choice here.

Solution.

3. Nearest Neighbor.

(a) Implement the 1-nearest neighbor algorithm in the `one_nearest_neighbor()` function in `hw3.py`. In the starter code you are given three torch tensors as input:

- `X` - training set
- `Y` - training labels
- `X_test` - testing set

Use the training set to determine labels for the testing set. Return the labels for the testing set as determined by your nearest neighbor implementation.

(b) Plot the Voronoi diagram of your nearest neighbor results. Use the data set returned from `load_one_nearest_neighbor_data()` in `hw3_utils.py`. You may use the function `voronoi_plot()` provided to you in `hw3_utils.py` to help generate the diagram. There is no need to submit code for this part, only submit the plots in the written portion.

Solution.

4. Robustness of the Majority Vote Classifier.

The purpose of this problem is to further investigate the behavior of the majority vote classifier (*Slide 5, Lecture 12*) using Hoeffding's inequality (*Slide 20, Lecture 13; will be included in Lecture 12*). Simplified versions of Hoeffding's inequality are as follows.

Theorem 1. *Given independent random variables $(Z_1, \ldots, Z_k)$ with $Z_i \in [0, 1]$,*

$$\Pr\left[\sum_{i=1}^{k} Z_i \geq \sum_{i=1}^{k} \mathbb{E}[Z_i] + k\epsilon\right] \leq \exp\left(-2k\epsilon^2\right), \tag{1}$$

and

$$\Pr\left[\sum_{i=1}^{k} Z_i \leq \sum_{i=1}^{k} \mathbb{E}[Z_i] - k\epsilon\right] \leq \exp\left(-2k\epsilon^2\right). \tag{2}$$

In this problem we have an odd number n of classifiers $(f_1, \ldots, f_n)$ and only consider their behavior on a fixed data example (x, y); by classifier we mean $f_i(x) \in \{\pm 1\}$. Define the majority vote classifer $\textsc{Maj}$ as

$$\textsc{Maj}(x) := 2 \cdot \mathbb{1}\left[\sum_{i=1}^{n} f_i(x) \geq 0\right] - 1 = \begin{cases} +1 & \sum_{i=1}^{n} f_i(x) > 0, \\ -1 & \sum_{i=1}^{n} f_i(x) < 0, \end{cases}$$

where we will not need to worry about ties since n is odd.

To demonstrate the utility of Theorem 1 in analyzing $\textsc{Maj}$, suppose that $\Pr[f_i(x) = y] = p > 1/2$ independently for each i. Then, by defining a random variable $Z_i := \mathbb{1}[f_i(x) \neq y]$ and noting $\mathbb{E}Z_i = 1 - p$,

$$\begin{aligned}
\Pr[\textsc{Maj}(x) \neq y] &= \Pr\left[\sum_{i=1}^{n} \mathbb{1}[f_i(x) \neq y] \geq \frac{n}{2}\right] \\
&= \Pr\left[\sum_{i=1}^{n} Z_i \geq n(1-p) - \frac{n}{2} + np\right] \\
&= \Pr\left[\sum_{i=1}^{n} Z_i \geq n\mathbb{E}Z_1 + n(p - 1/2)\right] \\
&\leq \exp\left(-2n(p - 1/2)^2\right).
\end{aligned}$$

The purpose of this problem is to study the behavior of $\textsc{Maj}(x)$ when not all of the classifiers $(f_1, \ldots, f_n)$ are independent.

(a) Assume n is divisible by 6 and $5n/6$ is odd, and that of the n classifiers $(f_1, \ldots, f_n)$, now only $5n/6$ of them have independent errors on x, specifically $\Pr[f_i(x) = y] = p := 4/5$ for $5n/6$ of the classifiers. By contrast, make no assumption on the other $n/6$ classifiers and their errors. Now use Hoeffding's inequality to show that the majority vote classifier over all n classifiers is still good, specifically showing

$$\Pr\left[\textsc{Maj}(x) \neq y\right] \leq \exp(-n/15).$$

Remark: This problem shows that even with corruption levels more than 16%, the robustness of the majority vote classifier allows for a reasonable probability of correctness for sufficiently large n.

Hint: Use Hoeffding's inequality (Eq (1)) on the $5n/6$ classifiers which have independent errors. However, in this case the majority vote can be incorrect when less than half of the $\frac{5n}{6}$ (independent) classifiers are in error, due to the arbitrary behavior of the remaining $\frac{n}{6}$ classifiers. A good point to start is by modifying the calculations for $\Pr[\textsc{Maj}(x) \neq y]$ in the preamble of this problem.

For full points: You need to derive the inequality $\Pr\left[\textsc{Maj}(x) \neq y\right] \leq \exp(-n/15)$ rigorously for ANY possible behavior of the $\frac{n}{6}$ arbitrary classifiers.

(b) Suppose again that n is divisible by 5 and $3n/5$ is odd, but now that only $3n/5$ of the classifiers have independent errors, and are correct with probability $\Pr[f_i(x) = y] = p := 2/3$. Describe malicious behavior for the remaining $2n/5$ classifiers so that

$$\Pr\left[\textsc{Maj}(x) = y\right] \leq \exp(-n/30).$$

Remark: This problem shows that the performance of the majority vote classifier degrades significantly with higher corruption levels and increased probability of error.

Hint: This may require the use of "reverse" form of Hoeffding's inequality, i.e., Eq (2) of Theorem 1. First, specify the malicious behavior of the $\frac{2n}{5}$ arbitrary classifiers. Consider various deterministic choices for the malicious behavior as potential candidates.

For full points: Describe the malicious behavior of the arbitrary classifiers AND derive the inequality $\Pr\left[\textsc{Maj}(x) = y\right] \leq \exp(-n/30)$.

Solution.

References

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.