EMORY UNIVERSITY
Department of Computer Science
CS 334 Section 2 — Machine Learning
Fall 2024

Homework 4, Issued: Fri. 10/25, Due: Fri. 11/8 at 11:59pm

Submission Instructions: The homework is due on Gradescope in **one** part.

- **Upload PDF to HW4-Written**: Create a single high-quality PDF with your answers to the non-coding problems. Your submission may be typed or handwritten (can be scanned or from a note-taking software), but the pages must be tagged with the relevant questions appropriately on Gradescope. Code is not autograded but you should attach the function you implemented in the write-up. Please also include *a **SIGNED** honor code statement* that reads as follows:

```
THIS CODE IS MY OWN WORK, IT WAS WRITTEN WITHOUT CONSULTING CODE
WRITTEN BY OTHER STUDENTS OR LARGE LANGUAGE MODELS SUCH AS CHATGPT.
/* Your_Name_Here */
I collaborated with the following classmates for this homework:
<names of classmates>
```

Note on programming part of this assignment: For Q3, you should only need $<$ 15 lines of code, but running the code will take about 2 minutes.

1 Kernels [15 pts]

Let's kernelize the perceptron algorithm (shown below). We'll map each example $\vec{x}$ to a feature vector $\phi(\vec{x})$, and then reformulate the algorithm such that it only uses the kernel function $K(\vec{x}, \vec{x}')$ corresponding to $\phi(\vec{x}) \cdot \phi(\vec{x}')$ without explicitly computing any $\phi(\vec{x})$.

```
1 Initialization: $\vec{\theta} = \vec{0}$
2 Repeat until convergence:
3     for $i = 1, \ldots, N$:
4         if $y^{(i)}(\vec{\theta} \cdot \vec{x}^{(i)}) \leq 0$ then:
5             Update $\vec{\theta} \leftarrow \vec{\theta} + y^{(i)}\vec{x}^{(i)}$
6 Classification function of a test point $\vec{x}$:
7 $h(\vec{x}) = \text{sign}(\vec{\theta} \cdot \vec{x})$
```

```
1 Initialization: ________
2 Repeat until convergence:
3     for $i = 1, \ldots, N$:
4         if ____________ then:
5             Update ____________
6 Classification function of a test point $\vec{x}$:
7 $h(\vec{x}) = \text{sign}(\_\_\_\_\_\_\_\_\_\_\_\_)$
```

(a) (5 pts) First, let's think about how we might update the classifier function $h(\vec{x}; \vec{\theta})$ (line 7). Since $\phi(\vec{x})$ (and by extention, the $\vec{\theta}$ after being kernelized) may lie in a possibly infinite dimensional space, we need to find a new way of defining h without explicitly using $\vec{\theta}$. Suppose each $\vec{x}^{(i)}$ is misclassified α_i times during training. Given $K(\vec{x}^{(i)}, \vec{x})$ and α_i for each i, and the training data $\{\vec{x}^{(i)}, y^{(i)}\}$ **how would you classify the test point $\vec{x}$?** Hint: Review HW2 problem 2(a).

(b) (5 pts) Now, let's consider the update (line 5) which updates the parameters of our model. Again, we can't update $\vec{\theta}$ explicitly. Based on your answer for part (a), **how should we update the parameters?**

(c) (5 pts) Finally, **complete** the kernelized perceptron algorithm by updating steps (1), (4), (5), (7).

2 Decision Trees [18 pts]

You are tasked with learning a decision tree to predict which NBA teams are going to make the playoffs, based on data collected over the first half of the regular season. You collected data regarding three point percentage (3P%), steals per game (SPG), and rebounds per game (RPG) for a subset of teams shown here:

3P%>0.23	SPG≥7.8	RPG>44	Made Playoffs
F	T	T	YES
F	T	F	NO
F	F	T	NO
F	F	F	NO
T	F	F	YES
F	T	T	YES
T	T	T	YES
F	F	F	NO

(a) [3 pts] Calculate the entropy of the dataset, $H(Y)$, where Y is the outcome.

(b) [5 pts] Calculate the conditional entropy, $H(Y|X)$ for each feature X in (3P%>0.23, SPG≥7.8, RPG>44). Which feature(s) has the highest conditional entropy?

(c) [5 pts] Calculate the information gain, $IG(X, Y)$ for each feature with respect to the outcome. Which feature(s) result in the highest information gain?

(d) [5 pts] Construct (however you want) a decision tree with no more than 4 leaf nodes that achieves zero training error on this dataset.

3 Ensemble Methods [18 pts]

Let's compare the performance of two ensemble methods, random forest vs bagging with decision trees. We'll use a subset of the MNIST dataset, which contains examples of handwritten digits classified into 10 classes (0 through 9). Here, we will consider a downsampled version (14×14 image having 196 features) pertaining to the three digits 1, 3, 5. The skeleton code can be found in `ensemble.py`.

(a) (8 pts) **Implement** `random_forest(X_train, y_train, X_test, y_test, m, n_clf)`. Random forest consists of `n_clf`-many decision trees where each tree is trained independently on a bootstrap sample of the training data. For each node, a subset of m features are randomly selected as candidates for splitting on. Here, the final prediction is determined by a majority vote of these `n_clf` decision trees. In case of ties, randomly sample among the plurality classes (i.e. the classes that are tied). You should use `sklearn.tree.DecisionTreeClassifier` with `criterion='entropy'` and pass in `max_features`. Do not set `max_depth`.

(b) (0 pt) Review `bagging_ensemble(X_train, y_train, X_test, y_test, n_clf)` which we've already implemented for you. This function calls `random_forest()` but passes in `max_features=None` to consider all features at every split of the decision tree.

(c) (4pts plot + 6pts write-up) Now, we will compare the performance of these two ensembling approaches. **Run** the script which measures the average performance across 50 random splits of the dataset into training (80%) and test (20%) sets. **Attach the generated plot** of classifier accuracy over the range of m values (the size of the randomly selected subset of features) specified in the skeleton code. **How** does the generalization performance of the two methods compare as m varies?

4 Boosting [25 pts]

Consider the labeled training data points shown below, where × is positive and ● is negative. We will apply Adaboost with decision stumps to solve the classification problem. In each boosting iteration, we select the stump that minimizes the weighted training error, breaking ties arbitrarily.

Optional: we have provided `Adaboost.py` so that you can recreate the plot yourself and/or implement the Adaboost algorithm to check your answers.

(a) (5 pts) **Draw** the decision boundary corresponding to the first decision stump that the boosting algorithm would choose. **Label** this boundary as `(1)`, and also **indicate** the `+/-` sides of the decision boundary.

(b) (5 pts) **Circle** the point(s) that have the largest weight after the first boosting iteration.

(c) (5 pts) **What is the weighted error** of the first decision stump after the first boosting iteration, i.e., after the points have been re-weighted?

(d) (5 pts) **Draw** the decision boundary corresponding to the second decision stump, again in Figure 1, and **label** it as `(2)` also **indicating** the `+/-` sides of the boundary.

(e) (5 pts) Will any of the points be misclassified by the combined classifier after the two boosting iterations? **Provide** a brief justification. You may calculate the ensemble predictions or otherwise reason about the ensemble classifier's decision boundaries.

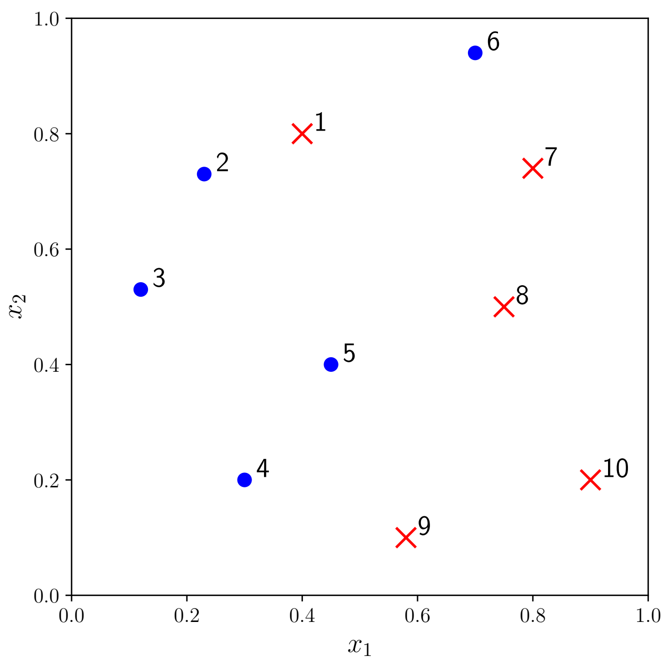
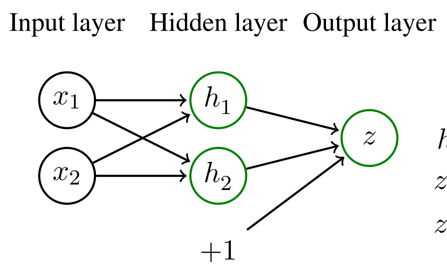


Figure 1: ●-negative points and ×-positive points.

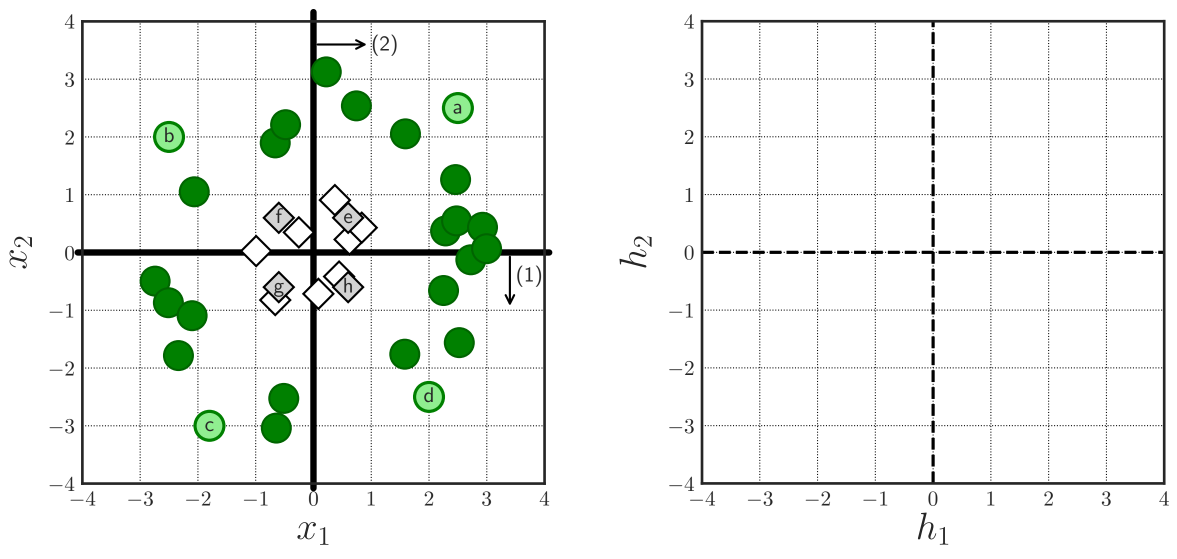
5 Neural Networks [24 pts]

Consider a 2-layer feed-forward neural network that takes as input two-dimensional examples $\vec{x} = [x_1, x_2]^T$ and has two ReLU hidden units.



$$h(\vec{x};\vec{\theta}) = z = V_1 h_1 + V_2 h_2 + V_0$$
$$z_1 = W_{11}x_1 + W_{21}x_2, \quad h_1 = \max\{0, z_1\}$$
$$z_2 = W_{12}x_1 + W_{22}x_2, \quad h_2 = \max\{0, z_2\}$$

(a) [8 pts] The weights in the hidden layers, W_{ij}, are set such that they result in the z_1 and z_2 "classifiers" given in figure (A) below by the "decision boundaries" and the corresponding normal vectors marked (1) and (2). Assume that $W_{11}^2 + W_{21}^2 = 1$ and $W_{12}^2 + W_{22}^2 = 1$. In figure (B) on the right, sketch how the input data (only points 'a' through 'h') map to the 2-dimensional space of hidden unit activations h_1 and h_2. Be sure to label the mapped points 'a' through 'h'.



(A) (B)

(b) [6 pts] Suppose we are solving a binary classification task, and the goal is to separate the ◇'s (negative) from the ●'s (positive) in figure (A). If we keep the hidden layer parameters W_{ij}'s fixed, and train the output layer parameters V_j's, **what** is the best (minimum) misclassification error rate we can achieve on points 'a' through 'h'? **Draw** a possible "best" decision boundary in figure (B).

(c) [5 pts] Now, suppose we keep the hidden layer parameters W_{ij}'s fixed, but instead of the output layer, we add and train additional hidden layers (applied after this layer) to further transform the data. Could the resulting neural network perfectly classify all points? Justify your answer.

(d) [5 pts] Supposed we stick to the current 2-layer architecture but add one more ReLU hidden unit. Given this new architecture with three hidden ReLU units, is it possible to learn the weights that perfectly separates all points in figure (a)? If so, provide an example. Otherwise, explain why it is not possible.