CMU 10-715: Final Exam

DUE: Dec. 16, 2020, 11:40 AM Eastern Standard Time.

Instructions:

- It is a 24-hour exam. The submission site will close at 11:40 AM, Dec 16.
- You can refer to your own notes, the scribe notes, homework solutions, the text book, and lecture videos. You are **NOT** allowed to use any other resources (e.g., no searching on the Internet).
- You are **NOT** allowed to discuss the exam with anyone in the duration of the exam. For any questions, please post on Diderot.
- For the programming question(s), you can feel free to use any code you wrote for any of the homeworks.
- Submit your solutions in a pdf file to Gradescope. Handwritten or typed solutions are both accepted. Start **each question on a new** page and make sure you select the corresponding page(s) for each question during submission to Gradescope.
- We recommend taking a brief look at all questions first and then starting to answer them. You may find some questions easier than others. Good luck!

Distribution of Marks

Question	Points	Score
1	10	
2	10	
3	10	
4	10	
5	10	
6	10	
7	10	
Total:	70	

1) Neural Networks and Circuit Building Blocks [10]

In a lecture you saw that neural networks are very expressive, for instance, they can represent all functions from $\{-1,1\}^d$ to $\{-1,1\}$. In this question you will show that basic building blocks of circuits – AND, OR, NOT functions – can be represented via neural networks.

Consider $\mathcal{X} = \{-1,1\}^2$ and $\mathcal{Y} = \{-1,1\}$. (Please take care to note that these values are in $\{-1,1\}$ and not in $\{0,1\}$.) Consider the following three functions: AND, OR and NOT.

$$\text{AND}(x_1, x_2) = \begin{cases} 1 & \text{if } x_1 > 0 \text{ and } x_2 > 0 \\ -1 & \text{if } x_1 > 0 \text{ and } x_2 < 0 \\ -1 & \text{if } x_1 < 0 \text{ and } x_2 > 0 \\ -1 & \text{if } x_1 < 0 \text{ and } x_2 < 0 \end{cases} \tag{1}$$

$$\text{OR}(x_1, x_2) = \begin{cases} 1 & \text{if } x_1 > 0 \text{ and } x_2 > 0 \\ 1 & \text{if } x_1 > 0 \text{ and } x_2 < 0 \\ 1 & \text{if } x_1 < 0 \text{ and } x_2 > 0 \\ -1 & \text{if } x_1 < 0 \text{ and } x_2 < 0 \end{cases} \tag{2}$$

$$\text{NOT}(x) = \begin{cases} -1 & \text{if } x > 0 \\ 1 & \text{if } x < 0 \end{cases} \tag{3}$$

Consider a neuron with weights $\mathbf{w}$ and bias b. And a its transformation $f : \mathcal{X} \mapsto \mathbb{R}$, with $f(\mathbf{x}) = \text{sign}(\mathbf{w}^\intercal \mathbf{x} + b)$.

(a) (3 points) Show there exists a neuron (that is, there exist weights and bias $\mathbf{w} \in \mathbb{R}, b \in \mathbb{R}$) with the behavior of the NOT function.

(b) (3 points) Show there exists a neuron (that is, there exist weights and bias $\mathbf{w} \in \mathbb{R}^2, b \in \mathbb{R}$) with the behavior of the AND function.

(c) (4 points) Show there exists a neuron (that is, there exist weights and bias $\mathbf{w} \in \mathbb{R}^2, b \in \mathbb{R}$) with the behavior of the OR function.

Learning goal: Further understand the expressivity of neural networks in a different manner. We know that AND, OR and NOT are the basic building blocks of digital circuits as they can be used to implement any Boolean function. We take such a "building blocks" approach to understand that neural networks are indeed very expressive, that is, they can indeed represent all Boolean functions. Furthermore, this approach can allow us to leverage the massive literature on digital circuits to represent various functions, and also compute guarantees on the number of gates (and hence neural-net parameters) required to express any given function.

2) Graphical models, causality, and Dr. X [10]

A scientist, Dr. X, is trying to find the efficacy of a certain medicine on COVID patients in Pittsburgh. On January 1 2021, Dr. X contacts all COVID patients in Pittsburgh and asks them to try out the medicine.

It turns out that not every person is interested to take the medicine. A key factor in the participants' decisions is the age of the participant – younger participants are much more open to taking this medicine. Unfortunately, Dr. X has no way of measuring whether any participant actually complied with the instructions or not.

Note that a crucial aspect of COVID to keep in mind is that the prospect of getting healthy depends on the age – younger patients are more likely to recover quickly.

Finally, on February 1 2021, Dr. X measures the fraction of participants from each group who became healthy again.

Questions:

(a) (5 points) Write down a directed graphical model that represents the setting above, that is, which represents the joint distribution regarding the above features (where the distribution is across all people with COVID on Jan 1). The graphical model should have three nodes:

1. Did the person take the medicine? Call this node "M".
2. Is the person young? Call this node "Y".
3. Did the person get healthy on February 1? Call this node "H".

Draw edges on this graph, and explain your choice of edges (or absence of edges).

(b) (5 points) Dr. X finds that 80% of people who took the medicine became healthy again whereas only 50% of people who did not take the medicine became healthy. Based on this, Dr. X concludes that the medicine does help in curing COVID. What could go wrong with such a conclusion?

Learning goal: To get your hands dirty in modeling a relevant, real-world problem as a graphical model (albeit a toy version of it, since this is an exam). A second goal is to get some experience in thinking critically about various assumptions and "confounding factors", rather than just looking at some aggregate numbers in the data, in order to draw conclusions.

3) Learning with Experts and Power of Two Choices [10]

The power of two choices is a powerful technique that is useful, in both theory and practice, in many different fields. At a high level it says that if you have to choose one of some d options, then pick two of the options uniformly at random, and then pick the best one of these two.[1]

In this question we will try out the power of two choices approach in the "learning from experts" problem. In particular, consider $d \geq 3$ experts, where the problem is to design a no-regret algorithm (in the worst/adversarial case). Consider the following algorithm:

- At the beginning of each day, choose two experts uniformly at random from the set of d experts.
- See which of the two experts has a lower error so far.
- Output the prediction of this expert (for that day).

Show that this algorithm is NOT a no-regret algorithm.

Note:

1. Assume the algorithm has access to the error of any expert over all the previous days (not just the ones that have been sampled).
2. When there is a tie, the algorithm breaks it uniformly at random.

Learning goal Intuitively, this algorithm combines two appealing properties: (i) randomness, which we saw in the lecture is necessary for a no-regret algorithm, and (ii) the well-known powerful nature of "power of two choices". One learning goal is to give you a negative result, where despite individual pieces of an algorithm being quite useful, they don't yield you the desired result. A second goal is to introduce to the very useful tool of power-of-two-choices (although it wasn't too useful here!).

[1] For example, this is useful in scheduling jobs in data centers. Google needs to decide which server it wants to send your request to. Ideally it would like to send it to the server with the lowest load. However, doing so would need querying all servers to find their respective loads. Instead, under the power of two choices strategy, it picks two servers uniformly at random, queries only these two servers for their loads, and then sends your request to the one of these two servers with a lower load.

4) Adversarial Machine Learning [10]

Adversarial Machine Learning pertains to adversarially manipulating the algorithm's output. One way to do so is by corrupting the training data.

Let us consider a toy example. Let $\mathcal{X} = \mathbb{R}$ and let $\mathcal{Y} = \{-1, 1\}$, the training data is denoted by $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^n$. Suppose the algorithm picks the classifier defined by the weight and bias $w \in \mathbb{R}, b \in \mathbb{R}$ that minimizes the square loss over all possible linear classifiers

$$\hat{w}, \hat{b} = \text{argmin}_{w \in \mathbb{R}, b \in \mathbb{R}} \sum_{i=1}^{n} (y_i - (wx_i + b))^2.$$

The decision rule of the classifier $\hat{y}(x) = \hat{f}_n(x, \hat{w}, \hat{b})$ with

$$\hat{f}_n(x, \hat{w}, \hat{b}) = \begin{cases} 1 & \text{if } \hat{w}x + \hat{b} > 0 \\ -1 & \text{otherwise} \end{cases}$$

Now consider the following perfectly separable training data with $n = 10$ points:
$\mathcal{S} = \{(-9, 1), (-7, 1), (-5, 1), (-3, 1), (-1, 1), (1, -1), (3, -1), (5, -1), (7, -1), (9, -1)\}$.

Suppose you are an adversary whose goal is to make the learnt output classify $y = 1$ for $x = 2$. You can move the "x" of any one of the 10 aforementioned training points to anywhere in $\mathcal{X}$. Is it possible to achieve this adversary's goal? Please supplement your answer with a proof. You can try to do it analytically or use your computer for a numerical solution.

Learning goal: Introducing you to adversarial machine learning via a toy example that you can work out by hand and experience this topic.

5) Categorical Log Likelihood and Cross Entropy [10]

Let $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{1, \ldots, K\}$. Consider the training data $\mathcal{S} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$. Each $\mathbf{x}_i$ is drawn i.i.d. from some unknown distribution $\mathcal{D}$.

Based on the value of $\mathbf{x}_i$, the corresponding label $y_i \in \mathcal{Y}$ is drawn as follows. We consider a hypothesis class $\mathcal{H}$ where each hypothesis in this class is parameterized by some parameters $\theta \in \Theta$ (for some set Θ). Then under the hypothesis parameterized by some θ, the label is distributed over $\mathcal{Y} = \{1, \ldots, K\}$ via the distribution $p(y = k|\mathbf{x}) = \pi_k(\theta, \mathbf{x})$. Here $\pi_1, \ldots, \pi_K$ are some known functions (but the parameter θ is unknown).[2]

With this context, the likelihood for the data $\mathcal{S}$, under the hypothesis class represented by θ, is given by:

$$L\left(\theta, \mathcal{S}\right) = \prod_{i=1}^{n} \prod_{k=1}^{K} \pi_k(\theta,\ \mathbf{x}_i)^{\mathbb{1}\{y_i = k\}}.$$

On a separate note, we will now introduce you to a loss that is a popular loss used for classification problems. This is called the cross-entropy loss. The (empirical) cross-entropy for the data $\mathcal{S}$, under the hypothesis class represented by θ, is given by:

$$E(\theta, \mathcal{S}) = -\sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}\{y_i = k\} \log(\pi_{y_i}(\theta,\ \mathbf{x}_i)).$$

Prove that the parameters θ that maximize the likelihood also minimize the empirical cross-entropy loss on the training data, that is

$$\text{argmax}_\theta L(\theta, \mathcal{S}) = \text{argmin}_\theta E(\theta, \mathcal{S})$$

Learning goal Introduce the concept of cross-entropy loss, one of the most popular losses used for classification problems in machine learning. Students will find the equivalence between this loss and the likelihood. A second learning goal is to get some idea about multi-class classification since in the lecture we largely focused on binary classification.

[2] This is called a categorical distribution. In words, this distribution is a discrete probability distribution and it can take one of K possible categories, with the probability of each category specified by $\pi_k(\theta, x)$ for all $k \in \{1, ..., K\}$.

6) Shapley Values and ML Interpretability [10]

Let $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \mathbb{R}$. Each $\mathbf{x}_i$ is drawn i.i.d. from some unknown distribution $\mathcal{D}$. Let $h : \mathcal{X} \mapsto \mathbb{R}$ be a model that uses all the d features and we denote $h_S : \mathcal{X} \mapsto \mathbb{R}$ to be a model which uses a subset S of the features. Let $F = \{1, ..., d\}$ be the feature indexes.

The Shapley value of a single feature $f \in F$ for a prediction $h(\mathbf{x})$ is denoted by $\phi_f(h(\mathbf{x}))$, and it measures the average difference in the predictions with and without this feature f. Formally, the Shapley value of any feature f is defined as:

$$\phi_f(h(\mathbf{x})) = \frac{1}{d} \sum_{S \subseteq F \setminus \{f\}} \frac{1}{\binom{d-1}{|S|}} \left(h_{S \cup \{f\}}(\mathbf{x}) - h_S(\mathbf{x}) \right).$$

Note that in the expression above, the summation is over all subsets $S \subseteq F \setminus \{f\}$.

Let us now consider a concrete machine learning task of fitting a regression model to predict the value of houses, using the following three features: The area of the terrain x_1, an indicator variable x_2 to denote if the house has backyard, and an indicator variable x_3 to denote if its located in a high crime zone. The model predicts the value of a house $\mathbf{x}$ with $\mathbf{x} = (x_1, x_2, x_3) = (500, True, True)$ and $h(\mathbf{x}) = 2,300$. Table 1 shows the predicted value for house $\mathbf{x}$ for h_S with all possible subsets S of features.

model	prediction
$h_\emptyset(\mathbf{x})$	1,740
$h_{\{1\}}(\mathbf{x})$	2,490
$h_{\{2\}}(\mathbf{x})$	1,790
$h_{\{3\}}(\mathbf{x})$	1,500
$h_{\{1,2\}}(\mathbf{x})$	2,540
$h_{\{2,3\}}(\mathbf{x})$	1,550
$h_{\{1,3\}}(\mathbf{x})$	2,250
$h_{\{1,2,3\}}(\mathbf{x})$	2,300

Table 1: Value predicted by h on each subset of features.

(a) (6 points) To understand the contribution of each feature to this prediction $h(\mathbf{x})$ compute the Shapley values for the three features $\phi_1(h(\mathbf{x}))$, $\phi_2(h(\mathbf{x}))$, $\phi_3(h(\mathbf{x}))$ using the expression above and information on Table 1.

(b) (4 points) Report the computational complexity of the calculation of a single Shapley value as a function on the number of features d. What are the implications?

Learning goal: Further understand the concept of Shapley values with a practical simple example. A second goal is to understand the computational complexity of calculating Shapley values and potential limitations. Finally, note that in order to compute the Shapley value the predictions of h_S and $h_{S \cup \{f\}}$ models are needed for all subsets.

While we did a brute force calculation of ϕ_f in this toy question, in practice usually only the model with the full set of features F is trained with the training data. Then features are "removed" to make a prediction for $\mathbf{x}$ by replacing their values with their mean for the feature f on the training data, this is denoted by $\bar{x}_f$.

7) A Bayesian approach to Multi-armed Bandits [10]

Recall the multi-armed bandit (MAP) problem where we have K arms with unknown means $\mu_1, \cdots, \mu_K$. At each iteration, one selects an arm and receives a reward. The goal is to identify the arm with the highest mean as soon as possible, or to minimize the regret. The core of an MAP algorithm is to balance exploitation (sampling from the best arm as much as possible) and exploration (pulling different arms to better identify the best one). For example, the Upper Confidence Bound algorithm discussed in class constructs a confidence interval for each arm to tackle the exploitation-exploration trade-off.

In this question we will explore a 'Bayesian' approach to this trade-off called Thompson sampling. The Thompson sampling algorithm first defines a prior distribution for the mean of each arm. At each iteration, the algorithm draws a sample from the posterior distribution of each arm and pulls the arm whose drawn sample had the maximum value among all arms. It will then update the posterior distribution of the pulled arm with the reward it observed. This way we encode the uncertainty of the estimate of μ_k's using the prior and posterior distributions which enables us to handle the exploitation-exploration trade-off.

Let's consider a simple case where we have two arms both following a Bernoulli distribution with means $\mu_1 = 0.4, \mu_2 = 0.6$. A popular approach is to assume that the prior distribution for each arm follows a $Beta(1, 1)$ distribution.[3] After t iterations, if we observe S_k successes (reward = 1) and F_k failures (reward = 0) for arm k, the posterior distribution of arm k is simply $Beta(1 + S_k, 1 + F_k)$.[4] Hence the Thompson sampling algorithm for Bernoulli bandits is:

```
Algorithm 1: Thompson sampling for Bernoulli bandits
  $S_k = 0, F_k = 0, k = 1, 2$
  for $t \leftarrow 1$ to $T$ do
      For $k = 1, 2$, sample $\lambda_k \sim Beta(1 + S_k, 1 + F_k)$
      Pull the arm $a = argmax_{k \in \{1,2\}} \lambda_k$ and observe a binary reward $r_t$
      $S_a = S_a + 1$ if $r_t = 1$, otherwise $F_a = F_a + 1$
  end
```

[3] The pdf of a $Beta(\alpha, \beta)$ random variable is: $f(x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)}$ where $x \in [0, 1]$, $B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$ and $\Gamma(z) = \int_0^\infty x^{z-1}e^{-x}dx$ is a gamma function. $Beta(1, 1)$ is equivalent to the uniform distribution on $[0, 1]$.

[4] If $\mu \sim Beta(\alpha, \beta)$, after observing n samples, $x_1, \cdots, x_n$ from a Bernoulli distribution, the posterior distribution has a closed form: $\mu | x_1, \cdots, x_n \sim Beta(\alpha + \sum_i x_i, \beta + n - \sum_i x_i)$. Since the posterior and prior are in the same distribution family (both are Beta), the prior (Beta) is called the 'conjugate prior' of the likelihood (Bernoulli).

Consider the Bernoulli bandit described above with $\mu_1 = 0.4, \mu_2 = 0.6$. Code the Thompson sampling method and run the algorithm with $T = 2500$ iterations and 1000 repetitions. Plot the regret (averaged over 1000 repetitions) vs iteration t. On the same plot, also include the 5th and 95th quantiles of the regret (over 1000 repetitions) vs iteration t. You can use a solid line to represent the mean, and dashed lines/bands to represent the percentiles.

Note:

1. Please compute the **actual** regret with the observed reward. For example, in one repetition, if you pulled arm 1 and 2 for the first two iterations and received reward 0 and 1, your regret at iteration 2 should be $0.6 * 2 - (0 + 1)$. This is consistent with what we studied in class. Feel free to use any built-in functions for sampling from a Beta distribution and plotting quantiles. If you are using python, you may want to look at numpy.random.beta and numpy.quantile.
2. Please append your code to your pdf submission.

Learning goal: To get your hands dirty with a Bayesian perspective for handling exploration-exploitation trade-off in the MAP problem and learn the popular "Thompson sampling" algorithm.