6.390: Final Exam, Fall 2022

Solutions

- This is a closed book exam. Two pages (8 1/2 in. by 11 in) of notes, front and back, are permitted. Calculators are not permitted.
- The total exam time is 3 hours.
- The problems are not necessarily in any order of difficulty.
- Record all your answers in the places provided. If you run out of room for an answer, continue on a blank page and mark it clearly.
- If a question seems vague or under-specified to you, make an assumption, write it down, and solve the problem given your assumption.
- If you absolutely *have* to ask a question, come to the front.
- **Write your name on every piece of paper.**

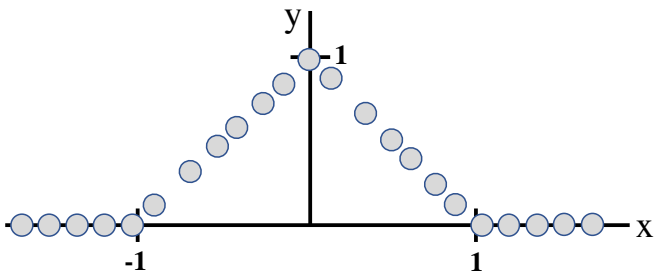
Name: _____ MIT Email: _____

Question	Points	Score
1	12	
2	10	
3	10	
4	8	
5	14	
6	18	
7	18	
8	10	
Total:	100	

Name: _____

Mountain Ascent

1. (12 points) Randy got back from hiking his favorite mountain. At various points of time x (in hours relative to before and after reaching the summit) he recorded his altitude y (in miles) to form a dataset $\mathcal{D}_n = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$. The dataset is plotted below.



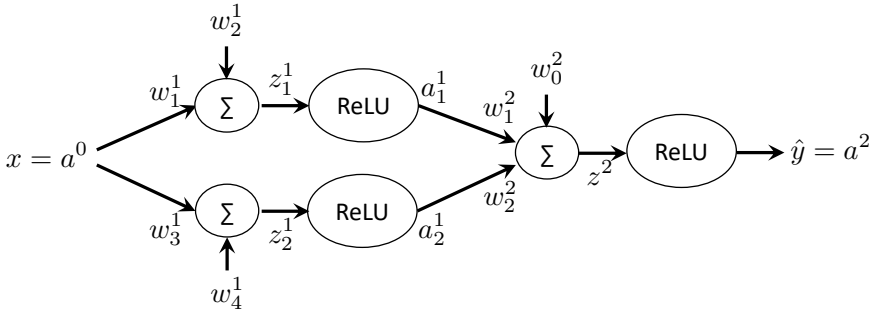
(a) Having learned about linear regression, he wonders if he can fit a piecewise linear function to the data. For each of the following ranges, give the formula for a linear function which perfectly fits the data in the range.

Solution:

$$y = \begin{cases} \underline{\quad \mathbf{0} \quad}, & x < -1, \\ \underline{\quad \mathbf{x + 1} \quad}, & -1 \le x < 0, \\ \underline{\quad \mathbf{1 - x} \quad}, & 0 \le x < 1, \\ \underline{\quad \mathbf{0} \quad}, & 1 \le x. \end{cases}$$

(b) Randy wonders whether it might be possible to fit the following two-layer neural network with ReLU activations to perfectly predict his altitude.

With our notation, e.g., w_2^2 denotes the second weight of the second layer, and multiplies the first layer activation from the bottom ReLU, a_2^1. Here w_2^1, w_4^1 and w_0^2 are constant offsets.



Is it possible to find weights to fit the dataset? If so, specify the weights as they appear in the diagram above. If not, explain why it is not possible using the provided neural network architecture.

Solution: √ **Possible** ◯ Impossible

$w_1^1 = 1$ $w_2^1 = 0$

$w_3^1 = -1$ $w_4^1 = 0$

$w_0^2 = 1$ $w_1^2 = -1$ $w_2^2 = -1$

Note that given the symmetry of the network, there are other correct solutions. For example, students may swap the top and bottom paths.

(c) Randy wants to see whether PyTorch can find a good fit to the data. Can you help him remember how to express this neural network architecture in PyTorch? Select one of the following.

Solution:

◯
```
[Linear(in_features=1, out_features=2, bias=True), ReLU(),
 Linear(in_features=2, out_features=2, bias=True),
 Softmax(dim=-1)]
```

◯
```
[Linear(in_features=1, out_features=1, bias=True),
 Linear(in_features=1, out_features=1, bias=True), ReLU(), ReLU(),
 Linear(in_features=2, out_features=1, bias=True), ReLU()]
```

◯
```
[Linear(in_features=1, out_features=2, bias=False), ReLU(),
 Linear(in_features=2, out_features=1, bias=False), ReLU()]
```

√
```
[Linear(in_features=1, out_features=2, bias=True), ReLU(),
 Linear(in_features=2, out_features=1, bias=True), ReLU()]
```

◯
```
[Linear(in_features=1, out_features=1, bias=True), ReLU(),
 Linear(in_features=2, out_features=1, bias=True), ReLU()]
```

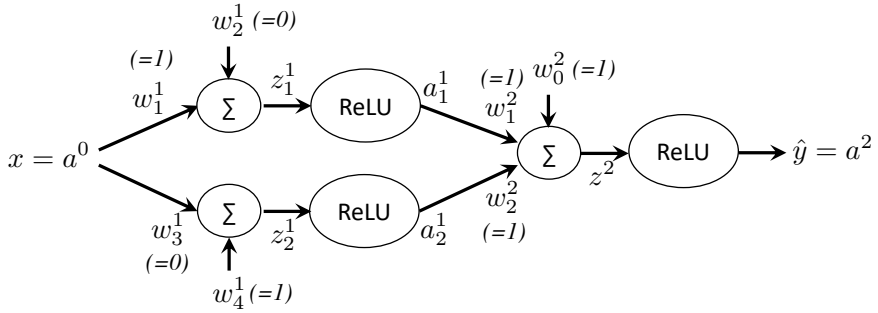
(d) Randy wants to understand what PyTorch is doing behind the scenes, so he decides to manually derive gradient descent updates for each of the weight parameters. He chooses the squared-error loss function, $\mathcal{L}(y, \hat{y}) = (y - \hat{y})^2$.

He initializes the weights of the neural network as:

$$w_1^1 = 1,\ w_2^1 = 0,\ w_3^1 = 0,\ w_4^1 = 1,\ w_0^2 = 1,\ w_1^2 = 1,\ w_2^2 = 1.$$

These initial weights are shown on the figure below, in italics.

Name: _____



With these initialized weights, what does the network predict for each of the following data points? What is the value of the loss function evaluated at each data point?

Solution:

$(x = 0, y = 1)$: $\hat{y} = 2$ loss= 1.

$(x = 1, y = 0)$: $\hat{y} = 3$ loss= 9.

(e) What is the gradient of the loss function with respect to the weight w_1^1? Derive the gradient in symbolic form (i.e., as a function of x, z's, etc.) using backpropagation. Then, compute the value of the gradient at the datapoint $(x, y) = (1, 0)$. Show all of your work.

Solution:

$$\begin{aligned}\frac{\partial \mathcal{L}(y,\hat{y})}{\partial w_1^1} &= \frac{\partial \mathcal{L}(y,\hat{y})}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z^2}\frac{\partial z^2}{\partial a_1^1}\frac{\partial a_1^1}{\partial z_1^1}\frac{\partial z_1^1}{\partial w_1^1} \\ &= -2(y-\hat{y})1[z^2>0]w_1^2 1[z_1^1>0]x.\end{aligned}$$

Evaluated at data point $(x, y) = (1, 0)$:

$$\begin{aligned}\frac{\partial \mathcal{L}(y,\hat{y})}{\partial w_1^1}(1,0) &= -2(0-3)\cdot 1\cdot 1\cdot 1\cdot 1 \\ &= 6\end{aligned}$$

(f) With a step size $\eta = 1$, what is the new value of w_1^1 after one iteration of stochastic gradient descent with respect to the data point $(x, y) = (1, 0)$?

Solution:

$$\begin{aligned}w_1^1 &\leftarrow w_1^1 - \eta\frac{\partial \mathcal{L}(y,\hat{y})}{\partial w_1^1} \\ &= 1-6 \\ &= -5\end{aligned}$$

Name: _____

Grey Matter

2. (10 points) Grey Matter, Inc. wants to build a neural network to convert 2×2 RGB color images into greyscale. The color images have three channels: channel 0 is red, channel 1 is green, and channel 2 is blue. For each pixel, Grey Matter use the following `RGB2Grey` conversion:

$$\text{grey} = 0.30 \text{ red} + 0.59 \text{ green} + 0.11 \text{ blue}.$$

Each pixel value (in both greyscale images and color image channels) is a real number in the range $[0, 1]$.

(a) Consider the following pixel values for a 2×2 color image:

Channel 0 (red)

1	1
0	0

Channel 1 (green)

0	1
0	1

Channel 2 (blue)

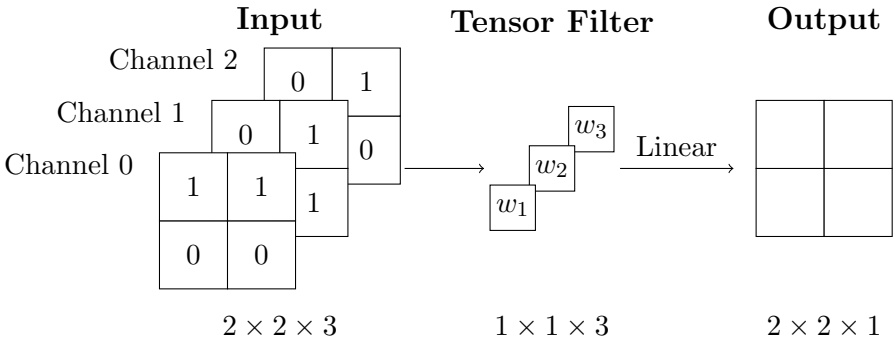
0	1
1	0

What are the corresponding pixel values in the greyscale image?

Solution:

0.30	1.0
0.11	0.59

(b) Blake is charged with building a convolutional neural network (CNN) to implement the `RGB2Grey` conversion. They heard a rumor that 1×1 convolutional tensor filters are useful in CNNs. They try the following architecture, where the input RGB color images have shape $2 \times 2 \times 3$, the tensor filter has shape $1 \times 1 \times 3$ with bias term $w_0 = 0$ and stride 1, a Linear activation function is used, and the output greyscale image has shape $2 \times 2 \times 1$:



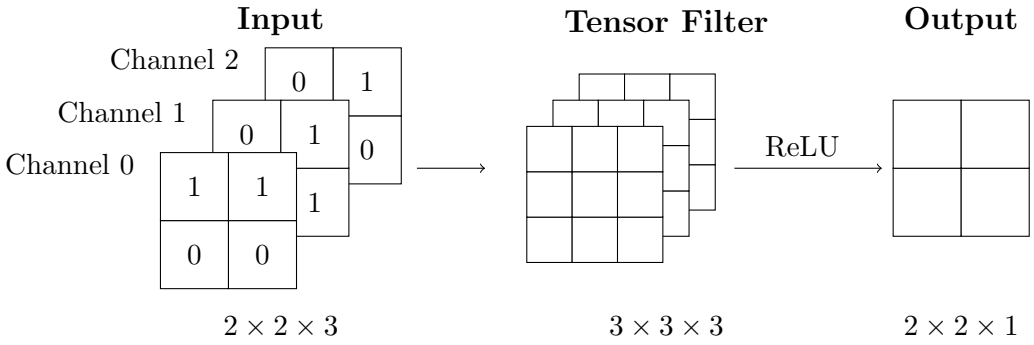
With hundreds of different training samples, can this CNN learn the greyscale conversion? If so, provide a set of values for the weight parameters w_1, w_2, and w_3 that implements the converter. If not, explain why the converter cannot be learned with this CNN.

Solution: Yes:

$$w_1 = 0.30, w_2 = 0.59, w_3 = 0.11.$$

With sufficient training examples and gradient descent with a squared error loss function, we can converge to this linear combination across channels.

(c) Cade believes that the architecture above needs more capacity to deal with a large variety of images. He replaces the 1×1 convolutional filter with a $3 \times 3 \times 3$ tensor filter with offset term $w_0 = 0$, and replaces the Linear activation with a ReLU:



Assume the filter is applied with stride 1. In order to obtain a $2 \times 2 \times 1$ output, is padding of the input necessary? If so, describe where padding values are needed, and what the padding values should be.

Solution: Zero-padding is needed, adding one pixel of zeros in both the horizontal and vertical dimensions around each channel of the input. (That requires 12 zero-values surrounding each channel of the 2 x 2 input image, or 36 total zero-values for padding for all three channels.)

(d) With hundreds of different training samples, can Cade's CNN from part (c) learn the greyscale conversion, assuming that zero-padding has been applied if or as necessary? If so, fill in a set of weights for the $3 \times 3 \times 3$ tensor filter above. If not, explain why the grey-scale converter cannot be learned with this CNN.

Solution:

Yes, this tensor filter can be learned, e.g., with gradient descent and squared error loss.

Channel 0

0	0	0
0	0.30	0
0	0	0

Channel 1

0	0	0
0	0.59	0
0	0	0

Channel 2

0	0	0
0	0.11	0
0	0	0

Name: _____

Rainbow Matter

3. (10 points) Grey Matter, Inc., is hoping to learn beyond just filters to spin-off a new company, now called Rainbow Matter, Ltd.!

   (a) They task Cade to consult with Dana (a photographer) to conduct some market research. Cade asks Dana, "What are some of your 'go-to' moves when you edit photos? We'd like to try machine learning a model to achieve those moves!" Dana replies, "Hmm. I often need to rotate my photos."

   Cade believes that a convolutional filter can be learned to rotate photos, i.e., to move the pixel values by one-quarter turn (90 degrees) clockwise in the image matrix:

Input		Desired output (after convolution)	
1	2	4	1
4	3	3	2

   As a "proof-of-concept," Cade tries to hand-design a two-dimensional convolutional filter to achieve the goal. To his delight, Cade finds that the following filter (with offset 0) succeeds in rotating the image above, using zero-padding of size 1 on the input, and an (unusual!) **stride of 2**.

   $$\begin{array}{|c|c|}\hline a & b \\ \hline d & c \\ \hline\end{array} = \begin{array}{|c|c|}\hline \frac{2}{3} & \frac{3}{4} \\ \hline \frac{1}{2} & c \\ \hline\end{array}$$

   What is the missing filter weight c?

   Solution: c=4.

   With the zero-padding of size 1, the original image would be padded as

   $$\begin{array}{|c|c|c|c|}\hline 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 2 & 0 \\ \hline 0 & 4 & 3 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline\end{array},$$

   and with a stride of 2, after applying the filter once, the output will be $\begin{array}{|c|c|}\hline c & 2d \\ \hline 4b & 3a \\ \hline\end{array}$.

   So we need $\begin{array}{|c|c|}\hline c & 2d \\ \hline 4b & 3a \\ \hline\end{array} = \begin{array}{|c|c|}\hline 4 & 1 \\ \hline 3 & 2 \\ \hline\end{array}$.

   Indeed, equality conditions on a, b, and d are satisfied by Cade's filter. And we just need c=4.

   (b) Cade is super excited; he believes he has found this magical rotational filter. Dana has serious doubts though. She says that Cade's filter is a one-time wonder, believing that it can only rotate that particular original image. She challenges Cade to demonstrate his success again, by rotating the previous output another 90 degrees using his filter.

Keeping the zero-padding and stride scheme the same as before, when the filter Cade found in part (a) is applied to the new input image, will it produce the desired output below?

New Input

4	1
3	2

Desired output (after convolution)

3	4
2	1

Solution: ◯ Yes √ **No**

To get this input output relationship, we need to have $\begin{array}{|c|c|}\hline 4c & 1d \\ \hline 3b & 2a \\ \hline \end{array} = \begin{array}{|c|c|}\hline 3 & 4 \\ \hline 2 & 1 \\ \hline \end{array}$.

Recall from part (a) that $c = 4$; this contradicts $4c = 3$.

(c) Dana's influencer friend Emily heard about the collaboration. She suggests: "the more business-savvy choice might be to learn a detector that can predict whether a filtered photo will get any 'Like's or not!" Dana was like, yeah, cool idea! She collected 1000 of her recent photos posted on social media, and extracted the Like counts.

The goal is now to predict whether a post will receive any Likes or not, (i.e., more than zero), and we need to help Cade design a complete CNN architecture.

i. What output encoding could Cade use on top of Dana's Like counts as the output? Specifically, describe how Cade should transform Dana's Like counts to obtain the labels used for training, the number of output units, and the loss function that should be used.

Solution: It should be a single output unit. Cade should construct a single binary output which is 1 if a post has >= 1 Likes, and 0 otherwise, e.g., by thresholding on a sigmoidal output. We can use the NLL loss.

ii. Suppose that Cade has access to these six layer and activation types:

- Convolutional Layer
- Fully Connected Layer
- Max Pooling Layer
- ReLU Activation
- Sigmoid Activation
- tanh Activation

Fill in the blanks below with the preferable layer or activation to help Cade complete the architecture for his model (preferable in the computational-cost sense):

Solution: (input image) → Convolution layer → max pool → ReLU → convolutional → max pool → ReLU→ Fully connected → Sigmoid → (binary output)

Or alternatively (swapping order or RelU and max pooling layers):
(input image) → Convolution layer → ReLU → max pool → convolutional → ReLU → max pool → Fully connected → Sigmoid → (binary output)

(d) Cade grew unsatisfied with just predicting if a photo will get any Likes. He thought, wouldn't it be more fun to predict $[p_1, p_{10}, p_{100}]$, i.e., the probability that the image would receive at least one like, at least 10 likes, and at least 100 likes? He believes a simple change of his network architecture to use a softmax activation at the output layer would achieve this new goal.

Is he correct? If not, please explain what activation function would be more appropriate.

Solution: No. $p_1 + p_{10} + p_{100} \neq 1$. Alternatively, Cade could use three sigmoid activation functions.

(e) Cade's friend Fish raises another question to Cade: regardless of whether the softmax is a valid idea, your CNN couldn't possibly have accurate predictions out in the real world. Specifically, it isn't possible to accurately to predict whether an image will have at least 100 likes by only looking at the image. You need more features to make the prediction!

Do you agree with Fish? What could be such a real-world feature that's missing as input, in addition to the photo itself?

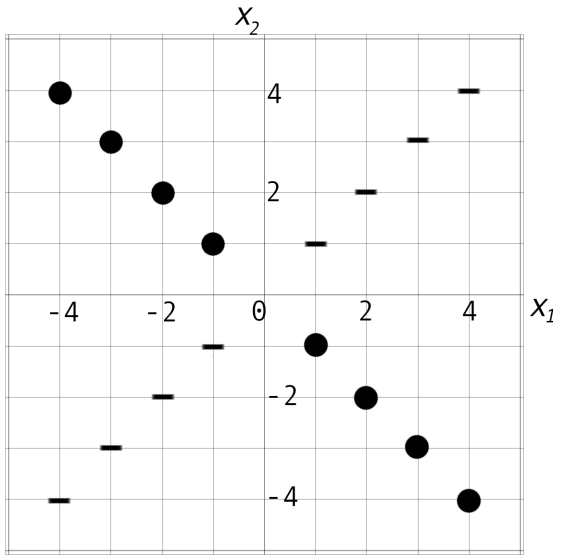
(This is an open-ended question. We're only expecting you to identify a feature; you do not need to specify how to encode this feature or how to construct a model taking in this feature.)

Solution: Whether the poster is likely to even have 100 visitors?

Name: _____

Decisions, Decisions

4. (8 points) You have 16 data points in your training set, plotted below. These data points in the plot just below are located exactly where gridlines intersect; for instance, there is a point at $(x_1, x_2) = (1, 1)$. There are two classes: circles and dashes. You are interested in building a decision tree for classification.



(a) What is the entropy for the entire data set? Your final answer should be a single numeric value (not an equation).

Solution: Let I_m denote the indices of all our data points.

$$\begin{aligned} H(I_m) &= -\hat{P}_{m,\text{circle}} \log_2 \hat{P}_{m,\text{circle}} - \hat{P}_{m,\text{dash}} \log_2 \hat{P}_{m,\text{dash}} \\ &= -\left(0.5 \cdot \log_2 \frac{1}{2}\right) - \left(0.5 \cdot \log_2 \frac{1}{2}\right) \\ &= -\log_2 \frac{1}{2} = 1 \end{aligned}$$

(b) First, we think about making a decision tree with a single split. What is the weighted average entropy of the split in dimension 1 at location 0 for this data set? Your final answer should be a single numeric value (not an equation).

Solution: This split has 50% of each class in the two subsets of the split, so the average weighted entropy is still $\hat{H} = 1$. Mathematically, the average weighted entropy $\hat{H}$ is now

$$\hat{H} = \frac{8}{16} H(I^-) + \frac{8}{16} H(I^+)$$

where

$$\begin{aligned} H(I^+) &= -\hat{P}_{+,\text{circle}} \log_2 \hat{P}_{+,\text{circle}} - \hat{P}_{+,\text{dash}} \log_2 \hat{P}_{+,\text{dash}} \\ &= -(0.5 \cdot -1) - (0.5 \cdot -1) = 1 \end{aligned}$$

and similarly $H(I^-) = 1$, giving

$$\hat{H} = \frac{1}{2} H(I^-) + \frac{1}{2} H(I^+) = 1.$$

(c) What is the weighted average entropy of the split in dimension 1 at location 1.5 for this data set? Your final answer should be a single numeric value (not an equation).

Solution: As in the previous part, this split also has 50% of each class in the two leaves, so the average weighted entropy is still 1. The details change, but the end result works out the same:

$$\hat{H} = \frac{10}{16} H(I^-) + \frac{6}{16} H(I^+)$$

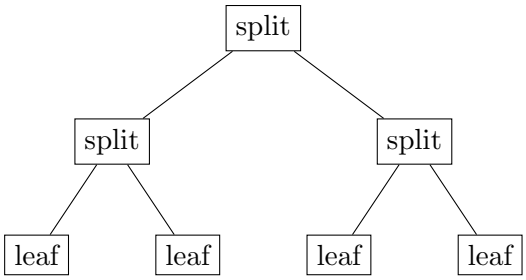
but $H(I^+) = H(I^-) = 1$ as before, so that

$$\hat{H} = \frac{10}{16} H(I^-) + \frac{6}{16} H(I^+) = 1.$$

(d) Now consider any possible split, on dimension j at location s. Which such split of this data set has the smallest weighted average entropy? If there are ties, describe all the splits with smallest weighted average entropy.

Solution: All single-feature splits on this data set have average weighted entropy of 1, because all such splits will have an equal number of each class in both subsets of the split. It's not until we do a second split on this data that we start getting regions with more of one class than the other in split regions.

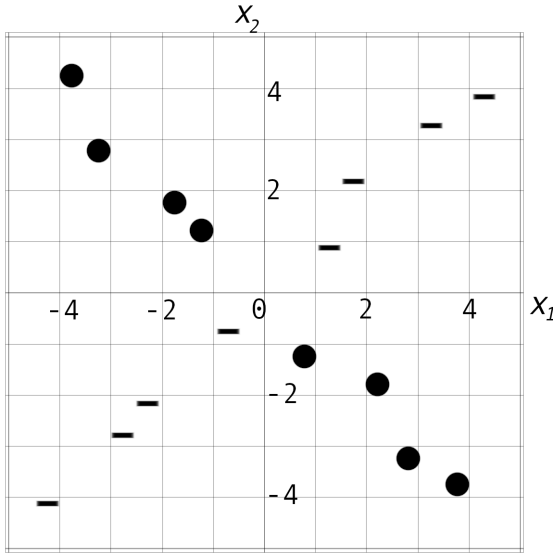
(e) More generally, suppose we take a greedy approach to growing a tree of the form depicted just below on any data set. That is, we first choose an optimal split for the root node (according to weighted average entropy). Second, we choose optimal splits for the left and right children of the root node. Is this procedure guaranteed to find the tree of the form drawn below that has the highest accuracy on the training data? Be sure to justify your answer.



Solution: No, it is not guaranteed to find the tree with this structure that gives the highest accuracy on training data.

The data set above is a counterexample. With only the information from the possible first splits, we can't guarantee that we choose the central split. (They all have the same weighted average entropy.) While there is a tree with one root node and two child splits that gets perfect accuracy on this training data, we can't guarantee we'll find it with this greedy procedure.

(f) Now suppose a little noise is added to our data; see the plot just below. So no two data points share the exact same feature value anymore. Will the value of the smallest weighted average entropy across possible splits change from part d? Will the split with smallest weighted average entropy change? Why or why not? (Note: you do **not** need to report any numeric values for weighted average of the new, noisy data in your answer.)



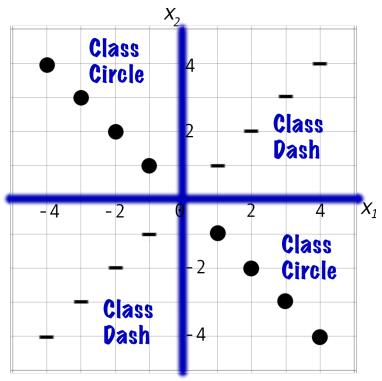
Solution: Yes; a single split is able to have a different number of classes across some of the single feature splits, so the average weighted entropy for those splits will be less than 1. For example, a split at $x_1 \geq 4$ will enable the entropy $H(I^+)$ to be 0, which will reduce the averaged weighted entropy for the split to be less than 1. It's not clear what the best split (or lowest average entry) will be, because that depends on the particular noisy locations of the data.

(g) Now let's return to the original data. Draw the decision boundary and predicted classes

Name: _____

for 1-nearest neighbors on the plot in the box below. (The plot is duplicated from the plot at the start.)

Solution:



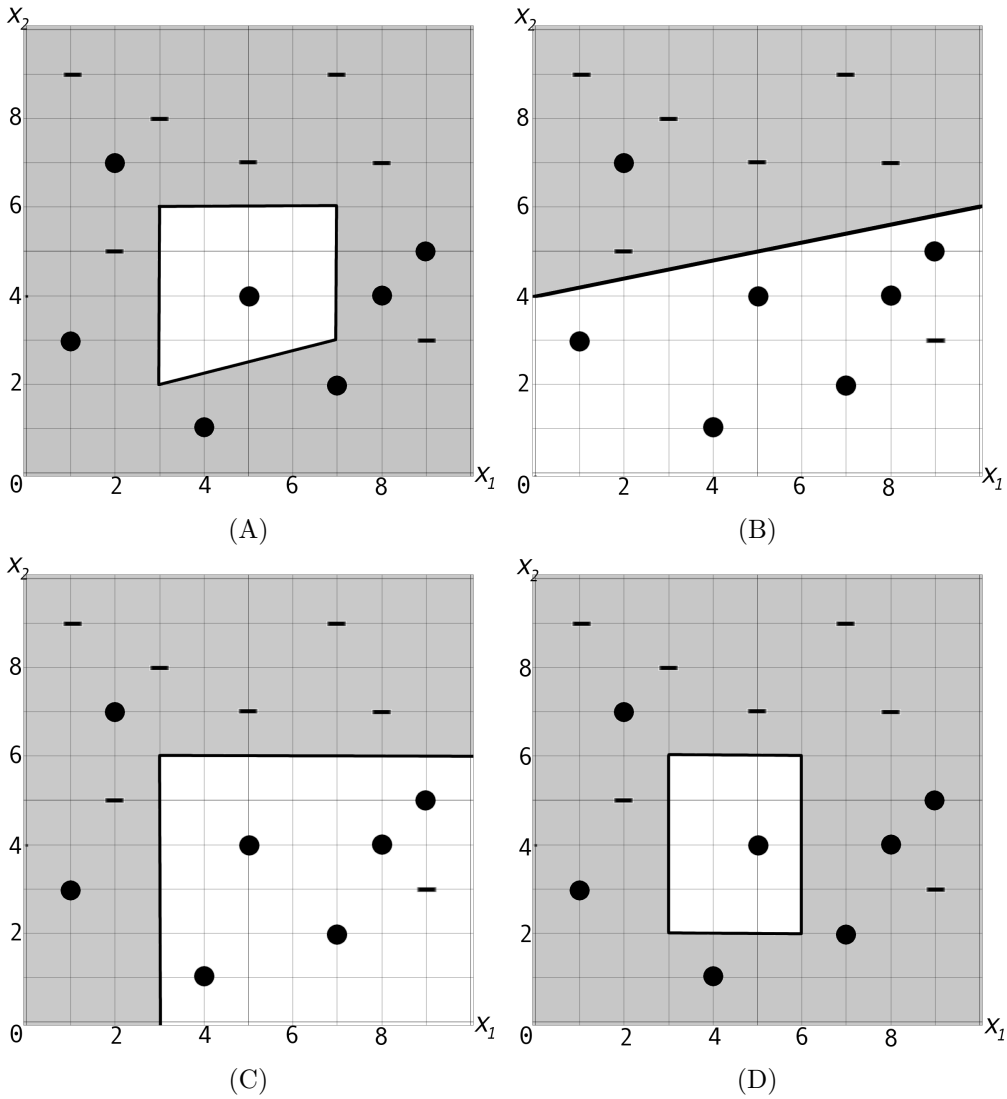
Name: _____

Game, Set, Match

5. (14 points) Rowan has been trying out various different classifiers on her data, but she forgot which is which. She hopes you can help her remember. Rowan's data has two classes: circles and dashes. Each picture below represents a binary classifier, where the decision boundary is in black. The region shaded gray predicts the dash class; the region shaded white predicts the circle class.

   In the questions below, Rowan is curious which of the pictured classifiers, if any, might belong to a stated hypothesis class. You can choose one picture, multiple pictures, or no pictures. Assume the features x_1 and x_2 are used directly; there are no feature transformations prior to applying the predictor.

   Unless the question refers specifically to the pictured data, you can assume Rowan is asking about *any* member of the hypothesis class, *not* just the optimal hypothesis with respect to the data.



(A) (B)

(C) (D)

(a) Rowan thinks she might have tried a linear classifier. Identify all of the plots above which could represent such a predictor; mark with an X all that apply.

Solution:
◯ (A) √ **(B)** ◯ (C) ◯ (D) ◯ None

(b) For every linear classifier (A) through (D) that you identified in the previous part, provide the decision boundary (i.e., the separator) in the form $0 = \theta^\top x + \theta_0$. Identify θ and θ_0. Additionally, write the normal vector for the classifier that points to the half-space classified as positive (+1), where we think of the circle class as corresponding to a positive (+1) prediction.

Solution:
Note that student answers can vary by constants c_1, c_2 in the answers below. We must have $c_1 \neq 0$ and c_1 must agree between θ and θ_0. We must have $c_2 > 0$.
(B) $\theta = c_1[1, -5]^\top$, $\theta_0 = 20c_1$, normal vector $= c_2[1, -5]^\top$
(Note: $c_2[-1, 5]^\top$ is **incorrect**.)

(c) Rowan thinks she might have tried a decision tree with two splits of the form $x_j \geq s$. Identify all of the plots above which could represent such a predictor; mark with an X all that apply.

Solution:
◯ (A) ◯ (B) √ **(C)** ◯ (D) ◯ None

(d) For every classifier you identified in the previous part, fill in the corresponding tree. In particular, we've set up the tree structure for you below. You will need to fill in the splits of the form $x_j \geq s$, fill in the leaf values, and choose "yes" or "no" as appropriate. As a reminder, you only need to specify a tree that produces the decision boundary (if any), without worrying about whether any particular learning algorithm would find it.

Solution:
(C) There are two possible answers.
Answer 1: The top split is $x_1 \geq 3$. No ⇒ dash class. Yes ⇒ lower split $x_2 \geq 6$. Yes ⇒ dash class; No ⇒ circle class.
Answer 2: The top split is $x_2 \geq 6$. Yes ⇒ dash class. No ⇒ lower split $x_1 \geq 3$. Yes ⇒ circle class; No ⇒ dash class.
We won't take off for using $>$ instead of $\geq$ since the boundary behavior isn't obvious from the picture. And using $<$ or $\leq$ is fine as long as they get the yeses and nos right.

(e) Rowan thinks she might have tried a decision tree with an unconstrained number of splits of the form $x_j \geq s$. Identify all of the plots above which could represent such a predictor; mark with an X all that apply.

Solution: ◯ (A) ◯ (B) √ **(C)** √ **(D)** ◯ None

(f) Rowan thinks she might have tried a 2-layer neural network with two hidden neurons and one output neuron, where each neuron uses the ReLU activation function. That is, the first layer takes the two features (x_1 and x_2) as inputs and has two outputs; the second layer takes the two outputs from the first layer as inputs and has one output (the final predicted class). Identify all of the plots above which could represent such a predictor; mark with an X all that apply.

Solution:
◯ (A) √ **(B)** √ **(C)** ◯ (D) ◯ None

(g) Rowan thinks she might have tried a 2-layer neural network with an unconstrained number of hidden neurons and one output neuron, where each neuron uses the ReLU activation function. Identify all of the plots above which could represent such a predictor; mark with an X all that apply.

Solution:
√ **(A)** √ **(B)** √ **(C)** √ **(D)** ◯ None

(h) Rowan thinks she might have tried a 1-nearest neighbors classifier using Euclidean distance on the full set of pictured data. (Unlike the previous questions, this question really does depend on Rowan's full data set, shown in the picture.) Identify all of the plots above which could represent such a predictor; mark with an X all that apply.

Solution:
◯ (A) ◯ (B) ◯ (C) ◯ (D) √ **None**

Name: _____

Word

6. (18 points) "Word" is a word game where one tries to guess a three-letter word in four guesses, where each guess must be a three-letter word. In our game, we will use five common letters in the English language $\{A, E, R, S, T\}$.

   (a) We will consider all valid three-letter English words as character sequences $c = [c_0, c_1, c_2]$. For the letters $\{A, E, R, S, T\}$, we will use the following encoding function ϕ:

$$\phi(A) = [1, 0, 0, 0, 0],$$
$$\phi(E) = [0, 1, 0, 0, 0],$$
$$\phi(R) = [0, 0, 1, 0, 0],$$
$$\phi(S) = [0, 0, 0, 1, 0],$$
$$\phi(T) = [0, 0, 0, 0, 1].$$

   Consider the character sequence $[E, A, R]$. Provide the encoding of this sequence, i.e., $[\phi(E), \phi(A), \phi(R)]$.

   Solution: We can encode the full word as an ordered list of our one-hot encodings for each position in the word, in sequence. Thus our encoding for EAR would be:

$$[\phi(E), \phi(A), \phi(R)] = [[0, 1, 0, 0, 0], [1, 0, 0, 0, 0], [0, 0, 1, 0, 0]].$$

   Similar to a list of list representation, would could alternatively represent the output as a matrix, where the first row is the vector for $\phi(E)$, second for $\phi(A)$, and third row for $\phi(R)$.

   (b) We want to build a model to help us play the game better. We want the model to take a game state (which we will encode later in the problem) as input, and to predict the probability of all letters at all positions in the word. How should we encode the output and what is the dimension of the output?

   Solution: A reasonable encoding is to output five real values (between 0 and 1), one for each possible letter, for each of the three letters (positions) in the word. E.g., a probability matrix P as a 5 by 3 matrix, with 15 total probabilities. We can use the same ordering down the column, for the probabilities in order of A, E, R, S, and T, where the column corresponds to the position in the word.

   (c) Which (potentially nonlinear) output function should we use to build this output, and which sets of dimensions should this output be applied to?

   Solution: Because we want probabilities across multiple classes, we should use a softmax function to output the probabilities of our possible letters, for each letter position in the word. Softmax should be applied independently for each letter position. So building on our result from (c), our encoding would be a 2D matrix P of probabilities, where column 0 in P will have the first softmax outputs for the first letter (position)

in the word, column 1 in P for the second letter (position) in the word, and column 2 in P for the third letter (position) in the word.

(d) We indicate the probability for letter i in position j as $a_j^{(i)}$, where $j \in \{1, 2, 3\}$. Give an example output for the word ERA with the probability for the first position $a_1^{(E)} = 1$; for the second position $a_2^{(R)} = 0.9$ and $a_2^{(T)} = 0.1$; and for the third position $a_3^{(A)} = 0.9$ and $a_3^{(T)} = 0.1$.

Solution: Using our definition of probability matrix P (where column j indicates position j in the word), and the probabilities down each column uses our same A, E, R, S, and T row ordering, we would get:

$$\begin{bmatrix} 0.0 & 0.0 & 0.9 \\ 1.0 & 0.0 & 0.0 \\ 0.0 & 0.9 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.1 & 0.1 \end{bmatrix}$$

Alternatively, if one is using row i to correspond to position i in the word, and the ordering of probabilities along the row uses our same A, E, R, S, and T ordering, then the resulting probability matrix output is a transpose of the matrix above.

$$\begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.9 & 0.0 & 0.1 \\ 0.9 & 0.0 & 0.0 & 0.0 & 0.1 \end{bmatrix}$$

(e) Suppose we know that the first letter is an E, the last two letters are not an E, S, or T, and we want the output to indicate 80% probability that the word is EAR and 20% probability that the word is ERA (and zero probability that it is any other word, like ERR). Can we represent that output with the representation above? If so, show the output. If not, explain why not.

Solution: We cannot represent this output with the encoding above, because our output encoding does not encode the probability of a whole word; it only encodes probabilities of letters in positions.

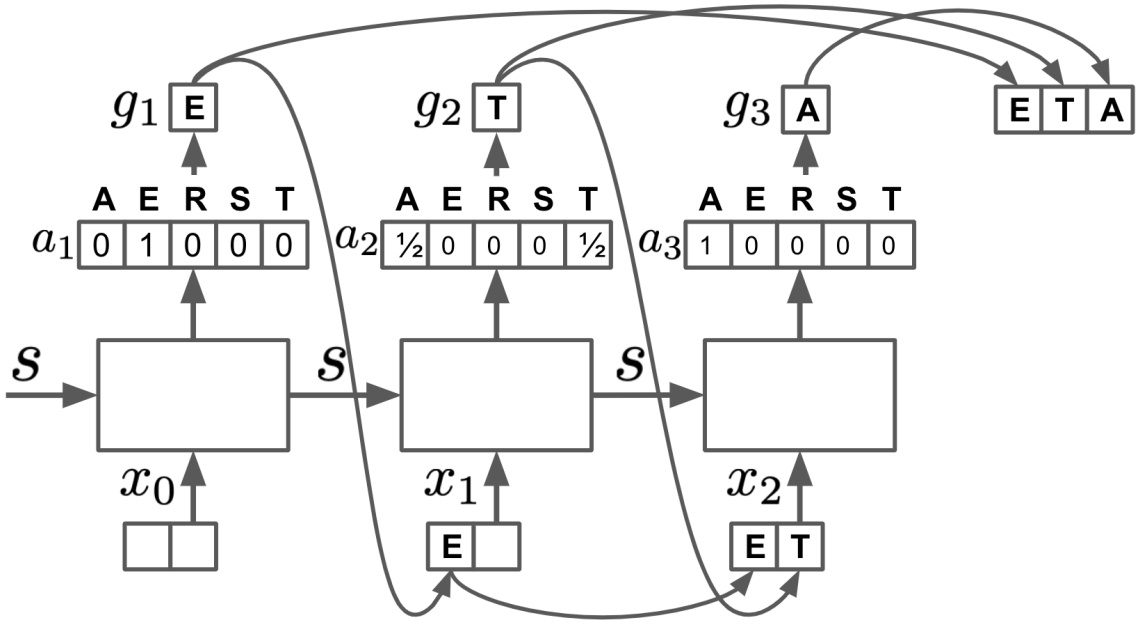
For example, if we attempted to encode (80% EAR and 20% ERA) using our probability representation as below, then that would still allow ERR (and even other non-valid words like EAA) to be consistent with the resulting letter probabilities:

$$\begin{bmatrix} 0.0 & 0.8 & 0.2 \\ 1.0 & 0.0 & 0.0 \\ 0.0 & 0.2 & 0.8 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}.$$

(f) Rather than having our model making predictions for all the letters at once, we instead decide to use a recurrent neural network (RNN) to output its prediction as a sequence of letters. We will use the all-zeros encoding $\phi('-') = [0, 0, 0, 0, 0]$ for the token $'-'$ (dash), a special character that indicates no letter. Our RNN has the following design made of three identical blocks:

1. The RNN first takes as input the game state s and $x_0 = [\phi('-'), \phi('-')]$ and outputs the probability that each letter appears in the first position, a_1. We then sample the first letter with the likelihood given by the probabilities a_1 to produce our guess of the first letter, g_1.
2. To the second block, we pass in the same game state s from before and $x_1 = [\phi(g_1), \phi('-')]$, with the guess g_1 from the previous block and a blank second letter. The second block outputs the probability that each letter appears in the second position, a_2. We then sample the second letter with the likelihood given by the probabilities a_2 to produce our guess g_2.
3. Finally we pass the same game state s and $x_2 = [\phi(g_1), \phi(g_2)]$ to the final block. The third block predicts g_3, the third letter, with the likelihood given by the probabilities a_3.

Our final prediction is the sequence of letters $[g_1, g_2, g_3]$. Note that while in the diagram we are using the letters as input, we actually would use the encoding ϕ from part (a).



i. For every guess, the game will return the game state s, which reflects what we know about the word based on our previous guesses. For each position $j \in \{1, 2, 3\}$ in the word, we use the following encoding for game state at that position, s_j:

1. $[0, 0]$ for when the letter is not in this position in the word (including not in the word at all),

2. $[0, 1]$ for when we don't know if the letter is in this position or not,
3. $[1, 1]$ for when the letter is in the word in this position.

What is the name for this type of encoding?

> **Solution:** Thermometer encoding

ii. The game has given us the following information from our previous guesses:
1. A is not the second letter (not in the second position in the word);
2. E is the first letter (the first position in the word);
3. R is not the third letter (not in the third position in the word);
4. S is not in the word;
5. and our guesses have not generated additional information about T.

The encoding of the game state for the first position (first letter) in the word is

$$s_1 = [[0, 0], [1, 1], [0, 0], [0, 0], [0, 0]]$$

where each element in the list is the encoding corresponding to knowledge about letter A, E, R, S, and T, in sequence. The game state for the second letter in the word is:

$$s_2 = [[0, 0], [0, 1], [0, 1], [0, 0], [0, 1]] \ .$$

Using the encoding above, give the encoding of the game state for the third position in the word:

> **Solution:** The game state for the third position in the word is:
>
> $$s_3 = [[0, 1], [0, 1], [0, 0], [0, 0], [0, 1]$$
>
> Again, other encodings are possible, such as a 2D matrix where a row or column corresponds to the encoding for our knowledge about a particular letter in the position of interest.

iii. Recall the output activation function from part (c). If we want the predicted probability of a given letter to be (very close to) zero, what value should the pre-activated input to the output activation function be?

> **Solution:** Since the output activation in (c) is the softmax function, if we want the probability to be close to zero, the pre-activated number going into the softmax should be very negative, i.e., much less than the values for other letters.

iv. We want to guarantee that the model will not predict letters that we know from the game state s cannot be at the given position. To do this, we will apply the following neural network to the pre-activations z_j of the output a_j, where j indicates the j^{th} position. The pre-activations take into consideration x_{j-1}, and this neural network accounts for s.

```
# s_j : game state at position j shape [5, 2] -> [letter, encoding]
# z_j : pre-activation of a_j, shape [5]
# w : shape [1], b : shape [1]
s_j_summed = s_j.sum(axis=1)  # shape [5]
condition, if_true, if_false = s_j_summed < 1, s_j_summed, 1
s_where = np.where(condition, if_true, if_false)  # shape [5]
c_j = f_1(w * s_where + b)
a_j = f_2(z_j - c_j)
```

Assume `w` and `b` are forced to be values between -10 and 10, and `f_2` is the output activation function. Specify what the function `f_1` should be, and values for `w` and `b`, such that `c_j` is nonzero when `s_j[i] == [0, 0]` (the state for the i^{th} letter in the j^{th} position) for all i but zero when `s_j[i]` is `[0, 1]`, or `[1, 1]` for any i. Note that the value `c_j` is subtracted from `z_j` and keep in mind your answer to part (iii).

Solution: We want the argument of `f_2` to be very negative and therefore `c_j` to be very positive. We can make `f_1` a ReLU or linear and `w` and `b` equal in magnitude with opposite sign, arriving at `w = -10` and `b = +10`.

Name: _____

Markovian Spins

7. (18 points) Pauli is in the lab measuring the direction of the spin of a qubit. We're going to help Pauli model their observations. The spin of the qubit can be in the state up $\uparrow$ or down $\downarrow$ along specific directions x or z, denoted as $\uparrow x$, $\downarrow x$, $\uparrow z$, and $\downarrow z$.

   Pauli can apply one of two measurements, M_x or M_z, to observe the spin along the directions x or z.

   Pauli knows that qubits obey the following physics. If Pauli applies measurement...

   1. M_x on $\uparrow x$ or $\downarrow x$, they get back the same state, $\uparrow x$ or $\downarrow x$, respectively.
   2. M_z on $\uparrow z$ or $\downarrow z$, they get back the same state, $\uparrow z$ or $\downarrow z$, respectively.
   3. M_x on state $\uparrow z$ or $\downarrow z$, they observe a transition to $\uparrow x$ with 50% likelihood or to $\downarrow x$ with 50% likelihood.
   4. M_z on state $\uparrow x$ or $\downarrow x$, they observe a transition to $\uparrow z$ with 50% likelihood or $\downarrow z$ with 50% likelihood.

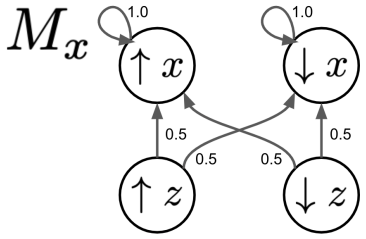
   We would like to model Pauli's measurements of the qubit as a Markov Decision Process. To do this, we must define our states, actions, and transitions between states for each action.

   (a) What are the state space $\mathcal{S}$ and action space $\mathcal{A}$?

   Solution: $\mathcal{S} = \{\uparrow x, \downarrow x, \uparrow z, \downarrow z\}$, and $\mathcal{A} = \{M_x, M_z\}$

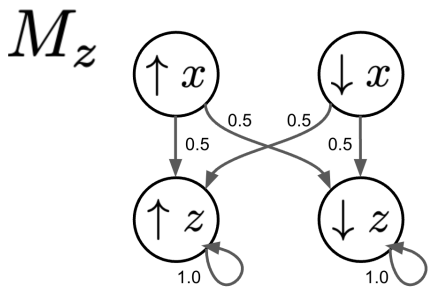
   (b) Draw a diagram showing the transitions between states via actions M_x. Be sure to write the probability of transitioning between states.

   Solution:

   (c) Draw a diagram showing the transitions between states via actions M_z. Be sure to write the probability of transitioning between states.

Solution:



(d) We want to calculate the value of the following policy π_0:

- $\pi_0(\uparrow x) = \pi_0(\downarrow x) = \pi_0(\uparrow z) = M_z$
- $\pi_0(\downarrow z) = M_x$

Suppose we get a reward of 100 for measuring the state $\uparrow z$ with M_z, $R(\uparrow z, M_z) = 100$, and the reward of all other state-action pairs is 0. Calculate the value of the policy π_0 for horizon of 0 and horizon of 1 starting in $\uparrow z$ or $\downarrow z$. Where appropriate, first express your answer in terms reward function $R(s, a)$ for the appropriate state s and action a determined by the policy π_0, before reducing the answer to the numerical values.

Solution:

$V^0_{\pi_0}(\uparrow z) = 0$
$V^1_{\pi_0}(\uparrow z) = R(\uparrow z, M_z) + 0 = 100$
$V^0_{\pi_0}(\downarrow z) = 0$
$V^1_{\pi_0}(\downarrow z) = R(\downarrow z, M_x) + 0 = 0$

(e) Calculate the value of the policy for horizon of 2, starting from $\uparrow x$. Assume a discount factor of γ, meaning leave your answers in terms of γ. Where appropriate, first express your answer in terms reward function $R(s, a)$ and transition model $T(s, a, s')$ for the appropriate state s and s' and action a determined by the policy π_0, before reducing the answer to the numerical values.

Solution:

$$\begin{aligned} V^2_{\pi_0}(\uparrow x) &= R(\uparrow x, M_z) + \gamma \sum_{s'} T(\uparrow x, M_z, s') V^1_{\pi_0}(s') \\ &= R(\uparrow x, M_z) + \gamma \cdot 0.5 \cdot V^1_{\pi_0}(\uparrow z) + \gamma \cdot 0.5 \cdot V^1_{\pi_0}(\downarrow z) \\ &= 0 + \gamma \cdot 0.5 \cdot 100 + \gamma \cdot 0.5 \cdot 0 = 50\gamma \end{aligned}$$

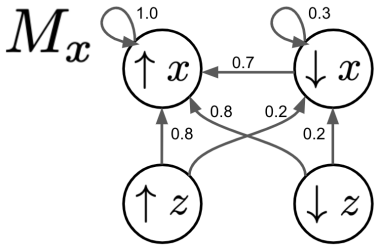
(f) We turn on a magnetic field pointing along the $\uparrow x$ direction, and this changes what transitions occur when we make a measurement, in the following ways.

1. If we are in state $\uparrow z$ or $\downarrow z$ and measure with M_x, we are 80% likely to transition to $\uparrow x$ and 20% likely to transition to $\downarrow x$.

2. If we are in state $\downarrow x$ and measure with M_x, we are 30% likely to stay $\downarrow x$ and 70% likely to transition to $\uparrow x$.
3. All other transitions stay the same.

Draw a diagram showing the new transitions between states via actions M_x. Be sure to write the probability of transitioning between states.

Solution:



(g) We continue with the magnetic field turned on. Suppose we get a reward of 100 for measuring the state $\uparrow z$ with M_z and now we also get a reward of 10 for measuring the state $\uparrow x$ with M_x, i.e., $R(\uparrow z, M_z) = 100$ and $R(\uparrow x, M_x) = 10$, respectively. Does the value of the policy π_0 for horizon of 2 starting in $\uparrow x$ change from part (e)? Why or why not? If it does change, please calculate it. Assume a discount factor of γ, meaning leave your answer in terms of γ. Where appropriate, first express your answer in terms reward function $R(s, a)$ for the appropriate state s and action a determined by the policy π_0, before reducing the answer to the numerical values.

Solution: $V^2_{\pi_0}(\uparrow x)$ remains unchanged under the new magnetic field.

$$\begin{aligned} V^2_{\pi_0}(\uparrow x) &= R(\uparrow x, M_z) + \gamma \sum_{s'} T(\uparrow x, M_z, s') V^1_{\pi_0}(s') \\ &= R(\uparrow x, M_z) + \gamma \cdot 0.5 \cdot V^1_{\pi_0}(\uparrow z) + \gamma \cdot 0.5 \cdot V^1_{\pi_0}(\downarrow z) \\ &= 0 + \gamma \cdot 0.5 \cdot 100 + \gamma \cdot 0.5 \cdot 0 = 50\gamma \end{aligned}$$

(h) We want to calculate the value of a new policy π_1:

- $\pi_1(\uparrow x) = \pi_1(\downarrow x) = \pi_1(\downarrow z) = M_x$
- $\pi_1(\uparrow z) = M_z$

We continue with the rewards and transition probabilities from part (g) above.

If we start in $\uparrow x$, what is value of new policy π_1 with a horizon of 2? Again, assume a discount factor of γ, meaning leave your answer in terms of γ. Where appropriate, first express your answer in terms reward function $R(s, a)$ for the appropriate state s and action a determined by the policy π_1, before reducing the answer to the numerical values.

Solution:

$$\begin{aligned} V^2_{\pi_1}(\uparrow x) &= R(\uparrow x, M_x) + \gamma \sum_{s'} T(\uparrow x, M_x, s') V^1_{\pi_1}(s') \\ &= 10 + \gamma \left(R(\uparrow x, M_x) + \gamma V^0_{\pi_1}(\uparrow x) \right) \\ &= 10 + \gamma(10 + \gamma \cdot 0) = 10 + 10\gamma \end{aligned}$$

(i) To maximize expected reward, for what range of discount factors γ should Pauli choose policy π_1 instead of π_0, for horizon 2 measurement of state $\uparrow x$?

Solution:

$$\begin{aligned} \text{Choose } \pi_1 \text{ when: } \quad V^2_{\pi_0}(\uparrow x) &< V^2_{\pi_1}(\uparrow x) \\ 50\gamma &< 10 + 10\gamma \\ \gamma &< \frac{1}{4} \end{aligned}$$

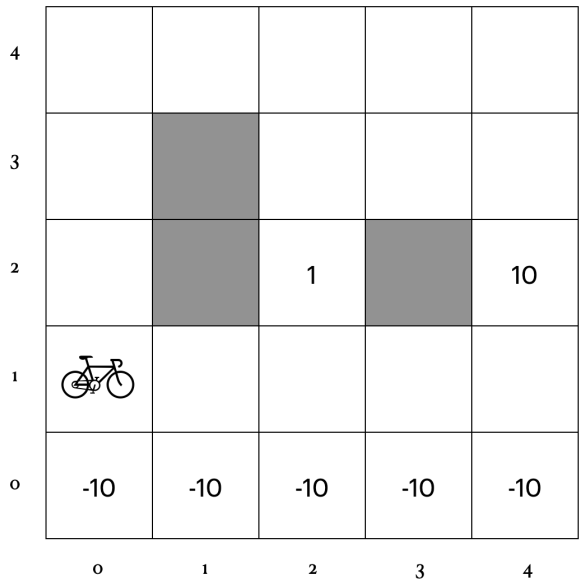
Name: _____

Griddy-Greedy World

8. (10 points) We're riding a bike in a 5-by-5 grid-world. We always start at the square with our bicycle, (0,1). The shaded squares (1,2), (1,3), and (3,2) are pillars. Four possible actions, North, South, East and West, allow us to navigate this world (the transitions will be detailed in the parts below).

   The state's rewards are written on the squares, regardless of the action taken. For instance, we receive a reward of -10 for any action taken in state (0,0). Those squares without numbers written on them all incur 0 reward for any actions taken in them.

   Squares (2,2) and (4,2) are special. When we reach either square, we get to make one final action, collect a 1 reward if we were in (2,2), or a 10 reward if we were in (4,2), and the game terminates. Our goal is to maximize the total reward we collect along the path we choose.



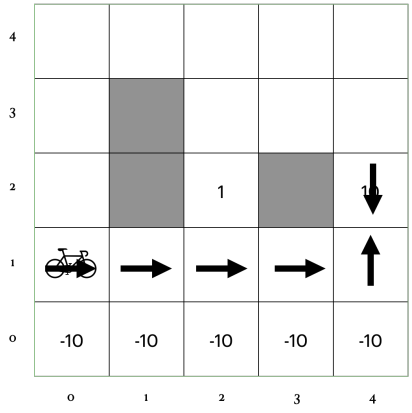
Name: _____

(a) In this part, we assume a discount factor of 1.0 and deterministic transitions. If our action would take us out of this world or into a pillar, we stay where we are. For instance, in state $(0, 1)$, a North action takes us to state (0,2); whereas in state (1,1), a North action leaves us in state (1,1).

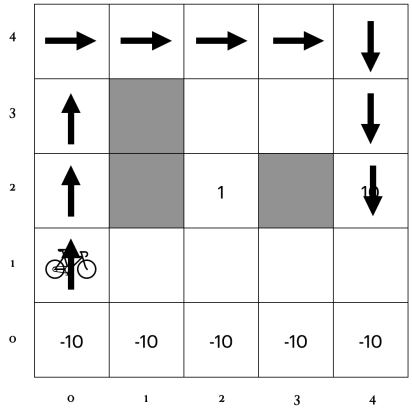
Suppose we run the value iteration algorithm for 1000 iterations. What would be the optimal value of our initial state (0,1)? Is the corresponding best path unique (unique up to but not including the last action chosen)? Draw out the actions as arrows taken at the states along your best (or one of the best) path(s).

Solution: Best value is 10. The best path is not unique.

One possibility: East, East, East, East, North, Any.



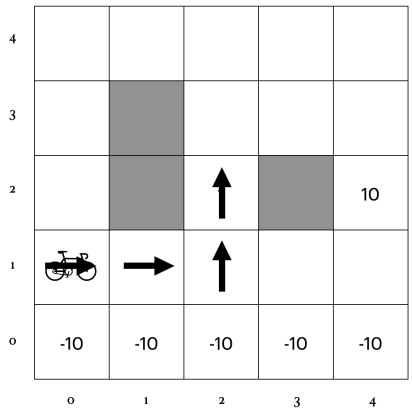
Another possibility: North, North, North, East, East, East, East, South, South, Any (recall that there's no discounting, we get to "procrastinate" all we want).



(b) In this part, we keep all of the setup from the previous part, except that we change the discount factor to 0.1.

Suppose we run the value iteration algorithm for 1000 iterations. What would be the optimal value of our initial state (0,1)? Is the best path unique (unique up to but not including the last action chosen)? Draw out the actions as arrows taken at the states along your best (or one of the best) path(s).

Solution: Best value is $(0.1)^3 \cdot 1 = 0.001$, and the path is unique this time: East, East, North, Any.



To see this, we can work backwards:

$$V^*(2,2) = 1$$
$$V^*(2,1) = 0 + \gamma \cdot V^*(2,2) = \gamma \cdot 1$$
$$V^*(1,1) = \gamma^2$$
$$V^*(0,1) = \gamma^3 \, .$$

Because of strong discounting, it is better to go for the reward 1 square, than for the reward 10 square (where V^* with discount would be $\gamma^5 \cdot 10 = 0.0001$).

(c) In this part, we now assume a discount factor of 0.99 (close to 1). As before, if our action would take us out of this world or into a pillar, we deterministically stay where we are.

However, now all other transitions are no longer deterministic; instead, they all have a "noisy transition" probability. Specifically, with probability 0.5, the next state is the "intended" target state (i.e, the state in the direction indicated by the action). However, with probability 0.5, the next state will instead be one of the horizontal or vertical reachable neighbors of the target state (with equal probability).

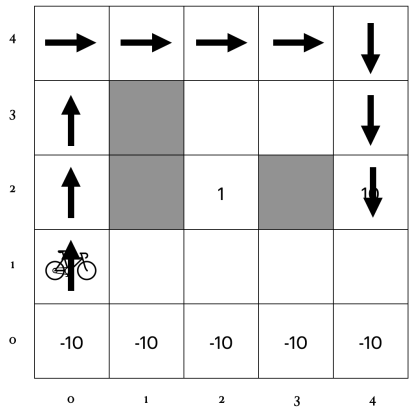
As a concrete example, in state $(0, 1)$, an East action takes us to state $(1, 1)$ with probability 0.5, or to one of the other three reachable immediate neighbors of $(1, 1)$ (which are $(0, 1)$, $(1, 0)$, and $(2, 1)$ but not $(1, 2)$ because that is a pillar), each with probability $0.5 \cdot \frac{1}{3}$.

Would we still be able to run the value iteration algorithm? If yes, suppose we run the value iteration algorithm for 1000 iterations. What would be the best path (or one of the best paths) with respect to the expected value? Draw out the path.

If we cannot run the value iteration algorithm, explain what variable(s) might be missing that prevents us from running value iteration in this part.

Solution: Yes, we'd still be able to run value iteration. This is because we still have full access to both the transition and reward models (despite the fact that the transition model is more complicated now).

The path with the best expected value is now unique: North, North, North, East, East, East, East, East, South, South, Any.



This avoids moving along row 1, where there is a substantial chance we will noisily transition to the bottom row and get one or more -10 rewards as part of our overall path reward. Similarly, we want to stay as far away as possible from the (2,2) square, as we might noisily transition there and only get reward of 1.

(d) In this part, we keep all of the setup from part (c), including noisy transitions. We will try reinforcement learning using Q-learning. We use a learning rate of 0.2 and "greedy" action selection (that is, we always pick action with best Q value). In case of action selection ties, we randomly select from the tied actions.

We run the Q-learning algorithm for 10000 episodes. Each episode starts in the square (0,1), and we take actions until the game in that episode terminates.

Draw out the best path learned by our Q-learning algorithm.

Solution: All solutions are the same as part (c). Even though we are running a purely greedy strategy, the random action tie-breaks as well as the noisy transitions will "nudge" us off any particular path – as long as we run the algorithm long enough. Indeed, since we run one-thousand episodes we're highly likely to thoroughly explore and learn near-optimal Q values on this small board.