Gradient descent

This week's lab and homework explore the concept of machine learning as optimization, building on the lecture and lecture notes on margin maximization and gradient descent.

Group information

1) Explore gradient descent

We've established that machine learning problems can be posed as optimization problems. We begin by studying general strategies for finding the minimum of a function. In general, unless the function is convex, it may be computationally difficult to find its global minimum.

Note: A function is convex if the line segment between any two points on the graph of the function lies above or on the graph.

We will sometimes study convex objectives, but in other cases we will content ourselves with finding a local minimum (where the gradient is zero) which may not be a global minimum. One method to find a local minimum of a function is called gradient descent (there are better ways, but this one is simple and computationally efficient in high dimensions and with lots of data). The idea is that we start with an initial guess, x_0, and move "downhill" in the direction of the gradient, leading to an update step

$$x^{(1)} = x^{(0)} - \eta \nabla_x f(x^{(0)})$$

where η is a "step size" parameter with the constraint that $\eta > 0$. We continue updating until $x^{(i+1)}$ does not differ too much from $x^{(i)}$. This approach is guaranteed to find the minimum if the function is convex and η is sufficiently small.

The questions below are concerned with running gradient descent on the parabola

$$f(x) = (2x + 3)^2 \ .$$

1A) Formulate the update rule that will be executed on every step when performing gradient descent on $f(x)$. You may use `eta` and `x` in your Python expression, where `eta` represents η.

x ← x - eta * (2*(2*x+3)*2)

Check Syntax | Submit | View Answer | Ask for Help **100.00%**

You have infinitely many submissions remaining.

1B)

What is the optimal value of x that minimizes f? -3/2

Submit | View Answer | Ask for Help **100.00%**

You have infinitely many submissions remaining.

1C) This question asks you to explore convergence of gradient descent for our $f(x)$, to see how the step size affects the rate of convergence and whether there are oscillations or lack of convergence.

We implement minimization of $f(x)$ using a function `t1`, which runs gradient descent for the minimization of $f(x)$. We halt the algorithm when the value of x changes by less than 10^{-5}. In general, we may use some small tolerance such as this to say whether gradient descent has *converged*. Conversely, *divergence* is defined as being when x will not converge to a single finite value, even with an infinite number of updates.

When you click Submit, the tutor question generates a plot of $f(x)$ in blue and the history of x values you have tried in red with a blue 'x' at the initial x value. You can change the values for `step_size` or `init_val` and click Submit again (all submits get 100%).

Experiment with the following step sizes: [0.01, 0.1, 0.2, 0.3]. For which one(s) does $x^{(k)}$ converge without oscillation? For which one(s) does $x^{(k)}$ diverge?

```
def run():
    return t1(step_size= 1/8, init_val = 0)

```

Run Code | Submit | View Answer | Ask for Help **100.00%**

You have infinitely many submissions remaining.

Why do oscillations happen for some values of `step_size` and not others? To gain insight into this behavior (and to understand how gradient descent proceeds), we note that we can rewrite our update rule:

$$x^{(k+1)} = x^{(k)} - \eta \nabla_x f(x^{(k)})$$

in the form:

$$z^{(k+1)} = \alpha z^{(k)}$$

for some $z = x - C$, and then consider how or if z approaches 0 for large k (or equivalently, x approaches C for large k). In particular, this formulation will soon be useful for us to analyze how the step-size η affects convergence.

1D) Show that the gradient descent update rule for our function $f(x) = (2x + 3)^2$ can be written in the form:

$$x^{(k+1)} + 3/2 = (1 - 8\eta)(x^{(k)} + 3/2)$$

and equivalently in the form

$$z^{(k+1)} = \alpha z^{(k)}$$

by defining $z^{(k)}$ and α appropriately. What is $z^{(k)}$ in terms of $x^{(k)}$? What is α in terms of η?

Written in this form, we make the following key observation: **we can think of our gradient descent step as simply a multiplication of the previous value by α on each step.**

Inductively, we can see that the value of z (and x) at step k is related to the initial value as

$$z^{(k)} = \alpha^k z^{(0)}$$

and equivalently

$$x^{(k)} + 3/2 = (1 - 8\eta)^k (x^{(0)} + 3/2) \ .$$

(Note: if this is not clear, please feel free to join the help queue.)

The above equations are useful because they explain the relationship between the initial value of z (or x) and the output of gradient descent as the repeated multiplication by the same factor α on each step. From this, we can identify cases of convergence, oscillation, and divergence by the following values on α:

$\alpha > 1$	**Gradient descent diverges without oscillation;** $z \to \infty$
$\alpha = 1$	$z^{(k)} = z^{(0)}$**, so no gradient descent steps occur**
$1 > \alpha \geq 0$	α^∞ **approaches 0, so gradient descent converges;** $z \to 0$
$0 > \alpha > -1$	α^∞ **approaches 0 while changing signs every step, so converges with some oscillation**
$\alpha = -1$	**At every step, the sign of** z **flips. Gradient descent oscillates between** $z^{(0)}$ **and** $-z^{(0)}$ **endlessly**
$-1 > \alpha$	**Gradient descent diverges with oscillation, since** z **grows but the sign of** z **flips at every step**

Since our ultimate goal is convergence, we are interested in the cases where $|\alpha| < 1$.

We can use the rules above about α to reason about how η affects convergence, given that $\alpha = 1 - 8\eta$.

Answer the following questions algebraically (using the expressions above).

1E)

What is the range of step size, as an interval, that causes x to converge to the global minimum starting from an arbitrary initial value? Use ( ) for open intervals and [ ] for closed intervals

(0, 0.25)

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

Does your algebraic answer agree with your numerical experiments above?

1F)

What is the largest step size that causes x to converge without oscillating?

1/8

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

How many iterations are needed for convergence in this case? Run `t1` with this value of step size.

1G)

For a step size of 0.1, is there an initial value of x_0 for which x does not converge to the global minimum? No

Submit View Answer Ask for Help 100.00%

You have infinitely many submissions remaining.

1H) What value(s) of step size in the set $\{0.1, 0.11, 0.12, 0.13, 0.14, 0.15\}$ makes gradient descent take the most steps before convergence? Find the answer algebraically and enter your value(s) in a Python list. (Hint: think about the magnitude of $1 - 8\eta$ and how this might affect the rate of convergence).

Enter your answer as a python list: [0.1, 0.15]

Submit View Answer Ask for Help 100.00%

You have infinitely many submissions remaining.

Now try running `t1` with the above step sizes. Which ones are slowest? Which ones oscillate? Do these behaviors match with your algebraic results?

2) Where to meet?

A group of friends is planning to host a baby shower over the weekend. They want to find a location for the party that minimizes the sum of *squared* distances from their houses to the location of the party. Assume for now that they can host the party at any location in the town.

Assuming that the friends live in a 1-dimensional town, solve the following problems:

2A) Pose this problem as an (unconstrained) optimization problem. Assume there are n friends and the i-th friend is located at l_i. Denote the location of the party by p. What is the objective as a function of p? Write it down.

2B) Compute the gradient (write down/show your computation). Where is it zero?

2C) Which of the following is true?

- [x] There is necessarily a unique location that minimizes the objective function
- [ ] The optimization problem may have local minima that are not global minima
- [ ] The party can be always be hosted in one of the houses without loss of optimality
- [x] There is necessarily a choice of step size that makes gradient descent converge with oscillations for this problem

Submit View Answer Ask for Help 100.00%

You have infinitely many submissions remaining.

Regression

During this week, we are exploring regression. Along the way, we will also deepen our understanding and experience with gradient descent, particularly as applied to regression. The previous notes on gradient descent will be useful, as well as the notes on regression.

In this lab, we will start by running code for solving regression problems and doing gradient descent on the mean square loss to develop an understanding for how it behaves. In the homework, you will implement all of the important functions necessary to create these demos.

Exploring gradient descent for regression and classification

In many problems, we want to predict a real value, such as the actual gas mileage of a car, or the concentration of some chemical. Luckily, we can use most of a mechanism we have already spent building up, and make predictors of the form:

$$y = \theta^T x + \theta_0$$

This is called a *linear regression* model.

We would like to learn a linear regression model from examples. Assume X is a d by n array (as before) but that Y is a 1 by n array of *floating-point* numbers (rather than +1 or -1). Given data (X, Y) we need to find θ, θ_0 that does a good job of making predictions on new data drawn from the same source.

We will approach this problem by formulating an objective function. There are many possible reasonable objective functions that implicitly make slightly different assumptions about the data, but they all typically have the form:

$$J(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} L(x^{(i)}, y^{(i)}, \theta, \theta_0) \text{ (without regularization)}$$

$$J(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} L(x^{(i)}, y^{(i)}, \theta, \theta_0) + \lambda R(\theta) \text{ (with regularization)}$$

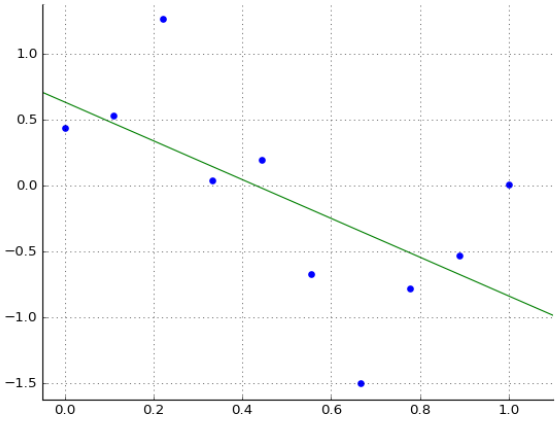
where L is our loss function and R is our regularization in terms of θ. For regression, we most frequently use *squared loss*, in which

$$L_s(x, y, \theta, \theta_0) = (\hat{y} - y)^2 = (\theta^T x + \theta_0 - y)^2$$

where our prediction is $\hat{y} = \theta^T x + \theta_0$ and y is our actual data.

We will come back to our regularization later...

In this lab, we will experiment with linear regression, gradient descent, and polynomial features. We will start with a simple data-set with one input feature and 10 training points, which is easy to visualize. In the plot below, the horizontal axis is our x value, and the vertical axis is the corresponding y value.



1) Least squares regression

In least squares regression problems, we assume that our objective function $J(\theta, \theta_0)$ comes without a regularization term; in other words,

$$J(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} L(x^{(i)}, y^{(i)}, \theta, \theta_0).$$

Recall from the lecture and notes that it is possible to solve a least-squares regression problem directly via the matrix algebra expression for the parameter vector θ in terms of the input data X and desired output vector Y. In this section, we will explore this *analytic* solution strategy.

For the rest of this lab, we will be computing solutions by transforming our one-dimensional input features with a polynomial basis. Specifically, we will be transforming our single input feature, which we will call x, into the vector $\phi(x) = (x^1, ..., x^k)$ if we are using a k-th order basis. This means that solutions to our regression problem will take the form

$$\theta^T \phi(x) + \theta_0 = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_k x^k$$

To answer the next set of problems, you are given the function `t1` to experiment with, which:

- Takes as input `order`, which is the order of the polynomial basis.
- Outputs θ and θ_0 that minimizes the regression objective $J(\theta, \theta_0)$.
- Finds θ and θ_0 **analytically** (we will explore the analytical solution in the homework).

In the codebox below, experiment with `t1` and try different values of order k to examine the effects of the order of the polynomial basis on the resulting solution. Based on your observations, answer the questions below the codebox.

To answer 1A and 1B below, you should try to answer the questions first without running `t1`, and then run `t1` to match with your expectations.

```
def run():
    return t1(order=0)

```

Run Code | Submit | View Answer | Ask for Help 100.00%

You have infinitely many submissions remaining.

1A) Consider the solution with the 0th-order basis (the regression polynomial described by only θ_0). What is the shape of the solution that you expect, and why? How does this solution depend on training data? Does this match what you observe in running `t1` with order 0?

1B) Polynomial order 9 is particularly interesting. What do you expect to be true about the solution, and why? (Hint: you are given 10 points in the training data). Does this match what you observe in running `t1` with order 9?

To answer the questions below, you should run the codebox above with various orders from 1 to 9 and observe plots and values of θ, θ_0. Make sure to click "submit".

1C) From your observation in the previous question, what is problematic if the polynomial order gets too big?

1D) Look at the values of θ and θ_0 you obtain, shown in the results. How does the magnitude of θ change with order? (You don't have to answer this quantitatively.)

1E) What polynomial order do you feel represents the hypothesis that will be the most predictive for new data?

1F) When we run 10-fold cross validation on this data set, varying the order from 0 to 8 (because we train on 9 training data points for each instance of 10-fold cross validation), we get the following mean squared error results:

```
[0.69206670716618535, 0.53820006043438084, 0.73424762793041687, 0.2835495578961193, 0.74338564580774558,
0.61422551802155112, 6.156711267187811, 356.8873742619765, 408.34678081302491]
```

What is the best order, based on this data? Does it agree with your previous answer?

1G) Abstractly, if we were to use a polynomial model with orders higher than 9 (e.g., 12), should it be possible for that polynomial to go through all the points?

1H) Now, actually run `t1` with orders higher than 9 (e.g., 12). Does the learned polynomial go through all the points? Remember that `t1` is fitting the polynomial analytically (directly using matrix inversion) as discussed in the notes on regression. Would we expect matrix inversion to work well for models with orders higher than 9 for this data?

2) Regularizing the parameter vector

Recall that we can add a regularization term $R(\theta)$ to the empirical risk. If we use a squared-norm regularizer, we get the so-called *ridge regression* objective:

$$J_{ridge}(\theta, \theta_0) = \frac{1}{n}\sum_{i=1}^{n} L_s(x^{(i)}, y^{(i)}, \theta, \theta_0) + \lambda\|\theta\|^2$$

The procedure `t2(order, lam)` performs ridge regression on polynomial features of order `order` with regularization coefficient `lam`. In the resulting plots, `t2` draws the original least-squares solution in orange, and raws the regularized version in green.

Using a 9-th order polynomial feature function, experiment with λ. Look at the detailed results after submitting your code, paying attition to the visualization of the polynomial that was fit using your chosen λ.

```
def run():
    return t2(order=9, lam = 0.1)

```

Run Code | Submit | View Answer | Ask for Help **100.00%**

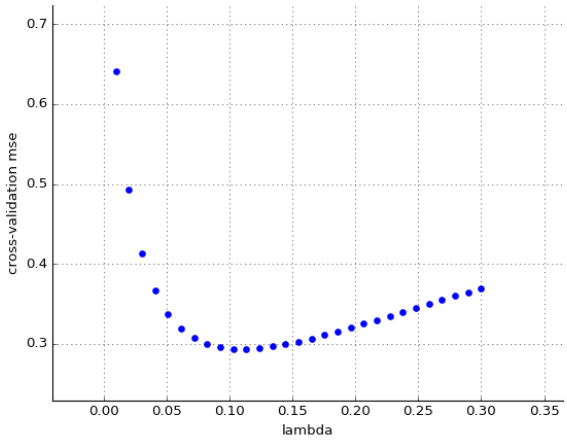
You have infinitely many submissions remaining.

2A) What happens to both $J_{ridge}(\theta, \theta_0)$ and the learned regression line, with very large (e.g., infinite) and very small (0) values of λ?

2B) If our goal is to solely minimize $\frac{1}{n}\sum_{i=1}^{n} L(x^{(i)}, y^{(i)}, \theta, \theta_0)$, what would be the best value of λ?

2C) What value of λ do you feel will give good performance on new data from the same source?

2D) When we run leave-one-out cross validation on this data set, using 9th order features, varying λ from 0.01 to 0.3, we get the following plot:



What is the best value of λ, in terms of generalization performance, based on this data? Does it agree with your previous answer? Is it the same as the best value for performance on the training set?

3) Gradient descent

Computing the analytic solution requires inverting a d by d matrix; as the size of the feature space increases, this becomes difficult. In addition, there are lots of other useful machine-learning models for which there is no closed-form analytic solution. So, we'll play with gradient descent here and see how it works. The procedure

```
t3(order, lam, step_size, max_iter)
```

performs gradient descent on the ridge regression objective using polynomial features of order `order`. You can specify the maximum number of iterations (we capped it at 10,000 to keep from killing the server) and the step size. It will print a convergence plot (objective value versus iteration number) and then show the solution it found in green and the analytic solution in orange for comparison.

Keeping $\lambda = 0$, experiment with this method for solving regression problems. (Note the difference in the plots when the solution diverges and when the solution converges very quickly.) Click 'Show/Hide Detailed Results' after submitting your code to view the convergence plot generated.

```
1 def run():
2     return t3(order=9, lam=0, step_size= 0.001, max_iter = 10000)
3
```

Ask for Help **100.00%**

You have infinitely many submissions remaining.

Here is the solution we wrote:

```
def run():
    #return t3(order=2, lam=0, step_size= 0.3, max_iter = 1000)
    # Note: Ignore this solution code! Click "Show Detailed Results" to view the plots.
    return 'None'
```

3A) With `max_iter = 1000` in the code snippet above, what step sizes were needed to match the analytic solution almost exactly for 1st and 2nd order bases? (Use the convergence plot to help decide on whether the two curves "match almost exactly.")

3B) What step size allowed you to get similar curves to the analytical solution for 3rd order? (Use the convergence plot to help decide whether the two curves are "similar.")

3C) For 9th order, play around for a while (no more than 10 minutes) to see how close you can get to the analytic solution. How does it compare to the analytical solution?

Neural Networks

For this lab, you will need to understand the material in the notes on neural networks up through and including section 6 on loss functions.

1) Exploring neural networks

Notes on the functions in the code boxes below:

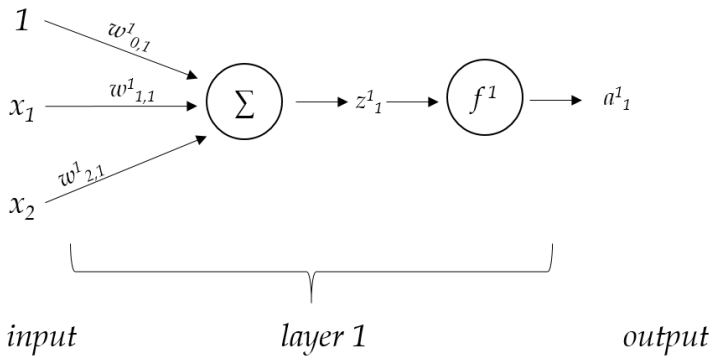
- The initialization of weights in a neural network is random, so subsequent runs might produce different results.
- `iters` indicates how many single steps of SGD to run, each using one training point.
- Pay attention to the decision boundaries, shown in the answers after you run the code. Also note that the accuracy of the network on the training data is printed above the graph.

1.1) Simple separable data set

Here's a very simple data set.

```
X = np.array([[2, 3, 9, 12],
              [5, 2, 6, 5]])
Y = np.array([[1, 0, 1, 0]])
```

The code below runs a very simple neural network composed of a single sigmoid unit with two inputs, using negative log likelihood (NLL) loss:



The function `t1` in the code box below plots the classifier found after training for the specified number of iterations with the specified "learning rate" (step size). The classifier predicts label +1 if the output is greater than 0.5 and label 0 otherwise -- remember that the range of the sigmoid function is $(0, 1)$.

1.1.A) What is the shape of the separator produced by this network? Explain.

1.1.B) Are you able to get relatively consistent 100% accuracy with some combination of `iters` and learning rate `lr`?

```
def run():
    return t1(iters=1000, lr=0.2)

```

Run Code | Submit | View Answer | Ask for Help **100.00%**

You have infinitely many submissions remaining.

1.2) Not-XOR

The XOR dataset is a classic example showing the need for more than one layer of non-linear units. Here, we will consider a function outputting the negation of XOR, Not-XOR:

```
X = np.array([[0, 1, 0, 1],
              [0, 1, 1, 0]])
Y = np.array([[1, 1, 0, 0]])
```

1.2.A) Draw a plot showing the locations of these data points in x_1, x_2 coordinate space, with the corresponding labels. Now consider a network with no hidden layers as in part 1.1 above, which just has input units connected (via weights) to an output sigmoid unit. Can this network learn a separator for the given dataset? Explain.

1.2.B) Assume we add a hidden layer with ReLU activation units, connected to sigmoid unit in the output layer. Draw by hand the smallest network that can separate the data. Show all the weights (and biases) for the network units. Explain how it works (with reference to your earlier drawing of the Not-XOR data in x_1 and x_2 space), and explain why a smaller network won't work. (Try to spend just 10 minutes on this, then ask for tips!)

1.2.C) In the run box below, first try to see if you can get a network of the size you found above to separate the data reliably (several times in a row). If not, try larger networks. Explain what's going on.

The `t2` function in the code below runs on the dataset using a network with a single hidden layer of ReLU units, a sigmoid output layer, and NLL loss. You can specify the number of hidden units (pretty please, don't pass in anything other than a positive integer!), the number of iterations, and the learning rate.

```
def run():
    return t2(hidden_units=2, iters=10000, lr=0.05)

```

Run Code | Submit | View Answer | Ask for Help **100.00%**

You have infinitely many submissions remaining.

1.3) Hard data set

In this next example we use a much harder data set that is **barely** separable (although not linearly). We'll try a two-layer network architecture.

1.3.A) Running the code below, can you get this architecture to separate the data set well (at least 95% accuracy)? I f you can't, explain why not. If you can, explain why. Make sure your accuracy is reliable by running the code several times.

1.3.B) Do you think it's a good idea to try to find a "perfect" separator for this data? Explain.

The network here has two hidden layers of ReLU units and a sigmoid output unit with NLL loss. In the `t3` function below, you can specify the number of hidden units in each hidden layer, the number of iterations, and the learning rate.

```
def run():
    return t3(hidden_units=18, iters=10000, lr=0.05)

```

Run Code | Submit | View Answer | Ask for Help **100.00%**

You have infinitely many submissions remaining.

2) Crime and Punishment

One important part of designing a neural network application is understanding the problem domain and choosing

- A representation for the input
- The number of output units and what range of values they can take on
- The loss function to try to minimize, based on actual and desired outputs

We have studied input representation (featurization) in a previous lab, so in this problem we will concentrate on the number of output units, activation function on the output units, and loss function. These should generally be chosen jointly.

Just as a reminder, among different loss functions and activation functions, we have studied:

- Activation functions: linear, ReLU, sigmoid, softmax
- Loss functions: hinge, negative log likelihood (NLL a.k.a. cross-entropy), quadratic (mean squared)

For each of the following application domains, specify good choices for the number of units in the output layer, the activation function(s) on the output layer, and the loss function. When you choose to use multiple output units, be very clear on the details of how you are applying the activation and the loss. **Please write your answers down!**

2.A) Map the words on the front page of the New York Times to the predicted (numerical) change in the stock market average.

2.B) Map a satellite image centered on a particular location to a value that can be interpreted as the probability it will rain at that location sometime in the next 4 hours.

2.C) Map the words in an email message to which one of a user's fixed set of email folders it should be filed in.

2.D) Map the words of a document into a vector of outputs, where each index represents a topic, and has value 1 if the document addresses that topic and 0 otherwise. Each document may contain multiple topics, so in the training data, the output vectors may have multiple 1 values.

Filters

For this lab:

- You will need to understand the material in the notes on convolutional neural networks.
- We suggest that you write down your answers/explanations
- It will be useful to sketch the networks, input/outputs, and convolution operations to answer the questions in this Lab.

1) Filters

Assume that the input to a CNN is a one-dimensional (1D) image, that is, a vector of values corresponding to light intensity, A common operation that we want to perform on images is to detect where there are "edges" (i.e., rapid changes in the intensity) since these sometimes correlate with changes in material, lighting, depth, etc. The early layers of the visual systems of animals appear to do processing of this type.

Let's start simple and imagine that the input "image" is a vector of three elements, $[x_1, x_2, x_3]$. We want to build a network that has a high output when there is a rapid change of intensity in the image and low otherwise. What this will mean is that we will call x_2 an "edge" location iff x_1 is high and x_3 is low or vice versa. Another way of saying this is that we want to detect locations where $|x_3 - x_1| > 1$. (In general, the choice of 1 is arbitrary and it should be a learned bias. **But here, we will use 1 as the threshold for all of the following problems**.)

Let's consider building an "edge classifier" using a single unit, with three inputs and ReLU activation, whose output should be greater than 0 for an edge location defined as above and 0 otherwise. The weights here are either -1, 0, or 1.

1A. Pick a set of weights and bias for this unit that can do this task (or determine if no such set of weights exist). Be prepared to explain your answer.

Enter a list of four numbers for the unit weights and bias $[w_1, w_2, w_3, b]$ or the string 'None' if no such weights exist: 'None'

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

1B. Now, let's consider a network with two units (each with three inputs) on the first layer and one unit (with two inputs) on the output layer. All units have ReLU activation. Pick a set of weights and biases for these units that can do the edge detection task or determine if no such set of weights exist. Be prepared to explain your answer.

Enter a list of three lists of weights and bias, first for the two units on the first layer (in the order of $[w_1, w_2, w_3, b]$) and then for the output unit (in the order of $[w_1, w_2, b]$), or 'None' in no such weights and biases exist.

[[1, 0, -1, -1], [-1, 0, 1, -1], [1, 1, 0]]

Ask for Help **100.00%**

You have infinitely many submissions remaining.

Solution: `[[1, 0, -1, -1], [-1, 0, 1, -1], [1, 1, 0]]`

1C. Now, let's think about 3x3 "images" and try to detect diagonal "edges." Consider a 3x3 image patch

$$x_1, x_2, x_3$$
$$x_4, x_5, x_6$$
$$x_7, x_8, x_9$$

We might care about patterns that have:

- $|(x_1 + x_2 + x_4) - (x_9 + x_8 + x_6)| > 1$ (diagonal edge x_3-x_5-x_7), or
- $|(x_3 + x_2 + x_6) - (x_7 + x_8 + x_4)| > 1$ (diagonal edge x_1-x_5-x_9)

We can do this with four ReLU units in the first layer and one output unit. Pick a set of nine weights (and a bias) for **just one** of the first layer units, that responds to one of the four patterns.

Enter a list of 10 numbers: 9 weights (w_i) and a bias. [1, 1, 0, 1, 0, -1, 0, -1, -1, -1]

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

2) Convolution

The weights for these units that we have constructed are sometimes called a "filter kernel." It's a (usually) small vector (for 1D images) or matrix (for 2D images) that is "tuned" to produce a high value for certain kinds of patterns/features (such as edges). Note that you can think of these filters allowing a **limited receptive field** which only focus on part of the image instead of the whole image. In order to detect the patterns/features anywhere in the (large) input image, we **slide** the filters over the image and produce a new feature map. This process is known as **convolving the filter with the image** and this is where CNNs get their name [*see footnote 1*].

Assume that we have a 1D input x of size 15 (note that the x index starts from 1 here):

$$x = [x_1, x_2, \cdots, x_{15}] = [0, 0, 0, 2, 2, 2, 4, 4, 4, 2, 2, 2, 0, 0, 0]$$

Consider we have a filter $f = [w_1, w_2, w_3]$ of size 3. Now we want to convolve this filter f with the 1D input x to obtain a 1D feature map $\phi(x)$ of the same size 15. When we perform convolution at input location i, the feature map at location i is $\phi_i =$

$w_1 \cdot x_{i-1} + w_2 \cdot x_i + w_3 \cdot x_{i+1}$. We then send the feature maps to non-linear activation functions $\sigma(\cdot)$ which might, for example, be ReLU units.

Note that we will assume zero padding, such that the input value x_i is 0 if some part of the filter falls outside the image, i.e., $x_i = 0, \forall i \notin \{1, 2, \cdots, 15\}$. Also, we slide the filter over the input by the stride, which is 1 here.

2A. Visualize the input in terms of gray-scale plot. (You are free to make assumptions about your gray-scale mapping, etc.)

2B. You are given a ReLU unit that detects $x_3 - x_1 > 1$. Enter a list of indices i where the output of this ReLU unit is positive, i.e., $\sigma(\phi_i) > 0$:

Enter a list of indices in ascending order: [3, 4, 6, 7]

Ask for Help **100.00%**

You have infinitely many submissions remaining.

Solution: `[3, 4, 6, 7]`

2C. You are given a ReLU unit that detects $x_1 - x_3 > 1$. Enter a list of indices where the output of this ReLU unit is positive, i.e., $\sigma(\phi_i) > 0$:

Enter a list of indices in ascending order: [9, 10 , 12, 13]

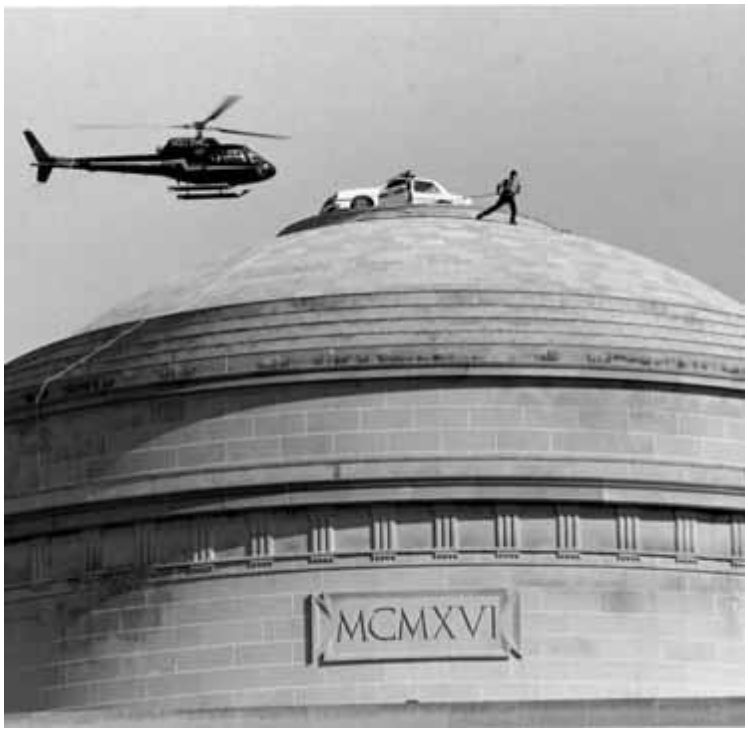
Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

Note that in the above processing, "step" patterns are detected anywhere in the image and independent of the overall "brightness." These are important properties for image processing.

3) Experiment with convolution

Executing the code below will convolve one of the filters f_1, f_2, f_3 (including the bias) with a familiar image and return an image that is white where the output is greater than 0, and black elsewhere. Thus, the image returned and shown in the detailed output is binary. Here's the original image:



3A. Try changing the filters and the bias to get a feeling for what is going on when a CNN learns the weights and biases for the filters. The range of values in the images after filtering is on the order of ± 100; that should help you pick the bias value. Click "Show/Hide Detailed Results" to see the output image.

```
# Some interesting filters to try:
f1 = ((1./8)*np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])).tolist()
f2 = ((1./8)*np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])).tolist()
f3 = ((1)*np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]])).tolist()
# You will need to change the bias to get nice results... think about the sign.
def run():
    return filter(weights=f3, bias=10)

```

Run Code | Submit | View Answer | Ask for Help **100.00%**

You have infinitely many submissions remaining.

3B. Consider these "features":

A) Points brighter than neighborhood
B) Vertical edges (right brighter)
C) Horizontal edges (top brighter)

Which filter from the code above best detects each feature?

Enter a python list of numbers (1, 2, 3 corresponding to the filters f1, f2, f3) for features A, B, C:

[3, 1, 2]

Submit | View Answer | Ask for Help **100.00%**

You have infinitely many submissions remaining.

3C. How is the output image affected if you increase or decrease the bias value? Explain your observations.

3D. Keeping the bias constant, change the coefficients of each filter array and consider how they affect the output image. What happens when the bias is zero? What happens when the bias is non-zero?

3E. Given a 1D black and white "image" (think of this as a sequence of 0's and 1's), we define an object to be a consecutive sequence of black pixels (0's) (e.g., 0101 has two objects, 10110011000 has three objects, etc.). Design a simple CNN that can count the number of objects in a 1D image. Specify your choice of filter. How many hidden layers and filters do you need? What is your choice of activation functions and why? Hint: you can use a fully connected layer at the last layer.

4) Conversion from Convolutional Layer to Fully Connected Layer

Consider a 1D input x of size 4, a 1D feature map $\phi(x)$ of the same size 4, and a filter f of size 3: $(w_1, w_2, w_3) = (5, 6, 7)$ with stride of 1. Assume that when doing convolutions, inputs that fall outside the image indices are treated as zeros, as in Problem 2. The effect of convolving a filter with the input layer x is actually equivalent to constructing a matrix M of weights between two layers of a fully connected network, such as we saw in HW 6, but where the same weight may occur more than once in the matrix, i.e.,

$$\phi(x) = M^\top x \ ,$$

where M is a 4 by 4 matrix.

4A. What is the convolution-equivalent matrix M? Hint: What is the feature map $\phi(x)$?

Enter the matrix M as a list of rows, where each row is a list of the matrix elements in that row (i.e., in the same format you would use to construct a 2D numpy array):

[[6, 5, 0, 0], [7, 6, 5, 0], [0, 7, 6, 5], [0, 0, 7, 6]]

Submit | View Answer | Ask for Help **100.00%**

You have infinitely many submissions remaining.

By looking at the matrix you can see the "sliding" of the filter during the convolution. In a fully-connected network every entry in the matrix is a different weight value that has to be learned; in contrast, in the convolutional case above you can see that each weight appears several times in the matrix (for essentially every output unit) and many weights are zero. The "re-use" of weights is referred to as "parameter sharing" or "parameter tying." This network can be trained with error back-propagation as we already understand it; care must just be taken to sum over all "error values" that a particular weight can contribute to.

4B. In general, we have a 1D input x of size d, a 1D filter f with size k and with stride of s, and 1D zero-paddings on both sides with size p. In the above problem 4A, we have $d = 4, k = 3, s = 1, p = 1$. Discuss what the size of feature map $\phi(x)$ is, in terms of general d, k, s, and p.

Footnotes

[Footnote at Problem 2] If you were already familiar with what a convolution is, you might have noticed that the definition given in the lab corresponds to a correlation and not to a convolution. Indeed, correlation and convolution refer to different operations in signal processing. However, in Neural Networks literature, most libraries implement the correlation (as described in this lab) but call it convolution. The distinction is not significant; in principle, if convolution is called for the network would learn the same weights, just flipped. For a discussion on the difference between convolution and correlation and the conventions used in the literature you can read section 9.1 (pages 327-329) from the deep learning book.

State Machines and MDPs

For this lab:

- You will need to understand the material in the notes on state machines and Markov decision processes.
- We suggest that you write down your answers/explanations and make sure you discuss and understand them during the Lab Check-off.

1) MDP Formulation

We will try to model some aspects of a very simple factory as a Markov decision process.

There is a single machine that has three possible operations: "wash", "paint", and "eject", with corresponding buttons. Parts are put into the machine, and each time you push a button, something is done to the part. It's an old machine, and not very reliable. However, it has a camera inside that can clearly detect what is going on with the part and will output the state of the part: either dirty, clean, or painted.

In this question, you will devise a policy that will take as input the state of a part and select a button to press, until finally you press the eject button (which sends you to a state which always transitions to itself and gets zero rewards).

- All parts start out dirty.
- If you perform the "wash" operation on any part, whether it's dirty, clean, or painted, it will end up clean with probability 0.9 and otherwise become dirty.
- If you perform the "paint" operation on a clean part, then with probability 0.8 it becomes nicely painted, with probability 0.1 the paint misses but it stays clean, and with probability 0.1 it dumps rusty dust all over the part and it becomes dirty.
- If you perform the "paint" operation on a painted part, it stays painted with probability 1.0.
- If you perform the "paint" operation on a dirty part, it stays dirty with probability 1.0.
- If you perform an "eject" operation on any part, the part comes out of the machine and this fun game is over.

You get reward +10 for ejecting a painted object, reward 0 for ejecting a non-painted object, and reward -3 for every action that is not "eject".

1A) Write out a careful and complete specification of the **state space**, **action space**, **transition model**, and **reward function**. Provide both a state diagram and a transition matrix.

1B) Use your intuition to find the infinite horizon optimal policy when the discount factor $\gamma = 1$ (no discount). What is its format?

1C) How does the policy from 2 change if $\gamma = 0.1$?

1D) How does the policy from 2 change if the horizon is 2?

Check this box and submit when you have finished question 1.

Submit | View Answer | Ask for Help **100.00%**

You have infinitely many submissions remaining.

2) Simple Value Iteration Example

We are going to look at a couple of two-dimensional "grid-world" examples, in which there is a robot that can move North, South, East, or West (N, S, E, W). It cannot move off the board. The transitions are somewhat noisy; when commanding a move there is a small chance that you will end up in one of the neighbor states of the desired state.

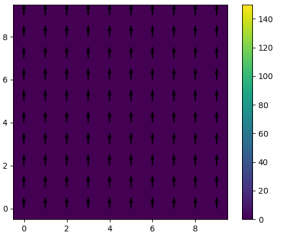
We have defined a special subclass of `MDP` called `GridWorld` for representing these MDPs. One can specify a particular "floor plan" with a list of strings of characters. Consider this world:

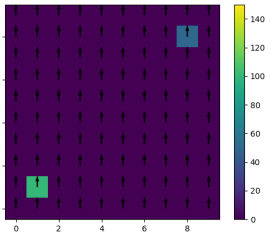
```
['.........',
 '........*.',
 '.........',
 '.........',
 '.........',
 '.........',
 '.........',
 '.........',
 '.$.......',
 '.........']
```

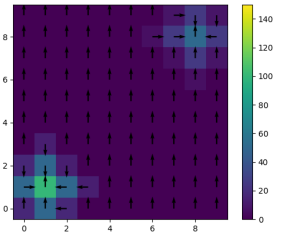
Each character corresponds to a square in the grid. The meanings of the characters are:

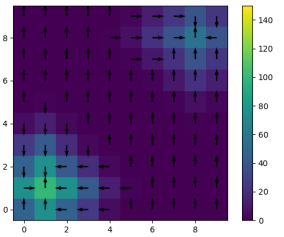
- '.' : a normally habitable square, from which the robot can move.
- '$' : reward of 100; a terminal state, every action leads to a zero-reward state that cannot be escaped.
- '*' : reward of 50 and next state is chosen uniformly at random from all occupiable states (this reward can be claimed multiple times).

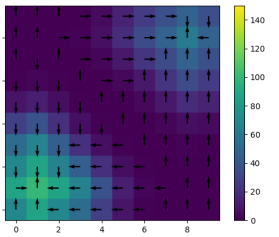
These are plots of the values of the states as we run 20 iterations of value iteration with $\gamma = 1$. The arrows represent the current best policy, pointing N, S, E, or W. Be sure you understand how the policy learned on iteration i relates to the value function for horizon i.

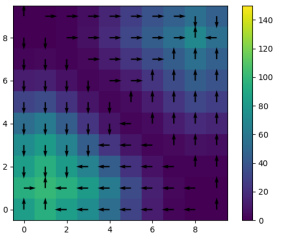
1 

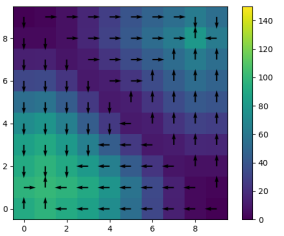
2 

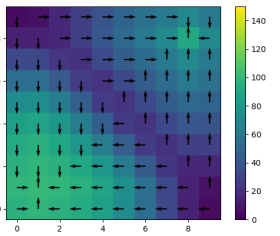
3 

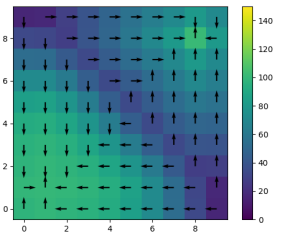
4 

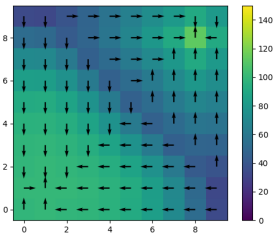
5 

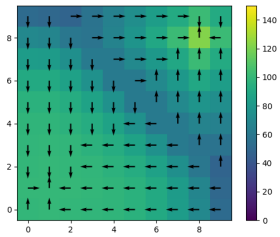
6 

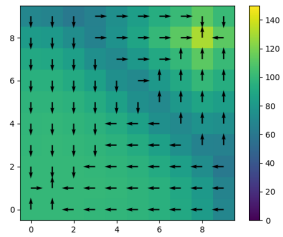
7 

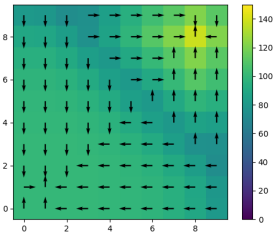
8 

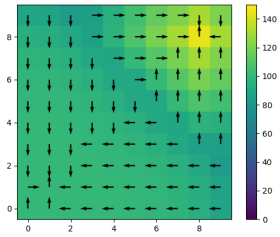
9 

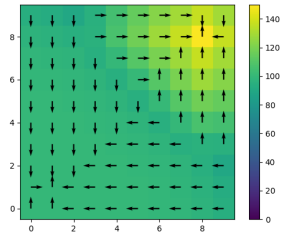
10 

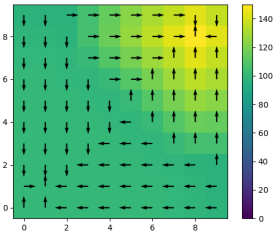
11 

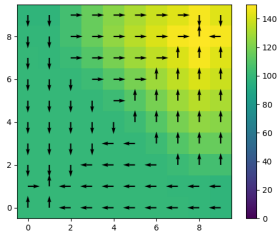
12 

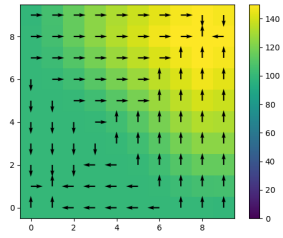
13 

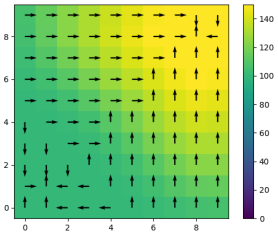
14 

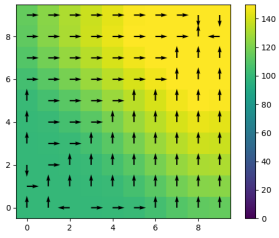
15 

16 

17 

18 

19 

20 

2A) In the first picture all the values are zero. After one iteration we have two non-zero states. What are they? What values do they have?

2B) What happens in roughly iterations 2 - 7?

2C) Look at iteration 11. Why does the upper right state now have higher value than the lower left one?

2D) What's happening around iteration 16?

2E) What's happening around iteration 20?

2F) If we keep going, what will the final map look like? Do we expect the robot to eventually terminate?

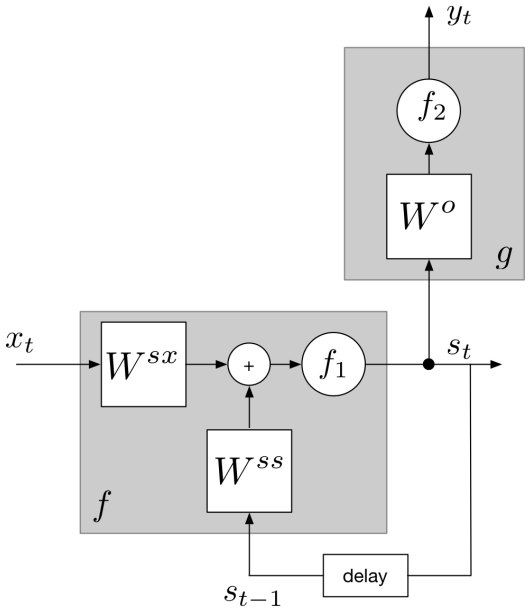
3) Autoregressive RNN

A neural network is a (learned) function that maps from input vectors to output vectors. A recurrent neural network (RNN) is a (learned) state machine that maps input sequences to output sequences. We will learn more about RNNs in week 11; for this lab, we will look at a very simple instance, relating to MDPs. In particular, an RNN has a transition function and an output function, each of which is defined in terms of weight matrices, offset vectors and activation functions, analogously to standard neural networks.

The behavior is defined as follows:

$$s_t = f_1(W^{ss} s_{t-1} + W^{sx} x_t + W_0^{ss})$$
$$y_t = f_2(W^o s_t + W_0^o)$$

where f_1 and f_2 are two activation functions, such as linear, softmax, or tanh. This arrangement is shown below, illustrating that it is an instance of the state machine model that we saw in the Week 9 exercises.



We would like to use an RNN to implement an autoregressive model, so that:

$$y_t = 1y_{t-1} - 2y_{t-2} + 3y_{t-3}$$

We will use an RNN in which:

$$x_t = y_{t-1}$$

This is what makes it "autoregressive." Assume that x_t is a scalar (1x1).

3A) If $y_0 = 5$ and all previous y values are 0, what are y_1, y_2, y_3?

Enter a list of three numbers for $[y_1, y_2, y_3]$: [5, -5, 0]

Submit | View Answer | Ask for Help **100.00%**

You have infinitely many submissions remaining.

3B) What is the best choice for f_1, where f_1 is one of the common activation functions (linear, softmax, tanh, sigmoid, relu) we've studied? Be prepared to explain your answer.

linear

Ask for Help **100.00%**

You have infinitely many submissions remaining.

Solution: linear

Explanation:

Be prepared to discuss your answer during the checkout.

3C) What is the best choice for f_2? Be prepared to explain your answer.

linear

Ask for Help **100.00%**

You have infinitely many submissions remaining.

Solution: linear

Explanation:

Be prepared to discuss your answer during the checkout.

3D) What is the smallest dimensionality for the state s that will allow this to be implemented exactly?

Enter a list of two numbers [a, b] for the dimensions $a \times b$ of state s :

[3,1]

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

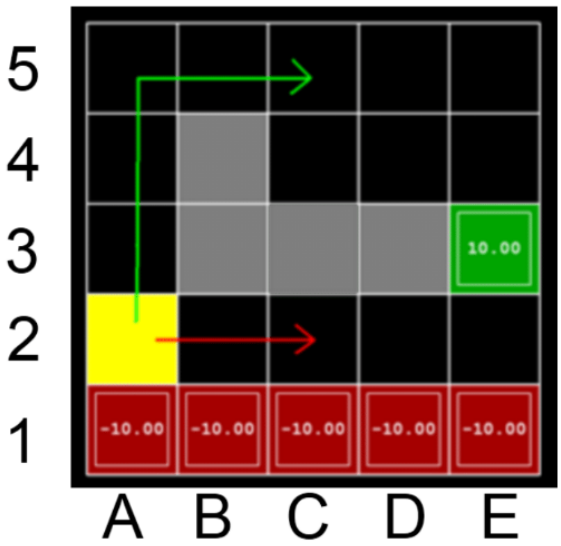
3E) Provide matrices W^{ss}, W^{sx}, W^{o}, W_0^{ss} and W_0^{o} that implement this model. You should also specify the initial state vector s_0 consistent with $y_0 = 5$ for your model; your initial state vector may contain non-zero elements.

For this, you should be sure to understand the notes on Reinforcement Learning.

1) MDPacMan: The Wind-up Cart Chronicles

During a summer hike with some of the friends you made in 6.036 (or 6.862), you accidentally stumble upon the cavernous lair of a giant Pac-Man who is trying to eat you in retribution for all you have done to him during your childhood.

You can see a layout of the cave below: the light grey squares are impassable terrain. Both the cliffs (red cells, get reward −10 for entering) and the cave exit (green cells, get reward +10 for entering) are terminal states. This means the game is over when you reach a -10 or +10 state (you die or escape).



You jump in a mining cart, denoted by the yellow square, to escape, which has the actions LEFT, RIGHT, UP, DOWN. Unfortunately the steering system has not been used in a while, and thus it will not always take the cart in its originally intended direction. An action will take the cart

1. In the direction intended with $1-2\tau$ probability,
2. In one of the two adjacent directions [1] with τ probability each, and
3. In the opposite direction with probability 0.

If the result of an action would take the cart into a wall or off the grid, it ends up in the same location. There is a constant "cost of living" C incurred on every action you take (i.e., there is a negative reward of magnitude C for each action taken).

This mining cart is no ordinary cart — it does not have a steering wheel! Instead, there are three dials, to control the parameters γ, τ, C, which are used by value iteration to determine the optimal path. The possible paths are the upper (green) path or the lower (red) path in the figure. Suppose the current setting of γ, τ, C yields the lower path by the −10 cliff, and you really would rather take the longer but safer path.

Please note that the variables τ and C are attributes of the environment, and γ is the discount factor used in value iteration.

[1]: By "adjacent", we mean the orthogonal directions right next to the intended direction; so for example UP and DOWN are directions adjacent to LEFT or RIGHT, and vice versa.

1.1) Longer path

How would you change each of the values of γ, τ, C to help your cart learn to take the longer path (the green one)? Assume you are changing **one of the variables at any given time**, not all three of them together.

1.1A) How do you change γ?

increase

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

1.1B) How do you change τ?

increase

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

1.1C) How do you change the magnitude of C?

decrease

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

1.2) On-line Q-learning

Now, imagine a situation where the cart is going to learn the optimal behavior by executing on-line Q-learning over a single lifetime while moving through the cave. Remember that unlike value iteration, Q-learning doesn't use known transition probabilities. Instead, it learns by interacting with the real world environment without ever constructing any explicit models for transition probabilities.

The cart starts with a Q value table that is empty (filled with 0s and some fixed action selection in case of Q value ties), and it turns out that with such a Q value table and a 0.5-greedy policy, the cart does not take the longer path.

How would you change the γ, τ, or C variable this time such that the cart learns to take the longer path?

For your reference, the Q-learning update formula is the following:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a'))$$

1.2A) How do you change γ?

don't change (no effect)

Ask for Help **100.00%**

You have infinitely many submissions remaining.

Solution: don't change (no effect)

Explanation:

Be prepared to explain your answer during your checkoff.

1.2B) How do you change τ?

don't change (no effect)

Ask for Help **100.00%**

You have infinitely many submissions remaining.

Solution: don't change (no effect)

Explanation:

Be prepared to explain your answer during your checkoff.

1.2C) How do you change the magnitude of C?

don't change (no effect)

Ask for Help **100.00%**

You have infinitely many submissions remaining.

Solution: don't change (no effect)

Explanation:

Be prepared to explain your answer during your checkoff.

1.3) Simulating multiple episodes

After being inspired (or scared) by the result of the previous problem, you decide to change the cart such that it uses a simulation of the cave to run the online Q-learning algorithm for multiple episodes. Then it computes its route in the real world

using the values in the learned Q value table and a greedy policy. In this scenario, how would you change the γ, τ, or C variable to make the cart take the longer path?

1.3A) How do you change γ?

increase

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

1.3B) How do you change τ?

increase

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

1.3B) How do you change the magnitude of C?

decrease

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

2) Q-Learning: Goal vs. SSP

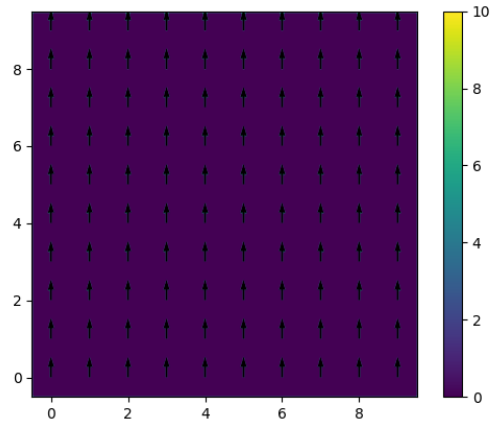
This problem will explore Q-learning in a simple 2D grid setting (like in last week's lab) with a single goal location. There are two typical ways to set up reward structures for domains with a goal state:

- **Goal-reward based**: Put a positive reward on taking any action from the goal state, and make the goal state a *terminal state*, from which every action leads to a zero-reward state that can never be escaped. Put zero rewards everywhere else.
- **Stochastic-shortest-path (SSP) based**: Put zero reward on the goal state, and make the goal state a *terminal state*, from which every action leads to a zero-reward state that can never be escaped. Put -1 rewards everywhere else.

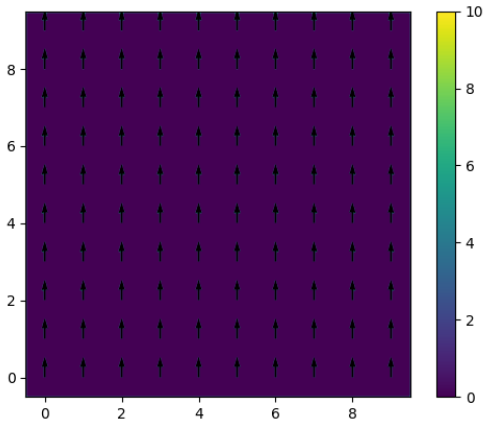
Some things to note about our implementation:

1. The agent follows an epsilon-greedy policy with $\epsilon = 0.1$;
2. Whenever the agent reaches the goal, we start a new episode on the next iteration by sampling a start state uniformly at random and following the epsilon-greedy policy from then on (until it reaches the goal again, then the process repeats).
3. In the case of a tie among $\arg\max_a Q(s, a)$ actions, the first action in the action space among the tied values is selected.

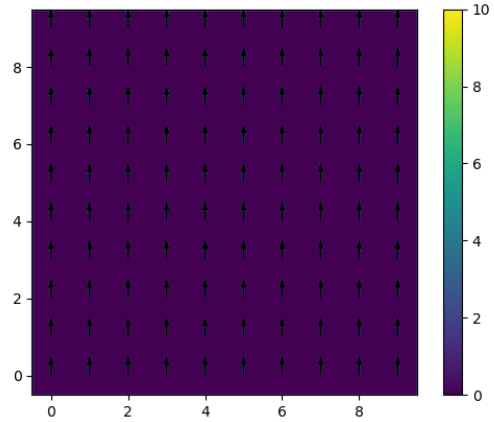
These are plots of the values of the states using the Goal-reward formulation as we run 50,000 iterations of Q-value learning, plotting every 5,000 iterations. **Note that the scale of the colors changes across the different plots, per the bar on the right of each plot.**


0 2 4 6 8
0 2 4 6 8
0 2 4 6 8 10

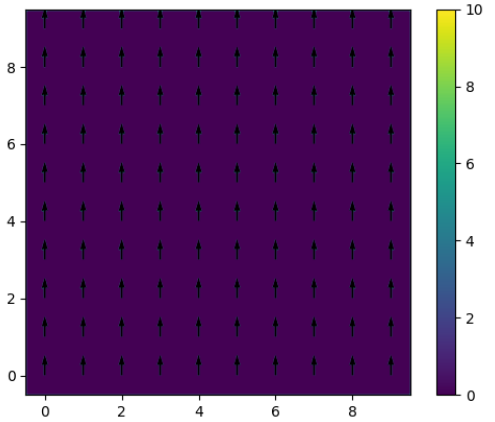
1


0 2 4 6 8
0 2 4 6 8
0 2 4 6 8 10

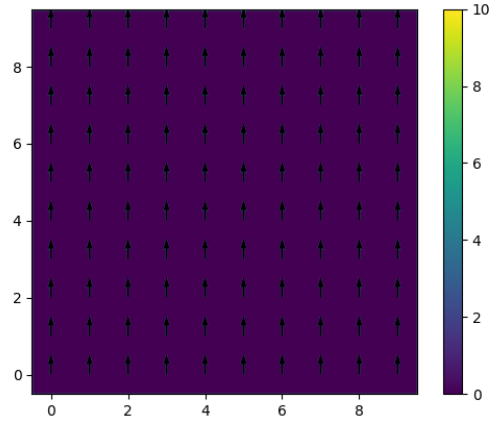
2


0 2 4 6 8
0 2 4 6 8
0 2 4 6 8 10

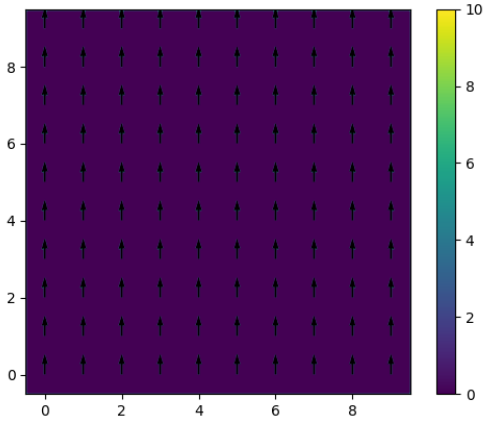
3


0 2 4 6 8
0 2 4 6 8
0 2 4 6 8 10

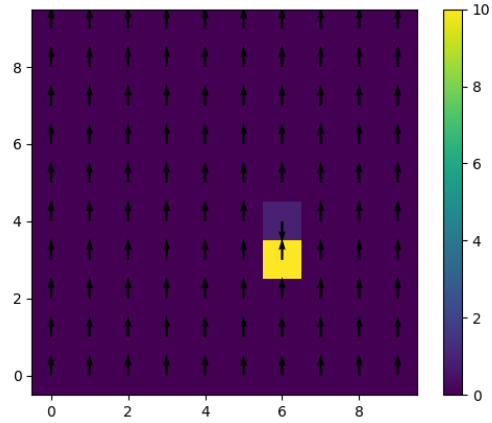
4


0 2 4 6 8
0 2 4 6 8
0 2 4 6 8 10

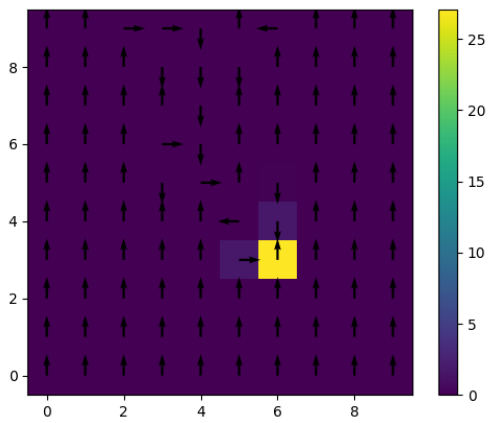
5


0 2 4 6 8
0 2 4 6 8
0 2 4 6 8 10

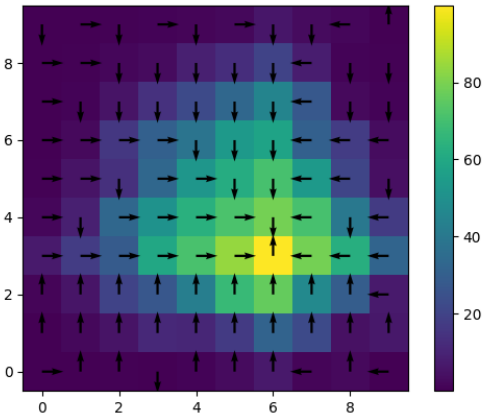
6



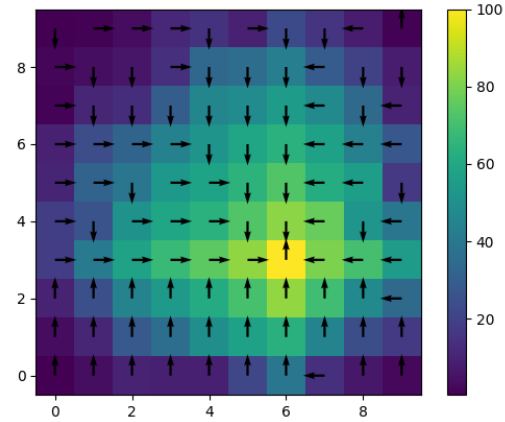
7



8



9



10

2A) What is the difference between Q-learning (as in this question) and value iteration (as in question 2 from last week's lab)?

2B) What is going on in the first few graphs here?

2C) What about all those pictures that are purple with one yellow spot?

2D) Why does the Q function suddenly "bloom" out in the last two pictures?

2E) What positive reward value do you think our implementation assigned to taking any action from the goal state?

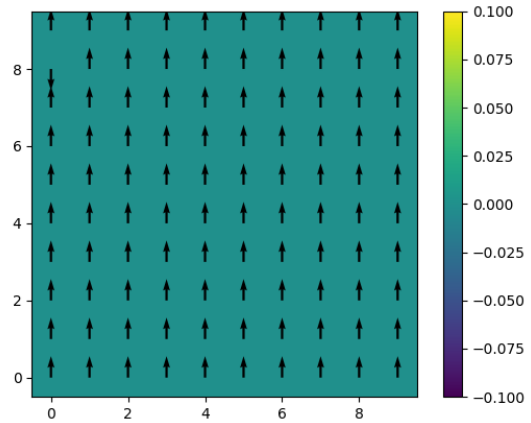
2F) Recall that in this implementation, in the case of a tie among $\arg\max_a Q(s,a)$ actions, the first action in the action space among the tied values is selected. What would change if instead a random action among the tied values were to be selected instead?

Check this box and submit when you have finished questions 2A through 2E.

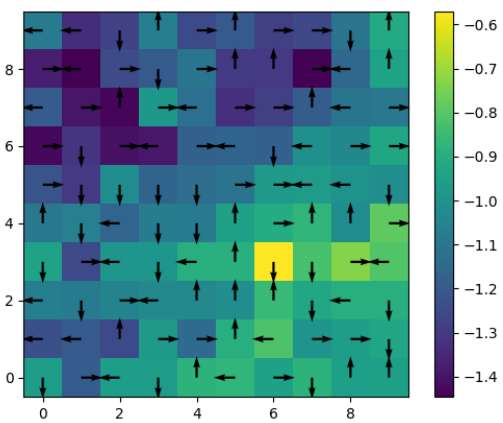
Submit | View Answer | Ask for Help **100.00%**

You have infinitely many submissions remaining.

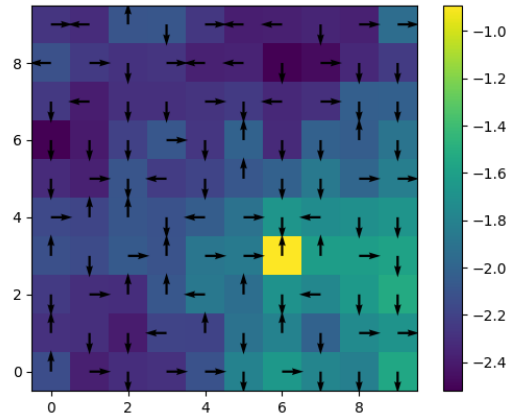
These are plots of the values of the states using the SSP formulation as we run 50,000 iterations of Q-value learning, plotting every 5,000 iterations. **Note that the scale of the colors changes across the different plots, per the bar on the right of each plot.**


0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100

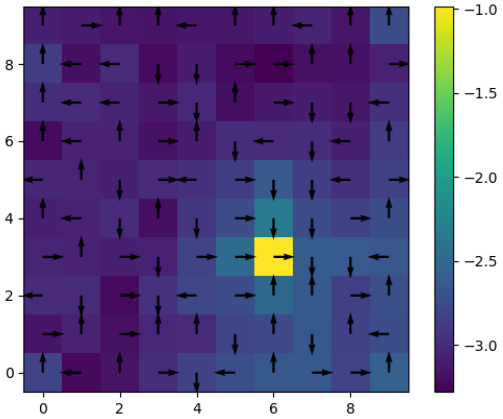
1


−0.6
−0.7
−0.8
−0.9
−1.0
−1.1
−1.2
−1.3
−1.4

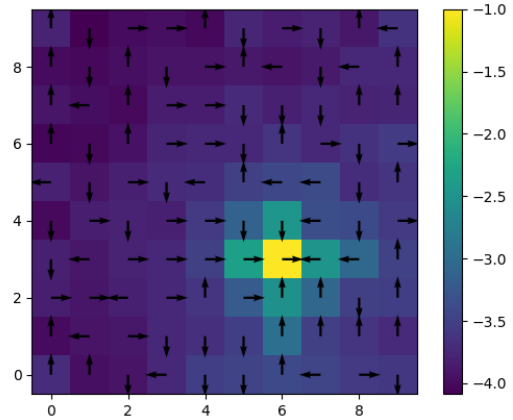
2


−1.0
−1.2
−1.4
−1.6
−1.8
−2.0
−2.2
−2.4

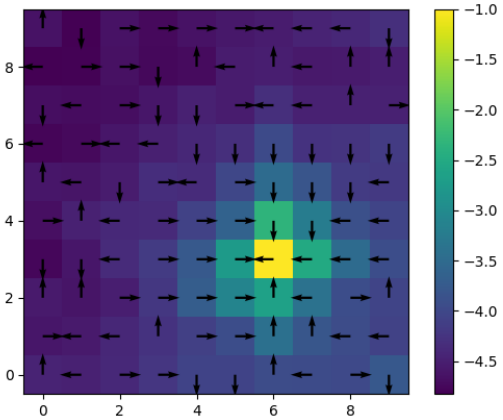
3


−1.0
−1.5
−2.0
−2.5
−3.0

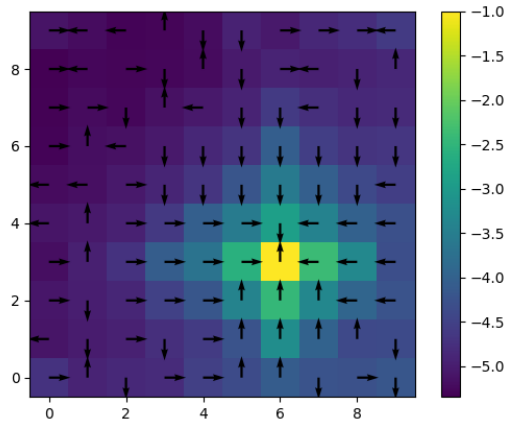
4


−1.0
−1.5
−2.0
−2.5
−3.0
−3.5
−4.0

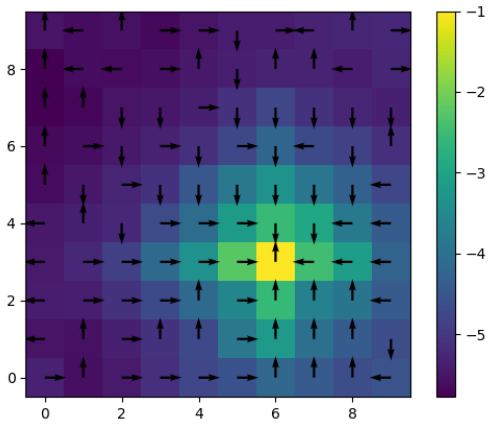
5


−1.0
−1.5
−2.0
−2.5
−3.0
−3.5
−4.0
−4.5

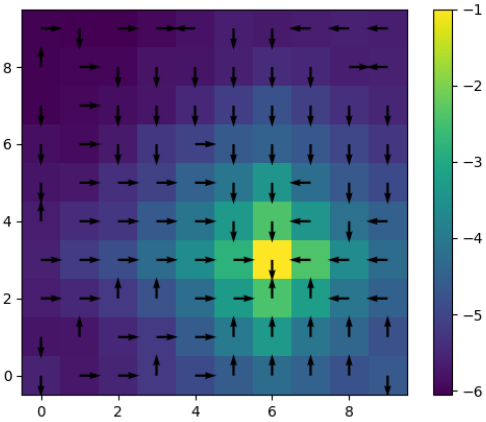
6



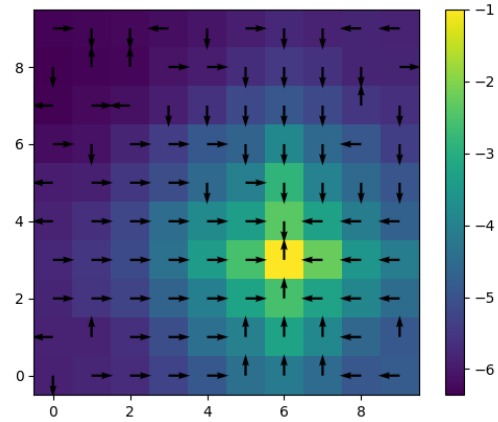
7



8



9



10

2G) What is going on in the first few graphs here?

2H) Discuss how and why these plots differ from those in the Goal-reward formulation -- especially in the first 6 graphs.

2I) Which formulation (Goal-reward or SSP) of the reward function works better here? Why?

Check this box and submit when you have finished question 2.

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

3) A Wild Epsilon Search

ϵ typically denotes an "explore vs. exploit" tradeoff in reinforcement learning, specifically Q-learning algorithm. As a reminder, when a Q-learning algorithm interacts with the environment, it takes a completely random move with probability ϵ and the best move according to its current Q-table with probability $1 - \epsilon$.

We will try to teach a computer how to play a simple "pong" game with a Q-learning algorithm. We will observe how the learning differs based on different values of ϵ.

3.1) Expectations

Before you move on, write down how you expect the Q-learning algorithm to behave for different values of ϵ in the set $\{0, 0.5, 1\}$. Think about how the learned Q-value function with a greedy action selection policy based on that Q-value will evolve for each ϵ.

3.1A) Which of these epsilon values risks **never** finding the optimal policy?

0

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

3.1B) Which of these epsilon values risks spending way too much time exploring parts of the space that are unlikely to be useful?

1

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

3.1C) Which of these epsilon values is guaranteed to cause optimal behavior **during** learning?

none

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

3.2) Versus Reality

For this part, you will use a colab notebook we have prepared for you. If you have not used colab before, ask your partner or a TA/LA for help. You can find the colab notebook here. To run it, you may need to create a copy of it in your google drive.

Once you run the code wait patiently until you see a yellow and purple square on a teal background (you may need to scroll down from the "score" and "reward" text lines printed out). Ignore everything else for now. Click play in the button right below the square. This is a movie of a policy playing the game No Exit. It's kind of like Pong: the purple square is the "ball" and the yellow square is your "paddle". The actions are to move the paddle up, down, or keep it still.

The state is specified by the positions and velocities of the ball and paddle, with a special added "game over" state.

The transition model is a very approximate physics model of the ball reflecting off walls and the paddle, except if the ball gets past the paddle in the positive x direction, the game is over.

The agent gets a reward of +1 on every step it manages to survive.

When watching the game play out, you'll sometimes see that the purple square gets near the right-hand border and then suddenly it changes to a state with the purple square in the bottom left and the yellow one in the upper right -- this means that the game terminated and then reset to the initial state.

Now we can go back and look at the other output in the notebook:

- First, we show a table of what happens during learning: after every 10 iterations of batch Q learning, we take the current greedy policy and run it to see what its average score is. This score represents how long the episode ran before the ball ran off the map, or 100 if it lasted for that long.
- Next is a plot of the score as a function of the amount of training.
- Finally, we run the greedy policy with respect last Q-value function for 10 games and report the rewards achieved on each game. We also make a movie of these 10 games, which is what we started out looking at.

3.2A) Run the code given on the notebook for values of ϵ in the set $0, 0.5, 1$. Does your observation of the learning behavior match with what you wrote down?

Remember that this is a small instance, so sometimes the random noise of the environment might prevent you from seeing any useful information. Run the notebook two or three times if something doesn't line up with your expectation, and then ask for help.

(Optional) Play with the number of iterations in the colab until you get all of the models to converge, and then observe the gameplay of the agents. You might expect the model that learned with $\epsilon = 1$ has a more jittery gameplay than the agent that learned with $\epsilon = 0$ or 0.5. Does that hold?