Solutions

10-601 Machine Learning	**Name:**
Spring 2021	**Andrew Email:**
Exam 2 Practice Problems	**Room:**
April 11, 2021	**Seat:**
Time Limit: N/A	**Exam Number:**

Instructions:

- Fill in your name and Andrew ID above. Be sure to write neatly, or you may not receive credit for your exam.
- Clearly mark your answers in the allocated space **on the front of each page.** If needed, use the back of a page for scratch space, but you will not get credit for anything written on the back of a page. If you have made a mistake, cross out the invalid parts of your solution, and circle the ones which should be graded.
- No electronic devices may be used during the exam.
- Please write all answers in pen.
- You have N/A to complete the exam. Good luck!

1 Logistic Regression

1. **[2 pts]** If today I want to predict the probability that a student sleep more than 8 hours on average (SA) given the Course loading (C), I will choose to use linear regression over logistic regression.

   Circle one: True False

   False.

2. Answer the following questions with brief explanations where necessary.

   a) **[2 pts]** A generalization of logistic regression to a multiclass settings involves expressing the per-class probabilities $P(y = c|x)$ as the softmax function $\frac{\exp(w_c^T x)}{\sum_{d \in C} \exp(w_d^T x)}$, where c is some class from the set of all classes C.

   Consider a 2-class problem (labels 0 or 1). Rewrite the above expression for this situation, to end up with expressions for $P(Y = 1|x)$ and $P(Y = 0|x)$ that we have already come across in class for binary logistic regression.

   $P(y = 1|x) = \frac{\exp(w_1^T x)}{\exp(w_0^T x) + \exp(w_1^T x)} = \frac{\exp((w_1 - w_0)^T x)}{1 + \exp((w_1 - w_0)^T x)} = \frac{\exp(w^T x)}{1 + \exp(w^T x)} = p$
   Therefore, $1 - p = \frac{1}{1 + \exp(w^T x)}$

   b) **[3 pts]** Given 3 data points $(1, 1), (1, 0), (0, 0)$ with labels $0, 1, 0$ respectively. Consider 2 models: Model 1: $\sigma(w_1 x_1 + w_2 x_2)$, Model 2: $\sigma(w_0 + w_1 x_1 + w_2 x_2)$ ($\sigma(z)$ is the sigmoid function $\frac{1}{1+e^{-z}}$) that compute $p(y = 1|\mathbf{x})$. Using the given data, we can learn parameters $\hat{w}$ by maximizing the conditional log-likelihood.

   Suppose we switched $(0, 0)$ to label 1 instead.

   Do the parameters learnt for Model 1 change?

   Circle one: **True** **False**
   One-line explanation:

   False. The parameters learnt for Model 1 don't change because $w_1 x_1 + w_2 x_2 = 0$ for (0, 0). Hence $p = 0.5$ irrespective of the labels or the values of w.

   What about Model 2?

   Circle one: **True** **False**
   One-line explanation:

   True. This model has a bias term which remains non-zero for (0, 0), and can thus change the model depending on the label assigned.

   c) **[2 pts]** For logistic regression, we need to resort to iterative methods such as gradient descent to compute the $\hat{w}$ that maximizes the conditional log likelihood. Why?

   There is no closed-form solution.

d) **[3 pts]** Considering a Gaussian prior, write out the MAP objective function $J(w)_{MAP}$ in terms of the MLE objective $J(w)_{MLE}$. Name the variant of logistic regression this results in.

$J_{MAP}(\mathbf{w}) = J_{MLE}(\mathbf{w}) - \lambda\|\mathbf{w}\|_2^2$. This is L2 regularized logistic regression.

3. Given a training set $\{(x_i, y_i), i = 1, \ldots, n\}$ where $x_i \in \mathbb{R}^d$ is a feature vector and $y_i \in \{0, 1\}$ is a binary label, we want to find the parameters $\hat{w}$ that maximize the likelihood for the training set, assuming a parametric model of the form

$$p(y = 1|x; w) = \frac{1}{1 + \exp(-w^T x)}.$$

The conditional log likelihood of the training set is

$$\ell(w) = \sum_{i=1}^{n} y_i \log p(y_i, |x_i; w) + (1 - y_i) \log(1 - p(y_i, |x_i; w)),$$

and the gradient is

$$\nabla \ell(w) = \sum_{i=1}^{n} (y_i - p(y_i|x_i; w))x_i.$$

a) [5 pts.] Is it possible to get a closed form for the parameters $\hat{w}$ that maximize the conditional log likelihood? How would you compute $\hat{w}$ in practice?

There is no closed form expression for maximizing the conditional log likelihood. One has to consider iterative optimization methods, such as gradient descent, to compute $\hat{w}$.

b) [5 pts.] For a binary logistic regression model, we predict $y = 1$, when $p(y = 1|x) \geq 0.5$. Show that this is a linear classifier.

Using the parametric form for $p(y = 1|x)$:

$$\begin{aligned} p(y = 1|x) \geq \frac{1}{2} &\implies \frac{1}{1 + \exp(-w^T x)} \geq \frac{1}{2} \\ &\implies 1 + \exp(-w^T x) \leq 2 \\ &\implies \exp(-w^T x) \leq 1 \\ &\implies -w^T x \leq 0 \\ &\implies w^T x \geq 0, \end{aligned}$$

so we predict $\hat{y} = 1$ if $w^T x \geq 0$.

c) Consider the case with binary features, i.e, $x \in \{0, 1\}^d \subset \mathbb{R}^d$, where feature x_1 is rare and happens to appear in the training set with only label 1. What is $\hat{w}_1$? Is the gradient ever zero for any finite w? Why is it important to include a regularization term to control the norm of $\hat{w}$?

If a binary feature fired for only label 1 in the training set then, by maximizing the conditional log likelihood, we will make the weight associated to that feature be infinite. This is because, when this feature is observed in the training set, we will want to predict predict 1 irrespective of everything else. This is an undesired behaviour from the point of view of generalization performance, as most likely we do not believe this rare feature to have that much information about class 1. Most likely, it is spurious co-occurrence. Controlling the norm of the weight vector will prevent these pathological cases.

2 Feature Engineering and Regularization

1. **Model Complexity:** In this question we will consider the effect of increasing the model complexity, while keeping the size of the training set fixed. To be concrete, consider a classification task on the real line $\mathbb{R}$ with distribution D and target function $c^* : \mathbb{R} \to \{\pm 1\}$ and suppose we have a random sample S of size n drawn iid from D. For each degree d, let ϕ_d be the feature map given by $\phi_d(x) = (1, x, x^2, \ldots, x^d)$ that maps points on the real line to $(d+1)$-dimensional space.

   Now consider the learning algorithm that first applies the feature map ϕ_d to all the training examples and then runs logistic regression as in the previous question. A new example is classified by first applying the feature map ϕ_d and then using the learned classifier.

   a) [4 pts.] For a given dataset S, is it possible for the training error to increase when we increase the degree d of the feature map? **Please explain your answer in 1 to 2 sentences.** No. Every linear separator using the feature map ϕ_d can also be expressed using the feature map ϕ_{d+1}, since we are only adding new features. It follows that the training error must decrease for any given sample S.

   b) [4 pts.] Briefly **explain in 1 to 2 sentences** why the true error first drops and then increases as we increase the degree d. When the dimension d is small, the true error is high because it is not possible to the target function is not well approximated by any linear separator in the ϕ_d feature space. As we increase d, our ability to approximate c^* improves, so the true error drops. But, as we continue to increase d, we begin to overfit the data and the true error increases again.

3 Neural Networks

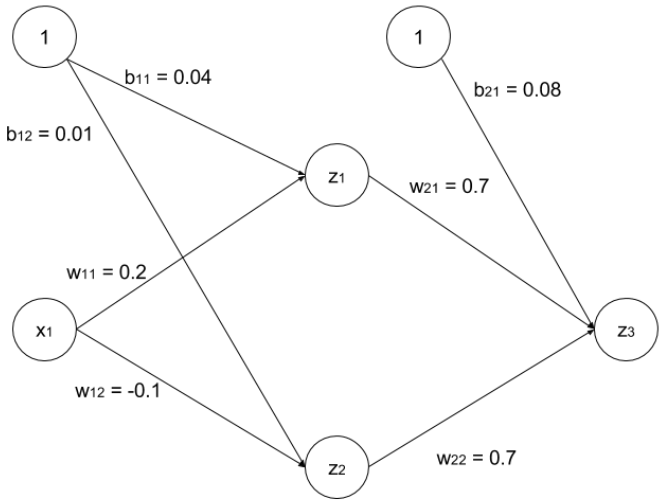


Figure 1: neural network

1. Consider the neural network architecture shown above for a 2-class $(0, 1)$ classification problem. The values for weights and biases are shown in the figure. We define:

$a_1 = w_{11}x_1 + b_{11}$

$a_2 = w_{12}x_1 + b_{12}$

$a_3 = w_{21}z_1 + w_{22}z_2 + b_{21}$

$z_1 = \text{relu}(a_1)$

$z_2 = \text{relu}(a_2)$

$z_3 = \sigma(a_3)$, $\sigma(x) = \frac{1}{1+e^{-x}}$

Use this information to answer the questions that follow.

(i) **[6 pts]** For $x_1 = 0.3$, compute z_3, in terms of e. **Show all work.**
$z_3 =$

$z_3 = \frac{1}{1+e^{-0.15}}$

(ii) **[2 pts]** To which class does the network predict the given data point ($x_1 = 0.3$), i.e., $\hat{y} =$? Note that $\hat{y} = 1$ if $z_3 > \frac{1}{2}$, else $\hat{y} = 0$.

Circle one: 0 1

$\hat{y}(x_1 = 0.3) = 1$

(iii) **[6 pts]** Perform backpropagation on the bias b_{21} by deriving the expression for the gradient of the loss function $L(y, z_3)$ with respect to the bias term b_{21}, $\frac{\partial L}{\partial b_{21}}$, in terms of the partial derivatives $\frac{\partial \alpha}{\partial \beta}$, where α and β can be any of $L, z_i, a_i, b_{ij}, w_{ij}, x_1$ for all valid values of i, j. Your backpropagation algorithm should be as explicit as possible—that is, make sure each partial derivative $\frac{\partial \alpha}{\partial \beta}$ cannot be decomposed further into simpler partial derivatives. Do *not* evaluate the partial derivatives.

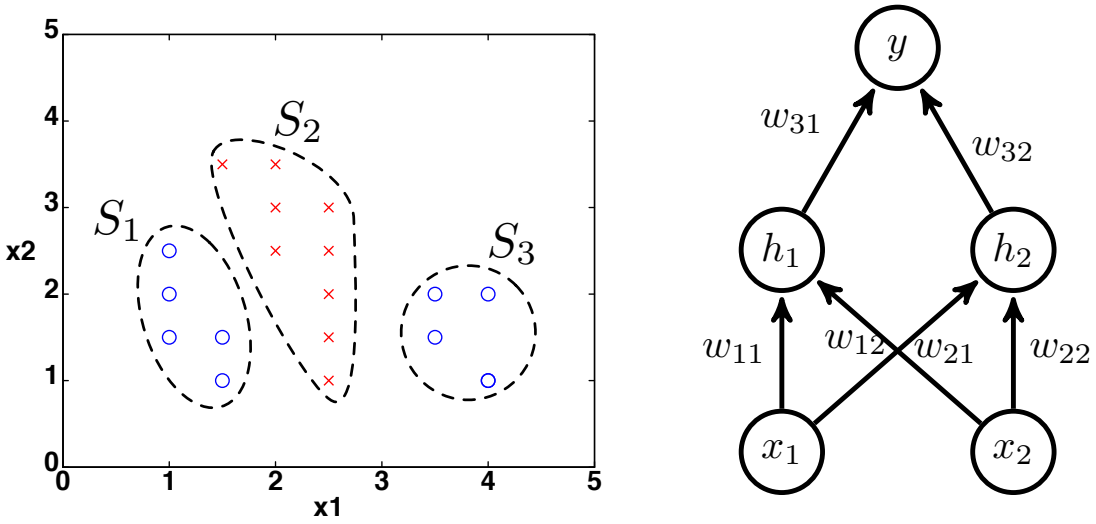
$$\frac{\partial L}{\partial b_{21}} = \frac{\partial L}{\partial z_3}\frac{\partial z_3}{\partial a_3}\frac{\partial a_3}{\partial b_{21}}$$

(iv) **[6 pts]** Perform backpropagation on the bias b_{12} by deriving the expression for the gradient of the loss function $L(y, z_3)$ with respect to the bias term b_{12}, $\frac{\partial L}{\partial b_{12}}$, in terms of the partial derivatives $\frac{\partial \alpha}{\partial \beta}$, where α and β can be any of $L, z_i, a_i, b_{ij}, w_{ij}, x_1$ for all valid values of i, j. Your backpropagation algorithm should be as explicit as possible—that is, make sure each partial derivative $\frac{\partial \alpha}{\partial \beta}$ cannot be decomposed further into simpler partial derivatives. Do *not* evaluate the partial derivatives.

$$\frac{\partial L}{\partial b_{12}} = \frac{\partial L}{\partial z_3}\frac{\partial z_3}{\partial a_3}\frac{\partial a_3}{\partial z_2}\frac{\partial z_2}{\partial a_2}\frac{\partial a_2}{\partial b_{12}}$$

2. In this problem we will use a neural network to classify the crosses ($\times$) from the circles ($\circ$) in the simple dataset shown in Figure 2a. Even though the crosses and circles are not linearly separable, we can break the examples into three groups, S_1, S_2, and S_3 (shown in Figure 2a) so that S_1 is linearly separable from S_2 and S_2 is linearly separable from S_3. We will exploit this fact to design weights for the neural network shown in Figure 2b in order to correctly classify this training set. For all nodes, we will use the threshold activation function

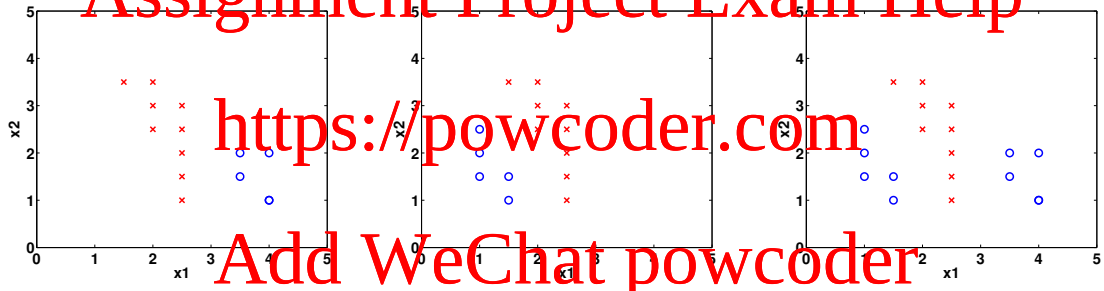
$$\phi(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0. \end{cases}$$



(a) The dataset with groups S_1, S_2, and S_3.

(b) The neural network architecture

Figure 2



(a) Set S2 and S3

(b) Set S1 and S2

(c) Set S1, S2 and S3

Figure 3: NN classification.

(i) First we will set the parameters w_{11}, w_{12} and b_1 of the neuron labeled h_1 so that its output $h_1(x) = \phi(w_{11}x_1 + w_{12}x_2 + b_1)$ forms a linear separator between the sets S_2 and S_3.

(a) [1 pt.] On Fig 3a, draw a linear decision boundary that separates S_2 and S_3.

(b) [1 pt.] Write down the corresponding weights w_{11}, w_{12}, and b_1 so that $h_1(x) = 0$ for all points in S_3 and $h_1(x) = 1$ for all points in S_2. One solution would suffice and the same applies to (ii) and (iii).

$w_{11} = -1, w_{12} = 0, b_1 = 3$

(ii) Next we set the parameters w_{21}, w_{22} and b_2 of the neuron labeled h_2 so that its output $h_2(x) = \phi(w_{21}x_1 + w_{22}x_2 + b_2)$ forms a linear separator between the sets S_1 and S_2.

(a) [1 pt.] On Fig 3b, draw a linear decision boundary that separates S_1 and S_2.

(b) [1 pt.] Write down the corresponding weights w_{21}, w_{22}, and b_2 so that $h_2(x) = 0$ for all points in S_1 and $h_2(x) = 1$ for all points in S_2.

$w_{21} = 3, w_{22} = 1, b_2 = -7$

(iii) Now we have two classifiers h_1 (to classify S_2 from S_3) and h_2 (to classify S_1 from S_2). We will set the weights of the final neuron of the neural network based on the results from h_1 and h_2 to classify the crosses from the circles. Let $h_3(x) = \phi\big(w_{31}h_1(x) + w_{32}h_2(x) + b_3\big)$.

(a) [1 pt.] Compute w_{31}, w_{32}, b_3 such that $h_3(x)$ correctly classifies the entire dataset. $w_{31} = 1, w_{32} = 1, b_3 = -1.5$

(b) [1 pt.] Draw your decision boundary in Fig 3c.

(iv) **Back propagation**

In the above example, we need to learn the weights by according to the data. At first step, we need to get the gradients of the parameters of neural networks.

Suppose there m data points x_i with label y_i, where $i \in [1, m]$. x_i is a $d \times 1$ vector and $y_i \in \{0, 1\}$. We use the data to train a neural network with one hidden layer:

$$h(x) = \sigma(W_1 x + b_1)$$
$$p(x) = \sigma(W_2 h(x) + b_2),$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the sigmoid function, W_1 is a n by d matrix and b_1 is a n by 1 vector, W_2 is a 1 by n matrix and b_1 is a 1 by 1 vector.

We use cross entropy loss function and minimize the negative log likelihood to train the neural network.

$$l = \frac{1}{m}\sum_i l_i = \frac{1}{m}\sum_i -(y_i \log p_i + (1 - y_i)\log(1 - p_i)),$$

where $p_i = p(x_i), h_i = h(x_i)$.

(a) Describe how you would drive the gradients w.r.t the parameters W_1, W_2 and b_1, b_2. (No need to write out the detailed mathematical expression.) Use chain rule.

(b) When m is large, we typically use a small sample of all the data set to estimate the gradient, this is call stochastic gradient descent (SGD). Explain why we use SGD instead of gradient descent.
SGD converges faster than gradient descent.

(c) Work out the following gradient: $\frac{\partial l}{\partial p_i}, \frac{\partial l}{\partial W_2}, \frac{\partial l}{\partial b_2}, \frac{\partial l}{\partial h_i}, \frac{\partial l}{\partial W_1}, \frac{\partial l}{\partial b_1}$. When deriving the gradient w.r.t. the parameters in lower layers, you can may assume the gradient in upper layers are available to you (i.e., you can use them in your equation). For example, when calculating $\frac{\partial l}{\partial W_1}$, you can assume $\frac{\partial l}{\partial p_i}, \frac{\partial l}{\partial W_2}, \frac{\partial l}{\partial b_2}, \frac{\partial l}{\partial h_i}$ are known.

$$\frac{\partial l}{\partial p_i} = \frac{1}{m}(-\frac{y_i}{p_i} + \frac{1-y_i}{1-p_i})$$
$$\frac{\partial l}{\partial W_2} = \frac{1}{m}\sum_i \frac{\partial l_i}{\partial p_i}\frac{\partial p_i}{\partial W_2} = \frac{1}{m}\sum_i \frac{\partial l_i}{\partial p_i} p_i(1-p_i)h_i^T$$
$$\frac{\partial l}{\partial b_2} = \frac{1}{m}\sum_i \frac{\partial l_i}{\partial p_i} p_i(1-p_i)$$
$$\frac{\partial l}{\partial h_i} = \frac{\partial l_i}{\partial p_i}\frac{\partial p_i}{\partial h_i} = \frac{\partial l_i}{\partial p_i} p_i(1-p_i)W_2^T$$
$$\frac{\partial l}{\partial W_1} = \frac{1}{m}\sum_i \frac{\partial l_i}{\partial h_i}\frac{\partial h_i}{\partial W_1} = \frac{1}{m}\sum_i \left[\frac{\partial l_i}{\partial h_i} \circ h_i \circ (1-h_i)\right] x_i^T$$
$$\frac{\partial l}{\partial b_1} = \frac{1}{m}\sum_i \frac{\partial l_i}{\partial h_i}\frac{\partial h_i}{\partial b_1} = \frac{1}{m}\sum_i \frac{\partial l_i}{\partial h_i} \circ h_i \circ (1-h_i)$$

4 Reinforcement Learning

4.1 Markov Decision Process

Environment Setup (may contain spoilers for Shrek 1)

Lord Farquaad is hoping to evict all fairytale creatures from his kingdom of Duloc, and has one final ogre to evict: Shrek. Unfortunately all his previous attempts to catch the crafty ogre have fallen short, and he turns to you, with your knowledge of Markov Decision Processes (MDP's) to help him catch Shrek once and for all.

Consider the following MDP environment where the agent is Lord Farquaad:

Figure 4: Kingdom of Duloc, circa 2001

Here's how we will define this MDP:

- S **(state space):** a set of states the agent can be in. In this case, the agent (Farquaad) can be in any location (row, col) and also in any orientation $\in \{N, E, S, W\}$. Therefore, state is represented by a three-tuple (row, col, dir), and $S =$ all possible of such tuples. Farquaad's start state is $(1, 1, E)$.
- A **(action space):** a set of actions that the agent can take. Here, we will have just three actions: turn right, turn left, and move forward (turning does not change row or col, just dir). So our action space is $\{R, L, M\}$. Note that Farquaad is debilitatingly short, so he cannot travel through (or over) the walls. Moving forward when facing a wall results in no change in state (but counts as an action).
- $R(s, a)$ **(reward function):** In this scenario, Farquaad gets a reward of 5 by moving into the swamp (the cell containing Shrek), and a reward of 0 otherwise.
- $p(s'|s, a)$ **(transition probabilities):** We'll use a deterministic environment, so this will bee 1 if s' is reachable from s and by taking a, and 0 if not.

1. What are $|S|$ and $|A|$ (size of state space and size of action space)?

   $|S| = 4$ rows $\times$ 4 columns $\times$ 4 orientations $= 64$
   $|A| = |\{R, L, M\}| = 3$

2. Why is it called a "Markov" decision process? (Hint: what is the assumption made with p?)

   $p(s'|s, a)$ assumes that s' is determined only by s and a (and not any other previous states or actions).

3. What are the following transition probabilities?

   $$p((1,1,N)|(1,1,N),M) =$$
   $$p((1,1,N)|(1,1,E),L) =$$
   $$p((2,1,S)|(1,1,S),M) =$$
   $$p((2,1,E)|(1,1,S),M) =$$

   $p((1,1,N)|(1,1,N),M) = 1$
   $p((1,1,N)|(1,1,E),L) = 1$
   $p((2,1,S)|(1,1,S),M) = 1$
   $p((2,1,E)|(1,1,S),M) = 0$

4. Given a start position of $(1, 1, E)$ and a discount factor of $\gamma = 0.5$, what is the expected discounted future reward from $a = R$? For $a = L$? (Fix $\gamma = 0.5$ for following problems).

   For $a = R$ we get $R_R = 5 * (\frac{1}{2})^{16}$ (it takes 17 moves for Farquaad to get to Shrek, starting with $R, M, M, M, L...$)

   For $a = L$, this is a bad move, and we need another move to get back to our original orientation, from which we can go with our optimal policy. So the reward here is:

   $R_L = (\frac{1}{2})^2 * R_R = 5 * (\frac{1}{2})^{18}$

5. What is the optimal action from each state, given that orientation is fixed at E? (if there are multiple options, choose any)

R	R	M	R
R	R	L	R
M	R	L	R
M	M	L	-

(some have multiple options, I just chose one of the possible ones)

6. Farquaad's chief strategist (Vector from Despicable Me) suggests that having $\gamma = 0.9$ will result in a different set of optimal policies. Is he right? Why or why not?

Vector is wrong. While the reward quantity will be different, the set of optimal policies does not change. (it is now $5 * (\frac{9}{10})^{16}$) (one can only assume that Lord Farquaad and Vector would be in kahoots: both are extremely nefarious!)

7. Vector then suggests the following setup: $R(s, a) = 0$ when moving into the swamp, and $R(s, a) = -1$ otherwise. Will this result in a different set of optimal policies? Why or why not?

It will not. While the reward quantity will be different, the set of optimal policies does not change. (Farquaad will still try to minimize the number of steps he takes in order to reach Shrek)

8. Vector now suggests the following setup: $R(s, a) = 5$ when moving into the swamp, and $R(s, a) = 0$ otherwise, but with $\gamma = 1$. Could this result in a different optimal policy? Why or why not?

This will change the policy, but not in Lord Farquaad's favor. He will no longer be incentivized to reach Shrek quickly (since $\gamma = 1$). The optimal reward from each state is the same (5) and therefore each action from each state is also optimal. Vector really should have taken 10-301/601...

9. Surprise! Elsa from Frozen suddenly shows up. Vector hypnotizes her and forces her to use her powers to turn the ground into ice. Now the environment is now stochastic: since the ground is now slippery, when choosing the action M, with a 0.2 chance, Farquaad will slip and move two squares instead of one. What is the expected future-discounted rewards from $s = (2, 4, S)$?

Recall that $R_{exp} = max_a E[R(s, a) + \gamma R_{s'}]$

(notation might be different than in the notes, but conceptually, our reward is the best expected reward we can get from taking any action a from our current state s.)

In this case, our best action is obviously to move forward. So we get

R_{exp} = (expected value of going two steps) + (expected value of going one step)

$E[2_{steps}] = p((4,4,S)|(2,4,S),M) \times R((4,4,S),(2,4,S),M) = 0.2 \times 5 = 1$

$E[1_{step}] = p((4,3,S)|(2,4,S),M) \times (R((4,3,S),(2,4,S),M) + \gamma R_{(4,3,S)})$

where $R_{(4,3,S)}$ is the expected reward from $(4,3,S)$. Since the best reward from here is obtained by choosing $a = M$, and we always end up at Shrek, we get

$E[1_{step}] = 0.8 \times (0 + \gamma \times 5) = 0.8 \times 0.5 \times 5 = 2$

giving us a total expected reward of $R_{exp} = 1 + 2 = 3$

(I will be very disappointed if this is not the plot of Shrek 5)

4.2 Value and Policy Iteration

1. Which of the following environment characteristics would increase the computational complexity per iteration for a value iteration algorithm? Choose all that apply:

   ☐ Large Action Space

   ☐ A Stochastic Transition Function

   ☐ Large State Space

   ☐ Unknown Reward Function

   ☐ None of the Above

   A and C (state space and action space). The computational complexity for value iteration per iteration is $O(|A||S|^2)$

2. Which of the following environment characteristics would increase the computational complexity per iteration for a policy iteration algorithm? Choose all that apply:

   ☐ Large Action Space

   ☐ A Stochastic Transition Function

   ☐ Large State Space

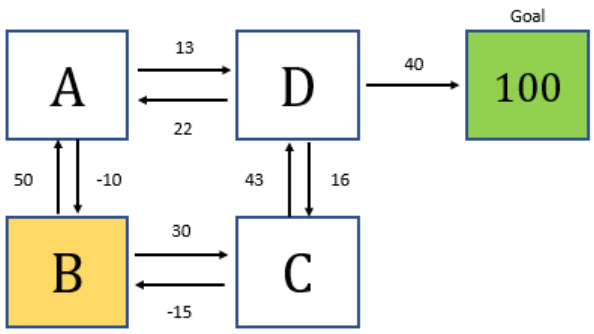
   ☐ Unknown Reward Function

   ☐ None of the Above

   A and C again. The computational complexity for policy iteration per iteration is $O(|A||S|^2 + |S|^3)$

3. In the image below is a representation of the game that you are about to play. There are 5 states: A, B, C, D, and the goal state. The goal state, when reached, gives 100 points as reward. In addition to the goal's points, you also get points by moving to different states. The amount of points you get are shown next to the arrows. You start at state

B. To figure out the best policy, you use asynchronous value iteration with a decay (γ) of 0.9.



(i) When you first start playing the game, what action would you take (up, down, left, right) at state B?

Up

(ii) What is the total reward at state B at this time?

50 (immediate reward of 50, and future reward (value at state A) starts at 0)

(iii) Let's say you keep playing until your total values for each state has converged. What action would you take at state B?

C

(iv) What is the total reward at state B at this time?

174 (30 from the immediate action, and 144 from the future reward (value at state C))

4.3 Q-Learning

1. For the following true/false, circle one answer and provide a one-sentence explanation:

(i) One advantage that Q-learning has over Value and Policy iteration is that it can account for non-deterministic policies.

Circle one: True False

False. All three methods can account for non-deterministic policies

(ii) You can apply Value or Policy iteration to any problem that Q-learning can be applied to.

Circle one: True False

False. Unlike the others, Q-learning doesn't need to know the transition probabilities (p(s' | s, a)), or the reward function (r(s,a)) to train. This is its biggest advantage.

(iii) Q-learning is guaranteed to converge to the true value Q* for a greedy policy.

Circle one: True False

False. Q-learning converges only if every state will be explored infinitely. Thus, purely exploiting policies (e.g. greedy policies) will not necessarily converge to Q*, but rather to a local optimum.

2. For the following parts of this problem, recall that the update rule for Q-learning is:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \left(q(\mathbf{s}, a; \mathbf{w}) - (r + \gamma \max_{a'} q(\mathbf{s}', a'; \mathbf{w}) \right) \nabla_{\mathbf{w}} q(\mathbf{s}, a; \mathbf{w})$$

(i) From the update rule, let's look at the specific term $X = (r + \gamma \max_{a'} q(\mathbf{s}', a'; \mathbf{w}))$ Describe in English what is the role of X in the weight update.

Estimate of true total return (Q*(s,a)). This may get multiple answers, so grade accordingly

(ii) Is this update rule synchronous or asynchronous?

Asynchronous

(iii) A common adaptation to Q-learning is to incorporate rewards from more time steps into the term X. Thus, our normal term $r_t + \gamma * max_{a_{t+1}} q(s_{t+1}, a_{t+1}; w)$ would become $r_t + \gamma * r_{t+1} + \gamma^2 \max_{a_{t+2}} q(\mathbf{s}_{t+2}, a_{t+2} : \mathbf{w})$ What are the advantages of using more rewards in this estimation?

Incorporating rewards from multiple time steps allows for a more "realistic" estimate of the true total reward, since a larger percentage of it is from real experience. It can help with stabilizing the training procedure, while still allowing training at each time step (bootstrapping). This type of method is called N-Step Temporal Difference Learning.