6.036: Final Exam: Fall 2017

Solutions

- This is a closed book exam. Calculators not permitted. **Useful formulas on page 1**.
- The problems are not necessarily in any order of difficulty.
- Record all your answers in the places provided. If you run out of room for an answer, continue on the back of the page. Show your work neatly.
- If a question seems vague or under-specified to you, make an assumption, write it down, and solve the problem given your assumption. **Don't ask for clarification.**
- **Write your name on every page.**

Name: _____ Athena ID: _____

Question	Points	Score
1	8	
2	10	
3	8	
4	10	
5	10	
6	10	
7	10	
8	10	
9	12	
10	12	
Total:	100	

Name: _____

Potentially useful facts

In the definitions below, NLL takes two arguments, so it's $\text{NLL}(y, \hat{y})$ where y is the true value and $\hat{y}$ is the predicted value. We have sometimes seen it with the arguments in the other order.

$$\text{sign}(z) = \begin{cases} +1 & \text{if } z > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\tanh(z) = \frac{e^{2z} - 1}{e^{2z} + 1}$$

$$\text{NLL}(y, \hat{y}) = y \log \hat{y} + (1 - y) \log(1 - \hat{y})$$

$$\frac{\partial \text{NLL}(y, \hat{y})}{\partial \hat{y}} = \left(-\frac{y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}} \right)$$

$$\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1 - \sigma(z))$$

$$\frac{\partial \text{NLL}(y, \sigma(z))}{\partial z} = (\sigma(z) - y)$$

The Bellman Q-value equation (both versions correct):

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} P(s' \mid s, a) \max_{a'} Q(s', a')$$

$$Q(s, a) = \sum_{s'} P(s' \mid s, a)(R(s, a, s') + \gamma \max_{a'} Q(s', a'))$$

Definition of margin

$$\gamma(x, y, \theta, \theta_0) = \frac{y(\theta^T x + \theta_0)}{\|\theta\|}$$

Hinge loss with respect to a reference margin γ_{ref}:

$$L_H\left(\frac{\gamma(x, y, \theta, \theta_0)}{\gamma_{ref}}\right) = \begin{cases} 1 - \gamma(x, y, \theta, \theta_0)/\gamma_{ref} & \text{if } \gamma(x, y, \theta, \theta_0) < \gamma_{ref} \\ 0 & \text{otherwise} \end{cases}$$

Name: _____

1 Regression

1. (8 points) We're given a data set $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$, where $x^{(i)} \in R^d$ and $y^{(i)} \in R$. Let X be a $d \times n$ matrix in which the $x^{(i)}$ are the columns and let Y be a $1 \times n$ vector containing the values of $y^{(i)}$. Using the ordinary least-squares formula, we can compute

$$W_{ols} = (XX^T)^{-1}XY^T$$

Using ridge regression, we can compute

$$W_{ridge} = (XX^T + \lambda I)^{-1}XY^T$$

We decide to try to use these methods to initialize a single-unit "neural network" with a linear activation function. Assume that XX^T is neither singular nor equal to the identity matrix, and that neither W_{ols} nor W_{ridge} is equal to $(0, 0, \ldots, 0)$.

(a) If we initialized our neuron with W_{ols} and did batch gradient descent (summing the error over all the data points) with squared loss and a fixed small step size, which of the following would most typically happen:

- ◯ The weights would change substantially at the beginning, but then converge back to the original values.
- √ **The weights would not change.**
- ◯ The weights would make small oscillations around the initial weights.
- ◯ The weights would converge to a different value.
- ◯ Something else would happen.

(b) Explain why.

> **Solution:** These weights are an optimum of the objective and the gradient will be (nearly) zero.

(c) If we initialized our neuron with W_{ols} and did stochastic gradient descent (one data point at a time) with squared loss and a fixed small step size, which of the following would most typically happen?

- ◯ The weights would change substantially at the beginning, but then converge back to the original values.
- ◯ The weights would not change.
- √ **The weights would make small oscillations around the initial weights.**
- ◯ The weights would converge to a different value.
- ◯ Something else would happen.

(d) Explain why.

> **Solution:** In expectation the steps should be small motions around the optimum. If someone says it will (or might) do something else (like hop out of this and get stuck somewhere else) that's most of the credit if it is well reasoned and explained.

(e) Consider a neuron initialized with W_{ridge}. Provide an objective function $J(W)$ that depends on the data, such that batch gradient descent to minimize J will have no effect on the weights, or argue that one does not exist.

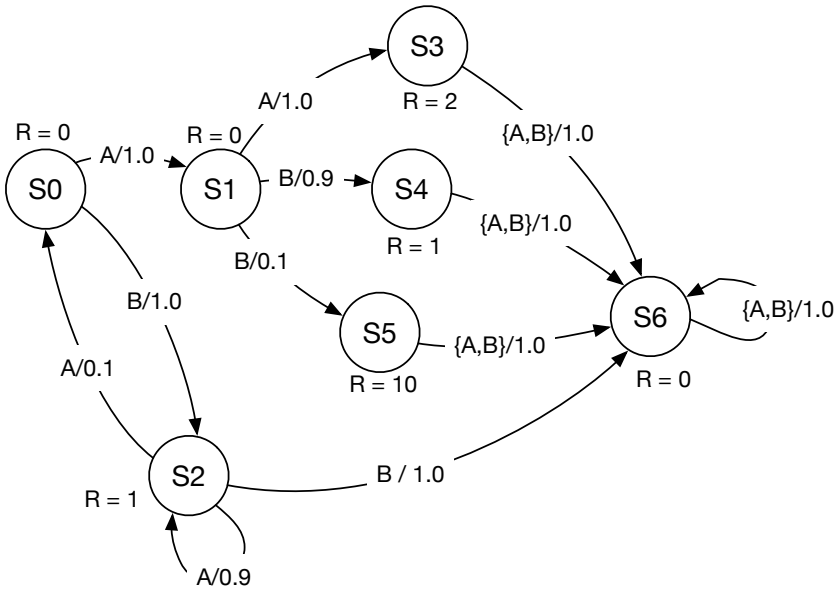
Solution: $J(W) = (W^T X - Y)^2 + \lambda \|W\|^2$

(f) Reggie has solved many problems like this before and the solution has typically been close to $W_0 = (1, ..., 1)^T$. Define an objective function that would result in good estimates for Reggie's next problem, even with very little data.

Solution: $J(W) = (W^T X - Y)^2 + \lambda \|W - \mathbf{1}\|^2$

Name: _____

2 MMMMMM DP



2. (10 points)

Consider the MDP shown above. It has states $S_0, \ldots, S_6$ and actions A, B. Each arrow is labeled with one or more actions, and a probability value: this means that if any of those actions is chosen from the state at the start of the arrow, then it will make a transition to the state at the end of the arrow with the associated probability.

Rewards are associated with states, and independent, in this example, from the action that is taken in that state. Remember that with horizon $H = 1$, the agent can collect the reward associated with the state it is in, and then terminates.

(a) Consider the policy π that takes action B in S_0 and action A in S_2. If the system starts in S_0 or S_2, then under that policy, only those two states (S_0 and S_2) are reachable.

Assuming the discount factor $\gamma = 0.5$, what are the values of $V_\pi(S_0)$ and $V_\pi(S_2)$? It is sufficient to write out a small system of linear equations that determine the values of those two variables; you do not have to take the time to solve them numerically.

Solution:

$$\begin{aligned} V_\pi(S_0) &= 0 + 0.5 \cdot V_\pi(S_2) \\ V_\pi(S_2) &= 1 + 0.5 \cdot (0.9 V_\pi(S_2) + 0.1 V_\pi(S_0)) \end{aligned}$$

Name: _____

(b) What is the optimal value $V(s) = max_a Q(s, a)$ for each state for horizon $H = 1$ with no discounting?

i. S_0 **0**

ii. S_1 **0**

iii. S_2 **1**

iv. S_3 **2**

v. S_4 **1**

vi. S_5 **10**

vii. S_6 **0**

(c) What is the optimal action and value $V(s)$ for each state for horizon $H = 2$ with no discounting?

i. S_0 a: **B** v: **1**

ii. S_1 a: **A** v: **2**

iii. S_2 a: **A** v: **1.9**

iv. S_3 a: **A or B** v: **2**

v. S_4 a: **A or B** v: **1**

vi. S_5 a: **A or B** v: **10**

vii. S_6 a: **A or B** v: **0**

Name: _____

(d) Are there any policies that result in infinite-horizon Q_π values that are finite for all states even when $\gamma = 1$? If so, provide such a policy. If not, explain why not.

i. S_0 **A**

ii. S_1 **A**

iii. S_2 **A or B**

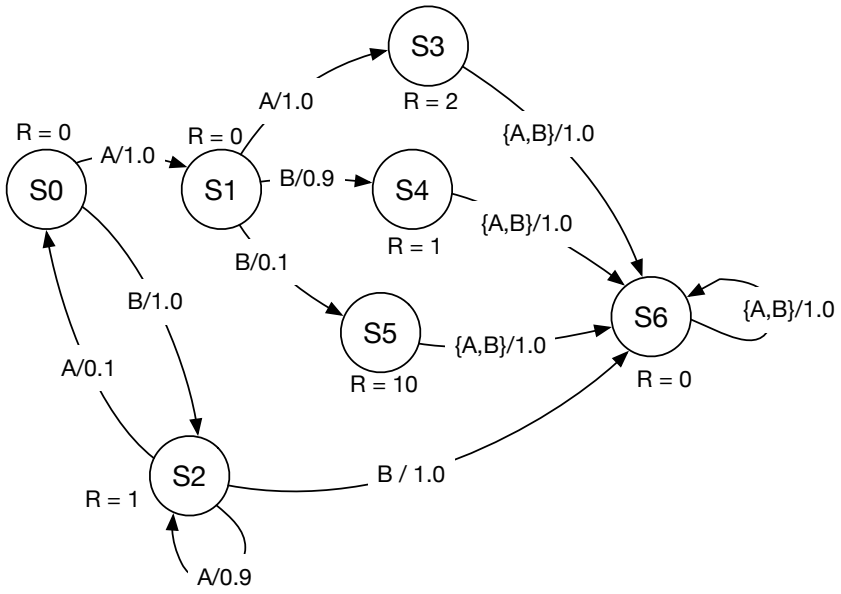
iv. S_3 **A or B**

v. S_4 **A or B**

vi. S_5 **A or B**

vii. S_6 **A or B**

Solution: There's a policy.



Name: _____

3 ABCs of RL

3. (8 points) Consider an MDP with four states, called A, B, C, and D, and with two actions called **Move** and **Stay**. The discount factor $\gamma = 0.9$.

   Here is a reminder of the Q-learning update formula, based on experience tuple (s, a, r, s'):

   $$Q(s,a) := (1-\alpha)Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a')\right) \quad .$$

   Let $\alpha = 1$.

   Assume we see the following state-action-reward sequence:
   A, Move, 0
   B, Move, 0
   C, Move, 1
   A, Move, 0
   B, Move, 0
   C

   With Q-values all starting at 0, we run the Q-learning algorithm on that state-action sequence.

   (a) Specify *each of the five updates to the Q table* by saying which entry or entries are updated, and with which values. Please include an update even if the value is 0.

   i. $Q($**A**, **Move**$) =$ **0**

   ii. $Q($**B**, **Move**$) =$ **0**

   iii. $Q($**C**, **Move**$) =$ **1**

   iv. $Q($**A**, **Move**$) =$ **0**

   v. $Q($**B**, **Move**$) =$ **0.9**

   (b) Characterize the weakness of Q-learning demonstrated by this example, which would be worse if there were a long sequence of states $B_1, \ldots, B_{100}$ between A and C. Very briefly describe a strategy for overcoming this weakness.

   Solution: It doesn't propagate the value all the way back the chain. Do the updates backward along the trajectory; or save your experience and replay it.

(c) Imagine that we continue acting (using the Q function we arrived at, at the end of the previous question) and receive the following additional state-action-reward sequence:
A, Move, 0
B, Move, 0
D

What additional updates would occur?

i. $Q($**A**, **Move**$) =$ **.81**

ii. $Q($**B**, **Move**$) =$ **0**

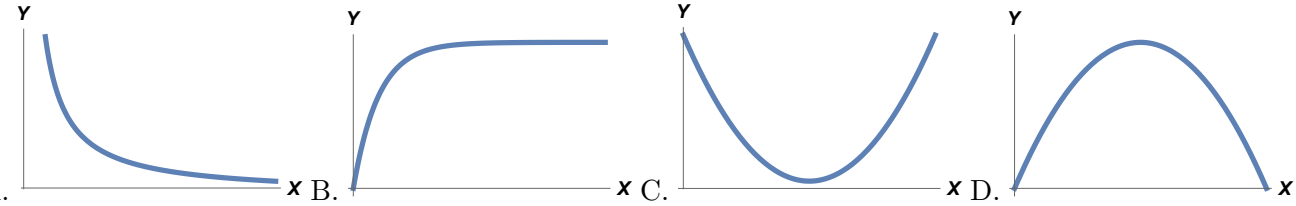
(d) What problem with our algorithm is revealed by this example? Very briefly explain a small change to the method or parameters we are using that will solve this problem.

Solution: Use a smaller learning rate.

Name: _____

4 The Plot Thickens

4. (10 points) Consider just the general shape of the following plots. For each of the following possible interpretations of the quantities being plotted on the X and Y axes, indicate which of the plots would most typically be the result, or mark "none" if none are appropriate.



Assume all quantities other than X are held constant during the experiment. Error quantities reported are averages over the data set they are being reported on.

Provide a one-sentence justification for each answer.

(a) **X axis**: Number of training examples **Y axis**: test set error

√ **A** ◯ B ◯ C ◯ D ◯ none

Solution: With more training data, we are better able to find a good hypothesis.

(b) **X axis**: Number of training examples **Y axis**: training error

◯ A √ **B** ◯ C ◯ D ◯ none

Solution: It's easy to fit a small amount of data exactly; harder as we get more data.

(c) **X axis**: Order of polynomial feature set **Y axis**: test set error

◯ A ◯ B √ **C** ◯ D ◯ none

Solution: Underfits if too low; overfits if too high.

(d) **X axis**: Order of polynomial feature set **Y axis**: training set error
√ **A** ◯ B ◯ C ◯ D ◯ none

Solution: More features makes it easier to fit complex data.

(e) **X axis**: Order of polynomial feature set **Y axis**: cross validation error
◯ A ◯ B √ **C** ◯ D ◯ none

Solution: Same as test set error.

Name: _____

5 Convenient network

5. (10 points) We'll consider here a simple one-dimensional convolutional neural network layer. This is a feature map created by a single filter whose parameters we must learn. The filter represents a local pattern detector that is applied in every position of the input signal. The feature map therefore transforms an input vector (one dimensional signal) into another vector (one-dimensional feature map). To train such a layer on its own, i.e., not as part of a bigger network as we typically do, we can imagine having training pairs (x, y) where x is the input signal as a vector and y is a binary vector representing whether the relevant pattern appeared in a particular position or not. Specifically,

   - Input x is a one-dimensional vector of length d.
   - Target y is also a one-dimensional vector of length. d. One "pixel" in the output, y_j, has value 1 if the input pixels x_{j-1}, x_j, x_{j+1}, centered at j, exhibit the target pattern and 0 if they do not.
   - The filter is represented by a weight vector w consisting of three values.
   - The output of the network is a vector $\hat{y}$ whose j^{th} coordinate (pixel) is $\hat{y}_j = \sigma(z_j)$ where $z_j = [x_{j-1}; x_j; x_{j+1}]^T w$ and $\sigma(\cdot)$ is the sigmoid function. Assume that x_0 and x_{d+1} are 0 for the purposes of computing outputs.
   - We have a training set $D = (x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$.
   - We measure the loss between the target binary vector y and the network output $\hat{y}$ pixel by pixel using cross-entropy (Negative Log-Likelihood or NLL). The aggregate loss over the whole training set is

$$L(w, D) = \sum_{i=1}^{n} \sum_{j=1}^{d} \text{NLL}(y_j^{(i)}, \hat{y}_j^{(i)})$$

   (a) What is the "stride" for this feature map?

   Solution: 1

(b) Provide a formula for $\nabla_w \text{NLL}(y_j, \hat{y}_j)$, which is the gradient of the loss with respect to pixel j of an example with respect to $w = [w_1, w_2, w_3]^T$, in terms of x, y, and z values only.

Solution:

$$(\sigma(z_j) - y_j) \begin{bmatrix} x_{j-1} \\ x_j \\ x_{j+1} \end{bmatrix}$$

(c) We'd like to use a simple stochastic gradient descent (SGD) algorithm for estimating the filter parameters w. But wait... how is the algorithm stochastic? Given the framing of our problem there may be multiple ways to write a valid SGD update.

For each of the following cases of SGD, write an update rule for w, in terms of step size η, $\nabla_w \text{NLL}$, target pixel values $y_j^{(i)}$ and actual pixel values $\hat{y}_j^{(i)}$.

i. Update based on all pixels of example i

Solution: $w \leftarrow w - \eta \sum_{j=1}^{d} \nabla_w N(y_j^{(i)}, \hat{y}_j^{(i)})$

ii. Update based on pixel j of all examples

Solution: Select j (position) at random, $w \leftarrow w - \eta \sum_{i=1}^{n} \nabla_w N(y_j^{(i)}, \hat{y}_j^{(i)})$

iii. Update based on pixel j of example i

Solution: Select i (example) and j (position) at random, $w \leftarrow w - \eta \nabla_w N(y_j^{(i)}, \hat{y}_j^{(i)})$

Name: _____

6 The deep end of the pool

6. (10 points) We are going to consider two different simple convolutional networks over one dimensional (vector) inputs. Each network has a single convolutional layer with a single filter of size 3 and stride 1. Let $(z_1, \ldots, z_d)$ be the output of this convolutional layer, i.e., $(z_1, \ldots, z_d)$ represents the feature map constructed from the input vector $(x_1, \ldots, x_d)$. For simplicity, you can think of z_j just as a linear map $z_j = [x_{j-1}; x_j; x_{j+1}]^T w$ where w are the filter parameters. Our two networks differ in terms of how the feature map values are pooled to a single output value.

   Network A has a single *max-pooling* layer with input size d, so that the output of the network $\hat{y} = \sigma(\max(z_1, \ldots, z_d))$ where $\sigma(\cdot)$ is the sigmoid function.

   Network B has a single *min-pooling* layer with input size d, so that the output of the network $\hat{y} = \sigma(\min(z_1, \ldots, z_d))$.

   When the filter's output value is high it represents a positive detection of some pattern of interest.

   (a) For which network does a high output value correspond, qualitatively, to "every pixel in x corresponds to an instance of the desired pattern"?

   ◯ A ✓ **B** ◯ None

   (b) For which network does a high output value correspond, qualitatively, to "at least half of the pixels in x correspond to an instance of the desired pattern"?

   ◯ A ◯ B ✓ **None**

   (c) For which network does a high output value correspond, qualitatively, to "there is at least one instance of the desired pattern in this image"?

   ✓ **A** ◯ B ◯ None

   (d) Assume for simplicity that all $z_1, \ldots, z_d$ have distinct values (we can ignore the corner cases where some of the values are equal). What is $\partial\hat{y}/\partial z_i$ for network A?

   Solution: $\sigma(z_i)(1 - \sigma(z_i))$ if $z_i = \max(z_1, \ldots, z_d)$, and 0 otherwise.

(e) Now, suppose we are just given a single training pair (x, y) where the target y is binary 0/1. The loss that we are minimizing is again just

$$\text{NLL}(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

which is minimized when $\hat{y}$ matches the target y. We are interested in understanding qualitatively how the filter parameters w get updated in the two networks if we use simple gradient descent to minimize $\text{NLL}(y, \hat{y})$. For each of the four qualitative behaviors of this process described below, specify the situation in which it would occur:

- Which network (A, B, or it doesn't matter)
- Target y (1, 0, or it doesn't matter)

The behaviors are:

1. The filter weights w become increasingly aligned in the direction of a particular triplet $[x_{j-1}; x_j; x_{j+1}]$
   i. Network: √ **A** ◯ B ◯ irrelevant
   ii. Target y: √ **1** ◯ 0 ◯ irrelevant

2. The filter weights w become increasingly aligned in the *negative* direction of a particular triplet $[x_{j-1}; x_j; x_{j+1}]$.
   i. Network: ◯ A √ **B** ◯ irrelevant
   ii. Target y: ◯ 1 √ **0** ◯ irrelevant

3. Each update moves the filter weights w in the direction of some triplet but the specific triplet keeps changing from one update to another
   i. Network: ◯ A √ **B** ◯ irrelevant
   ii. Target y: √ **1** ◯ 0 ◯ irrelevant

4. Each update causes the filter weights w to move in the *negative* direction of some triplet but the specific triplet may change from one update to next
   i. Network: √ **A** ◯ B ◯ irrelevant
   ii. Target y: ◯ 1 √ **0** ◯ irrelevant

Name: _____

7 Q learning and NNs

7. (10 points) Chris wants to use Q-learning to solve a video-game problem in which there is a ball moving on an n by n pixel screen, similar to the one we studied in class. However, instead of moving a paddle up and down along the right wall, there is a "photon cannon" fixed in the middle of the right-hand side, and the player is allowed to instantaneously set the angle of the cannon and to try to shoot. If the photon beam hits the ball, the ball will reflect backwards. It takes 10 time steps for the cannon to recharge after being fired, however, before it can be fired again. Our goal here is to try to understand how to apply deep Q-learning to this problem.

   The state of the system is composed of five parts:

   - Ball position x (1 .. n)
   - Ball position y (1 .. n)
   - Ball velocity x (-1, 1)
   - Ball velocity y (-1, 0, 1)
   - Number of time steps until the cannon is ready to shoot again (0,. . . ,10)

   The possible actions at each time step involve both the aim and whether to try to shoot:

   - Cannon angle in degrees (-60, -30, 0, 30, 60)
   - Shoot cannon (1, 0)

   The options for us in terms of solving the game include how we represent the states and actions and how these are mapped to Q-values. We won't worry about the exploration problem here, only about representing Q-values. Our learning algorithm performs gradient descent steps on the squared Bellman error with respect to the parameters in the Q-values.

   (a) How many states and actions are there in this game?

   Solution: $n^2 * 2 * 3 * 11$ states and 6 (or $5 * 2$) actions

   (b) Let S be the set of states, and A the set of actions. Suppose we construct a simple one layer neural network to represent Q-values. The network has $|S| + |A|$ input units, no hidden units, and just one linear output unit to represent the associated Q-value. The pair (s, a) is fed into the model by concatenating a one-hot vector for s and a one-hot vector for a. Could this model learn to match the correct Q-values for each state-action pair? Briefly describe why/why not.

   Solution: No, since the model is restricted. The value is predicted as a sum of a state-dependent and action-dependent parts without any cross-talk.

   (c) Suppose we modify the network a bit by giving it $|S|$ input units and $|A|$ output units where the output units represent the Q-values $Q(s, a)$, $a \in A$, for the state s fed in as a one-hot vector. Again, we have no hidden units. Could this model match the correct Q-values? Why/why not.

Solution: Yes, it could. We can specify arbitrary outgoing weights for each input state thus can set the Q-values without restriction.

(d) What if we only introduced five input units, one unit for each of the five parts of the state. With m hidden units, and $|A|$ output units, could we now always represent the correct Q-values? Why/why not.

Solution: Yes, we can, provided that m is large enough. A neural network with a single hidden layer is able to represent arbitrary mappings.

(e) If we increase n and also include many more angle gradations for the aim, so that $|S|$ and $|A|$ are very large, which of the following architectures would we prefer for representing Q-values?

- ◯ $|S|$ input units (one-hot vector for s), $|A|$ output units
- ◯ 5 input units, some m hidden units, $|A|$ output units
- √ **5+2 input units for the five part state, two-part action, m hidden units and one output unit**
- ◯ 5 input units, some m hidden units, and two output units

Briefly explain your choice.

Solution: Need to be able to generalize over both input and output space.

Name: _____

8 RNN

8. (10 points) Consider three RNN variants:

   1. The **basic** RNN architecture we studied was

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

   where W^{ss} is $m \times m$, W^{sx} is $m \times d$, and f is an activation function to be specified later. We omit the offset parameters for simplicity (set them to zero).

   2. **Randdy** thinks the basic RNN is representationally weak, and it would be better not to decompose the state update in this way. Randdy's proposal is to instead

   $$s_t = f(W^{ssx} \mathit{concat}(s_{t-1}, x_t))$$
   $$y_t = W^o s_t$$

   where $\mathit{concat}(s_{t-1}, x_t)$ is a vector of length $m + d$ obtained by concatenating s_{t-1} and x_t, so W^{ssx} has dimensions $m \times (m + d)$.

   3. **Orenn** wants to try yet another model, of the form:

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

   Lec Surer insists on understanding these models a bit better, and how they might relate.

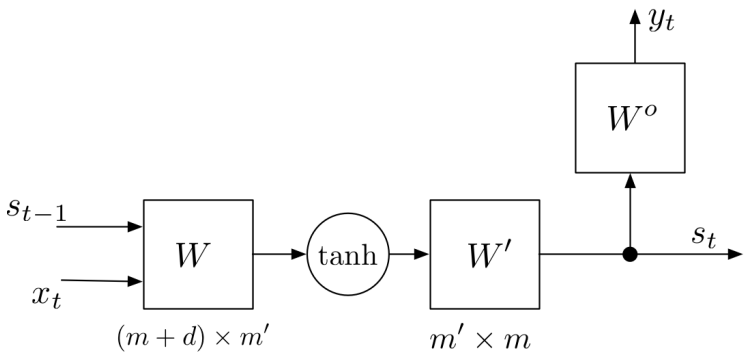
   (a) Select the correct claim and answer the associated question.

   (1) Claim: The three models are all equivalent when $f(z) = z$. In this case, define W^{ssx} in terms of W^{ss} and W^{sx} so that the equivalence holds.

   (2) Claim: The three models are not all equivalent when $f(z) = z$. In this case, assume $m = d = 1$ and provide one setting of W^{ssx} in Randdy's model such that W^{ss} and W^{sx} cannot be chosen to make the basic and Orenn's models the same as Randdy's.

Solution: Claim 1 $W^{ssx} = \mathit{hstack}(W^{ss}, W^{sx})$

(b) Lec Surer thinks that something interesting happens with Orenn's model when $f(z) = \tanh(z)$. Specifically, it supposedly corresponds to the architecture shown in the figure below, which includes an additional hidden layer. Specify what W, W', and m' are so that this architecture indeed corresponds to Orenn's model.



Ignore the dimensions written on the figure above; they are backwards.

i. m'

Solution: $2m$

ii. W

Solution: A block-diagonal matrix of the form

$$\begin{bmatrix} W^{ss} & 0 \\ 0 & W^{sx} \end{bmatrix}$$

iii. W'

Solution: $hstack(I(m); I(m))$

(c) Assume again that $f(z) = \tanh(z)$. Suppose $s_0 = 0$ (vector) and we feed $x_1, \ldots, x_n$ as the input sequence to Orenn's model, obtaining $y_1, \ldots, y_n$ as the associated output sequence. If we change the input sequence to $-x_1, \ldots, -x_n$, which of the following is the best characterization of the resulting output sequence?

◯ The new output sequence will alternate between positive and negative values.

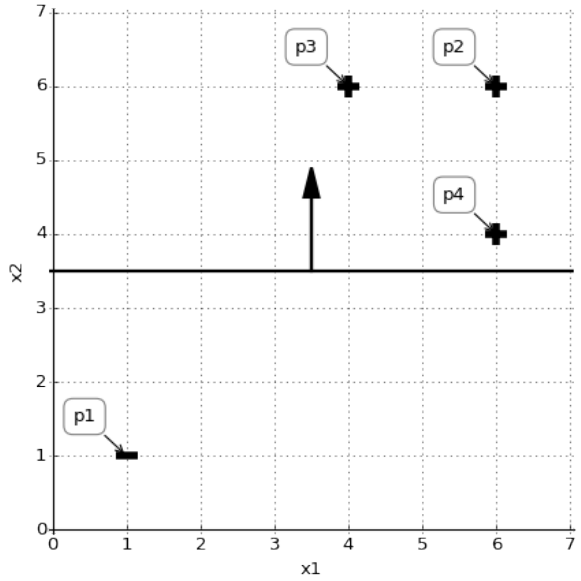
◯ The new output sequence depends on the parameters.

√ **The new output is just the negative of the previous output sequence**

Name: _____

9 Marginalia

9. (12 points) Here is a data set and a separator:



(a) What is the margin for each point with respect to the separator shown in the figure?

i. p_1 **2.5**

ii. p_2 **2.5**

iii. p_3 **2.5**

iv. p_4 **0.5**

Let our objective be the regularized average hinge loss with respect to γ_{ref}:

$$J(\theta, \theta_0, \gamma_{ref}) = \frac{1}{n} \sum_{i=1}^{n} L_H \left(\frac{\gamma(x, y, \theta, \theta_0)}{\gamma_{ref}} \right) + \lambda \frac{1}{\gamma_{ref}^2}$$

(b) If $\lambda = 0$, and we're still referring to the separator in the figure, what range of values of γ_{ref} achieves the optimal value of J?

Solution: $\gamma_{ref} < 0.5$

(c) Now let $\lambda = \epsilon$, a very small value, and consider that same separator. What value of γ_{ref} achieves the closest to optimal value of J?

- ◯ -10
- ◯ -1
- ◯ -0.5
- ◯ 0
- √ **0.5**
- ◯ 1
- ◯ 10

(d) When $\lambda = \epsilon$, and using the maximum margin separator, supply a value of γ_{ref} that approximately minimizes J.

$\gamma_{ref} =$ $2\sqrt{2}$

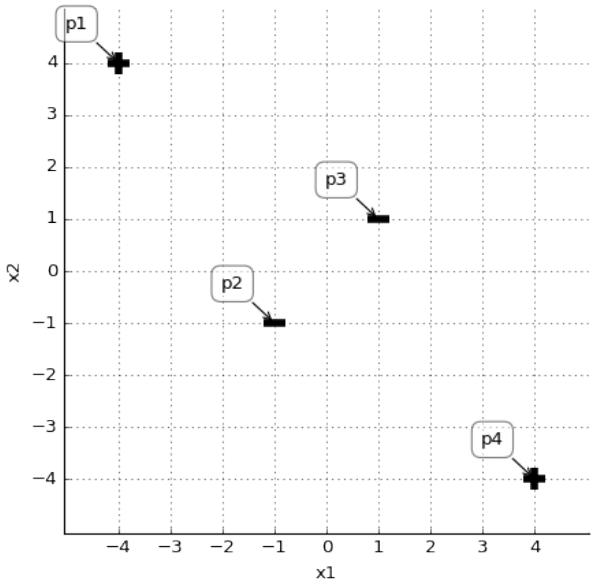
(e) Why might we prefer a maximum-margin separator over the one originally provided?

Solution: We expect it will generalize better because it is not as dependent on the data points (a small variation in the data probably won't change the result too much).

Name: _____

10 Kopy Kat

10. (12 points) Consider the following data. Clearly, the points are not linearly separable, so we will try three alternative approaches to solving the problem.



Approach 1: Nested linear classifiers

Let $w = \begin{bmatrix} +1 \\ -1 \end{bmatrix}$ and $a = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$ where

$$a_1 = \text{sign}(w^T x + 4)$$
$$a_2 = \text{sign}(w^T x - 4)$$
$$y_{predicted} = \text{sign}(v_1 a_1 + v_2 a_2 + v_0)$$

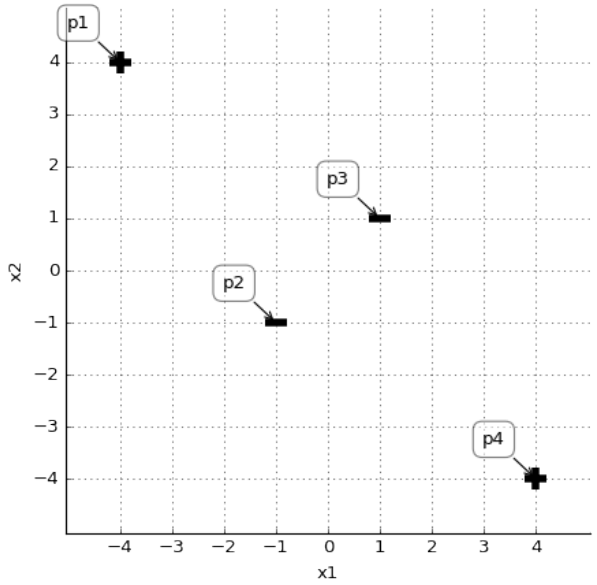
(a) Select values of these parameters so that the nested classifier correctly predicts the value in the data set.

i. v_1 **-1**

ii. v_2 **1**

iii. v_0 **.5**

Name: _____



Approach 2: Using kernels

Suppose $K(x, x') = e^{-\|x-x'\|^2}$ and for the points in our data set, $K(p_i, p_j) = \mathbf{K}_{ij}$ where the matrix

$$\mathbf{K} \approx \begin{bmatrix} 1 & \exp(-15) & \exp(-15) & \exp(-56) \\ \exp(-15) & 1 & \exp(-4) & \exp(-15) \\ \exp(-15) & \exp(-4) & 1 & \exp(-15) \\ \exp(-56) & \exp(-15) & \exp(-15) & 1 \end{bmatrix}$$

(b) Now we will use the kernel perceptron algorithm to find the α values in the classifer, which has the form

$$y_{predicted} = \text{sign}(\sum_{i=1}^{4} \alpha_i y^{(i)} K(x^{(i)}, x))$$

Assuming that we go through the points in order $p1, p2, p3, p4$ as many times as necessary to correctly classify the data, what are the values of

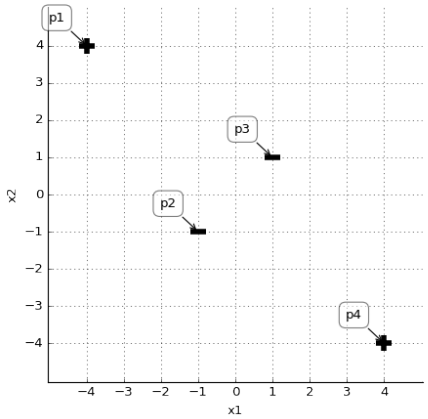
i. α_1 **1**

ii. α_2 **1**

iii. α_3 **0**

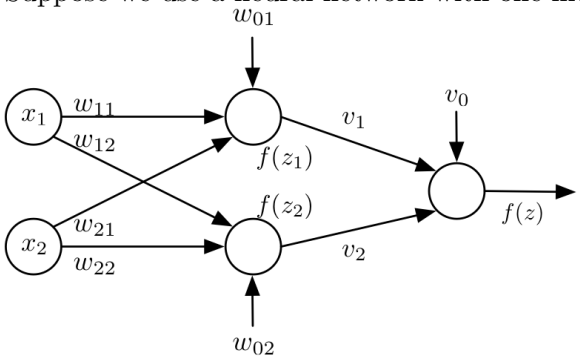
iv. α_4 **1**

Name: _____



Approach 3: Neural nets

Suppose we use a neural network with one hidden layer to classify the points, as shown below.



(c) We can classify the points correctly if f is tanh. Assume that
$w_{11} = w_{12} = +1$ and
$w_{21} = w_{22} = -1$.
Provide the rest of the weights so this network will correctly classify the given points.

i. v_1 **-1**

ii. v_2 **1**

iii. v_0 **.5**

iv. w_{01} **4**

v. w_{02} **-4**

Name: _____

Work space

Name: _____

Work space