6.036: Final, Spring 2018

Solutions

- This is a closed book exam; you are allowed 2 pages of notes.
- Calculators and phones are not permitted.
- The problems are not necessarily in any order of difficulty.
- Record all your answers in the places provided. If you run out of room for an answer, continue on a blank page and mark it clearly.
- If a question seems vague or under-specified to you, make an assumption, write it down, and solve the problem given your assumption.
- **Write your name on every page.**
- Come to the front to ask questions.

Name: _____ Athena ID: _____

Question	Points	Score
1	20	
2	8	
3	14	
4	10	
5	8	
6	15	
7	10	
8	7	
9	8	
Total:	100	

Name: _____

Feature mapping

1. (20 points) In this problem, we will consider two-dimensional input data vectors $x = [x_1, x_2]^T$. We will explore the impact of a feature transformation on various learning methods.

   We will be using the feature transformation

   $$\phi(x) = [1, x_1, x_2, x_1 x_2, x_1^2, x_2^2]^T$$

   Consider the following 2D data sets:

   Classic XOR: positive: $(0, 1), (1, 0)$ and negative: $(0, 0), (1, 1)$.

   Signed XOR: positive: $(-1, 1), (1, -1)$ and negative: $(-1, -1), (1, 1)$

   (a) For each dataset, is there a vector θ^T for a linear classifier through the origin ($\theta^T x$) that perfectly separates the data? If so, give the θ vector, else explain briefly why not. Make sure that the prediction has the correct sign.

   Classic XOR: positive: $(0, 1), (1, 0)$ and negative: $(0, 0), (1, 1)$.

   Solution: No. Data is not linearly separable.

   Signed XOR: positive: $(-1, 1), (1, -1)$ and negative: $(-1, -1), (1, 1)$

   Solution: No. Data is not linearly separable.

(b) For each dataset, is there a vector θ^T for a linear classifier through the origin ($\theta^T \phi(x)$) that perfectly separates the data? If so, give the θ vector, else explain briefly why not. Make sure that the prediction has the correct sign.

$$\phi(x) = [1, x_1, x_2, x_1 x_2, x_1^2, x_2^2]^T$$

Classic XOR: positive: $(0, 1), (1, 0)$ and negative: $(0, 0), (1, 1)$.

Solution: Yes. $(-\epsilon, 0.5, 0.5, -1, 0, 0)$, for example.

Signed XOR: positive: $(-1, 1), (1, -1)$ and negative: $(-1, -1), (1, 1)$

Solution: Yes. $0, 0, 0, -1, 0, 0$

(c) For the dataset indicated below, could a one-hidden-layer neural network with x_1 and x_2 as inputs, a layer of **up to four** relu units and a final tanh output unit be trained to separate the data set? If yes, show the network with weights, including offsets if any. If no, explain briefly why not. Make sure that the prediction has the correct sign.

The network is specified as follows:

$$z = W^T x + W_0$$
$$o = \tanh(V^T \text{relu}(z) + V_0)$$

Assuming you use $k \leq 4$ hidden units, W is $2 \times k$, W_0 is $k \times 1$ and V is $k \times 1$ and V_0 is 1×1. Write down W, W_0, V, V_0.

Signed XOR: positive: $(-1, 1), (1, -1)$ and negative: $(-1, -1), (1, 1)$

Solution: Yes. The simplest way is to have four ReLU units. The ith ReLU unit is responsible for being positive when given the ith input, and negative when given any of the other three inputs. The connection between the ith ReLU unit and the tanh layer should be a large positive number when the ith label is +1, and a large negative number when the ith label is -1.

(d) For the dataset indicated below, could a one-hidden-layer neural network with the entries in $\phi(x)$ as inputs, a layer of **up to four** relu units and a final tanh output unit be trained to separate the data set? If yes, show the network with weights, including offsets if any. If no, explain briefly why not. Make sure that the prediction has the correct sign.

The network is specified as follows:

$$z = W^T \phi(x) + W_0$$
$$o = \tanh(V^T \text{relu}(z) + V_0)$$

Assuming you use $k \leq 4$ hidden units, W is $6 \times k$, W_0 is $k \times 1$ and V is $k \times 1$ and V_0 is 1×1. Write down W, W_0, V, V_0.

$$\phi(x) = [1, x_1, x_2, x_1 x_2, x_1^2, x_2^2]^T$$

Signed XOR: positive: $(-1, 1), (1, -1)$ and negative: $(-1, -1), (1, 1)$

Solution: Yes, using weights as in part (b).

Name: _____

Trees

2. (8 points) We will continue the example from the previous question.

   (a) For the dataset indicated below, construct a decision tree (using the algorithm from class, based on weighted entropy) with the original features $x = [x_1, x_2]^T$. Use tests of the form $f < v$. If there is a tie in the choice of split, first prefer x_1 and then smaller thresholds. You do **not** need to provide numerical values of the weighted entropy.

   Classic XOR: positive: $(0, 1), (1, 0)$ and negative: $(0, 0), (1, 1)$.

Solution:

```
x_1 < 0.5
 T: x_2 < 0.5
    T: -1
    F: +1
 F: x_2 < 0.5
    T: +1
    F: -1
```

(b) For the dataset indicated below, construct a decision tree (using the algorithm from class, based on weighted entropy) with features from $\phi(x)$. If there is a tie in the choice of split, first prefer features that appear earlier in the $\phi(x)$ vector and then smaller thresholds. Use tests of the form $f < v$. You do **not** need to provide numerical values of the weighted entropy.

$$\phi(x) = [1, x_1, x_2, x_1x_2, x_1^2, x_2^2]^T$$

Classic XOR: positive: $(0,1), (1,0)$ and negative: $(0,0), (1,1)$.

Solution:

```
x_1 x_2 < 0.5
 T: x_1 < 0.5
    T: x_2 < 0.5
       T: -1
       F: +1
    F: +1
 F: -1
```

(c) For **any** dataset with only positive-valued x_1 and x_2, what features in $\phi(x)$ **cannot** possibly appear in a decision tree computed by the algorithm from class. Assume the splitting rule described earlier: if there is a tie in the choice of split, first prefer features that appear earlier in the $\phi(x)$ vector and then smaller thresholds. Explain your answer.

$$\phi(x) = [1, x_1, x_2, x_1x_2, x_1^2, x_2^2]^T$$

Solution: Features 1, x_1^2 and x_2^2 cannot appear. The first one provides no information and the square terms (for positive data values) create the same splits in the data as the x_1 and x_2 features.

Name: _____

Generalization

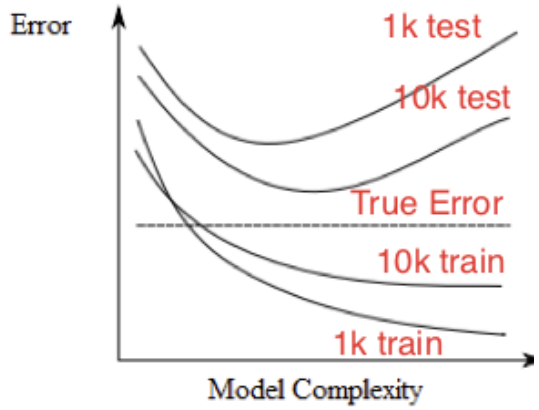
3. (14 points) (a) Assume two data sets are sampled from the same distribution where data set 1 has 1,000 elements and data set 2 has 10,000 elements. Also assume we randomly construct train and test sets from both data sets by dividing them into 90% training and 10% testing.

   We will explore the effect of using models of increasing complexity (you can think of this as decreasing regularization).

   - Draw **two curves**, for training error and test error, **for each data set** with the y-axis denoting the error and the x-axis denoting the model complexity.
   - You should have total of 4 curves: one training error and one test error curve for each dataset.
   - Draw all 4 of them in the same diagram below. We have included the *true error* value on the diagram; this is the error that the correct model has on this data.
   - Clearly mark your curves with the labels: 1K train, 1K test, 10K train, 10K test.

   The following factors will be used for grading:

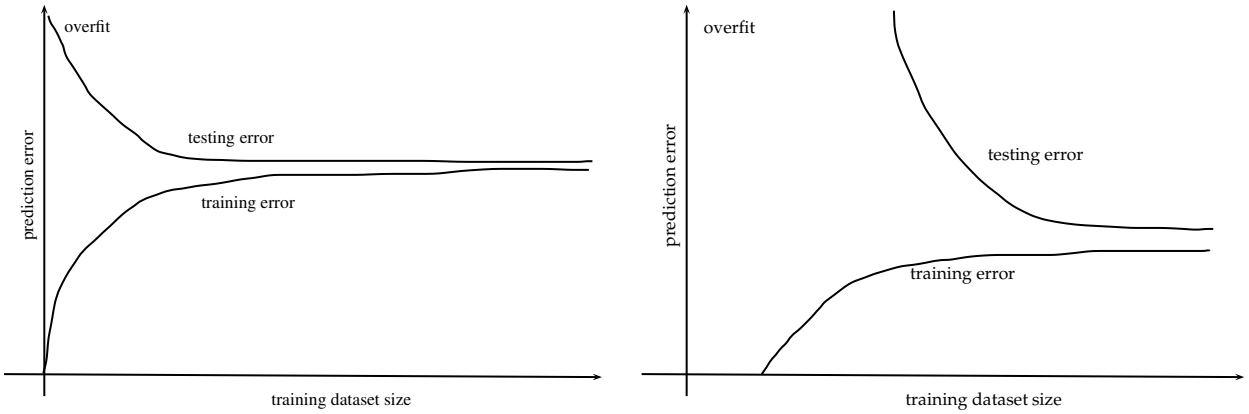
   - The general shape of the curves.
   - The relative ordering of the curves in the "Prediction Error" direction.



Solution:

- Training error is lower than the true error (with sufficient model complexity), while test error is higher, as we are fitting to the training data
- Training error decreases with increasing model complexity, as we have increased capacity to fit the data
- Test error initially decreases with increasing model complexity and then increases, as we start to fit the data better and then proceed to overfit
- The 10k dataset makes it more difficult to overfit, so training error is higher and test error lower compared to their 1k counterparts.

(b) Consider these training and test curves as a function of training dataset size. These are for two models: one simple and one complex. Which is which? Explain your choice.



Left: √ **simple** ◯ complex Right: ◯ simple √ **complex**

Solution: The complex model more easily overfits, so test error is initially worse (and training error better), but with sufficient data (to prevent overfitting) the more complex model performs better.

(c) In some cases, we will have a validation set in addition to training and test sets. Assume the validation set is approximately the same size as the test set. This validation set is often used to tune hyperparameters such as λ. Imagine we have trained a classifier using regularization, with λ chosen based on performance on the validation set.

i. Which will generally have the highest accuracy (check two if you expect a tie)?
◯ training set √ **validation set** ◯ test set

ii. Which will generally have the lowest accuracy (check two if you expect a tie)?
◯ training set ◯ validation set √ **test set**

(d) If we instead used the performance on the training set to choose λ,

i. Which will generally have the highest accuracy (check two if you expect a tie)?
√ **training set** ◯ validation set ◯ test set

ii. Which will generally have the lowest accuracy (check two if you expect a tie)?
◯ training set ◯ validation set √ **test set**

(e) An alternative to cross-validation for estimating prediction error is to use "bootstrap samples". These are datasets constructed by randomly sampling points from the original training set with replacement, that is, we do not remove previously sampled points, so a data point could appear more than once in a bootstrap sample. Consider the following alternative methodologies, assuming the training dataset contains N samples.

1. Generate K bootstrap samples of size N, train on each sample and evaluate on the original training dataset. Return average of results.
2. Generate K bootstrap samples of size N, train on the original training dataset and evaluate on each sample. Return average of results.
3. Generate K bootstrap samples of size N, train on each sample and evaluate on points in the original training dataset but not in the sample (assume there are always some such points). Return average of results.

Order these (from best to worst) by how accurate you expect the estimates of prediction error on unseen test data to be. Explain your answer.

Solution: 3, 1, 2
The more unfamiliar your test data, the more accurate the evaluation will be.

Name: _____

At a loss

4. (10 points) Consider a classification problem in which there are K possible output classes, $1, \ldots, K$. We have studied using NLL as a loss function in such cases, but that assumes that all mistaken classifications are equally significant.

   Instead, we'll consider a case where some mistakes are worse than others (e.g., mis-identifying a cow as a horse is not as bad as calling it a mouse). Define the cost matrix $c_{g,a}$ to be the cost for guessing class g when the actual class is a. For convenience, we'll write c_j for the column of the matrix $[c_{1,j}, c_{2,j}, \ldots, c_{K,j}]^T$ representing the costs of all the possible guesses when j is the actual value.

   We will use a simple neural network with a softmax activation function, so our prediction p will be a $K \times 1$ vector:

   $$p = \texttt{softmax}(z)$$
   $$z = W^T x$$

   Assume inputs are $d \times 1$ so W is $d \times K$.

   Our loss function, for a prediction vector p when the target output is value $y \in \{1, \ldots, K\}$ is the expected cost of the prediction:

   $$L_c(p, y) = \sum_{k=1}^{K} p_k c_{ky} = p^T c_y$$

   So, the overall training objective is to minimize, over a data set of n points,

   $$J_c(W) = \sum_{i=1}^{n} L_c(p^{(i)}, y^{(i)}) \ .$$

   (a) Select which of the following cost matrices c corresponds to each situation described below.

   A. $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ B. $\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$ C. $\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$ D. $\begin{bmatrix} 0 & .5 & .5 & .5 \\ 2 & 0 & 1 & 1 \\ 2 & 1 & 0 & 1 \\ 2 & 1 & 1 & 0 \end{bmatrix}$ E. $\begin{bmatrix} 0 & 2 & 2 & 2 \\ .5 & 0 & 1 & 1 \\ .5 & 1 & 0 & 1 \\ .5 & 1 & 1 & 0 \end{bmatrix}$

   i. Any output that is not the single preferred answer is penalized equally.
   ◯ A ✓ **B** ◯ C ◯ D ◯ E

   ii. There are two pairs of outputs that are interchangeable with no penalty.
   ◯ A ◯ B ✓ **C** ◯ D ◯ E

   iii. It is worse to miss predicting a particular bad outcome than to predict that outcome by mistake.
   ◯ A ◯ B ◯ C ✓ **D** ◯ E

(b) What would the change to the weights W be, in one step of stochastic gradient descent on J_c, with input x and target output y, and step size η?

Computing $\partial p/\partial z$ is kind of hairy. It is a $K \times K$ matrix. You can write your answer in terms of it without computing it. You may also use x, y, W, and/or c in your solution.

Solution:

$$-\eta \cdot x \cdot (\frac{\partial p}{\partial z} \cdot c_y)^T$$

. To calculate the SGD update, we first need to calculate $\frac{\partial J_c}{\partial W}$. We use chain rule.

$$\frac{\partial J_c}{\partial W} = \frac{\partial J_c}{\partial L_c}\frac{\partial L_c}{\partial p}\frac{\partial p}{\partial z}\frac{\partial z}{\partial W} = 1 \cdot (c_y^T)(\frac{\partial p}{\partial z}^T) \cdot x = x \cdot (\frac{\partial p}{\partial z} \cdot c_y)^T$$

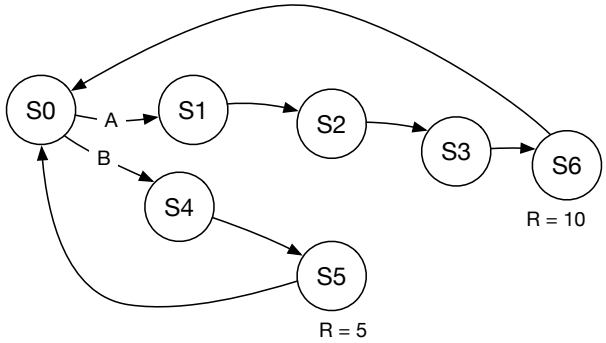
The SGD update is then

$$-\eta \cdot x \cdot (\frac{\partial p}{\partial z} \cdot c_y)^T$$

Name: _____

Murky decision problem

5. (8 points) Consider the following Markov decision process:



Assume:

- Reward is 0 in all states, except +10 in s6 and +5 in s5; the reward is received when *exiting* the state.
- Transitions out of s0 are deterministic, and depend on the choice of action (A or B).

(a) Assume in this part that all transitions are deterministic, following the arrows indicated with probability 1. When horizon = 3 and discount factor $\gamma = 1$, provide values for:

i. $Q(s_0, A)$ **0**

ii. $Q(s_0, B)$ **5**

(b) Still assuming that all transitions are deterministic, but letting horizon = 5 and discount factor $\gamma = 1$, provide values for:

i. $Q(s_0, A)$ **10**

ii. $Q(s_0, B)$ **5**

(c) Now, assume that transitions out of s_0 are deterministic, but that all other transitions follow the arrows indicated with probability 0.9 and stay in the current state with probability 0.1.

For policy $\pi(s_0) = B$, write a system of equations that can be solved in order to compute $V_\pi(s_0)$ when the horizon is infinite and $\gamma = 0.8$.

Do not solve the equations!

Solution:

$$v_0 = 0.8v_4$$
$$v_4 = 0.8(0.1v_4 + 0.9v_5)$$
$$v_5 = 5 + 0.8(0.1v_5 + 0.9v_0)$$

Name: _____

Groundhog day

6. (15 points) We will be performing Q-learning in an MDP with states s_0 through s_5, and actions a_1 and a_2. Let the discount factor $\gamma = 0.8$ and learning rate $\alpha = 1$.

   Note: do not assume anything more about the MDP from which this experience was generated. It is not necessarily the same as the one from question 5.

   (a) The Q-learning algorithm begins with $Q(s, a) = 0$ for all s and a. It receives the following experience, in order. Each tuple represents a state, action, reward, and next state.

$$(s_0, a_2, 0, s_2)$$
$$(s_2, a_1, 0, s_3)$$
$$(s_3, a_1, 0, s_1)$$
$$(s_1, a_1, 10, s_0)$$
$$(s_0, a_1, 0, s_5)$$
$$(s_5, a_1, 0, s_4)$$
$$(s_4, a_1, 5, s_0)$$

   Fill in the resulting Q values in the following table:

	s_0	s_1	s_2	s_3	s_4	s_5
a_1	0	10	0	0	5	0
a_2	0	0	0	0	0	0

   (b) Iyaz suggests that, rather than getting new experience, it would be a good idea to replay this data over several times using the regular Q-learning update. What's the minimum number of times you would have to iterate through this data before $Q(s_0, a_2) > Q(s_0, a_1$? *Note: it should be possible to answer this question by thinking about the structure of the problem, rather than by grinding through more Q-learning update calculations.*

4

Solution: 4, including the first update whose values we recorded in the table).

(c) Now, imagine that you have a very large memory full of experience that explores the whole environment effectively. In the following questions, you can assume that a neural network `nn` has the following methods:

- `nn.train(data, epochs)` takes a list of (x, y) pairs and does supervised training on it for the specified number of epochs. If `epochs` is `None` it trains until convergence.
- `nn.predict(x)` takes an input x and returns the predicted y value
- `nn.init()` randomly reassigns all the weights in the network.

Consider the following code.

```
# nn = dictionary of neural networks, one for each action
# each nn[a] maps state s into Q(s, a)
gamma = 0.8
for t in range(max_iterations):
    for a in actions:
        data[a] = [(s, <fill in>) for (s, a_prime, r, s_prime) in memory
                                                  if a_prime == a]
    for a in actions:
        nn[a].train(data[a], <epochs>)
```

What is a correct expression for `<fill in>` above?

Don't panic about detailed Python syntax; you may use words to clarify if you think you might be wrong.

Solution: `r + 0.8 * max([nn[a_prime].predict(s_prime) for a_prime in actions])`. We want the Q-learning update since we want to pick the max predicted value for the given state over all actions.

What is an appropriate value for `<epochs>` above?

Solution: `None`. We want to train until convergence.

(d) If we change the loop to have the form

```
for t in range(max_iterations):
    for (s, a, r, s') in memory:
        data = [(s, <fill in>)]   # a single data point
        nn[a].train(data, <epochs>)
```

Provide a value for `<epochs>` above that will cause this algorithm to converge to a correct solution or explain why no such value exists.

Solution: 1. With 1 epoch we will look at every piece of experience in memory once per iteration.

(e) Would it be okay to call `nn[a].init()` on the line before calling `train` in the code loop in part c?

√ **Yes** ◯ No

Provide a short explanation of your answer.

Solution: We have used the old networks to compute targets for all our training points, so it's okay to start over and train up the function approximators again.

(f) Would it be okay to call `nn.init()` on the line before calling `train` in the code loop in part d?

◯ Yes √ **No**

Provide a short explanation of your answer.

Solution: We are just making a small gradient step after each data point here, so we rely on the networks maintaining their values.

(g) We often use ϵ-greedy exploration in Q learning, in which we execute the action with the highest Q value in the current state with probability $1 - \epsilon$ and execute a random action with probability ϵ. What problem might occur if we set ϵ to be too small?

Solution: We might get stuck for a long time doing a sub-optimal action choice due to lack of exploration.

Name: _____

Commendation

7. (10 points) Consider a recommender system that has n users and m movies, with sparse ratings available for each user and movie.

   Our typical strategy is to look for matrices U and V such that UV^T is a rank k approximation to the true matrix at the locations it has been observed.

   (a) M.A. Trix suggests a new decomposition of the solution matrix X into UWV^T where W is a $k \times k$ matrix, and U and V are as in the original approach.

   Is M.A. Trix's approach able to represent:

   ◯ A richer class of models than the original?

   ◯ A smaller class?

   √ **The same class?**

   Provide a short concrete justification of your answer.

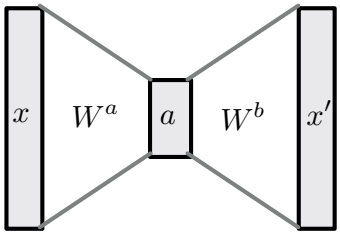
   Solution: You could just multiply W directly into U or V^T and end up with the original model.

   (b) Otto N. Coder thinks there's a whole different and interesting way to approach this problem. Consider a neural network with two layers of weights and tanh activation functions:

$$a = \tanh(W^{aT} x)$$
$$x' = \tanh(W^{bT} a)$$

   where x is a $m \times 1$ vector representing a single user's movie-watching experience. We will assume just binary ratings (+1 means the user liked the movie and -1 that they did not; a value of 0 indicates that the user has not yet rated the movie). The vector a is $k \times 1$ where k is significantly less than m.

   We can make a data-set with n such vectors, one for each user, and then train this network as an "auto-encoder", which takes in an x vector and attempts to recreate it as its output, but which is forced to go through a much smaller representation, as shown below.



   To train this network, we would use ordinary supervised training, but with pairs (x, x) with one x vector for each user in the training data, used as both the input and the desired output of the network.

   The loss function $L(x', x)$ where x is the true output vector and x' is the prediction, would be

$$L(x', x) = \sum_{j=1}^{m} \begin{cases} 0 & \text{if } x_j = 0 \\ (x_j - x'_j)^2 & \text{otherwise} \end{cases}$$

i. What is $L(x', x)$ if this user has never watched any movies? **0**

ii. How much, if any, more loss is incurred, with respect to a particular movie, for predicting +1 when the answer should be -1 than is incurred for predicting -1 when the answer should be +1? **0**

iii. In terms of making good predictions, would it be disastrous, just fine, or only mildly bad if we were to leave out the tanh activation function on the output layer? Explain.

> **Solution:** Only mildly bad. We would get predictions that go outside the bounds of +1 and -1, but they would probably be usable for picking the max. Note that choosing the max is the "right" thing to do here since we want to make recommendations and the thing to recommend should have the maximum prediction value.

(c) After training this network, we could feed in a particular user's x vector and receive an output x'. How could we use the x' value to select the best movie to recommend to that user?

Provide your answer in completely detailed math, code, or English that could be unambiguously converted into math or code.

> **Solution:**
>
> $$m = \text{argmax}_{\{i|x_i=0\}} x'_i$$

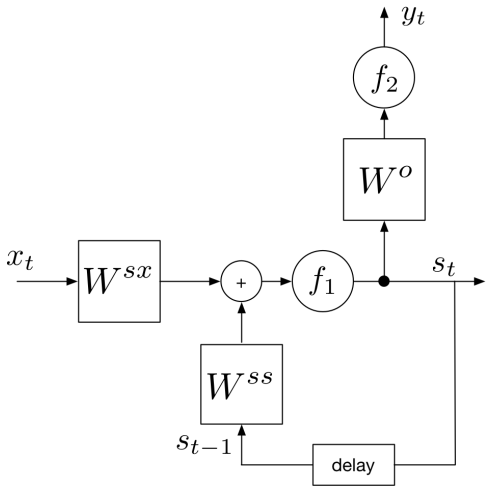
Name: _____

Double trouble

8. (7 points) One of the RNN architectures we studied was

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

where W^{ss} is $m \times m$, W^{sx} is $m \times l$ and W^o is $n \times m$. Assume f_i can be any of our standard activation functions. We omit the offset parameters for simplicity (set them to zero).



(a) Suppose we modify the original architecture as follows:

$$s_t = f_1(W^{ss1} f_3(W^{ss2} s_{t-1}) + W^{sx} x_t)$$

i. Provide values for the original W^{ss} that make the original architecture equivalent to this one, or explain why none exist.

$W^{ss} =$ _____

Solution: This architecture can represent state machines that can't be represented by the original architecture, because the class of state transition functions that can be modeled in the modified architecture is bigger.

ii. Provide values for W^{ss2}, f_3 and W^{ss1} that make this new architecture equivalent to the original, or explain why none exist.

$f_3 =$ **linear**

$W^{ss1} = W^{ss}$

$W^{ss2} = I$

Solution: See above

(b) Now, we'll consider two strategies for making the RNN generate two output symbols for each input symbol. Assume the symbols are drawn from a vocabulary of size n.

Model A: We use a separate softmax output for each symbol, so

$$y_t^1 = \text{softmax}(W^{o1} s_t)$$
$$y_t^2 = \text{softmax}(W^{o2} s_t)$$

where W^{o1} and W^{o2} are $n \times m$.

Model B: We use a single softmax output, but it ranges over n^2 possible pairs of symbols, so

$$y_t^1, y_t^2 = \text{softmax}(W^{o3} s_t)$$

i. What would the dimension of W^{o3} need to be?

Solution: $n^2 \times m$

ii. Which of the following is true:

◯ Models A and B can express exactly the same set of RNN models.

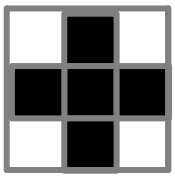
◯ Model A is more expressive than model B.

√ **Model B is more expressive than model A.**

Name: _____

Convoluted network

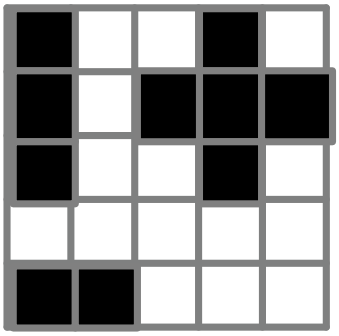
9. (8 points) We will explore how convolutional neural networks operate by designing one. Our objective is to be able to locate the pattern



in an image. Throughout this problem, treat dark squares as having value +1 and light squares as having value -1.

(a) Consider the image that would result from convolving the image below with a filter that is the same as the pattern above. (Use our definition of convolution, in which we slide the filter over the image and compute the dot product.) Assume that the edges are padded with -1 and that use a stride of 1.

Indicate which pixel in the resulting image will have the maximum value by writing the resulting pixel value in the appropriate cell of the image on the right below.



3		1		1
5			9	
3		1		1
1				
3	3	1		

(b) In order to detect this pattern, we would create a network that has

- a convolutional layer with a single filter, corresponding to the desired pattern,
- a max-pooling layer with input size equal to the image size, and finally
- a single ReLU unit.

Provide a value for the offset W_o on the input to the ReLU that, for any image, would guarantee the output of the ReLU is positive if and only if there is a perfect instance of this pattern in the image.

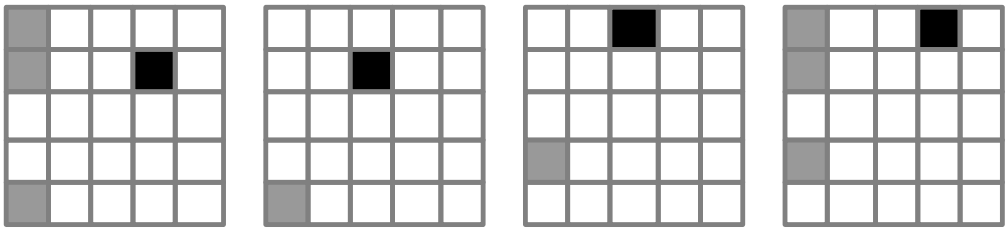
Solution: -8

A perfect score is 9. The next best match would be 8 correct and 1 wrong, which would total to 7. Any value between 7 and 9 would be correct here.

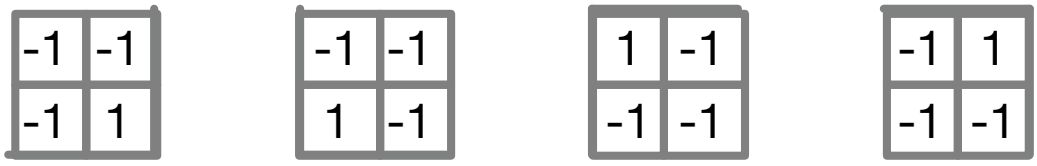
(c) Kanye Volution thinks that instead of having this single convolution layer with a single filter matching the whole desired pattern, it would be better to start with a convolutional layer with four smaller filters, shown below:

The following images are the result of convolving the original image with these 4 simple filters and running through a ReLU. Black squares have value +1, grey squares have value +0.5, and the rest have value 0.

It is slightly unusual to have 2 x 2 filters (usually they have odd dimension). When we apply them, we place the upper-left pixel of the filter on top of the image pixel whose value we are computing.



The next layer of Kanye's network now takes an input of depth 4 and applies a single 2 x 2 x 4 filter. Specify a filter on the output of the simple filters that will generate an image with a high value at the pixel located at the upper left corner of the pattern and lower values elsewhere. Fill weight values (either +1 or -1) into the squares below.



Name: _____

Work space