Final Exam, Fall 2025
ESE 577
SOLUTIONS

Instructor: Jorge Mendez-Mendez

Date: Tuesday December 16th, 2025

Do not tear exam booklet apart!

- This is a closed book exam. Two sheets (8 1/2 in. by 11 in.) of notes, front and back, are permitted. Calculators are not permitted.
- The total exam time is 150 minutes.
- The problems are not necessarily in any order of difficulty.
- Record all your answers in the places provided. If you run out of room for an answer, continue on a blank page and mark it clearly.
- If a question seems vague or under-specified to you, make an assumption, write it down, and solve the problem given your assumption.
- If you absolutely *have* to ask a question, come to the front.
- **Write your name on every piece of paper.**

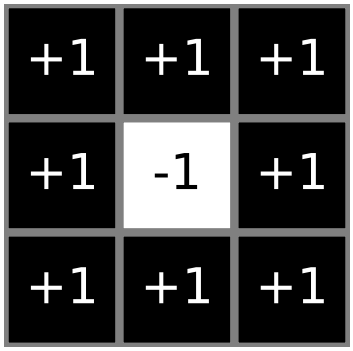
Name: _____ SBU email: _____

Question	Points
1	24
2	26
3	14
4	16
5	20
Total:	100

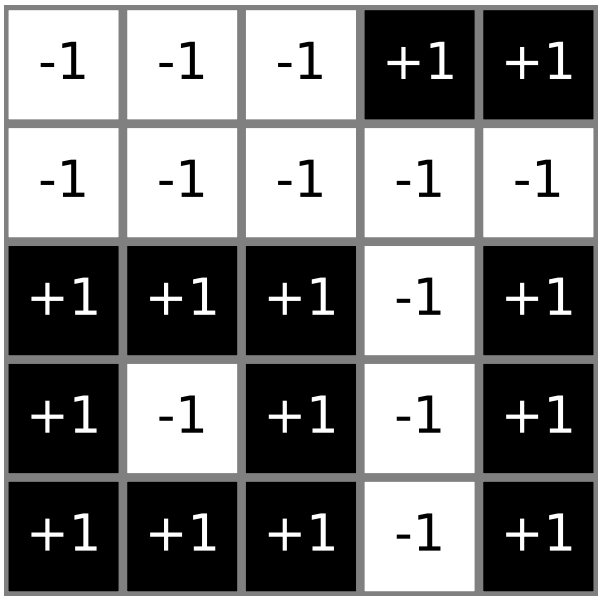
Name: _____

Building a ConvNet

1. (24 points) We will build a convolutional neural network to detect the presence of the following pattern in an image, where black pixels take a value +1 and white pixels take a value −1:



Suppose that we have the following image, where we would like to detect the pattern:



(a) We use a filter that is identical to the pattern and convolve it with our image. We pad the image with one column of −1 left and right, and one row of −1 above and below. Using our definition of convolution, we slide the **center** of the filter across the image and compute the sum of element-wise products. Use a stride of 1.

i. What is the resulting size of the convolved image?

Rows:

Columns:

Channels:

ANSWER: The resulting image is the same size as the original image, 5×5. Using a 3×3 filter and padding with one column/row on either side results in no shape reduction. Using one filter results in no additional channels.

ii. Which pixel in the resulting image has the largest value, and what is the value? Use $[0, 0]$ as the index of the top-left corner.

Row:

Column:

Value:

ANSWER: Row 3, column 1, value 9. The highest value is obtained by aligning the filter with the occurrence of the pattern in the image. Because all entries match, we get a value of 9.

(b) We will aim to detect this pattern using a very simple network with:

- One convolutional layer with the filter that matches our pattern, with a stride of 1
- No activation function
- One max-pooling layer with window size equal to the image size
- One output unit with weights $W = 1$ and offset W_0

i. What is the shape of the output image from the max-pooling layer?

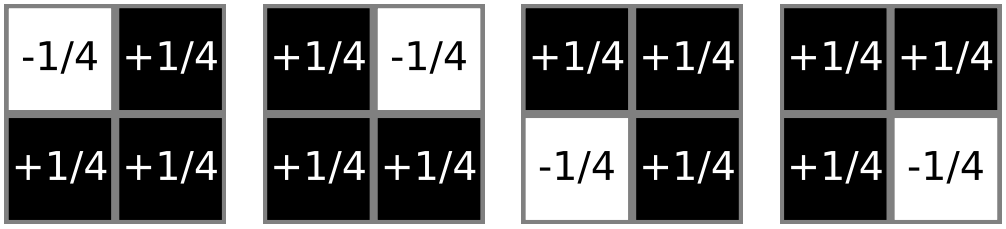
ANSWER: 1. Because the max-pooling is the same size as the image, we obtain a single pixel at the output.

ii. Provide a value for the offset W_0 in sigmoid unit such that, for any image, the output of the sigmoid is > 0.5 if and only if there is an exact match for the pattern in the image.

ANSWER: -8. If there is no match, the largest possible value from our filter at any pixel is 7, and so the output of the max-pool is at most 7, making the sigmoid input -1, which produces $\sigma(-1) < 0.5$. If there is a match, the input to the sigmoid will be 1, giving $\sigma(1) > 0.5$. Note that any value $\in (-9, -7]$ is a correct solution.

(c) Kan Volution thinks that instead of having this single convolution layer with one filter for the whole pattern, it would be better to start with a convolution layer with four small filters, shown below, where black pixels have value $+1/4$ and white pixels have value $-1/4$:

Name: _____



(a) (b) (c) (d)

Because our filters are even-sized, we slide the **upper-left pixel** (instead of the usual center) across the image. We use -1 padding for one column on the right and one row on the bottom of the input image, and use a stride of 1.

i. What is the resulting size of the convolved image?

Rows:

Columns:

Channels:

ANSWER: The resulting image is the same size as the original image, 5×5, with 4 channels. Using a 2×2 filter and padding with one column/row on one side results in no shape reduction. Using four filter results in four output channels.

ii. Write down the numerical values of applying each of the four filters above to the original image and passing the outputs through a ReLU activation. Your answer should be four grids of appropriate shape, clearly indicating which filter each grid corresponds to. For your convenience, the original image is replicated below.

-1	-1	-1	+1	+1
-1	-1	-1	-1	-1
+1	+1	+1	-1	+1
+1	-1	+1	-1	+1
+1	+1	+1	-1	+1

Name: _____

ANSWER:

0	0	0	0	0
0.5	0.5	0	0	0
0	0	0	0.5	0
0	+1	0	0.5	0
0	0	0	0	0

(a)

0	0	0	0	0
0.5	0.5	0	0	0
0	0	0.5	0	0.5
+1	0	0.5	0	0.5
0	0	0	0	0

(b)

0	0	0	0.5	0
0	0	0	0	0
0	+1	0	0.5	0
0	0	0	0.5	0
0.5	0.5	0	0	0

(c)

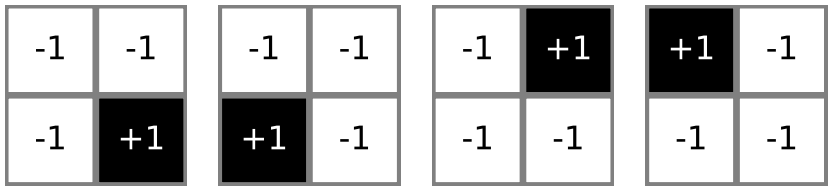
0	0	0	0.5	0
0	0	0	0	0
+1	0	0.5	0	0.5
0	0	0.5	0	0.5
0.5	0.5	0	0	0

(d)

iii. The next layer of Kan's network takes in the 3D tensor and applies one $2 \times 2 \times 4$ filter, with the same top-left corner alignment of the filter and asymmetric padding (right-bottom) as before; this time, we use a padding value of 0. Specify a filter for this layer that will generate a final image with the highest value at the pixel located in the upper-left corner of the original 3×3 pattern we are trying to detect. Each weight should be either $+1$ or -1.Your answer should be four 2×2 grids, clearly indicating which channel each grid corresponds to.

Name: _____

ANSWER:



Name: _____

Transformer Sequence Averager

2. (26 points) We will construct a transformer that computes the average over the sequence of values seen so far. The model receives a sequence of tokens. Each token may be a **content** token (containing a number to be included in the average) or a **trigger** token (requesting to compute the average over all previous content tokens).

   Each input token is encoded with three features: $x^{(i)} = [\texttt{is_trigger}, \texttt{is_content}, n]^\top$, where `is_trigger` and `is_content` are binary indicators, and n is 0 for trigger tokens and the actual number for content tokens. For example:

   - $x^{(i)} = [0, 1, 5]^\top$ is a content token with $n = 5$
   - $x^{(i)} = [1, 0, 0]^\top$ is a trigger token

   Given a sequence of input tokens $x^{(1)}, \ldots, x^{(N)}$, the transformer should produce an output sequence of the same length $y^{(1)}, \ldots, y^{(N)}$. At positions where the input is a trigger token, the output should be the average over previous content tokens. At positions where the input is a content token, the output should be 0. Assume that all sequences begin with a trigger token, and that the average over an empty sequence should be 0. For example:

   $$\underset{\substack{\uparrow\\ \text{trigger}}}{\begin{bmatrix}1\\0\\0\end{bmatrix}} \quad \begin{bmatrix}0\\1\\5\end{bmatrix} \quad \begin{bmatrix}0\\1\\3\end{bmatrix} \quad \begin{bmatrix}1\\0\\0\end{bmatrix} \quad \begin{bmatrix}0\\1\\2\end{bmatrix} \quad \begin{bmatrix}1\\0\\0\end{bmatrix} \rightarrow \underset{\substack{\uparrow\\ \text{avg([])}}}{0} \quad 0 \quad 0 \quad \underset{\substack{\uparrow\\ \text{avg([5,3])}}}{4} \quad 0 \quad \underset{\substack{\uparrow\\ \text{avg([5,3,2])}}}{3.33}$$

   Our transformer learns projection matrices $W_Q \in \mathbb{R}^{d_k,3}, W_K \in \mathbb{R}^{d_k,3}, W_V \in \mathbb{R}^{d_v,3}$ that map raw tokens $x^{(i)}$ to queries $q^{(i)} = W_Q \cdot x^{(i)}$, keys $k^{(i)} = W_K \cdot x^{(i)}$, and values $v^{(i)} = W_V \cdot x^{(i)}$, respectively. For each query $q^{(i)}$, we will compute unnormalized self-attention scores $a^{(i)} = \text{softmax}({q^{(i)}}^\top k^{(1)}, \ldots, {q^{(i)}}^\top k^{(N)})$ for all N keys in the sequence. The i-th output will be computed as the sum of values weighted by the self-attention scores: $y^{(i)} = a_1^{(i)} v^{(1)} + \cdots + a_N^{(i)} v^{(N)}$.

   (a) We will use self-attention as the key mechanism that allows each output to be computed based on the appropriate tokens. It will help to think through all 4 subparts of this question (i–iv) before answering.

   i. Should our transformer use causal masking? Briefly justify your answer (in one sentence).

   Yes No
   Justification:

   ANSWER: Yes. For each trigger, we want to compute the average over previous content tokens, which requires attending only over previous tokens.

   ii. For the example sequence, write the numerical value of the self-attention matrix that would allow the transformer to compute each output token by using the relevant input tokens. Your answer should be a 6×6 matrix, where each row corresponds to a query and must sum to 1. (Hint: think about which input tokens could allow us to compute 0 outputs).

ANSWER:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0.33 & 0.33 & 0 & 0.33 & 0 \end{bmatrix}$$

Trigger tokens attend equally to all previous content tokens, and content tokens attend equally to all previous trigger tokens.

iii. Write the numerical value of the input to the softmax that computes the desired self-attention matrix. This corresponds to the similarity scores ${q^{(i)}}^\top k^{(j)}$, (optionally) multiplied by the causal mask. (Hint: for any unmasked entry that must result in 0 attention, assume that $\frac{1}{c+e^{100}} \approx 0$ for any small constant c.)

Name: _____

ANSWER:

$$\begin{bmatrix} 0 & -\infty & -\infty & -\infty & -\infty & -\infty \\ 100 & 0 & -\infty & -\infty & -\infty & -\infty \\ 100 & 0 & 0 & -\infty & -\infty & -\infty \\ 0 & 100 & 100 & 0 & -\infty & -\infty \\ 100 & 0 & 0 & 100 & 0 & -\infty \\ 0 & 100 & 100 & 0 & 100 & 0 \end{bmatrix}$$

The $-\infty$s represent our causal masking. The fact in the hint implies that we can use similarity scores of 100 for entries that we want to attend to and 0 for entries that we do not want to attend to. For example, consider the softmax in the fourth row:

$$\begin{aligned} \text{softmax}([0, 100, 100, 0]) &= \left[\frac{e^0}{2e^0 + 2e^{100}}, \frac{e^{100}}{2e^0 + 2e^{100}}, \frac{e^{100}}{2e^0 + 2e^{100}}, \frac{e^0}{2e^0 + 2e^{100}}\right] \\ &= \left[\frac{1}{2 + 2e^{100}}, \frac{e^{100}}{2 + 2e^{100}}, \frac{e^{100}}{2 + 2e^{100}}, \frac{1}{2 + 2e^{100}}\right] \\ &\approx \left[0, \frac{e^{100}}{2 + 2e^{100}}, \frac{e^{100}}{2 + 2e^{100}}, 0\right] \\ &\approx [0, 0.5, 0.5, 0] \end{aligned}$$

iv. Using 2-dimensional keys and queries (i.e., $d_k = 2$), write numerical values of the projection matrices W_K and W_Q that result in the desired attention matrix for any sequence of trigger and content tokens. Your answers should be two 2×3 matrices.

$W_K = \begin{bmatrix} & \\ & \end{bmatrix}$ $W_Q = \begin{bmatrix} & \\ & \end{bmatrix}$

ANSWER:

$W_K = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \end{bmatrix}$ $W_K = \begin{bmatrix} 0 & 10 & 0 \\ 10 & 0 & 0 \end{bmatrix}$

We want the similarity score to be 0 between content-queries and content-keys, and 100 between content-queries and trigger-keys. We also want the similarity between trigger-queries and trigger-keys to be 0, and the similarity between trigger-queries and content-keys to be 100. We can achieve this by making query vectors orthogonal to key vectors for the same token, and parallel to key vectors for the other tokens. We achieve this by making our keys $k = [10, 0]$ or $k = [0, 10]$, and our queries $q = [0, 10]$ or $q = [10, 0]$. The projection matrices above achieve this result.

(b) Given our attention matrix, we need to define our values to use for computing the final output of our model.

i. What should be the dimensionality of our values, d_v?

$d_v =$

ANSWER: $d_v = 1$. The outputs of our transformer are weighted sum of values, and the problem requires scalar outputs, so $d_v = 1$.

ii. Define a weight matrix W_v that produces values $v^{(i)}$, which, when weighted by our attention matrix, produce the desired outputs. Your answer should be a matrix of shape $d_v \times 3$

$W_v = \begin{bmatrix} & & \end{bmatrix}$

ANSWER: $W_v = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$. This produces value n for content tokens and 0 for trigger tokens. Overall, our transformer will equally weight all previous contents n for trigger tokens, and equally weight all previous instances of 0 for content tokens.

Name: _____

Linear Q-Network

3. (14 points) We will use the deep Q-learning algorithm to train a linear Q-function. We consider a problem with two states **A** and **B**. Our states are represented by a scalar feature x, such that $x_A = 1$ for state **A** and $x_B = 2$ for state **B**. The agent can execute two actions, a_1 or a_2. We will estimate our Q-values with a linear function with parameters $\theta = [\theta_1, \theta_2]$: $Q_\theta(s, a_i) = \theta_i \cdot x$.

   Throughout this problem, we will use $\gamma = 0.9$, a learning rate $\alpha = 0.05$, and initial parameters $\theta = [1, 1]$. In the first step, the agent executes action a_1, observes a transition from **A** to **B**, and obtains a reward $r = 0$.

   (a) First, compute the error before any deep Q-learning updates.

      i. What is the predicted Q-value, $Q(s, a_1)$?

      $Q(\mathbf{A}, a_1) =$

      ANSWER: $Q(s, a_1) = 1 \cdot 1 = 1$

      ii. What is the target value, $y = r + \gamma \max_i Q(\mathbf{B}, a_i)$?

      $y =$

      ANSWER: $y = 0 + 0.9 \cdot \max[1, 1] \cdot 2 = 1.8$

      iii. What is the squared error $L_{t=1} = (Q(\mathbf{A}, a_1) - y)^2$?

      $L_{t=1} =$

      ANSWER: $L_{t=1} = (1 - 1.8)^2 = 0.64$

   (b) Compute the updated value of θ after executing one step of deep Q-learning. Recall that the update rule of deep Q-learning is $\theta_{\text{new}} = \theta_{\text{old}} - \alpha \cdot \nabla_\theta (Q_\theta(s, a) - y)^2$.

   $\theta_{\text{new}} =$

   ANSWER:

$$\begin{aligned}
\nabla_\theta (Q_\theta(\mathbf{A}, a_1) - y)^2 &= 2 \cdot (Q_\theta(\mathbf{A}, a_1) - y) \cdot [x_A, 0] \\
&= 2(1 - 1.8) * [1, 0] \\
&= 2 \cdot [-0.8, 0] \\
\theta_{\text{new}} &= \theta_{\text{old}} - \alpha \cdot 2 \cdot [-0.8, 0] \\
&= [1, 1] - 0.1 \cdot [-0.8, 0] \\
&= [1.08, 1]
\end{aligned}$$

   (c) Now, compute the error after the first deep Q-learning update (i.e., using the updated θ_{new}).

      i. What is the predicted Q-value, $Q(s, a_1)$?

Name: _____

$Q(\mathbf{A}, a_1) =$

ANSWER: $Q(s, a_1) = 1.08 \cdot 1 = 1.08$

ii. What is the target value, $y = r + \gamma \max_i Q(\mathbf{B}, a_i)$?

$y =$

ANSWER: $y = 0 + 0.9 \cdot \max[1.08, 1] \cdot 2 = 1.944$

iii. What is the squared error $L_{t=2} = (Q(\mathbf{A}, a_1) - y)^2$?

$L_{t=2} =$

ANSWER: $L_{t=2} = (1.08 - 1.944)^2 \approx 0.746$

(d) Compare the losses before and after the deep Q-learning udpate, $L_{t=1}$ and $L_{t=2}$. Did the loss decrease or increase? If it increased, briefly explain (in 1–2 sentences) why and what technique we could use to fix this issue.

ANSWER: The loss increased because the target value moved faster than the current Q-value. Using a target Q-network is the standard technique to get around this challenge.

Name: _____

Deploying LeafNet

4. (16 points) Dr. Flora Green is a computational botanist working on LeafNet, a deep learning model designed to classify drone images of crops into two categories: "Healthy" and "Infected". She has collected a dataset of 5,000 images, where 4,500 are "Healthy" and 500 are "Infected." She asks for your help to finalize the deployment pipeline.

   (a) Dr. Green splits her data into a training set (4,000 images) and a validation set (1,000 images). She trains her first model and enthusiastically reports a validation accuracy of 90%. She claims that her model is performing incredibly well and is ready for deployment. Do you agree with her assessment? Why or why not?

   ANSWER: Dr. Green is overestimating her model's performance. For this problem, a commonsense baseline would always predict the "Healthy" (majority) class and achieve 90% accuracy. Her model is merely matching the performance of a trivial model.

   (b) Dr. Green decides to train the network using Stochastic Gradient Descent (SGD). In her first trials, she trained the model using a batch size of 1, but found that the training was proceeding too slowly. She learned of mini-batch gradient descent, and decided to try with a batch size of 2,000. Unfortunately, the training process was advancing even more slowly. What could be the reason for training to slow down? How could Dr. Green address this issue?

   ANSWER: It is possible that the batch size is too large for her GPU. Instead of parallelizing the gradient computation across $2,000$ data points, Dr. Green may be spending a lot of time moving smaller batches to the GPU, slowing down training. Dr. Green should choose a smaller batch size that she is sure can fit in her GPU.

   (c) Dr. Green suspects that her model is underfitting. She decides to increase the model size to learn how big of a model she can train before it starts overfitting. She increases the depth of her Convolutional Neural Network (CNN) from 10 layers to 100 layers. Surprisingly, both her training loss and validation loss increase. Explain why adding more layers caused the training error to increase (hint: it's not overfitting), and propose an architectural change to fix this issue.

   ANSWER: Training very deep networks with vanilla CNNs results in *vanishing gradients*, where the gradient of earlier layers collapses to 0. Adding skip connections or residual connections enables training very deep networks without vanishing gradients.

   (d) Dr. Green adds batch normalization layers to her network to help with the training. When she deploys the model to the drone to make real-time predictions on single images, the model outputs terrible predictions. She discovers that she was computing the mean and variance of the *current*

input during deployment. Why does this fail when processing a single image? What statistics should the batch normalization layer use during deployment?

ANSWER: The variance of a single image is 0, and dividing by a small value (even if being careful to add a small constant ϵ to avoid dividing by 0) would ruin the learned features. Even for batches greater then 1, a small test batch may not match the training distribution. During deployment, batch normalization should use the *running mean and variance* accumulated during training.

Name: _____

Regression / classification

5. (20 points) We're given a dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$ where $x^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}$. Let X be the data $n \times d$ data matrix and Y be the $n \times 1$ vector of labels. The analytical regression formula gives us:

$$\theta_{\text{ana}} = (X^\top X)^{-1} X^\top Y \ .$$

Using ridge regression, we can compute

$$\theta_{\text{ridge}} = (X^\top X + \lambda I)^{-1} X^\top Y \qquad (\lambda > 0) \ .$$

We will try using these methods to initialize a "neural network" with a single linear unit and no offset:

$$h(x; W) = W^\top x \ .$$

Assume that $X^\top X$ is invertible and that neither θ_{ana} nor θ_{ridge} are all-zeros.

(a) If we initialize our unit with $W = \theta_{\text{ridge}}$ and did gradient descent (summing the error over all data points) with squared loss, no regularization, and a fixed small step size, which of the following is most likely to happen? Choose one.

- ◯ W would change substantially at the beginning but then converge back to θ_{ridge}.
- ◯ W would not change.
- ◯ W would make small oscillations around θ_{ridge}.
- ◯ W would converge to a different vector with smaller magnitude.
- ◯ W would converge to a different vector with larger magnitude.

ANSWER: W would converge to a different vector with larger magnitude.

(b) Explain why.

ANSWER: GD will converge to the optimal solution without regularization, which is θ_{ana}. $\|\theta_{\text{ana}}\|_2 > \|\theta_{\text{ridge}}\|$ < because it is unregularized.

(c) If we initialized our unit with θ_{ridge} and did stochastic gradient descent (one data point at a time) with squared loss, regularization with the same λ, and a fixed small step size, many different things could happen. For each outcome below, explain briefly the circumstances in which it might occur, or explain why it is unlikely.

i. The weights would not change.

ANSWER: This is unlikely to happen because it would require the gradient for *each data point* to be 0. This could only happen if all the x's are scaled versions of θ_{ridge}, which in turn means that all x's are scaled versions of each other. (To see why, observe that the gradient for one data point is $\nabla_\theta = 2(\theta^\top x - y)x + 2\lambda\theta = 2 \cdot err \cdot x + 2\lambda\theta$.)

ii. The weights would make small oscillations around the initial weights.

ANSWER: Since all data points will have non-zero gradient, and the learning rate is small, then on expectation the steps should be small motions around the optimum.

iii. The weights would converge to a different value.

ANSWER: This cannot happen because the ridge objective has a single optimum, and SGD with a small learning rate converges to this optimum.

(d) Mike has trained a network to solve a very similar problem from this particular domain before, and the solution was W^*. Define an objective function $J(W)$ that depends on the data that we could minimize in order to obtain a good estimate for Rory's next problem, even with very little data, assuming that the solution should be similar to W^*.

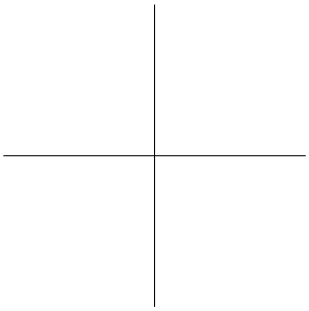
$J(W) =$

ANSWER: $J(W) = \frac{1}{n}\sum_{i=1}^{n}(W^\top x^{(i)} - y^{(i)})^2 + \lambda\|W - W^*\|_2^2$. This ensures that the new parameter W is not too far from W^*.

(e) Cheng thinks that they can get a better hypothesis by using knowledge about neural networks, and considers the hypothesis class

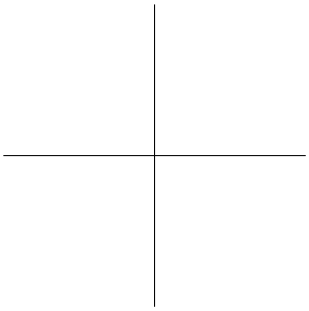
$$g = w_a \cdot \text{ReLU}(w_b \cdot x + w_c) + w_d \ .$$

Assume that the inputs x are 1-dimensional and recall that $\text{ReLU}(z) = \max(0, z)$.
Provide a dataset with 3 points for which Cheng's hypothesis class can reach a lower training MSE than the original analytical solution or argue that one does not exist.



ANSWER: $(0,0), (1,0), (2,1)$.

(f) Provide a dataset with 3 points for which the original analytical hypothesis class can reach a **substantially** lower training MSE than Cheng's hypothesis class or argue that one does not exist.



ANSWER: Does not exist. Given any three points that are co-linear, you can always set values for the weights that shift/scale/mirror the ReLU to obtain the minimum-cost line.