

You want to train a neural network to drive a car. Your training data consists of grayscale 64×64 pixel images. The training labels include the human driver's steering wheel angle in degrees and the human driver's speed in miles per hour. Your neural network consists of an input layer with $64 \times 64 = 4,096$ units, a hidden layer with 2,048 units, and an output layer with 2 units (one for steering angle, one for speed). You use the ReLU activation function for the hidden units and no activation function for the outputs (or inputs).

- (1) [2 pts] Calculate the number of parameters (weights) in this network. You can leave your answer as an expression. Be sure to account for the bias terms.

$$4097 \times 2048 + 2049 \times 2$$

- (2) [3 pts] You train your network with the cost function $J = \frac{1}{2}|\mathbf{y} - \mathbf{z}|^2$. Use the following notation.

- \mathbf{x} is a training image (input) vector with a 1 component appended to the end, \mathbf{y} is a training label (input) vector, and \mathbf{z} is the output vector. All vectors are column vectors.
- $r(\gamma) = \max\{0, \gamma\}$ is the ReLU activation function, $r'(\gamma)$ is its derivative (1 if $\gamma > 0$, 0 otherwise), and $r(\mathbf{v})$ is $r(\cdot)$ applied component-wise to a vector.
- \mathbf{g} is the vector of hidden unit values before the ReLU activation functions are applied, and $\mathbf{h} = r(\mathbf{g})$ is the vector of hidden unit values after they are applied (but we append a 1 component to the end of \mathbf{h}).
- V is the weight matrix mapping the input layer to the hidden layer; $\mathbf{g} = V\mathbf{x}$.
- W is the weight matrix mapping the hidden layer to the output layer; $\mathbf{z} = W\mathbf{h}$.

Derive $\partial J / \partial W_{ij}$.

$$\begin{aligned} \frac{\partial J}{\partial W_{ij}} &= (\mathbf{z} - \mathbf{y})^\top \frac{\partial \mathbf{z}}{\partial W_{ij}} \\ &= (\mathbf{z}_i - \mathbf{y}_i) \mathbf{h}_j \end{aligned}$$

- (3) [1 pt] Write $\partial J / \partial W$ as an outer product of two vectors. $\partial J / \partial W$ is a matrix with the same dimensions as W ; it's just like a gradient, except that W and $\partial J / \partial W$ are matrices rather than vectors.

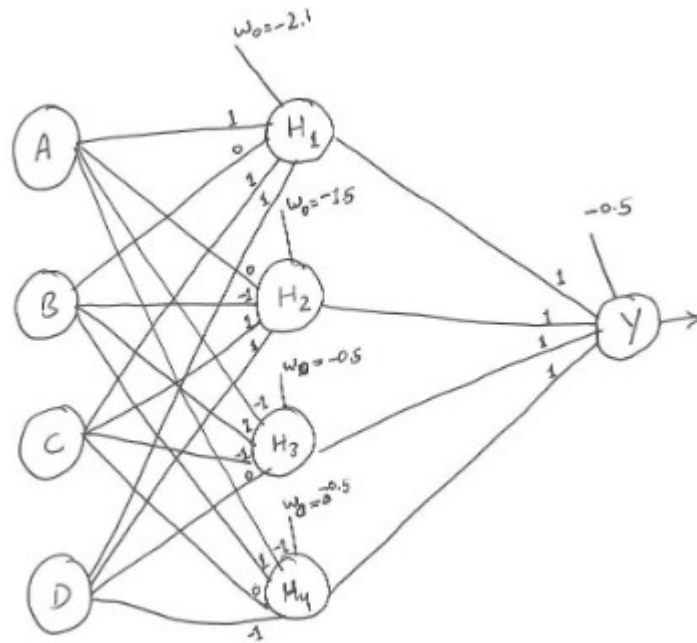
$$\frac{\partial J}{\partial W} = (\mathbf{z} - \mathbf{y}) \mathbf{h}^\top$$

- (4) [4 pts] Derive $\partial J / \partial V_{ij}$.

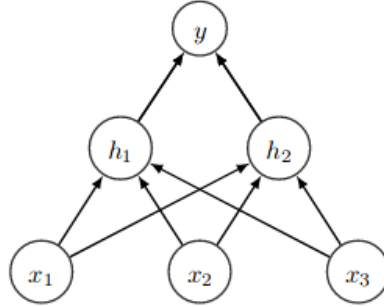
$$\begin{aligned} \frac{\partial J}{\partial V_{ij}} &= (\mathbf{z} - \mathbf{y})^\top \frac{\partial \mathbf{z}}{\partial V_{ij}} \\ &= (\mathbf{z} - \mathbf{y})^\top W \frac{\partial \mathbf{h}}{\partial V_{ij}} \\ &= (\mathbf{z} - \mathbf{y})^\top W [0, \dots, r'(\mathbf{g}_i) \mathbf{x}_j, \dots, 0]^\top \\ &= ((\mathbf{z} - \mathbf{y})^\top W)_i r'(\mathbf{g}_i) \mathbf{x}_j. \end{aligned}$$

Create a neural network with only one hidden layer (of any number of units) that implements $(A \vee \neg B) \oplus (\neg C \vee \neg D)$. Draw your network, and show all weights of each unit.

★ **SOLUTION:** Note that XOR operation can be written in terms of AND and OR operations: $p \oplus q = (p \wedge \neg q) \vee (\neg p \wedge q)$. Given this, we can rewrite the formula as $(A \wedge C \wedge D) \vee (\neg B \wedge C \wedge D) \vee (\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge B \wedge \neg D)$. This formula can be represented by a neural network with one hidden layer and four nodes in the hidden layer (one unit for each parenthesis). An example is shown in Figure 2.



The following graph shows the structure of a simple neural network with a single hidden layer. The input layer consists of three dimensions $x = (x_1, x_2, x_3)$. The hidden layer includes two units $h = (h_1, h_2)$. The output layer includes one unit y . We ignore bias terms for simplicity.



We use linear rectified units $\sigma(z) = \max(0, z)$ as activation function for the hidden and the output layer. Moreover, denote by $l(y, t) = \frac{1}{2}(y - t)^2$ the loss function. Here t is the target value for the output unit y .

Denote by W and V weight matrices connecting input and hidden layer, and hidden layer and output respectively. They are initialized as follows:

$$W = \begin{bmatrix} 1 & 0 & 1 \\ 1 & -1 & 0 \end{bmatrix} \text{ and } V = \begin{bmatrix} 0 & 1 \end{bmatrix} \text{ and } x = [1, 2, 1] \text{ and } t = 1.$$

Also assume that we have at least one sample (x, t) given by the values above.

1. Write out *symbolically* (no need to plug in the specific values of W and V just yet) the mapping $x \rightarrow y$ using σ, W, V .

Solution:

$$y = \sigma(V\sigma(Wx))$$

2. Assume that the current input is $x = (1, 2, 1)$. The target value is $t = 1$. Compute the *numerical* output value y , clearly show all intermediate steps. You can reuse the results of the previous question. Using matrix form is recommended but not mandatory.

Solution:

$$h = \sigma(Wx) = (2, 0)^T$$

$$y = \sigma(Vh) = 0$$

3. Compute the gradient of the loss function with respect to the weights. In particular, compute the following terms *symbolically*:

- The gradient relative to V , i.e. $\frac{\partial l}{\partial V}$
- The gradient relative to W , i.e. $\frac{\partial l}{\partial W}$
- Compute the values *numerically* for the choices of W, V, x, y given above.

Solution:

The gradients are computed with the chain rule as follows:

$$\begin{aligned}\frac{\partial l}{\partial V} &= \left(\frac{\partial y}{\partial V} \right)^\top \frac{\partial l}{\partial y} \\ \frac{\partial l}{\partial W} &= \left(\frac{\partial h}{\partial W} \right)^\top \left(\frac{\partial y}{\partial h} \right)^\top \frac{\partial l}{\partial y}\end{aligned}$$

Plugging in numerical values yields

$$\frac{\partial l}{\partial V} = \left(\frac{\partial V h}{\partial V} \right)^\top \left(\frac{\partial y}{\partial V h} \right)^\top \frac{\partial l}{\partial y} = h^\top g(y - t) = [-2g, 0]$$

where $0 \leq g \leq 1$ is the subgradient of ReLU

$$\frac{\partial l}{\partial W} = \left(\frac{\partial W x}{\partial W} \right)^\top \left(\frac{\partial h}{\partial W x} \right)^\top \left(\frac{\partial y}{\partial h} \right)^\top \frac{\partial l}{\partial y} = M V^\top (y - t) x^\top = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \text{ where } M = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

1. (28 points) Give short answers for each of the following questions – 2 points each

- a) For a fully-connected deep network with one hidden layer, increasing the number of hidden units should have what effect on bias and variance? Explain briefly.(2 points)

Adding more hidden units should decrease bias and increase variance. In general, more complicated models will result in lower bias but larger variance, and adding more hidden units certainly makes the model more complex. See the Lecture 2 slides on the “rule of thumb” for bias and variance.

- b) What’s the risk with tuning hyperparameters using a test dataset? (2 points)

Tuning model hyperparameters to a test set means that the hyperparameters may overfit to that test set. If the same test set is used to estimate performance, it will produce an overestimate. Using a separate validation set for tuning and test set for measuring performance provides unbiased, realistic measurement of performance.

- c) What kinds of dataset are difficult for a linear classifier to correctly classify? (2 points)

Linear classifiers delimit linearly separable classes, which roughly corresponds to a single exemplifier. In input space, each class must be a convex set. The simplest counter-example is a non-convex set which is a bimodal class, e.g. which contains images of two very different breeds of cat.

[2 pts] A neural network with multiple hidden layers and sigmoid nodes can form non-linear decision boundaries.

☒ True ☐ False

[2 pts] All neural networks compute non-convex functions of their parameters.

☐ True ☒ False

[2 pts] For logistic regression, with parameters optimized using a stochastic gradient method, setting parameters to 0 is an acceptable initialization.

☒ True ☐ False

[2 pts] For arbitrary neural networks, with weights optimized using a stochastic gradient method, setting weights to 0 is an acceptable initialization.

☐ True ☒ False

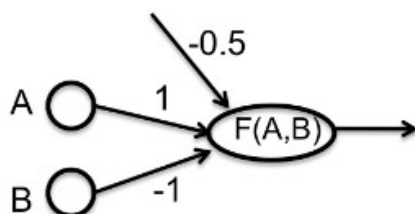
[3 points] Can you represent the following boolean function with a single logistic threshold unit (i.e., a single unit from a neural network)? If yes, show the weights. If not, explain why not in 1-2 sentences.

A	B	$f(A,B)$
1	1	0
0	0	0
1	0	1
0	1	0

Solution:

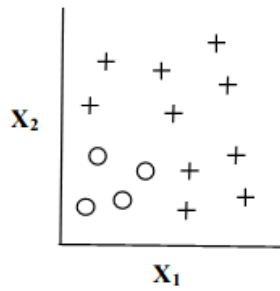
Yes, you can represent this function with a single logistic threshold unit, since it is linearly separable. Here is one example.

$$F(A, B) = 1\{A - B - 0.5 > 0\}$$

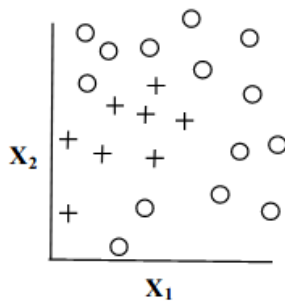


(1)

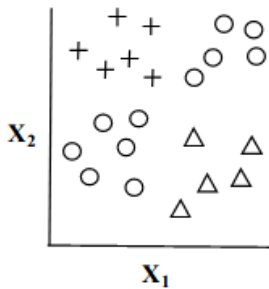
Recall that the output of a perceptron is 0 or 1. For each of the three following data sets, select the perceptron network with the fewest nodes that will separate the classes, and write the corresponding letter in the box. *You can use the same network more than once.*



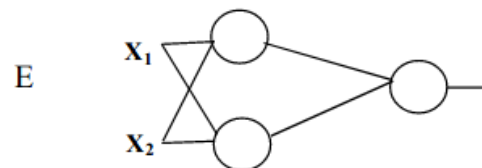
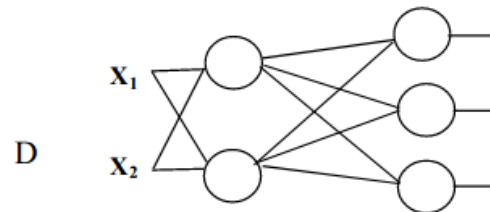
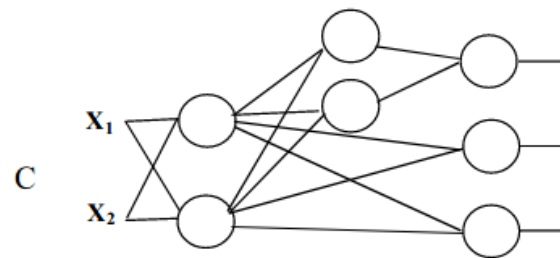
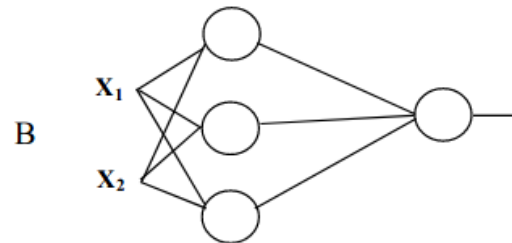
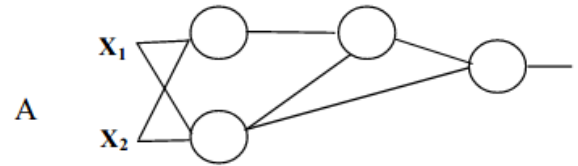
E



B



C



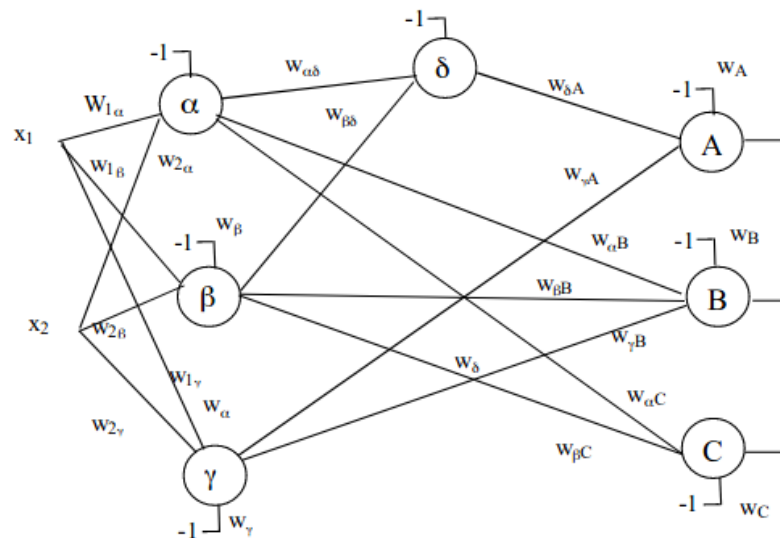
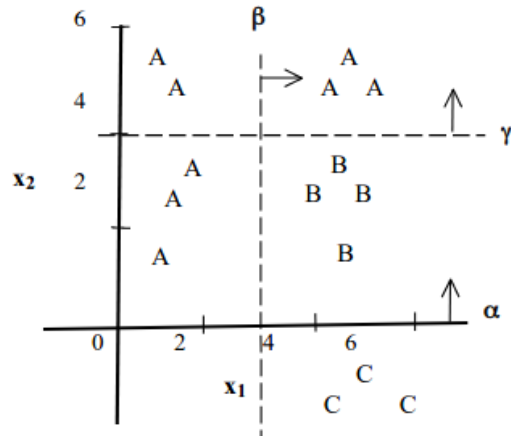
Fill in the missing weights for each of the nodes in the perceptron network.
Make the following assumptions:

Perceptrons output 0 or 1

A, B, C are classes

The lines labeled α (same as abscissa, the x_1 axis), β , γ represent decision boundaries

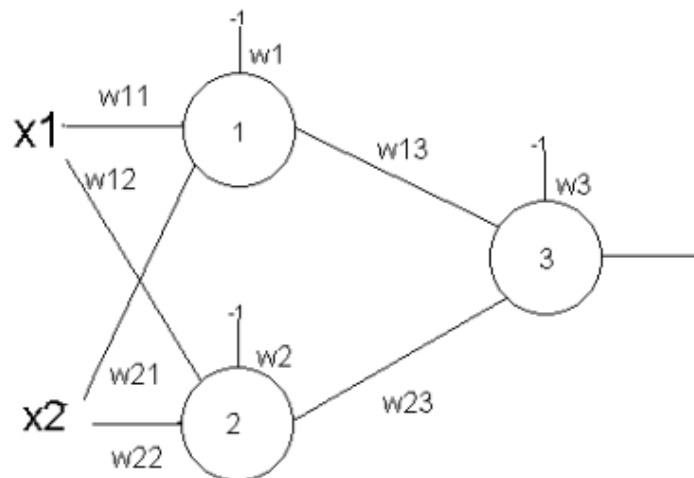
The directions of the arrows shown on the graph represent the side of each boundary that causes a perceptron



α	β	γ	δ	A	B	C
$w_{1\alpha}$	$w_{1\beta}$	$w_{1\gamma}$	$w_{\alpha\delta}$	$w_{\delta A}$	$w_{\alpha B}$	$w_{\alpha C}$
$w_{2\alpha}$	$w_{2\beta}$	$w_{2\gamma}$	$w_{\beta\delta}$	$w_{\gamma A}$	$w_{\beta B}$	$w_{\beta C}$
w_{α}	w_{β}	w_{γ}	w_{δ}	w_A	w_B	w_C

α	β	γ	δ	A	B	C
$w_{1\alpha}$	$w_{1\beta}$	$w_{1\gamma}$	$w_{\alpha\delta}$	$w_{\delta A}$	$w_{\alpha B}$	$w_{\alpha C}$
$w_{2\alpha}$	$w_{2\beta}$	$w_{2\gamma}$	$w_{\beta\delta}$	$w_{\gamma A}$	$w_{\beta B}$	$w_{\beta C}$
w_{α}	w_{β}	w_{γ}	w_{δ}	w_A	w_B	w_C

The following sigmoid network has 3 units labeled 1, 2, and 3. All units are **negative sigmoid** units, meaning that their output is computed using the equation $n(z) = -\frac{1}{1+e^{-z}}$, which differs from the standard sigmoid by a minus sign. The equation for the derivative of $n(z)$ is: $\frac{dn(z)}{dz} = n(z)(1+n(z))$. Additionally, this network uses a **non-standard error** function $E = \frac{1}{2}(2y^* - 2y)^2$.



Part B1: Forward propagation (4 Points)

Using the initial weights provided below, and the input vector $[x1, x2] = [2, 0.5]$, compute the output at each neuron after forward propagation. *Use the negative sigmoid values given in the table*

Weight	w1	w11	w12	w13	w2	w21	w22	w23	w3
Value	0.5	1	1	0.5	0.5	1	1	0.25	0.5

y1 (output at neuron 1) $n(w11x1 + w21x2 - w1) = n((1)(2) + (1)(.5) - .5) = n(2) = -.88$

y2 (output at neuron 2) $n(w12x1 + w22x2 - w2) = n((1)(2) + (1)(.5) - .5) = n(2) = -.88$

y3 (output at neuron 3) $n(w13y1 + w23y2 - w3) = n((.5)(-.88) + (.25)(-.88) - .5) = n(-1.16) = -.24$

Part B2: Backward propagation (6 Points)

Using a **learning rate of 1**, and a **desired output of 0**, backpropagate the network by computing the δ values for nodes 2 and 3, and write the new values for the selected weights in the table below. Assume the initial values for the weights are as specified in Part B1, and assume the following values for the neuron outputs:

output at node 1, $y_1 = -1.0$

output at node 2, $y_2 = -1.0$

output at node 3, $y_3 = -0.2$

Express δ_2 and δ_3 in terms of derivative-free expressions.

$$\delta_3 \quad 4(y_3 - y^*)(y_3)(1 + y_3) = 4(-0.2)(-0.2)(1 - 0.2) = 0.13$$

$$\delta_2 \quad y_2(1 + y_2)(w_{23}\delta_3) = 0$$

Express the weights in terms of δ s and numbers.

Weight	w_{22}	w_{33}
Value	$1 - 0.5\delta_2 = 1$	$0.5 + \delta_3 = 0.63$

Negative Sigmoid Values

z	n(z)	z	n(z)
-3.00	-0.05	0.25	-0.56
-2.75	-0.06	0.50	-0.62
-2.50	-0.08	0.75	-0.68
-2.25	-0.10	1.00	-0.73
-2.00	-0.12	1.25	-0.78
-1.75	-0.15	1.50	-0.82
-1.50	-0.18	1.75	-0.85
-1.25	-0.22	2.00	-0.88
-1.00	-0.27	2.25	-0.90
-0.75	-0.32	2.50	-0.92
-0.50	-0.38	2.75	-0.94
-0.25	-0.44	3.00	-0.95
0.00	-0.50	3.25	-0.96

An efficient method of implementing gradient descent for neural networks.

The formulas below assume a regular sigmoid unit, $s(z) = \frac{1}{1+e^{-z}}$, and an error function of $E = \frac{1}{2} \sum (y^* - y)^2$.

Descent Rule $w_{i \rightarrow j} = w_{i \rightarrow j} - r \frac{dE}{dw_{i \rightarrow j}} = w_{i \rightarrow j} - r \delta_j y_i$

Backprop rule $\delta_j = \frac{ds(z_j)}{dz_j} \sum_k \delta_k w_{j \rightarrow k}$

1. Initialize weights to small random values
2. Choose a random sample input feature vector
3. Compute total input (z_j) and output (y_j) for each unit (forward prop)
4. Compute δ_n for output layer $\delta_n = \frac{ds(z_n)}{dz_n} (y_n - y_n^*) = y_n(1 - y_n)(y_n - y_n^*)$
5. Compute δ_j for preceding layer by backprop rule (repeat for all layers)
6. Compute weight change by descent rule (repeat for all weights)

