COMP9444
Neural Networks and Deep Learning
Term 2, 2024

Week 1 Tutorial: Perceptrons (Sample Solution)

1. **Introduce yourselves, get to know your fellows and your tutor.**

2. **Perceptron Learning**

   (a) Construct by hand a Perceptron which correctly classifies the following data; use your knowledge of plane geometry to choose appropriate values for the weights w_0, w_1, and w_2.

Training Examples	x1	x2	Class
a.	0	1	-1
b.	2	0	-1
c.	1	1	+1

$y = kx + b$

slope: $\frac{y_2 - y_1}{x_2 - x_1} = \frac{0-1}{2-0} = -\frac{1}{2} = k$.

interset: should be $(\frac{1}{2}, 1)$

then $y = (-\frac{1}{2}) \times \frac{1}{2} + b = 1 \Rightarrow -\frac{1}{4} + b = 1$

$b = \frac{5}{4}$

The first step is to plot the data on a 2-D graph, and draw a line which separates the positive from the negative data points:

$y = -\frac{1}{2}x + \frac{5}{4}$

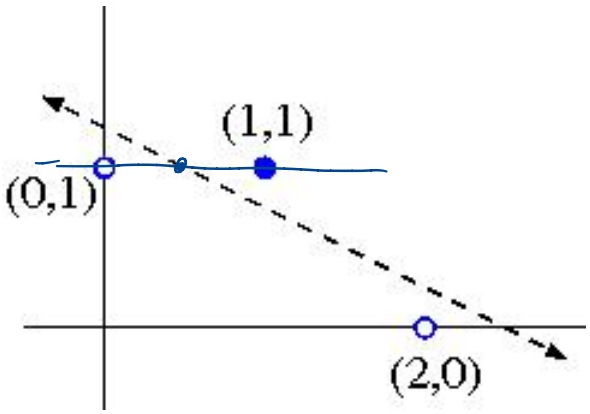
$x_2 + \frac{1}{2}x_1 - \frac{5}{4} = 0$

$4x_2 + 2x_1 - 5 = 0$

$w_0 = 4$

$w_1 = 2$

$w_2 = -5$



This line has slope $\frac{-1}{2}$ and x_2-intersect $\frac{5}{4}$, so its equation is:

$$x_2 = \frac{5}{4} - \frac{x_1}{2}$$

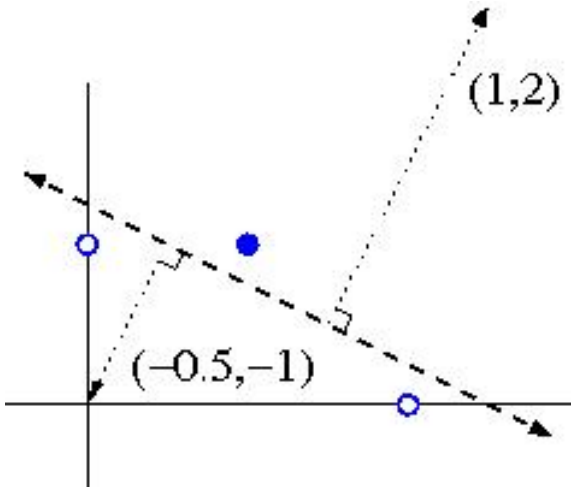
i.e.,

$$2x_1 + 4x_2 - 5 = 0$$

Taking account of which side is positive, this corresponds to these weights:
w_0 = -5
w_1 = 2

$w_2 = 4$

Alternatively, we can derive weights w1=1 and w2=2 by drawing a vector normal to the separating line, in the direction pointing towards the positive data points:



The bias weight w_0 can then be found by computing the dot product of the normal vector with a perpendicular vector from the separating line to the origin. In this case $w_0 = 1(-0.5) + 2(-1) = -2.5$
(Note: these weights differ from the previous ones by a normalising constant, which is fine for a Perceptron)

(b) Demonstrate the Perceptron Learning Algorithm on the above data, using a learning rate of 1.0 and initial weight values of
w_0 = -1.5
w_1 = 0
w_2 = 2

learning rate η

感知器公式 $S = w_0 + w_1 x_1 + w_2 x_2$

权重更新规则：若分类错误

① 负样本(实际-1) $w_j = w_j - \eta x_j$

② 正样本(实际+1) $w_j = w_j + \eta x_j$

In your answer, you should clearly indicate the new weight values at the end of each training step. The first three steps are shown here:

Iteration	w_0	w_1	w_2	Training Example	x_1	x_2	Class	$s = w_0 + w_1 x_1 + w_2 x_2$	Action
1	-1.5	0	2	a.	0	1	-	+0.5	Subtract
2	-2.5	0	1	b.	2	0	-	-2.5	None
3	-2.5	0	1	c.	1	1	+	-1.5	Add

Below is the solution table. Note the final weights when all items are correctly classified.

第一次迭代 $S = w_0 + w_1 x_1 + w_2 x_2$
$= -1.5 + 0.0 + 2.1$
$= 0.5 > 0$，与class不符
∴更新权重 $w_0 = -1.5 - 1 = -2.5$
$w_1 = 0 - 1 \times 0 = 0$
$w_2 = 2 - 1 \times 1 = 2 - 1 = 1$

第2次迭代 $S = w_0 + w_1 x_1 + w_2 x_2$
$= -2.5 + 0 \cdot 2 + 1 \cdot 0$
$= -2.5 < 0$ class<0 ✓

3: $S = w_0 + w_1 x_1 + w_2 x_2$
$= -2.5 + 0 \times 1 + 1 \times 1 = -1.5 < 0$

实际为正，更新权重：$w_0 = w_0 + \eta x_0 = -2.5 + 1 = -1.5$
$w_1 = w_1 + \eta x_1 = 0 + 1 \times 1 = 1$
$w_2 = w_2 + \eta x_2 = 1 + 1 \times 1 = 2$

Iteration	w_0	w_1	w_2	Training Example	x_1	x_2	Class	$s = w_0 + w_1x_1 + w_2x_2$	Action
1	-1.5	0	2	a.	0	1	-	+0.5	Subtract
2	-2.5	0	1	b.	2	0	-	-2.5	None
3	-2.5	0	1	c.	1	1	+	-1.5	Add
4	-1.5	1	2	a.	0	1	-	+0.5	Subtract
5	-2.5	1	1	b.	2	0	-	-0.5	None
6	-2.5	1	1	c.	1	1	+	-0.5	Add
7	-1.5	2	2	a.	0	1	-	+0.5	Subtract
8	-2.5	2	1	b.	2	0	-	+1.5	Subtract
9	-3.5	0	1	c.	1	1	+	-2.5	Add
10	-2.5	1	2	a.	0	1	-	-0.5	None
11	-2.5	1	2	b.	2	0	-	-0.5	None
12	-2.5	1	2	c.	1	1	+	+0.5	None

3. Computing any Logical Function with a 2-layer Network

Recall that any logical function can be converted to **Conjunctive Normal Form** (CNF), which means a conjuction of terms where each term is a disjunction of (possibly negated) literals. This is an example of an expression in CNF:
$(A \vee B) \wedge (\neg B \vee C \vee \neg D) \wedge (D \vee \neg E)$

Assuming False=0 and True=1, explain how each of the following could be constructed. You should include the bias for each node, as well as the values of all the weights (input-to-output or input-to-hidden and hidden-to-output, as appropriate).

(a) Perceptron to compute the OR function of m inputs,

Set the bias weight to $\frac{-1}{2}$ and all other weights to 1.
It makes sense for the input-to-output weights to be 1, because any of the inputs being True makes it more likely for the output to be True. In fact, the ONLY way the output can be False is if ALL the inputs are False. By setting the bias to $\frac{-1}{2}$, we insure that the linear combination is slightly negative when all of the inputs are False, but becomes positive when any of the inputs is True.

(b) Perceptron to compute the AND function of n inputs,

Set the bias weight to $(\frac{1}{2} - n)$, all other weights to 1. The ONLY way the conjunction can be True is if ALL the inputs are True. By setting the bias to $(\frac{1}{2} - n)$, we insure that the linear combination is slightly positive when all of the inputs are True, but becomes negative when any of the inputs is False.

(c) Two-layer Neural Network to compute the function
$(A \vee B) \wedge (\neg B \vee C \vee \neg D) \wedge (D \vee \neg E)$

Hint: first consider how to construct a Perceptron to compute the OR function of m inputs, with k of the m inputs negated.

first layer: bias $= -\frac{1}{2}$, weight$_A = 1$, $W_{13} = 1$

Sed layer: bias $=$, $W_{13} = -1$, $W_c = 1$, $W_D = -1$
$x_C = 1$, $x_E = 0$

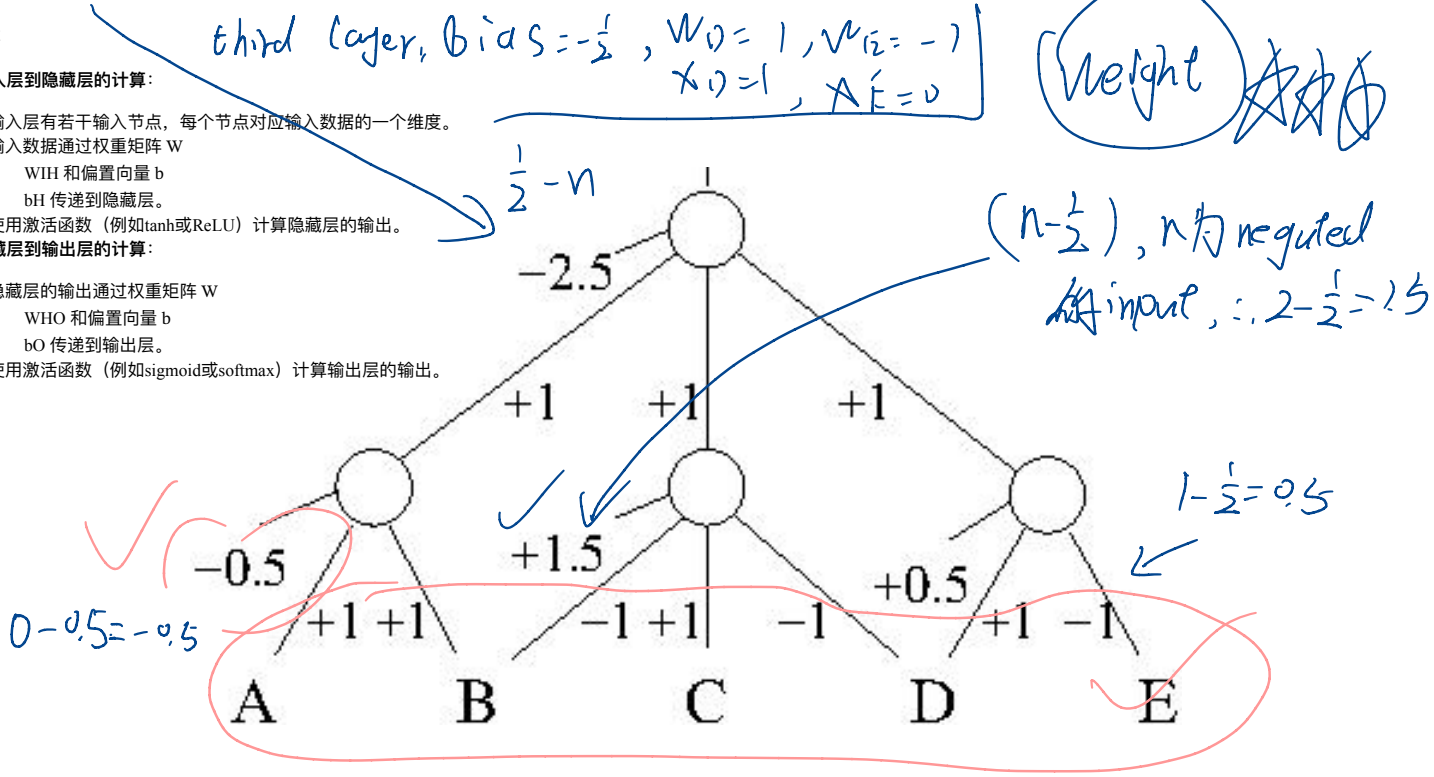
解释步骤

1 **输入层到隐藏层的计算**:

输入层有若干输入节点，每个节点对应输入数据的一个维度。
输入数据通过权重矩阵 W
WIH 和偏置向量 b
bH 传递到隐藏层。
使用激活函数（例如tanh或ReLU）计算隐藏层的输出。

2 **隐藏层到输出层的计算**:

隐藏层的输出通过权重矩阵 W
WHO 和偏置向量 b
bO 传递到输出层。
使用激活函数（例如sigmoid或softmax）计算输出层的输出。



Each hidden node should compute one disjunctive term in the expression. The input-to-hidden weights are -1 for items that are negated, $+1$ for the others. The output node then computes the conjunction of all the hidden nodes, as in part (b).

With reference to this example, explain how a two-layer neural network could be constructed to compute any (given) logical expression, assuming it is written in Conjunctive Normal Form.

As in the example above, each hidden node should compute one disjunctive term in the expression; the output node then computes the conjunction of all these hidden nodes. The input-to-hidden weights should be -1 for items that are negated, $+1$ for the others. The bias for each hidden node should be $(k - \frac{1}{2})$ where k is the number of items that are negated in the disjunctive term corresponding to that node.

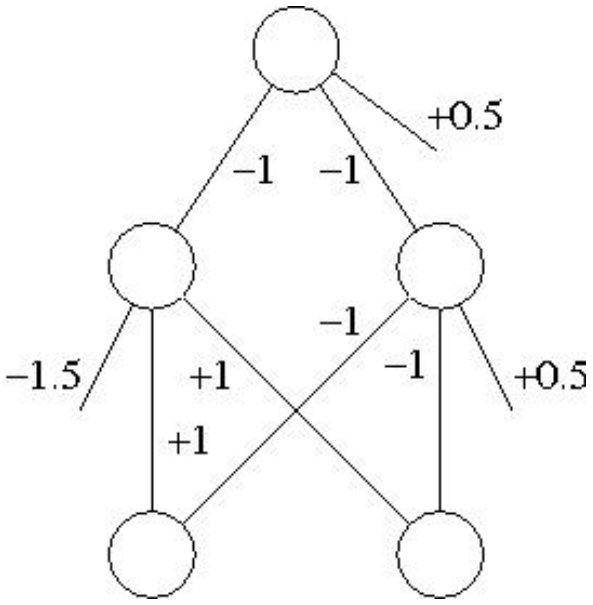
4. XOR Network

Construct by hand a Neural Network (or Multi-Layer Perceptron) that computes the XOR function of two inputs. Make sure the connections, weights and biases of your network are clearly visible.

There are a number of ways to express XOR as a combination of simpler functions that are linearly separable. For example, using NOR as an abbreviation for "NOT OR", (x_1 XOR x_2) can be written as:

(x_1 AND x_2) NOR (x_1 NOR x_2)

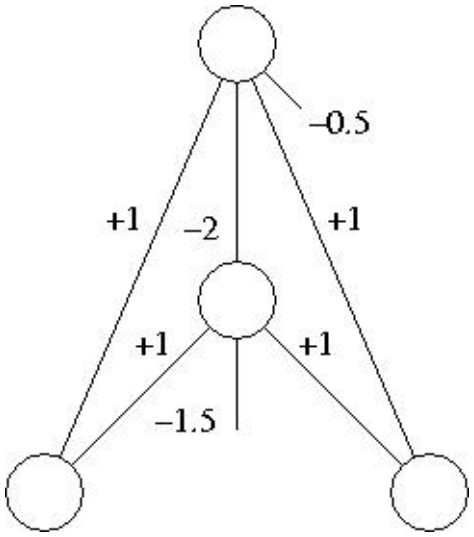
This decomposition allows us to compute XOR with a network like below:



Challenge: Can you construct a Neural Network to compute XOR which has only one hidden unit, but also includes shortcut connections from the two inputs directly to the (one) output?

Hint: Start with a network that computes the inclusive OR, and then try to think of how it could be modified.

Exclusive OR (XOR) is very similar to normal (inclusive) OR, except for the case where both inputs are True, i.e. where (x_1 AND x_2) is True. We therefore introduce a single hidden unit which computes (x_1 AND x_2). This hidden unit is connected to the output with a negative weight, thus forcing the overall output to be False when the output from this hidden node is positive. See the network below.



5. **Implications of Deep Learning**

   What potential benefits and dangers might Deep Learning pose for education, entertainment, the economy, and society in general?