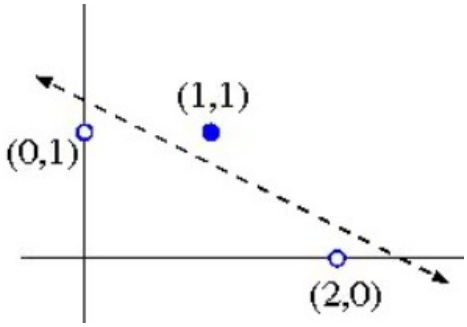
COMP3411 Tutorial - Week 9
Neural Nets and ILP

Question 1

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 Example	x_1	x_2	Class
a.	0	1	-1
b.	2	0	-1
c.	1	1	+1

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:



Based on the two intersection points, we can derive the following line slope:

$$Slope = m = \frac{(y_2 - y_1)}{(x_2 - x_1)} = \frac{(0 - 1)}{(2 - 0)} = -0.5$$

With the slope, the corresponding line equation can be partially calculated, utilising an identified point on the line, which will be (0.5, 1) for this example (the point between (0,1) and (1,1)):

$$y = mx + b$$
$$\cong 1 = -0.5 * 0.5 + b$$
$$b = 5/4 \equiv 1.25$$
$$y = -0.5x + 1.25$$

Since x_2 is equivalent to y based on our dataset, we can deduce a quadratic equation using our line equation which will provide the weightings:

$$x_2 = -0.5x_1 + 1.25$$
$$\equiv 0 = 0.5x_1 + x_2 - 1.25$$
$$\equiv 0 = 2x_1 + 4x_2 - 5$$

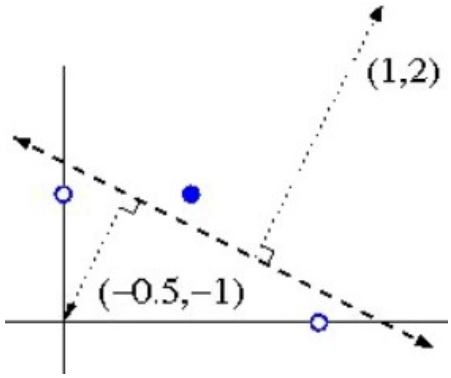
This results in the following weights:

$w_0 = -5$

$w_1 = 2$

$w_2 = 4$

Alternatively, we can derive weights w_1=1 and w_2=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 the following initial weight values:

$w_0 = -0.5$

$w_1 = 0$

$w_2 = 1$

Be sure to round the predicted result to the closest class value available (-1 or 1 in this case).

Iteration	w_0	w_1	w_2	Example	x_1	x_2	Class	Prediction	Rounded	Action
1	-2.5	0.0	-1.0	a	0.0	1.0	-1	0.5	1.0	Subtract
2	-2.5	0.0	-1.0	b	2.0	0.0	-1	-2.5	-1	None
3	-0.5	2.0	1.0	c	1.0	1.0	1	-3.5	-1	Add
4	-2.5	2.0	-1.0	a	0.0	1.0	-1	0.5	1	Subtract
5	-4.5	-2.0	-1.0	b	2.0	0.0	-1	1.5	1	Subtract
6	-2.5	0.0	1.0	c	1.0	1.0	1	-7.5	-1	Add
7	-2.5	0.0	1.0	a	0.0	1.0	-1	-1.5	-1	None
8	-2.5	0.0	1.0	b	2.0	0.0	-1	-2.5	-1	None
9	-0.5	2.0	3.0	c	1.0	1.0	1	-1.5	-1	Add
10	-2.5	2.0	1.0	a	0.0	1.0	-1	2.5	1.0	Subtract
11	-4.5	-2.0	1.0	b	2.0	0.0	-1	1.5	1.0	Subtract
12	-2.5	0.0	3.0	c	1.0	1.0	1	-5.5	-1	Add
13	-4.5	0.0	1.0	a	0.0	1.0	-1	0.5	1	Subtract
14	-4.5	0.0	1.0	b	2.0	0.0	-1	-4.5	-1	None
15	-2.5	2.0	3.0	c	1.0	1.0	1	-3.5	-1	Add
16	-4.5	2.0	1.0	a	0.0	1.0	-1	0.5	1.0	Subtract
17	-4.5	2.0	1.0	b	2.0	0.0	-1	-0.5	-1.0	None
18	-2.5	4.0	3.0	c	1.0	1.0	1	-1.5	-1	Add
19	-4.5	4.0	1.0	a	0.0	1.0	-1	0.5	1.0	Subtract
20	-6.5	0.0	1.0	b	2.0	0.0	-1	3.5	1	Subtract
21	-4.5	2.0	3.0	c	1.0	1.0	1	-5.5	-1	Add
22	-4.5	2.0	3.0	a	0.0	1.0	-1	-1.5	-1	None
23	-4.5	2.0	3.0	b	2.0	0.0	-1	-0.5	-1	None
24	-4.5	2.0	3.0	c	1.0	1.0	1	0.5	1	None

Note that we only have three training examples (a, b, c), but we keep iterating over them until no corrections are required for the network to produce the correct output.

Question 2

Explain how each of the following could be constructed:

a) Perceptron to compute the OR function of m inputs

Set the bias weight to -½, all other weights to 1.

The OR function is almost always True. The only way it can be False is if all inputs are 0. Therefore, we set the bias to be slightly less than zero for this input.

b) Perceptron to compute the AND function of n inputs

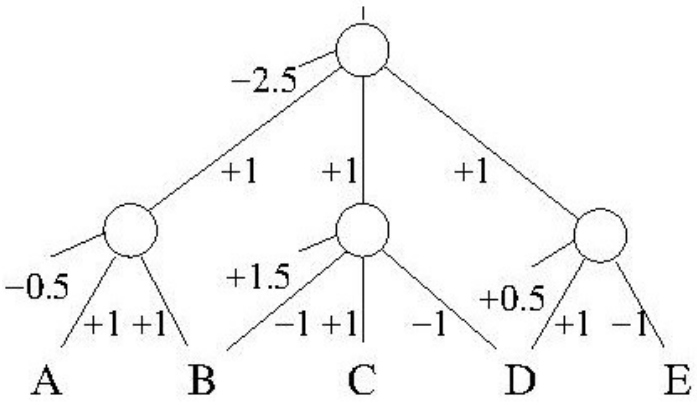
Set the bias weight to $\left(\frac{1}{2} - n\right)$*, all other weights to 1.*

AND function is almost always False. The only way it can be True is if all inputs are 1. Therefore, we set the bias so that, when all inputs are 1, the combined sum is slightly greater than 0.

c) 2-Layer Neural Network to compute any (given) logical expression, assuming it is written in Conjunctive Normal Form.

Each hidden node should compute one disjunctive term in the expression. The weights should be -1 for items that are negated, +1 for others. The bias should be k - ½ where k is the number of items that are negated. The output node then computes the conjunction of all hidden nodes.

For example, here the network computes $(A \vee B) \wedge (\neg B \vee C \vee \neg D) \wedge (D \vee \neg E)$



Question 3

a) Find the least general generalisation of the following terms:

```
f(g(a, b), [1, 2, [3, 4], 5], 1 + 2 * 6)
f(g(a, h(x, y)), [1, 2, [3, 4, 5]], 1 + 6)
```

To find the LGG, look for where the terms differ and replace with a variable. Reuse the same variable if the same terms appear in corresponding positions.

```
lgg = f(g(a, X), [1, 2, [3, 4 | Y] | Y], 1 + Z)
subst = [X/{b, h(x, y)}, Y/{[], [5]}, Z/{2*6, 6}]
```

The list is tricky because the inverse substitution Y/{[], [5]} *applies also as* Y/{[5], []} .

b) Find the least general generalisation of the following clauses:

```
q(f(a)) :- p(a, b), r(b, c), r(b, e).
q(f(x)) :- p(x, y), r(y, z), r(w, z).
```

To find the LGG of two clauses, match the head, then find all combinations of possible LGGs between literals that have consistent inverse substitutions.

```
q(f(X)) :- p(X, Y), r(Y, Z), r(W, Z), R(Y, E), r(W, E)
subst = [X/{a, x}, Y/{b, y}, Z/{c, z}, W/{b, w}, E/{e, z}]
```