

## HW5

**Q1. An almost clique is any graph that is the result of deleting one edge from a clique. Prove that the problem of whether a graph  $G$  has an almost clique of size  $t$  is  $NP$ -complete.**

Let ALMOST-CLIQUE be the language:  $\{\langle G \rangle \mid G \text{ is a graph with an almost clique of size } t\}$

Consider an algorithm  $A(\langle G \rangle, \langle ac \rangle)$ , which:

1. Checks whether  $\langle G \rangle$  is in the form of a graph and  $\langle ac \rangle$  is in the format of a set of  $t$  vertices for this graph. Otherwise, it rejects the inputs.
2. Checks whether  $\langle ac \rangle$  is an almost clique by checking in  $G$  whether all except one of the  $\binom{t}{2}$  vertex pairs in  $\langle ac \rangle$  is connected with an edge

Since,  $\langle G \rangle$  represents a set of vertices and edges in the graph, and  $\langle ac \rangle$  represents only a subset of  $t$  vertices in this graph:  $|\langle G \rangle| < |\langle ac \rangle|$ .

Since,  $\langle G \rangle$  represents a set of vertices and edges in the graph, the number of edges checked to verify almost clique will be less than or equal to the total number of edges in the graph. Hence, the verification will take  $O(|\langle G \rangle|)$  time.

Hence, ALMOST-CLIQUE is the language:  $\{\langle G \rangle \mid \langle G \rangle \in \{0, 1\}^* : \exists \langle ac \rangle, |\langle ac \rangle| < |\langle G \rangle| \text{ and } A(\langle G \rangle, \langle ac \rangle) = 1\}$

Therefore, ALMOST-CLIQUE is in NP.

To prove ALMOST-CLIQUE is NP-Hard, we reduce CLIQUE to ALMOST-CLIQUE.

1. Consider a graph  $G$  with a clique of size  $t$ .
2. Add two new vertices to  $G$ :  $v_{n+1}$  and  $v_{n+2}$ . Connect them to all the other  $n$  vertices, but not each other. Hence, this graph,  $G'$  will contain an almost clique of size  $t + 2$ .
3. For  $G'$  to have a  $t + 2$  almost clique, there are 3 possibilities:
  - a. The almost clique contains the two new vertices  $\{v_{n+1}, v_{n+2}\}$ . This implies that the missing edge must be the one between these two vertices, and the remaining vertices form a  $t$ -size clique
  - b. The almost contains one of the vertices  $\{v_{n+1}, v_{n+2}\}$ . This implies that the missing edge must be inside  $G$ , say  $e = \{u, v\} \in G$ . If we remove  $u$  and  $v$  then the other  $t$  vertices in  $G$  must form a  $t$ -size clique.
  - c. The almost clique does not contain any of the vertices  $\{v_{n+1}, v_{n+2}\}$ . This implies that  $G$  must contain a  $t$ -size clique.

Hence, since CLIQUE is reduced to ALMOST-CLIQUE in polynomial time, and CLIQUE is in NP, ALMOST-CLIQUE is NP-Hard.

Since ALMOST-CLIQUE is in NP and is also NP-Hard, it is NP Complete.

**Q2. Show 0-1-2 integer programming is NP-complete where 0-1-2 integer programming is the problem: Given a list of  $m$  linear inequalities with rational coefficients over  $n$  variables  $u_1, \dots, u_n$  (i.e.,  $m$  inequalities of the form  $a_1 u_1 + a_2 u_2 + \dots + a_n u_n \leq b$  where the  $a_i$  and  $b$  are fraction  $p/q$  for some integers  $p$  and  $q$ ), decide if there is an assignment of the numbers 0, 1, or 2 to the variables that satisfies all the inequalities.**

We first prove that BinLP (0-1 integer programming) is NP-complete. We then use this result to prove 0-1-2 integer programming is NP-complete.

Let us represent the  $m$  linear inequalities in matrix form as:

$AU \leq B$ , where

$A$  is a  $m \times n$  matrix of coefficients  $a_1$  to  $a_n$  for the  $m$  inequalities

$U$  is an  $n \times 1$  matrix of variables  $u_1$  to  $u_n$

$B$  is an  $m \times 1$  matrix for  $b$  in the  $m$  inequalities

Hence, witness  $U$  will be of length  $n$ . With matrix multiplication, we can verify the witness in polynomial time. Hence, the BinLP is in NP.

Now, we prove BinLP is NP-Hard by reducing 3SAT to BinLP:

Let the variables in the 3SAT formula be  $x_1, x_2, \dots, x_n$ . We will have corresponding variables  $u_1, u_2, \dots, u_n$  in our BinLP program. First, we constraint each variable to be 0 or 1:

$$\forall i, z_i \in \{0, 1\}$$

Assigning  $u_i = 1$  in BinLP is equivalent to setting  $x_i = \text{true}$  in 3SAT;  $u_i = 0$  is equivalent to  $x_i = \text{false}$ .

For each clause such as  $(x_1 \text{ OR } \text{not}(x_2) \text{ OR } \text{not}(x_3))$ , we introduce the constraint:

$$u_1 + (1 - u_2) + (1 - u_3) \leq 1$$

To satisfy this inequality we must set at two among  $u_1, u_2, u_3$  as 1, and correspondingly, two among  $x_1, x_2$ , and  $x_3$  as true.

This can be implemented in the above matrix equation by adding a row of 1's to the bottom of  $A$  to make it a  $m+1 \times n$  matrix (hence,  $B$  becomes an  $m+1 \times 1$  matrix).

If the given instance  $I$  outputs true for 3SAT then  $f(I)$  is true for BinLP. Just take a satisfying assignment  $A$  to the variables  $x_i$  and set each  $u_i$  to 0 or 1 accordingly. Since  $A$  satisfied at least one literal in each clause, this means the associated sum is  $\leq 1$ . In the other direction, any solution to the BinLP must set at least one of the associated literals to 1, since each is an integer 0 or 1.

Hence, 3SAT is reduced to BinLP, proving BinLP to be NP-Hard. Since BinLP is in NP and is NP-Hard, BinLP is NP-complete.

Now, we prove 0-1-2 integer programming is NP complete using the above results.

0-1-2 integer programming is in NP following the same logic as for BinLP above.

To the above proof, we add the constraint that no individual variable can be greater than 1. This can be implemented in the matrix equation by appending an  $n \times n$  identity matrix below A and n 1's below B.

This proves that any solution to 0-1-2 programming can also be a solution to BinLP. Since, BinLP is NP-Hard, this makes 0-1-2 programming NP-hard.

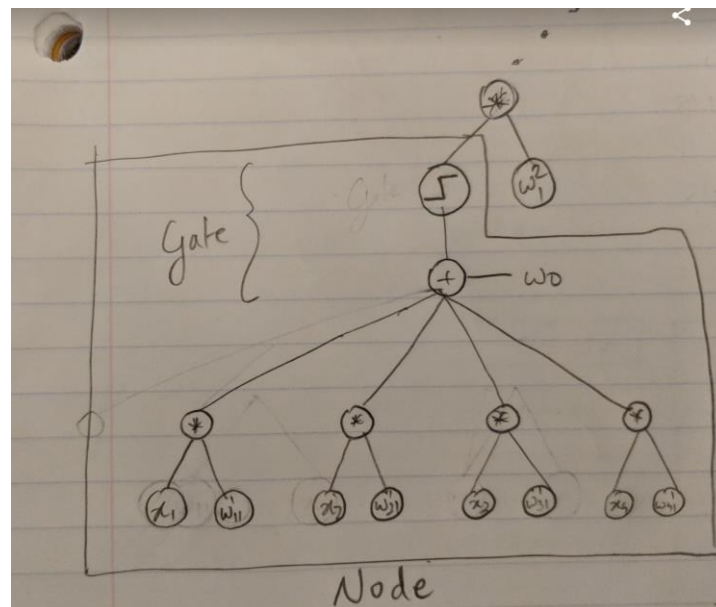
Since 0-1-2 programming is in NP and is NP-Hard, it is NP-Complete.

Q3. A neural net gate  $NN(x_1, \dots, x_n, w_0, w_1, \dots, w_n)$  outputs 1 if  $w_0 + \sum w_i x_i > 0$  and 0 otherwise. Here  $x_i$  are viewed as the inputs and  $w_i$  are called weights. We imagine weights are fixed after some training process. A neural network is a directed acyclic graph where the nodes are labeled with  $NN$  gates. The output of such a network is computed in the natural way by evaluating gates which are immediately connected to the inputs, followed by gates all of whose inputs now have values, and so on. Define the neural network understanding (NNU) problem to be given a neural network  $N$  and a setting for its weights  $\vec{w}$ , decide if there is a setting of its inputs  $\vec{x}$  which makes it output 1. Show the  $NNU$  problem is  $NP$ -complete.

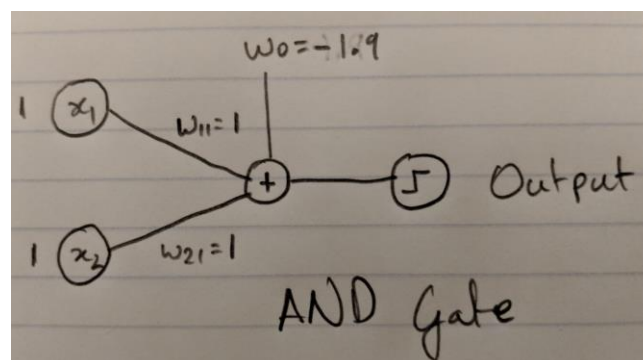
NNU is in NP as for a given NN, we can in polynomial time verify that  $w_0 + \sum w_i x_i > 0$  for each node in the NN.

We reduce CIRCUIT-SAT to NNU to prove that NNU is NP-Hard.

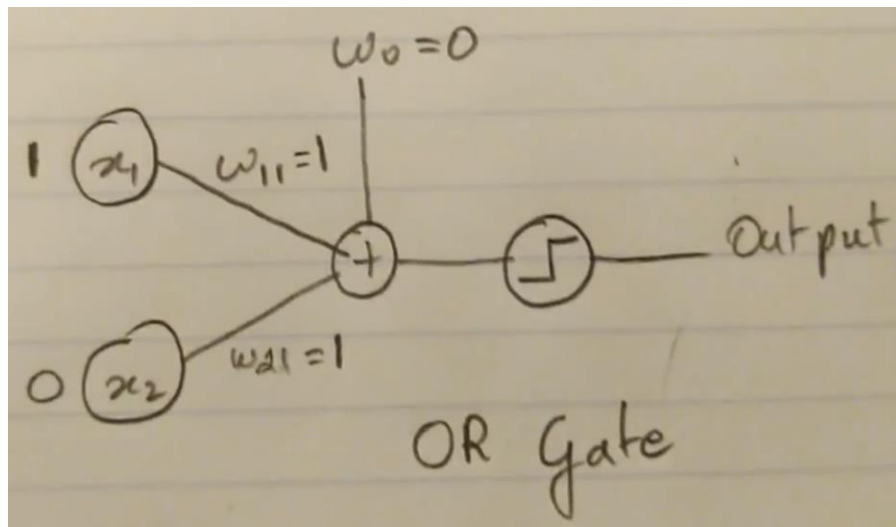
The intuition for this is that an NN itself can be considered a circuit, as shown in the following example:



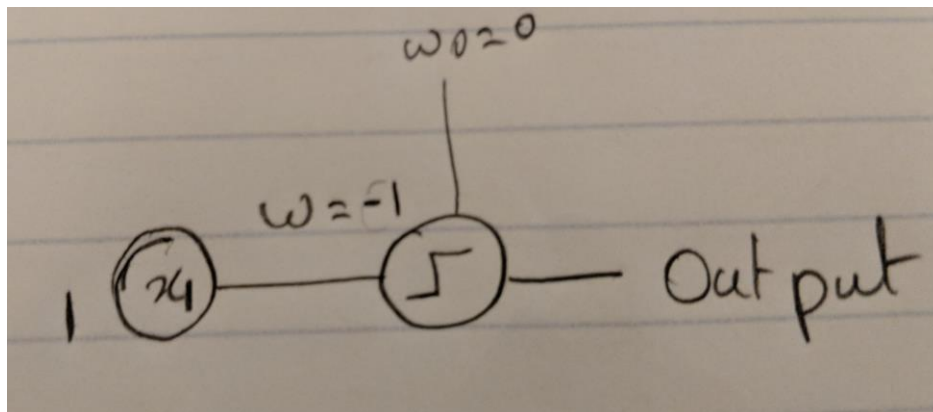
Further, an NN node can be used to simulate an AND gate if we give binary inputs and keep the weights as 1 for the activation functions ( $w_0 + \sum w_i x_i > 0$ ) provided above:



Similarly, an NN node can be used to simulate an OR gate if we keep the weights as 1 and employ the bias  $w_0$  with the given activation function, as follows:



Finally, it can also be used to simulate a NOT gate:



Hence, we see that CIRCUIT-SAT can be simulated as an NNU problem with AND, OR, and NOT gates. Any solution to the NNU problem will yield a solution to CIRCUIT-SAT. Since, each gate in a circuit can be translated to an NN node, the reduction of CIRCUIT-SAT to NNU can be done polynomially with respect to the number of nodes. And since CIRCUIT-SAT is NP-Hard, this implies that NNU is also NP-Hard.

Since, NNU is in NP and is NP-Hard, NNU is NP-Complete.