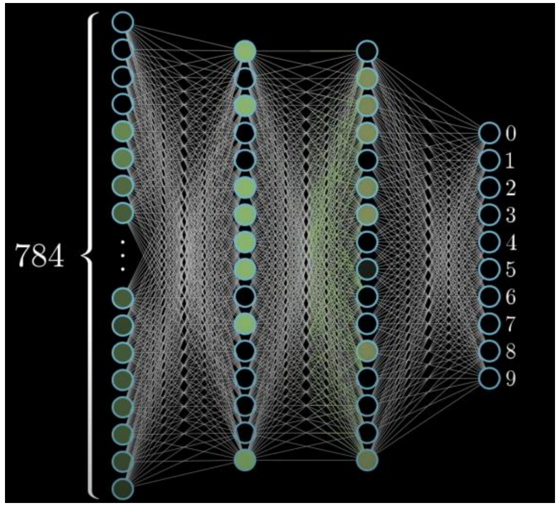
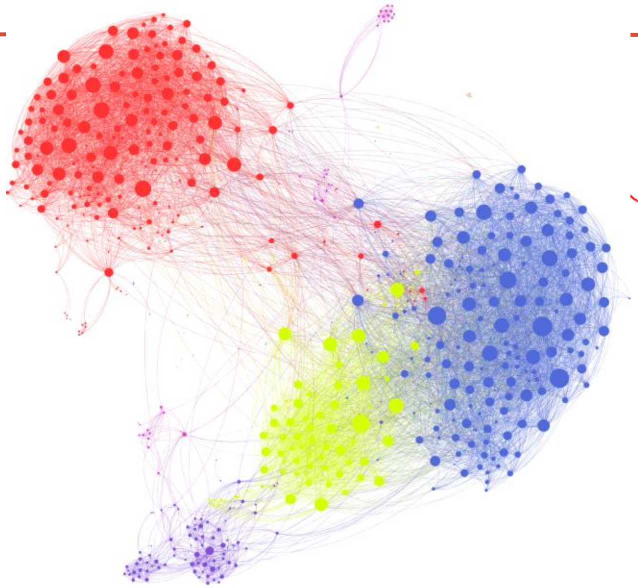
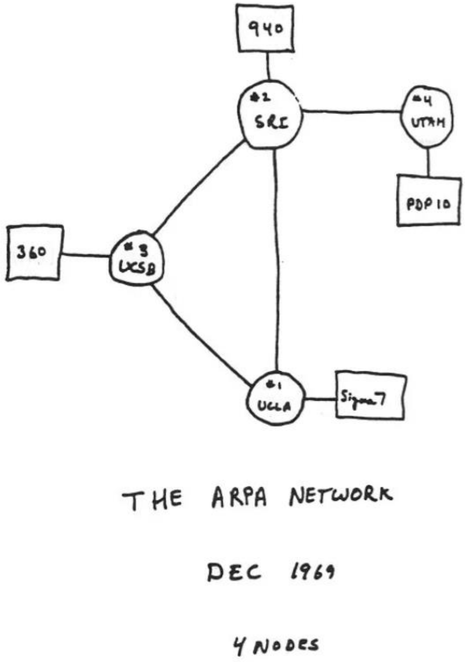
Handout: https://bit.ly/NeuralNets-GraphRepresentation

Book your mock interview today!

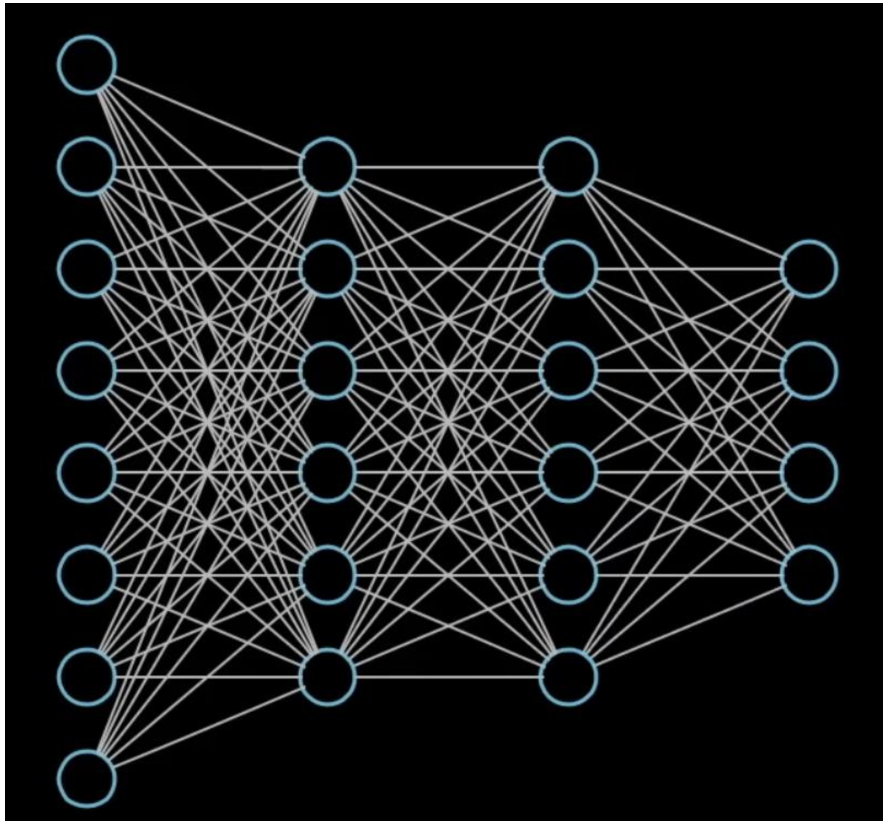
GRAPH REPRESENTATION

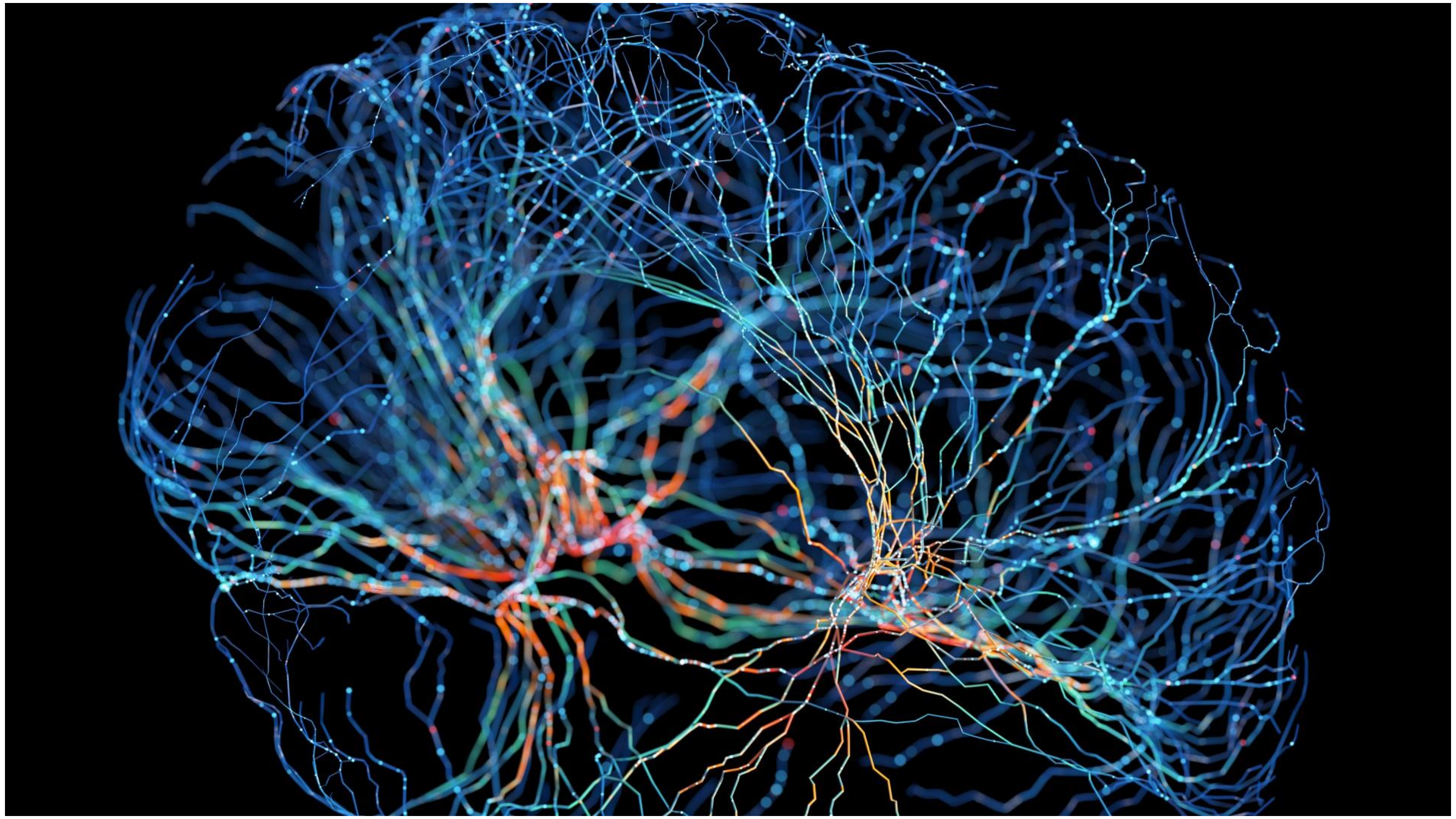






Neural Networks: Biologically inspired structure



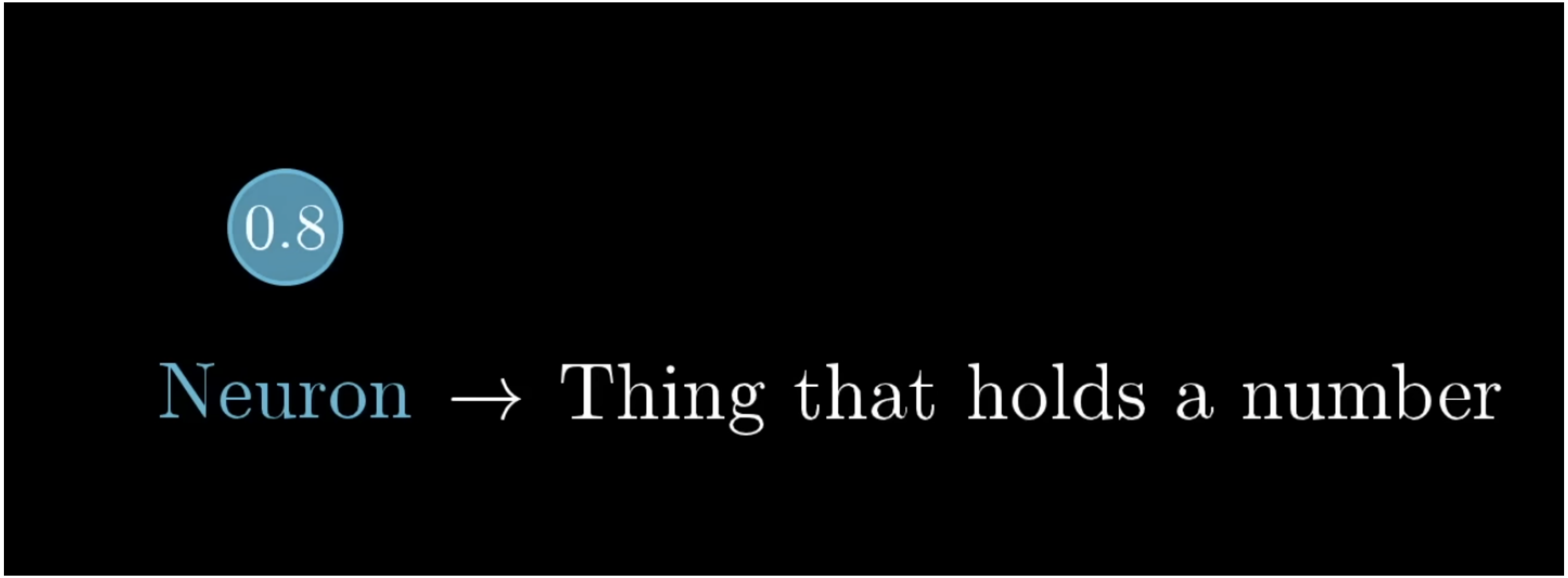


Neural Network: Collection of connected neurons modeled after the brain

Human brain has billions of neurons
Each neuron is connected to thousands of other neurons

Video intro to neural networks: https://youtu.be/aircAruvnKk?feature=shared

What is an artificial neuron?

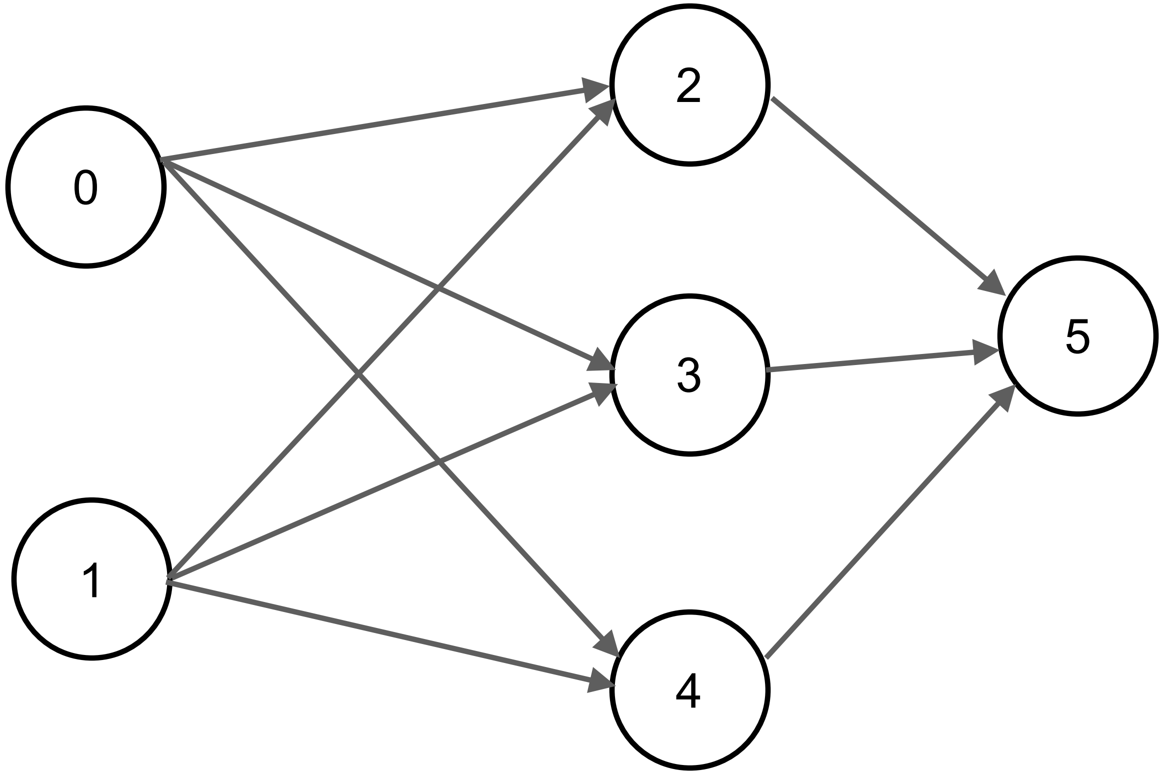


If a neuron just holds a number…where does that number come from?

Terminology: the number stored in the neuron is also called its activation value or value for short

What decides a neuron's value?

A neuron's activation is computed based on the activations of neurons connected to it.
Think of connections as pathways of information flow!

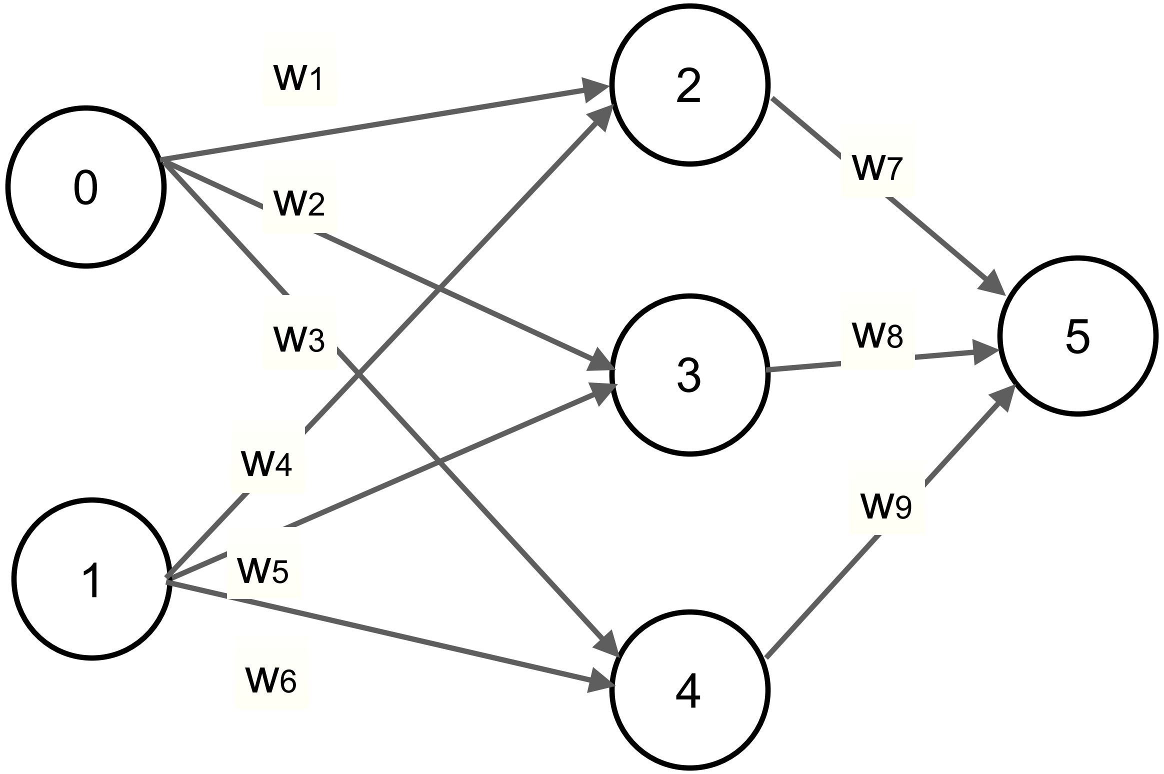


Which neurons will determine the activation value of neuron 3?

An example NN with 6 neurons and 9 connections

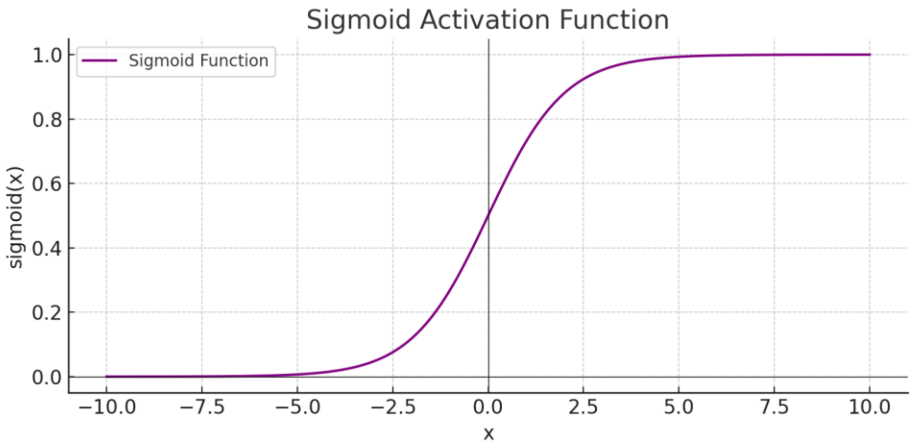
What decides a neuron's value?

The activation for a neuron is computed as the weighted sum of its input neurons (with some adjustments).

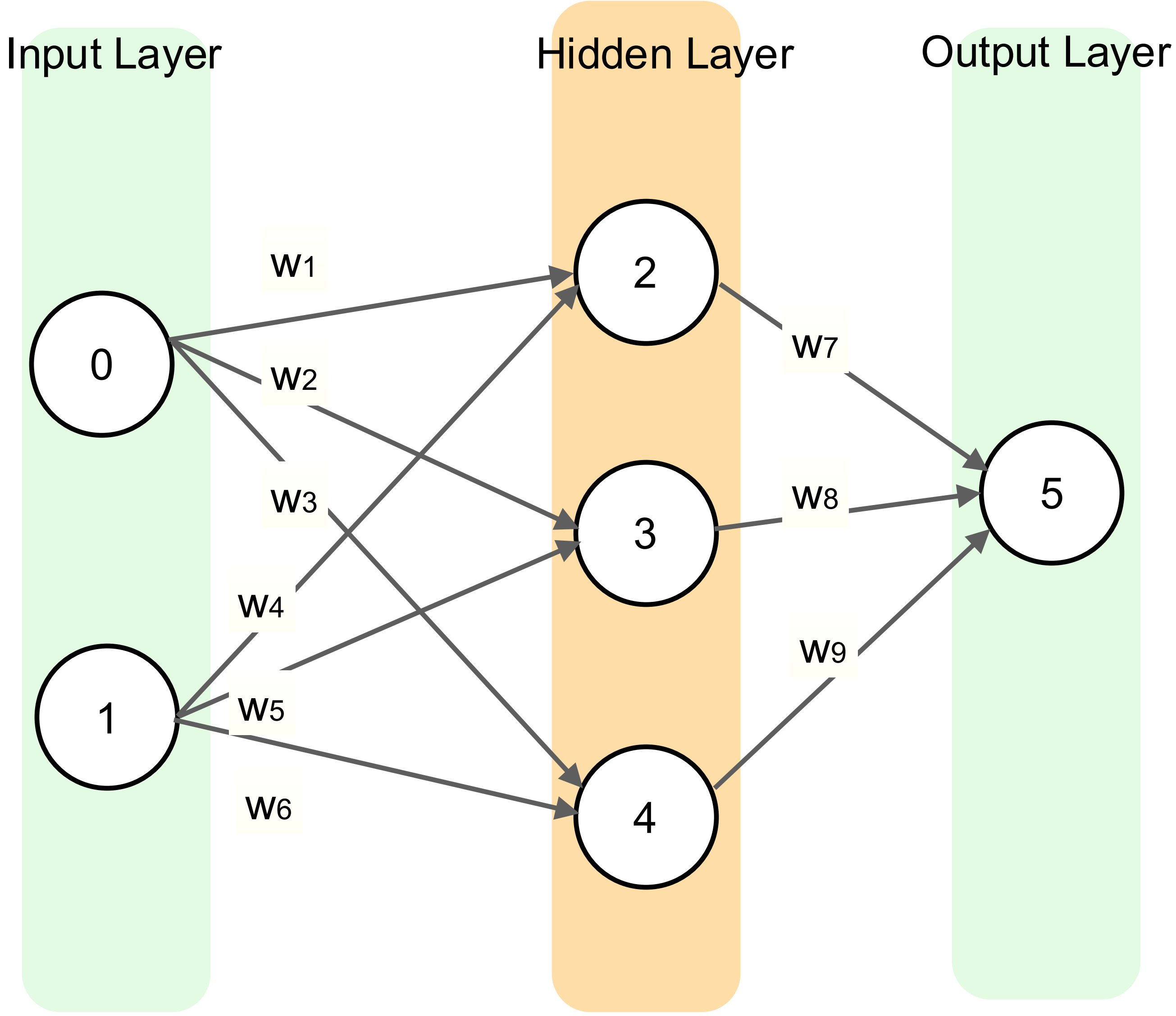


An example NN with 6 neurons and 9 connections

Compute the activation of neuron 3



How does a feed-forward neural network compute?



A **feedforward neural network (FNN)** is the simplest type of neural network where data flows in one direction — from input to output — without looping back.

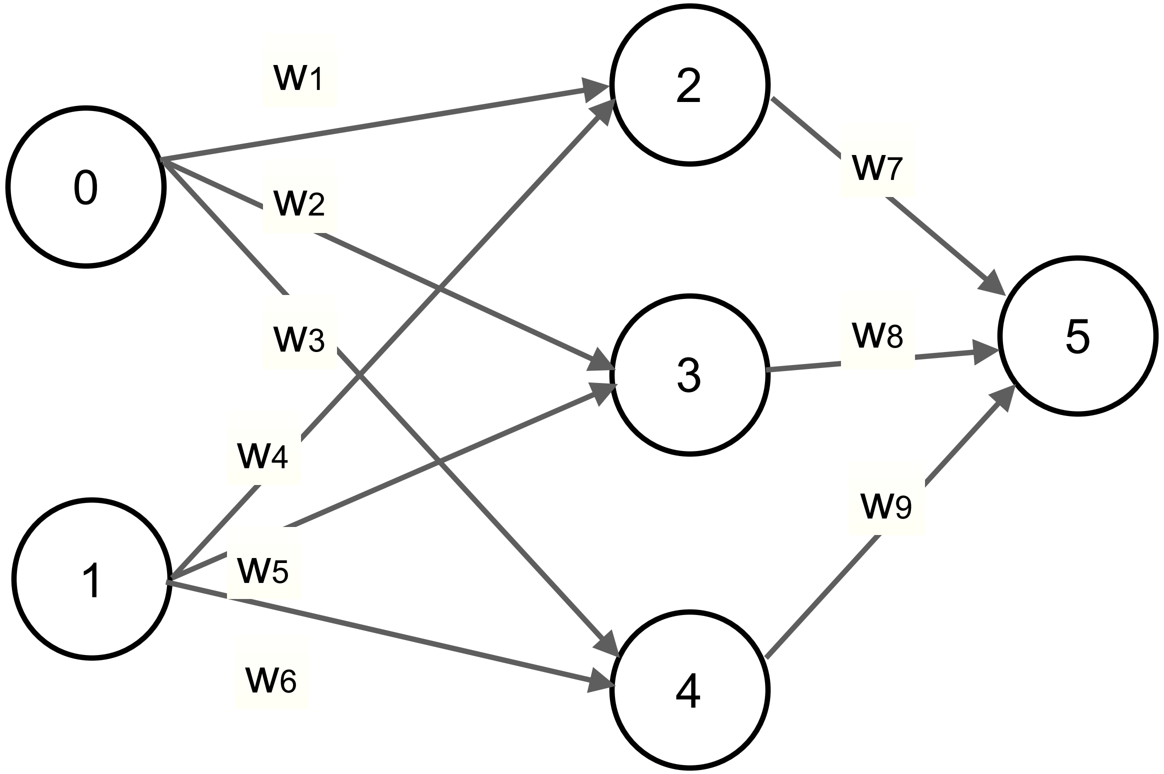
NN Terminology

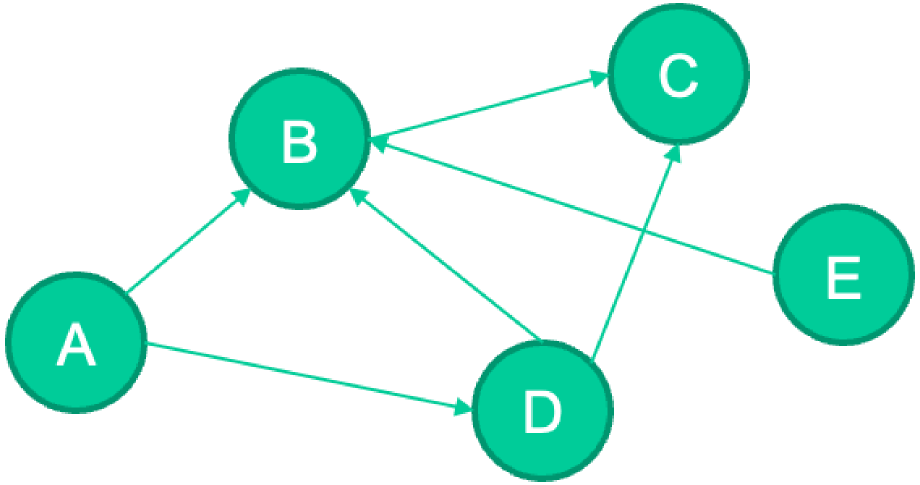
- Neurons
- Connections
- Input Layer
- Hidden Layer(s)
- Output Layer
- Neuron Info: activation value, bias, activation function

How would you represent the neural net using the data structures learned so far?

Neural Networks as Graphs

(PA03) Model a neural network as a ______________________________ graph.





NN is a set of neurons and connections

- Connections are directed (one-way)
- Connections have weights (strength of connection)

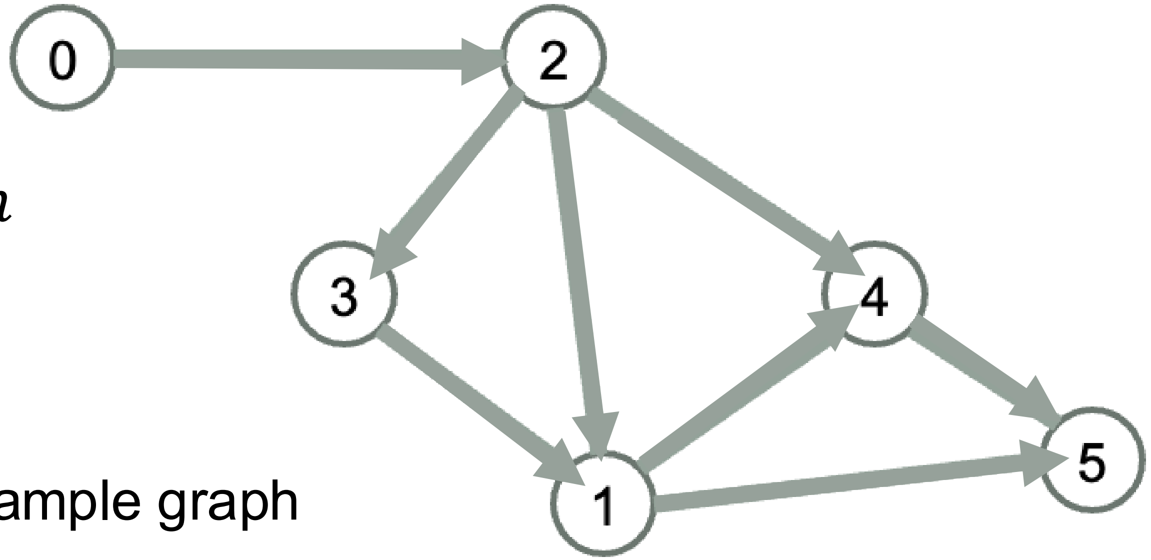
Graph is a set of nodes (vertices) and edges

- Directed graph: Edges go one way
- Undirected graph: Edges go both ways
- Weighted graph: Edges have weights

Graph Terminology and Notation

Graph $\mathrm{G} = \{V, E\}$

- Vertices $\mathrm{V} = \{0,\ 1,\ 2,\ 3, \dots, n-1\}; |V| = n$
- Edges $\mathrm{E} = \{(\mathrm{u}, \mathrm{v}) |\ \mathrm{u} \in V, v \in V\};\ |E| = m$



Activity 1: Write the vertices and edges for the example graph

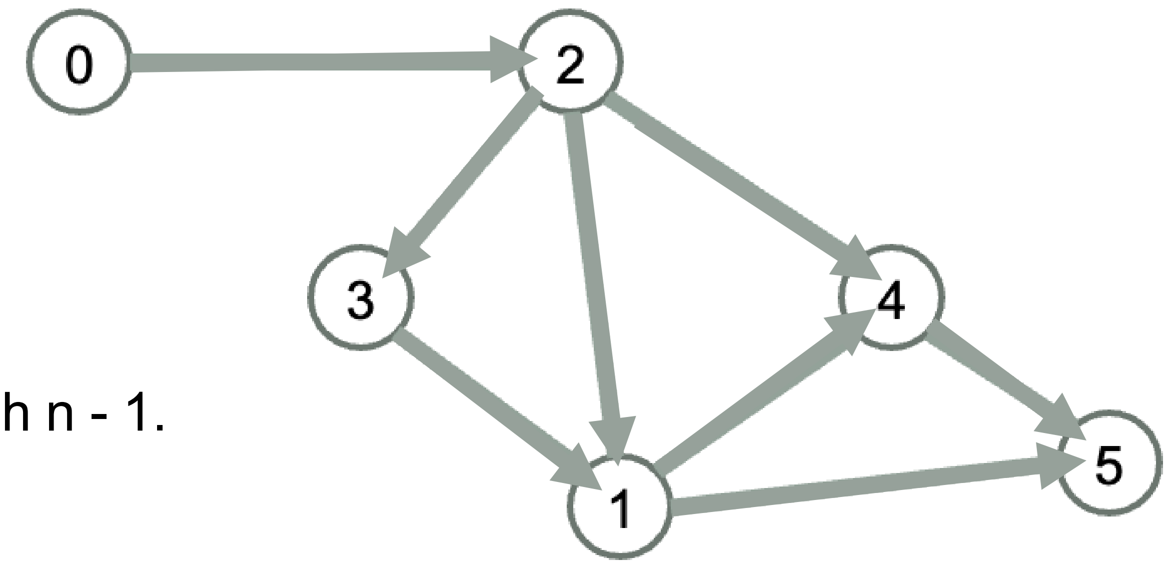
$V =$

$E =$

$n =$

$m =$

Representing vertices



Graph $\mathrm{G} = \{V, E\}$

$$\mathrm{V} = \{0, 1, 2, 3, \ldots, n-1\}$$

Activity 2: Your graph has vertices labeled 0 through n - 1.

What's the best way to store the set of vertices?

A. Vector, why?

B. Hashtable, why?

C. BST, why?

D. Something else?

Consider:

1. What do you need to do with the vertices?

(Look them up? Loop through them?)

2. Does it help that the vertex labels are just 0, 1, 2, ..., n - 1?

Convince your neighbor why your pick is the best.

Representing vertices with associated info

```
struct NodeInfo {
    // The value of this Node before activation.
    double preActivationValue;
    // The value of this Node after activation.
    double postActivationValue;
    double bias;  //bias
    // other fields associated with a neuron
};
std::vector<NodeInfo> nodes(n);  // where n is known ahead of time
OR
std::vector<NodeInfo*> nodes(n);
```

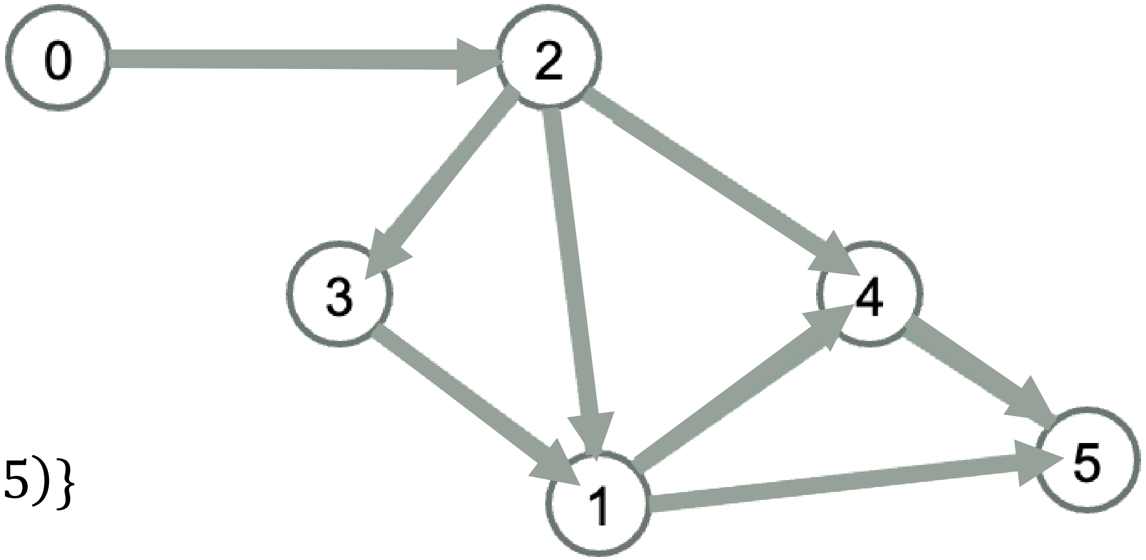
Representing the graph using an adjacency list

An **adjacency list** is a way to represent a graph where each vertex stores a list of its neighbors (the ones its connected to by an edge).

$G = (V, E)$

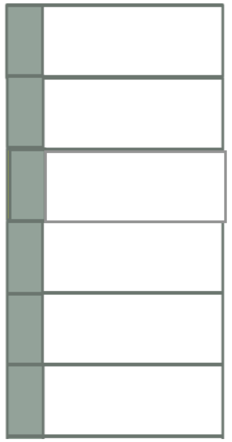
$V = \{0, 1, 2, 3, 4, 5\}$

$E = \{(0,2), (2,3), (2,1), (2,4), (3,1), (1,4), (1,5), (4,5)\}$



Guiding question: Given a vertex (v), how efficiently can we find all its neighbors?

Activity 3: Complete the adjacency list representation of the example graph.



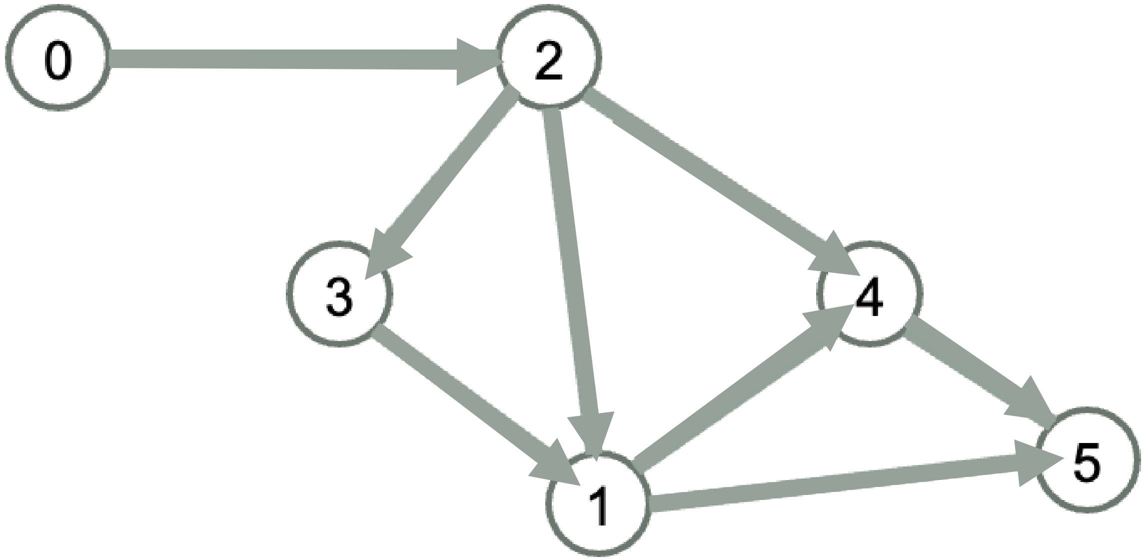
Does this look like any other data structure you have seen so far?

Representing a graph using an adjacency list

```
class graph{
  ...
  private:
   vector<NodeInfo*> nodes;
   _______________________ adjlist;
};
```

Choose the ADT to represent adjlist

A. vector<int>
B. vector<unordered_set<int>>
C. list<vector<int>>
D. vector<list<int>>
E. set<list<int>>

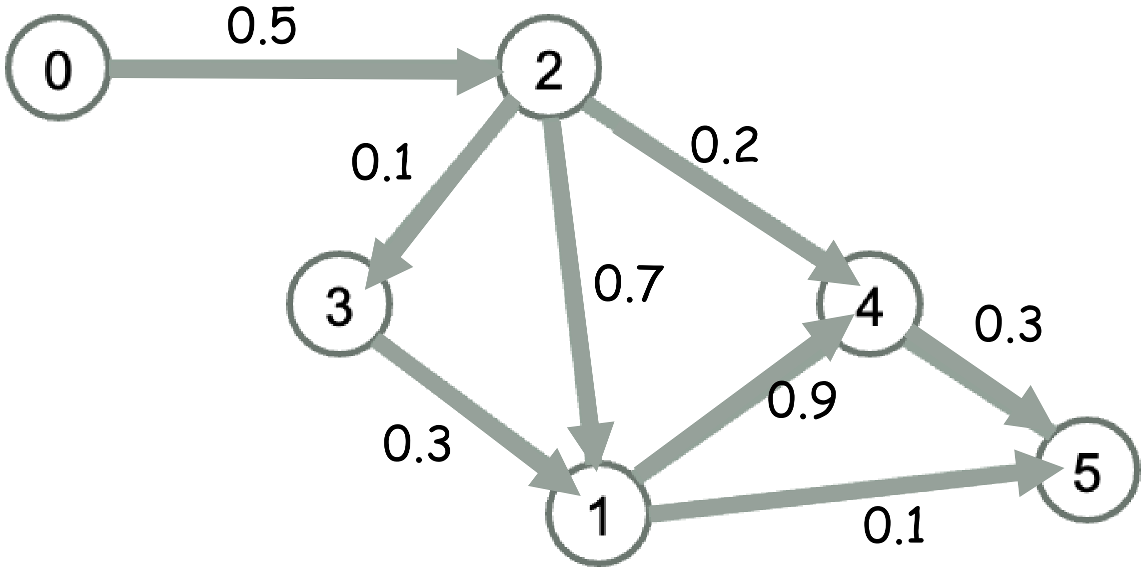


What if edges had weights?

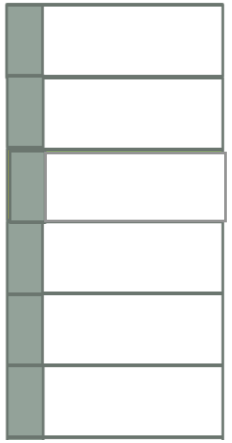
```
class graph{
  ...
  private:
   vector<NodeInfo*> nodes;
   ______________________ adjlist;
};
```

How would you represent adjlist for a weighted graph?

// Use when edges have no weights
vector<unordered_set<int>> adjlist;

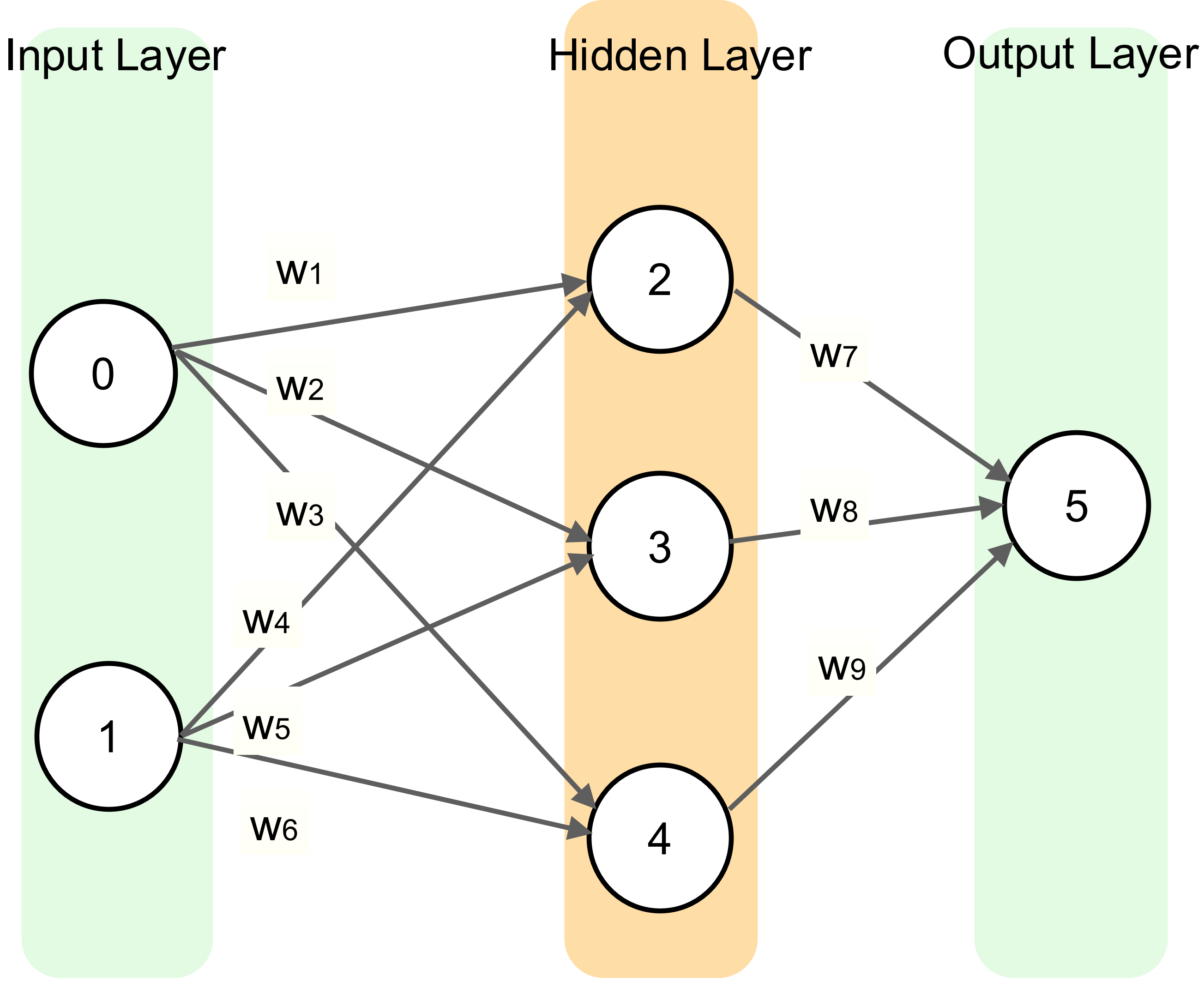


adjlist



Neural Network structure for upcoming assignment

```
typedef std::vector<std::unordered_map<int, Connection> > AdjList;
```




```
class Graph {
  std::vector<NodeInfo*> nodes;
  AdjList adjacencyList;
};
```

Understanding the Graph and NeuralNetwork classes

```
typedef std::vector<std::unordered_map<int, Connection> > AdjList;
```

```
class Graph {

  public:
    Graph();
    Graph(int size);
    // Constructors and destructor

    // TODO: graph methods
    void updateNode(int id, NodeInfo n);
    NodeInfo* getNode(int id) const;
    void updateConnection(int v, int u, double w);

  protected:
    // protected to give NeuralNetwork access

    // adjacency list containing weights for edges.
    AdjList adjacencyList;

    // vector storing node info
    std::vector<NodeInfo*> nodes;

    //Other functions
};
```

```
class NeuralNetwork : public Graph {

  public:

  // Constructors and public functions

  private:

  // each index of layers holds a vector which
    contains the id's of every node in that layer.
     std::vector<std::vector<int> > layers;

  // contains ids of input nodes
     std::vector<int> inputNodeIds;

  // contains ids of output nodes
     std::vector<int> outputNodeIds;


// since NeuralNetwork inherits from Graph, you can imagine all of the
graph members here as well...

};
```

Activity 5: Draw the final neural net and its representation in memory

```
void test_algorithm() {
  cout << "test_algorithm" << endl;
  NeuralNetwork nn(6);

  NodeInfo n0("ReLU", 0, -0.2);
  NodeInfo n1("ReLU", 0, 0.2);
  NodeInfo n2("identity", 0, 0);
  NodeInfo n3("sigmoid", 0, 0.98);
  NodeInfo n4("ReLU", 0, 0.11);
  NodeInfo n5("identity", 0, 0);

  nn.updateNode(0, n0);
  nn.updateNode(1, n1);
  nn.updateNode(2, n2);
  nn.updateNode(3, n3);
  nn.updateNode(4, n4);
  nn.updateNode(5, n5);

  nn.updateConnection(2, 1, 0.1);
  nn.updateConnection(2, 4, 0.2);
  nn.updateConnection(2, 0, 0.3);
  nn.updateConnection(5, 1, 0.4);
  nn.updateConnection(5, 4, 0.5);
  nn.updateConnection(5, 0, 0.6);
  nn.updateConnection(1, 3, 0.7);
  nn.updateConnection(4, 3, 0.8);
  nn.updateConnection(0, 3, 0.9);

  nn.setInputNodeIds({2, 5});
  nn.setOutputNodeIds({3});
```

Next lecture preclass activities

- Review pa03 tutorial: https://ucsb-cs24.github.io/s25/pa/pa03-tutorial/

- Finish watching the neural net intro video: https://youtu.be/aircAruvnKk?feature=shared

- Do the assigned reading: Breadth First Search and Depth First Search on graphs.