Neural Networks

CS4881 Artificial Intelligence

Jay Urbain

Credits:

Nazli Goharian and David Grossman, IIT

Machine Learning, Tom Mitchell

AIMA, Russell and Norvig

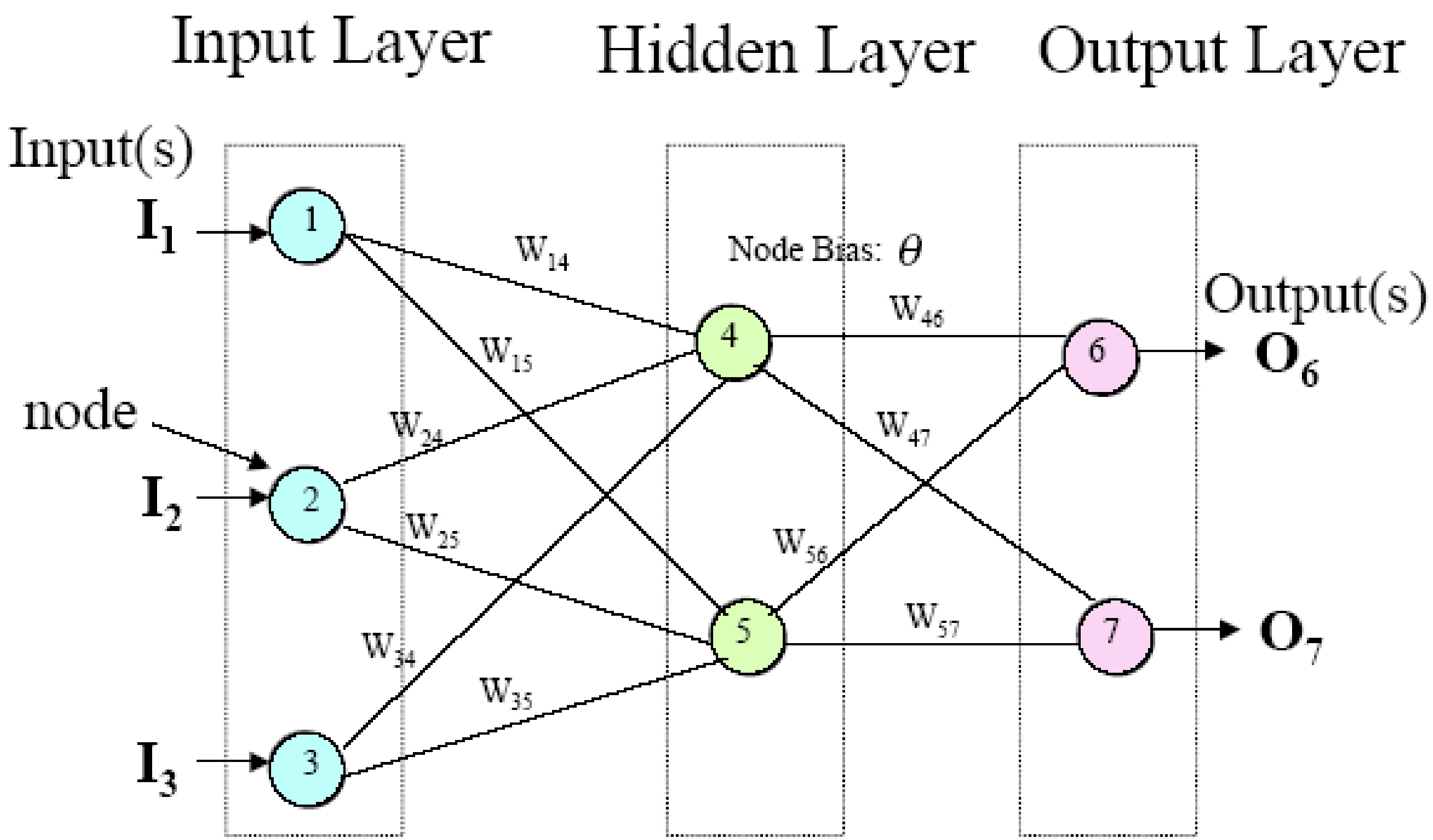
Neural Network Classification

- Loosely based on concept of neurons in the brain.
- Set of connected nodes with weights for nodes and arcs.
- Weights are initially assigned randomly.
- Typically involves a long learning process.
- Tolerance to noise data.

Neural Network Components

- Input Layer
- Hidden Layer
- Output Layer
- Connections, arcs
- Weights
- Activation Functions
- Training, test sets
- Learning algorithms

Neural Network Components



Input Layer

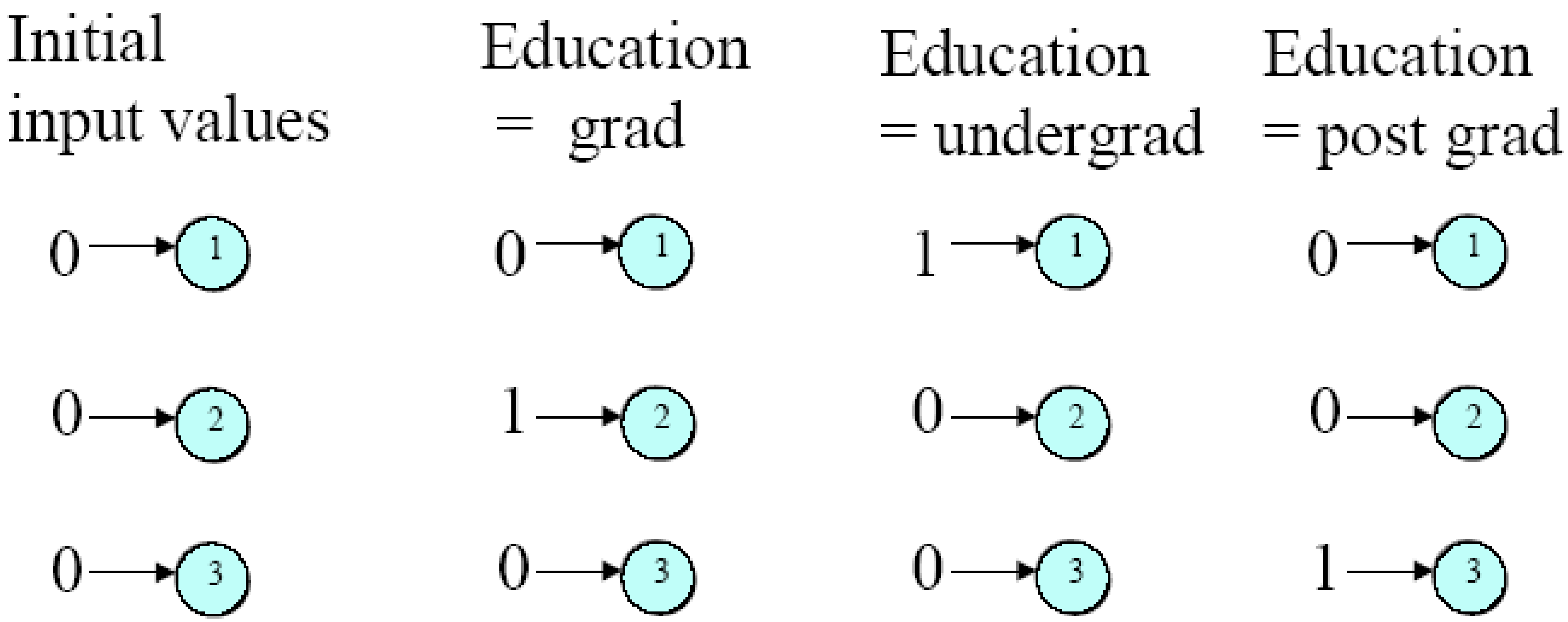
- Input layer has attributes used for classification.
- Select attributes by examining the data and utilizing domain knowledge.
- Inputs are the attribute values for each tuple:
  - {age=30, education=MS, salary=90,000}
- Input values should be normalized and discrete values are encoded.

Input Layer 2

- Number of nodes in input layer is typically defined by the number of attributes and the number of attribute types.
  - Continuous attributes like salary are typically normalized between {0,1} and fed into one node.
  - Numeric/ordinal attributes which do not have a continuous range such as age, an attribute is created for each discrete value _or_ age is transformed into a range of categories.
  - Categorical attributers (or continuous attributes transformed into categories) are first encoded and one node is created for each category.

Example input layer

Education: {undergrad, grad, post grad}



Hidden Layer

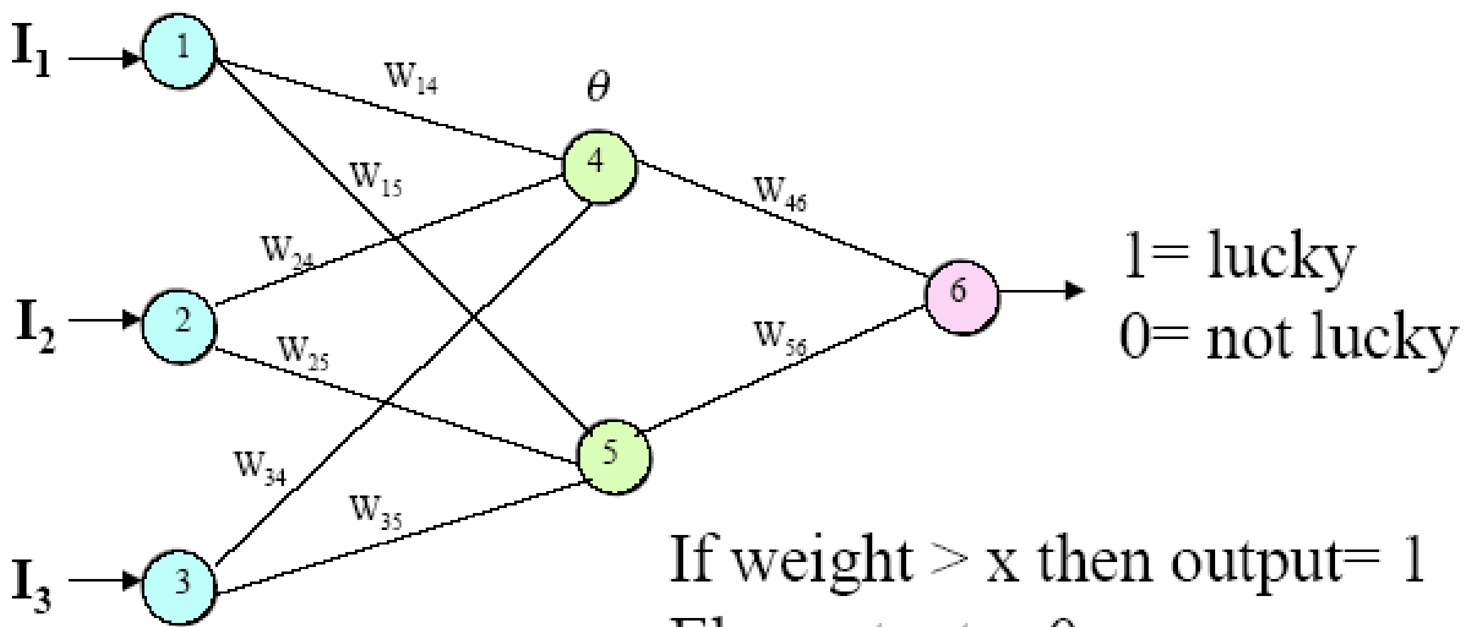
- Hidden layer allows networks to solve complex nonlinear problems.
- A network can have one or more hidden layers.
- The number of nodes in the hidden layer(s) is determined via experimentation. *~6 is a good start, or some number between #inputs & outputs.*
- Too many nodes => over-fitting
- Too few nodes => reduced classification accuracy.

Output Layer

- Result of the classification is the output of a node in the output layer.
- Weights and activation functions determine the output.
- Output layer may have one or more nodes.
- There is typically one output node for each class:
  - E.g., 3 output nodes for high-income, mid-income, and low-income classes.
  - If two classes, i.e., binary classification, e.g., {lucky, not lucky} you can use one node {1=lucky, 0=not lucky}.

Example Output Layer

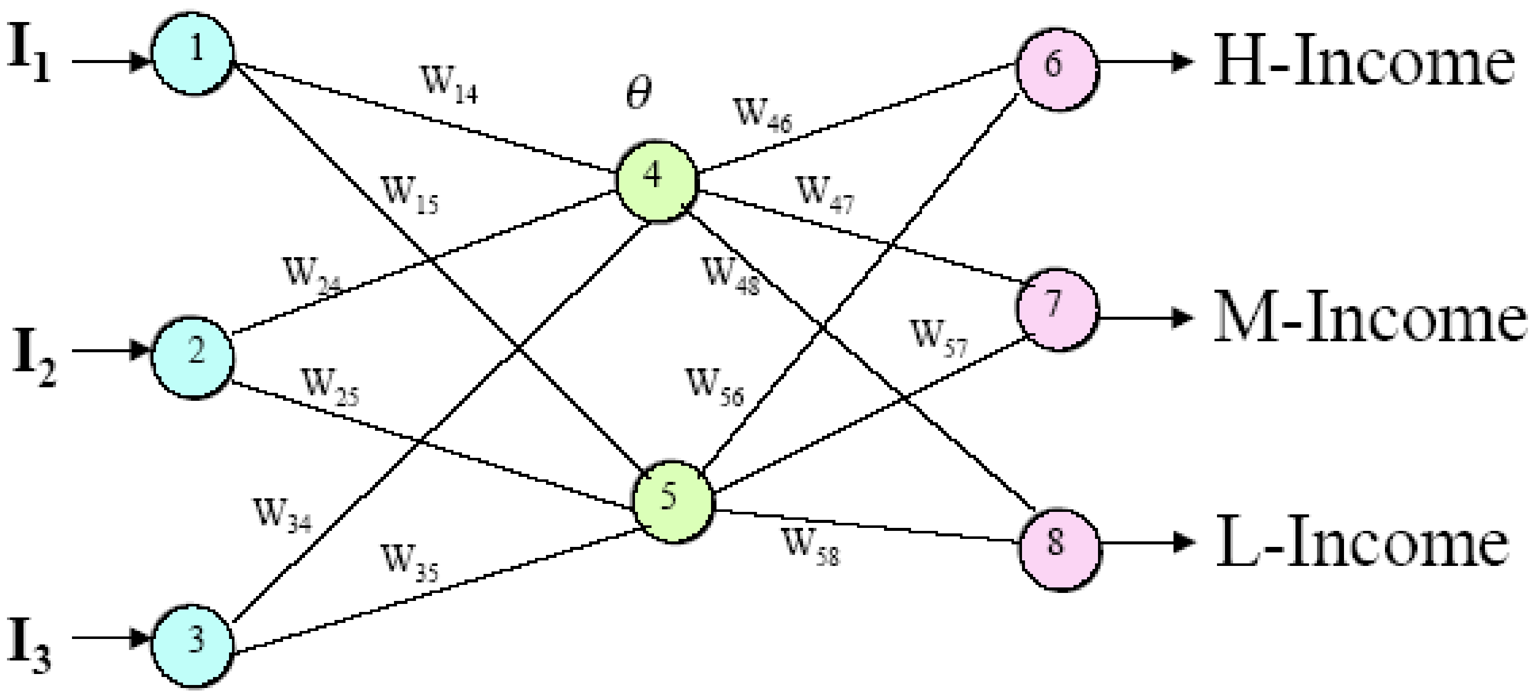
Class labels: lucky, not lucky



If weight > x then output= 1
Else output = 0

Example Output Layer 2

Class labels: H-Income, M-Income, L-Income



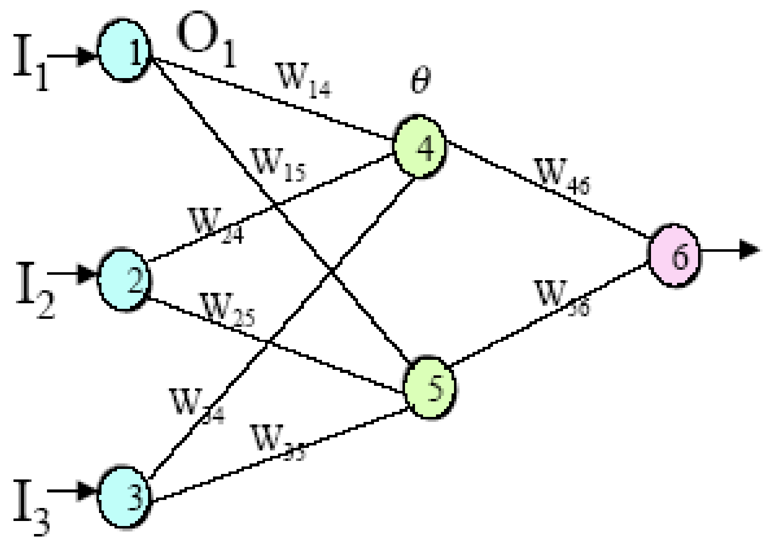
Example: Arcs and weights

In input layer: $O_i = I_i$

In hidden or output layer:

- Input to node j:

$$I_j = \left(\sum_i w_{ij} . O_i \right) + \theta_j$$



- Output from node j using *sigmoid* activation function:

$$O_j = \frac{1}{1 + e^{-I_j}}$$

Activation Function

- Different activation functions can be used, *sigmoid* is typical.

- Also called a *squashing function*
  - as it squashes the output value into a range of {0 to 1} to reduce the weak or gray area by pushing it toward one class or another.

Training a Neural Network

- Run an example from the training set, by giving its attribute values as input (normalized of course!).

- *Feed-forward* process
  - Summation of weights and activation functions are applied at each node of hidden and output layers, until an output is generated.

- *Back-propagation* process
  - If the output does not match, go back from the output layer to each hidden layer, (layer by layer) and modify the arc weights and biases of nodes.

- Eventually the weights will (*should*) converge and processing stops.

Feed-Forward Process

- Process starts from input nodes to hidden nodes:

  For each training sample *X* do

  For each hidden or output layer node j

  Calculate input I_J to that node: $I_j = \left(\sum_i w_{ij} \,.\, O_i \right) + \theta_j$

  Calculate output O_J from that node: $O_j = \dfrac{1}{1 + e^{-I_j}}$

- At this point, the final output is generated.

Back Propagation Process

- Calculate *error* and update weights

  For each node j in the output layer do

  Calculate the error:

$$Err_j = \underbrace{O_j(1-O_j)}_{\text{Derivative of squashing function}}(T - O_j)$$

Expected result

- For each node j in hidden layer (last to first)

  Calculate the error:

$$Err_j = O_j(1-O_j)\left(\sum_k Err_k \, . \, w_{jk}\right)$$

Back Propagation Process cont.

- For each weight w_{ij}

Calculate weight increment: $\Delta w_{ij} = l \, . \, Err_j \, . \, O_i$

$\underset{\text{Learning rate}}{l}$

Update weight: $w_{ij} = w_{ij} + \Delta w_{ij}$

- For each node bias

Calculate the error:

$$Err_j = O_j(1 - O_j)\left(\sum_k Err_k \, . \, w_{jk}\right)$$

Summary

- Set of connected nodes along with the weights for nodes and arcs.
- Different network topologies based on trial and error, though there has been considerable research into which topologies are optimal for different classes of problems.
- Strengths:
  - Tolerance to noise
  - Works well with complex, nonlinear problems that are difficult to characterize.
  - *Can be highly accurate.*
- Weaknesses
  - Not intuitive, difficult to extract human understandable description from learned weights
  - Can have a long learning process
  - Prone to overfitting