COMP4388: MACHINE LEARNING

Artificial Neural Networks

Dr. Radi Jarrar
Department of Computer Science
Birzeit University

Artificial Neural Networks

"a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."

Dr. Robert Hecht-Nielson, "Neural Network Primer: Part I" by Maureen Caudill, AI Expert, February. 1989

Artificial Neural Networks (2)

- In brains, neurons are cells that are connected together and can transmit electrical signals
- The human brain is made-up of around 90 billion neurons
- The set of connected neurons form-up the neural network
- In the human body, these are tens of billions of interconnected neurons

Artificial Neural Networks (3)

- Artificial neural networks have many applications:
  - Google uses them for their Data Centre Management
  - It is used to adjust the settings and cooling equipment in the data center
  - Robotics: they have been used in pattern recognition and sensory data to determine what move to take

Artificial Neural Networks (4)

- Medical Monitoring: medical attention can be achieved by medical machinery that involves constant updating of many variables such as heart-rate, blood pressure, breathing rate, sugar-level, etc.
- Sophisticated models of weather and climate, fluid dynamics, and many other complex tasks

Artificial Neural Networks (5)

- ANN models the relationship between an input signals and an output signal by using a model that is based on the understanding of how the brain responds to stimuli from a sensory inputs
- ANN uses a network of nodes (artificial neurons) to tackle the learning problem
- ANN are best applied to problems in which the input and output signals are well-understood yet the mapping from input to output is complex

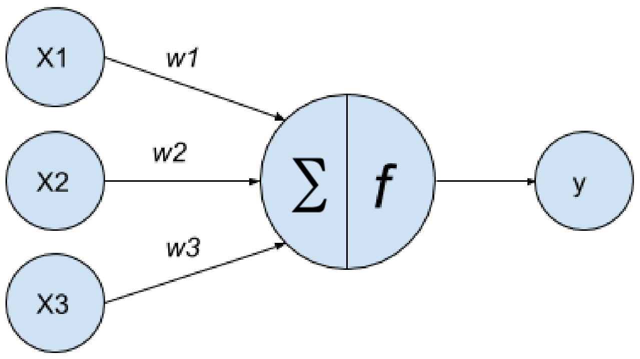
Perceptrons

- Perceptrons are the basis of the Neural Networks
- Perceptron receives an input signal and then passes a value through some functions
- Perceptrons deal with numbers (i.e., numbers or vectors of numbers) passed to it as input values
- The input value can be added to a constant or multiplied by a constant value (in the easiest implementation)

Perceptrons (2)

- In more sophisticated implementation, the input values are passed to a function to calculate the output value
- This function is called ***Activation Function***

Perceptrons (3)



Perceptrons (4)

- Neuron, unit, node…is the basic computational unit in ANN
- Receives an input and computes an output
- Each input has a weight indicating the relevance importance of the input feature

Perceptrons (5)

- The perceptron sums all the input signals within its body
- If the sum passes some threshold, then the signal is passed through activation function ***f*,** denoted by variable y=1, and if the value is below the threshold, then y=0
- A typical implementation of an artificial neuron is through the following equation

$$y(x) = f\left(\sum_{i=1}^{n} w_i x_i\right)$$

Perceptrons (6)

- Weights are given to allow input signals to contribute lesser or greater amounts to the sum of input signals
- The net total is used by the activation function f(x)
- The activation function adds a non-linearity into the output of a neuron (as most real world data is not linear and the aim to teach the neurons this non-linearity)
- The resulting signal y(x) is the output axon
- A set of similar neurons are used to build a Neural Network handling complex models

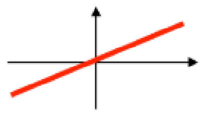
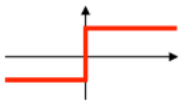
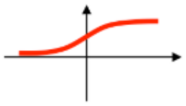
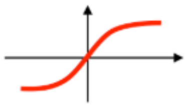
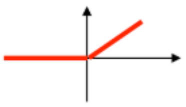
Characteristics of ANN

- The Activation Function
  - Transforms the input signals into a single output signal to be further broadcasted in the Neural Network
- Network Architecture
  - Describes the number of neurons in the network and the number of layers
- Training Algorithm
  - Describes how connection weights are set in order to stop/excite neurons in proportion to the input signal

Activation Function

- Is the processing that takes place after the input signals are passed into the neuron
- The result of the activation function decides whether the value is passed to the next neuron in the network
- A common implementation for the activation function is the **Sigmoid Function**
- Think of Logistic Sigmoid: the output isn't binary; any value between 0 and 1

Activation Function (2)

• $f(x) = x$	Linear	
• $f(x) = \begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases}$	Binary step	
• $f(x) = \frac{1}{1+e^{-x}}$	Logistic	
• $f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	Hyperbolic Tangent	
• $f(x) = \max(0, x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases}$	Rectified Linear Unit (ReLU)	

Activation Function (3)

- Step function
  - Works well for two classes. However, adding more classes will cause problems
  - Translates the input ranged in [-inf, +inf] to [0, 1]
- Linear function
  - The activation is proportional to the input
  - The weighted sum from neuron
  - It gives a range of activations instead of a binary activation

Activation Function (4)

- Sigmoid function
  - Non-linear
  - Smooth step function
  - The output of this function is always in the range (0, 1) unlike linear (-inf, +inf)
  - Small change in the value of X causes the value of Y to change significantly. Meaning that this function has the tendency to bring the value of Y to the ends of the curve
  - Issues: towards the end of the curve, small changes in X won't reflect much on Y (the gradient at that region is very small). This means that the network will refuse to learn more or it becomes very small

Activation Function (5)

- Tanh function
  - Non-linear
  - Similar characteristics with the sigmoid functions
  - Maps X to the rangs of (-1, 1)
  - Very popular in ANN
  - It has the same problem of slow gradient as in sigmoid function

Activation Function (6)

- ReLU function
  - Non-linear
  - X if X is positive, zero otherwise
  - It will map input X to a value in [0, +inf)
- When using Sigmoid or Tanh functions, almost all neurons will activate and send a signal forward (i.e., dense activation). This is costly. ReLU overcomes this issue.
  - Assume a network with random weights and around 50% of the network yields 0 activation (as ReLU outputs 0 for –ve values of X). This would result in fewer neurons are firing (sparse activation), thus a faster network.

Activation Function (7)

- Still an issue when the gradient goes towards zero (i.e., weights will not get adjusted during descent and neurons that go into this state will not respond to variations in input as gradients is zero, then nothing will change
- Computationally faster than Sigmoid and Tanh, thus used in Deep Learning

Network Architecture

- The capacity of a neural network is decided by its architecture:
  - The number of layers
  - Number of nodes in each layer
  - Ability of information to travel backward

Network Architecture (2)

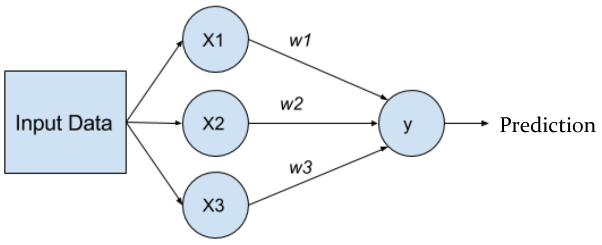
- These properties decide the complexity of the ANN
- The more the complex the ANN, the more the ability for the ANN to identify more complex decision boundaries

Number of Layers

- Typically, there are three types of layers:
- Input layer: contains nodes that receive the features (unprocessed signal)
- Hidden layer: receives signals from the previous layer and using activation functions passes a signal to the next layer
- Output layer: outputs the final prediction based on some activation function
- These nodes are arranged within layers in the neural network

Number of Layers

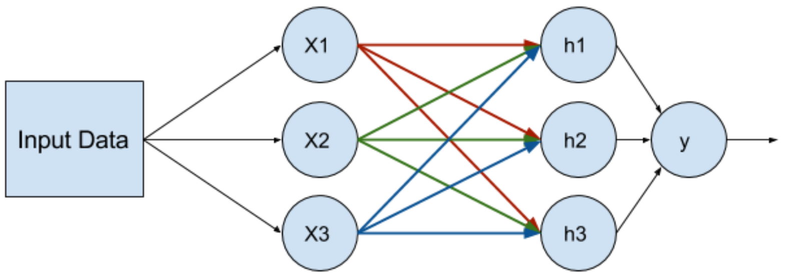
- The simplest form of a network is the single-layer network



Number of Layers (2)

- This form of networks (Single-layered) can be applied to simple pattern classification problem in which the patterns are linearly separable
- If the pattern is not linearly separable, more complex networks are required to solve such cases
- This is achieved by adding more layers (hidden) between the input and output layers

Number of Layers (3)



Number of Layers (4)

- A multilayer network contains one or more hidden layers that process signals between the input nodes and the output node
- Multilayer networks are fully connected (Dense networks) in most case
- Prior to deep learning, a neural network that contains more than two hidden layers are rare
- In most cases though, single hidden layer is enough

Number of Layers (5)

- No hidden layer: linear separable function between input & output
- A single hidden layer: approximates a continuous value from one finite space to another
- 2 hidden layers: can represent more complex decision boundary
- More than 2 hidden layers would represent more complex representations (sort of automatic feature engineering)

Number of Nodes in Each Layer

- The number of input nodes is predetermined by the number of features in the input data
- The number of output nodes is predetermines by the number of outcomes (i.e., the number class labels)
- The number of nodes in the hidden layers is left to be decided by the user once the network is designed!
  - Should be between the size of the input layer and the size of the output layer

Number of Nodes in Each Layer (2)

- The appropriate number of neurons depend on the number of input nodes (i.e., number of features), number of training data, complexity of learning task, and many other factors
- A greater number of neurons will result in a model that can solve more complex problems
- This has the risk of overfitting as well large and complex networks are computationally expensive and slow to train

Number of Nodes in Each Layer (3)

- A good practice is to use the fewest number of nodes that result in a good performance on the validation test
- Mostly, a few number of hidden nodes often results in a neural network with a great amount of learning ability

Back propagation

- The weights on the network connections reflect the patterns observed over time
- The greater the weight, the more patterns observed for a specific feature and vise versa
- These are learnt through the learning stage from the observed data
- Adjusting the weights is computationally expensive

Back propagation (2)

- Back propagation: "Backward Propagation of Errors"; is an algorithm to calculate the gradients and to map the correct inputs to the correct outputs
- Consists of two steps in iterative scheme: propagation phase & updating of weights
- Each iteration of the algorithm over the entire training set is called epoch
- The weights are set to randomly prior to beginning

Back propagation (3)

- The algorithm will run until a stopping criterion is reached
- The Forward Phase: the neurons are activated from input layer to the output layer
- The neurons' weights and activation functions are set along the way
- The Backward Phase: the network's output signal is compared to the true target value in the training phase

Back propagation (4)

- The difference between the predictions (the error) is propagated backwards in the network to modify the connection weights between neurons and reduce future error
- The network uses the information sent backward to reduce the total error of the network

Back propagation (5)

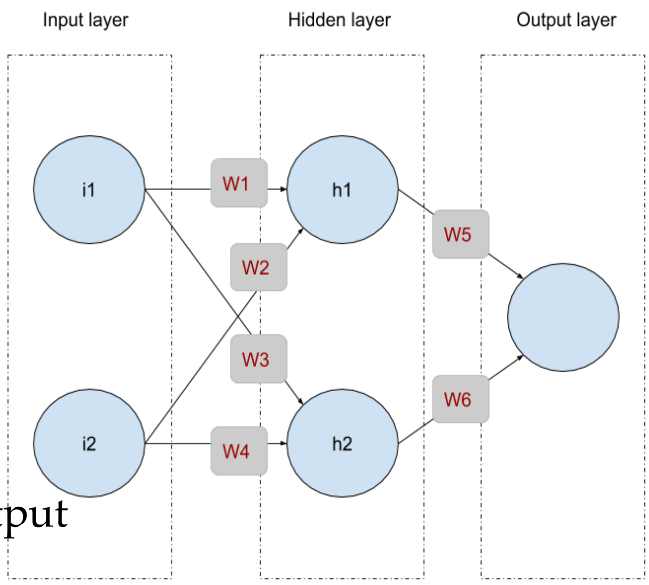
- This algorithm decides on whether a weight should be updated or not by using the **Gradient Descent** algorithm
- The gradient suggests how steeply the error will be reduced or increased for a change of weights

Back propagation (6)

- The algorithm will change the weights that result in the greater reduction in error by using the learning rate
- The greater the learning rate, the larger the learning rate, the faster the algorithm to descent down which results in a faster training time BUT with the risk of overshooting the minimum value!

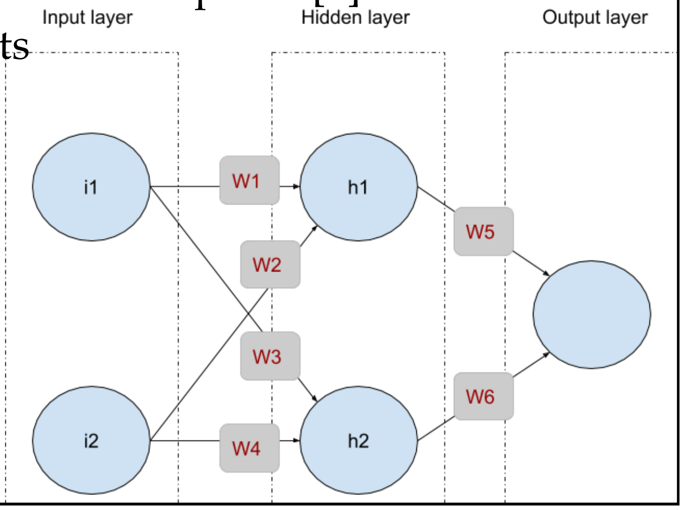
Example

- NN training is about finding weights that minimize the prediction error
- Suppose this is the network
- The training starts with randomly generated networks
- Later, backpropagation updates the weights to to correctly map input to output



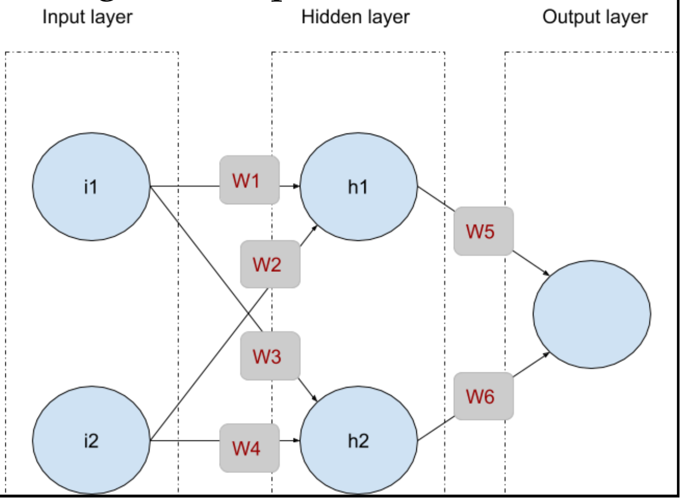
Example (2)

- Assume the input is [3, 4] and the output is [1]
- Assume the random weights are: w1=0.11, w2=0.21, w3=0.12, w4=0.08, w5=0.14, and w6=0.15



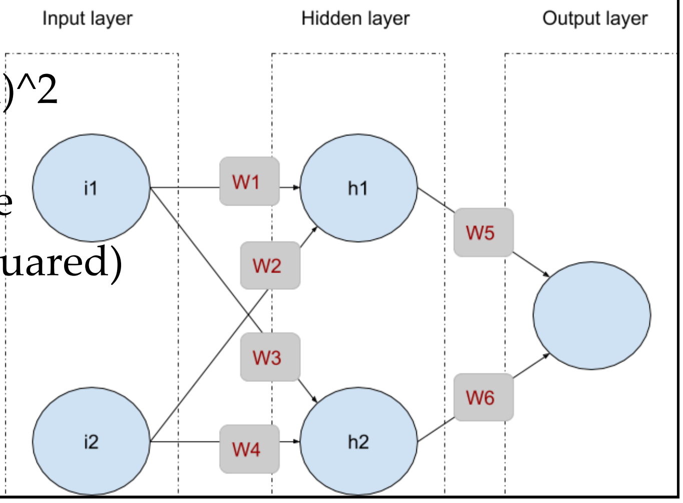
Example — Pass forward

- Inputs are multiplied by the weights and passed forward
- 2*0.11+3*0.21=0.85
  2*0.12+3*0.08=0.48
- 0.85*0.14+0.48*0.15 = 0.191



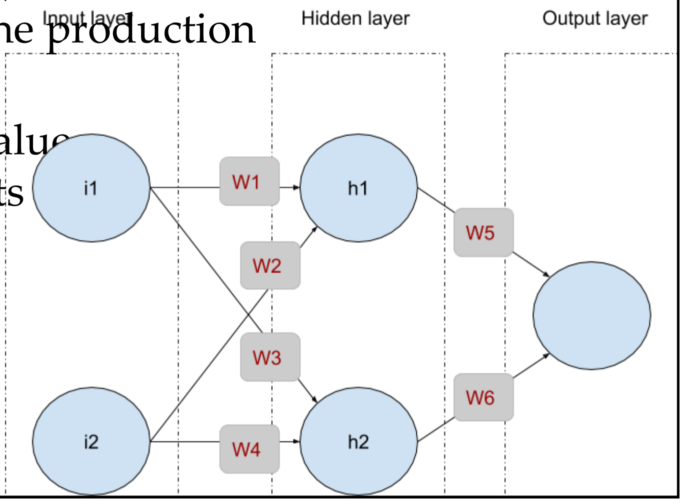
Example — Calculating errors

- Calculate the error as the difference between the actual output and the prediction
- Error=1/2(prediction-actual)^2
- 1/2 is added to ease the calculation of the derivative
- Error is always positive (squared)
- Error here is 0.327



Example — Reducing errors

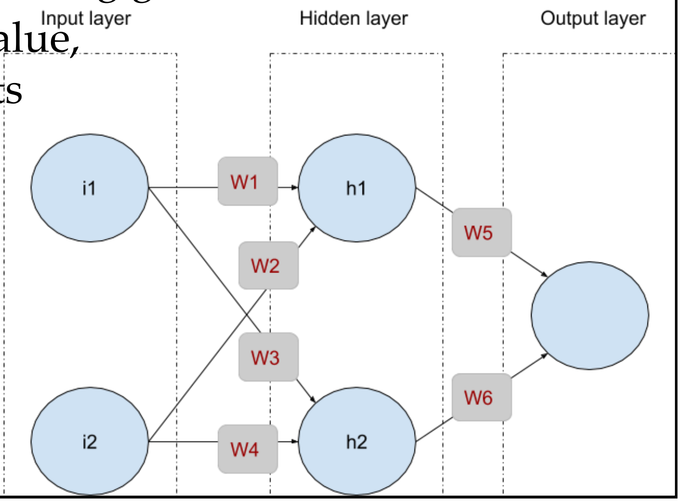
- The goal is to reduce error (the difference between actual and the prediction). Only the prediction can change
- To change the prediction value once can change the weights



Example — Backpropagation

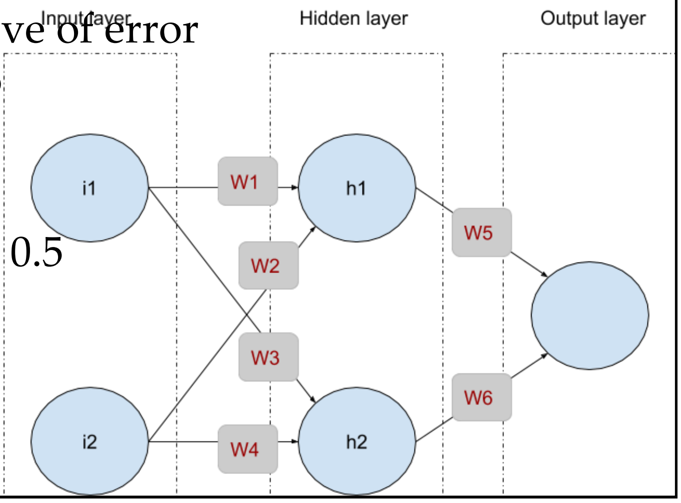
- Used to update the weights using gradient descent
- To change the prediction value, once can change the weights
- f(s)

$$\underset{\text{New weight}}{{}^{*}W_x} = \overset{\text{old weight}}{W_x} - \underset{\text{learning rate}}{a}\ \overset{\text{Derivative of error with respect to weight}}{\left(\frac{\partial Error}{\partial W_x}\right)}$$



Example — Backpropagation

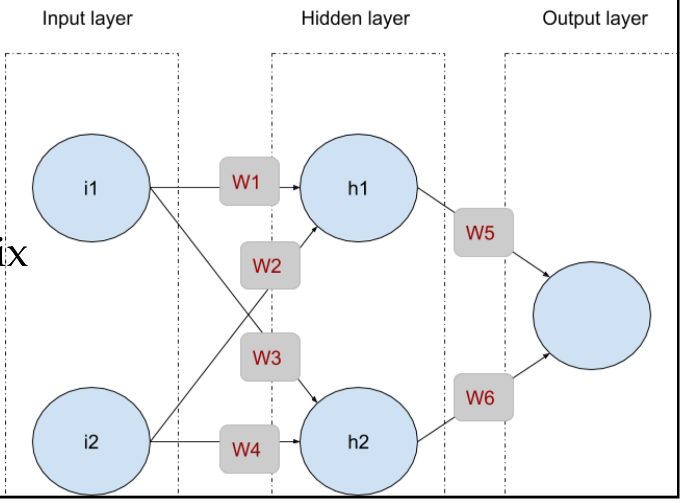
- For example, to update w6, take the current w6 and subtract the partial derivative of error function with respect to w6
- Δ = prediction - actual
  = 0.191 - 1 = -.809
- Suppose the learning rate = 0.5
- Then
  W`6=W6 - aΔh2
  W`5=W5 - aΔh2



Example — Backpropagation

- W`4=W4 - a(i2 . ΔW6)
  W`3=W3 - a(i1 . ΔW6)
  W`2=W2 - a(i2 . ΔW5)
  W`1=W1 - a(i1 . ΔW5)

- or it can be written as matrix multiplication as follows:



Example — Backpropagation

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - a\,\Delta \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - \begin{bmatrix} ah_1\Delta \\ ah_2\Delta \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - a\,\Delta \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} . \begin{bmatrix} w_5 & w_6 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - \begin{bmatrix} a\, i_1 \Delta w_5 & a\, i_1 \Delta w_6 \\ a\, i_2 \Delta w_5 & a\, i_2 \Delta w_6 \end{bmatrix}$$

- Assume learning rate = 0.5, Δ = 0.191-1 = -.809

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - 0.05(-0.809) \begin{bmatrix} 0.85 \\ 0.48 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - \begin{bmatrix} -0.034 \\ -0.019 \end{bmatrix} = \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix}$$

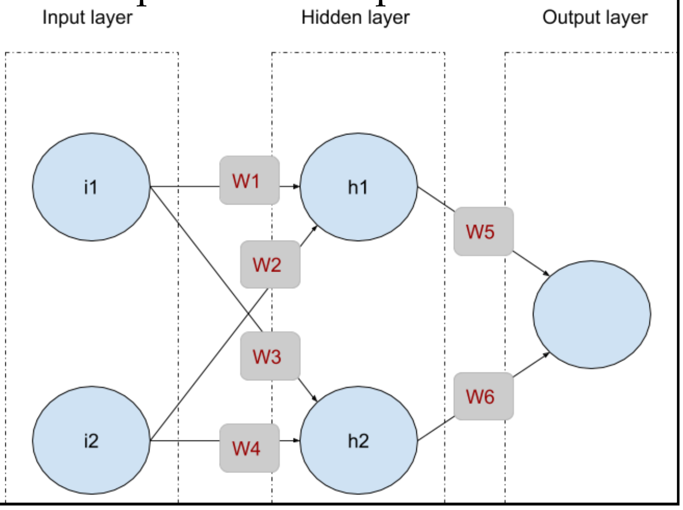
$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - 0.05(-0.809) \begin{bmatrix} 2 \\ 3 \end{bmatrix} . \begin{bmatrix} 0.14 & 0.15 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - \begin{bmatrix} -0.011 & -0.012 \\ -0.017 & -0.018 \end{bmatrix} = \begin{bmatrix} .12 & .13 \\ .23 & .10 \end{bmatrix}$$

https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Example — Backpropagation

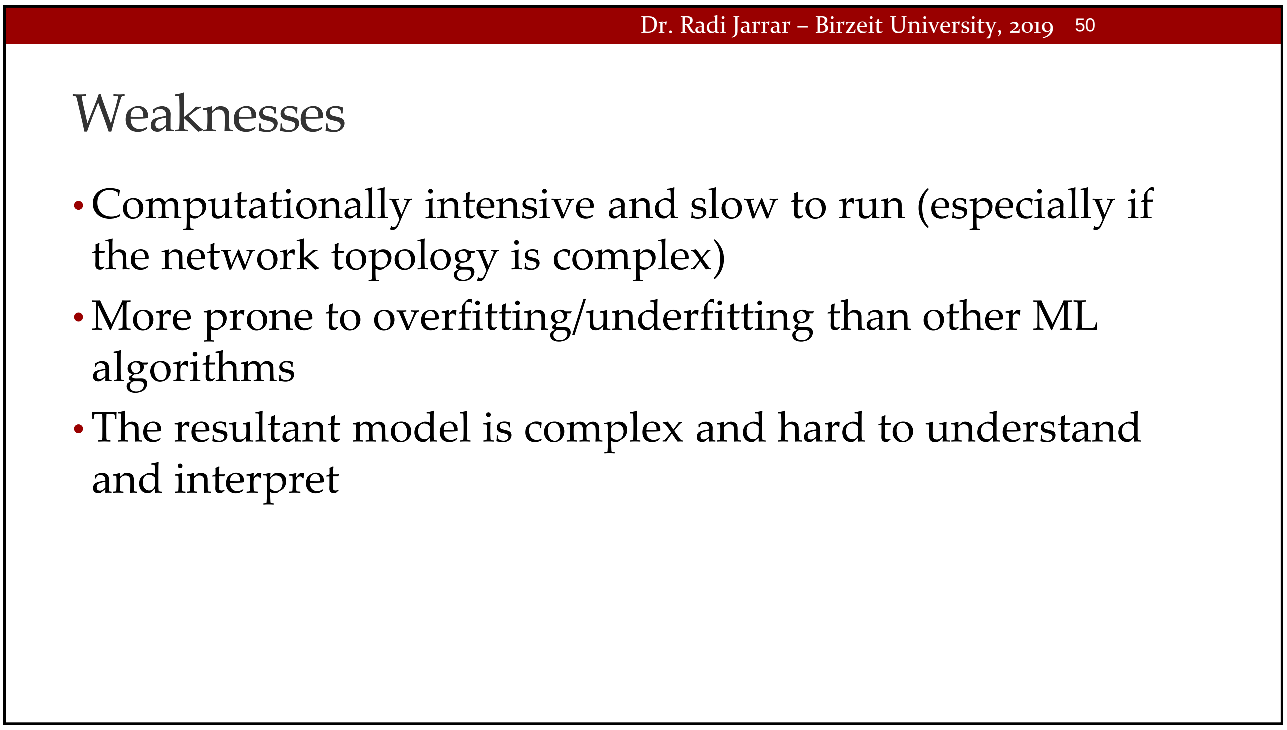
- Once updated the weights, we can repeat forward pass:
  2*0.12 + 3*0.23 = 0.92
  2*0.13 + 3*0.10 = 0.56

  0.92*0.17 + 0.56*0.17 = 0.26
- The error now is 0.273
- Repeat the same process of back/forward pass until the error is close to or equal zero



Strengths

- One of the most accurate modelling approaches
- Can be used for classification and regression problems
- Doesn’t make much assumptions about the underlying relations between input data



Weaknesses

- Computationally intensive and slow to run (especially if the network topology is complex)
- More prone to overfitting/underfitting than other ML algorithms
- The resultant model is complex and hard to understand and interpret