Deep Learning
Part 1: Neural Networks

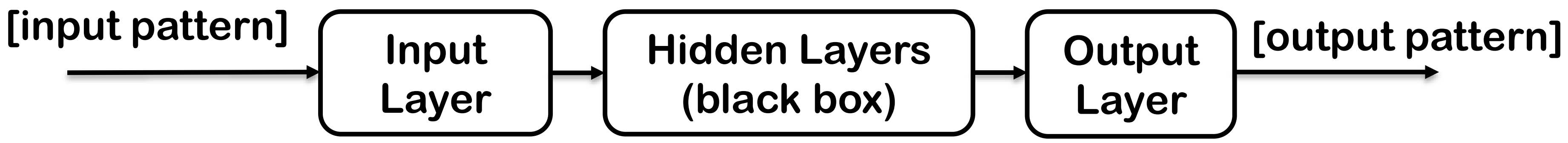
Nagiza F. Samatova, samatova@csc.ncsu.edu
Professor, Department of Computer Science
North Carolina State University

Senior Scientist, Computer Science & Mathematics Division
Oak Ridge National Laboratory

Outline

- Typical Neural Network (NN): Core Components
- Neural Network Roadmap by Problem Domains
- Neural Network Basics
  - Neurons and Layers
  - Neuron Types
  - Activation Functions
  - Logic Gates
- Boltzmann Machines and Restricted Boltzmann Machines
- Feedforward Neural Networks
  - Classification
  - Regression
  - Network Layers
  - Normalization

A Typical Neural Network (NN)



- NN accepts a pattern and returns a pattern it recognizes
- NN consists of layers of similar neurons / units
- Most NNs have input and output layers
- Hidden layers: black box that applies transformations on data
- NN's operate ***synchronously***:
    - it will only output when it has input
    - unlike human brain that responds to input but produces output any time it feels like it!

NN as a Hash Table

- NN acts somewhat like a **dictionary** that maps input patterns (i.e. key) to the output patterns (i.e., value) it recognizes
    - "hear" → "to perceive or apprehend by ear"
    - "run" → "to go faster than a walk"
- Typically, **input and output patterns** are **vectors of floating-point or binary numbers**
    - NN Input: [-0.25, 0.28, 0.0]
    - NN Output: [0.78, 0.54]
- Important question about NN:
    - How to translate a pattern recognition problem (e.g., over text, images, speech) into a fixed-length array of floating-point numbers?

XOR NN as a Hash Table

For an XOR to be TRUE, both of the sides must be different from each other

- False XOR False = False
- True XOR False = True
- False XOR True = True
- True XOR True = False

$\downarrow$

Input and ideal expected output for the NN (as a hash table example)

- **[0.0 , 0.0] → [0.0]**
- **[1.0 , 0.0] → [1.0]**
- **[0.0 , 1.0] → [1.0]**
- **[1.0 , 1.0] → [0.0]**

A Neural Network Roadmap
NN Types & Problem Domains

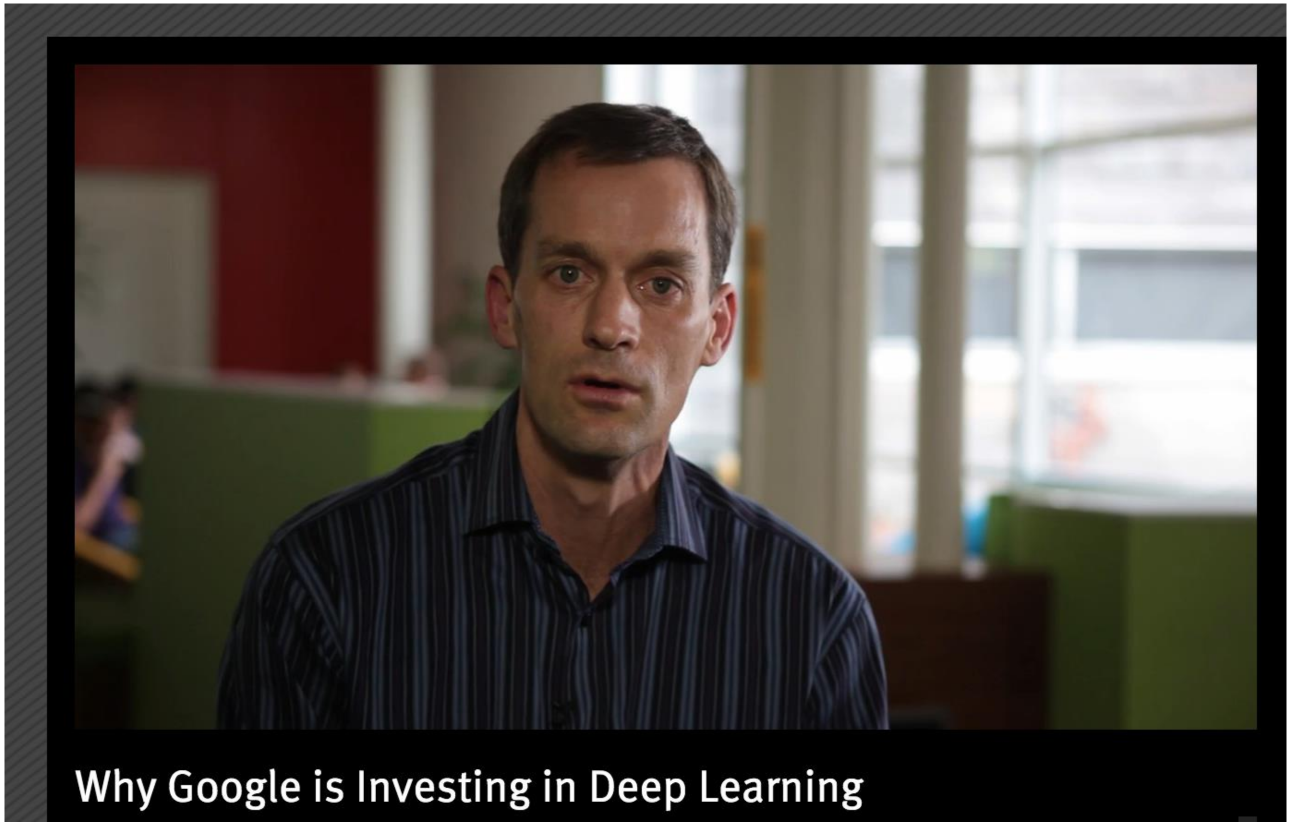
	Clustering	Regression	Classification	TS Forecast	Robotics	Vision	Optimization
Self-organizing Map	XXX				X	X	
Feedforward		XXX	XXX	XX	XX	XX	
Boltzmann Machine			X				XX
Deep Belief Network			XXX		XX	XX	
Convolutional Network		X	XXX		XXX	XXX	
Recurrent Network		XX	XX	XXX	XX	X	

- Robotics: using sensors and motor controls
- Vision: understanding images
- Optimization: e.g., best ordering or set of values to achieve objective

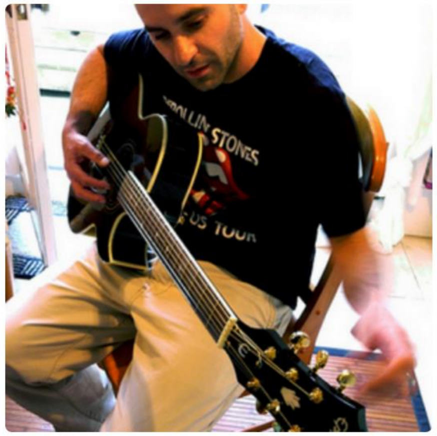
Why Google Is Investing in Deep Learning

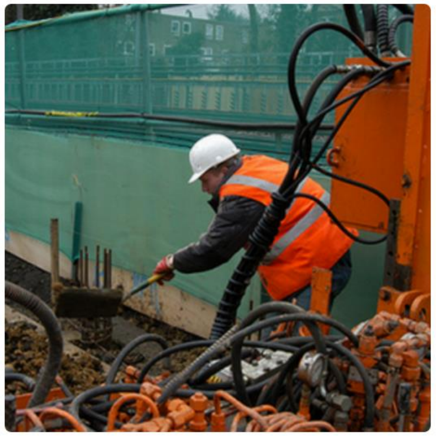
Because:

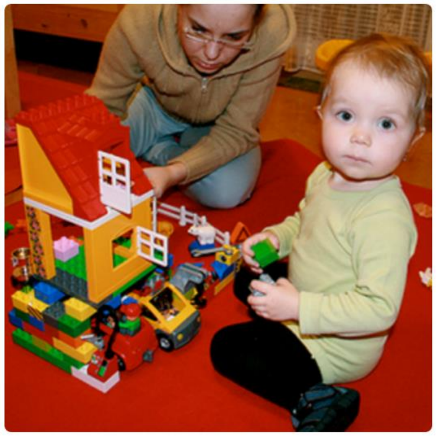
- It's self-powered
- Apply to almost anything

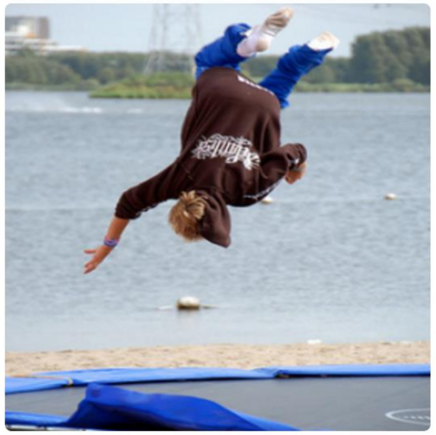


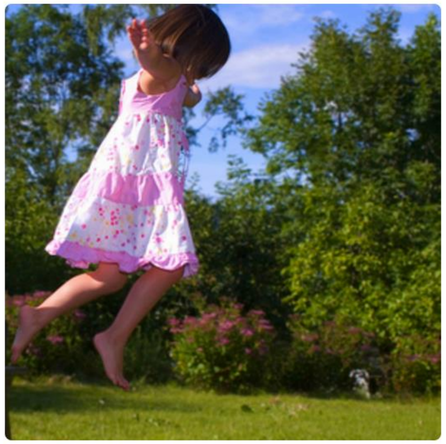
Deep Visual-Semantic Alignments for Generating Image Descriptions

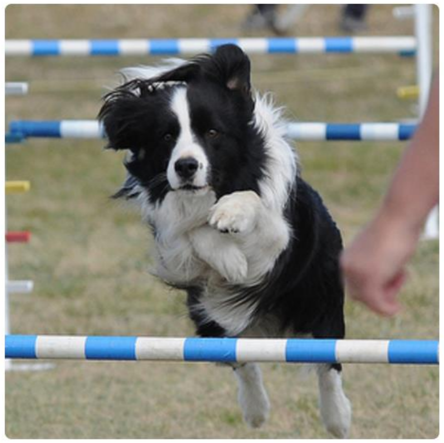

"man in black shirt is playing guitar."

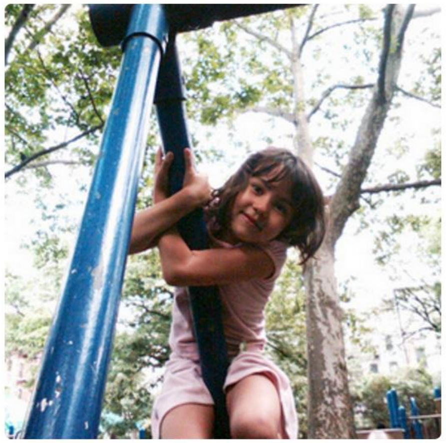

"construction worker in orange safety vest is working on road."

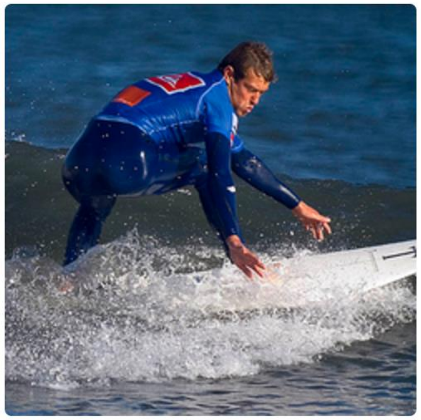

"two young girls are playing with lego toy."


"boy is doing backflip on wakeboard."


"girl in pink dress is jumping in air."


"black and white dog jumps over bar."


"young girl in pink shirt is swinging on swing."


"man in blue wetsuit is surfing on wave."

Karpathy, A., & Fei-Fei, L. (2014). Deep visual-semantic alignments for generating image descriptions. arXiv:1412.2306.

Zero to expert in 8 hours: These robots harness deep learning

Hamilton Spectator
By Pavel Alpeyev

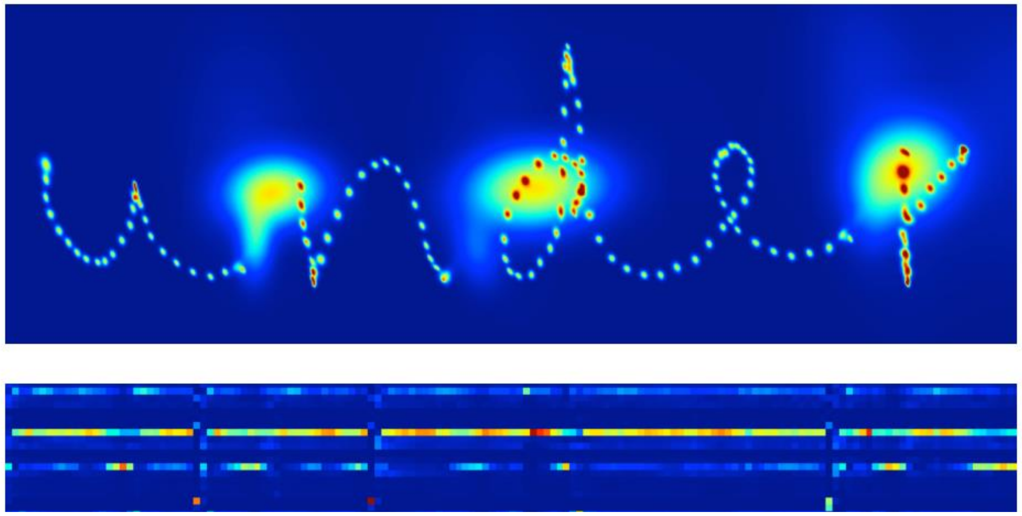
TOKYO — A yellow robotic arm pauses over a pile of metal cylinders, snaps a photo, then proceeds to confidently pick pieces out of the jumble. What's impressive is that just eight hours ago, its bin-picking skills were about zero.

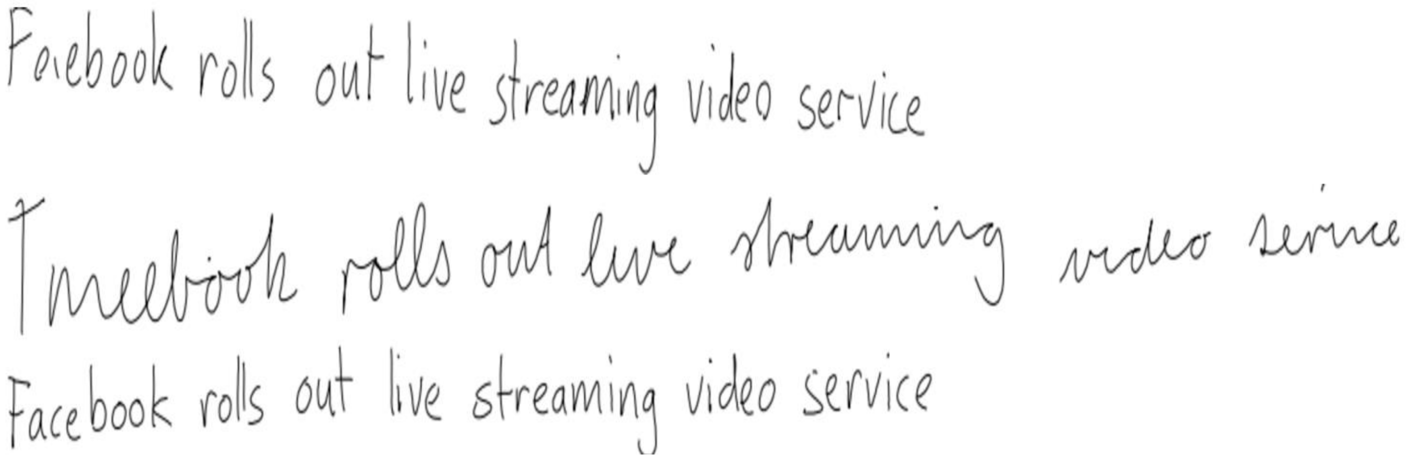
Fanuc Corp. is showing off the first results of its partnership with artificial intelligence startup Preferred Networks Inc. at Tokyo's International Robot Exhibition this week. The world's largest maker of automation equipment is using so-called "deep learning" to enable machines that can acquire skills independently.

Left overnight, a robot empowered by the algorithms can use trial and error to figure out how to pick up randomly positioned objects with 90 percent accuracy. Eight machines working simultaneously and sharing their lessons can do that in an hour. A veteran Fanuc engineer would need several days to write a teaching program that even approaches that performance.

FANUC ROBOT
Tomohiro Ohsumi,Bloomberg/Washington Post
A Fanuc Corp. industrial robot moves a vehicle during a demonstration at the International Robot Exhibition 2015 in Tokyo, Japan, on Wednesday, Dec. 2, 2015. The expo runs through Dec. 5 under the motto "making a future with robots."

Recurrent Neural Networks based on Hand Writing Generation*




Facebook rolls out live streaming video service
Facebook rolls out live streaming video service
Facebook rolls out live streaming video service

Graves, A. (2013). Generating sequences with recurrent neural networks. arXiv:1308.0850.

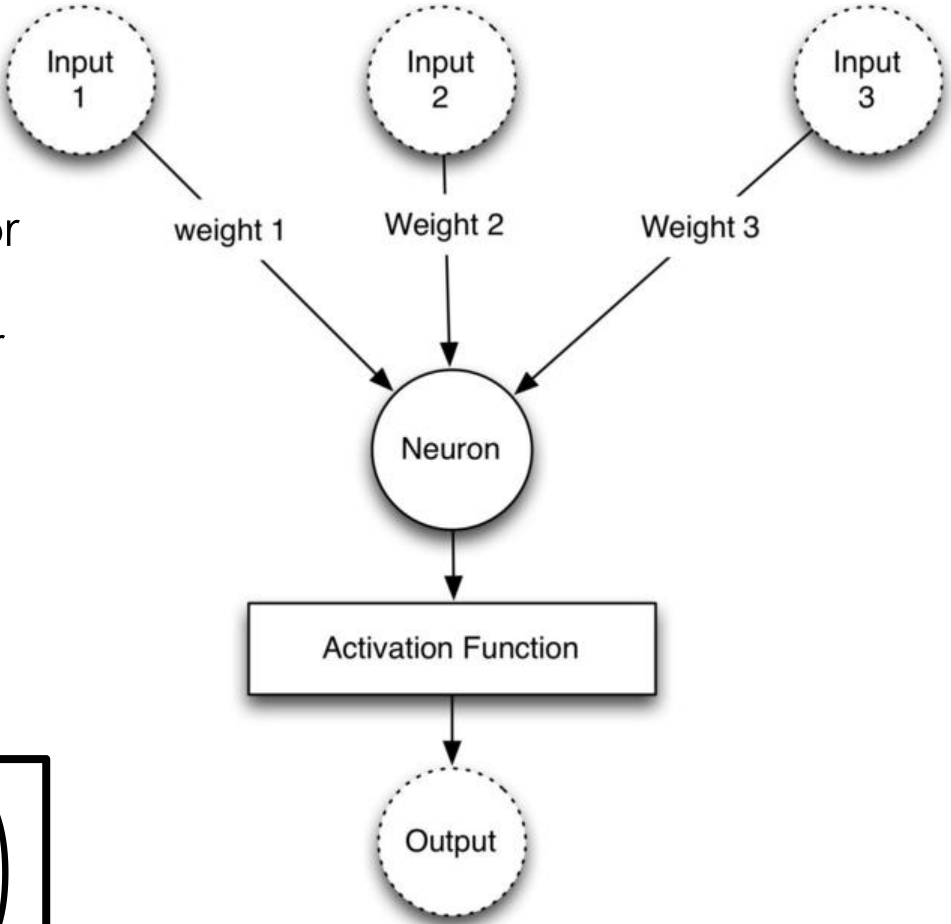
Outline

- Typical Neural Network (NN): Core Components
- Neural Network Roadmap by Problem Domains
- Neural Network Basics: Commonalities among different NNs
  - Neurons and Layers
  - Neuron Types
  - Activation Functions
  - Logic Gates
- Boltzmann Machines and Restricted Boltzmann Machines
- Feedforward Neural Networks
  - Classification
  - Regression
  - Network Layers
  - Normalization

An Artificial Neuron (= Unit, Node of a NN)

- Input: floating-point or binary vector
- Neuron receives input from one or more sources (e.g. other neurons or data fed into the network)
- Neuron multiplies each of its inputs by a weight and adds these multiplications
- Neuron passes this sum to an **activation function (AF)**:

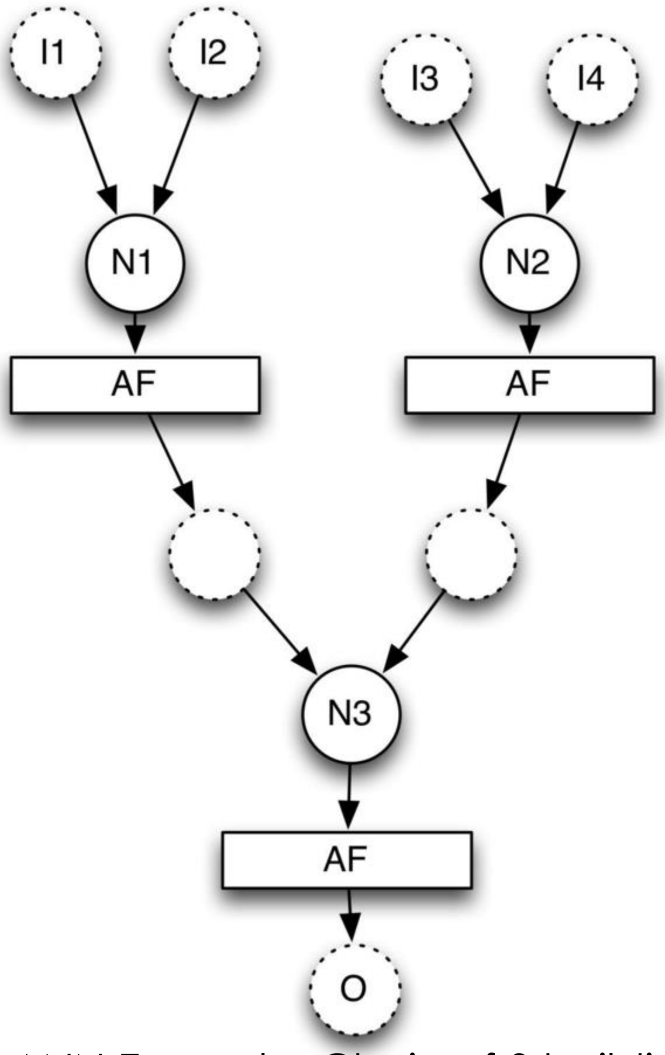
$$\boxed{f(x_i, w_i) = \phi\left(\sum_i (w_i \times x_i)\right)}$$



One building block with a single artificial neuron

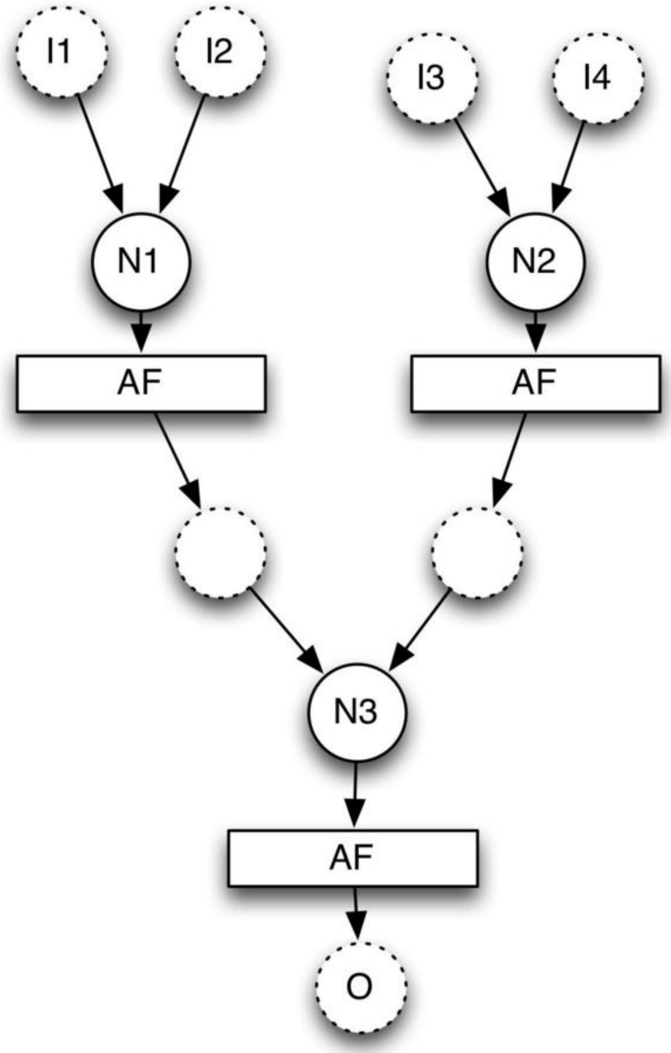
Simple Artificial Neural Network (ANN)

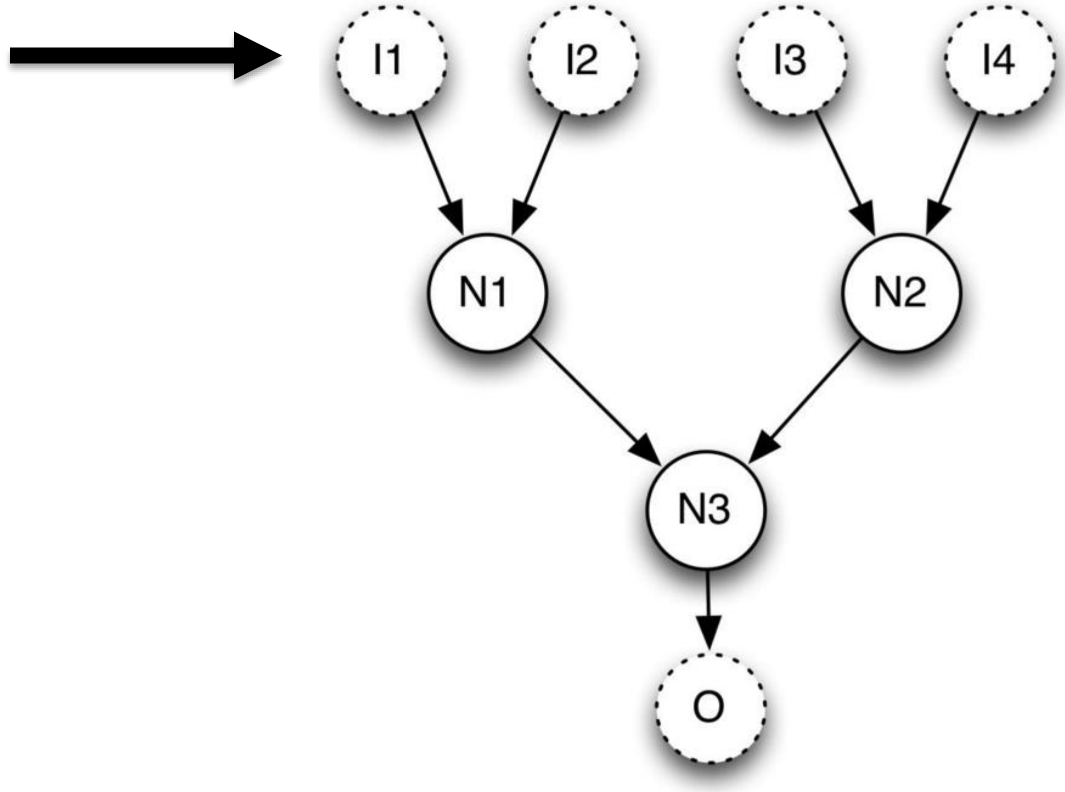
- Building blocks of an artificial neuron can be chained together to build an **artificial neural network** (**ANN**)
- Artificial neurons in ANN are building blocks for which inputs and outputs are connectors
- Note that the activation function (AF) will be called for each neuron



ANN Example: Chain of 3 building blocks of artificial neurons

Simple View of the ANN

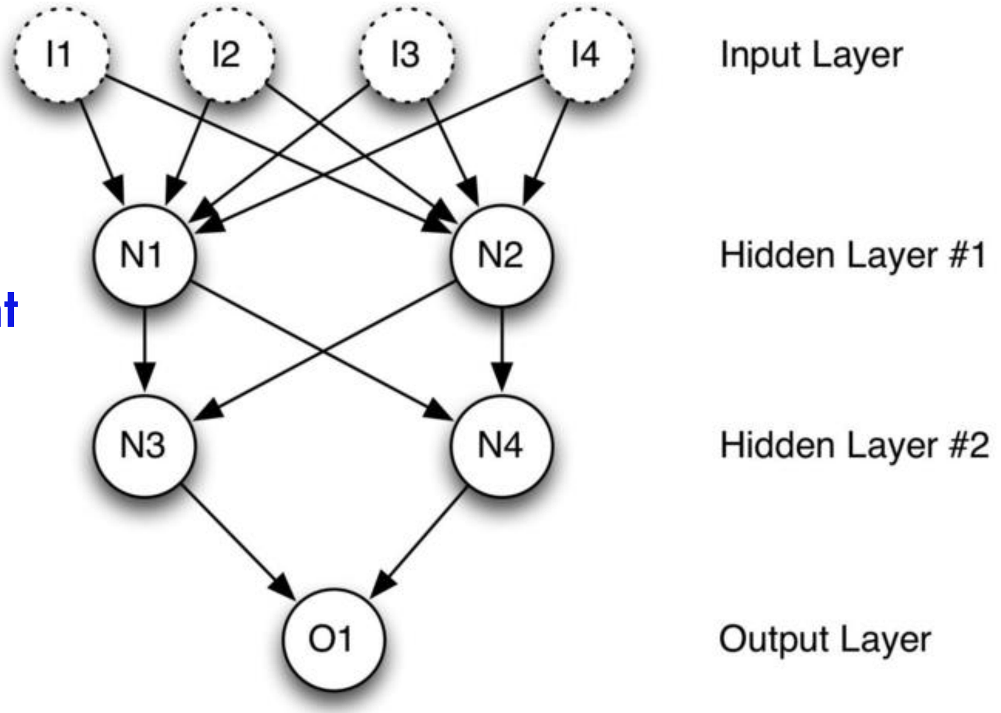




- Inputs & outputs: dotted line circles
- Neurons are arranged in **layers**
- Neurons in the same layer share the same activation function (AF)

Fully Connected Neural Network

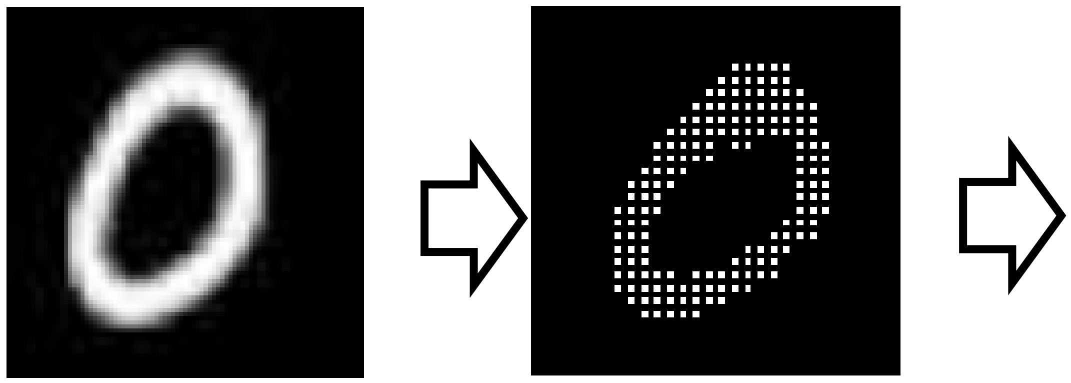
- Neurons are arranged in **layers**
- Neurons in the same layer share the same activation function (AF)
- Different layers may have **different activation functions**
- Layers are **fully connected** to the next layer: every neuron in one layer has connection to all the neurons in the previous layer



Example: **Two-hidden-layer** network

Transform a Question for a NN

Hand Written Recognition

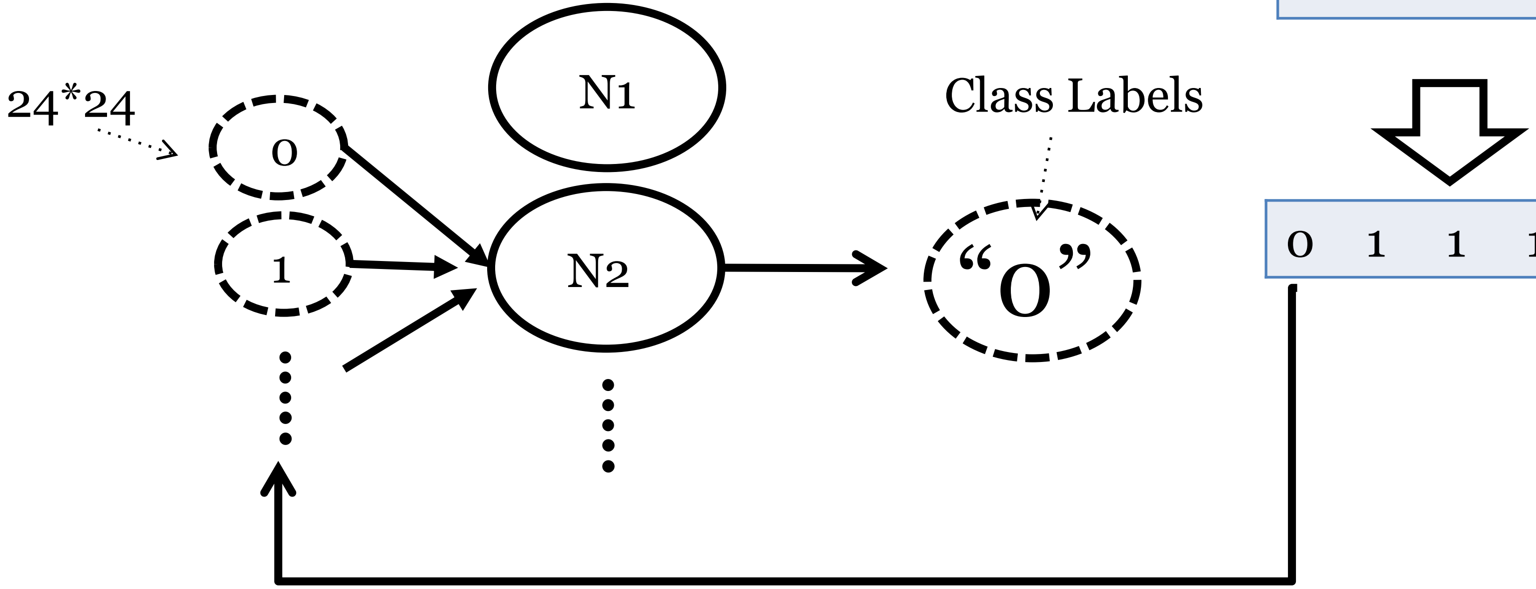


0	1	1	1	1	1	...
0	0	1	0	0	1	...
0	0	1	0	0	0	...
0	1	1	0	0	0	...
0	1	1	0	0	0	...
0	1	1	0	0	1	...
...	...	...	...	...	...	...

24*24

Flatten out

0	1	1	1	1	1	0	....



Types of Neurons

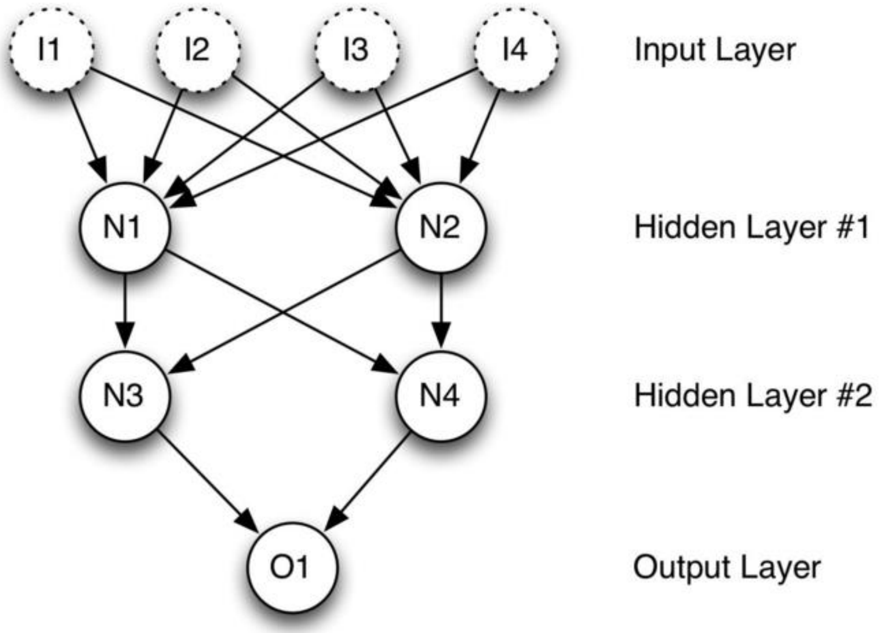
- Input and Output Neurons
- Hidden Neurons
- Bias Neurons
- Context Neurons
- Other Neuron Types (Boltzmann Machines as Complex Neurons)

Input and Output Neurons

- **Input neurons**:
  - Input to a NN: an array or vector of floating-point numbers
  - The number of input vector elements = the number of input neurons
  - Input neurons accept data from the program to the network
  - Input neurons do NOT have activation functions
- **Output neurons:**
  - Output from a NN: an array or vector of floating-point numbers
  - The number of output vector elements = the number of output neurons
  - Output neuron provides data from the network back to the program
- **Input and output layers:**
  - Input & output neurons are grouped together into separate layers

Hidden Neurons

- **Hidden neurons**:
  - Only receive input from other neurons such as input neurons or other hidden neurons
  - Hidden neurons only output to other neurons such as output neurons or other hidden neurons

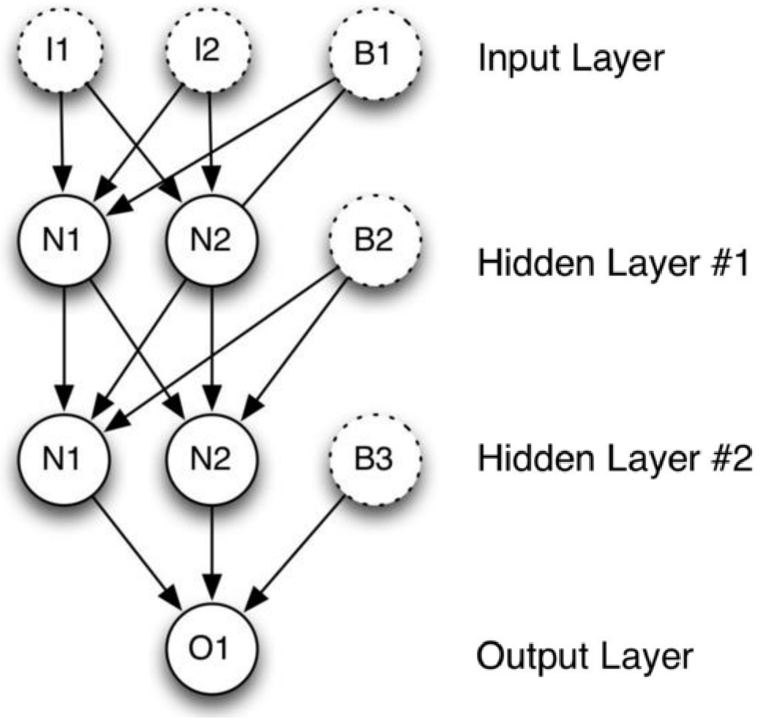


- **Hidden layers:**
  - Hidden neurons are grouped together into one or more hidden layers

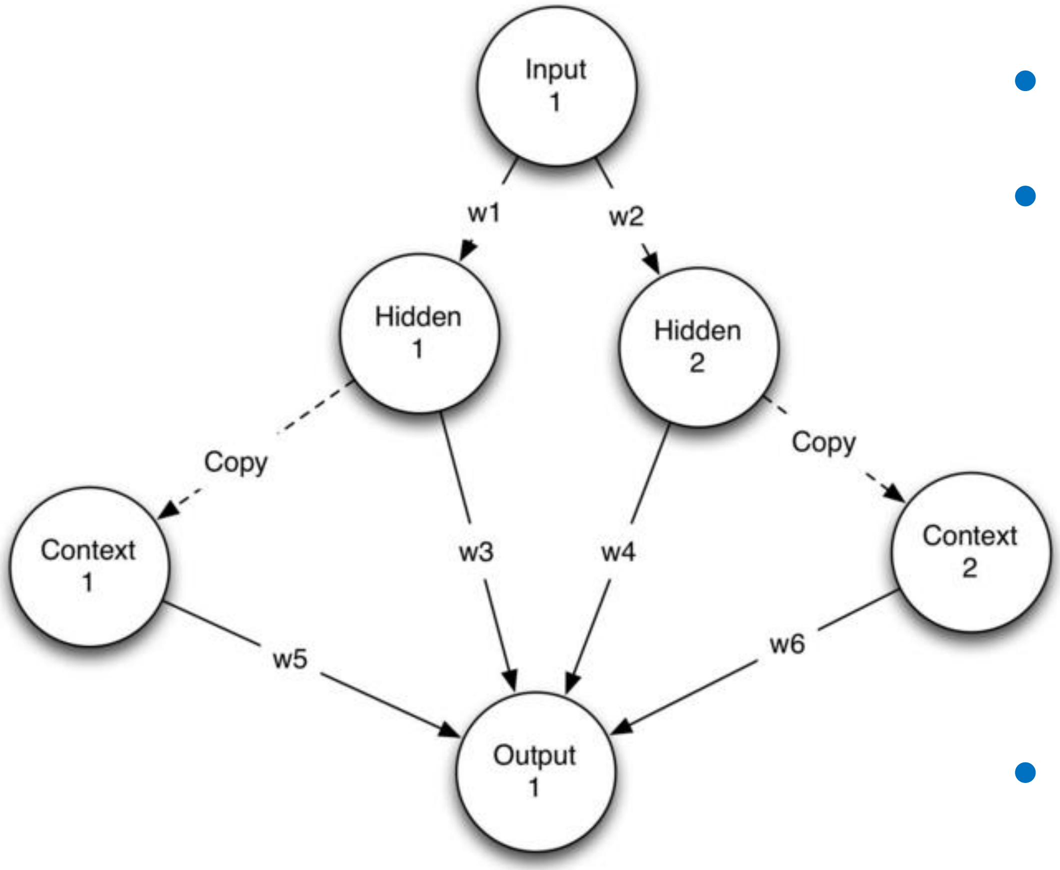
- **Questions about NN architecture:**
  - What should be the number of hidden neurons in the NN?
  - How many hidden layers should the NN include?
    - Theoretically, a single-hidden-layer NN can learn anything
    - Deep learning (NN's w/ >one hidden layer facilitates a more complex representation of patterns in the data)

Bias Neurons

- Bias neurons are added to NN to help them learn patterns
- Bai neurons function like input neurons that always produce the value of 1, which is the bias activation (but could be set to values ≠ 1
- Bias neurons are NOT connect to the previous layer
- Bias neurons allow the output of an activation function to be shifted



Context Neurons (to maintain state)



- Context neurons are used in Recurrent Neural Networks (RNN):
- They allow a NN to maintain state

- Example:
  - hearing the noise while crossing the street vs. hearing the horn while watching an action movie cause different responses
  - prior inputs give the context for processing the audio input of a horn

- Application:
  - Time series: speech recognition or prediction of trends in security prices

Activation Functions (AF)

- Linear AF
- Step AF
- Sigmoid AF
- Hyperbolic Tangent AF
- Rectified Linear Units (ReLU)
- Softmax AF

The Role of Activation Function in NNs

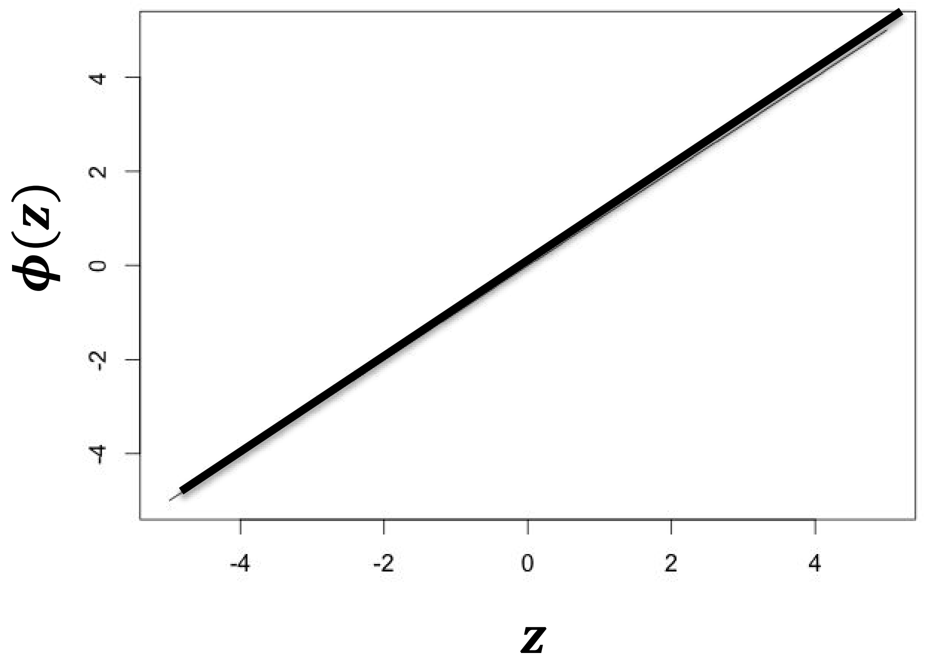
- Activation functions:
  - Establish bounds for the output neurons
  - Transform the signals, hence, they are also called the transfer functions, to facilitate learning complex and non-linear relationships

Linear Activation Function

$$\boxed{f(x_i, w_i) = \phi\left(\sum_i (w_i \times x_i)\right)}$$

$$\downarrow$$

$$\boxed{\phi(z) = z}$$



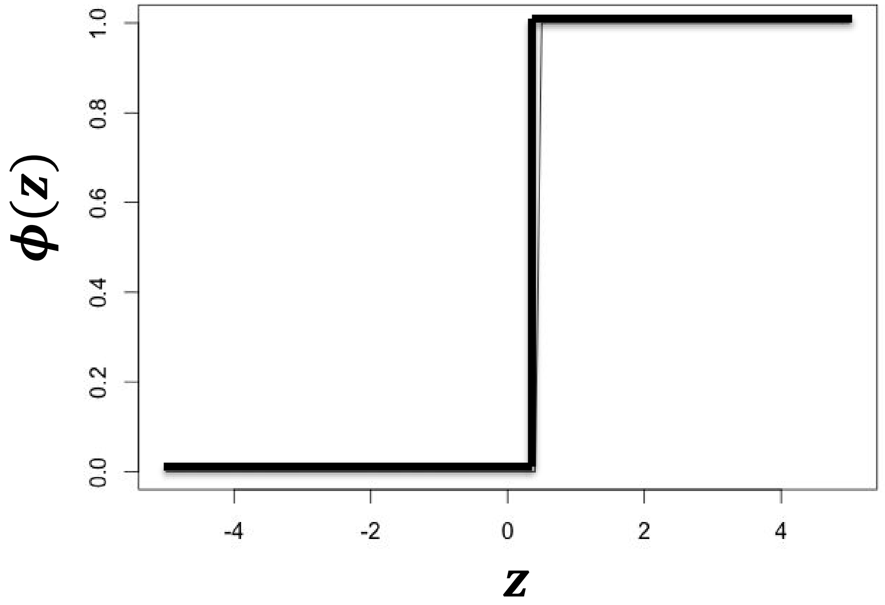
- Applications:
  - **Regression neural networks**

Step Activation Function = Threshold f()

$$f(x_i, w_i) = \phi\left(\sum_i (w_i \times x_i)\right)$$

$$\downarrow$$

$$\phi(z) = \begin{cases} 1, if\ z \geq 0.5 \\ 0, otherwise \end{cases}$$



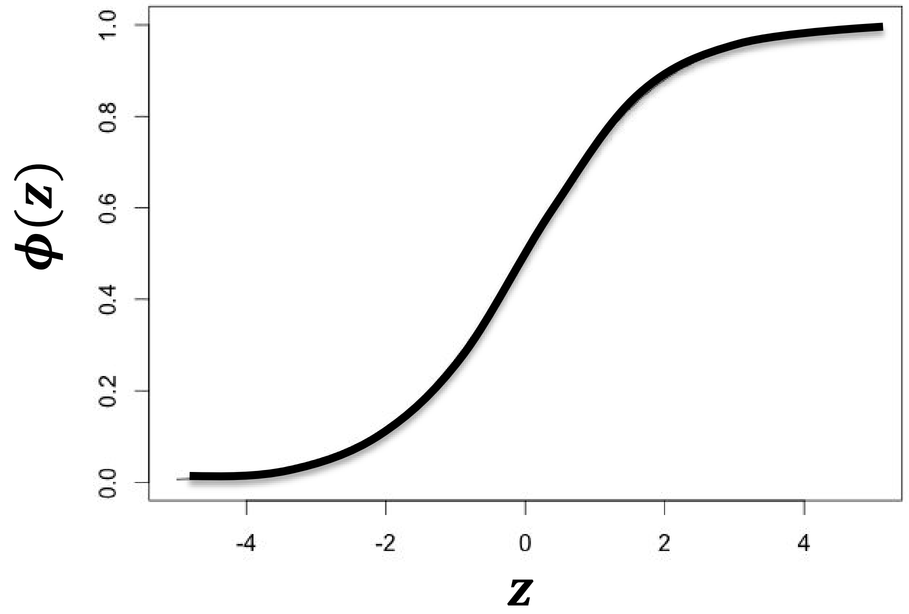
- Applications:
    - **Classification neural networks**

Sigmoid Activation Function = Logistic f()

$$\boxed{f(x_i, w_i) = \phi\left(\sum_i (w_i \times x_i)\right)}$$

$$\downarrow$$

$$\boxed{\phi(z) = \frac{1}{1 + e^{-z}}}$$



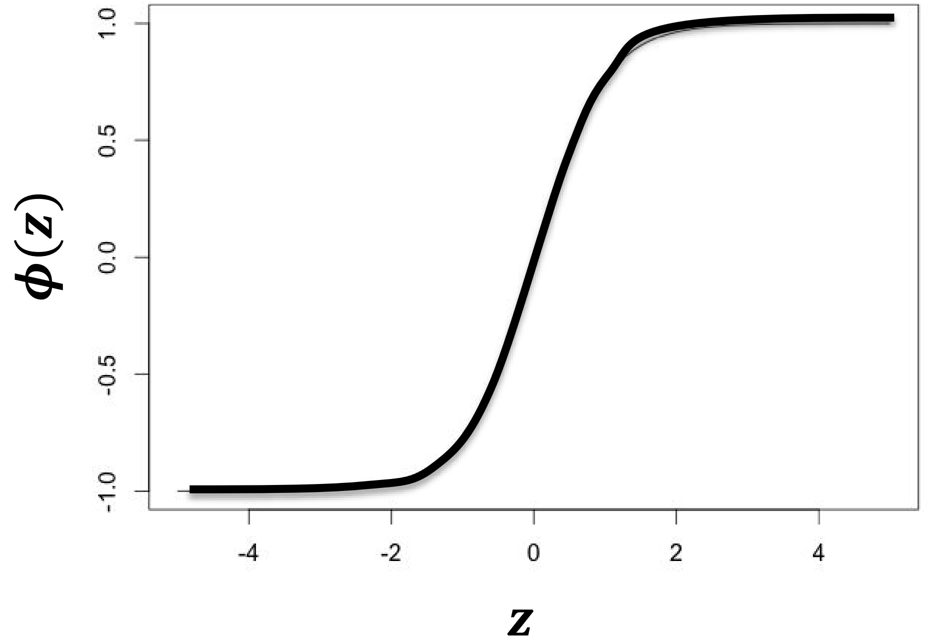
- **Applications:**
  - Feedforward neural networks that need to output only positive numbers
  - Used to ensure that the value stays within a relatively small range
  - Probabilistic classification

Hyperbolic Tangent Activation Function

$$\boxed{f(x_i, w_i) = \phi\left(\sum_i (w_i \times x_i)\right)}$$

$$\Big\downarrow$$

$$\boxed{\phi(z) = tanh\,(z)}$$



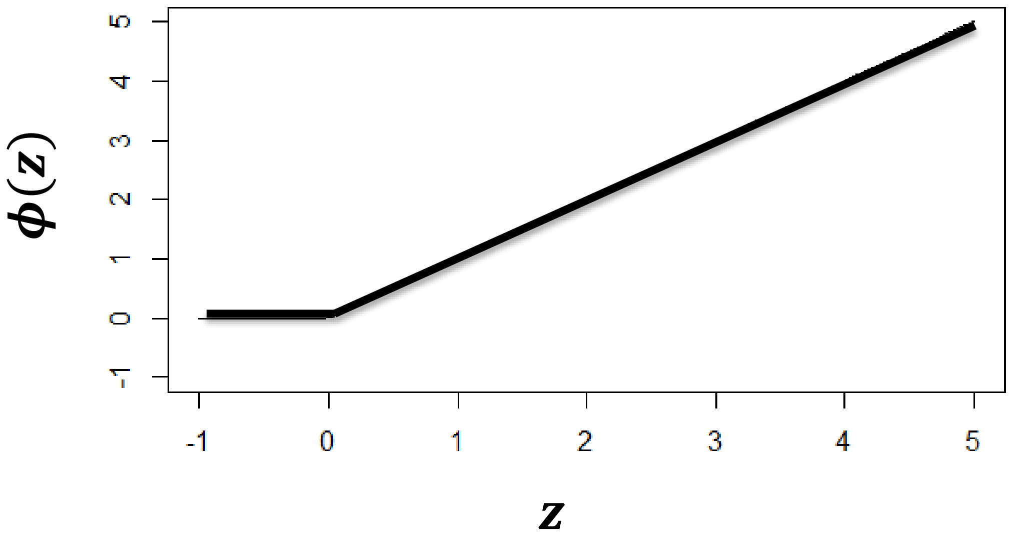
- **Applications:**
  - Used to ensure that the output values stay in the range $[\,-1; 1\,]$

Rectified Linear Units (ReLU)

$$\boxed{f(x_i, w_i) = \phi\left(\sum_i (w_i \times x_i)\right)}$$

$$\downarrow$$

$$\boxed{\phi(z) = max\ (0, z)}$$



- Introduced by Teh & Hinton (2000)
- Highly recommended for getting superior training results:
    - Because it is a linear, non-saturating function
    - Unlike sigmoid/logistics or hyperbolic tangent AFs, it does not saturated to -1, 0, or 1
- NNs should utilize ReLUs on hidden layers and linear / softmax functions on the output layer

Softmax Activation Function

$$\boxed{f(x_i, w_i) = \phi\left(\sum_i (w_i \times x_i)\right)}$$

$$\downarrow$$

$$\boxed{\phi(z_i) = \frac{e^{z_i}}{\sum_{j \in group} e^{z_j}}}$$

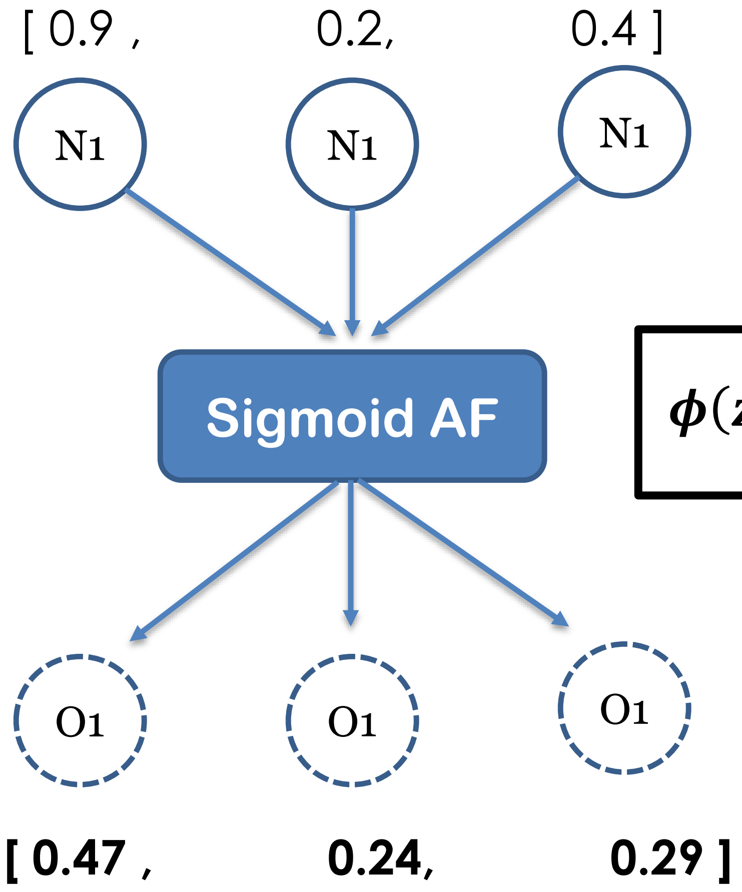
```
def softmax(neuron_output):
  sum = 0
  for v in neuron_output:
    sum = sum + v

  sum = math.exp(sum)
  proba = []
  for i in range(len(neuron_output))
proba[i] = math.exp(neuron_output[i])/sum
  return proba
```

- Usually used in the output layer of a NN
- Used in classification NN:
  - the neuron that has the highest value claims the input as the member of its class (the number of output neurons = # of classes)
- It forces the output of the NN to represent the **probability** that the input falls into each of the classes:
  - without software the neuron's output are simply numeric values, with the highest indicating the winning class (the winner takes all)

Example: Softmax AF using iris data

Neuron 1 (N1): setosa : 0.9
Neuron 2 (N2): versicolour: 0.2
Neuron 3 (N3): virginica: 0.4

- Note that these values are NOT probabilities; they sum up to 1.5 rather than 1.0

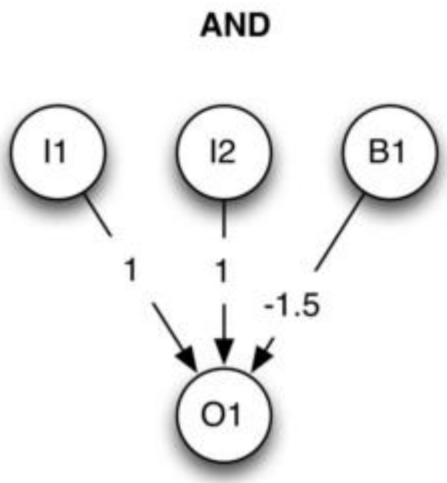


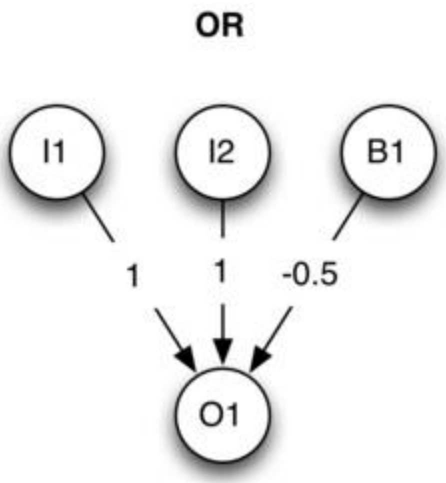
$$\boxed{\phi(z_i) = \frac{e^{z_i}}{\sum_{j \in group} e^{z_j}}}$$

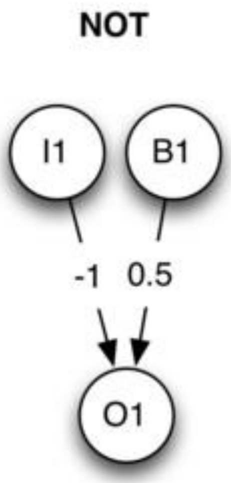
- These values do sum up to 1 and can be treated as probabilities

Logic Gates with Neural Networks

- Exercise: Show that using a step activation function will recognize the corresponding logic gates: AND, OR, and NOT







Creating more complex logical statements

- Suppose you want a house with a nice view and a large backyard but will be satisfied with a house that has a small backyard yet is near a park

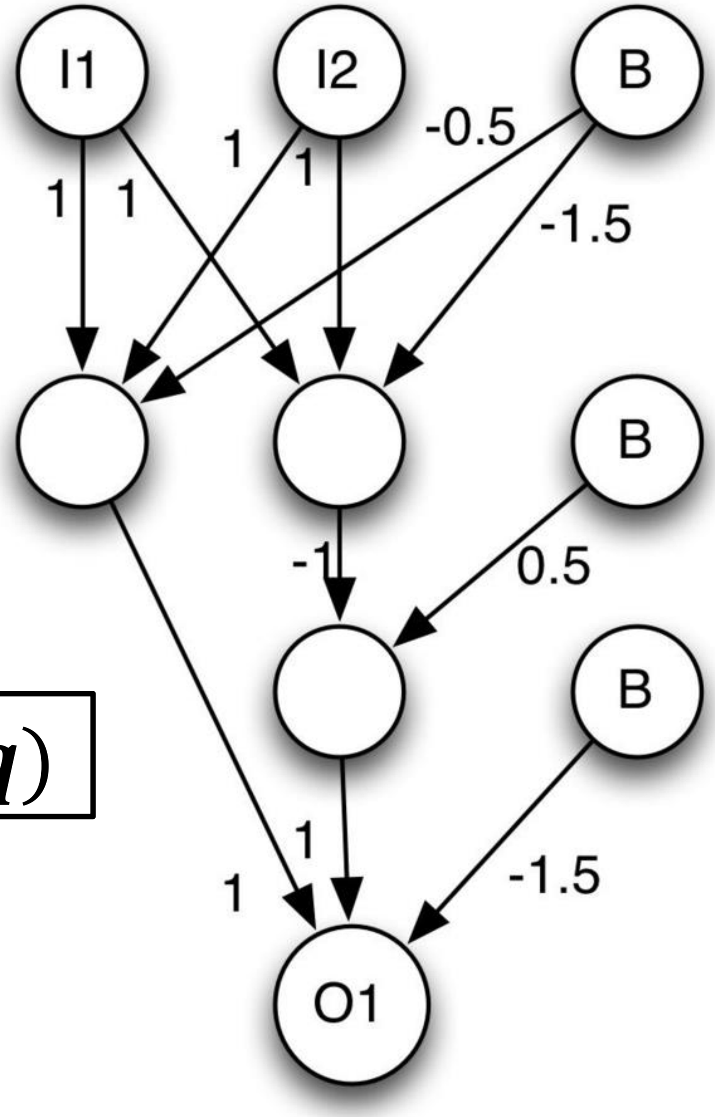
([nice view] AND [large yard]) OR ((NOT [large yard]) and [park])

$$(\boldsymbol{NV} \ \wedge \ \boldsymbol{LY}) \vee (\neg \boldsymbol{LY} \ \wedge \boldsymbol{PK})$$

XOR Neural Network: Validate Correctness

- False XOR False = False
- True XOR False = True
- False XOR True = True
- True XOR True = False

$$\boxed{p \otimes q = (p \vee q) \wedge \neg (p \wedge q)}$$



Outline

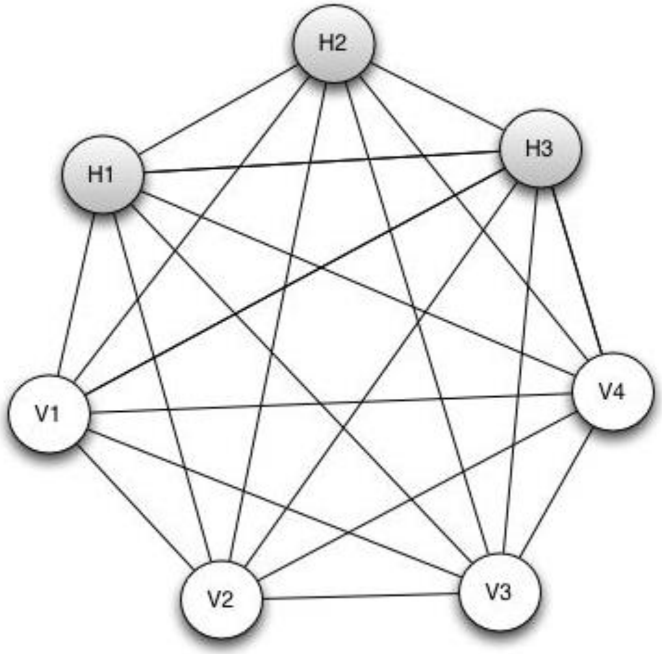
- Typical Neural Network (NN): Core Components
- Neural Network Roadmap by Problem Domains
- Neural Network Basics
  - Neurons and Layers
  - Neuron Types
  - Activation Functions
  - Logic Gates
- **Boltzmann Machines and Restricted Boltzmann Machines**
- Feedforward Neural Networks
  - Classification
  - Regression
  - Network Layers
  - Normalization

The Boltzmann Machine

	Clustering	Regression	Classification	TS Forecast	Robotics	Vision	Optimization
Self-organizing Map	XXX				X	X	
Feedforward		XXX	XXX	XX	XX	XX	
Boltzmann Machine			X				XX
Deep Belief Network			XXX		XX	XX	
Convolutional Network		X	XXX		XXX	XXX	
Recurrent Network		XX	XX	XXX	XX	X	

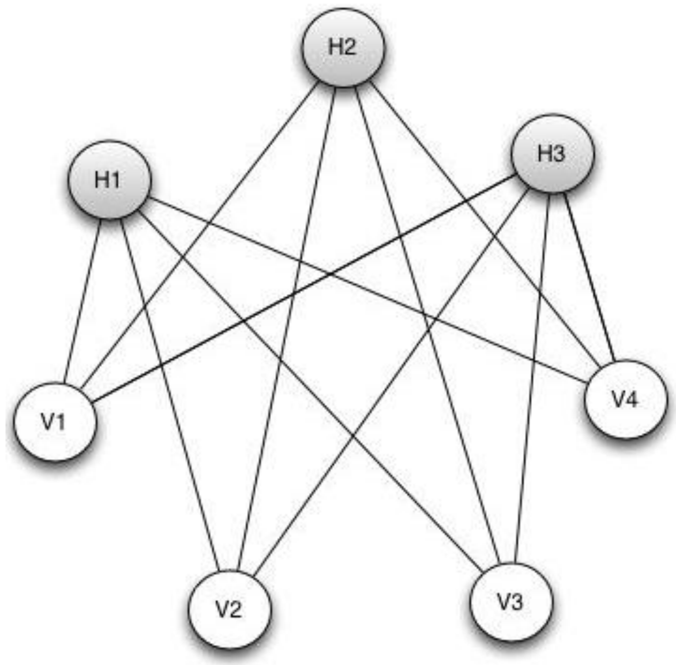
- The Boltzmann machine is a classic NN
- It is not used in modern AI on its own but
- Forms the foundation of the deep belief neural network (DBNN), which is one of the core algorithms of deep learning

The Boltzmann Machine Architecture



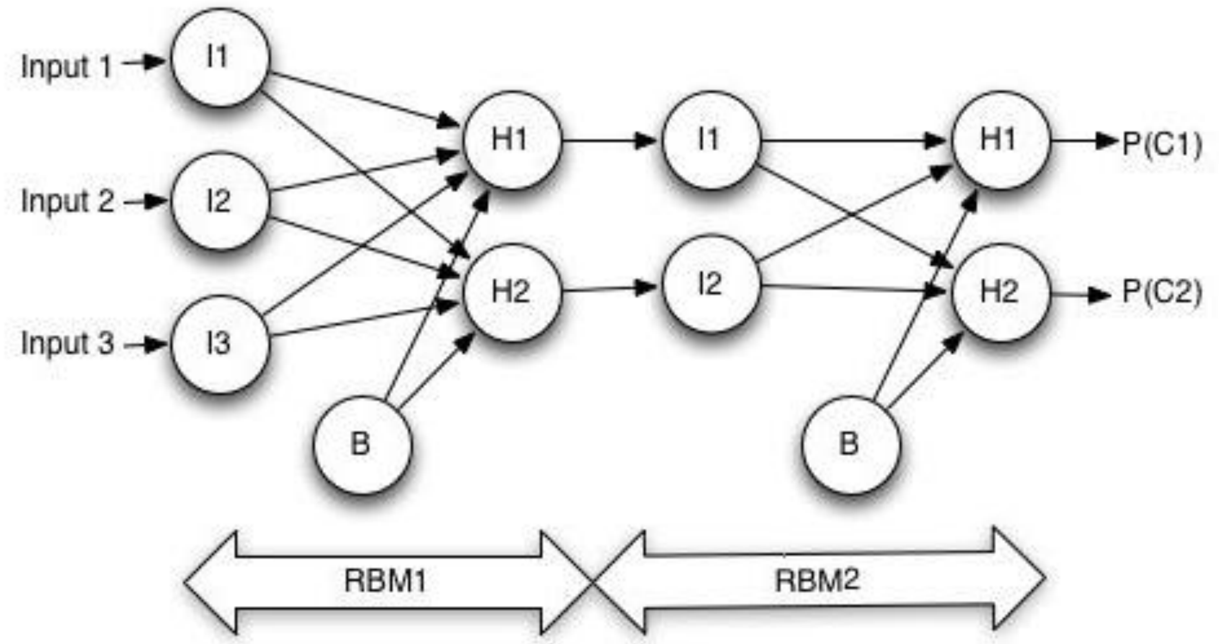
- A Boltzmann machine is a **fully connected, two-layer neural network**:
  - **Clique** in an undirected graph without self-edges
- Visual and hidden layers:
  - Visual layer = input layer
  - Hidden layer functions as the output layer
- NO Hidden layer between the input and output layers
- Boltzmann machines can stacked to form layers

RBM: Restricted Boltzmann Machine



- A Restricted Boltzmann Machine (RBM) is a two-layer neural network without connections between the neurons of the same layer:
  - **Bipartite clique** in an undirected graph without self-edged
- RBMs is the core technology underlying deep learning and deep belief neural network

Stacked RBMs in Deep NNs



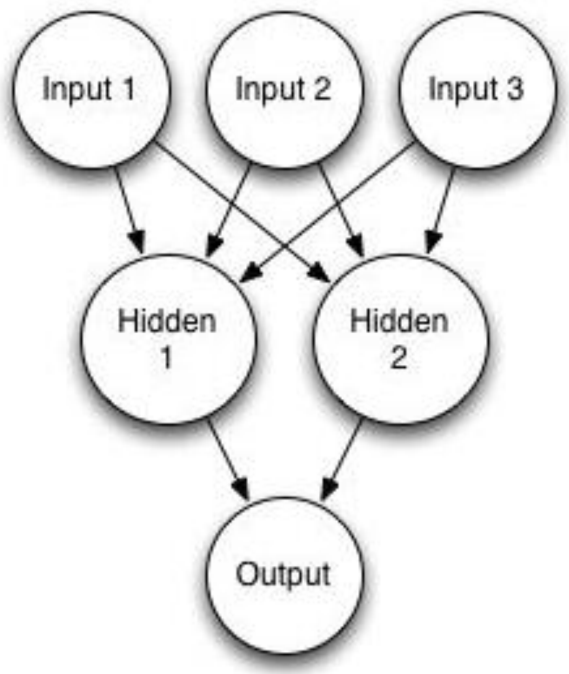
- Note that the hidden units in RBM1 pass their output directly to the inputs (visible units) of RBM2
- There are NO weights between RBM1 and RBM2

Outline

- Typical Neural Network (NN): Core Components
- Neural Network Roadmap by Problem Domains
- Neural Network Basics
- Boltzmann Machines and Restricted Boltzmann Machines
- **Feedforward Neural Networks**
  - **Data Structure**
  - **Calculating Feedforward Output**
  - **Weights of the NN**
  - **Xavier Weight Initialization**
  - **Radial-Basis Function Feedforward NN and it use for**
    - Classification
    - Regression
  - **Network Layers**
  - **Normalization**

Single-Output Feedforward Network

- Applications: Single-Output Feedforward NN
  - **Regression**
  - **Binary Classification**
- Multi-class Classification
  - Have one output neuron for each class



Single-Output Feedforward NN for Regression

- Data set: MPG : https://archive.ics.uci.edu/ml/datasets/Auto+MPG

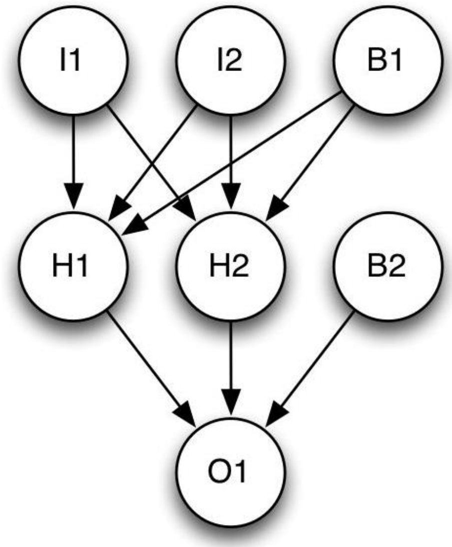
```
mpg,cylinders,displacement,horsepower,weight,acceleration,m
odel_year,origin,car_name
18,8,307,130,3504,12,70,1,"chevrolet chevelle malibu"
15,8,350,165,3693,11,70,1,"buick skylark 320"
18,8,318,150,3436,11,70,1,"plymouth satellite"
16,8,304,150,3433,12,70,1,"amc rebel sst"
```

- Output neuron: MPG value
- Input neurons: cylinders, displacement, horsepower, etc.

- Question:
    - Can NN perform regression on Multiple Output Neurons:
        - MPG value and Weight Value
    - Technically, yes. Multi-output NN is capable of performing regression on multiple output / response variables
    - But you achieve better results with separate NN for each regression outcome

Example: Data Structure

- Input Layer: 2 neurons, 1 bias
- Hidden Layer: 2 neurons, 1 bias
- Output Layer: 1 neuron
- Total: **7 neurons**



- Neuron vector of **length 7**:
  - Neuron 0: Output 1
  - Neuron 1: Hidden 1
  - Neuron 2: Hidden 2
  - Neuron 3: Bias 2 (init to 1)
  - Neuron 4: Input 1
  - Neuron 5: Input 2
  - Neuron 6: Bias 1 (init to 1)

- Additional vectors in the data structure
  - *layerFeedCounts*: [1, 2, 2] : the count of non-bais neurons in each layer
  - *layerCounts*: [1, 3, 3] : # of neuron in each layer
  - *layerIndex*: [0, 1, 4] : where each layer begins in the *layerOutput* vector
  - *layerOutput*: [0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0] : current value of each neuron (initially 0's except for bias neurons that are 1's)
  - *weightIndex*: [0, 3, 9] : holds indexes to location of each layer in weight vector

- Weights are stored as vectors & structured as:
  - Weight 0 : H1 → O1
  - Weight 1: H2 → O1
  - Weight 2: B2 → O1
  - Weight 3: I1 → H1
  - Weight 4: I2 → H1
  - Weight 5: B1 → H1
  - Weight 6: I1 → H2
  - Weight 7: I2 → H2
  - Weight 8: B1 → H2

Ex: Calculating the Output

Calculate Feedforward Output

$$f(x_i, w_i) = \phi\left(\sum_i (w_i \times x_i)\right)$$

```
def compute(net, input):
  sourceIndex = len(net.layerOutput)
    - net.layerCounts[len(net.layerCounts) - 1]
  # Copy the input into the layerOutput vector
  array_copy(input, 0, net.layerOutput, sourceIndex,
net.inputCount)
  # Calculate each layer
  for i in reversed(range(0,len(layerIndex))):
    compute_layer(i)
  # update context values
  offset = net.contextTargetOffset[0]
  # Create result
  result = vector(net.outputCount)
  array_copy(net.layerOutput, 0, result, 0,
net.outputCount)
   return result
```

```
def compute_layer(net,currentLayer):
  inputIndex = net.layerIndex[currentLayer]
  outputIndex = net.layerIndex[currentLayer - 1]
  inputSize = net.layerCounts[currentLayer]
  outputSize = net.layerFeedCounts[currentLayer - 1]
  index = this.weightIndex[currentLayer - 1]
  limit_x = outputIndex + outputSize
  limit_y = inputIndex + inputSize
  # weight values
  for x in range(outputIndex,limit_x):
    sum = 0;
    for y in range(inputIndex,limit_y):
      sum += net.weights[index] * net.layerOutput[y]
        net.layerSums[x] = sum
        net.layerOutput[x] = sum
        index = index + 1

  net.activationFunctions[currentLayer - 1]
     .activation_function(
 net.layerOutput, outputIndex, outputSize)
```

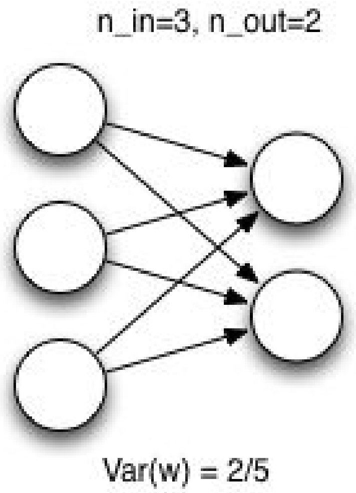
Weights of the NN

- About the Weights of the NN:
  - The weights determine the output for the NN
  - Training adjusts these weights so the output produced is useful
  - Initialization of weights will affect iterative training convergence
  - Training progresses in iterations by continuously improving the weights to produce better output
- How to initialize the weights?
  - Bad Strategy: Assign random weights and if the NN model is of poor performance, restart with a new set of random weights
  - Wish list:
    - How consistently the algorithm provides good weights upon changes to weight init values and architectural changes to NN?
    - How much of an advantage do the weights of the algorithm provide?

Xavier Weight Initialization

- Sets all of the weights to normally distributed random numbers
- Weights within the same layer have the same init value
- The weights are always centered at 0
- And the standard deviation (or sqrt of the variance) defined as a function of the number of neurons in the in-layer and the out-layer provided the edge connects nodes between in-to-out layers:

$$Var\ (weight) = \frac{2}{n_{in} + n_{out}}$$



Radial-Basis Function (RBF) Feedforward NN

- Introduced by Broomhead & Lowe (1988)
- Can be used for both classification & regression
- RBF can handle non-linearity in the data
- But they loose popularity as they are not used with deep learning
- RBF:
  - Accepts multi-dimensional input
  - Returns a single value that is the function of the distance between the input vector and the data center vector:

$$\boldsymbol{r_i} = \| \boldsymbol{x_c} - \boldsymbol{x_i} \| \longleftarrow \text{Euclidean distance}$$

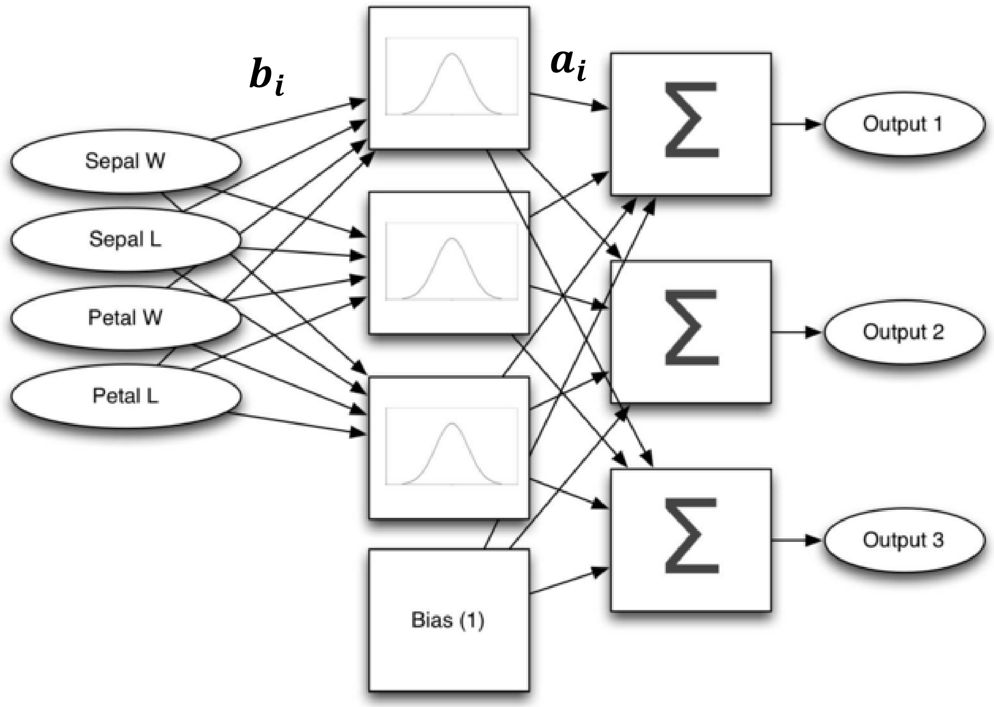
$$\boxed{\boldsymbol{\phi(r_i) = e^{-r_i^2}}} \longleftarrow \textbf{Gaussian RBF}$$

$$\boxed{\boldsymbol{f(X) = \sum_i a_i \times \phi(\|b_i X - c_i\|)}} \longleftarrow \textbf{RBF Network Output}$$

- a_i : weight for each RBF
- b : weight for each input attribute
- c_i : the vector center for each RBF

Example: RBF NN for the Iris Data

The RBF Network for the Iris Data



Input Neurons **RBFs & Bias Node** **Output Neurons**

Normalizing the Data

- One-of-N Encoding (for categorical values)
- Range Normalization
- Z-score Normalization
- Complex / Heterogeneous Normalization

References: TBD

- Books
  - TBD
  - TBD
- Papers
  - TBD
  - TBD