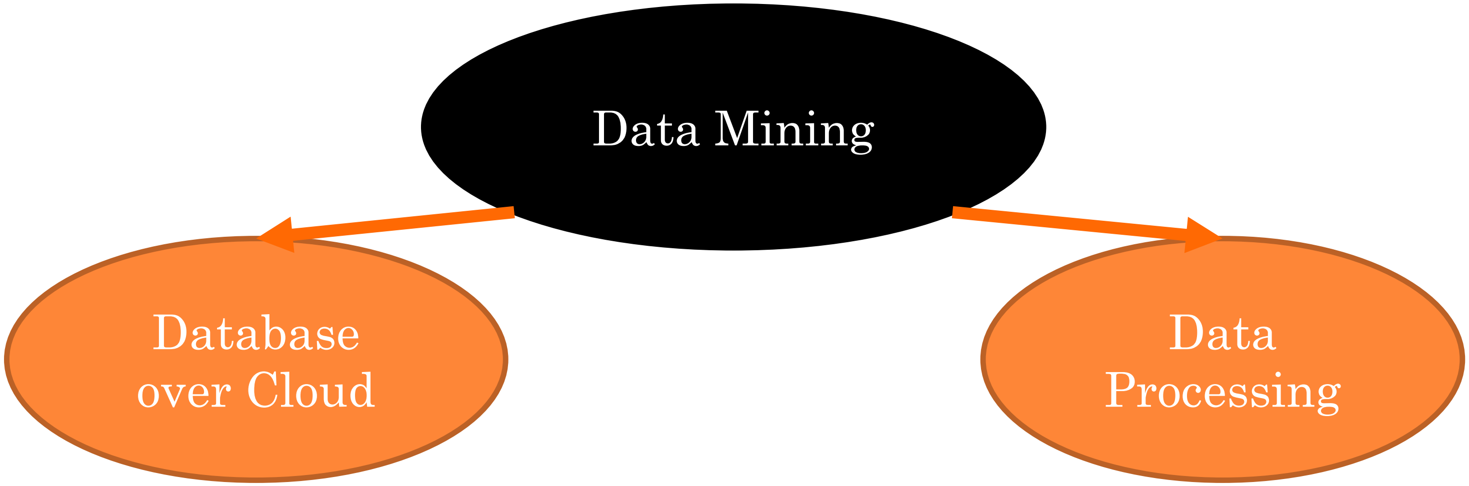
SCC.413 Applied Data Mining
Week 14 Introduction

BEHIND DATA MINING

- Ontology

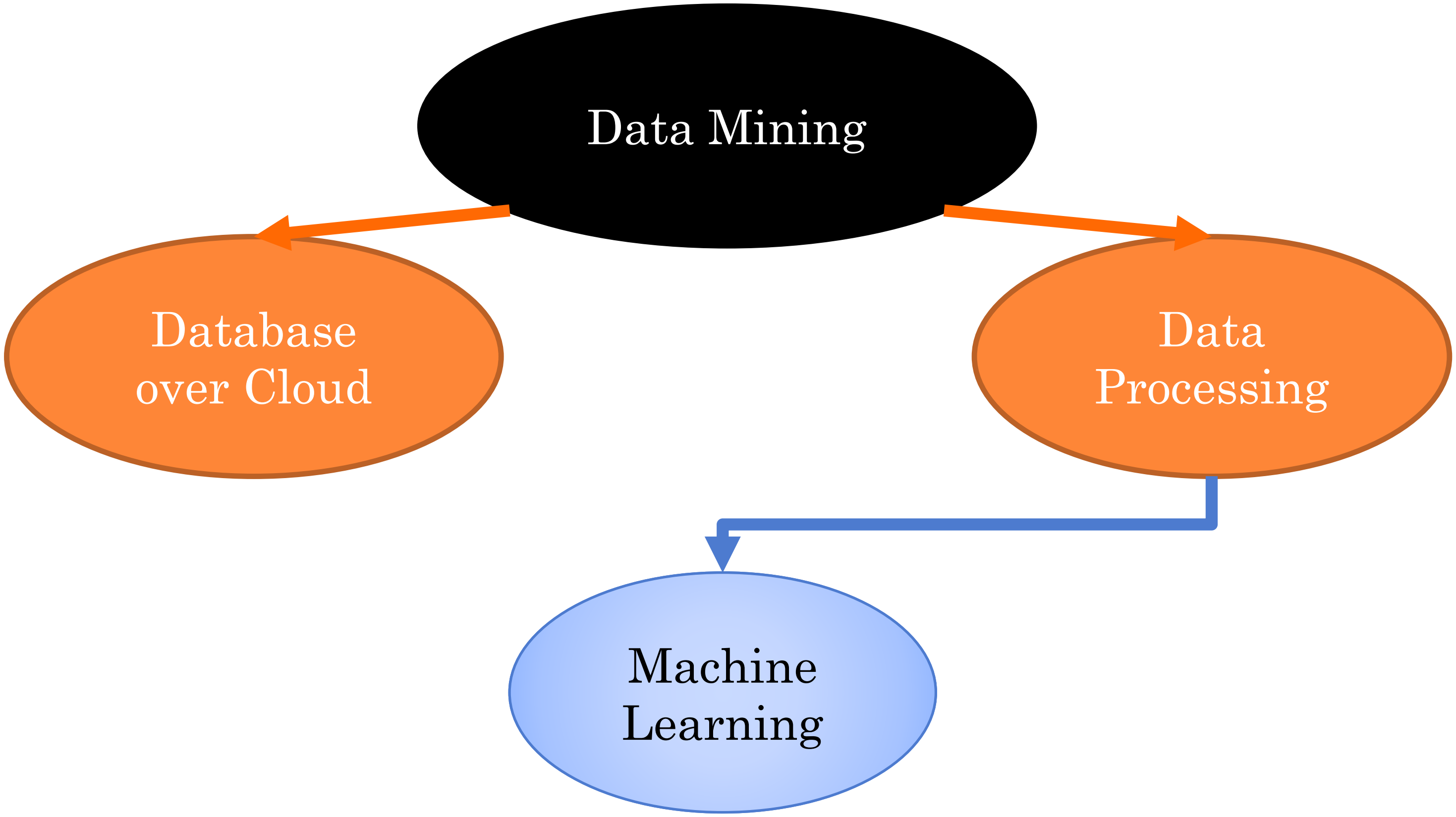
Behind Data Mining

- Ontology



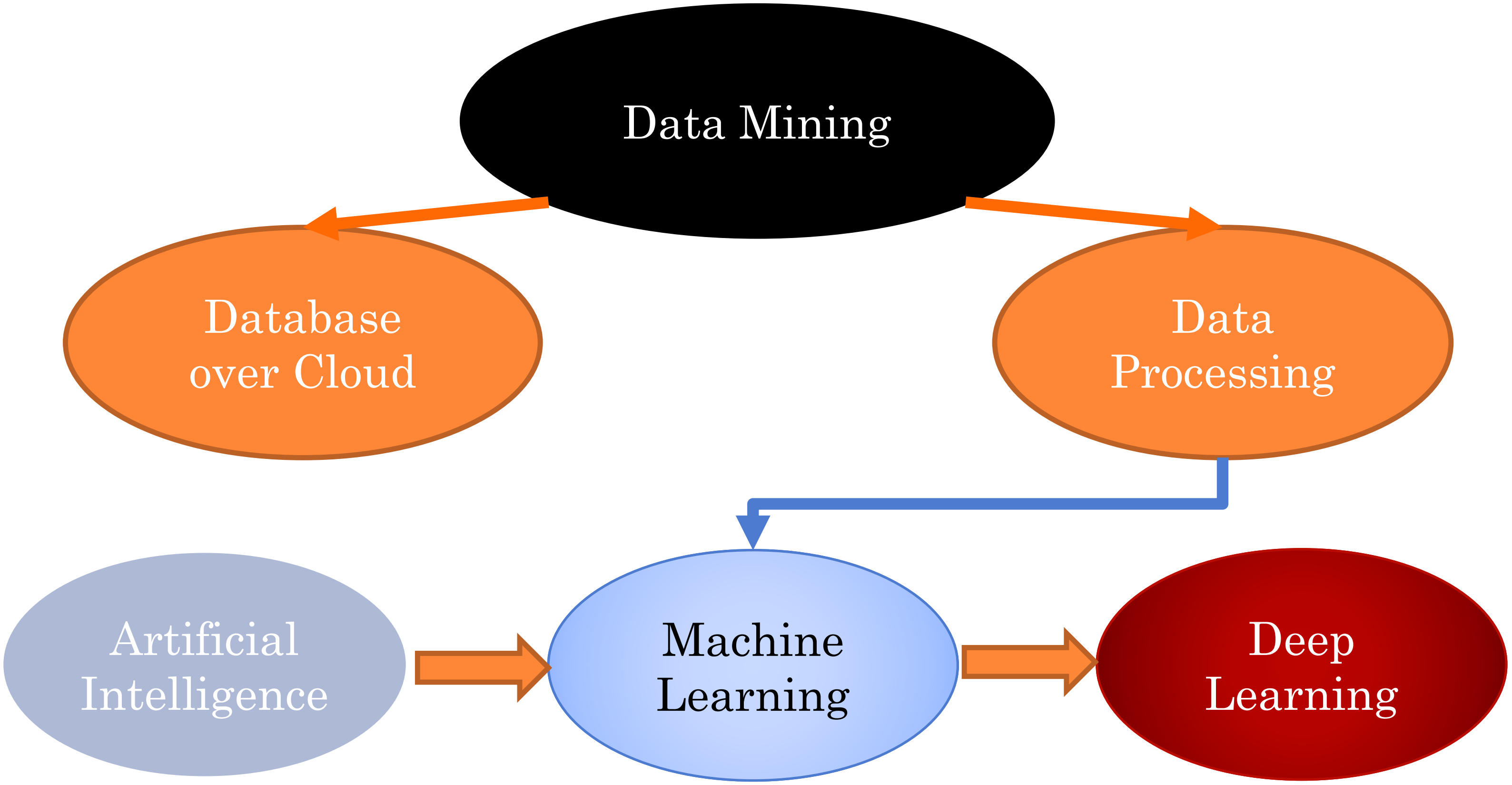
BEHIND DATA MINING

- Ontology

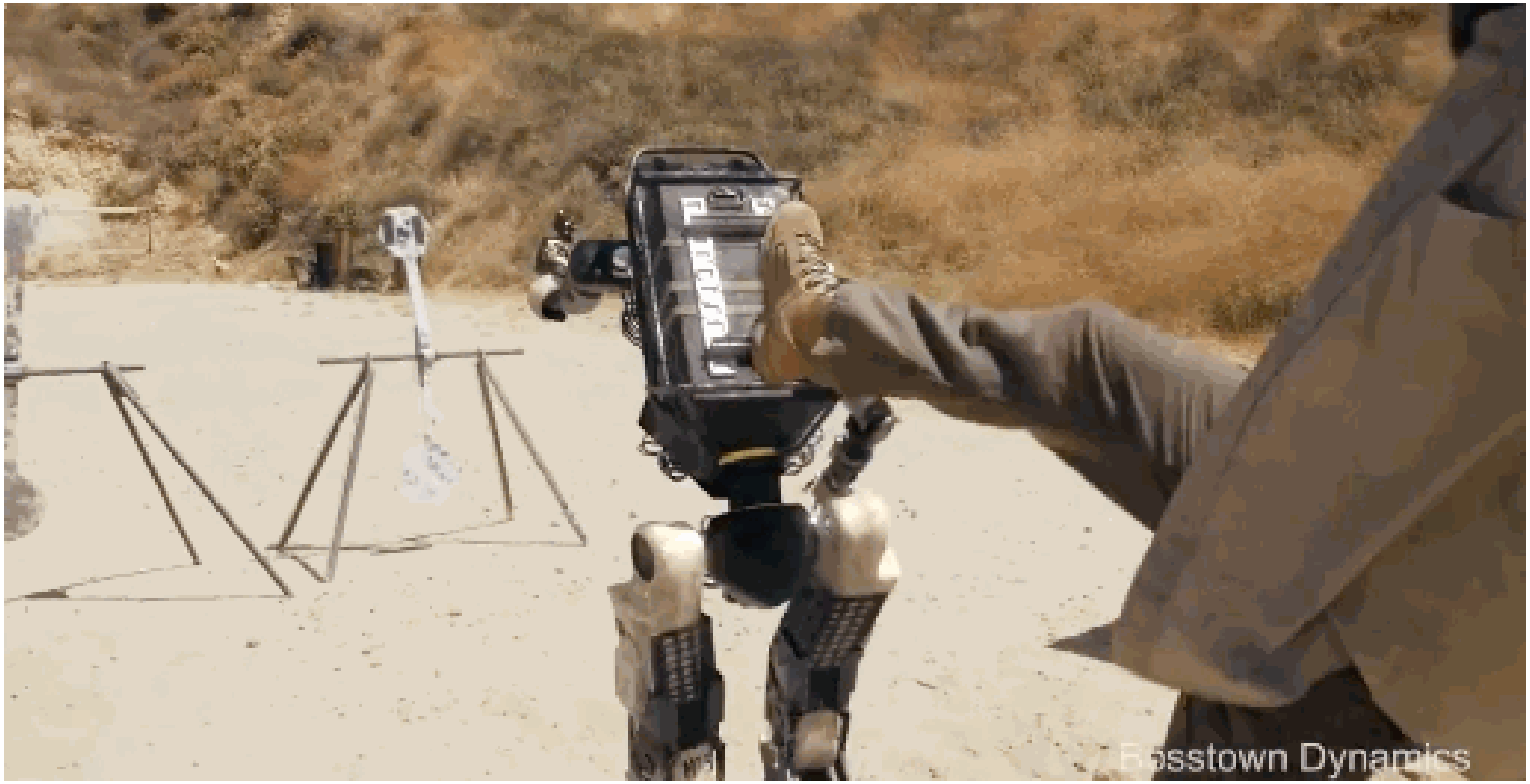


Behind Data Mining

- Ontology



Pretty much for computer vision




Bosstown Dynamics


Bosstown Dynamics

Outline

- The history of neural networks
- Fundamentals of neural networks
- Single layer perceptrons
- Training and test of a perceptron learning
- Multi-layer perceptron (MLP)

The history of neural nets

- In 1943, McCulloch and Pitts developed a neural network model based on their understanding of neurology (but the models were typically limited to formal logic simulations).

The history of neural nets

- In 1943, McCulloch and Pitts developed a neural network model based on their understanding of neurology (but the models were typically limited to formal logic simulations).
- It wasn't until the late 1950s that promising models began to emerge (models were built to understand human memory and learning)...

THE HISTORY OF NEURAL NETS

- In 1943, McCulloch and Pitts developed a neural network model based on their understanding of neurology (but the models were typically limited to formal logic simulations).

- It wasn't until the late 1950s that promising models began to emerge (models were built to understand human memory and learning)...

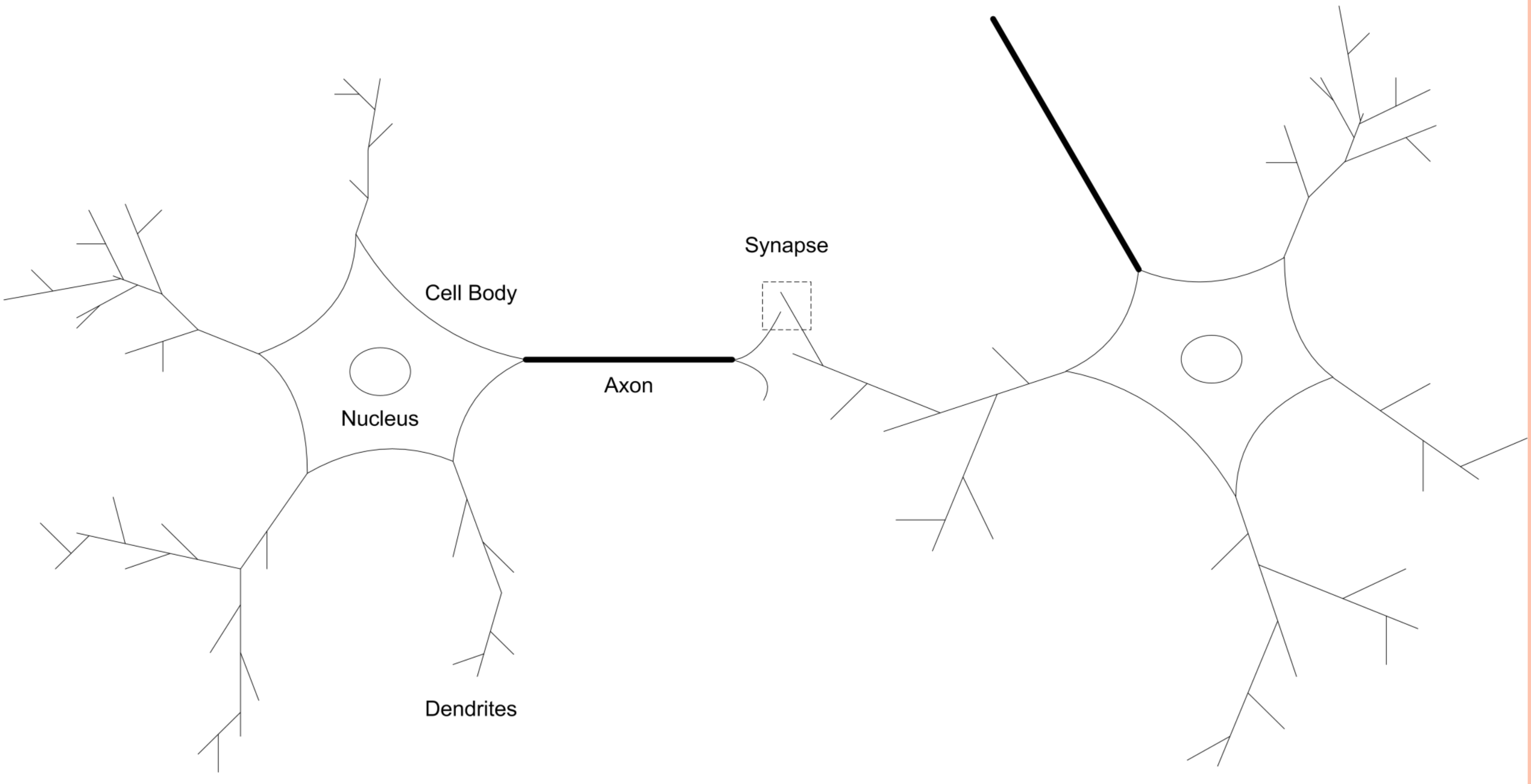
- In the 1960s, the growing popularity of neural networks was brought to a halt. Minsky and Papert wrote a book entitled "Perceptrons" to discuss the limitations of the single-layer perceptrons, which led to research funding cut...

The history of neural nets

- In 1943, McCulloch and Pitts developed a neural network model based on their understanding of neurology (but the models were typically limited to formal logic simulations).
- It wasn't until the late 1950s that promising models began to emerge (models were built to understand human memory and learning)...
- In the 1960s, the growing popularity of neural networks was brought to a halt. Minsky and Papert wrote a book entitled "Perceptrons" to discuss the limitations of the single-layer perceptrons, which led to research funding cut...
- In 1974, Werbos developed the backpropagation algorithm, which permitted successful learning in multi-layer neural networks.

BIOLOGICAL MOTIVATION

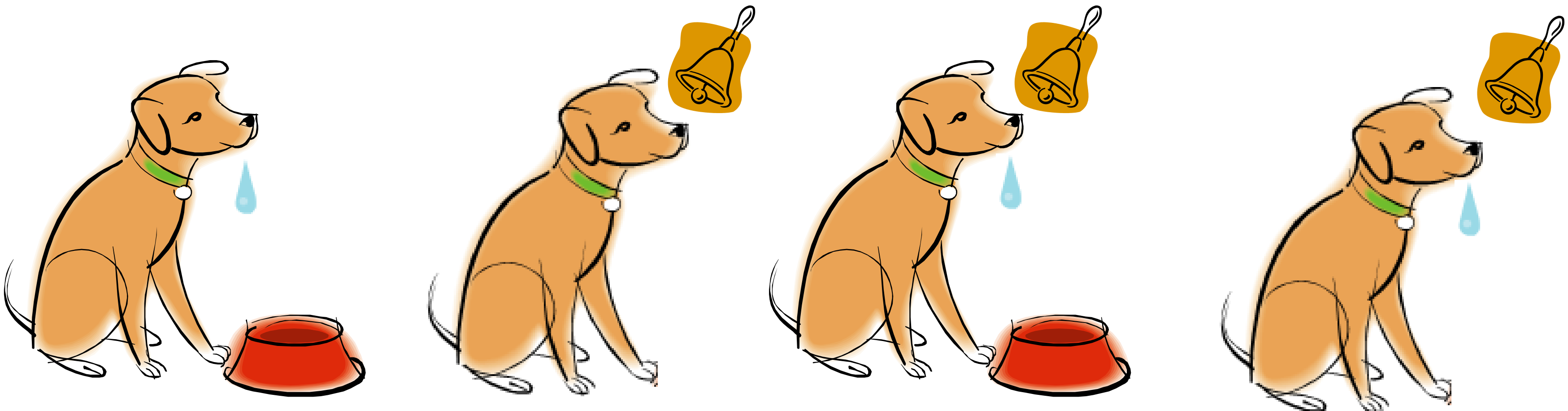
- In 1943, McCulloch & Pitts built a new model for information processing based on their knowledge on neurology.



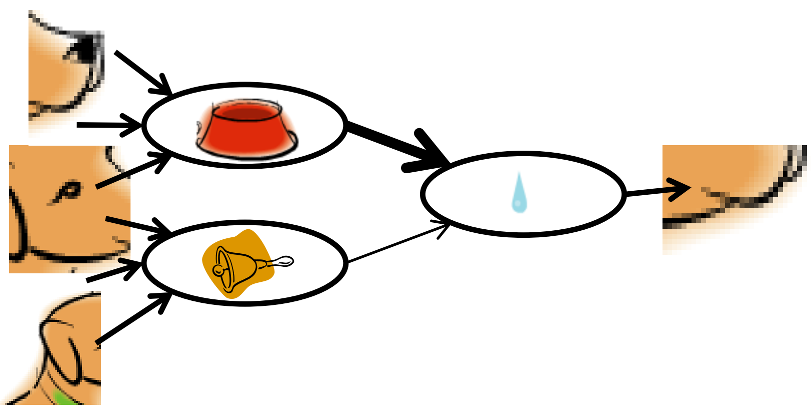
Hebbian Learning

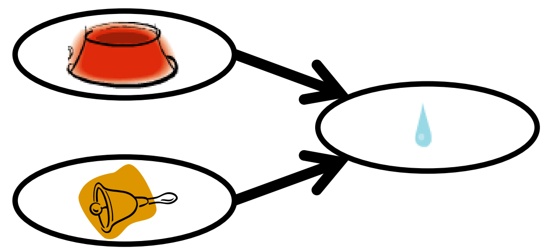
"The general idea is that any two cells or systems of cells that are repeatedly active at the same time will tend to become 'associated', so that activity in one facilitates activity in the other." [Hebb 1949]

Hebbian Learning



Before conditioning　Conditioning　After Conditioning





"The general idea is that any two cells or systems of cells that are repeatedly active at the same time will tend to become 'associated', so that activity in one facilitates activity in the other." [Hebb 1949]

Artificial neuron

ARTIFICIAL NEURON

- The neuron is a simple processing device that has inputs (known as dendrites) and outputs (known as axons).

Artificial neuron

- The neuron is a simple processing device that has inputs (known as dendrites) and outputs (known as axons).

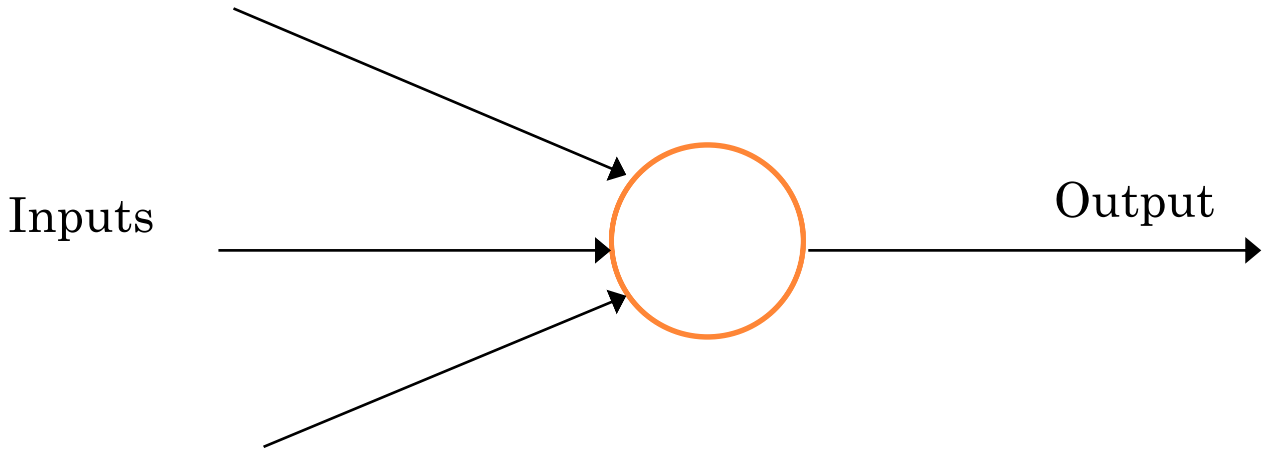
- When a neuron receives excitatory inputs that exceed its inhibitory inputs, a signal is transmitted down to its axon to other neurons.

Artificial neuron

- The neuron is a simple processing device that has inputs (known as dendrites) and outputs (known as axons).
- When a neuron receives excitatory inputs that exceed its inhibitory inputs, a signal is transmitted down to its axon to other neurons.
- Learning can then be defined as the altering of the synaptic junctions that change the manner in which one neuron is influenced by others.

Artificial Neurons

- A cartoon model of a real neuron
- Output is a simple function of the inputs
- Exact value determined by weight of each connection, and an (optional) threshold value
- Present inputs in turn and find outputs
- Adjust weights to get behaviour we want



Fundamentals of neural networks

- A neural network is made up of one or more neurons, which is the basic processing element.

Fundamentals of Neural Networks

- A neural network is made up of one or more neurons, which is the basic processing element.
- A neuron has one or more inputs, each of which are individually weighted.
- A neuron has one or more outputs that are weighted when connecting to other neurons.
-

FUNDAMENTALS OF NEURAL NETWORKS

- A neural network is made up of one or more neurons, which is the basic processing element.

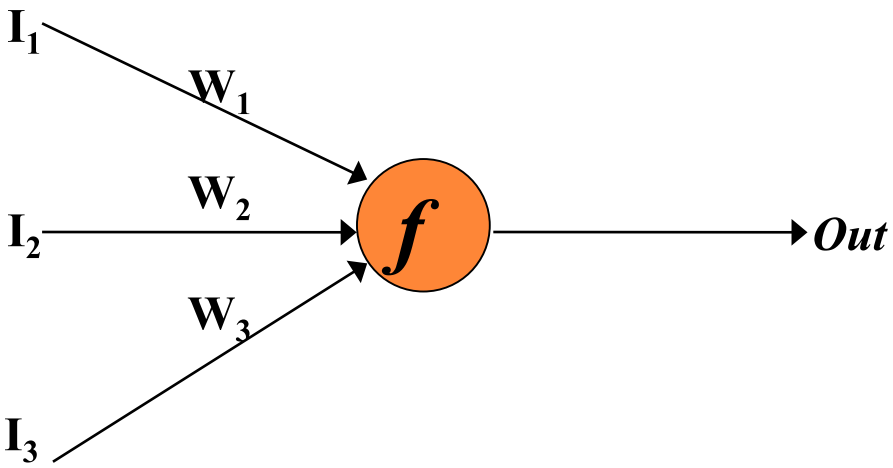
- A neuron has one or more inputs, each of which are individually weighted.

- A neuron has one or more outputs that are weighted when connecting to other neurons.

- The neuron itself includes a function that incorporates its inputs (via summarization) and then normalizes its output via a transfer function.

ACTIVATION

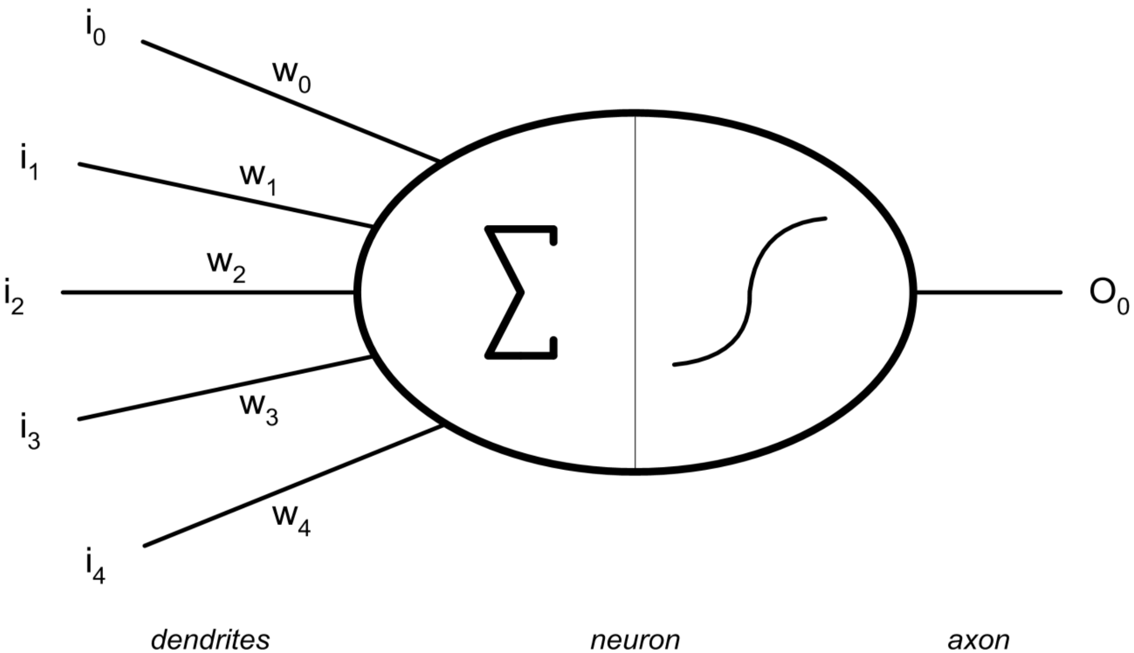
- Each input receives a value
- The inputs are multiplied by respective weights and added together
- Output (*Activation*) is a simple function of weighted input



$$sum = \sum I_i w_i$$
$$= I_1 w_1 + I_2 w_2 + I_3 w_3$$

$$out = f(sum)$$

Fundamentals of Neural Networks



The following equation is provided for this simple neuron.

$$O_0 = f\left[\sum_{j=0}^{n}(i_j W_j)\right]$$

Simple neuron with biological equivalents

Activation Functions

- Different types of activation function produce different types of output
-

Activation Functions

- Different types of activation function produce different types of output

- Can picture activation function as a graph of output against input sum

-

Activation Functions

- Different types of activation function produce different types of output

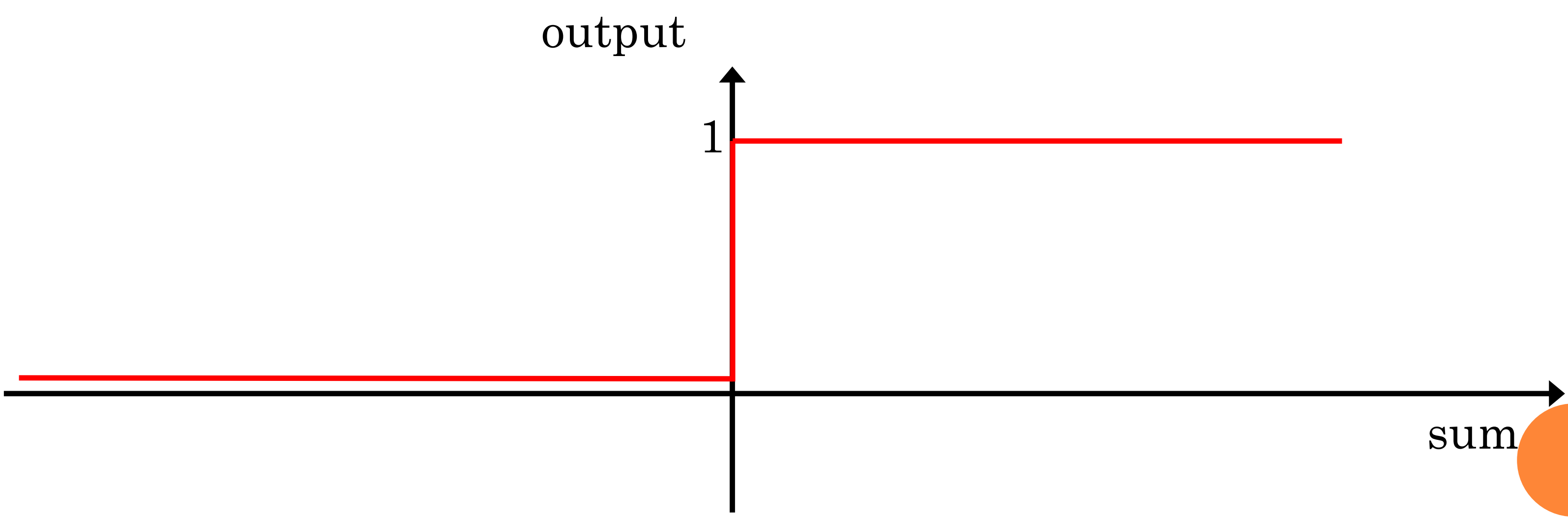
- Can picture activation function as a graph of output against input sum

- The type of activation function depends on what kind of output we want

  - Binary (1 or 0, *yes* or *no*, *A* or *B*)
  - Continuous (any number)
  - Continuous range (any number 0-1)

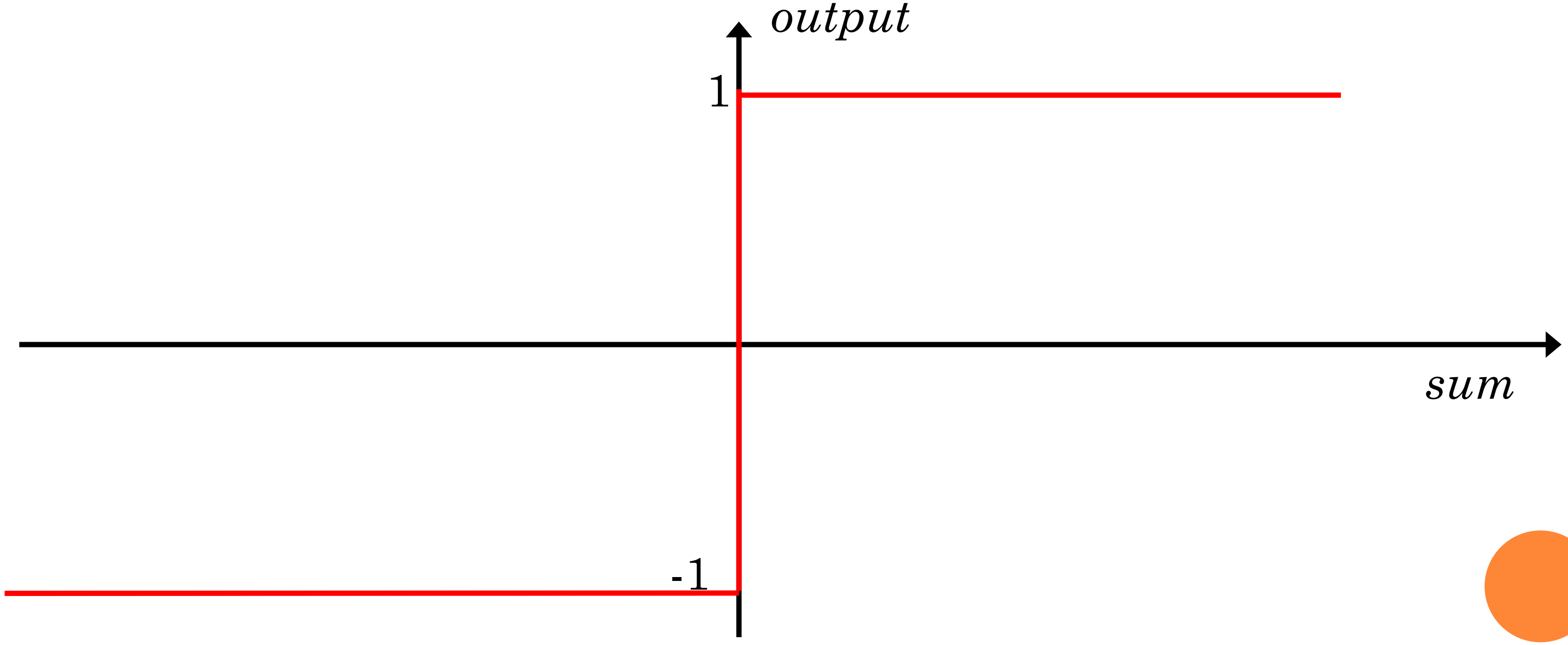
Binary Step Function

- Produces binary output (0 or 1)
- Good for yes/no type questions
- *Output = 1 if sum>0, 0 otherwise*



Bipolar Step Function

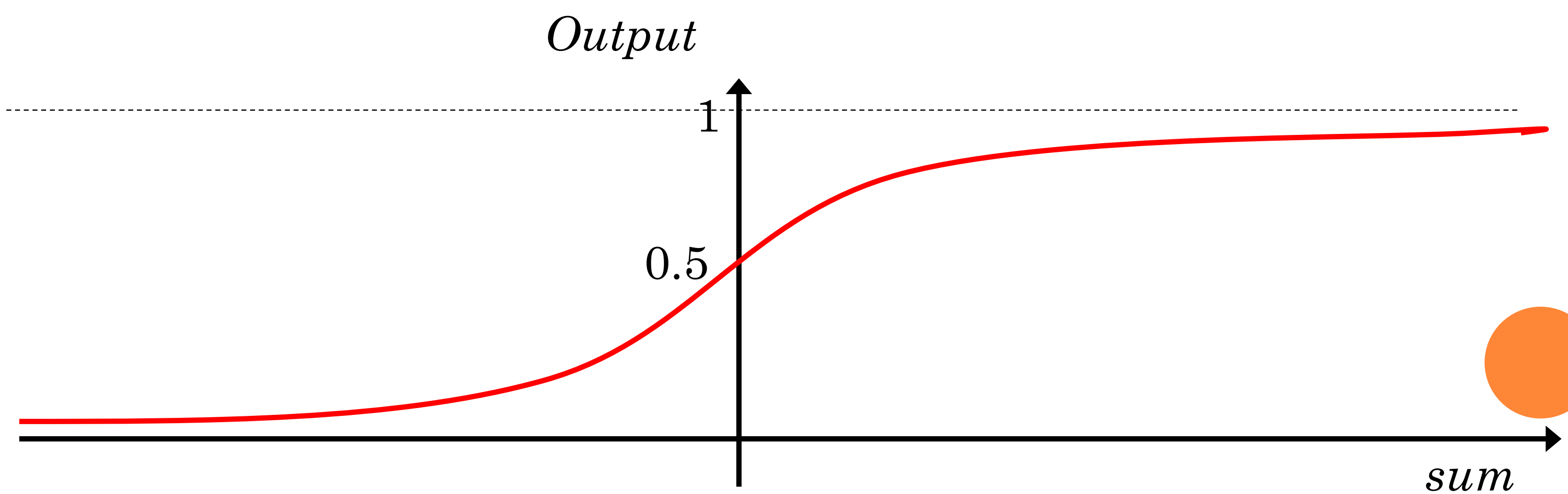
- Or *bipolar* output: -1 or 1
- *output = +1 if sum>0, -1 otherwise*
- (Very similar to binary, but occasionally learns better)



Sigmoidal (Squashing) Function

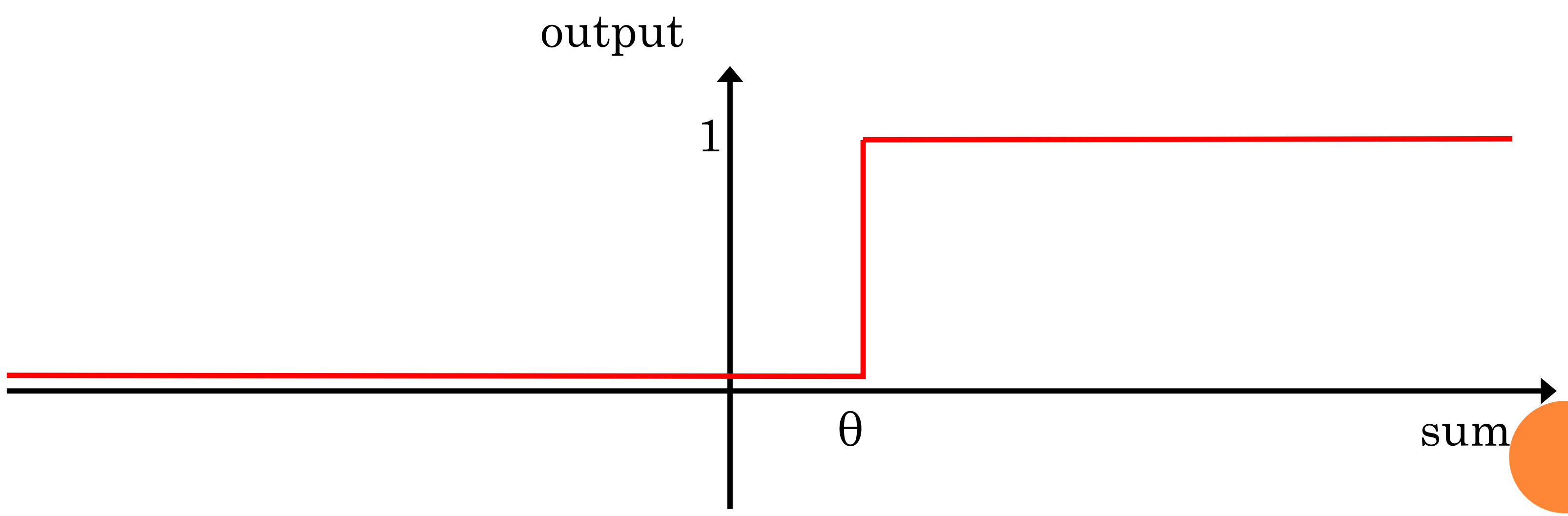
- Produces continuous output in a limited range (0,1)
- Two *asymptotes*:
  - *output*→*1 as sum*→∞
  - *output*→*0 as sum*→-∞

$$output = \frac{1}{1+e^{-sum}}$$



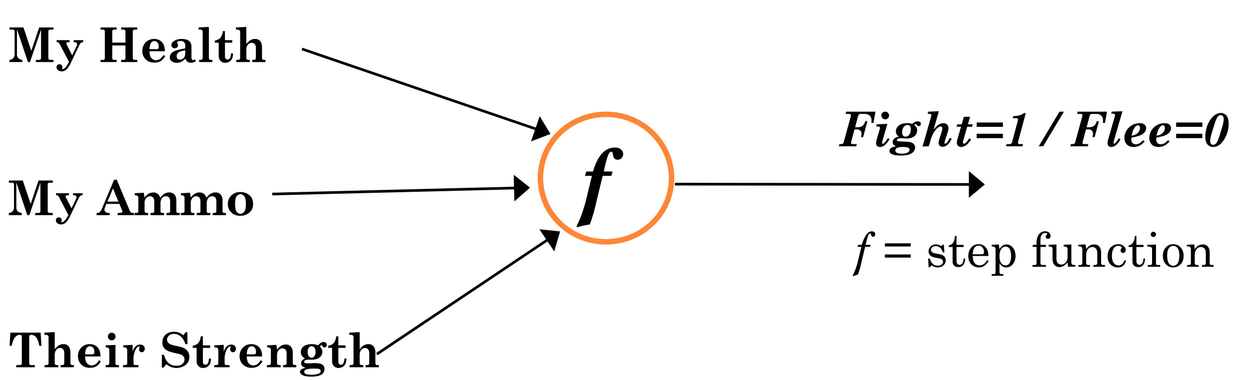
Thresholded Step Function

- Often useful to add a threshold
- *output = 1 if sum>θ, 0 otherwise,* or
- *output =1 if sum-θ>0, 0 otherwise*



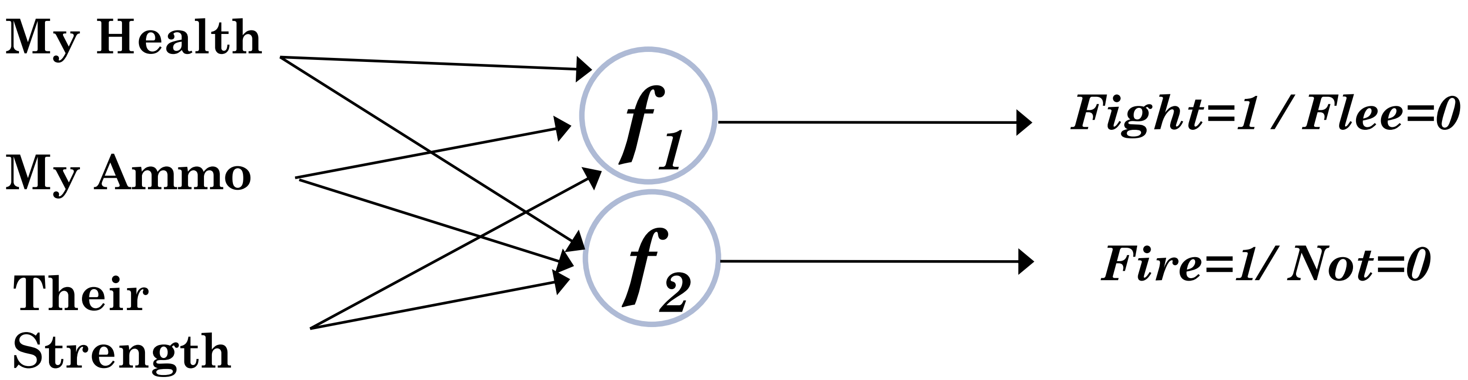
Games Example

Input			Output
Your Health	*Your Ammo*	*Their Strength*	*Action*
0.73	0.34	0.49	*Fight*
0.70	0.09	0.66	*Flee*
0.49	0.60	0.61	*Flee*
0.12	0.03	0.31	*Fight*
0.46	0.90	0.91	*Flee*
0.29	0.98	0.34	*Fight*
0.01	0.11	0.55	*?*



Games Example

Input			Output	
Your Health	***Your Ammo***	***Their Strength***	***Fire***	***Flee***
0.73	0.34	0.49	*Fire*	*Stay*
0.70	0.09	0.66	*Don't*	*Flee*
0.49	0.60	0.61	*Don't*	*Flee*
0.12	0.03	0.31	*Don't*	*Stay*
0.46	0.90	0.91	*Fire*	*Flee*
0.29	0.98	0.34	*Fire*	*Stay*
0.01	0.11	0.55	?	?



Supervised vs unsupervised learning

- Two basic categories of learning algorithms for neural networks: supervised and unsupervised learning.

Supervised vs unsupervised learning

- Two basic categories of learning algorithms for neural networks: supervised and unsupervised learning.

- In supervised learning paradigm, the neural network is trained with data that has known right and wrong answers.

-

Supervised vs unsupervised learning

- Two basic categories of learning algorithms for neural networks: supervised and unsupervised learning.
- In supervised learning paradigm, the neural network is trained with data that has known right and wrong answers.
- By calculating the output of the neural network and comparing this to the expected output for the given test data, we can identify the error and adjust the weights accordingly.
-

Supervised vs unsupervised learning

- Two basic categories of learning algorithms for neural networks: supervised and unsupervised learning.

- In supervised learning paradigm, the neural network is trained with data that has known right and wrong answers.

- By calculating the output of the neural network and comparing this to the expected output for the given test data, we can identify the error and adjust the weights accordingly.

- Examples of supervised learning algorithms include: Perceptron learning, Least-Mean-Squares Learning and Backpropagation.

Single layer perceptrons (SLPs)

SINGLE LAYER PERCEPTRONS (SLPS)

- Single layer perceptrons (SLPs) can be used to emulate logic functions such as NOT, NOR, OR, AND and NAND, but cannot be used to emulate the XOR function (2 layers of neurons are required).

Single layer perceptrons (SLPs)

- Single layer perceptrons (SLPs) can be used to emulate logic functions such as NOT, NOR, OR, AND and NAND, but cannot be used to emulate the XOR function (2 layers of neurons are required).

- A bias is also commonly applied to each neuron, which is added to the weighted sum of the inputs. It determines the level of incoming activations (value of weighted inputs) that are required in order for the neuron to fire.

SINGLE LAYER PERCEPTRONS (SLPS)

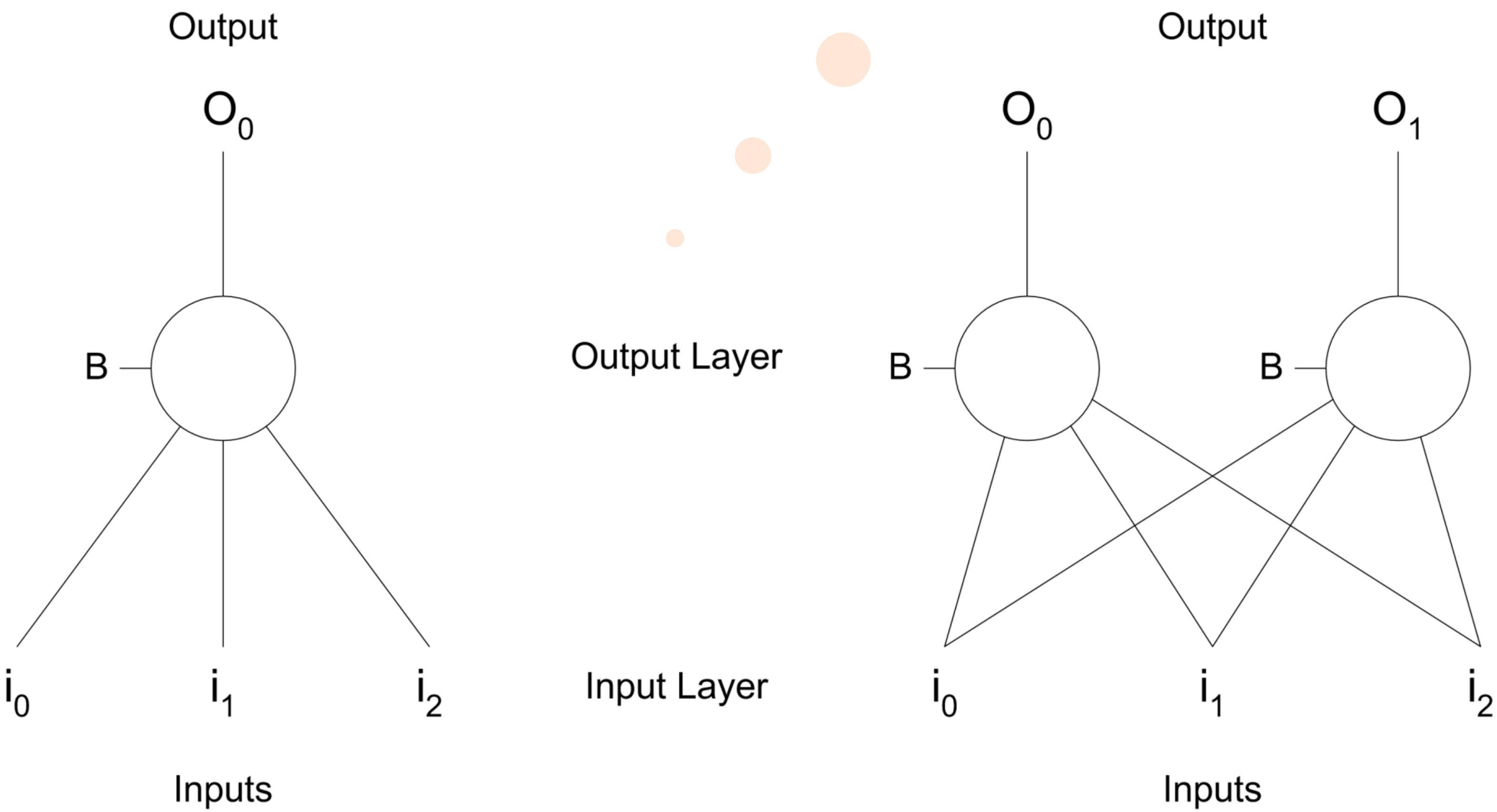
- Single layer perceptrons (SLPs) can be used to emulate logic functions such as NOT, NOR, OR, AND and NAND, but cannot be used to emulate the XOR function (2 layers of neurons are required).

- A bias is also commonly applied to each neuron, which is added to the weighted sum of the inputs. It determines the level of incoming activations (value of weighted inputs) that are required in order for the neuron to fire.

- A bias is commonly set to 1. A weight is also applied to the bias which can be tuned by the learning algorithm.

Examples of SLPs

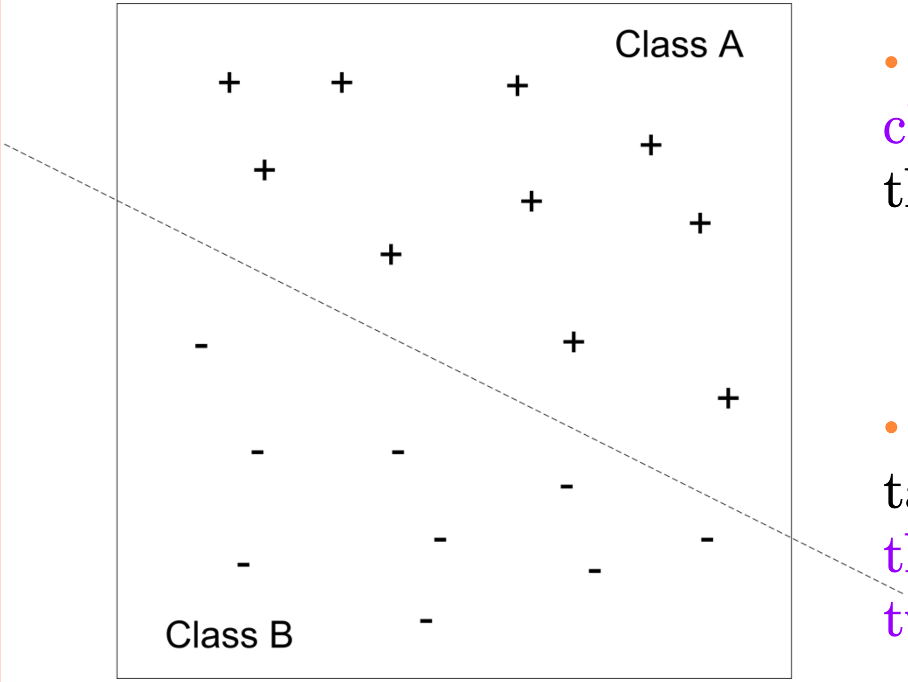
All consisting of
a single layer



The perceptron learning

- A perceptron is a single neuron neural network that was first introduced by Rosenblatt in the late 1950s.
- It is a simple model for neural networks that can be used for a certain class of simple problems called linear separable problems.
- It is often used to classify whether a pattern belongs to one of two classes.

An example problem that a perceptron can solve



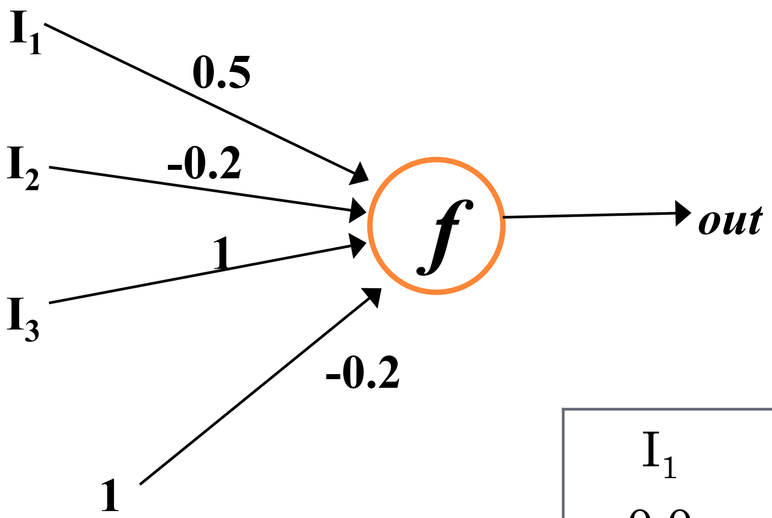
- A perceptron has the ability to classify the data into two classes if the data is *linearly separable*.

- Given the set of possible inputs, the task then is to identify the weights that correctly classify the data into two classes.

The Perceptron

- The simplest form of neural network
- Used as a *decision system*
  - *ie* a two-class classifier
- Thresholded step activation function
- Binary (or sometimes bipolar) output
- http://en.wikipedia.org/wiki/Perceptron

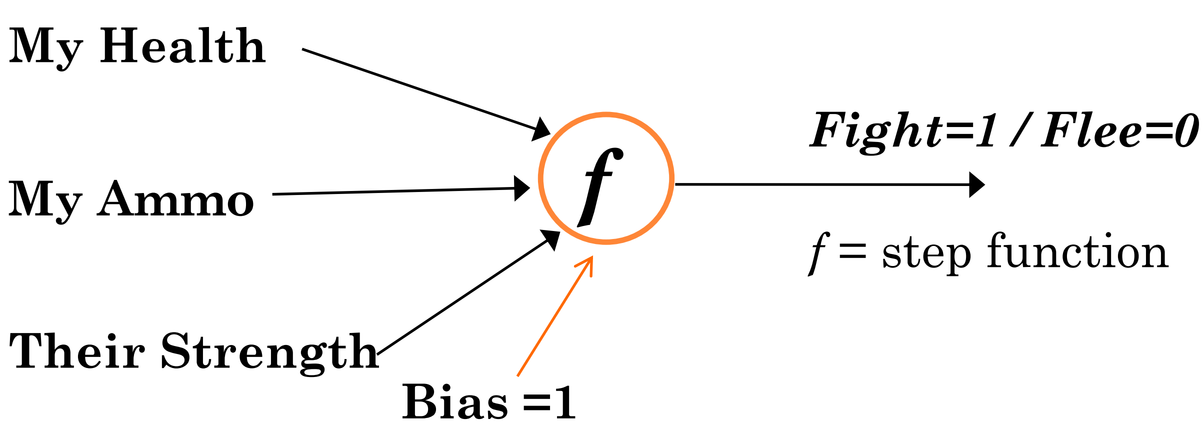
Perceptron Example



I_1	I_2	I_3	sum	out
0.0	1.0	0.5	0.1	1
-0.2	0.0	-1.0	-1.3	0
0.2	-0.5	0.2	0.2	1
-0.8	0.2	-0.6	-1.24	0

Games Example

Input			Output
Your Health	*Your Ammo*	*Their Strength*	*Action*
0.73	0.34	0.49	*Fight*
0.70	0.09	0.66	*Flee*
0.49	0.60	0.61	*Flee*
0.12	0.03	0.31	*Fight*
0.46	0.90	0.91	*Flee*
0.29	0.98	0.34	*Fight*
0.01	0.11	0.55	?



Perceptron Learning

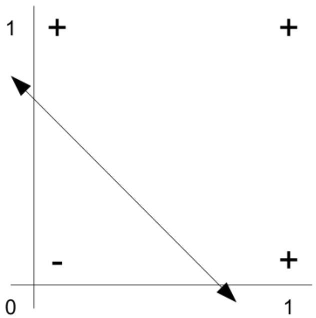
- How do we get a perceptron to learn to classify examples correctly?
- Present each of the examples in the training set
- See what output you get
- Adjust the weights to get the output ‘more right’
- Until it does what you want

Training a Perceptron

1. Start weights at random
2. Present inputs and calculate outputs
3. Find error compared with desired output
4. Adjust weights
5. Repeat 2-4 until:
   - *Either* got the outputs you want
   - *Or* results not getting any better
6. Then use network to make predictions

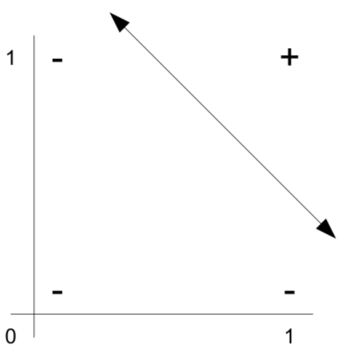
USING THE PERCEPTRON FOR SIMPLE BOOLEAN FUNCTIONS

- The perceptron can accurately classify the standard boolean functions, such as AND, OR, NAND, and NOR. But the XOR function is linearly inseparable.



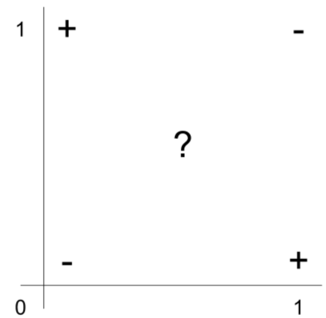
Or

0	0	0
0	1	1
1	0	1
1	1	1



And

0	0	0
0	1	0
1	0	0
1	1	1



XOR

0	0	0
0	1	1
1	0	1
1	1	0

A bias provides the offset of the line from the origin.

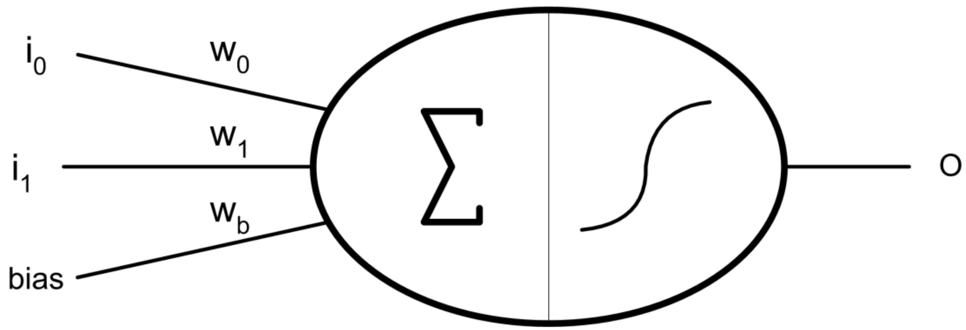
Perceptron learning algorithm (1)

- Each sample from the training set is applied to the perceptron, and the error (expected result minus the actual result) is used to adjust the weights.

- A learning rate is also applied (small number between 0 and 1) to minimize the changes that are applied at each step.

- The bias is normally set to 1, but the weight for the bias will be adjusted to alter its affect.

SIMPLE PERCEPTRON USED FOR BINARY FUNCTION CLASSIFICATION



- Calculating the output of the perceptron can be defined as the following:

$$R = step\ (i_0 w_0 + i_1 w_1 + w_b)$$

The step function simply pushes the result to 1.0 if it exceeds a threshold; otherwise, the result is -1.0.

Perceptron learning algorithm (2)

- The weights are adjusted using the following equation:

$$w_i = w_i + \alpha * T * i_i$$

- In this equation:
  -- α is the learning rate (between 0 and 1);
  -- T is the target (or expected) result;
  -- i_i is the input value for the current weight w_i.

- The algorithm continues until no changes are made to the weights because all tests are properly classified.

Another Learning Rule

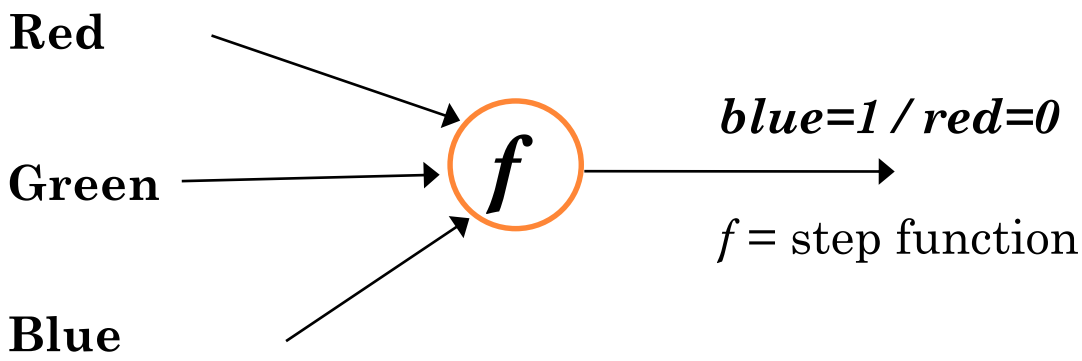
$$error = output - target$$

$Learning\ rate = 0.01$ (for example)

$$W_{new} = W_{old} + (rate \times error \times input)$$

Tutorial task – building a perceptron neural network

- This perceptron learning has three RGB values as inputs.

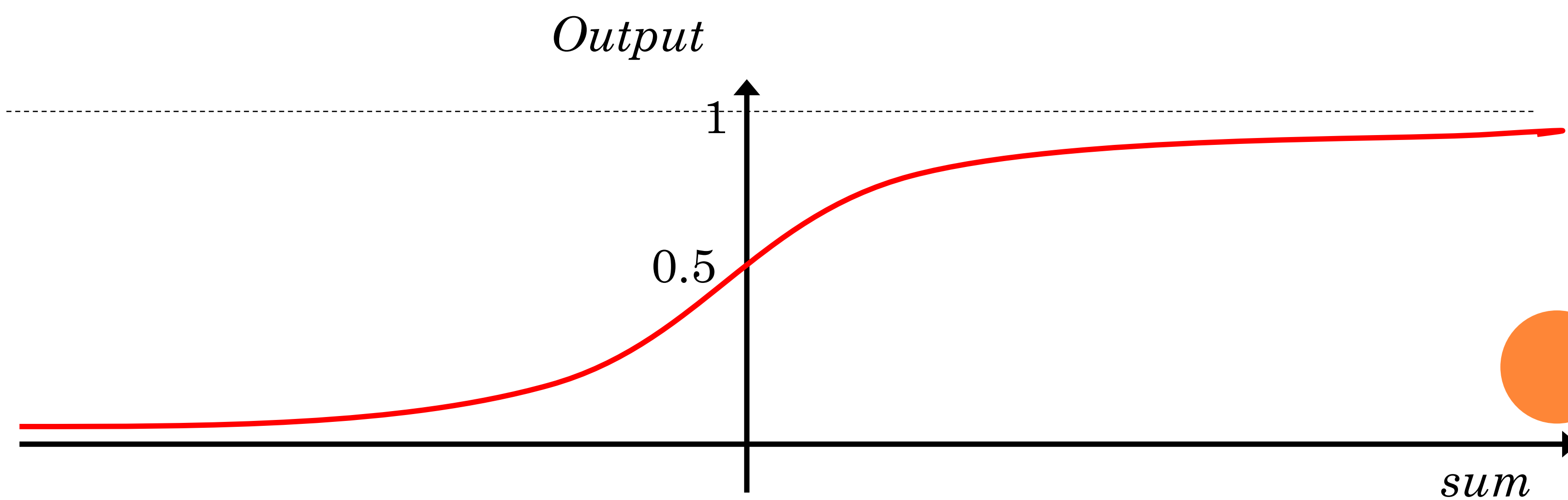


- The output layer has one node to indicate the color that the three input values represent. Sigmoid function is used to finalise the output of the perceptron learning.

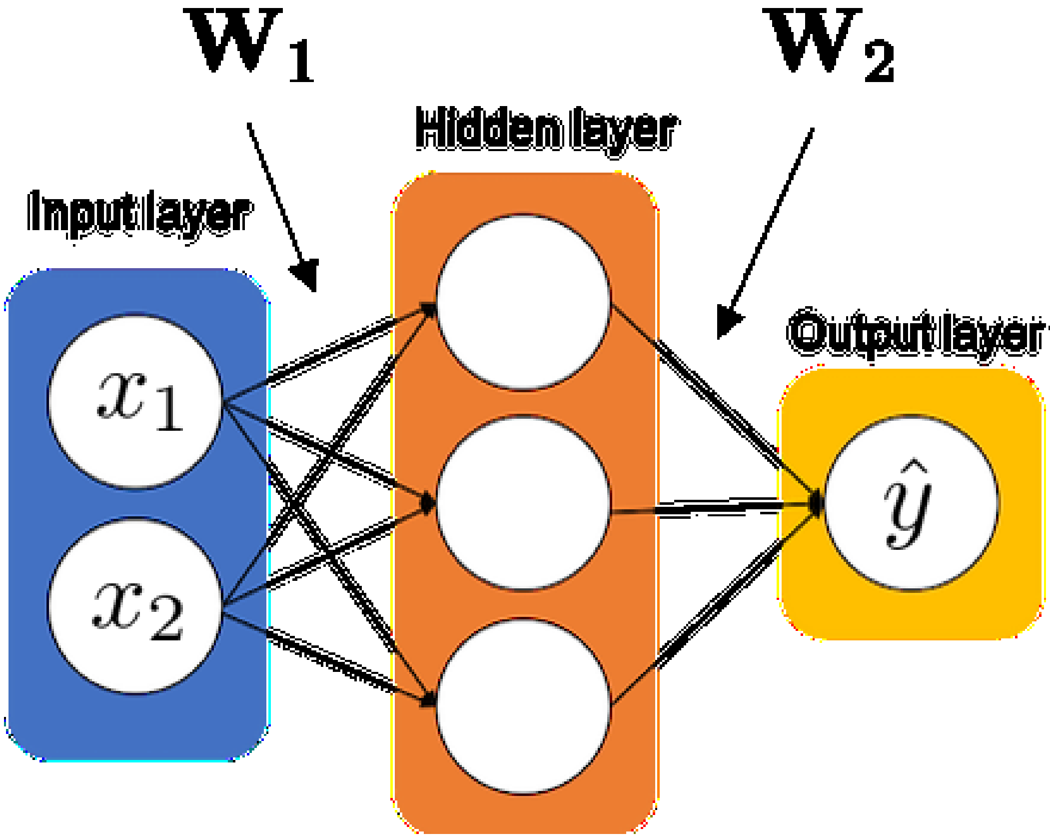
OUTPUT: SIGMOIDAL (SQUASHING) FUNCTION

- Produces continuous output in a limited range (0,1)
- Two *asymptotes*:
  - *output→1 as sum→∞*
  - *output→0 as sum→-∞*

$$output = \frac{1}{1+e^{-sum}}$$



Example of a multiple layer neural network



Backpropagation and Training

- Batch-based Forward Pass
  - Train dataset is large and we can split it to batches,
    - 1000 data samples = 100 batches * 10 samples per batch

Backpropagation and Training

- Batch-based Forward Pass
  - Train dataset is large and we can split it to batches,
    - 1000 data samples = 100 batches * 10 samples per batch
    - Put all 10 samples $\{x_i\}$ together, forms a matrix $\mathbf{X}$
    - For the MLP architecture with two layers, we have,

      $\hat{\mathbf{y}} = \sigma(\mathbf{X}\mathbf{W}_1)\mathbf{W}_2$, where: $\mathbf{H} := \mathbf{X}\mathbf{W}_1$, $\mathbf{A} := \sigma(\mathbf{H})$
    - $\mathbf{W_1}$ are weights in layer 1 and $\mathbf{W_2}$ in layer 2
    - σ is the activation function

Backpropagation and Training

- Batch-based Forward Pass
  - Train dataset is large and we can split it to batches,
    - 1000 data samples = 100 batches * 10 samples per batch
    - Put all 10 samples $\{x_i\}$ together, forms a matrix $\mathbf{X}$
    - For the MLP architecture with two layers, we have,

      $\hat{\mathbf{y}} = \sigma(\mathbf{X}\mathbf{W}_1)\mathbf{W}_2$, where: $\mathbf{H} := \mathbf{X}\mathbf{W}_1$, $\mathbf{A} := \sigma(\mathbf{H})$
    - $\mathbf{W_1}$ are weights in layer 1 and $\mathbf{W_2}$ in layer 2
    - σ is the activation function
- Batch-based Loss Function:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}||\hat{y}_i - y_i||_2^2$$

Backpropagation and Training

- Batch-based Backpropagation:
  - Layer 2:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_2} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{W}_2}, \quad \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} = \left[\frac{\partial \mathcal{L}}{\partial \hat{y}_1}, \ldots, \frac{\partial \mathcal{L}}{\partial \hat{y}_N} \right], \quad \frac{\partial \mathcal{L}}{\partial \hat{y}_i} = \frac{1}{N} (\hat{y}_i - y_i)$$

Backpropagation and Training

- Batch-based Backpropagation:
  - Layer 2:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_2} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{W}_2}, \quad \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} = \left[\frac{\partial \mathcal{L}}{\partial \hat{y}_1}, \ldots, \frac{\partial \mathcal{L}}{\partial \hat{y}_N} \right], \quad \frac{\partial \mathcal{L}}{\partial \hat{y}_i} = \frac{1}{N}(\hat{y}_i - y_i)$$

  - Layer 1:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{W}_1} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \underset{\substack{\Uparrow \\ W_2}}{\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{A}}} \underset{\Uparrow}{\frac{\partial \mathbf{A}}{\partial \mathbf{H}}} \underset{\substack{\Uparrow \\ \mathbf{X}}}{\frac{\partial \mathbf{H}}{\partial \mathbf{W}_1}}$$

sigmoid $\longrightarrow$ $\dfrac{\partial \sigma(\mathbf{x})}{\partial \mathbf{x}} = \sigma(\mathbf{x})(1 - \sigma(\mathbf{x}))$

Backpropagation and Training

- Batch-based Backpropagation:
  - Layer 2:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_2} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{W}_2}, \quad \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} = \left[\frac{\partial \mathcal{L}}{\partial \hat{y}_1}, \ldots, \frac{\partial \mathcal{L}}{\partial \hat{y}_N} \right], \quad \frac{\partial \mathcal{L}}{\partial \hat{y}_i} = \frac{1}{N}(\hat{y}_i - y_i)$$

  - Layer 1:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{W}_1} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \underbrace{\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{A}}}_{W_2} \underbrace{\frac{\partial \mathbf{A}}{\partial \mathbf{H}}}_{\text{sigmoid}} \underbrace{\frac{\partial \mathbf{H}}{\partial \mathbf{W}_1}}_{\mathbf{X}}$$

sigmoid $\rightarrow$ $$\frac{\partial \sigma(\mathbf{x})}{\partial \mathbf{x}} = \sigma(\mathbf{x})(1 - \sigma(\mathbf{x}))$$

- Update weights with gradients & learning rates

$$W_{new} = W_{old} + rates \times grads$$

BACKPROPAGATION AND TRAINING

- Batch-based Backpropagation:
  - Layer 2:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_2} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{W}_2}\ , \quad \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} = \left[\frac{\partial \mathcal{L}}{\partial \hat{y}_1}, \ldots, \frac{\partial \mathcal{L}}{\partial \hat{y}_N}\right], \quad \frac{\partial \mathcal{L}}{\partial \hat{y}_i} = \frac{1}{N}(\hat{y}_i - y_i)$$

  - Layer 1:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{W}_1} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{A}} \frac{\partial \mathbf{A}}{\partial \mathbf{H}} \frac{\partial \mathbf{H}}{\partial \mathbf{W}_1}$$

W_2 $\quad$ $\mathbf{X}$

sigmoid $\longrightarrow$ $\frac{\partial \sigma(\mathbf{x})}{\partial \mathbf{x}} = \sigma(\mathbf{x})(1 - \sigma(\mathbf{x}))$

- Update weights with gradients & learning rates

$$W_{new} = W_{old} + rates \times grads$$

- Consider bias: $y = WX+b$, how to do BP training?

Lab Session

- MLP implementation for XOR gate

- Your excercise - expand the code for
  - Epoch, batches, & including bias in your model
  - Test it for datasets: MINIST, ECG

- Your coursework is a report based on labs.