Introduction to Machine Learning

Presentation Title

H M Sabbir Ahmad

Ph.D., Systems Engineering.

09/12/2024

What is Machine Learning?

"Learning is any process by which a system improves performance from experience."

- Herbert Simon

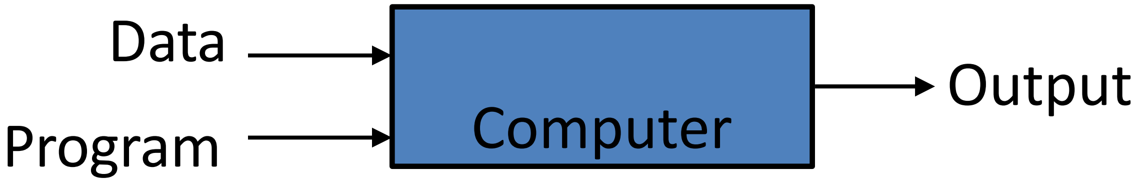
Definition by Tom Mitchell (1998):

Machine Learning is the study of algorithms that

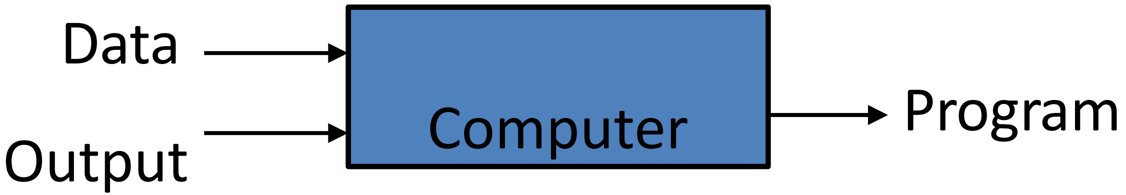
- improve their performance P
- at some task T
- with experience E.

A well-defined learning task is given by $<P, T, E>$.

Traditional Programming



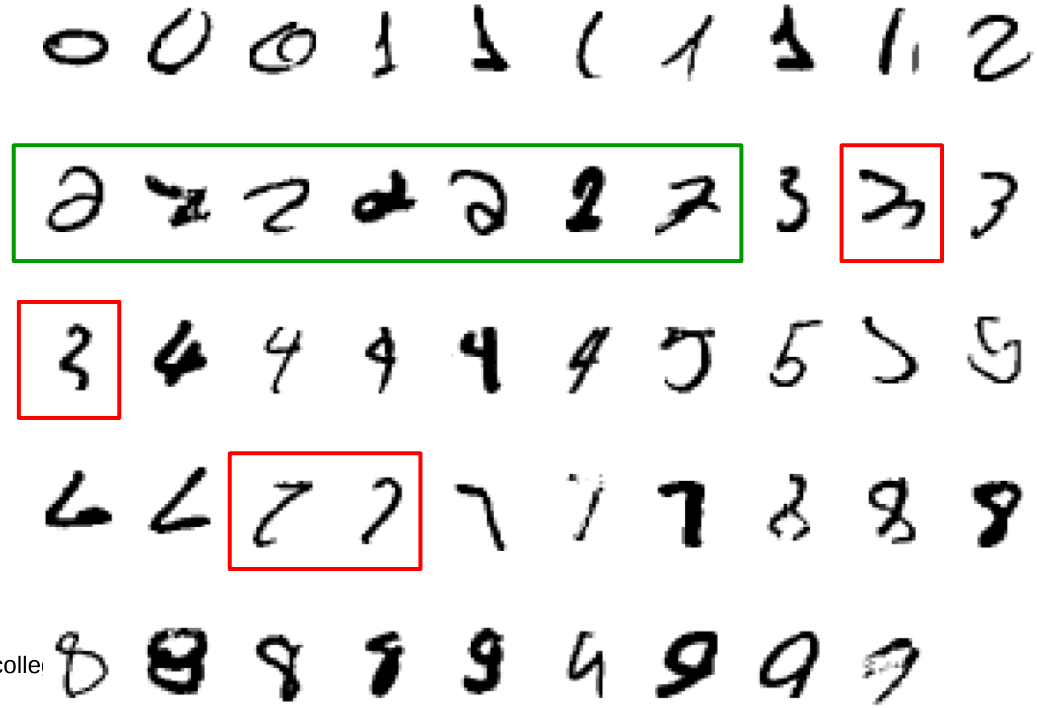
Machine Learning



Slide credit: Pedro Domingos

What is Machine Learning?

A classic example of a task that requires machine learning: It is very hard to say what makes a 2



Slide credit: Geoffrey Hinton

Some more examples of tasks that are best solved by using a learning algorithm

- Recognizing patterns:
  - Facial identities or facial expressions
  - Handwritten or spoken words
  - Medical images
- Generating patterns:
  - Generating images or motion sequences
- Recognizing anomalies:
  - Unusual credit card transactions
  - Unusual patterns of sensor readings in a nuclear power plant
- Prediction:
  - Future stock prices or currency exchange rates

Slide credit: Geoffrey Hinton

Samuel's Checkers-Player

"Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed."   -Arthur Samuel (1959)

Types of Learning

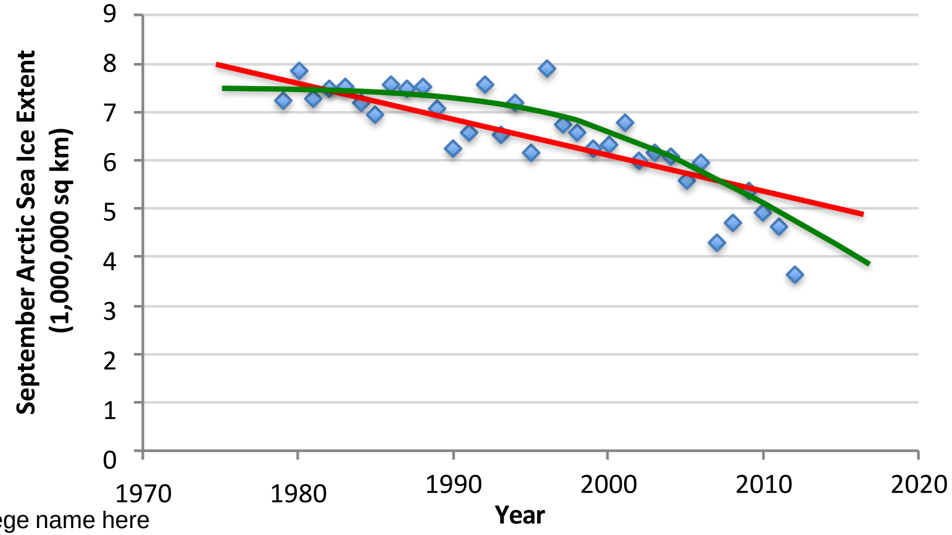
- Supervised (inductive) learning
  - Given: training data + desired outputs (labels)
- Unsupervised learning
  - Given: training data (without desired outputs)
- Semi-supervised learning
  - Given: training data + a few desired outputs
- Reinforcement learning

– Rewards from sequence of actions

Slide credit: Pedro Domingos

Supervised Learning: Regression

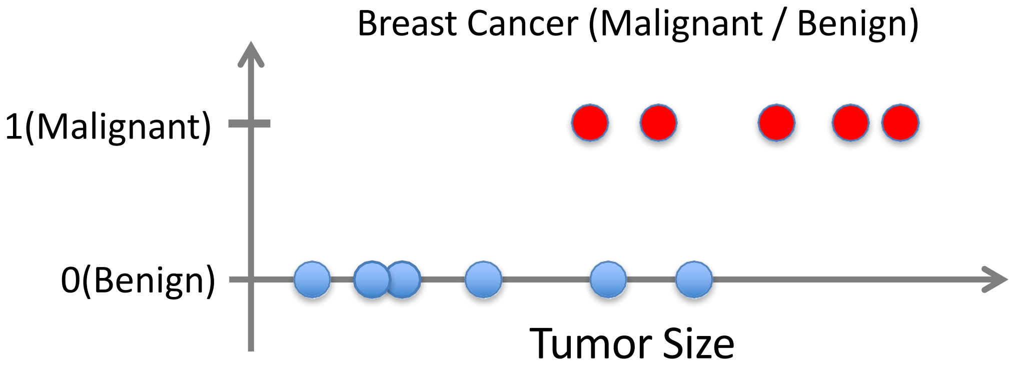
- Given $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
- Learn a function $f(x)$ to predict y given x
  - y is real-valued == regression



Data from G. Witt. Journal of Statistics Education, Volume 21, Number 1 (2013)

Supervised Learning: Classification

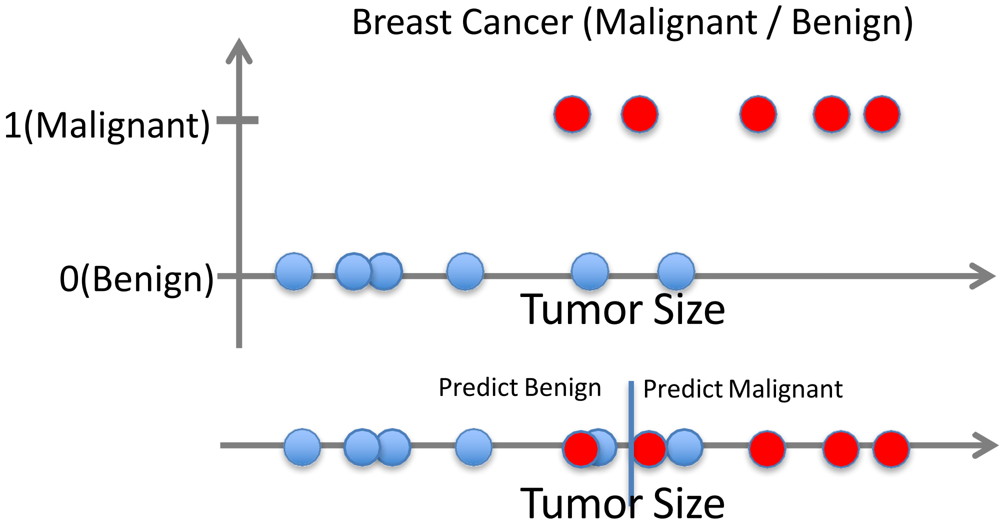
- Given $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
- Learn a function $f(x)$ to predict y given x
  - y is categorical == classification



Slide credit: Andrew Ng

Supervised Learning: Classification

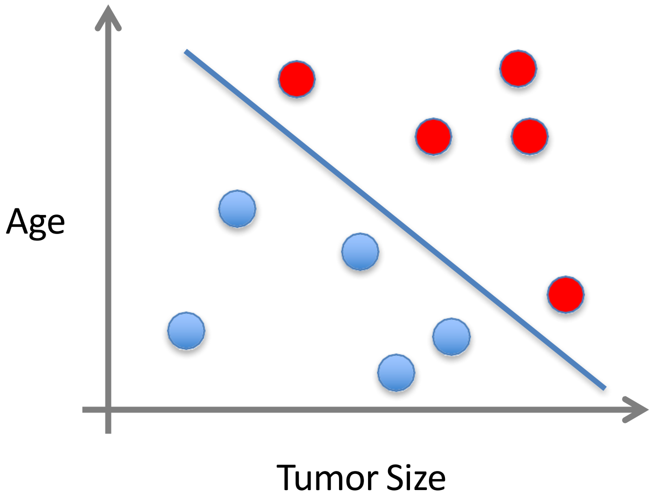
- Given $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
- Learn a function $f(x)$ to predict y given x
  - y is categorical == classification



Slide credit: Andrew Ng

Supervised Learning

- x can be multi-dimensional
  - Each dimension corresponds to an attribute

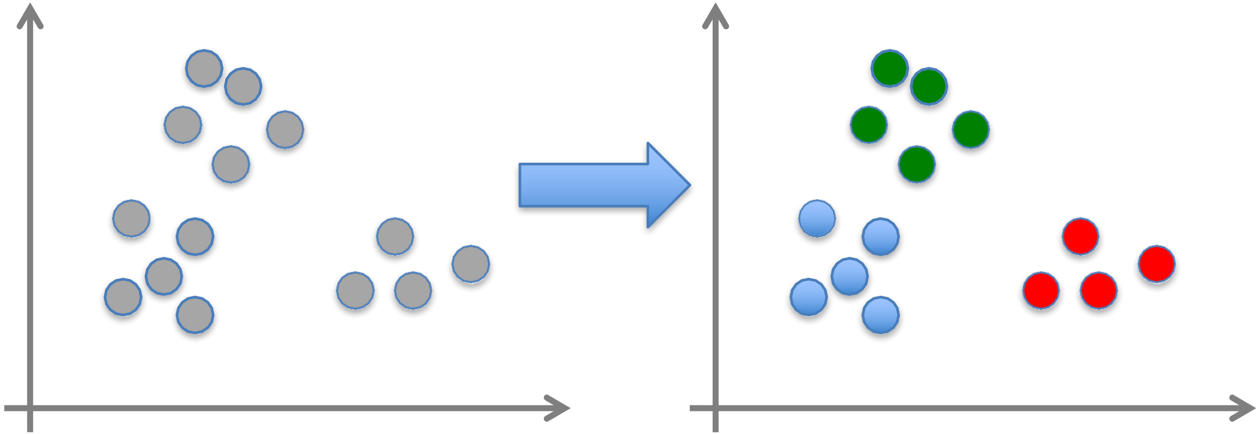


- Clump Thickness
- Uniformity of Cell Size
- Uniformity of Cell Shape

…

Unsupervised Learning

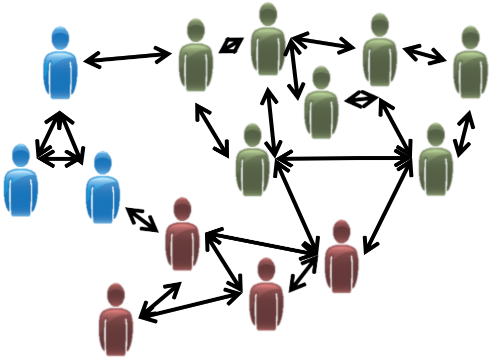
- Given $x_1, x_2, \ldots, x_n$ (without labels)
- Output hidden structure behind the x's
  - E.g., clustering



Slide credit: Andrew Ng

Unsupervised Learning

Organize computing clusters



Social network analysis

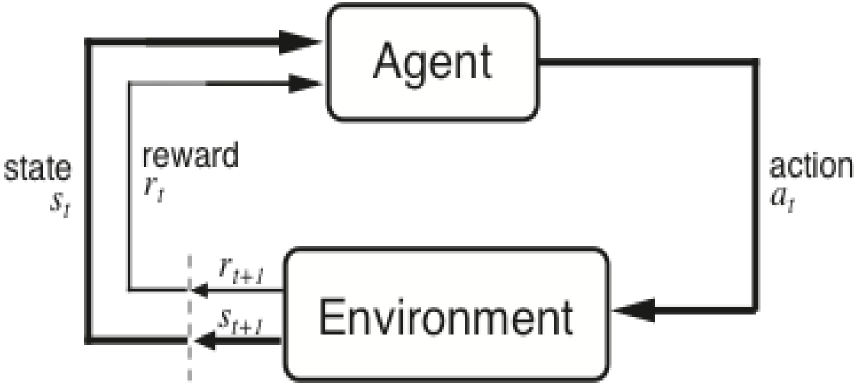
Market segmentation

Slide credit: Andrew Ng

Reinforcement Learning

- Given a sequence of states and actions with (delayed) rewards, output a policy
  - Policy is a mapping from states → actions that tells you what to do in a given state
- Examples:
  - Game playing
  - Robot in a maze
  - Balance a pole on your hand

The Agent-Environment Interface



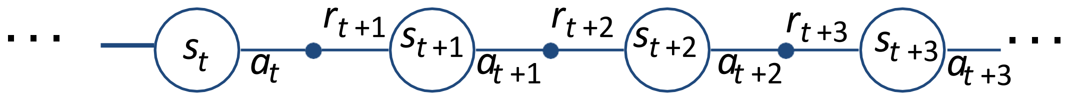
Agent and environment interact at discrete time steps : $t = 0, 1, 2, \ldots$

Agent observes state at step t: $s_t \in S$

produces action at step t: $a_t \in A(s_t)$

gets resulting reward : $r_{t+1} \in \Re$

and resulting next state : s_{t+1}



Slide credit: Sutton & Barto

Reinforcement Learning

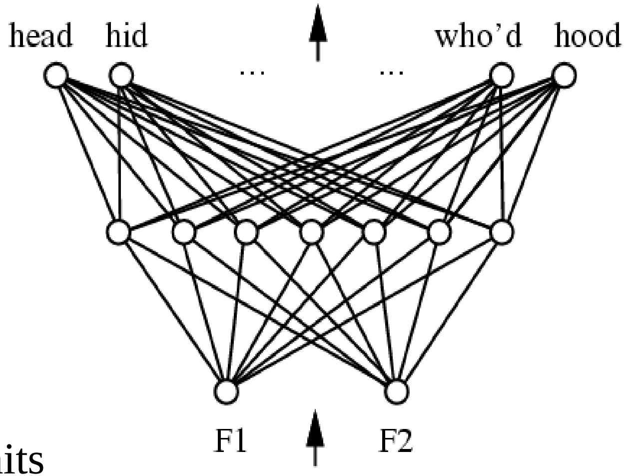
https://www.youtube.com/watch?v=4cgWya-wjgY

Neural Networks
Supervised learning

Artificial Neural Networks to learn f: X → Y

- $f_{\mathbf{w}}$ typically a non-linear function, $f_{\mathbf{w}}$: X → Y

- X feature space: (vector of) vars
- Y output space: (vector of) vars
- $f_{\mathbf{w}}$ *network* of basic units



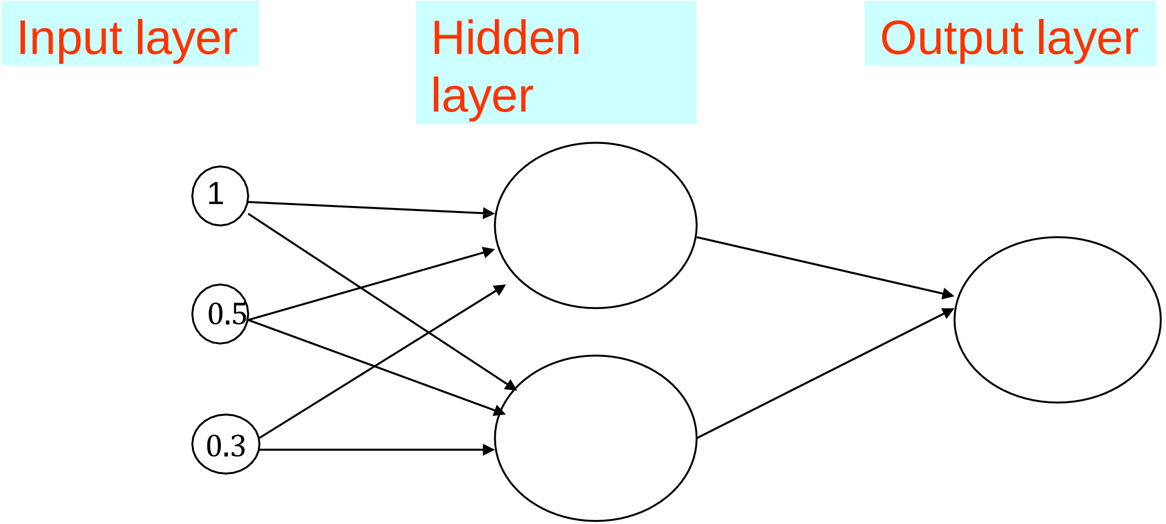
Learning algorithm: given $(x_d, t_d)_{d \in D}$, train weights w of all units to minimize sum of squared errors of predicted network outputs.

Find parameters w to minimize $\sum_{d \in D} (f_w(x_d) - t_d)^2$

Use gradient descent!

What type of units should we use?

- Classifier is a multilayer *network of units.*
- Each *unit* takes some inputs and produces one output. Output of one unit can be the input of another.



Slide Credit: Tom Mitchell and Maria Florina Balcan

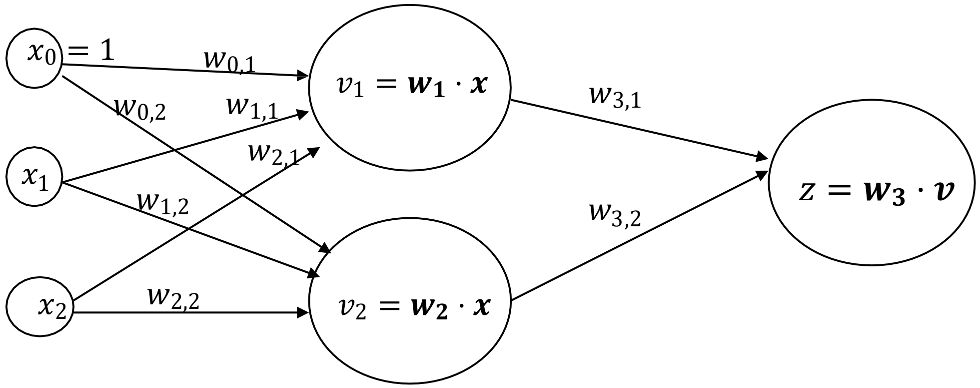
Multilayer network of Linear units?

- Advantage: we know how to do gradients on linear units

Input layer

Hidden layer

Output layer



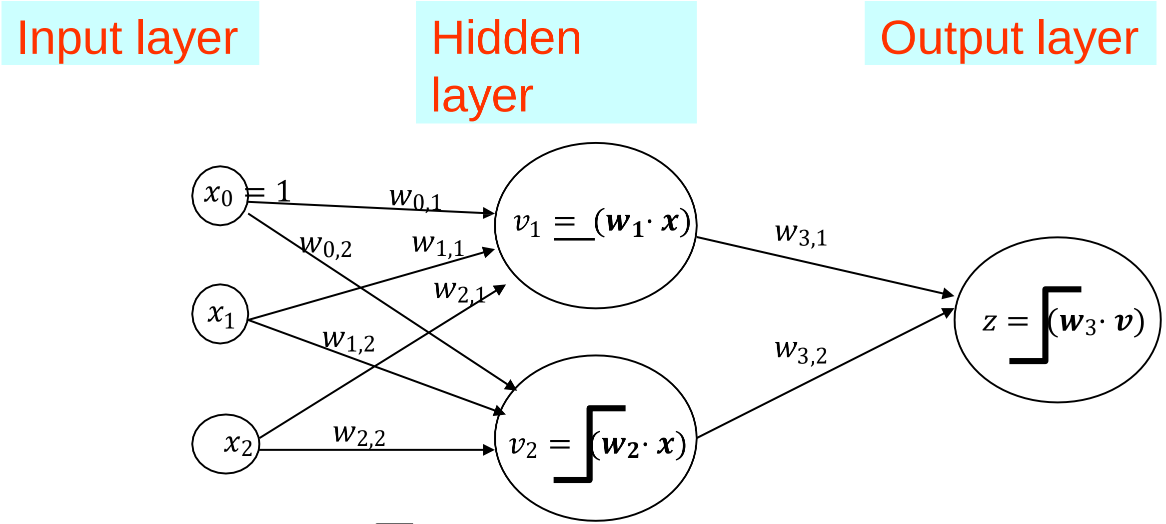
Problem: linear of linear is just linear.

$$z = w_{3,1} \; \boldsymbol{w_1} \cdot \boldsymbol{x} \; + w_{3,2} \; \boldsymbol{w_2} \cdot \boldsymbol{x} \; = \left(w_{3,1}\boldsymbol{w_1} + w_{3,2}\boldsymbol{w_2}\right) \cdot \boldsymbol{x} = \text{linear}$$

Slide Credit: Tom Mitchell and Maria Florina Balcan

Multilayer network of Perceptron units?

- Advantage: Can produce highly non-linear decision boundaries!



Threshold function: $x = 1$ if x is positive, 0 if x is negative.

Problem: discontinuous threshold is not differentiable. Can't do gradient descent.

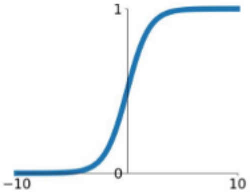
BOSTON UNIVERSITY

Slide Credit: Tom Mitchell and Maria Florina Balcan

Activation Functions

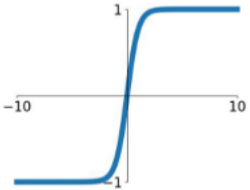
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



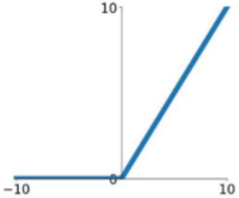
tanh

$\tanh(x)$



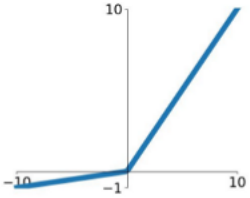
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

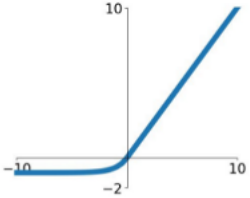


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

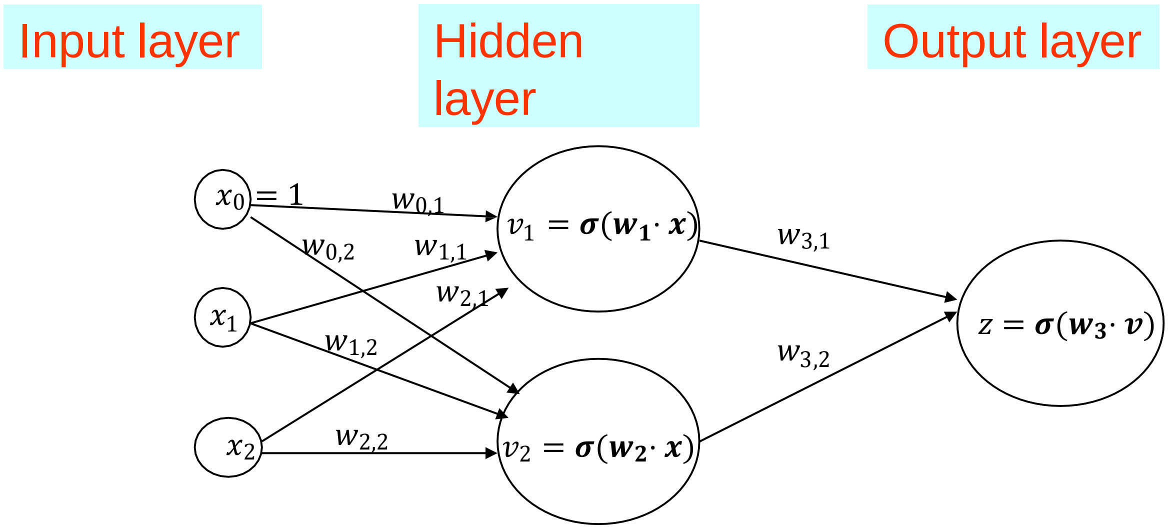
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



BOSTON UNIVERSITY

Multilayer network of sigmoid units

- Advantage: Can produce highly non-linear decision boundaries!
- Sigmoid is differentiable, so can use gradient descent

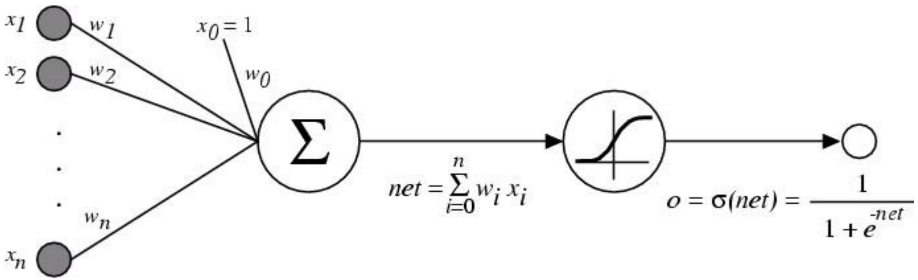


$$\boldsymbol{\sigma}(x) = \frac{1}{1 + e^{-x}} =$$ 

Very useful in practice!

Slide Credit: Tom Mitchell and Maria Florina Balcan

The Sigmoid Unit



σ is the sigmoid function; $\sigma(x) = \frac{1}{1+e^{-x}}$

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

We can derive gradients to train and learn the parameters:

- One sigmoid unit
- *Multilayer networks* of sigmoid units → Backpropagation

Gradient Descent to Minimize Squared Error

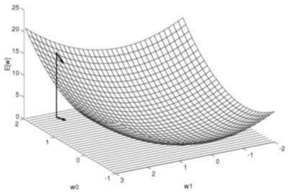
Goal: Given $(x_d, t_d)_{d \in D}$ find w to minimize $E_D[w] = \Sigma(f_w(x_d)\text{-}t_d)^2$

Mode 1

Batch mode Gradient Descent:

Do until satisfied

1. Compute the gradient $\nabla E_D[\boldsymbol{w}]$
2. $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla E_D[\boldsymbol{w}]$



$$\nabla E[\boldsymbol{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n}\right]$$

Mode 2

Incremental (stochastic) Gradient Descent: Do until satisfied

- For each training example d in D

1. Compute the gradient $\nabla E_d[\boldsymbol{w}]$
2. $\mathbf{w} \leftarrow \boldsymbol{w} - \eta \nabla E_d[\boldsymbol{w}]$

Note: Incremental Gradient Descent can approximate *Batch Gradient Descent* arbitrarily closely if η made small enough

BOSTON UNIVERSITY

Slide Credit: Tom Mitchell and Maria Florina Balcan

Gradient descent in weight space

Goal: Given $(x_d, t_d)_{d \in D}$ find w to minimize $E_D[w] = \frac{1}{2}\Sigma(f_w(x_d) - t_d)^2$

This error measure defines a surface over the "weight" space

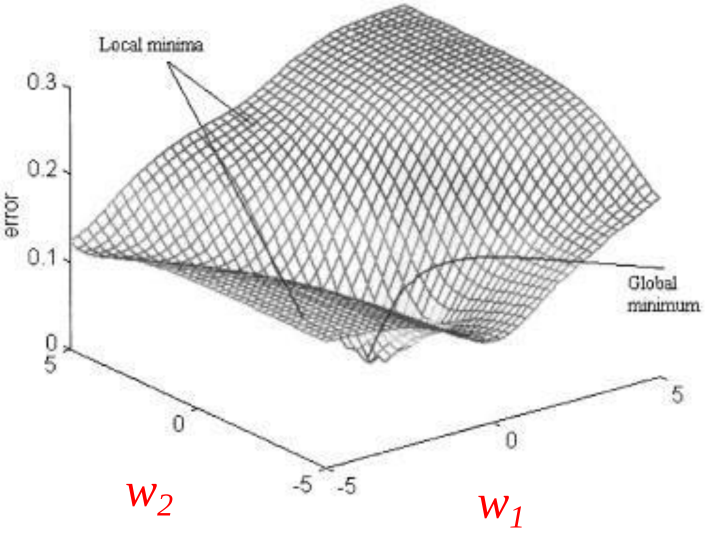


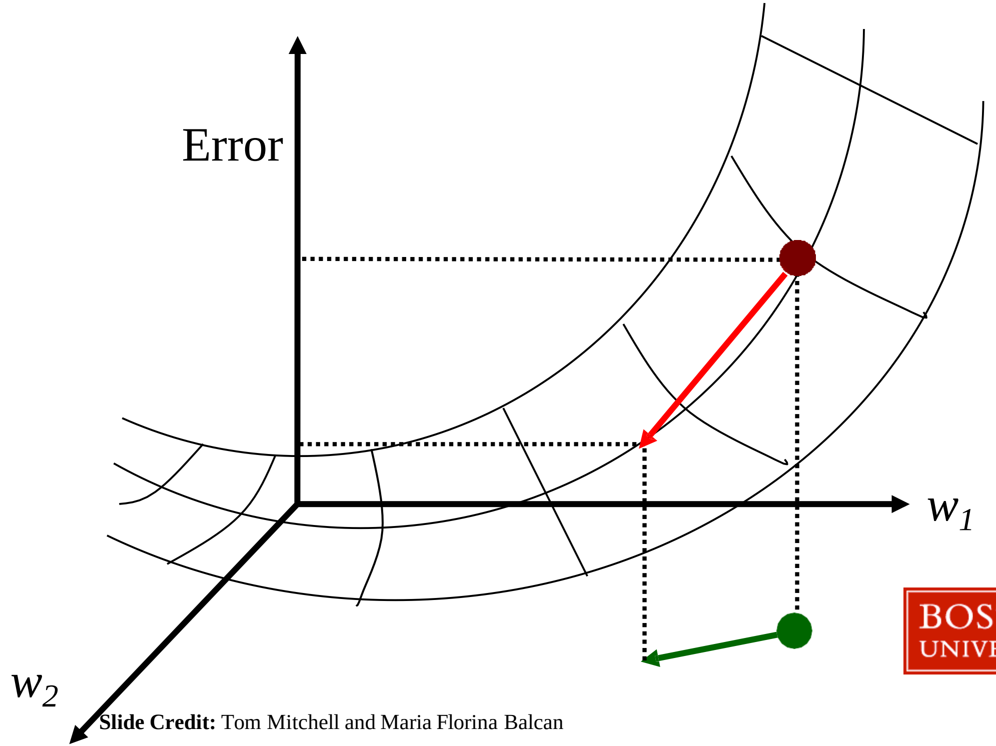
figure from Cho & Chow, *Neurocomputing* 1999

Gradient descent in weight space

Gradient descent is an iterative process aimed at finding a minimum in the error surface.

on each iteration

- current weights define a point in this space
- find direction in which error surface descends most steeply
- take a step (i.e. update weights) in that direction



Slide Credit: Tom Mitchell and Maria Florina Balcan

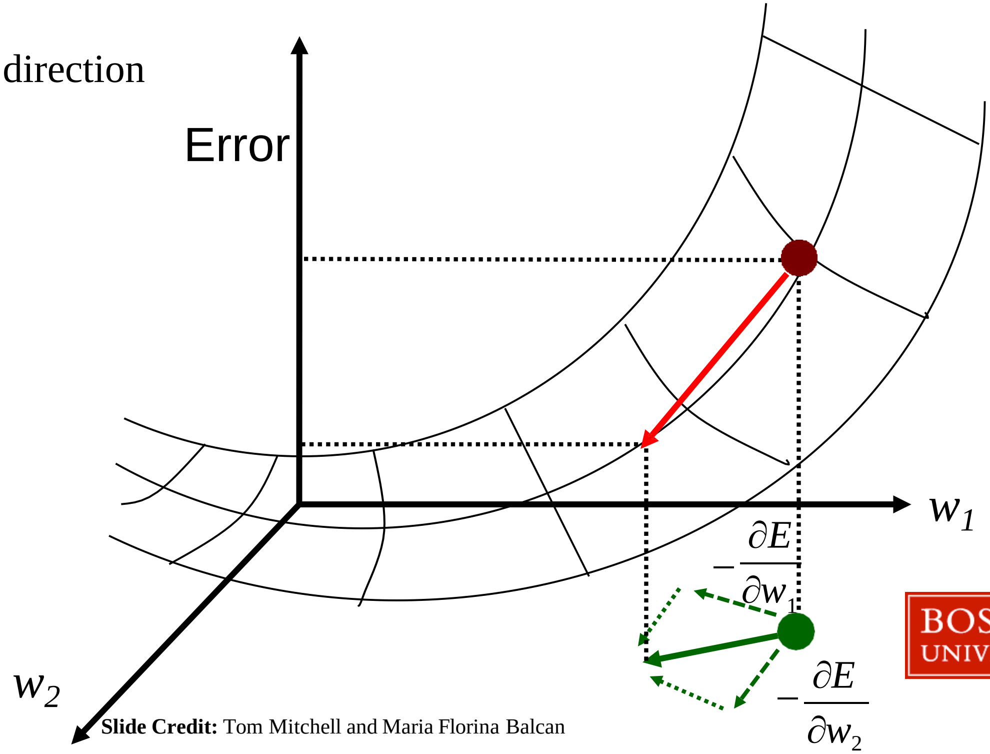
Gradient descent in weight space

Calculate the gradient of E: $\nabla E(\boldsymbol{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n} \right]$

Take a step in the opposite direction

$$\Delta \boldsymbol{w} = -\eta \, \nabla E(\boldsymbol{w})$$

$$\Delta w_i = -\eta \, \frac{\partial E}{\partial w_i}$$



Slide Credit: Tom Mitchell and Maria Florina Balcan

Taking derivative: chain rule

Recall the chain rule from calculus

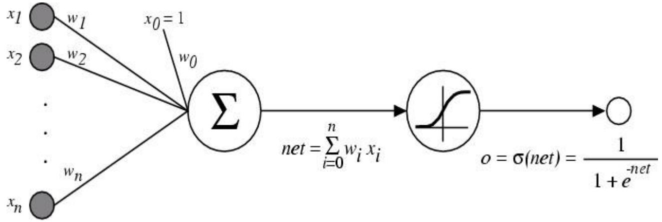
$$y = f(u)$$
$$u = g(x)$$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial x}$$

Slide Credit: Tom Mitchell and Maria Florina Balcan

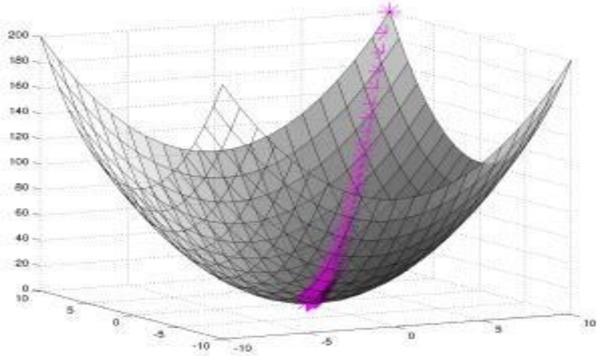
Gradient Descent for the Sigmoid Unit

Given $(x_d, t_d)_{d\in D}$ find **w** to minimize $\Sigma_{d\in D}(o_d - t_d)^2$



o_d =observed output for x_d

$o_d = \sigma(net_d); net_d = \sum_i w_i x_{i,d}$



$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2 = \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_{d\in D} 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) = \sum_{d\in D}(t_d - o_d)\left(-\frac{\partial o_d}{\partial w_i}\right)$$

$$= -\sum_{d\in D}(t_d - o_d)\frac{\partial o_d}{\partial net_d}\frac{\partial net_d}{\partial w_i}$$

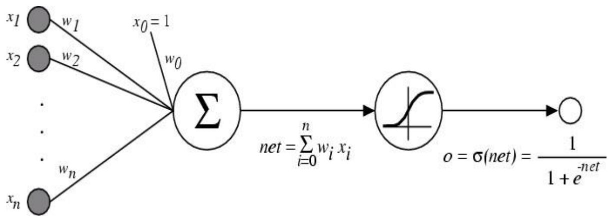
But we know: $\frac{\partial o_d}{\partial net_d} = \frac{\partial \sigma(\text{ net }_d)}{\partial net_d} = o_d(1 - o_d)$ and $\frac{\partial net_d}{\partial w_i} = \frac{\partial(\mathbf{w}\cdot\mathbf{x}_d)}{\partial w_i} = x_{i,d}$

So: $\frac{\partial E}{\partial w_i} = -\sum_{d\in D}(t_d - o_d)o_d(1 - o_d)x_{i,d}$

Slide Credit: Tom Mitchell and Maria Florina Balcan

Gradient Descent for the Sigmoid Unit

Given $(x_d, t_d)_{d \in D}$ find **w** to minimize $\sum_{d \in D}(o_d - t_d)^2$



o_d =observed output for x_d

$o_d = \sigma(net_d); net_d = \sum_i w_i x_{i,d}$

$$\frac{\partial E}{\partial w_i} = -\sum_{d \in D} (t_d - o_d)\, o_d(1 - o_d) x_{i,d}$$

δ_d error term $t_d - o_d$ multiplied by $o_d(1 - o_d)$ that comes from the derivative of the sigmoid function

$$\frac{\partial E}{\partial w_i} = -\sum_{d \in D} \delta_d x_{i,d}$$

Update rule: $w \leftarrow w - \eta \nabla E[w]$

Slide Credit: Tom Mitchell and Maria Florina Balcan

Backpropagation Algorithm

Incremental/stochastic gradient descent

Initialize all weights to small random numbers.

Until satisfied, Do:

- **For** each training example (x, t) **do**:
  1. Input the training example to the network and compute the network outputs
  2. For each output unit k:

     $$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

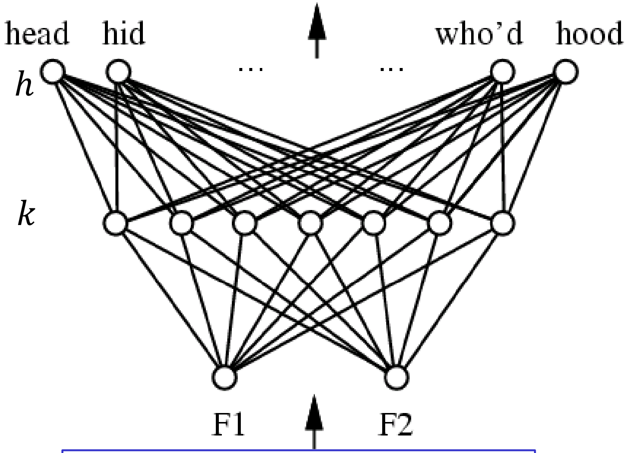
  3. For each hidden unit h:

     $$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

  4. Update each network weight $w_{i,j}$

     $$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

     where $\Delta w_{i,j} = \eta \delta_j x_{i,j}$



o = observed unit output

t = target output

x = input

$x_{i,j}$ = ith input to jth unit

w_{ij} = wt from i to j

Slide Credit: Tom Mitchell and Maria Florina Balcan

The End

Thanks for your attention.

I would be glad if you have any question.