Introduction to Machine Learning

Lecture 8
Supervised Learning

- Artificial Neural Networks
- Backpropagation Algorithm

- Some materials are courtesy of Vibhave Gogate, Eric Xing and Tom Mitchell.
- All pictures belong to their creators.

Supervised Learning

- $< x, y >$
- Estimate $\Pr[y|x]$ by Bayesian Theory: $\Pr[y|x] = \frac{\Pr[x|y]\Pr[y]}{\Pr[x]}$
- Naïve Bayes.

- Estimate $\Pr[y|x]$ directed by assuming a certain form
- Logistic regression.

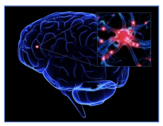
- Assume the form of $y = f(x)$
- Error-driven.

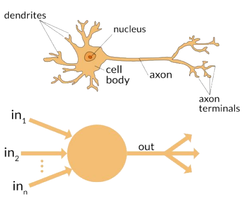
$\Pr[x|y]$ and $\Pr[y]$ are hard to compute.
No good form for $\Pr[y|x]$
No prior Knowledge on $f(x)$.

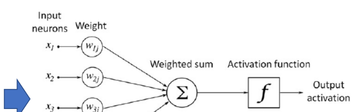
You can try Neural Networks.

Neural Networks

- Input x, output y
- Given x, how to compute y?
- x and y are generally real vectors.



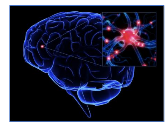


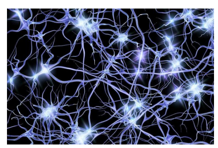


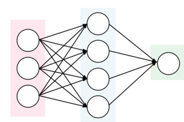
perceptron
Sigmoid unit

Neural Networks

- Input x, output y
- Given x, how to compute y?
- x and y are generally real vectors.

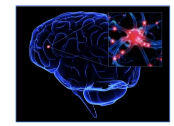




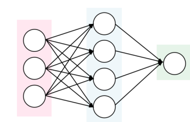


Neural Networks

- Input x, output y
- Given x, how to compute y?
- x and y are generally real vectors.



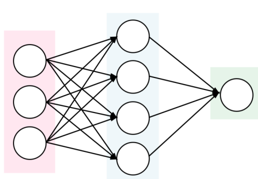
- Many neuron-like units (perceptron, sigmoid)
- Weighted interconnections between units.
- Parallel distributed process.



Neural Networks

- Input x, output y
- Given x, how to compute y?
- x and y are generally real vectors.

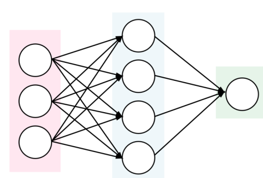
- Design a neural network:
- What is the function within each unit?
- How are the units connected?



Neural Networks

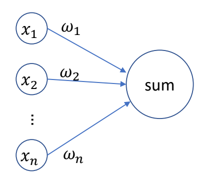
- Input x, output y
- Given x, how to compute y?
- x and y are generally real vectors.

- Units:
- Input Unit
- Processing Units
  - Receive input from other units
  - Output result to other units
- Edges
  - weights



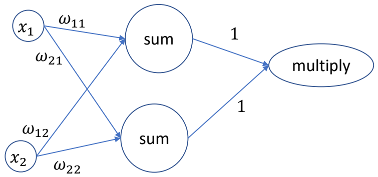
Examples

- Linear function $y = \omega_1 x_1 + \cdots + \omega_n x_n$
- Input $(x_1, \dots, x_n)$ output y.



Examples

- Quadratic function $y = (\omega_{11}x_1 + \omega_{12}x_2)(\omega_{21}x_1 + \omega_{22}x_2)$
- Input $(x_1, \dots, x_n)$ output y.

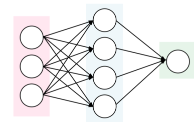


Train a Neural Network

- Define the error E
- Update the weight ω of each edge to minimize E
- $\omega \leftarrow \omega - \eta \frac{\partial E}{\partial \omega}$ (delta rule)

Calculate $\frac{\partial E}{\partial \omega}$

- Batch Mode
  - E: the total error among training data.
  - Consider all the training data each time.
- **Incremental Mode**
  - **E: the error for one training instance.**
  - **Consider training instances one by one.**

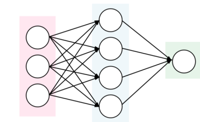


Train a Neural Network

- Define the error E
- Update the weight ω of each edge to minimize E
- $\omega \leftarrow \omega - \eta \frac{\partial E}{\partial \omega}$ (delta rule)

Calculate $\frac{\partial E}{\partial \omega}$

- **Incremental Mode**
  - **E: the error for one training instance.**
  - **Consider training instances one by one.**



Neural Networks

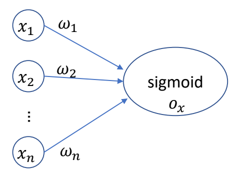
- One layer of sigmoid with one output
- Two layers of single functions
- Two layers of multiple sigmoid units with one output.
- Two layers of multiple sigmoid units with multiple outputs.

- Goal: how to find the parameters that can minimize the error.

Neural Networks

- One layer of sigmoid with one output

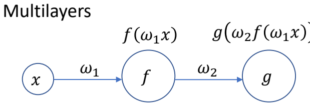
$$o(x_1, \dots, x_n) = \frac{1}{1 + e^{-\omega \cdot x}}$$



$$\frac{\partial E_x}{\partial \omega_i} = -(t_x - o_x)\frac{\partial o_x}{\partial \omega_i} = -(t_x - o_x) \cdot x_i \cdot o_x(1 - o_x)$$

Neural Networks

- Multilayers



(x, t_x)

$$E = \frac{1}{2}\left|t_x - g\right|^2$$

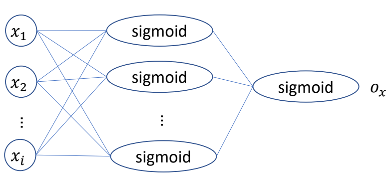
$$\frac{\partial E}{\partial \omega_2} = -(t_x - g)\frac{\partial g}{\partial \omega_2}$$
$$\frac{\partial g}{\partial \omega_2} = \frac{\partial g}{\partial \omega_2 f}\frac{\partial \omega_2 f}{\partial \omega_2} = \frac{\partial g}{\partial \omega_2 f} f$$

$$\frac{\partial E}{\partial \omega_1} = -(t_x - g)\frac{\partial g}{\partial \omega_1}$$
$$\frac{\partial g}{\partial \omega_1} = \frac{\partial g}{\partial \omega_2 f}\frac{\partial \omega_2 f}{\partial \omega_1}$$
$$= \frac{\partial g}{\partial \omega_2 f}\omega_2 \frac{\partial f}{\partial \omega_1}$$
$$= \frac{\partial g}{\partial \omega_2 f}\omega_2 \frac{\partial f}{\partial \omega_1 x}\frac{\partial \omega_1 x}{\partial \omega_1}$$
$$= \frac{\partial g}{\partial \omega_2 f}\omega_2 \frac{\partial f}{\partial \omega_1 x} x$$

- Update ω_1 or ω_2 first?

Neural Networks

- Two layers of multiple sigmoid units with one output.



For each x in D
$E_x = \frac{1}{2}(t_x - o_x)^2$

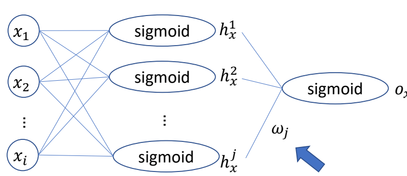
today's weather → tomorrow's temp (normalized)

x $f(x, \omega)$ $t_x = f(x)$

ω: weight on each edge

Neural Networks

- Two layers of multiple sigmoid units with one output.

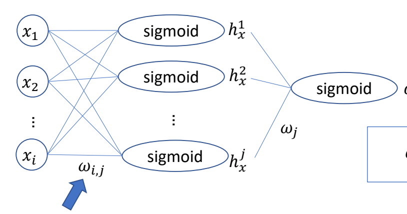


For each x in D
$E_x = \frac{1}{2}(t_x - o_x)^2$

$$\frac{\partial E_x}{\partial \omega_j} = -(t_x - o_x)\frac{\partial o_x}{\partial \omega_j} = -(t_x - o_x) \cdot h_x^j \cdot o_x(1 - o_x)$$

Neural Networks

- Two layers of multiple sigmoid units with one output.



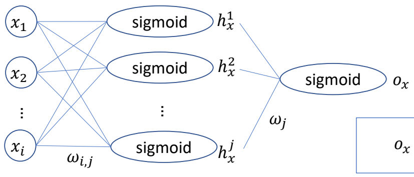
For each x in D
$E_x = \frac{1}{2}(t_x - o_x)^2$

$$o_x = \frac{1}{1 + e^{-\Sigma \omega_j \cdot h_x^j}}$$

$$\frac{\partial E_x}{\partial \omega_{i,j}} = -(t_x - o_x)\frac{\partial o_x}{\partial \omega_{i,j}} = -(t_x - o_x)\frac{\partial o_x}{\partial h_x^j}\frac{\partial h_x^j}{\partial \omega_{i,j}}$$

Neural Networks

- Two layers of multiple sigmoid units with one output.



For each x in D
$E_x = \frac{1}{2}(t_x - o_x)^2$

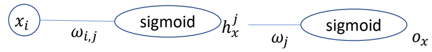
$$o_x = \frac{1}{1 + e^{-\Sigma \omega_j \cdot h_x^j}}$$

$$\frac{\partial o_x}{\partial h_x^j} = \omega_j \cdot o_x(1 - o_x)$$

$$\frac{\partial h_x^j}{\partial \omega_{i,j}} = x_i \cdot h_x^j(1 - h_x^j)$$

Neural Networks

- Two layers of multiple sigmoid units with one output.



$$\frac{\partial E_x}{\partial \omega_j} = -(t_x - o_x) \cdot o_x(1 - o_x) \cdot h_x^j$$

$$\frac{\partial E_x}{\partial \omega_{i,j}} = -(t_x - o_x)\, \omega_j \cdot o_x(1 - o_x)\, x_i \cdot h_x^j(1 - h_x^j)$$

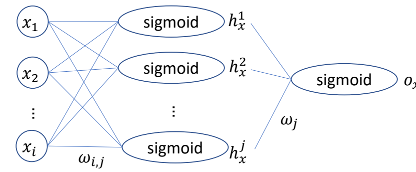
$\omega \leftarrow \omega + \Delta\omega$
$\Delta\omega = -\eta \frac{\partial E}{\partial \omega}$
$\Delta\omega_j = \eta\, h_x^j \delta_x^o$
$\Delta\omega_{i,j} = \eta\, x_i \delta_x^{h,j}$

$\delta_x^o = (t_x - o_x)\, o_x(1 - o_x)$
$\delta_x^{h,j} = h_x^j(1 - h_x^j)\omega_j \delta_x^o$

$$\frac{\partial E_x}{\partial \omega_j} = -h_x^j \delta_x^o \qquad \frac{\partial E_x}{\partial \omega_{i,j}} = -x_i \delta_x^{h,j}$$

Neural Networks

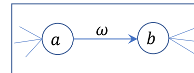
- Two layers of multiple sigmoid units with one output.



$\omega \leftarrow \omega + \Delta\omega$
$\Delta\omega_j = \eta\, h_x^j\, \delta_x^o$
$\Delta\omega_{i,j} = \eta\, x_i \delta_x^{h,j}$

$$\delta = -\frac{\partial E}{\partial\, unit}$$

h_x^j and x_i, input of the edge
δ_x^o and δ_x^j, error term associated with the units

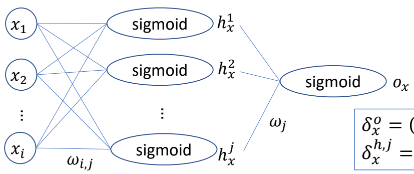


Input from a
Error term $\frac{\partial E}{\partial b}$ on b
$\Delta\omega = \eta a \frac{\partial E}{\partial b}$

Neural Networks

- Two layers of multiple sigmoid units with one output.

$\omega \leftarrow \omega + \Delta\omega$
$\Delta\omega_j = \eta\, h_x^j \delta_x^o$
$\Delta\omega_{i,j} = \eta\, x_i \delta_x^{h,j}$



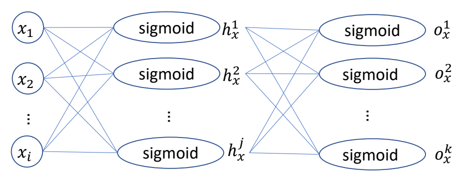
$\delta_x^o = (t_x - o_x)\, o_x(1 - o_x)$
$\delta_x^{h,j} = h_x^j(1 - h_x^j)\omega_j \delta_x^o$

For each x in D
Forward, calculate h_x^j and o_x
Backward, calculate δ_x^o and $\delta_x^{h,j}$
Update ω_j and $\omega_{i,j}$

Backpropagation Framework

Neural Networks

- Two layers of multiple sigmoid units with multiple outputs.



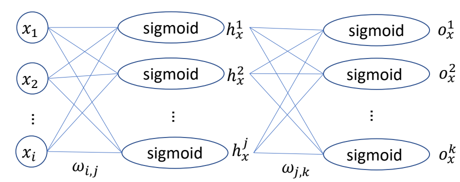
today's weather → tomorrow's weather (normalized)

x — $f(x, \omega)$ — $t_x = f(x)$

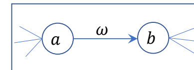
ω: weight on each edge

Neural Networks

- Two layers of multiple sigmoid units with multiple outputs.



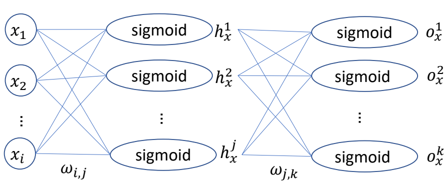
For each x in D
$$E_x = \frac{1}{2}\sum_k (t_x^k - o_x^k)^2$$



Input from a
Error term $\frac{\partial E}{\partial b}$ on b
$\Delta\omega = \eta a \frac{\partial E}{\partial b}$

Neural Networks

- Two layers of multiple sigmoid units with multiple outputs.



For each x in D
$$E_x = \frac{1}{2}\sum_k (t_x^k - o_x^k)^2$$

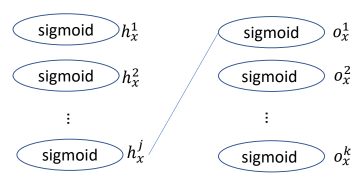
$\Delta\omega_{j,k} = \eta\, h_x^j\, \delta_x^{o,k}$

One ouput: $\delta_x^o = (t_x - o_x)\, o_x(1 - o_x)$

Multiple output: $\delta_x^{o,k} = (t_x^k - o_x^k)\, o_x^k(1 - o_x^k)$

Neural Networks

- Two layers of multiple sigmoid units with multiple outputs.



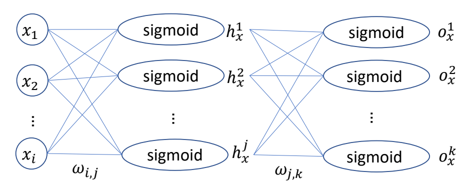
One ouput: $\delta_x^{h,j} = h_x^j(1 - h_x^j)\, \omega_j\, \delta_x^o$

Multiple output: $\delta_x^{h,j} = h_x^j(1 - h_x^j) \sum_k \omega_{j,k} \delta_x^{o,k}$

If not fully connected, sum over the connected units.

Neural Networks

- Two layers of multiple sigmoid units with multiple outputs.



For each x in D
$$E_x = \frac{1}{2}\sum_k (t_x^k - o_x^k)^2$$

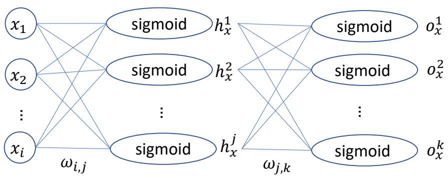
$\Delta\omega_{i,j} = \eta\, x_i\, \delta_x^{h,j}$

One ouput: $\delta_x^{h,j} = h_x^j(1 - h_x^j)\, \omega_j\, \delta_x^o$

Multiple output: $\delta_x^{h,j} = h_x^j(1 - h_x^j) \sum_k \omega_{j,k} \delta_x^{o,k}$

Neural Networks

- Two layers of multiple sigmoid units with multiple outputs.



$\omega \leftarrow \omega + \Delta\omega$
$\Delta\omega_{j,k} = \eta \; h_x^j \; \delta_x^{o,k}$
$\Delta\omega_{i,j} = \eta \; x_i \; \delta_x^{h,j}$

Backpropagation Framework

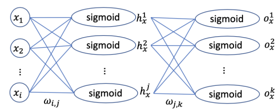
For each x in D
Forward, calculate h_x^j and o_x^k
Backward, calculate $\delta_x^{o,k}$ and $\delta_x^{h,j}$
Update $\omega_{j,k}$ and $\omega_{i,j}$

$\delta_x^{o,k} = (t_x^k - o_x^k)\, o_x^k (1 - o_x^k)$
$\delta_x^{h,j} = h_x^j (1 - h_x^j)\, \sum_k \omega_{j,k} \delta_x^{o,k}$

Neural Networks

- Two layers of multiple sigmoid units with multiple outputs.
- Initialize the weights with random values.
- Do until converge
  - For each $x \in D$
    - (Forward) Calculate h_x^j and o_x^k
    - (Backward) calculate $\delta_x^{o,k}$ and $\delta_x^{h,j}$
    - Update $\omega_{j,k}$ and $\omega_{i,j}$

$\omega \leftarrow \omega + \Delta\omega$
$\Delta\omega_{j,k} = \eta \; h_x^j \; \delta_x^{o,k}$
$\Delta\omega_{i,j} = \eta \; x_i \; \delta_x^{h,j}$

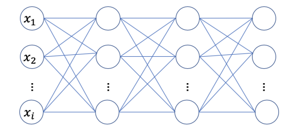


$\delta_x^{o,k} = (t_x^k - o_x^k)\, o_x^k (1 - o_x^k)$
$\delta_x^{h,j} = h_x^j (1 - h_x^j)\, \sum_k \omega_{j,k} \delta_x^{o,k}$

Neural Networks

- Train a neural network with:
- One sigmoid units.
- Two layers of multiple sigmoid units with one output.
- Two layers of multiple sigmoid units with multiple outputs.

- Backpropagation framework: any units, any acyclic graph.
- **Update the weight from right to left.**

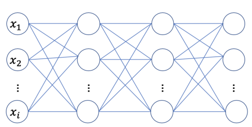


Neural Networks

- Backpropagation framework

- Good news: using the network is fast.
- Bad news: training the network is slow
- More bad news: it may converge to local minima.
- More good news: it performs well in practice.

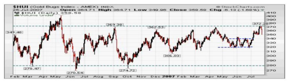
- To avoid local minima
- Add a momentum
  - $\Delta\omega_n^* = \Delta\omega_n + \alpha\Delta\omega_{n-1}^*$
- Train with different initializations.



Neural Networks

- Representational Power of Neural Networks.

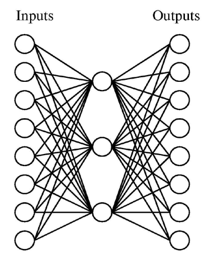
- Boolean functions: every Boolean function can be represented exactly by some network with **two layers** of units.
- Continuous functions: every bounded continuous function can be approximated with arbitrarily small error by a network with **two layers of units**.(sigmoid + linear will do)
- Arbitrary function: any function can be approximated to arbitrary accuracy by a network with **three layers of units**. (sigmoid + sigmoid+ linear will do)



An example (Mitchell)

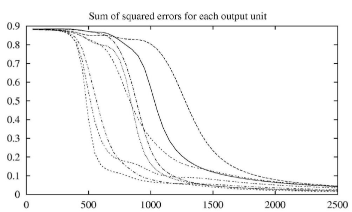
- Learn an identity function with eight training examples.

Input		Output
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001



An example (Mitchell)

- Learn an identity function with eight training examples.

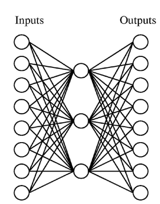


Input		Output
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

An example (Mitchell)

- Learn an identity function with eight training examples.

Input		Hidden Values				Output	
10000000	→	.89	.04	.08	→	10000000	1 0 0
01000000	→	.01	.11	.88	→	01000000	0 0 1
00100000	→	.01	.97	.27	→	00100000	0 1 0
00010000	→	.99	.97	.71	→	00010000	1 1 1
00001000	→	.03	.05	.02	→	00001000	0 0 0
00000100	→	.22	.99	.99	→	00000100	0 1 1
00000010	→	.80	.01	.98	→	00000010	1 0 1
00000001	→	.60	.94	.01	→	00000001	1 1 0



Overfitting

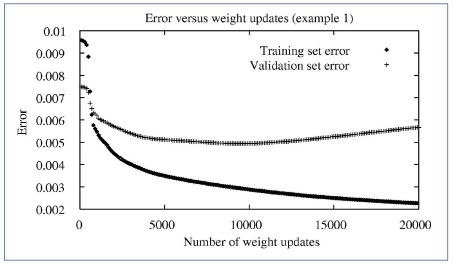
- Overfitting is everywhere…

- Overfitting may happen, when
- The learned model is too complicated
- Fitting data too well when data has noise
- When training set is small

- Avoid large parameters.

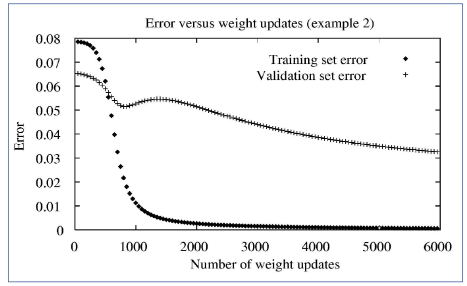
Overfitting

- Overfitting is everywhere…



Overfitting

- Overfitting is everywhere…



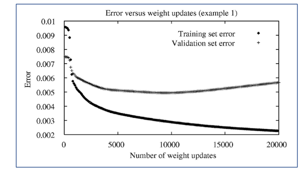
Overfitting

- Overfitting is everywhere…

- Weight Decay:
- Decrease each weight by some small factor during each iteration?

- Penalize large weights:
- $Error = Error + \gamma \sum \omega_i^2$

- Using Validation data.



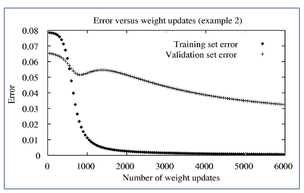
Overfitting

- Overfitting is everywhere…

- Weight Decay:
- Decrease each weight by some small factor during each iteration?

- Penalize large weights:
- $Error = Error + \gamma \sum \omega_i^2$

- Using Validation data.



Summary

- What is an artificial neural network?
- Process the input by the **connected units.**

- How to training a neural network?
- Error-driven.
- Backpropagation algorithm.
  - Two layers of sigmoid units.