CS-F415: Data Mining

14 Neural Network

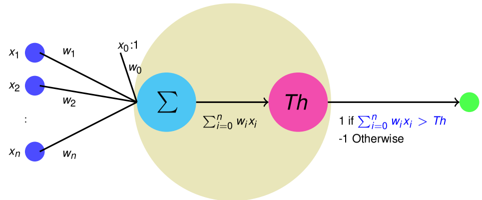
Dr. Kamlesh Tiwari
Associate Professor, Department of CSIS,
BITS Pilani, Pilani Campus, Rajasthan-333031 INDIA

March 04, 2024 M/W/F 4:00pm 6101 @ BITS-Pilani [Jan-May 2024]

http://ktiwari.in/dm

A Single Perceptron

Perceptron representation

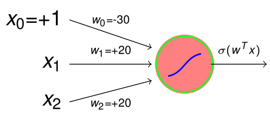


- Any m-of-n function (at least m of the n inputs must be true) can be represented by perceptron. OR (m=1) and AND (m=n)

Two layer NN can represent any boolean function (Consider SOP)

An Example

Consider a perceptron with output 0/1 as below



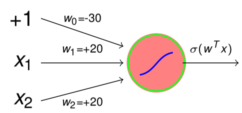
x_1	x_2	Output
0	0	0
0	1	0
1	0	0
1	1	1

This perceptron computes logical AND

- w_0=-10 gives logical OR
- w_0=10, w_1=-20 with single input gives logical NOT
- XOR is not possible (MLP[1] can do it))

[1] MLP (multi layer perceptron) with one-hidden-layer is *universal boolean function*, capable of expressing any truth table

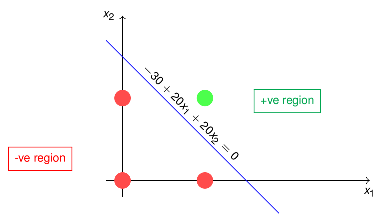
Essentially it Represents A Decision Boundary



Provides positive classification if

$$-30 + 20x_1 + 20x_2 \geq 0$$

Represents a linear decision boundary

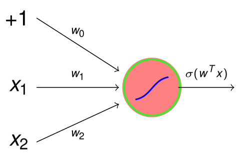


An Example

Design a **perceptron** for

x_1	x_2	Classification
0	0	0
0	1	0
1	0	1
1	1	0

Let us assume following



We have following four equations

$$w_0 + w_1 \times (0) + w_2 \times (0) < 0 \quad (1)$$
$$w_0 + w_1 \times (0) + w_2 \times (1) < 0 \quad (2)$$
$$w_0 + w_1 \times (1) + w_2 \times (0) \geq 0 \quad (3)$$
$$w_0 + w_1 \times (1) + w_2 \times (1) < 0 \quad (4)$$

By (1) $w_0 < 0$ so let $w_0 = -1$
By (2) $w_0+w_2<0$ so let $w_2 = -1$
By (3) $w_0+w_1 \geq 0$ so let $w_1 = 1.5$
By (4) $w_0+w_1+w_2<0$ that is valid

So $(w_0, w_1, w_2) = (-1, -1, 1.5)$

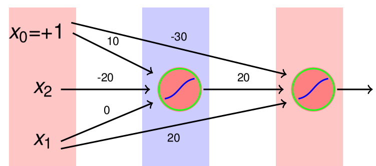
Other possibilities are also there

An Example

Design a **neural network** for

x_1	x_2	Classification
0	0	0
0	1	0
1	0	1
1	1	0

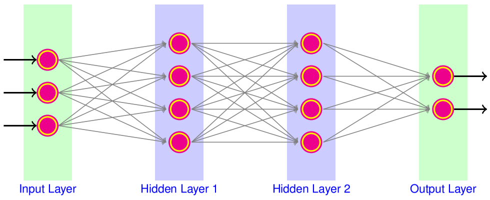
x_1	x_2	$\bar{x}_2$	$(x_1)AND(\bar{x}_2)$
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0



This arrangement is mostly avoided, as training is very challenging

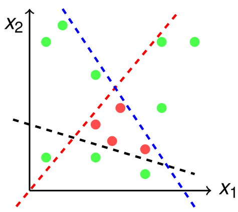
Neural Network

When neurons are interconnected in layers



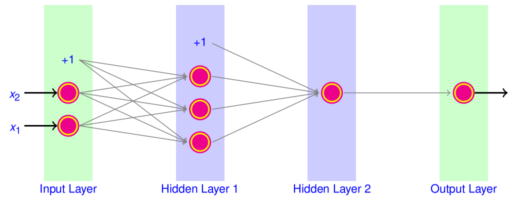
- Number of layers may differ
- Nodes in each intermediate layers may also differ
- Multiple output neurons are used for different class
- **Two levels deep** NN can represent any boolean function

More Example: Design NN for the following data



Whether is it green?

Red-line	Blue-line	Black-line	Color
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



Note: Weights and activation in subsequent layers add power to the model in terms of non linearity.

Neural Network Applications

NN is appropriate for problems with the following characteristics:

- Instances are provided by many attribute-value pairs (more data)
- The target function output may be discrete-valued, real-valued, or a vector of several real or discrete valued attributes
- The training examples may contain errors
- Long training times are acceptable
- Fast evaluation of the target function may be required
- The ability of humans to understand the learned target function is not important

Perceptron Training (delta rule)

When data is not linearly-separable; error fluctuates with *parameter training* updates. It is difficult to decide, when to stop?

- Delta rule converges to a best-fit approximation of the target
- Uses gradient descent
- Consider unthresholded perceptron, $o(\vec{x}) = \vec{w}.\vec{x}$
- Training error is defined as

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

- Gradient would specify direction of steepest increase $\nabla E(\vec{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_n}\right]$
- Weights can be learned as $w_i = w_i - \eta\frac{\partial E}{\partial w_i}$
- It can be seen that $\frac{\partial E}{\partial w_i} = \sum_{d\in D}(t_d - o_d)(-x_{id})$

Perceptron Training (delta rule)

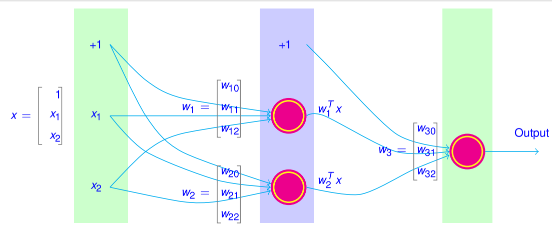
Algorithm 1: Gradient Descent (D, η)

```
Initialize w_i with random weights
repeat
    For each w_i, initialize △w_i = 0
    for each training example d ∈ D do
        Compute output o using model for d whose target is t
        For each w_i, update △w_i = △w_i + η(t−o)x_i
    For each w_i, set w_i = w_i + △w_i
until termination condition is met;
return w
```

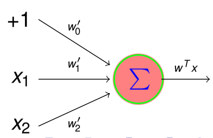
- A date item $d \in D$, is supposed to be multidimensional $d = (x_1, x_2, ..., x_n, t)$
- Algorithm converges toward the minimum error hypothesis.
- Linear programming can also be an approach

Linear Activation is Not Much Interesting

NN with perceptrons have limited capability, even with many layers



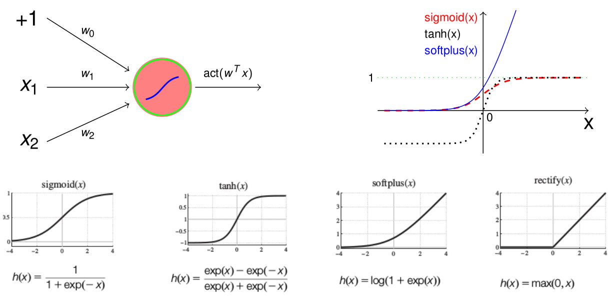
$$\begin{aligned} Output &= w_{30}\times 1 + w_{31}\times(w_1^T x) + w_{32}\times(w_2^T x)\\ &= w_{30}\times 1 + w_{31}\times[w_{10}\times 1 + w_{11}\times x_1 + w_{12}\times x_2]\\ &\quad + w_{32}\times[w_{20}\times 1 + w_{21}\times x_1 + w_{22}\times x_2]\\ &= (w_{30} + w_{31}w_{10} + w_{32}w_{20}) + (w_{31}w_{11} + w_{32}w_{21})\times x_1\\ &\quad + (w_{31}w_{12} + w_{32}w_{22})\times x_2\\ &= w_0' + w_1'\times x_1 + w_2'\times x_2 \end{aligned}$$



Expression of single perceptron

Neuron

Neuron uses nonlinear activation functions (*sigmoid*, *tanh*, *softplus*, *ReLU* leaky *ReLU etc.*) at the place of thresholding



To avoid *dying reLU* problem, leaky-reLU is used $max(0.1x, x)$

Multilayer Networks and Backpropagation

- Single perceptron can only express linear decision surface
- We need units whose output is a nonlinear function of its inputs AND is also differentiable (Use Neuron not ~~Perceptron~~)

$$o(\vec{x}) = \sigma(\vec{w}.\vec{x})$$

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

- Backpropagation algorithm learns the weights for a fixed set of units and interconnections
- It employs gradient descent to minimize the error between the network output values and the target values for these outputs
- Let Error function is redefined as

$$E(\vec{w}) = \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd} - o_{kd})^2$$

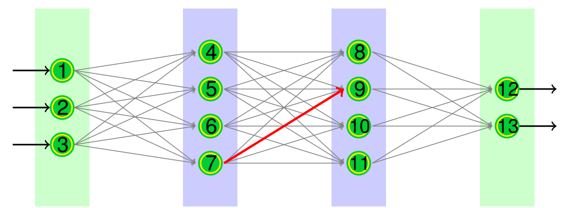
Backpropagation (for 2 layers)

Algorithm 2: Backpropagation(D, η, n_{in}, n_{out}, n_{hidden})

```
Create the feedforward network with n_in, n_out, n_hidden layers
Randomly initialize weights to small values ∈ [−0.05, +0.05]
repeat
    for each <x⃗, t⃗> ∈ D do
        o_u = get output from network ∀ unit u
        δ_k = o_k(1 − o_k)(t_k − o_k) for all output unit k
        δ_h = o_h(1 − o_h) Σ_{k∈outputs}(w_kh δ_k) for all hidden unit h
        w_ji = w_ji + △w_ji      where △w_ji = η δ_j x_ji
until converge;
```

- Recall error function is $E(\vec{w}) = \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd} - o_{kd})^2$
- For a single training example $E_d(\vec{w}) = \frac{1}{2}\sum_{k\in outputs}(t_k - o_k)^2$
- Weight w_{ji} is updated by adding $\triangle\ w_{ji} = -\eta\frac{\partial E_d}{\partial w_{ji}}$

Conventions Over The Network



x_{ji}	ith input to unit j (x_{94} is highlighted)
w_{ji}	weight associated with ith input to unit j
net_j	be $\sum_i w_{ji}x_{ji}$ the weighted sum of input for unit j
o_j	output computed by unit j. Consider it as $\sigma(net_j)$
t_j	target output for unit j
outputs	set of units in final layer ({12, 13} in our case)
Downstream(j)	units whose immediate input is the output of unit j

We are interested in $\frac{\partial E_d}{\partial w_{ji}}$ it is $\frac{\partial E_d}{\partial net_j} \times \frac{\partial net_j}{\partial w_{ji}}$ and therefore, $\frac{\partial E_d}{\partial net_j} \times x_{ji}$

Recap: Training Multilayer Networks with Backpropagation

- Single perceptron can only express linear decision surface
- We need units whose output is a nonlinear function of its inputs AND is also differentiable (using Neuron not ~~Perceptron~~) $o(\vec{x}) = \sigma(\vec{w}.\vec{x})$ where $\sigma(y) = \frac{1}{1+e^{-y}}$

Backpropagation algorithm learns weights

for a fixed set of units and interconnections

- It employs gradient descent to minimize the error between the network output values and the target values for these outputs
- Let Error function is redefined as

$$E(\vec{w}) = \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd} - o_{kd})^2$$

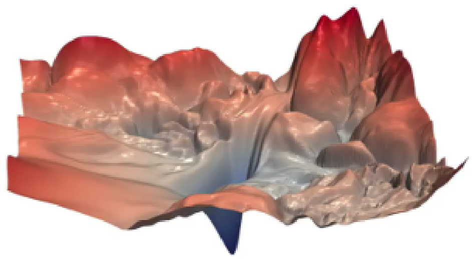
Recap: Backpropagation (for 2 layers)

Algorithm 3: Backpropagation(D, η, n_{in}, n_{out}, n_{hidden})

```
Create the feedforward network with n_in, n_out, n_hidden layers
Randomly initialize weights to small values ∈ [−0.05, +0.05]
repeat
    for each <x⃗, t⃗> ∈ D do
        o_u = get output from network ∀ unit u
        δ_k = o_k(1 − o_k)(t_k − o_k) for all output unit k
        δ_h = o_h(1 − o_h) Σ_{k∈outputs}(w_kh δ_k) for all hidden unit h
        w_ji = w_ji + △w_ji      where △w_ji = η δ_j x_ji
until converge;
```

- Recall error function is $E(\vec{w}) = \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd} - o_{kd})^2$
- For a single training example $E_d(\vec{w}) = \frac{1}{2}\sum_{k\in outputs}(t_k - o_k)^2$
- Weight w_{ji} is updated by adding $\triangle\ w_{ji} = -\eta\frac{\partial E_d}{\partial w_{ji}}$

Loss Landscape [2]



Backpropagation over multilayer networks converge to a local minimum, NOT necessarily to the global minima

[2] Visualizing the Loss Landscape of Neural Nets, Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, Tom Goldstein, 2018

Backpropagation

- **Adding Momentum:** weight update during n^{th} iteration depend partially on the update that occurred during the $(n-1)^{th}$ iteration

$$\triangle w_{ji}(n) = \eta \delta_j x_{ji} + \alpha \triangle w_{ji}(n-1)$$

- **Learning in arbitrary acyclic network:** for feed-forward networks of arbitrary depth, δ_r value for a unit in hidden layer is determined as

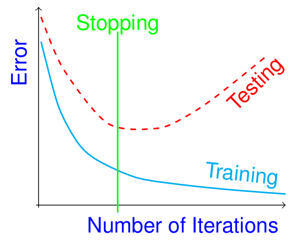
$$\delta_r = o_r(1-o_r) \sum_{s \in Downstream(r)} w_{sr} \times \delta_s$$

Backpropagation

- Result of Backpropagation over multilayer networks is only guaranteed to converge toward some local minimum and not necessarily to the global minimum error
- No methods are known to predict with certainty when local minima will cause difficulties
- Suggested to use momentum, true gradient descent or multiple networks (initialized with different random weights)
- Any boolean function can precisely be represented by some network having only **two** layers of units (Cybenko 1989; Hornik et al. 1989)
- Every bounded continuous function can be approximated with arbitrarily small error (under a finite norm) by a network with two layers of units
- Any function can be approximated to arbitrary accuracy by a network with three layers of units (Cybenko 1988)

Generalization, Overfitting, and Stopping Criterion

Continue training until the error on the training examples falls below some predetermined threshold could be a poor strategy



- Weight decay or use of validation set (k-fold ?) is suggested
- Input or output encoding can be used

Thank You!

Thank you very much for your attention!

Queries ?