Data Mining

Lecture Notes for Chapter 4

Artificial Neural Networks

Introduction to Data Mining , 2nd Edition

by

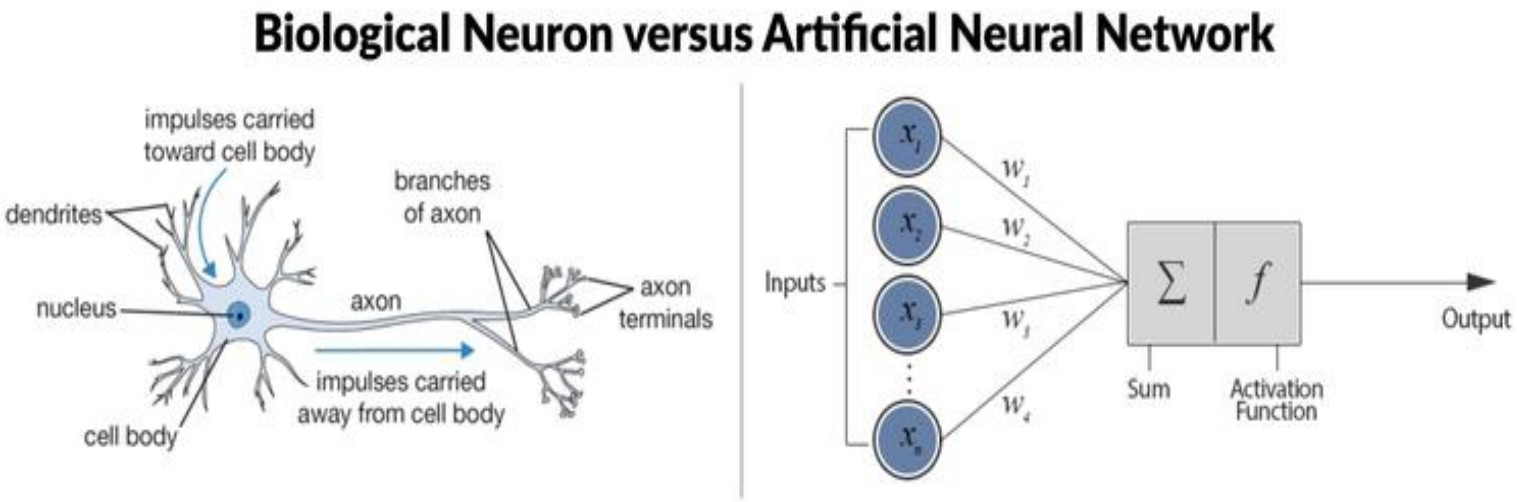
Tan, Steinbach, Karpatne, Kumar

Artificial Neural Networks (ANN)

- Yapay sinir ağları (YSA), biyolojik sinir sistemlerini simüle etme girişimlerinden esinlenmiştir.
- İnsan beyni, esas olarak, **akson** adı verilen lif ipleri aracılığıyla diğer nöronlarla birbirine bağlanan **nöron** adı verilen sinir hücrelerinden oluşur.
  - Aksonlar, nöronlar her uyarıldığında sinir uyarılarını bir nörondan diğerine iletmek için kullanılır.
- Bir nöron, diğer nöronların aksonlarına, nöronun hücre gövdesinden uzantılar olan **dendritler** yoluyla bağlanır.

Artificial Neural Networks (ANN)

- Bir dendrit ile bir akson arasındaki temas noktasına sinaps (**synapse**) denir.
- Nörologlar, insan beyninin, aynı dürtüyle tekrarlanan uyarımla nöronlar arasındaki **sinaptik bağlantının** gücünü değiştirerek öğrendiğini keşfettiler.



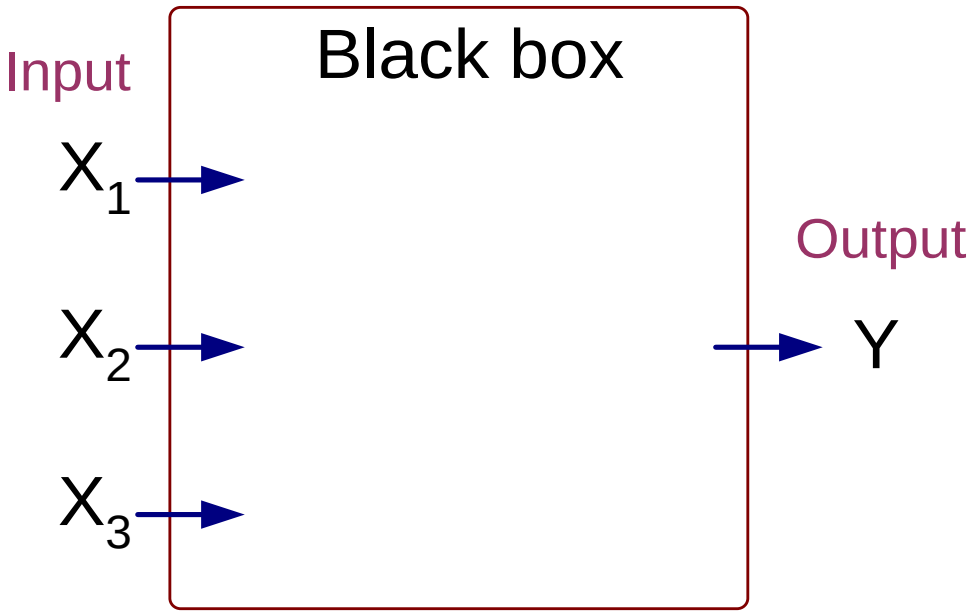
Artificial Neural Networks (ANN)

- İnsan beyni sinir adı verilen yaklaşık 10^{11} hesaplama elemanından oluşur.
- Bu sinirlerin her biri 10^{4} tane sinir eklemine sahiptir.
- Merkezi sinir sistemi reseptörlerden aldığı bilgileri kendi içinde işler ve efektörleri kumanda ederek çeşitli aksiyonlar biçiminde cevaplar üretir.

Sinirler, çözülmüş kimyasal iyonlar içeren akışkanlarla doldurulmuş ve kuşatılmıştır. Bu kimyasal iyonlar sodyum (Na^+), kalsiyum (Ca^{++}), potasyum (K^+) ve klördür (Cl^-). K^+ iyonları sinir hücresinin içinde yoğun olarak bulunurken, Na^+ iyonları ise hücre mebranının dışında yoğunlaşmıştır. Na^+ ve K^+ iyonları sinir darbesi adı verilen aktif sinir cevabının üretilmesinden sorumludur.

Artificial Neural Networks (ANN)

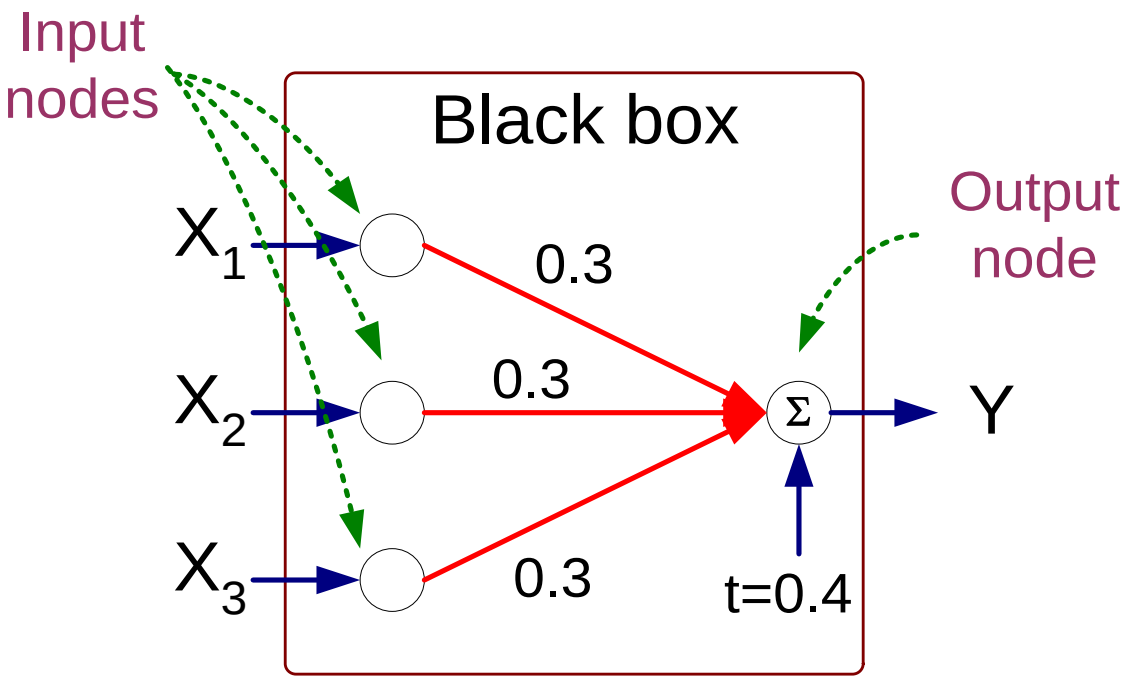
X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



Output Y is 1 if at least two of the three inputs are equal to 1.

Artificial Neural Networks (ANN)

X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



$$Y = sign(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

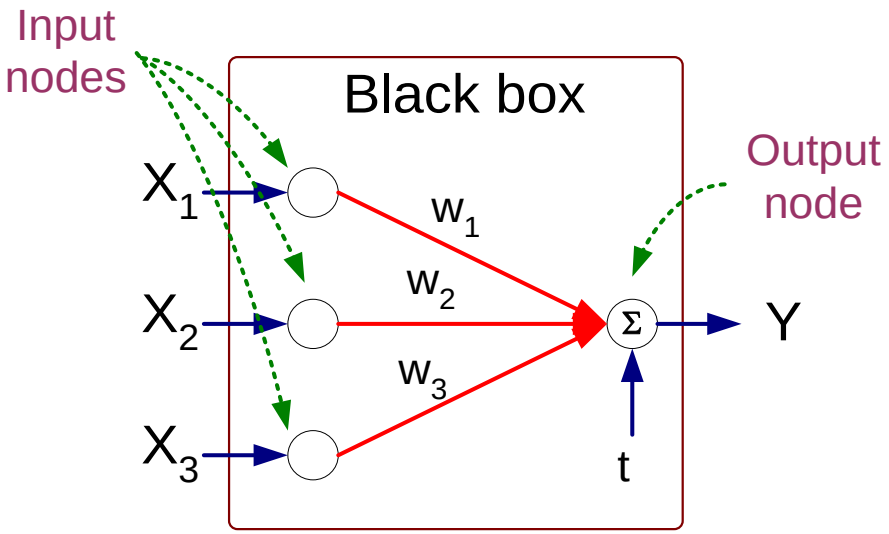
$$\text{where } sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Artificial Neural Networks (ANN)

- Model, birbirine bağlı düğümlerin ve ağırlıklandırılmış bağlantıların bir birleşimidir

- Çıkış düğümü, giriş değerlerinin her birini bağlantılarının ağırlıklarına göre toplar

- Çıkış düğümünü bir *t* eşik değeriyle karşılaştırır

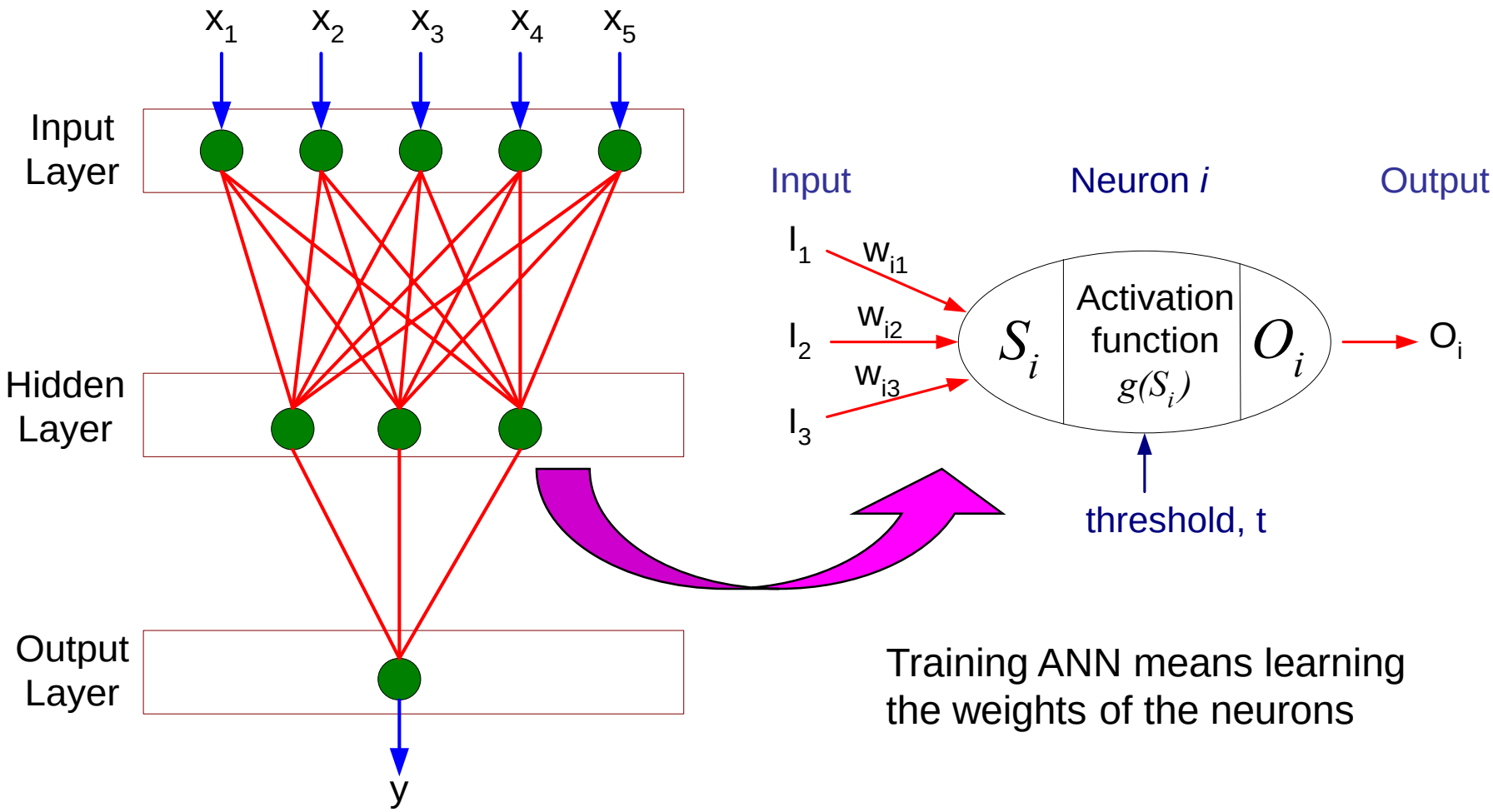


Perceptron Model

$$Y = sign(\sum_{i=1}^{d} w_i X_i - t)$$

$$= sign(\sum_{i=0}^{d} w_i X_i)$$

General Structure of ANN



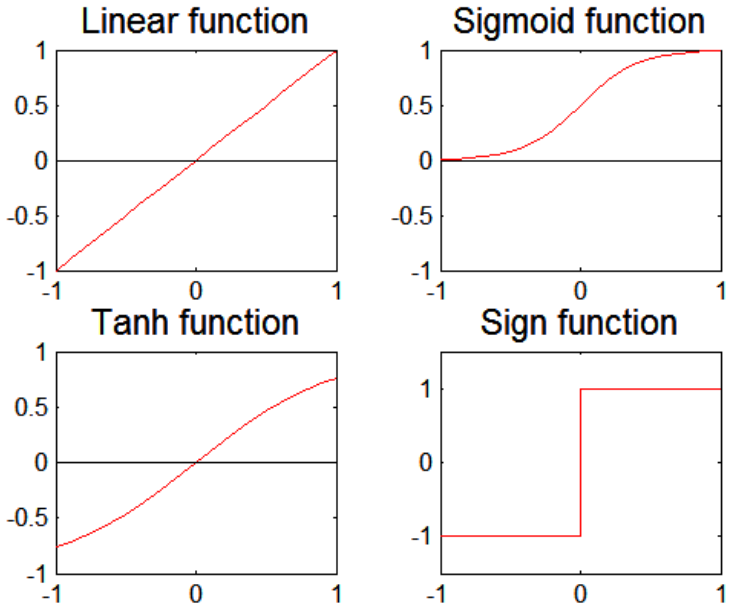
Training ANN means learning the weights of the neurons

Artificial Neural Networks (ANN)

- Various types of neural network topology
  - single-layered network (perceptron) versus multi-layered network
  - Feed-forward versus recurrent network

- Various types of activation functions (f)

$$Y = f(\sum_i w_i X_i)$$



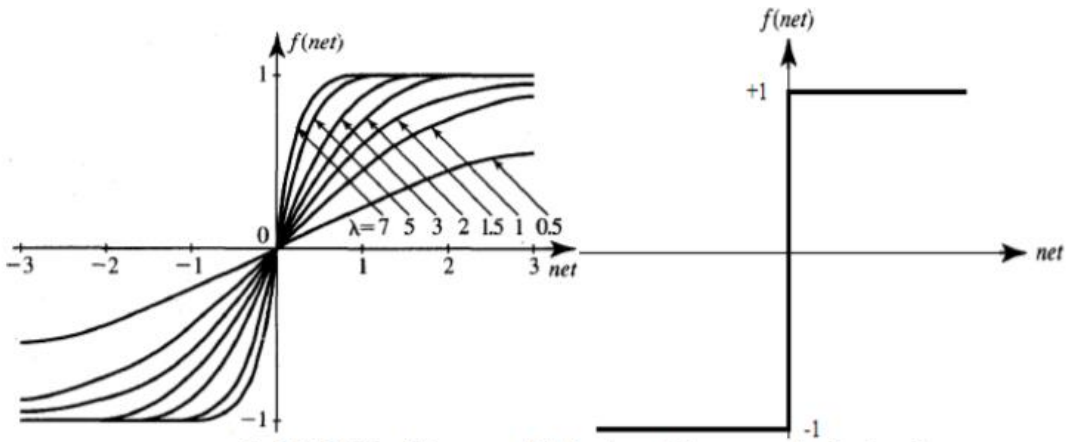
Artificial Neural Networks (ANN)

Sinir, süreçleme düğümü olarak, kendi ağırlıklandırılmış girişlerinin toplanması işlemini veya net'i elde etmek için skaler çarpma işlemini gerçekler. Ardından, kendi aktivasyon fonksiyonu sayesinde $f(\text{net})$ lineer olmayan işlemini yapar. Kullanılan aktivasyon fonksiyonları

$$f(net) \triangleq \frac{2}{1+\exp(-\lambda net)} - 1 \tag{5}$$

$$f(net) \triangleq \operatorname{sgn}(net) = \begin{cases} +1, & net > 0 \\ -1, & net < 0 \end{cases} \tag{6}$$

$\lambda > 0$ katsayısı net = 0 yakınında $f(\text{net})$ sürekli aktivasyon fonksiyonunun derinliğini belirleyen sinir kazancı ile orantılıdır. $\lambda \to \infty$ için sürekli aktivasyonun limiti (6)



Şekil 3. Sürekli ve ayrık bipolar aktivasyon fonksiyonları

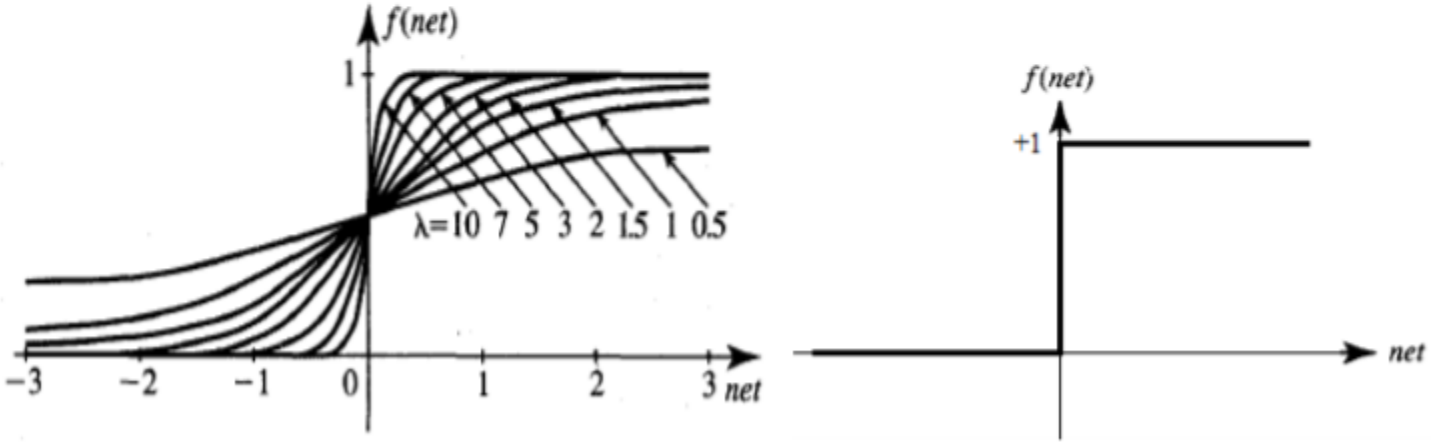
Artificial Neural Networks (ANN)

Aktivasyon fonksiyonları tek kutuplu olabilir, bu durumda sinirlerin çıkışları sadece pozitif değerler alır.

$$f(net) \triangleq \frac{1}{1 + \exp(-\lambda net)} \tag{7}$$

$$f(net) \triangleq \begin{cases} 1, & net > 0 \\ 0, & net < 0 \end{cases} \tag{8}$$

(7) ve (8) bağıntılarına karşı düşen aktivasyon fonksiyonları şekil 4'te gösterilmiştir



Şekil 4. Tek kutuplu sürekli ve ayrık aktivasyon fonksiyonları

Perceptron

- Single layer network
  - Contains only input and output nodes

- Activation function: f = sign(w•x)

$$\hat{y} = sign[w_d x_d + w_{d-1} x_{d-1} + \ldots + w_1 x_1 + w_0 x_0] = sign(\mathbf{w} \cdot \mathbf{x}),$$

- Applying model is straightforward

$$Y = sign(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

$$\text{where } sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

  - $X_1 = 1, X_2 = 0, X_3 = 1 \Rightarrow y = sign(0.2) = 1$

Perceptron Learning Rule

- Initialize the weights (w_0, w_1, …, w_d)
- Repeat
  - For each training example (x_i, y_i)
    - Compute f(w, x_i)
    - Update the weights:

$$w^{(k+1)} = w^{(k)} + \lambda\left[y_i - f(w^{(k)}, x_i)\right]x_i$$

- Until stopping condition is met

Perceptron Learning Rule

Algorithm 5.4 Perceptron learning algorithm.

1: Let $D = \{(\mathbf{x}_i, y_i) \mid i = 1, 2, \ldots, N\}$ be the set of training examples.
2: Initialize the weight vector with random values, $\mathbf{w}^{(0)}$
3: **repeat**
4: **for** each training example $(\mathbf{x}_i, y_i) \in D$ **do**
5: Compute the predicted output $\hat{y}_i^{(k)}$
6: **for** each weight w_j **do**
7: Update the weight, $w_j^{(k+1)} = w_j^{(k)} + \lambda\left(y_i - \hat{y}_i^{(k)}\right)x_{ij}$.
8: **end for**
9: **end for**
10: **until** stopping condition is met

Perceptron Learning Rule

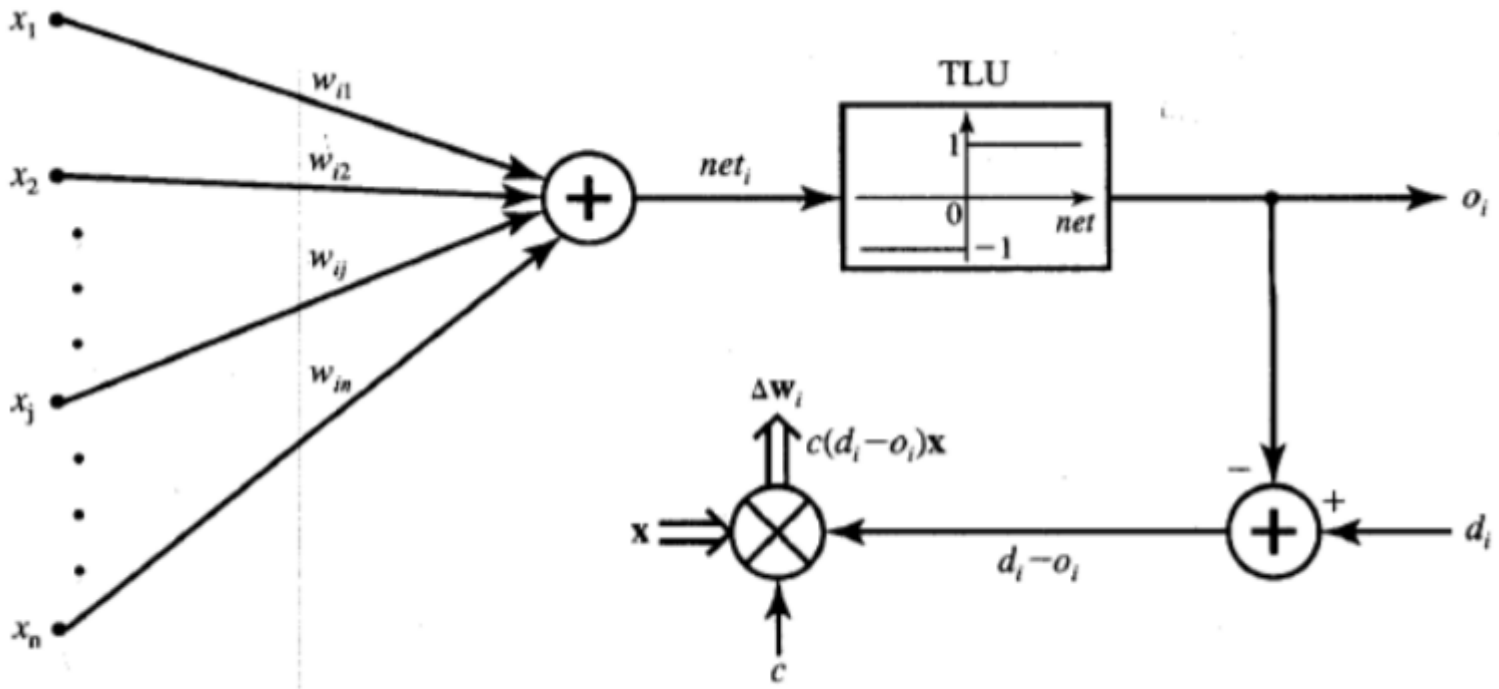
- Weight update formula:

$$w^{(k+1)} = w^{(k)} + \lambda\left[y_i - f(w^{(k)}, x_i)\right]x_i \;;\; \lambda : \text{learning rate}$$

- Intuition:
  - Update weight based on error: $e = \left[y_i - f(w^{(k)}, x_i)\right]$
  - If y=f(x,w), e=0: no update needed
  - If y>f(x,w), e=2: weight must be increased so that f(x,w) will increase
  - If y<f(x,w), e=-2: weight must be decreased so that f(x,w) will decrease

Perceptron Learning Rule

Threshold Logic Unit (TLU)



Şekil 7. Perception öğrenme kuralı

Example of Perceptron Learning

$$w^{(k+1)} = w^{(k)} + \lambda\left[y_i - f(w^{(k)}, x_i)\right]x_i$$

$$Y = sign(\sum_{i=0}^{d} w_i X_i)$$

$$\lambda = 0.1$$

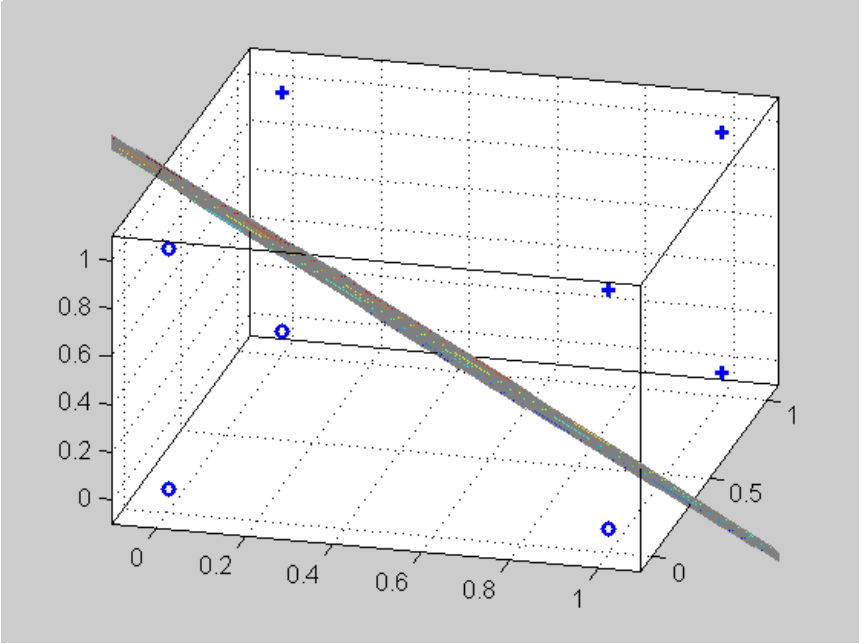
X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1

	w_0	w_1	w_2	w_3
0	0	0	0	0
1	-0.2	-0.2	0	0
2	0	0	0	0.2
3	0	0	0	0.2
4	0	0	0	0.2
5	-0.2	0	0	0
6	-0.2	0	0	0
7	0	0	0.2	0.2
8	-0.2	0	0.2	0.2

Epoch	w_0	w_1	w_2	w_3
0	0	0	0	0
1	-0.2	0	0.2	0.2
2	-0.2	0	0.4	0.2
3	-0.4	0	0.4	0.2
4	-0.4	0.2	0.4	0.4
5	-0.6	0.2	0.4	0.2
6	-0.6	0.4	0.4	0.2

Perceptron Learning Rule

- f(w,x), giriş değişkenlerinin doğrusal bir kombinasyonu olduğundan, karar sınırı doğrusaldır

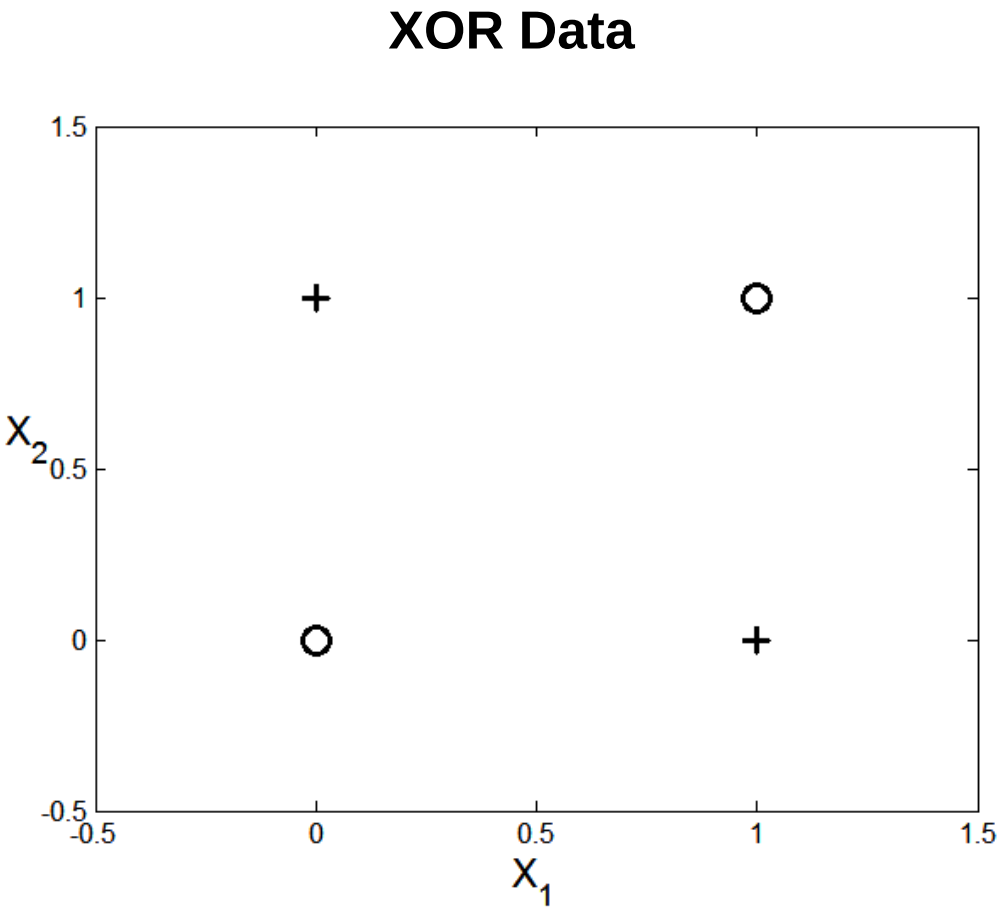


- Doğrusal olmayan ayrılabilir problemler (nonlinearly separable problems) için, **perceptron** öğrenme algoritması başarısız olacaktır çünkü hiçbir doğrusal hiper düzlem verileri mükemmel şekilde ayıramaz.

Nonlinearly Separable Data

$$y = x_1 \oplus x_2$$

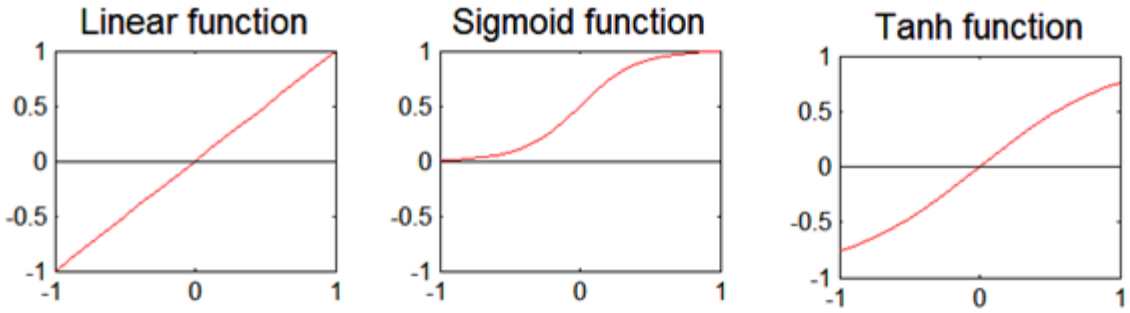
x_1	x_2	y
0	0	-1
1	0	1
0	1	1
1	1	-1



Multilayer Neural Network

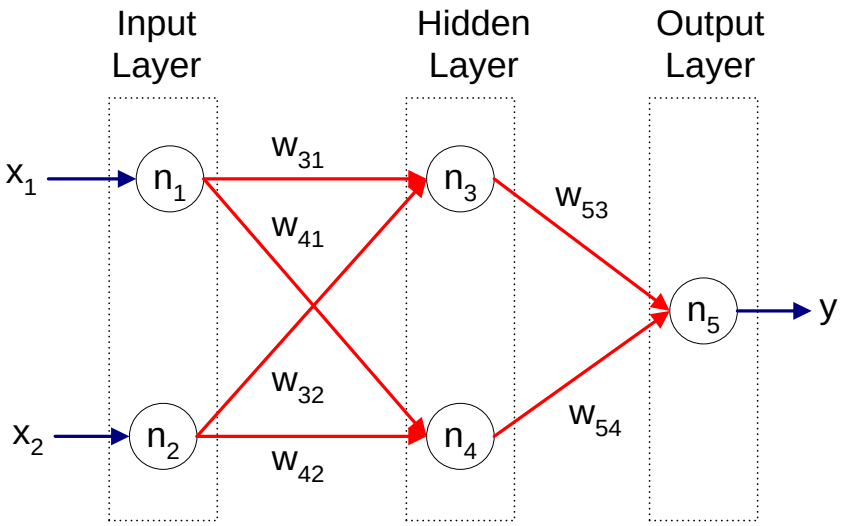
- Hidden layers
  - intermediary layers between input & output layers

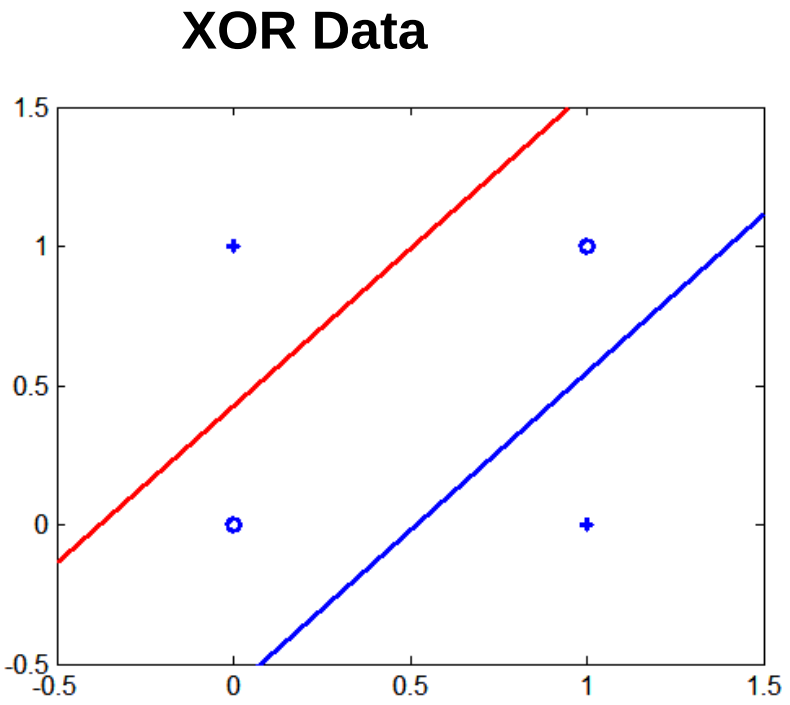
- More general activation functions (sigmoid, linear, etc)



Multi-layer Neural Network

- Çok katmanlı sinir ağı (*Multi-layer neural network*), doğrusal olmayan karar yüzeylerini içeren her türlü sınıflandırma görevini çözebilir





Learning Multi-layer Neural Network

- Gizli düğümler dahil her düğüme perceptron öğrenme kuralı uygulayabilir miyiz?
  - Perceptron öğrenme kuralı, hata terimini e = y-f(w,x) hesaplar ve ağırlıkları buna göre günceller
    - Problem: Gizli düğümler için y'nin gerçek değeri nasıl belirlenir?
  - Çıkış düğümlerindeki hataya göre gizli düğümlerde yaklaşık hata
    - Problem:
      - Gizli düğümlerdeki ayarlamanın genel hatayı nasıl etkilediği net değil
      - Optimum çözüme yakınsama garantisi yok

Learning the ANN Model

- YSA öğrenme algoritmasının amacı, toplam karesel hataların toplamını en aza indiren bir dizi ağırlık **w** belirlemektir.

Objective function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2.$$

- Hataların karesinin toplamının **w** 'ye bağlı olduğuna dikkat edin, çünkü tahmin edilen $\hat{y}$ sınıfı, gizli katman ve çıkış düğümlerine atanan ağırlıkların bir fonksiyonudur.

Learning the ANN Model

- Şekil 5.20, iki parametresi olan w_1 ve w_2'nin bir fonksiyonu olarak hata yüzeyinin bir örneğini göstermektedir.

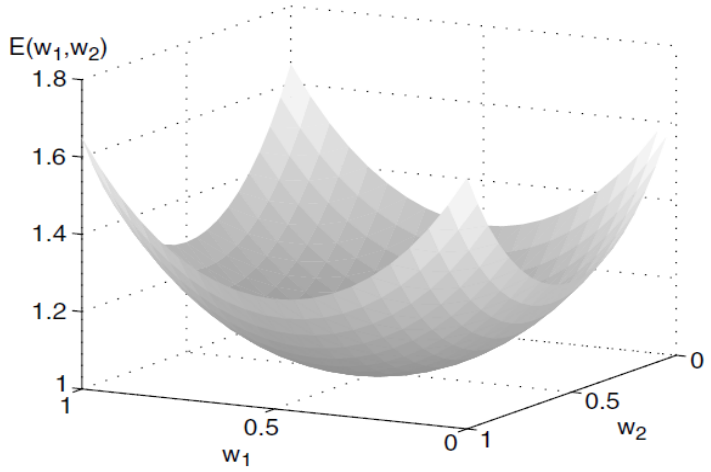


Figure 5.20. Error surface $E(w_1, w_2)$ for a two-parameter model.

- Bu tür bir hata yüzeyi tipik olarak $\hat{y}_i$, **w** parametresinin doğrusal (linear) bir fonksiyonu olduğunda karşılaşılır.

Learning the ANN Model

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2.$$

- $\hat{y}$= **w** · **x** ifadesini önceki denklemde yerine koyarsak, hata fonksiyonu parametrelerinde ikinci dereceden olur
  - ve global bir minimum çözüm kolayca bulunabilir.
- Çoğu durumda, bir YSA çıktısı, aktivasyon fonksiyonlarının seçimi nedeniyle (örneğin, **sigmoid** veya **tanh** fonksiyonu) parametrelerinin doğrusal olmayan (**nonlinear**) bir fonksiyonudur.
- Sonuç olarak, **w** için global olarak optimal olması garanti edilen bir çözüm bulmak artık kolay değil.

Gradient Descent for MultiLayer NN

- Optimizasyon problemini verimli bir şekilde çözmek için gradyan iniş (**gradient descent**) yöntemine dayananlar gibi açgözlü (Greedy) algoritmalar geliştirilmiştir.
- Gradyan iniş yöntemi tarafından kullanılan **ağırlık güncelleme formülü** şu şekilde yazılabilir:

$$w_j \longleftarrow w_j - \lambda \frac{\partial E(\mathbf{w})}{\partial w_j},$$

λ öğrenme oranıdır (*learning rate*).

- İkinci terim, ağırlığın genel hata terimini azaltan bir yönde artırılması gerektiğini belirtir.

Gradient Descent for MultiLayer NN

- Bununla birlikte, hata fonksiyonu doğrusal olmadığından, **gradyan iniş yönteminin** lokal minimumda yakalanması olasıdır.

- Gradyan iniş yöntemi, bir sinir ağının çıkış ve gizli düğümlerinin ağırlıklarını hesaplamak için kullanılabilir.

Gradient Descent for MultiLayer NN

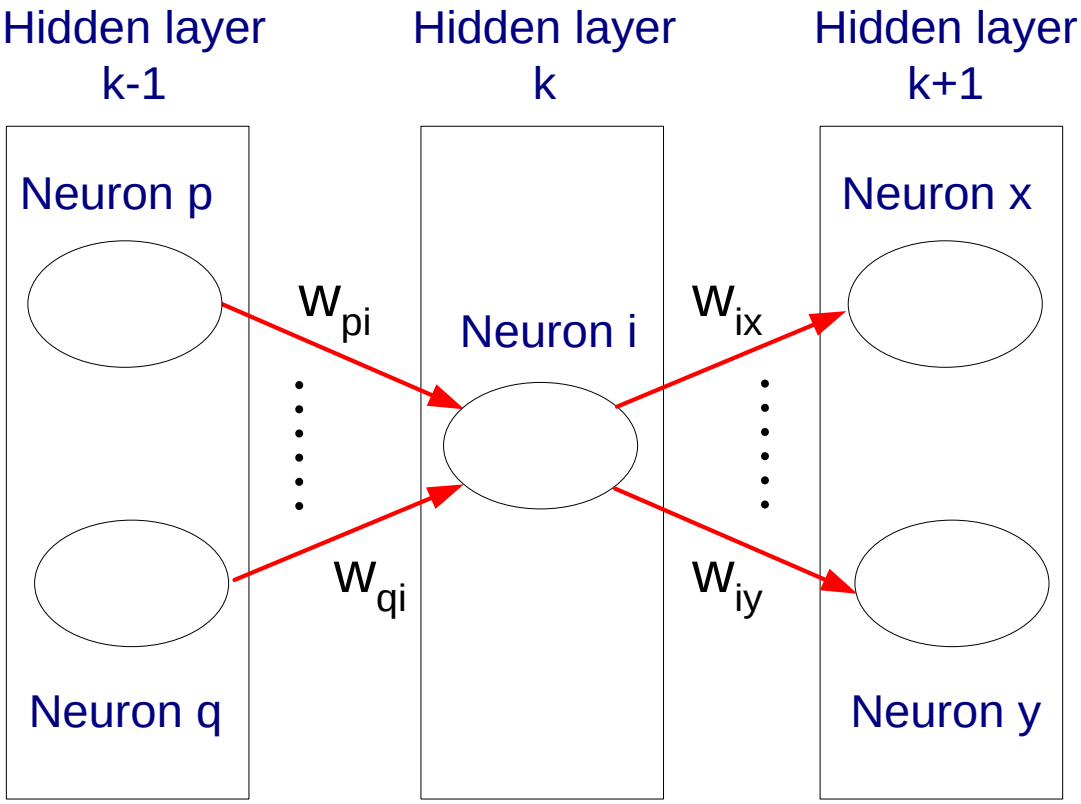
- Gizli düğümler (*hidden nodes*) için hesaplama kolay değildir çünkü çıkış değerlerinin ne olması gerektiğini bilmeden hata terimleri $\partial E/\partial w_j$ 'yi değerlendirmek zordur.
- **backpropagation** olarak bilinen bir teknik, bu sorunu çözmek için geliştirilmiştir.

- Algoritmanın her iterasyonunda iki aşama vardır: ileri faz (*forward phase*) ve geri faz (*backward phase*).

Gradient Descent for MultiLayer NN

- İleri faz (*forward phase*) sırasında, önceki iterasyonda elde edilen ağırlıklar, ağdaki her bir nöronun çıkış değerini hesaplamak için kullanılır.
  - Hesaplama ileri yönde ilerler; yani, *k* seviyesindeki nöronların çıktıları, *k* + 1 seviyesindeki çıktıların hesaplanmasından önce hesaplanır.
- Geriye doğru (backward phase) aşamada, ağırlık güncelleme formülü **ters yönde** uygulanır.
  - Diğer bir deyişle, *k* + 1 seviyesindeki ağırlıklar, *k* seviyesindeki ağırlıklar güncellenmeden önce güncellenir.
  - Bu geri yayılma (back-propagation) yaklaşımı, *k* + 1 katmanındaki nöronlara ait hataları *k* katmanındaki nöronların hatalarını tahmin etmek için kullanmamızı sağlar.

Gradient Descent for MultiLayer NN



- **Activation function f must be differentiable**

Design Issues in ANN

- Giriş katmanındaki düğüm sayısı(*Number of nodes in input layer*)
  - Her bir binary/sürekli öznitelik için bir giriş düğümü (*One input node per binary/continuous attribute)*
  - k değerli her kategorik düğüm öznitelik için k veya $\log_2$ k adet düğüm (*k or $\log_2$ k nodes for each categorical attribute with k values*)
- Çıkış katmanındaki nöron sayısı(*Number of nodes in output layer*)
  - İkili sınıflandırma problemi için 1 tane çıkış (*One output for binary class problem)*
  - K-sınıflı problem için k veya $\log_2$ k adet (*k or $\log_2$ k nodes for k-class problem*)
- Gizli katmandaki nöron sayısı (*Number of nodes in hidden layer*)
- Başlangıç ağırlıkları ve bias değerleri (Initial weights and biases)

Characteristics of ANN

- Çok katmanlı YSA (Multilayer ANN) evrensel tahmin edicilerdir (**universal approximators**) ancak ağ çok büyükse overfitting'den zarar görebilir
- Gradyan inişi **lokal minimuma** yakınsayabilir
  - Yerel minimumdan kaçmanın bir yolu, ağırlık güncelleme formülüne bir momentum terimi eklemektir.
- Model oluşturmak çok **zaman alıcı** olabilir, ancak test etme çok hızlı olabilir
- Ağırlıklar otomatik olarak öğrenildiği için **gereksiz niteliklerin üstesinden gelebilir**
- Eğitim verilerinde gürültüye duyarlı
- Eksik niteliklerin üstesinden gelmek zor

Recent Noteworthy Developments in ANN

- Use in deep learning and unsupervised feature learning
  - Seek to automatically learn a good representation of the input from unlabeled data
- Google Brain project
  - Learned the concept of a ‘cat’ by looking at unlabeled pictures from YouTube
  - One billion connection network