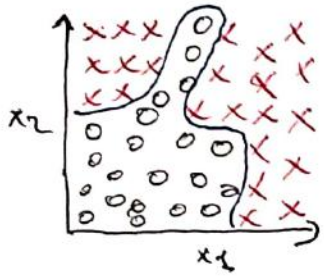
Neural Networks

Non-linear Hipotez



$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \dots)$

Bu şekilde iki özellikli bir sınıflandırma problemini lojistik regresyon ile çözebiliriz.

x_1 = size
x_2 = bedrooms
x_3 = floors
x_4 = age
...
x_{100}

Ancak böyle 100 özelliğe sahip bir problemi modele uydurmak için polinomial yapmalıyız ve bu durum özellik sayımızı 5000'e çıkarır. Bu durumda overfitting'e, hesaplama hantallığına yol açabilir. Yaklaşık olarak özellik sayınızın karesi kadar özellik sayısı elde edersiniz. $O(n^2) \cong \frac{n^2}{2}$

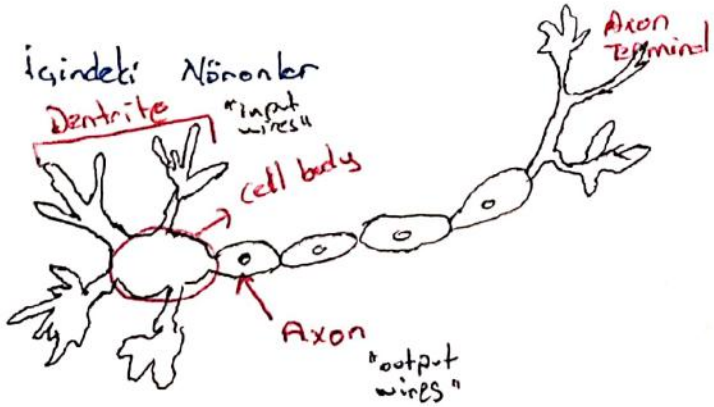
Kuadratik veya kübik özellikler eklemeyle birlikte basit lojistik regresyon, n çok büyük olduğunda doğrusal olmayan hipotezleri öğrenmede iyi bir yol değildir. Sinir ağları işte böyle çoklu karmaşık özelliklerde oldukça işe yarayan bir yöntemdir.

Neurons and Brains

Sinir ağları, beyni taklit etme amacıyla geliştirilen eski bir algoritmadır. 80-90'lı yıllarda kullanılmış ve 90'ların sonlarında bırakılmış. Ancak günümüzde bilgisayarların artan donanım gücü ve veri patlaması nedeniyle popüler olmuştur.

Beynin farklı bir bölgesine farklı bir görevi öğretebiliyoruz. Bu inanılmaz bir şey. Eğer beynin nasıl öğrendiğinin algoritmasını çıkarabilirsek yapay zekaya onlarıyla amacına kavuşacaktır.

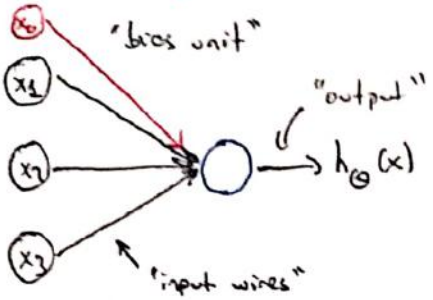
Beyin İçindeki Nöronlar



Beyin hücreleri dentrite denen bağlantı ağlarından sinyal alır. Cell body noktasında işler ve axon aracılığıyla yeni sinyali diğer nöronlara iletir. Nöronlar elektrik yolluyor gibi yollamayarak birbirlerine veri aktarır.

Bu durum 0-1 anlamına gelir. Bizde bir nöron bir lojistik birim olarak modelleyeceğiz.

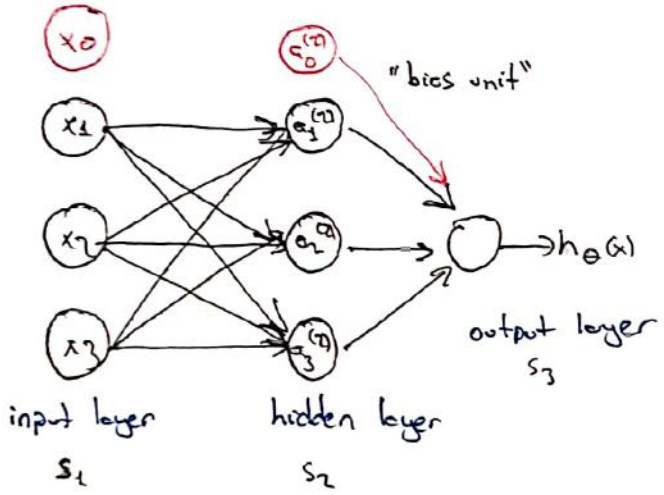
Nöron Model: Lojistik Birim



Daha önce dediğim gibi nöron modellerini lojistik regresyon olarak modelleyeceğiz.
Θ yüzden $h_\Theta(x) = \frac{1}{1+e^{-\Theta x}}$ 'dur.

x_0 olarak çizdiğimiz düğüm bias unit veya bias nöron olarak adlandırılır. Ancak x_0 ile eşit olduğundan çizebiliriz çizmeyebiliriz.

Sinir ağları terminolojisinde Θ parametrelerine, ağırlık (weights) olarak adlandırılır. Aktivasyon fonksiyonu (hipotez) olarak sigmoid kullanacağız.



x_0 ve $a_0^{(2)}$ 1 olduğu için onları yazmayabilirsiniz. İlk katman input layer'dır. Son katman output layer'dır. Ara katmanlara hidden layer denir.

$a_i^{(j)}$: j katmanı içinde i birim aktivasyonu

$\Theta^{(j)}$: j katmanının Θ matrisini bildirir.

$\Theta^{(j)}$ matrisi : $s_{j+1} \times (s_j + 1)$ şeklinde belirlenir. Örneğin yukarıdaki örnekte;
$\Theta^{(1)} = 3\times4$ ve $\Theta^{(2)} = 1\times4$ matrise sahiptir.

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3) = g(z_1^{(2)})$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3) = g(z_2^{(2)})$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3) = g(z_3^{(2)})$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

Vectorization

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$z^{(2)} = \Theta^{(1)}x$ veya $\Theta^{(1)}a^{(1)}$

$a^{(2)} = g(z^{(2)})$

Add $a_0^{(2)} = 1$

$z^{(3)} = \Theta^{(2)}a^{(2)}$

$h_\Theta(x) = a^3 = g(z^{(3)})$

General formul

$z^{(j)} = \Theta^{(j-1)}a^{(j-1)}$

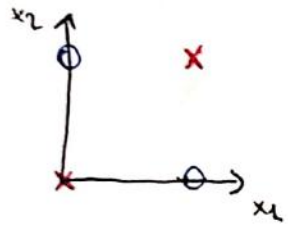
$a^{(j)} = g(z^{(j)})$

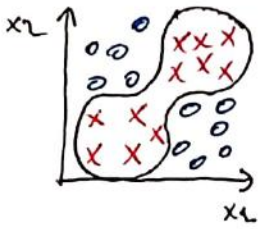
Son katman hesaplanırken $j-1$'e a_0^j bileşenini eklemeliyiz.

Dikkat ettiyseniz her katmanın çıktı değerleri ile lojistik regresyon hesaplıyoruz. Bu yaptığımız aslında lojistik bir model oluşturmak ancak hem özellikleri bir defa hariçinde kullanmıyoruz. Nöronlar her yeni özellik çıktısını aldığı için daha karmaşık hesaplamaları bu sinir ağı sayesinde yapabiliyorlar.

Non-linear clasification example: XOR / XNOR

x_1 ve x_2, binary (0 ve 1)

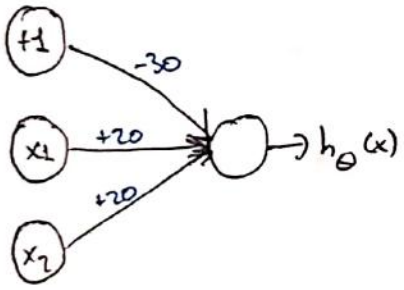




Simple example: AND

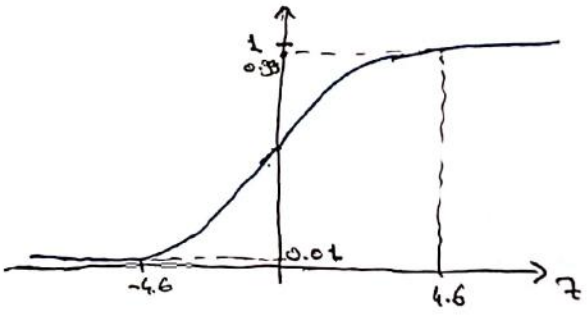
$x_1, x_2 \in \{0,1\}$

$y = x_1$ AND x_2



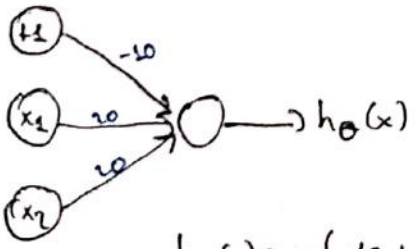
$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$

$\Theta^{(1)}_{10}$ $\Theta^{(1)}_{11}$ $\Theta^{(1)}_{12}$



x_1	x_2	$h_\Theta(x)$
0	0	$g(-30) \cong 0$
0	1	$g(-10) \cong 0$
1	0	$g(-10) \cong 0$
1	1	$g(10) \cong 1$

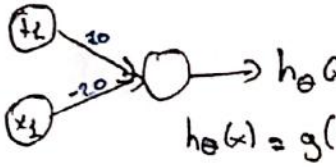
Simple example: OR



$h_\Theta(x) = g(-10 + 20x_1 + 20x_2)$

x_1	x_2	$h_\Theta(x)$
0	0	$g(-10) \cong 0$
0	1	$g(10) \cong 1$
1	0	$g(10) \cong 1$
1	1	$g(30) \cong 1$

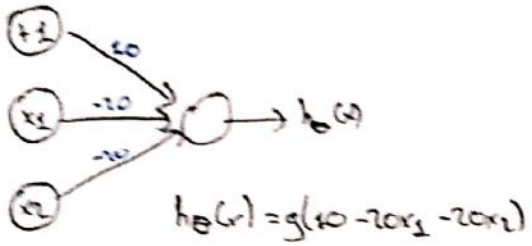
Simple example: NOT



$h_\Theta(x) = g(10 - 20x_1)$

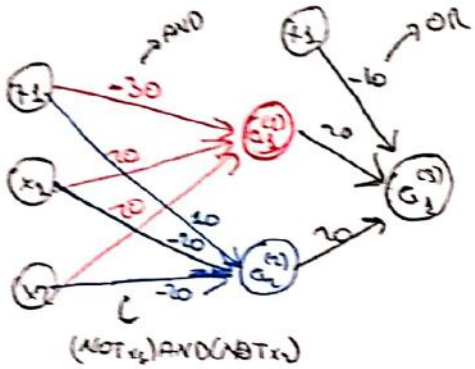
x_1	$h_\Theta(x)$
0	$g(10) \cong 1$
1	$g(-10) \cong 0$

Simple example : (Not x_1) AND (Not x_2)



x_1	x_2	$h_\Theta(x)$
0	0	$g(10) \cong 1$
0	1	$g(-10) \cong 0$
1	0	$g(-10) \cong 0$
1	1	$g(-30) \cong 0$

Main example : XNOR

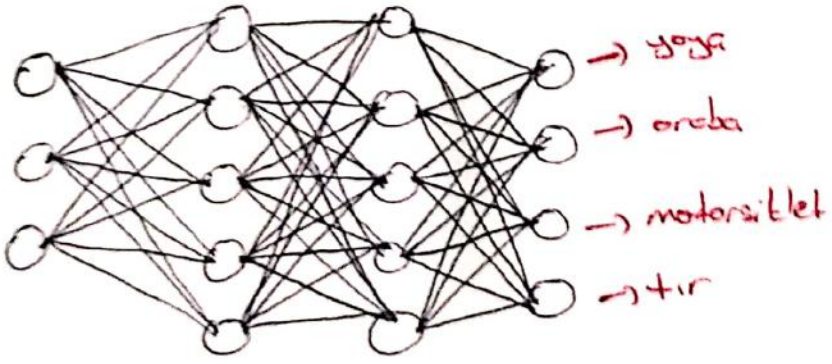


x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x) = a_1^{(3)}$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Yukarıda XNOR'u hesaplayan sinir ağ mimarisinin gizli katmanları gösterdiğimiz gibi içerde karmaşık sınıflandırmanın ufak ayrıntı özelliklerini işleyerek XNOR hesaplayan bir sinir ağ yaratmıştır.

Multiclass Classification

Çoklu sınıflandırma problemlerinde sınıf adedi kadar nöronla sinir ağımızı tanımlarız. Bunun nedeni her bir nöron bir sınıfa verilir ve o nöron o sınıfın 0'mı 1'mi olduğunu döndürür. Sınıflandırma yaptığımız için 1 tanesi 1 dönecek bu da bize problemi sınıflandırmış olacak. $h_\Theta(x)$ sonucumuz (class adedi) x 1 vektör olarak geri dönecektir.



$$y^{(i)} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

Yukarıda bir fotoğrafın yaya mı, araba mı, motor mu, tır mı olduğunu sınıflandıran bir örnek bulun matematiksel

$$h_\Theta(x) = y^{(i)}$$

Exercise Code

```
data = load('ex2data1.txt');
X = data(:, [1,2]);  y=data(:,3);
```

Part 1: Plotting

% Pozitif ve negatif verilerin tabloda x_1 ve x_2 özelliklerine göre grafiği

```
plotData(X, y);
hold on;
xlabel('Exam 1 Score')
ylabel('Exam 2 Score')
legends('Admitted (Geçti)', 'Not admited');
```

Part 2: Compute Cost and Gradient

% logistic regresyonun cost ve gradient'i hesaplanacaktır.

```
[m,n] = size(X);
X = [ones(m,1), X];
initial_theta = zeros(n+1, 1);

[cost, grad] = CostFunction(initial_theta, X,y);
fprintf('Cost at initial theta(zeros): %f \n', cost);
fprintf('Expected cost (approx): 0.693 \n');
fprintf('Gradient at initial theta : \n %f \n', grad);
fprintf('Expected grad : \n -0.1000 \n -12.0092 \n -11.26);

test_theta = [-24; 0.2 ; 0.2];
[cost,grad] = costFunction(test_theta, X,y);
fprintf('Cost at test theta: %f \n', cost);
fprintf('Expected cost (approx): 0.218 \n');
fprintf('Gradient at test theta : \n %f \n', grad);
fprintf('Expected grad: \n 0.043 \n 2.566 \n 2.647 \n);
```

```
function plotData(X,y)
figure; hold on;
pos = find(y==1);
neg = find(y==0);

plot(X(pos,1), X(pos,2), 'k+', 'LineWidth', 2, 'MarkerSize', 7);
plot(X(neg,1), X(neg,2), 'ko', 'MarkerFaceColor', 'y');
hold off;
```

Renk ve marker tipi belirler — kalınlığını tanımlar — Marker'ın boyutu — Marker'ın içini boyar

```
function g = sigmoid(z)
g = zeros(size(z));
g = 1./(1+ exp(-z));
end
```

```
function [J, grad] = costFunction(theta, X, y)
m = length(y);
J = 0;
grad = zeros(size(theta));

h_theta = sigmoid(X * theta);
J = (1/m)* (-y' * log(h_theta) - (1-y)' * log(1-h_theta));

delta = h_theta - y;
grad(1) = (1/m)* (X(:,1)' * (delta));
grad(2) = (1/m)* (X(:,2)' * (delta));
grad(3) = (1/m) * (X(:,3)' * (delta));
```

Part 3: Optimizing using fminunc

```
% Degrade initiali loop seklinde kullanarak
degil fminunc algoritmasinda costFunction
fonksiyonumuzu kullanarak optimum theta
degerlerimizi bulacagiz.

options = optimset('GradObj', 'on', 'MaxIter', 400);
[theta, cost] = ...
    fminunc(@(t)(costFunction(t, X, y)), initial_theta, options);

fprintf('Cost at theta found by fminunc: %f\n', cost);
fprintf('Expected cost (approx): 0.203\n');
fprintf('theta: \n %f \n', theta);
fprintf('Expected theta:\n -25.161\n 0.206\n 0.201\n');

plotDecisionBoundry(theta, X, y);
hold on;
xlabel('Exam 1 score');
ylabel('Exam 2 score');
legend('Admitted', 'Not admitted');
hold off;
```

Part 4: Predict and Accuracies (Doğruluk oranı)

```
% 1. sinavi 47, 2. sinavi 85 olan bir ögrenci
gecme tahmini yapilmali ve training setin basarim
oranini bulunuz

prob = sigmoid([1 45 85] * theta);
fprintf('For a student with scores 45 and 85, we
        predict an admission probability of %f\n', prob);
fprintf('Expected value: 0.775 +/- 0.02');

% Compute accuracy on our training set
p = predict(theta, X);
fprintf('Train accuracy: %f\n' mean(double(y==p))*100);
fprintf('Expected accuracy (approx): 89.0\n');
```

```
function plotDecisionBoundry(theta, X, y)
plotData(X(:, 2:3), y);
hold on;

if size(X,2) <=3
 % Decision boundry çizebilmek için
 iki uç noktaya ihtiyacimiz var.
plot_x = [min(X(:,2)) -2 , max(X(:,2))+2];

 % Seçtiğimiz iki x1 noktasinin
 karşiliği olan iki x2 noktasi seçeceğiz.
 %  x2 = (θ0 + θ1x1) * (-1/θ2)
plot_y = (-1./theta(3)).*(theta(2).*plot_x + theta(1));

 % Noktaları verilerin üzerine çizdiricez.
 plot(plot_x, plot_y);
 axis([30, 100, 30, 100]);

else
   u = linspace(-1, 1.5, 50);
   v = linspace(-1, 1.5, 50);
   z = zeros(length(u), length(v));
   % z = theta * x
   for i = 1:length(u)
       for j = 1:length(v)
           z(i,j) = mapFeature(u(i), v(j))*theta;
       end
   end
   z = z';
   contour(u, v, z, [0,0], 'LineWidth', 2);
end
```

```
function p = predict(theta, X);
m = size(X, 1);
p = zeros(m, 1);
p = double(sigmoid(X*theta) >=0.5);
```

Exercise Code

```
data = load('ex2data2.txt');
X = data(:, [1,2]); y = data(:,3);

plotData(X, y);
hold on;
xlabel('Microphic test 1');
ylabel('Microphic tes 2');
legend =('y=1', 'y=0');
hold off;
```

Part1 : Regularized Logistic Regression
% Elimizdeki veri setimize doğrusal bir model uygulamayacağımızdan verilere polinom özellikler eklemeliyiz ve bunun içinde mapFeature fonksiyonumuz var ^6 'ya kadar polinom oluşturur.
X verimiz 118 x28 olur.

```
X = mapFeature(X(:,1), X(:,2));
initial_theta = zeros(size(X,2), 1);

lambda = 1
[cost, grad] = costFunctionReg(initial_theta, X, y, lambda);

fprintf('Cost at initial theta (zeros): %f\n', cost);
fprintf('Expected cost (approx): 0.693\n');
fprintf('Gradient at initial theta(zeros)-first five value\n');
fprintf('%f\n', grad(1:5));
fprintf('Expected grad:\n 0.0085\n 0.0188\n 0.0001\n 0.0503\n 0.0115');

test_theta = ones(size(X,2), 1);
[cost, grad] = costFunctionReg(test_theta, X, y, 10);

fprintf('\nCost at test theta: %f\n', cost);
fprintf('Expected cost (approx): 3.16\n');
fprintf('Gradient at test theta - first five value: %f\n', grad(1:5));
fprintf('Expected grad:\n 0.3460\n 0.1614\n 0.1948\n 0.2269\n 0.0922');
```

```
function out = mapFeature(X1, X2)
degree = 6;
out = ones(size(X1(:,1)));
for i = 1:degree
    for j = 0:i
        out(:, end+1) = (X1.^(i-j)).*(X2.^j);
    end
end
```

```
function [J, grad] = costFunctionReg(theta, X, y, lambda)
m = length(y);
J = 0;
grad = zeros(size(theta));

h_theta = sigmoid(X*theta);
J = (1/m)*(-y'*log(h_theta) - (1-y)'*...
    log(1-h_theta)) + lambda/(2*m)*(theta(2:size(theta,1),:)'*theta(2:size(theta,1),:));

delta = h_theta - y;
grad(1) = (1/m)*(X(:,1)'*(delta));
for i=2:size(theta,1)
    grad(i) = (1/m)*(X(:,i)'*(delta)) + ...
        (lambda/m)*theta(i);
end;
```

Part 2: Regularisation and Accuracies

```
initial_theta = zeros(size(X,2),1);
lambda = 1;

options = optimset('GradObj','on','MaxIter',400);

[theta, J, exit_flag] = fminunc(@(t)(costFunctionReg(t, X, y, lambda)), initial_theta, options);

% Plot Boundry
plotDesicionBoundry(theta, X, y);
hold on;
title(sprintf('lambda = %g', lambda));
xlabel('Microphic Test 1');
ylabel('Microphic Test 2');
legend('y=1', 'y=0', 'Decision Boundry');
hold off;

% Compute accuracy on our training set
p = predict(theta, X);
fprintf('Train Accuracy: %f\n', mean(double(p == y)) * 100);
```