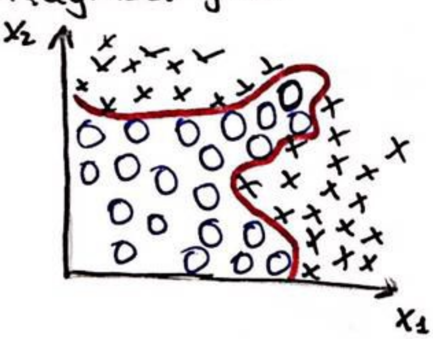
-W4- NN Representations -

- Yeni learning algorithm modelimiz "Neural Networks" State of the art technique for many different machine learning problems.
- Why do we need another learning algorithm models than linear and logistic regression? Why do we need Neural Networks?
    - Basitçe cevap şu: n large iken non-linear classifiers build etmek çok zor çünkü feature space will be too large bu da overfitting'e yol açabilir ve computationally expensive.

Ex: Supervised classification problem için bir training set olsun, aşağıdaki gibi:



- One thing we could do is, apply logistic regression with a lot of non-linear features like:
- $$\boxed{g(\theta_0+\theta_1 x_1+\theta_2 x_2+\theta_3 x_1 x_2+\theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \cdots)}$$
- Yeterince polynomial kullanırsak belki kırmızı çizgi gibi bir decision boundary elde edebiliriz.

- Logistic Regression yukarıdaki metodla çalışır ancak bunun için feature sayısının (x_1, x_2) az olması lazım. 2 feature için çalışır.
- Ancak bir çok machine learning problemi için 2'den çok daha fazla feature'ımız vardır.
    - Mesela housing için evin satılma olasılığını bulmaya çalıştığımız bir classification problem olsun.
    - x_1 = size, x_2 = # bedrooms, x_3 = # floors, --- x_{100} = ... gibi 100 tane feature olsun ($n=100$)
    - $n=100$ için yukarıdaki gibi bir dağılımda kullanacağımız bir hipotez için tüm quadratic functions'ı dahil etsek:

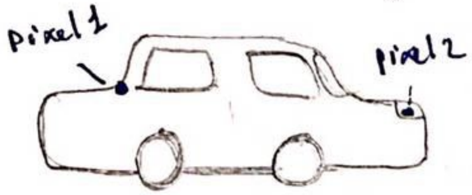
$$\boxed{\begin{matrix} x_1^2, & x_1 \cdot x_2, & x_1 \cdot x_3, & x_1 \cdot x_4, & \cdots & x_1 x_{100} \\ x_2^2, & x_2 x_3, & x_2 \cdot x_4, & \cdots & & x_2 \cdot x_{100} \\ & & \vdots & & & \\ x_{100}^2 & & & & & \end{matrix}} \approx \boxed{\frac{n^2}{2} \approx 5000}$$

features!

- Daha küçük bir feature space için ise yukarıdaki gibi bir data set ayrılamaz.
- Cubic termleri de alsak 170 000 term!

- W4 - NN Representations -

- For many machine learning problems n will be large! Let's consider the problem of computer vision.
- Suppose we wanna use machine learning to train a classifier to examine an image and tell us whether or not the image is car.



- Computer'in gördüğü pixel intensity values'dan oluşan bir matrix'ten fazlası değil. Her pixel'in brightness'ı yani pixel intensity'si tutulur.

- Training set için farklı car pictures kullanılır. Training set learning algorithm'e beslenir ve bir classifier elde edilir.

→ Şimdi basitçe problemi açıklığa kavuşturmak için feature'lar imin pixel 1 ve pixel 2 adında 2 kritik pixel olduğunu kabul ediyorum.

"+" cars
"−" non-cars

- Her datanın bir araba veya araba-değil örneği teşkil ettiğine dikkat et!

- Bu dağılımı ayırabilmek için kırmızı çizgiyi yaratacak bir non-linear hypothesis gerekir.

- 50×50 pixel images kullansak dahi → 2500 pixel gibi bir initial feature size oluyor yani $n = 2500$ bunu sadece linear lines için kullanabiliriz.

0-255 pixel intensity values

$$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2 intensity} \\ \vdots \\ \text{pixel 2500 intensity} \end{bmatrix}$$

- Tüm quadratic functions içeren bir nonlinear hypothesis kullanmak için ise $(x_i \times x_j) \approx$ $\boxed{\text{3 million features}}$

Neural Networks are much better way to learn complex nonlinear hypothesis even when n is large!

- W4 - Neurons and the Brain -

- Origins of NNs: Algorithms that try to mimic the brain.
- Was widely used in 80s and 90s; popularity diminished in late 90s.

Recently resurgence occured: State-of-the-art technique for many applicati

---

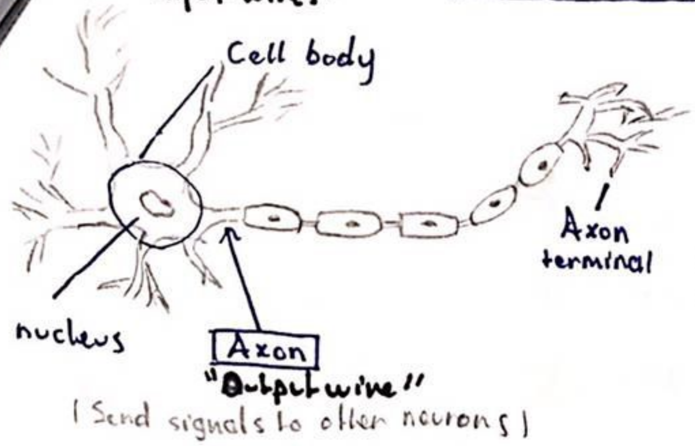
- Beynimin bunca karmaşık fonksiyonu yerine getirmek için 1000'lerce farklı programdan oluşmalı diye düşünebilirsin ama bu yanlış.
- Bunun yerine beyin tek bir learning algorithm ile her işi öğreniyor

  - Bazı kanıtlardan bahsedebiliriz:
    - Örneğin duyma işini yapan Auditory Cortex'e kulaktan giden bağlantı kesilip yerine gözden gelen data verilirse auditory cortex learns to see
    - Somatosensory Cortex dokunma hissini sağlar. Buna da re-wiring yapıp görsel veri sağlarsak yine görmeyi öğrenir.

  - Yani almost any sensor almost any part of brain'e bağlanınca brain figures it out how to deal with it. Biz de belki bu öğrenme algoritmasına yaklaşarak sonuç elde edebiliriz.

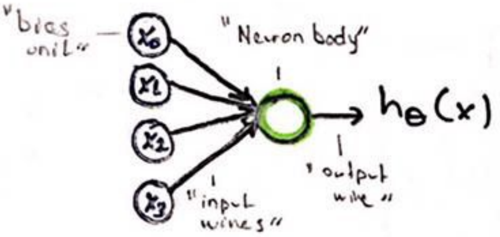
  - Dil ile görme (Seeing with your tongue).
  - Human echolocation (sonar).
  - Haptic belt. Direction sense.
  - Implanting a 3rd eye.

-W4- NN Model Representations I-



- At a simplistic level, a neuron is a computational unit that gets a number of inputs through dentrites, does some computations then sends outputs via its axon to other neurons.

Artificial Neuron: Neuron'u bir logistic unit olarak modelleyeceğiz.

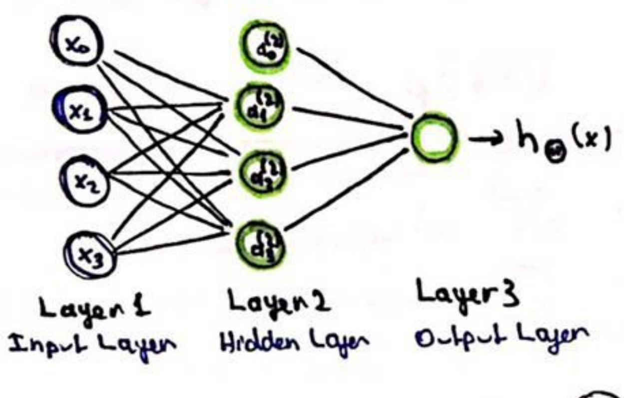


$$\boxed{h_\theta(x) = \frac{1}{1+e^{-\theta^T \cdot x}}} \quad \text{where } x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}, \ \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

"weights" or "parameters"

- Bias unit, x_0 bazen çizilir bazen çizilmez!
- This an artificial neuron with a Sigmoid / Logistic Activation Function.

Neural Network:



- $a_i^{(j)}$: "activation" of unit i in layer j.
- $\Theta^{(j)}$: Matrix of weights controlling function mapping from layer j to layer $j+1$.

Activation: The value that's computed and output by a specific unit or neuron.

$$a_1^{(2)} = g(\Theta_{10}^{(1)} \cdot x_0 + \Theta_{11}^{(1)} \cdot x_1 + \Theta_{12}^{(1)} \cdot x_2 + \Theta_{13}^{(1)} \cdot x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} \cdot x_0 + \Theta_{21}^{(1)} \cdot x_1 + \Theta_{22}^{(1)} \cdot x_2 + \Theta_{23}^{(1)} \cdot x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} \cdot x_0 + \Theta_{31}^{(1)} \cdot x_1 + \Theta_{32}^{(1)} \cdot x_2 + \Theta_{33}^{(1)} \cdot x_3)$$

$\Theta^{(1)} \in \mathbb{R}^{3 \times 4}$
Layer 1 ile Layer 2 bağlar

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} \cdot a_0^{(2)} + \Theta_{11}^{(2)} \cdot a_1^{(2)} + \Theta_{12}^{(2)} \cdot a_2^{(2)} + \Theta_{13}^{(2)} \cdot a_3^{(2)})$$

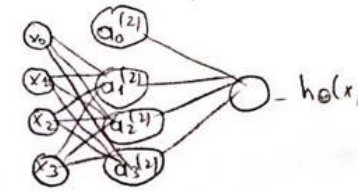
$\Theta^{(i)}$ size: $i+1$. layerdaki units (w/o bias) $\times$ i. layerdaki units (w bias)

$\Theta^{(2)} \in \mathbb{R}^{4 \times 1}$
Layer 2 ile Layer 3 bağlar

- W4 - NN Model Representations II -

Forward Propagation: Vectorised Implementation

$$a_1^{(2)} = g(\Theta_{10}^{(1)} \cdot x_0 + \Theta_{11}^{(1)} \cdot x_1 + \Theta_{12}^{(1)} \cdot x_2 + \Theta_{13}^{(1)} \cdot x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} \cdot x_0 \quad \Theta_{21}^{(1)} \cdot x_1 \quad \Theta_{22}^{(1)} \cdot x_2 \quad \Theta_{23}^{(1)} \cdot x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} \cdot x_0 \quad \Theta_{31}^{(1)} \cdot x_1 \quad \Theta_{32}^{(1)} \cdot x_2 \quad \Theta_{33}^{(1)} \cdot x_3)$$



$$h_\Theta(x) = g(\Theta_{10}^{(2)} \cdot a_0^{(2)} + \Theta_{11}^{(2)} \cdot a_1^{(2)} + \Theta_{12}^{(2)} \cdot a_2^{(2)} + \Theta_{13}^{(2)} \cdot a_3^{(2)})$$

- $x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$
- $z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} = \overbrace{\begin{bmatrix} \Theta_{10}^{(1)} \cdot x_0 + \Theta_{11}^{(1)} \cdot x_1 + \Theta_{12}^{(1)} \cdot x_2 + \Theta_{13}^{(1)} \cdot x_3 \\ \Theta_{20}^{(1)} \cdot x_0 + \Theta_{21}^{(1)} \cdot x_1 + \Theta_{22}^{(1)} \cdot x_2 + \Theta_{23}^{(1)} \cdot x_3 \\ \Theta_{30}^{(1)} \cdot x_0 + \Theta_{31}^{(1)} \cdot x_1 + \Theta_{32}^{(1)} \cdot x_2 + \Theta_{33}^{(1)} \cdot x_3 \end{bmatrix}}^{\Theta^{(1)} \cdot x}$

- $z^{(2)} = \Theta^{(1)} \cdot x = \Theta^{(1)} \cdot a^{(1)}$
- $a^{(2)} \in \mathbb{R}^3 = g(z^{(2)}) \in \mathbb{R}^3 \longrightarrow a^{(2)} = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$ Bunu bulmuş olduk ama $h_\Theta(x)$ için $a_0^{(2)} = 1$'e de ihtiyaç var!
- Add $a_0^{(2)} = 1$ then $a^{(2)} \in \mathbb{R}^4$ → Şimdi $h_\Theta(x)$'i bulabiliriz.
- $h_\Theta(x) = a^{(3)} = g(z^{(3)})$ This is a relatively efficient way of computin $h(x)$

- This process of computin $h_\Theta(x)$ is called "Forward Propagation".

What NN is doing and how they help us to learn complex nonlinear hypothesis.

- ilk layer'ı görmezden gelirsek aslında sadece 1 logistic regression unit var, ve 4 input var yani 4 features: $(a_0^{(2)}, a_1^{(2)}, a_2^{(2)}, a_3^{(2)})$
Bu zaten yaptığımız bir şey.

$$h_\Theta(x) = g(\Theta_{10}^{(2)} \cdot a_0 + \Theta_{11}^{(2)} \cdot a_1 + \Theta_{12}^{(2)} \cdot a_2 + \Theta_{13}^{(2)} \cdot a_3)$$

- NN works like logistic regression except rather than using the original features x_0, x_1, x_2, x_3 it uses this new features and these new features are learned as functions of inputs.

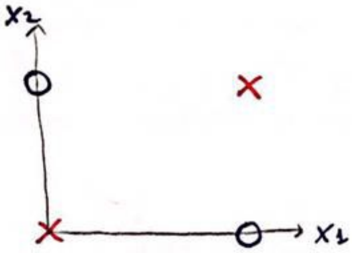
NNs : Examples and Intuitions

-W4-

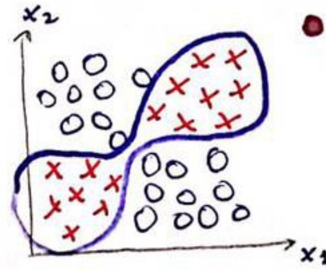
How a NN can compute a complex non-linear function of the input?

Non-linear Classification Example: XOR/XNOR

- x_1, x_2 are binary (0 or 1)



- Bu aslında sağdaki örneğin simplified hali!

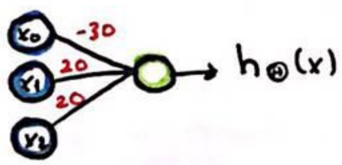


- Learning a non-linear decision boundary. Bu problem çözülebilir.

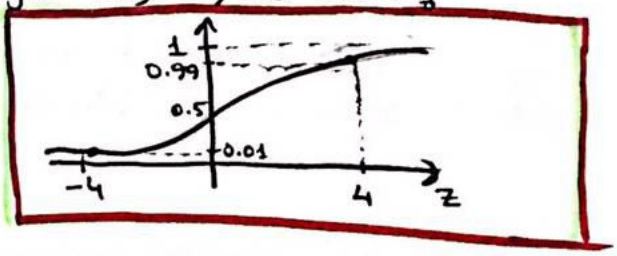
- Yukarıdaki örnek bir $y = x_1$ XNOR x_2 örneği. 11 ve 00 için $y=1$
- Bize böyle bir training set verildi. NN ile öyle bir $h_\Theta(x)$ elde edeceğiz ki, XNOR fonksiyonunun görevini yapsın!
  - Bunu başarmak için daha basite ineceğiz önce "AND" ile başlayacağız.

Simple Example: AND

- $x_1, x_2 \in \{0,1\}$ (Binary Inputs)
- $y = x_1$ AND x_2 (Target labels)



- $\boxed{h_\Theta(x) = g(\Theta^{(1)}_{10} \cdot x_0 + \Theta^{(1)}_{11} \cdot x_1 + \Theta^{(1)}_{12} \cdot x_2)}$ Formunda olduğunu biliyoruz şimdi parametrelere elle değer atayalım ve sırasıyla -30, 20, 20 olsun!

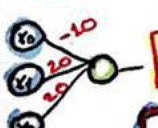


- $\boxed{h_\Theta(x) = g(-30 + 20x_1 + 20x_2)}$

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

$\Rightarrow \boxed{h_\Theta(x) \approx x_1 \text{ AND } x_2}$

Example: OR

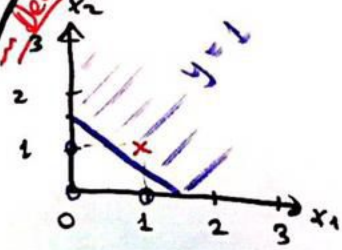


$\boxed{h_\Theta(x) = g(-10 + 20 \cdot x_1 + 20 x_2)}$

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

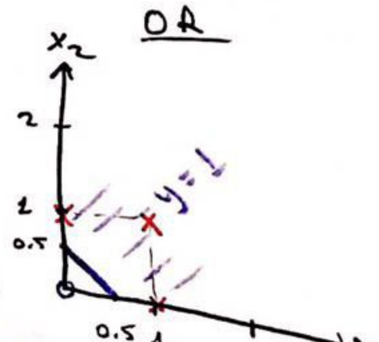
~ Decision Boundaries ~

AND



$20x_1 + 20x_2 \geq 30$

OR



$20x_1 + 20x_2 \geq 10$

- NNs: Examples and Intuitions -

- W4 -

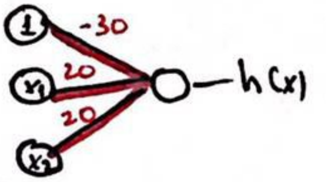
NOT Function or Negation:

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

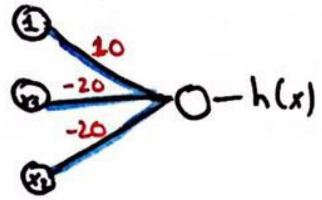
x_1	$h_\Theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

- x_0'ın başına "+" diğerlerinin başına large negative parameters!

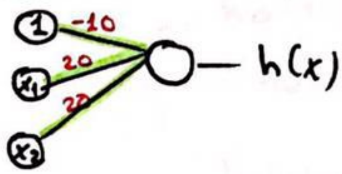
Putting it Together: x_1 XNOR x_2:



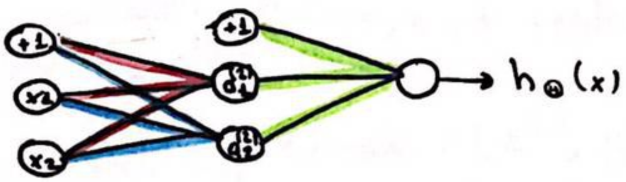
x_1 AND x_2



(NOT x_1) AND (NOT x_2)

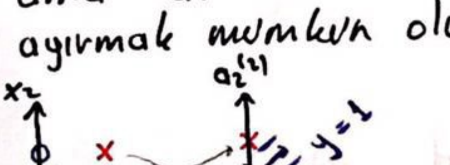


x_1 OR x_2



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

- x_1 ve x_2'yi kullanarak datasetini tek bir line ile ayırmak imkansız ama $a_1^{(2)}$ ve $a_2^{(2)}$ featurelarını kullanınca tek bir line ile datasetini ayırmak mümkün oluyor.



Eklenen her layer ile yeni bir eksen takımı yeni yeni features oluşturulmuş oluyor. Training set mirrorlanıyor.

Eklenen her neuron ise yeni bir line yani a_3 eklenseydi bir önceki layer eksenine ($x_1 - x_2$) 3 çizgi çekmiş olacaktık, daha mantıklısı son layer'a ekleseydik a_1, a_2'ye 2 çizgi çekmiş olurduk.

NN şöyle bir şey yapıyor, original feature ekseninde çok complex şekiller ile ayrılması gereken bir data seti kendi yeni featurelarını yeni layerlar ile oluşturarak daha basitçe ayırıyor.