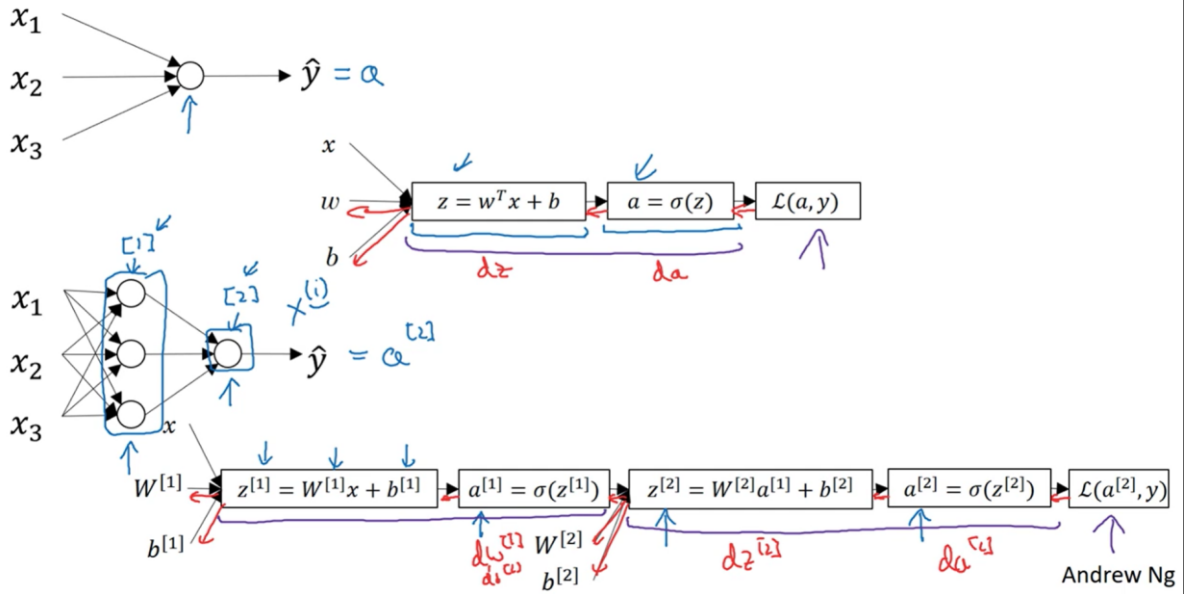


COURSE 1 W3

Vid 25

Neural Network Overview

What is a Neural Network?



Bildiğimiz üzere, NN dediğimiz log. Reg. unitlerinin bir araya gelmesinden oluşuyor. Önceden tek bir unit olduğu için forward propagation için parametreler ve x ile z hesaplanıyordu sonra buradan a yani çıkış hesaplanıyordu daha sonra loss hesaplanabilir.

NN için ise yukarıdaki notation'ı kullanıyoruz 1 dediğimizde 1. Layer ile ilgili parametre ve çıkışlardan bahsediyoruz demektir. Sonuçta görüldüğü gibi önce a^1 sonra a^2 yi hesaplarız. Backward propagation için ise yine sondan başa doğru adım adım gidilir.

Vid 26

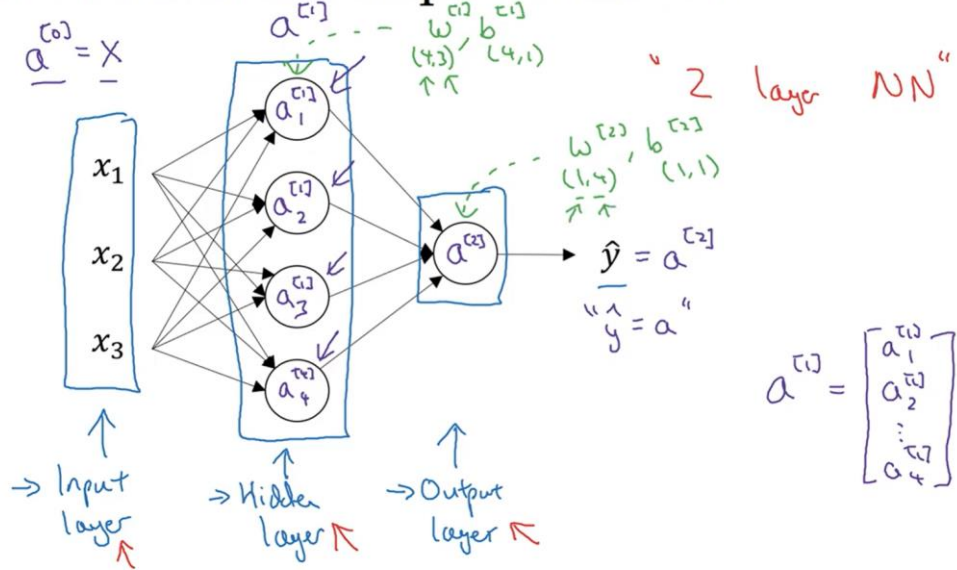
Neural Network Representation

Aşağıdaki örnek 2 layerlı bir NN. Input layer sayılmaz. Input layer'a a_0 diyebiliriz, a_1, a_2 diye gider.

a_1 dediğimizde 4×1 'lik bir vektörden bahsediyoruz. Her satırında bir başka elemanın çıkışı var.

w_1 dediğimizde layer1'in her uniti için 3 adet (3 input var) parametre olacak bu sebeple 4×3 olur, benzer şekilde her unit için bir b olacak 4×1 olur. Son layer ise a_1 'i input alır.

Neural Network Representation

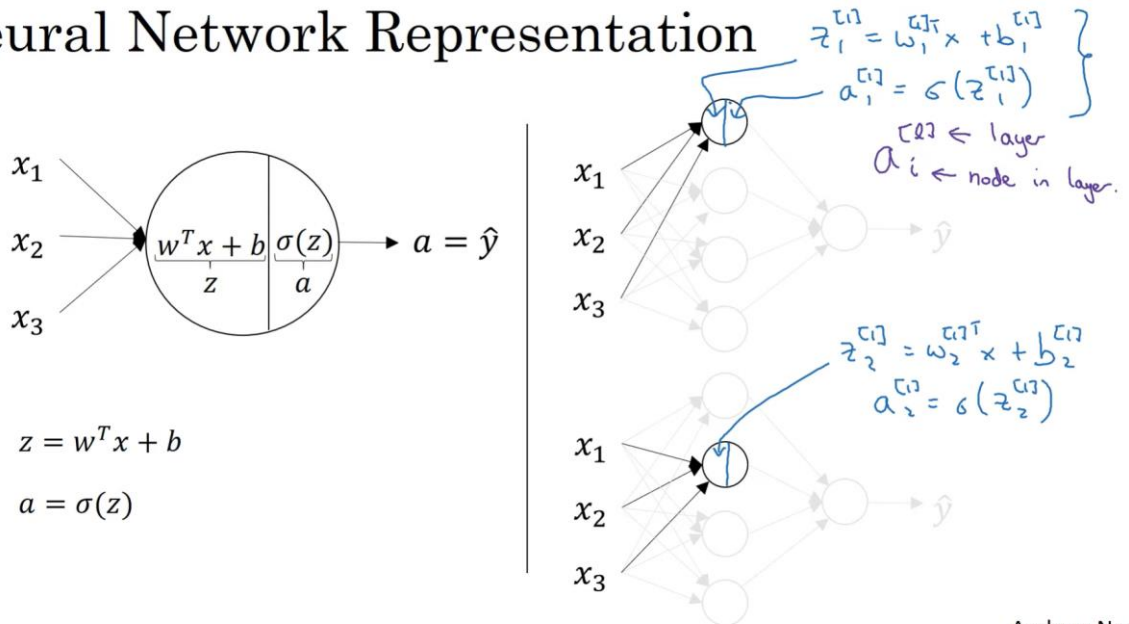


Andrew Ng

Vid 27

Neural Network Output

Neural Network Representation

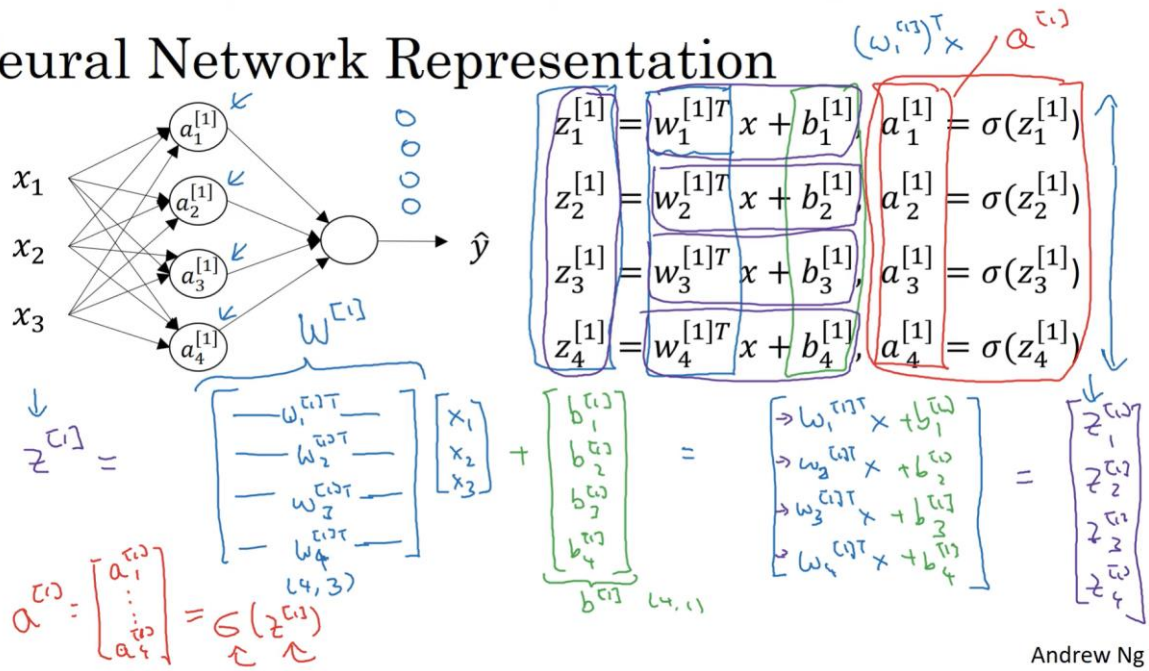


Andrew Ng

Superscript layer'ı gösterirken subscript ilgili layer'ın hangi node'u olduğunu gösterir. Her node için yapılan işlem logistic regression ile aynı.

Her eleman için z hesaplamasını daha düzgün biçimde yazalım ve bu hesaplamaları nasıl vektörize ederiz ona bakalım:

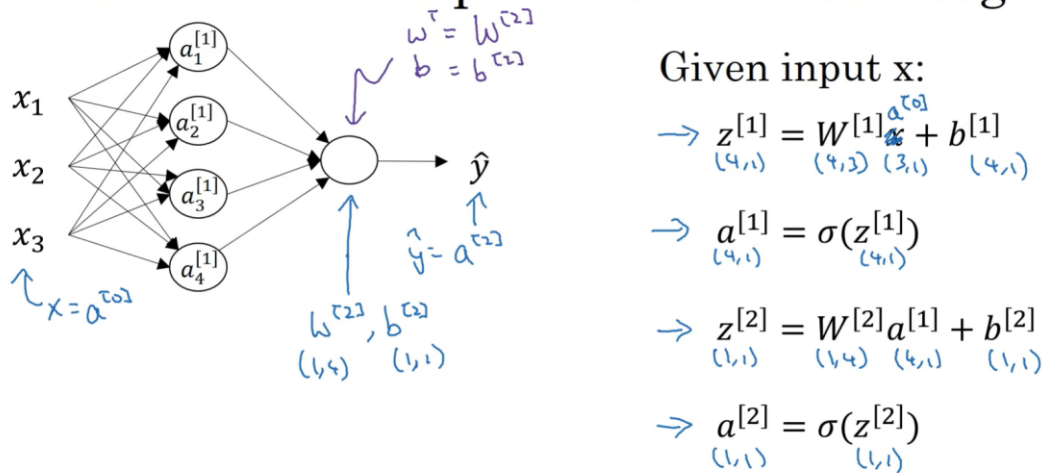
Neural Network Representation



T transpose anlamında, $w^{[1]}$ 'in transpose'u alınmış. Yani temelde her unit için z hesaplanıyor. 4 adet unit var aynı işlem 4 kere yapılabilir bunu vektörize edersek elle yazılan kısım ortaya çıkar. A 'yı bulmak içinde $z^{[1]}$ 'in sigmoidi alınır.

Biraz daha temiz yazarsak aşağıdaki gibi olur. Burada W_2 dediğimizin logistic regression için w^T 'ye denk düştüğünü unutma.

Neural Network Representation learning

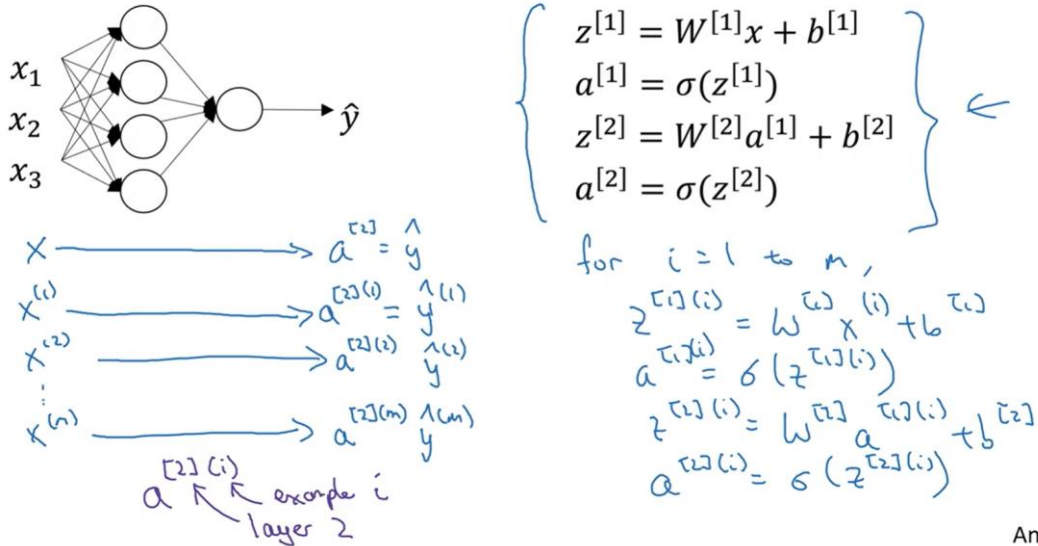


Tek bir training example için vectorization'ı gördük. Bir sonraki kısımda m training example için bu computations nasıl vectorize edilebilir ona bakacağız.

Vid 28

Vectorizing Across Multiple Examples

Vectorizing across multiple examples



Andrew Ng

Tek bir training ex. için sağ üstte görüldüğü gibi çıkışı hesaplayabiliyoruz. Vektörize etmezsek, aynı işlemi x_1 'den x_m 'e kadar her training ex. için yapmamız gerekecek. Bunun için gerekli for loop yukarıda verilmiş, fakat daha önce de söylediğimiz gibi looping verimsiz olacaktır, o yüzden vektörizasyon uygulamalıyız.

Burada $a[2](i)$ dediğimizde i . training example için bulunan $a[2]$ 'den bahsediliyor. Bu $a[1](i)$ olsa aslında bir vektör olacağına dikkat et. Çünkü $a[1]$ aslında hidden layer'ın tüm unitlerinin çıktısını tutan bir vektör.

Peki nasıl vektörize ederiz ve bu for looptan kurtuluruz?

Vectorizing across multiple examples

for $i = 1$ to m :

$$\begin{aligned} z^{[1]}(i) &= W^{[1]}x^{(i)} + b^{[1]} \\ a^{[1]}(i) &= \sigma(z^{[1]}(i)) \\ z^{[2]}(i) &= W^{[2]}a^{[1]}(i) + b^{[2]} \\ a^{[2]}(i) &= \sigma(z^{[2]}(i)) \end{aligned}$$

$$X = \begin{bmatrix} x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ \uparrow & \uparrow & & \uparrow \\ & (n_x, m) & & \end{bmatrix}$$

$$\begin{aligned} z^{[1]} &= W^{[1]}X + b^{[1]} \\ \Rightarrow A^{[1]} &= \sigma(z^{[1]}) \\ \Rightarrow z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\ \Rightarrow A^{[2]} &= \sigma(z^{[2]}) \end{aligned}$$

$$\begin{aligned} z^{[1]} &= \begin{bmatrix} z^{[1]}(1) & z^{[1]}(2) & \dots & z^{[1]}(m) \\ 1 & 1 & & 1 \end{bmatrix} \\ A^{[1]} &= \begin{bmatrix} a^{[1]}(1) & a^{[1]}(2) & \dots & a^{[1]}(m) \\ 1 & 1 & & 1 \end{bmatrix} \end{aligned}$$

Andrew Ng

Yukarıda sağ üstte görülen form ile bu loopingden kurtuluruz. Sağda Z, X ve A'ların capital olduğuna dikkat et.

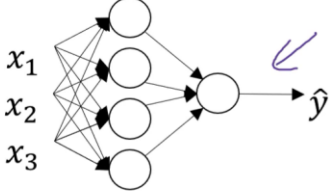
Solda tek tek her training ex. için z hesaplarken, sağda X'i kullanarak tüm training example için Z bulunur. Daha sonra bu sonuçlardan A1, Z2 ve A2 sırasıyla elde edilebilir.

Bu noktada matrisler yukarıda açıklanmış. X matrisinde sütun bir training ex.'a karşılık geliyor her satır ise bir feature'a karşılık geliyor. Benzer şekilde Z1 matrisinde her sütun bir training example'a karşılık elde edilen Z1 vektörüdür, her satır bir başka unit'e karşılık gelir. Mesela en sol en üstteki eleman 1. Training example için 1. Hidden unit'e karşılık gelen Z'dir. A da aynı şekilde.

Vid 29

Explanation for Vectorized Implementation

Recap of vectorizing across multiple examples



$$X = \begin{bmatrix} | & | & \dots & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & \dots & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & \dots & | \\ a^{1} & a^{[1](2)} & \dots & a^{[1](m)} \\ | & | & \dots & | \end{bmatrix}$$

for i = 1 to m

- $z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$
- $a^{[1](i)} = \sigma(z^{[1](i)})$
- $z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$
- $a^{[2](i)} = \sigma(z^{[2](i)})$

$Z^{[1]} = W^{[1]}X + b^{[1]}$ ← $W^{[1]}A^{[0]} + b^{[1]}$
 $A^{[1]} = \sigma(Z^{[1]})$
 $Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$
 $A^{[2]} = \sigma(Z^{[2]})$

Handwritten notes: $x = a^{[0]}$, $x^{(i)} = a^{[0](i)}$

Andrew Ng

Vid 30

Activation Functions

Neuronlar için hangi activation function'ı seçeceğimiz karar vermeliyiz. Şuana kadar hep sigmoid kullandık ama, bu her zaman en iyi seçim değildir.

Genel durumda aşağıdaki sigmoid fonksiyonu yani $\sigma(z)$ yerine $g(z)$ yazabiliriz. Burada g için tanh, relu, veya linear gibi farklı activation function'lar seçilebilir.

Bu da kullanılabilir ama Relu da iş görür en sık kullanılan reludur. İkisinin de bir avantajı z 'nin çok büyük bir range için derivative'inin 0'dan oldukça büyük olmasıdır. Bu da hızlı training sağlıyor.

Never use sigmoid except for the output unit, mostly use relu or leaky relu, you can try tanh as well.

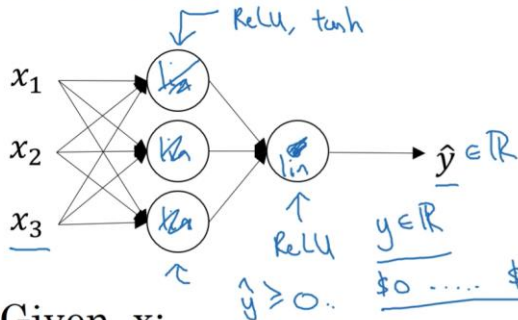
Vid 31

Why NonLinear Activation Functions?

Bir neural network'un ilerleyen layerlar ile birlikte complex functions hesaplayabilmesi için yani complex decision boundaries oluşturabilmesi için mutlaka ve mutlaka nonlinear activation functions kullanılmalı (relu da buna dahil oluyor), çünkü linear function kullanmazsak yani $a=g(z)$ yerine $a=z$ dersek, aynen linear regression için yaptığımız gibi, ve bu şekilde bir çok linear regression unit kullanırsak sonuçta elde edilecek çıkış aslında yine doğrusal bir fonksiyondan ötesi olmayacak, doğrusal olarak ayrılamayan data olursa patlayacak.

Son olarak şundan da bahsedelim, linear activation'ı output da kullanabiliriz mesela -sonsuz ile sonsuz arasında bir çıkış almam gerekiyor o zaman linear activation kullanabilir. Bu durumda bile eğer housing prices gibi bir tahmin yapmak istiyorsak Relu kullanabiliriz.

Activation function



Given x :

$$\begin{aligned} \rightarrow z^{[1]} &= W^{[1]}x + b^{[1]} \\ \rightarrow a^{[1]} &= g^{[1]}(z^{[1]}) = z^{[1]} \\ \rightarrow z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\ \rightarrow a^{[2]} &= g^{[2]}(z^{[2]}) = z^{[2]} \end{aligned}$$

$g(z)=z$
"linear activation function"

$$\begin{aligned} a^{[1]} &= z^{[1]} = W^{[1]}x + b^{[1]} \\ a^{[2]} &= z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} &= W^{[2]}(W^{[1]}x + b^{[1]}) + b^{[2]} \\ &= (W^{[2]}W^{[1]})x + (W^{[2]}b^{[1]} + b^{[2]}) \\ &= W'x + b' \\ g(z) &= z \end{aligned}$$

Andrew Ng

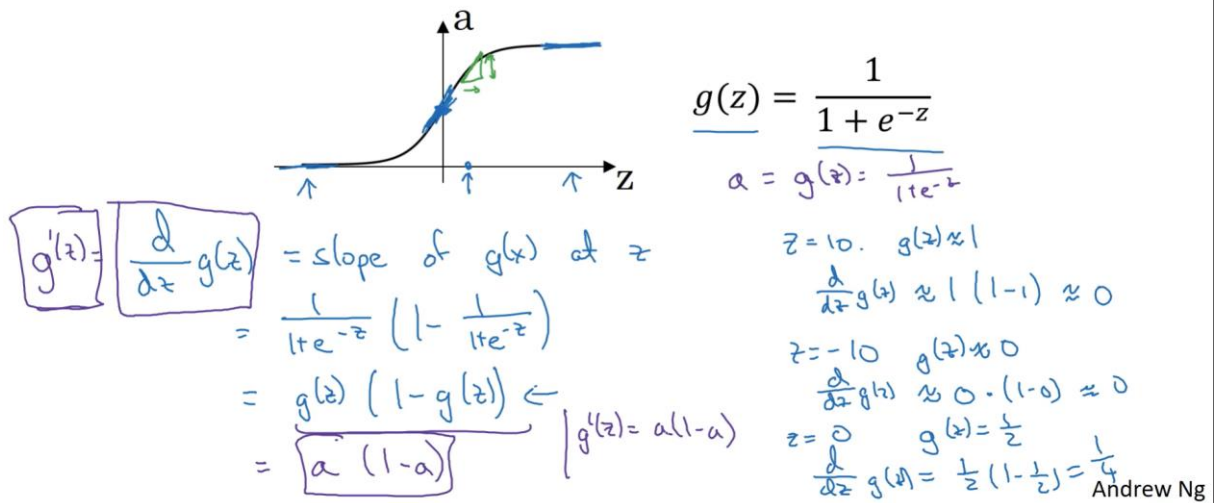
Vid 32

Derivatives of Activation Functions

İlk olarak sigmoid activation function için derivative hesaplırsak aşağıdaki gibi bulunur, $g(z)$ yerine z koyabileceğimizi unutma.

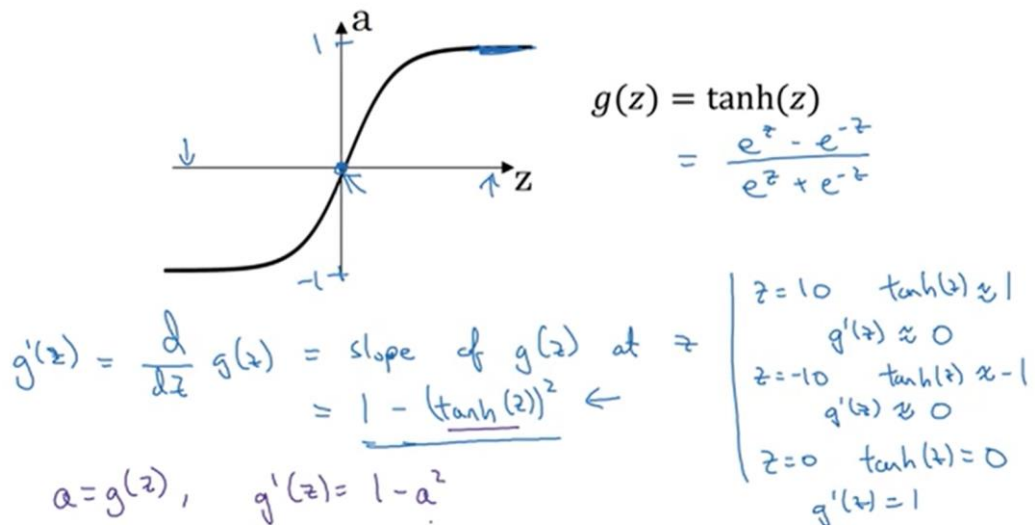
z büyükken veya küçükken slope 0'a gider, $z=0$ iken ise slope $\frac{1}{4}$ olarak bulunur.

Sigmoid activation function



Tanh'a bakarsak:

Tanh activation function

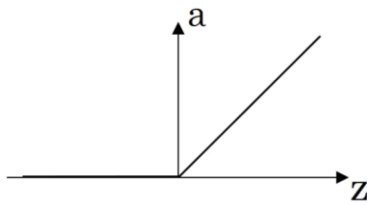


Andrew Ng

Son olarak Relu ve Leaky Relu'ya bakalım:

Burada matematiksel olarak $z=0$ iken türev undefined olur ama pratikte bunu hesaba katmazsak da gayet içi çalışır, $z=0$ iken türevi 1 veya 0 kabul edebiliriz.

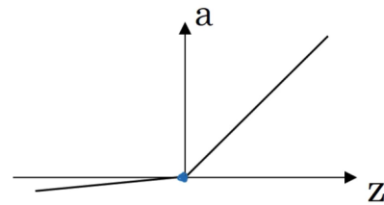
ReLU and Leaky ReLU



ReLU

$$g(z) = \max(0, z)$$

$$\rightarrow g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$



Leaky ReLU

$$g(z) = \max(0.01z, z)$$

$$g'(z) = \begin{cases} 0.01 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

Andrew Ng

Vid 33

Gradient Descent for NNs

Bu kısımda back propagation ve gradient descent için gerekli algoritmayı göreceğiz, bir sonraki kısımda ise back propagation ile ilgili intuiton edinmeye çalışacağız.

Gradient descent for neural networks

Parameters: $\begin{matrix} w^{[0]} & b^{[0]} & w^{[1]} & b^{[1]} \\ (n^{[0]}, 1) & (n^{[1]}, 1) & (n^{[2]}, 1) & (n^{[3]}, 1) \end{matrix}$ $n_x = n^{[0]}$, $n^{[1]}$, $n^{[2]} = 1$

Cost function: $J(w^{[1]}, b^{[1]}, \underbrace{w^{[2]}}_{\uparrow a^{[1]}}, \underbrace{b^{[2]}}_{\uparrow a^{[2]}}) = \frac{1}{n} \sum_{i=1}^n \ell(\hat{y}_i, y_i)$

Gradient Descent:

→ Report {

→ Compute products $(y^{(i)}, i=1, \dots, m)$

$$\underline{dw^{(i)}} = \frac{\partial J}{\partial w^{(i)}} , \quad \underline{db^{(i)}} = \frac{\partial J}{\partial b^{(i)}} , \quad \dots$$

$$W^{(i)} := W^{(i)} - \alpha \partial W^{(i)}$$

$$b^{(1)} = b^{(2)} - \alpha \nabla b^{(2)}$$

(2) \dots (2)

Yukarı n_0, n_1, n_2 dediğimiz 0. 1. Ve 2. Layerdaki unit sayısı. Cost function'ı ortalama loss olarak hesaplayabiliriz. Bunları birleştirince Gradient Descent için yukarıdaki gibi bir algoritma çıkar, önce forward p. ile prediction'ı ve loss 'u hesaplarız sonra bunu kullanarak gradients hesaplanır ve bu gradients ile bir step atılır.

Formulas for computing derivatives

$$\begin{array}{l}
 \text{Forward propagation:} \\
 z^{[1]} = w^{[0]}x + b^{[0]} \\
 A^{[1]} = g^{[1]}(z^{[1]}) \leftarrow \\
 z^{[2]} = w^{[1]}A^{[1]} + b^{[1]} \\
 A^{[2]} = g^{[2]}(z^{[2]}) = \sigma(z^{[2]})
 \end{array}
 \quad
 \begin{array}{l}
 \text{Back propagation:} \\
 dz^{[2]} = A^{[2]} - Y \leftarrow Y = [y^{(1)} \ y^{(2)} \dots \ y^{(n)}] \\
 dw^{[2]} = \frac{1}{n} dz^{[2]} A^{[1]T} \\
 db^{[2]} = \frac{1}{n} \text{np.sum}(dz^{[2]}, \text{axis}=1, \text{keepdims}=\text{True}) \\
 dz^{[1]} = \underbrace{w^{[1]T}}_{(n^{[1]}, m)} dz^{[2]} \times \underbrace{g^{[1]'}(z^{[1]})}_{\text{element-wise product}} \quad (n^{[1]}, m) \\
 dw^{[1]} = \frac{1}{n} dz^{[1]} x^T \\
 db^{[1]} = \frac{1}{n} \text{np.sum}(dz^{[1]}, \text{axis}=1, \text{keepdims}=\text{True})
 \end{array}$$

Solda forward propagation'ı görüyoruz sağda ise backpropagation var. Keepdims=True dememizin sebebi output olarak rank1 vektör yerine boyutları belirli vektör almak.

Bir sonraki kısımda bu backpropagation operasyonunun intuion'unu göreceğiz...

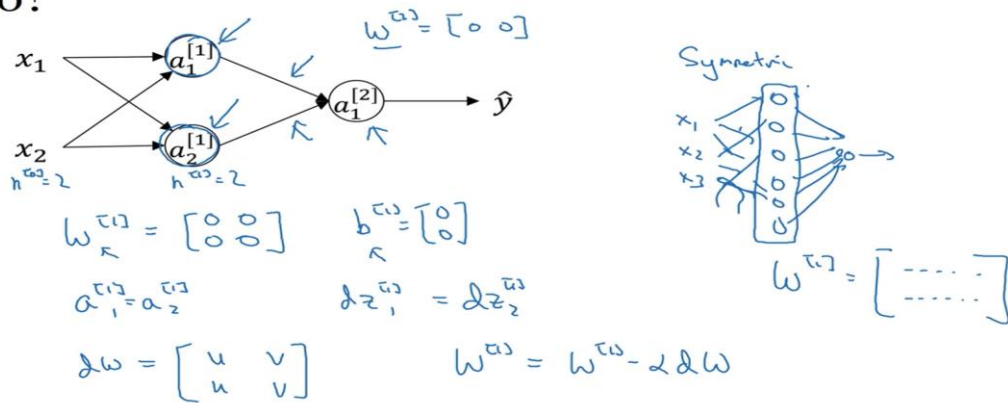
Vid 35

Random Initialization

Logistic regression için weightleri 0 olarak initialize etmiştik, ancak NN için bunu yapamayız. b term'i 0 olarak initialize edebiliriz ancak w'yü edemeyiz. Eğer edersek, hidden unitler aynı olacaktır ve backpropagation ile ikisi içinde gradient aynı çıkacaktır sonuçta aynı hidden units birebir aynı olur.

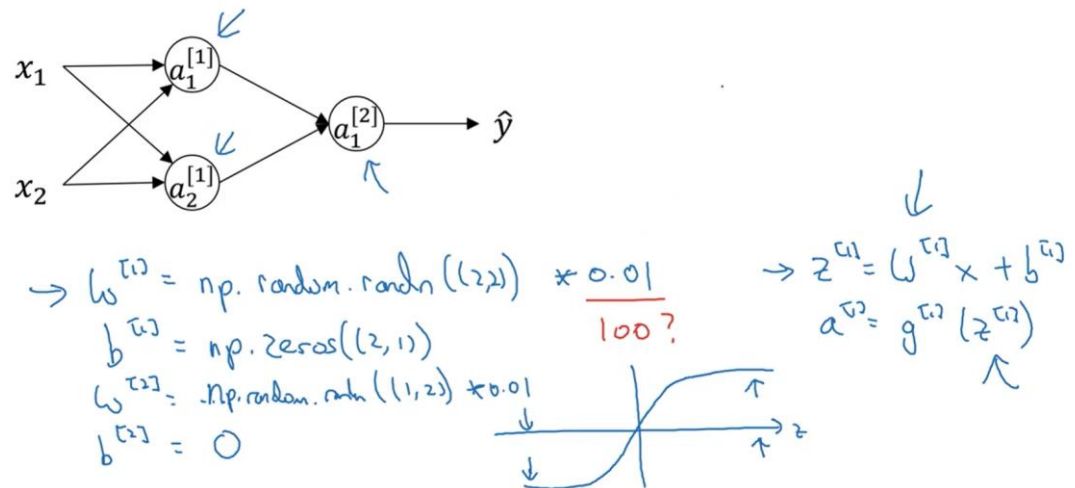
Bu yüzden 0'a yakın fakat random sayılar ile w'yi initialize ederiz. 0 dememizin sebebi sigmoid veya tanh için büyük veya küçük w değerleri sonuçta büyük veya küçük z değerlerine yol açabilir ve nihayetinde slow learning gerçekleşir.

What happens if you initialize weights to zero?



Andrew Ng

Random initialization



Andrew Ng