

# Week-3

## Neural Network Representation

Çizilen sinir ağlarının neler ifade ettiğini göstereceğiz.

Daha öncesinde girdileri ve alternatif notasyonları belirtmek için **x vektörünü ifade ediyorduk**.

Bu, a üstü parantez içinde sıfır eşittir x olarak ifade edilir. (Sinir ağının Hidden kısmında)

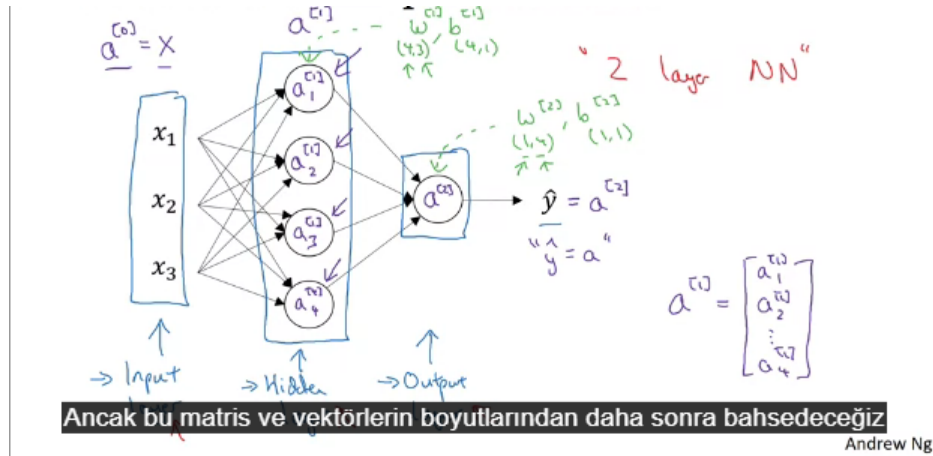
Yeni sinir ağ bağlantımızda hangi katmanın nereden geldiğini açıkça görmek için köşeli parantez kullanılır.

Görülen sinir ağı, iki katmanlı sinir ağı olarak adlandırılır.

Sinir ağlarındaki katmanları saydığımızda "Input" katmanını saymayız. Sadece HiddenLayer sayısı ve output çıktısı katman sayısında sayılır. ( 2 LAYER NN )

w ve b değerleri hidden katmanıyla bağlantılı olarak output ekranında  $\begin{bmatrix} w \\ b \end{bmatrix}$  şeklinde outputlara yanıt verebilir.

Bu matris ve vektörlerin boyutlarından daha sonra bahsedeceğiz.



## Computing a Neural Network's Output

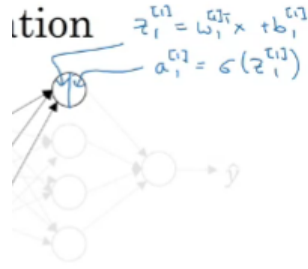
Bu videoda sinir ağlarının çıktılarını tam olarak nasıl çıktığını öğreneceğiz.

İlk olarak görüldüğü gibi z hesaplanır sonra da z'nin sigmoid fonksiyonu ile a(aktivasyon) hesaplanır. Sinir ağı bunu defalarca tekrarlar.

Şimdi de gizli katmandaki bir düğüme odaklanalım.

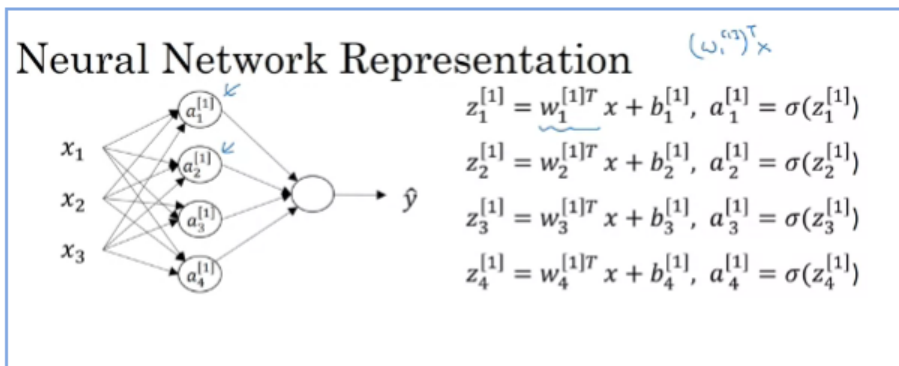
İlk adım düğümün sol yarısında olan ki bu düğüm  $z[1] = wTx + b$

İkinci adım  $a[1,1] = \text{sigmoid of } z[1,1]$  gibi hesaplamaları yapar. ( $a[1] = \text{sigmoid}[z[1]]$ )



Köşeli parantez içindeki  $a[n]$  katman numarasıdır.

Alt indis "i" ise bu katmandaki düğümü ifade eder.



Görüldüğü üzere her sinir ağı(indis) için iki bölmeli işlemler yapılır. Fakat indisler değişse de aynı katman üzerinde işlemler yapılmaktadır.

Sinir ağı oluşturmada bu dört denklem tek başına işlevsiz olsa da daha sonra işlevsel hale getirmenin ilk adımı olan vektörleşme işlemine başlıyoruz. Burada Z'nin nasıl vektör olarak işlem yapacağını ve dönüştürmeyi göreceğiz.

İlk adım W değerlerini bir matrise yükleyelim. (Elimize  $w^T$  değerlerinden oluşan bir sütun vektörü geçer.

$$\begin{bmatrix} w_1^{[1]T} \\ w_2^{[1]T} \\ w_3^{[1]T} \\ w_4^{[1]T} \end{bmatrix}$$

Ardından sinir ağıımızda bulunan  $x_1, x_2$  ve  $x_3$  için bir sütun vektörü oluşturuyoruz.

Ardından hesaplamalarda bulunan b değerleri içinde aynı şekilde bir sütun vektörü oluşturuyoruz.

W matrisi ve x matrisini kartezyen çarpımına sokup + b matrisini ekliyoruz.

$$\begin{bmatrix} w_1^{[1]T} \\ w_2^{[1]T} \\ w_3^{[1]T} \\ w_4^{[1]T} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix}$$

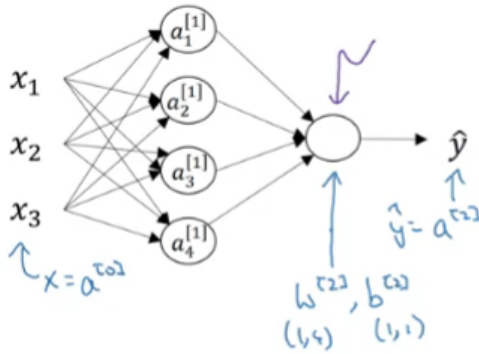
Düğümün sol tarafı için gayet uyumlu bir matris ortaya çıktı ve bu matris tamamiyle  $Z[1]$  (KATMAN) 'a eşittir. Bir sonraki işlem "a" değerlerini işleme koymak, tabi doğal olarak  $a[1]$  olarak tanımlanacak.(KATMAN)

$$a^{(1)} = \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \\ a_4^{(1)} \end{bmatrix} = \begin{bmatrix} 1 \\ -0.5 \\ 0.5 \\ 0 \end{bmatrix}$$

Şimdi de

Şimdi de ikinci katmanda işlemlere bakalım. Aynı işlemi yapıyoruz. 4 tane gelen W ağırlık ile çıkan a değerlerini çarpıyoruz.

## Neural Network Representation learning



Given input x:

$$\rightarrow z^{[1]} = W^{[1]} a^{(0)} + b^{[1]}$$

$\begin{pmatrix} 4,1 \end{pmatrix} \quad \begin{pmatrix} 4,3 \end{pmatrix} \quad \begin{pmatrix} 3,1 \end{pmatrix} \quad \begin{pmatrix} 4,1 \end{pmatrix}$

$$\rightarrow a^{[1]} = \sigma(z^{[1]})$$

$\begin{pmatrix} 4,1 \end{pmatrix} \quad \begin{pmatrix} 4,1 \end{pmatrix}$

$$\rightarrow z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$

$\begin{pmatrix} 1,1 \end{pmatrix} \quad \begin{pmatrix} 1,4 \end{pmatrix} \quad \begin{pmatrix} 4,1 \end{pmatrix} \quad \begin{pmatrix} 1,1 \end{pmatrix}$

$$\rightarrow a^{[2]} = \sigma(z^{[2]})$$

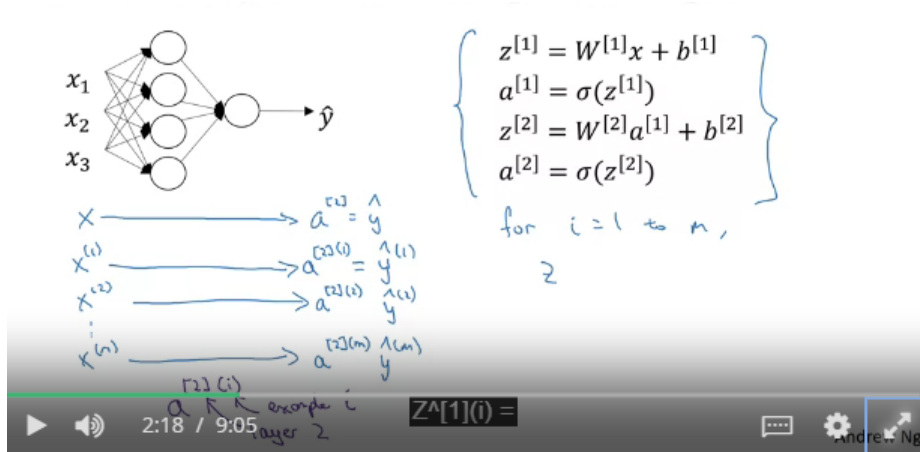
$\begin{pmatrix} 1,1 \end{pmatrix} \quad \begin{pmatrix} 1,1 \end{pmatrix}$

Gizli katmanlı bir sinir ağı biriminiz olduğunda çıktıyı işleyebilmek için sadece bu dört denkleği uygulamak gerekir.

### Vectorizing Across Multiple Examples

Bu videoda, birden fazla eğitim örneğinde nasıl vektörelleştireceğimizi göreceğiz ve sonuç lojistik regresyon için gördüğünüz ile oldukça benzer olacaktır.

Önceki videoda yaptığımız işlem sadece girilen bir "x" inputu için geçerliydi. Şimdi ise örnek olarak bir eğitim seti veya test seti için bu işlemleri nasıl yaparız, nasıl bir vektör haline getiririz onu göreceğiz.



Her veri için ayrı ayrı işlem yapıyor.  $a[2],[i]$  ifadesinde "2" katmanı, i ise örneğin sayısını ifade eder.

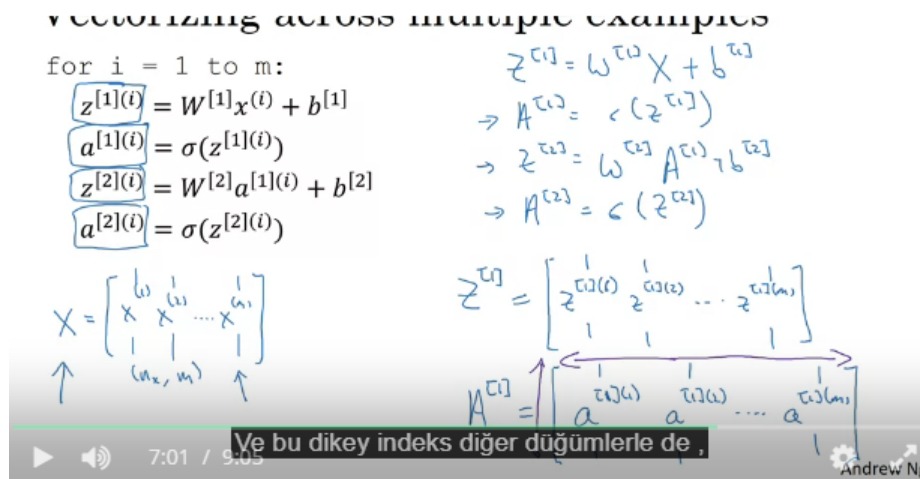
Tüm i değerleri için hesaplayacak olursak  $1 = 1$  için m yapmak gerekir ve ardından 4 denklemi de uygulamak gerekir.

for  $i = 1$  to  $m$ ,  
 $z^{(1)(i)} = W^{(1)} x^{(i)} + b^{(1)}$   
 $a^{(1)(i)} = \sigma(z^{(1)(i)})$   
 $z^{(2)(i)} = W^{(2)} a^{(1)(i)} + b^{(2)}$   
 $a^{(2)(i)} = \sigma(z^{(2)(i)})$

**denklem (i) üzeri ekleyerek**

Andrew Ng

Bu kod bloğunu düzenlemeye geçelim.



Yatayda baktığımızda her veri için bir sütun sıralaması mevcuttur. Dikey baktığımızda ise, diğer düğümlerle de sinir ağına uyum sağlar.

← Examples dikeyde ise farklı hidden layerlara denk gelir.(diğer sinir düğümleri.)

### Explanation for Vectorized Implementation

Bir önceki videoda yatay ekseninde x verilerinin sıralanışını ve işleyişini gördük. Bunun neden doğru bir method olduğunu açıklayalım.

İlk alıştırmaya örneğinin yayılım hesabını ele alalım. B değerini gözardı edip işlemlerimize devam edelim.

**Justification for vectorized implementation**

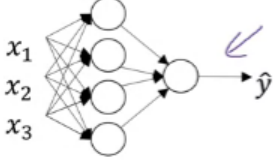
$$z^{(1)} = w^{(1)} x + b^{(1)}, \quad z^{(2)} = w^{(2)} x + b^{(2)}, \quad z^{(3)} = w^{(3)} x + b^{(3)}$$
$$w^{(1)} = \begin{bmatrix} w_{11}^{(1)} \\ w_{21}^{(1)} \\ w_{31}^{(1)} \end{bmatrix}, \quad w^{(2)} x = \begin{bmatrix} w_{11}^{(2)} x \\ w_{21}^{(2)} x \\ w_{31}^{(2)} x \end{bmatrix}, \quad w^{(3)} x = \begin{bmatrix} w_{11}^{(3)} x \\ w_{21}^{(3)} x \\ w_{31}^{(3)} x \end{bmatrix}$$

Her x değeri için bize farklı bir  $Wx$  matrisi verecektir. Şimdi tüm eğitim setini gösteren "X" oluşturalım ve tüm eğitim örneklerini yığma yoluyla birleştirelim.

Bu örnek sadece veri seti 3 adet olan bir eğitim seti için geçerli. Fazla olsaydı elbette yatayda x'm e kadar yığılacaktı.

$$w^{(1)} \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & x^{(3)} \\ | & | & | \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} | & | & | \\ z^{(1)(1)} & z^{(1)(2)} & z^{(1)(3)} \\ | & | & | \end{bmatrix} = z^{(1)}$$

Unutulmaması gereken husus oluşan  $Z =$  değerinde matrisin dışında  $+b$  matrisi olmasıdır. Bu kısımda sadece  $Z[1]$  değerinin hesaplanması görüldü. Farklı katmanlar arasında pek bir fark yok aynı işlemleri yapar. Sadece indeks sayısı artıyor.


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

for  $i = 1$  to  $m$

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

Şimdiye kadar sigmoid fonksiyonunu kullandık. Bakalım başka fonksiyonlar bize seçenek olabilir mi? Devamında başka Activation fonksiyonlarını göreceğiz.

### Activation Functions

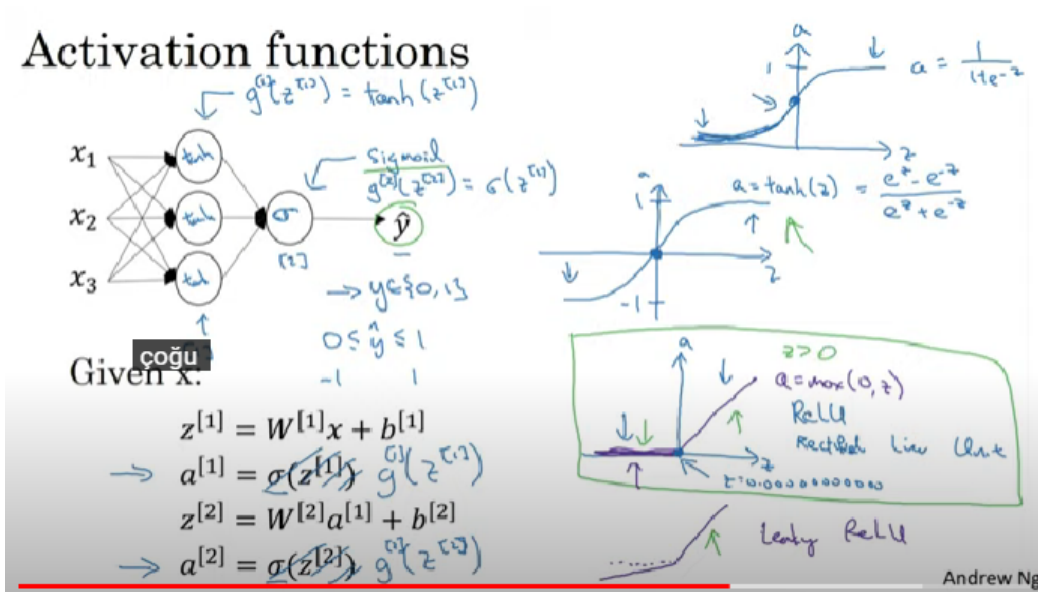
Bu zamana kadar sadece sigmoid fonksiyonunu gördük, başka neler kullanabiliriz?

Sigmoid fonksiyonu 1 ve 0 arasında gider. Alternatif olarak kullanabileceğimiz bir diğer fonksiyon "tanh" fonksiyonu(hiperbolik tanjant fonksiyonu) ise +1 ile -1 arasında gider. Sigmoid fonksiyonunun kaydırılmış halidir ek olarak (0,0) noktasından geçer. TanH fonksiyonu, sigmoid fonksiyonundan daha çok kullanılır ve daha kullanışlıdır. Fakat output katmanı istisnadır. Çünkü y değerinin 1 ve 0 değerleri arasında olması gerektiğini biliyoruz.

Burdan çıkarabileceğimiz bir diğer sonuç ise aktivasyon fonksiyonları her katman için farklı olabilir.

Bu iki fonksiyonun ortak bir dezavantajı vardır. Z'nin çok büyük ya da çok küçük olmasıdır. O zaman bu fonksiyonun gradyanı, türevi ya da eğimi çok küçük olur. Bu durumda değerlerin eğim inişini yavaşlatabilir.

Bu durumda imdadımıza RELU fonksiyon geliyor. Fakat sıfıra gitmek öğrenmeyi yavaşlatır. Bu durumda sigmoid ve tanH ön plana çıkıyor.



Hızlıca özet geçmek gerekirse;

Her aktivasyon fonksiyonlarının kendisine göre artıları ve eksileri vardır.

Çıkış(y)(output) katmanı haricinde sigmoid fonksiyonunun kullanılması önerilmez. 1 ve 0 arasında ikili değer döndüğünden sigmoid fonksiyonu önerilir.

TANH fonksiyonu sigmoidden daha üstündür ve ardından RELU gelir ve leaky RELU vardır. İki RELU fonksiyonu arasında değer aralıkları farkı vardır.

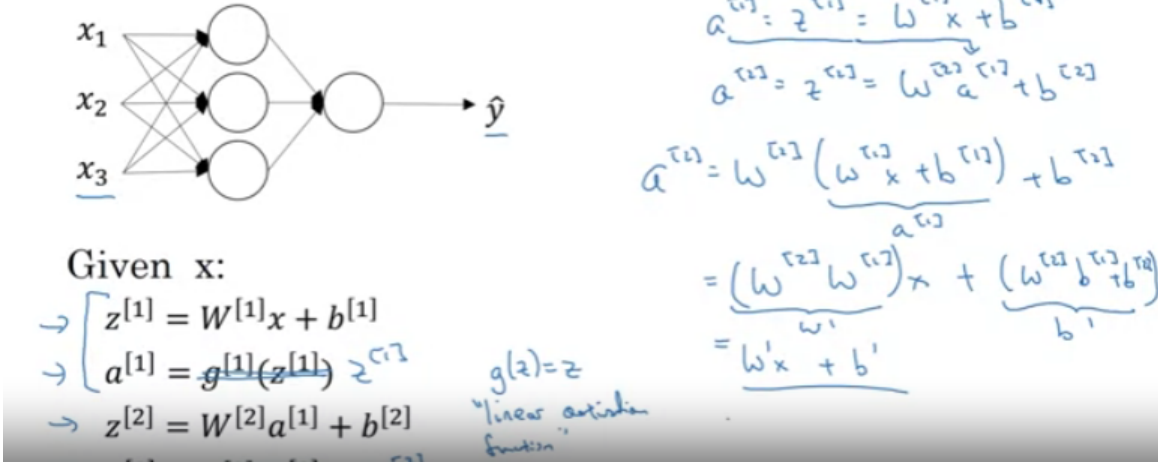
### Why do you need Non-Linear Activation Functions?

Bir sinir ağı neden lineer olmayan bir aktivasyon fonksiyonuna ihtiyaç duyar?

İlginç olarak nitelendirdiğimiz fonksiyonları hesaplayabilmesi için lineer olmayan bir aktivasyon fonksiyonuna ihtiyaç duyabiliriz.

"g(z) = z" fonksiyonunu örnek gösterecek olursak bu lineer bir aktivasyon fonksiyonudur ve ya da diğer adıyla identity activation(kimlik aktivasyon) fonksiyonu diyebiliriz. Çünkü girdi neyse çıktı o olur.

# Activation function



g'yi ortadan kaldırdığımızda elimize  $a = z$  gelir, denklemde yerinde koyduğumuzda ise elimize  $a = z = wx + b$  fonksiyonu gelir.

2 katman için bu işlemi yaptıktan sonra birbirine dair bir ilişki fonksiyonu çıkarmak istediğimizde sadeleştirmelerin aldırda  $w'x + b'$  fonksiyonunu elde ederiz.

Öyle görünüyor ki eğer lineer etkinleştirme fonksiyonunu kullanırsanız ya da alternatif olarak etkilenleştirme fonksiyonunuz yoksa sinir ağınızın kaç katmanının bulunduğu bir önemi yoktur, sinir ağınızın bütün yaptığı sadece lineer etkinleştirme fonksiyonunu hesaplamaktır.

Daha önce bahsedildiği gibi burada bir lineer aktivasyon fonksiyonu ve şurada bir sigmoid fonksiyonu varsa, o zaman bu model gizli katmanı olmayan standart lojistik regresyondan farkı yoktur.

Alınması gereken ders, lineer olan gizli bir katman pek işe yaramaz, çünkü iki lineer fonksiyonun birleşimi kendisine yani bir lineer fonksiyona eşittir. Yan buraya bir lineersizlik almazsanız ağda daha derine gitmenize rağmen ilginç fonksiyonları hesaplamıyor oluruz. Lineer aktivasyon fonksiyonunu kullanabileceğimiz sadece bir yer olabilir,  $g(z) = z$  ve bu sadece regresyon probleminde makine öğrenmesi uygulamak olur.

Örnek verecek olursak  $y \in \mathbb{R}$  (  $y$  reel sayıdır ) ve konut fiyatlarını tahmin ediyorsanız,  $y$  sıfır değildir ama bir reel sayıdır ve sıfır TL'den bir evin fiyatı nereye kadar çıkıyorsa o kadardır.

Eğer  $y$  fiyat olan reel değerleri alırsa o zaman burada lineer aktivasyon fonksiyonunu kullanmak normal olabilir, böylece çıktı olan  $y'$  da eksi sonsuzla artı sonsuz arasında bir reel sayı olur. **HIDDEN LAYER BULUNAN BİR ÖĞRENME ALGORİTMASINDA LİNEER ETKİNLEŞTİRME FONKSİYONU KULLANILMAMALIDIR.**

Hidden layerlar RELU, TANH veya Leaky RELU aktivasyon fonksiyonlarını kullanmalıdır. Yani lineer etkinleştirme fonksiyonunu kullanabileceğiniz yer genellikle output(çıktı) katmanıdır.

## Derivatives of Activation Functions

Yapay sinir ağınız için geri yayılımı uygularken, aktivasyon fonksiyonlarının türev eğimlerini hesaplayabiliyor olmanız gerekir. Bu sebeple aktivasyon fonksiyonlarımızda eğimlerin nasıl hesaplanacağını öğrenmemiz gerekiyor.

İlk başta bize en tanıdık olan Sigmoid aktivasyon fonksiyonu ile başlayalım.

$g(z)$  sigmoid fonksiyonunun eğimi  $(d/dz) * g(z)$  olur. Sigmoid fonksiyonunun türevi kendisine eşittir.

z fonksiyonuna örnek olarak 10 diyelim ve  $g(z)$  değeri  $\sim 1$ 'e yakın olur ve yandaki formüle baktığımızda  $d/dz * g(z) \sim 1 \cdot (1-1) = 0$  olur.

Bu sonuç doğrudur, çünkü  $z$  çok büyük bir değer ise eğimi 0'a yakındır.

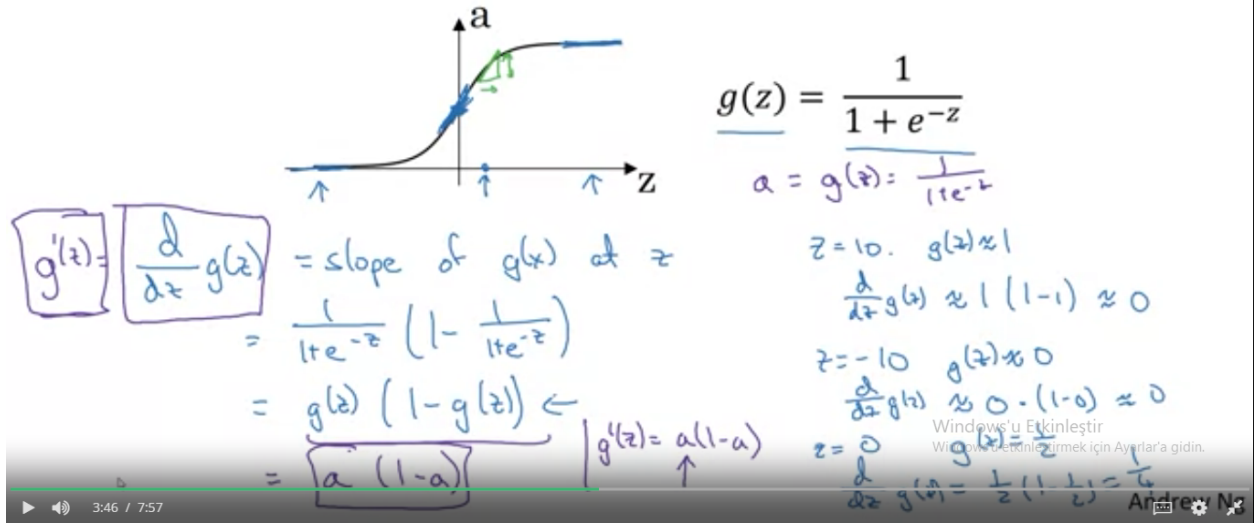
Tersi bir duruma baktığımızda  $z = -10$  dediğimizde yani çıkış yolu var ise  $g(z)$  0'a yakındır. Soldaki formüle baktığımızda  $d/dz * g(z)$ 'ye yakın olacak ve buda  $0 \cdot (1-0)$  olacaktır.

Son olarak  $z = 0$  ise  $g(z) = 1/2$  olur. Türev için baktığımızda  $1/2(1-1/2) = 1/4$  olur.

Her zaman tanımlı olan formül  $d/dz * g(z) = g(z)(1-g(z))$  yazmak yerine  $g'(z)$  ifadesini kullanırız.

Bazen sinir ağlarında  $a=g(z)$  ifadesini görebiliriz. Sonucunda eğim formülü şuna benzer;  $a(1-a) = a' = g'(z)$ .

## Sigmoid activation function



Şimdi de tanH aktivasyon fonksiyonu için türev işlemine bakalım.

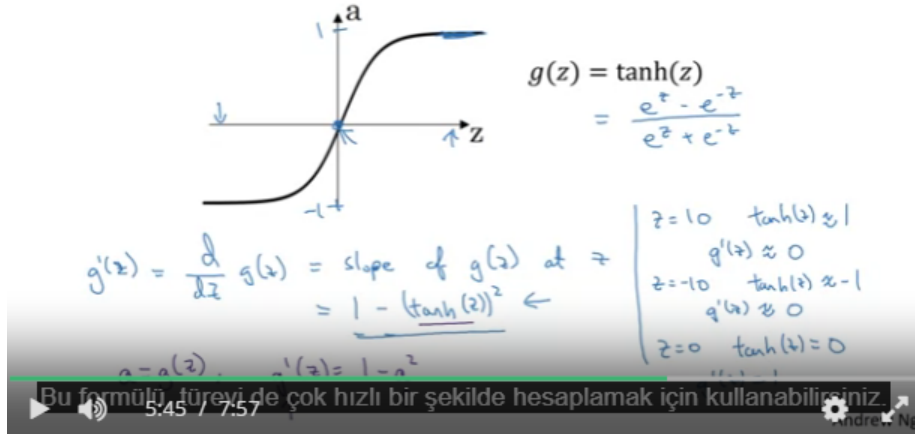
Tanımlı olan tanH fonksiyonunun tanımı  $g(z) = \tanh(z)$ 'dir. Türev işlemi uygulandığında ise;  $g'(z) = d/dz * g(z) = 1 - (\tanh(z))^2$  olur. Örnekler üstünden gidecek olursak;  $z = 10$  için konuştuğumuzda  $\tanh(z) \sim 1$  olur. Bu durumda  $g'(z)$  ise  $\sim 0$ . Sonuç olarak  $z$  çok büyükse eğim sıfıra yakındır. Aksine baktığımızda  $z$  için küçük bir değer,  $z=0$  dediğimizde  $\tanh(z)$  değeri  $\sim 0$  olur ve türevde sıfıra yakın olur.

Son olarak eğer  $z=0$  ise,  $\tanh(z)$ 'de  $\sim 0$  olur. Formüle koyduğumuzda eğim yaklaşık değil direkt olarak gerçekten 1'e eşittir.  $z$ 'nin sıfır olduğu durum tanH fonksiyonu için eğimdir.

Bir önceki aktivasyon fonksiyonunda gördüklerimiz  $a$  tanımı, burada ki fonksiyon içinde geçerlidir.  $a = g(z)$  olduğu durumda,  $g'(z) = 1-a^2$  olur. Yani bir kez daha, eğer  $a$  değerini zaten hesapladıysanız bu türevi de çok hızlı bir şekilde hesaplamak için kullanabilirsiniz.



## Tanh activation function

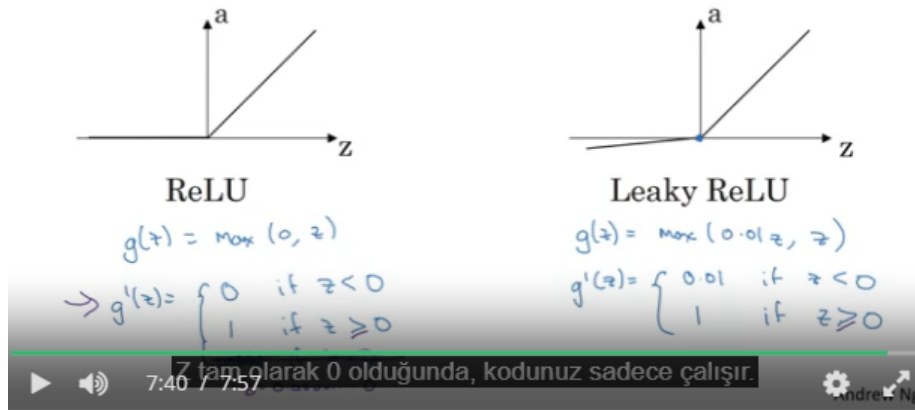


Son olarak ReLU ve Leaky ReLU aktivasyon fonksiyonlarımızın türevlerini nasıl hesaplıyoruz ona bakalım.

ReLU fonksiyonunun tanımlı olduğu değer aralıkları  $g(z) = \max(0, z)$ . Bu nedenle  $g'(z)$   $z$  sıfırdan küçük olduğunda 0,  $z$  sıfırdan büyük olduğunda 1 değerini alır. Eklemek gerekirse  $z$  aslında 0'a eşit olursa tanımsız bir değer alır.

Son olarak bir sinir ağına Leaky ReLU fonksiyonunu kullanıyorsanız,  $g(z) = \max(0.01z, z)$  olacaktır ve böylece  $g'(z)$  için  $z$  sıfırdan küçükse 0.01,  $z$  sıfırdan büyükse 1 olarak tanımlanır.

## ReLU and Leaky ReLU



### Gradient Descent for Neural Networks

Bu videoda bir katmana gradient descent kuralını nasıl uygulayacağımızı öğreneceğiz. Basit bir örnek üzerinden başlamak gerekirse tek katmanlı ve parametreleri  $w[1], b[1], w[2], b[2]$  olan bir sinir ağı olduğunu varsayalım.

İlk olarak cost function için ikili parametre yapacağımızı varsayalım.

Ardından algoritmalarımızı eşitlemek için gradient descent uygulamamız gerekiyor. Başlangıç değerlerini sıfıra atamaktansa rastgele değerler atamak her zaman önemlidir. Daha sonra neden böyle olduğunu göreceğiz. Cost function'a göre  $w$  ve  $b$  değerlerinin türevini alırız. Her bir katman için ayrı ayrı.

# Gradient descent for neural networks

Parameters:  $W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}$   $n_x = n^{(1)}, n^{(2)}, \underline{n^{(2)} = 1}$

Cost function:  $J(W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}) = \frac{1}{n} \sum_{i=1}^n \ell(\hat{y}_i, y_i)$

Gradient descent:

→ Repeat {  
 → Compute gradients  $(\hat{y}^{(i)}, i=1, \dots, n)$   
 $\frac{\partial J}{\partial W^{(1)}}, \frac{\partial J}{\partial b^{(1)}}, \dots$   
 $W^{(1)} := W^{(1)} - \alpha \frac{\partial J}{\partial W^{(1)}}$   
 $b^{(1)} := b^{(1)} - \alpha \frac{\partial J}{\partial b^{(1)}}$

Windows'u Etkinleştir  
 Windows'u etkinleştirmek için Ayarlar'a gidin.

İleri yayılım fonksiyonunu özet geçmek gerekirse,

Forward propagation:  
 $z^{(1)} = W^{(1)}x + b^{(1)}$   
 $A^{(1)} = g(z^{(1)})$   
 $z^{(2)} = W^{(2)}A^{(1)} + b^{(2)}$   
 $A^{(2)} = g(z^{(2)}) = \sigma(z^{(2)})$

Şimdi sırada sinirsel ağıımız için türevlerini hesaplayalım. ( Geri yayılım hesabı yapmak için türev almamız gerektiğini daha önce söylemiştik ).

Aşağıdaki formül üçlüsü sonuç olarak db(2)'yi (n,1) boyutunda bir vektör olarak çıkarmasını sağlıyor.

# Formulas for computing derivatives

Forward propagation:

$$z^{[1]} = w^{[1]}x + b^{[1]}$$

$$a^{[1]} = g^{[1]}(z^{[1]}) \leftarrow$$

$$z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = g^{[2]}(z^{[2]}) = \sigma(z^{[2]})$$

Back propagation:

$$dz^{[2]} = a^{[2]} - y$$

$$dw^{[2]} = \frac{1}{n} dz^{[2]} a^{[1]T}$$

$$db^{[2]} = \frac{1}{n} \text{np.sum}(dz^{[2]}, \text{axis}=1, \text{keepdims}=\text{True})$$

$$dz^{[1]} = \underbrace{w^{[2]T}}_{(n^{[2]}, m)} dz^{[2]} \times \underbrace{g^{[1]'}(z^{[1]})}_{\text{element-wise product}} (n^{[1]}, m)$$

$$dw^{[1]} = \frac{1}{n} dz^{[1]} x^T$$

$$db^{[1]} = \frac{1}{n} \text{np.sum}(dz^{[1]}, \text{axis}=1, \text{keepdims}=\text{True})$$

$$y = [y^{(1)} y^{(2)} \dots y^{(n)}]$$

$$(n, 1) \leftarrow$$

$$(n^{[2]}, 1) \leftarrow$$

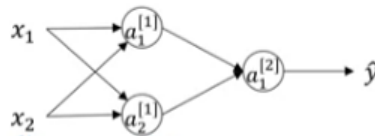
Bu 6 formül denkleminde ise türevle geri yayılımı elde ediyoruz.

## Random Initialization

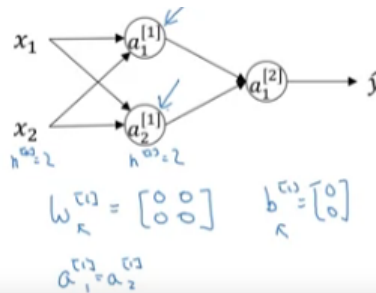
Lojistik regresyon modelinde ağırlıkların sıfır ayarlanmasında bir sakınca yoktur, fakat derin öğrenme algoritması altında katmanlarla çalışıyorsak bu durumda ağırlıklar rastgele değerler seçilmelidir.

Örnek üstünden gidecek olursak 3 katmanlı bir sinir ağı üzerinde sadece sıfırlarla çalışacağımızda nasıl bir sonuç alacağımıza göz atalım.

What happens if you initialize weights to zero?

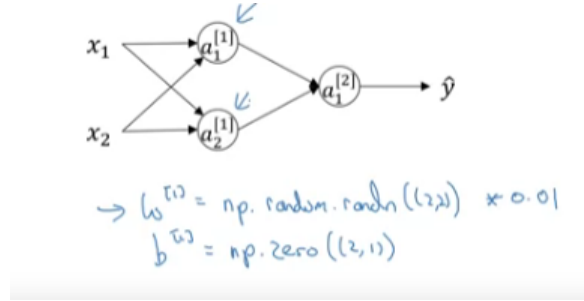


$W = (2,2).$ zeros ve  $b(2,1).$ zeros olarak tanımladıktan sonra 1. hidden layerındaki aktivasyon değerlerinin ikisinde aynı gelir çünkü aynı fonksiyon üzerinden hesaplamalar yapmaktadırlar.



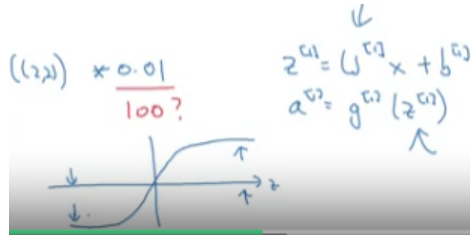
Aynı şekilde geri yayılım durumuna baktığımızda da  $dz1$  ve  $dz2$  birbirine eşit olacaktır. Bu iki gizli katmanda birbiri ile aynı şekilde tanımlanacaktır. Her anlamda simetrik bir fonksiyon olur ve yerine getirilmesi gereken her farklı katman birbiri ile aynı olur. Bu durumda her iki katmanda çıkış katmanına aynı etkiyi verir. Bu sebeple  $W$  başlangıç değeri sıfır olan bir sinir ağı algoritmasında çoklu gizli katman gereksiz olur çünkü aynı değerleri döndürürler.(Lojistik regresyonda kullanılabilir.)

Bu durum tek çözümü  $W$  ağırlık değerlerini rastgele başlatmaktır.



Yukarıda tanımlanan  $W$ , random (2,2) matrisinde Gauss sayıları üretir ve onu 0.01 sayısı ile çarpar. "b" ile yaşanması muhtemel simetri sorununa çözüm üretir.(Symmetry breaking problem)

Neden 0.01? sorusuna yanıt verecek olursak genellikle ağırlıkların rastgele küçük değerlerle başlamasını tercih ettiğimiz ortaya çıkıyor.



Eğer ki çok küçük değerlerle başlamazsak  $W$  değeri büyük olacağından  $Z$ 'de büyük olur ve aktivasyon fonksiyonu büyük tanımlanır. Haliyle eğimi çok küçük olduğu için okla belirtilen kısımlarda eğim çıkma olasılığı daha yüksek olur. Bu durum gradient descent çok daha yavaş olacak demektir. Buda öğrenmeyi bir hayli yavaşlatır.