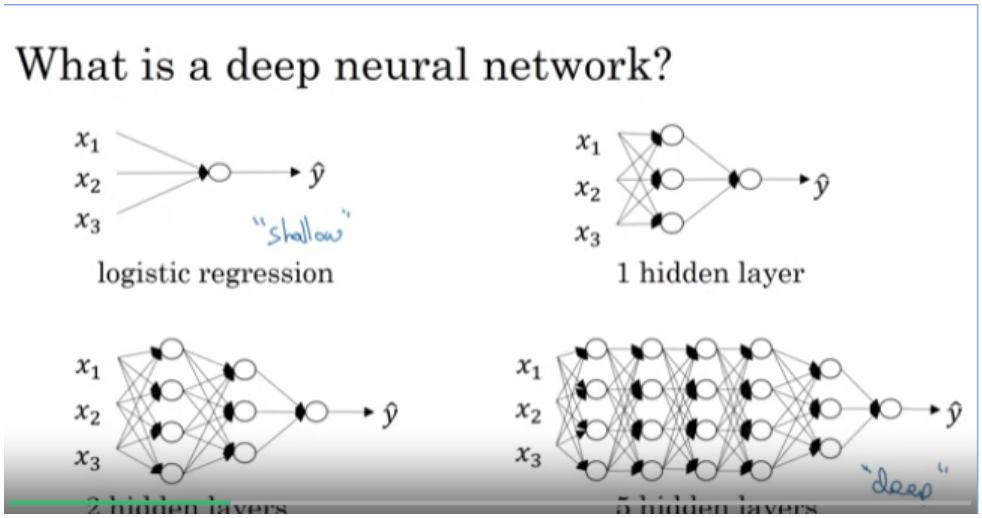
Week-4

Deep L-layer Neural Network

Bu hafta yapacağımız şey, kendi derin sinir ağınızı uygulamak için, bu konuları birbirleriyle birleştirmek olacak.

Lojistik regresyon modeli, derin öğrenmeye örnek değildir. Derin olmayan(sığ,shallow) modeldir.



Aşağıdaki görselde 3 gizli olan, 4 katmanlı bir sinir ağı bulunmaktadır ve 5,5,3 olmak üzere bir tane de üst birim bulunmaktadır.

Ağdaki katman sayısını göstermek için “L” ifadesini kullanırız bu durumda L = 4 katmanların sayısını verir.

n^[l], ifadesi katmanda düğümlerin ve birimlerin sayısını ifade eder.

Örneğin, n^[1] = 5 olacaktır. Çünkü orada 5 sinir ağı mevcuttur.

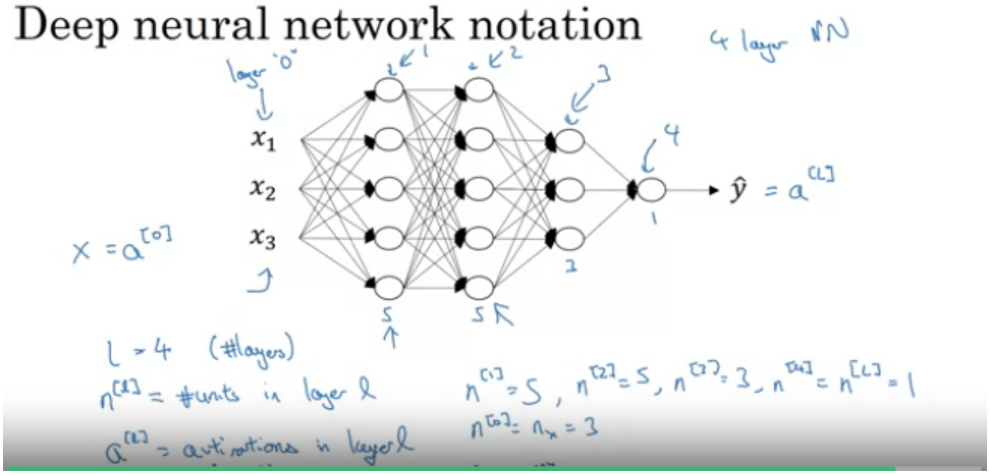
Input katmanı için yazacak olursak n^[0] = nx = 3 olacaktır.

Her bir katmandaki aktivasyonları göstermek için a^[l] ifadesini kullanacağız.

Aktivasyon fonksiyonu için yayılım ifadesi yazdığımızda ise, a^[l] = g^[l](z^[l])

Input katmanı için aktivasyon tanımı ise x = a^[0] olur.

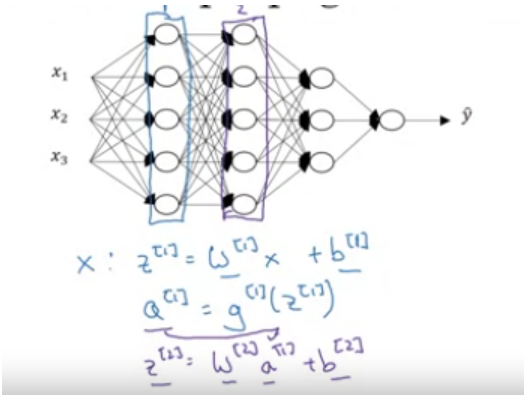
Output(yşapka) için ise = a^[L] olur.



Forward Propagation in a Deep Network

İleri yayılım nasıl hesaplanır ve verilen tek X girdisine göre nasıl hesaplanır onun üzerinde konuşalım. Vektör halindeki input üzerinde ileri yayılımın nereye uygulanacağını, tüm eğitim seti aynı zamanda verilen tek X girdisinde inceleyeğiz. Daha sonra aktivasyon değerlerini nasıl hesaplayacağımızı göreceğiz.

İlk katmanda aktivasyon hesaplarken, z^[1] = w^[1]*x + b^[1] w1 ve b1 parametreleri birinci katmandaki aktivasyonları etkiler. a^[1] = g(z1) eşit olur ve hedef fonksiyon g, bulunduğun katmana bağlıdır. Birinci katmanda hesaplanan aktivasyon, diğer katmanda W2 değerinin bulunmasına yardımcı olur. Z2'yi hesaplamak için W[2]*A[1] + b[2] olur.

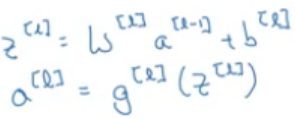


A[2] = g(Z2) olur.

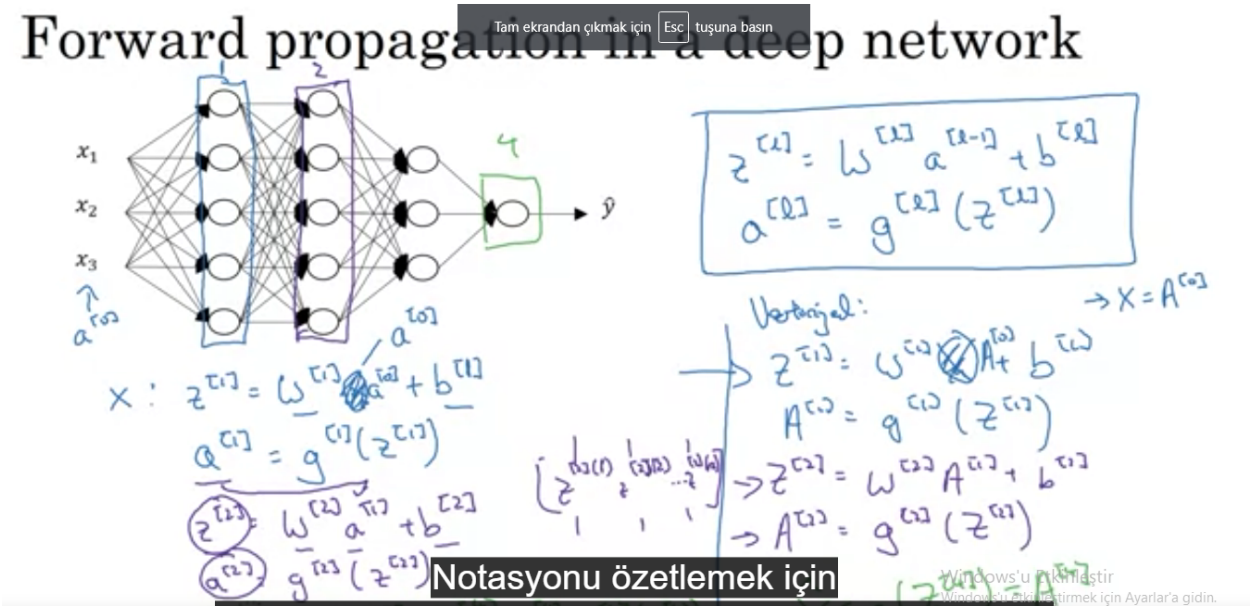
Bu durum diğer katmanlar içinde output katmanına gelene kadar aynı şekilde devam edilir.

Katman 4'ün hesabını yaparken, Z[4] = W[4]*A[3] + b[4] ve A[4] = g(Z4) olur.

Temel olarak katman hesaplamalarını yaparken çıkarabileceğimiz kural:



Tek öğrenme için tüm bu işlemleri yaptık. Bütün veri seti için bu işlemleri vektör olarak nasıl yapmalıyız ona bakalım. Denklemler öncekiler gibi görünecektir.



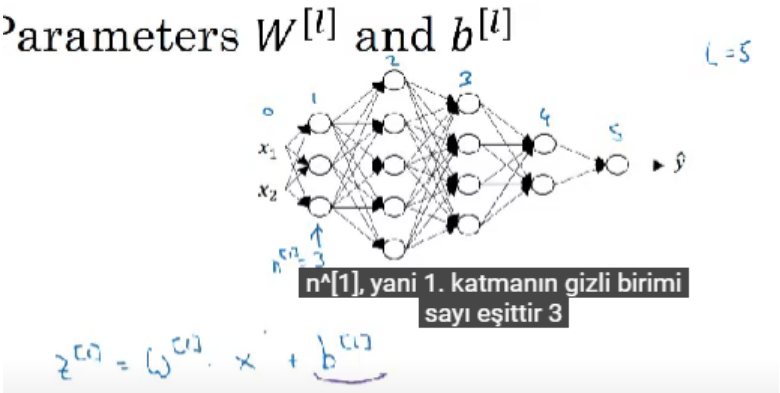
Bu durum m. eğitim örneğine kadar devam eder. Z,X ve W değerleri sütun vektörü olarak birikir. Bu işlemler sonunda y^ elde edilir ve bu da G(Z4) olur ve bu da A[4]'e eşittir.

Her katman için ayrı ayrı hesaplama yapmak zor olacağı için bir for döngüsü gereklidir, for l=1-4

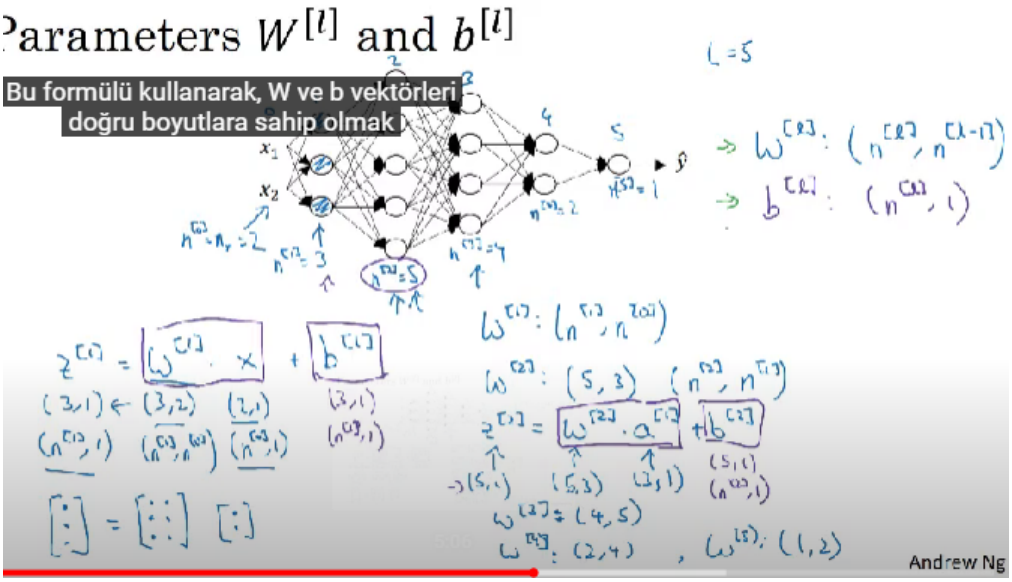
Getting your Matrix Dimensions Right

Derin sinir ağları oluştururken kodun doğruluğunu kontrol etmenin bir kaç yolu vardır. Bu hata ayıklama araçlarını görelim.

L = 5, 1. katman için yazacak olursak Z[1] = W[1]*x // bias değerini görmezden gelelim. n[1] = 3



Matris değerlerine katman gözünden bakıp yazacak olursak:



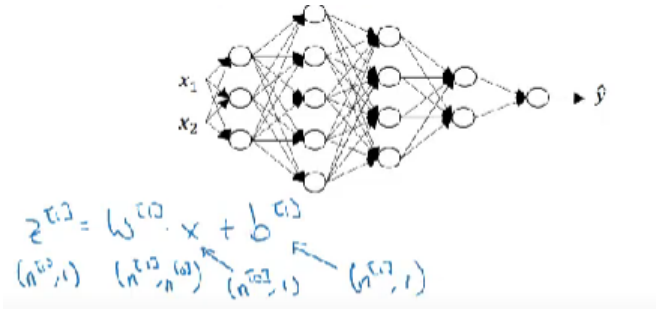
Eğer geri yayılım uygularsanız, dW[l] : (n[l], n[l-1] yani W[l] boytuna eşittir.

db[l] = (n[l],1)

Ayrıyetten kontrol için aktivasyon fonksiyonuna da bakabiliriz.

z[l] = g[l](a[l]) eşitliğinden z ve aynı boyuta sahip olmalıdır.

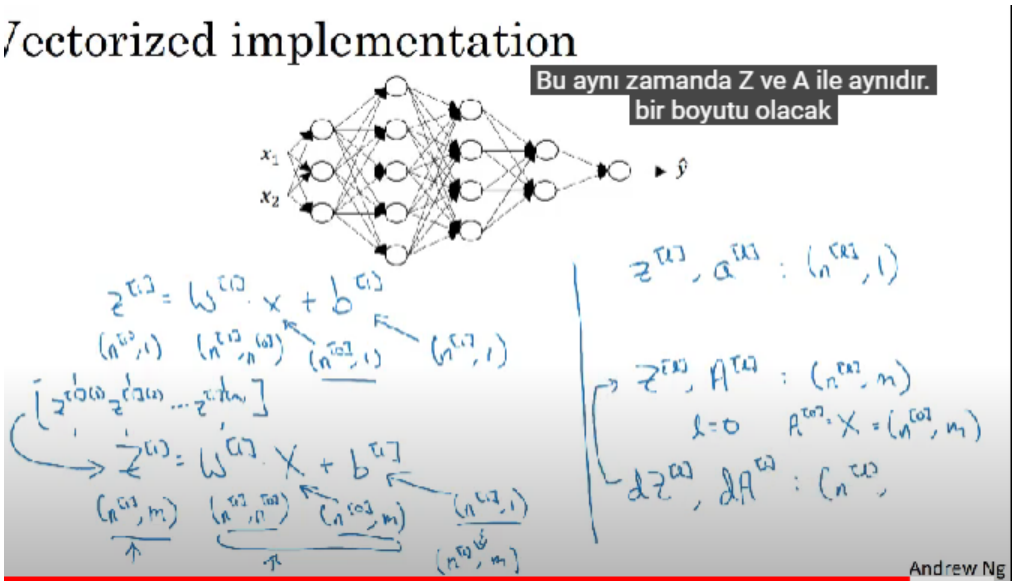
Şimdi aynı anda birden fazla veriyi görüntülemeyi, vektörize bir uygulama ile nasıl görüneceğini görelim. Tabii ki vektörleştirilmiş bir uygulamada bile W,b,dW,db aynı boyutlara sahiptir. Fakat vektörize bir uygulamada z,a,x değerleri farklı olabilir.



Normal bir Z[1] yazılımı için gösterim bu şekildedir. Fakat vektörize bir uygulamada durum biraz değişiyor.

Z[1] = W[1].X + b[1] olarak baktığımızda Z[1] bireysel verilerdendir. Yani z[1][1] .. z[1]..[m] şeklinde ilerler. (m, eğitim kümesinin boyutu)

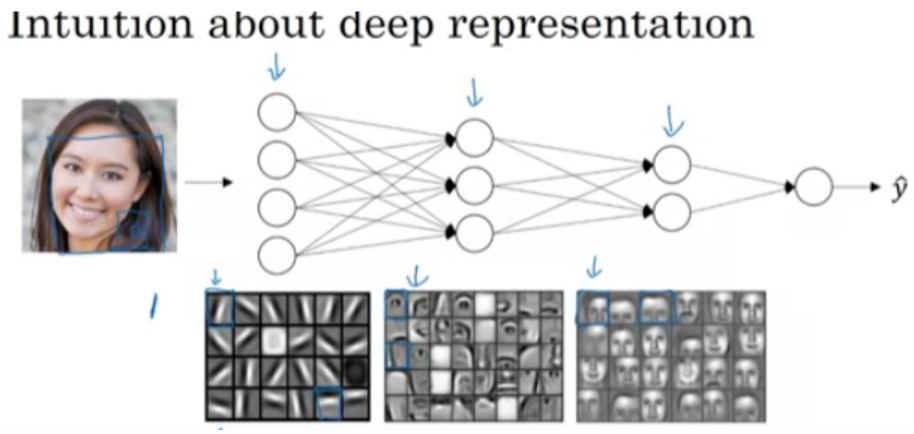
Yani bu durumda Z[1]'in matris shape'i [n[1],m] olur. W1'in boyutu aynı kalır. X'in shape'de eğitim kümesine göre değişiklik gösterip (n[0],m) olur. b[1] = (n[1],m) with python for b.



Why Deep Representations?

Derin ağların neden iyi çalışabileceğiyle ilgili biraz fikir edinelim.

İlk olarak derin ağlar neler hesaplıyor. Varsayalım bir yüz tanıma ya da algılama sistemi oluşturuyorsunuz. Deep Learning bu sistemde ne yapıyor olacak? Muhtemelen sisteme bir yüz resmi gireceksiniz sonrada sinir ağının ilk katmanı gelecek, bu katmanı öznitelik bulucu ya da kenar bulucu gibi düşünebilirsiniz. Şimdi resimdeki pikselleri gruplayarak resimdeki kenarların yerleri hakkında biraz düşünelim. Ardından ikinci katmanda kenarları ve grup kenarlarını birlikte algılayarak yüzlerin parçalarını oluşturabilir. Örneğin bir nöronlardan biri gözü bulmaya çalışırken farklı bir nöron bunun bir parçasını bulmaya çalışabilir. Böylece bir çok kenarı bir araya getirerek yüzün farklı parçalarını algılamaya başlayabilir. Son olarak üçüncü katmanda göz, burun, kulak veya çene gibi yüzün farklı parçalarını bir araya getirerek farklı tipteki yüzleri algılamaya yada tespit etmeye çalışacaktır. Genel olarak baktığımızda sinir ağının ilk katmanının kenar gibi basit işlevleri tespit edici olarak düşünebilirsiniz ve onları sinir ağının daha sonraki katmanlarında bir araya getirerek daha karmaşık işlevleri öğrenebilir. Bu tür basitten karmaşığa hiyeraşisel gösterim veya yapısal gösterim görüntü işlemeden ziyade diğer türden verilerde de geçerlidir.



Derin sinir ağları çoklu gizli katmanlarla ve önceki katmanlardaki düşük seviye temel özellikleri öğrenerek çalışır ve sonraki daha derin katmanlarla bir araya getirerek basit şeylerin tespiti ile daha kompleks şeylerin tespitini

gerçekleştirebilir

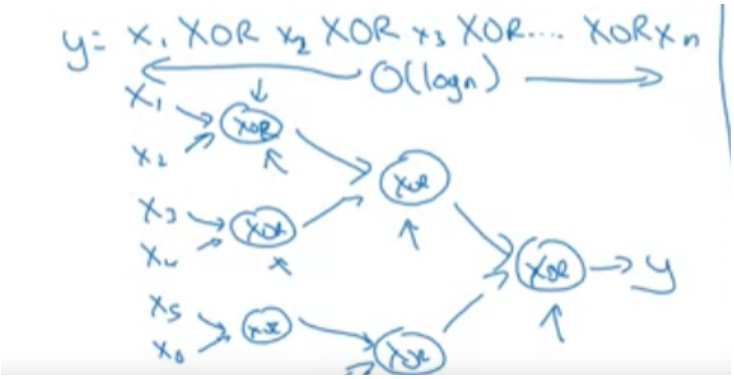
Bazı insanlar derin sinir ağları ile insan beyni arasında benzetme yapmak ister ve bizim veya nörologların söylediğine göre insan beyni de temel şeyleri tespit etmekle başlar mesela kenarlar gibi gördüğünüz şeyleri inşa eder ve daha karmaşık şeyleri tespti eder.

Videonun diğer parçası ise derin ağların iyi çalışıyor gözükmesinin sebebi şu şekilde.

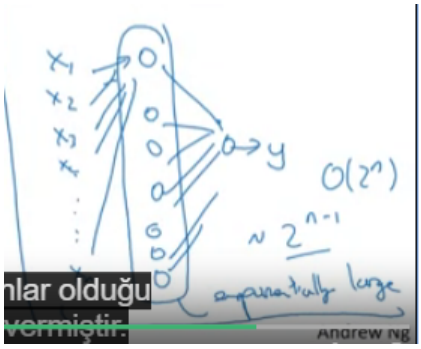
İşlevleri nispeten küçük fakat derin bir yapay sinir ağı ile hesaplanır, küçük olarak demek istediğimiz gizli katman sayısının nispeten küçük olmasıdır. Ama eğer aynı işlemi daha sığ bir ağ ile hesapladığınızda, yeterli olmayan gizli katman sayısı, sonrasında üstel olarak daha fazla gizli katmana ihtiyaç duyabilirsiniz. Örnek verecek olursak:

Diyelim ki tüm giriş özelliklerinin özel OR durumunu hesaplıyoruz.

XOR hesabını yapabilmek için tek katmandan ziyade bir devre ağacı kurmak daha mantıklıdır. XOR değerini hesaplamak için ağlar logN sırasında olur. Bu şekilde bir XOR ağacımız olacak.



Ama şimdi, eğer çoklu gizli sinir ağları kullanma imkanınız yoksa ve eğer bu fonksiyonu sadece bir gizli katman ile hesaplamak zorunda kalırsanız tüm inputlar gizli katmana doğru gidecek ve katmandaki nöronların hepsi outputa gidecektir. Sonra bu XOR fonksiyonunu hesaplamak için, bu gizli katmanın üstel olarak büyük olması gerekir, çünkü sonunda 2'den N'e kadar olan muhtemel biçimleri detaylıca yapılanıdrmanız gerekir. Sırayla 2'den N'e giriş bitlerinin muhtemel biçimleri XOR'un 1 veya 0 olması ile sonuçlanır. Teknik olarak bunu 2^N-1 gizli katman ile yapabiliriz.



Building Blocks of Deep Neural Networks

Derin bir ağ oluşturmak için bu bileşenleri nasıl bir araya getirebileceğinizi görelim.

Sadece bir katman üzerindeki hesaplamalara bakarak ilerleyelim.

Layer l = W[l], b[l]

Forward —> Input: a[l-1] , Output:a[l]

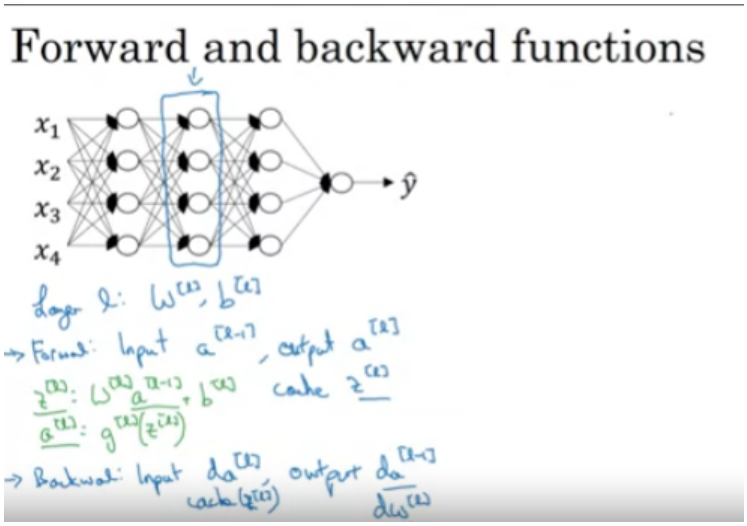
Z[l]: W[l]*a[l-1] + b[l]

a[l]: g[l](z[l])

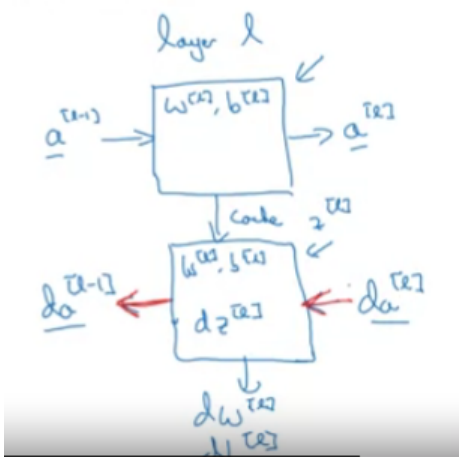
Z[l] değerini önbelleğe almak(cache) yapmak önemlidir çünkü z[l] değerini saklamak geriye yayılım için faydalı olacaktır.

Backward ——> Input da[l], output da[l-1]

Inputta cachelediğimiz Z[l] değeri kullanılabilir aynı zamanda output için eğimlerin çıktılarını da üretebiliriz.dw[l] (Öğrenme için eğim inişini uygulamak amacıyla.)



Şekil itibariyle özetlemek gerekirse

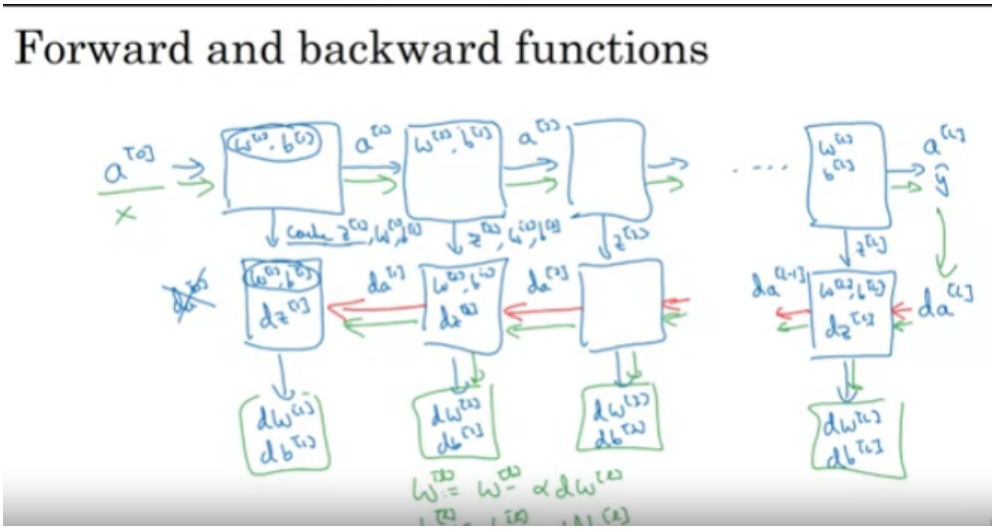


Örnek verecek olursak input a[0] özniteliği, diğer katmanın aktivasyonlarını hesaplamaya yardımcı olacaktır.(W1 ve b1 ile) Çıkan Z1 değerini saklayacağız(cache).

A[l] çıktısını üretene kadar bu durum böyle devam eder. Buda y^ya eşittir. Bu yol boyunca tüm Z değerlerini cache ettik ve sonuç olarak bu bir ileri yayılımdır.(Forward)

Sırada yapacağımız şey, geriye doğru gideceğimiz ve eğimleri hesaplayacağımız bir geri dönüş dizisi olacaktır.Yol boyunca da[l] ile inputlayıp da[l-1] çıktısı alarak ilerler ve her katmanda dw ve db parametrelerini üretir.

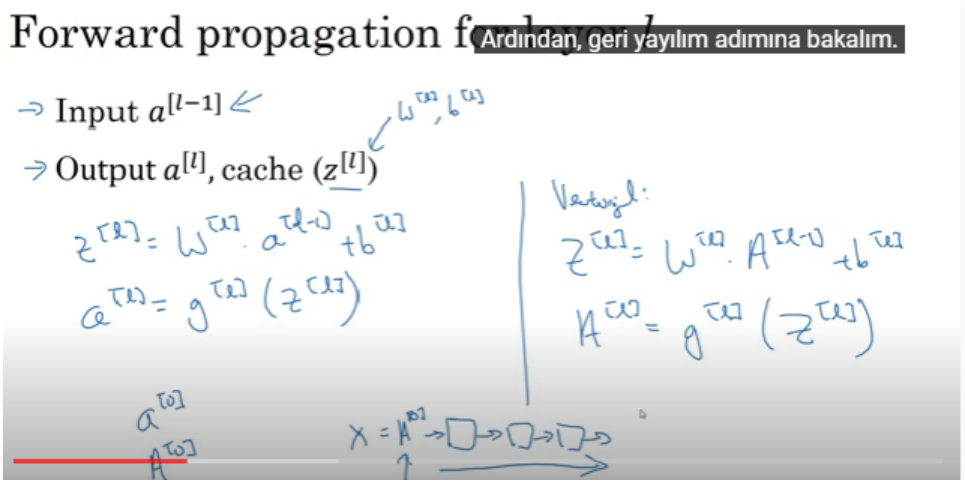
Ayrıca her bir katman için W değeri, W-(learningrate*dw) olarak güncellenir.Benzer durum b içinde geçerlidir.



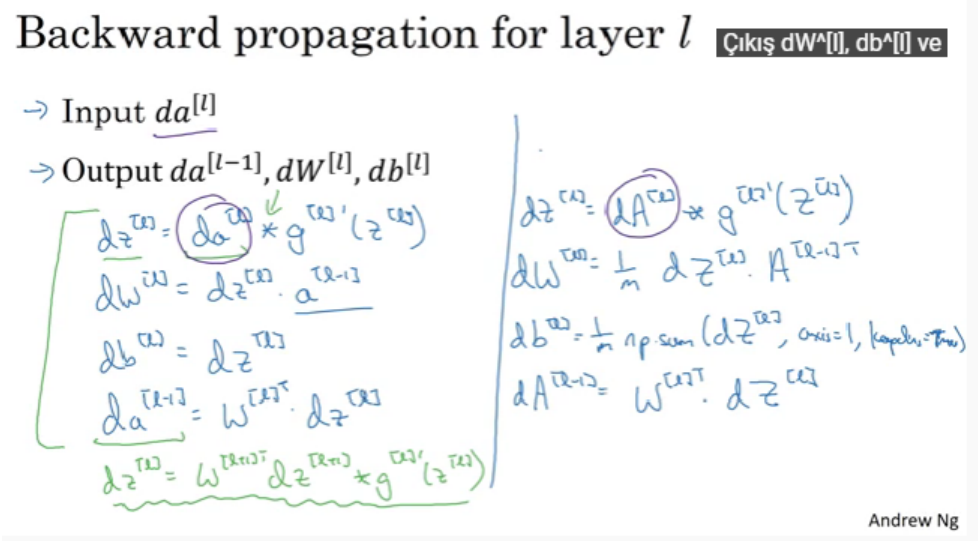
Forward and Backward Propagation

Önceki videoda her katman için uygulayabileceğimiz ileri ve geri yayımları gördük. Şimdi bu adımları nasıl uygulayabileceğimizi görelim.

İleri yayılımla başlayalım. İlk önce temel denklemlere göz atalım.



Şimdi de geri yayılım adımına bir göz atalım.



Özetlemek gerekirse X girişi alıyoruz ve ReLU aktivasyon fonksiyonunu uyguluyoruz birinci katmanda. Ikıncı katmanda başka bir ReLU aktivasyon fonksiyonu uyguluyoruz.Üçüncü katman bir sigmoid aktivasyon fonksiyonu kullanılıyor.Output katmanına yakın olduğu için yanı yşapka ikili sınıflandırma değeri döndüreceği için sigmoid kullanılıyor.L(y^,y)

Devamında işleme geri yayılım için bakalım.(kırmızı okla)

Her katmanda alınan dw ve db çıktıları elde edilir ve bu hesaplamalar yapılırken ileri yayılım yaparken elde ettiğimiz z1 z2 .. değerlerini cache(yedek) yapıyoruz. Son katman için elimize geçen L(y^,y) ifadesinin eşiti bir nebze matematiksel karşılığı -y/a + (1-y)/(1-a)'dır.

Sağ alt kısımda görülen dA[L] kısmı vektörleştirilmiş hali için yazılabilir.

Parameters vs Hyperparameters

Bu zamana kadar parametreler üzerinde çalıştık birazda hyperparameters bahsedelim.

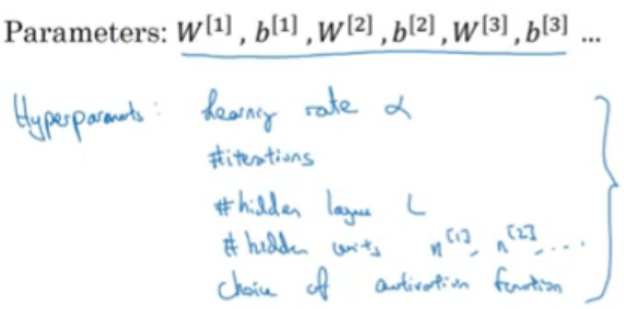
Öğrenme algoritmanız için özellikle öğrenmemiz gereken hyperparameters değerleri için örnek parametrelerimiz w1,b1,w2,b2,w3,b3 şeklinde tanımlanmıştır.

Hyperparameters diye tanımlanan ifade bu durumda learningrate(alfa) olur. Alfa değeri parametlerin nasıl değişeceğini veya belki iterasyon sayısını gradyan inişi için gerekli olan ve öğrenme algoritmasının sayıları vardır.

Hidden layers L, hidden units n[1],n[2]...

choise of activation fonction(ReLU,sigmoid)

Bunlar nihai parametrelerdir W ve B'yi kontol eder yani bunlar W ve B parametleri için hyperparameters'tır.



Derin öğrenme uygulamaları çok kırılgandır, genellikle örneklenebilecek bir fikrin olabilir, en iyi değer için öğrenme hızı mesela, Idea = alfa = 0.0.1 olabilir. Kodlamanın içinde nasıl çalıştığına bakıp ve sonra temel aldığı çıktıya göre, diyebilirsiniz ki bunu değiştirmek istiyorum ve artırmak istiyorum.Her zaman en iyi hyperparametreyi bulmaya çalışırız ve denemekten asla çekinmeyiz. Görmek için denemek zorundayım.