# Logistic Regression

İkili sınıflandırma (0 ve 1 çıkış) denilince akla ilk gelen lojistik regresyondur.

Ancak, derin öğrenme eğitiminde oradaki lojistik regresyonla ne yapmalı?

Cevap, lojistik regresyon aslında çok basit bir sinir ağıdır.

Bu arada sinir ağı ve derin öğrenme aynı şeydir. Yapay sinir ağına geldiğimizde "derin" gibi terimleri detaylı olarak anlatacağım.

Lojistik regresyonu (basit derin öğrenme) anlamak için önce hesaplama grafiğini öğrenelim.

# Computation Graph (Hesaplama Grafiği)

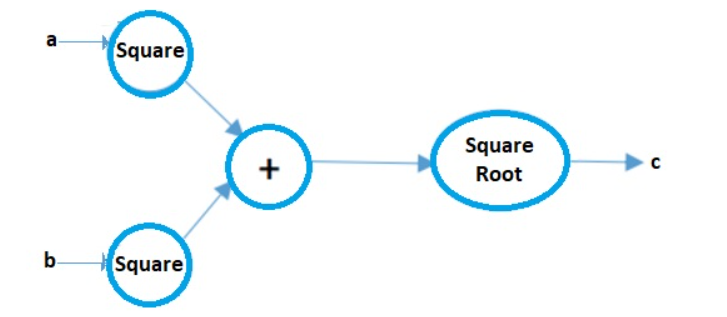
Hesaplama grafikleri, matematiksel ifadeler hakkında düşünmenin güzel bir yoludur.

Matematiksel ifadelerin görselleştirilmesi gibidir.

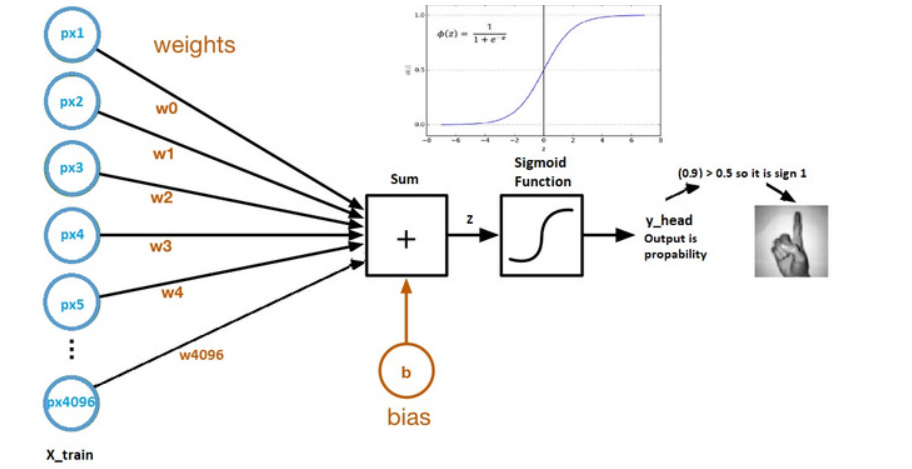
örneğin bizde



Hesaplama grafiği bu. Gördüğünüz gibi matematiği grafikle ifade ediyoruz.



Şimdi lojistik regresyonun hesaplama grafiğine bakalım



Parametreler ağırlık (weight) ve sapma(bias)dır.

weight: her pikselin katsayıları (coefficients of each pixels)

bias: araya girmek (intercept)

z = (w.t)x + b => z eşittir (transpose of weights times input x) + bias

Başka bir deyişle => z = b + px1w1 + px2w2 + ... + px4096\*w4096

y\_head = sigmoid(z)

Sigmoid işlevi, z'yi sıfır ile bir arasında yapar, yani bu olasılıktır(probability). Hesaplama grafiğinde(computation graph) sigmoid fonksiyonunu görebilirsiniz.

Neden sigmoid işlevini kullanıyoruz?

Olasılık sonucu verir(probabilistic result)

Türevdir (derivative), bu yüzden gradient descent algorithm kullanabiliriz (en kısa zamanda göreceğiz.)

Örnek yapalım:

Diyelim ki z = 4'ü bulduk ve z'yi sigmoid fonksiyonuna koyduk. Sonuç (y\_head) neredeyse 0,9'dur. Bu, sınıflandırma sonucumuzun %90 olasılıkla(probability) 1 olduğu anlamına gelir.

Şimdi baştan başlayalım ve hesaplama grafiğinin(computation graph) her bir bileşenini daha ayrıntılı olarak inceleyelim.

# Parametreleri başlatma(Initializing parameters)

Bildiğiniz gibi input, 4096 piksele sahip resimlerimizdir (her resim x\_train'de).

Her pikselin kendi weight değeri vardır.

İlk adım, her pikseli kendi weight değerleriyle çarpmaktır.

Soru şu ki, weight değerlerinin başlangıç değeri nedir?

Yapay sinir ağlarında anlatacağım bazı teknikler var ama bu sefer başlangıç ağırlıkları (initial weights) 0.01.

Tamam, weights 0,01 ama weight array shape nedir? Lojistik regresyonun hesaplama grafiğinden anlayacağınız gibi (4096,1)

Ayrıca initial bias(başlangıç değeri) 0'dır.

# Forward Propagation

Piksellerden(pixels) to cost kadar olan tüm adımlara forward propagation denir.

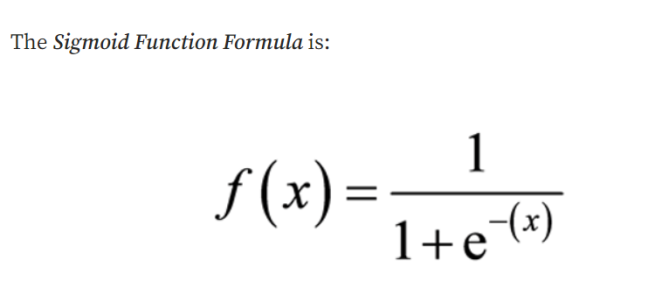
z = (w.T)x + b => bu denklemde pixel array olan x'i biliyoruz, w (weights) ve b (bias) biliyoruz, bu yüzden gerisi hesaplama. (T -> transpose)

Sonra z'yi y\_head(probability) döndüren sigmoid işlevine koyarız. Kafanız karıştığında gidin ve computation graph bakın. Ayrıca sigmoid fonksiyonunun denklemi computation graph içindedir.

Sonra loss(error) fonksiyonunu hesaplıyoruz.

Cost fonksiyonu, tüm loss(error) toplamıdır.

z ile başlayalım ve z'yi giriş parametresi olarak alan ve y\_head(probability) değerini döndüren yazma sigmoid tanımı(yöntemi) ile başlayalım.



Sigmoid yöntemini yazıp y\_head'i hesaplarken. Loss(error) fonksiyonunun ne olduğunu öğrenelim

Örnek verelim, girdi olarak bir resim koyuyorum, sonra weight ile çarpıyorum ve bias ekleyerek z'yi buluyorum. Sonra z'yi sigmoid yöntemine koyun, böylece y\_head'i bulurum. Bu noktaya kadar ne yaptığımızı biliyoruz. Daha sonra örneğin y\_head 0,5'ten büyük olan 0,9 oldu, bu nedenle tahminimiz(prediction) görüntü bir görüntü işaretidir. Tamam her şey yolunda gibi görünüyor. Ancak tahminimiz(prediction) doğru mu ve doğru olup olmadığını nasıl kontrol edeceğiz? Cevap, loss (error) işleviyle:

Log loss (error) işlevinin matematiksel ifadesi şudur:



Yanlış tahmin(wrong prediction) yaparsanız loss(error) büyük olur diyor.

Örnek: gerçek görüntümüz işaret 1 ve etiketi 1 (y = 1), o zaman y\_head = 1 tahminini(prediction) yaparız. y ve y\_head'i loss(error) denklemine koyduğumuzda sonuç -> 0 olur. loss 0'dır. Ancak y\_head= 0 gibi yanlış tahmin yaparsak loss(error) sonsuz olur.

Cost function, loss function toplamıdır. Her görüntü loss function oluşturur. Cost function, her bir girdi görüntüsü tarafından oluşturulan loss function’ların toplamıdır.

# Optimization Algorithm with Gradient Descent

Eh, şimdi error olan cost ne olduğunu biliyoruz.

Bu nedenle cost değerini düşürmemiz gerekiyor çünkü bildiğimiz gibi cost değerinin yüksek olması yanlış tahmin yaptığımız anlamına geliyor.

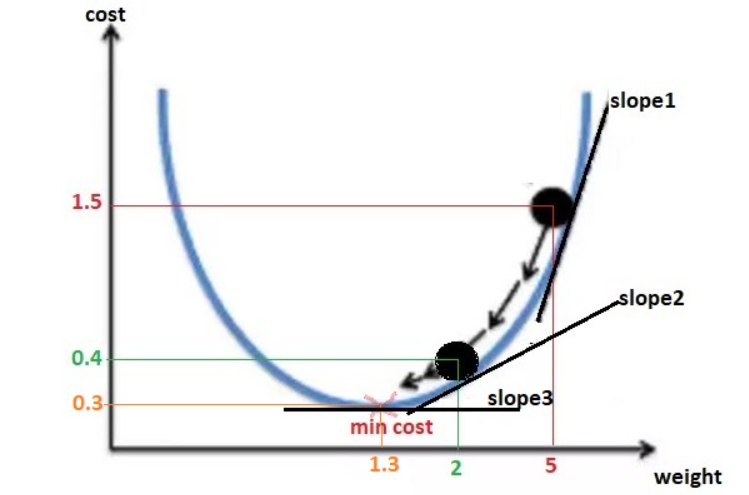
İlk adımı düşünelim, her şey weights ve bias başlatmakla (initalizing) başlar. Bu nedenle cost onlara bağlıdır.

Cost değerini düşürmek için weights ve bias güncellememiz gerekiyor.

Başka bir deyişle, modelimizin cost fonksiyonunu en aza indiren parametrelerin weights ve bias değerlerini öğrenmesi gerekir. Bu tekniğe gradient descent denir.

Bir örnek yapalım:

w = 5 ve bias = 0'a sahibiz (bu yüzden şimdilik bias’ı görmezden gelin). Sonra forward propagation yapıyoruz ve cost fonksiyonumuz 1.5. (Kırmızı çizgiyi takip et.)



Grafikten de görebileceğiniz gibi, cost fonksiyonunun minimum noktasında değiliz. Bu nedenle minimum cost geçmemiz gerekiyor. Tamam, weight güncelleyelim. ( := sembolü güncelleniyor)

w := w - adım. Soru şu ki, bu adım nedir? Adım eğim1(slope1)'dir. Tamam, dikkat çekici görünüyor. Minimum noktayı bulmak için eğim1(slope1)'i kullanabiliriz. O zaman eğim1(slope1) = 3 diyelim ve weight güncelleyelim. w := w - eğim1(slope1) => w = 2.

Şimdi w weight 2'dir. Hatırlayacağınız gibi yine forward propagation cost fonksiyonunu bulmamız gerekiyor.

Diyelim ki w = 2 ile forward propagation göre cost fonksiyonu 0,4. Hmm,cost fonksiyonumuz düştüğü için doğru yoldayız. Cost fonksiyonu için cost = 0.4 olan yeni bir değerimiz var. Bu yeterli mi? Aslında bilmiyorum bir adım daha deneyelim.

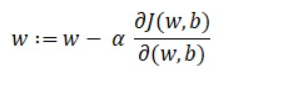
Slope2 = 0.7 ve w = 2. w : = w - step(slope2) => w = 1.3 olan yeni weight güncelleyelim. O halde yeni cost bulalım.

w = 1.3 ve cost = 0.3 ile bir forward propagation daha yapın. Tamam, cost bile düştü, iyi gibi görünüyor ama yeterli mi yoksa bir adım daha mı atmamız gerekiyor? Cevap yine bilmiyorum, deneyelim.

Slope3 ​​= 0,01 ve w = 1,3. weight güncelleme w := w - step(slope3) => w = 1.29 ~ 1.3. Yani weight değişmez çünkü minimum cost noktası fonksiyonunu buluruz.

Her şey güzel görünüyor ama eğimi(slope) nasıl buluyoruz? Liseden veya üniversiteden hatırlarsanız, verilen noktada (verilen ağırlıkta(weight)) fonksiyonun (cost fonksiyonu) eğimini(slope) bulmak için verilen noktada fonksiyonun türevini(derivative) alıyoruz. Ayrıca eğimi(slope) iyi bulduk tamam mı diye sorabilirsiniz ama nereye gittiğini nereden biliyor. Minimum noktaya gitmek yerine daha yüksek cost değerlerine gidebileceğini söyleyebiliriz. Cevap, slope (türev) hem adımı hem de adım yönünü vermesidir. O yüzden merak etmeyin :)

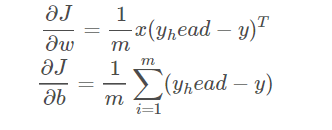
Güncelleme denklemi şudur. Bir cost işlevi olduğunu söylüyor (weight ve bias alıyor). Weight ve bias’a göre cost fonksiyonunun türevini alın. Sonra α learning rate(öğrenme oranı) ile çarpın. Ardından weight güncelleyin. (Açıklamak için bias’ı görmezden geliyorum ancak bu adımların tümü bias için de uygulanacaktır)



Şimdi eminim hiç bahsetmediğim öğrenme oranı(learning rate) nedir diye soruyorsunuzdur. Öğrenme oranını(learning rate) belirleyen çok basit bir terimdir. Ancak hızlı öğrenmek ve asla öğrenmemek arasında bir etki vardır. Örneğin, Paris'tesiniz (mevcut maliyet) ve Madrid'e gitmek istiyorsunuz (asgari maliyet). Hızınız (öğrenme oranınız) düşükse Madrid'e çok yavaş gidebilirsiniz ve çok uzun zaman alır. Öte yandan, hızınız (öğrenme oranınız) büyükse çok hızlı gidebilirsiniz ama belki kaza yaparsınız ve Madrid'e asla gidemezsiniz. Bu nedenle hızımızı (öğrenme oranımızı,learning rate) akıllıca seçmemiz gerekiyor.

Öğrenme hızı (learning rate), seçilmesi ve ayarlanması gereken hiperparametre(hyperparameter) olarak da adlandırılır. Diğer hiperparametreler(hyperparameters) ile yapay sinir ağlarında daha detaylı anlatacağım. Şimdilik, önceki örneğimiz için öğrenme oranının 1 olduğunu söyleyin.

Artık forward propagation (weights ve bias’dan, cost’a) ve backward propagation (cost’tan ,weight ve bunları güncellemek için bias’a) arkasındaki mantığı anladığınızı düşünüyorum. Kodu uygulamadan önce, cost fonksiyonunun weight ve bias’a göre türevini(derivative) nasıl aldığımızla ilgili bir şey daha öğrenmeniz gerekir. Python veya kodlama ile ilgili değildir. Saf matematiktir. İki seçenek var ilki google'da log loss function türevi nasıl alınır, ikincisi google log loss function türevi nedir ?



Basit sinir ağının (lojistik regresyon) arkasındaki mantığı ve bunun nasıl uygulanacağını öğrendik.

Artık mantığı öğrendiğimize göre, lojistik regresyon için tüm adımları elle uygulamaktan daha kolay olan **sklearn** kütüphanesini kullanabiliriz.

Kaynak : <https://www.kaggle.com/code/kanncaa1/deep-learning-tutorial-for-beginners/notebook>