Bir Sinir Ağını Kağıda Dökelim

Mert Cobanov Follow
May 20, 2019 · 6 min read

Bugün sizlerle oldukça basit bir sinir ağı inşa edeceğiz ve bunu bir hesap makinesi yardımıyla kağıt üzerinde canlandıracağız. Fakat öncesinde size bahsetmek istediğim birkaç şey var.

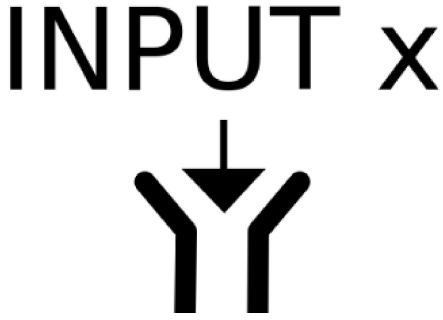
Sinir ağlarının müthiş güçlerine her gün yeniden şahit oluyoruz, özellikleri tespit ediyorlar, problemleri çözüyorlar ve sizi çılgın bir bilim adamı gibi gösteriyorlar.

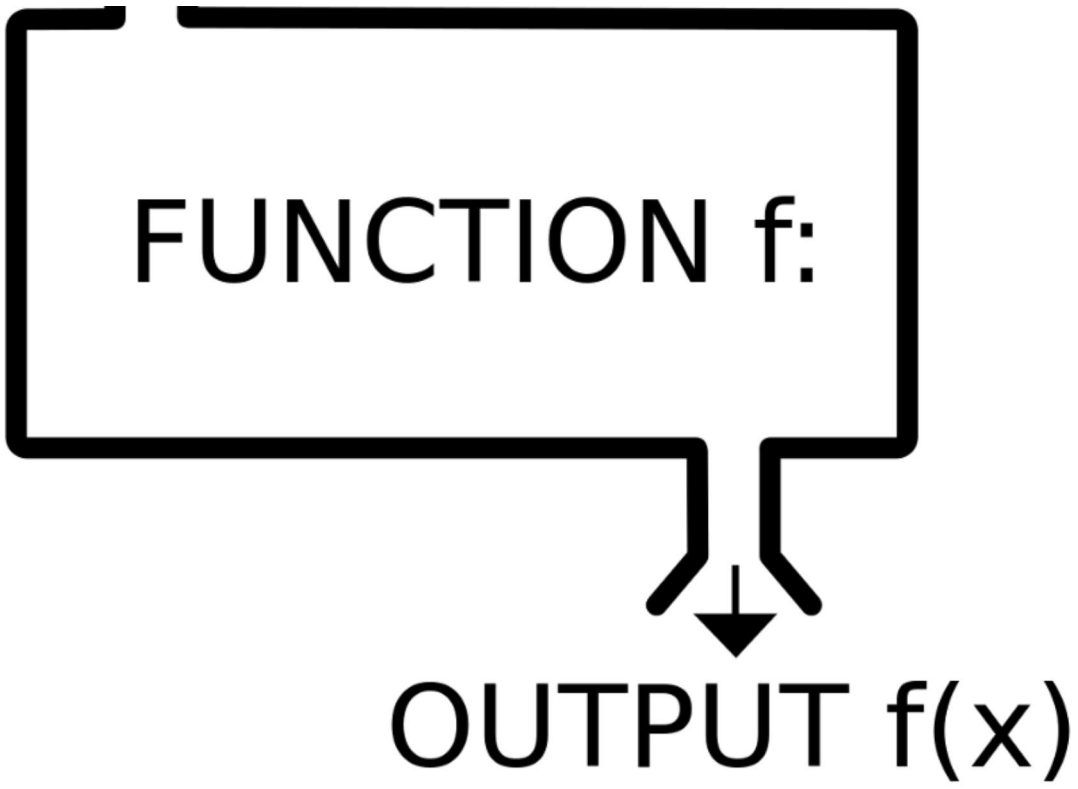
Peki neden bu kadar güçlüler? Kısa cevap; çünkü güçlerini yapılarındaki basitlikten kazanıyorlar. Uzun cevabını ise bu yazının devamında alacaksınız.

Hiç düşündünüz mü neden hemen hemen her alanda onları kullanabiliyoruz ve bilgisayarlarımızı daha güçlü kılıyorlar? Büyülü şeyler olmadıklarına emin olduğumuza göre yapılan her şeyin mantıklı bir sebebi olmalı, hadi gelin bu yazıda biraz içlerine bakalım ve aslında neler yaptıklarına bir göz atalım.

Fonksiyonlar

Birazdan size algılayıcıları anlatmak üzere birkaç terminolojiden bahsedeceğim fakat daha iyi anlamak adına biraz işin felsefesine girmekte yarar görüyorum. Ufak bir kutu hayal edin, kutunun içerisine bir şeyler atıyorsunuz ve kutudan bir takım sonuçlar elde ediyorsunuz. Oldukça basit, neyden bahsettiğimi eminim ki biliyorsunuz, aslında bu kutu bir matematiksel fonksiyon.





Her şey iyi gidiyor, elinizde bir takım girdileriniz var ve bu girdilerden çıktı üretecek bir fonksiyon tanımlıyorsunuz ve çalıştırıyorsunuz, artık elinizde çıktılarınız veya sonuçlarınız olacaktır.

Bir saniye... Ya elimizde girdilerimiz olsaydı ve aynı zamanda çıktılara da sahip olsaydık? O zaman bulmacanın eksik parçası fonksiyonun kendisi olurdu değil mi? O halde probleme farklı bir pencereden bakalım ve girdilerden elimizdeki çıktıyı üretecek kutuyu tasarlayalım. Oldukça basit değil mi? Aslında gerçekten de öyle, işte algılayıcılar (perceptrons) bu problemi çözmeye yarayan, arayıp da bulamadağımız kutsal kase. Şimdi bu nasıl yapılıyor buna bakalım.

Algılayıcılar (Perceptrons)

Perceptron, ne kadar etkileyici bir isim değil mi? Birazdan dünyayı yok eden büyük kırmızı butona basacağınız veya farklı güneş sistemlerinden gelen büyük ve istilacı devasa robotları alt edebileceğiniz bir silah gibi fakat göründüklerinden daha basitler.

Bir algılayıcı, girişlerine verdiğiniz değerleri, bir ağırlık (weight) katsayısı ile çarpan ve bir yanlılık değeri ekleyen, elde ettiği sonuç ile olmasını beklediğiniz çıktı değerlerinizi karşılaştırıp ilgili ağırlık ve yanlılık parametrelerini güncelleyerek istediğiniz çıktıyı elde etmenizi sağlayan matematiksel bir modeldir.

Biraz daha soyut düşünmekten vazgeçip hemen anlaşılmalarını basitleştirecek bir senaryo düşünelim ve hayata geçirelim. Mantık konusundan hatırlayacağınız önermeler konusunu ele alalım, bunlardan ikisi "ve" ve "veya" önermelidir. Bu kuralın aynı algılayıcıların gücünü basitliklerinden alması gibi çok önemli sonuçları var, bilgisayarların hemen hemen tüm mimarisi, elektronik süreçler ve hatta günlük hayatınızdaki kararların dahi indirgenebileceği bir yapıyı modelleyebiliyorlar.

Hızlıca mantıksal önermelere bir göz atalım.

"Ve" Bağlacı

Elinizde iki önerme olsun ve bu önermelerin değerlendirmeleri de basitçe doğru, yanlış olarak adlandırılabilsin. "Ve" bağlacı size bu önermelerin sadece aynı anda ikisinin de doğru olduğu anda pozitif çıktı veren bağlaç olarak tanımlanır.

Örnek verelim:

"Mert ve Berk birlikte okula gitti." bu sizin sonucunuz ve bunun çıktısını pozitif veya "1" kabul edelim.

Mert eğer okula gitmiş ve o gün Berk okula gitmediyse, buna "1" ve "0" olarak baktığınızda önermeniz yanlış ve bu bağlamda çıktınız "0" olacaktır. Keza Mert'in okula gitmeyip, Berk'in okula gittiği tam tersi durumda da bu çıktı halen "0" olduğunu görebiliyoruz. Hatta sorulacak son soruyu da sorup ikisinin de okula gitmediği duruma bakalım çıktımız halen "0" yani yukarıdaki ifade halen yanlış olacaktır. Fakat Mert ve Berk o gün okula gittiği "1" ve "1" durumu için ifadeniz doğru yani "1" çıktısını verecektir. O zaman bunu hemen bir tablo haline getirelim ve bakalım.

Inputs		Output
A	B	Y = A.B
0	0	0

0	1	0
1	0	0
1	1	1

Harikulade! “Veya” bağlacına girip konuyu çok fazla dağıtmadan ve hazır bilgilerimiz tazeyken şimdi bu yazıda bahsettiğimiz konulara hızlıca bir dönüş yapalım ve bir bağlantı kuralım. Tabloya bakalım, elimizde girdilerimiz var ve aynı zamanda bu girdilere göre değişen çıktılarımız var, size de umarım aynı şeyi anlatıyordur.***Algılayıcılar.*** Şimdi işin biraz daha eğlenceli tarafına atlayalım ve çok çok basit tek katmanlı bir sinir ağı yapısıyla - ki buna perceptron diyoruz- bu problemi çözelim.

Önceki yazılarımı okuyan takipçiler bilecektir ki Keras kütüphanesini öve öve bitiremiyorum, hemen bir Jupyter notebook açıyorum ve Keras’ı sahneye davet ediyorum, basit bir sinir ağı inşa edip girdilerimi ve çıktılarımı yazacağım bakalım bugün bizim için elinde neler var, görelim.

Sinir Ağını İnşa Edelim

Her zamanki gibi önce importlarımızla başlayalım, fakat bu sefer sadece iki kütüphane bizim için yeterli olacaktır.

Elimizdeki girdileri ve bu girdilerin çıktılarını tanımlayalım. Gördüğünüz gibi sırasıyla [0, 0] [0,1] [1, 0] ve [1, 1] girdilerimi yazıyorum, bu girdilere verilecek cevapları biraz önce tabloda çıkartmıştık. Kısaca onlar da [0] [0] [0] ve [1] olacaktır.

Algılayıcımız ilk katmanında 2 ve çıkışta da sadece 1 nöronu olan basit bir yapı olacak, bu “ve” bağlacı veya mühendislik alanında kullanıldığı ismiyle “ve kapısı” problemini

çözmemiz için yeterli olacaktır.

Modeli çalıştırıp bakalım sinir ağımızı gerçekten de bizim için doğru sonucu verecek yapıyı oluşturmuş mu? “0” ve “0” gönderelim.

Benim oluşturduğum model bu kod bloğuna karşılık bu cevabı veriyor:

```
array([[0.03678421], dtype=float32)
```

Enteresan, bir şeyler elde ettik fakat bu benim için bir anlam ifade etmiyor, denemeye devam edelim bu kez de “1” ve “1” gönderelim bakalım ne elde edeceğim.

```
array([[0.5142086]], dtype=float32)
```

Evet bir şeyler yakalamış gibi hissedebiliyorum, biraz daha mühendis gibi düşünelim ve tablomuzdaki olası girişleri değişkenlere tanımlayalım, sonrasında da tüm girdilerimi basit bir for döngüsü yazarak her birine hangi çıktıyı verecek yazdırmayı deneyelim.

[[0.03678421]]
[[0.364112]]
[[0.18688078]]
[[0.5142086]]

Şimdi görmesi çok daha basit bir hale getirmiş olduk. Girdilerimi verdim, sinir ağımın ideal durumdaki (tablodaki) çıktılarımı verecek şekilde ağırlıklarını ve yanlılık parametrelerini güncellemesini istedim sonrasında da oluşturduğum sinir ağına yeniden girişleri uyguladım ve baktım ne kadar iyi sonuçlar elde ediyorum.

Gördüğümüz sayılar halen ideal duruma yaklaşmış gözükmüyor fakat şu anda burada bir eşik değeri belirleseydim ve eğer çıktı 0.5 değerinden büyük olduğunda "1" ve küçük olduğunda "0" olacak şekilde bir ayarlama yapsaydım ne elde ederdim. Hemen bir fonksiyon tanımlayalım ve bu konuştuklarımızı bilgisayarımıza yaptırmaya çalışalım.

[[0.03678421]]
Bu yanlış

[[0.364112]]
Bu yanlış

[[0.18688078]]
Bu yanlış

[[0.5142086]]
Bu doğru

Mükemmel değil mi? Basit bir algılayıcı veya tek katmanlı bir sinir ağıyla “ve kapısı” problemini çözmüş olduk. Şimdi size yazının başında bahsettiğim gibi bunu basit bir hesap makinesiyle nasıl kağıda döküp gerçekleştirebileceğimizi göstereceğim.

Bunu yapmak için sinir ağımızın içerisindeki daha önce bahsettiğimiz ağırlık (weight) ve yanlılık (bias) değerlerine ihtiyacımız var. Onları da alalım ve algılayıcıların matematiğine başlayalım.

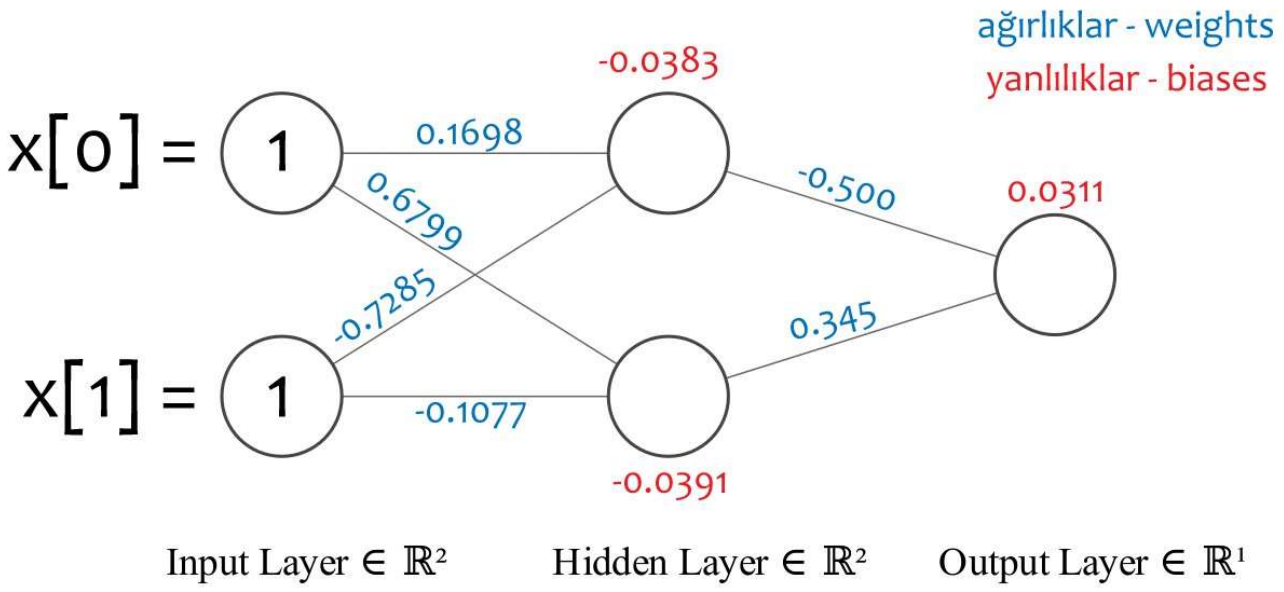
---

ilk katman ağırlıkları
[[0.16980149 0.679962]
[-0.7285618 -0.10772391]]

ilk katman yanlılıkları
[-0.03839757 -0.03918195]

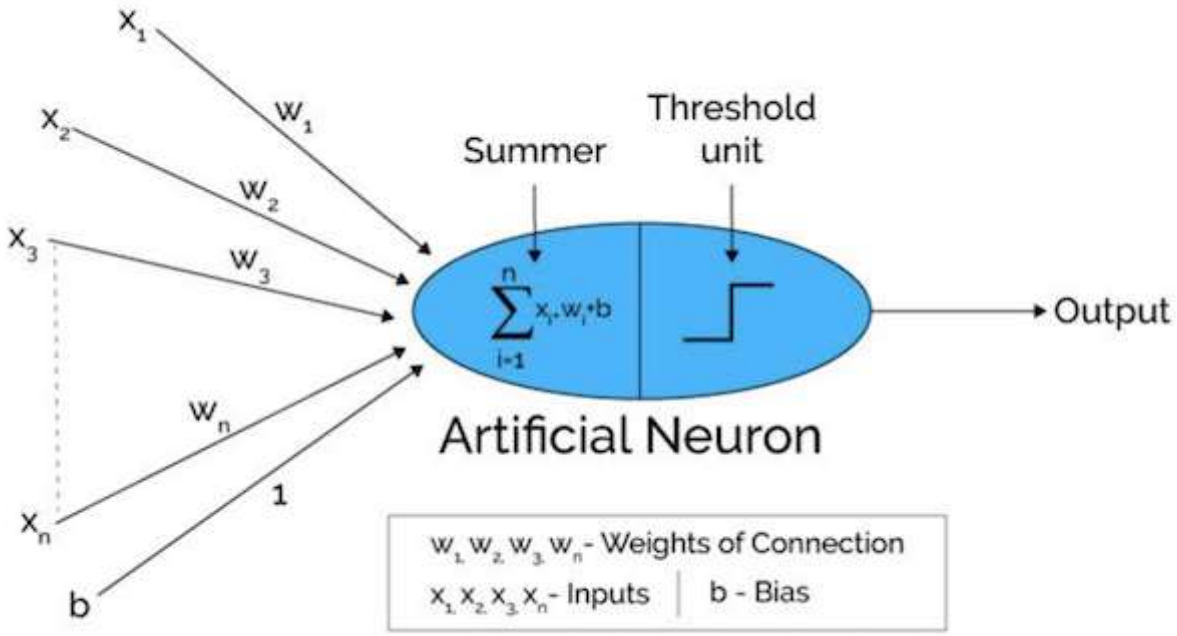
çıkış katmanı ağırlıkları
[[-0.5003943]
[0.34570208]]

çıkış katmanı yanlılıkları
[0.03111557]



Hesap Makinesini Göreve Çağırıyoruz

Şimdi elde ettiğimiz değerleri yerine yazıp nasıl bir şey beklediğimizi yukarıdaki fotoğrafta gördük, bir algılayıcı nasıl bir matematiksel ifadeyle çalışıyormuş bakalım sonrasında da bulduğumuz değerler ile hesaplama yapalım.



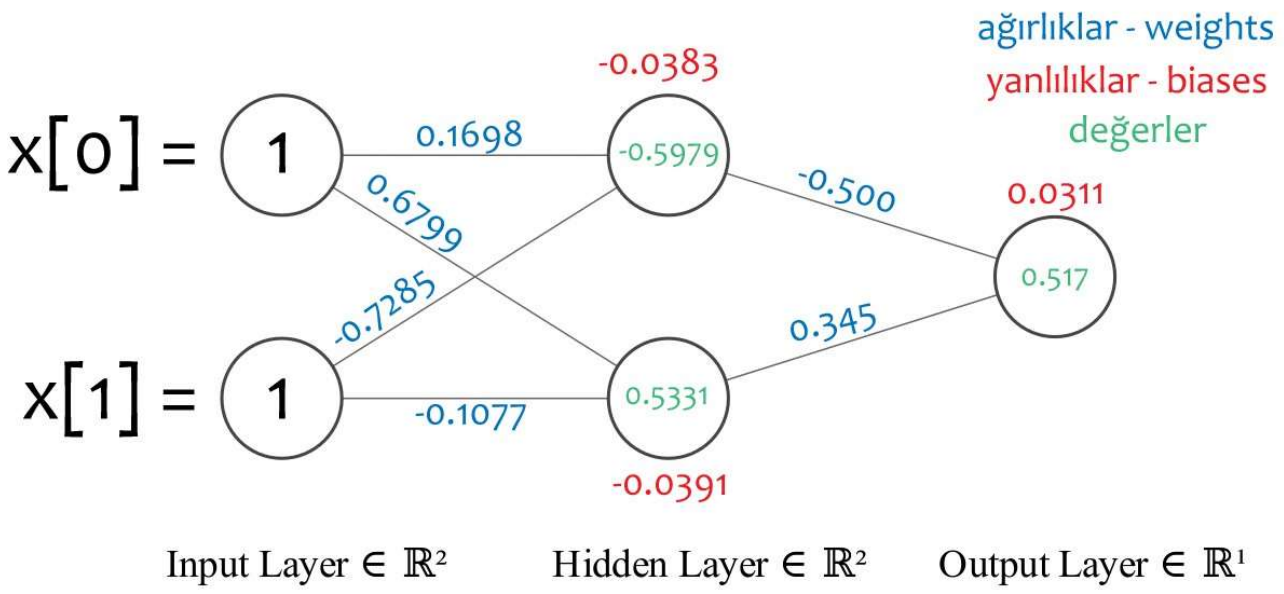
Her bir nörona gelen girdileri ağırlıkları ile çarpıyoruz ve hepsini topluyoruz, sonrasında yanlılığı ekliyoruz. Notasyon normal koşulllarda katmanların ve o katmanlardaki nöronların indisleriyle temsil ediliyor fakat bu sefer kolaylık olması açısından daha basit şekilde göstereceğim.

Hesaplaması kolay olması açısından girişlere [1,1] uyguluyorum.

Gizli Katmandaki ilk nöronun değeri:
[[1.0 x 0.1698] + [1.0 x -0.7285]] + [-0.0383] = -0.5979

Gizli Katmandaki ikinci nöronun değeri:
[[1.0 x 0.6799] + [1.0 x -0.1077]] + [-0.0391] = 0.5331

Çıkış Katmanındaki nöronun değeri:
z3 = [[-0.5979 x -0.500] + [0.5331 x 0.354]] + [0.0311] = 0.517



Yuvarlama hatalarımızla çıktımızda, [1,1] girişlerimize gerçekten de 0.517 değerini elde ederek, eşik değerimizin üzerinde yani 1 çıktısını elde ettik. Diğer girişler için de olası sonuçlara aynı yöntem ile bakabilirsiniz. Ayrıca siz eğitimi yaptığınızda sayılar ufak farklılıklar gösterebilir, bunlar doğaldır fakat yine de eşik değeriniz ile aynı sonuçları elde edeceksiniz.

Bu çalışma içerisinde ağırlık ve yanlılık parametrelerinin nasıl güncellendiğine ve nasıl belirli değerlere yaklaştığına değinmedik, bunu Keras'ın bizim için yapmasını istedik, keza geri yayılım kısmı bu kadar küçük bir ağ için dahi bir çok işlem getirecektir. Sinir ağları prensiplerinin en temel işlemi bu şekilde çalışıyor. Projenin tamamına ve diğer mantıksal kapıların nasıl çözüleceğine dair proje Github hesabımda duruyor, aşağıya linklerini bırakıyorum.

Kaynaklar

https://github.com/cobanov/NN_Gate