

ysa detaylı bilgi

Yapay Sinir Ağları ve Bilgisayar Simülasyonları: Bir Ders Kılavuzu

Bölüm 1: Yapay Zekanın Biyolojik Esin Kaynağı: Nöronlardan Ağlara

1.1. Giriş: Beyni Taklit Etmek

netlogo web

Yapay sinir ağları (YSA), en temelinde, insan beyninin karmaşık yapısını ve bilgi işleme yeteneklerini taklit etme arzusundan doğan bir hesaplama modelidir. Bu teknoloji, beynin öğrenme, hatırlama ve yeni durumlara genelleme yapma gibi olağanüstü kabiliyetlerini matematiksel ve algoritmik bir çerçeveye oturtma çabasını temsil eder. Yapay zekanın bu alt dalı, bilgisayarların yalnızca önceden programlanmış görevleri yerine getirmesini değil, aynı zamanda verilerden öğrenerek kendi kurallarını oluşturmasını ve zamanla performansını artırmasını hedefler. Bu sayede, belgeleri özetlemekten yüz tanımaya kadar birçok karmaşık problemi, hatalarından ders çıkararak ve sürekli gelişerek çözme yeteneği kazanırlar.

Bu alanın kökenleri, 20. yüzyılın ortalarına dayanmaktadır. 1958 yılında Frank Rosenblatt, bir sineğin gözünün çalışma prensiplerinden esinlenerek, yapay sinir ağlarının en temel yapı taşı olan Perceptron modelini geliştirmiştir. Bu gelişmeyi, 1959'da Bernard Widrow ve Marcian Hoff'un, gerçek bir dünya problemine (telefon hatlarındaki yankıyı filtreleme) uygulanan ilk sinir ağı olan MADALINE'i yaratması izlemiştir. Bu model, ticari olarak hala kullanımda olan bir sinir ağıdır ve YSA'ların teorik bir konsept olmaktan çıkıp pratik çözümler üretebileceğinin ilk kanıtlarından biri olmuştur.

1.2. Biyolojik Nöronun Yapısı ve İşleyişi

Yapay sinir ağlarının arkasındaki temel ilham kaynağını anlamak için, biyolojik sinir sisteminin temel birimi olan nörona bakmak gerekir. Sinir sistemi, nöron adı verilen milyarlarca özelleşmiş hücreden oluşur. Tipik bir nöron üç ana bölümden meydana gelir: dendritler, hücre gövdesi ve akson. Dışarıdan gelen bir uyarı, elektrokimyasal bir değişim yaratarak "impuls" adı verilen bir elektriksel sinyal oluşturur. Bu sinyal, ağaç dallarına benzeyen **dendritler** tarafından algılanır ve **hücre gövdesine** iletilir. Hücre gövdesinde işlenen sinyal, eğer yeterince güçlüyse, **akson** adı verilen uzun bir uzantı boyunca diğer nöronlara iletilir.

Nöronlar arasındaki bağlantı noktalarına **sinaps** denir. Bir nöronun akson ucu, diğer bir nöronun dendritine veya hücre gövdesine bu sinapslar aracılığıyla bağlanır. Öğrenme ve hafıza gibi bilişsel süreçlerin temelinde bu sinaptik bağlantıların gücünün ayarlanması yatar. Bir deneyim yaşandığında veya yeni bir bilgi öğrenildiğinde, belirli sinapslar güçlenir, zayıflar veya yeni bağlantılar oluşur. Bu sinaptik plastisite, beynin adaptasyon ve öğrenme yeteneğinin biyolojik temelidir.

1.3. Yapay Sinir Hücresi: Matematiksel Bir Model

Biyolojik nöronlardan esinlenerek oluşturulan yapay sinir hücresi (veya düğüm), bu biyolojik süreci basitleştirilmiş bir matematiksel modele dönüştürür. Bir yapay nöron, beş temel bileşenden oluşur :

1. **Girdiler (Inputs):** Bunlar, nörona dış dünyadan veya ağın bir önceki katmanındaki diğer nöronlardan gelen sayısal verilerdir. Her bir girdi, genellikle bir veri örneğinin belirli bir özelliğini (feature) temsil eder.
2. **Ağırlıklar (Weights):** Her bir girdi bağlantısının bir ağırlığı vardır. Bu ağırlık, o girdinin nöronun çıktısı üzerindeki etkisini veya "önemini" belirleyen bir katsayıdır. Biyolojik modeldeki sinapsların gücünü temsil ederler. Bir sinir ağının "öğrenme" süreci, esasen bu ağırlık değerlerinin doğru şekilde ayarlanmasıdır.
3. **Toplama Fonksiyonu:** Nörona gelen tüm girdiler, kendi ağırlıklarıyla çarpılır ve bu sonuçlar toplanır. Bu işlem, nöronun "net girdisini" oluşturur. Bu, basit bir ağırlıklı toplam işlemidir.
4. **Bias:** Bu, girdilerden bağımsız olarak toplama fonksiyonunun sonucuna eklenen bir sabit değerdir. Bias, nöronun aktivasyon eşiğini yukarı veya aşağı kaydırarak modele daha fazla esneklik kazandırır. Başka bir deyişle, tüm girdiler sıfır olsa bile nöronun bir çıktı üretmesini sağlayabilir.
5. **Aktivasyon Fonksiyonu:** Toplama fonksiyonu ve bias'tan elde edilen net girdi değeri, bir aktivasyon fonksiyonundan geçirilir. Bu fonksiyon, genellikle doğrusal olmayan bir fonksiyondur ve nöronun nihai çıktısını belirler. Bu çıktı, ya ağın nihai sonucu olur ya da bir sonraki katmandaki nöronlar için bir girdi haline gelir.

Bu yapı, en karmaşık derin öğrenme modellerinin bile temelinde yatan "ağırlıklı toplam + aktivasyon" prensibini oluşturur. Öğrenme, bu model için en iyi sonucu verecek ağırlık (w) ve bias (b) parametre değerlerini hesaplama işlemidir.

1.4. En Basit Ağ: Perceptron Modeli ve Çalışma Mantığı

Perceptron, yapay sinir ağlarının en temel ve en eski formudur. Frank Rosenblatt tarafından geliştirilen bu model, tek bir yapay sinir hücresinden oluşan tek katmanlı bir sinir ağıdır. Temel amacı, bir grup girdiyi analiz ederek bu girdilerin belirli bir sınıfa ait olup olmadığına karar vermektir. Bu nedenle, Perceptron özünde bir **ikili sınıflandırıcı (binary classifier)** olarak çalışır.

Perceptron'un çalışma mantığı oldukça basittir:

1. Girdiler (x_1, x_2, \dots, x_n) alınır.
2. Her girdi, kendi ağırlığıyla (w_1, w_2, \dots, w_n) çarpılır.
3. Bu ağırlıklı girdiler toplanır.
4. Elde edilen toplam, bir **eşik (threshold)** değeriyle karşılaştırılan bir aktivasyon fonksiyonundan geçirilir. Genellikle bu, bir basamak fonksiyonudur (step function).
5. Eğer toplam, belirlenen eşik değerini aşarsa, nöron "ateşlenir" ve çıktı olarak 1 (veya pozitif bir sınıf) üretir. Aksi takdirde, çıktı olarak -1 veya 0 (veya negatif bir sınıf) üretir.

Bu basit mekanizma, Perceptron'un bir "karar sınırı" öğrenmesini sağlar. Eğer veri noktaları iki farklı sınıfa aitse ve bu iki sınıf tek bir doğru çizgi (iki boyutta) veya bir hiperdüzlem (daha yüksek boyutlarda) ile ayrılabilirse, Perceptron bu sınırı bularak verileri doğru bir şekilde sınıflandırabilir. Bu tür problemlere **doğrusal olarak ayrılabilir (linearly separable)** problemler denir.

Bölüm 2: Perceptron'un Sınırları ve Çok Katmanlı Ağların Doğuşu: XOR Problemi

2.1. Doğrusal Ayrılabilirlik Kavramı: AND ve OR Kapıları

Tek katmanlı bir Perceptron'un gücünü ve sınırlarını anlamak için en iyi yol, temel mantık kapılarını incelemektir. **Doğrusal ayrılabilirlik**, bir veri setindeki farklı sınıflara ait noktaların, bir doğru (2D uzayda), bir düzlem (3D uzayda) veya bir hiperdüzlem (daha yüksek boyutlarda) ile tamamen birbirinden ayrılabilmesi anlamına gelir.

Örneğin, **AND** ve **OR** mantık kapıları, doğrusal olarak ayrılabilir problemlere mükemmel örneklerdir.

- **OR Kapısı:** Girdilerden en az biri 1 ise çıktı 1'dir, aksi halde 0'dır. Bu kapının doğruluk tablosundaki (0,0), (0,1), (1,0), (1,1) noktalarını bir grafiğe yerleştirdiğimizde, (0,0) noktasının (çıkıtı 0) diğer üç noktadan (çıkıtı 1) tek bir doğru çizgi ile ayrılabilirliğini görürüz. Perceptron, bu ayırıcı çizgiyi bularak OR fonksiyonunu kolayca öğrenebilir.
- **AND Kapısı:** Sadece her iki girdi de 1 ise çıktı 1'dir, diğer tüm durumlarda 0'dır. Benzer şekilde, (1,1) noktası (çıkıtı 1) diğer üç noktadan (çıkıtı 0) tek bir doğru ile ayrılabilir. Bu nedenle, AND kapısı da Perceptron tarafından çözülebilir.

Bu basit örnekler, tek katmanlı bir ağın, veriler arasında doğrusal bir ilişki veya desen olduğunda etkili bir sınıflandırıcı olabildiğini göstermektedir.

2.2. Çözümeyen Problem: XOR ve Doğrusal Olmayan Sınırlar

Perceptron'un en temel sınırlaması, doğrusal olarak ayrılabilen problemleri çözemesidir. Bu sınırlamayı en çarpıcı şekilde ortaya koyan problem, **XOR (Özel VEYA)** problemidir. XOR kapısının mantığı şöyledir: Girdiler birbirinden farklıysa (biri 0, diğeri 1 ise) çıktı 1'dir; girdiler aynıysa (her ikisi de 0 veya her ikisi de 1 ise) çıktı 0'dır.

XOR'un doğruluk tablosundaki noktaları bir grafiğe yerleştirdiğimizde durum karmaşıklaşır:

- (0,0) ve (1,1) noktaları "0" sınıfına aittir.
- (0,1) ve (1,0) noktaları "1" sınıfına aittir.

Bu noktaları birbirinden ayırmak için tek bir düz çizgi çizmek imkansızdır. "0" sınıfı ve "1" sınıfı birbirinin içine geçmiş durumdadır. Bu nedenle, XOR problemi **doğrusal olarak ayrılabilir (non-linearly separable)** bir problemdir.

Bu durum, 1969 yılında Marvin Minsky ve Seymour Papert'in "Perceptrons" adlı kitaplarında vurgulanmış ve tek katmanlı ağların bu tür temel problemleri bile çözemediği gösterilmiştir. Bu yayın, yapay sinir ağları alanındaki araştırmaların büyük ölçüde yavaşlamasına ve "Yapay Zeka Kışı" (AI Winter) olarak bilinen bir durgunluk dönemine girmesine neden olmuştur. Basit modellerin bu bariz yetersizliği, tüm alana olan güveni sarsmıştır.

2.3. Çözüm: Gizli Katmanların Gücü ve Çok Katmanlı Algılayıcı (MLP) Mimarisi

XOR probleminin ve benzeri doğrusal olmayan problemlerin yarattığı çıkmaz, **Çok Katmanlı Algılayıcı (Multi-Layer Perceptron - MLP)** mimarisinin geliştirilmesiyle aşılmıştır. MLP, en az bir veya daha fazla **gizli katman (hidden layer)** içeren bir ileri beslemeli sinir ağıdır. Bu ek katmanlar, ağı karmaşık ve doğrusal olmayan ilişkileri öğrenmesini sağlar.

Tipik bir MLP mimarisi üç tür katmandan oluşur :

1. **Giriş Katmanı (Input Layer):** Dış dünyadan ham veriyi alan katmandır. Bu katmanda herhangi bir hesaplama yapılmaz, sadece veriyi bir sonraki katmana iletir.
2. **Gizli Katman(lar) (Hidden Layer):** Giriş ve çıkış katmanları arasında yer alan katmanlardır. MLP'nin gücü bu katmanlardan gelir. Gizli katmanlardaki nöronlar, bir önceki katmandan gelen bilgileri alır, ağırlıklar ve aktivasyon fonksiyonları aracılığıyla doğrusal olmayan dönüşümlere uğratar. Bu süreç, verinin daha üst düzey ve soyut özelliklerini (feature) öğrenmeyi sağlar. Esasen, gizli katmanlar, orijinal girdi uzayını, verilerin doğrusal olarak ayrılabilirliği yeni bir temsil uzayına dönüştürür.
3. **Çıkış Katmanı (Output Layer):** Gizli katmanlarda işlenen bilgiyi alarak ağı nihai tahminini veya kararını üreten katmandır.

XOR probleminin çözümü, bu mimarinin gücünü ortaya koyar. Girdi katmanı (2 nöronlu) veriyi alır. Bir gizli katman (örneğin, 2 nöronlu), bu veriyi öyle bir şekilde dönüştürür ki, yeni temsil

uzayında "0" ve "1" sınıfları artık tek bir doğru ile ayrılabilir hale gelir. Son olarak, çıkış katmanı (1 nöronlu) bu yeni ve ayrılabilir uzayda basit bir doğrusal sınıflandırıcı gibi davranarak doğru sonucu üretir. Bu şekilde, "derinlik" (yani gizli katmanların varlığı), ağı sadece daha fazla hesaplama gücü eklemekle kalmaz, aynı zamanda verinin temsilini temelden değiştirerek Perceptron'un çözemediği problemleri çözülebilir hale getirir.

Bölüm 3: Bir Sinir Ağının Anatomisi: İleri Yayılım ve Aktivasyon Fonksiyonları

3.1. Verinin Yolculuğu: İleri Yayılım (Forward Propagation) Adım Adım

Bir sinir ağı bir tahminde bulunduğunda, veri ağı içinde belirli bir rota izler. Bu süreçte **ileri yayılım (forward propagation)** veya **ileri besleme (feedforward)** denir. Bu süreçte bilgi, ağı girişinden çıkışına doğru, geriye dönmekten tek yönde akar. Bu, bir sinir ağının tahmin üretme mekanizmasının temelidir.

İleri yayılım süreci adım adım şu şekilde işler:

1. **Giriş:** Veri örneği, giriş katmanındaki nöronlara verilir. Her nöron, verinin bir özelliğine (feature) karşılık gelir.
2. **İlk Gizli Katmana Geçiş:** Giriş katmanındaki değerler, ilk gizli katmandaki her bir nörona gönderilir.
3. **Nöron İçi Hesaplama:** Gizli katmandaki her bir nöron, aşağıdaki işlemleri gerçekleştirir:
 - a. **Ağırlıklı Toplam:** Bir önceki katmandan gelen tüm girdileri, ilgili bağlantı ağırlıklarıyla çarpar.
 - b. **Bias Ekleme:** Bu çarpımların toplamına, o nörona ait bias değerini ekler. Bu adıma **lineer adım** denir ve matematiksel olarak $z = Wx + b$ şeklinde ifade edilebilir.
 - c. **Aktivasyon:** Elde edilen net girdi (z), nöronun aktivasyon fonksiyonundan geçirilir. Bu adıma **aktivasyon adımı** denir ve sonuç $a = g(z)$ olarak gösterilir.
4. **Katmanlar Arası İlerleme:** Bir nöronun aktivasyon fonksiyonundan çıkan sonuç (a), bir sonraki katmandaki tüm nöronlar için bir girdi haline gelir.
5. **Tekrarlama:** Adım 3 ve 4, veri ağı tüm gizli katmanlarından geçip çıkış katmanına ulaşana kadar tekrarlanır.
6. **Nihai Çıktı:** Çıkış katmanındaki nöronlar, ağı nihai tahminini üretir. Bu tahmin, bir sınıflandırma problemi için bir sınıf olasılığı veya bir regresyon problemi için sayısal bir değer olabilir.

Bu deterministik süreç, ağı mevcut ağırlıkları ve bias'ları ile belirli bir girdi için hangi çıktıyı üreteceğini tanımlar.

3.2. Doğrusal Olmayanlığın Rolü: Aktivasyon Fonksiyonları

Aktivasyon fonksiyonları, bir sinir ağının en kritik bileşenlerinden biridir. Onların temel görevi, ağı **doğrusal olmayanlık (non-linearity)** katmaktır. Eğer bir sinir ağındaki tüm nöronlar sadece doğrusal aktivasyon fonksiyonları kullansaydı (veya hiç kullanmasaydı), ağ ne kadar derin olursa olsun (yani kaç katman içerirse içersin), matematiksel olarak tek bir doğrusal modele eşdeğer olurdu. Böyle bir ağ, bir Perceptron'dan daha fazlasını yapamaz ve XOR gibi en basit doğrusal olmayan problemleri bile çözemezdi.

Doğrusal olmayan aktivasyon fonksiyonlarının katman katman üst üste uygulanması, ağın son derece karmaşık, eğrisel ve çok boyutlu karar sınırları öğrenmesini mümkün kılar. Bu yetenek, **Evrensel Yaklaşım Teoremi (Universal Approximation Theorem)** ile teorik olarak desteklenir. Bu teorem, doğrusal olmayan bir aktivasyon fonksiyonuna sahip en az bir gizli katmanı olan bir ileri beslemeli sinir ağının, herhangi bir sürekli fonksiyonu istenen doğrulukta taklit edebileceğini belirtir. Bu, sinir ağlarının neden bu kadar güçlü ve esnek modelleme araçları olduğunun teorik temelidir.

3.3. Aktivasyon Fonksiyonlarının Karşılaştırılması: Sigmoid vs. ReLU

Aktivasyon fonksiyonu seçimi, bir sinir ağının öğrenme verimliliğini ve genel performansını doğrudan etkileyen önemli bir tasarım kararıdır. En yaygın kullanılan iki fonksiyon Sigmoid ve ReLU'dur.

Sigmoid Fonksiyonu

Sigmoid fonksiyonu, S-şekilli bir eğriye sahiptir ve herhangi bir reel sayı girdisini (0,1) aralığındaki bir değere dönüştürür. Bu "sıkıştırma" özelliği, onu özellikle ikili sınıflandırma problemlerinin çıkış katmanında bir olasılık değeri üretmek için kullanışlı kılar.

- **Matematiksel Formül:** $\sigma(x) = \frac{1}{1 + e^{-x}}$
- **Avantajları:** Çıktısının (0,1) aralığında olması, olasılık yorumuna olanak tanır. Fonksiyon pürüzsüz ve türevlenebilirdir, bu da gradyan tabanlı öğrenme için uygun olmasını sağlar.
- **Dezavantajları:** En büyük dezavantajı, **kaybolan gradyan (vanishing gradient)** problemine yol açmasıdır. Girdi değerleri çok büyük veya çok küçük olduğunda, Sigmoid fonksiyonunun türevi (gradyanı) sıfıra çok yaklaşır. Derin ağlarda, bu küçük gradyanlar geri yayılım sırasında katman katman çarpıldıkça daha da küçülür ve ilk katmanlardaki ağırlıkların neredeyse hiç güncellenmemesine neden olur. Bu durum, öğrenme sürecini ciddi şekilde yavaşlatır veya durdurur.

ReLU (Rectified Linear Unit) Fonksiyonu

ReLU, modern derin öğrenme mimarilerinde gizli katmanlar için fiili standart haline gelmiş bir aktivasyon fonksiyonudur. Çalışma prensibi son derece basittir.

- **Matematiksel Formül:** $\text{ReLU}(x)=\max(0,x)$
- **Avantajları:**
 - **Hesaplama Verimliliği:** Sadece bir eşikleme işlemi olduğu için Sigmoid veya Tanh gibi üstel fonksiyonlara göre hesaplaması çok daha hızlıdır.
 - **Kaybolan Gradyan Sorununu Çözme:** Pozitif girdiler için gradyan sabittir (1'dir). Bu, gradyanların geri yayılım sırasında kaybolmasını önler ve derin ağların çok daha hızlı ve etkili bir şekilde eğitilmesini sağlar.
 - **Seyreklik (Sparsity):** Negatif girdileri sıfıra eşleyerek, ağdaki bazı nöronların belirli bir girdi için aktif olmamasına neden olur. Bu "seyreklik", modelin daha verimli ve genelleştirilebilir olmasına yardımcı olabilir.
- **Dezavantajları:** En bilinen sorunu **ölen ReLU (dying ReLU)** problemidir. Bir nöronun ağırlıkları, sürekli olarak negatif bir net girdi üretecek şekilde güncellenirse, bu nöronun çıktısı kalıcı olarak sıfır olur. Sonuç olarak, bu nöronun geçen gradyan da her zaman sıfır olacağından, nöron bir daha asla öğrenme sürecine katılamaz ve "ölür".

Aşağıdaki tablo, bu iki temel aktivasyon fonksiyonunun özelliklerini özetlemektedir.

Tablo 1: Aktivasyon Fonksiyonları Karşılaştırması

Özellik	Sigmoid Fonksiyonu	ReLU (Rectified Linear Unit)
Matematiksel Formül	$\sigma(x)=1+e^{-x}$	$\text{ReLU}(x)=\max(0,x)$
Grafik	S-şekilli eğri	0'da bükülen doğrusal parça
Çıktı Aralığı	(0,1)	\$ Modelin tahminleri gerçek değerlere ne kadar yakınsa kayıp o kadar düşük, ne kadar uzaksa kayıp o kadar yüksek olur. Bir sinir ağının eğitimdeki tek ve nihai amacı, bu kayıp fonksiyonunun değerini mümkün olduğunca minimize etmektir.

Kayıp fonksiyonu seçimi, çözülen problemin türüne bağlıdır :

- **Regresyon Görevleri İçin:** Modelin sürekli bir sayı (örneğin, ev fiyatı) tahmin etmesi gereken durumlarda genellikle **Ortalama Kare Hata (Mean Squared Error - MSE)** kullanılır. MSE, tahmin ile gerçek değer arasındaki farkın karesini alarak büyük hataları daha fazla cezalandırır. Formülü $MSE=\frac{1}{n}\sum_{i=1}^n(y_i-y^i)^2$ şeklindedir, burada y_i gerçek değer ve y^i modelin tahminidir.

- **Sınıflandırma Görevleri İçin:** Modelin bir girdinin hangi kategoriye ait olduğunu (örneğin, bir e-postanın spam olup olmadığını) tahmin etmesi gereken durumlarda **Çapraz Entropi (Cross-Entropy)** kaybı kullanılır. Bu fonksiyon, modelin tahmin ettiği olasılık dağılımı ile gerçek sınıf dağılımı arasındaki farkı ölçer.

4.2. Hatayı Geriye Yaymak: Geri Yayılım (Backpropagation) Sezgisel Anlatımı

Modelin ne kadar hatalı olduğunu kayıp fonksiyonu ile ölçtüktten sonraki adım, bu hatadan hangi parametrelerin ne kadar sorumlu olduğunu bulmaktır. **Geri yayılım (backpropagation)** algoritması tam olarak bunu yapar. Bu süreci sezgisel olarak bir "suçu paylaşırma" mekanizması olarak düşünebiliriz.

Geri yayılım süreci şu adımlarla işler:

1. **İleri Yayılım ve Tahmin:** Ağ, bir girdi alır ve ileri yayılım sürecini kullanarak bir tahminde bulunur.
2. **Hata Hesaplama:** Modelin tahmini ile gerçek değer, kayıp fonksiyonu kullanılarak karşılaştırılır ve toplam hata (kayıp) değeri hesaplanır.
3. **Hatanın Geriye Yayılması:** Algoritma, ağın en son katmanından (çıkış katmanı) başlar ve geriye doğru, katman katman ilerler. Bu geri yolculuk sırasında, matematikteki **zincir kuralını (chain rule)** kullanarak her bir ağırlığın ve bias'ın toplam hataya ne kadar katkıda bulunduğunu hesaplar. Örneğin, çıkış katmanındaki bir nöronun çıktısı büyük bir hataya yol açıyorsa, o nörona gelen bağlantıların ağırlıkları bu hatadan daha fazla "sorumlu" tutulur. Bu sorumluluk (yani gradyan), bir önceki katmanlardaki ağırlıklara orantılı olarak dağıtılır. Bu işlem, giriş katmanına kadar devam eder.

Sonuç olarak geri yayılım, kayıp fonksiyonunun ağıdaki her bir ağırlığa göre kısmi türevini (yani gradyanını) verimli bir şekilde hesaplayan bir yöntemdir. Bu gradyan, her bir ağırlığın, kaybı artırmak veya azaltmak için hangi yönde ve ne kadar değiştirilmesi gerektiğini gösteren bir "yol haritası" sunar.

4.3. Minima Yolculuk: Gradyan İnişi (Gradient Descent) ile Ağırlıkların Güncellenmesi

Geri yayılım, hatanın "yol haritasını" (gradyanları) çıkardıktan sonra, **gradyan inişi (gradient descent)** algoritması bu haritayı kullanarak ağırlıkları fiilen günceller ve modeli "eğitir".

Bu süreci bir analogi ile anlamak daha kolaydır: Kayıp fonksiyonunu, derin vadileri ve yüksek tepeleri olan engebeli bir arazi olarak hayal edin. Amacımız, bu arazinin en alçak noktasına (global minimum), yani kaybın en düşük olduğu yere ulaşmaktır. Ağırlıkların mevcut değerleri, bu arazideki mevcut konumumuzu belirler.

Gradyan inişi şu şekilde çalışır:

1. **Eğimi Hesapla:** Mevcut konumumuzda durup, arazinin en dik yokuş aşağı yönünü buluruz. Bu yön, geri yayılım ile hesaplanan gradyanın tam tersidir. Gradyan en dik yokuş yukarı yönü gösterirken, negatif gradyan en dik yokuş aşağı yönü gösterir.
2. **Adım At:** Bu en dik yokuş aşağı yönde küçük bir adım atarız. Bu adım, bizi vadinin tabanına bir miktar daha yaklaştırır.
3. **Ağırlıkları Güncelle:** Bu "adım atma" işlemi, ağırlıkları güncellemek anlamına gelir. Güncelleme formülü şöyledir: $\text{yeni_ağırlık} = \text{eski_ağırlık} - \text{öğrenme_oranı} \times \text{gradyan}$.
4. **Tekrarla:** Bu süreç (eğimi hesapla, adım at, güncelle), vadinin tabanına ulaşana veya daha fazla inilemeyecek bir noktaya (yerel veya global minimum) gelene kadar binlerce, hatta milyonlarca kez tekrarlanır.

Bu döngüde **öğrenme oranı (learning rate)** kritik bir hiperparametredir. Bu değer, attığımız adımların ne kadar büyük olacağını kontrol eder. Çok büyük bir öğrenme oranı, vadinin dibini "iskalamamıza" ve karşı yamaca tırmanmamıza neden olabilir (salınım). Çok küçük bir öğrenme oranı ise vadiye çok yavaş inmemize ve eğitimin çok uzun sürmesine yol açabilir.

Özetle, **Tahmin Et (İleri Yayılım) -> Hatayı Ölç (Kayıp Fonksiyonu) -> Düzelt (Geri Yayılım + Gradyan İnişi)** döngüsü, bir sinir ağının verilerden öğrenmesini sağlayan temel mekanizmadır.

Bölüm 5: Simülasyon ve Uygulama Atölyesi

Teorik bilgileri pekiştirmenin ve sezgisel bir anlayış geliştirmenin en etkili yolu, bu kavramları uygulamalı olarak deneyimlemektir. Bu atölye, öğrencileri kodsuz interaktif bir ortamdan başlayarak, temel kütüphanelerle sıfırdan bir ağ oluşturmaya ve son olarak yüksek seviyeli bir framework ile gerçek bir problemi çözmeye yönlendiren üç aşamalı bir yapı sunmaktadır.

5.1. Pratik 1: Kodsuz Sezgisel Keşif - TensorFlow Playground

TensorFlow Playground, bir sinir ağının nasıl çalıştığını kod yazmadan, tamamen görsel ve interaktif bir arayüzde keşfetmek için tasarlanmış web tabanlı bir araçtır. Bu araç, soyut kavramları somutlaştırmak için mükemmel bir başlangıç noktasıdır.

5.1.1. Arayüzü Tanıma

Öğrenciler, arayüzün ana bileşenlerini tanıyarak başlayacaklardır:

- **Veri Setleri:** Sol üst köşede, Circle, XOR, Gaussian ve Spiral gibi farklı karmaşıklık seviyelerine sahip önceden tanımlanmış veri setleri arasından seçim yapabilirler. Bu setler, doğrusal ve doğrusal olmayan sınıflandırma problemlerini temsil eder.
- **Özellikler (Features):** Giriş katmanına hangi verilerin besleneceği seçilir. Ham girdiler (X_1, X_2) veya bu girdilerden türetilmiş doğrusal olmayan özellikler ($X_{12}, X_{22}, X_1X_2, \sin(X_1)$)

kullanılabilir. Bu, özellik mühendisliğinin etkisini görselleştirmeye olanak tanır.

- **Mimari:** Ortadaki bölümde, "+" ve "-" butonları kullanılarak gizli katman sayısı ve her katmandaki nöron sayısı dinamik olarak değiştirilebilir. Bu, model karmaşıklığının etkisini anında gözlemlemeyi sağlar.
- **Hiperparametreler:** Üst kısımda, **Öğrenme Oranı (Learning Rate)**, **Aktivasyon Fonksiyonu** (ReLU, Tanh, Sigmoid, Linear), **Regülerizasyon** (L1/L2) ve **Problem Tipi** (Sınıflandırma/Regresyon) gibi kritik eğitim parametreleri ayarlanabilir.

5.1.2. XOR Problemini İnteraktif Olarak Çözme

Öğrenciler, XOR veri setini seçerek atölyeye başlarlar. İlk olarak, gizli katman olmadan (tek katmanlı bir Perceptron simüle ederek) ağı çalıştırmaları istenir. Bu durumda, ağı veri noktalarını ayıran tek bir doğru çizgi çizmeye çalıştığını ancak başarısız olduğunu gözlemleyeceklerdir. Ardından, bir gizli katman ekleyip nöron sayısını (örneğin 2 veya 3'e) artırdıklarında, ağı artık doğrusal olmayan, eğrisel bir karar sınırı oluşturarak XOR problemini başarıyla çözdüğünü göreceklerdir.

5.1.3. Karar Sınırlarının ve Kayıp Eğrisinin Evrimini Gözlemleme

"Play" düğmesine basıldığında, eğitim süreci başlar. Öğrenciler iki şeyi eş zamanlı olarak izleyebilirler:

1. **Karar Sınırı Evrimi:** Sağdaki ana görselleştirme alanında, arka plan renkleri ağı karar sınırlarını temsil eder. Eğitim ilerledikçe, bu renkli bölgelerin şekil değiştirerek mavi ve turuncu noktaları doğru bir şekilde sarmaya çalıştığını canlı olarak izlerler.
2. **Kayıp Eğrisi:** Sağ üst köşedeki grafikte, hem eğitim kaybı (Training loss) hem de test kaybı (Test loss) değerlerinin epoch'lar ilerledikçe nasıl düştüğünü takip ederler.

Bu interaktif deneyim, örneğin öğrenme oranını çok yükseğe ayarlamanın kayıp eğrisinde nasıl düzensiz salınımlara (oscillating loss) yol açtığını veya çok düşük ayarlamanın öğrenmeyi nasıl yavaşlattığını doğrudan görmelerini sağlar.

5.2. Pratik 2: Temelleri Pekiştirme - NumPy ile Sıfırdan XOR Ağı Kodlama

TensorFlow Playground ile kazanılan sezgisel anlayışın ardından, bu bölümde öğrenciler bir kütüphanenin soyutlaması olmadan, temel matematiksel işlemlerle bir sinir ağına öğrenme mekanizmasını kodlayacaklardır. Bu pratik, ileri ve geri yayılımın arkasındaki "sihri" ortadan kaldırır ve temel matris işlemlerine dayandığını gösterir.

5.2.1. Mimarinin Kurulması ve Ağırlıkların Başlatılması

XOR problemini çözmek için 2 giriş nöronu, 2 gizli katman nöronu ve 1 çıkış nöronu içeren bir mimari (2-2-1) tanımlanacaktır. Ağırlık matrisleri ve bias vektörleri, küçük rastgele sayılarla başlatılacaktır. Sıfırdan başlatma simetriyi kıramayacağı için öğrenmeyi engeller.

5.2.2. İleri ve Geri Yayılım Fonksiyonlarının Yazılması

İki ana fonksiyon kodlanacaktır:

- **forward()** : Bu fonksiyon, girdi verilerini ve mevcut ağırlıkları alarak, matris çarpımları (`np.dot`) ve Sigmoid aktivasyon fonksiyonunu kullanarak katman katman ileri ve nihai tahmini üretir.
- **backward()** : Bu fonksiyon, ileri yayılımdan gelen tahmin ile gerçek değer arasındaki hatayı hesaplar. Ardından, zincir kuralının matematiksel karşılığı olan türevleri kullanarak, her bir ağırlık ve bias için gradyanları (yani güncelleme miktarlarını) hesaplar ve döndürür.

5.2.3. Eğitim Döngüsü ve Ağırlık Güncellemelerinin Uygulanması

Bir `for` döngüsü içinde binlerce iterasyonluk bir eğitim süreci oluşturulur. Her bir iterasyonda:

1. Tüm eğitim örnekleri için `forward()` çağrılarak tahminler yapılır.
2. Bir kayıp fonksiyonu (örneğin, MSE) kullanılarak toplam hata hesaplanır.
3. `backward()` çağrılarak gradyanlar hesaplanır.
4. Her bir ağırlık ve bias, $\text{parametre} = \text{parametre} - \text{öğrenme_oranı} \times \text{gradyan}$ formülü kullanılarak güncellenir.

Bu pratik, öğrencilere `model.fit()` gibi sihirli görünen bir komutun arkasında aslında bu temel ve tekrarlı hesaplama döngüsünün yattığını somut bir şekilde öğretir.

5.3. Pratik 3: Gerçek Dünya Problemi - Keras ile MNIST Rakam Tanıma

Son aşamada, öğrenciler öğrendikleri tüm teorik ve pratik bilgileri, Keras gibi modern ve yüksek seviyeli bir derin öğrenme kütüphanesi kullanarak, yaygın bir bilgisayarlı görü problemi olan el yazısı rakam tanıma görevine uygulayacaklardır.

5.3.1. Veri Hazırlama

MNIST veri seti, 70,000 adet 28x28 piksel boyutunda, 0'dan 9'a kadar olan el yazısı rakamların gri tonlamalı görüntülerinden oluşur. Modelin bu veriyi işleyebilmesi için iki temel ön işleme adımı uygulanır :

- **Düzleştirme (Flattening):** Her bir 28x28'lik 2D görüntü matrisi, 784 elemanlı tek boyutlu bir vektöre dönüştürülür. Bu, standart bir `Dense` katmanının girdisi için gereklidir.

- **Normalizasyon:** Görüntülerin piksel değerleri genellikle aralığındadır. Bu değerler, aralığına (değerleri 255'e bölerek) veya $[-0.5, 0.5]$ aralığına ölçeklenir. Bu işlem, gradyan inişi optimizasyonunun daha stabil ve hızlı yakınsamasına yardımcı olur.

5.3.2. Keras Sequential Modeli ile Ağ Mimarisi Oluşturma

Keras'ın Sequential API'si, katmanları doğrusal bir şekilde üst üste ekleyerek model oluşturmayı kolaylaştırır. MNIST için basit bir MLP mimarisi şu şekilde oluşturulabilir :

1. **Giriş Katmanı:** `Flatten(input_shape=(28, 28))` katmanı eklenerek 28x28'lik girdilerin düzleştirileceği belirtilir.
2. **Gizli Katman(lar):** Bir veya daha fazla `Dense` katman eklenir. Örneğin, `Dense(128, activation='relu')` şeklinde 128 nöronlu ve ReLU aktivasyonlu bir gizli katman tanımlanabilir.
3. **Çıkış Katmanı:** Son katman, sınıf sayısına eşit sayıda nöron içermelidir. MNIST için 10 rakam (0-9) olduğundan, 10 nöronlu bir `Dense` katman kullanılır. Çok sınıflı bir sınıflandırma problemi olduğu için aktivasyon fonksiyonu olarak `softmax` seçilir. Softmax, 10 nöronun çıktısını, toplamı 1 olan bir olasılık dağılımına dönüştürür. Her bir çıktı, görüntünün ilgili rakama ait olma olasılığını temsil eder.

5.3.3. Modeli Derleme, Eğitme ve Performansını Değerlendirme

Model mimarisi tanımlandıktan sonra, eğitim süreci üç basit komutla yönetilir:

- `model.compile()` : Bu adımda öğrenme süreci yapılandırılır. Bir **optimizasyon algoritması** (genellikle `adam`), bir **kayıp fonksiyonu** (çok sınıflı sınıflandırma için `sparse_categorical_crossentropy`) ve izlenecek **performans metrikleri** (`accuracy` gibi) belirtilir.
- `model.fit()` : Bu komut, modeli eğitir. Eğitim verileri (`x_train`, `y_train`), veri seti üzerinden kaç kez geçileceği (**epochs**) ve her güncellemede kaç örneğin kullanılacağı (**batch_size**) gibi parametreler alır.
- `model.evaluate()` : Eğitim tamamlandıktan sonra, modelin genelleme yeteneğini ölçmek için daha önce hiç görmediği test verileri (`x_test`, `y_test`) üzerinde performansı değerlendirilir.

5.3.4. Eğitim Sürecini Yorumlama

`model.fit()` metodu, her epoch'taki kayıp ve doğruluk değerlerini içeren bir `history` nesnesi döndürür. Bu veriler kullanılarak eğitim ve doğrulama metriklerinin zaman içindeki değişimi görselleştirilebilir. Bu grafikler, modelin öğrenip öğrenmediğini, ne zaman öğrenmeyi durdurduğunu veya eğitim verisine aşırı uyum sağlayıp sağlamadığını (overfitting) anlamak için hayati öneme sahiptir.

Aşağıdaki tablo, Keras'ta oluşturulan basit bir MNIST modelinin `model.summary()` çıktısını ve bu çıktının nasıl yorumlanacağını göstermektedir. Bu özet, oluşturulan soyut modelin somut yapısını, katmanlarını ve öğrenilebilir parametre sayısını net bir şekilde ortaya koyar.

Tablo 2: Keras ile MNIST Modeli Özeti (`model.summary()`)

Katman Tipi (Layer Type)	Çıktı Boyutu (Output Shape)	Parametre Sayısı (Param #)	Açıklama
<code>flatten</code>	(None, 784)	0	28x28'lik girdiyi 784 elemanlı bir vektöre dönüştürür. Öğrenilebilir parametresi yoktur.
<code>dense</code>	(None, 128)	100480	İlk gizli katman. Parametre sayısı: (784 girdi x 128 nöron) + 128 bias = 100352 + 128 = 100480.
<code>dense_1</code>	(None, 32)	4128	İkinci gizli katman. Parametre sayısı: (128 girdi x 32 nöron) + 32 bias = 4096 + 32 = 4128.
<code>dense_2</code>	(None, 10)	330	Çıkış katmanı. Parametre sayısı: (32 girdi x 10 nöron) + 10 bias = 320 + 10 = 330.
Toplam Parametreler		104,938	Modelin eğitim sırasında öğreneceği toplam ağırlık ve bias sayısıdır.

Export to Sheets

Bu üç aşamalı atölye, öğrencilere sadece bir kütüphaneyi nasıl kullanacaklarını öğretmekle kalmaz, aynı zamanda bu kütüphanelerin çözdüğü temel problemleri ve bu çözümlerin arkasındaki mekanizmaları da derinlemesine anlamalarını sağlar. Bu yaklaşım, onları sadece bir "kütüphane kullanıcısı" olmaktan çıkarıp, problemleri temelden anlayabilen ve çözebilen yetkin programcılar haline getirir.

Sources used in the report



[

veribilimiokulu.com

Yapay Sinir Ağı(Artificial Neural Network) Nedir? - Veri Bilimi Okulu

Opens in a new window](<https://www.veribilimiokulu.com/yapay-sinir-agiartificial-neural->



[network-nedir/](#)) [

ennurcitir.wordpress.com

Yapay Sinir Ağları - ENNUR ÇITIR - WordPress.com

Opens in a new window](<https://ennurcitir.wordpress.com/2020/02/19/yapay-sinir-aglari/>) [



aws.amazon.com

Sinir Ağı Nedir? - Yapay Sinir Ağına Ayrıntılı Bakış - AWS

Opens in a new window](<https://aws.amazon.com/tr/what-is/neural-network/>) [



isikhanelif.medium.com

Perceptron Nedir?. Bu yazımda kısaca Perceptron konusuna... | by ...

Opens in a new window](<https://isikhanelif.medium.com/perceptron-nedir-83dd39aada0>) [



medium.com

XOR Problemi Nedir ve Çözümü. 1960'lı yılların sonunda yapay sinir ...

Opens in a new window]([https://medium.com/@ugurcan.soruc/xor-problemi-nedir-ve-](https://medium.com/@ugurcan.soruc/xor-problemi-nedir-ve-%C3%A7%C3%B6z%C3%BCm%C3%BC-197f9b110053)

[%C3%A7%C3%B6z%C3%BCm%C3%BC-197f9b110053](https://medium.com/@ugurcan.soruc/xor-problemi-nedir-ve-%C3%A7%C3%B6z%C3%BCm%C3%BC-197f9b110053)) [

bilgisayarkavramlari.com

Yahut Problemi (Özel Veya Problemi (XOR Problem, exclusive or)) - Bilgisayar Kavramları



Opens in a new window](<https://bilgisayarkavramlari.com/2008/10/05/ozel-veya-problemi-xor->



[problem-exclusive-or/](#)) [

educative.io

XOR problem in neural network - Educative.io

Opens in a new window](<https://www.educative.io/answers/xor-problem-in-neural-network>) [

aliosmangokcan.com

YSA ile XOR GATE Uygulaması - Öğr.Gör.Ali Osman GÖKCAN

Opens in a new window](<https://aliosmangokcan.com/index.php/kodlama/11-ysa-ile-xor->



[uygulamasi](#)) [

omclbb.medium.com

Yapay Sinir Ağları, XOR Problemi ve Python Dili Kullanılarak ...

Opens in a new window](<https://omclbb.medium.com/yapay-sinir-a%C4%9Flar%C4%B1-ve-xor-problemi-ve-python-dili-kullan%C4%B1larak-%C3%A7%C3%B6z%C3%BCm%C3%BCn-yap%C4%B1lmas%C4%B1-cb08c3c6ef99>) [

tarimorman.gov.tr

Yapay Sinir Ağları ile Hidrolojik Modelleme

Opens in a new window]

(<https://www.tarimorman.gov.tr/SYGM/Belgeler/Ta%C5%9Fk%C4%B1n%20SON/Yapay%20Sin>


[ir%20Aglar%C4%B1%20ile%20Hidrolojik%20Modelleme_Dr.%20Gok%C3%A7en%20UYSAL.p](#)



[df\)](#) [

medium.com

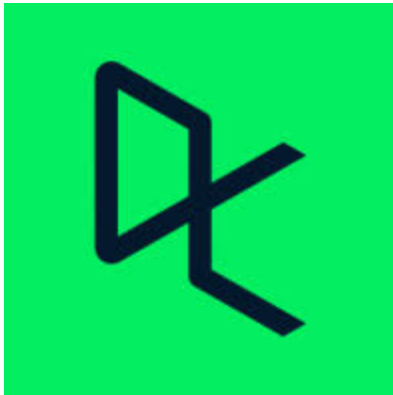
MLP (Çok Katmanlı Algılayıcı) Yapay Sinir Ağları | by Baristuzemenn ...

Opens in a new window](<https://medium.com/@baristuzemenn/mlp-%C3%A7ok-katmanli%C4%B1-alg%C4%B1lay%C4%B1c%C4%B1-yapay-sinir-a%C4%9Flar%C4%B1-a8c9c68748f6>) [

geeksforgeeks.org

Feedforward Neural Network - GeeksforGeeks

Opens in a new window](<https://www.geeksforgeeks.org/nlp/feedforward-neural-network/>) [



datacamp.com

Feed-Forward Neural Networks Explained: A Complete Tutorial ...

Opens in a new window]([https://www.datacamp.com/tutorial/feed-forward-neural-networks-](https://www.datacamp.com/tutorial/feed-forward-neural-networks-explained)

[explained\)](#) [

miuul.com

Yapay Sinir Ağlarında Öğrenme Süreci: Ağırlıklar, Forward ve Backpropagation - Miuul

Opens in a new window](<https://miuul.com/blog/yapay-sinir-aglarinda-ogrenme-sureci>) [



en.wikipedia.org

Activation function - Wikipedia

Opens in a new window](https://en.wikipedia.org/wiki/Activation_function) [

geeksforgeeks.org

Tanh vs. Sigmoid vs. ReLU - GeeksforGeeks

Opens in a new window]([https://www.geeksforgeeks.org/deep-learning/tanh-vs-sigmoid-vs-](https://www.geeksforgeeks.org/deep-learning/tanh-vs-sigmoid-vs-relu/)

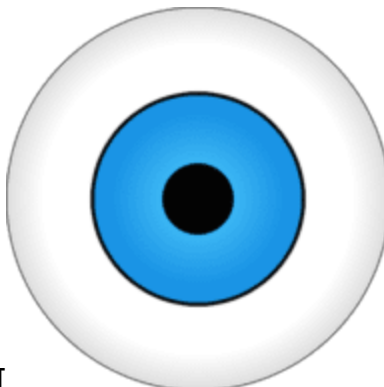


[relu/](#)) [

developers.google.com

Neural networks: Activation functions | Machine Learning - Google for Developers

Opens in a new window]([https://developers.google.com/machine-learning/crash-course/neural-](https://developers.google.com/machine-learning/crash-course/neural-networks/activation-functions)



[networks/activation-functions](#)) [

learnopencv.com

Understanding Feedforward Neural Networks | LearnOpenCV

Opens in a new window]([https://learnopencv.com/understanding-feedforward-neural-](https://learnopencv.com/understanding-feedforward-neural-networks/)



[networks/](https://learnopencv.com/understanding-feedforward-neural-networks/)) [

encord.com

Activation Functions in Neural Networks: 15 examples - Encord

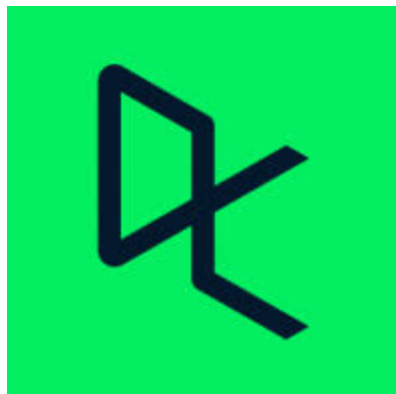
Opens in a new window](<https://encord.com/blog/activation-functions-neural-networks/>) [



builtin.com

Loss Functions in Neural Networks & Deep Learning | Built In

Opens in a new window](<https://builtin.com/machine-learning/loss-functions>) [



datacamp.com

Loss Functions in Machine Learning Explained - DataCamp

Opens in a new window](<https://www.datacamp.com/tutorial/loss-function-in-machine-learning>) [



ibm.com

What is Loss Function? | IBM

Opens in a new window](<https://www.ibm.com/think/topics/loss-function>) [



arxiv.org

Loss Functions in Deep Learning: A Comprehensive Review - arXiv

DP
AKADEMİK

Opens in a new window](<https://arxiv.org/html/2504.04242v1>) [

dergipark.org.tr

Yapay Sinir Ağları ile Tahmin ve Sınıflandırma Problemlerinin Çözümü İçin Arayüz Tasarımı - DergiPark

Opens in a new window](<https://dergipark.org.tr/tr/download/article-file/398974>) [



komtas.com

Backpropagation (geri yayılım) Nedir? - Komtaş

Opens in a new window](<https://www.komtas.com/glossary/backpropagation-nedir>) [



yazgit.com.tr

Yapay Sinir Ağlarında Ağırlık Başlatma Yöntemleri: Eğitim Sürecinin Gizli Mimarı - YAZGİT

Opens in a new window](<https://www.yazgit.com.tr/yapay-sinir-aglarinda-agirlik-baslatma-yontemleri-egitim-surecinin-gizli-mimari/>) [



yontemleri-egitim-surecinin-gizli-mimari/) [

medium.com

Yapay Sinir Ağlarında Öğrenme Süreci: Ağırlıklar, Forward and Backpropagation - Medium

Opens in a new window](<https://medium.com/@gurkanc/yapay-sinir-a%C4%9Flar%C4%B1nda-%C3%B6%C4%9Frenme-s%C3%BCreci-b10ec4925c1b>) [



medium.com

Building an XOR Neural Network from Scratch: Learn from the ...

](<https://medium.com/@derek246810/building-an-xor-neural-network-from-scratch-learn-from-the-basics-63a2a22495ae>)