10.Deep Learning Basics:Temelleri

176

Malware Data Science is set in New Baskerville, Futura, Dogma, andTheSans-

MonoCondensed.

Visit https://www.nostarch.com/malwaredatascience/ for updates, errata, and

other information.

Derin öğrenme, işleme gücündeki ve derin öğrenme tekniklerindeki gelişmeler nedeniyle son birkaç yılda hızla gelişen bir tür makine öğrenimidir. Genellikle derin öğrenme, görüntü tanıma ve dil çevirisi gibi çok karmaşık, genellikle tarihsel olarak insan merkezli görevleri gerçekleştirmede mükemmel olan derin veya çok katmanlı sinir ağlarını ifade eder.

Örneğin, bir dosyanın daha önce gördüğünüz bazı kötü amaçlı kodların tam bir kopyasını içerip içermediğini tespit etmek bir bilgisayar programı için basittir ve gelişmiş makine öğrenimi gerektirmez. Ancak bir dosyanın daha önce gördüğünüz kötü amaçlı koda biraz benzeyen kötü amaçlı kod içerip içermediğini tespit etmek çok daha karmaşık bir iştir. Geleneksel imza tabanlı algılama şemaları katıdır ve daha önce hiç görülmemiş veya karartılmış kötü amaçlı yazılımlarda kötü performans gösterirken, derin öğrenme modelleri yüzeysel değişiklikleri görebilir ve bir örneği kötü niyetli yapan temel özellikleri belirleyebilir. Aynısı ağ etkinliği, davranış analizi ve diğer ilgili alanlar için de geçerlidir. Bir yığın gürültü içindeki bu yararlı özellikleri seçme yeteneği, derin öğrenmeyi siber güvenlik uygulamaları için son derece güçlü bir araç haline getirir.

Derin öğrenme yalnızca bir tür makine öğrenimidir (makine öğrenimini genel olarak Bölüm 6 ve 7'de ele aldık). Ancak, genellikle önceki bölümlerde tartıştığımız yaklaşımlardan daha iyi doğruluk elde eden modellere yol açar, bu nedenle makine öğreniminin tüm alanı son beş yılda derin öğrenmeye vurgu yapmıştır. Güvenlik veri biliminin en son noktasında çalışmakla ilgileniyorsanız, derin öğrenmeyi nasıl kullanacağınızı öğrenmek çok önemlidir. Bununla birlikte, bir uyarı notu: Derin öğrenmenin anlaşılması, bu kitabın başlarında tartıştığımız makine öğrenimi yaklaşımlarından daha zordur ve tam olarak anlamak için biraz kararlılık ve lise düzeyinde matematik gerektirir. Bunu anlamak için harcadığınız zamanın, daha doğru makine öğrenimi sistemleri oluşturma beceriniz açısından güvenlik veri biliminize faydası olacağını göreceksiniz. Bu nedenle, bu bölümü dikkatlice okumanızı ve onu alana kadar anlamaya çalışmanızı tavsiye ediyoruz! Başlayalım.

\*What is Deep Learning?

Derin öğrenme modelleri, eğitim verilerini iç içe geçmiş bir kavramlar hiyerarşisi olarak görmeyi öğrenir ve bu da inanılmaz karmaşık kalıpları temsil etmelerine olanak tanır. Başka bir deyişle, bu modeller yalnızca onlara verdiğiniz orijinal özellikleri dikkate almakla kalmaz, aynı zamanda bu özellikleri yeni, optimize edilmiş meta özellikler oluşturmak için otomatik olarak birleştirir, daha sonra bunları daha da fazla özellik oluşturmak için birleştirir ve bu böyle devam eder.

neuron sinir

"Derin" aynı zamanda bunu başarmak için kullanılan mimariye de atıfta bulunur ve genellikle her biri önceki katmanın çıktılarını girdi olarak kullanan birden çok işlem birimi katmanından oluşur. Bu işlem birimlerinin her birine nöron denir ve bir bütün olarak model mimarisine sinir ağı veya çok sayıda katman olduğunda derin sinir ağı denir. Bu mimarinin nasıl yardımcı olabileceğini görmek için, görüntüleri bisiklet veya tek tekerlekli bisiklet olarak sınıflandırmaya çalışan bir program düşünelim. Bir insan için bu kolay bir iştir, ancak bir bilgisayarı bir piksel ızgarasına bakacak ve hangi nesneyi temsil ettiğini söyleyecek şekilde programlamak oldukça zordur. Bir görüntüde tek tekerlekli bisikletin var olduğunu gösteren belirli pikseller, tek tekerlekli bisiklet hafifçe hareket ettirildiyse, farklı bir açıyla yerleştirildiyse veya farklı bir renge sahipse, bir sonrakinde tamamen başka bir anlama gelecektir.

Derin öğrenme modelleri, sorunu daha yönetilebilir parçalara bölerek bunu aşar. Örneğin, derin bir sinir ağının ilk nöron katmanı önce görüntüyü parçalara ayırabilir ve görüntüdeki şekillerin kenarları ve sınırları gibi düşük seviyeli görsel özellikleri belirleyebilir. Oluşturulan bu özellikler, özellikler arasında kalıplar bulmak için ağın bir sonraki katmanına beslenir. Bu modeller daha sonra, ağ genel şekilleri tanımlayana ve sonunda nesneleri tamamlayana kadar sonraki katmanlara beslenir. Tek tekerlekli bisiklet örneğimizde, ilk katman doğruları bulabilir, ikincisi daire oluşturan çizgileri görebilir ve üçüncüsü, belirli dairelerin aslında tekerlek olduğunu belirleyebilir. Bu şekilde, model bir piksel kütlesine bakmak yerine, her görüntünün belirli sayıda "tekerlek" metafizatı olduğunu görebilir. Daha sonra, örneğin, iki tekerleğin muhtemelen bir bisikleti, bir tekerleğin ise tek tekerlekli bisiklet anlamına geldiğini öğrenebilir.

Bu bölümde, sinir ağlarının hem matematiksel hem de yapısal olarak gerçekte nasıl çalıştığına odaklanıyoruz. İlk olarak, bir nöronun tam olarak ne olduğunu ve bir sinir ağı oluşturmak için diğer nöronlara nasıl bağlandığını açıklamak için örnek olarak çok basit bir sinir ağını kullanıyorum. İkinci olarak, bu ağları eğitmek için kullanılan matematiksel süreçleri anlatıyorum. Son olarak, bazı popüler sinir ağları türlerini, nasıl özel olduklarını ve nelerde iyi olduklarını anlatıyorum. Bu sizi Python'da derin öğrenme modelleri oluşturacağınız 11. Bölüm için güzel bir şekilde hazırlayacak.

\*How Neural Networks Work:Sinir Ağları Nasıl Çalışır?

Makine öğrenimi modelleri basitçe büyük matematiksel işlevlerdir. Örneğin, girdi verilerini (bir dizi sayı olarak temsil edilen bir HTML dosyası gibi) alıyoruz, bir makine öğrenimi işlevi uyguluyoruz (bir sinir ağı gibi) ve bize HTML dosyasının ne kadar kötü niyetli göründüğünü söyleyen bir çıktı elde ediyoruz. Her makine öğrenimi modeli, eğitim sürecinde optimize edilen ayarlanabilir parametreleri içeren bir işlevdir.

Fakat derin öğrenme işlevi gerçekte nasıl çalışır ve neye benzer? Sinir ağları, adından da anlaşılacağı gibi, sadece birçok nöronun ağlarıdır. Yani, sinir ağlarının nasıl çalıştığını anlayabilmemiz için önce nöronun ne olduğunu bilmemiz gerekir.

\*Anatomy of a Neuron:Bir Nöronun Anatomisi

Nöronların kendileri sadece bir tür küçük, basit işlevdir. Şekil 10-1, tek bir nöronun neye benzediğini gösterir.

Giriş verilerinin soldan geldiğini ve sağda tek bir çıkış numarasının çıktığını görebilirsiniz (bazı nöron türleri birden fazla çıktı oluştursa da). Çıktının değeri, nöronun girdi verilerinin ve bazı parametrelerin (eğitim sırasında optimize edilen) bir fonksiyonudur. Giriş verilerini çıktıya dönüştürmek için her nöronun içinde iki adım gerçekleşir.

İlk olarak, nöronun girdilerinin ağırlıklı toplamı hesaplanır. Şekil 10-1'de, nörona giden her xi giriş numarası ilişkili bir ağırlık değeri olan wi ile çarpılır. Ortaya çıkan değerler, bir önyargı teriminin eklendiği (ağırlıklı bir toplam verecek şekilde) birbirine eklenir. Önyargı ve ağırlıklar, modeli optimize etmek için eğitim sırasında değiştirilen nöronun parametreleridir.

İkinci olarak, ağırlıklı toplam artı önyargı değerine bir aktivasyon fonksiyonu uygulanır. Bir aktivasyon fonksiyonunun amacı, nöronun girdi verilerinin doğrusal bir dönüşümü olan ağırlıklı toplama doğrusal olmayan bir dönüşüm uygulamaktır. Birçok yaygın etkinleştirme işlevi türü vardır ve bunlar oldukça basit olma eğilimindedir. Bir etkinleştirme işlevinin tek gerekliliği, farklılaştırılabilir olmasıdır, bu da parametreleri optimize etmek için geri yayılımı kullanmamızı sağlar (bu süreci kısaca "Sinir Ağlarının Eğitimi" sayfa 189'da tartışıyoruz).

Tablo 10-1, çeşitli diğer yaygın etkinleştirme işlevlerini gösterir ve hangilerinin hangi amaçlar için iyi olma eğiliminde olduğunu açıklar.

Rectified linear unit Doğrultulmuş doğrusal birim (ReLU), günümüzde kullanılan en yaygın etkinleştirme işlevidir ve basitçe maks (0, s). Örneğin, ağırlıklı toplam artı sapma değerinizin s olarak adlandırıldığını varsayalım. Eğer s sıfırın üzerindeyse, o zaman nöronunuzun çıktısı s olur ve eğer s sıfıra eşit veya sıfırın altındaysa, o zaman nöronunuzun çıktısı 0'dır. Bir ReLU nöronunun tüm fonksiyonunu basitçe maks (0, ağırlıklı girişler + önyargı) veya daha somut olarak, n giriş için aşağıdaki gibi:

Doğrusal olmayan aktivasyon fonksiyonları aslında bu tür nöronlardan oluşan ağların herhangi bir sürekli fonksiyona yaklaşabilmesinin temel nedenidir, bu da bu kadar güçlü olmalarının büyük bir nedenidir. Aşağıdaki bölümlerde, nöronların bir ağ oluşturmak için nasıl birbirine bağlandığını öğreneceksiniz ve daha sonra doğrusal olmayan aktivasyon fonksiyonlarının neden bu kadar önemli olduğunu anlayacaksınız.

\*A Network of Neurons:Nöron Ağı

Bir sinir ağı oluşturmak için, nöronları yönlendirilmiş bir grafikte (bir ağ), çok daha büyük bir işlev oluşturmak üzere bağlanan birkaç katmanla düzenlersiniz. Şekil 10 2, küçük bir sinir ağının bir örneğini göstermektedir.

Şekil 10-2: Verilerin bağlantılar yoluyla nörondan nörona aktarıldığı çok küçük, dört nöronlu bir sinir ağı örneği.

Şekil 10-2'de, orijinal girdilerimiz var: sol tarafta x1, x2 ve x3. Bu xi değerlerinin kopyaları, gizli katmandaki (çıktısı modelin son çıktısı olmayan bir nöron katmanı) her bir nörona bağlantılar boyunca gönderilir ve sonuçta her bir nörondan bir tane olmak üzere üç çıktı değeri elde edilir. Son olarak, bu üç nöronun her bir çıkışı, sinir ağının nihai sonucunu veren son bir nörona gönderilir.

Bir sinir ağındaki her bağlantı bir ağırlık parametresi ile ilişkilidir ve her nöron ayrıca bir önyargı parametresi, b (ağırlıklı toplama eklenir) içerir, bu nedenle temel bir sinir ağındaki optimize edilebilir parametrelerin toplam sayısı, bağlanan kenarların sayısıdır. bir nörona bir girdi, artı nöron sayısı. Örneğin, Şekil 10-2'de gösterilen ağda, toplamda 16 optimize edilebilir parametre sağlayan 4 toplam nöron artı 9 + 3 kenar vardır. Bu sadece bir örnek olduğu için, çok küçük bir sinir ağı kullanıyoruz - gerçek sinir ağları genellikle binlerce nörona ve milyonlarca bağlantıya sahiptir.

\*Universal Approximation Theorem:Evrensel Yaklaşım Teoremi

Sinir ağlarının çarpıcı bir yönü, evrensel tahmin ediciler olmalarıdır: yeterli nöron ve doğru ağırlık ve önyargı değerleri verildiğinde, bir sinir ağı temelde her tür davranışı taklit edebilir. Şekil 10-2'de gösterilen sinir ağı ileri beslemedir, bu da verilerin her zaman ileri doğru aktığı anlamına gelir (görüntüde soldan sağa).

Evrensel yaklaşım teoremi, evrensellik kavramını daha resmi olarak tanımlar. Doğrusal olmayan aktivasyon işlevlerine sahip tek bir gizli nöron katmanına sahip ileri beslemeli bir ağın, Rn.1'in kompakt bir alt kümesindeki herhangi bir sürekli işlevi yaklaştırabileceğini (keyfi olarak küçük bir hata ile) belirtir. Bu biraz ağız doludur, ancak bu sadece şu anlama gelir Yeterli nörona sahip bir sinir ağı, sınırlı sayıda girdi ve çıktıyla herhangi bir sürekli, sınırlı işlevi çok yakından tahmin edebilir.

1. Rn, tüm sayıların gerçek sayı olduğu n boyutlu bir Öklid uzayı olarak düşünülebilir. Örneğin, R2 (3.5, –5) gibi, 2 uzunluğundaki tüm olası gerçek değerleri temsil eder.

Başka bir deyişle, teorem yaklaştırmak istediğimiz fonksiyon ne olursa olsun, teorik olarak doğru parametrelere sahip bazı sinir ağlarının işi yapabileceğini belirtir. Örneğin, Şekil 10-3'teki gibi dalgalı, sürekli bir fonksiyon, f (x) çizerseniz, her olası x,f(x) ≈ network(x) girişi için, ne olursa olsun bazı sinir ağları vardır. f (x) işlevi ne kadar karmaşıktır. Bu, sinir ağlarının bu kadar güçlü olmasının bir nedenidir.

Şekil 10-3: Küçük bir sinir ağının korkak bir işleve funky function  nasıl yaklaştığına dair örnek. Nöron sayısı arttıkça, y ve ˆy arasındaki fark 0'a yaklaşacaktır.

Sonraki bölümlerde, doğru parametreler verildiğinde bu kadar farklı davranış türlerini nasıl ve neden modelleyebileceğimizi anlamanıza yardımcı olmak için elle basit bir sinir ağı oluşturacağız. Bunu sadece tek bir girdi ve çıktı kullanarak çok küçük bir ölçekte yapsak da, aynı ilke birden çok girdi ve çıktıyla ve inanılmaz derecede karmaşık davranışlarla uğraşırken de geçerlidir.

\*Building Your Own Neural Network:Kendi Sinir Ağınızı Kurmak

Bu evrenselliği iş başında görmek için kendi sinir ağımızı oluşturmayı deneyelim. Şekil 10-4'te gösterildiği gibi tek bir giriş x kullanarak iki ReLU nöronu ile başlıyoruz. Ardından, farklı işlevleri ve sonuçları modellemek için farklı ağırlık ve önyargı değerlerinin (parametrelerin) nasıl kullanılabileceğini görürüz.

Şekil 10-4: Giriş verileri ile beslenen iki nöronun görselleştirilmesi x

Burada, her iki nöronun ağırlığı 1'dir ve her ikisi de bir ReLU aktivasyon işlevi kullanır. İkisi arasındaki tek fark, nöron1'in –1'lik bir önyargı değeri uygularken, nöron2'nin –2'lik bir önyargı değeri uygulamasıdır. Nöron1'i birkaç farklı x değeri ile beslediğimizde ne olacağını görelim. Tablo 10-2 sonuçları özetlemektedir.

İlk sütun, x için bazı örnek girişleri gösterir ve ikincisi, ortaya çıkan ağırlıklı toplamı gösterir. Üçüncü sütun, önyargı parametresini ekler ve dördüncü sütun, belirli bir x girdisi için nöronun çıktısını elde etmek için ReLU aktivasyon fonksiyonunu uygular. Şekil 105, nöron1 fonksiyonunun grafiğini gösterir.

Şekil 10-5: Nöron1'in bir işlev olarak görselleştirilmesi. X ekseni, nöronun tek giriş değerini temsil eder ve y ekseni nöronun çıktısını temsil eder.

Nöron1'in –1'lik bir önyargısı olduğundan, nöron1'in çıktısı, ağırlıklı toplam 1'in üzerine çıkana kadar 0'da kalır ve ardından Şekil 10-5'te görebileceğiniz gibi, belirli bir eğimle yükselir. Bu 1 eğimi, 1'in wx → 1 ağırlık değeriyle ilişkilidir. 2 ağırlığında ne olacağını düşünün: Ağırlıklı toplam değer iki katına çıkacağından, Şekil 10-5'teki açı x = 0,5'te olacaktır x = 1 ve çizgi 1 yerine 2 eğimi ile yukarı çıkacaktır. Şimdi -2 önyargı değerine sahip nöron2'ye bakalım (bkz. Tablo 10-3).

Nöron2'nin sapması –2 olduğundan, Şekil 10-6'daki açı x = 1 yerine x = 2'de meydana gelir.

Şimdi, her ikisi de belirli bir süre boyunca hiçbir şey yapmayan, sonra 1 eğimle sonsuza kadar yükselen çok basit iki işlev (nöron) oluşturduk. ReLU nöronlarını kullandığımız için, her bir  \*!nöronun işlevinin eğimi onun tarafından etkilenir. ağırlıklar, eğim ve ağırlık terimleri eğimin başladığı yeri etkiler. Diğer etkinleştirme işlevlerini kullandığınızda, benzer kurallar geçerlidir. Parametreleri ayarlayarak, istediğimiz gibi her nöronun işlevinin açısını ve eğimini değiştirebildik.

Bununla birlikte, evrenselliği elde etmek için, nöronları bir araya getirmemiz gerekir, bu da daha karmaşık işlevlere yaklaşmamızı sağlar. Şekil 10-7'de gösterildiği gibi, iki nöronumuzu üçüncü bir nörona bağlayalım. Bu, nöron1 ve nöron2'den oluşan tek bir gizli katmana sahip küçük bir üç nöron ağı oluşturacaktır.

Şekil 10-7'de, giriş verisi x hem neuron1'e hem de neuron2'ye gönderilir. Daha sonra, nöron1 ve nöron2'nin çıktıları, ağın nihai çıktısını veren nöron3'e girdi olarak gönderilir.

Şekil 10-7'deki ağırlıkları incelerseniz, w1 → 3 ağırlığının 2 olduğunu ve nöron1'in nöron3'e katkısını ikiye katladığını göreceksiniz. Bu arada, w2 → 3 –1'dir ve nöron2'nin katkısını tersine çevirir. Özünde, neuron3, aktivasyon işlevini nöron1 \* 2 - nöron2'ye uygulamaktadır. Tablo 10-4, ortaya çıkan ağ için girdileri ve karşılık gelen çıktıları özetlemektedir.

İlk sütun, orijinal ağ girdisini (x) ve ardından nöron1 ve nöron2'nin sonuç çıktılarını gösterir. Sütunların geri kalanı, neuron3'ün çıktıları nasıl işlediğini gösterir: ağırlıklı toplam hesaplanır, önyargı eklenir ve son olarak, x için her orijinal giriş değeri için nöron ve ağ çıktılarını elde etmek için son sütunda ReLU aktivasyon işlevi uygulanır. Şekil 10-8, ağın işlev grafiğini gösterir.

Bu basit fonksiyonların birleşimiyle, Şekil 10 8'de yaptığımız gibi, farklı noktalarda istenen herhangi bir periyot veya eğim için yukarı çıkan bir grafik oluşturabileceğimizi görebiliriz. Başka bir deyişle, yapabilmeye çok daha yakınız. x girdimiz için herhangi bir sonlu fonksiyonu temsil etmek!

\*Adding Another Neuron to the Network:Ağa Başka Nöron Eklemek

186

Nöron ekleyerek ağımızın fonksiyon grafiğini nasıl yukarı (herhangi bir eğimle) yapacağımızı gördük, ancak grafiği nasıl aşağı indirebiliriz? Şekil 10-9'da gösterildiği gibi karışıma başka bir nöron (nöron4 neuron4) ekleyelim.

Şekil 10-9: Tek bir gizli katmana sahip küçük bir dört nöron ağının görselleştirilmesi

Şekil 10-9'da, giriş verileri x, nöron1, nöron2 ve nöron4'e gönderilir. Çıktıları daha sonra, ağın nihai çıktısını veren neuron3'e girdi olarak beslenir. Neuron4, nöron1 ve nöron2 ile aynıdır, ancak önyargısı -4'e ayarlanmıştır. Tablo 10-5, nöronun çıktısını özetlemektedir4.

Ağ grafiğimizin alçalmasını sağlamak için, nöron4'ü nöron3'e bağlayan ağırlığı -2'ye ayarlayarak nöron3'ün ağırlıklı toplamındaki nöron4'ün işlevini nöron1 ve nöron2'nin işlevinden çıkarıyoruz. Tablo 10-6, tüm ağın yeni çıktısını göstermektedir.

Şekil 10-10 bunun neye benzediğini gösterir.

Şekil 10-10: Dört nöron ağımızın görselleştirilmesi

Neyse ki, şimdi sinir ağı mimarisinin, sadece birkaç basit nöronu birleştirerek (evrensellik!) Çok daha karmaşık işlevler oluşturmak için daha fazla nöron eklemeye devam edebiliriz.

\*Automatic Feature Generation:Otomatik Özellik Oluşturma:

Tek bir gizli katmana sahip bir sinir ağının, yeterli nörona sahip herhangi bir sonlu işlevi yaklaşık olarak tahmin edebileceğini öğrendiniz. Bu oldukça güçlü bir fikir. Fakat birden fazla gizli nöron katmanına sahip olduğumuzda ne olur? Kısacası, sinir ağlarının belki de daha güçlü bir yönü olan otomatik özellik üretimi gerçekleşir.

  Tarihsel olarak, makine öğrenimi modelleri oluşturma sürecinin büyük bir kısmı özellik çıkarma idi. Bir HTML dosyası için, bir HTML dosyasının hangi sayısal yönlerinin (bölüm başlıklarının sayısı, benzersiz kelimelerin sayısı vb.) Modele yardımcı olabileceğine karar vermek için çok zaman harcanır.

Birden çok katmana ve otomatik özellik üretmeye sahip sinir ağları, bu işin çoğunu boşaltmamıza izin veriyor. Genel olarak, bir sinir ağına oldukça ham özellikler (bir HTML dosyasındaki karakterler veya kelimeler gibi) verirseniz, her bir nöron katmanı, bu ham özellikleri daha sonraki katmanlara girdi olarak işe yarayan yollarla temsil etmeyi öğrenebilir. Başka bir deyişle, bir sinir ağı bir HTML belgesinde a harfinin kaç kez göründüğünü saymayı öğrenecektir, eğer bu özellikle kötü amaçlı yazılımları tespit etmekle ilgiliyse, bir insanın öyle olup olmadığını söyleyen gerçek bir girdi olmadan.

  Görüntü işleme bisiklet örneğimizde, hiç kimse ağa kenarların veya tekerlek meta özelliklerinin yararlı olduğunu özellikle söylemedi. Model, bu özelliklerin eğitim sürecinde bir sonraki nöron katmanına girdi olarak yararlı olduğunu öğrendi. Özellikle yararlı olan, bu daha düşük seviyeli öğrenilen özelliklerin sonraki katmanlar tarafından farklı şekillerde kullanılabilmesidir; bu, derin sinir ağlarının, tek katmanlı bir ağın yapabileceğinden çok daha az nöron ve parametre kullanarak çok sayıda inanılmaz derecede karmaşık modeli tahmin edebileceği anlamına gelir.

Sinir ağları, daha önce çok zaman ve çaba gerektiren birçok özellik çıkarma işini gerçekleştirmekle kalmaz, bunu eğitim sürecinin rehberliğinde optimize edilmiş ve alanı verimli bir şekilde yapar.

\*Training Neural Networks:Sinir Ağlarının Eğitimi:

Şimdiye kadar, çok sayıda nöron ve doğru ağırlıklar ve önyargı terimleri verildiğinde, bir sinir ağının karmaşık işlevleri nasıl yaklaşık olarak tahmin edebileceğini keşfettik. Şimdiye kadarki tüm örneklerimizde, bu ağırlık ve önyargı parametrelerini manuel olarak belirledik. Bununla birlikte, gerçek sinir ağları normalde binlerce nöron ve milyonlarca parametre içerdiğinden, bu değerleri optimize etmek için verimli bir yola ihtiyacımız var.

Normalde, bir modeli eğitirken, bir eğitim veri kümesiyle ve bir grup optimize edilmemiş (rastgele başlatılmış) parametre içeren bir ağla başlarız. Eğitim, hedef bir işlevi en aza indirmek için parametrelerin optimize edilmesini gerektirir. Modelimizi "iyi huylu" için 0 ve "kötü amaçlı yazılım" için 1 gibi bir etiketi tahmin edebilecek şekilde eğitmeye çalıştığımız denetimli öğrenmede, bu hedef işlev, eğitim sırasında ağın tahmin hatasıyla ilişkili olacaktır. Belirli bir x girdisi için (örneğin, belirli bir HTML dosyası), bu, doğru olduğunu bildiğimiz y etiketi (örneğin, "kötü amaçlı yazılımdır" için 1.0) ile mevcut ağdan aldığımız çıktı ˆy arasındaki farktır (örneğin örnek 0.7). Hatayı, tahmin edilen ˆy etiketi ile bilinen, gerçek etiket y arasındaki fark olarak düşünebilirsiniz; burada ağ (x) = ˆy ve ağ, f (x) = y gibi bilinmeyen bir f fonksiyonuna yaklaşmaya çalışıyor. . Başka bir deyişle, ağ = ˆf.

Eğitim ağlarının arkasındaki temel fikir, bir ağa eğitim veri kümenizden bir gözlem (x) beslemek, biraz çıktı almak, ˆy ve ardından parametrelerinizi değiştirmenin hedefinize nasıl yaklaşacağını bulmaktır, y. Çeşitli düğmeleri olan bir uzay gemisinde olduğunuzu hayal edin. Her bir düğmenin ne yaptığını bilmiyorsunuz, ancak girmek istediğiniz yönü biliyorsunuz (y). Problemi çözmek için gaza basıp gittiğiniz yönü not edin (ˆy). Ardından, bir düğmeyi çok az çevirip gaza tekrar basıyorsunuz. Birinci ve ikinci yönleriniz arasındaki fark, bu düğmenin yönünüzü ne kadar etkilediğini size söyler. Bu şekilde, uzay gemisini nasıl uçuracağınızı sonunda oldukça iyi anlayabilirsiniz.

Bir sinir ağını eğitmek de benzerdir. İlk olarak, bir ağa eğitim veri kümenizden bir gözlem (x) beslersiniz ve bazı çıktılar alırsınız, ˆy. Bu adıma ileri yayılma denir çünkü son çıktınızı ˆy elde etmek için x girdinizi ağ üzerinden ileri beslersiniz. Daha sonra, her parametrenin çıktınızı nasıl etkileyeceğini belirlersiniz ˆy. Örneğin, ağınızın çıkışı 0,7 ise, ancak doğru çıktının 1'e yakın olması gerektiğini biliyorsanız, bir parametreyi (w) biraz artırmayı deneyebilirsiniz, ˆy'nin y'ye yaklaşıp yaklaşmadığını veya uzaklaşıp uzaklaşmadığını ve ne kadar.2 Buna, ˆy'nin w'ye göre kısmi türevi veya ˆy w denir. Ağın tamamındaki parametreler daha sonra, ˆy'nin y'ye biraz daha yaklaşmasına (ve dolayısıyla ağın f'ye daha yakın olmasına) neden olan bir yönde çok az sürüklenir. Eğer ˆy w pozitifse, o zaman w'yi küçük bir miktar artırmanız gerektiğini bilirsiniz (özellikle y ˆy w ile orantılı olarak), böylece yeni ˆy'niz 0.7'den biraz uzaklaşacak ve 1 (y ). Başka bir deyişle, ağınıza, bilinen etiketlerle eğitim verilerindeki hatalarını düzelterek bilinmeyen f işlevine yaklaşmayı öğretirsiniz. Bu kısmi türevleri yinelemeli olarak hesaplama, parametreleri güncelleme ve ardından tekrarlama işlemine gradyan inişi denir. Bununla birlikte, binlerce nöron, milyonlarca parametre ve genellikle milyonlarca eğitim gözlemi ağıyla, tüm bu analiz çok fazla hesaplama gerektirir. Bunu aşmak için, bu hesaplamaları hesaplama açısından uygun kılan, geri yayılım adı verilen düzgün bir algoritma kullanıyoruz. Özünde, geri yayılım, bir sinir ağı gibi hesaplama grafikleri boyunca kısmi türevleri verimli bir şekilde hesaplamamıza izin verir!

\*Using Backpropagation to Optimize a Neural Network:Bir Sinir Ağını Optimize Etmek İçin Geri Yayımlamayı Kullanma

Bu bölümde, geri yayılmanın nasıl çalıştığını göstermek için basit bir sinir ağı oluşturuyoruz. Değeri x = 2 ve ilişkili gerçek etiketi y = 10 olan bir eğitim örneğimiz olduğunu varsayalım. Genellikle, x birçok değerden oluşan bir dizi olabilir, ancak işleri basitleştirmek için tek bir değere bağlı kalalım. Bu değerleri yerine koyduğumuzda, Şekil 10-11'de ağımızın giriş x değeri 2 ile 5'lik bir ˆy değeri çıkardığını görebiliriz.

Şekil 10-11: Üç nöron ağımızın x = 2 girdisiyle görselleştirilmesi

Parametrelerimizi, x = 2 verildiğinde ağımızın ˆy çıktısının bilinen y değerimiz olan 10'a yaklaşması için dürtmek için, w1 → 3'ün nihai çıktımızı ˆy nasıl etkilediğini hesaplamamız gerekir. W1 → 3'ü biraz artırdığımızda ne olacağını görelim (diyelim, 0,01). Nöron3'teki ağırlıklı toplam 1.01 \* 2 + (1 \* 3) olur ve nihai çıktıyı 5'ten 5.02'ye değiştirerek 0.02'lik bir artışa neden olur. Başka bir deyişle, w1 → 3'e göre ˆy'nin kısmi türevi 2'dir, çünkü w1 → 3'ü değiştirmek, ˆy'deki değişimin iki katı verir. Y 10 olduğundan ve mevcut çıktımız ˆy (mevcut parametre değerlerimiz ve x = 2 verildiğinde) 5 olduğundan, y'yi 10'a yaklaştırmak için w1 → 3'ü küçük bir miktar artırmamız gerektiğini artık biliyoruz.

Bu oldukça basit. Ancak, sadece son katmandaki bir nörondaki parametreleri değil, ağımızdaki tüm parametreleri hangi yöne iteceğimizi bilmemiz gerekir. Örneğin, wx → 1 ne olacak? ˆY wx 1'i hesaplamak daha karmaşıktır çünkü sadece dolaylı olarak ˆy'yi etkiler. İlk olarak, nöron3'ün işlevine nöron1'in çıktısından nasıl etkilendiğini soruyoruz. Nöron1'in çıktısını 2'den 2,01'e değiştirirsek, nöronun 3 nihai çıkışı 5'ten 5,01'e değişir, yani nöron 1 1. Wx → 1'in ˆy'yi ne kadar etkilediğini bilmek için, sadece ∂ˆy ∂neuron1'i, nöron1'in çıktısını ne kadar wx → 1 etkilediğiyle çarpmalıyız. Wx → 1'i 1'den 1.01'e değiştirirsek, nöron1'in çıktısı 2'den 2.02'ye değişir, yani nöron 1 1 wx 2'dir. Bu nedenle:

Zincir kuralını kullandığımızı fark etmiş olabilirsiniz. 3 Başka bir deyişle, bir ağın derinliklerinde wx → 1 gibi bir parametrenin nihai çıktımızı ˆy nasıl etkilediğini anlamak için, kısmi türevleri, bizim aramızdaki yol boyunca her noktada çarpıyoruz. parametre wx → 1 ve ˆy. Bu, eğer wx → 1 çıktıları on başka nörona beslenen bir nörona beslenirse, wx → 1’in ˆy üzerindeki etkisinin hesaplanması, wx → 1'den ˆy'ye giden tüm yolların toplanmasını gerektireceği anlamına gelir. Şekil 10-12, numune ağırlığı parametresi wx → 2 tarafından etkilenen yolları görselleştirmektedir.

Bu ağdaki gizli katmanların tam olarak bağlantılı katmanlar olmadığına dikkat edin, bu da ikinci gizli katmanın alt nöronunun neden vurgulanmadığını açıklamaya yardımcı olur.

\*Path Explosion:Yol Patlaması

Peki ağımız daha da genişlediğinde ne olur? Düşük seviyeli bir parametrenin kısmi türevini hesaplamak için eklememiz gereken yol sayısı katlanarak artar. Çıktısı 1.000 nöronluk bir katmana beslenen, çıktıları 1.000 nörona beslenen ve çıktıları daha sonra nihai bir çıkış nöronuna beslenen bir nöronu düşünün. Bu bir milyon yolla sonuçlanır! Şans eseri, her bir yolu gözden geçirmek ve sonra onları y parametresini elde etmek için toplamak gerekli değildir. Geri yayılımın işe yaradığı yer burasıdır. Nihai çıktılarımıza (ˆy) götüren her bir yolda yürümek yerine, kısmi türevler, yukarıdan aşağıya veya geriye doğru başlayarak katman katman hesaplanır. Son bölümdeki zincir kuralı mantığını kullanarak, herhangi bir kısmi türevi y w hesaplayabiliriz; burada w, katman – 1'den bir çıktıyı katmantaki bir nöroniye bağlayan bir parametredir, tüm neuroni + 1 için aşağıdakileri toplayarak, burada her nöroni + 1, nöroninin (w'nin nöronu) bağlı olduğu katman + 1'deki bir nörondur:

Bu katman katman yukarıdan aşağıya doğru yaparak, her katmanda türevleri konsolide ederek yol patlamasını sınırlıyoruz. Başka bir deyişle, en üst düzey katman + 1'de hesaplanan türevler (ˆy i nöron 1 gibi), katmanlarda türevleri hesaplamaya yardımcı olmak için kaydedilir. Daha sonra katman – 1'deki türevleri hesaplamak için, katmanlardan kaydedilen türevleri kullanırız (∂ˆy ∂ i nöron gibi). Daha sonra katman – 2, katman – 1'den türevleri kullanır ve bu böyle devam eder. Bu numara, tekrar etmemiz gereken hesaplama miktarını büyük ölçüde azaltır ve sinir ağlarını hızlı bir şekilde eğitmemize yardımcı olur.

\*Vanishing Gradient:Ufuk Gradyan

Çok derin sinir ağlarının karşılaştığı bir sorun, yok olan gradyan problemidir. On katmanı olan bir sinir ağının ilk katmanındaki bir ağırlık parametresini düşünün. Geri yayılımdan aldığı sinyal, bu ağırlığın nöronundan nihai çıkışına kadar tüm yolların sinyallerinin toplamıdır. Sorun şu ki, her yolun sinyalinin inanılmaz derecede küçük olması, çünkü bu sinyali, tümü 1'den küçük sayılar olma eğiliminde olan on nöron derinliğindeki yol boyunca her noktada kısmi türevleri çarparak hesaplıyoruz. Bu, düşük olduğu anlamına gelir. -seviye nöronun parametreleri, çok sayıda çok küçük sayının toplamına göre güncellenir ve bunların çoğu birbirini iptal eder. Sonuç olarak, bir ağın daha düşük katmanlardaki parametrelere güçlü bir sinyal göndermesini koordine etmesi zor olabilir. Daha fazla katman ekledikçe bu sorun katlanarak daha da kötüleşir. Aşağıdaki bölümde öğrendiğiniz gibi, belirli ağ tasarımları bu yaygın sorunu aşmaya çalışır.

\*Types of Neural Networks:Sinir Ağı Türleri

Basitlik uğruna, size şimdiye kadar gösterdiğim her örnek, ileri beslemeli sinir ağı adı verilen bir ağ türü kullanıyor. Gerçekte, farklı problem sınıfları için kullanabileceğiniz başka birçok yararlı ağ yapısı vardır. En yaygın sinir ağları sınıflarından bazılarını ve bunların siber güvenlik bağlamında nasıl uygulanabileceğini tartışalım.

\*Feed-Forward Neural Network:İleri Beslemeli Sinir Ağı

En basit (ve ilk) tür sinir ağı, ileri beslemeli bir sinir ağı, aksesuarsız bir Barbie bebek gibidir: diğer sinir ağları türleri genellikle bu "varsayılan" yapının varyasyonlarından ibarettir. İleri beslemeli  feedforward architecture mimari kulağa tanıdık gelmelidir: nöron yığınlarından oluşur. Her nöron katmanı, bir sonraki katmandaki bazı veya tüm nöronlara bağlıdır, ancak bağlantılar asla geri gitmez veya döngü oluşturmaz, dolayısıyla "ileri besleme feed forward." adı verilir. İleri beslemeli sinir ağlarında, var olan her bağlantı, i katmanındaki bir nöronu (veya orijinal girişi) j> i katmanındaki bir nörona bağlar. Katman i'deki her bir nöronun, i + 1 katmanındaki her nörona bağlanması gerekmez, ancak tüm bağlantılar, önceki katmanları sonraki katmanlara bağlayarak ileriye doğru beslenmelidir. İleri beslemeli ağlar, mevcut sorun üzerinde özellikle iyi çalışan başka bir mimariyi (görüntü tanıma için evrişimli sinir ağları gibi) zaten bilmiyorsanız, genellikle ilk önce bir soruna attığınız türden ağlardır.

\*Convolutional Neural Network:Evrişimli Sinir Ağı

Evrişimli sinir ağı (CNN), her bir nöronu besleyen girişin, giriş alanı üzerinde kayan bir pencere tarafından tanımlandığı evrişimli katmanlar içerir. Yalnızca pencereden görünen piksellerin bir sonraki katmandaki belirli bir nörona bağlanacağı daha büyük bir resmin üzerinde kayan küçük kare bir pencere hayal edin. Daha sonra pencere kayar ve yeni piksel seti yeni bir nörona bağlanır. Şekil 10-13 bunu göstermektedir. Bu ağların yapısı, yerelleştirilmiş özellik öğrenmeyi teşvik eder. Örneğin, bir ağın alt katmanlarının, bir görüntü boyunca rastgele dağılmış pikseller arasındaki ilişkiye odaklanmaktan ziyade, bir görüntüdeki (kenarları, şekilleri vb. Oluşturan) yakın pikseller arasındaki ilişkiye odaklanması daha yararlıdır. çok demek). Kayar pencereler, bu odağı açıkça zorlar, bu da yerel özellik çıkarmanın özellikle önemli olduğu alanlarda öğrenmeyi geliştirir ve hızlandırır.

Şekil 10-13: 2 × 2 çıktı vermek için 1 adımda (adım boyutu) 3 × 3 giriş alanı üzerinde kayan 2 × 2 evrişimli pencerenin görselleştirilmesi

Girdi verilerinin yerelleştirilmiş bölümlerine odaklanma yetenekleri nedeniyle, evrişimli sinir ağları görüntü tanıma ve sınıflandırmada son derece etkilidir. Ayrıca, siber güvenlik açısından etkileri olan belirli doğal dil işleme türleri için etkili oldukları da gösterilmiştir.

Her bir evrişimsel pencerenin değerleri, evrişimli bir katmandaki belirli nöronlara beslendikten sonra, bu nöronların çıktıları üzerinde kayan bir pencere yeniden kaydırılır, ancak bunlar, her bir girişle ilişkili ağırlıklarla standart nöronlara (örneğin, ReLU'lar) beslenmek yerine, bunlar Ağırlıksız (yani, 1'de sabitlenmiş) ve maksimum (veya benzer) aktivasyon işlevi olmayan nöronlara yeniden beslenir. Diğer bir deyişle, evrişimli katmanın çıktıları üzerinde küçük bir pencere kaydırılır ve her pencerenin maksimum değeri alınır ve bir sonraki katmana aktarılır. Buna havuz katmanı denir. Katmanları bir araya toplamanın amacı, verileri (genellikle bir görüntüyü) “uzaklaştırmak”, böylece en önemli bilgileri korurken daha hızlı hesaplama için özelliklerin boyutunu azaltmaktır.

Evrişimli sinir ağları, bir veya daha fazla evrişimli ve havuz katman kümesine sahip olabilir. Standart bir mimari, bir evrişimli katman, bir havuzlama katmanı, ardından başka bir dizi evrişimsel ve havuzlama katmanları ve son olarak ileri beslemeli ağlarda olduğu gibi birkaç tam bağlantılı katman içerebilir. Bu mimarinin amacı, bu son tam bağlantılı katmanların girdi olarak oldukça yüksek seviyeli özellikler alması (tek tekerlekli bisiklet üzerinde düşünmek) ve sonuç olarak karmaşık verileri (görüntüler gibi) doğru bir şekilde sınıflandırabilmesidir.

\*Autoencoder Neural Network:Otomatik kodlayıcı Sinir Ağı

Otomatik kodlayıcı, orijinal eğitim girdisi ile sıkıştırılmış çıktı arasında minimum farkla bir girdiyi sıkıştırmaya ve ardından sıkıştırmayı açmaya çalışan bir tür sinir ağıdır. Bir otomatik kodlayıcının amacı, bir veri kümesi için verimli bir gösterimi öğrenmektir. Başka bir deyişle, otomatik kodlayıcılar, optimize edilmiş kayıplı sıkıştırma programları gibi davranır, burada giriş verilerini daha küçük bir temsilde sıkıştırır ve ardından orijinal giriş boyutuna geri döndürür.

Belirli bir x girdisi için bilinen etiketler (y) ve tahmin edilen etiketler (ˆy) arasındaki farkı en aza indirerek sinir ağı optimizasyon parametreleri yerine, ağ, orijinal girdi x ve yeniden yapılandırılmış çıktı ˆx arasındaki farkı en aza indirmeye çalışır.

Yapısal olarak, otomatik kodlayıcılar, Şekil 10-14'te gösterildiği gibi, orta katmanların erken ve geç aşama katmanlarından daha az nöron içermesi dışında, genellikle standart ileri beslemeli sinir ağlarına çok benzer.

Şekil 10-14: Bir otomatik kodlayıcı ağının görselleştirilmesi

Gördüğünüz gibi, orta katman, her biri aynı boyutta olan en soldaki (giriş) ve en sağdaki (çıktı) katmanlardan çok daha küçüktür. Son katman her zaman orijinal girdilerle aynı sayıda çıktı içermelidir, böylece her eğitim girdisi xi, sıkıştırılmış ve yeniden yapılandırılmış kuzeni ˆxi ile karşılaştırılabilir. Bir otomatik kodlayıcı ağı eğitildikten sonra, farklı amaçlar için kullanılabilir. Otomatik kodlayıcı ağları, basitçe verimli sıkıştırma / açma programları olarak kullanılabilir. Örneğin, görüntü dosyalarını sıkıştırmak için eğitilen otomatik kodlayıcılar, JPEG ile aynı boyutta sıkıştırılan aynı görüntüden çok daha net görünen görüntüler oluşturabilir.

\*Generative Adversarial Network:Üretken Tartışmalı Ağ

Üretken bir düşmanlık ağı (GAN), ilgili görevlerinde kendilerini geliştirmek için birbirleriyle rekabet eden iki sinir ağından oluşan bir sistemdir. Tipik olarak, üretici ağ rastgele gürültüden sahte örnekler (örneğin, bir tür görüntü) oluşturmaya çalışır. Daha sonra ikinci bir ayırt edici ağ, gerçek örnekler ile sahte, üretilen örnekler arasındaki farkı söylemeye çalışır (örneğin, bir yatak odasının gerçek görüntüleri ile oluşturulan görüntüler arasında ayrım yapma). Bir GAN'daki her iki sinir ağı, geri yayılımla optimize edilmiştir. Jeneratör ağı, belirli bir turda diskriminatör ağını ne kadar iyi kandırdığına göre parametrelerini optimize ederken, ayırıcı ağ, üretilen ve gerçek örnekleri ne kadar doğru bir şekilde ayırt edebildiğine bağlı olarak parametrelerini optimize eder. Başka bir deyişle, kayıp fonksiyonları birbirlerinin tam tersidir. GAN'lar, gerçek görünümlü veriler oluşturmak veya düşük kaliteli veya bozuk verileri geliştirmek için kullanılabilir.

\*Recurrent Neural Network:Tekrarlayan Sinir Ağı

Tekrarlayan ağlar (RRN'ler), nöronlar arasındaki bağlantıların aktivasyon işlevleri zaman adımlarına bağlı olan yönlendirilmiş döngüler oluşturduğu nispeten geniş bir sinir ağları sınıfıdır. Bu, ağın veri dizilerindeki kalıpları öğrenmesine yardımcı olan bir bellek geliştirmesine izin verir. RNN'lerde, girdiler, çıktılar veya hem girdi hem de çıktı bir çeşit zaman serisidir.

RNN'ler, bağlantılı el yazısı tanıma, konuşma tanıma, dil çevirisi ve zaman serisi analizi gibi veri sırasının önemli olduğu görevler için mükemmeldir. Siber güvenlik bağlamında, ağ trafiği analizi, davranışsal algılama ve statik dosya analizi gibi sorunlarla ilgilidir. Program kodu, bu sırayla doğal dile benzer olduğu için, bir zaman serisi olarak ele alınabilir.

RNN'lerle ilgili bir sorun, kaybolan gradyan probleminden dolayı, bir RNN'ye eklenen her zaman adımının, ileri beslemeli bir sinir ağındaki tüm ekstra katmana benzer olmasıdır. Geri yayılım sırasında, kaybolan gradyan problemi, daha düşük seviyeli katmanlardaki (veya bu durumda, daha önceki zaman adımlarındaki) sinyallerin inanılmaz derecede zayıf olmasına neden olur.

Uzun kısa süreli bellek  long short-term memory (LSTM) ağı, bu sorunu çözmek için tasarlanmış özel bir RNN türüdür. LSTM'ler, hangi bilgilerin hatırlanacağına ve hangi bilgilerin unutulacağına karar vermeye çalışan bellek hücreleri ve özel nöronlar içerir. Çoğu bilginin atılması, kaybolan gradyan sorununu büyük ölçüde sınırlar çünkü yol patlamasını azaltır.

\*ResNet:

ResNet (artık ağın kısaltması), ağın erken / sığ katmanlarındaki nöronlar arasında, bir veya daha fazla ara katmanı atlayarak daha derin katmanlara atlama bağlantıları oluşturan bir tür sinir ağıdır. Burada artık kelimesi, bu ağların sayısal bilgiyi, Tablo 10-1'de gösterdiğimiz türden aktivasyon fonksiyonlarından geçmek zorunda kalmadan, sayısal bilgiyi doğrudan katmanlar arasında geçirmeyi öğrendikleri gerçeğini ifade eder.

Bu yapı, kaybolan gradyan sorununu büyük ölçüde azaltmaya yardımcı olur ve bu da ResNets'in inanılmaz derecede derin, bazen 100'den fazla katman olmasını sağlar.

Çok derin sinir ağları, giriş verilerindeki son derece karmaşık, garip ilişkileri modellemede mükemmeldir. ResNets çok fazla katmana sahip olabildikleri için, özellikle karmaşık problemler için uygundurlar. İleri beslemeli sinir ağları gibi, ResNets de çok özel sorun alanlarındaki uzmanlıklarından ziyade karmaşık sorunları çözmedeki genel etkinlikleri nedeniyle daha önemlidir.

-> ÖZET:

Bu bölümde, nöronların yapısını ve sinir ağları oluşturmak için nasıl birbirine bağlandıklarını öğrendiniz. Ayrıca, bu ağların geri yayılım yoluyla nasıl eğitildiğini araştırdınız ve evrensellik, otomatik özellik üretimi ve kaybolan gradyan sorunu gibi sinir ağlarının sahip olduğu bazı faydaları ve sorunları keşfettiniz. Son olarak, birkaç yaygın sinir ağı türünün yapılarını ve faydalarını öğrendiniz.

Sonraki bölümde, Python’un Keras paketini kullanarak kötü amaçlı yazılımları tespit etmek için aslında sinir ağları oluşturacaksınız.