

Recurrent Neural Network (RNN)

HAZIRLAYANLAR:

Mert SARIKAYA (212923012)

Yunus Emre SELCUK (212923032)

Onur BARBAROS (212923036)

Emirhan GÜNGÖR (212923059)


Efe Ata AKAN (212923016)

AKADEMİSYEN:

DR. ÖĞR. ÜYESİ KEREM GENCER

İÇERİK:

1. Recurrent Neural Network (RNN) Nedir ?
2. Normal Sinir Ağından Farkı Ne?
3. Yapay Zeka vs. Sinir Ağları
4. Popüler Nöral Ağlar
5. Feedforward Neural Network
6. Neden RNN?
7. Uygulama Alanları
8. RNN Nasıl Çalışır?
9. RNN Türleri
10. Vanishing Gradient Problem
11. Exploding Gradient Problem
12. LSTM



Recurrent Neural Networks

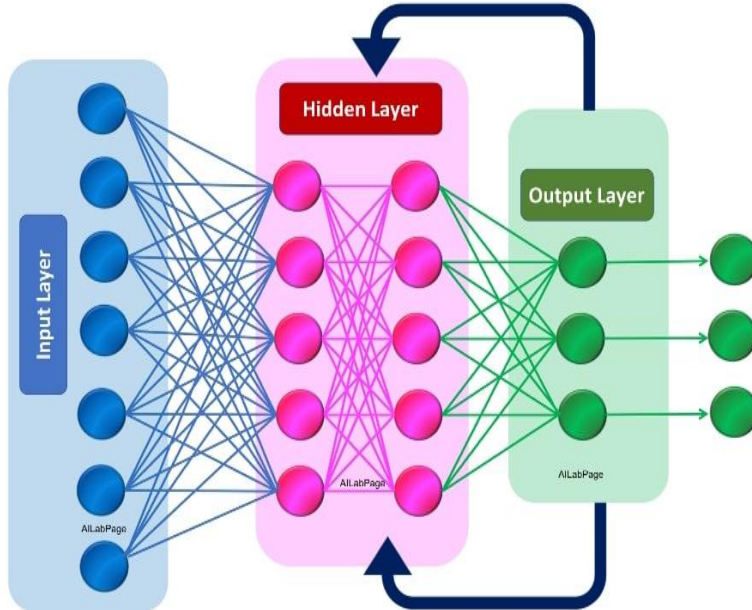
(RNN) Nedir?

Normal sinir ağlarına göre çok daha sağlıklı ve güvenilir sonuçlar elde etmemizi sağlayan Recurrent Neural Networks 'ün (RNN) amacı hata payını düşürmektir. Bu sunumumuzda Recurrent Neural Networks (RNN) kavramını ele aldık.

Beynimiz bir konu hakkındaki bilgiyi tam olarak kavrayabilmesi ve bilgiye karşı doğru bir yanıt oluşturabilmesi için o konu hakkındaki eskiden öğrenilen bilgiler ile yeni öğrenilmiş bilgileri kullanır. Yani beyin eski bilgilerle yeni öğrenilen bilgileri anlamlaştırarak bir bütün oluşturur. Eski bilgiler tecrübe yoluyla elde edilir. Eğer eski bilgiler yanlışsa, eski bilgiler unutulmaya zorlanır.

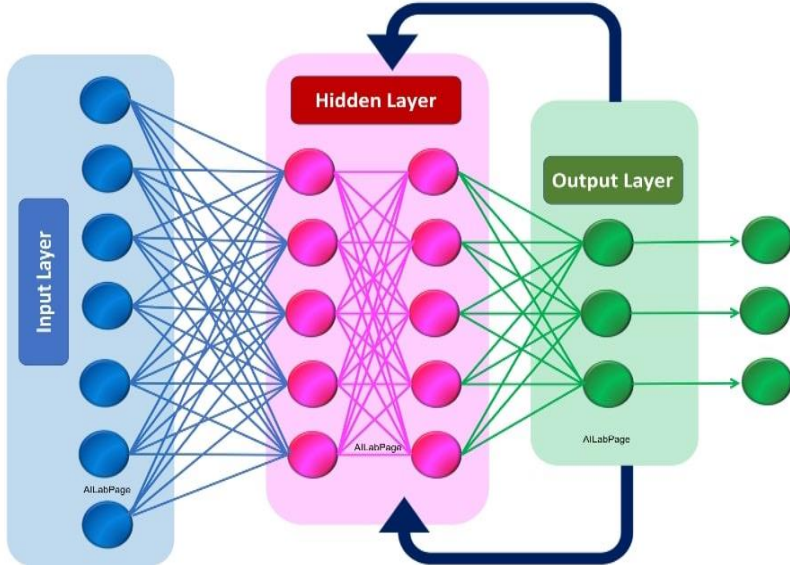


Recurrent Neural Networks



Recurrent Neural Network (RNN), bu olayı esas alır. RNN yapısında, sinir ağlarına gelen bilgi katmanlardaki belirli ağırlık sabitleri ile ilişkilendirilerek sonuçta bir tahmin oluşturulur. Bu tahmin gerçek veriler ile karşılaştırıldığında bir hata payı oluşursa ağırlık sabitleri değiştirilerek yeniden tahmin oluşturulur. Bu sinir ağında asıl amaç hata payını düşürmektir. RNN temel olarak dört katmandan oluşabilir.

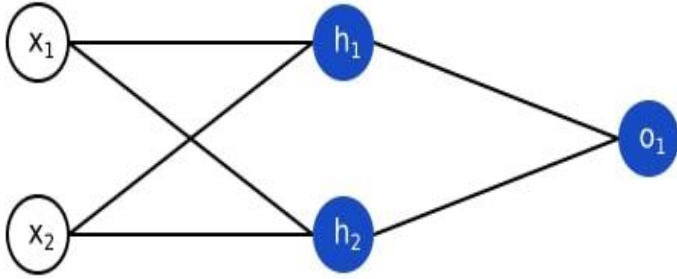
Recurrent Neural Networks



Görselde, basit bir RNN yapısı görülmektedir. Bu yapı 1 giriş katmanına, 2 gizli katmana ve 1 çıkış katmanına sahiptir. Yinelemeli (Recurrent) denmesinin sebebi çıkışta ölçülen hata payının azaltılması için elde edilen tahminin tekrar işleme sokulmasından dolayıdır. Bunun sayesinde sistemdeki hata payının azaltılması amaçlanır.

Normal Sinir Ağından Farkı Ne?

Input Layer Hidden Layer Output Layer



Temel Bir Sinir Ağı Yapısı

Basit bir sinir ağı bir fotoğraf içerisindeki nesneleri tanımak için sadece o anda verilen giriş bilgilerini kullanır. Sisteme daha önce verilen bilgileri kullanmaz. O bilgileri kullanmadan bir tahmin oluşturur. Böylece hedefe gitmek amaçlanır.

Normal Sinir Ağından Farkı Ne?



RNN ise yalnızca o an verilen bilgileri değil, sisteme daha önce tanıtılan bilgileri ve ayrıca sisteme daha sonra yüklenecek bilgileri de kullanır. Böylece bu sinir ağları bir belleğe sahip olan ağlardır. Eski veriler ile yeni verileri karşılaştırarak sonuca ulaşırlar.

Yinelemeli sinir ağları, doğal dil işleme, anomali tespiti, kelime ve cümle tahmini ve borsa tahmini gibi alanlarda kullanılır.



Dezavantajları

Yinelemeli Sinir Ağları, zaman içerisinde çok farklı tekrar yapabilir. Bu tekrar denemeleri yüzünden veriyle ilişkili olan bazı parametreleri veriyi az etkilediği gerekçesiyle sistemden atabilir. Eğer bu parametreler sinir ağına çok önce eklenmişse artık kullanıcı bu bilgiye ulaşmayabilir. Atılan ve unutulan bu bilgileri bir bellekte toplayabilmek için geliştirilmiş mimariler bulunmaktadır.

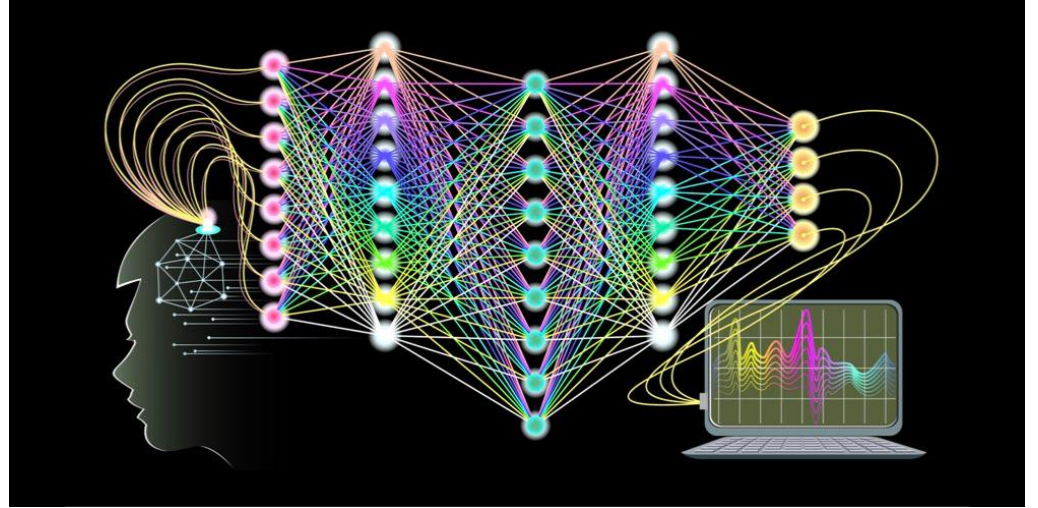


Sinir Ağları Nedir?

Bir sinir ağının yapısı, makine öğreniminin bir alt alanı olan derin öğrenme olarak bilinen bir eğitim süreci aracılığıyla çalışır. Bu, nöro-dilbilimsel programlamaya uygulanan yapay zeka (AI) teknolojisidir.

Sinir Ağları Nedir?

Temel olarak bir sinir ağı, insan beyninin bilgiyi işlerken nasıl çalıştığının modellenmesidir. Eğitim yoluyla, veriler arasındaki ilişkileri ve örüntüleri modellemeyi, karmaşık sorunlara çözüm sağlamayı ve doğal dil işlemeyi simüle etmeyi öğrenen ve sürekli olarak geliştiren uyarlanabilir bir sistemdir.



Bu ağı, birbirine bağlı ve katmanlar halinde organize edilmiş nöronlar veya düğümler içerir. Giriş verilerini alan bir giriş katmanı; verileri yayan bir veya daha fazla gizli katman ve analizin veya NLP'nin sonucunu gönderen bir çıkış katmanı vardır.

Yapay Zeka ve Sinir Ağları

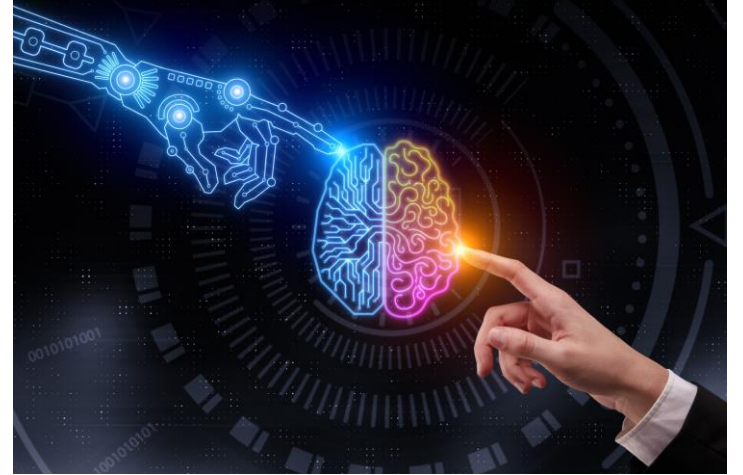
Makine öğrenimine gelince, hem YSA hem de makine öğrenimi modelleri çıktığı tahmin edebilir veya sınıflandırabilir. Tek fark, bir makine öğrenimi modelinin eğitim verilerine dayalı kararlar almasıdır. Bu nedenle, hem denetimli hem de denetimsiz makine öğrenimi modelleri insan denetimi gerektirir. Bir sinir ağı, ilk aşamalarda daha az insan müdahalesi gerektirir ve manuel çaba gerektirmeden doğru kararlar üretebilir.



Yapay zeka, YSA'lar da dahil olmak üzere bir dizi akıllı teknolojiyi kapsayan şemsiye bir terimdir. Sinir ağları ise derin öğrenmeye zemin hazırlayan ve büyük miktarda veriyi işleyebilen makine öğreniminin bir alt alanıdır. Derin öğrenme ve sinir ağları genellikle birbirlerinin yerine kullanılır.

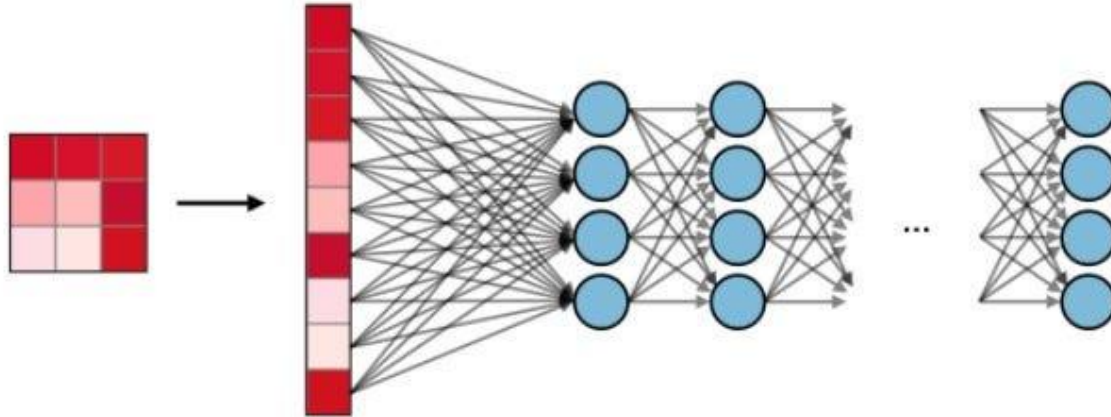
Teknik açıdan bakıldığında, makine öğrenimi modelleri tek bir veri giriş katmanından beslenir. Sinir ağı modelleri ise tam tersine birden fazla katmandan oluşur.

Genel olarak, yapay zeka, makine öğrenimi ve sinir ağları, her biri diğeri için bir temel sağlayan tamamlayıcı teknolojilerdir. Bu nedenle, makine öğrenimi fikirleri genellikle yüz tanıma ve nesne algılama gibi derin öğrenme fikirleriyle örtüşür.

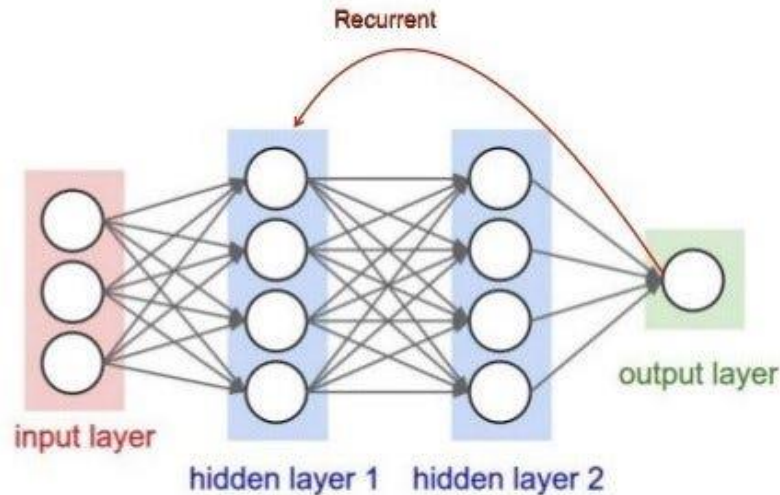


Popüler nöral ağlar

Convolutional Neural Networks (CNN): Görüntü işleme ve bilgisayarla görü uygulamalarında yaygın olarak kullanılır. Özellikle nesne tanıma ve görüntü sınıflandırma gibi görevlerde etkilidir.

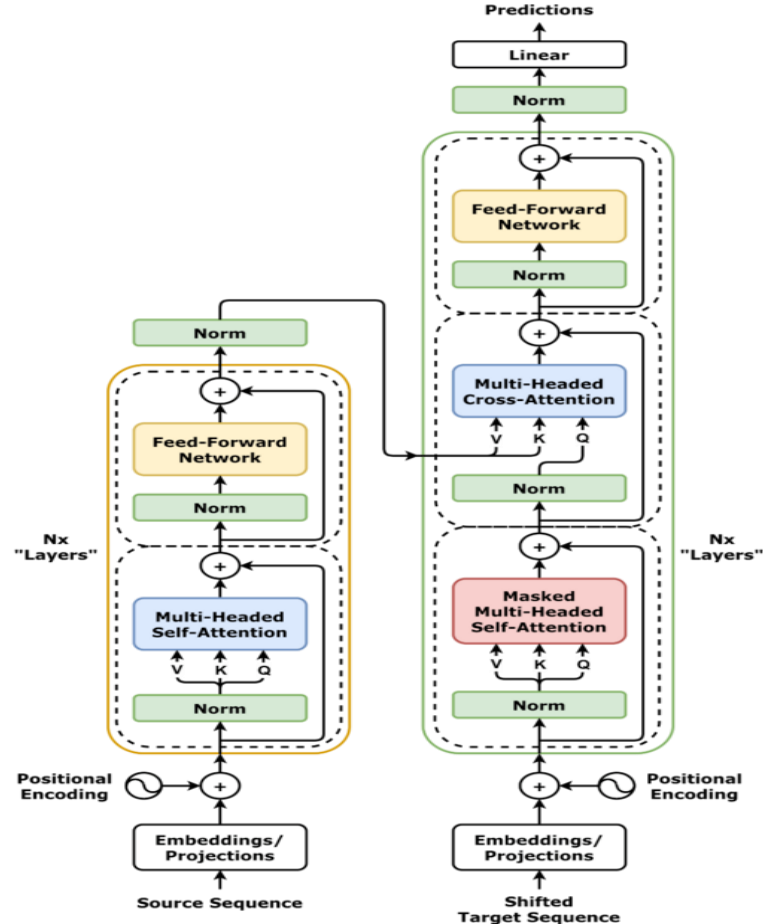


Recurrent Neural Networks (RNN): Zaman serisi verileri ve sıralı veriler için uygundur. Doğal dil işleme gibi alanlarda kullanılır, ancak LSTM (Uzun Kısa Süreli Bellek) ve GRU (Kapı Kontrollü Birimi) gibi gelişmiş varyantlarıyla daha da yaygın hale gelmiştir.

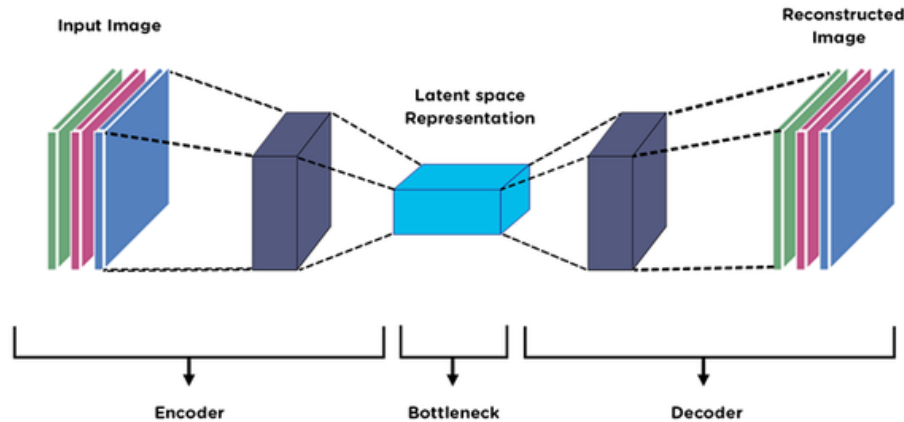


Transformers:

Doğal dil işleme alanında devrim yaratan bir mimaridir. BERT ve GPT gibi modeller, bağlamı daha iyi anlamak için dikkat mekanizmalarını kullanır.

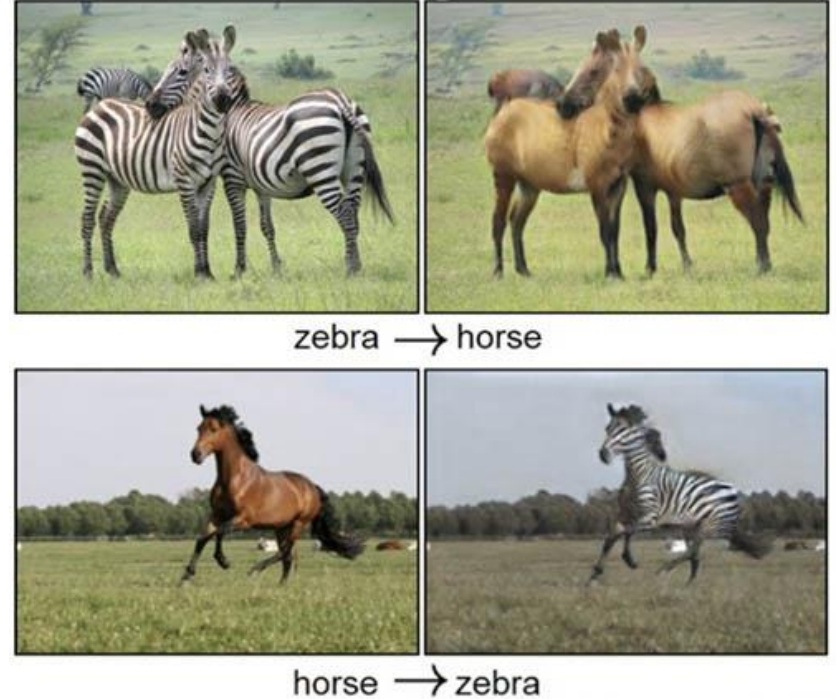


Otomatik Kodlayıcılar (Autoencoders): Veri sıkıştırma ve özellik çıkarımı için kullanılır. Anomalileri tespit etmek veya veri gürültüsünü azaltmak için de etkili olabilir.



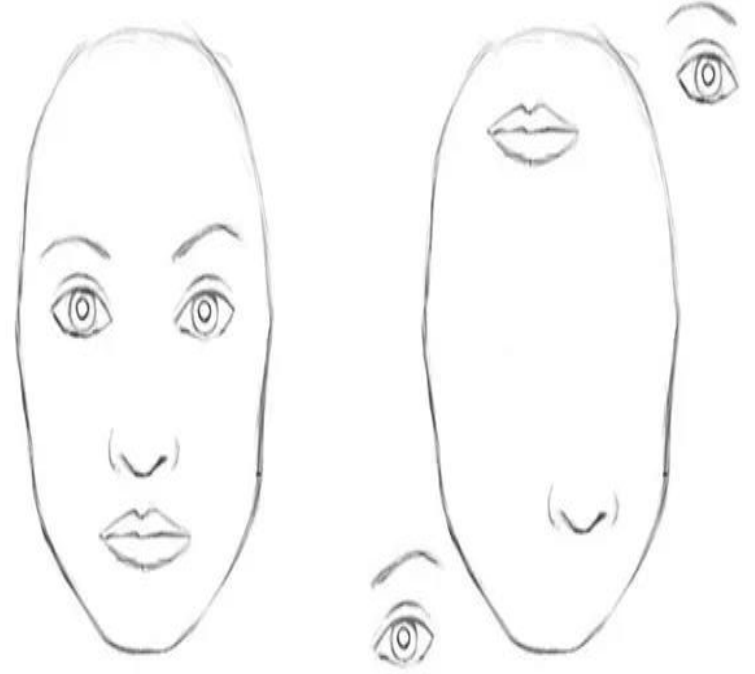
Generative Adversarial Network(GAN):

Gerçekçi veri üretimi için kullanılan bir yapıdır. Görüntü, ses ve metin gibi çeşitli veri türlerinde yeni örnekler oluşturabilir.



Dönüşümlü Sinir Ağları (Capsule Networks):

Görüntü verilerinde nesne hiyerarşisini anlamak için geliştirilmiş bir yapı. Özellikle görüntü tanıma görevlerinde bazı avantajlar sunabilir.





Feedforward Neural Network

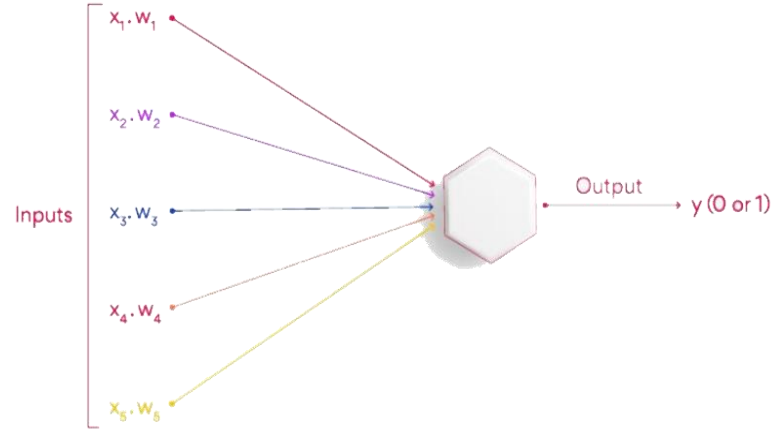
(İleri Beslemeli Sinir Ağları) Nedir?

Feedforward neural network, düğümlerin döngüler oluşturmadığı yapay sinir ağlardır . Bu tür sinir ağları, tüm bilgilerin yalnızca ileri iletildiği için çok katmanlı sinir ağı olarak da bilinir.

Veri akışı sırasında, giriş düğümleri gizli katmanlardan geçen verileri alır ve çıkış düğümlerinden çıkar. Ağda, çıkış düğümünden bilgi göndererek kullanılabilecek hiçbir bağlantı yoktur.

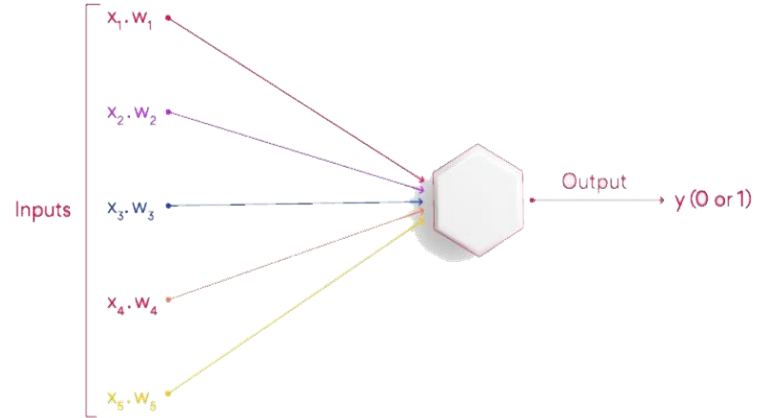
Feedforward neural network nasıl çalışır ?

FNN basitleştirildiğinde tek katmanlı bir algılayıcı olarak görülebilir.



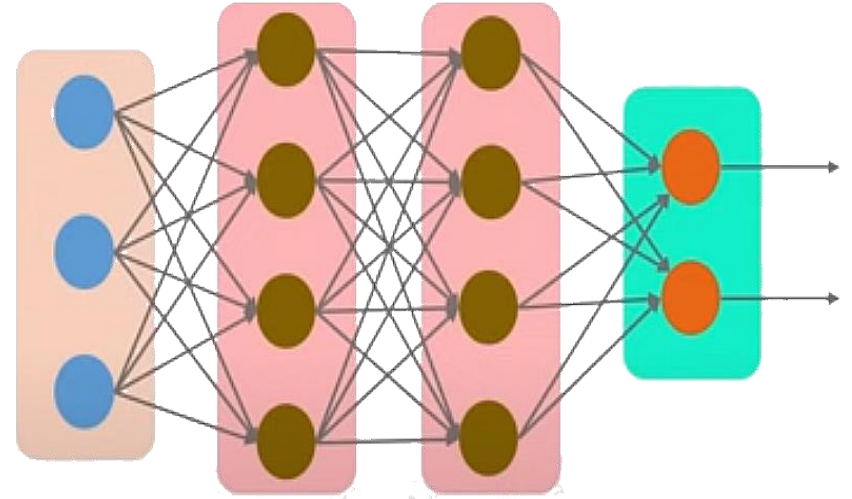
FNN nasıl çalışır ?

Feedforward bir sinir ağı modeli olarak, tek katmanlı algılayıcı genellikle sınıflandırma için kullanılır. Makine öğrenimi de tek katmanlı algılayıcılara entegre edilebilir.

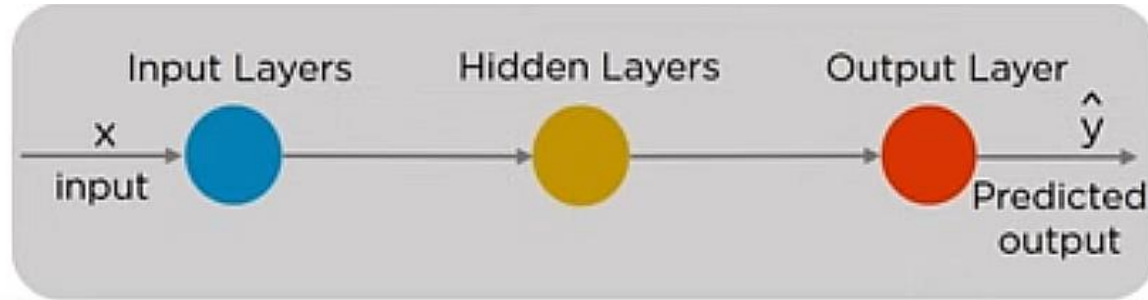


FNN nasıl çalışır ?

Feed-Forward Ağda, bilgi yalnızca ileri yönde akar, girdi düğümlerinden başlayarak (varsa) gizli katmanlar üzerinden çıkış düğümlerine kadar ilerler.



FNN nasıl çalışır ?



- Kararlar mevcut girdiye dayanır.
- Geçmiş hakkında hafıza yok.
- Gelecek için herhangi bir imkan yok.

FNN Avantajları

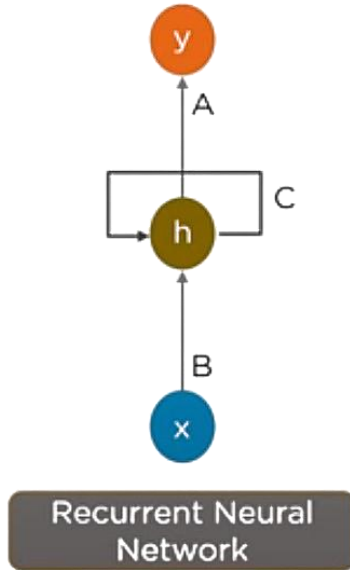
- İleri beslemeli sinir ağlarının basitleştirilmiş mimarisiyle makine öğrenimi güçlendirilebilir.
- Karmaşık görevler ağda birden fazla nörona ihtiyaç duyar.
- Yapay sinir ağları karar sınırlarının karmaşık problemiyle ilgilenir.

Feedforward Neural Network Sorunları



- 01 Sıralı veriyi işleyemez
- 02 Yalnızca mevcut girdiyi dikkate alır
- 03 Önceki girdileri hatırlayamaz

Feedforward Neural Network Çözümler



- 01 Sıralı veriyi işleyebilir
- 02 Mevcut girdiyi ve daha önce alınan girdileri dikkate alır
- 03 İçsel hafızası sayesinde önceki girdileri hatırlayabilir

Neden RNN ?



Sıralı Verilerle Baş Etme: RNN'ler, verilerin sırayla işlendiği durumlarda daha etkili çalışır çünkü önceki adımlardaki bilgileri dikkate alarak kararlar verebilir. Bu, önceki bilgilerin gelecekteki çıktılara olan etkisini yakalama yeteneği sağlar.

Neden RNN ?



Dahili Hafıza: RNN'lerin en önemli özelliklerinden biri, geçmişte işlenmiş girdileri "hatırlayabilmeleridir." Bu, önceki girdilerden gelen bilgileri tutma ve sonraki adımlarda kullanma yeteneği ile sağlanır. Bu sayede, veri setindeki her bir adımın geçmişle bağlantısını kurarak daha tutarlı ve ilgili sonuçlar elde edebilir.

Neden RNN ?



Zaman Bağımlılıkları: RNN'ler, zaman içinde değişen veya gelişen verilerle çalışmada avantajlıdır çünkü geçmiş adımlardaki değişikliklerin mevcut duruma nasıl etki edeceğini öğrenir. Bu, zaman içinde veri noktaları arasındaki ilişkileri anlamak ve bu ilişkileri kullanarak daha iyi sonuçlar elde etmek açısından önemlidir.

RNN Uygulama Alanları

Image Captioning



Bir köpek havada bir top yakalıyor.

RNN, görüntüdeki aktiviteleri analiz ederek bir görüntüyü altyazılamak için kullanılır

RNN Uygulama Alanları

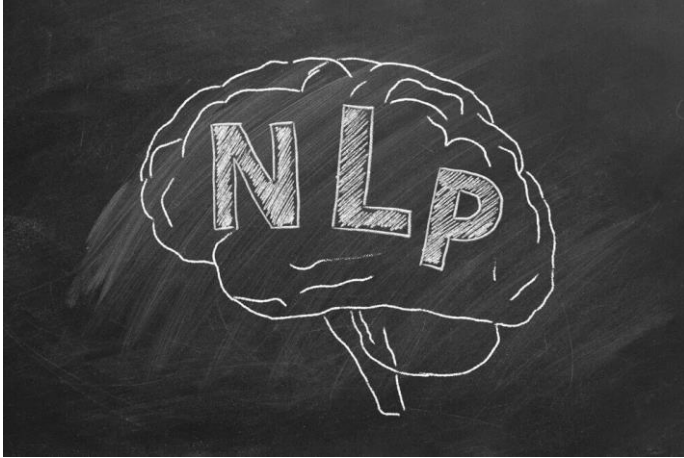
Time Series Prediction



RNN kullanılarak belirli bir ayda hisse senedi fiyatlarının tahmini gibi herhangi bir zaman serisi problemi çözülebilir

RNN Uygulama Alanları

NLP



Doğal Dil İşleme için RNN kullanılarak Metin Madenciliği ve Duygu Analizi yapılabilir

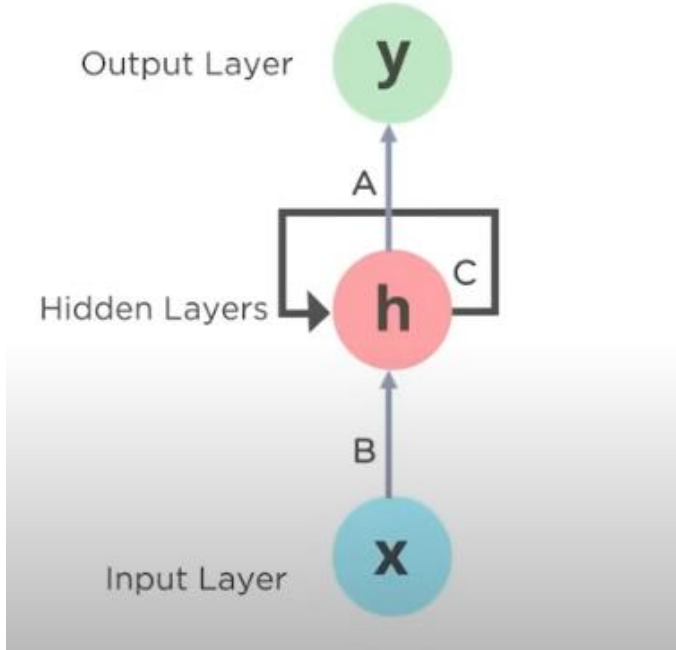
RNN Uygulama Alanları

Machine Translation



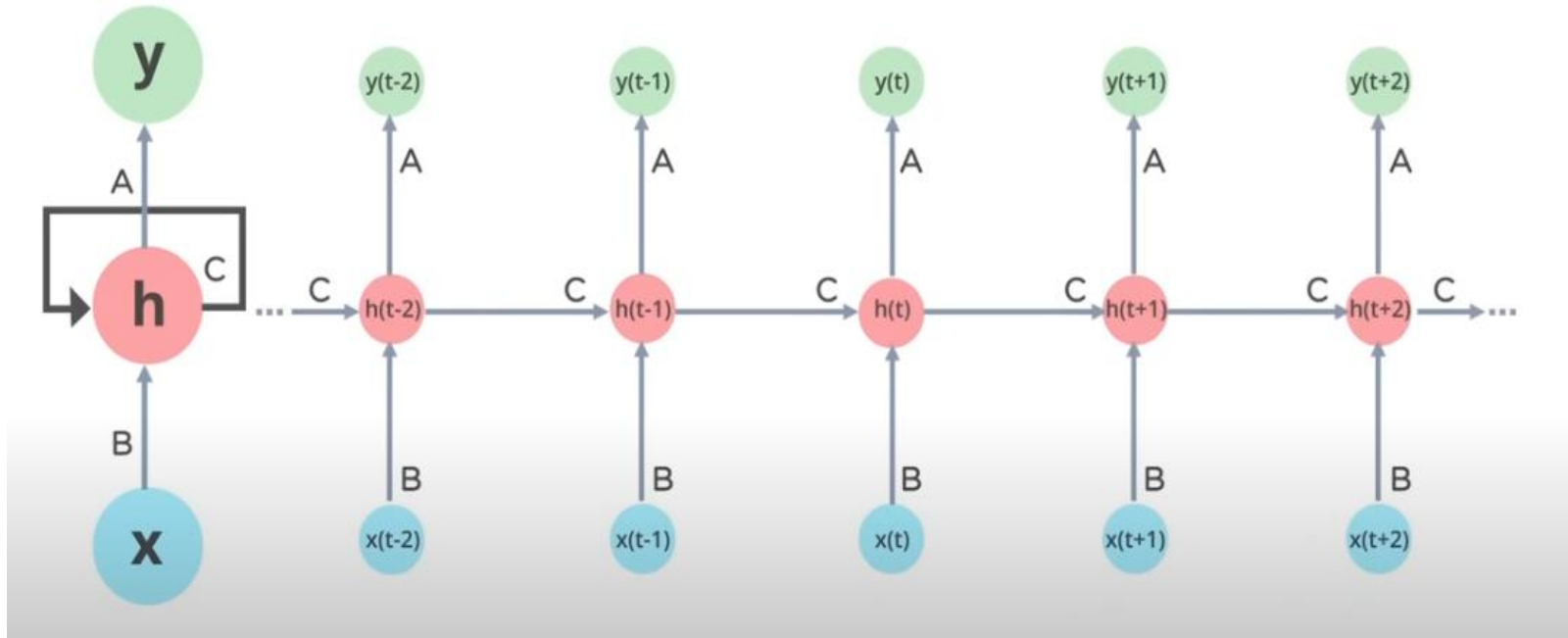
Bir dilde verilen bir girdi, RNN kullanılarak farklı dillerde çıktıya çevrilebilir.

RNN nasıl çalışır ?



Tekrarlayan sinir ağının (RNN) ne olduğuna bir göz atalım. Tekrarlayan sinir ağı (RNN), bir katmanın çıktısını kaydetme ve bunu katmanın çıktısını tahmin etmek için girişe geri besleme prensibiyle çalışır.

RNN nasıl çalışır ?



RNN nasıl çalışır ?



RNN'ler, karmaşık görevleri gerçekleştirmek için birlikte çalışan veri işleme düğümleri olan nöronlardan oluşur. Nöronlar girdi, çıktı ve gizli katmanlar olarak düzenlenmiştir. Giriş katmanı işlenecek bilgileri alır, çıktı katmanı ise sonucu sağlar. Veri işleme, analiz ve tahmin gizli katmanda gerçekleşir.

RNN nasıl çalışır?



RNN'ler, aldıkları sıralı verileri gizli katmanlara her seferinde bir adım ileterek çalışır. Bununla birlikte, kendi kendini döngüleyen veya *tekrarlayan* bir iş akışına da sahiptirler; gizli katman, kısa vadeli bir bellek bileşeninde gelecekteki tahminler için önceki girdileri hatırlayabilir ve kullanabilir. Bir sonraki diziyi tahmin etmek için mevcut girişi ve depolanan belleği kullanır.

RNN nasıl çalışır ?



Şu sıraya göz atın: Elma kırmızıdır. RNN'nin Elma giriş dizisini aldığı anda kırmızıyı tahmin etmesini istiyorsunuz. Gizli katman Elma kelimesini işlediğinde bir kopyasını hafızasında saklar. Daha sonra dır kelimeyi gördüğünde Elma kelimesini hafızasından hatırlar ve tam diziyi anlar: Elma x'tir. Daha sonra daha iyi doğruluk için kırmızı kelimesini tahmin edebilir. Bu, RNN'leri konuşma tanıma, makine çevirisi ve diğer dil modelleme görevlerinde faydalı kılar.

RNN nasıl çalışır ?



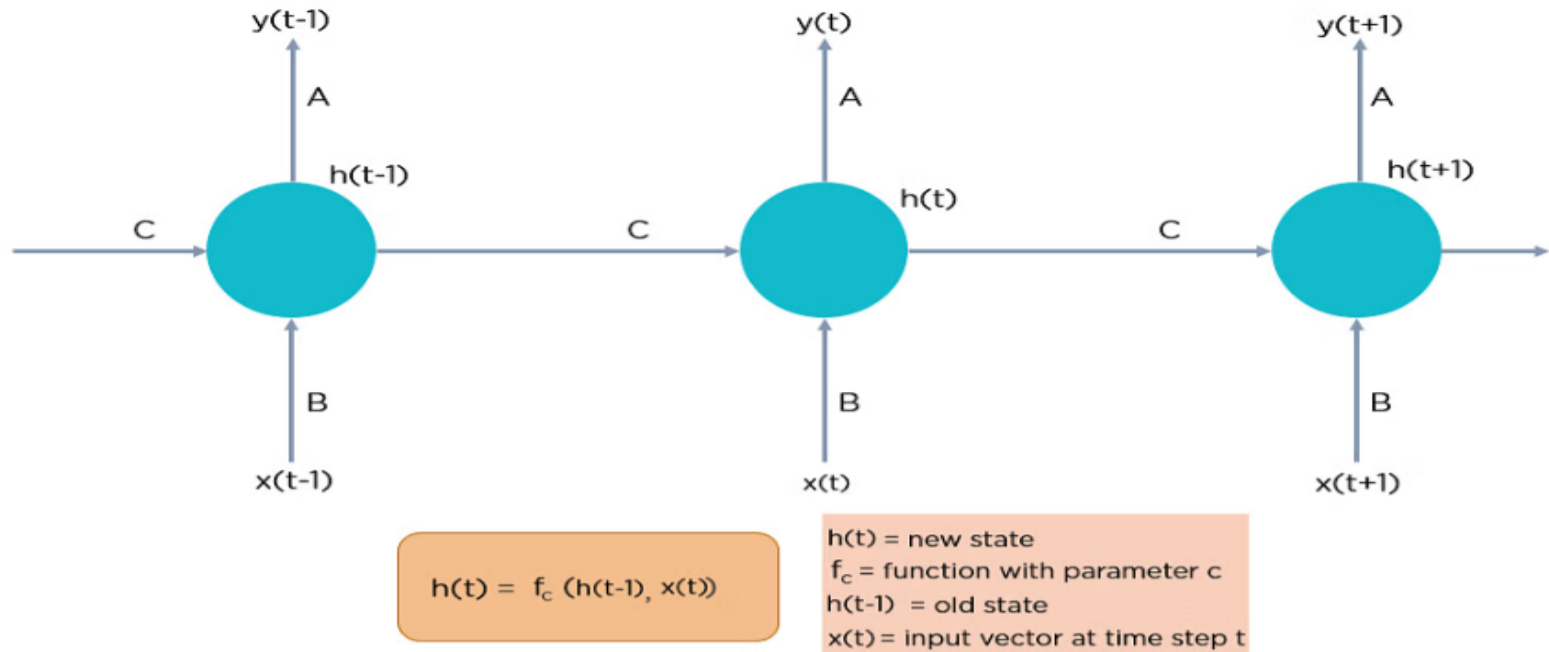
Makine öğrenimi (Machine Learning) (ML) mühendisleri, modeli eğitim verileriyle besleyerek ve performansını iyileştirerek RNN'ler gibi derin sinir ağlarını eğitir. ML'de nöronun ağırlıkları, eğitim sırasında öğrenilen bilgilerin çıktığı tahmin ederken ne kadar etkili olduğunu belirleyen sinyallerdir. Bir RNN'deki her katman aynı ağırlığı paylaşır.

RNN nasıl çalışır ?



ML mühendisleri, tahmin doğruluğunu artırmak için ağırlıkları ayarlar. Model hatasını hesaplamak ve ağırlığını buna göre ayarlamak için zaman içinde geri yayılma (BPTT) adı verilen bir teknik kullanırlar. BPTT, çıktıyı önceki zaman adımına geri döndürür ve hata oranını yeniden hesaplar. Bu şekilde, dizideki hangi gizli durumun önemli bir hataya neden olduğunu belirleyebilir ve hata payını azaltmak için ağırlığı yeniden ayarlayabilir.

RNN nasıl çalışır ?



RNN nasıl çalışır ?



Burada, "x" giriş katmanıdır, "h" gizli katmandır ve "y" çıkış katmanıdır. A, B ve C, modelin çıktısını iyileştirmek için kullanılan ağ parametreleridir. Herhangi bir t zamanında, geçerli girdi $x(t)$ ve $x(t-1)$ 'deki girdinin bir kombinasyonudur. Herhangi bir zamandaki çıktı, çıktıyı iyileştirmek için ağa geri getirilir.

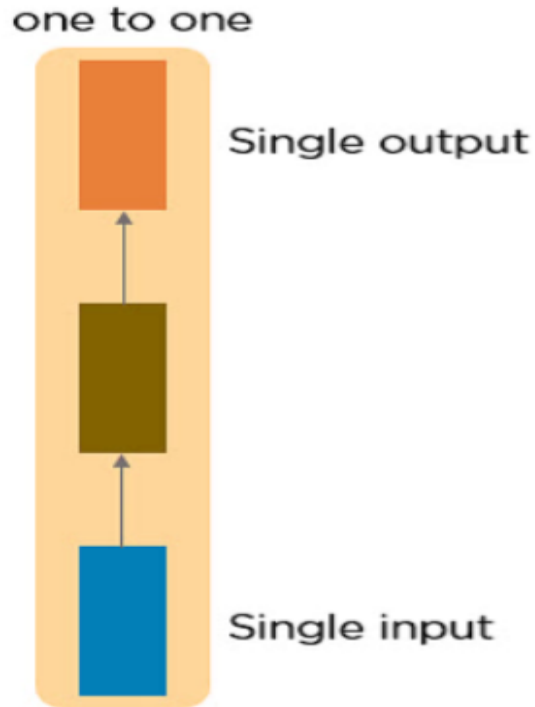
RNN Türleri



Dört tip RNN vardır:

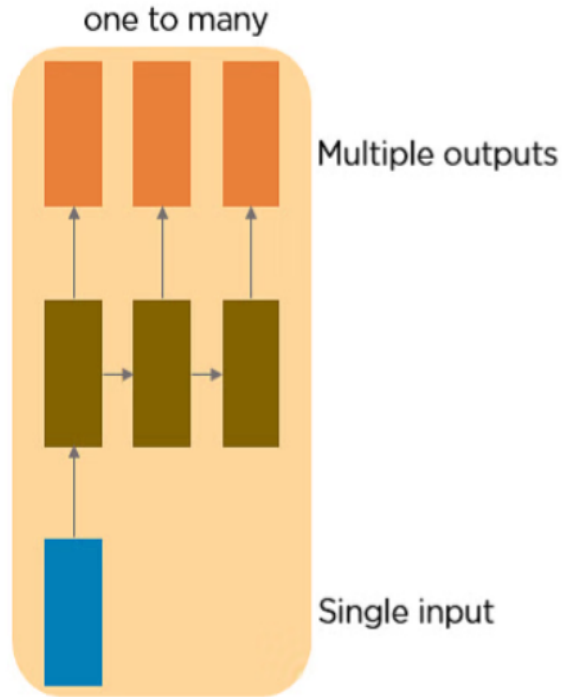
1. One to One (Bire Bir)
2. One to Many (Birden Çoğa)
3. Many to One (Çoktan Bire)
4. Many to Many (Çoktan Çoğa)

One to One RNN



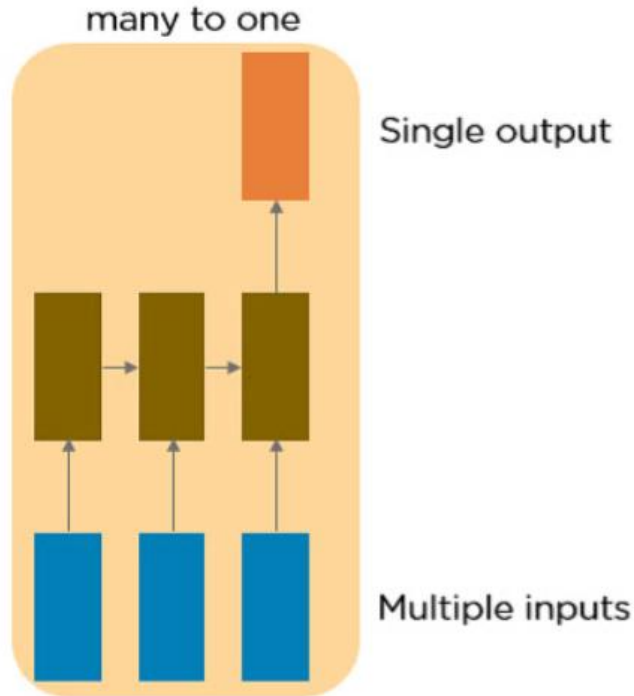
Bu tür sinir ağları “Vanilya Neural Network” olarak bilinir. Tek bir girdi ve tek bir çıktısı olan genel makine öğrenmesi sorunları için kullanılır.

One to Many RNN



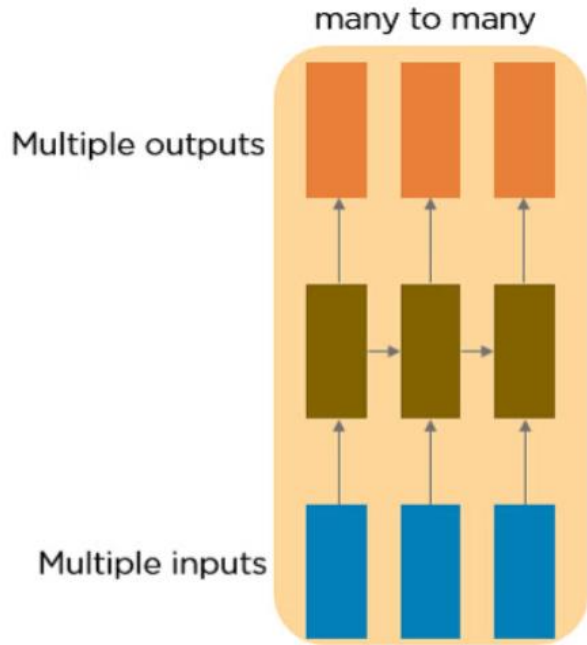
Bu tür sinir ağının tek bir girişi ve birden fazla çıkışı vardır. Bunun bir örneği image captioning 'dir.

Many to One RNN



Bu RNN bir dizi girdi alır ve tek bir çıktı üretir. Duygu analizi, verilen bir cümlenin olumlu veya olumsuz duyguları ifade ettiği şeklinde sınıflandırılabileceği bu tür ağların iyi bir örneğidir.

Many to Many RNN



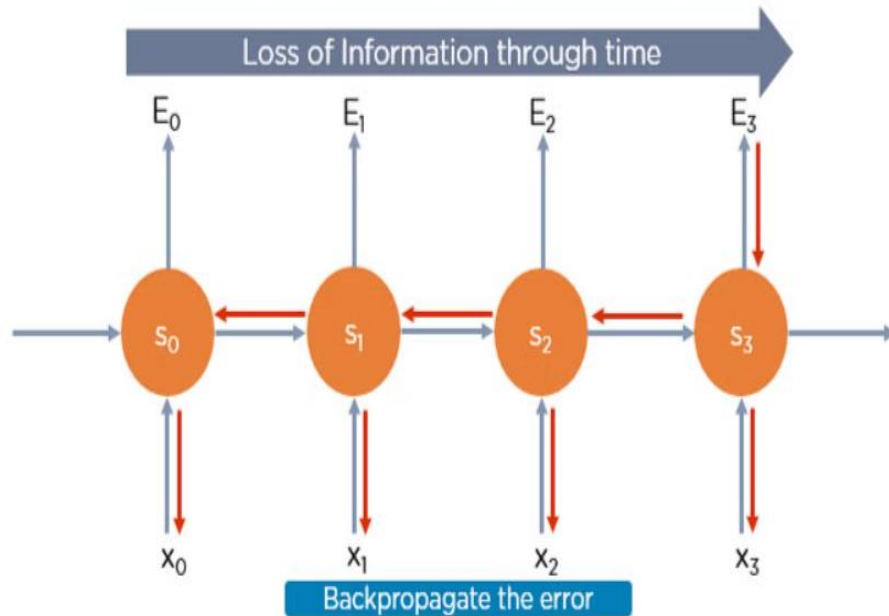
Bu RNN bir dizi girdi alır ve bir dizi çıktı üretir. Makine çevirisi örneklerden biridir.

Vanishing Gradient Problem (Kaybolan Gradyan Problemi)

RNN'ler, borsa tahmini, makine çevirisi ve metin üretimi gibi zamana bağlı ve ardışık veri problemlerini modellemenize olanak tanır. Ancak, Vanishing Gradient nedeniyle RNN'nin eğitilmesinin zor olduğunu göreceksiniz.

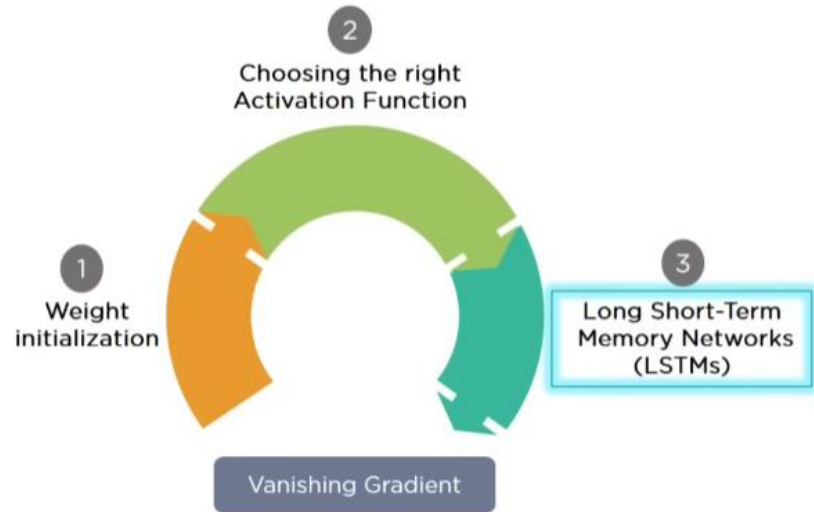
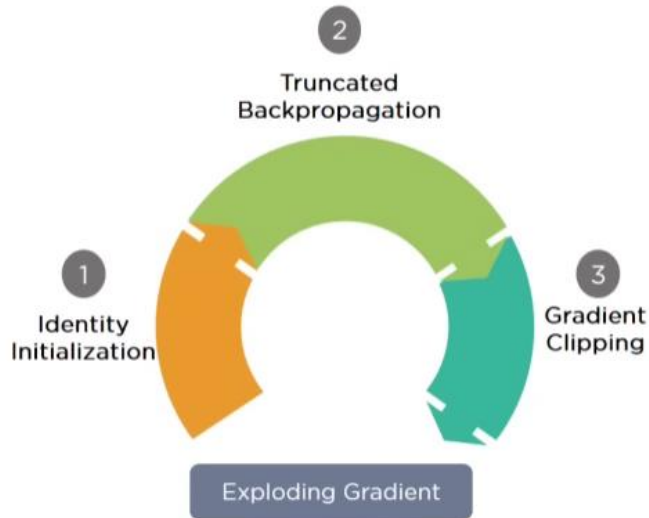
RNN'ler kaybolan gradyanlar sorunuyla karşı karşıyadır. Gradyanlar RNN'de kullanılan bilgileri taşır ve gradyan çok küçük olduğunda, parametre güncellemeleri önemsiz hale gelir. Bu, uzun veri dizilerinin öğrenilmesini zorlaştırır.

Vanishing Gradient Problem (Kaybolan Gradyan Problemi)

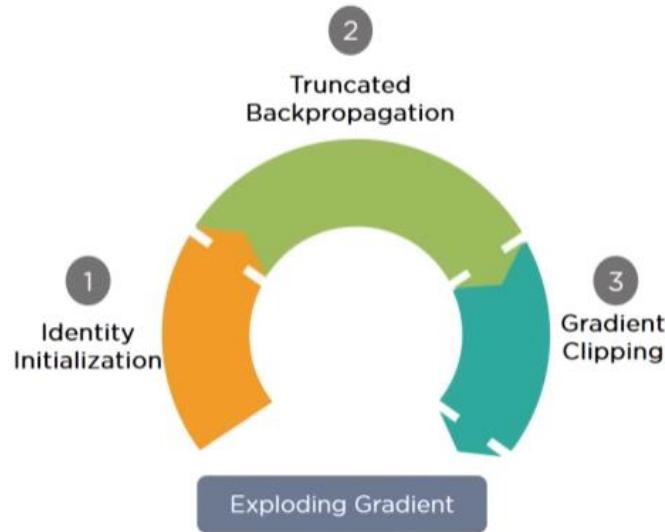


Bir RNN eğitirken, gradyanınız çok küçük veya çok büyük olabilir ve bu da eğitimi zorlaştırır. Gradyan çok küçük olduğunda, sorun "Kaybolan Gradyan Problemi" olarak bilinir.

Solution to Gradient Problem(Kaybolan Gradyan Problemi Çözümü)



Exploding Gradient Çözümleri



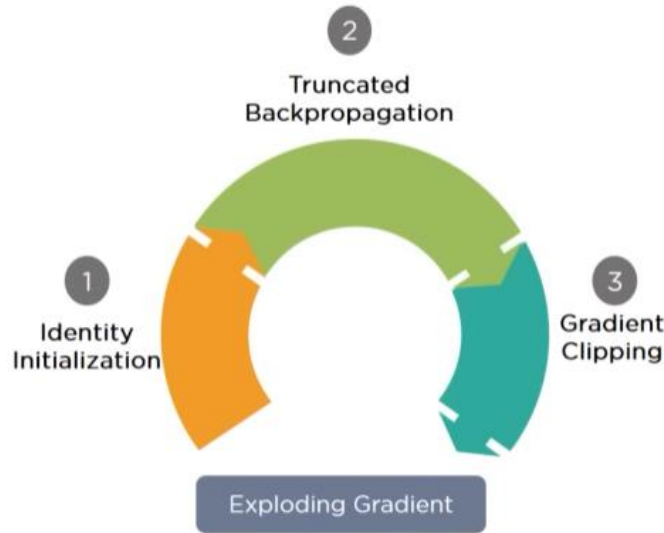
Identity Initialization (Kimlik Başlatma):

- Sadece önemli bilgilere odaklanarak daha kararlı bir öğrenme süreci sağlar.
- Ağırlıkların başlangıç değerleri, kimlik matrisine yakın seçilir. Bu sayede gradyanların büyümesi engellenmeye çalışılır.

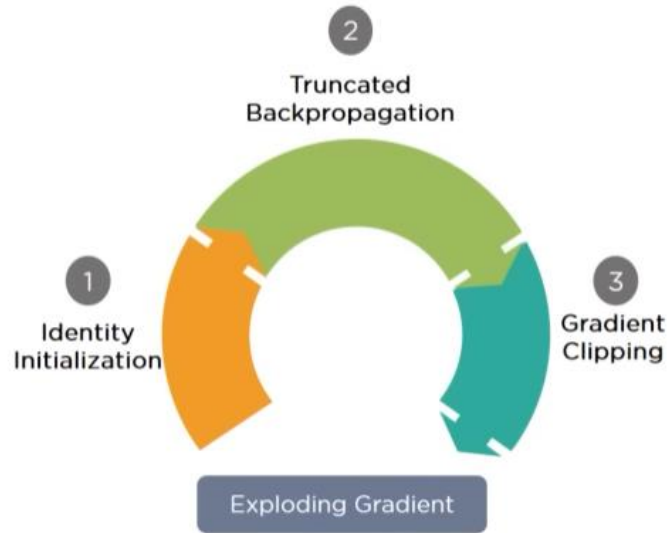
Exploding Gradient Çözümleri

Truncated Backpropagation (Kısaltılmış Geri Yayılım):

- Ağın derinliği azaltılarak, gradyanların geriye doğru yayılması sırasında kat ettiği yol kısaltılır. Bu sayede gradyanların çarpılarak büyümesi engellenir.



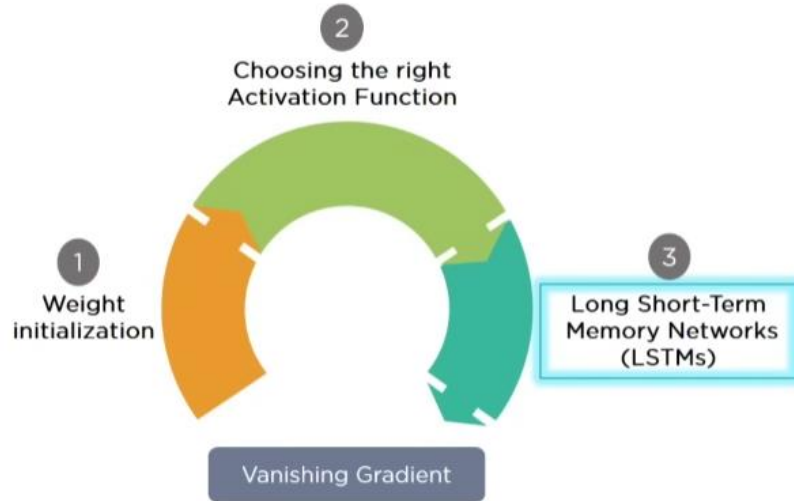
Exploding Gradient Çözümleri



Gradient Clipping (Gradyan Kırpma):

- Gradyanların belirli bir değerin üzerine çıkmasına izin verilmez. Bu sayede gradyanların aşırı büyümesi engellenir.

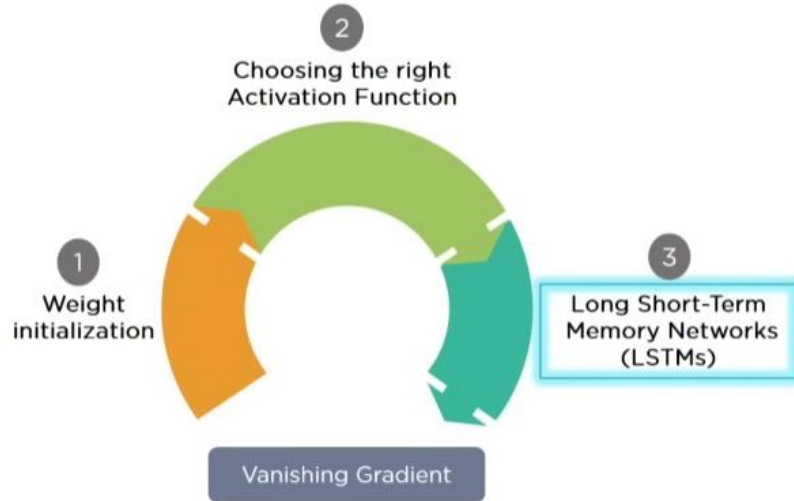
Vanishing Gradient Çözümleri



Weight Initialization (Ağırlık Başlatma):

- Ağırlıkların başlangıç değerleri, gradyanların çok küçük olmaması için dikkatlice seçilir. Xavier veya He başlatma gibi yöntemler kullanılır.

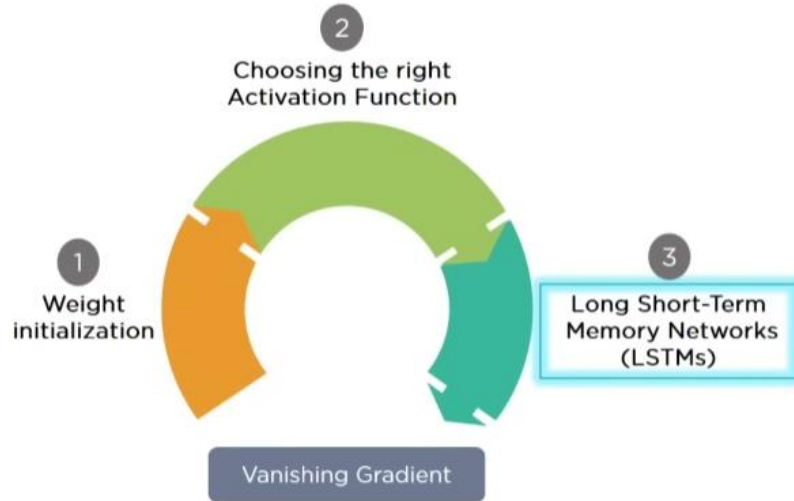
Vanishing Gradient Çözümleri



Activation Functions (Aktivasyon Fonksiyonları):

- Sigmoid gibi fonksiyonlar yerine ReLU gibi doğrusal olmayan fonksiyonlar kullanmak, gradyanların sıfıra yaklaşmasını engeller.

Vanishing Gradient Çözümleri



Long short-term memory (LSTM) Ağları:

- Özellikle uzun süreli bağımlılıkları öğrenmek için tasarlanmıştır. Hücre durumları ve kapılar sayesinde gradyanların kaybolmasını engeller.

Long - Term Dependencies (Uzun Süreli Bağımlılıklar)

Örnek 1:

Cümle: "Balıklar suda ..."

Son kelimeyi tahmin etmek kolaydır çünkü yakın kontekst yeterlidir.

Bir metindeki kelimelerin anlamı, sadece o kelimedden önceki birkaç kelimeye değil, daha önceki birçok kelimeye bağlı olabilir.

Long - Term Dependencies (Uzun Süreli Bağımlılıklar)

Örnek 2:

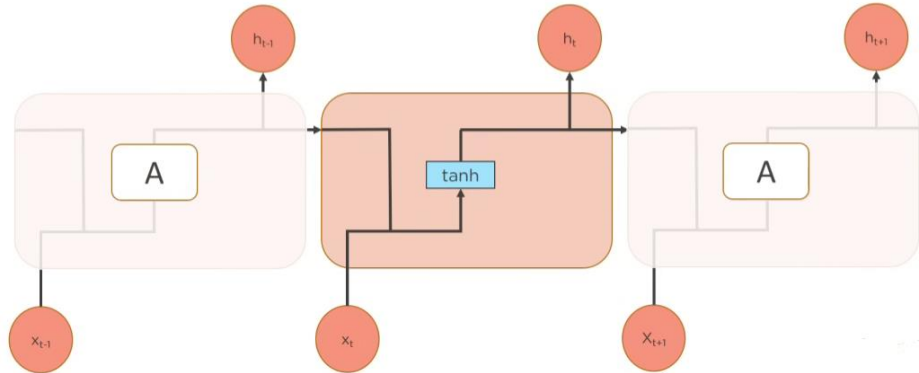
Cümle: "10 yıldır Fransa'da yaşıyorum. Akıcı bir şekilde ... konuşabiliyorum"

Eksik olan kelimeyi tahmin etmek için daha uzun bir kontekst gereklidir.

Burada, son kelimeyi tahmin etmek için Fransa bağlamına ihtiyacımız var. İlgili bilgi ile bu bilginin kullanılması gereken nokta arasındaki boşluk çok büyük olabilir. **LSTM'ler** bu problemi çözmemize yardımcı olur.

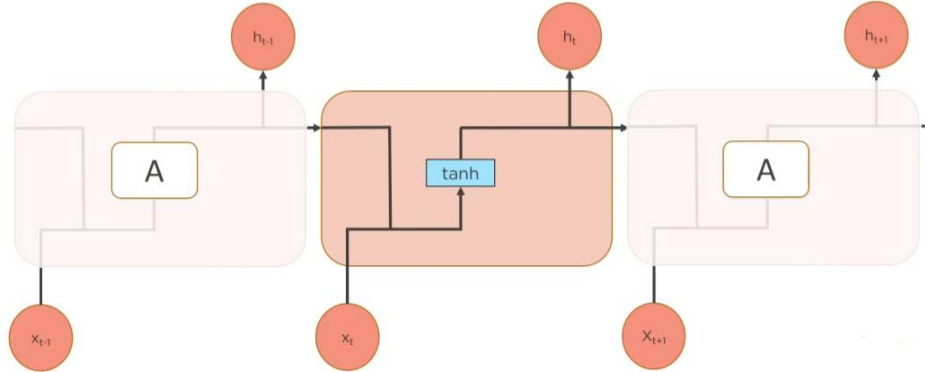
Çözüm: LSTM Ağları

LSTM'ler, uzun süreli bağımlılıkları öğrenme yeteneğine sahip özel bir Tekrarlayan Sinir Ağı türüdür. Bilgileri uzun süre hatırlamak onların doğal bir davranışıdır.



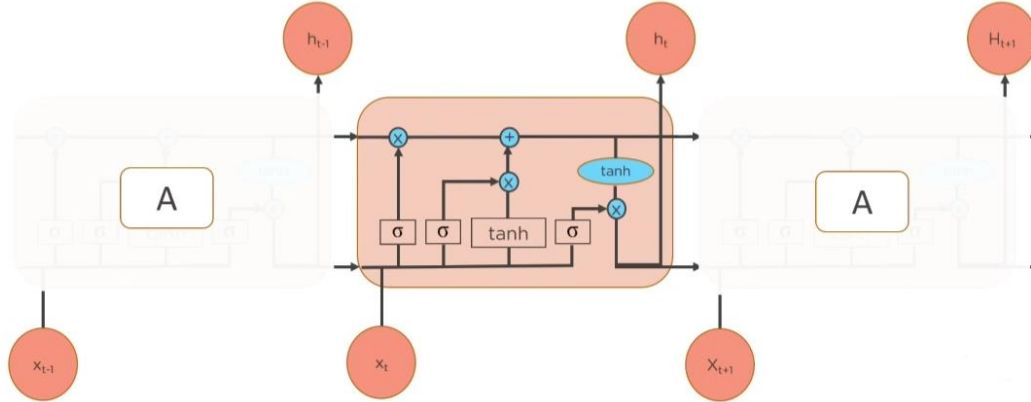
Çözüm: LSTM Ağları

Tüm tekrarlayan sinir ağları, birbirini tekrar eden sinir ağı modüllerinin bir zinciri şeklindedir. Standart RNN'lerde bu tekrar eden modül, tek bir tanh katmanı gibi çok basit bir yapıya sahiptir.



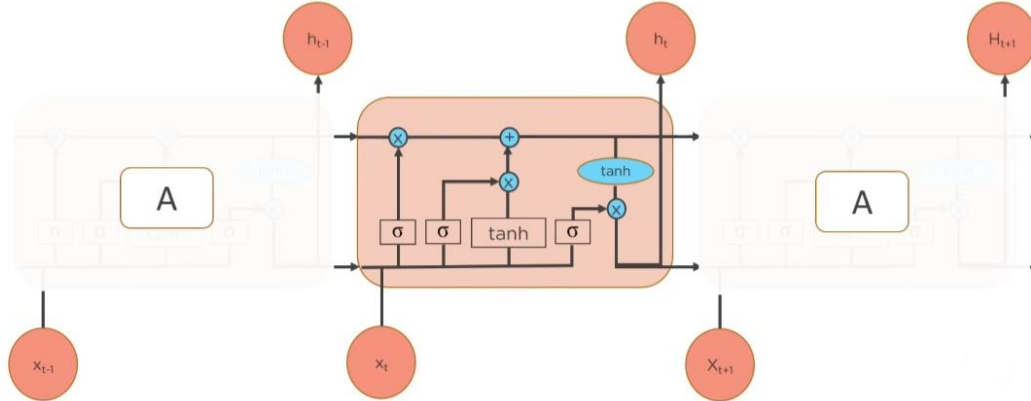
Çözüm: LSTM Ağları

Long short-term memory (LSTM): Özellikle zaman serisi verileri (örneğin, metin, konuşma) üzerinde çalışan ve uzun süreli bağımlılıkları yakalayabilen bir tür yapay sinir ağıdır. Bu sayede, örneğin bir cümlenin başındaki bilgileri sonuna kadar hatırlayarak daha doğru tahminler yapılabilir.



Çözüm: LSTM Ağları

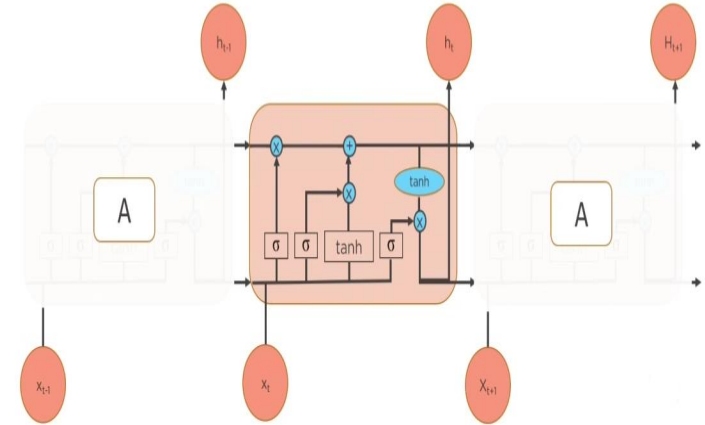
Recurrent Neural Network (RNN): Bilgileri sıralı bir şekilde işleyen ve önceki çıktıları bir sonraki girdi olarak kullanan bir tür sinir ağıdır. LSTM'ler de RNN'lerin bir türüdür.



Çözüm: LSTM Ağları

LSTM'ler, diğer RNN'lerden farklı olarak dört ana bileşene sahiptir:

- **Unutma Kapısı (Forget Gate):** Hücre durumundaki hangi bilginin atılacağına karar verir.
- **Giriş Kapısı (Input Gate):** Yeni bilginin hücre durumuna nasıl ekleneceğine karar verir.
- **Çıkış Kapısı (Output Gate):** Hücre durumundaki bilginin ne kadarının çıktıya aktarılacağına karar verir.
- **Hücre Durumu (Cell State):** Bilgilerin uzun süreli olarak saklandığı yerdir.

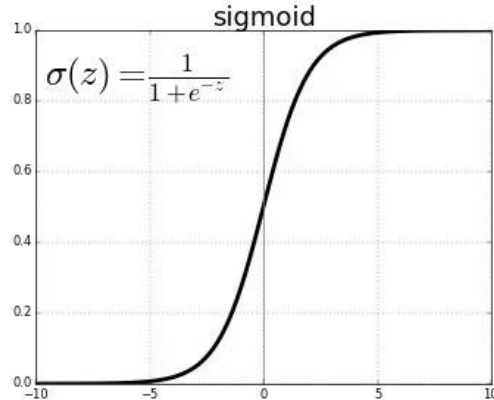


LSTM Nedir?

LSTM'in temel konsepti Cell State ve kullandığı çeşitli kapılardır(gates). **Cell State**, tahmin yapmak için anlamlı bilgileri hücreler boyunca taşıyan bir iletişim hattı ve ağın hafızası olarak açıklanabilir. Bu sayede kısa süreli bellek(short-term memory) problemi çözülerek eski veriler ağ zinciri boyunca taşınabilir. Cell State'in bu yolculuğu boyunca taşınması gereken bilgiler ise kapılar aracılığı ile belirlenir. Bu kapılar hangi bilginin gerekli veya gereksiz olduğunu belirleyebilir.

Sigmoid Fonksiyonu

Tekli sınıflandırmalarda kullanılır. Gelen girdi değerini (0,1) aralığında sıkıştırır. Fonksiyonun grafiği ve formülü şekildeki gibidir.





Örnek olarak eğer kedi olan bir resmi YSA modeline verip çıktı olarak sigmoid fonksiyonunda 0.85 değerini alıyorsak, bu resimde kedi olma ihtimalini %85 olarak değerlendirebiliriz. Tam tersi kedi olmayan resmi modelimize verip çıktı olarak 0.05 alıyorsak, bu resimde kedi olmama ihtimalini %99.95 olarak veya resimde kedi olma ihtimalini %0.05 olarak düşünebiliriz.

LSTM Örnek Uygulamaları

1. Model bir kelime dizisi ile beslendiğinde dil modelleme ve metin üretme uygulamaları için kullanılabilir.
2. Resimleri işleyebilir, resimler ve resimlerin açıklayıcı cümleleri ile beslendiğinde verilen resimleri açıklayabilir.(Image Captioning)
3. Konuşma ve el yazısı tanıma için kullanılabilir.
4. Müzik üretme için kullanılabilir.
5. Karakter veya kelime seviyesinde eğitilerek cümleleri bir dilden başka bir dile çevirebilir. Resim işlemede olduğu gibi 2 farklı dilde cümleler ile beslenerek eğitilir.



1-) Forget Gate (Unutma Kapısı):

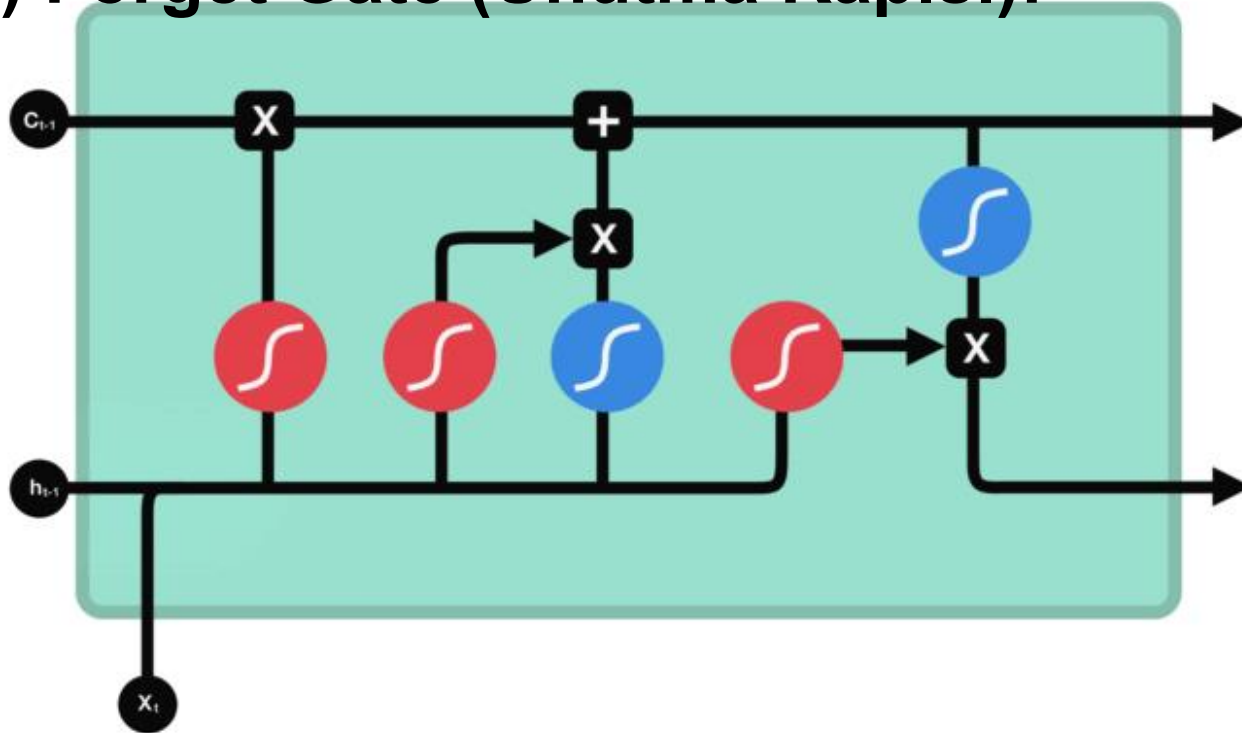
Hangi bilginin tutulacağı veya unutulacağına karar verir. Mantığı hiç karmaşık değil. Matematikte öğrendiğimiz gibi, bir sayı 0 ile çarpılırsa ne kadar büyük olursa olsun sonuç 0 olur. Burada da aynı mantıkla işlem yapılıyor. Unutmak için girdinin ağırlığına 0 verilir.



1-) Forget Gate (Unutma Kapısı):


Bir önceki gizli katmandan gelen bilgiler ve güncel bilgiler Sigmoid Fonksiyonundan geçer. 0'a ne kadar yakınsa o kadar unutulacak, 1'e ne kadar yakınsa o kadar tutulacak demektir.

1-) Forget Gate (Unutma Kapısı):



c_{t-1} previous cell state

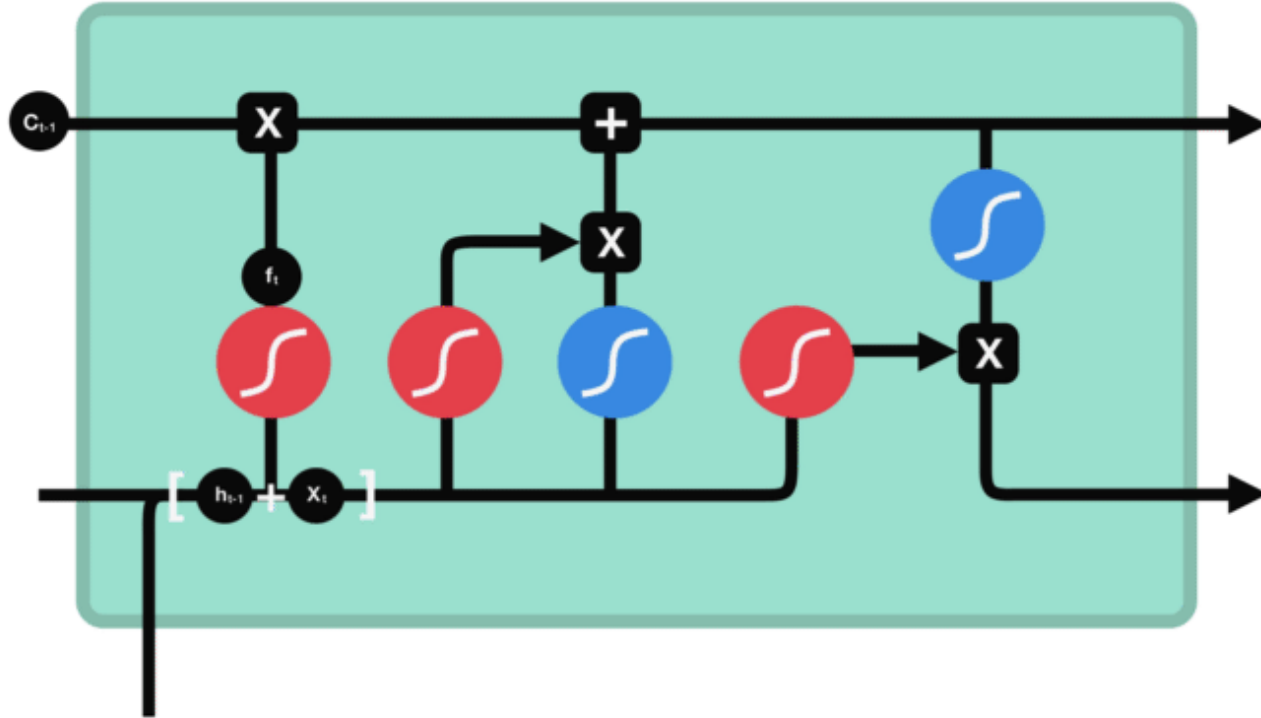
f_t forget gate output



2-) Input Gate (Girdi Kapısı):

Cell State'i güncellemek için kullanılır. Öncelikle Forget Gate'de (Unutma Kapısı) olduğu gibi Sigmoid fonksiyonu uygulanır, hangi bilginin tutulacağına karar verilir. Daha sonra ağı düzenlemek için Tanh fonksiyonu yardımıyla $-1,1$ arasına indirgenir ve çıkan iki sonuç çarpılır.

2-) Input Gate (Girdi Kapısı):



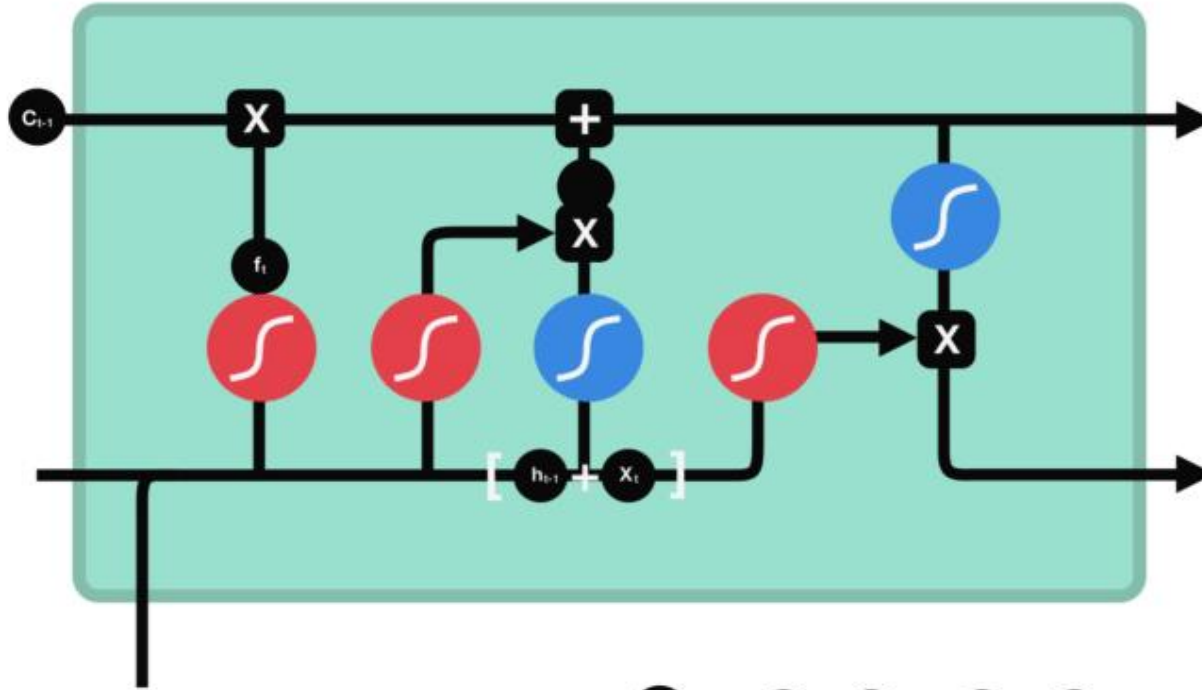
- c_{t-1} previous cell state
- f_t forget gate output
- i_t input gate output
- \hat{c}_t candidate



3-) Cell State:

Cell State'in hücre içerisindeki en önemli görevi bilgiyi taşımaktır. Taşınması gereken verileri alır ve hücre sonuna, oradan da diğer hücrelere taşır. Yani ağ üzerinde veri akışını Cell State yardımıyla sağlarız. İlk olarak Forget Gate'den (Unutma Kapısı) gelen sonuç ile bir önceki katmanın sonucu çarpılır. Daha sonra Input Gate'den (Girdi Kapısı) gelen değer ile toplanır.

3-) Cell State:



- c_{t-1} previous cell state
- f_t forget gate output
- i_t input gate output
- \tilde{c}_t candidate
- c_t new cell state

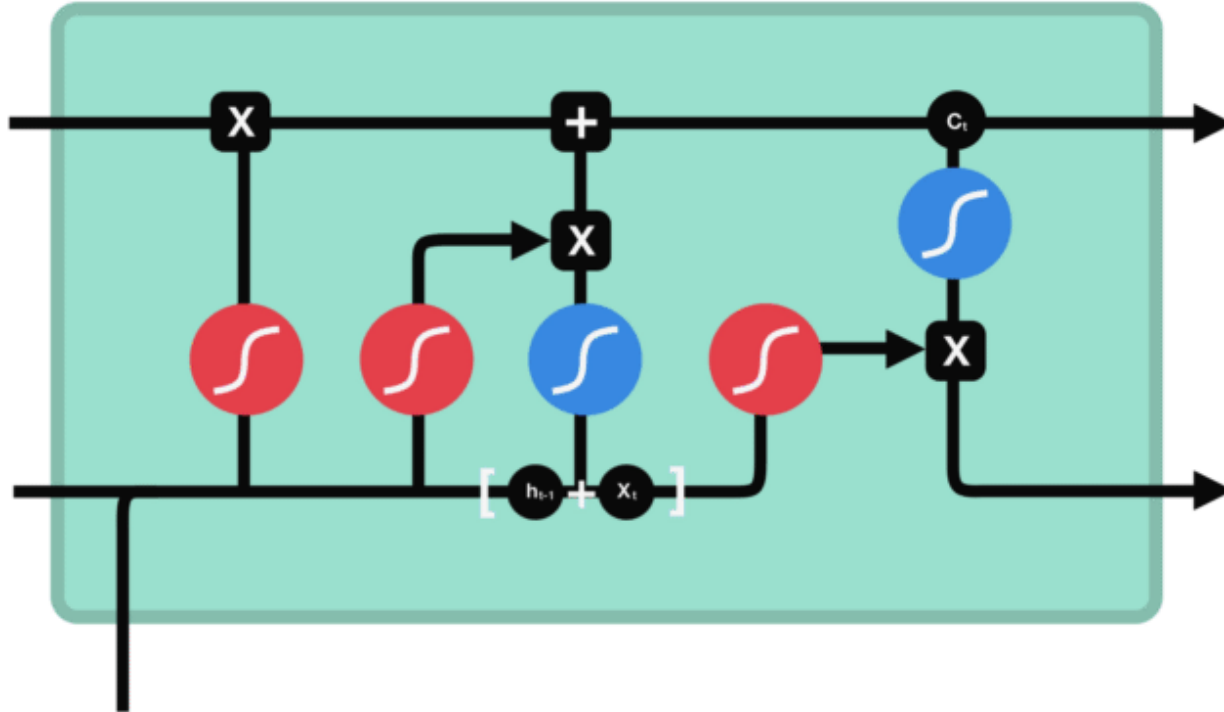
$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$



4-) Output Gate (Çıktı Kapısı):

Bir sonraki katmana gönderilecek değere karar verir. Bu değer, tahmin için kullanılır. Öncelikle bir önceki değer ile şu anki girdi Sigmoid fonksiyonundan geçer. Cell State'den gelen değer Tanh fonksiyonundan geçtikten sonra iki değer çarpılır ve bir sonraki katmana “Bir önceki değer” olarak gider. Cell State ilerler.

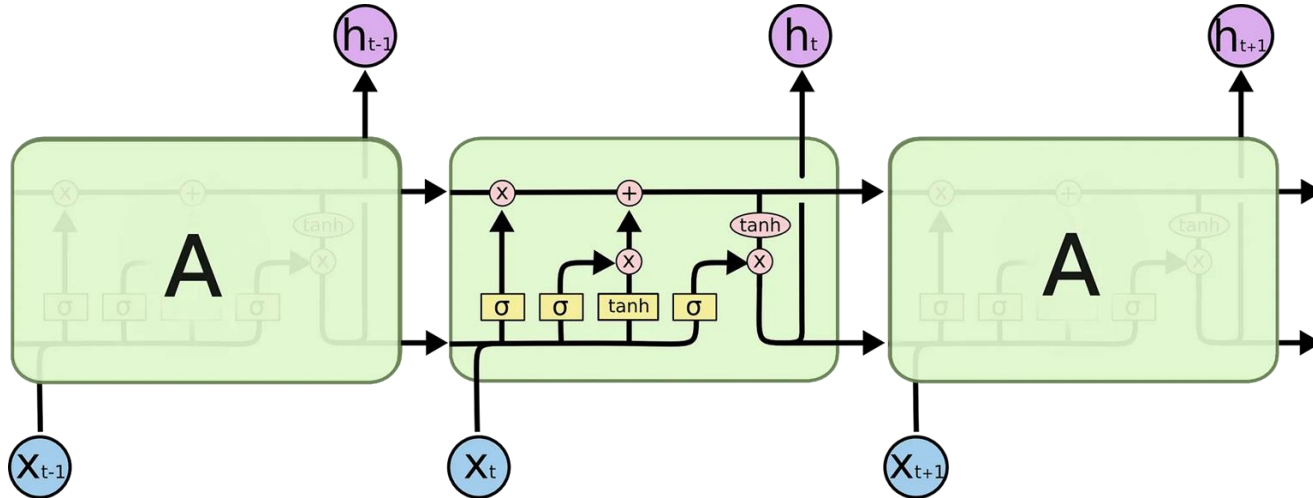
4-) Output Gate (Çıktı Kapısı):



- c_{t-1} previous cell state
- f_t forget gate output
- i_t input gate output
- \tilde{c}_t candidate
- c_t new cell state
- o_t output gate output
- h_t hidden state

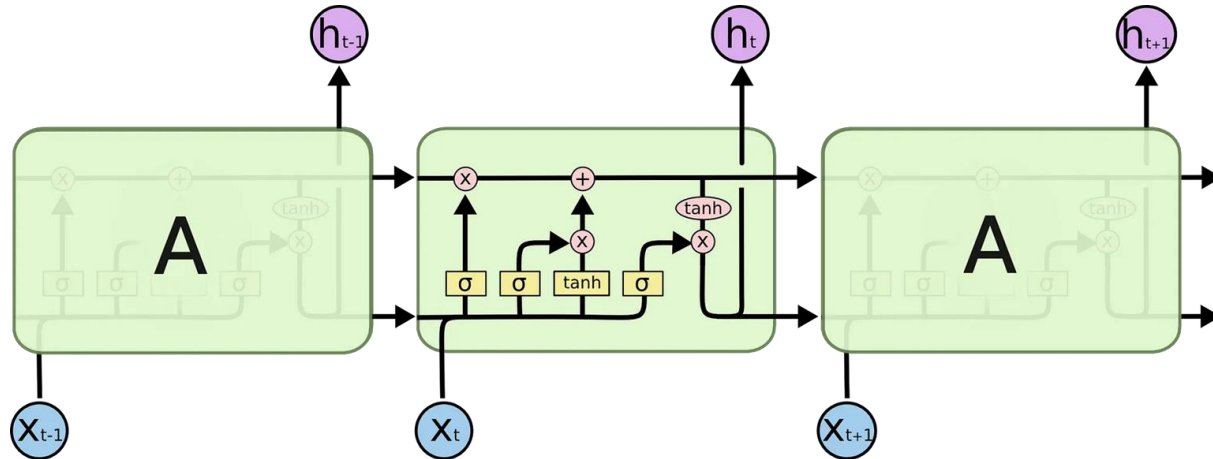
4-) Output Gate (Çıktı Kapısı):

Şu ana kadar anlattıklarım hücre içerisinde yapılan işlemlerdi. Daha geniş açıyla bakacak olursak:



4-) Output Gate (Çıktı Kapısı):

LSTM, duygu analizi, metin üretme ve zaman serileri gibi birçok konuda kullanılır.



Kod Örneği: Karakter Bazlı DNN ile İsimlerin Sınıflandırılması

```
from __future__ import unicode_literals, print_function, division
from io import open
import glob
import os

# Belirtilen bir dosya yolundaki tüm dosyaları bulan bir fonksiyon
def findFiles(path): return glob.glob(path)

# 'data/names/' klasöründeki tüm .txt dosyalarını bul ve listele
print(findFiles('data/names/*.txt'))

import unicodedata
import string
```



```
import unicodedata
import string
```

```
# ASCII harfler ve bazı özel karakterler; isim verisi temizliği için kullanılacak
all_letters = string.ascii_letters + " .,;"
n_letters = len(all_letters)
```

```
# Unicode karakterleri ASCII'ye dönüştüren fonksiyon
def unicodeToAscii(s):
    return ''.join(
        c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn'
        and c in all_letters
    )
```

Örnek bir isimde Unicode'dan ASCII'ye dönüşümün çıktısını gösterir

```
print(unicodeToAscii('Ślusàrski'))
```

Her bir kategoriye (ülkeye) ait isimlerin tutulacağı sözlük

```
category_lines = {}
```

Tüm kategorilerin (ülkelerin) tutulacağı liste

```
all_categories = []
```

Belirtilen dosyadan satırları okuyan fonksiyon

```
def readLines(filename):
```

Dosyadaki her satırı okuyup Unicode'dan ASCII'ye dönüştürür

```
lines = open(filename, encoding='utf-8').read().strip().split('\n')
```

```
return [unicodeToAscii(line) for line in lines]
```

```
# 'data/names/' klasöründeki her dosya için işlemler
for filename in findFiles('data/names/*.txt'):
    # Dosya adından kategori adını çıkar (örneğin, İngilizce için 'English')
    category = os.path.splitext(os.path.basename(filename))[0]
    all_categories.append(category) # Kategori listesini güncelle
    lines = readLines(filename)    # Dosyadaki isimleri oku
    category_lines[category] = lines # Kategoriye ait isimleri sözlüğe ekle

# Kategori (ülke) sayısını belirle
n_categories = len(all_categories)
```

```
# 'Italian' anahtarı altında bulunan ilk 5 ismi yazdır
print(category_lines['Italian'][:5])
```

```
import torch
```

```
# all_letters dizisinde belirtilen bir harfin indeksini bulan fonksiyon, örneğin "a" = 0
```

```
def letterToIndex(letter):
    return all_letters.find(letter)
```

```
# Bir harfi <1 x n_letters> boyutunda bir tensöre çevirme fonksiyonu (örnek için)
```

```
# Bu tensör bir "one-hot" vektörüdür, sadece belirli bir harfin konumunda 1 olur
```

```
def letterToTensor(letter):
    tensor = torch.zeros(1, n_letters) # <1 x n_letters> boyutunda sıfırlarla dolu bir tensör oluşturur
    tensor[0][letterToIndex(letter)] = 1 # Harfin indeksine göre tensörde 1 olarak işaretler
    return tensor
```

```
# Bir metni <line_length x 1 x n_letters> boyutunda tensöre çevirme fonksiyonu
# Bu tensör, harfleri "one-hot" vektörleri olarak temsil eden bir dizi içerir
def lineToTensor(line):
    tensor = torch.zeros(len(line), 1, n_letters) # Her harf için sıfırlardan oluşan bir tensör oluşturur
    for li, letter in enumerate(line): # Metindeki her harf için
        tensor[li][0][letterToIndex(letter)] = 1 # Harfin indeksine göre tensörde 1 olarak işaretler
    return tensor

# 'J' harfini "one-hot" tensör olarak yazdır
print(letterToTensor('J'))

# 'Jones' kelimesini tensör boyutunda yazdır
print(lineToTensor('Jones').size())
```



```
import torch.nn as nn

# RNN sınıfını tanımlıyoruz, nn.Module sınıfını miras alıyor
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__() # nn.Module'ın özelliklerini RNN'e aktarır

        self.hidden_size = hidden_size # Gizli katmanın boyutu

        # Giriş ve gizli durumun birleşiminden gizli duruma geçiş katmanı
        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)

        # Giriş ve gizli durumun birleşiminden çıkışa geçiş katmanı
        self.i2o = nn.Linear(input_size + hidden_size, output_size)

        # Çıkış katmanı için LogSoftmax aktivasyon fonksiyonu
        self.softmax = nn.LogSoftmax(dim=1)
```

İleri yayılım fonksiyonu, RNN'in giriş ve gizli durumu işleyerek çıktıyı üretmesini sağlar

```
def forward(self, input, hidden):  
    combined = torch.cat((input, hidden), 1) # Giriş ve gizli durumu birleştiriyoruz  
    hidden = self.i2h(combined)             # Yeni gizli durumu hesaplıyoruz  
    output = self.i2o(combined)             # Çıkış hesaplanıyor  
    output = self.softmax(output)           # LogSoftmax ile normalize ediliyor  
    return output, hidden                  # Çıktı ve yeni gizli durum döndürülüyor
```

Başlangıç gizli durumu oluşturma fonksiyonu (başlangıçta tüm değerler 0)

```
def initHidden(self):  
    return torch.zeros(1, self.hidden_size)
```

Gizli katman boyutu

```
n_hidden = 128
```

RNN modelini başlatıyoruz, giriş boyutu n_letters, çıkış boyutu n_categories

```
rnn = RNN(n_letters, n_hidden, n_categories)
```

'A' harfi için bir "one-hot" tensör oluşturuyoruz

```
input = letterToTensor('A')
```

Gizli durumun başlangıç değeri olarak sıfırlardan oluşan bir tensör oluşturuyoruz

```
hidden = torch.zeros(1, n_hidden)
```

RNN modeline 'A' harfi ve başlangıç gizli durumunu vererek çıktıyı ve bir sonraki gizli durumu elde ediyoruz

```
output, next_hidden = rnn(input, hidden)
```



```
# 'Albert' ismini temsil eden bir tensör oluşturuyoruz.  
# Bu tensör, her harfi "one-hot" olarak temsil eden bir dizi tensörden oluşur.  
input = lineToTensor('Albert')  
  
# Gizli durumun başlangıç değeri olarak sıfırlardan oluşan bir tensör oluşturuyoruz  
hidden = torch.zeros(1, n_hidden)  
  
# İlk harfi (input[0]) ve gizli durumu RNN modeline vererek çıktıyı ve yeni gizli durumu alıyoruz  
output, next_hidden = rnn(input[0], hidden)  
  
# Çıktıyı ekrana yazdırıyoruz; bu çıktı, modelin 'A' harfi ve başlangıç gizli durumu ile oluşturduğu tahmini temsil eder  
print(output)
```

Modelin çıktısından kategori tahminini belirleyen fonksiyon

def categoryFromOutput(output):

En yüksek değeri ve onun indeksini buluyoruz (bu kategori tahminidir)

top_n, top_i = output.topk(1) *# En yüksek olasılık değerine sahip olanı seçiyoruz*

category_i = top_i[0].item() *# İndeksi bir Python tamsayısına çeviriyoruz*

return all_categories[category_i], category_i *# Kategori adını ve indeksini döndürüyoruz*

Tahmin edilen kategori ve indeksini ekrana yazdırıyoruz

print(categoryFromOutput(output))

```
import random
```

```
# Listedeki rastgele bir öğe seçen fonksiyon
```

```
def randomChoice(l):
```

```
    return l[random.randint(0, len(l) - 1)]
```

```
# Rastgele bir eğitim örneği oluşturan fonksiyon
```

```
def randomTrainingExample():
```

```
    # Kategoriler arasından rastgele bir kategori seçiyoruz
```

```
    category = randomChoice(all_categories)
```

```
# Seçilen kategoriye ait isimler arasından rastgele bir isim seçiyoruz
```

```
    line = randomChoice(category_lines[category])
```

```
# Kategori için indeks tensörünü oluşturuyoruz (kategori numarasını uzunluk türünde tensor yapıyoruz)  
category_tensor = torch.tensor([all_categories.index(category)], dtype=torch.long)
```

```
# Seçilen ismi (line) tensöre çeviriyoruz  
line_tensor = lineToTensor(line)
```

```
# Kategori adı, isim, kategori tensörü ve isim tensörünü döndürüyoruz  
return category, line, category_tensor, line_tensor
```

```
# 10 adet rastgele eğitim örneği oluşturup yazdırıyoruz
```

```
for i in range(10):  
    category, line, category_tensor, line_tensor = randomTrainingExample()  
    print('category =', category, '/ line =', line)
```

```
# Negatif Log-Likelihood (NLL) kayıp fonksiyonunu tanımlıyoruz  
criterion = nn.NLLLoss()
```

```
# Öğrenme oranını tanımlıyoruz; bu değer çok yüksek olursa model kararsızlaşabilir,  
# çok düşük olursa model öğrenim sürecini etkili bir şekilde gerçekleştiremeyebilir  
learning_rate = 0.005
```

```
# Modelin eğitimini gerçekleştiren fonksiyon  
def train(category_tensor, line_tensor):  
    # RNN için başlangıç gizli durumunu başlatıyoruz  
    hidden = rnn.initHidden()  
  
    # RNN modelinin gradyanlarını sıfırlıyoruz  
    rnn.zero_grad()  
  
    # Girdi tensöründeki her harf için RNN modelini çalıştırıyoruz  
    for i in range(line_tensor.size()[0]):  
        output, hidden = rnn(line_tensor[i], hidden)  
  
    # Çıktıyı ve kategori tensörünü kullanarak kaybı hesaplıyoruz  
    loss = criterion(output, category_tensor)
```

Kayıp değerine göre gradyanları hesaplıyoruz

```
loss.backward()
```

Öğrenme oranı ile çarpılan gradyanları, parametre değerlerine ekliyoruz

Bu işlem, her parametrenin güncellenmesini sağlar

```
for p in rnn.parameters():
```

```
    p.data.add_(-learning_rate, p.grad.data)
```

Son çıktıyı ve kayıp değerini döndürüyoruz

```
return output, loss.item()
```

```
import time
import math

# Toplam iterasyon sayısı
n_iters = 100000
# Her 5000 iterasyonda çıktı vermek için ayarlama
print_every = 5000
# Her 1000 iterasyonda kayıp değerini grafiğe eklemek için ayarlama
plot_every = 1000

# Kayıpları takip etmek için bir liste
current_loss = 0
all_losses = []

# Geçen süreyi hesaplayan fonksiyon
def timeSince(since):
    now = time.time() # Şu anki zamanı al
    s = now - since # Başlangıç zamanından bu yana geçen süreyi hesapla
    m = math.floor(s / 60) # Dakika cinsinden süreyi hesapla
    s -= m * 60 # Saniye cinsinden süreyi hesapla
    return '%dm %ds' % (m, s) # Dakika ve saniyeyi döndür

start = time.time() # Eğitim süresini başlat
```

```

# Eğitim döngüsü
for iter in range(1, n_iters + 1):
    # Rastgele bir eğitim örneği oluştur
    category, line, category_tensor, line_tensor = randomTrainingExample()
    # Modeli eğit ve kaybı hesapla
    output, loss = train(category_tensor, line_tensor)
    current_loss += loss # Geçerli kaybı güncelle

    # Belirli bir iterasyonda çıktı ver
    if iter % print_every == 0:
        guess, guess_i = categoryFromOutput(output) # Modelin tahminini al
        # Doğruluğu kontrol et
        correct = '✓' if guess == category else 'X (%s)' % category
        # İterasyon numarasını, kaybı ve tahmini yazdır
        print('%d %d%% (%s) %.4f %s / %s %s' % (
            iter, iter / n_iters * 100, timeSince(start), loss, line, guess, correct))

    # Geçerli kayıp ortalamasını kayıplar listesine ekle
    if iter % plot_every == 0:
        all_losses.append(current_loss / plot_every) # Kayıp ortalamasını ekle
        current_loss = 0 # Geçerli kaybı sıfırla

```



```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

# Yeni bir grafik oluştur
plt.figure()
# Kayıp değerlerini grafiğe çiz
plt.plot(all_losses)

# Eksenleri ayarlamak için ticker kullan
plt.xlabel('Iterasyon (her 1000\'de bir)')
plt.ylabel('Kayıp (Loss)')
plt.title('Eğitim Kaybı Grafiği')

# Y ekseninde sayıları daha iyi göstermek için bir format belirle
plt.gca().yaxis.set_major_formatter(ticker.FormatStrFormatter('%.2f'))

# Grafiği göster
plt.show()
```

```

# Doğru tahminleri takip etmek için bir karmaşa matrisini başlat
confusion = torch.zeros(n_categories, n_categories) # n_categories x n_categories boyutunda sıfırlardan oluşan bir matris
n_confusion = 10000 # Değerlendirme için kullanılacak örnek sayısı

# Bir satıra verilen çıktıyı döndürmek için fonksiyon
def evaluate(line_tensor):
    hidden = rnn.initHidden() # RNN'in gizli durumunu başlat

    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden) # Her harf için çıkışı hesapla

    return output # Son çıkışı döndür

# Birçok örneği geçerek hangi tahminlerin doğru yapıldığını kaydet
for i in range(n_confusion):
    category, line, category_tensor, line_tensor = randomTrainingExample() # Rastgele bir eğitim örneği al
    output = evaluate(line_tensor) # Örneği değerlendir
    guess, guess_i = categoryFromOutput(output) # Tahmin edilen kategoriyi al
    category_i = all_categories.index(category) # Gerçek kategorinin indeksini bul
    confusion[category_i][guess_i] += 1 # Karmaşa matrisinde doğru tahmini güncelle

# Her satırı kendi toplamına bölerek normalize et
for i in range(n_categories):
    confusion[i] = confusion[i] / confusion[i].sum() # Satır toplamına böl

```

```
# Grafik ayarları
fig = plt.figure() # Yeni bir figür oluştur
ax = fig.add_subplot(111) # 1x1'lik bir ızgarada ilk alt grafiği oluştur
cax = ax.matshow(confusion.numpy()) # Karmaşa matrisini görselleştir
fig.colorbar(cax) # Renk çubuğunu ekle

# Eksen ayarları
ax.set_xticklabels([''] + all_categories, rotation=90) # X eksenindeki etiketleri ayarla
ax.set_yticklabels([''] + all_categories) # Y eksenindeki etiketleri ayarla

# Her tikte etiket zorla
ax.xaxis.set_major_locator(ticker.MultipleLocator(1)) # X eksenindeki her bir tike bir etiket yerleştir
ax.yaxis.set_major_locator(ticker.MultipleLocator(1)) # Y eksenindeki her bir tike bir etiket yerleştir

# Grafiği göster
plt.show()
```

```

def predict(input_line, n_predictions=3):
    # Kullanıcıdan alınan girdi satırını yazdır
    print('\n> %s' % input_line)
    with torch.no_grad(): # Gradyan hesaplamasını kapat (eğitim aşaması değil)
        output = evaluate(lineToTensor(input_line)) # Girdi satırını değerlendir

        # En yüksek N kategoriyi al
        topv, topi = output.topk(n_predictions, 1, True) # En yüksek n_predictions değerini ve indekslerini al
        predictions = [] # Tahminleri saklamak için bir liste oluştur

        # Tahmin edilen kategorileri ve değerlerini yazdır
        for i in range(n_predictions):
            value = topv[0][i].item() # Tahmin edilen değer
            category_index = topi[0][i].item() # Tahmin edilen kategorinin indeksi
            print('("%.2f) %s' % (value, all_categories[category_index])) # Değeri ve kategoriyi yazdır
            predictions.append([value, all_categories[category_index]]) # Tahminleri listeye ekle

# Örnek girdilerle tahminleri yap
predict('Dovesky') # 'Dovesky' ismi için tahmin yap
predict('Jackson') # 'Jackson' ismi için tahmin yap
predict('Satoshi') # 'Satoshi' ismi için tahmin yap

```

```

$ python predict.py Hazaki
(-0.42) Japanese
(-1.39) Polish
(-3.51) Czech

```

```

$ python predict.py Hinton
(-0.47) Scottish
(-1.52) English
(-3.57) Irish

```



TEŞEKKÜRLER

Mert SARIKAYA (212923012)

Yunus Emre SELCUK (212923032)

Onur BARBAROS (212923036)

Emirhan GÜNGÖR (212923059)

Efe Ata AKAN (212923016)

Kaynakça



1. <https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn>
2. <https://www.youtube.com/watch?v=IWkFhVq9-nc>
3. <https://aws.amazon.com/what-is/recurrent-neural-network/>
4. [https://www.elektrikport.com/makale-detay/yinelemeli-sinir-aglari-\(rnn\)-nedir/23277](https://www.elektrikport.com/makale-detay/yinelemeli-sinir-aglari-(rnn)-nedir/23277)
5. <https://bulutistan.com/blog/sinir-aglari-nedir-sinir-aglari-nasil-calisir/>

Kaynakça



6. https://www-turing-com.translate.goog/kb/mathematical-formulation-of-feed-forward-neural-network?_x_tr_sl=en&_x_tr_tl=tr&_x_tr_hl=tr&_x_tr_pto=wa&_x_tr_hist=true

7. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21?source=post_page-----326866fd8869-----

8. https://colah.github.io/posts/2015-08-Understanding-LSTMs/?source=post_page-----326866fd8869-----



KOD:

<https://colab.research.google.com/drive/1fnvH-CnPEfqD-SfBTP80YaQTHGrKXPps-?usp=sharing>