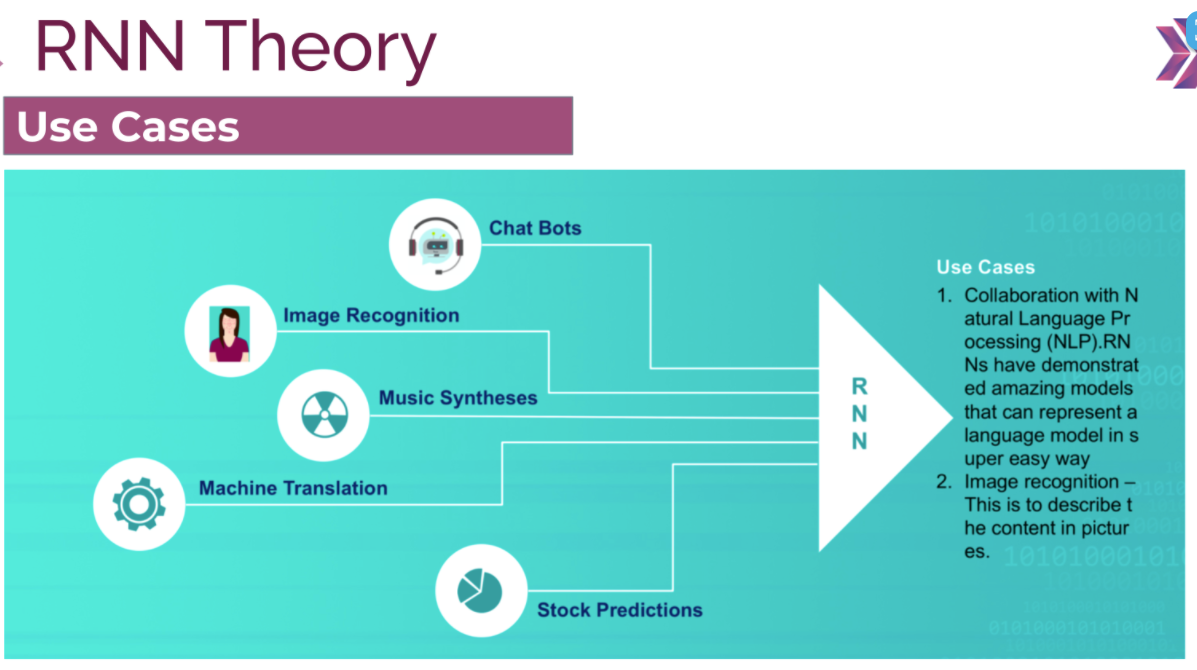
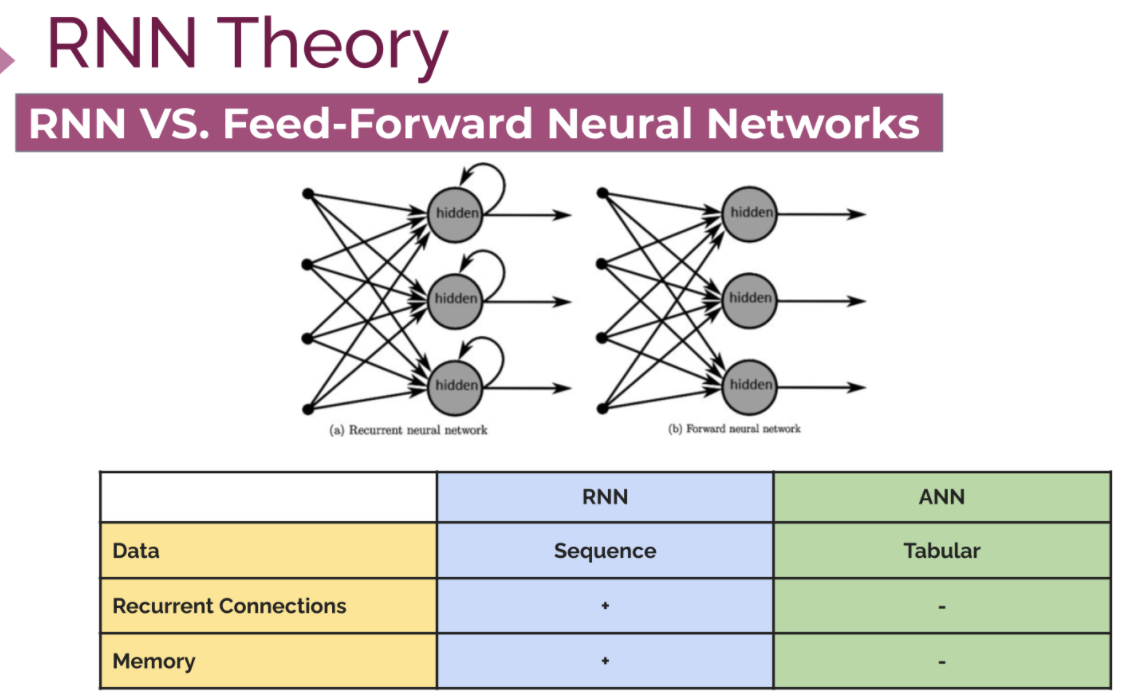
Eğer modeliminiz zamanlaması içeriğinden daha önemli ise biz RNN kullanırız

Mesela pikseller arasındaki bağı da siz işin içine katmak istiyorsanız, bir önceki piksel bir sonraki ile ilişkili ise. image recognition kısmında bunu hesaba katmanız gerekiyordur.



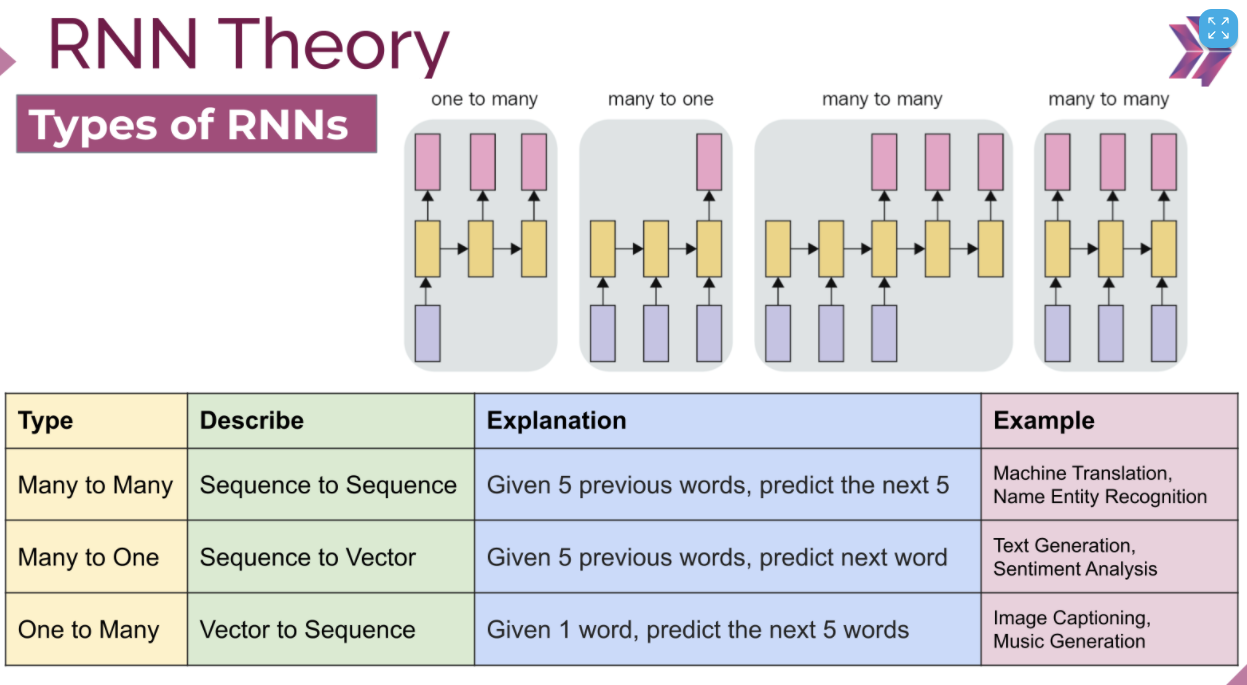
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

çıkış aynı nörona tekrar girerek geri besleme yapılıyor.



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

many yo many: 5 tane veri verdim modelimize 5 tane tahminde bulunup 5 kelime ile cevap verdi...

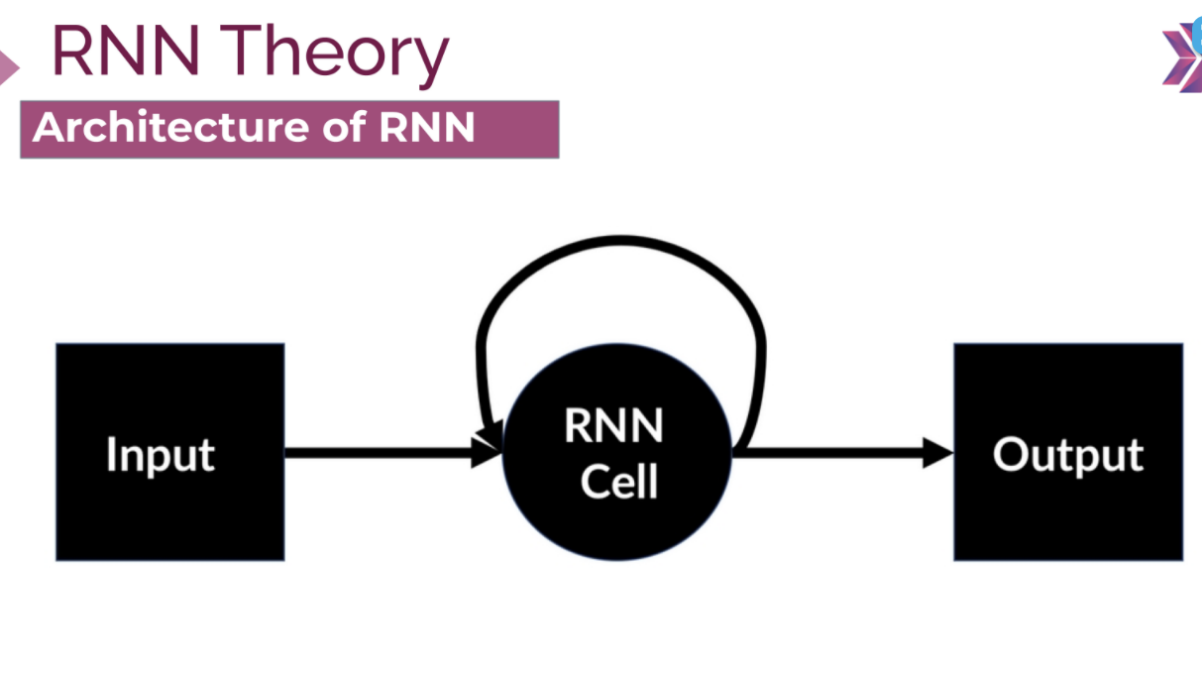


many to one: 5 kelime verdiniz, o da bir sonrakini tahmin ediyor.

one to many: 1 kelime veriyorsunuz, o da 5 kelime tahmini yapıyor. music generation gibi alanlarda kullanılıyor

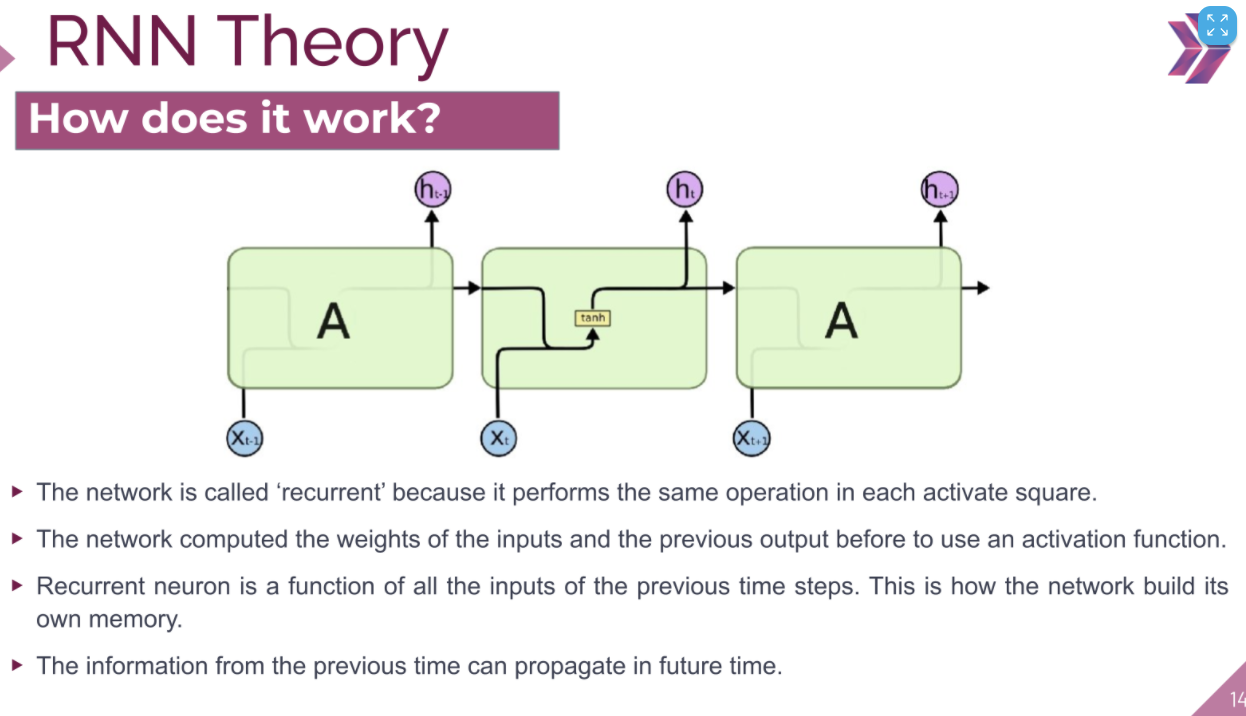
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Genel işleyiş: input geldi. RNN de işleme tabi tutuldu. aynı çıktı bir sonraki gelecek veriyle birlikte tekrar cell'e girerek işleme tabi tutuluyor. bu da dolayısıyla işleminize hafıza katıyor. memory den kastı bu



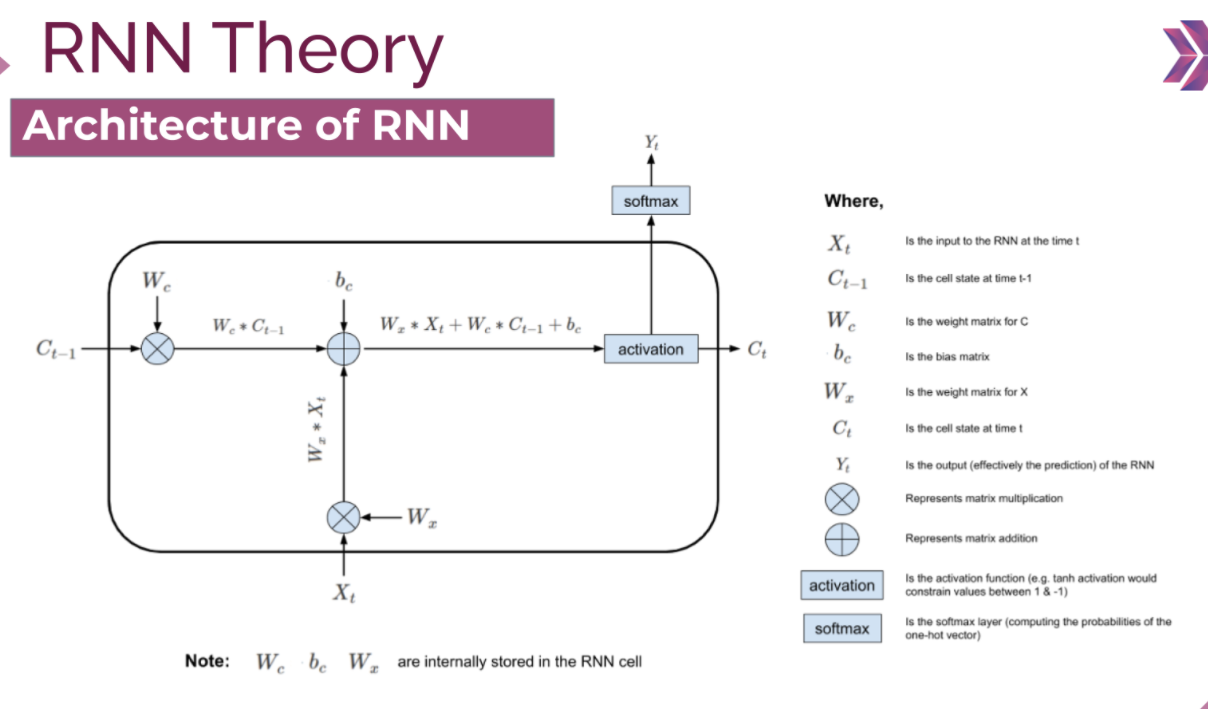
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

burada bir nöron var. aynı nöron zaman içerisinde üzerine yapılan girdilerle tekamül ediyor.  
birinci (t-1 zamanı) durumdan çıkan sonuca Xt den gelen veri giriyor ve bunlar toplanıp işleme giriyor ve çıktımız ht verisi oluyor. süreç bu şekilde devam ediyor.



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

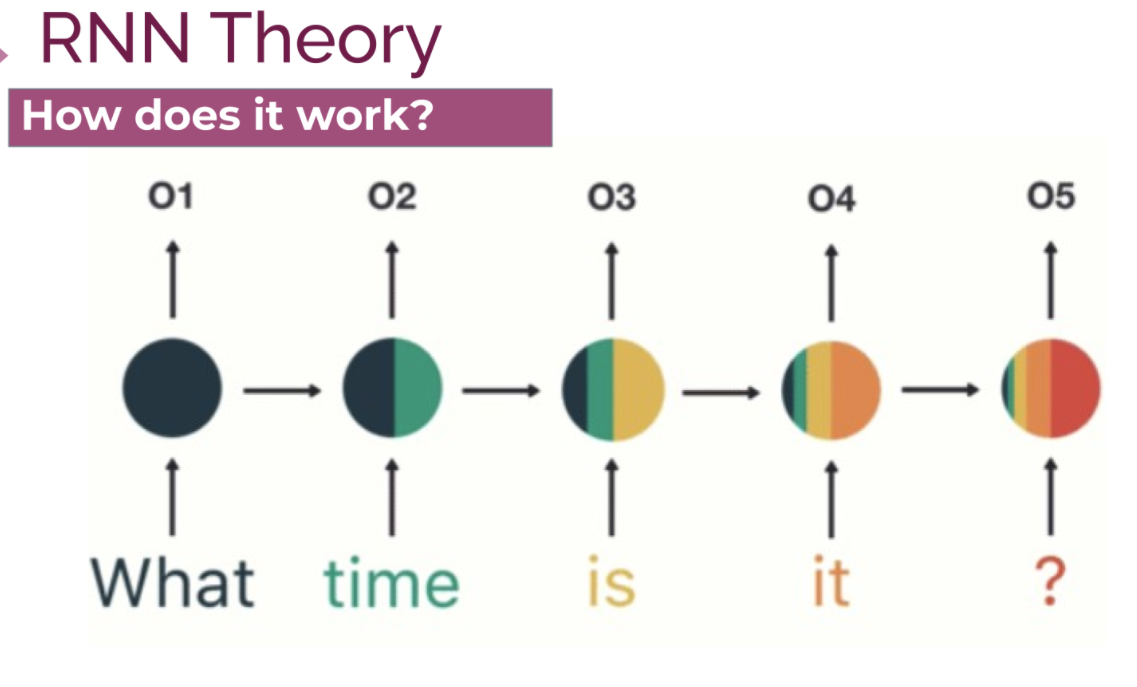
* C t-1 : bir önceki işlemlerden gelen hafıza.
* Wx, Wc : inputa giren çarpan (ağırlık). bacpropagation dan, gradient descent ten gelen ağırlıklar.
* önceki işlemden gelen veri Wc ağırlığıyla çarpılıyor. yeni veri (Xt) Wx ağırlığıyla çarpılıyor. bunlar toplanıyor ve bc sabiti ekleniyor ve bu şekilde aktivasyon işlemine tabi tutluyor. sonuçta Yt outputu ortaya çıkıyor. eğer tekrar besleme yapılacaksa bu output Ct olarak aynı nörona geri giriyor



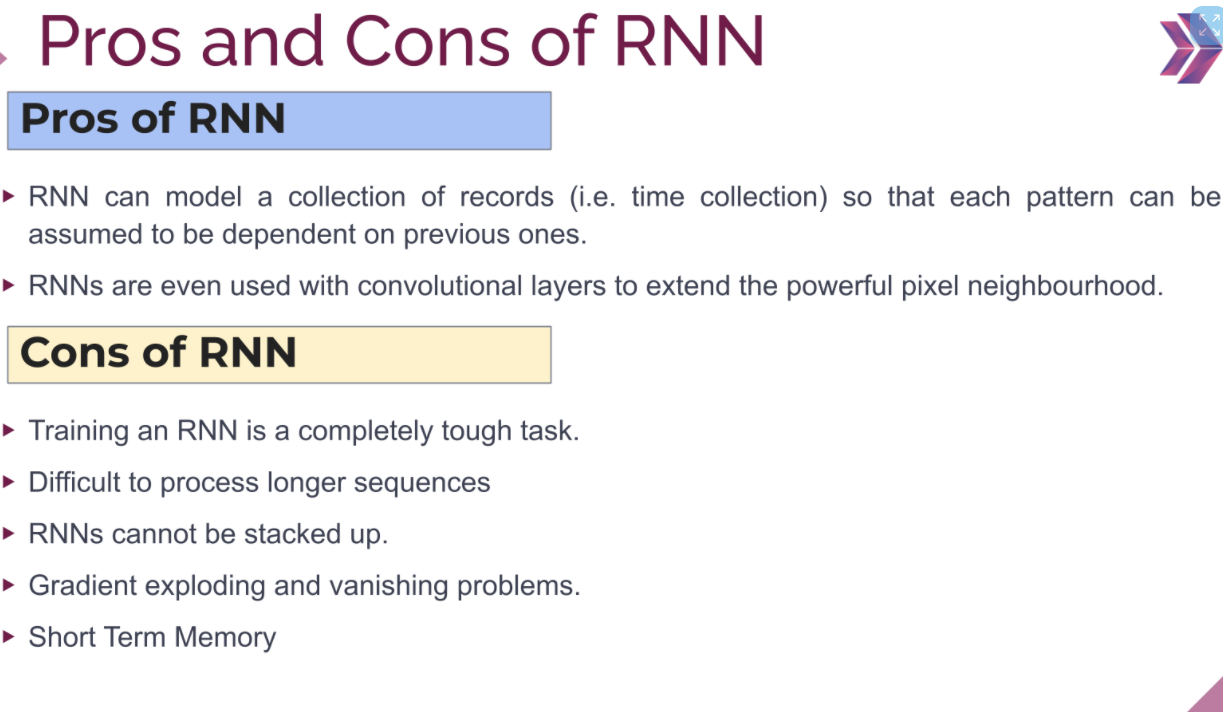
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 RNN tipi recurrent yapı örneği:

nörona önce what girdisi geldi. bir sonraki zamanda (yeni) time verisi girdi ve bir önceki what ile birlikte işleme tabi oldu. sonraki zamanda is girdi. yeni verinin ağırlığı diğerlerinden daha fazla. Memory'si var dolayısıyla önceki verileri hafızasında tutuyor ancak bunu yaparken önceki giren verilerin ağırlığı zaman içinde daha da azalıyor. ilk girenin ağırlığı en az oluyor. sonuçta tüm bu önceki girdilerin sonucunda 05. zamanda bize bir tahminde bulunuyor.



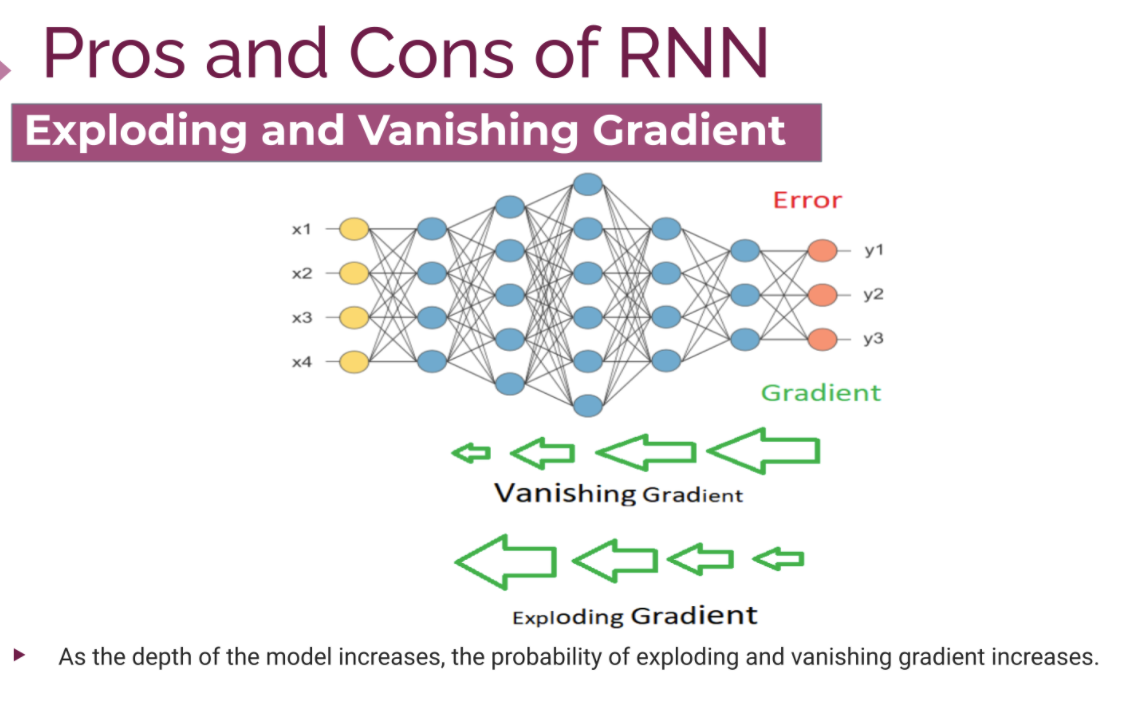
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

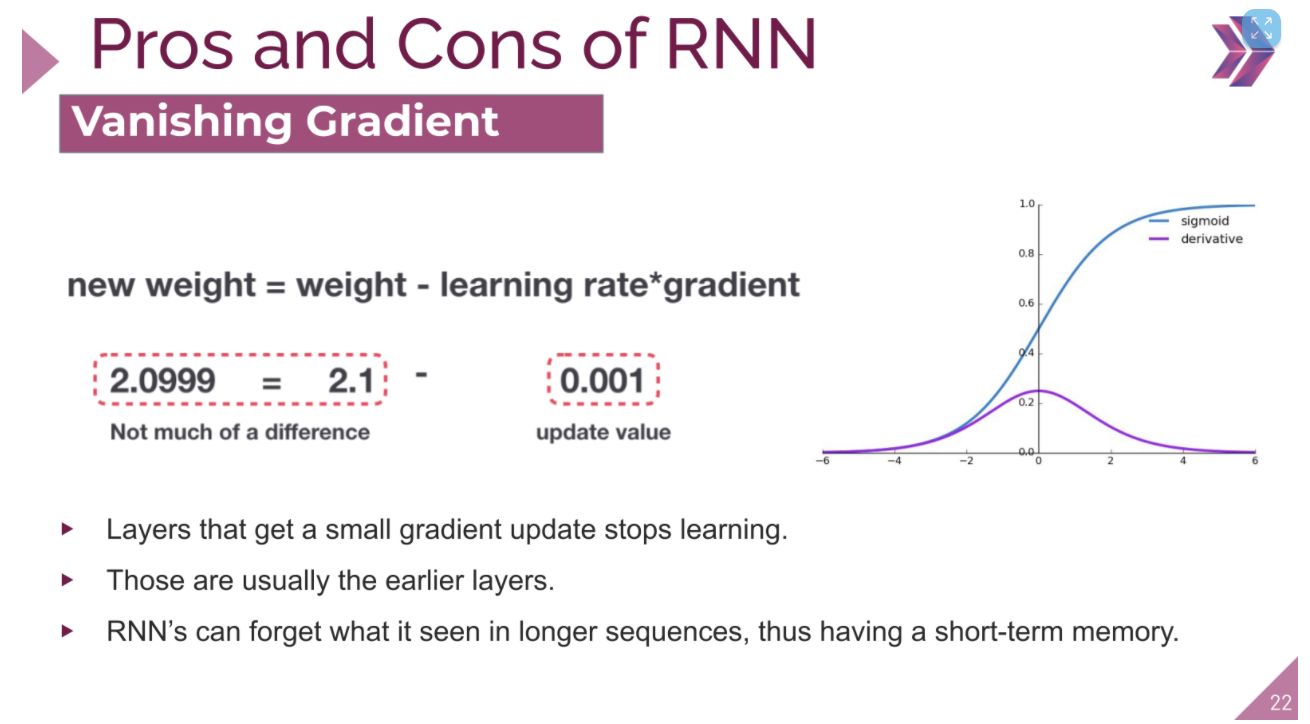
Backprogation ile en sondan başa doğru ağırlıklara düzeltme uygulayarak öğrenmeye katkı sağlanıyordu. türev alma işlemleri ile hata değerlerine göre ağırlıklar değiştiriliyordu. deep layer ne kadar derinleşirse hidden layer ne kadar artarsa gradient değeri kademe kademe azalıyor. geri dönüşte her layerde daha az gradient değeri ile ağırlıklar çarpılıyor. dolayısıyla git gide sönümlenme oluyor.

Buna Vanishing Gradient (vanish: sönümlenmek.. Gradient eğim demek, dikliğin eğimi demek, yani türevle alakalı..)  Erorların azalışındaki eğim, yani hatanın azalmasındaki değişim gradient değerini oluşturuyordu. ve o oranda da geriye dönük düzeltme uygulanıyordu. Geriye dönük düzeltme uygulanırken her layer da bu gradient değeri biraz daha azalarak uygulanıyordu. Bu değer azala azala Vanishing Gradient durumu oluşuyor. Takdir edersiniz ki layer sayısı arttıkça bu sönümlenme durumu daha fazla karşımıza çıkacaktır.



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

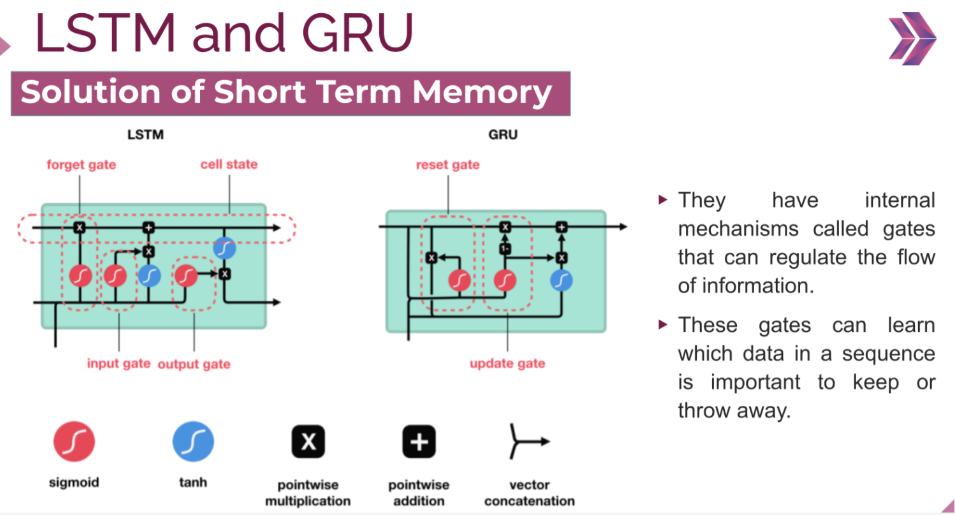
Modelimiz sondan başa gittikçe gradient seviyemiz düştüğünde (yani türev düştüğünde, eğim düştüğünde) öğrenmemiz de yavaşlayacak. sıfıra çok yakın değerlere ulaştığında model öğrenme durdu olarak kabul edecek.



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

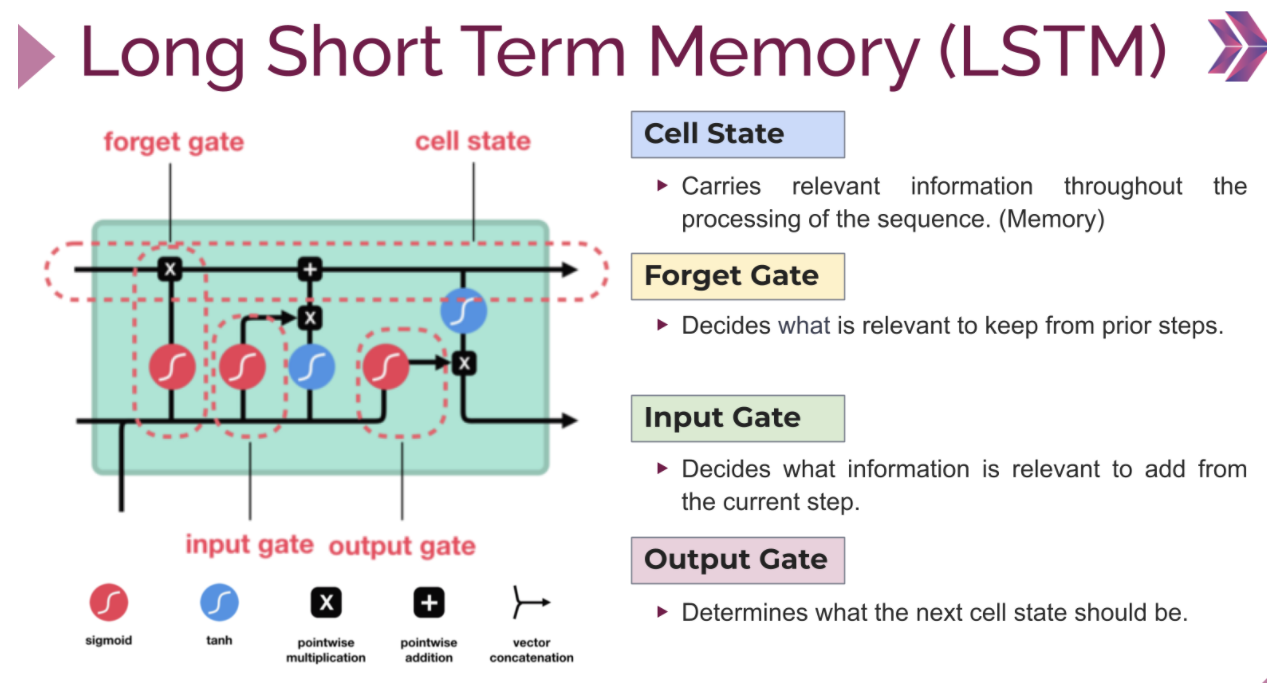
başa gittikçe Gradient Descent azaldığı için dolayısıyla başlardaki öğrenme azaldığı için önceki girilen verileri unutma durumu oluşuyor.süre geçtikçe t-1 den gelen verinin ağırlıklarında güncelleme daha az olacağı için unutma oluyor.

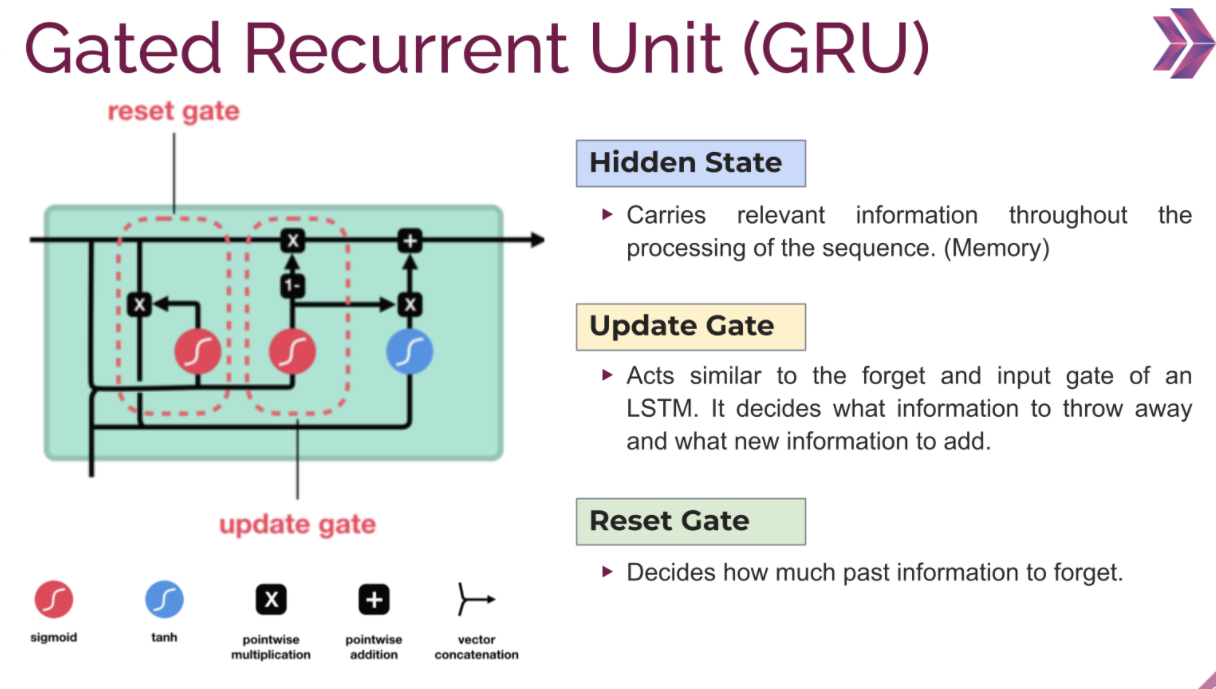
* Modelimizin öğrenmesini engelleyen bir durum vanishing gradient.
* geriye doğru gittikçe doğal olarak eğim (etki) düşüyor.
* hidden layer fazla ise ve ya activation function "tan" ise mesela, öğrenme azalıyor, en çok da diklik arttıkça meydana gelir.
* tek bir hidden layer varsa mesela "tanh" sıkıntı vermeyebilir ama layer sayıları (model uzunluğu) arttıkca relu yani "rectified" aktivasyon kullanılır.
* Karmaşık bir yapı oluşturulmuş.
* Ama genel anlamda:
* Yan tarafta yazılan gibi, kırmızılar kapı görevi gören aktivasyon fonksiyonlarıdır. Amacı ise önceki verilerden hangisi önemli hangisi değil, onu tutayım gibidir. Çünkü model uzadıkça baştaki girdilerden hangisi önemli onu akılda tutması gerekir.



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

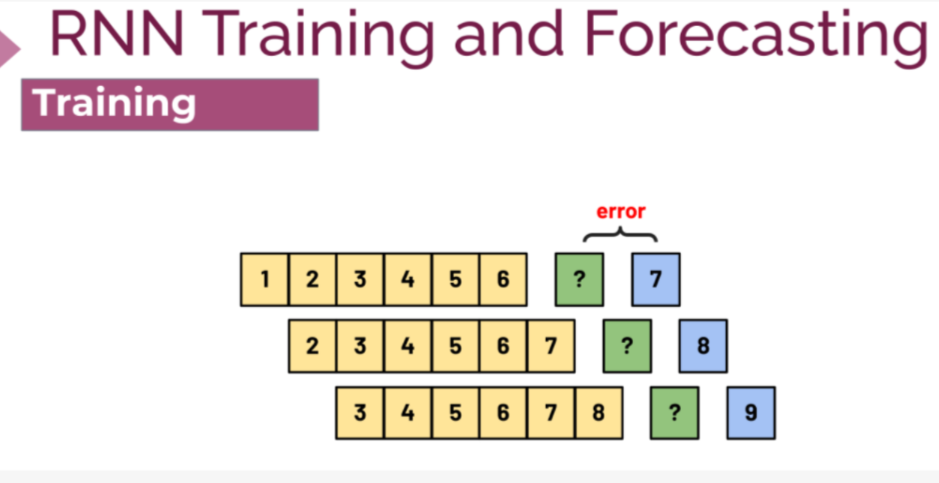
**cell state:** (yukardaki ok) aynı nörona geri beseleme için geri dönen veri.  
**forget gate**: (aşağıdaki ok) burada cell state teki aynı nörona geri giren veri aynı zamanda sigmoid fonksiyona tabi tutularak sonraki nörona giriyor.  
(gelen verileri **sigmoid fonk**. sokuyor. 1 e ne kadar yakınsa o veriler o kadar tutuluyor. 0 a ne kadar yakına o veri o kadar unutulması gerekiyor demek.)





\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

6 ya kadar datayı alacak ve 7. datayı tahmin edecek. Bu tahmin ile gerçek 7 değerini karşılaştırıp bir error bulacak (mesela 0.1) Bunu uygulayarak bu sefer 2 ile 7 arasındaki datayı alıp 8. datayı tahmin edecek ve bunu 8 ile karşılaştırıp örneğin 0.05 errorunu bulacak ve sonraki bölüme uygulayacak. Böylece errorları düşüre düşüre yani model eğitile eğitile devam edecek. Ta ki train seti bitene kadar.



* yeni veri olmaksızın, hataların en aza indirgenmiş haliyle tahminini yapacak
* bir sonraki tahminde, bir önceki hata oranı ile yapılan tahminden beslendiği için artık her defasında hata biraz daha artacaktır. dolayısıyla forecast'imizin çok uzun soluklu olması mümkün değildir.
* mesela modele 1 aylık veriyle eğitim yaptırdıysan en fazla 1 aylık forcast yapabilirsin. 1 haftalık veriyle eğitim yaptırırsan 1 haftalık forecast yaptırabilirsin

