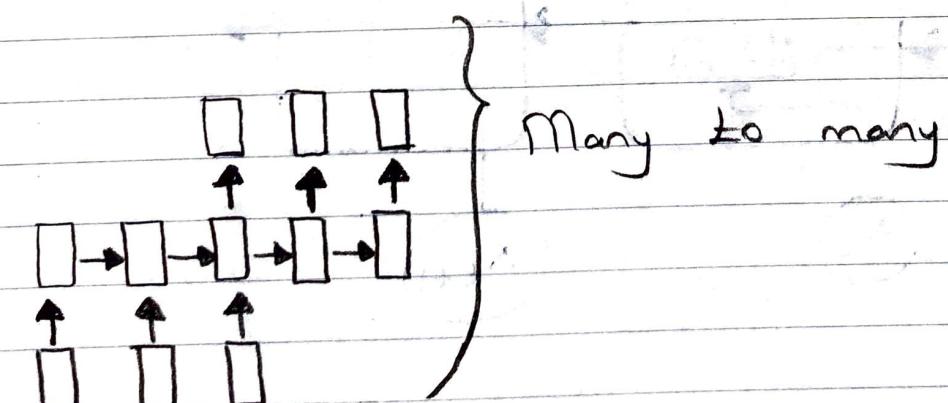
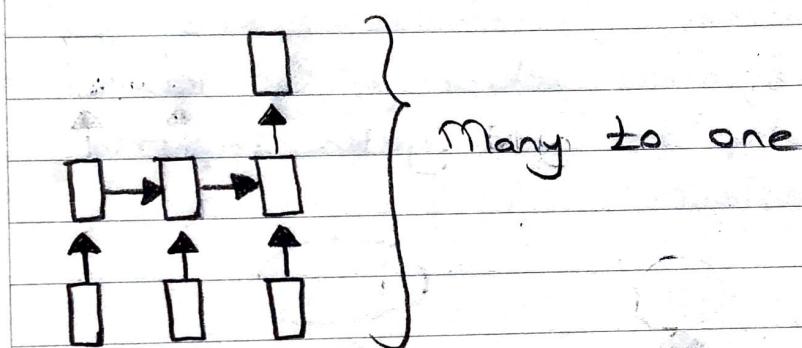
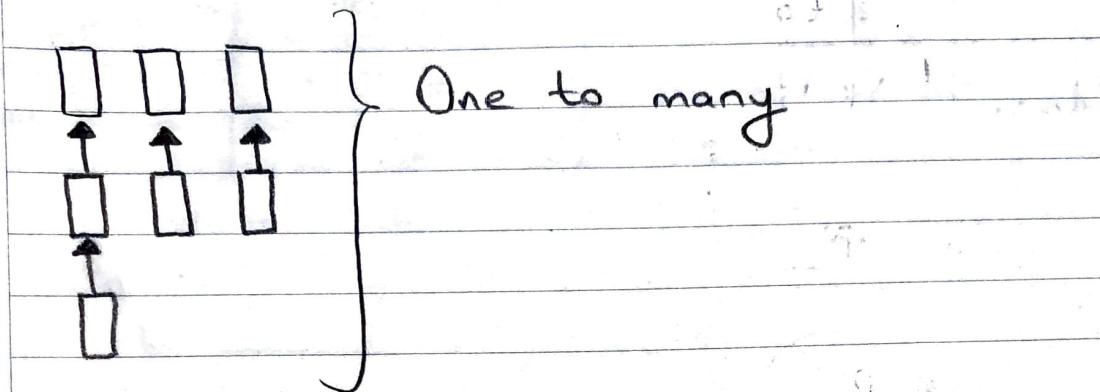








## Types of RNN :

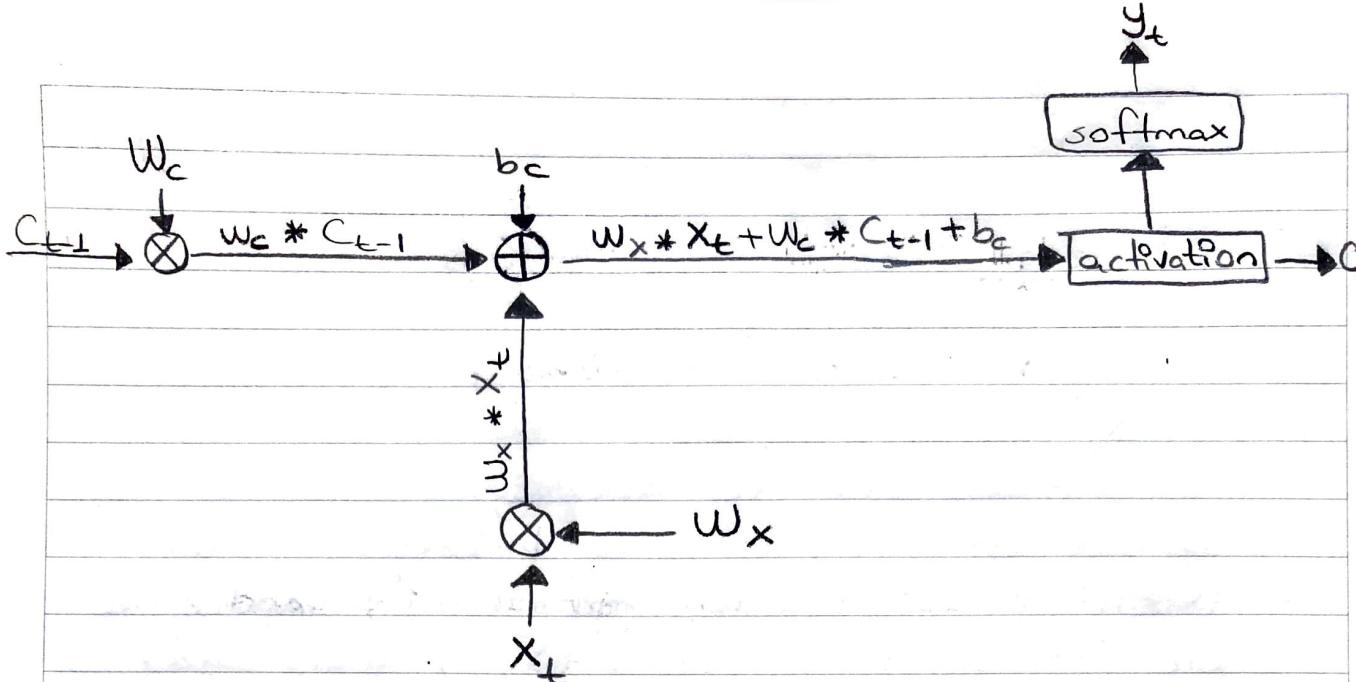




$x_t$  :  $t$  zamanda nörona  $-t-1$  zamanından  
gelen çıktı ( $h_{t-1}$ ) beslendi.

$x_t$  ve  $h_{t-1}$  vektörel olarak birbirlerine concatenated oldu ve aktivasyon fonksiyonundan gerek (default=tanh) regularization işlemi (veriler -1 ile 1 arasına sıkıştırıldı) yapıldı ve bir sonraki time step'e ( $x_{t+1}$ ) geçiş oldu.

- Many to one'da veriler bir sonraki katmana sequence zamanlandıgı zaman toplu olarak geceler.



$x_t$ : t'deki input.

$c_{t-1}$ : t-1'deki cell state.

$c_t$ : Nöronun çıktıları.

$w$ : Ağırlık

$b$ : bias

$y_t$ : Output

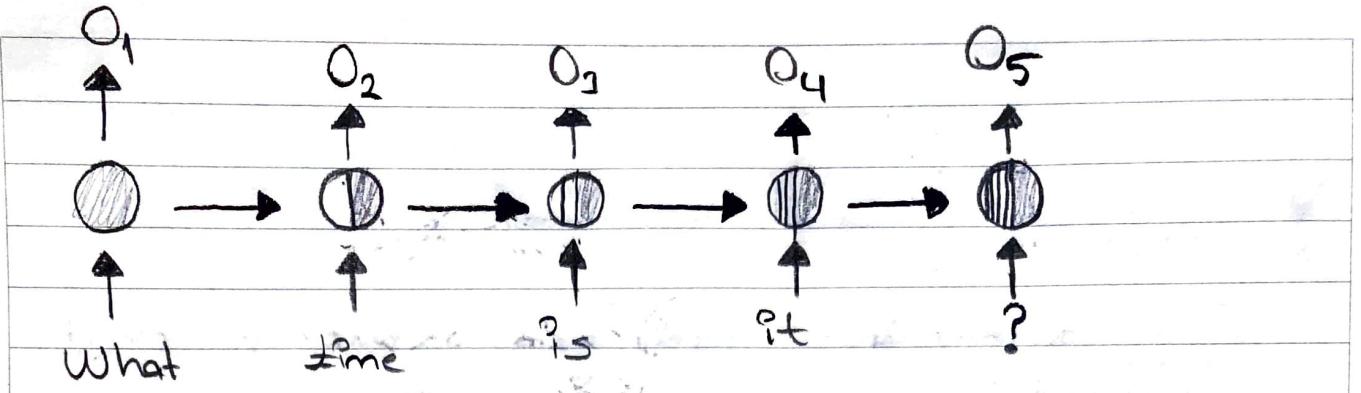
$\otimes$ : Çarpma

$\oplus$ : Toplama

 Hidden state: Short term memory

Cell state: Long term memory

- \*  $X_t$ 'de girdi gelmiş,  $W_x$  ağırlığı ile çarpılmış.
- \*  $C_{t-1}$ 'de bir önceki time step'ten gelen çıktı da  $W_c$  ağırlığı ile çarpılmış.
- \* Bu iki değeri vektörel olarak toplanmış  $\oplus$
- \*  $b_c$  bias değeri eklenmiştir.
- \* Bulunan değer activation fonksiyonundan geçerek bir sonraki time step'e ( $C_t$ ) çıktı olacak şekilde ayırmış.
- \* softmax kısmı, bütün layer'lar bittikten sonra ki output layer kısmı.



\* Bu tür kelimeler ayrı vektörler halinde RNN nöronlarına beslenir.

\* "What" kelimesi geldiğinde, nöron ?'a daha önce gelen bilgi olmadığını için tüm bilgi budur.

\* Sonraki aşamada "What" kelimesi sonraki time step'te aynı nörona beslenir ve yeni bir kelime daha getir, "Time". Bu iki kelime yaklasık olarak eşit ağırlıkta tensil edilir.

\* Sonraki time step'te "What time" aynı nörona gen beslenecek, yeni girdi geldi, "is". Yeni gelen veri ağırlık olarak daha fazla yer tutar, önceki veriler unutulmaya baslar, ağırlıkları azalmaya baslar.

\* En son step'te "What"ın ağırlığı oldukça azalır. Halbuki cümleyi tamamlama adına "What time" kısmı önemli.

Bu durum, simple RNN'lerin "short term memory" den kaynaklanan dezavantajıdır.

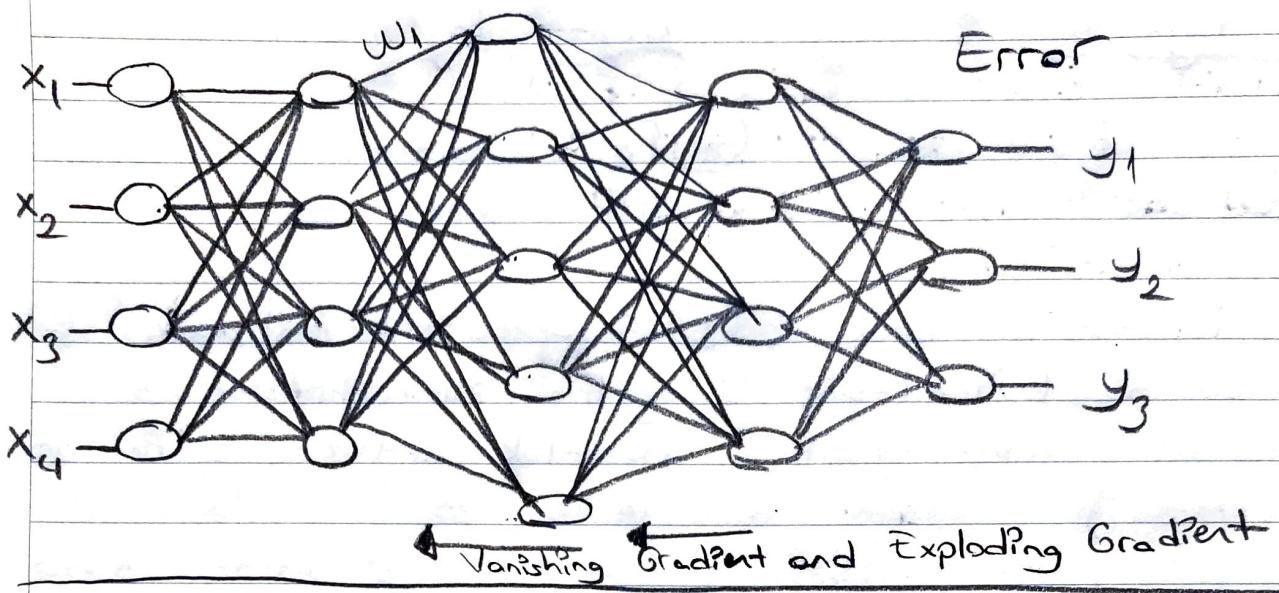
## RNN

### Avantajları

### Desavantajları

- ★ Sequence datalarında çalışma avantajı
- ★ Convolutional layerlarda çalışırken de kullanılabilir.  
(Resimlerdeki pixeller arası ilişkiler varsa.)
- ★ Fazla parametresi olduğu için eğitimi zor.
- ★ Sequence'lerde çalışmak da eğitimi zorlaştırır.  
(Uzunluk arttıkça eğitim zorlaşır.)
- ★ Parametre çokluğu ve uzun katmanlardan dolayı "Gradient exploding" ve "vanishing" problemleri ortaya çıkar.
- ★ Short term memory problem

## Exploding and Vanishing Gradient



ANN'de modeller eğitiliyordu, y aksasında prediction yapıldıktan hatta değer alınıyordu, sonra backpropagation ile türler alınarak model parametrelerini güncelleniyordu.

RNN'de Vanishing Gradient ve Exploding Gradient de backpropagation sırasında ortaya çıkar.

Modellerdeki katman sayıları arttıkça bu sonuçlarla karşılaşma olasılığı artar.

$\frac{\partial C_o}{\partial w_i} \rightarrow$  Cost function'un  $w_i$  ağırlığına göre türü.  
Bu tür "Chain Rule" ile alınır.





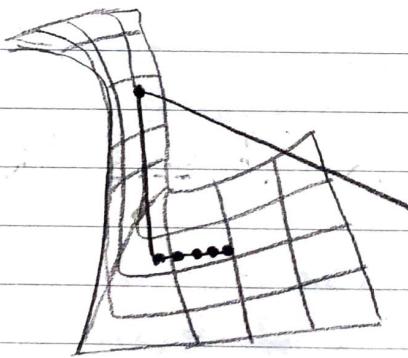


ReLU aktivasyon fonksiyonu back propagation'da değer 0 olmadığı zamanlarda değeri 1'e getir.

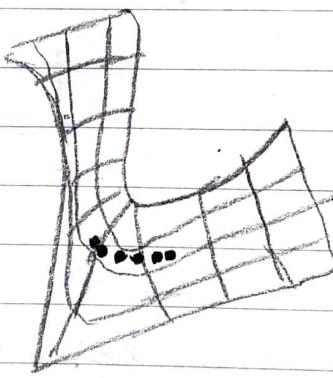
1, çarpma da etkisiz eleman olduğunu için back propagation'da öğrenmeyi engeller.  
(ReLU'nun türevleri de bu mantıkta işler.)

### Exploding Gradient Solution

Gradient Clipping : Gradient'in altına ve "üstüne threshold kayarak, bu değeri geçtiginde kırp" anlamına gelen parametreler var.



Without clipping



With clipping

Clipping yapılmadığında model yavaş yavaş öğrenmiş, gradient bir anda yükseldince patlama olmuş ve öğrenme de hızlanmış.

Bu yüzden geriye döndürerek tekrar öğrenmeye çalışmış. Bu tür öğrenmeklerin bosa gitmiş, model tekrar öğreneyecek. Bu da modelin eğitimi süresini artırır.

Uygun bir clipping değeriyle patlama olması engellenir.





\* Bulunan bu mean ( $M_B$ ) ve varyans ( $\sigma_B^2$ ) değerlerini normalization işlemiinde kullanır.

$$\hat{x}_i \rightarrow \frac{x_i - M_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad \left. \begin{array}{l} \\ \end{array} \right\} \text{Normalize}$$

Her bir  $x_i$  değerinin normalize edilmiş halini hesaplar.

\*  $y_i \rightarrow \gamma \hat{x}_i + \beta = BN_{\gamma, \beta}(x_i)$

Normalizasyon sonucunda  $\gamma$  ile çarpıp  $\beta$  ile topladı ki model bunları da eğitsin, her zaman  $\text{mean}=0$ ,  $\text{varianc}=1$  çıkması

$$y_i^{(k)} = \gamma^{(k)} \cdot \hat{x}_i^{(k)} + \beta^{(k)}$$

\*  $\beta=0$  (default)  
 $\gamma=1$  (default)

## ! Batch Normalization Nerede Kullanılır? !

- \* S-shaped fonksiyonlarda (sigmoid, tanh) bu fonksiyonlardan sonra kullanılmalı.
- \* ReLu vb. aktivasyon fonksiyonlarından önce kullanılmalı.

Bu şekilde kullanmak modelin öğrenmesini hızlandırır.

Batch Normalization Gradient Vanishing ve Exploding'i önlemek için ortaya çıktı fakat bazı yerlerde artıdışı da gözlemlenmiştir.

Batch Normalization'in modelde olan katkısı hala tartısma konusudur.

## Batch Normalization Faydalari:

- \* Eğitimi hızlandırır.
- \* Daha büyük learning rate'ler vermeye imkan sağlar.
- \* Başlangıç ağırlık değerleriyle ugrasmaya genel kalmaz.



Cell State : Uzun süreli hafıza burada muhafaza edilir.

■ LSTM ve GRU önemli verileri hatırlayıp önemlileri unutacak şekilde tasarlanmıştır.

\*) Cell state 'ı bir otoban gibi düşünebiliriz.

Forget Gate : Bu otobandan geçen verilerin hangisinin unutulması gerektiğini belirler.

Input Gate : Hangi verilerin sonraki steplerde hatırlanması gerektiğini belirler.

Output Gate : Bir sonraki time step'e hangi verilerin gitmesi gerektiğini belirler.

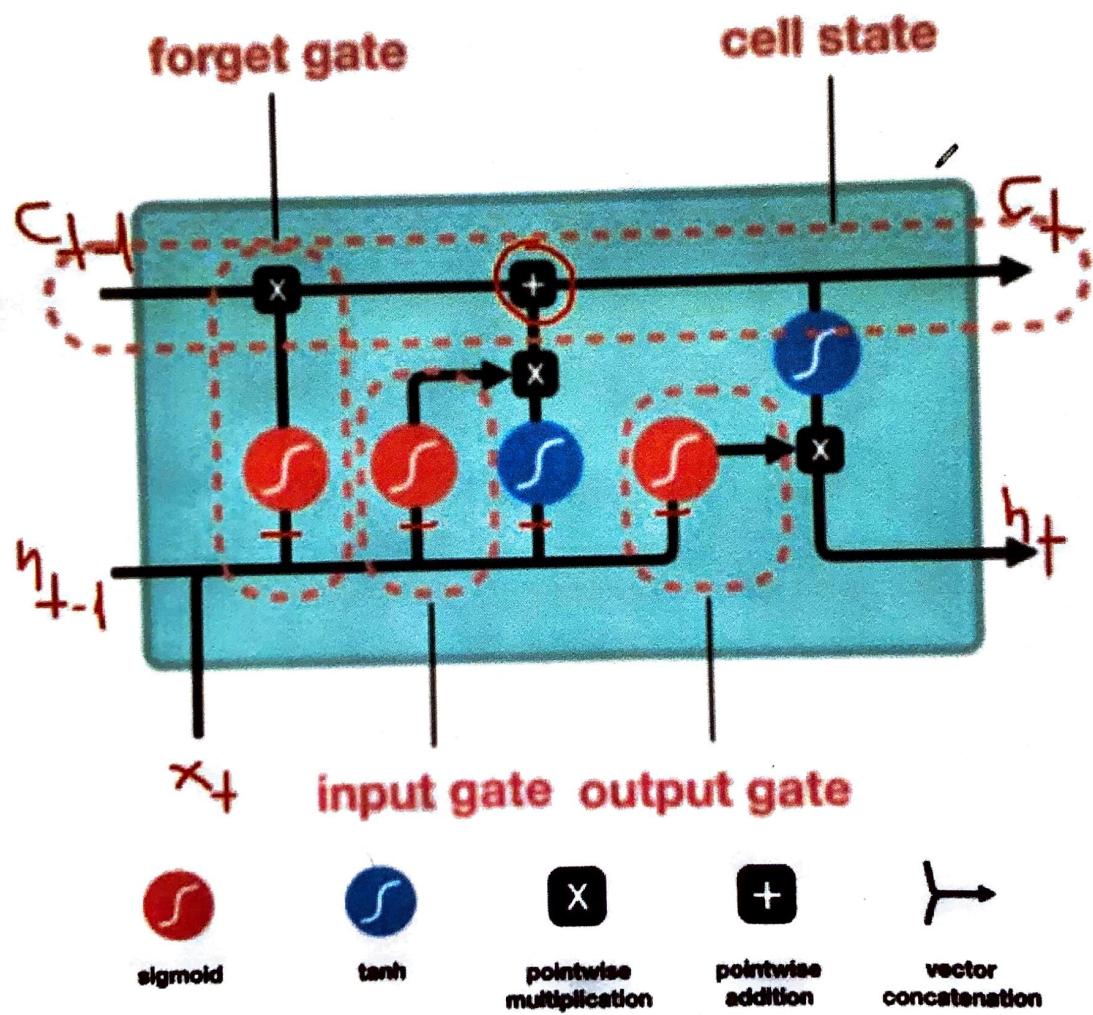
\*) "Onceki şekilde diyelim ki modeliniz eğitildi, artık nein önemli nein önemsiز olduğunu biliyor, ağırlıklar verildi.

\*) Yeni girdi ( $X_t$ ) ve onceki time stepten gelen veri ( $h_{t-1}$ ) geldi ve bunlar vektörel olarak concatenated edildi, sonra sigmoid (σ) fonksiyonuna geldiler.

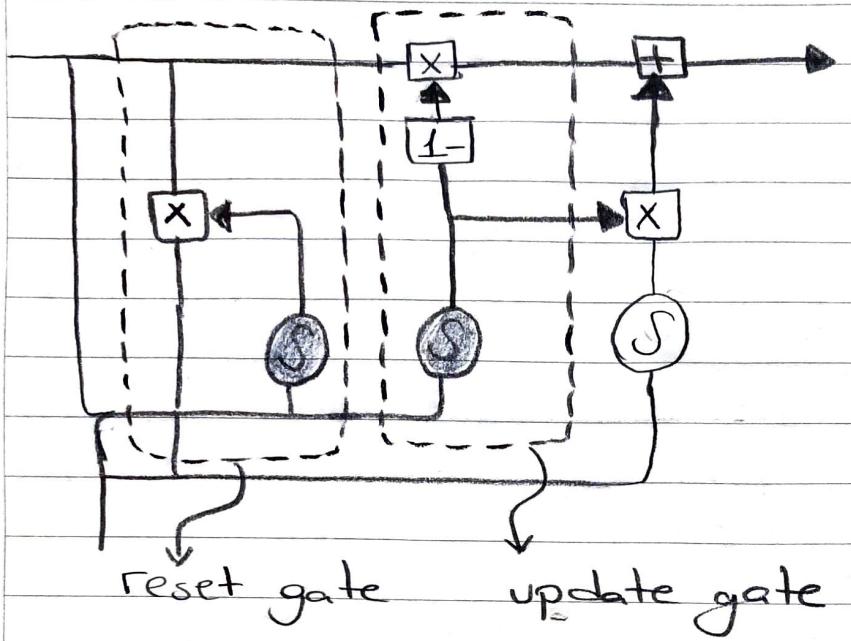
\*) Sigmoid fonk., verileri 0-1 arasına sıkıştırır.







## GRU (Gated Recurrent Unit),



( $S$ )  $\rightarrow$  Sigmoid

( $S$ )  $\rightarrow$  tanh

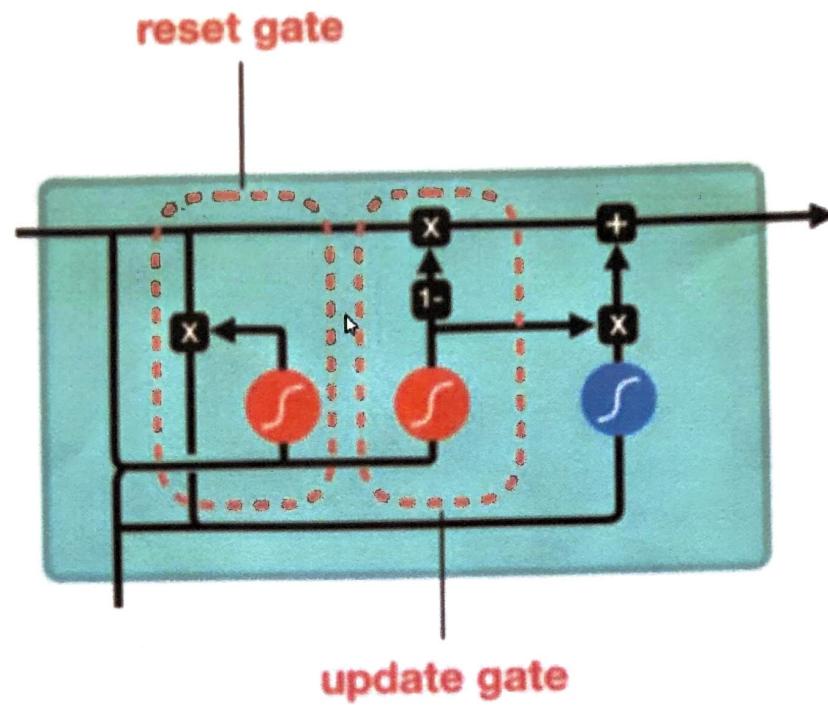
\* GRU, LSTM'den yaklaşık 15 yıl sonra keşfedilmiştir.

\* LSTM'in sistemi çok yavaşlattığını düşünen araştırmacılar GRU'yu keşfetmişler.

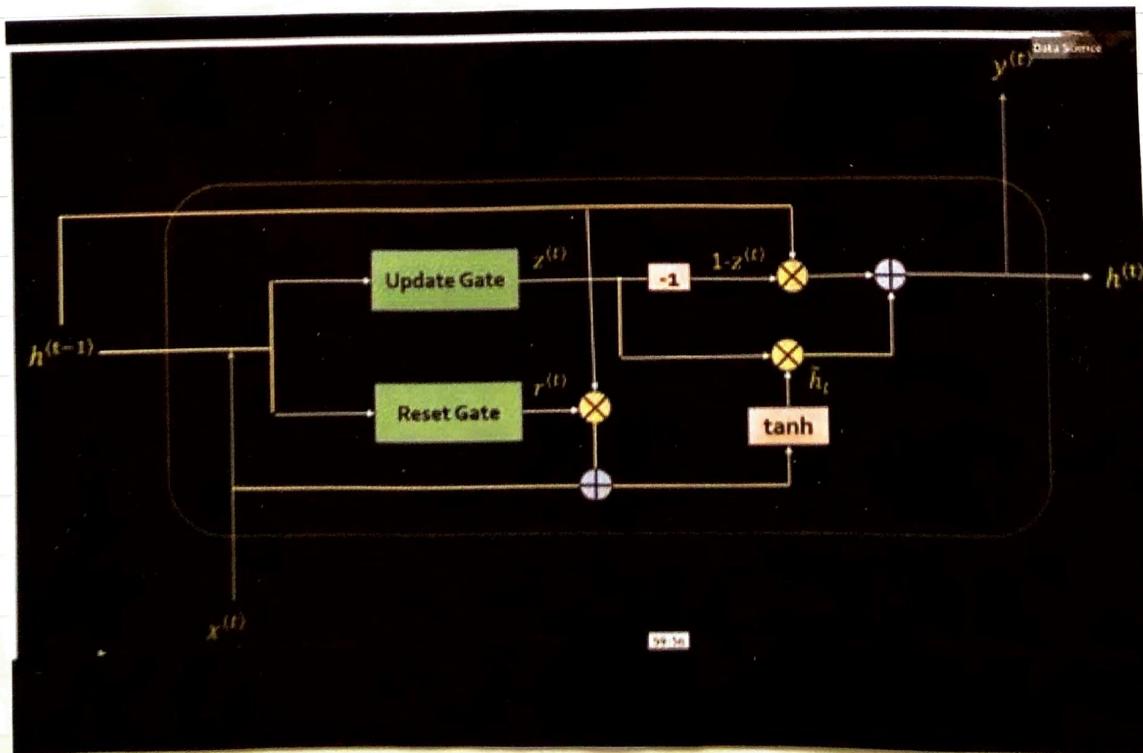
Update Gate : LSTM'deki forget ve input gate yerine gelmiş.

Reset Gate : LSTM'deki output gate yerine gelmiş.

\* LSTM ile GRU mertipi genel olarak aynıdır.

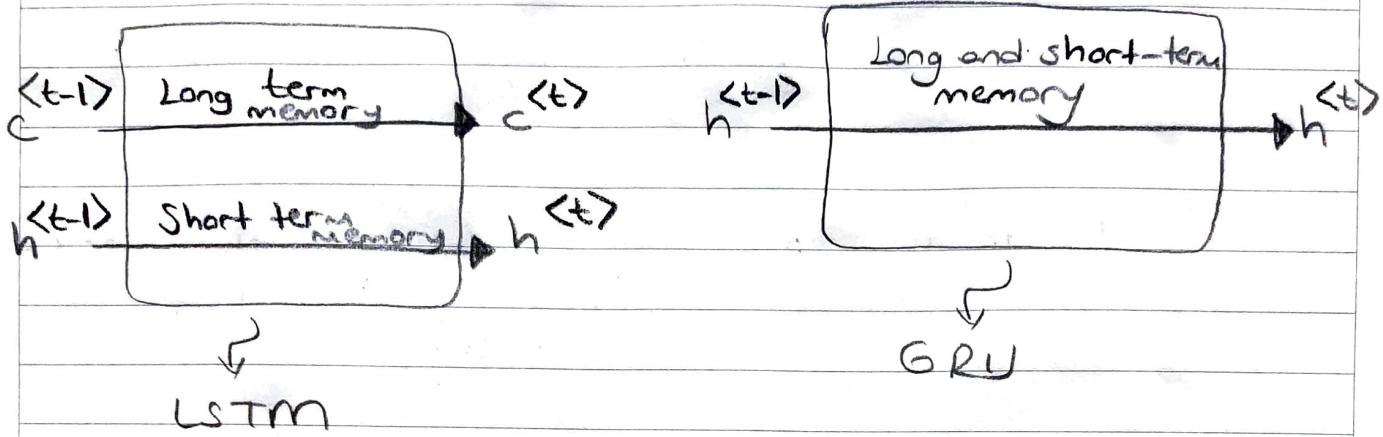


sigmoid	tanh	pointwise multiplication	pointwise addition	vector concatenation



LSTM'de  $\rightarrow$  Short term memory ve  
long term memory farklı yollarдан  
gider.

GRU'da  $\rightarrow$  short ve long term memory beraber  
yol alır.



\* LSTM'de nörona her seferinde 3 veri gelir:  
Yeni girdi, hidden state, cell state.

GRU'da yeni girdi ve hidden state.

Bu sayıların azalması işlem hızını artırır.

Parametre sayısı düşüğünde için işlem hızı da artar.

LSTM

GRU

\* 3 Gates (input, output, forget)      \* 2 Gates (reset, update)

\* Uzun sequence'larda daha iyi  
fakat hızı yavaş.

\* Hızlı daha iyi, daha popüler.

\* Invented: 1997 - 1994

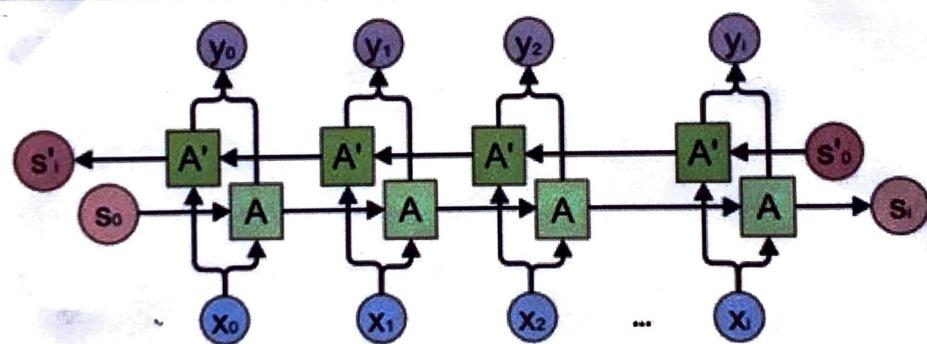
\* Invented 2014

## Bidirectional LSTM

LSTM' den farklı olarak datayı ters çevirdiğimizde o şekilde modelde yokluyor.

Text datalarında kelimelerin yer değiştirence anlam da değişebilir.

Dıştaki ve ters halıyla modele soktuğumuz dataların sonuçlarını toplayarak sonraki katmanı geliştirmemiz bazı durumlarda daha iyi sonuçlar almayı sağlar.



# Training

error  
nn

1	2	3	4	5	6	[?]	[7]
2	3	4	5	6	7	[?]	[8]
3	4	5	6	7	8	[?]	[9]



- \* 6 elemandan oluşan bir sequence olsun, 7. elemani tahmin etmek istiyorsunuz.

Gercekte olması gereken [7] ile tahmin edilen değer [?] arasında bir fark olacak.

- \* Eğer batch\_size=1 ise her batch'te bir backpropagation yapılacak ve ağırlıklar güncellenecek.

- \* Bir alt satırda sequence bir kayar, [8] değeri tahmin edilir ve gerçek değerle kıyaslanır, hata hesaplanır, backpropagation yapılır. Bu işlem sona kadar devam eder.

## Forecasting

52	53	54	55	56	54	Forecast Prediction
53	54	55	56	57	58.04	
54	55	56	57	58.04	59.09	
55	56	57	58.04	59.09	60.14	
56	57	58.04	59.09	60.14	61.28	
57	58.04	59.09	60.14	61.28		
						Completed Forecast

\*) 56 satırlı bir sequence olsun. Diyelim ki her bir sequence beş grupta halinde gruplandırıldı.

\*) İlk satırda 5 datadan 6. datayı tahmin etmeye çalışı, fakat bu 5 datayı karşılaştırabileceğimiz gerçek bir data yok. Burada tahmin edilen 54 forecast.

\*) Bir sonraki adımda bu forecast'i kullanacağız.

\*) 2. adımda 58 tahmin etmek gerektikten 58.04 tahmin ettik ve tahminlerimizi gidererek bozulmaya başladık.

\*) Yaptığımız prediction'lar üzerine yeni forecast'ları koymamız için forecast'lar git gide bozulmaya başlayacak.

\*) Örneğin, sequence'ler için 12 uzunluğunda bir length belirledik ve eğitimde bu şekilde yaptık. Model eğitildikten sonra da 12 adım ilerleye kadar tahmin yapabilir. Daha ilerisine gidilirse tahminler iyice bozulur.