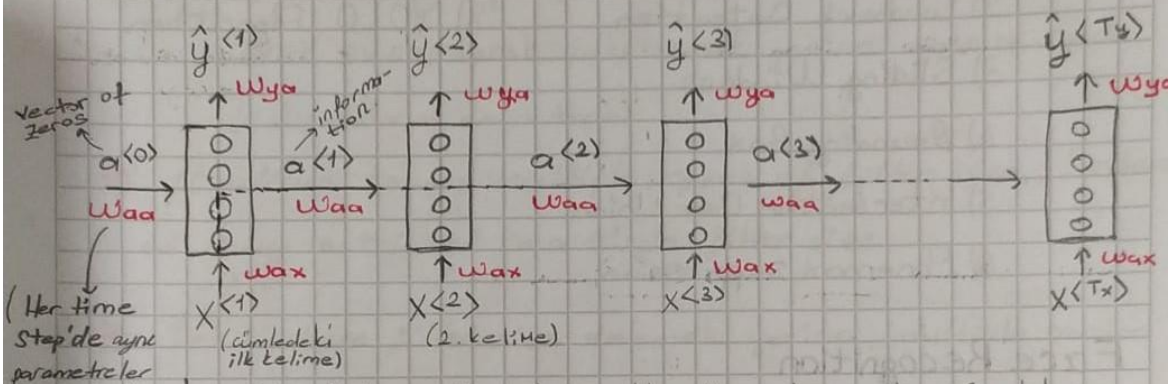
12) Recurrent Neural Networks

Recurrent Neural Network Model:

Simple Unidirectional Neural Network Architecture:



↳ Use 'information from the inputs earlier in the sequence; but not information later in the sequence.

$\hat{y}^{\langle 3 \rangle}$ prediction'u yaparken sadece $X^{\langle 3 \rangle}$'den değil; $X^{\langle 1 \rangle}$ ve $X^{\langle 2 \rangle}$'deki information'lardan da yararlanır.

Unidirectional modelde, $y^{(3)}$'ü tahmin ederken önceki information'ları kullanır; ancak x_4, x_5, x_6 - info'larını kullanamaz.

$\left(a^{\langle 1 \rangle} = h^{\langle 1 \rangle}\right)$

Forward Propagation:

$$a^{\langle 0 \rangle} = \vec{0}$$

$$a^{\langle 1 \rangle} = g_1(W_{aa}\, a^{\langle 0 \rangle} + W_{ax} X^{\langle 1 \rangle} + b_a)$$

$$\hat{y}^{\langle 1 \rangle} = g_2(W_{ya}\, a^{\langle 1 \rangle} + b_y)$$

$$a^{\langle t \rangle} = g(W_{aa}\, a^{\langle t-1 \rangle} + W_{ax} X^{\langle t \rangle} + b_a)$$

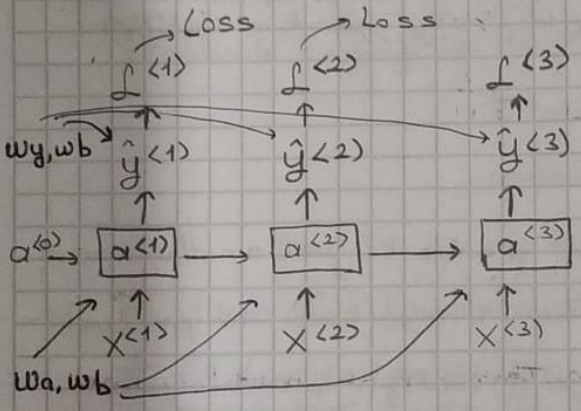
$$\hat{y}^{\langle t \rangle} = g(W_{ya}\, a^{\langle t \rangle} + b_y)$$

Backpropagation:

$$W_a = W_a - \frac{\alpha * \partial Loss(\hat{y}, y)}{\partial W_a} \qquad W_y = W_y - \frac{\alpha * \partial Loss(\hat{y}, y)}{\partial W_y}$$

$$W_x = W_x - \frac{\alpha * \partial Loss(\hat{y}, y)}{\partial W_x} \qquad b_a = b_a - \frac{\alpha * \partial Loss(\hat{y}, y)}{\partial b_a}$$

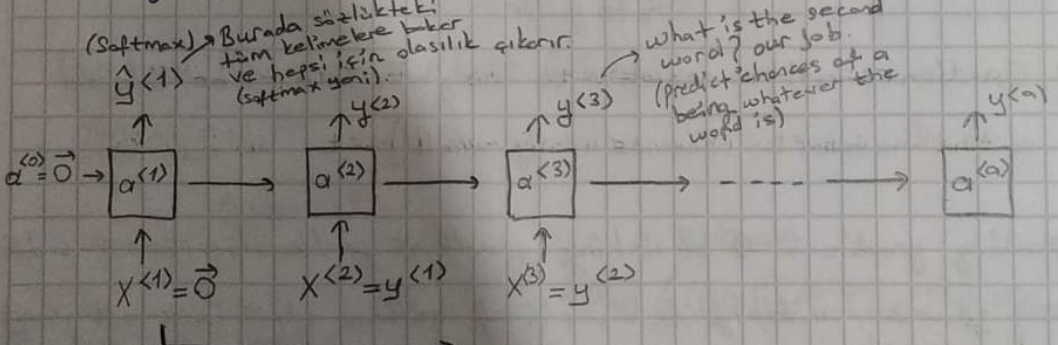
$$b_y = b_y - \frac{\alpha * \partial Loss(\hat{y}, y)}{\partial b_y}$$



$$\mathcal{L}^{\langle t\rangle}(\hat{y}^{\langle t\rangle}, y^{\langle t\rangle}) = -y^{\langle t\rangle} \log \hat{y}^{\langle t\rangle} - (1 - y^{\langle t\rangle}) \cdot \log(1 - \hat{y}^{\langle t\rangle})$$

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}^{\langle t\rangle}(\hat{y}^{\langle t\rangle}, y^{\langle t\rangle})$$

Language Model and Sequence Generation



we will give correct words. (Her adımda önceki adımlardaki doğru sözcükleri de almış oluyuz.)

Bu yüzden $x^{\langle 2\rangle} = y^{\langle 1\rangle}$ ... olur.

*

$$\mathcal{L}(\hat{y}^{\langle t\rangle}, y^{\langle t\rangle}) = -\sum_i y_i^{\langle t\rangle} \log \hat{y}_i^{\langle t\rangle}$$

$$\text{Overall } \mathcal{L} = \sum_t \mathcal{L}^{\langle t\rangle}(\hat{y}^{\langle t\rangle}, y^{\langle t\rangle})$$

Ör: $P(y^{\langle 1\rangle}, y^{\langle 2\rangle}, y^{\langle 3\rangle})$ → 3 sözcüklü cümle.

$$= P(y^{\langle 1\rangle}) \cdot P(y^{\langle 2\rangle} \mid y^{\langle 1\rangle}) \cdot P(y^{\langle 3\rangle} \mid y^{\langle 1\rangle}, y^{\langle 2\rangle})$$

↳ Probability of that 3-word sentence

Sequence Generation

↳ Character-level Language model veya Word-level Language model ile yapılabilir.

News

↳ Modeli haber-ler ile train edersek; "President says ..." gibi cümle generate etmesi çok muhtemel.

Shakespeare

↳ Shakespeare ile train edersek "The mortal moon ..." gibi birşey generate edilebilir.

Vanishing Gradients

Vanishing gradient ⇒ Your gradients get smaller and smaller in magnitude as you back-propagate through lower layers. Her katmandan geçerken biraz daha düşecek olan bu gradyan değerleri, baştaki katmanlara doğru sıfıra yaklaşmaya başlar, ve yapay sinir ağı öğrenemez hale gelir.

(Gradients with respect to weights in earlier layers of the network becomes really small)

Exploding gradient ⇒ Vanishing gradient probleminin tam tersidir.

(If we multiply a bunch of terms that are all greater than one, we gonna get something greater)

Backprop ile birlikte ağırlıkların yok olmasından dolayı RNN'de vanishing gradient problemi ile karşılaşmak çok olasıdır. Her bir katmandaki ağırlıklar Zincir Kuralı üzerinden ayarlandığından, gradient değerleri geriye doğru ilerledikçe katlanarak küçülecek ve sonunda

yok olacaktır. Örneğin; MLP uygulamasını ele alıyor olalım, Fiktya göre "What" ve "time" 'dan gelen bilgilerin tamamen ortadan kalktığını düşündüğümüzde "is" ve "it" tahminini nasıl yapabiliriz?

↳ Problemin kaynağı ⇒ 10.000 zaman adımı büyüklüğünde bir dizi verisini işleyen bir RNN, optimize edilmesi zor olan 10.000 derin katmana sahiptir.

* Uzun dizi boyutuna sahip RNN'lerde ortaya çıkar.

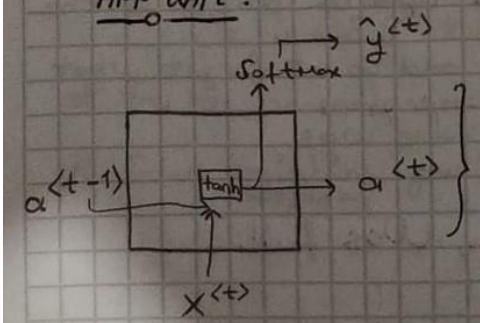
* Bu durumda, daha önceki katmanlar herhangi bir öğrenme yapamazlar. Bu da ilk Layer'lardaki bilgileri kaybetmelerine neden olur. (Basic RNN, Long-range dependency'leri yakalayamaz)

Vanishing (Kaybolan) Gradient problemini

- GRU
- LSTM yöntemleri ile çözebiliriz.

Gated Recurrent Unit (GRU)

RNN Unit:



$$a^{\langle t \rangle} = g(W_a[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_a)$$

(g: tanh)

GRU unit'i diğer sayfada gösterelim.

GRU (simplified):

* c : memory cell (provide a bit of memory to remember)

$c^{\langle t \rangle} = a^{\langle t \rangle}$

$\tilde{c}^{\langle t \rangle} = \tanh\left(W_c [c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_c\right)$

Γ_u (gamma u) = Gate ⇒ u stands for update gate.

$\Gamma_u = \sigma\left(W_u [c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_u\right)$ ← sigmoid func.

↳ always between zero and one.

$\tilde{c}^{\langle t \rangle}$ → Candidate for replacing c of it.

* Gate decides whether or not we actually update the c value using $\tilde{c}$.

* Gate also decides whan do you update these c values.

Specific equation used in GRU:

0

1) $c^{\langle t \rangle} = \Gamma_u * \tilde{c}^{\langle t \rangle} + (1 - \Gamma_u) * c^{\langle t-1 \rangle}$

($\tilde{c}$ ← candidate value)

$c^{\langle t \rangle}$ ↓ actual c value

a) when; $\Gamma_u = 0$ Do not update. Hung on to the value and do not forget what this value was.

The Cat (singular.) which already ate ..., Was (singular.) full.

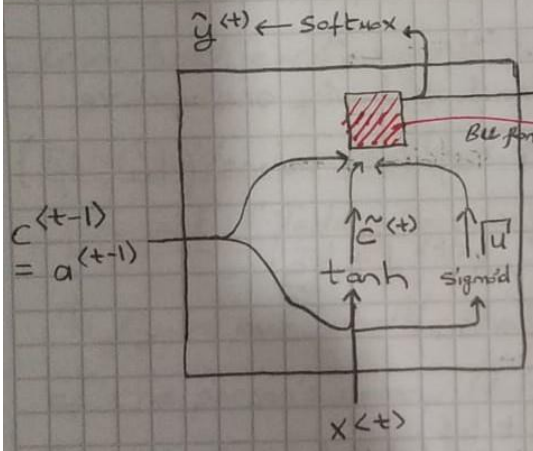
Let's say: $c^{\langle t \rangle} = 1$

GRU unit memorize the value of c all the way until Was.

$\Gamma_u = 1$ $\Gamma_u = 0$ $\Gamma_u = 0$ ----

decide when you update c value.

b) * when; $\Gamma_u = 1$; update the value of c with new candidate value.



$$\{c^{\langle t \rangle} = \Gamma_u * \tilde{c}^{\langle t \rangle} + (1-\Gamma_u) * c^{\langle t-1 \rangle}$$

↳ element wise.

* Γ_u allow neural network to go on even very long range dependencies. (Cat..... Was,) even they are separated by a lot of words in the middle.

Full GRU ↳ önceki fonksiyona çok ufak bir ekleme.

$$\tilde{c}^{\langle t \rangle} = \tanh(W_c[\Gamma_r * c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_c)$$

$$\Gamma_r = \sigma(W_r[c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_c)$$

$$\Gamma_u = \sigma(W_u[c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_u)$$

$$c^{\langle t \rangle} = \Gamma_u * \tilde{c}^{\langle t \rangle} + (1-\Gamma_u) * c^{\langle t-1 \rangle}$$

* Try to have longer range connections and
* solve vanishing gradient problems.

Long Short Term Memory (LSTM)

Slightly more powerful and more general version of the GRU.

* $a^{\langle t \rangle} \neq c^{\langle t \rangle}$
* New property of LSTM is instead of having one update gate control, we gonna have 2 seperate terms. (Γ_u, Γ_f)

$$\tilde{c}^{\langle t \rangle} = \tanh\left(W_c [a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_c\right)$$

(update) $$\Gamma_u = \sigma\left(W_u [a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_u\right)$$

(forget) $$\Gamma_f = \sigma\left(W_f [a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_f\right)$$

(output) $$\Gamma_o = \sigma\left(W_o [a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_o\right)$$

$$c^{\langle t \rangle} = \Gamma_u * \tilde{c}^{\langle t \rangle} + \Gamma_f * c^{\langle t-1 \rangle}$$

↳ forget gate.

$$a^{\langle t \rangle} = \Gamma_o * \tanh(c^{\langle t \rangle})$$

fig 1:

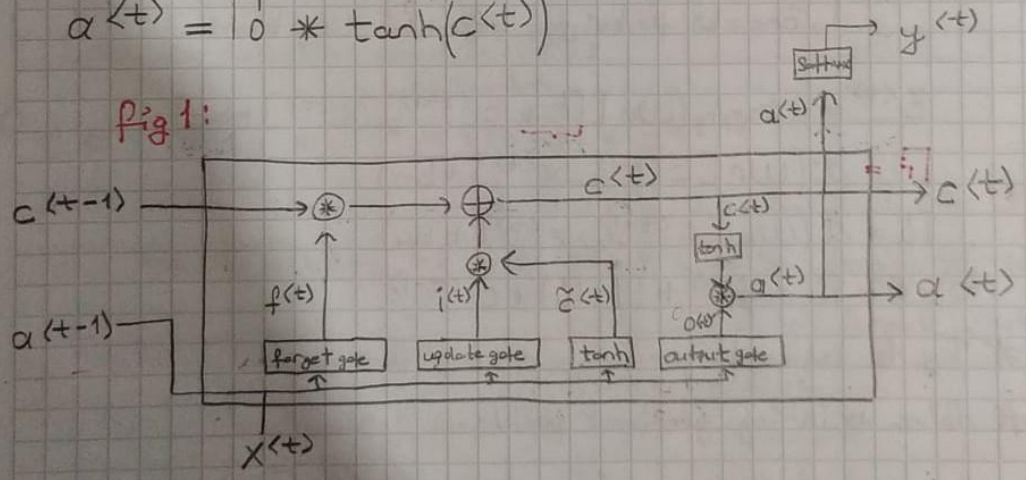
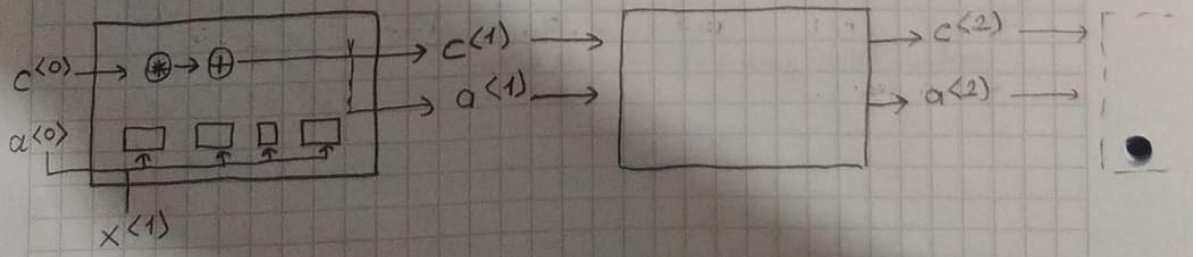


fig 2:



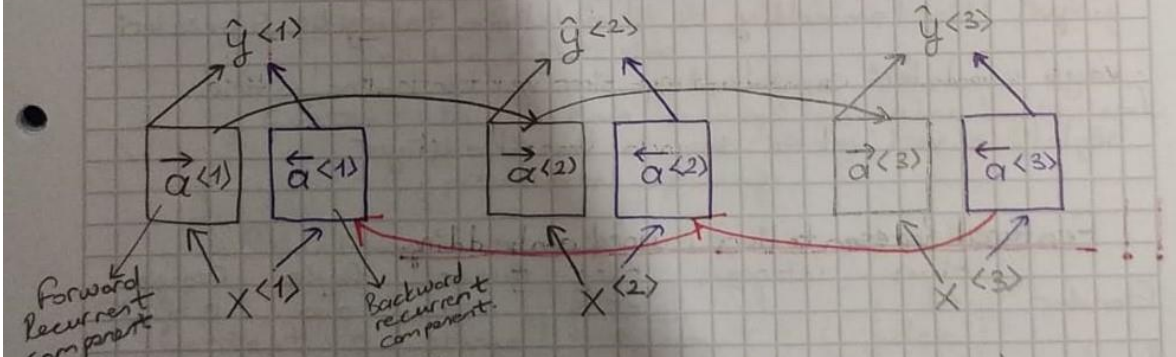
Bidirectional RNN (BRNN)

Lets you at a point in time to take information from both earlier and later in the sequence.

He said, "Teddy bears are on sale!"

He said, "Teddy Roosevelt was a great President!"

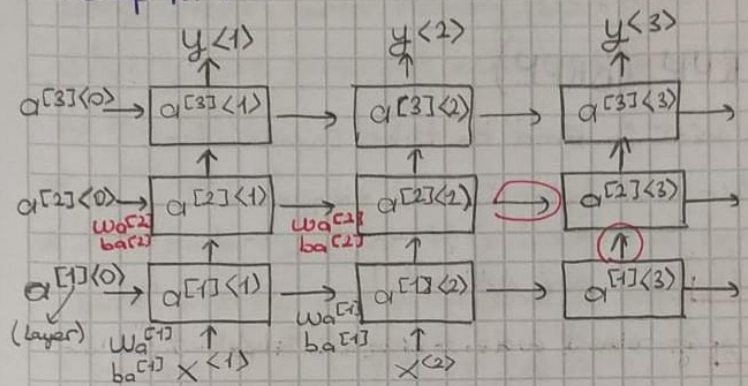
To figure out whether the 3rd. word Teddy is a part of the person's name, it is not enought to just look at the first part of the sentence shown as red colour. So to tell, if $\hat{y}^{<3>}$ = Teddy should be zero or one, we need more information than just the 3 words, because the first 3 words does not tell you whether they are talking about Teddy bears or former US president, Teddy Roosevelt.



$$\hat{y}^{<t>} = g\left(W_y\left[\vec{a}^{<t>}, \overleftarrow{a}^{<t>}\right] + b_y\right)$$

Information from past, present and future is taken.

Deep RNN:



* Her layer kendi parametrelerine sahiptir $(Wa^{[l]}, ba^{[l]})$

$$a^{[2]\langle 3\rangle} = g\left(Wa^{[2]}\left[a^{[2]\langle 2\rangle}, a^{[1]\langle 3\rangle}\right] + ba^{[2]}\right)$$

Natural Language Processing and Word Embeddings

Word Embedding: Metin verisini makinenin anlayacağı hale getirmemiz gerekiyor. Burada word embedding devreye giriyor. Word embedding bir dil modelleme tekniğidir. Sözcükleri veya cümleleri sayısallaştırıp birer vektör haline getirir böylelikle veri vektör uzayında temsil ediliyor.

!! Featured Representation : word embedding

	Man	Women	King	Queen	Apple	Orange
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
Size						

300

Birçok feature all the same, apple ve orange için

"Man" kelimesini represent etmek için 300 dimensional vector kullanılır.

y (Feature size)

Apple ve orange'ın feature'larının benzer olması, öğrenme algoritmasının işini kolaylaştırır. Örneğin;

* I want a glass of orange juice
* I want a glass of apple juice

] Generalize.

! Transfer Learning, Word embedding'de sıkça kullanılan bir metot'dur.

! 2D dimension'da visualize etmek için (word'leri) ve gruplamak için t-sne algoritması kullanılır.
↳ it takes 300-D data and it maps it in a very non-linear way to a 2D space.

Learning Word Embeddings:

1) Word2Vec: Taking as input one word like "orange" etc. and then trying to predict some words skipping a few words from the left or the right side.
Algorithm (Skip-Gram) Model.
↳ Context: Kaç kelime ve hangi kelimeler kullanılarak prediction yapılacağı.
↳ Skip-Gram modelde, just one word kullanılır.

Or) Context c ("orange") → Target t ("juice")
Target t ("my")

Take input one word (context word)
↳ predict what comes little bit before or after the context word.

Not ⇒ 2 farklı Word2Vec yaklaşımı vardır, biz Skip-Gram'i işledik, (Diğeri de CBOW'dur)

Skip-Grams Akış:

1) Pick a word to be the context word.
2) Pick another word within some window (örneğin ±5 context word) as a target word.

↳ Bu algoritma Softmax Classifer kullanıyor (Bu yüzden computationaly expensive)

↳ Bu problemi çözmek için "Negative Sampling" metodu kullanılır ve binary classification yolu ile problem çözülür.

2) GloVe Algorithm:

↳ Skip-Grams kadar yoğun kullanılmaz.
(Word2Vec'e eklenti olarak geliştirildi)

!!! Converting text data into numerical data, which is called Vectorization or in the NLP word, it is called word embeddings.