

# Redes Neurais Recorrentes





Jorge K. S. Kamassury

EEL7514/EEL7513 - Tópico Avançado em Processamento de Sinais

EEL410250 - Aprendizado de Máquina

EEL / CTC / UFSC

# Exemplos de dados sequenciais

Aplicações	X		Y
Reconhecimento de voz		→	Os alunos estão animados
Geração de música	∅	→	
Análise de sentimentos	"Não há nada de bom no filme"	→	
Análise de sequência de DNA	AGCCCCTGTGAGGAACTAG	→	AGCCCCTGTGAGGAACTAG
Tradução automática	Voulez – vous chanter avec moi?	→	Quer cantar comigo?
Reconhecimento de ações em vídeo		→	Correndo
Reconhecimento de entidade nomeada	Ontem, Rafael encontrou a Laís na sala de aula	→	Ontem, Rafael encontrou a Laís na sala de aula

# Notação

Exemplo motivador: reconhecimento de entidade nomeada

- ▶  $X$ : "Harry Potter and Hermoine Granger invented a new spell";

	$x^{<1>}$	$x^{<2>}$	$x^{<3>}$	$x^{<4>}$	$x^{<5>}$	$x^{<6>}$	$x^{<7>}$	$x^{<8>}$	$x^{<9>}$
$X$	Harry	Potter	and	Hermione	Granger	invented	a	new	spell
	↓	↓	↓	↓	↓	↓	↓	↓	↓
$Y$	1	1	0	1	1	0	0	0	0
	$y^{<1>}$	$y^{<2>}$	$y^{<3>}$	$y^{<4>}$	$y^{<5>}$	$y^{<6>}$	$y^{<7>}$	$y^{<8>}$	$y^{<9>}$

- ▶  $T_x$  e  $T_y$  correspondem aos comprimentos das sequências de entrada e saída, respectivamente;
- ▶  $x^{(i)<t>}$  é o  $t$ -ésimo elemento na sequência de entrada do  $i$  exemplo de treinamento;
- ▶  $y^{(i)<t>}$  é o  $t$ -ésimo elemento na sequência de saída do  $i$  exemplo de treinamento.

# Representação das palavras

## Enfoque: Processamento de Linguagem Natural

- ▶ Lista de vocabulário;
  - ▶ Cada palavra terá um índice exclusivo com a qual pode ser representada;
  - ▶ A classificação é realizada em ordem alfabética (neste caso);
  - ▶ Os tamanhos de vocabulários em aplicações modernas são da ordem 100000 palavras;
- ▶ Inicialmente, usaremos a codificação **one-hot** para cada palavra no conjunto de dados.

# Representação das palavras

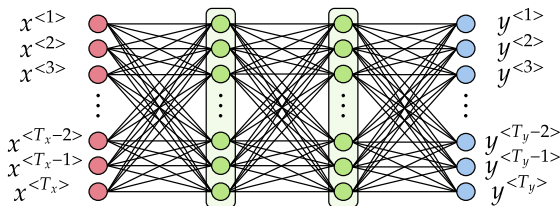
Enfoque: Processamento de Linguagem Natural

- ▶ Objetivo: usando uma representação de  $x$ , aprender um mapeamento via modelo de sequência para, então, obter uma sequência  $y$ ;
- ▶ Problema de aprendizado supervisionado.

Harry Potter and Hermione Granger invented a new speel

Vocabulary		$x^{<1>}$	$x^{<2>}$	$x^{<3>}$	...	$x^{<7>}$	$x^{<8>}$	$x^{<9>}$
1	a	0	0	0		1		
2	aaron	0	0	:		:		
	:	0	0	0		0		
367	and	0	0	1		0		
	:	:	:	:		:		
4075	Harry	1	0	0	...	0		
	:	:	:	:		:		
6830	Potter	0	1	0		0		
	:	0	:	0		0		
	:	0	0	0		0		
	Zulu	0	0	0		0		

# Porque não usar uma rede padrão?

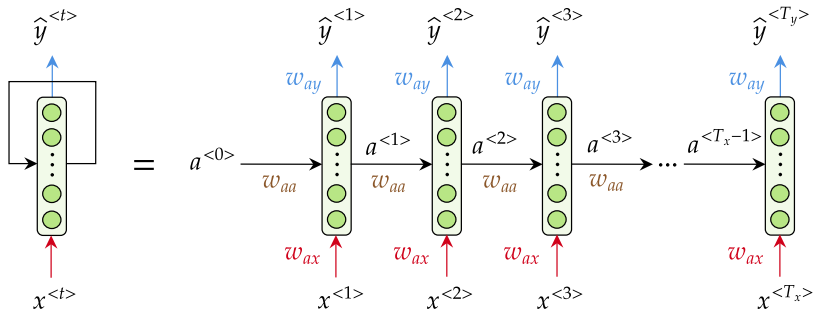


## Problemas:

- ▶ As entradas e as saídas podem ter comprimentos diferentes em exemplos diferentes ( $T_x \neq T_y$ );
- ▶ Rastreamento de dependências a longo prazo: informações sobre a ordem da sequência;
- ▶ Não compartilha *features* aprendidas em diferentes posições da sequência;
- ▶ A rede neural recorrente contorna ambos os problemas.

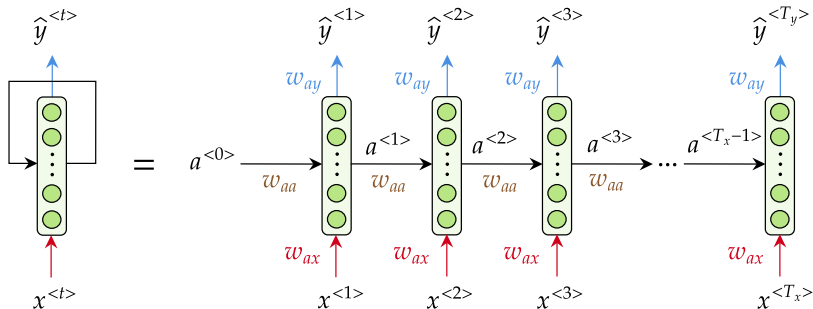
# Redes Neurais Recorrentes

- Exemplo: reconhecimento de entidade mencionada ( $T_x = T_y$ )



- $a^{<0>}$  é geralmente inicializado com zeros;
- Matrizes de pesos:  $W_{ax}$ ,  $W_{aa}$  e  $W_{ay}$ ;
- $W_{aa}$  é a *memória* que a RNN mantém das camadas anteriores.

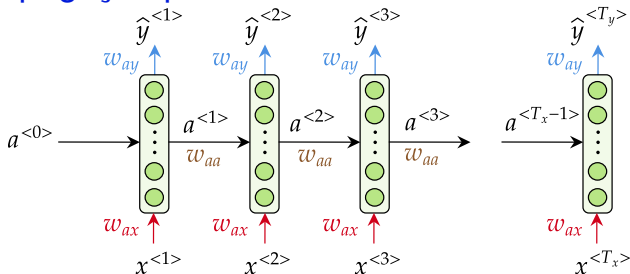
# Redes Neurais Recorrentes



- ▶  $a^{<t>} = f_W(a^{<t-1>}, x^{<t>})$ : estado oculto (estado da célula);
- ▶ A saída atual  $\hat{y}^{<t>}$  depende das entradas e ativações anteriores;
- ▶ Essa arquitetura não pode aprender com os elementos posteriores da sequência;
  - ▶ He said, **Teddy** Roosevelt was a great **President**.



## RNN: Propagação para frente

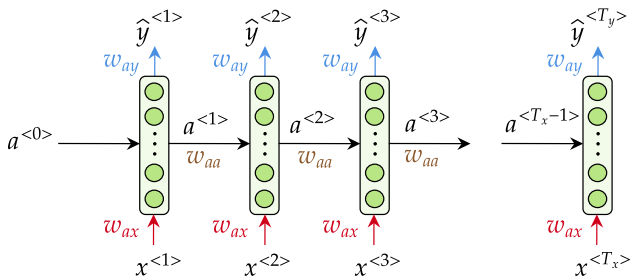


$$a^{<t>} = g(w_{aa}a^{<t-1>} + w_{ax}x^{<t>} + b_a) \quad (1)$$

$$\hat{y}^{<t>} = g(w_{ya}a^{<t>} + b_y) \quad (2)$$

- ▶ A função de ativação de  $a$  é geralmente Tanh ou ReLU;
- ▶ A função de ativação de  $\hat{y}$  depende da tarefa em questão.

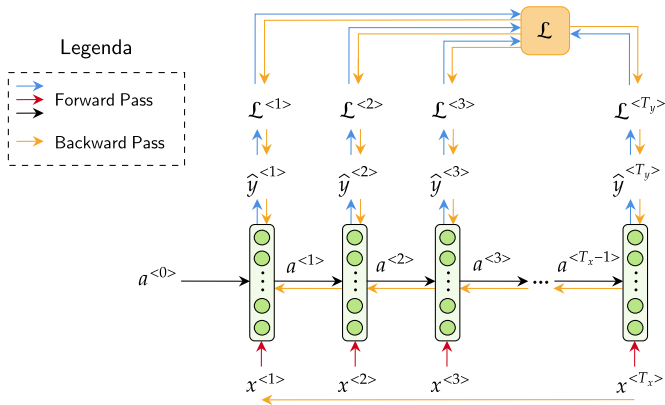
# RNN: Notação simplificada



$$\begin{aligned} a^{<t>} &= g(w_{aa}a^{<t-1>} + w_{ax}x^{<t>} + b_a) \\ &= g(w_a [a^{<t-1>}, x^{<t>}] + b_a) \end{aligned}$$

- ▶  $w_a = [w_{aa} | w_{ax}]$ : empilhados horizontalmente;
- ▶  $[a^{<t-1>}, x^{<t>}]$ : empilhados verticalmente;

# RNN: retropropagação através do tempo



- ▶ Função de perda:  $\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}(\hat{y}^{<t>}, y^{<t>})$ ;
- ▶ Retropropagação através do tempo: passamos a ativação  $a$  de um elemento de sequência para outro (como retrocedendo no tempo).

# Diferentes tipos de RNNs

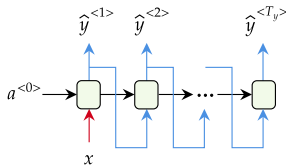
One to one

$$T_x = T_y = 1$$



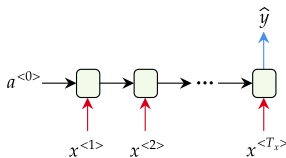
One to many

$$T_x = 1, T_y > 1$$



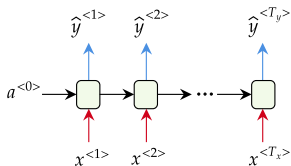
Many to one

$$T_x > 1, T_y = 1$$



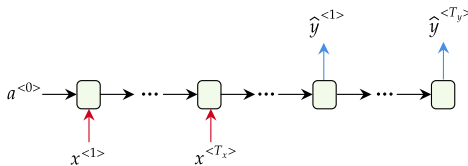
Many to many

$$T_x = T_y$$



Many to many

$$T_x \neq T_y$$



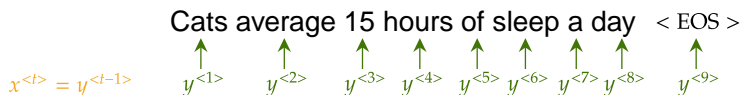
- Exemplos de aplicações: análise de sentimento, geração de música, tradução automática, etc.

# Linguagem e geração de sequência

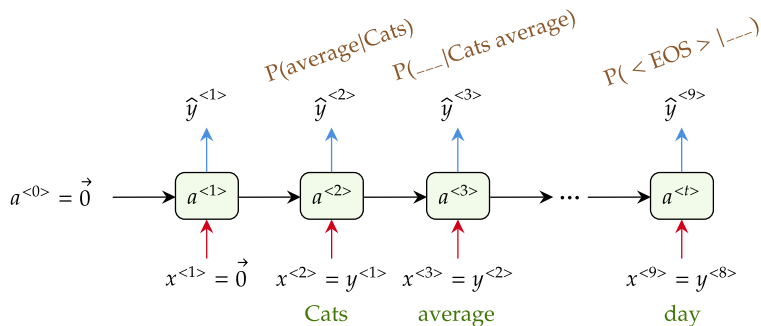
- ▶ O que é um modelo de linguagem?
- ▶ Exemplo (reconhecimento de voz):
  - ▶ The apple and **pair** salad/ The apple and **pear** salad.
  - ▶ Uma vez que **pair** e **pear** têm sons similares, como um sistema de reconhecimento de voz escolheria entre os dois termos?
- ▶ O trabalho de um modelo de linguagem é fornecer uma probabilidade de qualquer sequência de palavras dada.

# Como construir modelos de linguagem com RNN?

- ▶ De início, obter um conjunto de treinamento: um grande *corpus* de texto do idioma em questão;
- ▶ Tokenizar o conjunto de treinamento de forma a obter um dicionário (aplicação de *one-hot* para cada palavra);
- ▶ Uso de  $\langle \text{EOS} \rangle$  (para final de frase) e  $\langle \text{UNK} \rangle$  (para palavras desconhecidas)

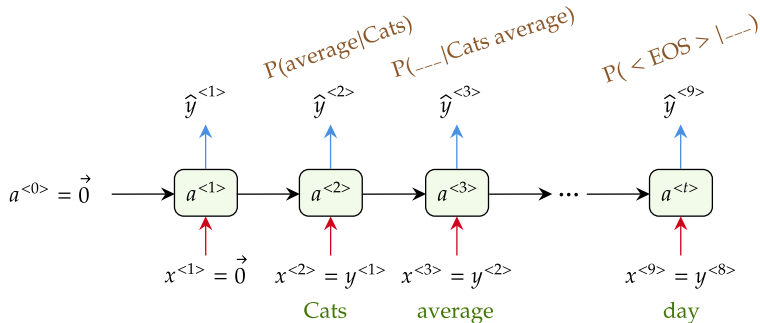


# Como construir modelos de linguagem com RNN?



- ▶ Função de perda:  $\mathcal{L}(\hat{y}^{<t>}, y^{<t>}) = - \sum_i y_i^{<t>} \log \hat{y}_i^{<t>}$ 
  - $i$  (p/ todos elementos do corpus) e  $t$  (p/ todos os passos temporais).
- ▶ Para prever a **próxima palavra**, alimentamos a sentença para a RNN e obtemos  $\hat{y}^{<t>}$ , classificando-o pela máxima probabilidade.

# Como construir modelos de linguagem com RNN?



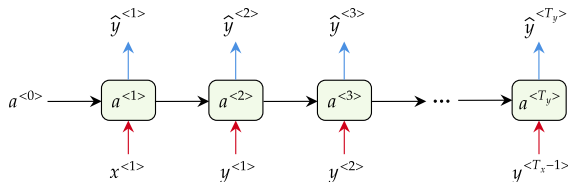
- Para calcular a probabilidade de uma frase, calculamos:

$$P(y^{<1>}, y^{<2>}, y^{<3>}) = P(y^{<1>}) P(y^{<2>}|y^{<1>}) P(y^{<3>}|y^{<2>}, y^{<1>})$$

- Isso é simplesmente alimenta a frase na rede e multiplicar as probabilidades (saídas)!

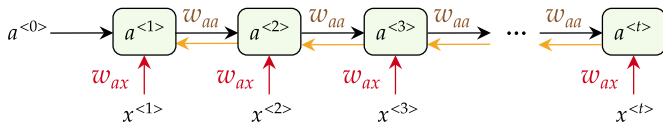


# Amostragem de novas seqüências



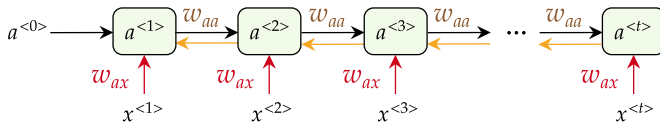
- i Consideramos  $a^{<0>} = \vec{0}$  e  $x^{<1>} = \vec{0}$ ;
- ii Seleccionamos uma previsão aleatoriamente da distribuição obtida por  $\hat{y}^{<1>}$ ;
  - Isto é, obter um início aleatório da frase cada vez que se executa uma nova seqüência de amostra.
- iii Passamos a última palavra prevista como  $a^{<1>}$  calculado;
- iv Continuamos as etapas ii e iii por um comprimento fixo ou até obter o token  $\langle \text{EOS} \rangle$ .
- Também é possível implementar **modelos de linguagem em nível de caractere** (embora sejam menos eficientes).

# Fluxo padrão de gradiente de uma RNN



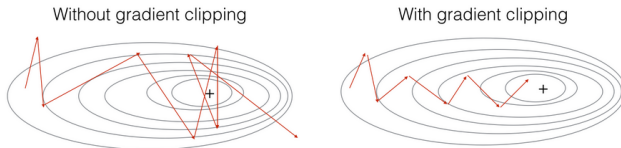
- ▶ O cálculo do gradiente em relação aos termos  $a$  envolvem muitos fatores de  $w_{aa}$  + repetidos cálculos de gradiente;
- ▶ Problemas:
  - ▶ Explosão de gradiente: Muitos valores  $> 1$ ;
  - ▶ Desaparecimento de gradiente: Muitos valores  $< 1$ ;

# RNN: explosão de gradiente



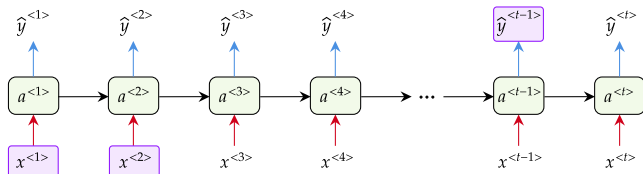
## ► Soluções:

- Retropropagação truncada: **não atualizar sempre todos os pesos**;
- Recorte de gradiente (clipping): abordagem mais empregada
  - Redimensionamento de parte do vetor de gradiente para que não seja maior que algum limite máximo pré-definido.



# RNN: desaparecimento de gradiente

Ex: "I grew up in France, ... and I speak fluent \_\_\_\_\_"

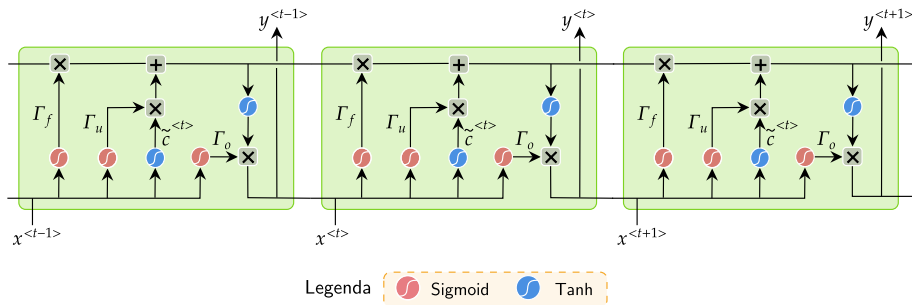


- ▶ Problema associado: dependências de longo prazo;
- ▶ Soluções:
  - ▶ Ideia 1: uso da função de ativação ReLU para evitar a redução dos gradientes para valores menores que 0;
  - ▶ Ideia 2: inicialização dos parâmetros para evitar que os pesos sejam reduzidos a 0;
  - ▶ Ideia 3: uso de uma unidade recorrente mais complexa com portas para controlar quais informações são passadas (LSTM, GRU, etc.).

# **Long Short Term Memory (LSTM) Networks**

# Long Short Term Memory (LSTM)

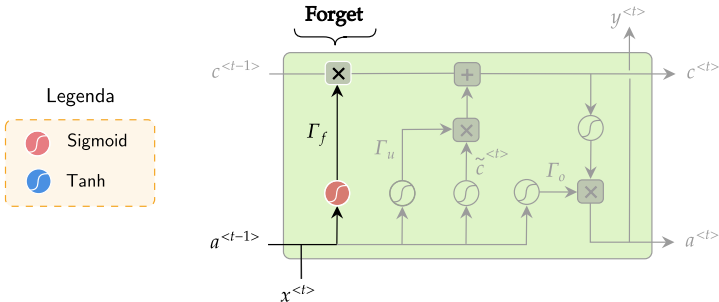
- ▶ Módulos LSTM contêm blocos computacionais que controlam os fluxos de informação;
- ▶ As células LSTM são capazes de rastrear informações ao longo de muitas etapas de tempo;
- ▶ A informação é adicionada ou removida através de estruturas chamadas portas (*gates*).



# Long Short Term Memory (LSTM): Gates

► I) **Forget** → II) **Store** → III) **Update** → IV) **Output**

**Esquece partes irrelevantes do estado anterior**



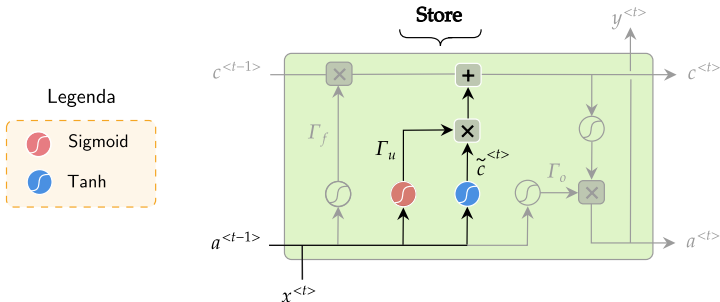
$$\Gamma_f = \sigma \left( W_f \left[ a^{<t-1>}, x^{<t>} \right] + b_f \right)$$

► Sigmoide: (0 → esquecer) ou (1 → manter)!

# Long Short Term Memory (LSTM): Gates

► I) Forget → II) Store → III) Update → IV) Output

**Armazena novas informações relevantes no estado da célula**



$$\Gamma_u = \sigma \left( W_u \left[ a^{<t-1>}, x^{<t>} \right] + b_u \right)$$
$$\tilde{c}^{<t>} = \tanh \left( W_c \left[ a^{<t-1>}, x^{<t>} \right] + b_c \right)$$

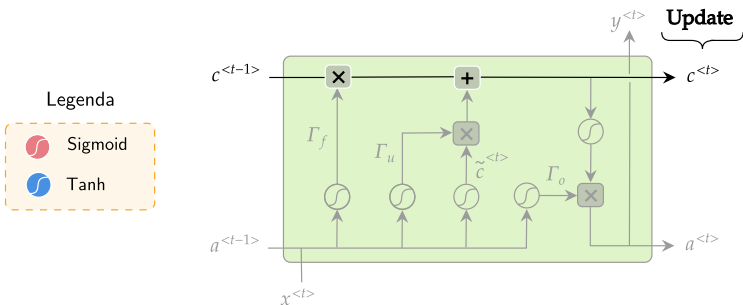
► Duas etapas: Sigmoide (o que atualizar?) e Tanh (valores candidatos)!



# Long Short Term Memory (LSTM): Gates

► I) Forget → II) Store → **III) Update** → IV) Output

**Atualiza seletivamente os valores do estado da célula**



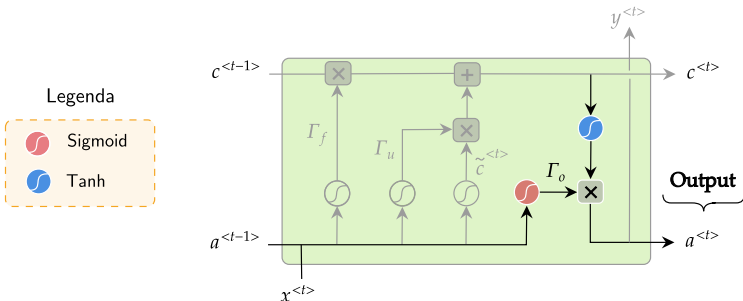
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

► Sem multiplicação matricial: **evita o desaparecimento de gradiente!**

# Long Short Term Memory (LSTM): Gates

► I) Forget → II) Store → III) Update → IV) Output

Controla quais informações são enviadas para a próxima etapa de tempo



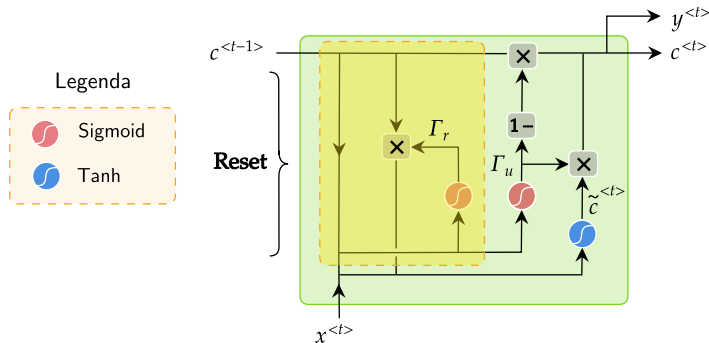
$$a^{<t>} = \Gamma_o * \tanh(c^{<t>})$$

► Versão filtrada do estado oculto!

# **Gated Recurrent Unit (GRU)**

# Gated Recurrent Unit (GRU): Gates

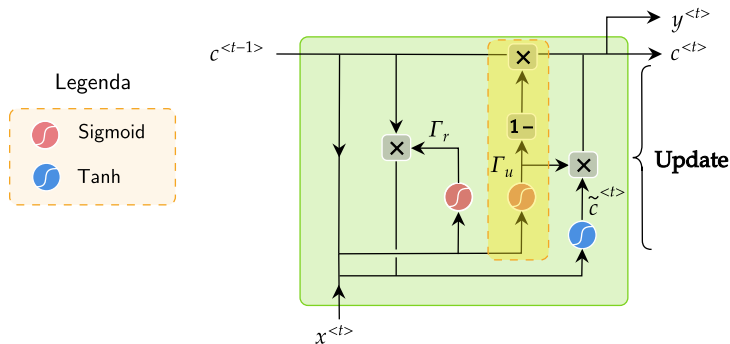
## ► I) Reset → II) Update



$$\Gamma_r = \sigma \left( W_r \left[ c^{<t-1>}, x^{<t>} \right] + b_r \right)$$

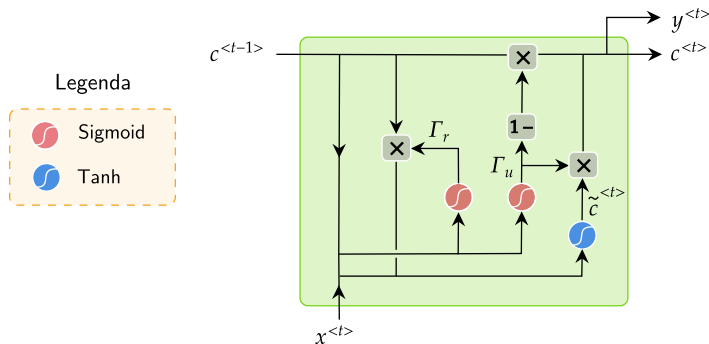
# Gated Recurrent Unit (GRU): Gates

## ► I) Reset → II) Update



$$\Gamma_u = \sigma \left( W_u \left[ c^{<t-1>}, x^{<t>} \right] + b_u \right)$$

# Gated Recurrent Unit (GRU): Gates



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

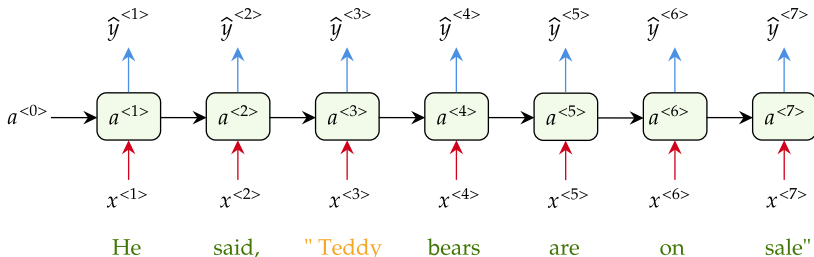
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

$$a^{<t>} = c^{<t>}$$

# **Bidirectional RNN (BRNN)**

# Bidirectional RNN (BRNN)

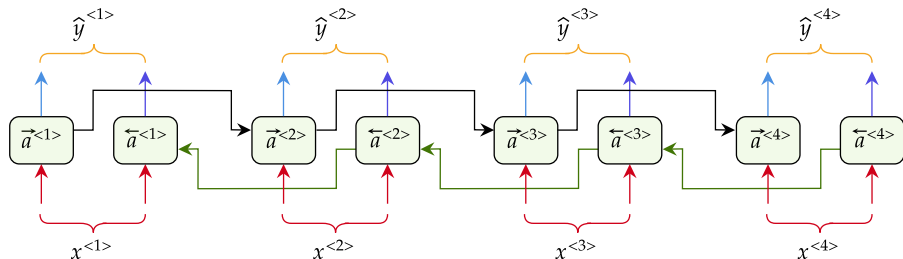
- ▶ Exemplo: problema de entidade mencionada



- ▶ O nome **Teddy** não pode ser aprendido com as palavras **He** e **said**, mas sim a partir de **bears**;
- ▶ BRNN soluciona esse problema!



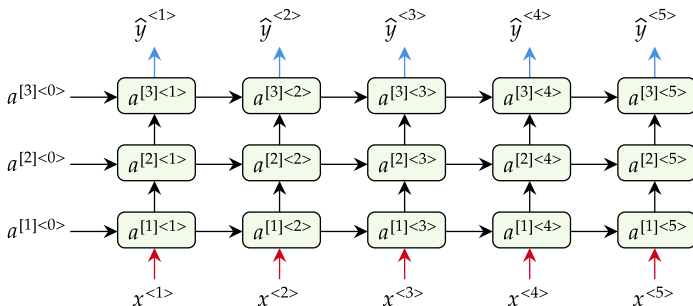
# Bidirectional RNN (BRNN)



- ▶ Gráfico acíclico;
- ▶ Para obter  $\hat{y}^{<t>}$  utilizamos as ativações que vêm de ambas direções;
- ▶ Os blocos podem ser RNNs, LSTMs ou GRUs;
- ▶ Principal desvantagem: precisa-se de **toda a sequência** antes de processá-la!

# **Deep RNNs (Deep-RNNs)**

# Deep RNNs

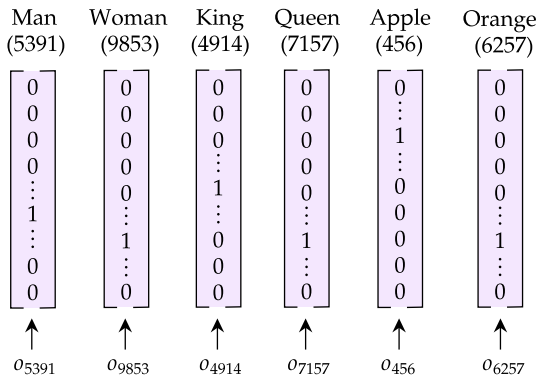


- ▶ Em muitos problemas, é útil empilhar algumas camadas RNN para formar uma rede mais profunda;
- ▶ No exemplo acima temos uma rede com 3 camadas RNN.

# **Introdução ao Word Embeddings**

# Representação de palavras

- ▶ NLP foi revolucionada pelo aprendizado profundo e especialmente pelas RNNs;
- ▶ Relembrando: exemplo de representação via one-hot.



## Representação de palavras: one-hot

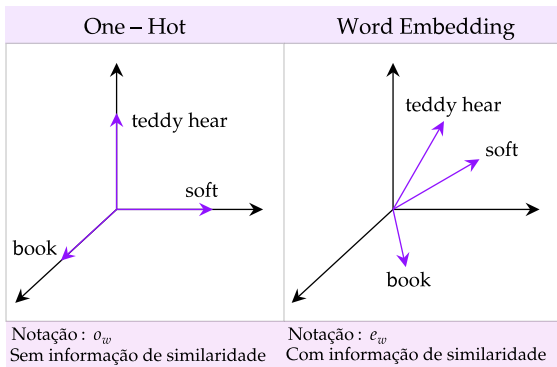
- ▶ Problema com essa abordagem: não considera a similaridade entre palavras, dificultando assim a generalização.
- ▶ Por exemplo:

I want a glass of orange ?  
I want a glass of apple ?

- ▶ Um modelo treinado e capaz de prever **juice** para a primeira frase, terá dificuldade para prever **juice** na segunda frase;
- ▶ Observe: as palavras **orange** e **apple** estão relacionadas;
- ▶ O produto interno entre qualquer vetor one-hot é nulo. Ademais as distâncias são as mesmas;
- ▶ Opção: [Word Embeddings](#).

# Representação de palavras: Word Embeddings

- Word Embeddings é um meio de representar palavras que leva em consideração a similaridade destas;



# Representação de palavras: Word Embeddings

Ex : 300

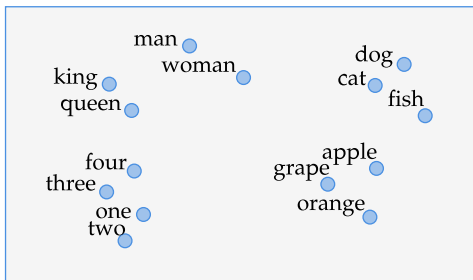
	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
⋮	⋮	⋮	⋮	⋮	⋮	⋮

- ▶ Nesse caso, cada palavra terá 300 features;
- ▶ Cada coluna de palavras é um vetor que corresponde a sua representação;
- ▶ Nesse caso,  $e_{5391}$  representa o vetor de features relacionado à palavra **man**;
- ▶ Observe que como as palavras **orange** e **apple** compartilham muitas features similares, a generalização é mais facilitada.



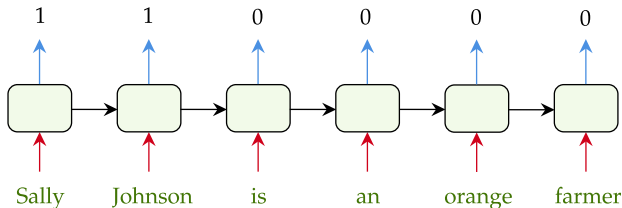
# Word Embeddings: t-SNE

- ▶  $t$ -SNE ( $t$ -distributed Stochastic Neighbor Embedding)
  - ▶ Técnica que visa reduzir embeddings de alta dimensão em um espaço dimensional inferior;
  - ▶ Na prática, é comumente usado para visualizar vetores de palavras no espaço 2D.



# Representação de palavras: Word Embeddings

Exemplo : reconhecimento de entidade nomeada



- ▶ Após treinar a rede com a sentença acima, o modelo é capaz de descobrir que para a sentença **Robert Lin is an apple farmer**, Robert Lin é um nome!;
- ▶ Isso é possível porque as palavras **orange** e **apple** têm representações próximas;
- ▶ Ao usarmos Word Embeddings, estamos aprendendo uma representação para cada palavra do vocabulário em questão.

# Word Embeddings

- ▶ Word Embeddings auxilia no raciocínio por analogia;

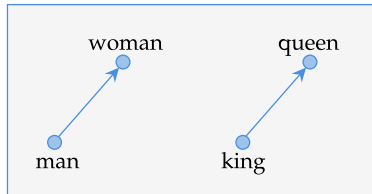
	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97

$e_{\text{Man}}$     $e_{\text{Woman}}$     $e_{\text{King}}$     $e_{\text{Queen}}$

- ▶ Se **Man** → **Woman**, então **King** → ?;
- ▶ Para prever a correspondência, consideremos as diferenças:
  - ▶  $e_{\text{Man}} - e_{\text{Woman}} = [-2 \ 0 \ 0 \ 0]$ ;
  - ▶  $e_{\text{King}} - e_{\text{Queen}} \approx [-1.9 \ 0 \ 0 \ 0]$

# Word Embeddings

- ▶ A diferença é sobre o gênero em ambos os casos;
- ▶ O vetor representa o gênero;



- ▶ Podemos reformular o problema para encontrar:

$$e_{Man} - e_{Woman} \approx e_{King} - e_{?}$$

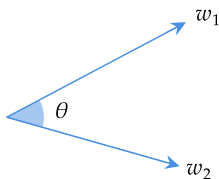
- ▶ Matematicamente, busca-se alcançar a correspondência ótima tal que:

$$\operatorname{argmax}_w \operatorname{sim}(e_w, e_{King} - e_{Man} + e_{Woman})$$

## Word Embeddings:

- Similaridade de cosseno: a função mais comumente usada;

$$\text{sim} = \frac{w_1 \cdot w_2}{\|w_1\| \|w_2\|} = \cos(\theta) \quad (3)$$



- Lembre-se:  $w_1 \cdot w_2$  representa o produto interno entre os vetores  $w_1$  e  $w_2$ .

# Embedding Matrix

- ▶ Para uma dada palavra  $w$ , a embedding matrix  $E$  é uma matriz que mapeia a representação one-hot  $o_w$  para uma representação embedding  $e_w$ , tal como:

$$e_w = E o_w$$

- ▶ Por exemplo:
  - ▶ Suponhamos que temos um vocabulário de 10000 palavras (incluindo o token);
  - ▶ Nesse caso, a ação do algoritmo criará uma matriz  $E$  de ordem (300,10000), onde o valor 300 corresponde a quantidade de features extraídas.

