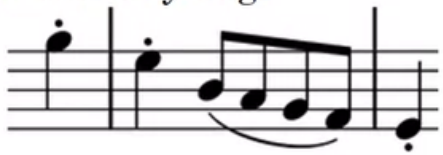
Semana 14 - Redes Neurais Recorrentes

João Florindo

Instituto de Matemática, Estatística e Computação Científica
Universidade Estadual de Campinas - Brasil
florindo@unicamp.br

Outline

Exemplos de Dados de Sequência

Speech recognition		"The quick brown fox jumped over the lazy dog."
Music generation		
Sentiment classification	"There is nothing to like in this movie."	
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	AGCCCCTGTGAGGAACTAG
Machine translation	Voulez-vous chanter avec moi?	Do you want to sing with me?
Video activity recognition		Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	Yesterday, Harry Potter met Hermione Granger.

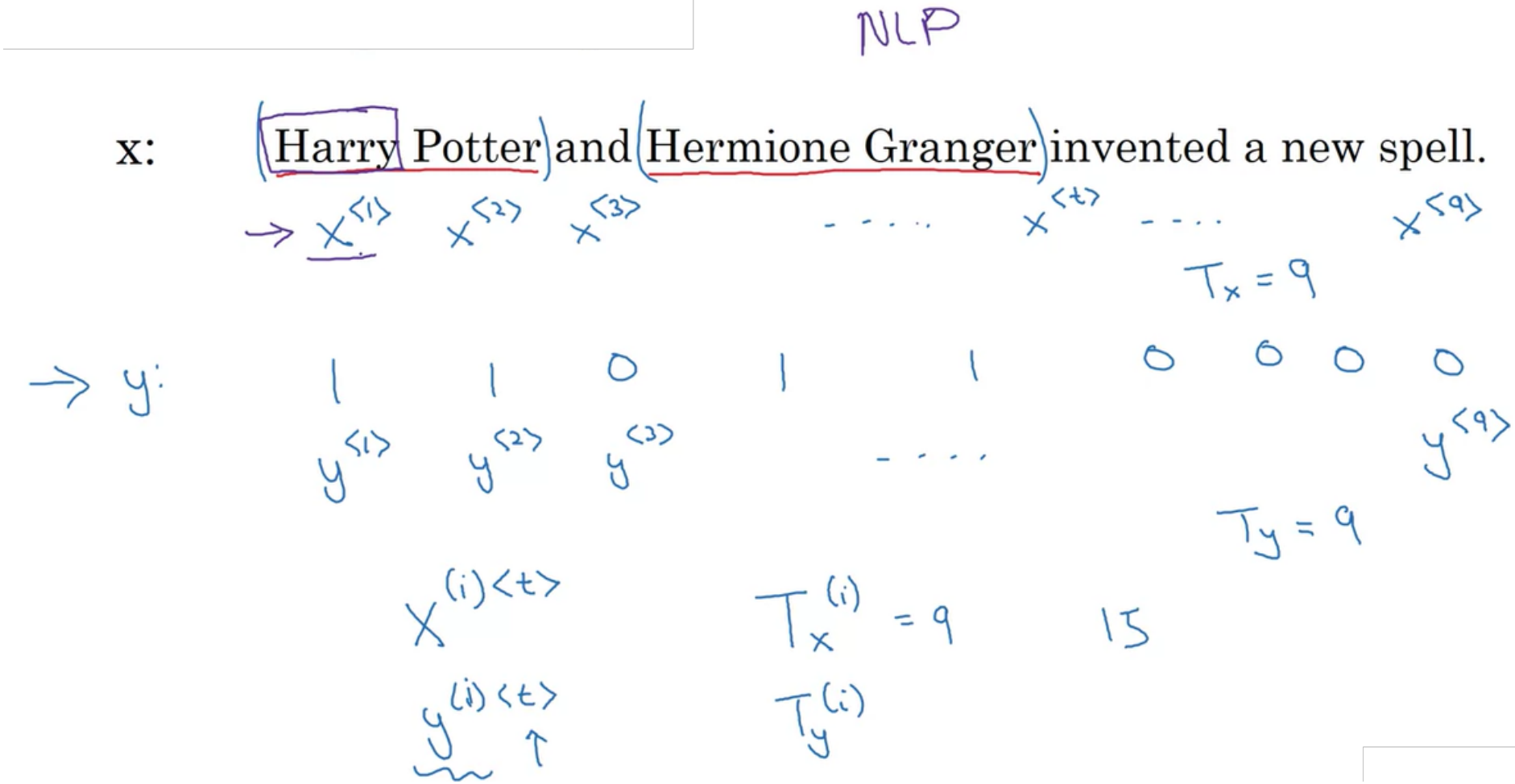
Exemplos de Dados de Sequência

- Exemplos:
  - Reconhecimento de fala
  - Geração de música
  - Classificação de sentimentos
  - Análise de sequências de DNA
  - Tradução automática
  - Reconhecimento de atividade em vídeo
  - Reconhecimento do nome de entidades (NER)

- x e y podem ser ambos sequências, com tamanhos diferentes (reconhecimento de áudio) ou iguais (DNA)

- Mas também pode ser que apenas x ou apenas y sejam sequências

Notação


NLP
x: Harry Potter and Hermione Granger invented a new spell.
$x^{\langle 1\rangle}$ $x^{\langle 2\rangle}$ $x^{\langle 3\rangle}$ .... $x^{\langle t\rangle}$ .... $x^{\langle 9\rangle}$
$T_x = 9$
y: 1 1 0 1 1 0 0 0 0
$y^{\langle 1\rangle}$ $y^{\langle 2\rangle}$ $y^{\langle 3\rangle}$ .... $y^{\langle 9\rangle}$
$T_y = 9$
$X^{(i)\langle t\rangle}$
$y^{(i)\langle t\rangle}$
$T_x^{(i)} = 9$
15
$T_y^{(i)}$

Notação

- Exemplo entrada (x) "Harry Potter and Hermione Granger invented a new spell"
- Queremos localizar os nomes na sentença: reconhecimento de entidades nomeadas
- Usado em mecanismos de busca, para filtrar, por exemplo, notícias nas últimas 24h sobre uma pessoa
- Usado também com nomes de empresas, cidades, países, moedas, etc.

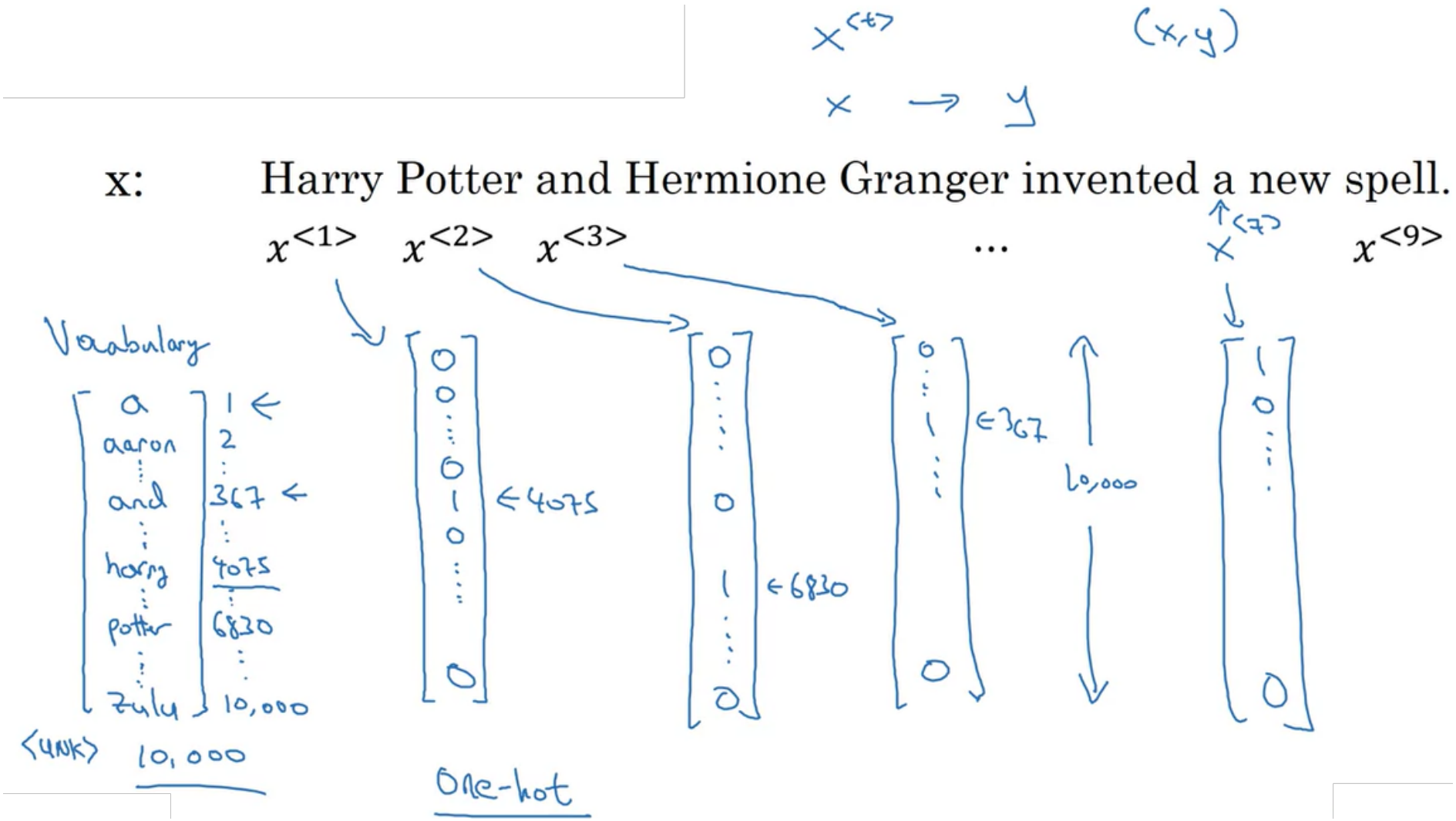
Notação

- y pode ser um vetor binário com um elemento para cada palavra de entrada, sendo 1 se a palavra é parte de um nome e 0 caso contrário
- A t-ésima palavra na entrada é $x^{<t>}$, em que t é o "instante de **tempo**" na sequência
- Notação na saída y é similar
- O comprimento da entrada é T_x e da saída é T_y. No caso $T_x = T_y = 9$, mas podiam ser diferentes

Notação

- A t-ésima entrada no i-ésimo exemplo de treinamento é $x^{(i)<t>}$
- $T_x^{(i)}$ é o comprimento do i-ésimo exemplo; podemos ter, por exemplo, $T_x^{(1)} = 9$ e $T_x^{(2)} = 15$
- Notação similar se aplica a cada y de treino correspondente

Representação de Palavras



Representação de Palavras

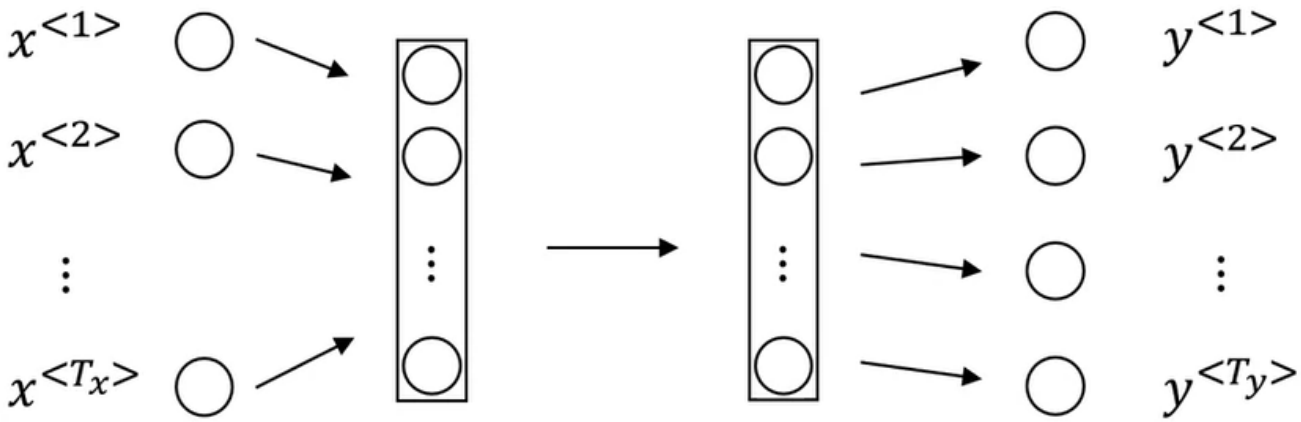
- Primeiro definimos um vocabulário (dicionário), que contém a lista de palavras usadas na sua representação
- Cada palavra no dicionário é indexada por um inteiro, na ordem, 1, 2, 3, ...
- No exemplo temos 10 mil entradas, o que é muito pequeno para aplicações modernas de NLP
- Uma forma de construir esse dicionário é olhar para as 10 mil palavras mais comuns no seu treinamento
- Ou pegar listas da internet, por exemplo, com as 10 mil palavras mais comuns em inglês

Representação de Palavras

- Pode-se usar então uma representação *one-hot*: apenas a posição da palavra no dicionário é 1 e todas as demais são 0
- Por exemplo, $x^{<1>}$ seria um vetor com 10 mil componentes, tendo 1 na posição 4075 e 0 em todas as demais
- Essa representação vai ser usada em um modelo supervisionado, com pares (x, y) no treino
- E se uma palavra na sentença de entrada não está no vocabulário?
- Cria-se uma "palavra" (*token*) falsa, que é a "palavra desconhecida", representada por $< UNK >$ (de *unknown*)

Outline

Por que não uma Rede Padrão?



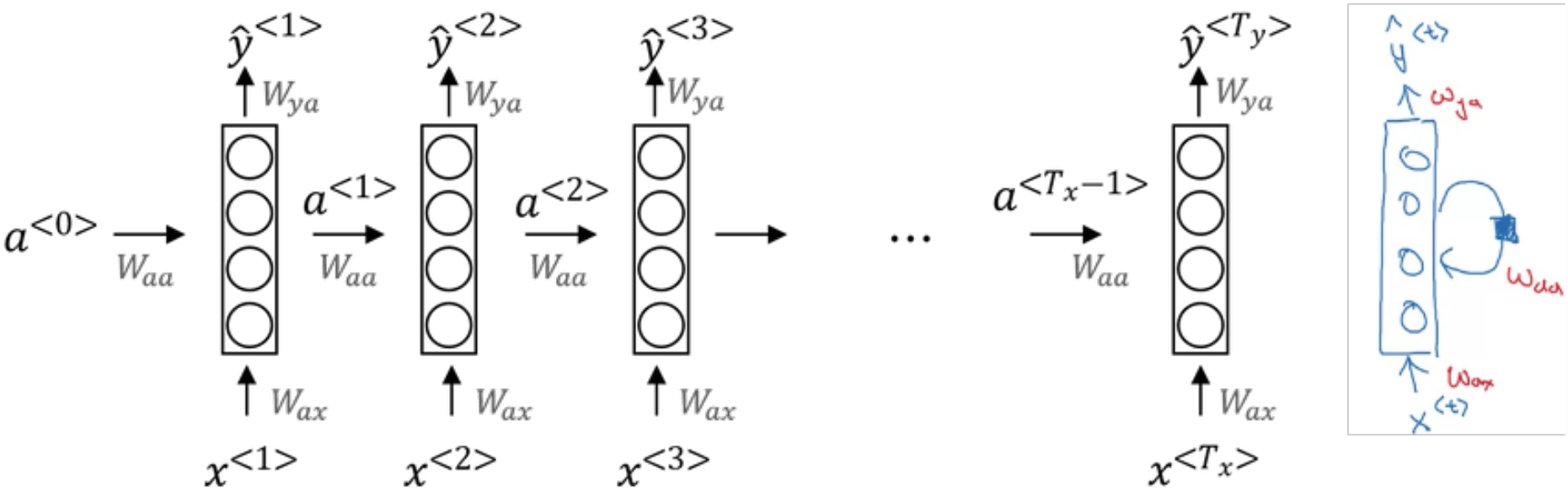
Problemas:

- Entradas e saída podem ter comprimentos diferentes
- Mais sério: uma rede clássica não compartilha *features* aprendidos em diferentes posições do texto

Por que não uma Rede Padrão?

- Suponha que a rede identifica que a palavra "Harry" em $x^{<1>}$ é um nome
- Então, se essa palavra fosse identificada em qualquer outra posição $x^{<t>}$, ela também deveria ser associada a um nome
- Isso lembra um pouco as redes convolucionais
- E assim como nas ConvNets, um modelo mais adequado a esse tipo de dado também vai reduzir o número de parâmetros
- Note que o tamanho da entrada no exemplo é 10000$\times$número de palavras: matriz de pesos gigante em uma rede clássica
- A rede que vai resolver ambos os problemas é chamada de **Rede Neural Recorrente** (RNN)

Rede Neural Recorrente



Rede Neural Recorrente

- Lendo a sentença da esquerda para a direita, a primeira palavra lida $x^{<1>}$ serve de entrada à camada escondida da rede, que devolve na saída $y^{<1>}$
- Em seguida, a rede recebe a segunda palavra $x^{<2>}$ E a ativação $a^{<1>}$ da camada anterior
- E o processo vai se repetindo, cada palavra é associada a um **passo de tempo**
- Note que aqui $T_x = T_y$, se isso não fosse verdade a arquitetura seria um pouco diferente

Rede Neural Recorrente

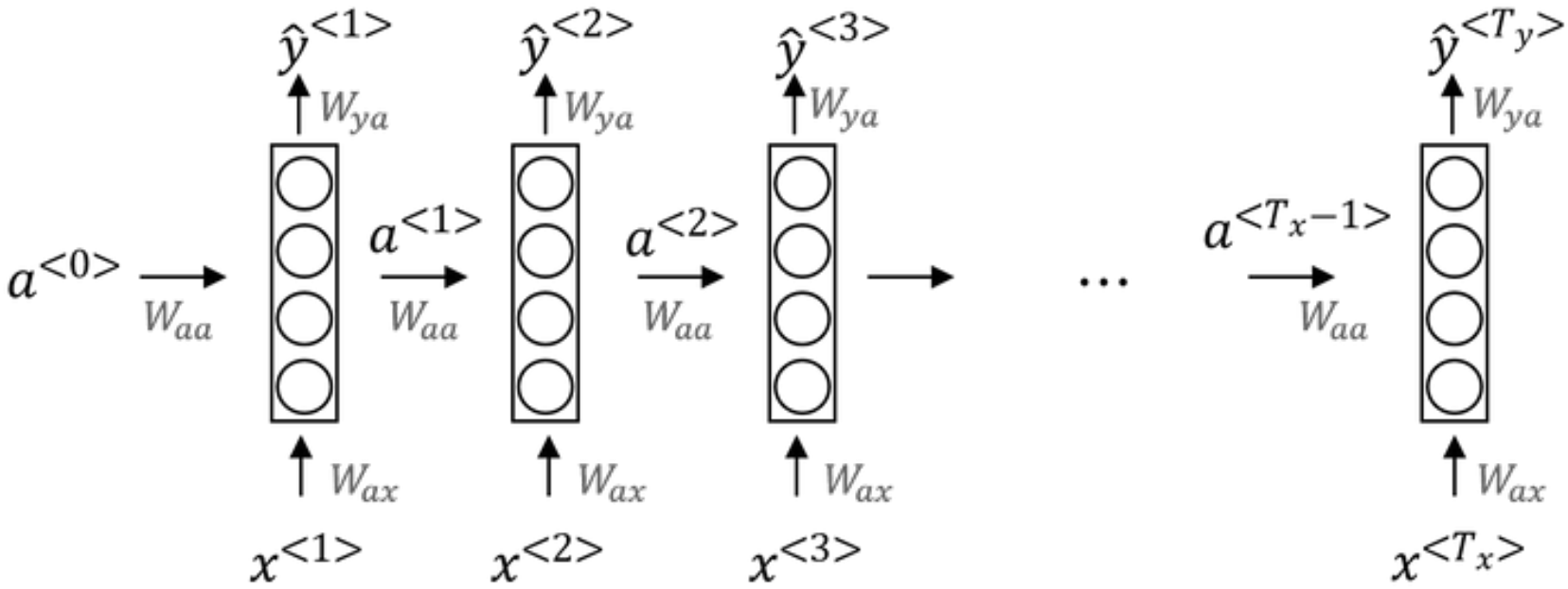
- NOTA 1: A ativação inicial $a^{<0>}$ pode ser implementada como um vetor de zeros ou de números aleatórios
- NOTA 2: Muitos artigos e livros representam redes recorrentes com uma seta em *loop*, por vezes com uma caixa sombreada na seta indicando o *delay* temporal
- ATENÇÃO: Os parâmetros W_{ax} que multiplicam a entrada na camada escondida são sempre os MESMOS em todos os passos de tempo; Eles são COMPARTILHADOS
- As ativações de um passo para o outro também são governadas por parâmetros W_{aa}, e que também são sempre os mesmos em cada passo de tempo
- O mesmo ocorre com os parâmetros W_{ya} na saída

Rede Neural Recorrente

- Deste modo, a previsão de $\hat{y}^{<3>}$ usa não apenas a informação de $x^{<3>}$, mas também de $x^{<2>}$ e $x^{<1>}$ passados pela ativação
- Um ponto fraco dessa arquitetura é que ela usa informação apenas do passado
- Imagine a sentença "Teddy Roosevelt was a great President"
- A última palavra é importante para definir que "Teddy" é um nome e não como em "Teddy bears are on sale!"
- Isso será resolvido usando uma RNN Bidirecional (BRNN)

Forward Propagation

Forward Propagation



$$a^{<0>} = \vec{0}$$

$$a^{<1>} = g_1(W_{aa}a^{<0>} + W_{ax}x^{<1>} + b_a)$$

Maybe $tanh$ or $relu$

$$\hat{y}^{<1>} = g_2(W_{ya}a^{<1>} + b_y)$$

Maybe $sigmoid$ for ner

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

$$\hat{y}^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

Forward Propagation

- Note que a função de ativação em a e $\hat{y}$ podem ser diferentes
- Na ativação é comum que se use tanh (às vezes ReLU); O desaparecimento do gradiente será resolvido de outra forma
- Na saída, pode ser uma sigmoide na classificação binária (caso do reconhecimento de entidade), softmax para multi-classes, etc.
- NOTA: A notação W_{ax} significa que os pesos vão ser multiplicados por x usados para calcular a; Similarmente, W_{ya} significa que os pesos vão ser multiplicados por a e usados para calcular y

Notação Simplificada

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

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

$$a^{<t>} = g(W_a[a^{<t-1>}, x^{<t>}] + b_a)$$

$$\hat{y}^{<t>} = g(W_y a^{<t>} + b_y)$$

$$W_a = [W_{aa} \quad W_{ax}] \qquad [a^{<t-1>}, x^{<t>}] = \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix}$$

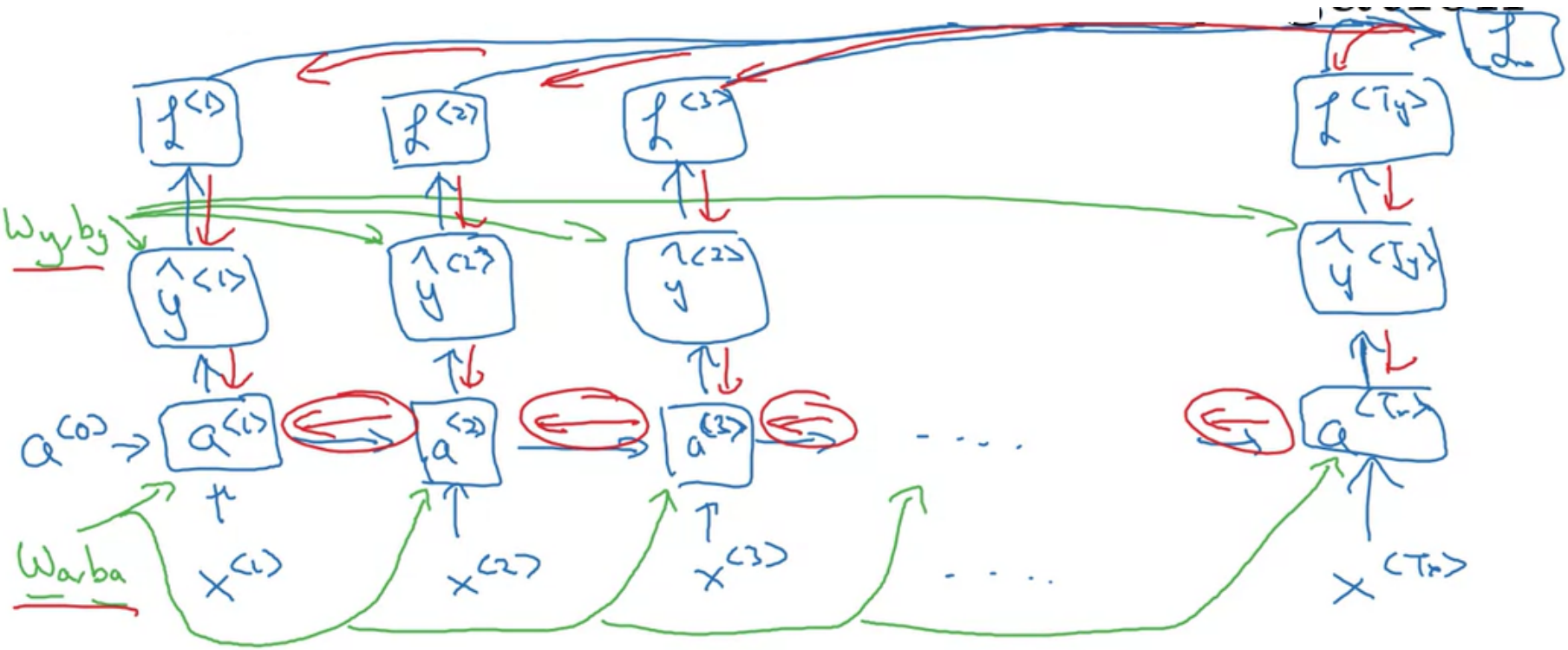
Notação Simplificada

- Aqui, as matrizes W_{aa} e W_{ax} são concatenadas horizontalmente gerando W_a
- Se temos, por exemplo, 100 unidades de ativação, a matriz W_{aa} vai ser 100×100; Se x tem dimensão 10000, então W_{ax} tem dimensão 100×10000; Neste caso, a matriz concatenada W_a vai ter dimensão 100×10100
- A notação $[a^{<t-1>}, x^{<t>}]$ representa os vetores $a^{<t-1>}$ e $x^{<t>}$ concatenados verticalmente; No exemplo, tal vetor teria dimensão 10100

Notação Simplificada

- Note que a multiplicação de W_a por $[a^{<t-1>}, x^{<t>}]$ recupera exatamente $a^{<t>}$
- A expressão para $\hat{y}^{<t>}$ também é ligeiramente alterada, chamando a matriz de pesos de W_y
- NOTA: Repare que temos um padrão em que o subscrito a em W_a se refere aos pesos usados no cálculo de a e y em W_y se refere ao cálculo de y

Backpropagation Through Time



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

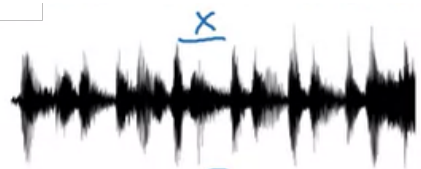
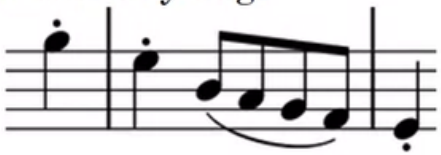
$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}^{\langle t \rangle}(\hat{y}^{\langle t \rangle}, y^{\langle t \rangle})$$

Backpropagation Through Time

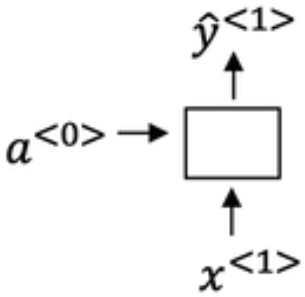
- Na implementação de RNNs em *frameworks* de *deep learning*, o *backpropagation* já vem implementado automaticamente
- Como antes, o *backpropagation* se dará na direção oposta ao *forward*
- Definimos uma *loss function*, tanto para cada palavra quanto para a sentença toda
- No caso é a nossa já conhecida *loss* da regressão logística
- Adicionamos essa *loss* ao nosso grafo computacional e o *backpropagation* se aplica como usual
- O nome "*Backpropagation through time*" vem do fato de o *forward* ser feito na sequência temporal da entrada

Outline

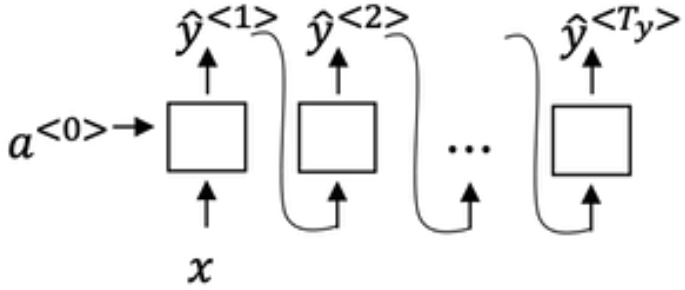
Aplicações

Speech recognition		"The quick brown fox jumped over the lazy dog."
Music generation		
Sentiment classification	"There is nothing to like in this movie."	
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	AGCCCCTGTGAGGAACTAG
Machine translation	Voulez-vous chanter avec moi?	Do you want to sing with me?
Video activity recognition		Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	Yesterday, Harry Potter met Hermione Granger.

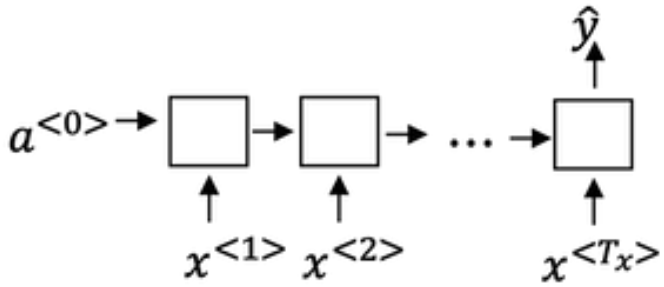
Tipos de Arquiteturas



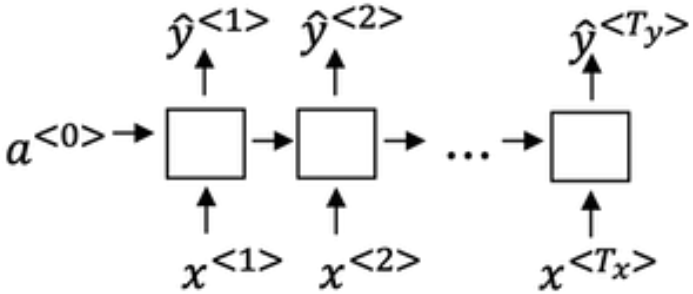
One to one



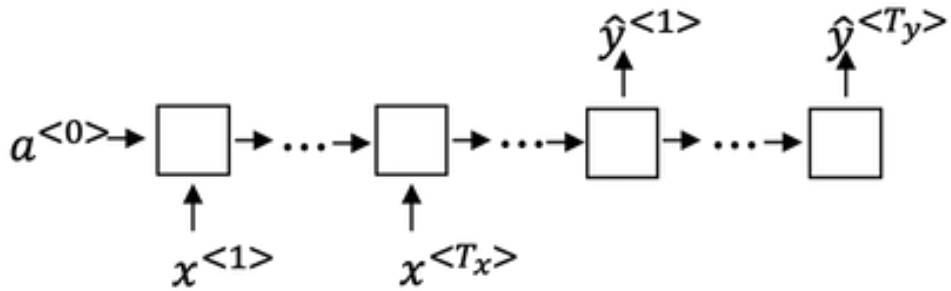
One to many



Many to one



Many to many



Many to many

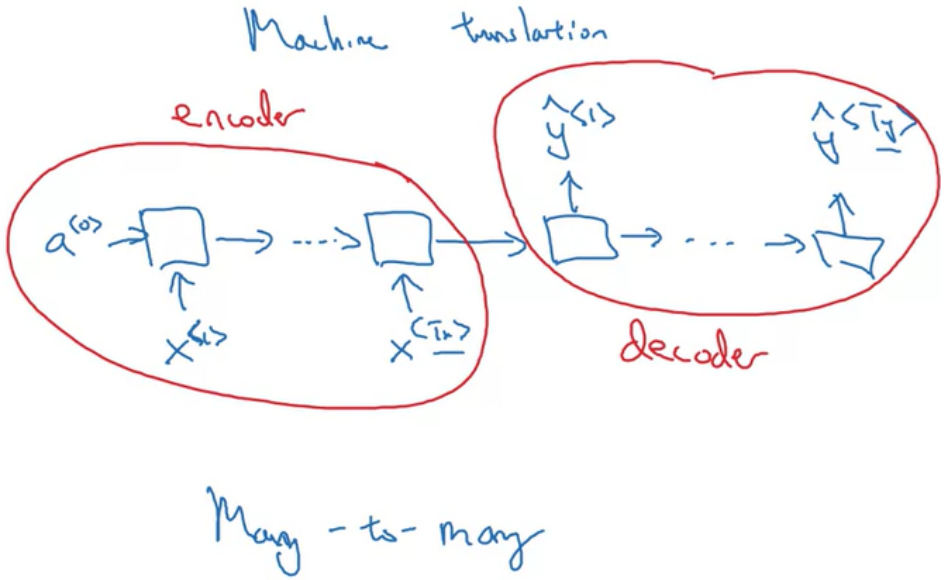
Tipos de Arquiteturas

- O exemplo visto com $T_x = T_y$ é uma arquitetura "many-to-many"
- Já na classificação de sentimento, x é um texto, como o *review* de um filme ("There is nothing to like in this movie") e y pode ser 0/1 (negativo/positivo) ou um número de 1 a 5 (estrelas)
- Neste caso, temos uma RNN com a saída apenas no último passo de tempo, após ter lido toda a sentença, chamada de "many-to-one"
- NOTA: Existe também uma arquitetura menos interessante, que é a "one-to-one", que equivale a uma rede neural clássica

Tipos de Arquiteturas

- A a geração de música usa uma arquitetura "one-to many"
- x pode ser um inteiro correspondendo ao gênero, ou à primeira nota, ou o conjunto vazio ou vetor de zeros
- No caso temos x entrando apenas no primeiro passo de tempo
- Detalhe técnico: na geração de sequências, a saída do passo anterior é usada como entrada no próximo passo
- Temos também uma versão alternativa do "many-to-many", em que os comprimentos da entrada e saída são diferentes; Tradução é uma tarefa típica em que isso ocorre

Tipos de Arquiteturas



- A rede primeiro lê a sentença original inteira (sem produzir saída) e depois começa a produzir a sentença traduzida (sem receber mais entradas)

- Pode ser dividida em duas partes, o *encoder*, que lê a sentença em francês, e o *decoder*, que produz a tradução para o inglês

Outline

Modelagem de Linguagem

- Uma das tarefas mais importantes em NLP
- Imagine um sistema de reconhecimento de fala cujas saídas são
  "The apple and pair salad"
  "The apple and pear salad"
- A sonoridade é a mesma, mas a frase que faz sentido é a segunda
- O sistema de reconhecimento usa uma modelagem de linguagem, que atribui uma probabilidade a cada sentença:
  $P(\text{"The apple and pair salad"}) = 3.2 \times 10^{-13}$
  $P(\text{"The apple and pear salad"}) = 5.7 \times 10^{-10}$
  em que a segunda sentença seria 3 ordens de grandeza mais provável

Modelagem de Linguagem

- Aqui, probabilidade é a chance de aquela sentença ser a próxima em um jornal, um site, um e-mail aleatório ou ouvir de alguém aquela sentença em particular

- Este é um componente importante também em tradução, que deve gerar a sentença traduzida mais provável

- Um modelo de linguagem recebe uma sentença com palavras $(y^{<1>}, y^{<2>}, \cdots, y^{<T_y>})$ e calcula a probabilidade daquela sequência particular de palavras:

$$P(\text{Sentença}) = ? \quad P(y^{<1>}, y^{<2>}, \cdots, y^{<T_y>})$$

- NOTA: Em modelos de linguagens é útil que a sentença seja representada como uma saída y em vez de uma entrada x

Construção de um Modelo de Linguagem com RNN

Training set: large corpus of english text.

Tokenize

Cats average 15 hours of sleep a day. <EOS>

$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ ... $y^{(8)}$ $y^{<9>}$

$x^{<t>} = y^{<t-1>}$

The Egyptian Mau is a bread of cat. <EOS>

<UNK>

10,000

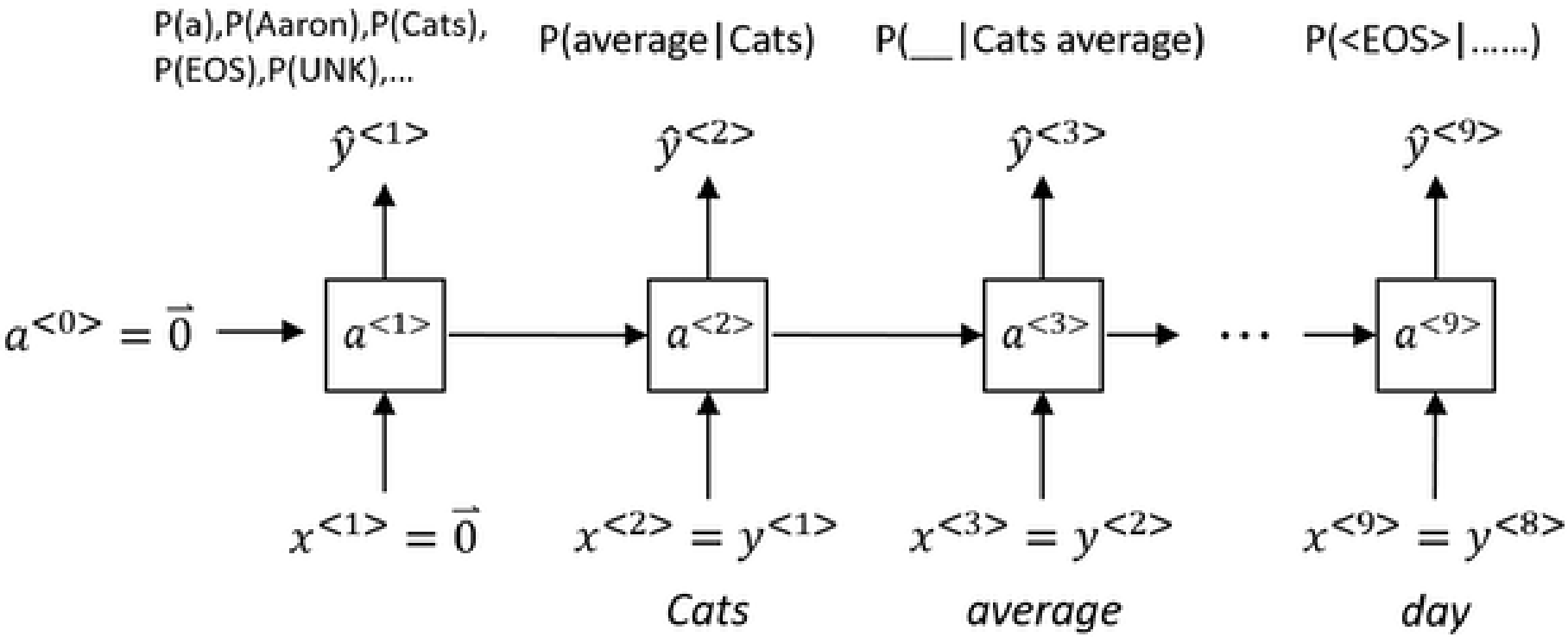
Construção de um Modelo de Linguagem com RNN

- Primeiro usamos um grande conjunto (*corpus*) de texto para o treinamento
- Suponha a frase "Cats average 15 hours of sleep a day."
- O primeiro passo é **tokenizar** a sentença, i.e., formar um vocabulário como vimos e associar cada palavra com um *one-hot vector*
- Outro procedimento usual é representar o final da sentença com um *token* extra EOS (*end of sentence*), anexado ao final de todas as sentenças no treinamento

Construção de um Modelo de Linguagem com RNN

- Se quisesse tratar cada pontuação como um *token*, precisaria adicioná-la ao vocabulário; Aqui o ponto final é ignorado
- E se uma sentença do treinamento tiver uma palavra que não está no vocabulário?
- EX.: "The Egyptian Mau is a bread of cat.$<$ *EOS* $>$" e um vocabulário com as 10 mil palavras mais comuns no inglês; "Mau" não seria reconhecida
- A palavra seria então representada pelo *token* UNK (*unknown*) e o modelo teria a probabilidade de UNK, em vez da palavra específica "Mau"

Construção de um Modelo de Linguagem com RNN



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

$$\mathcal{L} = \sum_t \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

Construção de um Modelo de Linguagem com RNN

- Uma prática que será adotada é considerar a entrada $x^{<t>}$ como sendo igual a $y^{<t-1>}$
- Neste caso $x^{<1>}$ será um vetor de zeros, como já fazíamos com $a^{<0>}$
- A saída $\hat{y}^{<1>}$ é dada por uma softmax, dando a probabilidade de a primeira palavra da sentença ser cada uma das palavras do vocabulário
- No segundo passo, a saída é a distribuição da probabilidade da segunda palavra ser uma palavra qualquer dadas as palavras anteriores
- Mas aqui a rede também recebe como entrada a primeira palavra verdadeira $y^{<1>}$ (no exemplo "Cats"); Ou seja, temos $x^{<2>} = y^{<1>}$

Construção de um Modelo de Linguagem com RNN

- No terceiro passo recebe a segunda palavra verdadeira ("average") como $x^{<3>}$ e a saída é a probabilidade de uma palavra qualquer dado que as palavras anteriores foram "cats average"
- E assim até que no final a entrada seja a última palavra ("day"); Espera-se que a palavra com maior probabilidade nesta saída seja EOS
- A *loss function* para o treinamento é a *loss* da softmax que já vimos para cada palavra e a *loss* final é a soma sobre todas as palavras
- Se essa rede for treinada sobre um grande conjunto de dados, ela vai ser capaz de, dado um conjunto de palavras, prever qual a probabilidade de cada nova palavra possível

Construção de um Modelo de Linguagem com RNN

- Já para calcular a probabilidade de uma sentença, deve-se multiplicar as probabilidades de cada palavra na ordem dadas as palavras anteriores, todas obtidas das saídas em cada passo de tempo

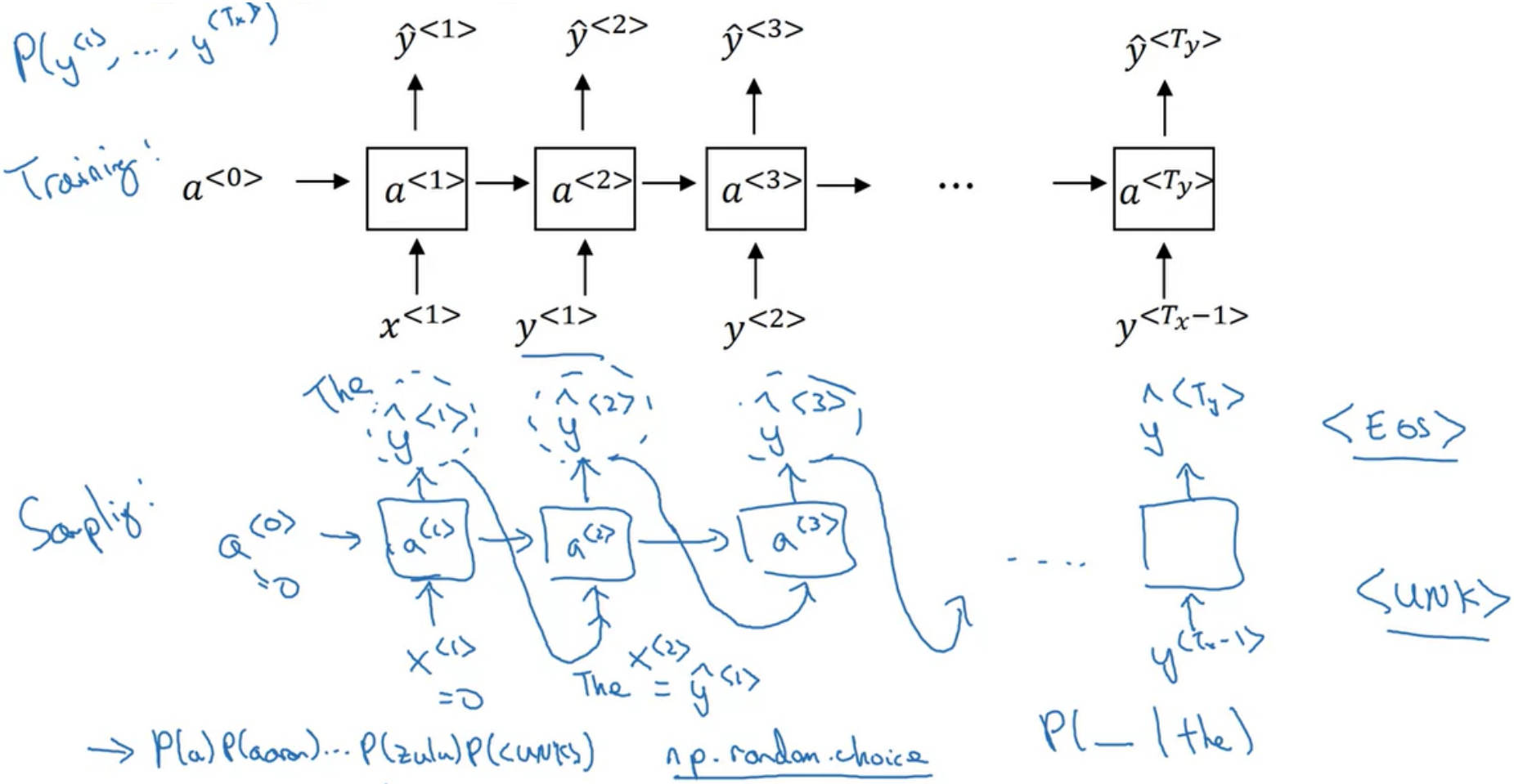
- EX.: Para uma sentença $y^{<1>}, y^{<2>}, y^{<3>}$:

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

Amostragem de Novas Sequências

- Uma das formas de se inferir informalmente o que um modelo de linguagem treinado aprendeu é amostrando novas sequências
- Vamos amostrar a distribuição de probabilidades para cada sequência particular de palavras
- A arquitetura na amostragem é um pouco diferente da usada no treino

Amostragem de Novas Sequências



Amostragem de Novas Sequências

- Novamente vamos definir $a^{<0>}$ e $x^{<1>}$ como vetores de zeros
- Então amostramos a primeira palavra a partir de $\hat{y}^{<1>}$ seguindo as probabilidades dadas pela softmax
- Pode-se usar para isso a função *np.random.choice*
- No segundo passo de tempo, a entrada vai ser a palavra amostrada no passo anterior $\hat{y}^{<1>}$
- A saída agora é usada para amostrar a segunda palavra e assim por diante

Amostragem de Novas Sequências

- Se o vocabulário inclui EOS, pode-se encerrar quando esse *token* for amostrado; Ou pré-definir um tamanho fixo de sentença
- NOTA: Este procedimento pode gerar uma palavra desconhecida (UNK); Se quiser evitar, pode rejeitar um UNK amostrado e continuar amostrando até que apareça algo diferente
- Note que até agora usamos um vocabulário no nível de palavras; Dependendo da aplicação, também podemos usar um vocabulário no nível de caracteres

Amostragem de Novas Sequências

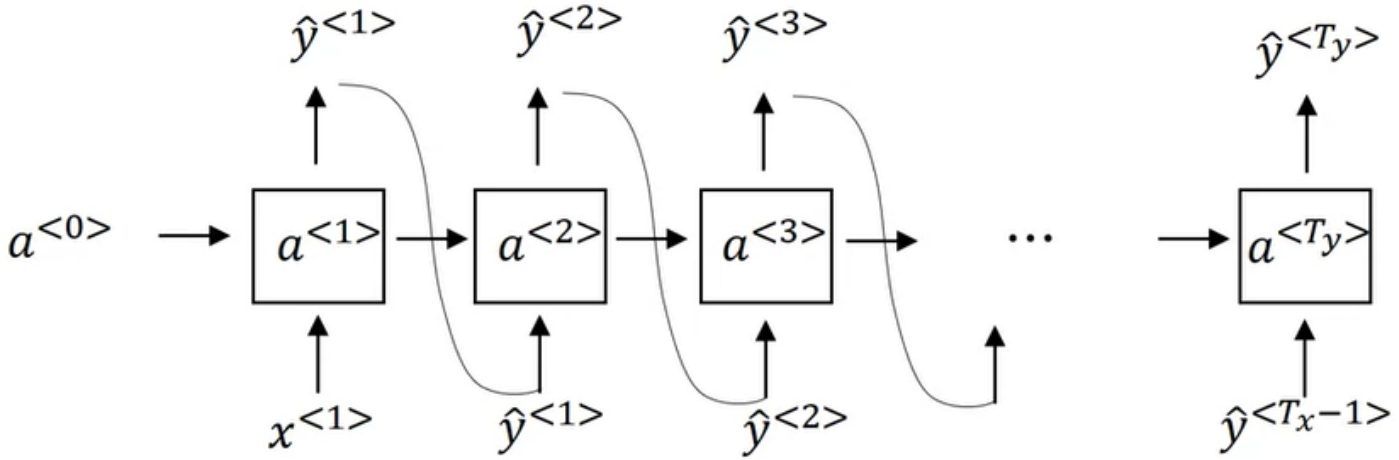
→ Vocabulary = [a, aaron, ..., zulu, <UNK>] ←

→ Vocabulary = [a, b, c, ..., z, ␣, ., ,, ;, 0, ..., 9, A, ..., Z]

$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ $y^{<4>}$

Cat average ↑↑↑↑ ...

May



Amostragem de Novas Sequências

- Poderia incluir letras maiúsculas e minúsculas, pontuações, dígitos e qualquer outro caractere presente no treinamento
- O uso de palavras ou caracteres tem prós e contras
- Uma vantagem de caracteres é não precisar se preocupar com palavras desconhecidas
- A palavra "Mau", por exemplo, mesmo não estando no vocabulário de palavras, é uma sequência válida de caracteres, com uma probabilidade associada

Amostragem de Novas Sequências

- Um problema é que entradas no nível de caracteres vão ser muito longas, comprometendo a capacidade do modelo de capturar dependências de longo alcance
- São também mais custosas computacionalmente para treinar
- Assim, exceto em situações com vocabulário mais especializado (palavras desconhecidas frequentes), não é comum que se use o nível de caracteres

Exemplos

News

President enrique peña nieto, announced sench's sulk former coming football langston paring.

"I was not at all surprised," said hich langston.

"Concussion epidemic", to be examined.

The gray football the told some and this has on the uefa icon, should money as.

Shakespeare

The mortal moon hath her eclipse in love.

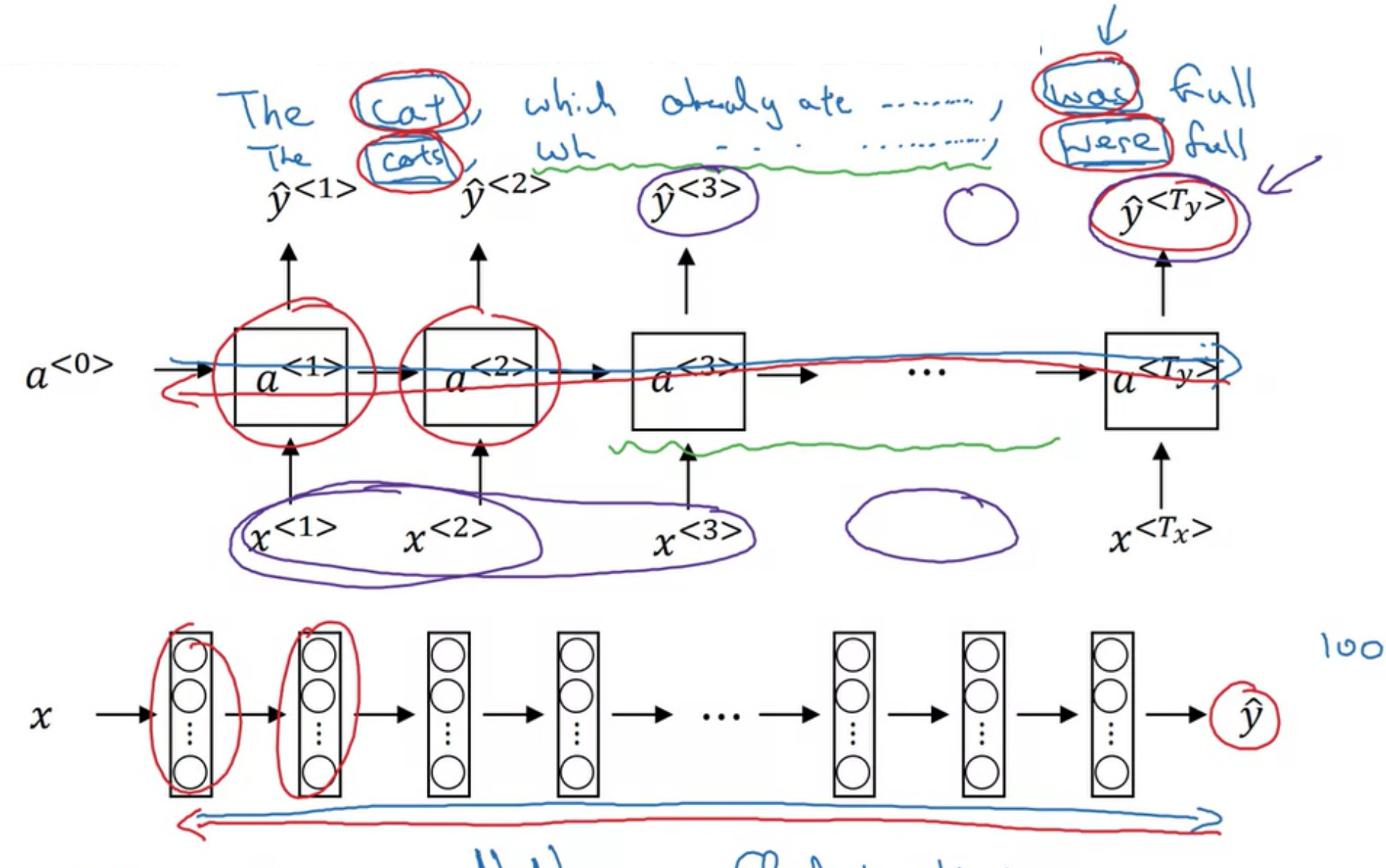
And subject of this thou art another this fold.

When besser be my love to me see sabl's.

For whose are ruse of mine eyes heaves.

Outline

Desaparecimento do Gradiente



Desaparecimento do Gradiente

- EX.: "The cat, which actually ate ..., was full." e "The cats, which actually ate ..., were full."
- Para a sentença ser consistente, o verbo was/were depende de cat/cats, que está muito antes na frase
- É uma dependência de longo alcance; RNNs não são boas em capturar isso
- Para entender porque isso ocorre, lembramos que o desaparecimento do gradiente é um problema em redes clássicas muito profundas, com 100 camadas por exemplo
- Algo similar ocorre na RNN já que o *backpropagation* precisa voltar por muitos passos de tempo propagando o erro

Desaparecimento do Gradiente

- Isso na prática dificulta a memorização de palavras distantes
- E note que em inglês essa sequência entre o sujeito e o verbo pode ser arbitrariamente grande
- Cada saída de uma RNN costuma então ser muito influenciada pelas entradas próximas no tempo
- Isso é uma limitação da RNN básica e que será tratada em breve
- Podemos ter também o problema do gradiente explodindo, que é quando ele aumenta exponencialmente

Desaparecimento do Gradiente

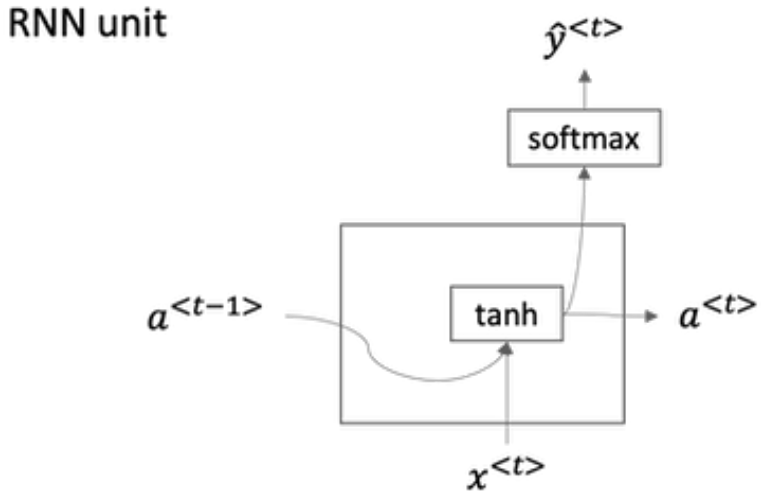
- O desaparecimento costuma ser um problema maior, mas quando há explosão isso também é catastrófico porque pode tornar os parâmetros tão grandes que bagunçam toda a rede
- Porém, a explosão do gradiente é mais fácil de detectar porque ela costuma acarretar no aparecimento de NaN nos cálculos da rede
- E uma solução para a explosão é usar o *gradient clipping*, em que o gradiente é re-escalado se for maior que um certo limiar
- O desaparecimento é mais difícil de resolver e será tema dos próximos tópicos

Outline

GRU[1]

- Adaptação da RNN mais adequada para capturar relações de longo alcance e amenizar o problema do desaparecimento do gradiente

- Lembramos da camada escondida de uma RNN:



$$a^{<t>} = g(W_a[a^{<t-1>}, x^{<t>}] + b_a)$$

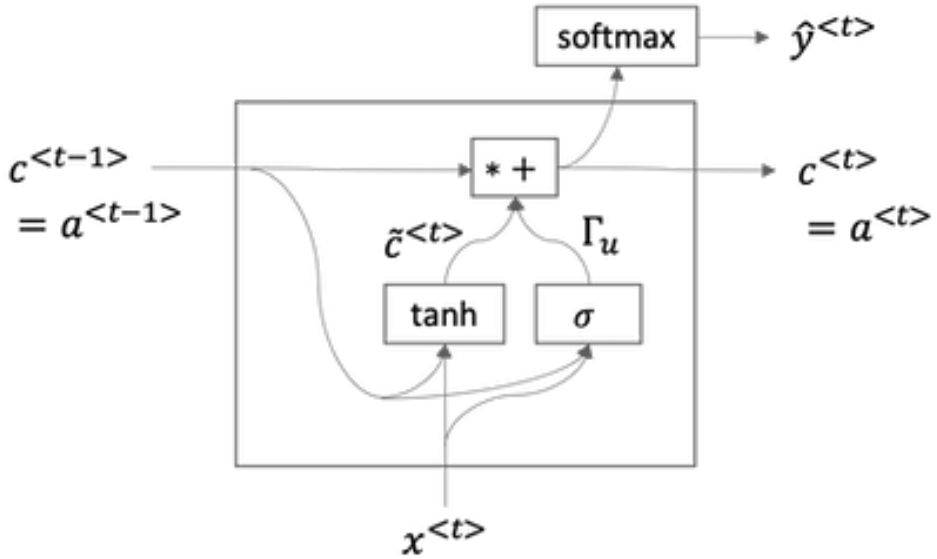
[1] Cho et al., 2014. On the properties of neural machine translation: Encoder-decoder approaches; Chung et al., 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence

GRU

- Autores: Junyoung Chung, Caglar, Gulcehre, KyungHyun Cho e Yoshua Bengio
- EX.: "The cat, which already ate ..., was full"
- A GRU vai ter uma nova variável c, que é a célula de memória, responsável por memorizar, por exemplo, se o sujeito é singular ou plural

GRU

GRU



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

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

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

Full GRU:

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

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

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

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

GRU

- No tempo t, a GRU devolve uma ativação de saída igual ao valor em c, i.e., $a^{<t>} = c^{<t>}$
- Vamos manter as letras diferentes a e c porque na LSTM que veremos mais tarde elas não são iguais
- Em cada tempo t, vamos considerar a possibilidade de sobrescrever a memória com um valor candidato $\tilde{c}^{<t>}$, que usa o valor anterior na célula de memória e a entrada atual:

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

GRU

- Mas a ideia mais importante da GRU é o conceito de porta ("*gate*"), que aqui será denotada por Γ_u (u de *update*)
- Γ_u é na realidade calculada com uma sigmoide e portante assume valores ENTRE 0 e 1:

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

- Mas, para a maior parte do intervalo possível de entradas, a sigmoide vai assumir valor muito próximos de 0 ou de 1
- Então, para desenvolver a intuição, vamos tratar Γ_u como sendo sempre 0 OU 1
- CURIOSIDADE: A letra Γ aqui é escolhida por lembrar uma porteira em uma cerca

GRU

- O papel crucial de Γ_u na GRU é decidir se a célula de memória $c^{<t>}$ vai ou não ser atualizada por $\tilde{c}^{<t>}$
- Imagine que $c^{<t>}$ representa o fato de o sujeito ser singular ($c^{<t>} = 1$) ou plural ($c^{<t>} = 0$)
- O valor de $c < t >$ pode ou não ser atualizado a cada nova palavra que a rede lê
- No exemplo, quando se lê "cat", temos $c^{<t>} = 1$; E este valor deve então permanecer até encontrar "was"
- Isso implica então em uma memorização da informação do sujeito (no caso o número) pela rede

GRU

- A equação para $c < t >$ incluindo agora Γ_u se torna:

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

- Note que $\Gamma_u = 1$ informa a rede de que a célula deve ser atualizada por $\tilde{c}^{<t>}$
- Já $\Gamma_u = 0$ implica que não há atualização pois $c^{<t>}$ manterá $c^{<t-1>}$
- No exemplo, Γ_u seria sempre 0 desde a palavra "cat" até o verbo "was", permitindo assim a memorização
- No diagrama, a caixa com $*+$ representa a equação que atualiza $c^{<t>}$
- Como na RNN básica, a saída também pode passar por uma softmax para fazer alguma previsão

GRU

- O uso da sigmoide logística em Γ_u também é estratégico já que isso vai fazer com que facilmente ele assuma valores muito próximos de zero
- Isso permite a memorização mesmo de palavras bem distantes entre si e ameniza significativamente o problema do desaparecimento do gradiente
- Note que aqui $c^{<t>}$, $\tilde{c}^{<t>}$ e Γ_u vão ser vetores com a mesma dimensão
- Então o asterisco na equação de Γ_u representa uma multiplicação componente-a-componente

GRU

- Só para a intuição, podemos pensar como se os valores de Γ_u fossem sempre 0 ou 1 e então é como se tivéssemos um vetor de bits
- Cada bit pode ser responsável por uma informação diferente a ser memorizada
- Por exemplo, podemos ter um bit para memorizar que o sujeito é singular e outro para memorizar que estamos falando de comida (quando encontra a palavra "ate" e que poderia estar relacionado com a palavra "full" no final)
- Essa versão descrita até aqui ainda é uma forma simplificada da unidade GRU; Vamos falar agora da unidade completa

GRU

- Uma primeira alteração é adicionar mais uma porta Γ_r (*relevance gate*), que vai dizer o quão relevante é $c^{<t-1>}$ para o cálculo da atualização candidata $\tilde{c}^{<t>}$
- O cálculo de Γ_r é similar ao de Γ_u, exceto por usar novos parâmetros W_r e b_r
- Muitas outras versões também foram tentadas por autores ao longo tempo, mas a forma que vimos acabou prevalecendo na prática
- Outra versão muito popular da GRU que veremos a seguir é a LSTM
- NOTA: Usamos aqui uma notação mais consistente entre a GRU e a LSTM, mas existem outras notações na literatura, como usar $\tilde{h}$, u, r e h

Outline

LSTM[2]

- LSTMs são ainda mais gerais e poderosas que GRUs na captura de relações de longo alcance
- O artigo original teve um impacto profundo no *deep learning*, embora seja bem difícil de ler, com muita teoria sobre o desaparecimento do gradiente
- Vamos comparar as equações da LSTM e GRU

[2] Hochreiter & Schmidhuber 1997. Long short-term memory

LSTM

LSTM units

GRU

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

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

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

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

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

LSTM

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

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

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

(output) $$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$

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

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

LSTM

- Não temos mais $a^{<t>} = c^{<t>}$
- Na versão mais comum também não temos a porta Γ_r (ainda que algumas versões usem)
- Outra novidade é a nova porta Γ_f (*forget gate*)
- Ela vai aparecer na equação que calcula $c^{<t>}$, no lugar onde aparecia $(1 - \Gamma_u)$
- Sua estrutura é similar a Γ_u, embora tenha outros parâmetros
- Temos ainda a porta Γ_o (*output gate*), também com estrutura similar

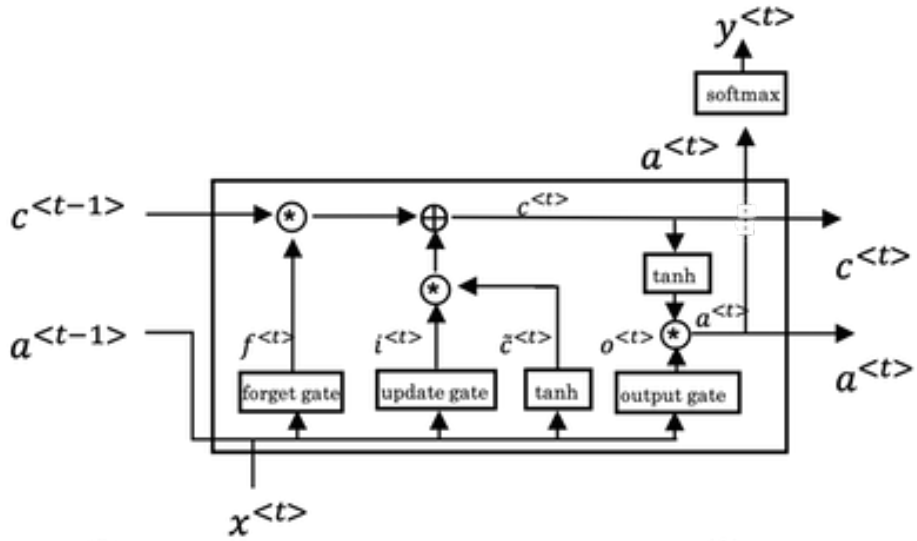
LSTM

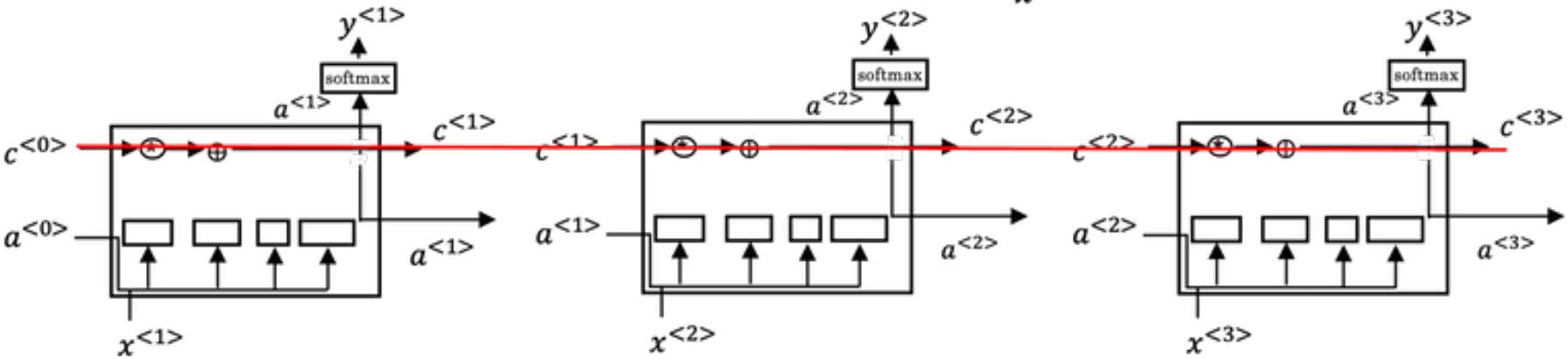
- A atualização de $c^{<t>}$ vai ter então a diferença de usar portas separadas para multiplicar (componente-a-componente) $\tilde{c}^{<t>}$ e $c^{<t-1>}$
- Por fim, em vez de $a^{<t>} = c^{<t>}$, temos $a^{<t>} = \Gamma_o * \tanh(c^{<t>})$
- Novamente, é comum que se use uma representação gráfica, embora as equações pareçam mais simples de intuir do que a figura

LSTM

LSTM

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$
$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$
$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$
$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$
$$a^{<t>} = \Gamma_o * \tanh(c^{<t>})$$





LSTM

- O ponto principal na figura é que $a^{<t-1>}$ e $x^{<t>}$ vão ser usados pelas 3 portas, além de passarem pela tangente hiperbólica no cálculo de $c^{<t>}$
- Esses valores são então combinados com $c^{<t-1>}$ para finalmente gerar $c^{<t>}$
- Outro ponto é que essas unidades LSTM vão ser conectadas entre si para processar cada entrada no tempo, de modo que tanto a quanto c são passados de um passo de tempo para o seguinte
- E aqui observamos uma linha vermelha no fluxo de c, e usando valores adequados para as portas de *forget* e *update*, é fácil para a rede memorizar uma informação por um longo intervalo de tempo

LSTM

- Uma variação comum da LSTM é a que introduz a *piphole connection*, em que as portas não vão depender apenas $a^{<t-1>}$ e $x^{<t>}$, mas também de $c^{<t-1>}$

- Um detalhe técnico sobre a *piphole connection* é que nela o primeiro elemento de $c^{<t-1>}$ afeta apenas o primeiro elemento da porta, o segundo afeta apenas o segundo, etc.

LSTM

- Sobre qual usar, LSTM ou GRU, isso vai depender do problema
- Mas deve-se observar que a GRU é mais simples, têm custo computacional menor e maior escalabilidade para modelos maiores
- Mas a LSTM é mais poderosa e costuma ser mais popular em termos de uso na prática
- CURIOSIDADE: Embora apresentamos aqui a GRU primeiro, historicamente a LSTM vem bem antes e a GRU veio como uma simplificação

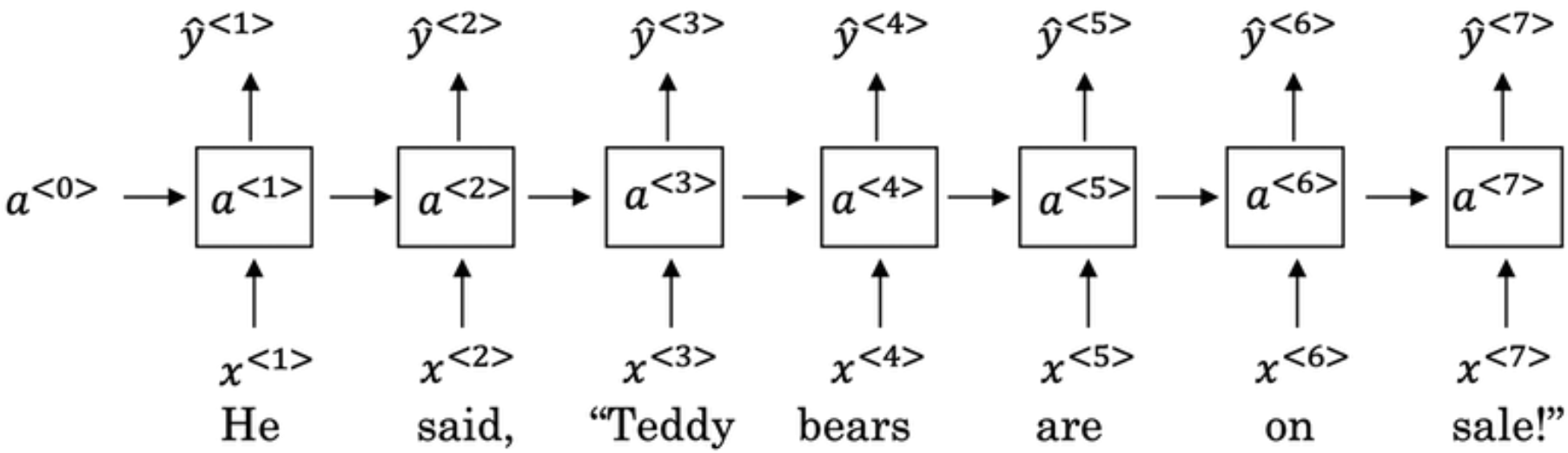
Outline

RNN Bidirecional

RNN on NER

He said, “Teddy bears are on sale!”

He said, “Teddy Roosevelt was a great President!”



RNN Bidirecional

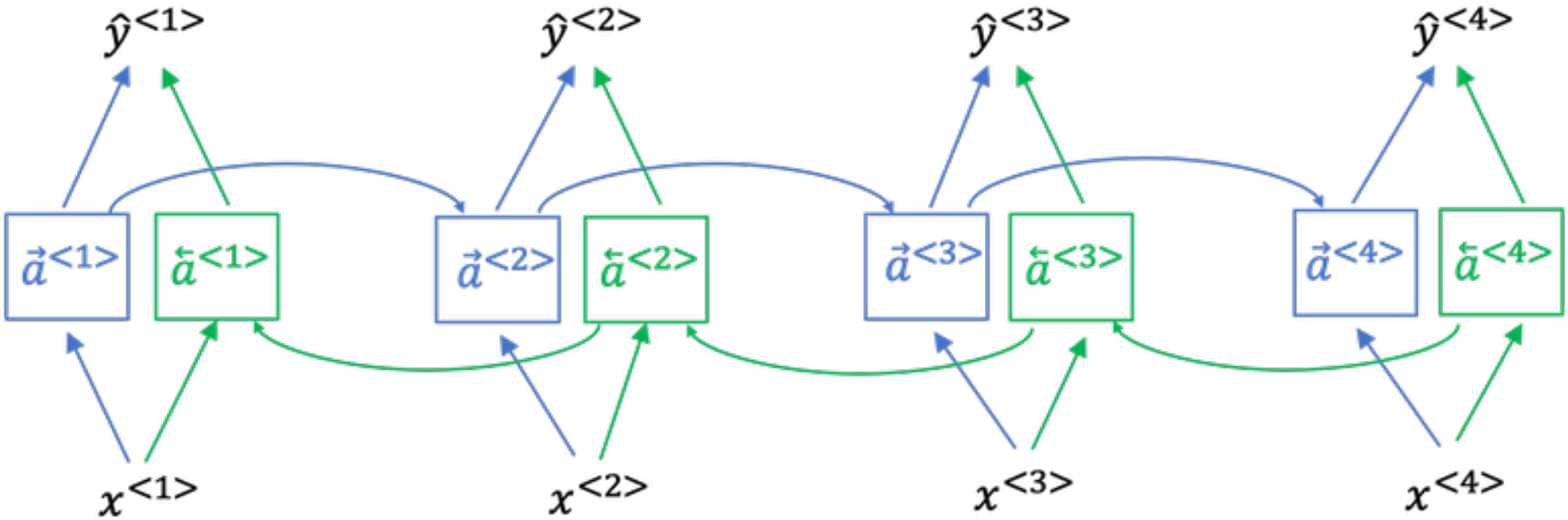
- Existem ideias que tornam as RNNs ferramentas ainda mais poderosas
- Uma delas é a RNN bidirecional, que permite o uso tanto de informações passadas quanto futuras na sequência
- Voltamos ao exemplo do reconhecimento de entidades nomeadas; Cada bloco por ser tanto uma RNN clássica quanto uma GRU ou LSTM
- E voltamos às nossas sentenças de exemplo:
  He said: "Teddy bears are on sale!"
  He said: "Teddy Roosevelt was a great President"

RNN Bidirecional

- Para saber se Teddy é ou não um nome, não basta olharmos para as 3 primeiras palavras
- Isso vai ser sempre um problema na rede unidirecional, independente do tipo de unidade
- A rede bidirecional resolve isso

RNN Bidirecional

BRNN



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

RNN Bidirecional

- A rede vai ter um componente de *forward* (púrpura), com uma seta para a direita sobre cada ativação para indicar essa direção
- As ativações são conectadas sequencialmente e geram também saídas, igual antes!
- Porém agora existe também uma camada recorrente *backward* (verde), dessa vez com a seta para a esquerda
- Temos aí um grafo acíclico; Primeiro se calcula toda a parte *forward*, depois todo o *backward* e por fim são geradas as saídas
- Não confundir com o *backpropagation* Tudo isso aqui é o *forwardpropagation*

RNN Bidirecional

- E cada saída agora depende não apenas da ativação *forward*, mas também da backward
- Isso permite que, por exemplo, a saída $y^{<3>}$ use informação não apenas do passado como antes ($x^{<1>}$, $x^{<2>}$ e $x^{<3>}$) mas também do futuro ($x^{<4>}$)
- É bastante comum que a rede bidirecional seja implementada com blocos LSTM (mas poderia usar GRU ou RNN clássica também)

RNN Bidirecional

- Uma desvantagem da BRNN é que ela requer a sequência inteira antes de fazer uma previsão em qualquer parte
- Isso é um problema, por exemplo, no reconhecimento de fala em tempo real, já que precisaria esperar a pessoa terminar de falar
- Para isso existem modelos mais complexos
- No caso de texto em NLP isso já não costuma ser um grande problema

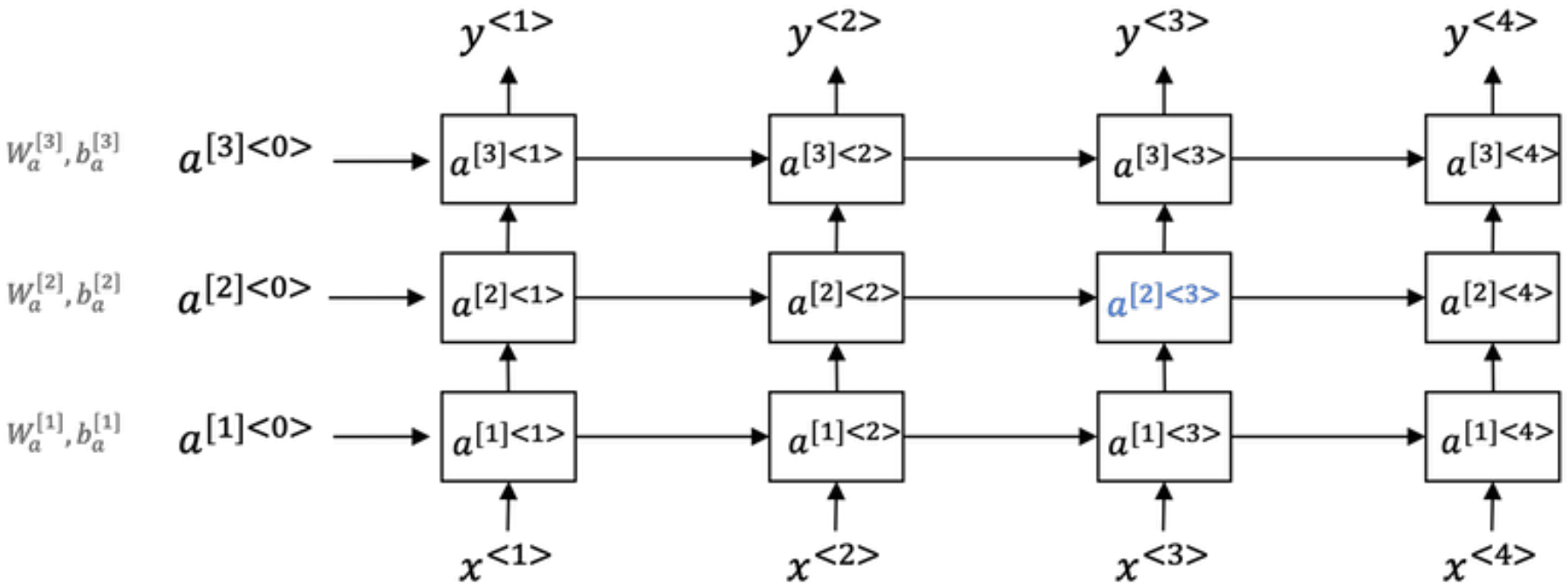
Outline

RNN Profunda

- Os blocos de RNN/GRU/LSTM podem ser empilhados em um modelo mais profundo, que assim permite o aprendizado de relações mais complexas
- Como vimos, na rede clássica, nós temos várias camadas, em que a ativação de uma serve de entrada para a próxima e ao final temos a predição; A ideia para RNNs é parecida

RNN Profunda

Deep RNN example



RNN
GRU
LSTM BRNN

$$a^{[2]<3>} = g(W_a^{[2]}[a^{[2]<2>}, a^{[1]<3>}] + b_a^{[2]})$$

RNN Profunda

- Note que adicionamos um [1] na ativação para denotar a 1a camada
- Então $a^{[l]<t>}$ é a ativação na camada l em tempo t
- Então empilhamos uma rede que vimos sobre a outra e temos na figura 3 camadas escondidas
- Vamos tomar como exemplo o cálculo de $a^{[2]<3>}$
- Ele tem duas entradas (de baixo e da esquerda) e no mais a fórmula é similar à que já conhecemos
- Atente-se apenas de que os parâmetros na mesma camada continuam compartilhados, porém, de uma camada para outra temos parâmetros diferentes!

RNN Profunda

- NOTA: Enquanto na rede padrão é comum que tenhamos até mais de 100 camadas, em uma RNN 3 camadas já é um número alto, já que temos a dimensão temporal, o que mesmo com poucas camadas já gera uma rede grande
- O que se vê com certa frequência é o uso de uma rede profunda após a saída da parte recorrente, mas com blocos que não possuem a conexão horizontal
- Por fim, essas arquiteturas podem novamente ser construídas com blocos de RNN, GRU ou LSTM e podem também ser bidirecionais