Módulo 4: Redes Neurais Recorrentes
Aula 4: Modelos de Atenção

Fabricio Murai

murai at dcc.ufmg.br

Aula Anterior

- Matemática dos Vanishing e Exploding Gradientes
- Problema da perspectiva de Funções Iteradas (opcional)
- Mantendo o gradiente estável
- Arquiteturas alternativas
  - Usando ideias de Deep Residual Networks (opcional)
  - GRU e LSTM
- LSTM: visualizações e fluxo do gradiente (opcional)

Aula de Hoje

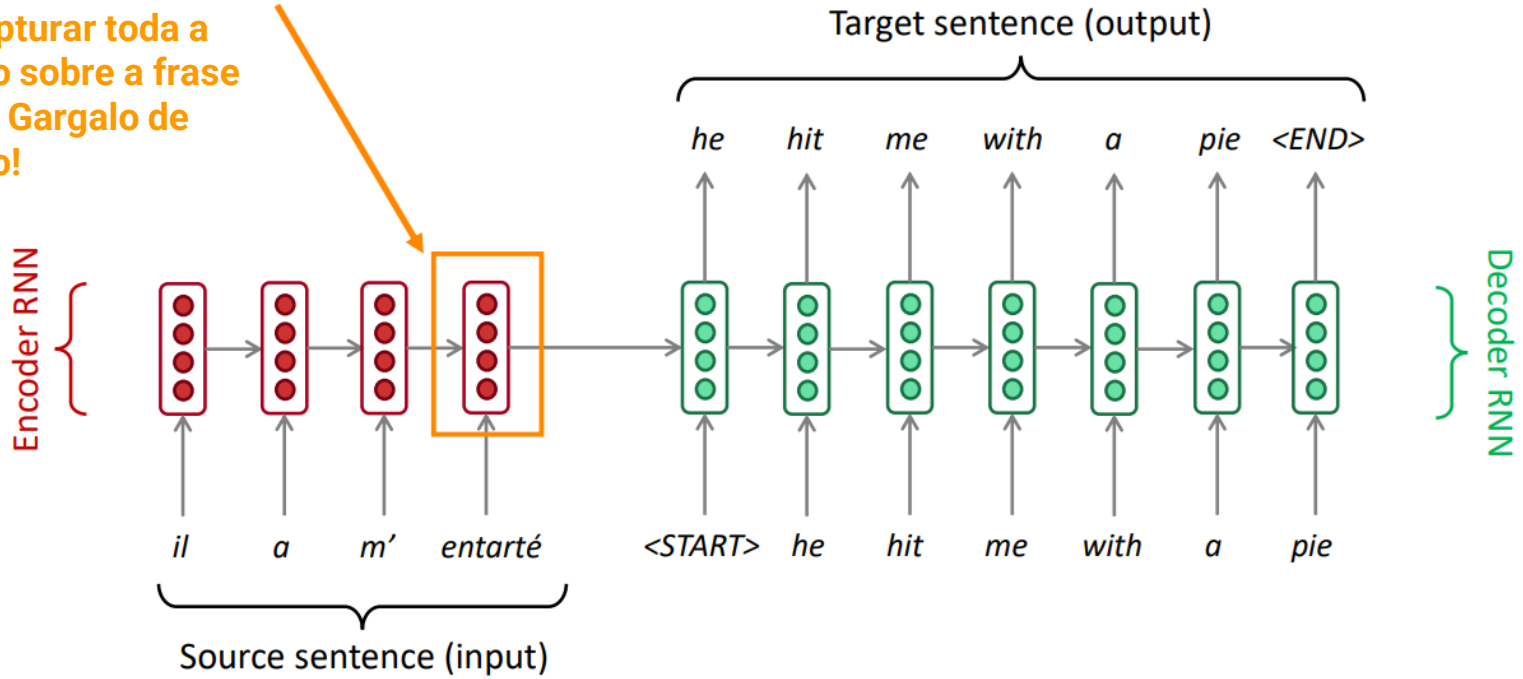
- Problema das sequências longas
- Mecanismo de Atenção
  - Exemplo (sem equações)
  - Exemplo (com equações)
- Tradução por máquina baseada em atenção
- Geração de legendas (opcional)
- Attention Transformers

Problema das sequências longas

Problema das sequências longas

Encoding of the source sentence.

Precisa capturar toda a informação sobre a frase de origem. Gargalo de informação!



Problems with this architecture?

Problema das sequências longas

Encoder deve ler a sentença e armazenar uma representação antes de passar ao decoder. **Decoder** então deve gerar a tradução.

Considere a entrada:

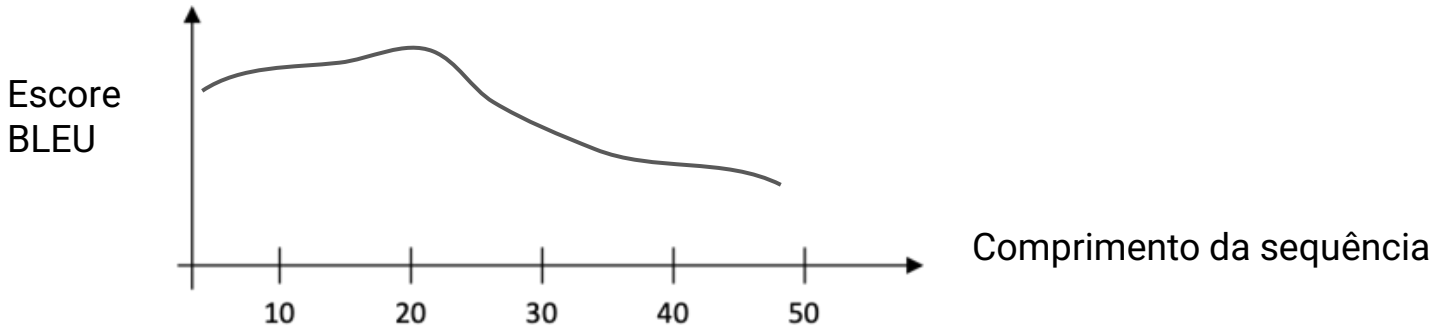
Jane s'est rendue en Afrique en septembre dernier, a apprécié la culture et a rencontré beaucoup de gens merveilleux; elle est revenue en parlant comment son voyage était merveilleux, et elle me tente d'y aller aussi.

Cuja tradução é

Jane went to Africa last September, and enjoyed the culture and met many wonderful people; she came back raving about how wonderful her trip was, and is tempting me to go too.

Problema das sequências longas

- Humanos não lêem a frase inteira, memorizam e depois começam a fazer a tradução.
- A arquitetura encoder-decoder funciona bem para sequências curtas (não muito curtas, que são um pouco difíceis de traduzir). Depois disso, o escore BLEU começa a decair.



- **Ideia dos attention models:** a cada passo do decoder, use conexões diretas com o encoder para focar em uma parte específica da sequência de entrada. Reduz o problema da queda no escore BLEU.
- Originalmente desenvolvido para tradução por máquina, mas se espalhou por outras aplicações também.

Visão Geral

- Vimos alguns modelos de previsão de sequência baseados em RNNs
- É desafiador gerar sequências longas quando os decoders tem apenas acesso ao(s) último hidden state do encoder.
  - Tradução por máquina: é difícil sumarizar sentenças longas em um único vetor, então vamos deixar o decoder espiar o input.
  - Visão: faça a rede olhar para uma parte da imagem de cada vez, de modo que possamos entender que informação está usando.
- Nesta aula discutiremos mecanismos de atenção, que melhoram drasticamente o desempenho em sequências longas.

Modelos de Atenção

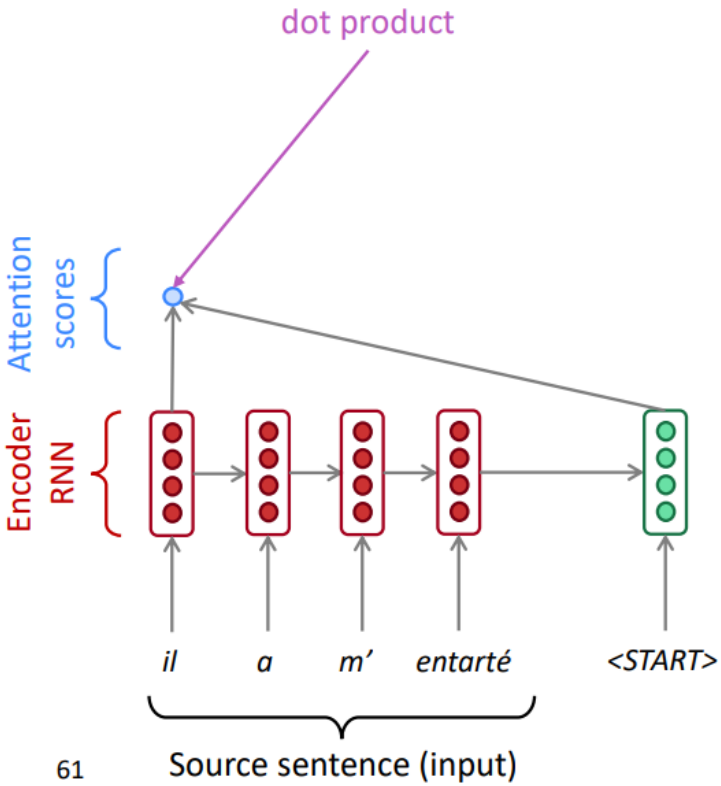
- Ideia básica: cada palavra de saída vem de uma ou de algumas poucas palavras da entrada. Talvez seja possível aprender a prestar atenção apenas às palavras relevantes conforme geramos a saída.

Bahdanau et al., Neural machine translation by jointly learning to align and translate. ICLR, 2015.

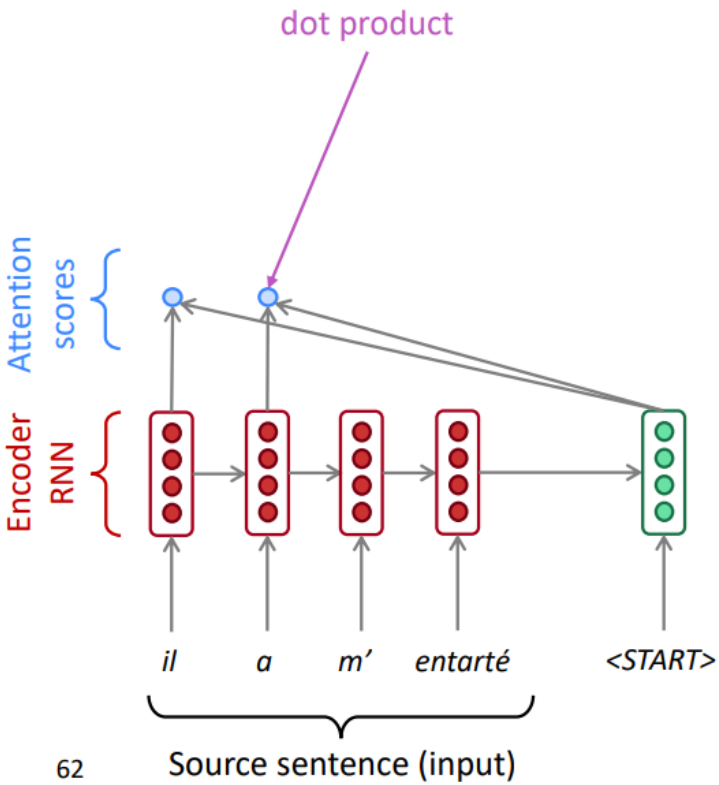
Mecanismo de Atenção

Exemplo passo-a-passo (sem equações)

Sequence-to-sequence with attention

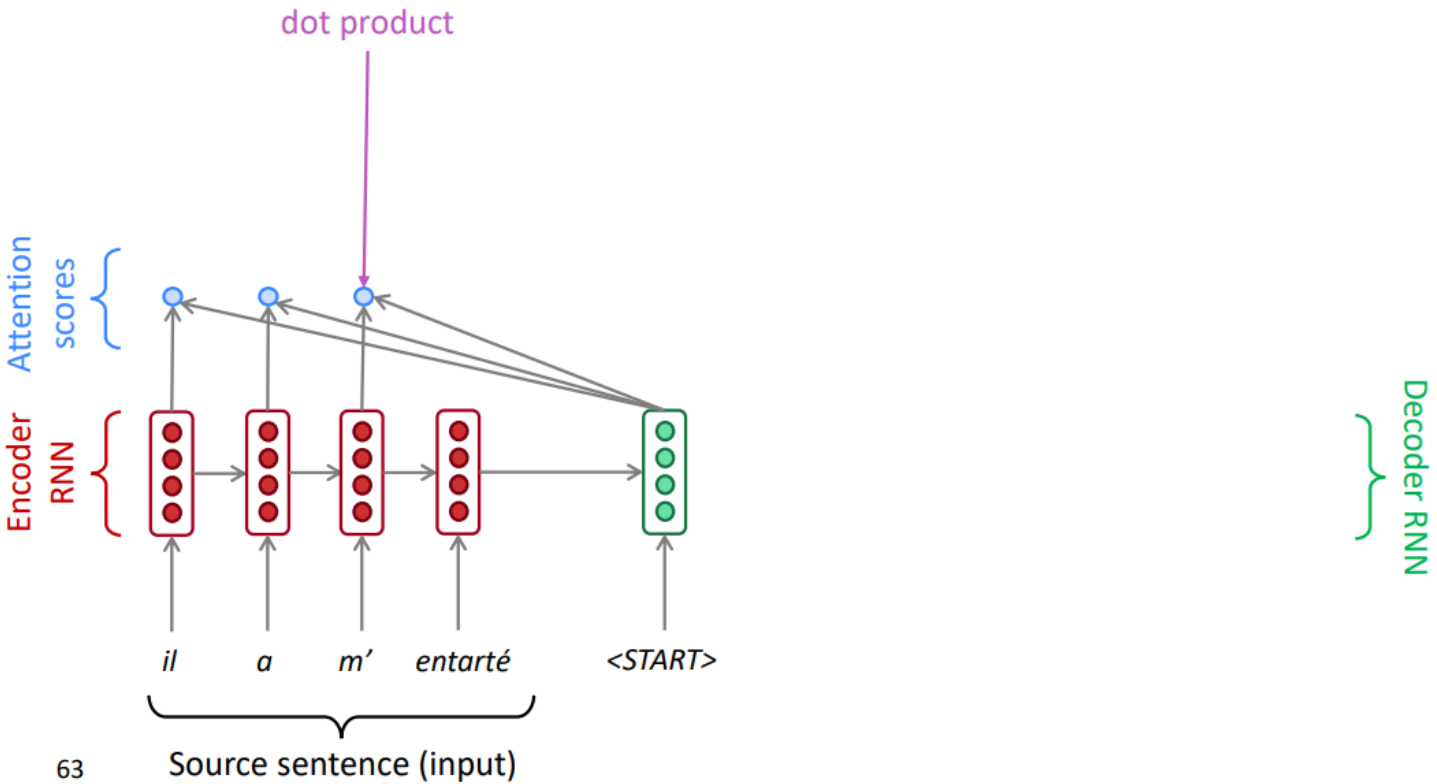


Sequence-to-sequence with attention

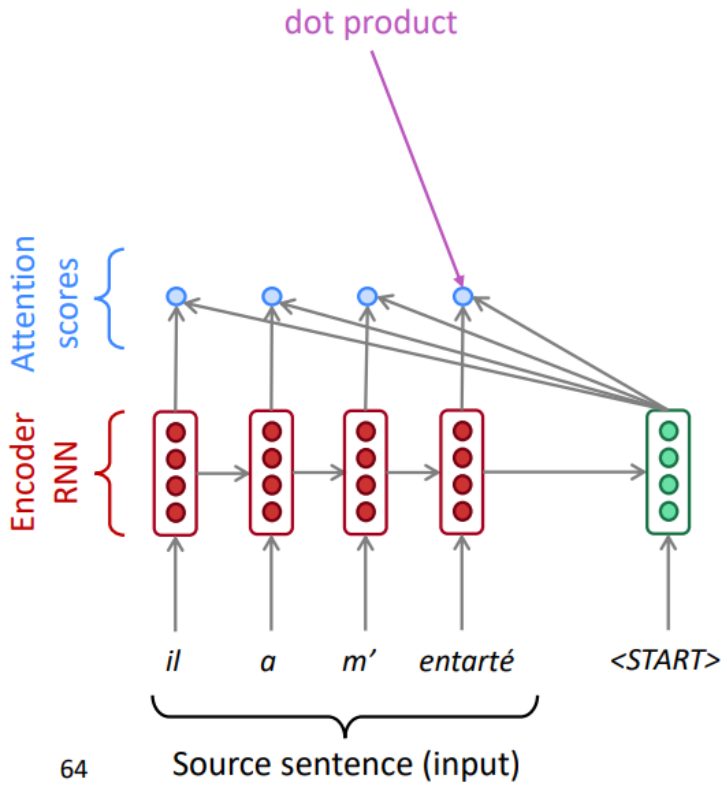


Decoder RNN

Sequence-to-sequence with attention

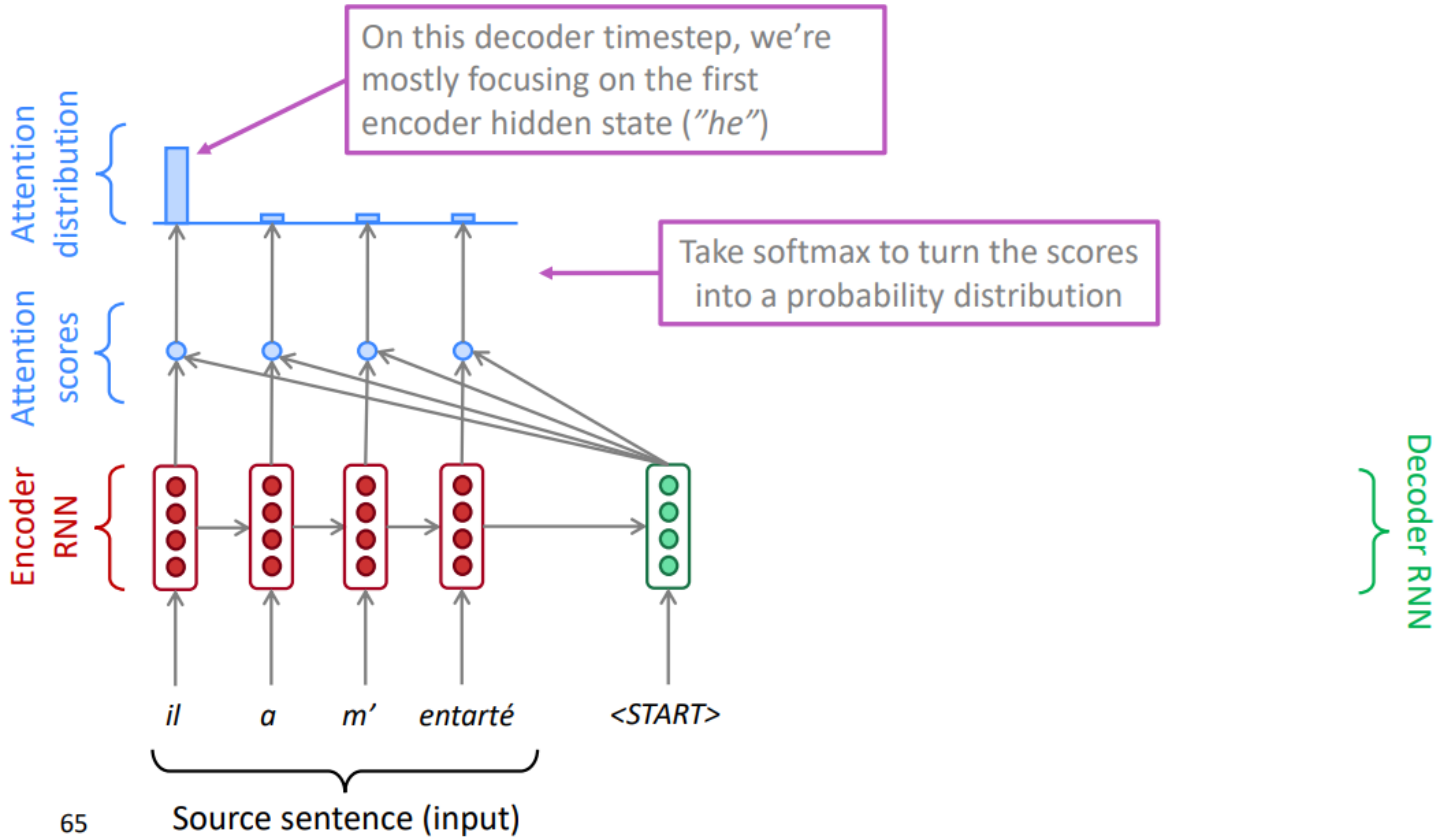


Sequence-to-sequence with attention

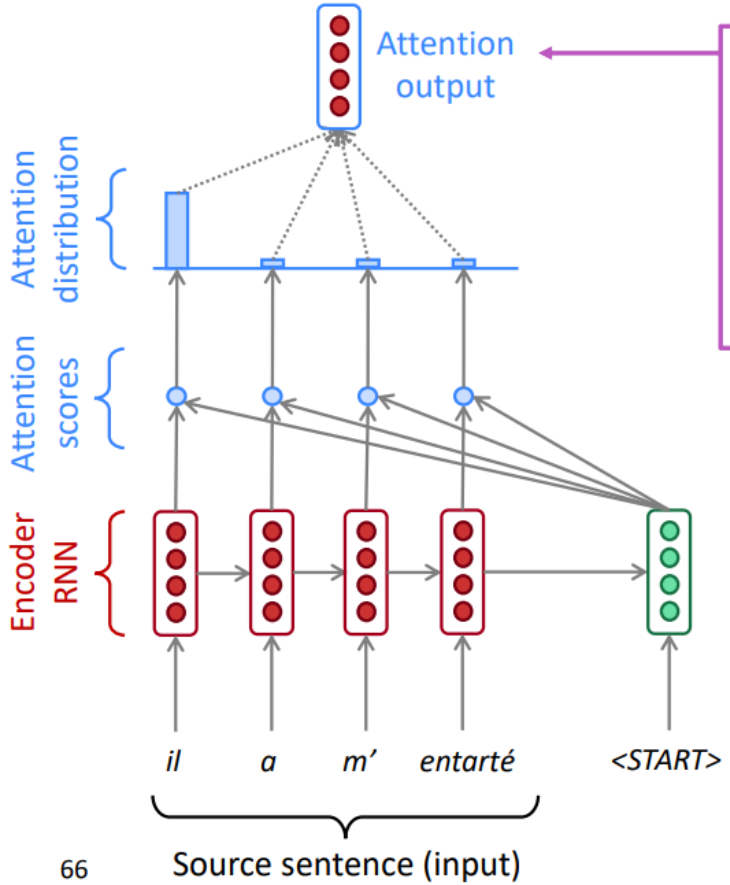


Decoder RNN

Sequence-to-sequence with attention



Sequence-to-sequence with attention

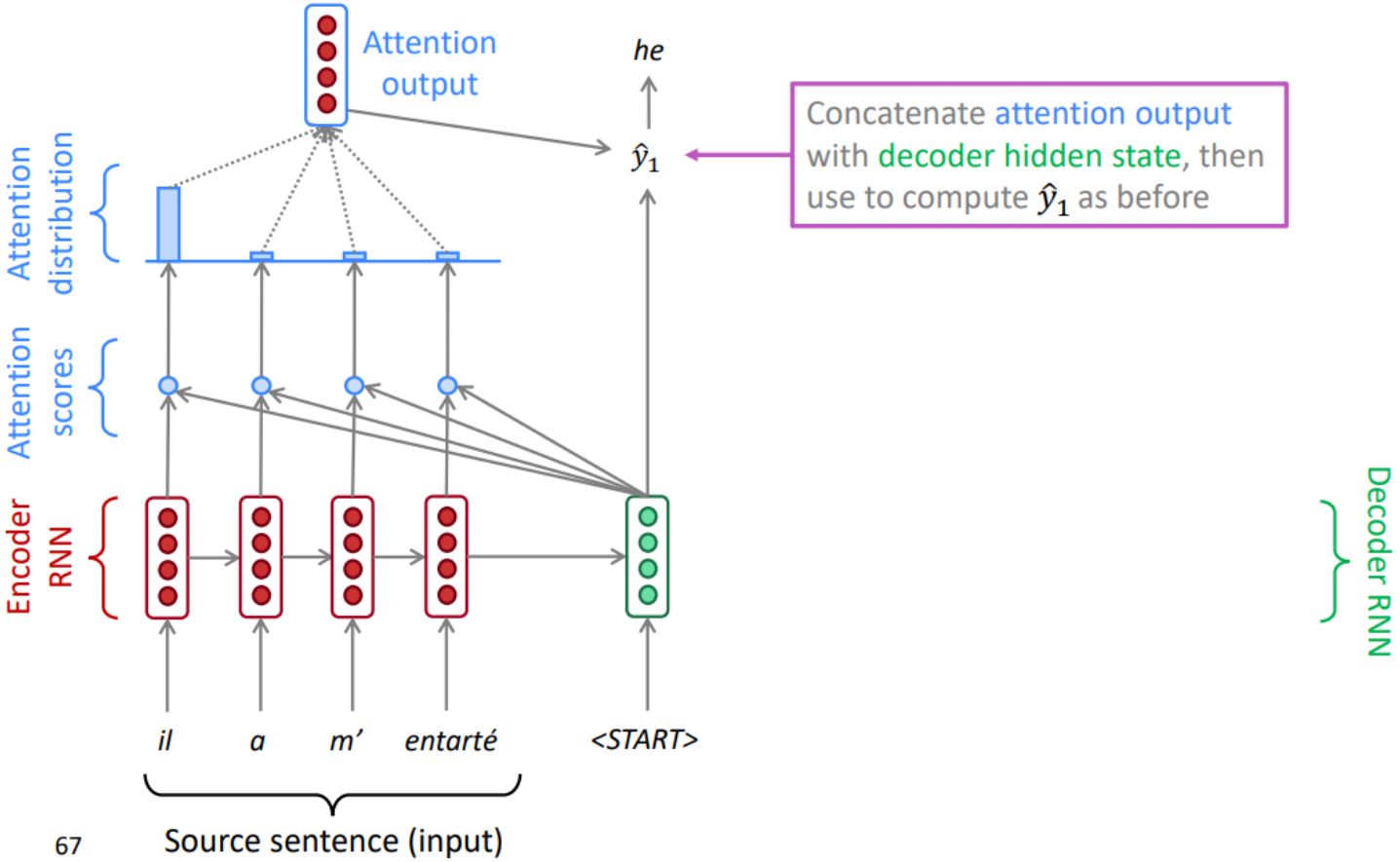


Use the attention distribution to take a **weighted sum** of the encoder hidden states.

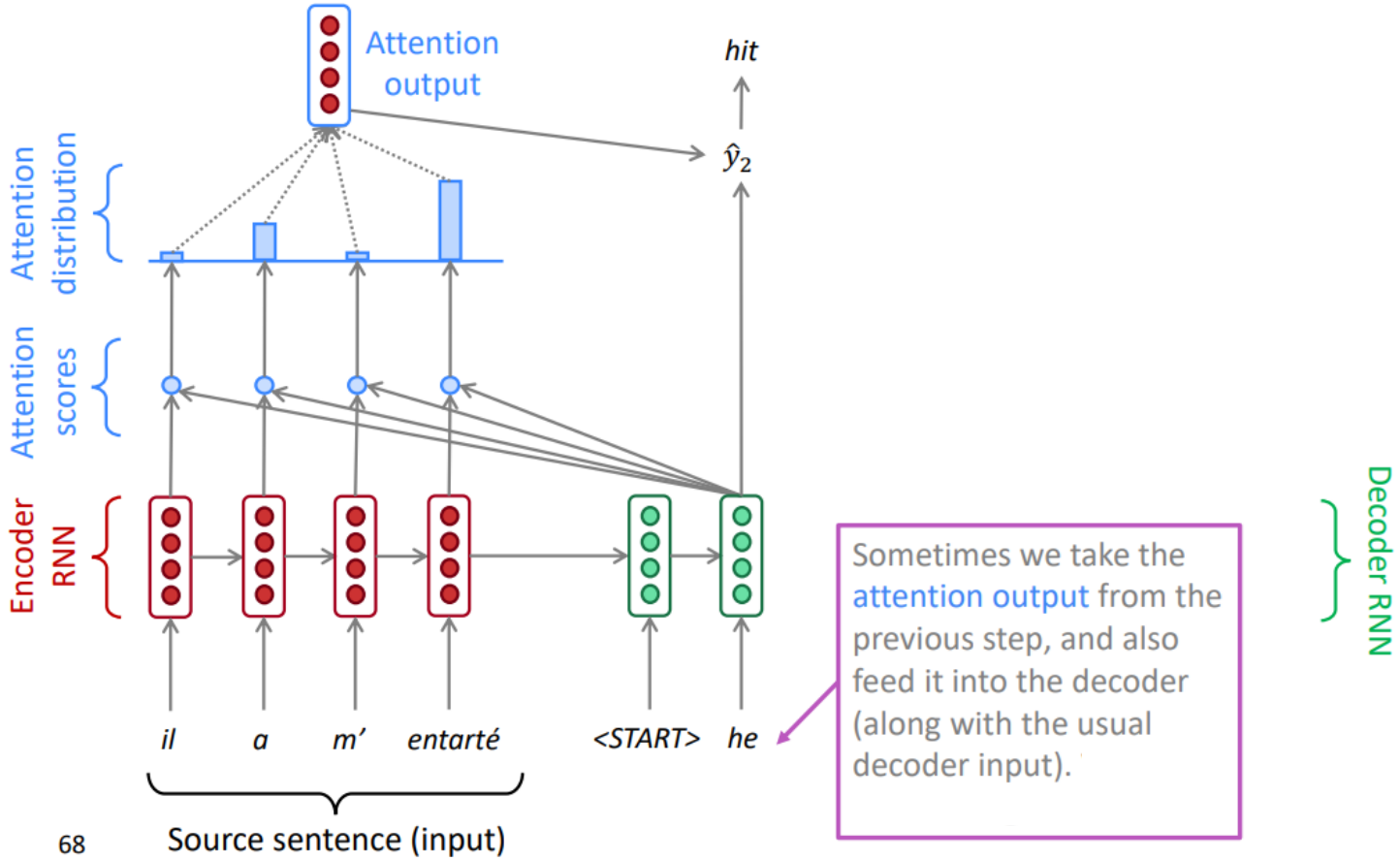
The attention output mostly contains information from the hidden states that received high attention.

Decoder RNN

Sequence-to-sequence with attention

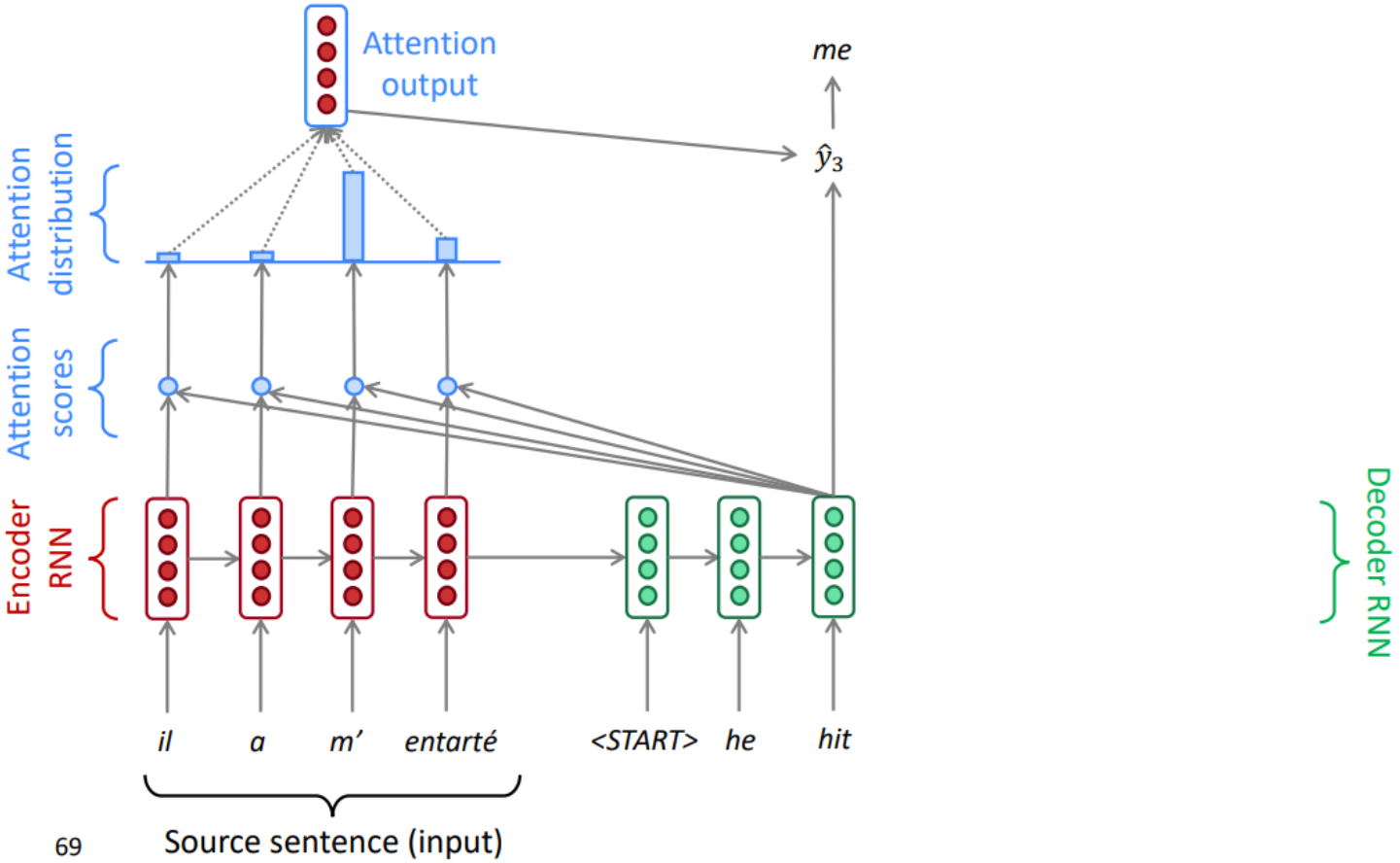


Sequence-to-sequence with attention

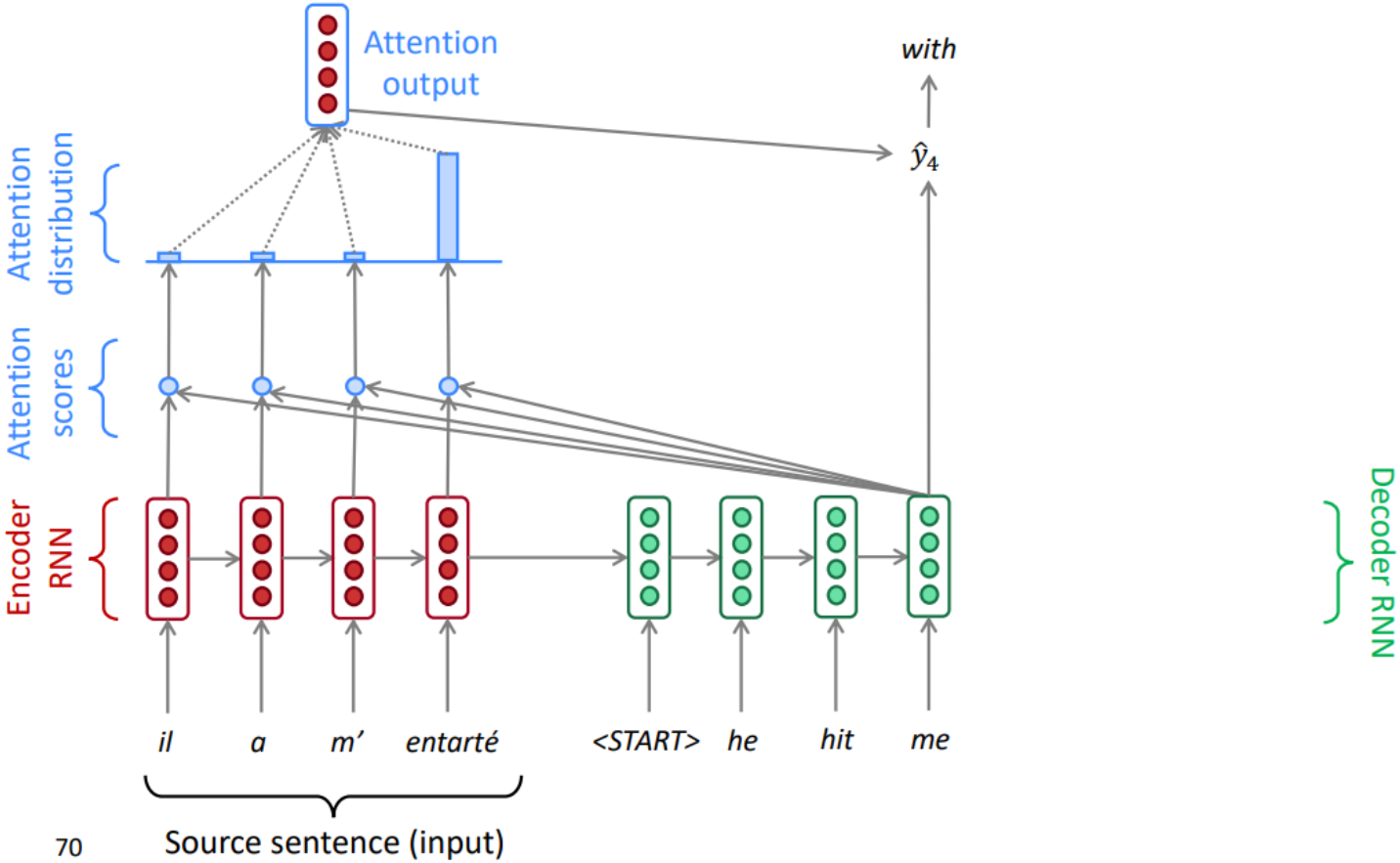


Sometimes we take the attention output from the previous step, and also feed it into the decoder (along with the usual decoder input).

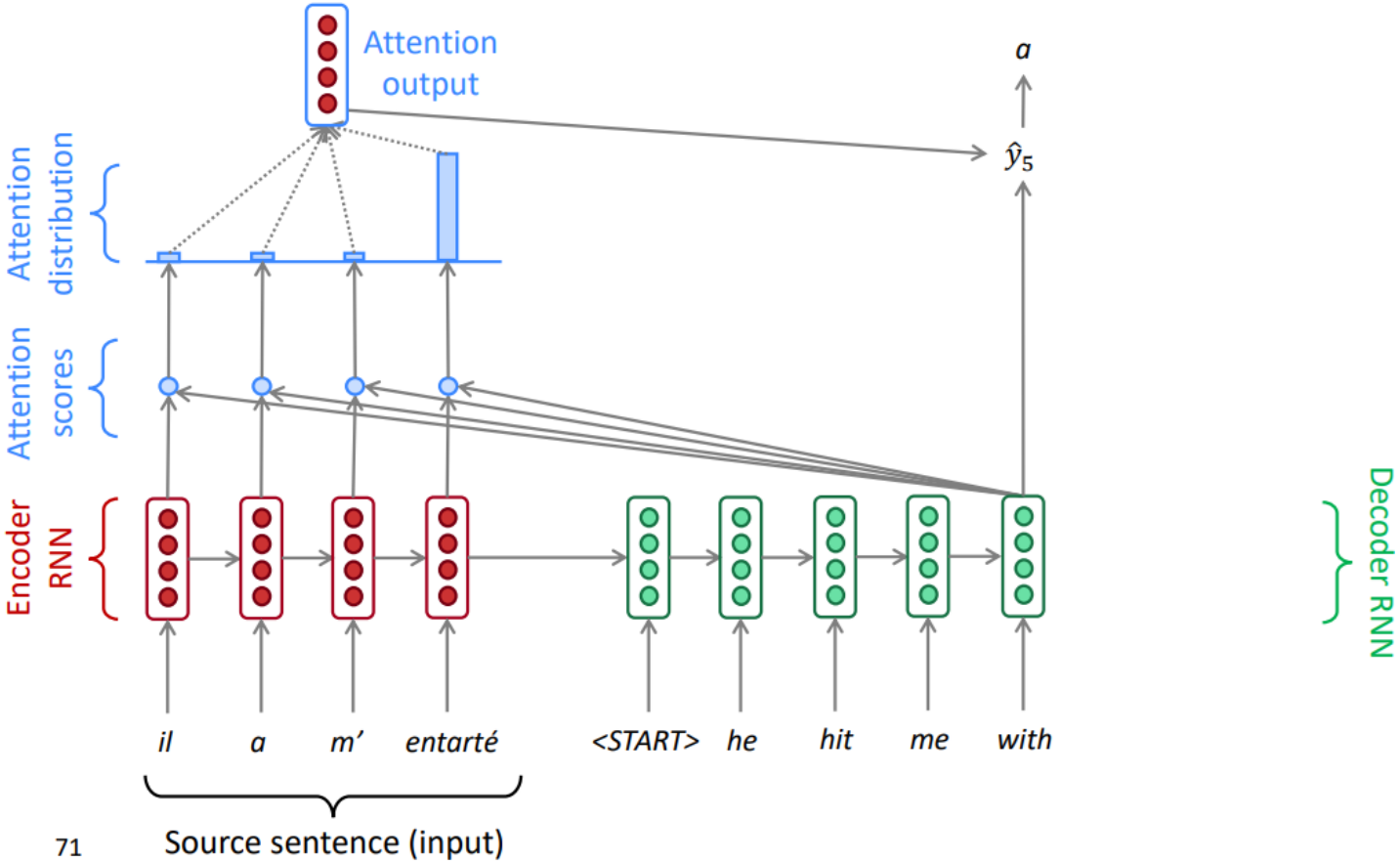
Sequence-to-sequence with attention



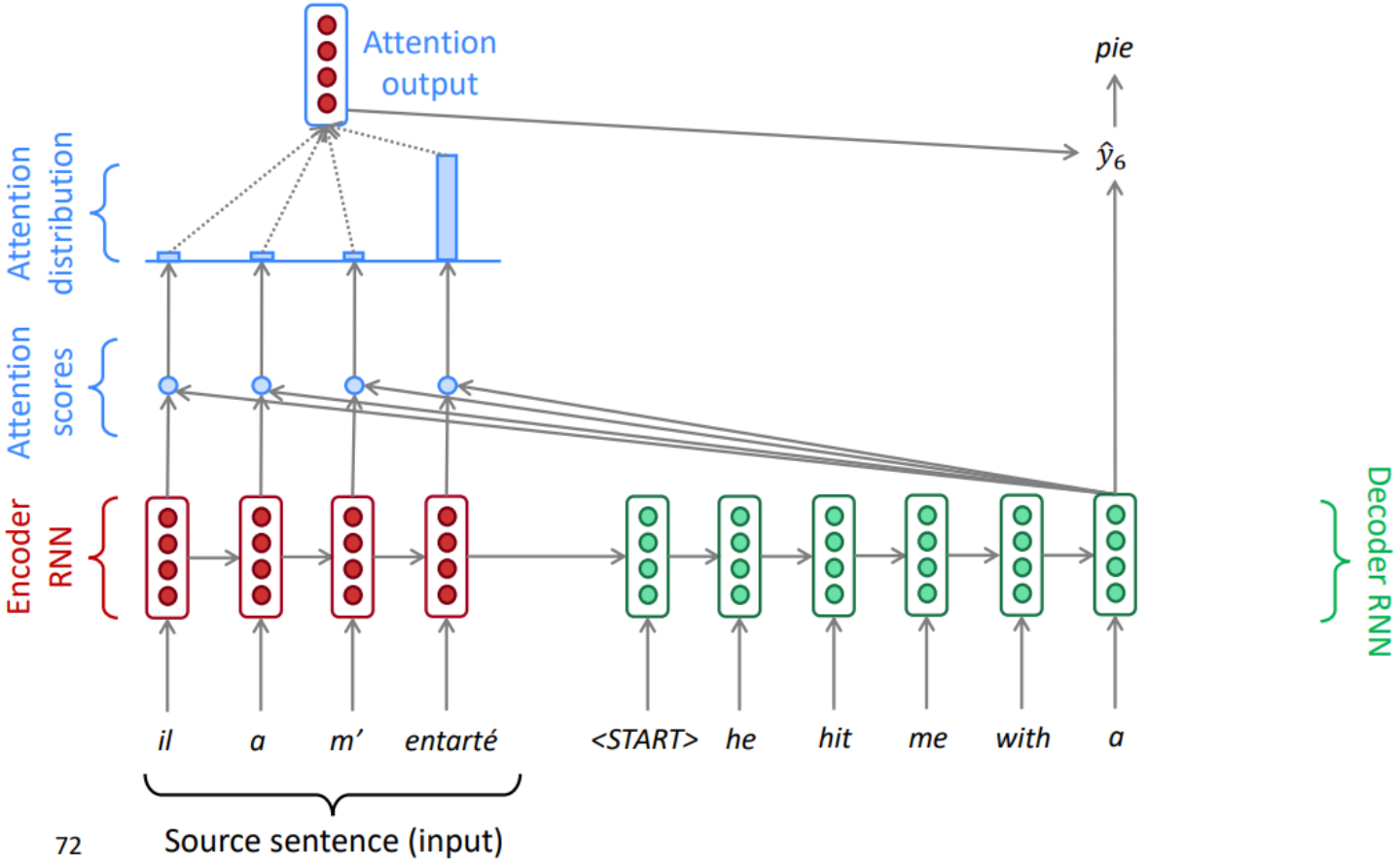
Sequence-to-sequence with attention



Sequence-to-sequence with attention

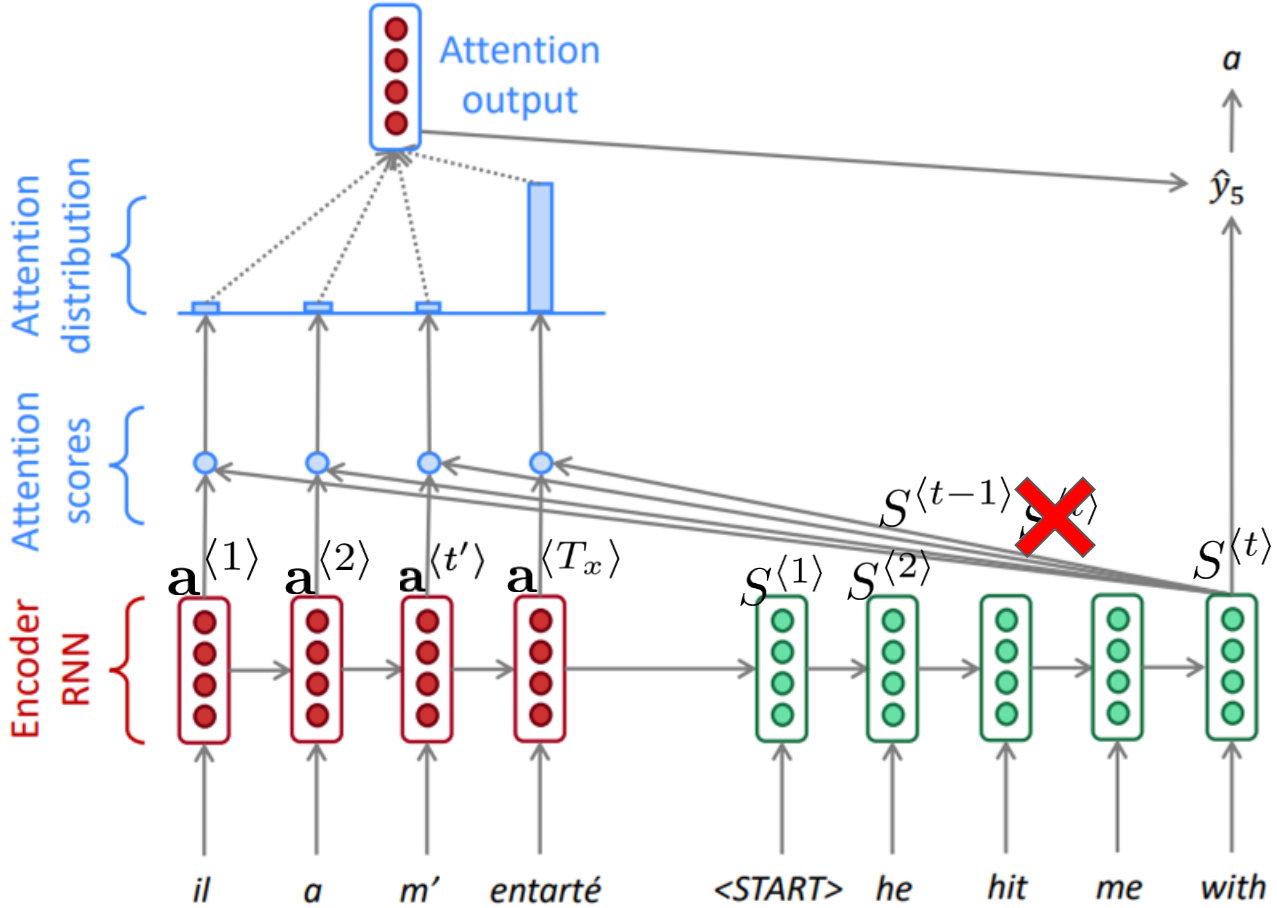


Sequence-to-sequence with attention

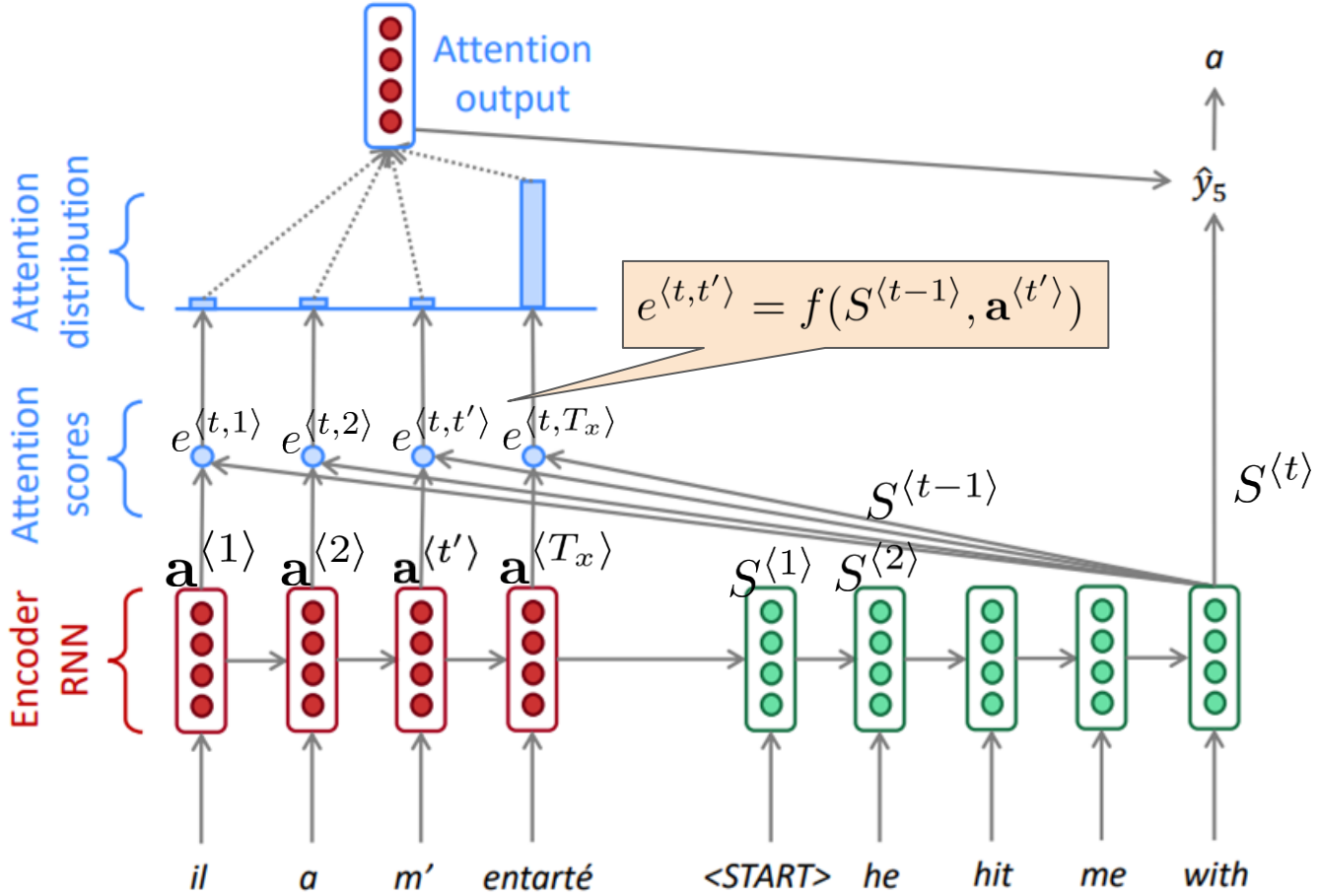


Exemplo passo-a-passo (com equações)

Atenção: as equações



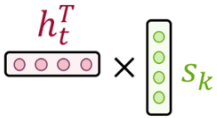
Atenção: as equações



Alignment Scoring Functions

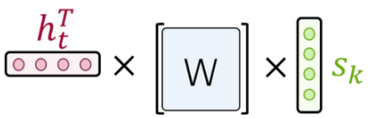
Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

Dot-product



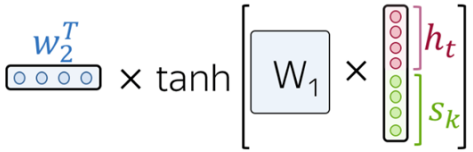
$\text{score}(h_t, s_k) = h_t^T \; s_k$

Bilinear



$\text{score}(h_t, s_k) = h_t^T \; W s_k$

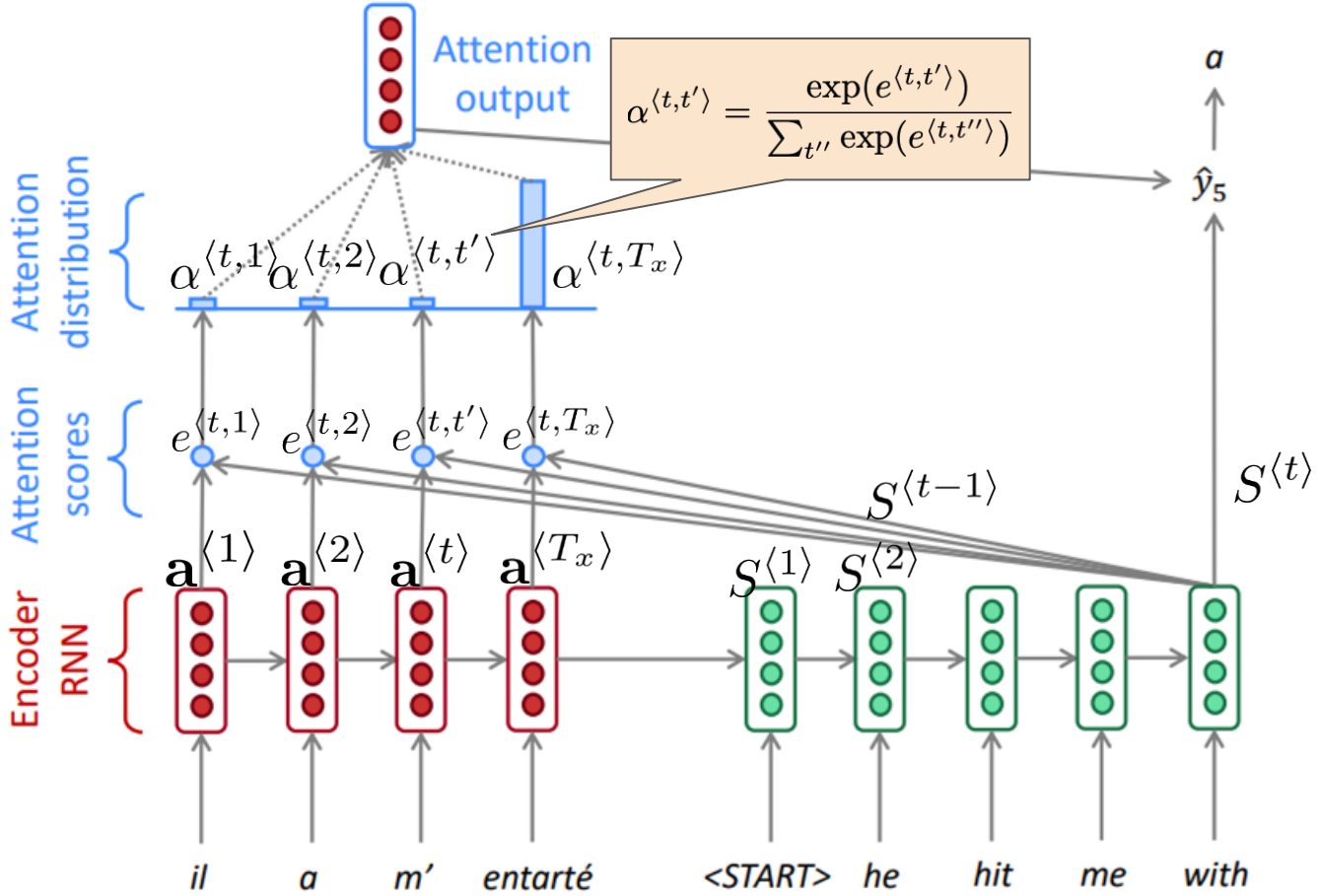
Multi-Layer Perceptron



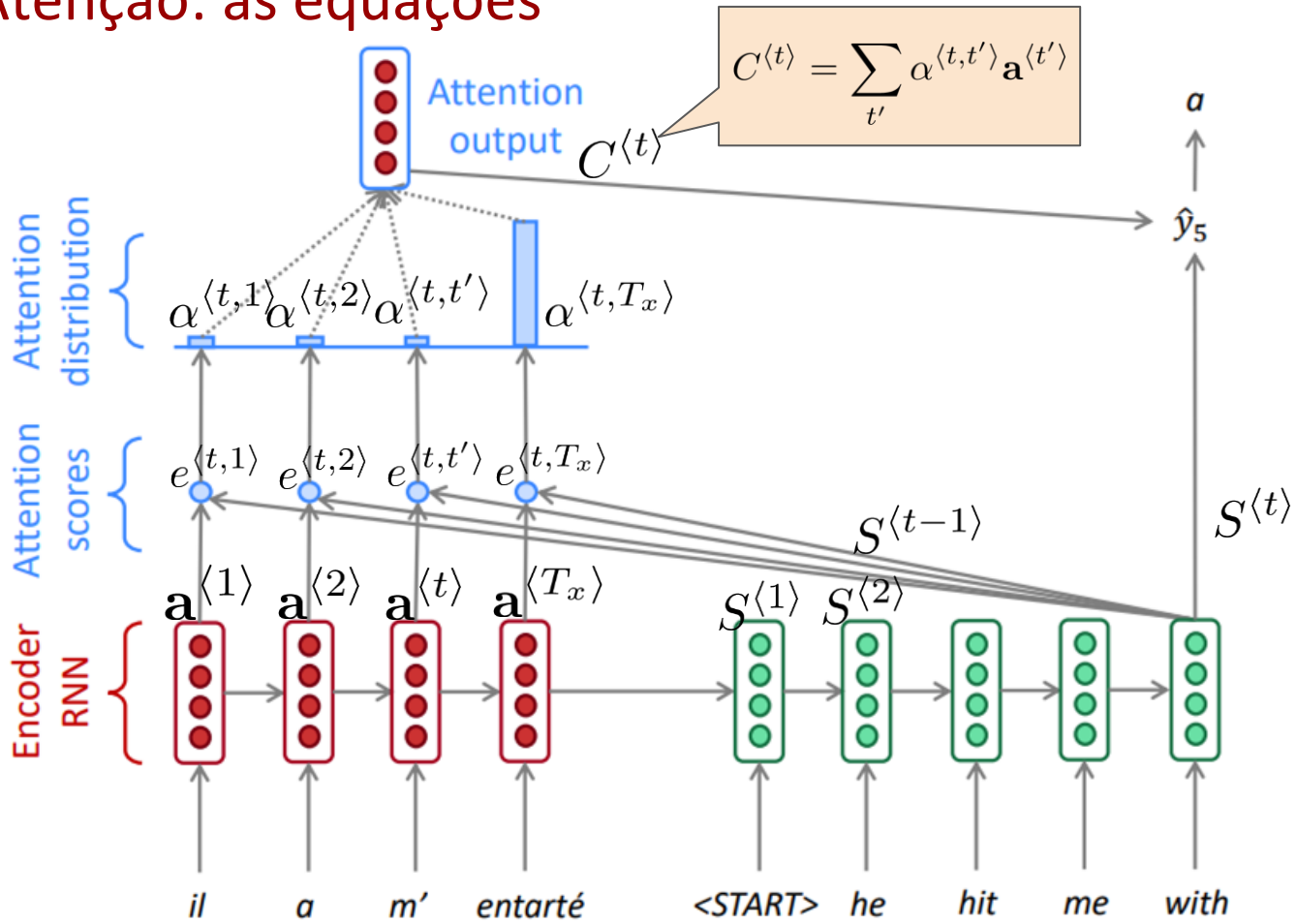
$\text{score}(h_t, s_k) = w_2^T \cdot \tanh(W_1[h_t, s_k])$

Sources: https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html
Seq2seq and Attention (lena-voita.github.io)

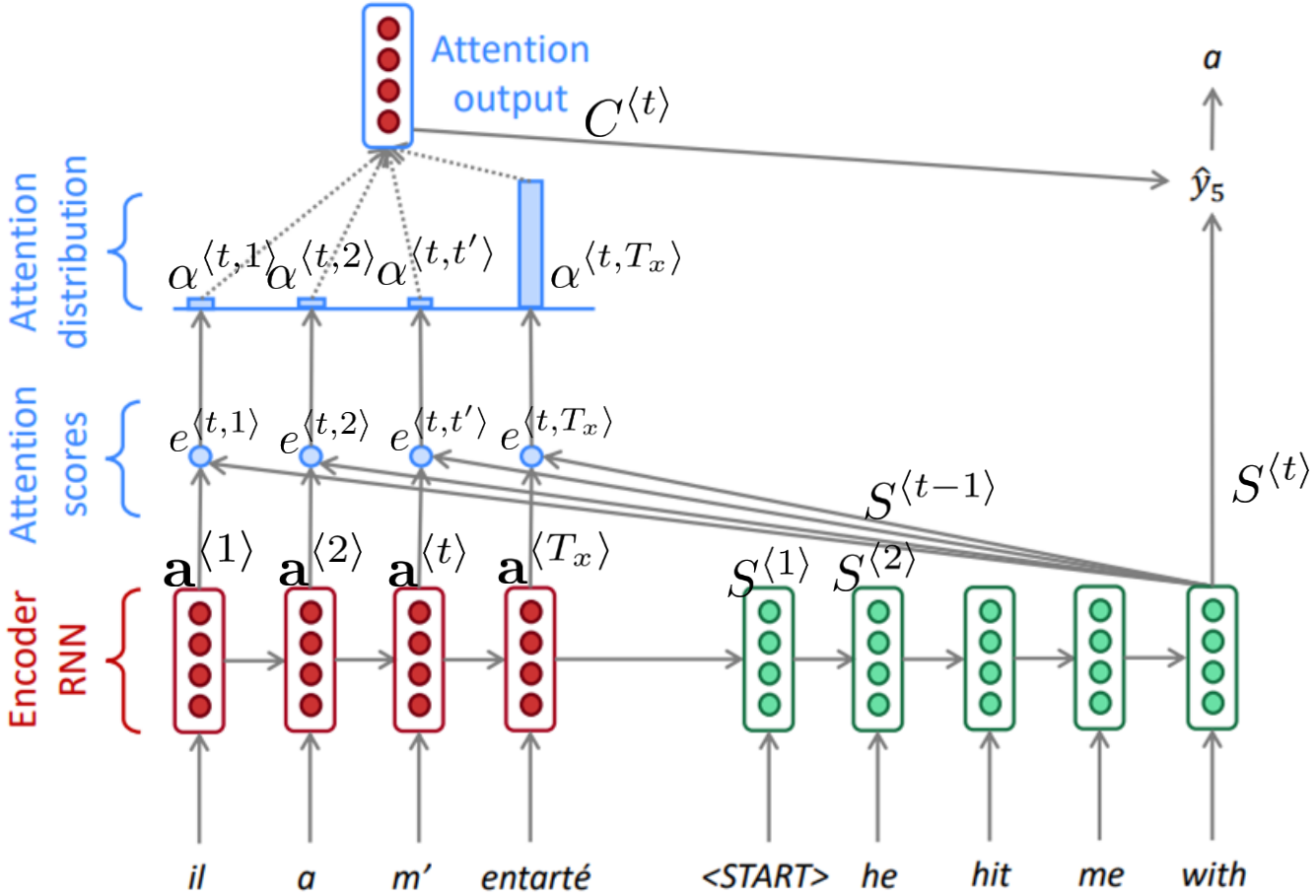
Atenção: as equações



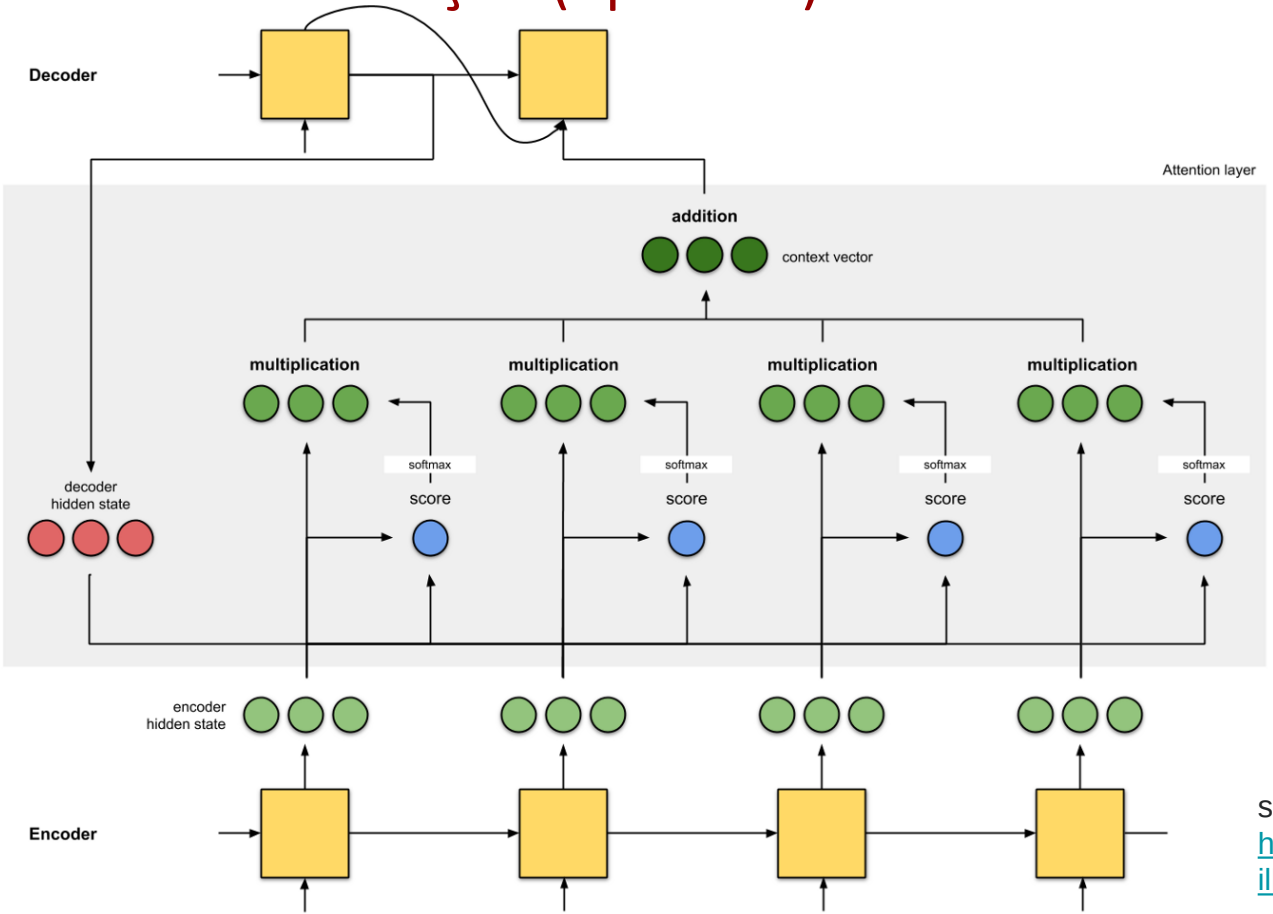
Atenção: as equações



Atenção: as equações



Outra visualização (opcional)



source:
https://towardsdatascience.com/attn-illustrated-attention-5ec4ad276ee3

Tradução por máquina baseada em atenção

- Vetor de contexto é calculado como uma média ponderada das anotações do encoder (values):

$$C^{\langle t \rangle} = \sum_{t'} \alpha^{\langle t,t' \rangle} \mathbf{a}^{\langle t' \rangle}$$

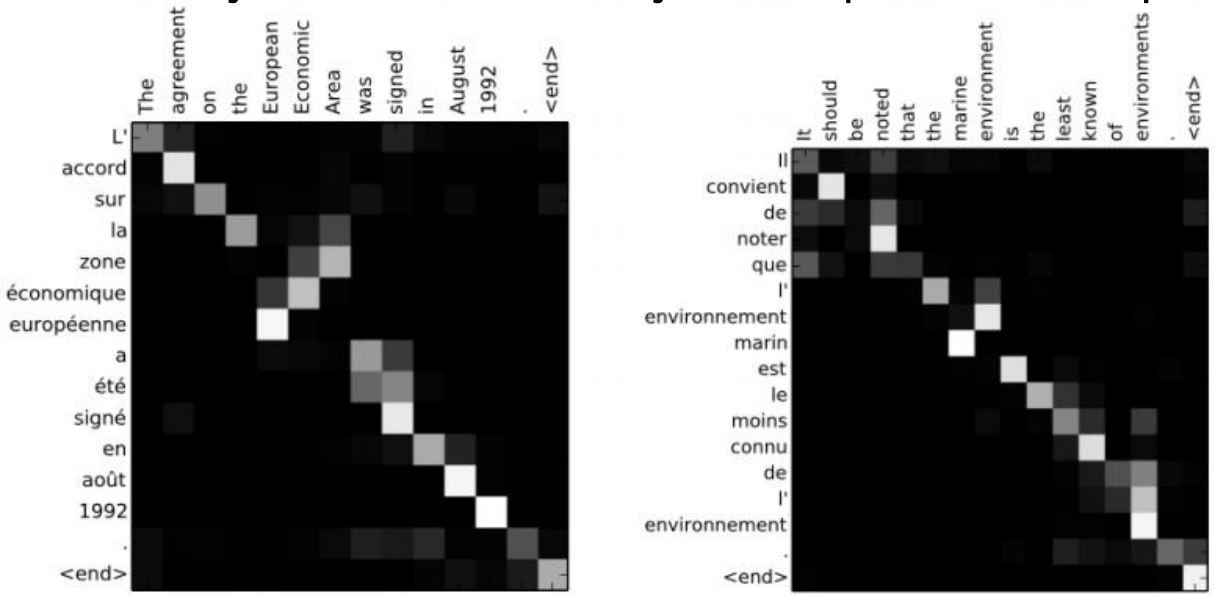
- Os pesos da atenção são calculados via softmax, onde as entradas dependem do estado do decoder (query) e das anotações (keys):

$$\alpha^{\langle t,t' \rangle} = \frac{\exp(e^{\langle t,t' \rangle})}{\sum_{t''} \exp(e^{\langle t,t'' \rangle})} \qquad e^{\langle t,t' \rangle} = f(S^{\langle t-1 \rangle}, \mathbf{a}^{\langle t' \rangle})$$

- Note que a função *f* depende do vetor de anotação e não da posição na frase. Isto é uma forma de endereçamento baseado em conteúdo.
  - E.g.: Meu modelo de linguagem diz que a próxima palavra deve ser um adjetivo. Encontre um adjetivo na entrada.

Tradução por máquina baseada em atenção

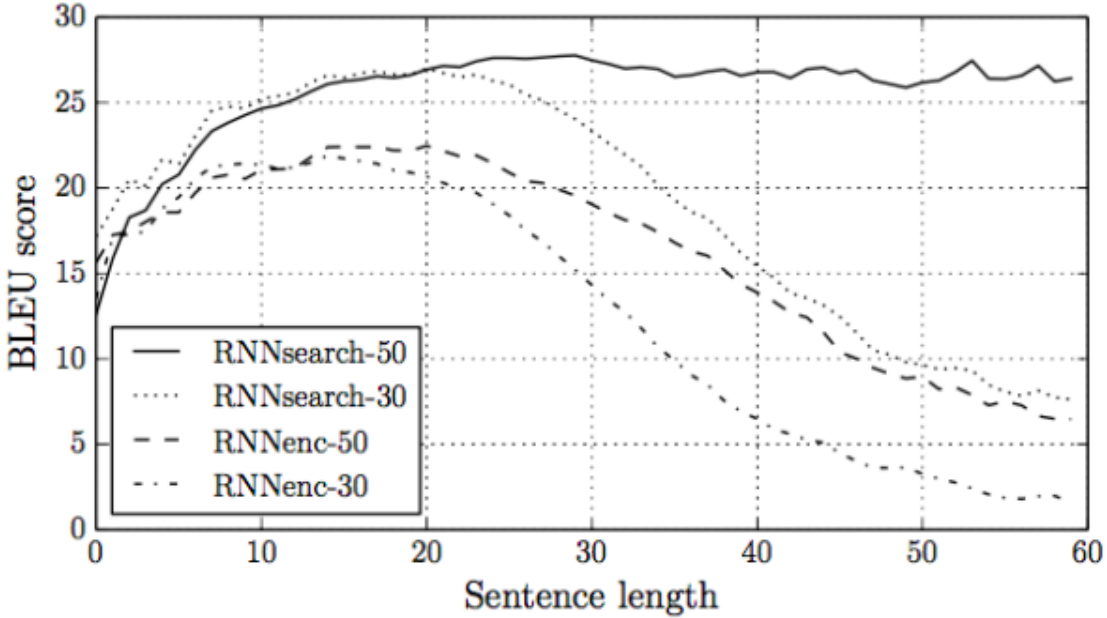
- Eis uma visualização de como a atenção é mapeada a cada passo.



- Nada força o modelo a avançar linearmente pela frase de entrada, mas de alguma forma ele aprende a fazê-lo.
  - Não é exatamente linear e.g., adjetivos em francês vem depois de substantivo

Tradução por máquina baseada em atenção

- O modelo baseado em atenção é muito melhor que modelo encoder/ decoder comum em frases longas.

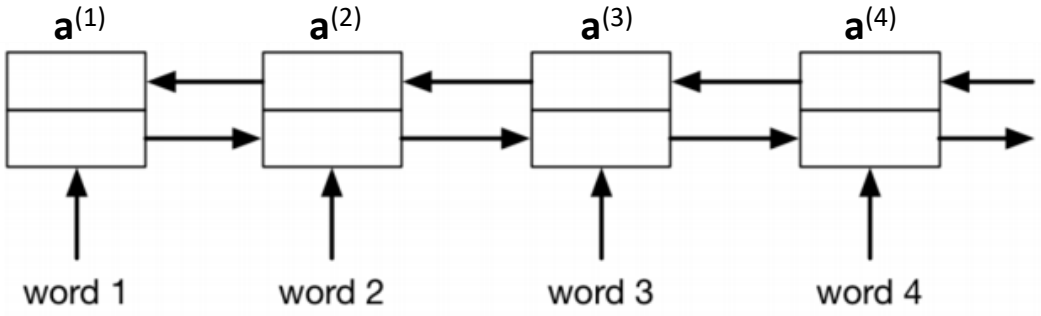


Attention é uma técnica geral de Deep Learning

- Vimos que attention é uma excelente maneira de melhorar modelo seq2seq para tradução por máquina (MT)
- Entretanto, pode-se usar attention em muitos modelos (não só seq2seq) e em muitas tarefas (não só MT)
- Definição mais geral: Dado um conjunto de *valores V* (vetores), e um vetor *query Q*, atenção é uma técnica geral para calcular uma soma ponderada dos valores, dependente da query
  - No modelo seq2seq + attention, cada estado oculto do decoder (Q) "attends to" (presta atenção) hidden states de todos os encoders (V)
  - No caso geral, usamos chaves *K* associadas aos valores *V* para calcular os scores, considerando $f(Q, K)$ (no exemplo, $K=V$)
- Desvantagem: Leva tempo quadrático $O(T_x T_y)$

Tradução por máquina baseada em atenção

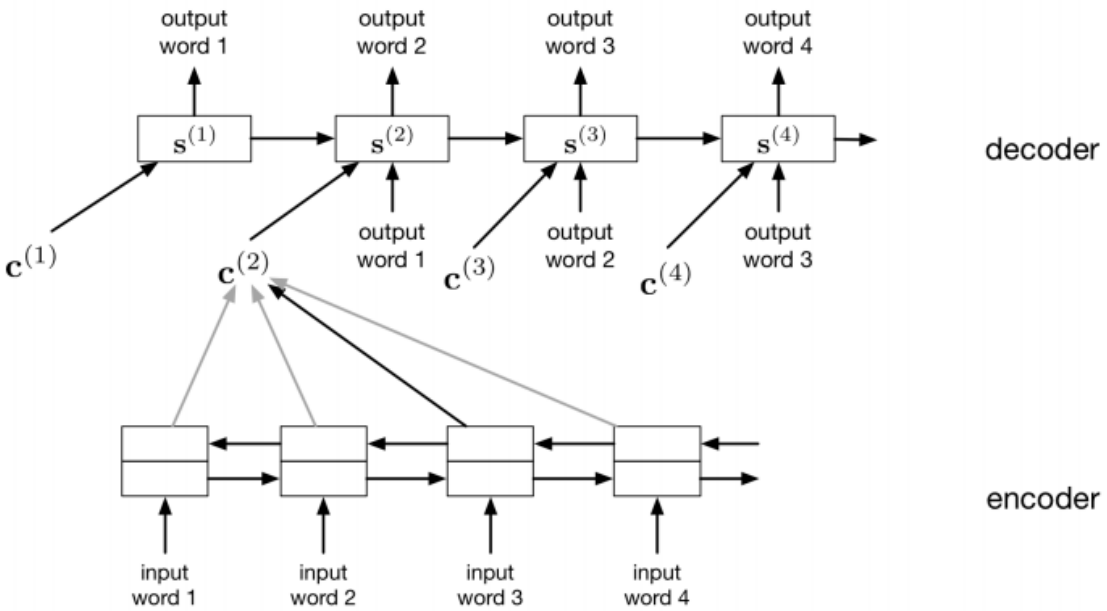
- Este modelo tem um encoder e um decoder. O encoder computa uma anotação de cada palavra na entrada.
- Para isso usa RNN bidirecional, ou seja, uma RNN "para frente" e uma RNN "para trás", cujos estados ocultos são concatenados
  - Encoder: informação anterior e posterior podem ajudar a desambiguar uma palavra, então precisamos de ambas as direções
  - Os estados ocultos retornados são chamados anotações do Encoder



Encoder

Tradução por máquina baseada em atenção

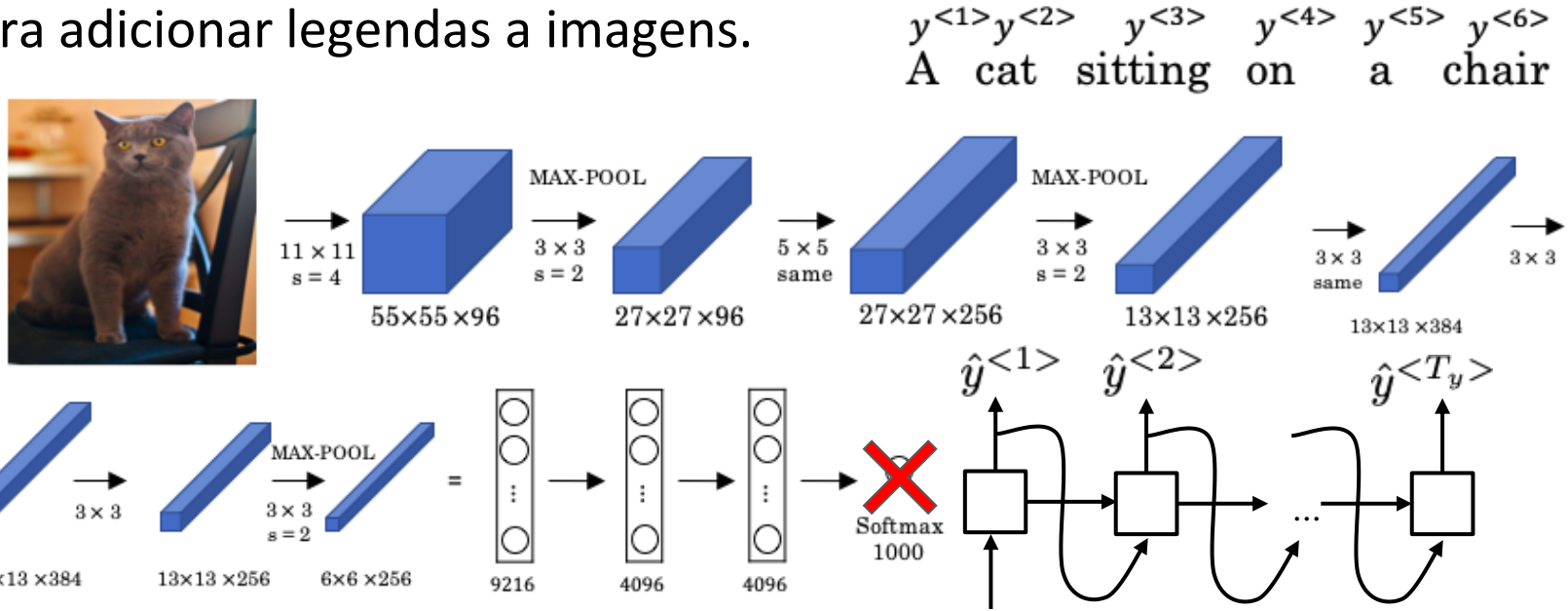
- Decoder também é uma RNN. Assim como modelo encoder/decoder de tradução, prevê uma palavra por vez, e as previsões são retro-alimentadas como entradas.
- A diferença é que também recebe um vetor de contexto $\mathbf{c}^{<t>}$ em cada passo, que é calculado considerando a entrada (output word) e as anotações do encoder.



Geração de legendas (opcional)

Geração de legendas - image captioning (opcional)

Arquitetura muito similar a de classificação de imagens também funciona para adicionar legendas a imagens.



[Mao et. al., 2014. Deep captioning with multimodal recurrent neural networks]
[Vinyals et. al., 2014. Show and tell: Neural image caption generator]
[Karpathy and Li, 2015. Deep visual-semantic alignments for generating image descriptions]

Geração de legendas baseado em atenção (opcional)

- Atenção também pode ser usado para entender imagens.
- Nós humanos não conseguimos processar uma cena inteira de vez.
  - A fóvea do olho nos dá uma visão de alta acuidade apenas em uma região bem pequena do campo de visão.
  - Em vez disso, precisamos integrar informação várias "olhadas".
- Próximos slides são baseados no paper
  - Xu et al. Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention. ICML, 2015.

Geração de legendas baseado em atenção (opcional)

- Tarefa de geração de legenda: pegue uma imagem como entrada e produza uma sequência descrevendo a imagem.
- Encoder: uma conv net de classificação (e.g., VGGNet).
  Calcula vários feature maps sobre a imagem.
- Decoder: uma RNN baseada em atenção, análoga ao decoder no modelo de tradução.
  - A cada passo, o decoder calcula uma mapa de atenção sobre toda a imagem, decidindo em que regiões se focar.
  - Recebe um vetor de contexto, que é a média ponderada dos feature maps.

Geração de legendas baseado em atenção (opcional)

- Nos permite entender onde a rede está olhando quando gera frase.

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Geração de legendas baseado em atenção (opcional)

- Também nos ajuda a entender os erros da rede.

A large white bird standing in a forest.

A woman holding a clock in her hand.

A man wearing a hat and a hat on a skateboard.

A person is standing on a beach with a surfboard.

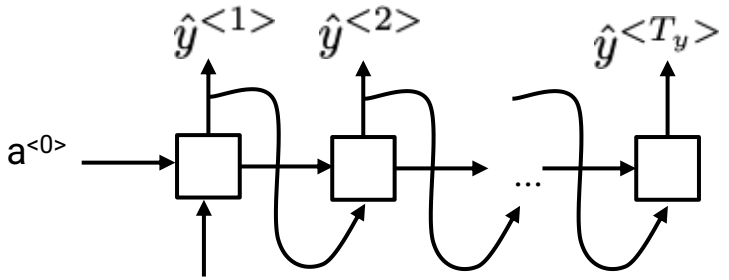
A woman is sitting at a table with a large pizza.

A man is talking on his cell phone while another man watches.

Sequência mais provável

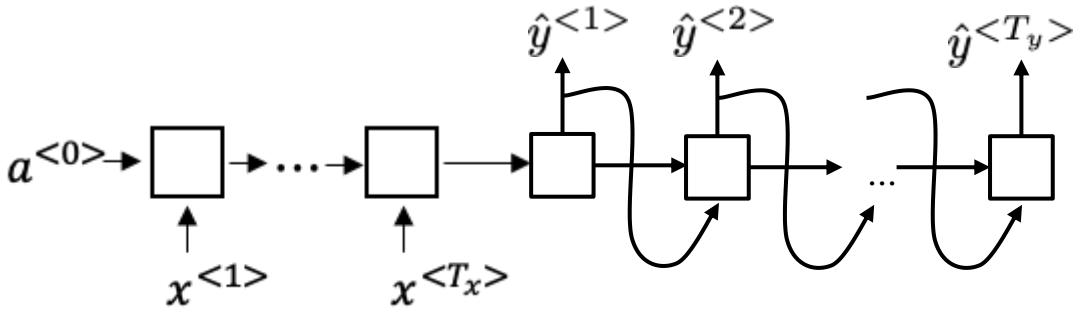
Tradução por máquina como um modelo de linguagem condicional

Modelo de linguagem:



$$P(y^{<1>}, \ldots, y^{<T_y>})$$

Tradução por máquina:



Pode ser visto como modelo de linguagem "condicional"

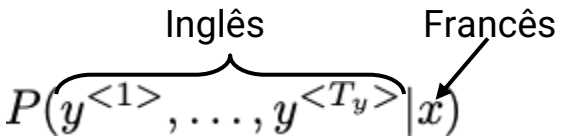
$$P(y^{<1>}, \ldots, y^{<T_y>} | x^{<1>}, \ldots, x^{<T_x>})$$

- Dois modelos anteriores são muito similares: o modelo de linguagem é igual ao decoder do modelo de tradução
- Porém, ao invés de começar com um vetor de 0s, o decoder recebe como entrada a sentença codificada (saída do encoder)
- Por isso, modelo de tradução pode ser visto como um modelo condicional de linguagem: ao invés de modelar a probabilidade de qualquer sequência, ele modela a probabilidade de uma tradução, dada uma sentença em francês

Encontrando a tradução mais provável

Jane visite l'Afrique en septembre.

- Jane is visiting Africa in September.
- Jane is going to be visiting Africa in September.
- In September, Jane will visit Africa.
- Her African friend welcomed Jane in September.

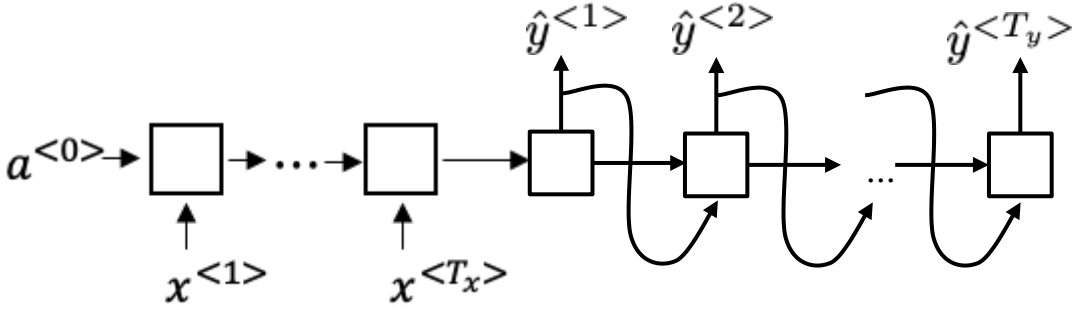


Não queremos que tradução seja uma amostra aleatória da distribuição.

Queremos encontrar $\arg\max_{y^{<1>},\ldots,y^{<T_y>}} P(y^{<1>}, \ldots, y^{<T_y>}|x)$

Que algoritmo devemos usar?

Por que não usar busca gulosa?



$$\arg \max_{\hat{y}^{<1>},\ldots,\hat{y}^{<T_y>}} P(\hat{y}^{<1>},\ldots,\hat{y}^{<T_y>}|x)$$

- Jane is visiting Africa in September.
- Jane is going to be visiting Africa in September.

A primeira tradução é melhor que a segunda. <u>No entanto, se</u>

P(Jane is going | x) > P(Jane is visiting | x),

Uma busca gulosa retornaria a 2a. tradução.

Busca exaustiva também não é factível.

Exemplo:

- Sequência de tamanho 10
- Tamanho do vocabulário: 10k
- Busca exaustiva: 10000^{10} possibilidades

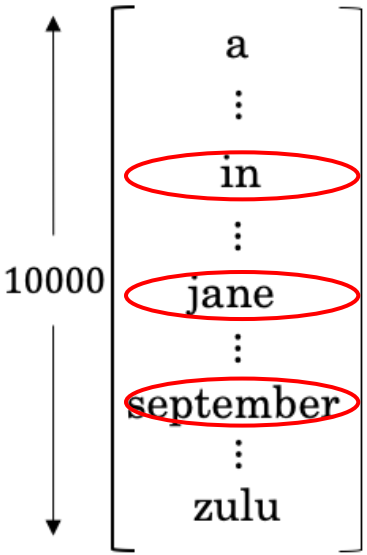
Solução precisa ser uma **busca aproximada.**

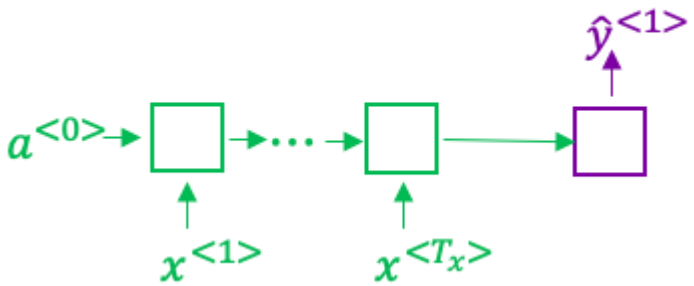
Beam Search

Algoritmo Beam Search

Para resolver (em parte) o problema da busca gulosa, o beam search mantém uma lista com B caminhos possíveis a cada passo. Exemplo com beam width B=3:

Passo 1: pegar 1a palavra



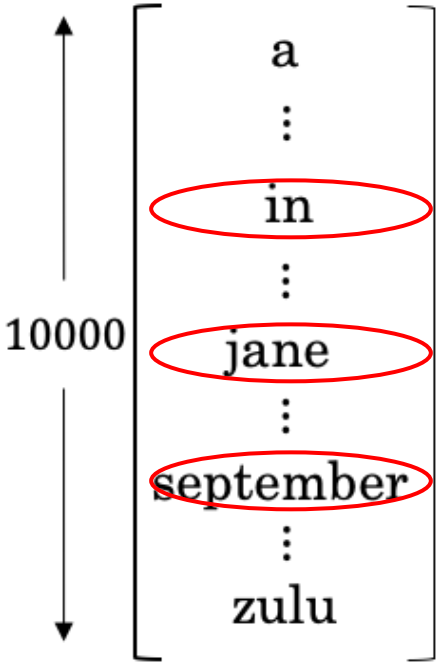


$$P(y^{<1>} \mid x)$$

- Lowercase usado para simplificar o problema
- Busca gulosa pega apenas a alternativa mais provável a cada passo
- Beam search considera B possibilidades (neste caso, B=3)
    - Após passar a sentença pelo encoder, a distribuição do softmax é utilizada para armazenar as B opções mais prováveis

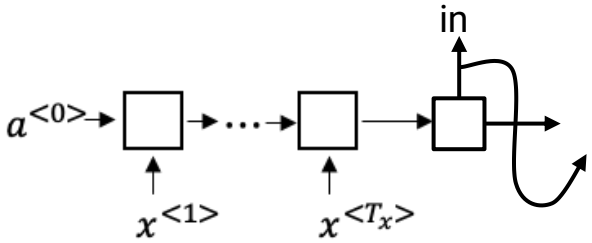
Algoritmo Beam Search (B=3)

Passo 1



Passo 2

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

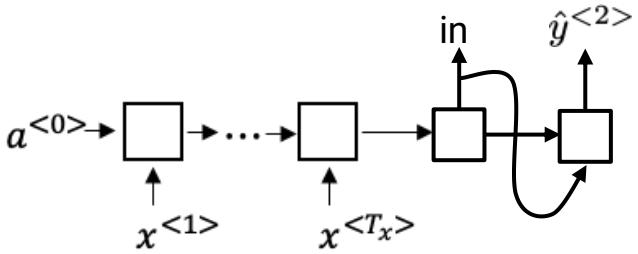


Algoritmo Beam Search (B=3)

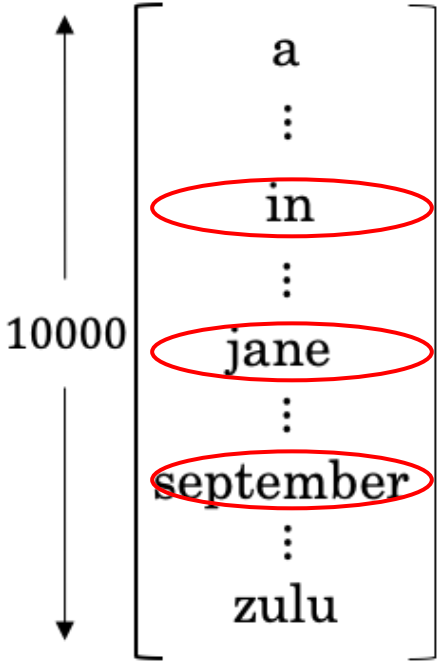
Passo 1

Passo 2

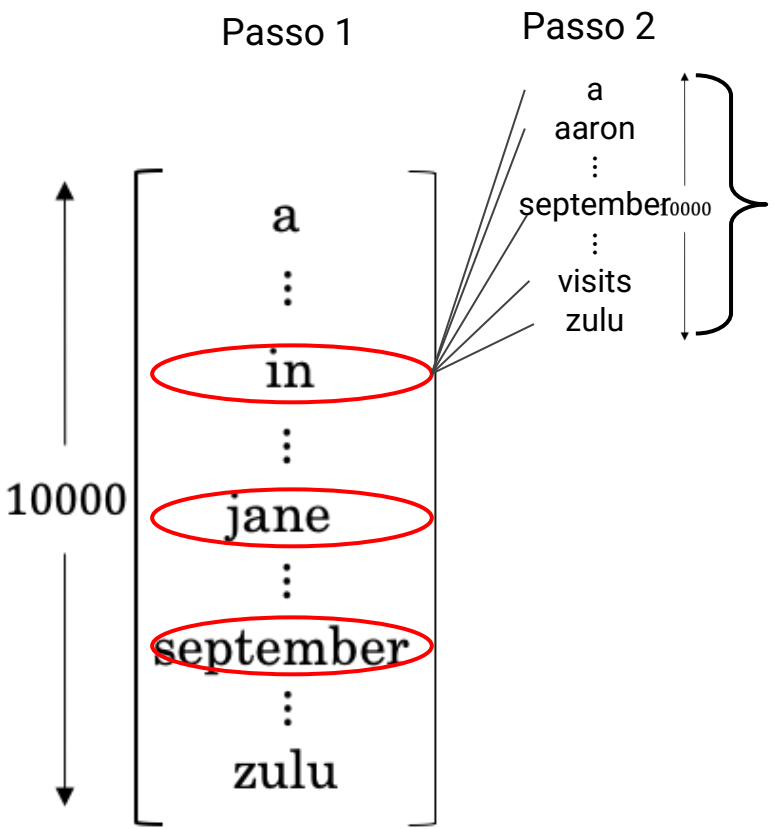
$$P(y^{<1>}, y^{<2>}|x) = P(y^{<1>}|x)P(y^{<2>}|x, y^{<1>})$$



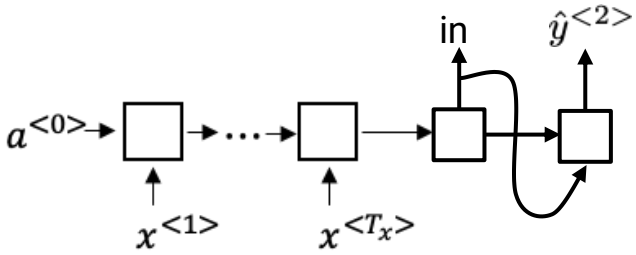
$P(y^{<2>}|x, \text{“in”})$



Algoritmo Beam Search (B=3)

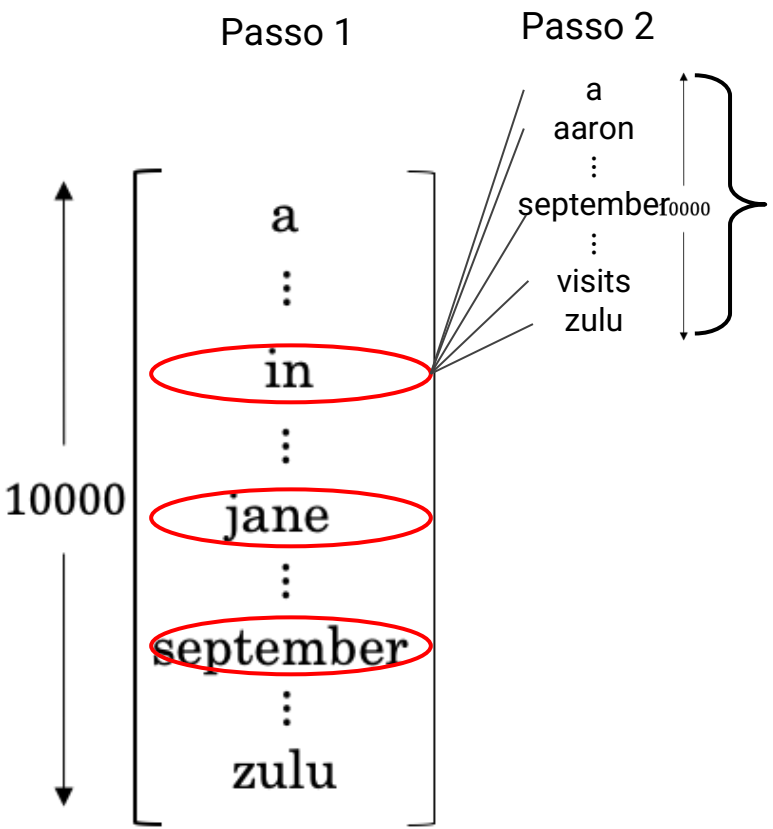


$$P(y^{<1>}, y^{<2>}|x) = P(y^{<1>}|x)P(y^{<2>}|x, y^{<1>})$$

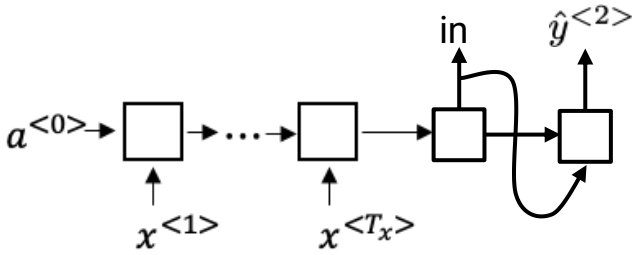


$P(y^{<2>}|x, \text{“in”})$

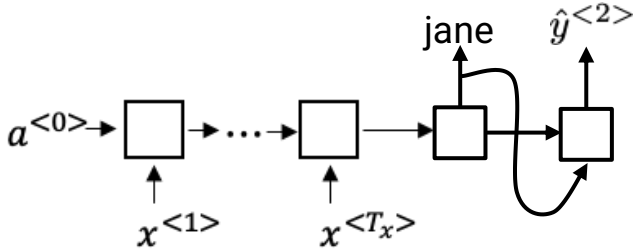
Algoritmo Beam Search (B=3)



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

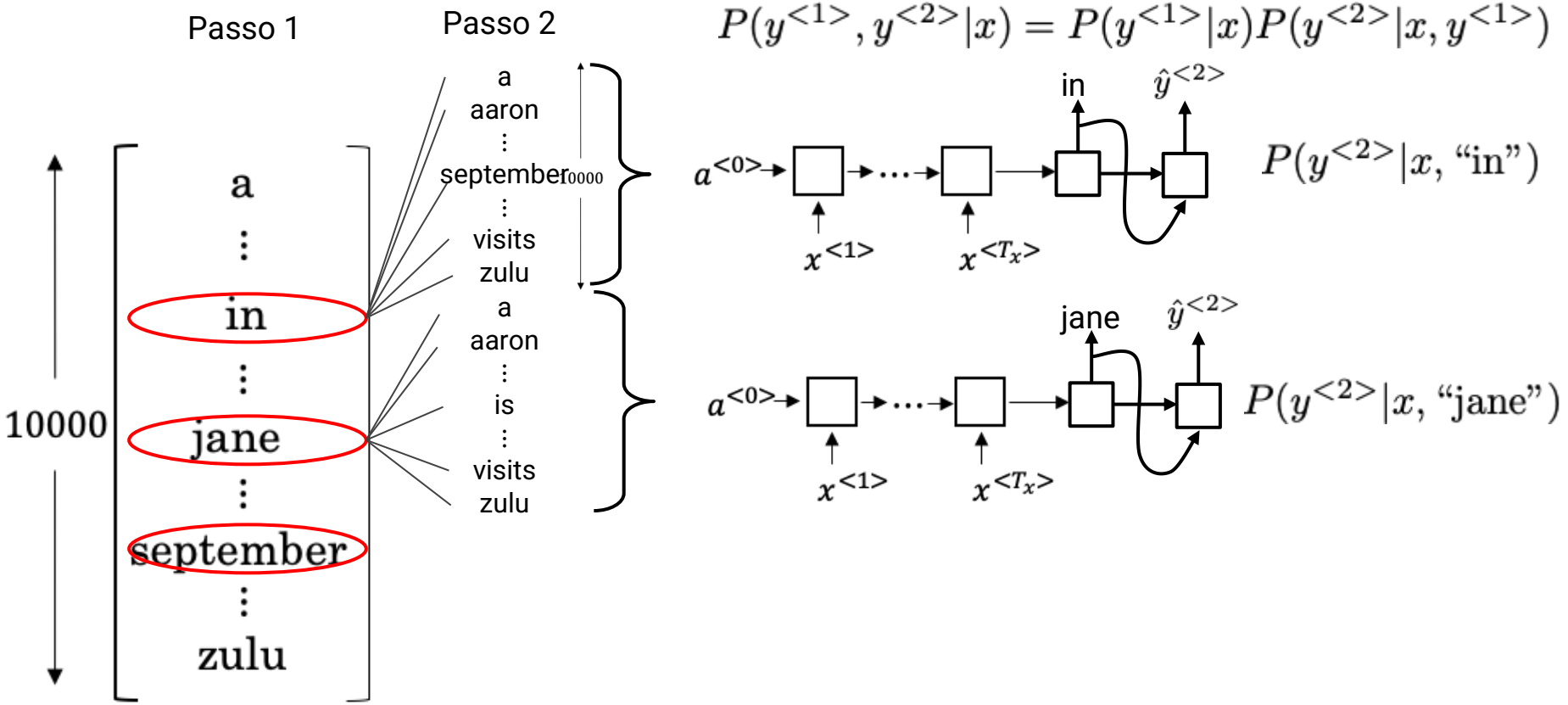


$P(y^{<2>}|x, \text{“in”})$

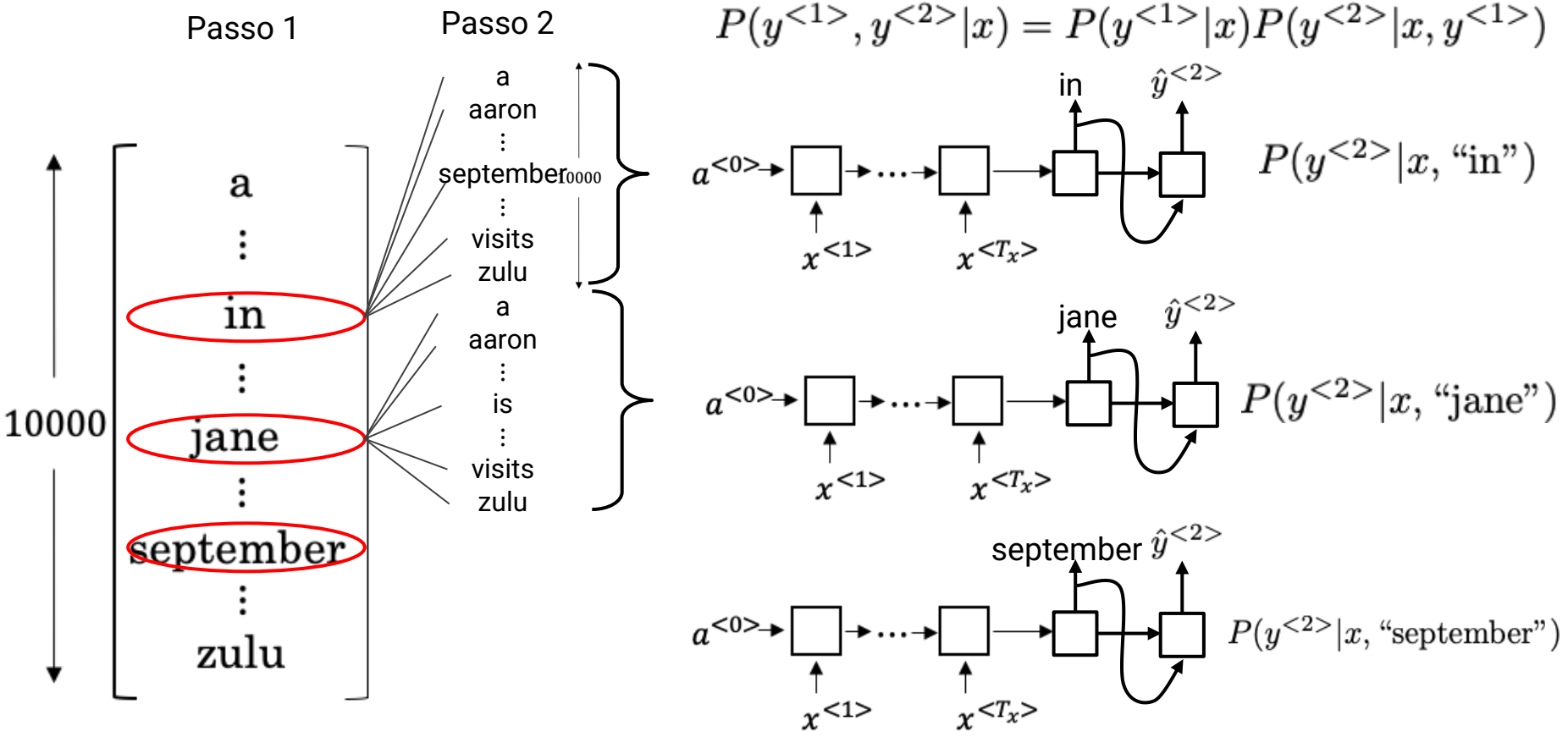


$P(y^{<2>}|x, \text{“jane”})$

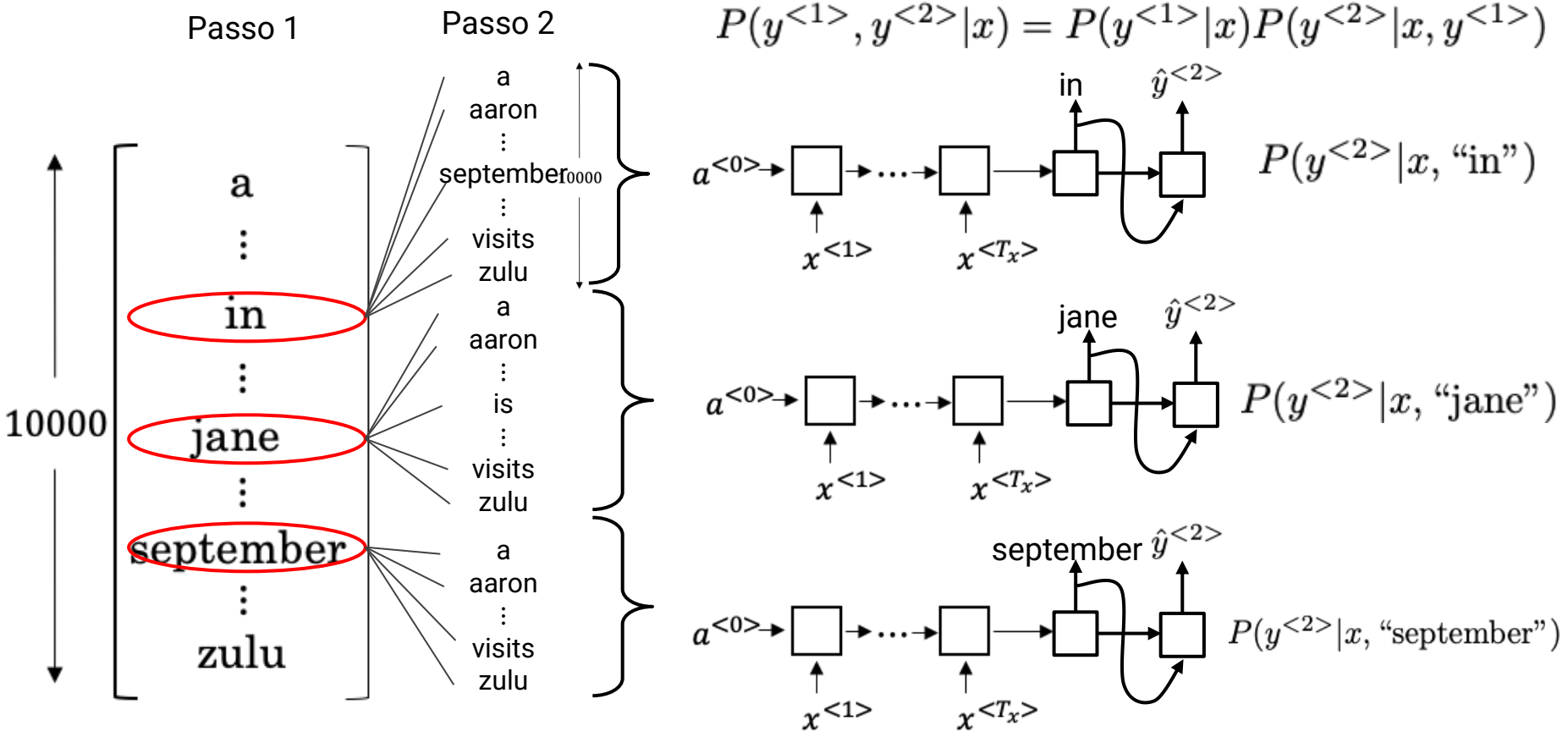
Algoritmo Beam Search (B=3)



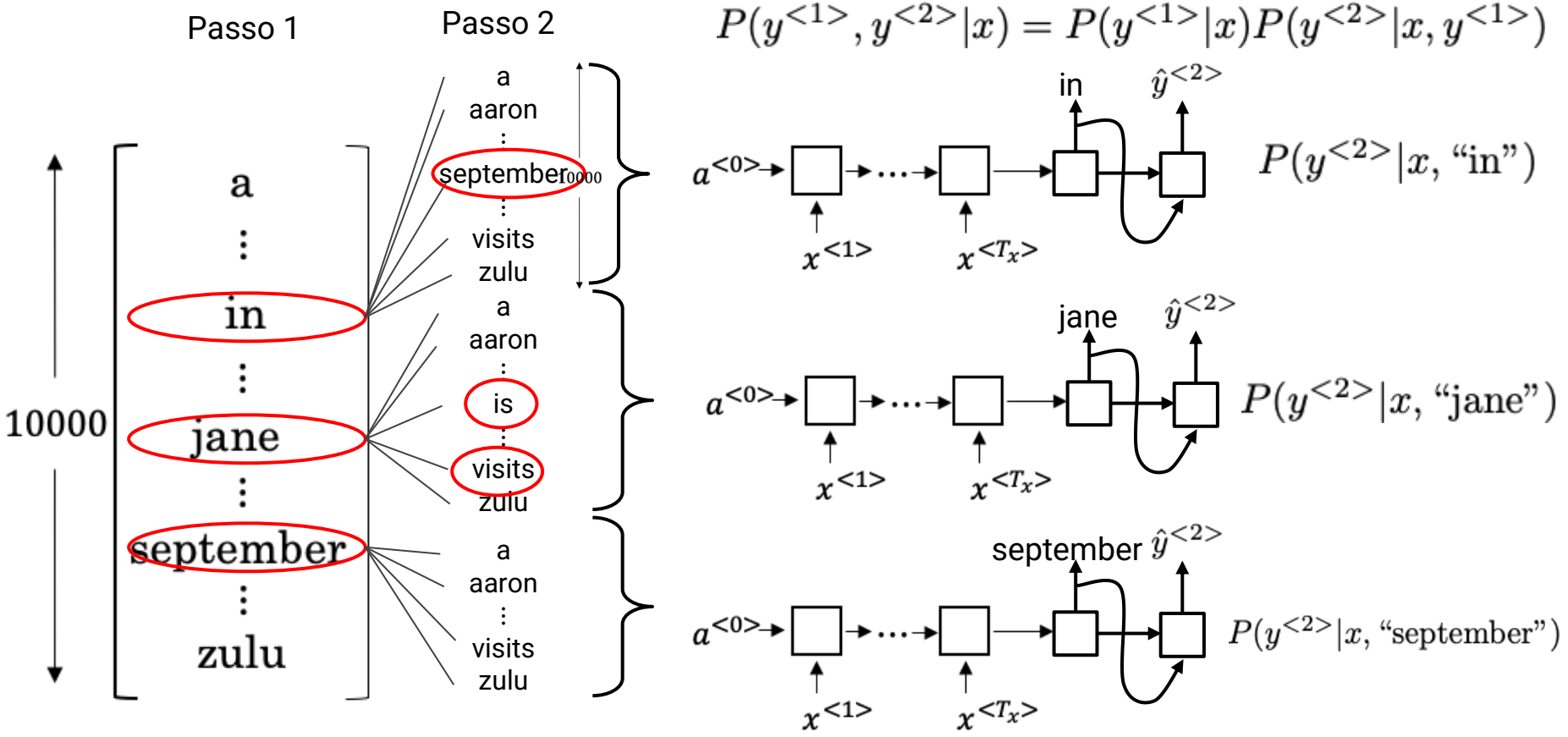
Algoritmo Beam Search (B=3)



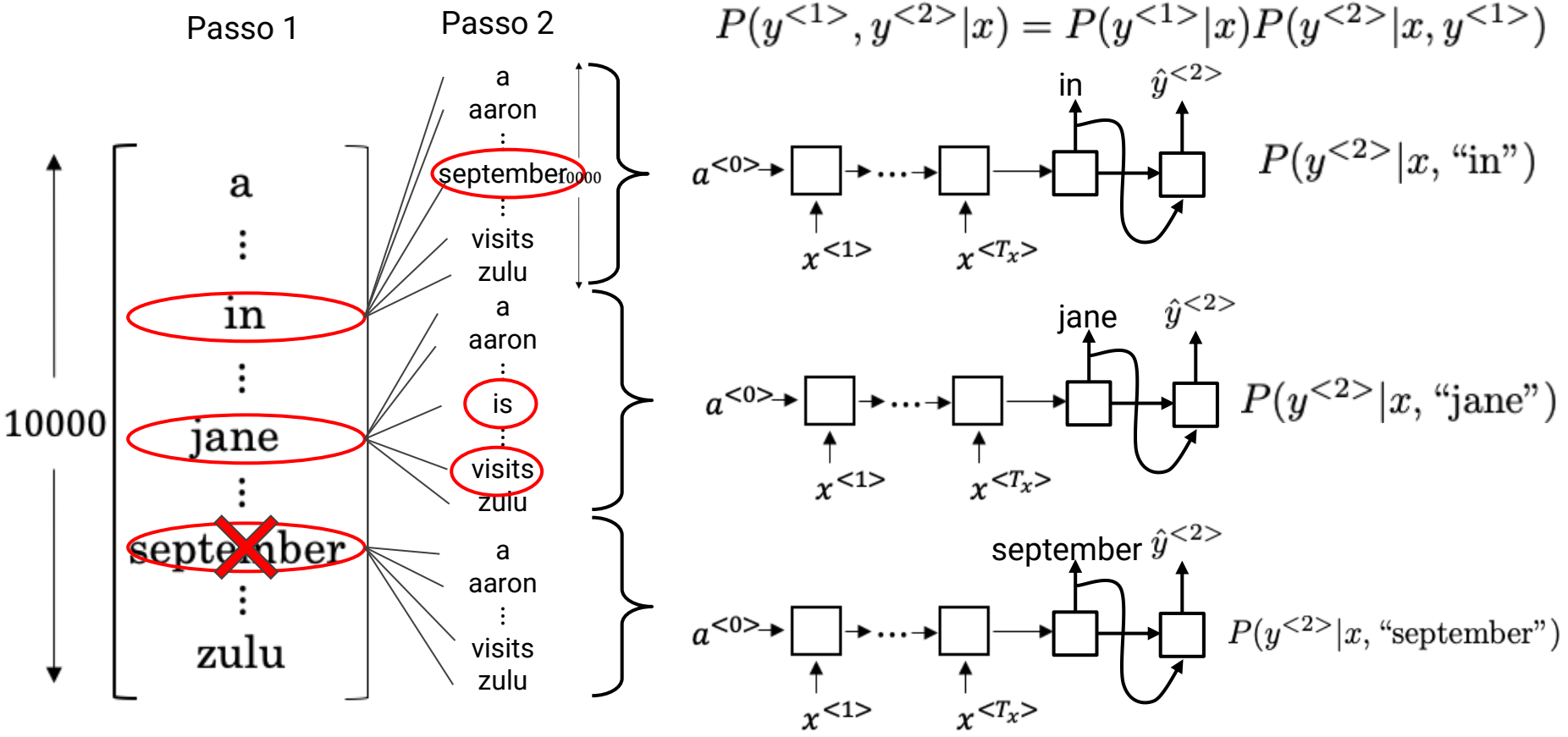
Algoritmo Beam Search (B=3)



Algoritmo Beam Search (B=3)



Algoritmo Beam Search (B=3)



- Para cada opção armazenada no passo 1, devemos considerar todas as possibilidades para a 2a palavra
- Para isso, é preciso fixar a 1a palavra e analisar a distribuição para a 2a
  Ou seja, deve-se instanciar B cópias da rede
- No passo 2, consideramos B|V| possibilidades (neste caso, 30k)
- Armazenamos apenas as B opções mais prováveis

Algoritmo Beam Search (B=3)

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

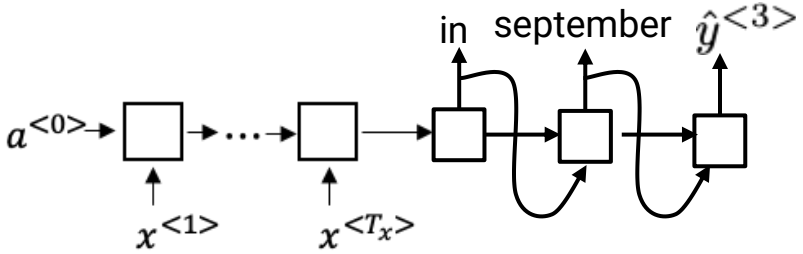
(fragmentos)

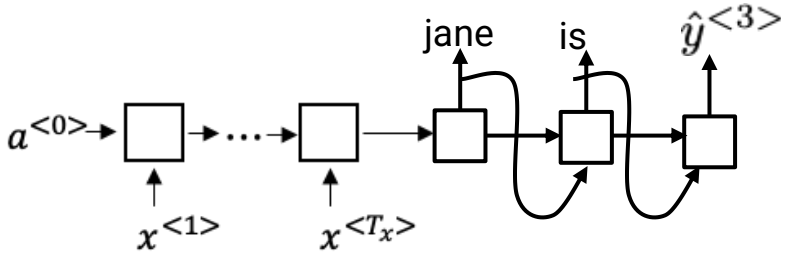
in september

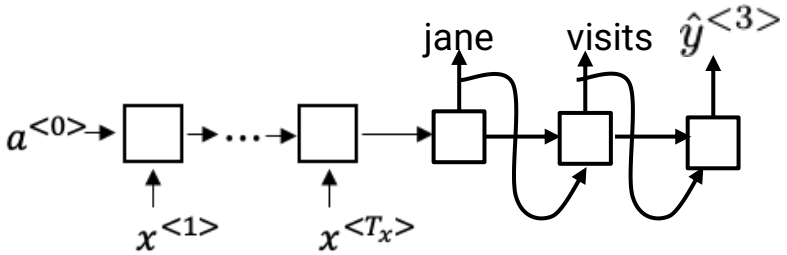
jane is

jane visits

a
aaron
⋮
jane
⋮
zulu







jane visits africa in september. <EOS>

Note que B = 1 é busca gulosa.

- Espera-se que ao final deste processo, uma sentença parecida com *jane visits africa in september* <EOS>
- Com B=1, beam search se degenera para a busca gulosa
- Considerando múltiplas possibilidades B=3, 10, etc beam search retorna resultados muito melhores que busca gulosa
- Existem alguns truques para tornar o beam search ainda melhor

Normalização do comprimento

Queremos maximizar a função de verossimilhança:

$$P(y^{<1>}, \ldots, y^{<T_y>}|x) = P(y^{<1>}|x)P(y^{<2>}|x, y^{<1>}) \cdots P(y^{<T_y>}|x, y^{<1>}, \ldots, y^{<T_{y-1}>})$$

Reescrevendo com produtório:

$$\arg\max_y \prod_{t=1}^{T_y} P(y^{<t>}|x, y^{<1>}, \ldots, y^{<t-1>})$$

Na prática, maximizamos o log da verossimilhança. É um problema equivalente, pois o logaritmo é monotonicamente crescente:

$$\arg\max_y \sum_{t=1}^{T_y} \log P(y^{<t>}|x, y^{<1>}, \ldots, y^{<t-1>})$$

Normalização do comprimento

$$\arg\max_{y} \sum_{t=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \ldots, y^{<t-1>})$$

Problema: em ambos os casos, sequências y maiores são necessariamente menos prováveis que suas subsequências.

Solução: normalizar pelo tamanho da sequência elevado a fator de desconto $0<\alpha<1$.

$$\arg\max_{y} \frac{1}{T_y^{\alpha}} \sum_{t=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \ldots, y^{<t-1>})$$

- Probabilidades no produtório em geral são números muito pequenos (~0)
- Além de ser mais fácil maximizar soma (ao invés do produto), o log evita problemas numéricos
- Ainda assim, mais termos na sequência reduzem a probabilidade
- Normalização do comprimento é uma pequena mudança no beam search que permite obter resultados muito melhores
- Se $\alpha = 1$, então estaríamos normalizando pelo comprimento.
  Se $\alpha = 0$, então não há normalização.
  Na prática, usa-se uma abordagem mais suave (e.g., $\alpha = 0.7$)
- Não há uma justificativa teórica para $\alpha < 1$, na prática são testados valores diferentes para otimizar este novo hiperparâmetro.

Como usar o beam search com normalização de comprimento?

- Conforme beam search é executado, são geradas sequências com T_y=1,2,3,... até um comprimento máximo.
- Quando B=3, são armazenadas as melhores sentenças para cada um destes comprimentos T_y=1,2,3,...
- Então, calcula-se um escore para cada sentença usando a função objetivo do slide 30
- Finalmente, retorna-se a sequência mais provável como saída (tradução)

Como escolher o beam width B?

B grande: resultados melhores, mais lento
B pequeno: resultados piores, mais rápido

Escolhas comuns:

- Produção: 10
- Produção, incomum: 100
- Artigos: 1000 até 3000

Efeito "diminishing returns": ganho é grande no início, mas diminui para *B* grande.

Ao contrário de algoritmos de busca exatos como BFS (busca em largura) ou DFS (busca em profundidade), Beam Search executa mais rápido, mas não garante encontrar o máximo exato para arg $\max_y P(y|x)$.

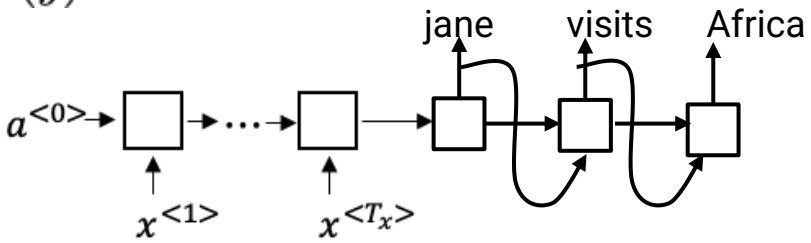
Análise de erro do beam search

Exemplo

Entrada: Jane visite l'Afrique en septembre.

Humano: Jane visits Africa in September. (y^*)

Algoritmo: Jane visited Africa last September. $(\hat{y})$



A predição tem 2 fontes de erro:

- Modelo RNN
- Beam search

Como usar as saídas anteriores para avaliar se o beam search está funcionando bem (mudar escolha de B)? Dois casos possíveis:

$$P(y^*|x) > P(\hat{y}|x)$$

$$P(y^*|x) \leq P(\hat{y}|x)$$

- Beam search não garante retornar o melhor resultado
- Análise de erro serve para identificar se o problema é RNN ou beam search
- Opções que podem ajudar, mas nunca prejudicam o desempenho (acurácia):
  - Conseguir mais dados de treinamento
  - Aumentar o beam width
- As duas opções podem ser muito custosas
- Como decidir se vale a pena aumentar o beam width?
- Usando as duas probabilidades no slide anterior, é possível atribuir um "culpado" a uma tradução imperfeita

Análise de erro do beam search

Humano: Jane visits Africa in September. (y^*)

Algoritmo: Jane visited Africa last September. $(\hat{y})$

Caso 1: $P(y^*|x) > P(\hat{y}|x)$

Beam search escolheu $\hat{y}$. Mas $y^\star$ obtém maior $P(y|x)$.

Conclusão: Beam search é o culpado.

Caso 2: $P(y^*|x) \leq P(\hat{y}|x)$

y^* é uma tradução melhor que $\hat{y}$. Mas o RNN previu $P(y^*|x) \leq P(\hat{y}|x)$.

Conclusão: Modelo RNN é o culpado.

Obs: No caso com normalização de comprimento, ao invés de avaliar estas probabilidades, deve-se avaliar a função objetivo mostrada no slide 3.

Processo de análise de erro

Human	Algorithm	$P(y^*\|x)$	$P(\hat{y}\|x)$	At fault?
Jane visits Africa in September.	Jane visited Africa last September.	2×10^{-10}	1×10^{-10}	B
		...	...	R
		...	...	B
.	.			.
.	.			.
.	.			.

(Development set)

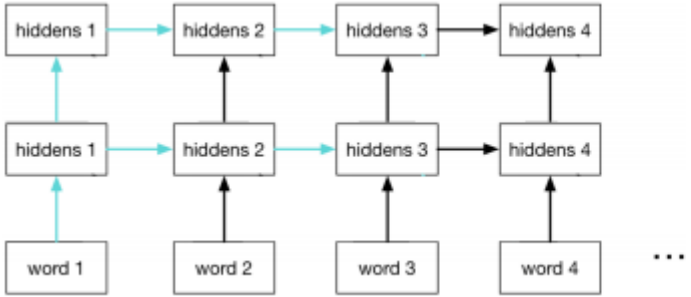
Calculando a média da última coluna é possível descobrir que fração de erros se deve ao beam search vs. modelo RNN.

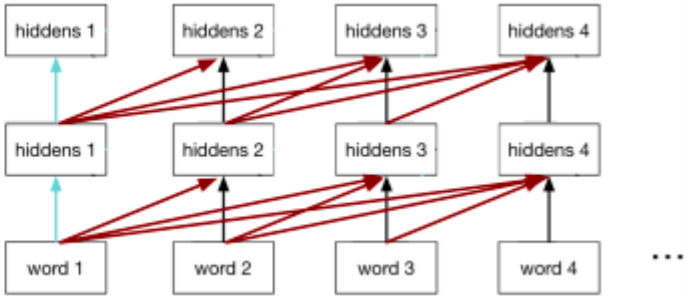
- Análise é feita sobre development set (conjunto de validação)
- Se a maioria dos erros for devido ao beam search, vale a pena aumentar o beam width
- Caso contrário, podemos tentar:
  - Conseguir mais dados de treinamento
  - Adicionar regularização
  - Arquitetura diferente
  - etc

Arquiteturas Transformers

Usando atenção para aumentar paralelismo

- RNNs custam tempo linear no tamanho T_x da sequência x devido às conexões laterais entre estados ocultos.
- Ideia: remover conexões laterais. Vejamos um modelo autoregressivo profundo, onde unidades ocultas dependem dos passos anteriores.

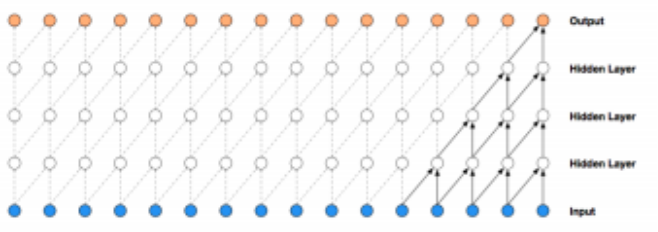


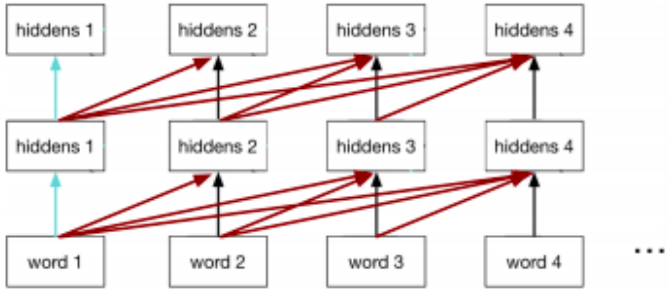


  - Vantagem: número de operações sequenciais independente do tamanho da sequência.

Attention is All You Need

- Vimos modelos de linguagem que usam contextos de tamanho fixo.
- Gostaríamos que nosso modelo tivesse acesso a todo o histórico em cada camada oculta.
- Mas o contexto terá diferente tamanho de entrada a cada passo (+ passos). Max/average pooling não são efetivos nesse caso.
- Podemos usar atenção para agregar informação de contexto prestando atenção a um ou alguns tokens importantes do histórico.





Vaswani, Ashish, et al. ”Attention is all you need.” Advances in Neural Information Processing Systems. 2017.

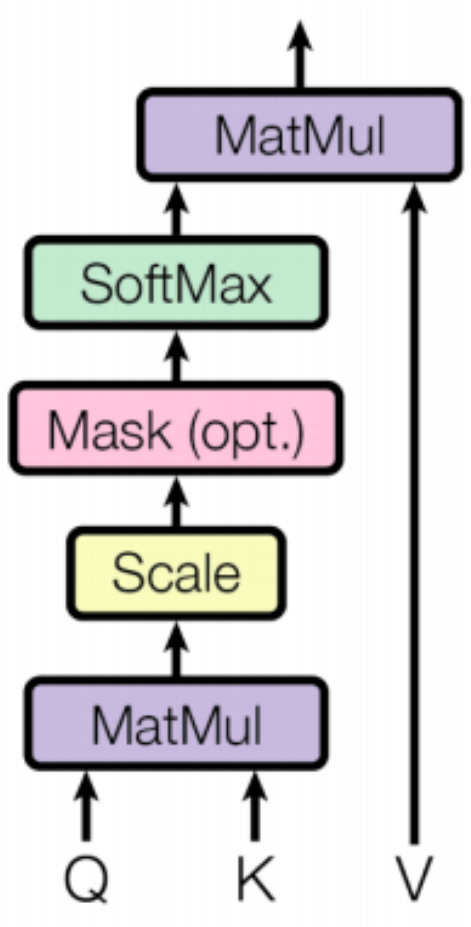
Attention is All You Need

- Vimos que mapeamentos de atenção podem ser descritos como uma função de uma query Q e um conjunto de pares chave-valor (K, V)
- Transformers usam "Scaled Dot-Product Attention" para obter contexto:

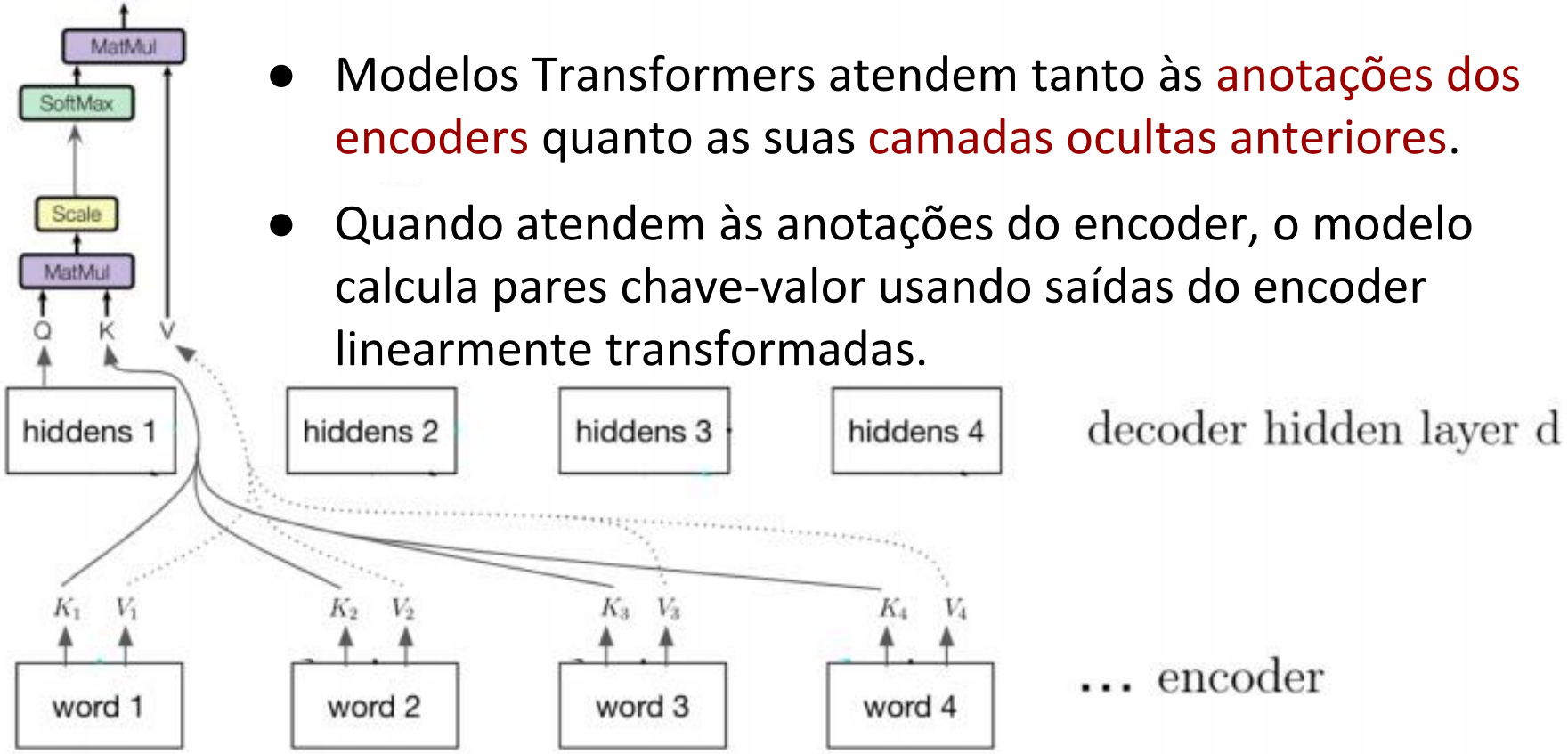
$$\mathbf{c}^{(t)} = \text{attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right)V,$$

Escalado pela raiz da dimensão da chave d_K.

- Conexões inválidas para o futuro são mascaradas para preservar a propriedade autoregressiva.



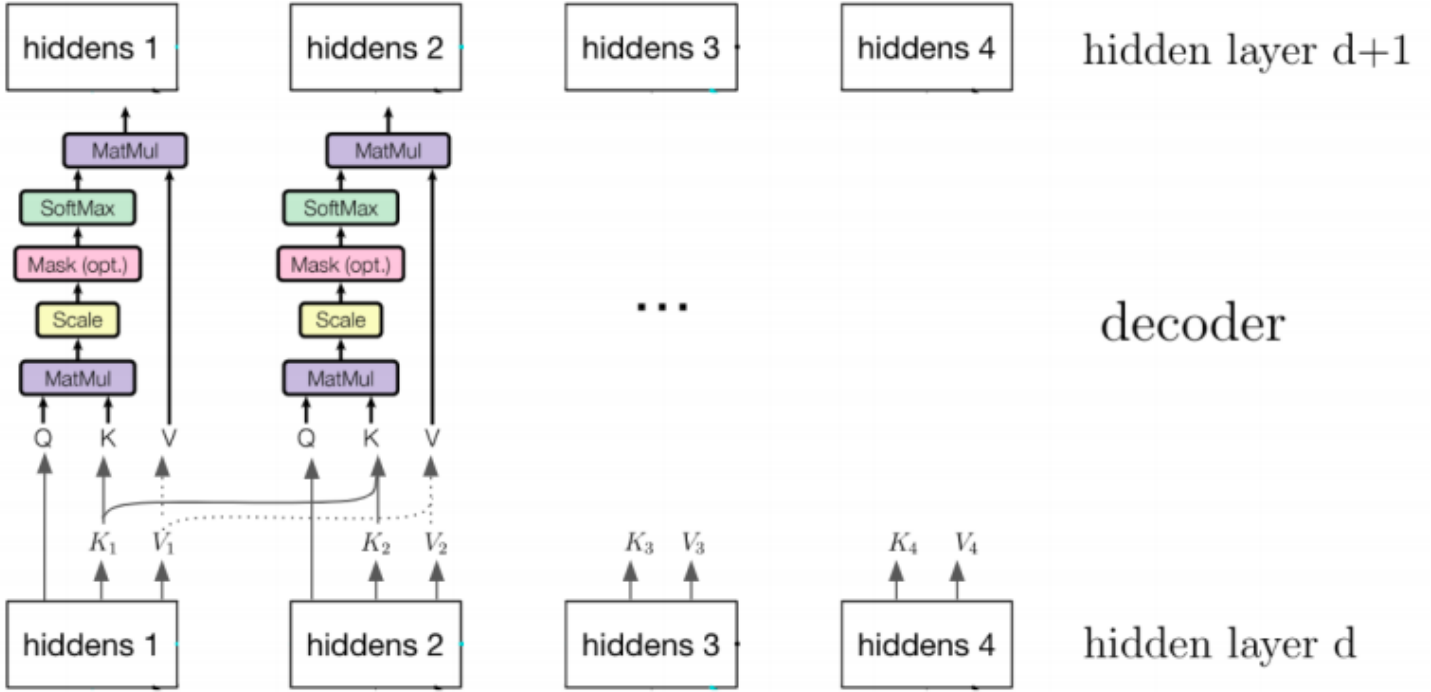
Attention is All You Need



- Modelos Transformers atendem tanto às anotações dos encoders quanto as suas camadas ocultas anteriores.
- Quando atendem às anotações do encoder, o modelo calcula pares chave-valor usando saídas do encoder linearmente transformadas.

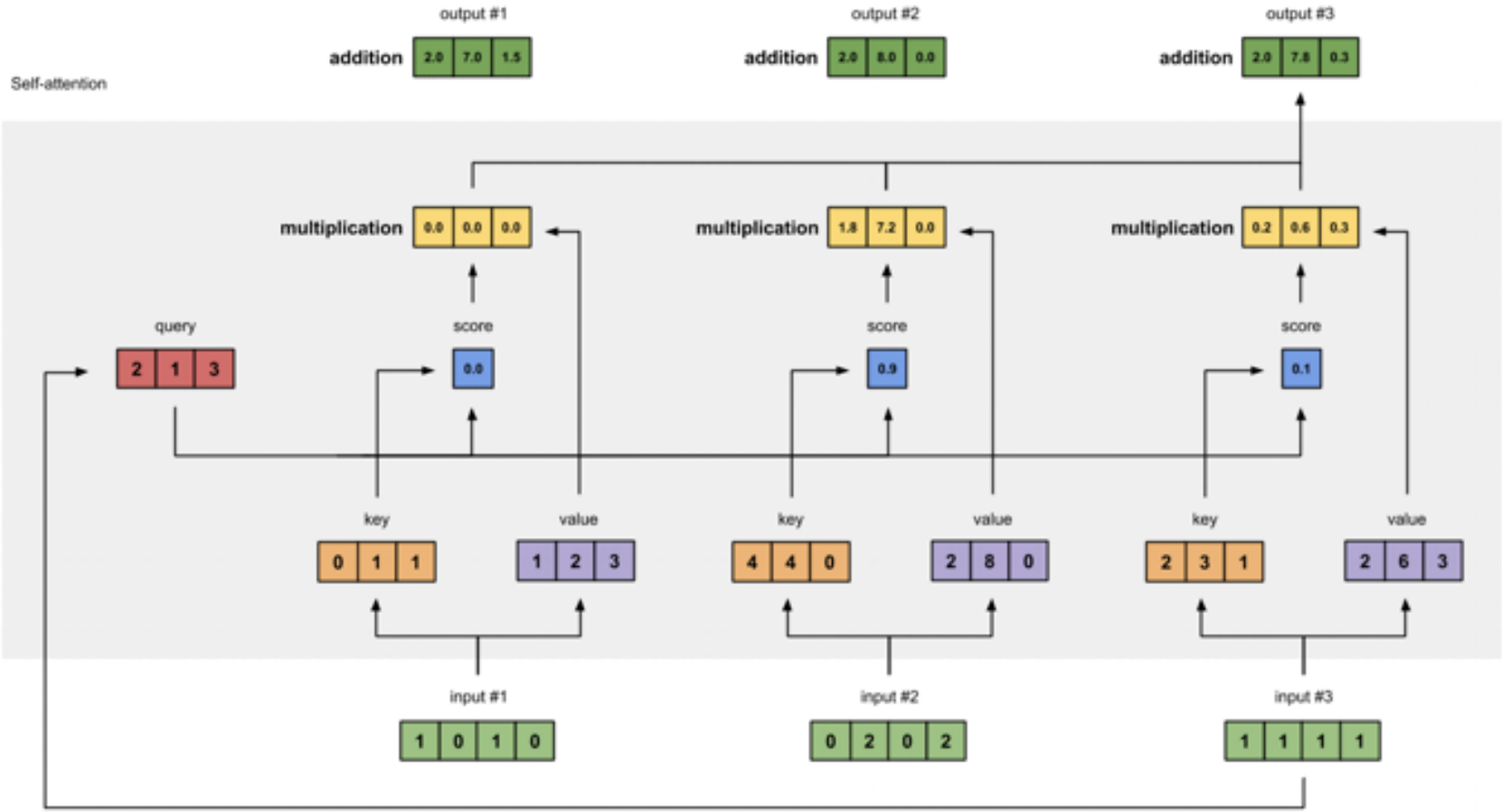
Attention is All You Need

- Modelos Transformers também usam self-attention nas camadas ocultas anteriores.
  - Self-attention: Q, K e V são gerados a partir de transformações lineares do mesmo vetor.
- Quando aplicam atenção às camadas anteriores, a estrutura causal é preservada (mask).



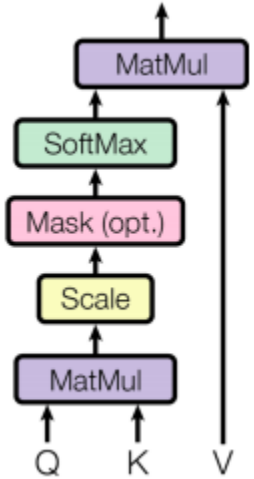
Self-attention layer

source: https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a



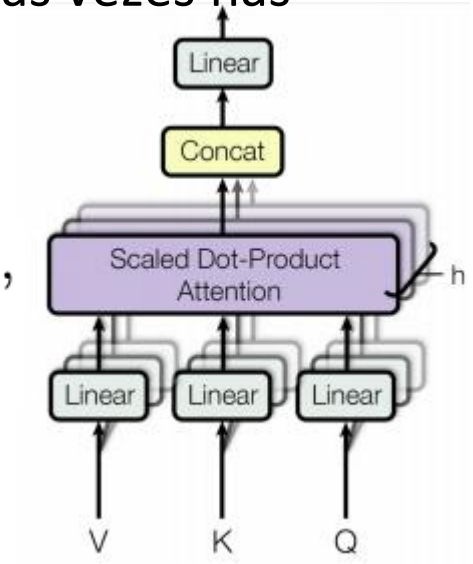
Attention is All You Need

- O Scaled Dot-Product Attention atende a uma ou mais entradas dentre os pares chave-valor.
  - Humanos conseguem atender a várias coisas simultaneamente.
- Ideia: aplicar o Scaled Dot-Product Attention múltiplas vezes nas entradas linearmente transformadas.



$$\text{MultiHead}(Q, K, V) = \text{concat}\,(\mathbf{c}_1, \cdots, \mathbf{c}_h)\, W^O,$$

$$\mathbf{c}_i = \text{attention}(QW_i^Q, KW_i^K, VW_i^V).$$



Encoding Posicional (opcional)

- Diferente de RNN e CNN encoders, as saídas do attention encoder não dependem da ordem das entradas (Por quê?)
- A ordem da sequência carrega informação importante para tarefas de tradução de máquina e modelagem de linguagem.
- Ideia: concatenar informação posicional aos embeddings de entrada.

$$\mathrm{PE}_{pos,2i} = sin(pos/10000^{2i/d_{emb}}),$$
$$\mathrm{PE}_{pos,2i+1} = cos(pos/10000^{2i/d_{emb}}),$$

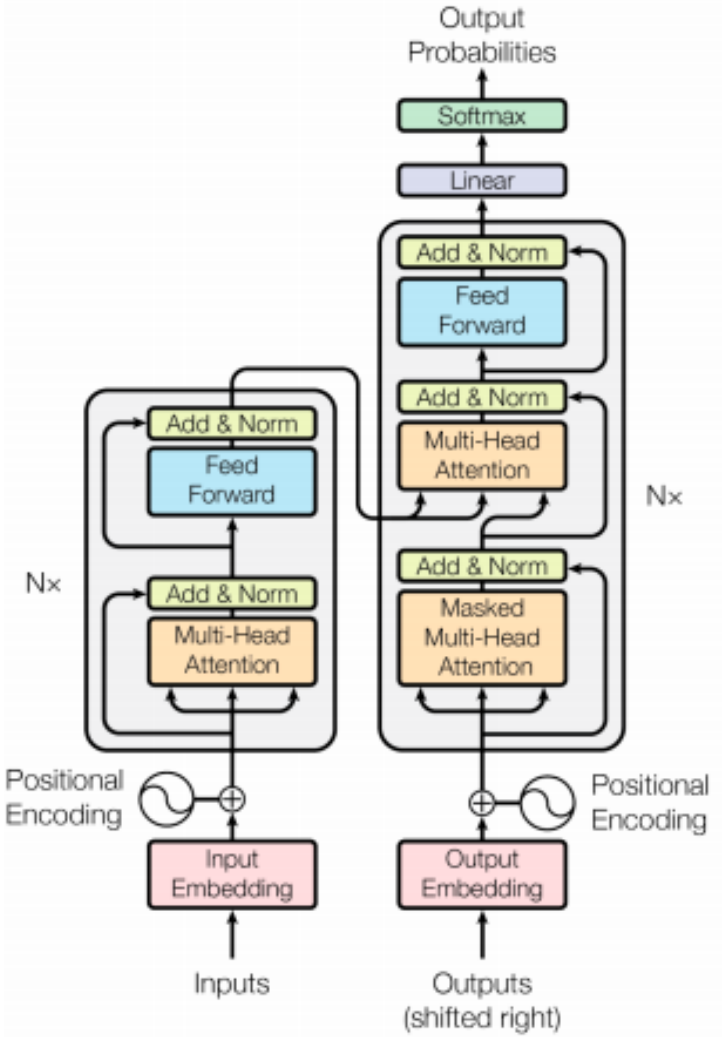
onde *pos* é a posição na sequência e *i* é a coordenada no embedding.

- Os embeddings finais de entrada são a soma do embedding treinável com o encoding posicional.

Intuição: Master Positional Encoding: Part I | by Jonathan Kernes | Towards Data Science

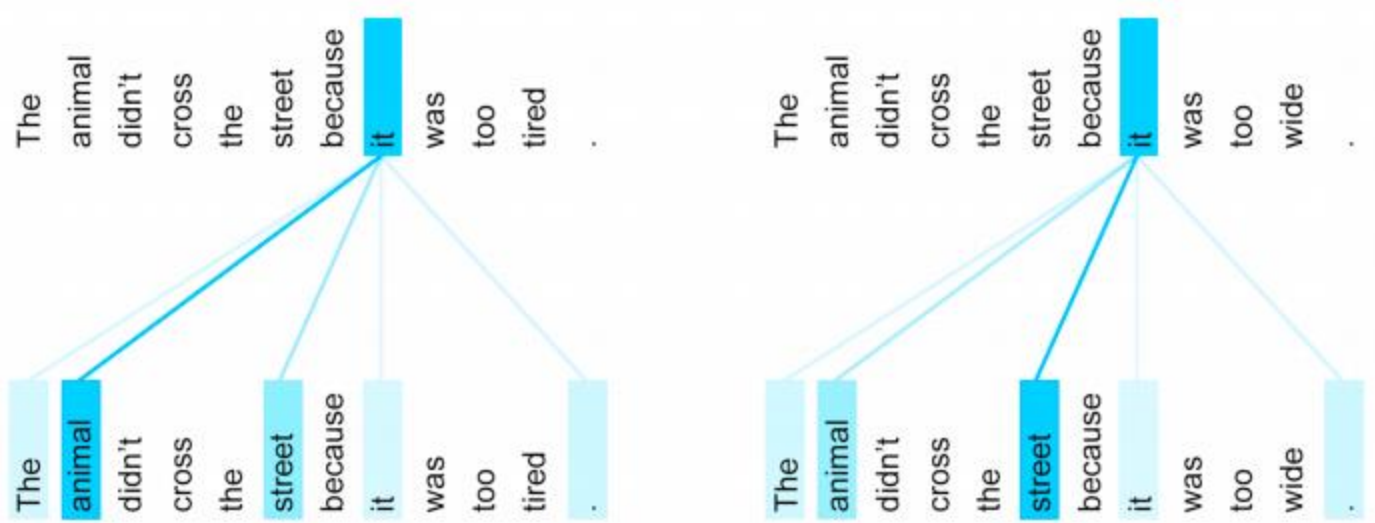
Tradução por máquina via Transformers

- Transformers tem uma arquitetura encoder/decoder similar aos modelos RNN anteriores.
  - Exceto que todas as conexões recorrentes são substituídas por módulos de atenção
- O modelo transformer usa N camadas de self-attention empilhadas
- Skip-connections ajudam a preservar informação posicional e de identidade das sequências de entrada



Tradução por máquina via Transformers

- Camadas de self-attention aprenderam que "it" pode se referir a diferentes entidades em diferentes contextos.



Visualização do 5o. para 6o. Self-attention layer no encoder.

https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

Complexidade Computacional e Paralelismo em Transformers

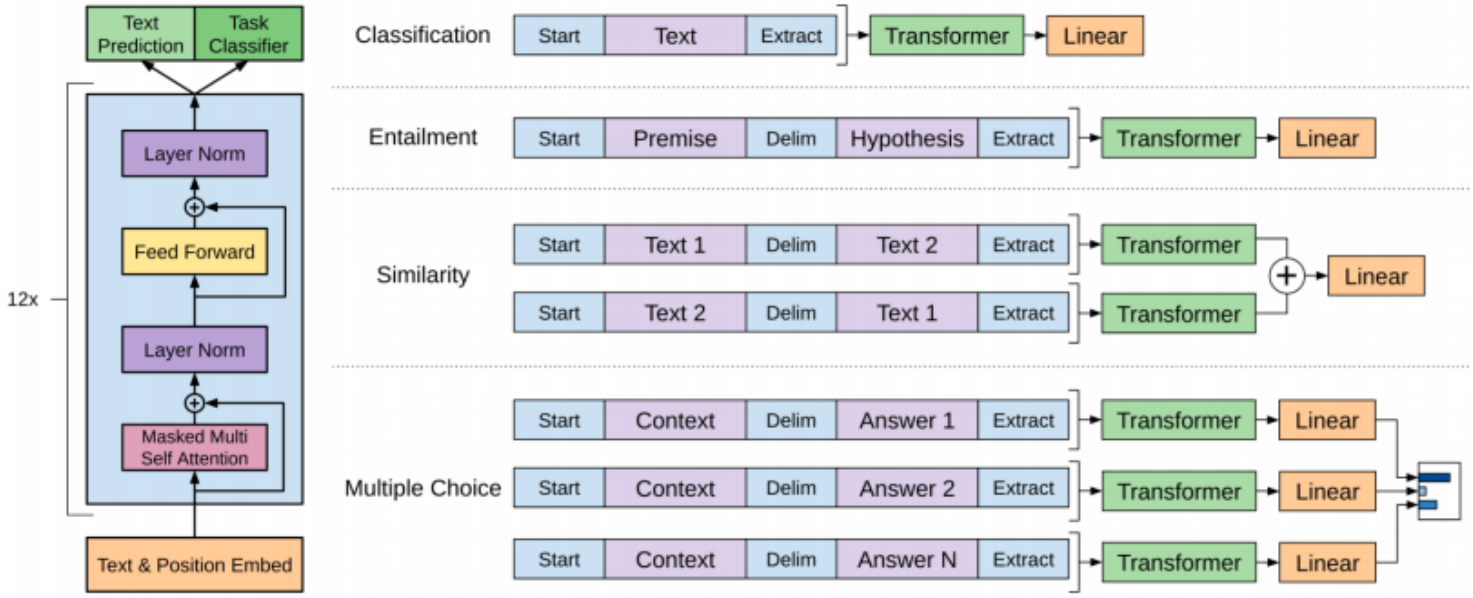
Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

- Normalmente, tamanho da sequência n << d
- Caminhos tem tamanho máximo constante

Pré-treinamento de Transformers para Linguagem

- Similar ao pré-treinamento de modelos de visão computacional no ImageNet, podemos pré-treinar modelo de linguagem p/ tarefas de NLP.
  - Modelo pré-treinado é então fine-tuned para tarefa desejada



Radford, Alec, et al. “Improving Language Understanding by Generative Pre-Training.” 2018

Pré-treinamento de Transformers para Linguagem

- Aumentar o tamanho do conjunto de treino e o tamanho do modelo traz melhorias notáveis no modelo de linguagem Transformer. Amostras cherry-picked de Radford et al. 2019

Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

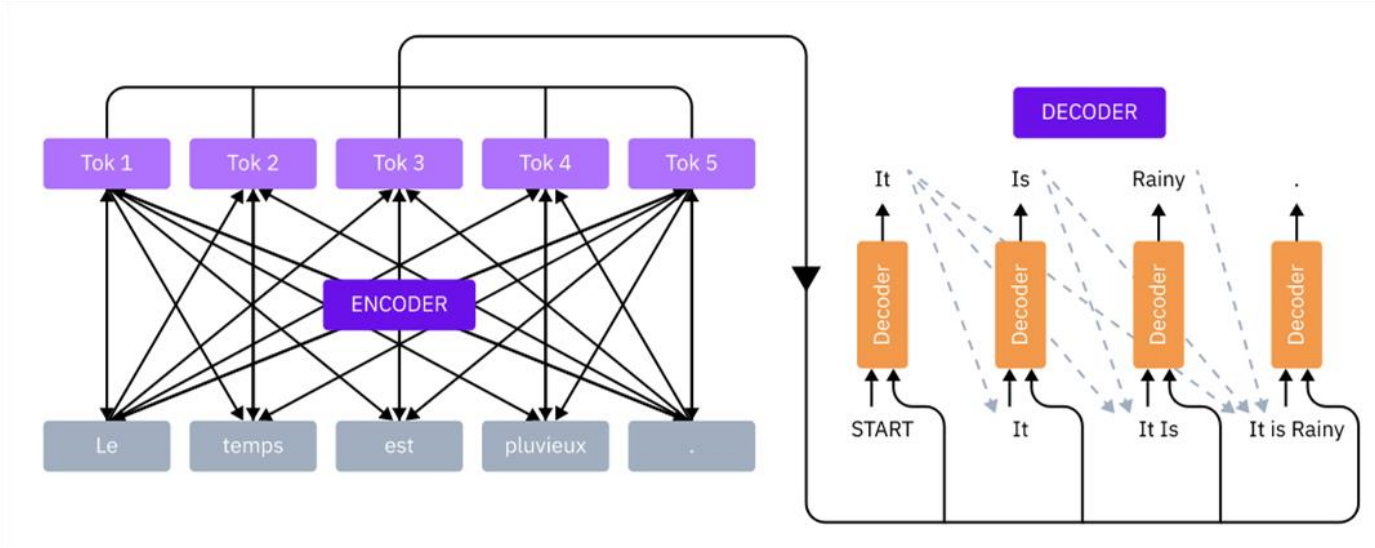
Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.

Exemplos completos em Radford, Alec, et al. "Language Models are Unsupervised Multitask Learners." 2019.

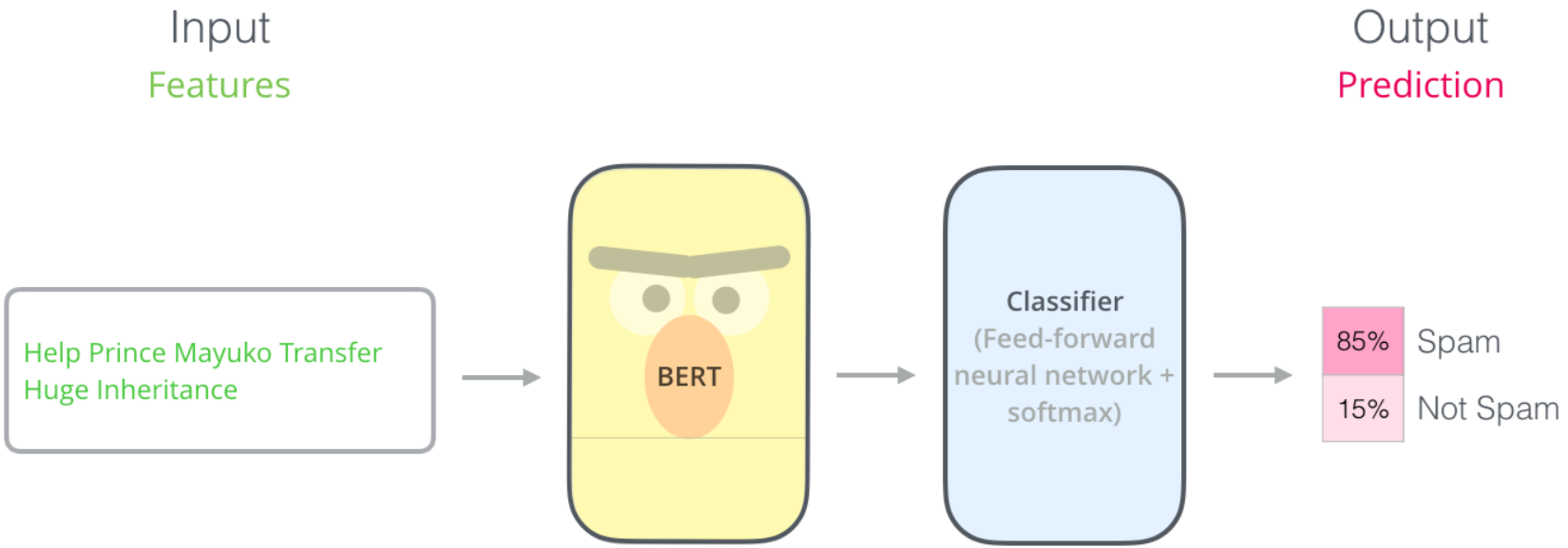
Transformers são lentos para gerar sequências

- No Decoder, não sabemos qual é a saída y_1 até que seja gerada
- Lentidão não acontece no treinamento graças ao *teacher forcing*



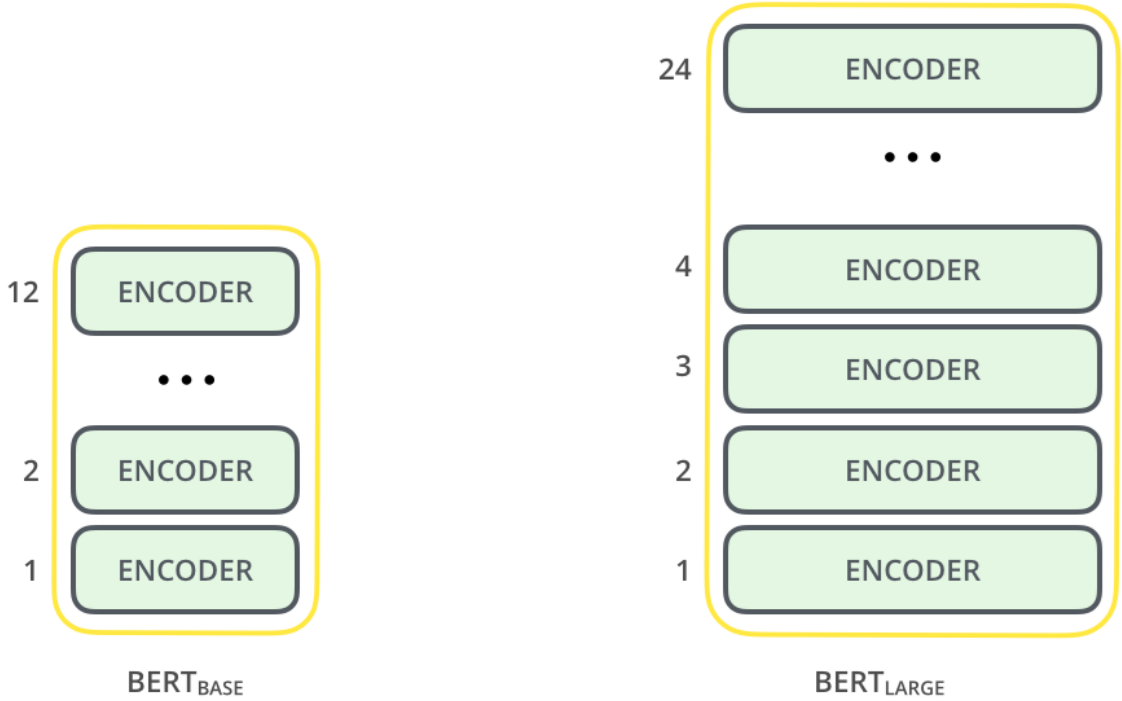
Source: https://scale.com/blog/pytorch-improvements

Sentence Classification using BERT (opcional)

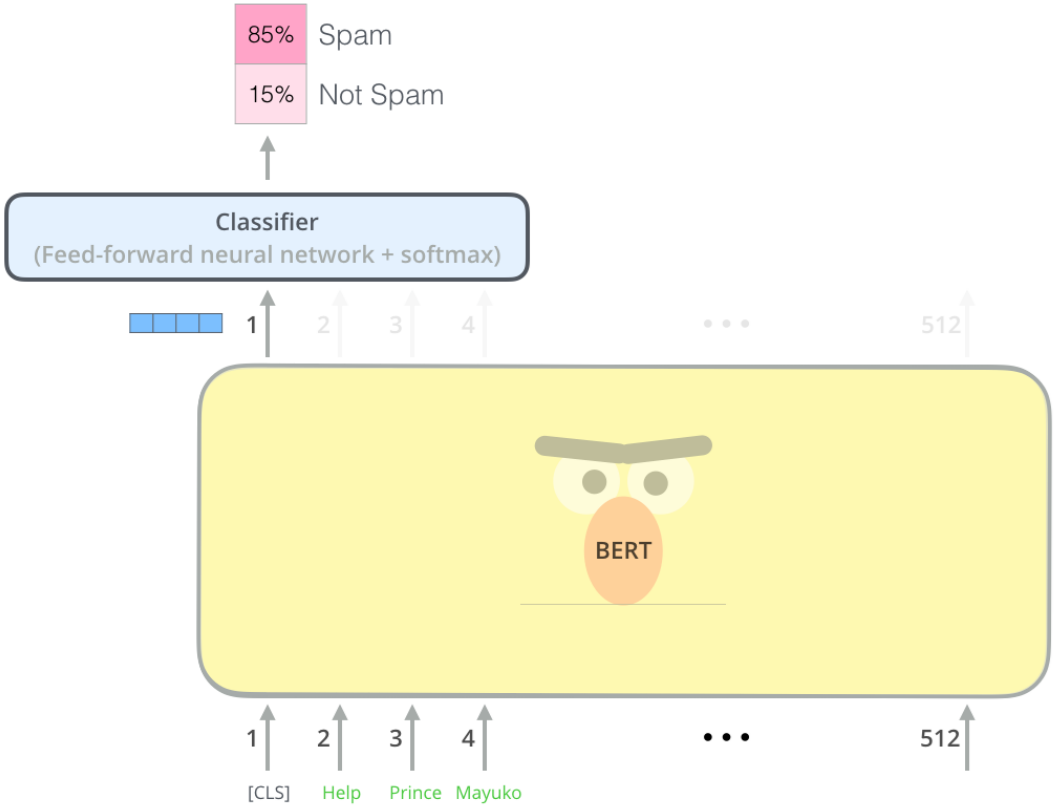


source: http://jalammar.github.io/illustrated-bert/

Sentence Classification using BERT (opcional)



Sentence Classification using BERT (opcional)



Recap do Módulo 5

- Modelos de Linguagem
- Modelos n-grama (limitações: espaço, frases não vistas)
- Representações distribuídas (modelos sem memória)
- Vanilla RNNs (vanishing/exploding gradients)
- GRU e LSTM (gargalo de informação para sequências longas)
- Mecanismo de atenção
- Attention Transformers

Obrigado!

<EOS>