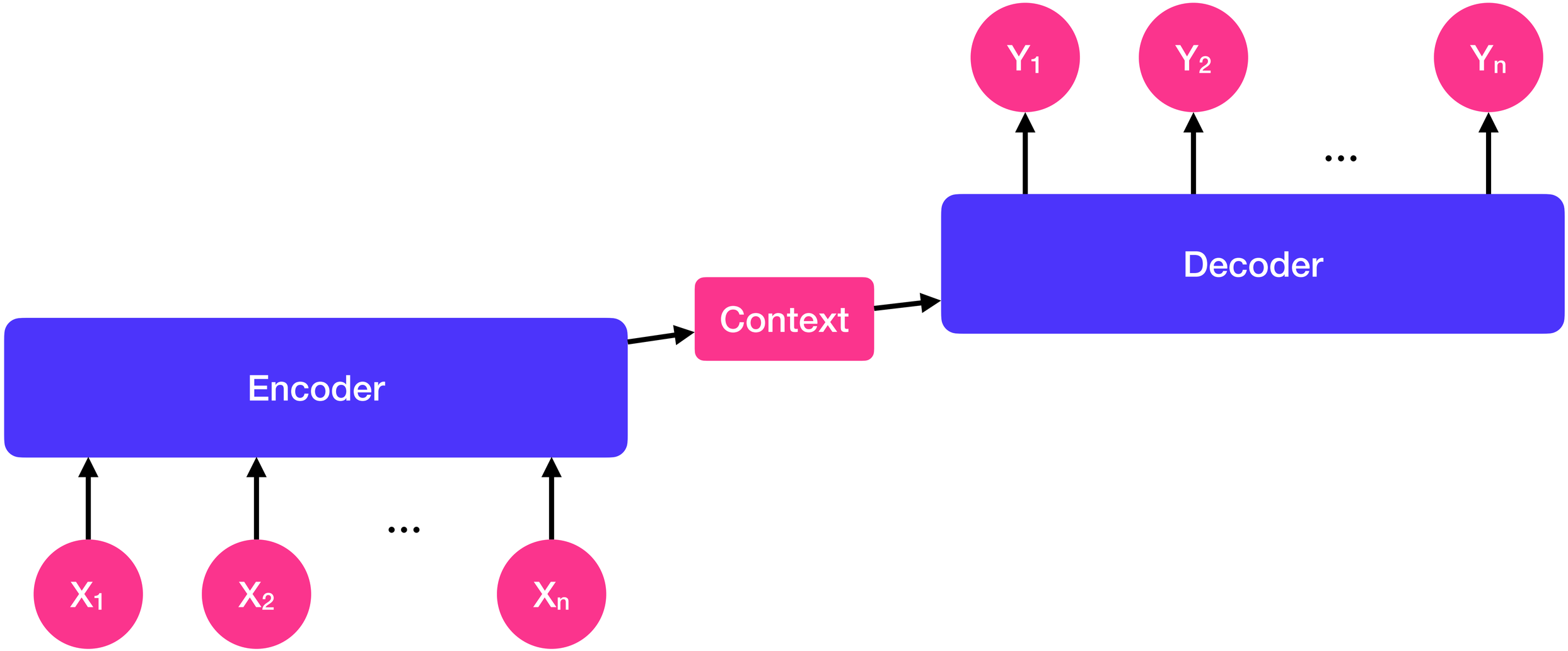
Processamento de Linguagem Natural

Sequence-to-sequence

Yuri Malheiros (yuri@ci.ufpb.br)

Introdução

- Redes sequence-to-sequence ou encoder-decoder são modelos capazes de gerar saídas sequenciais de tamanhos variados
- Elas são usadas em aplicações de tradução, sumarização, responder perguntas, etc.
- A rede encoder recebe a sequência de entrada e cria uma uma representação (contexto) como saída
- A rede decoder recebe o contexto e gera a sequência de saída


Y1
Y2
Yn
...
Decoder
Context
Encoder
...
X1
X2
Xn

Encoder-Decoder com RNNs

- Nós vimos que a probabilidade de uma sequência y pode ser calculada como:

$$p(y) = p(y_1)p(y_2 \mid y_1)P(y_3 \mid y_1 y_2) \ldots P(y_m \mid y_1, \ldots, y_{m-1})$$

- Numa RNN, em um determinado tempo t, passamos $t - 1$ tokens através da rede, sendo a última saída correspondente ao próximo token

Encoder-Decoder com RNNs

- Dado g uma função de ativação, t um instante no tempo, h_{t-1} o hidden state em $t-1$ e f uma função softmax, temos:

$$h_t = g(h_{t-1}, x_t)$$

$$y_t = f(h_t)$$

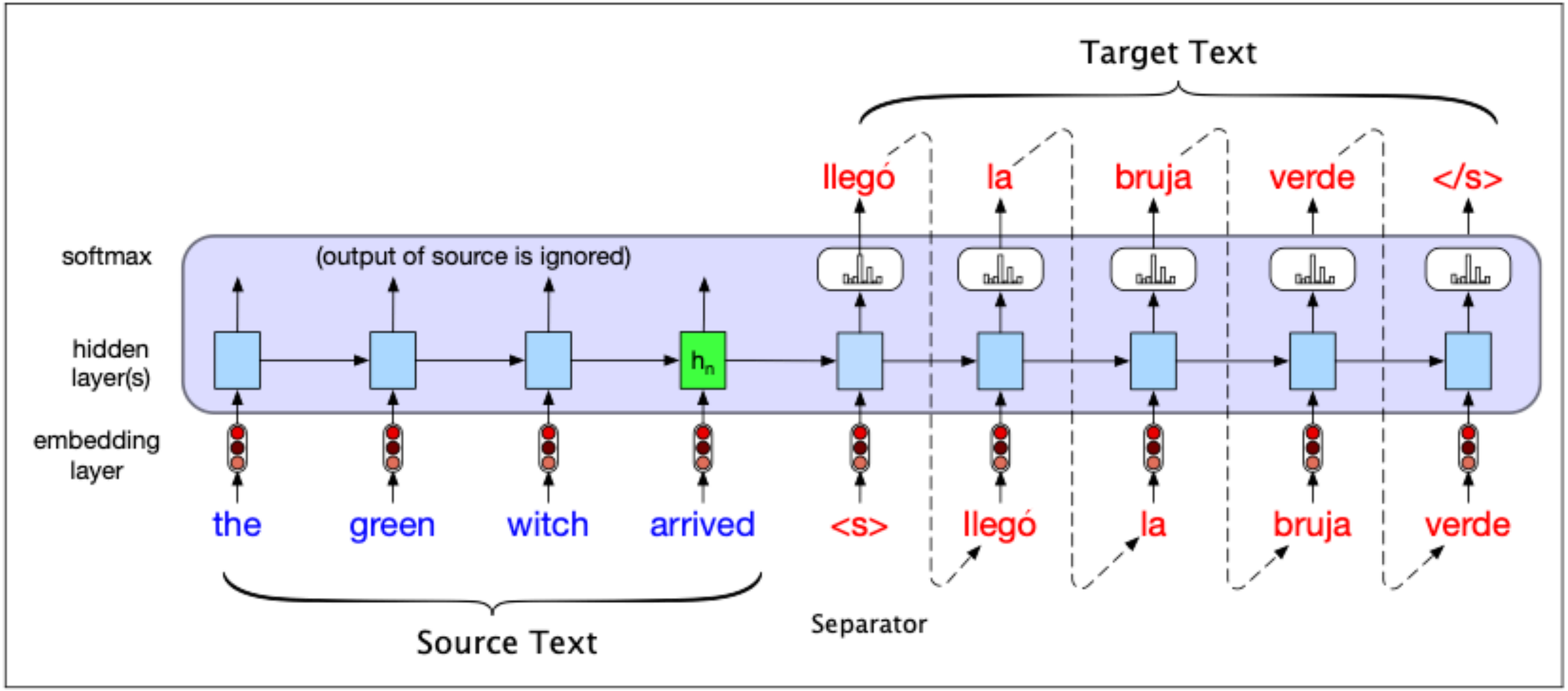
Encoder-Decoder com RNNs

- Para transformar esse modelo em um modelo de tradução, precisamos apenas de uma pequena mudança
- Para cada exemplo de treinamento, vamos adicionar um token especial no fim da entrada, depois concatenar a saída, por fim, adicionamos um token de fim da saída

Encoder-Decoder com RNNs

- Para gerar uma tradução, basta passar o texto de entrada para a rede até chegar no token especial de fim da entrada
- Em seguida, começamos a geração de texto usando a RNN até encontrar o token de fim da saída

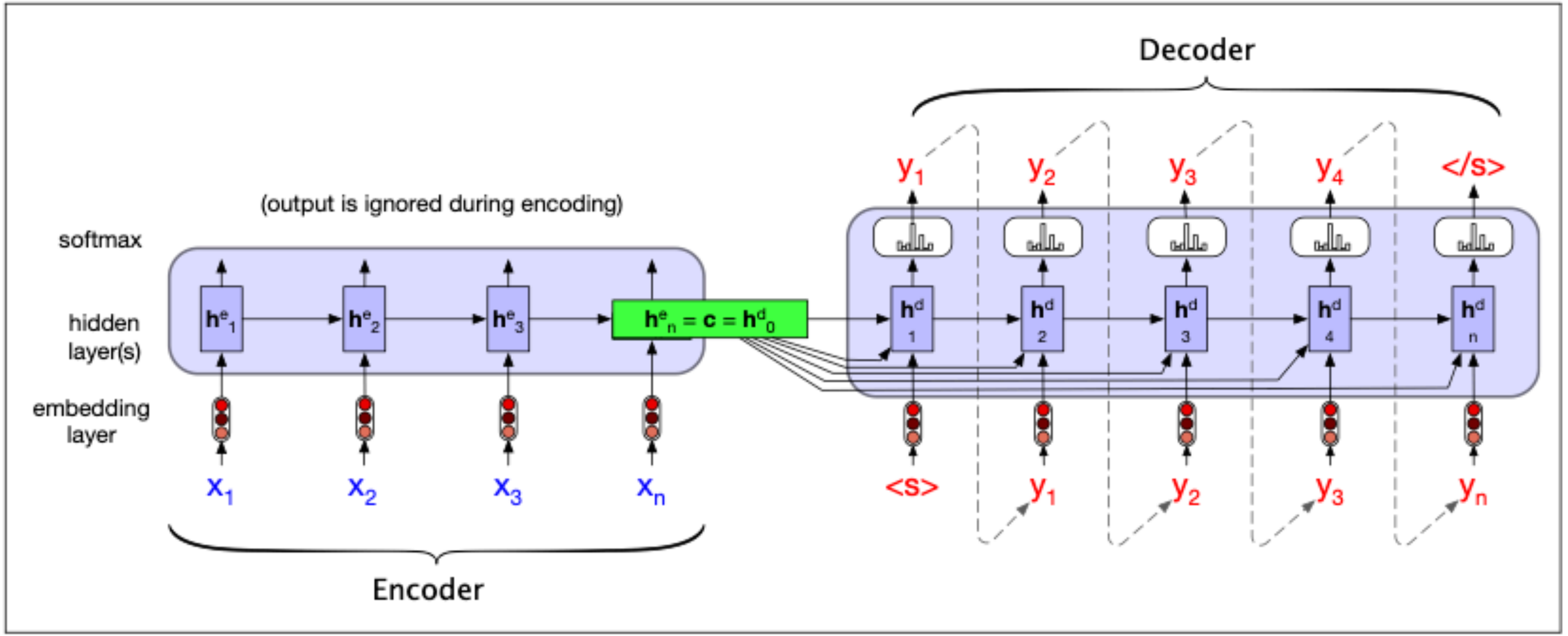
Encoder-Decoder com RNNs



Encoder-Decoder com RNNs

- Um problema dessa abordagem é que o contexto do encoder vai ter pouca influência em itens mais distantes da sequência de saída
- Uma solução para isso é deixar o contexto disponível para todos os passos do decoder

Encoder-Decoder com RNNs



Teacher Forcing

- Durante o treinamento, é comum utilizar a técnica de Teacher Forcing
- No treinamento, vamos tentar prever vários tokens de uma sequência, entretanto, a rede pode errar o token previsto
- No Teacher Forcing, ao tentar prever o próximo token, o treinamento sempre usará os tokens corretos anteriormente, mesmo que a rede tenha previsto um token errado

Encoder-Decoder com RNNs

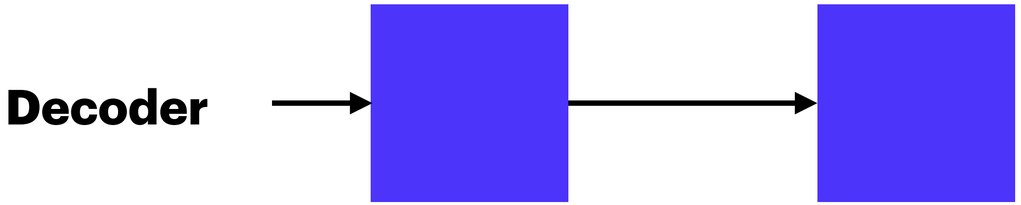
- Implementação...

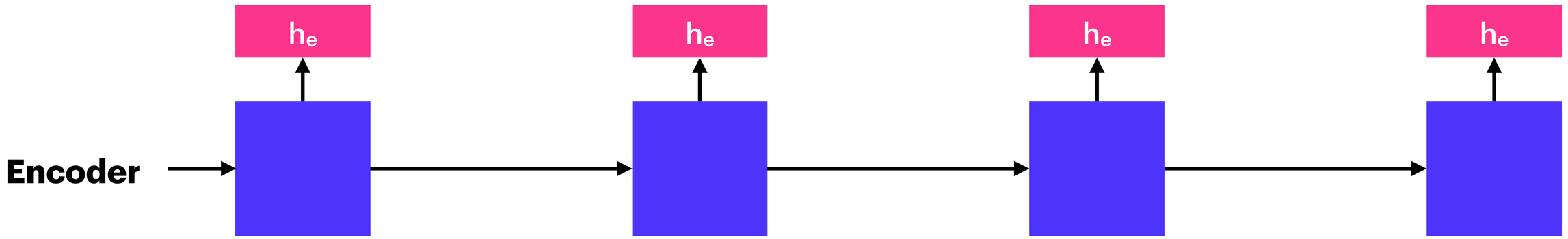
Attention

- O encoder constrói uma representação da entrada (contexto)
- O decoder usa essa representação para gerar a saída
- O contexto é um gargalo, pois ele precisa representar tudo sobre a entrada
- Informações sobre o início da entrada podem não ser bem representada
  - Principalmente para sequências longas

Attention

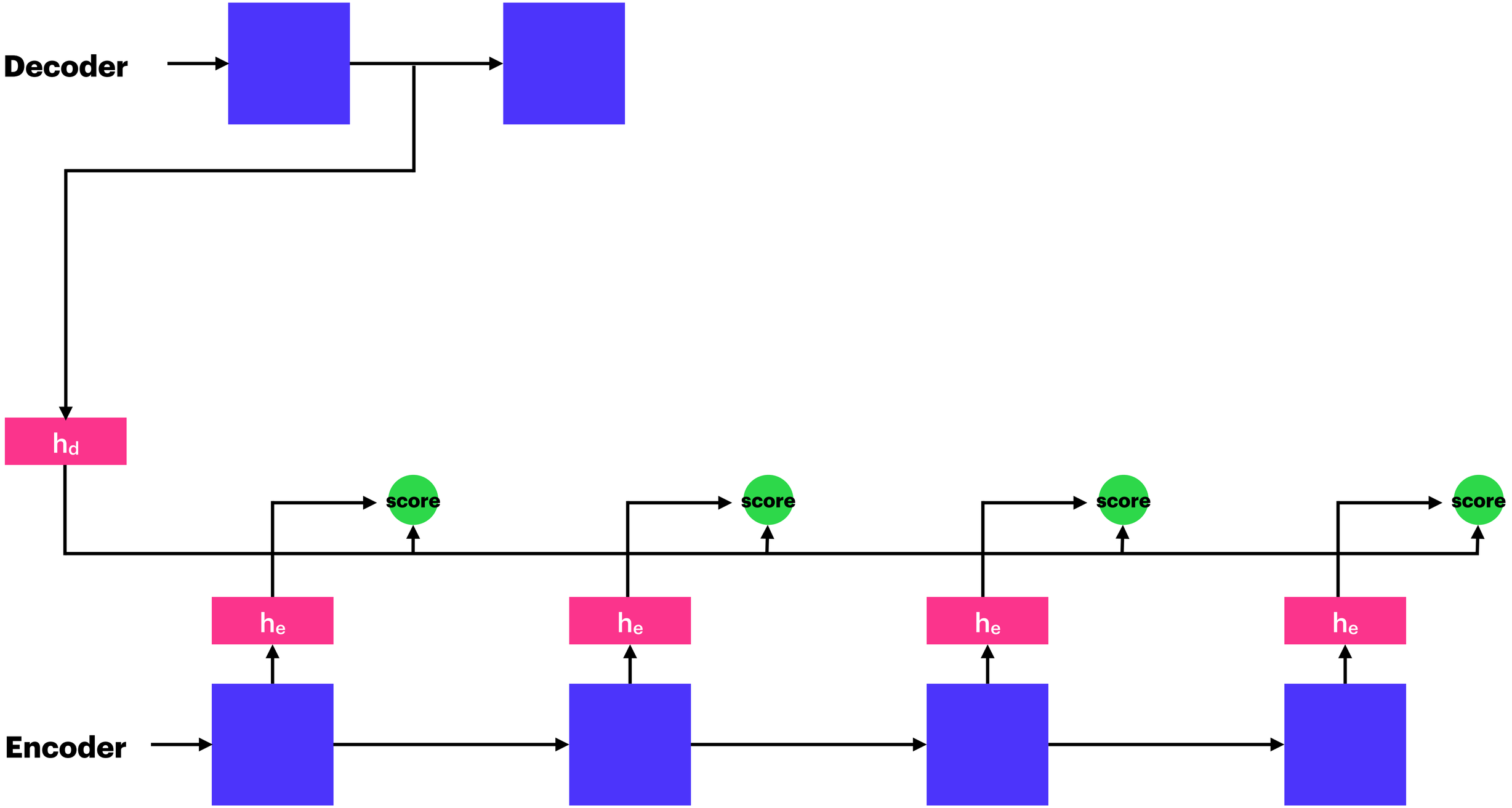
- O mecanismo de atenção (attention mechanism) é uma solução para esse gargalo
- O decoder pode usar a informação de todos os hidden states do encoder
- Como a quantidade de hidden states varia de acordo com a entrada, não podemos usar todos diretamente como contexto do decoder
- A ideia do attention é criar um vetor único de tamanho fixo para cada etapa do decoder para ser o contexto
  - Esse vetor é uma soma ponderada de todos os hidden states
  - O peso determina em que hidden states o decoder deve focar (prestar atenção)


Decoder


Encoder
he
he
he
he

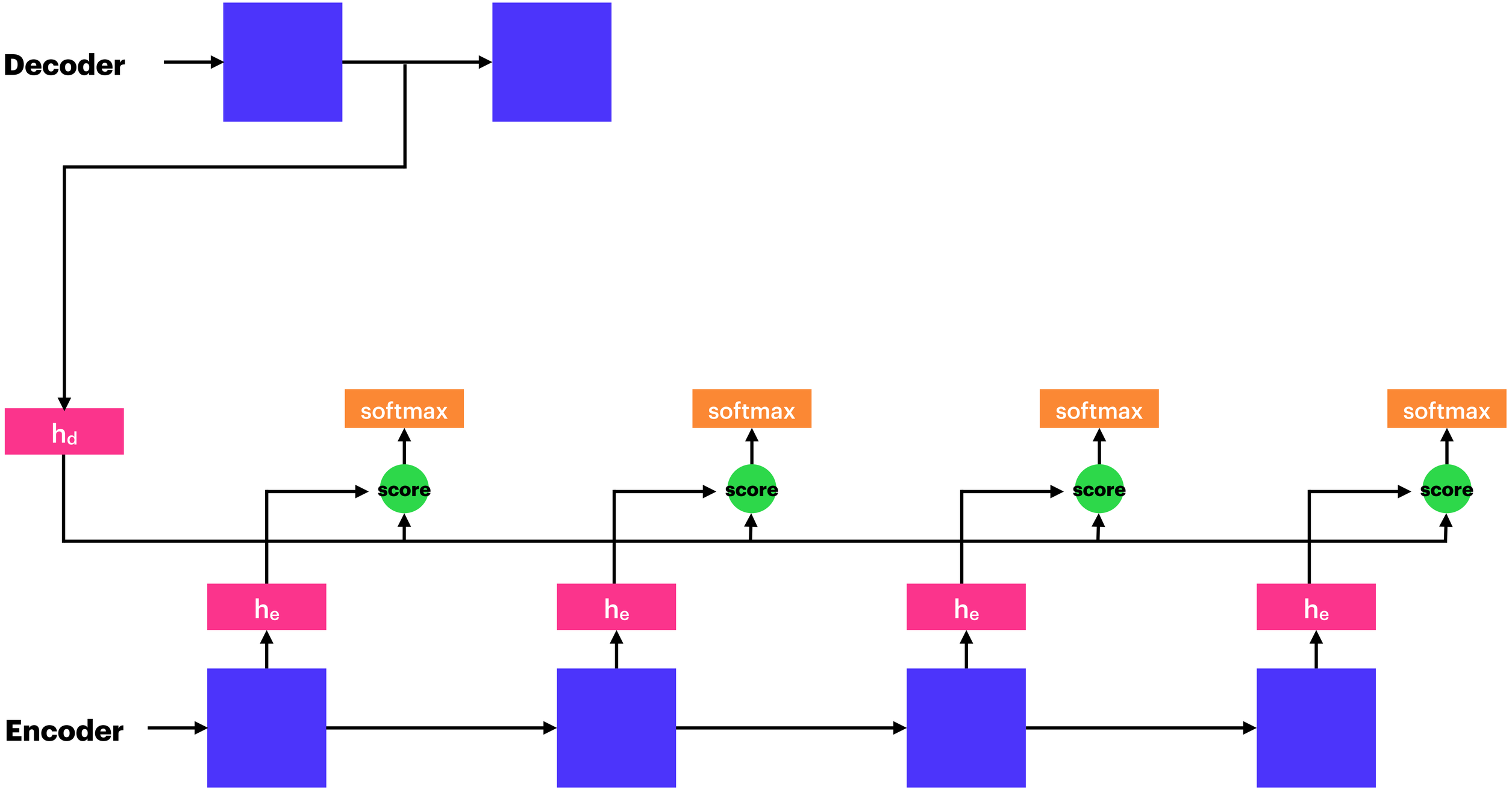
Attention

- O primeiro passo é calcular a relevância de um hidden state do encoder para um determinado passo do decoder
- Uma das maneiras de fazer isso é através do produto escalar entre os hidden states do encoder e o hidden state que um passo do decoder está recebendo
  - Temos um número de score para cada hidden state


Decoder
h_d
score
score
score
score
h_e
h_e
h_e
h_e
Encoder

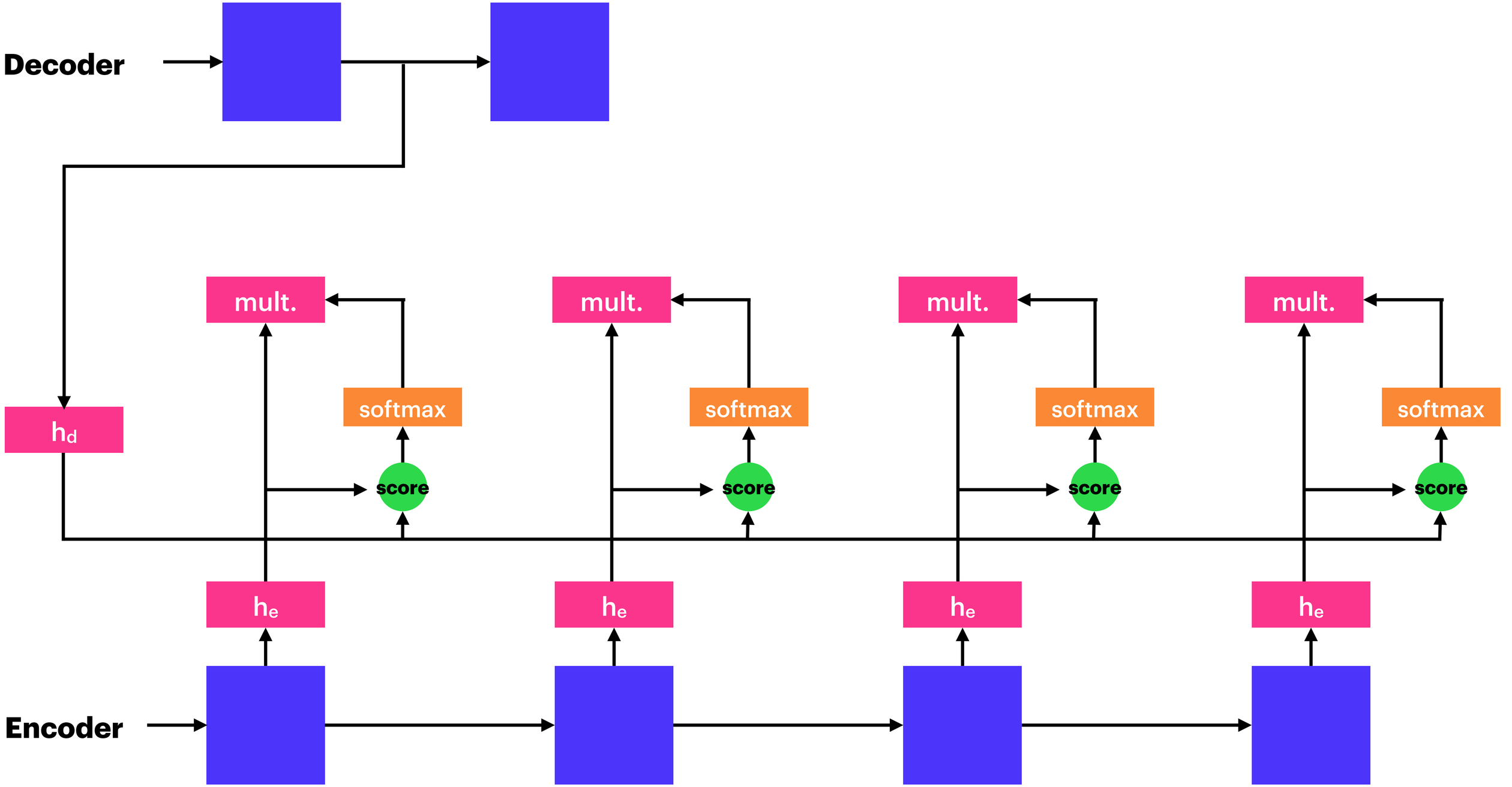
Attention

- Os scores vão gerar um vetor que será normalizado por uma função softmax
- Note que o softmax é um só que recebe todos os scores
  - O tamanho do vetor resultante do softmax é igual a quantidade de entradas recebidas pelo encoder


Decoder
h_d
softmax
softmax
softmax
softmax
score
score
score
score
h_e
h_e
h_e
h_e
Encoder

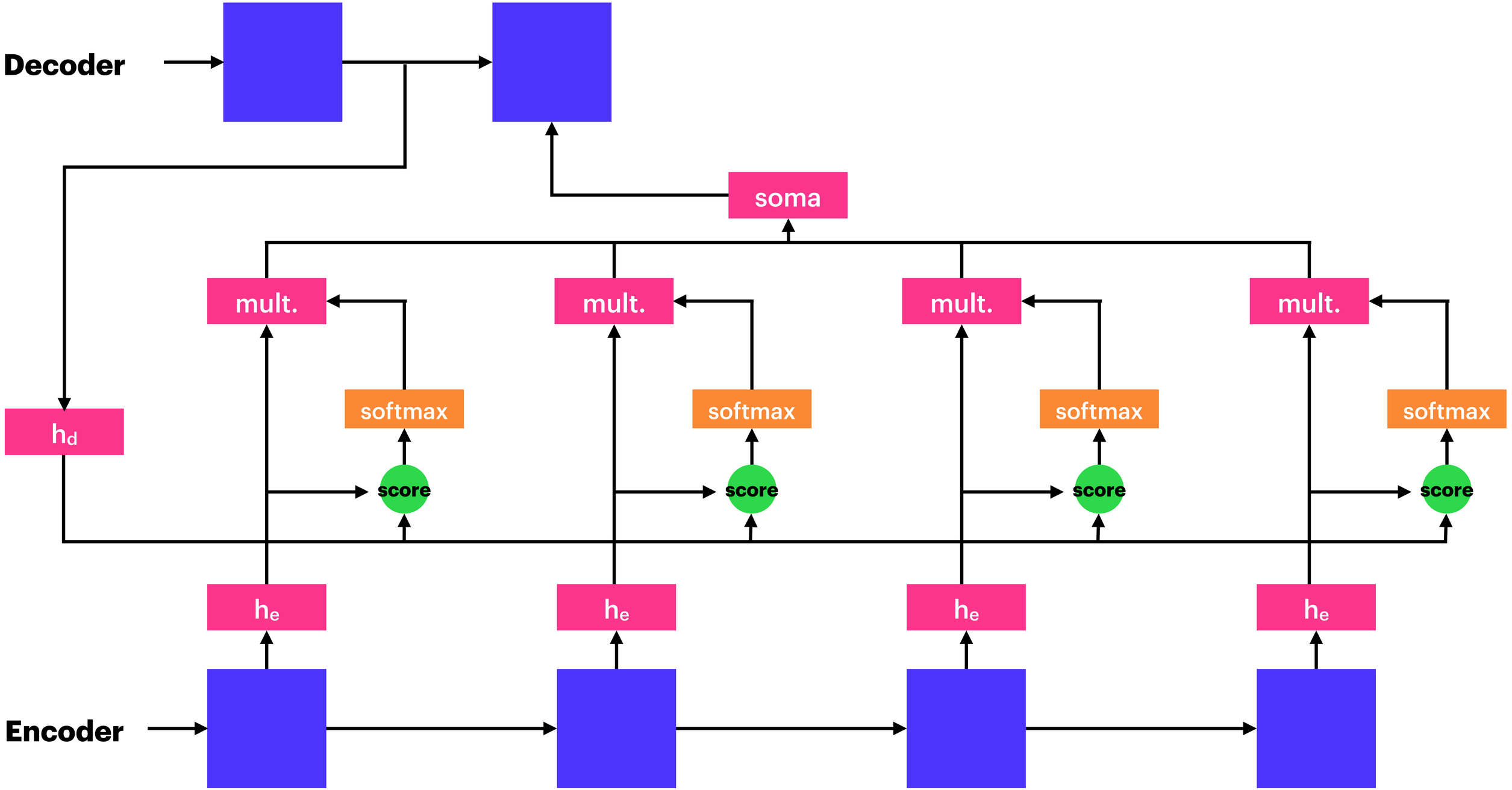
Attention

- Cada hidden state do encoder é multiplicado pelo resultado do softmax
- Assim, ponderamos cada vetor do hidden state pelo o seu score


Decoder
mult.
mult.
mult.
mult.
softmax
softmax
softmax
softmax
h_d
score
score
score
score
h_e
h_e
h_e
h_e
Encoder

Attention

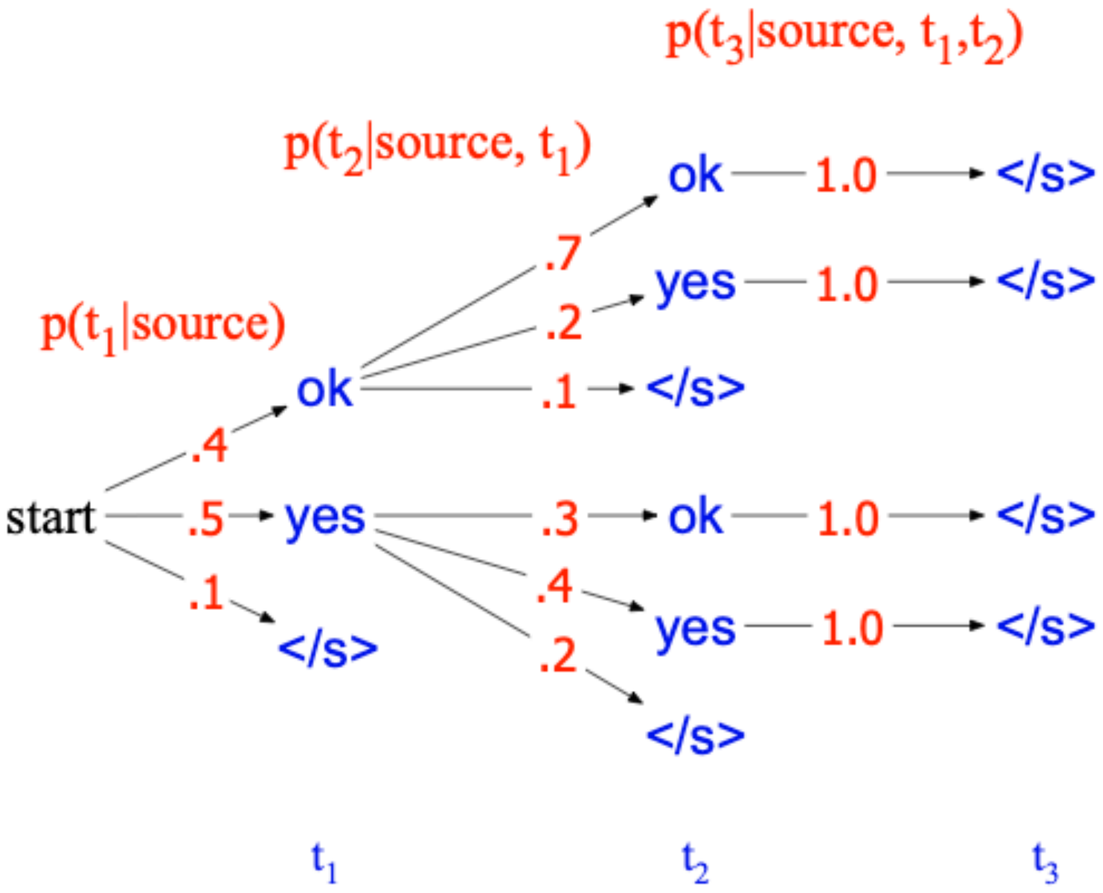
- Por fim, os vetores são somadas gerando apenas um vetor independentemente do tamanho da entrada
  - Todos os hidden states ponderados são somadas resultando em um vetor com tamanho igual ao dos hidden states


Decoder
soma
mult.
mult.
mult.
mult.
softmax
softmax
softmax
softmax
h_d
score
score
score
score
h_e
h_e
h_e
h_e
Encoder

Beam Search

- Na geração de texto e também na tradução nós usamos uma abordagem gulosa para escolher o próximo token
  - Ou pegamos o token com maior probabilidade
  - Ou fizemos amostragem de acordo com a probabilidade
- A abordagem gulosa pode funcionar, mas não é ótima

Beam Search


p(t3|source, t1,t2)
p(t2|source, t1)
p(t1|source)
start
.4
.5
.1
ok
yes
</s>
.7
.2
.1
.3
.4
.2
ok
yes
</s>
1.0
1.0
1.0
</s>
</s>
</s>
t1
t2
t3

Beam Search

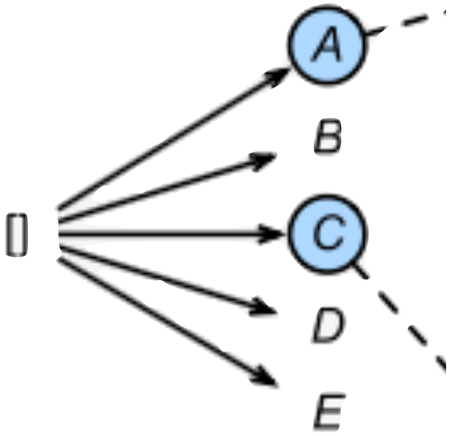
- Para geração de sequências longas é inviável gerar todas as possibilidades
- Por isso, vamos usar o Beam Search
- Nessa técnica, vamos manter **k** tokens em cada passo da geração
- Esses **k** valores são chamados de hipóteses
- As **k** hipóteses são passadas, cada uma, para uma decodificação

Beam Search

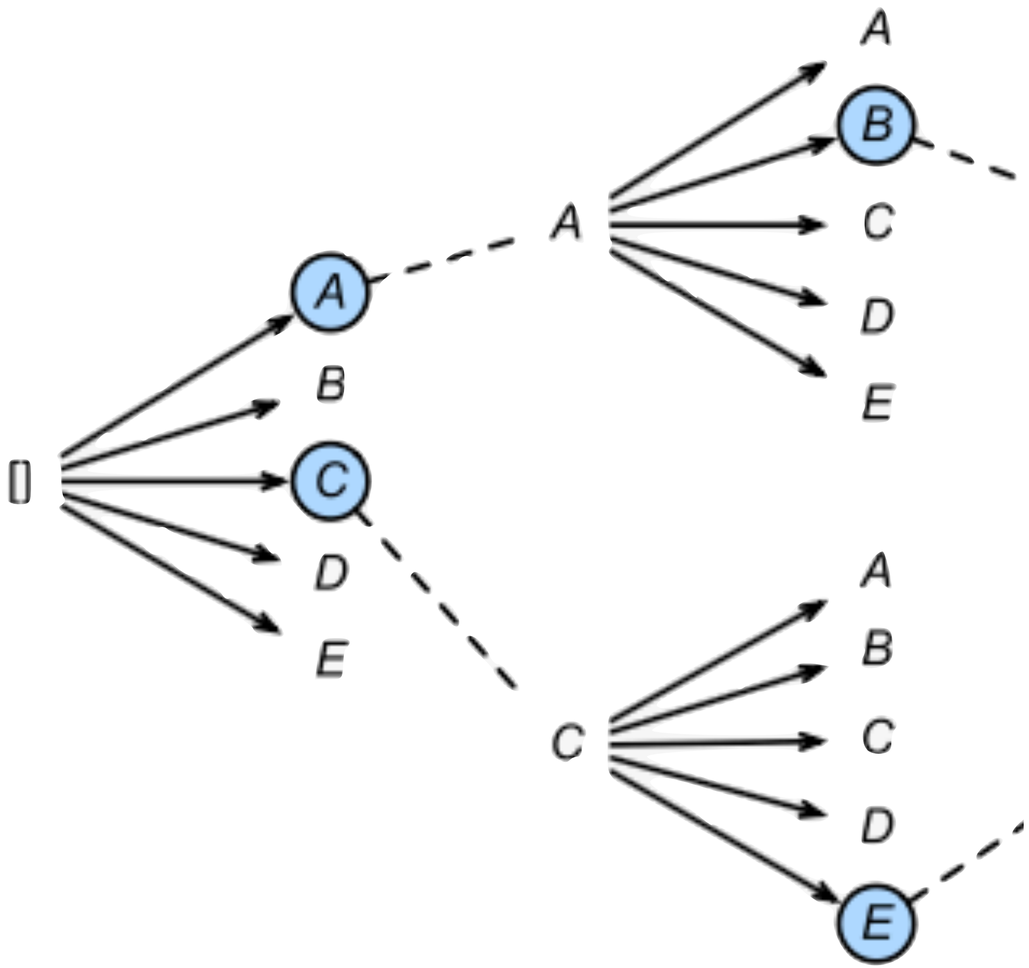
- Dado um vocabulário de tamanho **v**
- São gerados **k** vetores com **v** probabilidades
  - Totalizando **k*v** possibilidades de escolhas (novas hipóteses)
- Escolhemos as **k** hipóteses mais prováveis entre as novas hipóteses

- O processo continua até um critério de parada
  - Seja gerar n tokens ou encontrar um token de fim de sentença

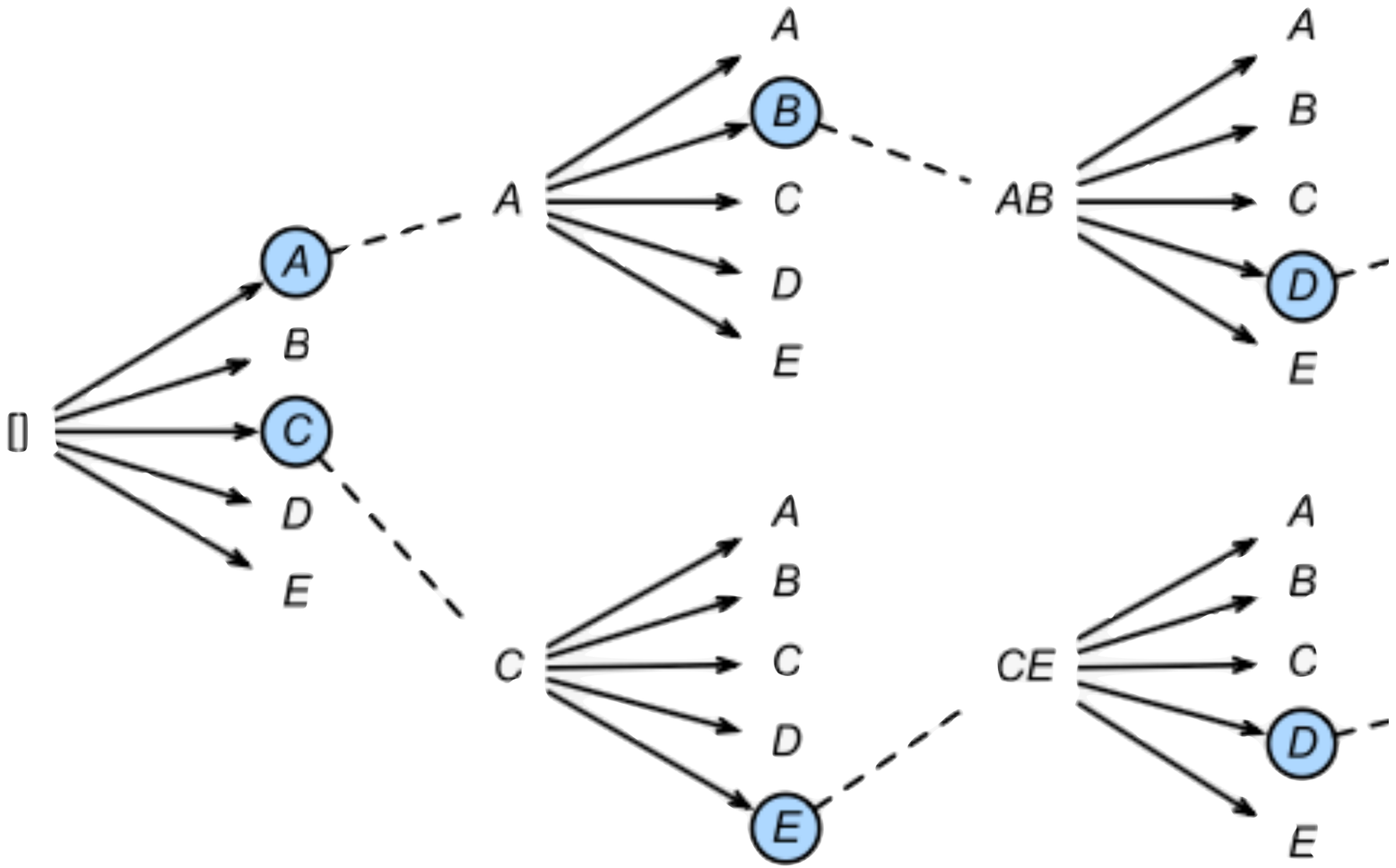
Beam Search



Beam Search



Beam Search



Beam Search