Decoding algorithms

THE
UNIVERSITY
OF UTAH

Where we are

We have seen different models for predicting sequences of tokens

- The underlying model could be a RNN or a transformer decoder

These autoregressive models

- Produce one token at a time
- The most recently predicted token is the input for the next step

The process of producing an output sequence involves chaining togther multiple decisions

This lecture: Decoding algorithms

How to predict a sequence

- The setup: An abstract auto-regressive model
- Decoding algorithms

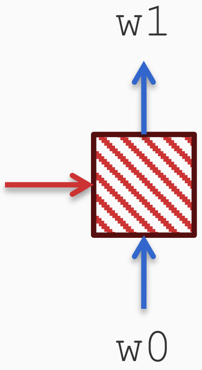
This lecture: Decoding algorithms

How to predict a sequence

- The setup: An abstract auto-regressive model
- Decoding algorithms

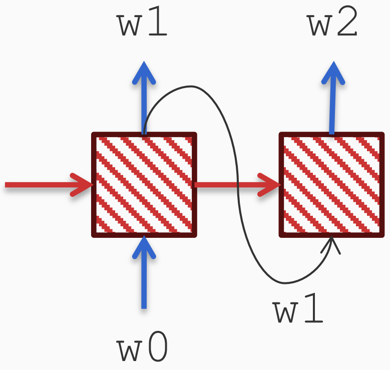
The setup: Any autoregressive model

The setup: Any autoregressive model



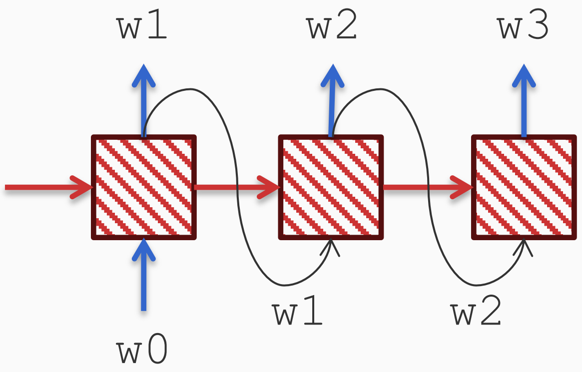
Recurrent neural network

The setup: Any autoregressive model



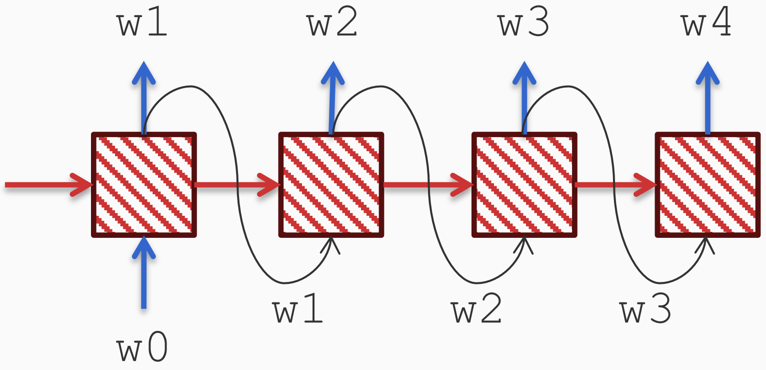
Recurrent neural network

The setup: Any autoregressive model



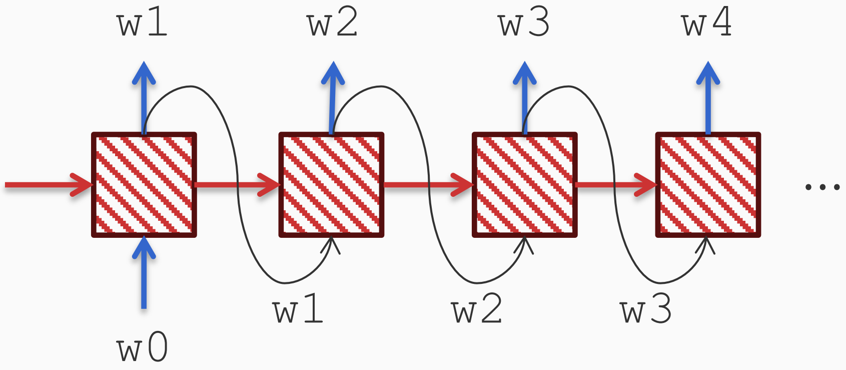
Recurrent neural network

The setup: Any autoregressive model



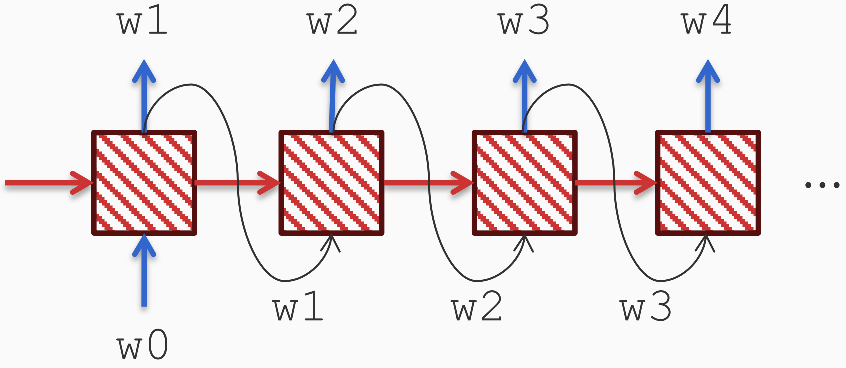
Recurrent neural network

The setup: Any autoregressive model

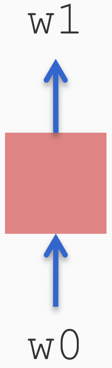


Recurrent neural network

The setup: Any autoregressive model

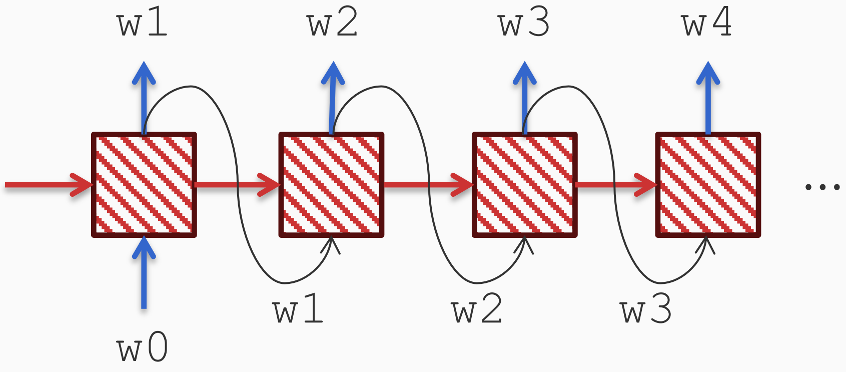


Recurrent neural network

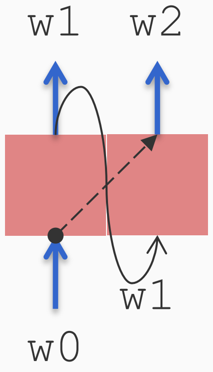


Transformer decoder

The setup: Any autoregressive model

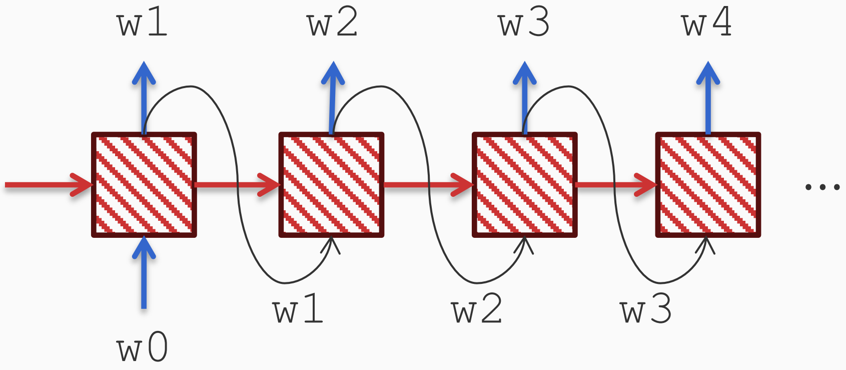


Recurrent neural network

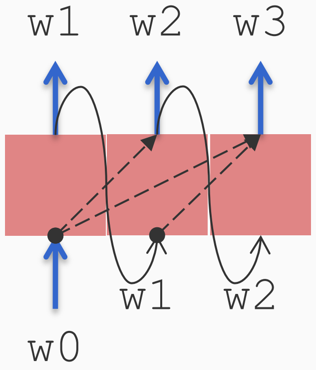


Transformer decoder

The setup: Any autoregressive model

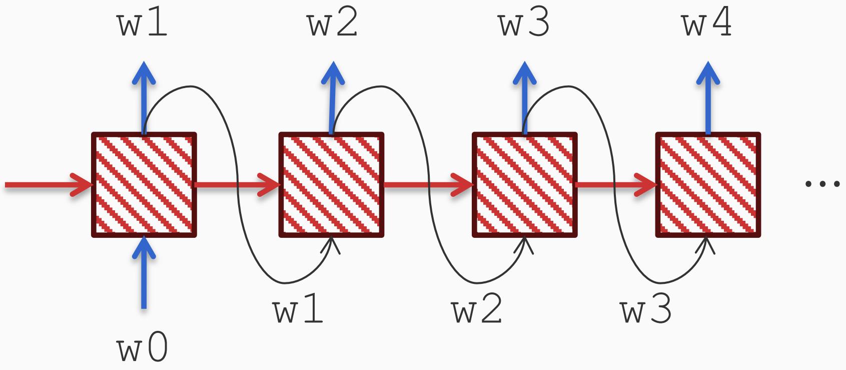


Recurrent neural network

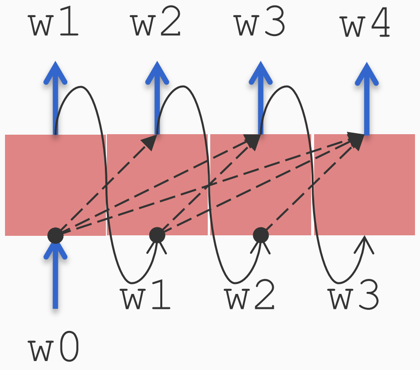


Transformer decoder

The setup: Any autoregressive model

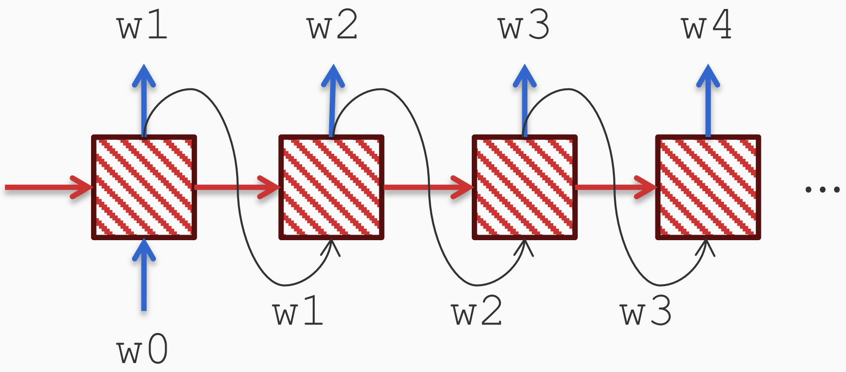


Recurrent neural network

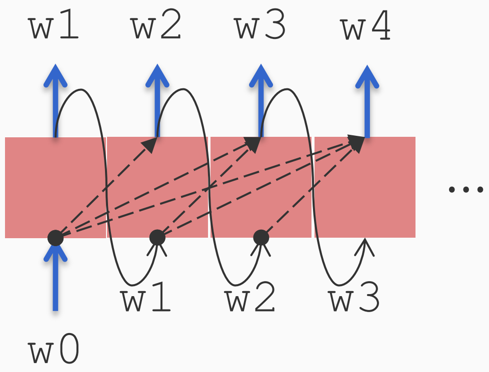


Transformer decoder

The setup: Any autoregressive model

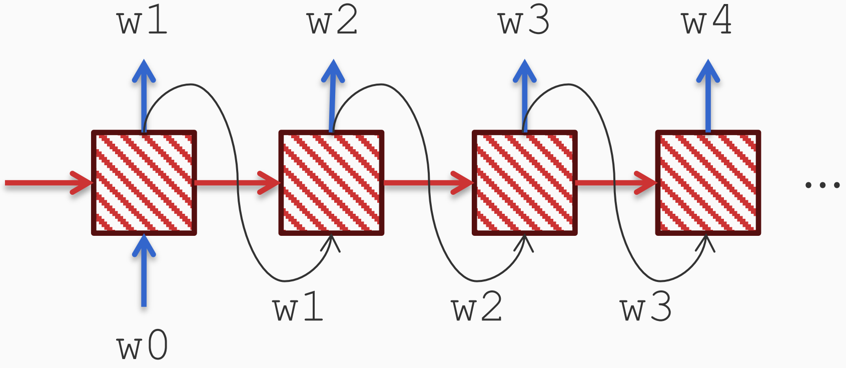


Recurrent neural network

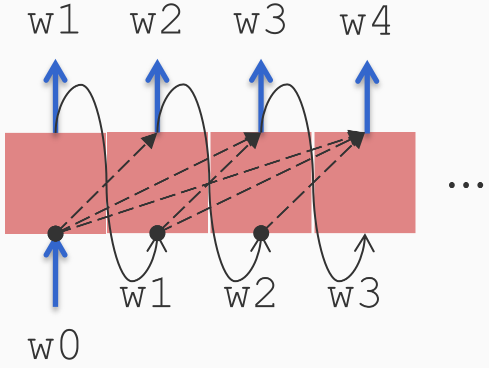


Transformer decoder

The setup: Any autoregressive model



Recurrent neural network

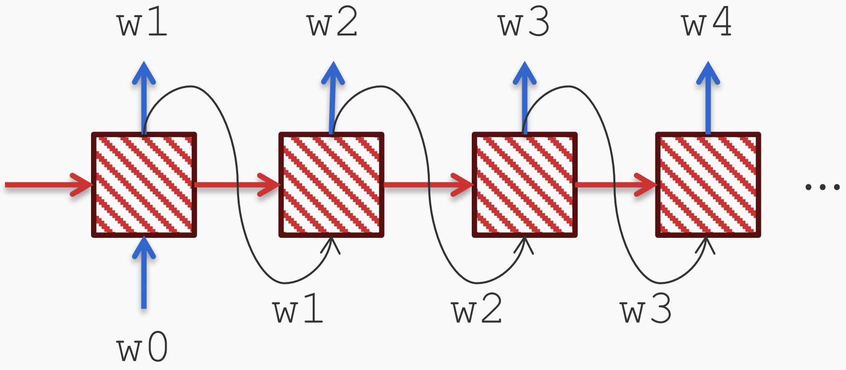


Transformer decoder

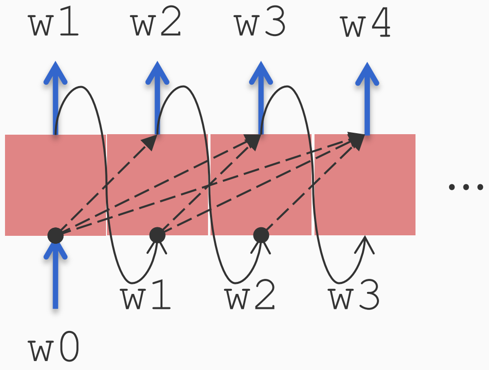
Both are ways to define probabilities for the next word given the sequence so far

The setup: Any autoregressive model

$$P(w_0 w_1 w_2 w_3 w_4 \cdots) = P(\text{token} \mid w_0) \cdot P(\text{token} \mid w_0 w_1) \cdot P(\text{token} \mid w_0 w_1 w_2) \cdot P(\text{token} \mid w_0 w_1 w_2 w_3) \cdots$$



Recurrent neural network



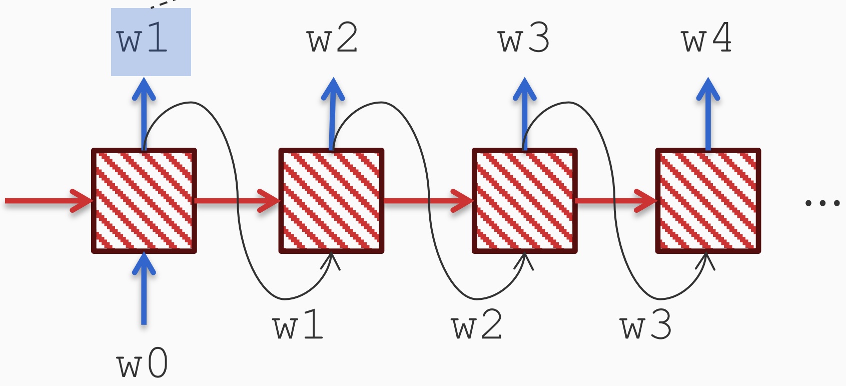
Transformer decoder

Both are ways to define probabilities for the next word given the sequence so far

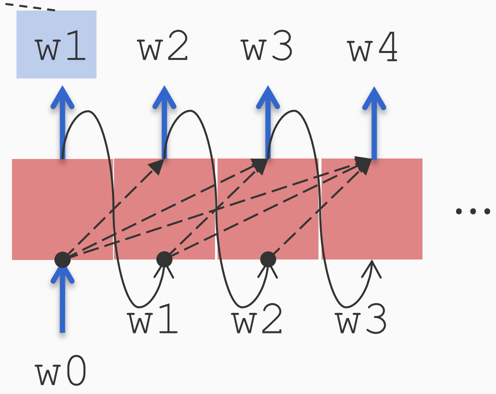
The setup: Any autoregressive model

The token `w1` is constructed using this distribution

$$P(w_0 w_1 w_2 w_3 w_4 \cdots) = P(\text{token} \mid w_0) \cdot P(\text{token} \mid w_0 w_1) \cdot P(\text{token} \mid w_0 w_1 w_2) \cdot P(\text{token} \mid w_0 w_1 w_2 w_3) \cdots$$



Recurrent neural network



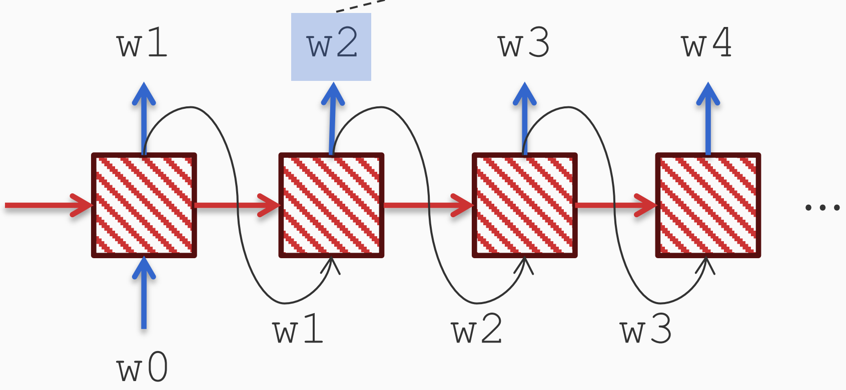
Transformer decoder

Both are ways to define probabilities for the next word given the sequence so far

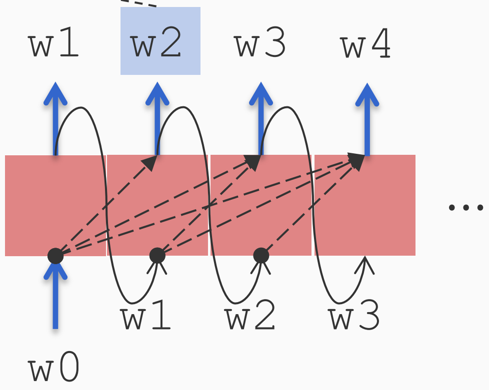
The setup: Any autoregressive model

The token w2 is constructed
using this distribution

$$P(w_0 w_1 w_2 w_3 w_4 \cdots) = P(\text{token} \mid w_0) \cdot \boxed{P(\text{token} \mid w_0 w_1)} \cdot P(\text{token} \mid w_0 w_1 w_2) \cdot P(\text{token} \mid w_0 w_1 w_2 w_3) \cdots$$



Recurrent neural network



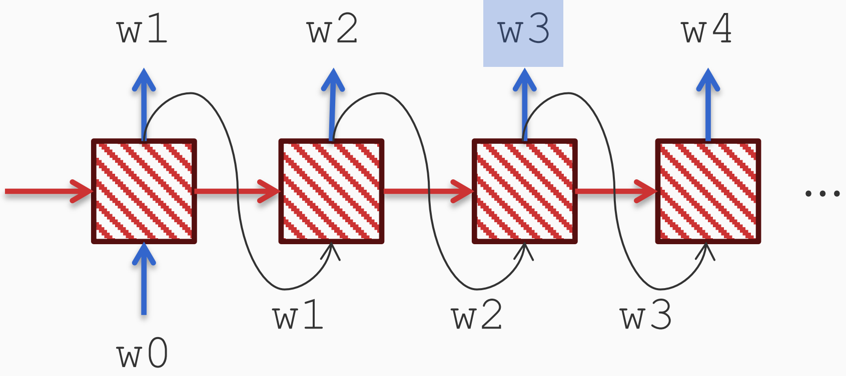
Transformer decoder

Both are ways to define probabilities for the next word given the sequence so far

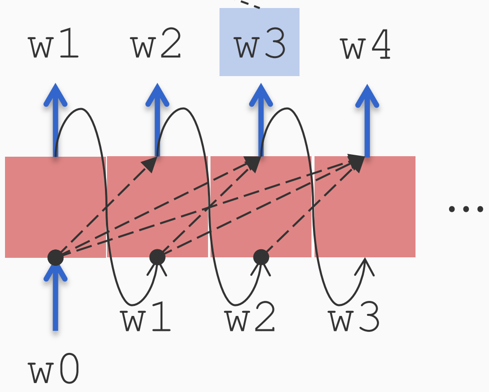
The setup: Any autoregressive model

The token `w3` is constructed using this distribution

$$P(w_0w_1w_2w_3w_4\cdots) = P(\text{token} \mid w_0)\cdot P(\text{token} \mid w_0w_1)\cdot P(\text{token} \mid w_0w_1w_2)\cdot P(\text{token} \mid w_0w_1w_2w_3)\cdots$$



Recurrent neural network



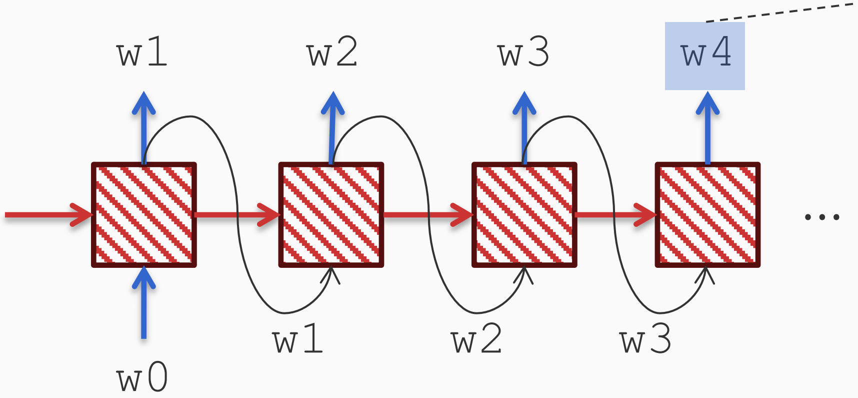
Transformer decoder

Both are ways to define probabilities for the next word given the sequence so far

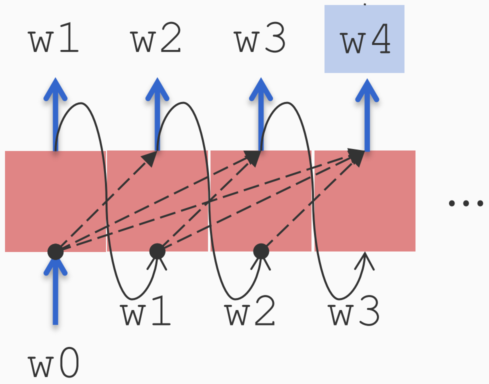
The setup: Any autoregressive model

The token `w4` is constructed using this distribution

$$P(w_0 w_1 w_2 w_3 w_4 \cdots) = P(\text{token} \mid w_0) \cdot P(\text{token} \mid w_0 w_1) \cdot P(\text{token} \mid w_0 w_1 w_2) \cdot \boxed{P(\text{token} \mid w_0 w_1 w_2 w_3)} \cdots$$



Recurrent neural network



Transformer decoder

Both are ways to define probabilities for the next word given the sequence so far

This lecture: Decoding algorithms

How to predict a sequence

- The setup: An abstract auto-regressive model
- Decoding algorithms

Decoding: The algorithmic question

Given some method to create a probability distribution $P(\mathbf{token} \mid w_0 w_1 w_2 \cdots)$
how should we predict the “best” sequence?

Decoding: The algorithmic question

Given some method to create a probability distribution $P(\textbf{token} \mid w_0 w_1 w_2 \cdots)$ how should we predict the "best" sequence?

The answer to this question does not depend on what kind of model we have underneath the probabilities

Decoding: The algorithmic question

The answer to this question does not depend on what kind of model we have underneath the probabilities

Given some method to create a probability distribution $P(\mathbf{token} \mid w_0 w_1 w_2 \cdots)$ how should we predict the "best" sequence?

What does *best* mean? Ideas?

Decoding: The algorithmic question

The answer to this question does not depend on what kind of model we have underneath the probabilities

Given some method to create a probability distribution $P(\textbf{token} \mid w_0 w_1 w_2 \cdots)$ how should we predict the "best" sequence?

What does *best* mean? Ideas?

Some notions of best when it comes to generating text

- Most probable
- Fast
- Does not repeat
- Diverse outputs
- ....

A toy example

Suppose our language model can pick from one of the following words at any step:

a, the, he, she, saw, him, her, apple

Some model predicts a conditional distribution given the words seen so far

Every token in the vocabulary is assigned a probability

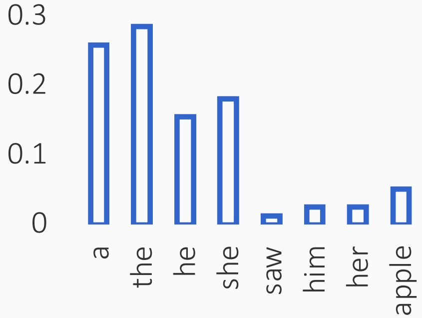
A toy example

Suppose our language model can pick from one of the following words at any step:

a, the, he, she, saw, him, her, apple

Some model predicts a conditional distribution given the words seen so far

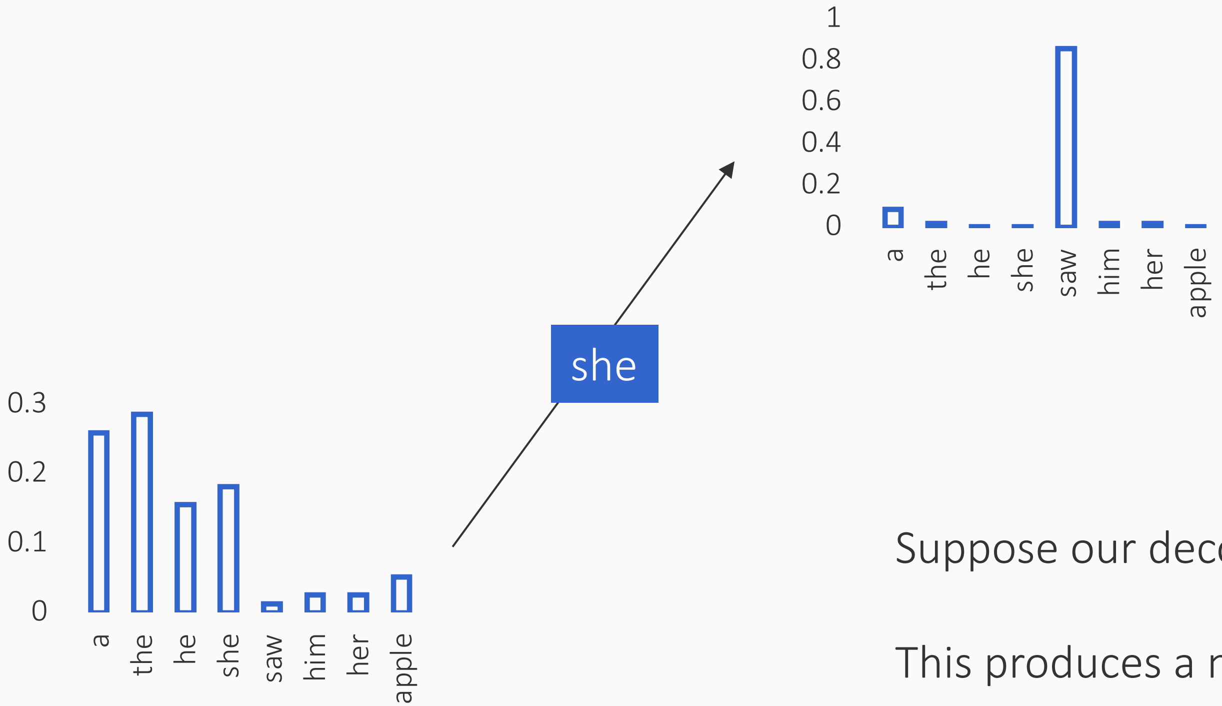
Every token in the vocabulary is assigned a probability



A toy example

Suppose our language model can pick from one of the following words at any step:

a, the, he, she, saw, him, her, apple



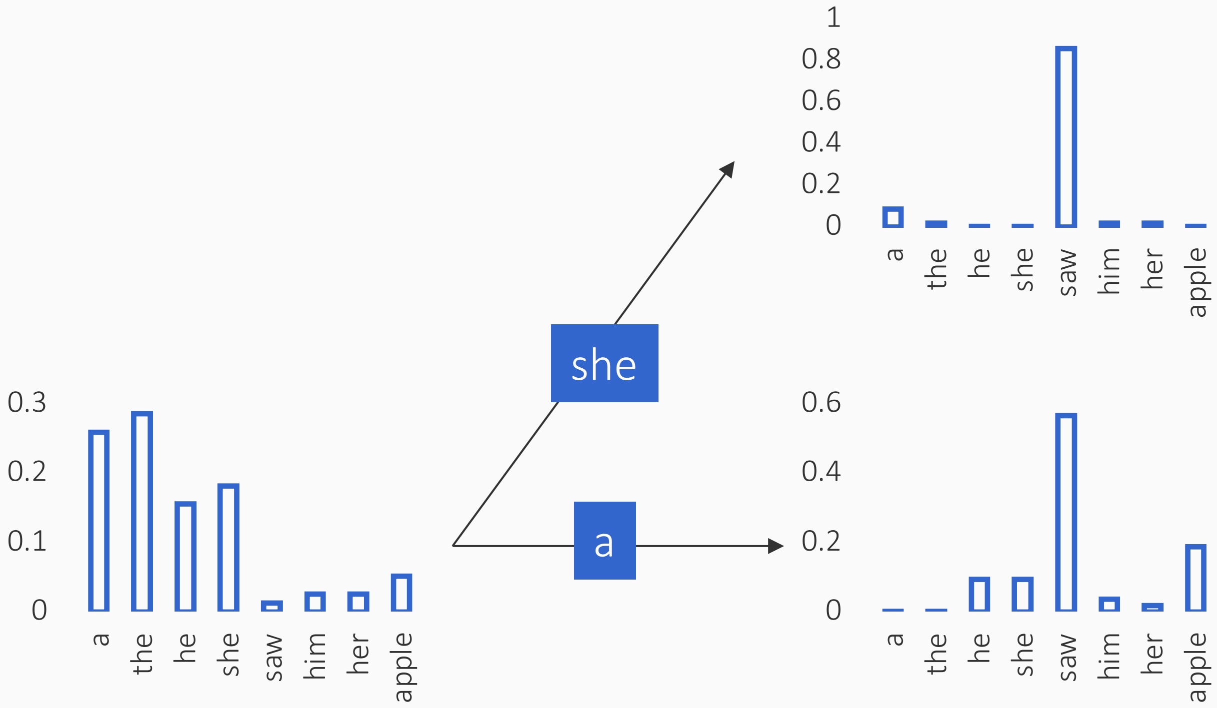
Suppose our decoder decides to pick the word "she"

This produces a new distribution over the next token

A toy example

Suppose our language model can pick from one of the following words at any step:

a, the, he, she, saw, him, her, apple

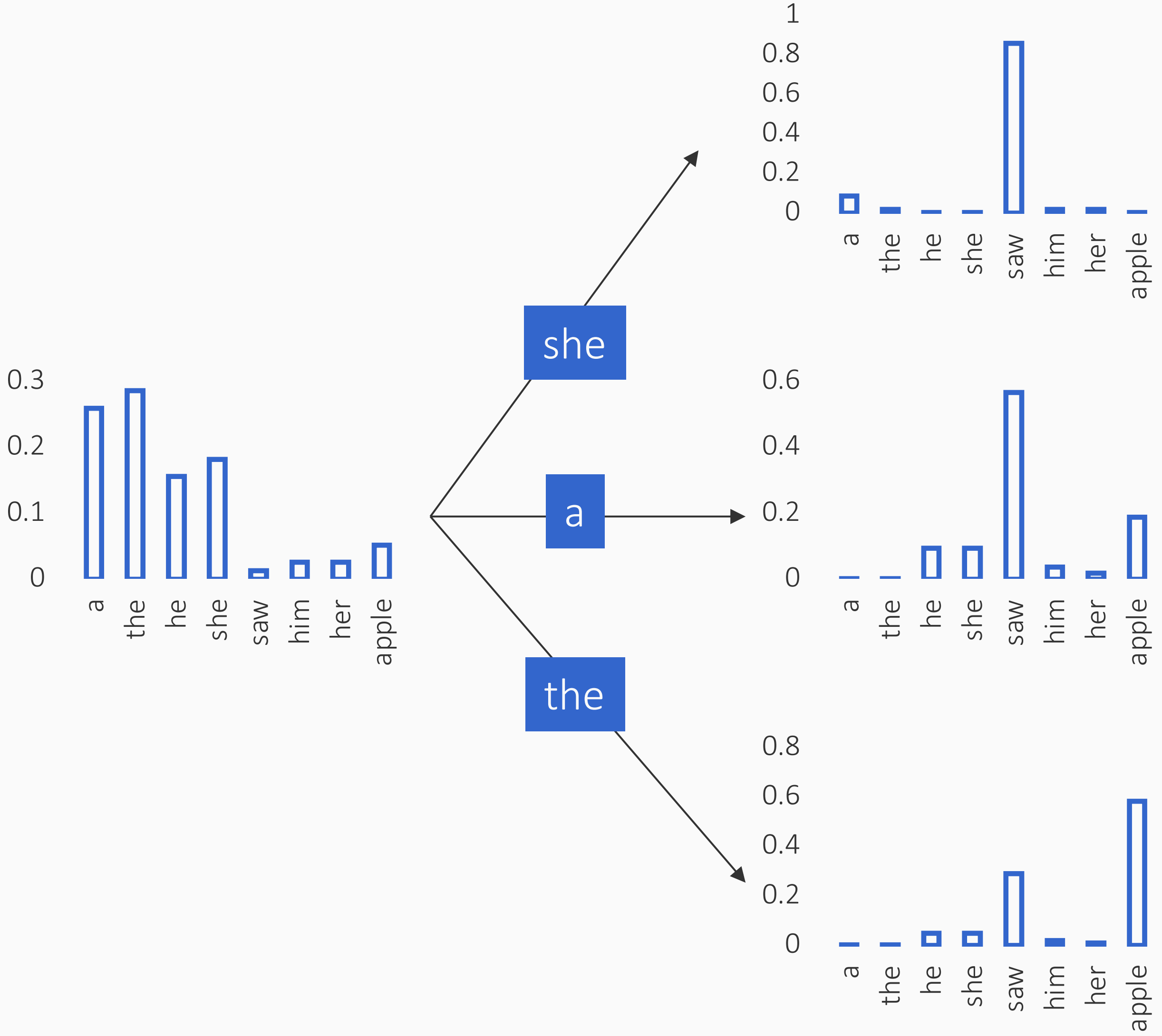


If it picked a different token, say “a”, then the next token distribution would be different.

A toy example

Suppose our language model can pick from one of the following words at any step:

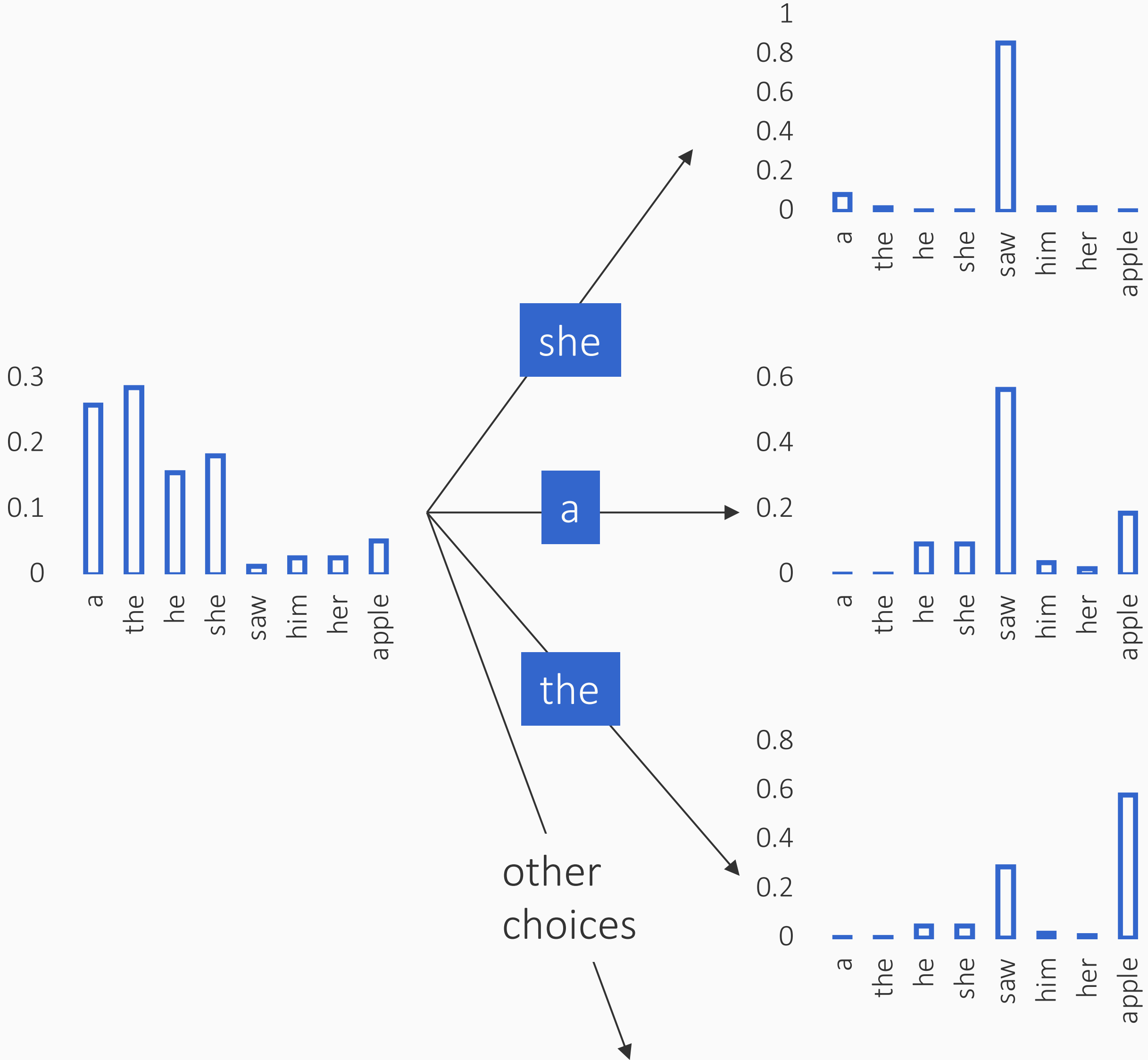
a, the, he, she, saw, him, her, apple



A toy example

Suppose our language model can pick from one of the following words at any step:

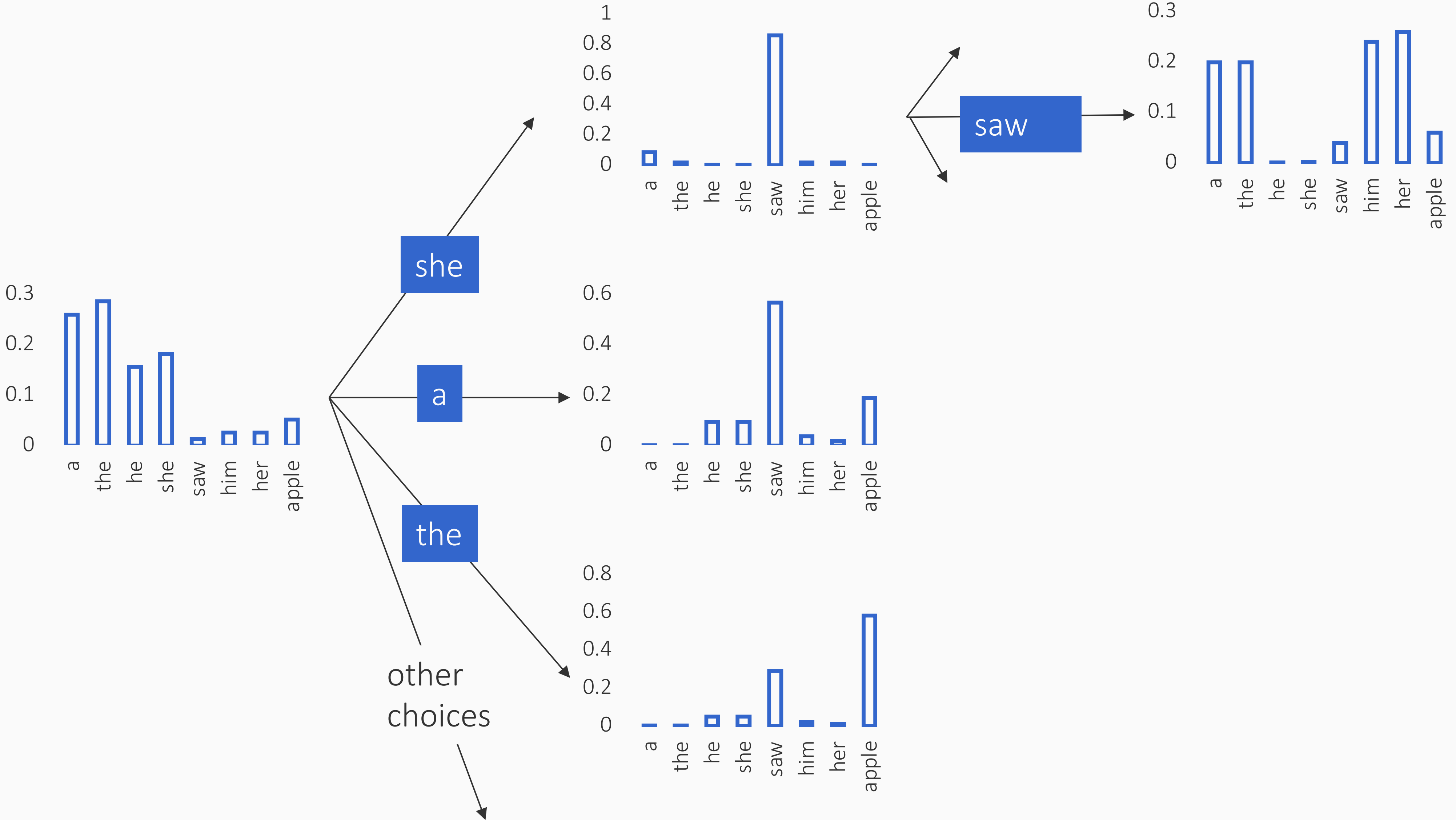
a, the, he, she, saw, him, her, apple



A toy example

Suppose our language model can pick from one of the following words at any step:

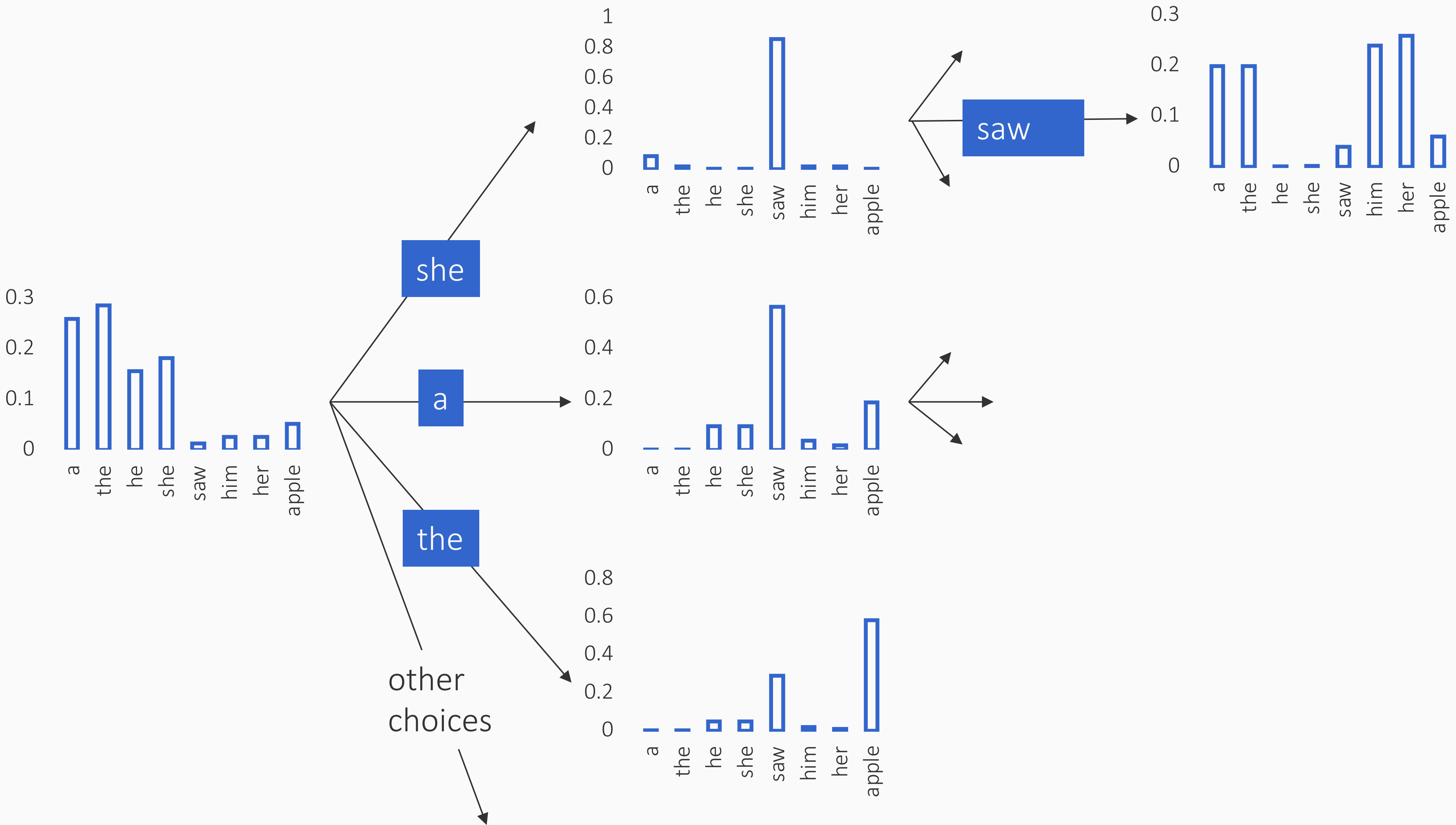
a, the, he, she, saw, him, her, apple



A toy example

Suppose our language model can pick from one of the following words at any step:

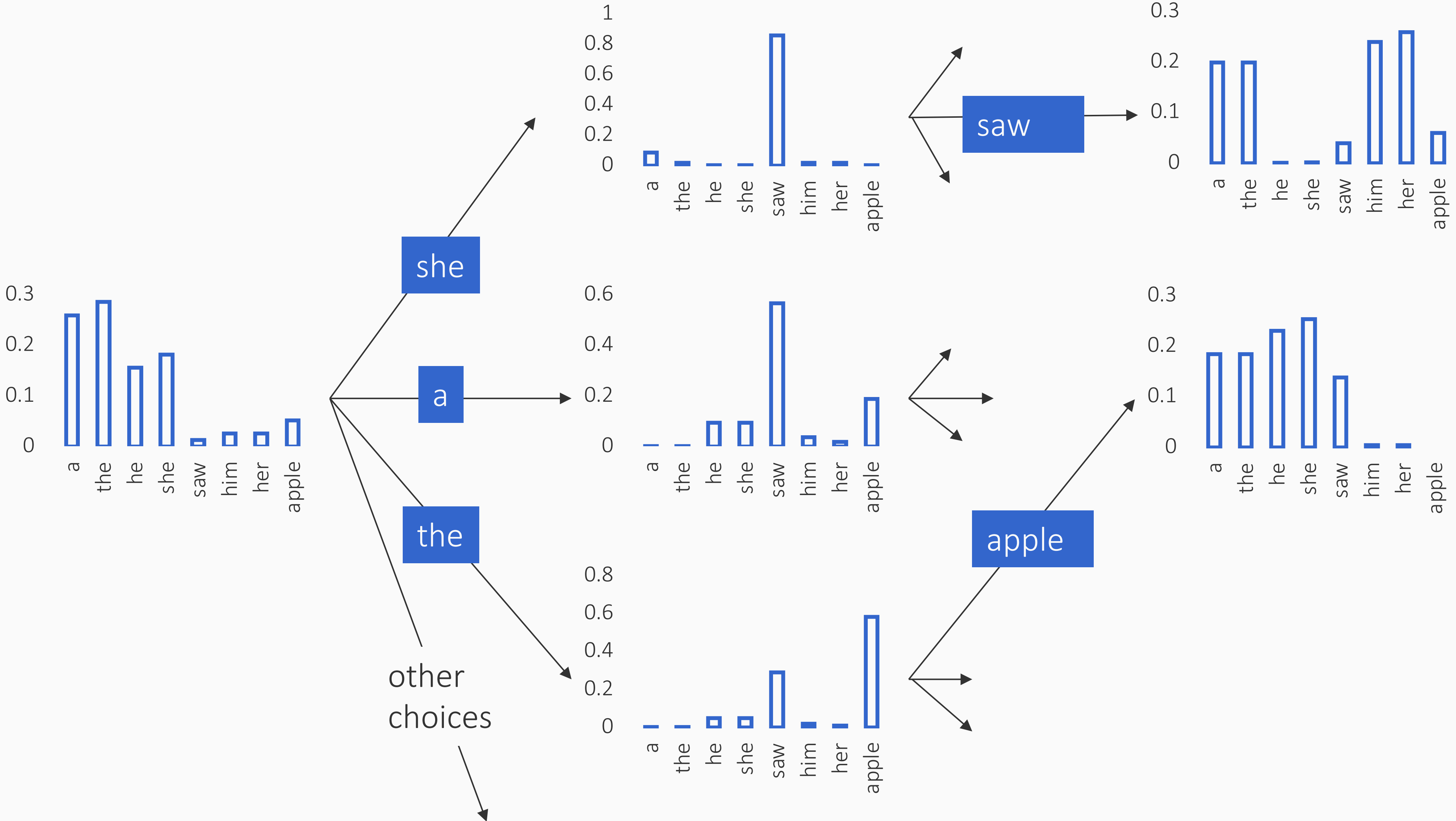
a, the, he, she, saw, him, her, apple



A toy example

Suppose our language model can pick from one of the following words at any step:

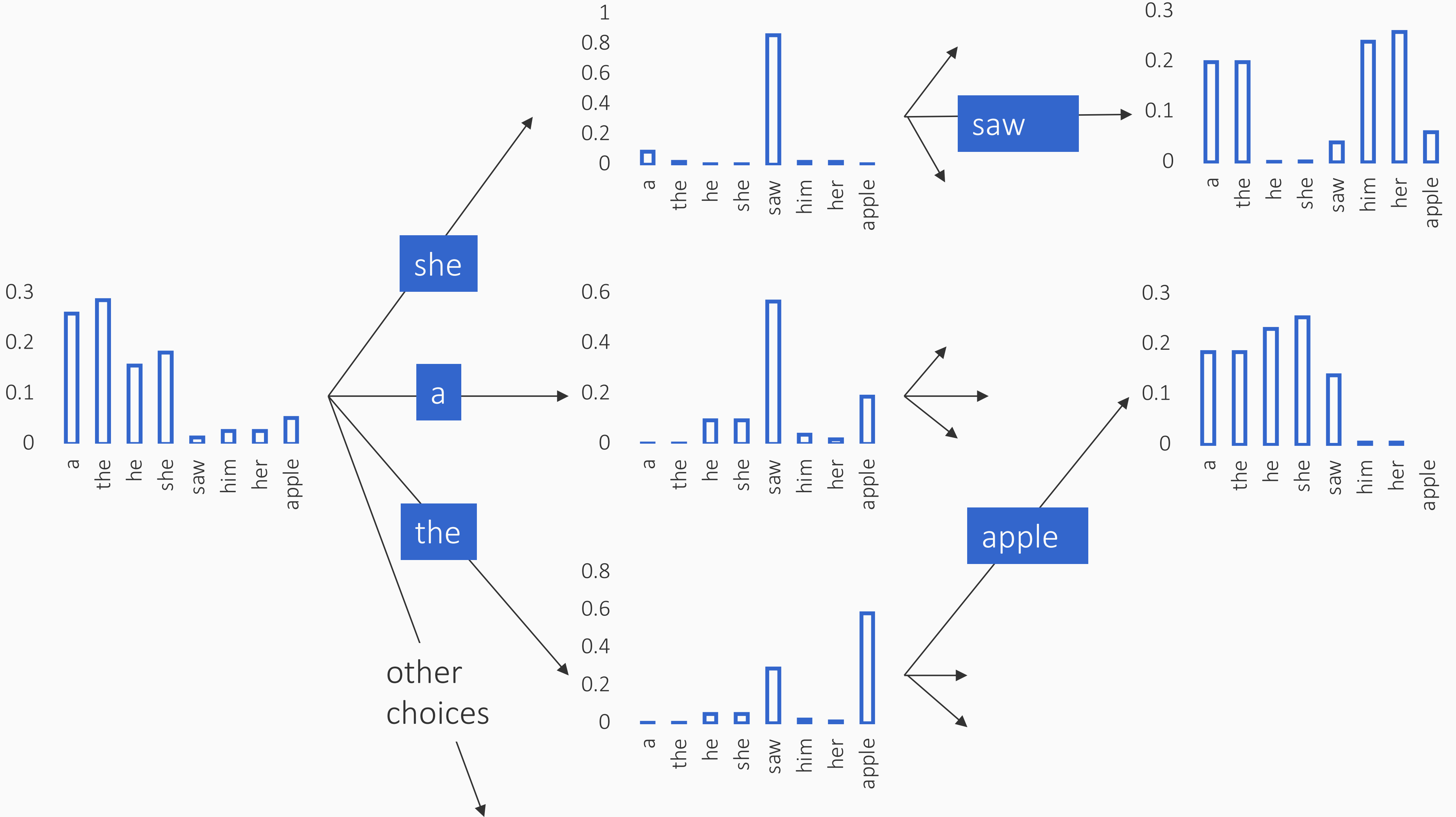
a, the, he, she, saw, him, her, apple



A toy example

Suppose our language model can pick from one of the following words at any step:

a, the, he, she, saw, him, her, apple



she

a

the

other choices

saw

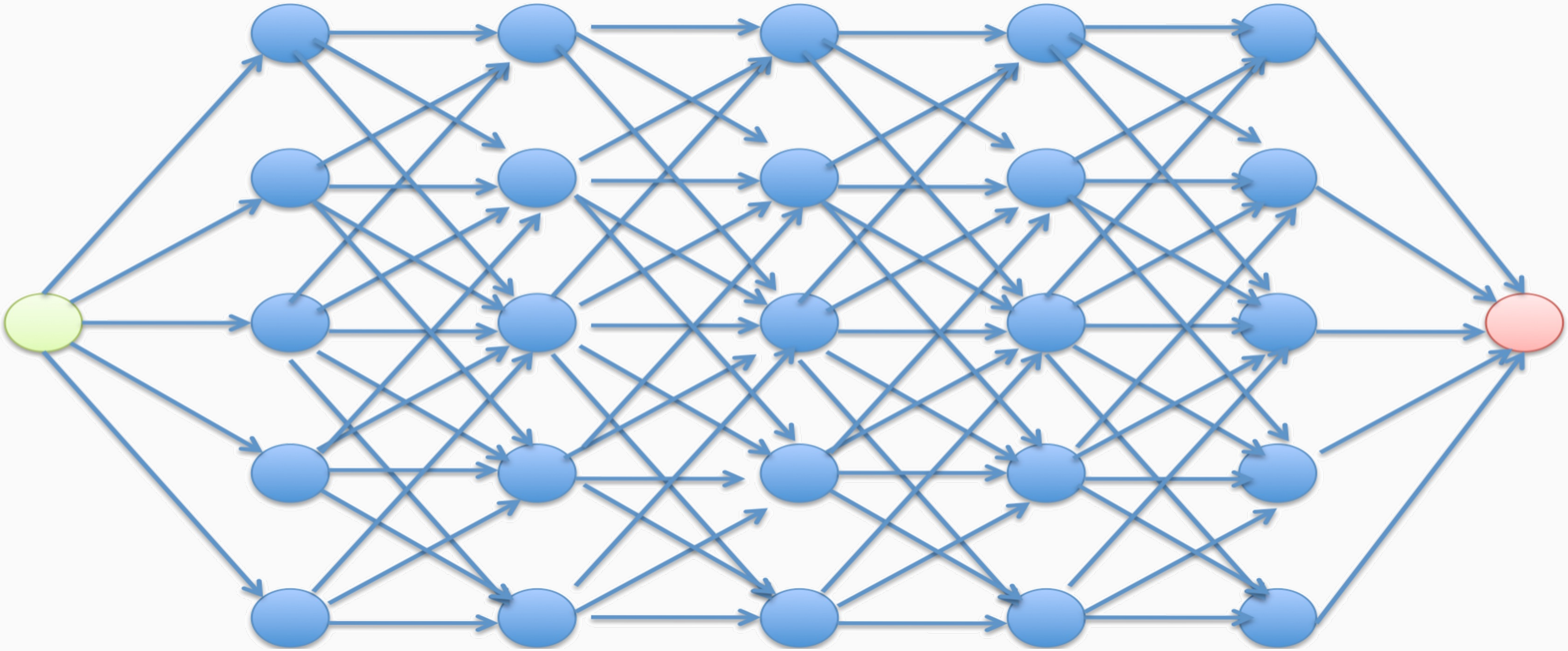
apple

And so on…

The search space for decoding

Each circle is a word that gets picked at a certain step

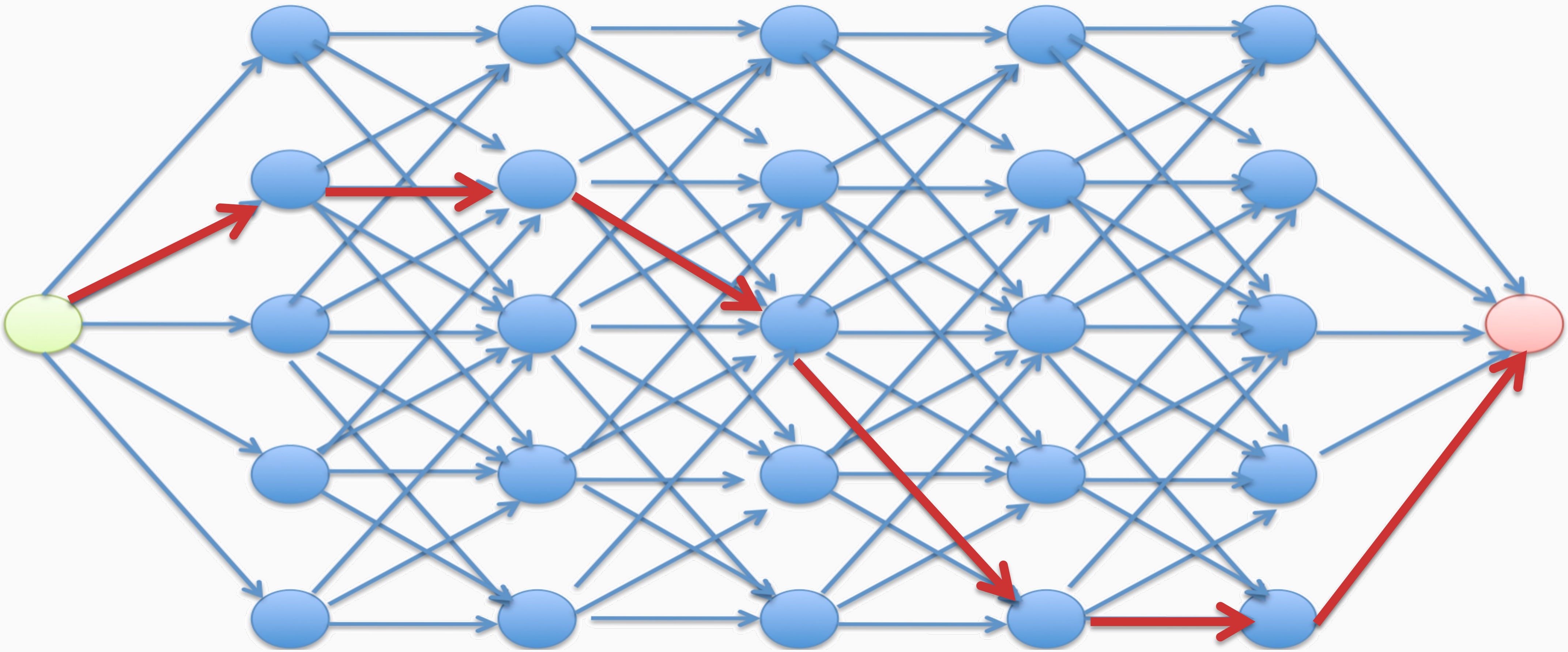
Goal: To find the "best" path in this graph



The search space for decoding

Each circle is a word that gets picked at a certain step

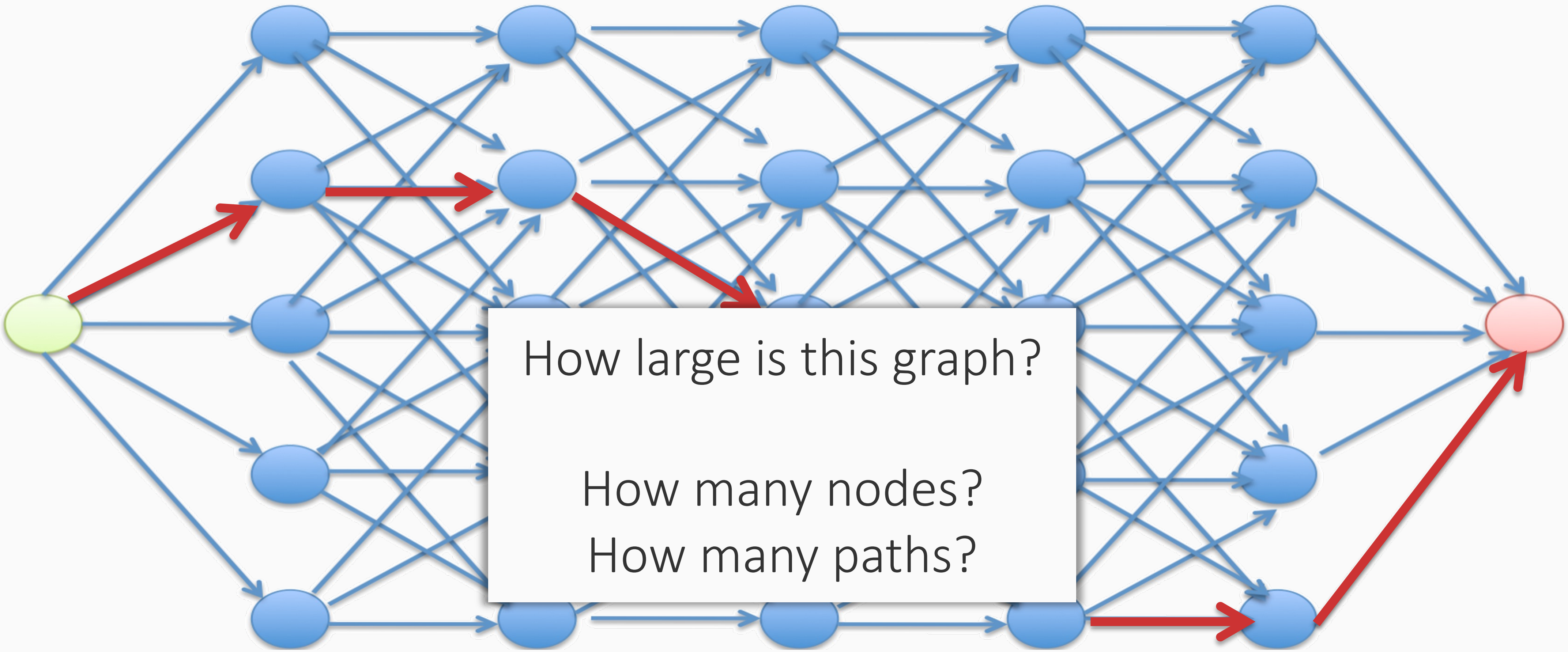
Goal: To find the "best" path in this graph



The search space for decoding

Each circle is a word that
gets picked at a certain step

Goal: To find the “best” path in this graph



How large is this graph?

How many nodes?
How many paths?

Different decoding strategies exist

Search based decoding

Sampling based decoding

Different decoding strategies exist

Search based decoding

Deterministic approaches that involves searching the space of sequences to select one

Sampling based decoding

Different decoding strategies exist

Search based decoding

Deterministic approaches that involves searching the space of sequences to select one

Sampling based decoding

Randomized approaches that involve sampling from the token conditional probability distribution

Different decoding strategies exist

Search based decoding

Deterministic approaches that involves searching the space of sequences to select one

- Greedy decoding
- Beam search

Sampling based decoding

Randomized approaches that involve sampling from the token conditional probability distribution

- Random sampling
- Top-K sampling
- Nucleus sampling

Different decoding strategies exist

Search based decoding

Deterministic approaches that involves searching the space of sequences to select one

- Greedy decoding
- Beam search

Sampling based decoding

Randomized approaches that involve sampling from the token conditional probability distribution

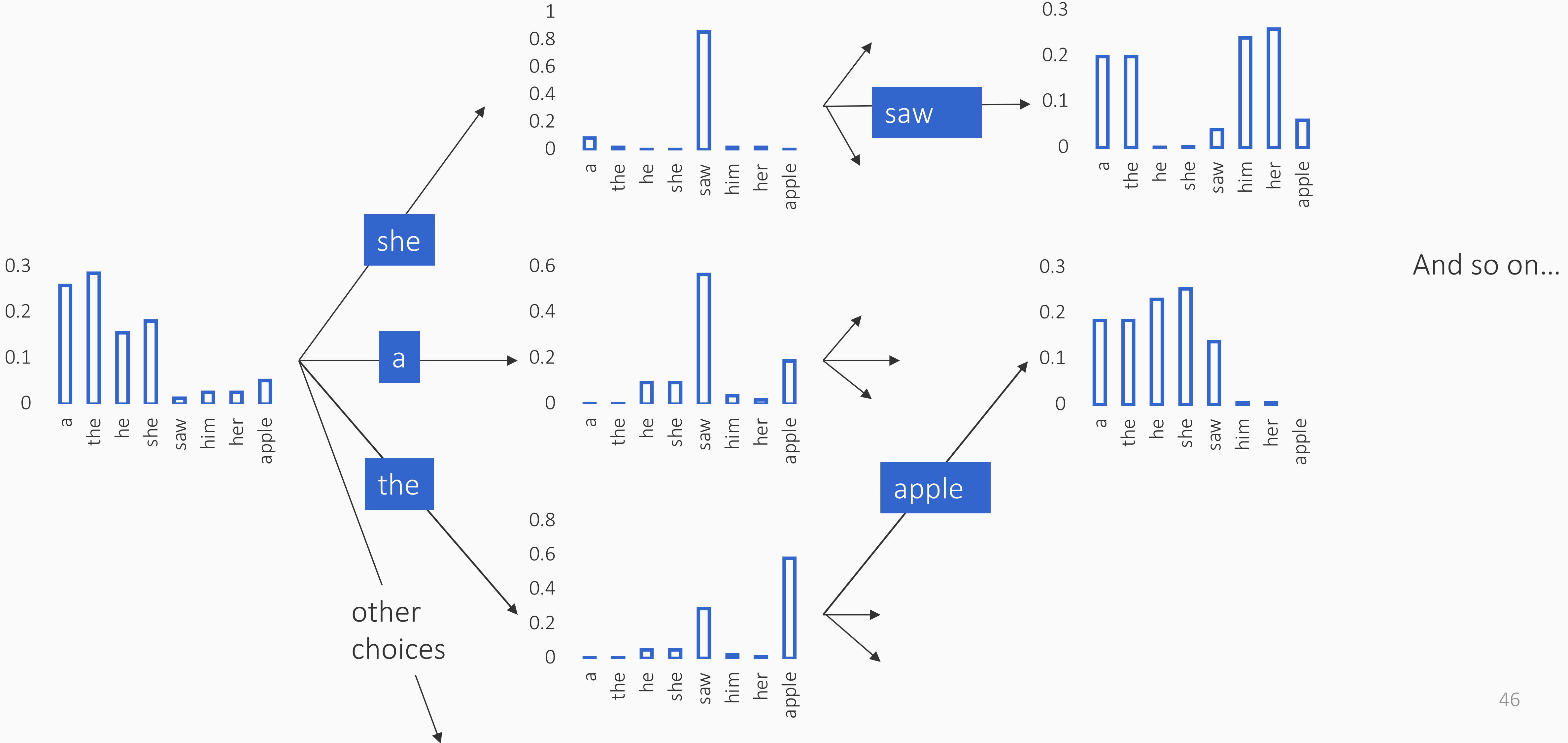
- Random sampling
- Top-K sampling
- Nucleus sampling

Greedy decoding

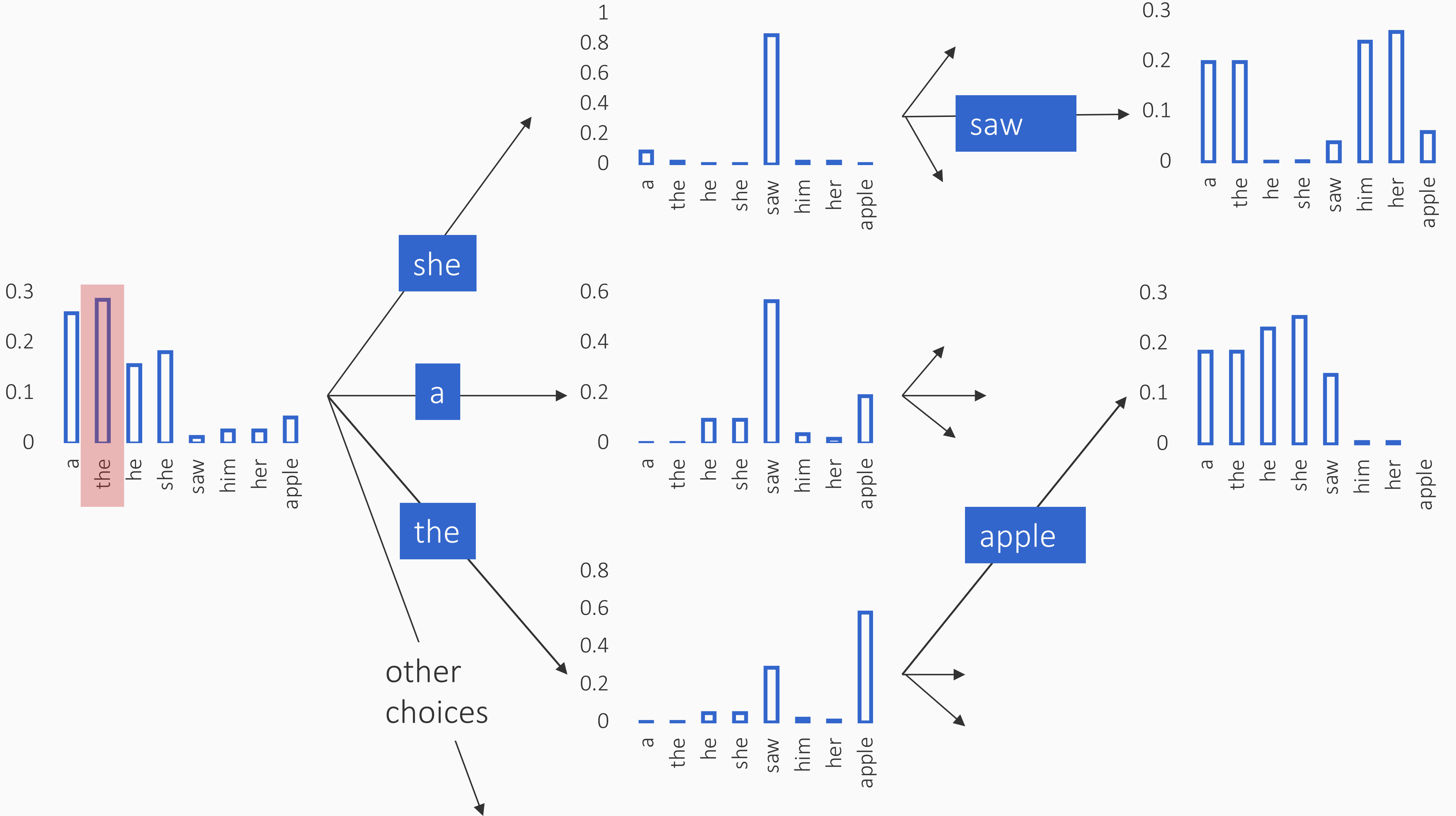
At each step, pick the token that has the highest probability

$$w_n = \underset{\{v \in \text{Vocabulary}\}}{\operatorname{argmax}} P\left(v \mid w_0 w_1 \cdots w_{\{n-1\}} \right)$$

Greedy decoding in our toy setting

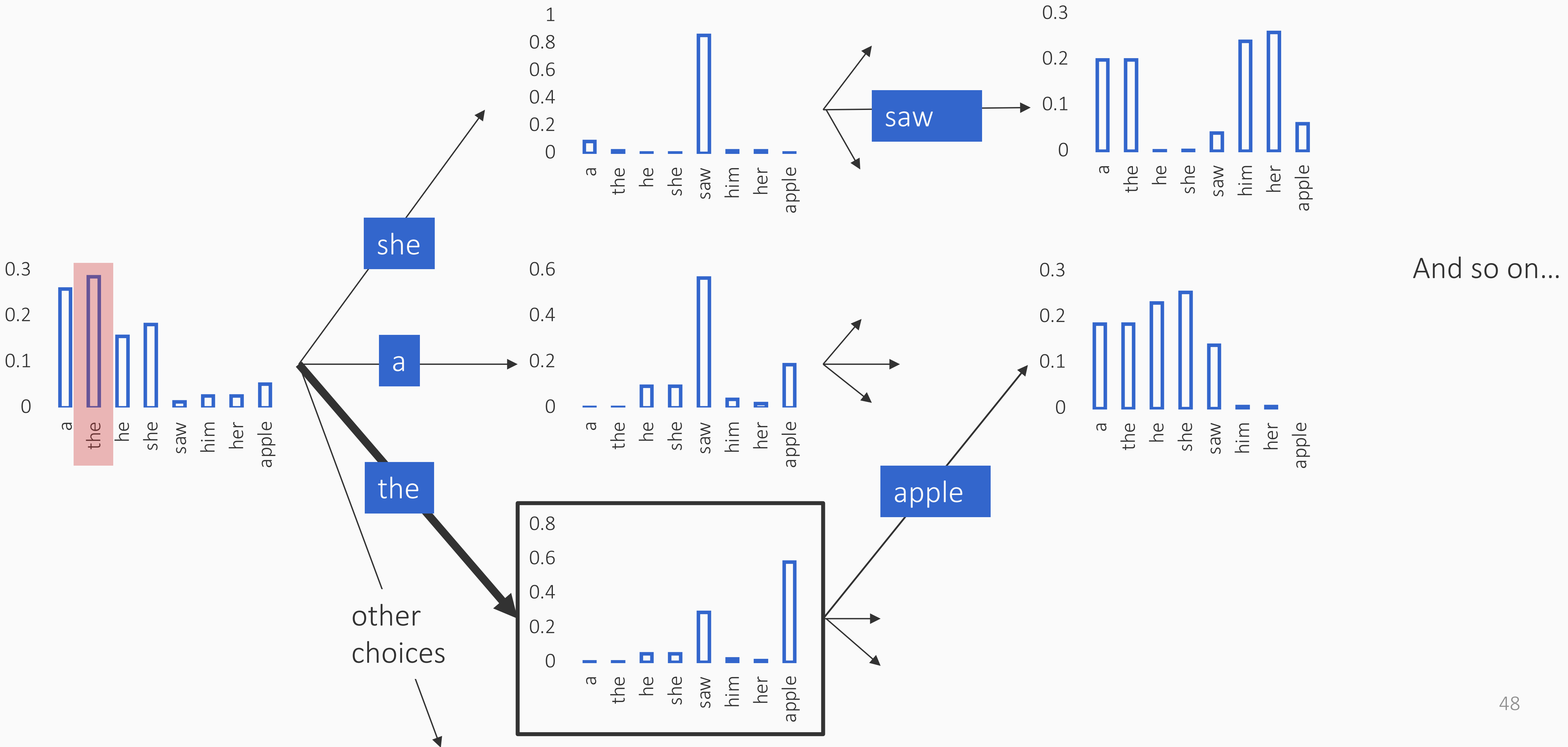


Greedy decoding in our toy setting

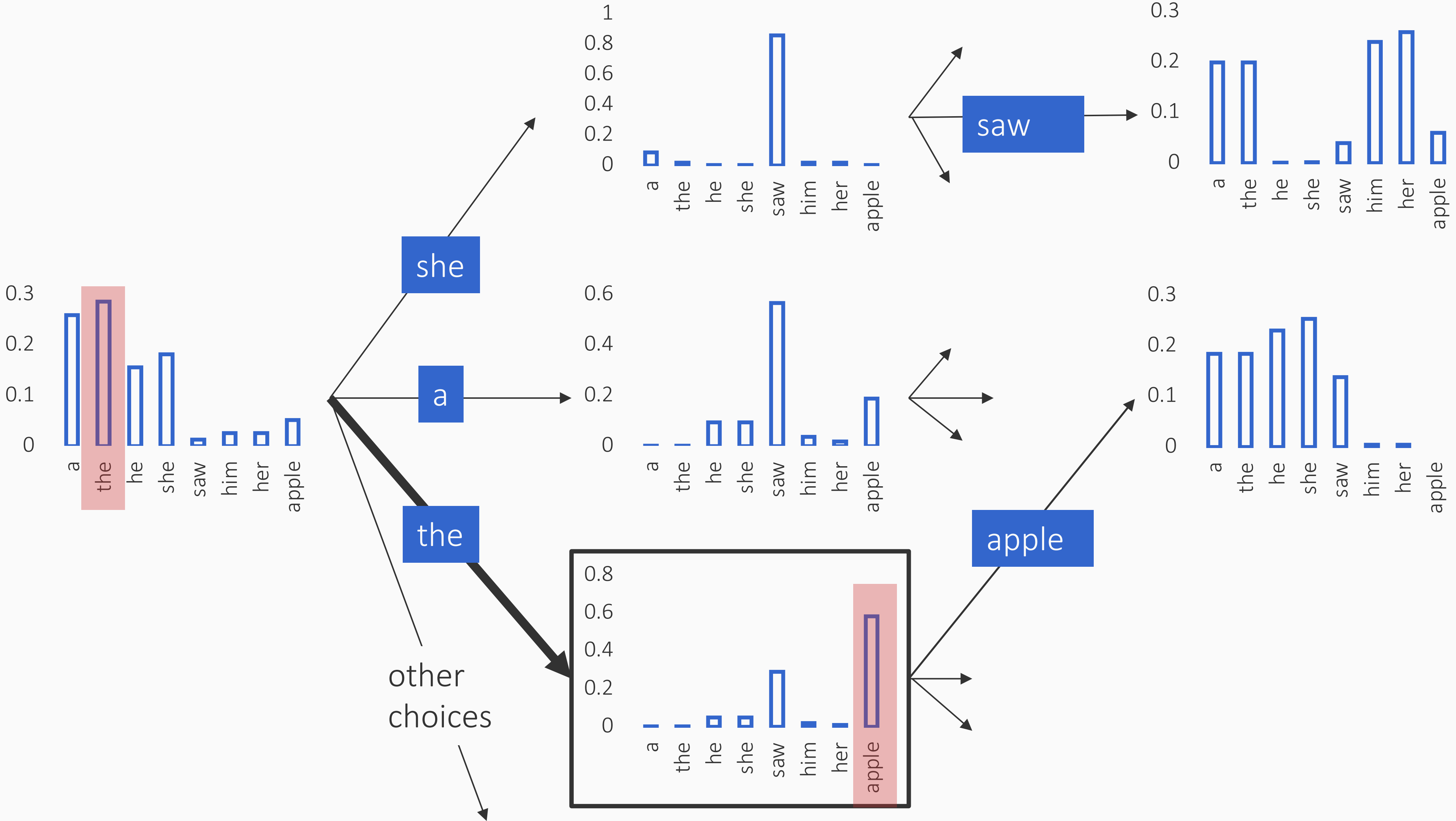


And so on…

Greedy decoding in our toy setting

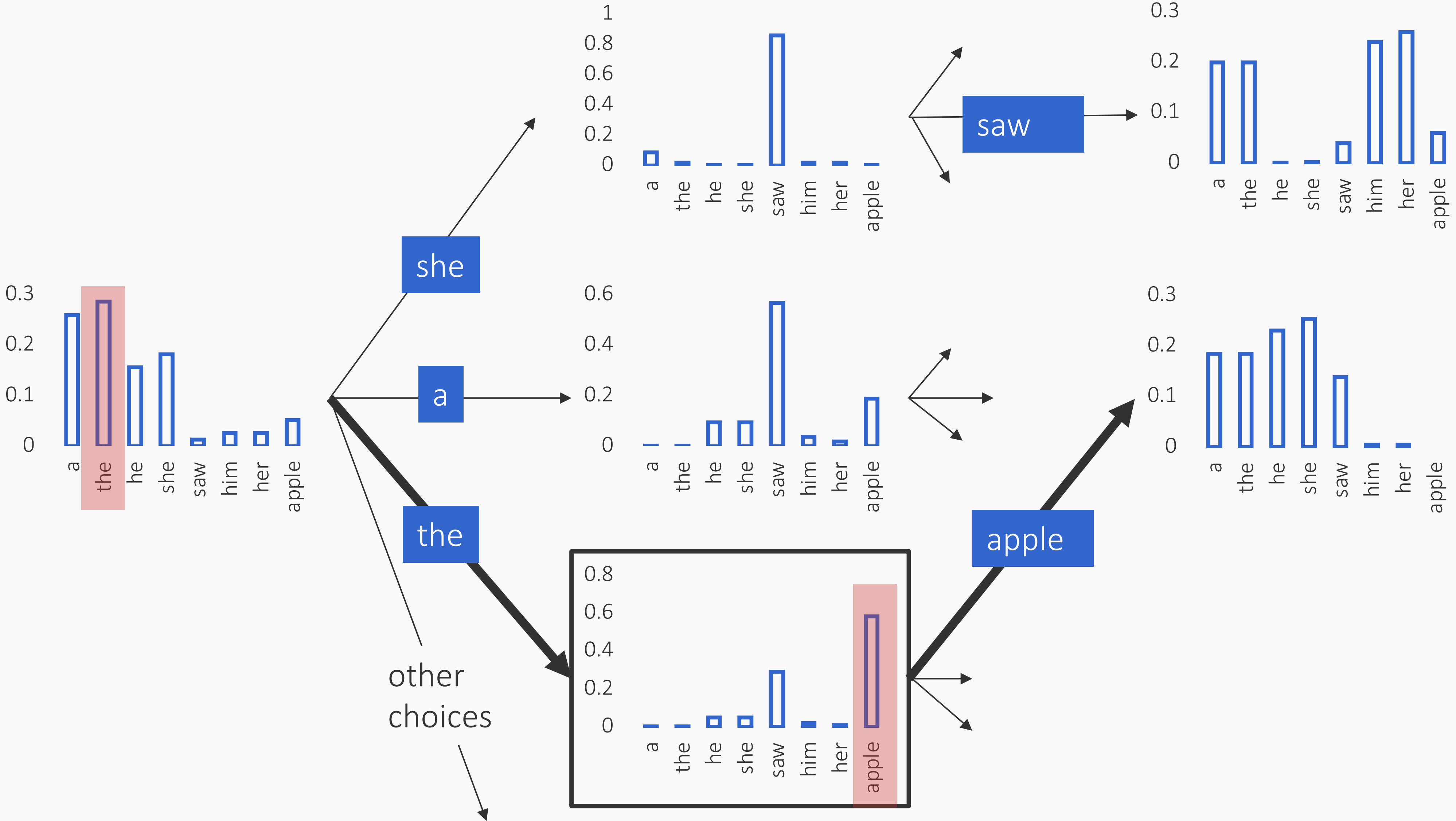


Greedy decoding in our toy setting



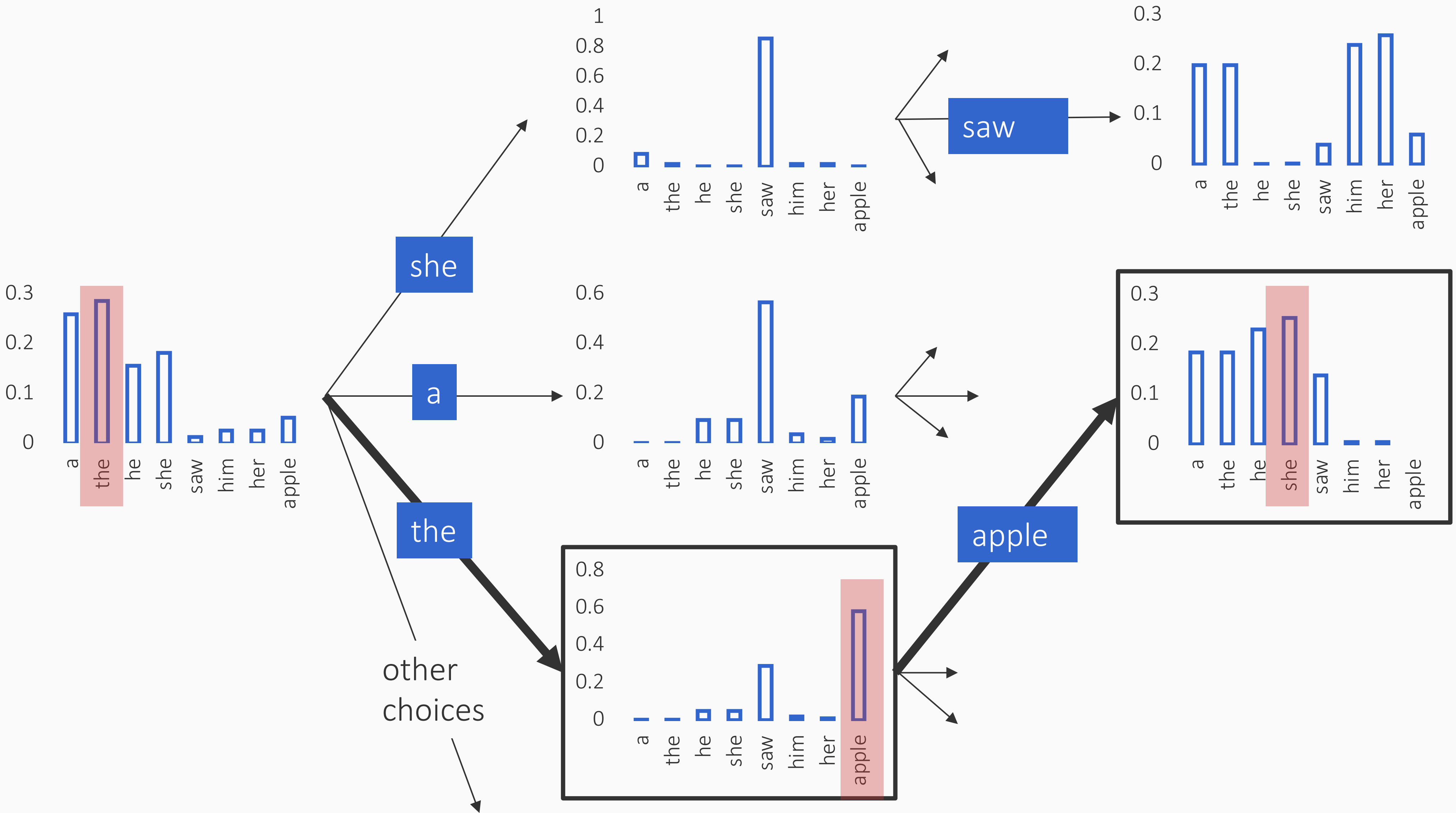
And so on…

Greedy decoding in our toy setting



And so on…

Greedy decoding in our toy setting



And so on…

Greedy decoding

At each step, pick the token that has the highest probability

$$w_n = \underset{\{v \in \text{Vocabulary}\}}{\operatorname{argmax}} P\left(v \mid w_0 w_1 \cdots w_{\{n-1\}} \right)$$

Pros:

- Simple
- Fast

Cons:

- If there is a high probability word later in the sequence, but to get there the model should have chosen a low probability word, this would get missed
- Also, in practice, the outputs can be repetitive

Different decoding strategies exist

Search based decoding

Deterministic approaches that involves searching the space of sequences to select one

- Greedy decoding
- Beam search

Sampling based decoding

Randomized approaches that involve sampling from the token conditional probability distribution

- Random sampling
- Top-K sampling
- Nucleus sampling

Beam search

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

What we might really want to do is to explore the full search space.

We cannot. Beam search is a compromise

Beam search: A compromise

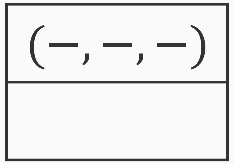
- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2

Beam search: A compromise

- Keep size-limited priority queue of states
    - Called the beam, sorted by probability for the state
- At each step:
    - Explore all transitions from the current state
    - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2



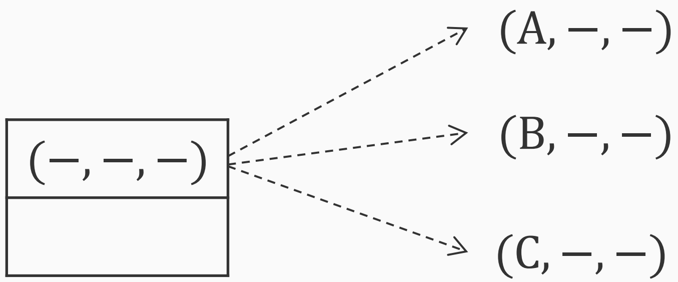
$(-, -, -)$

At the beginning, the beam has only one element, the start state

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2

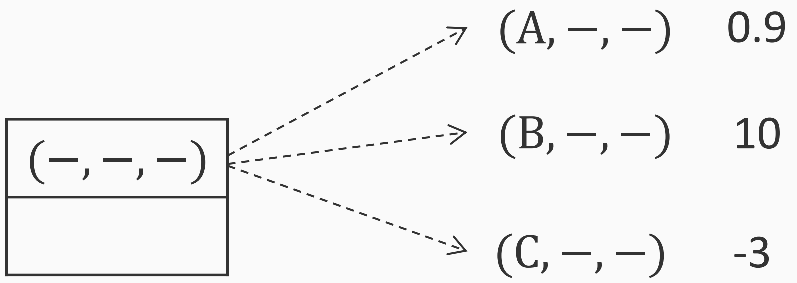


Expand all the states in the beam

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2



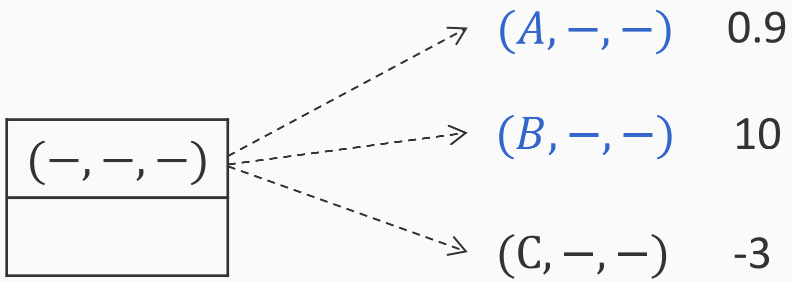
Expand all the states in the beam

Score the newly created states

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2



$(-,-,-)$

$(A,-,-)$ 0.9

$(B,-,-)$ 10

$(C,-,-)$ -3

Expand all the states in the beam

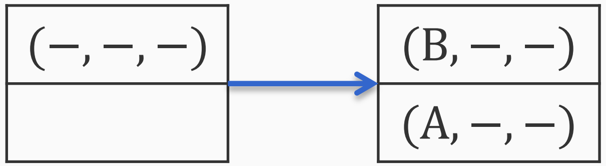
Score the newly created states

The top k new states form the new beam (sorted)

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2



$(-,-,-)$

$(B,-,-)$

$(A,-,-)$

Expand all the states in the beam

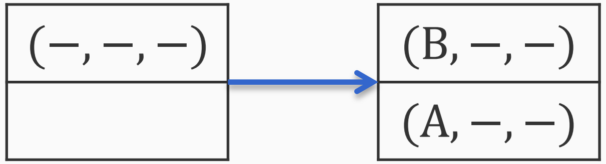
Score the newly created states

The top k new states form the new beam (sorted)

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2



$(-, -, -)$

$(B, -, -)$

$(A, -, -)$

Expand all the states in the beam

Score the newly created states

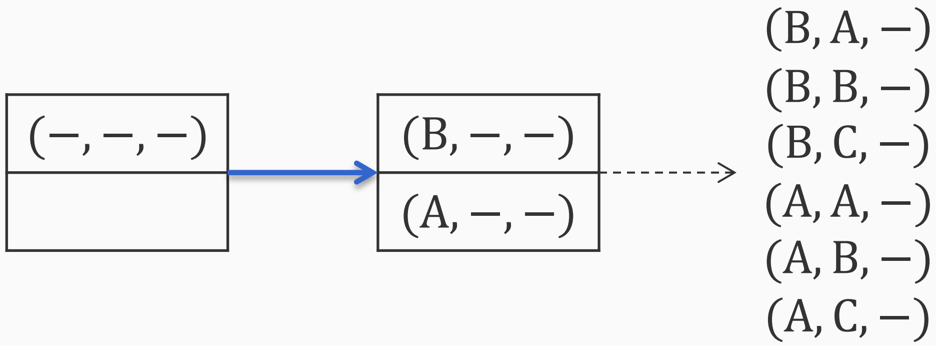
The top k new states form the new beam (sorted)

Now we are ready for the next step

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2



$(-,-,-)$

$(\mathrm{B},-,-)$
$(\mathrm{A},-,-)$

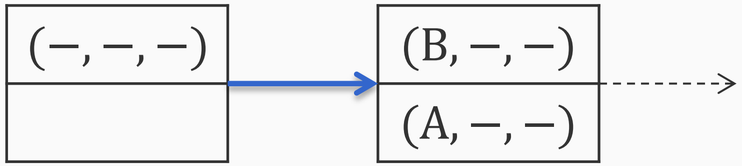
$(\mathrm{B},\mathrm{A},-)$
$(\mathrm{B},\mathrm{B},-)$
$(\mathrm{B},\mathrm{C},-)$
$(\mathrm{A},\mathrm{A},-)$
$(\mathrm{A},\mathrm{B},-)$
$(\mathrm{A},\mathrm{C},-)$

Expand all the states in the beam

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2



$(-,-,-)$

$(B,-,-)$

$(A,-,-)$

$(B, A, -)$	0.1
$(B, B, -)$	-3
$(B, C, -)$	10
$(A, A, -)$	20
$(A, B, -)$	-1
$(A, C, -)$	4.1

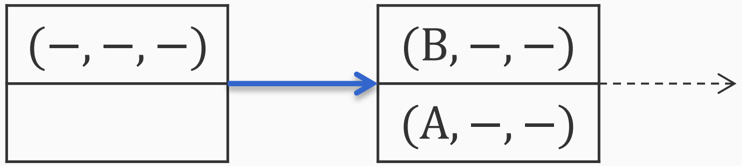
Expand all the states in the beam

Score the newly created states

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2



$(\mathrm{B}, \mathrm{A}, -)$	0.1
$(\mathrm{B}, \mathrm{B}, -)$	-3
$(\mathrm{B}, \mathrm{C}, -)$	10
$(\mathrm{A}, \mathrm{A}, -)$	20
$(\mathrm{A}, \mathrm{B}, -)$	-1
$(\mathrm{A}, \mathrm{C}, -)$	4.1

Expand all the states in the beam

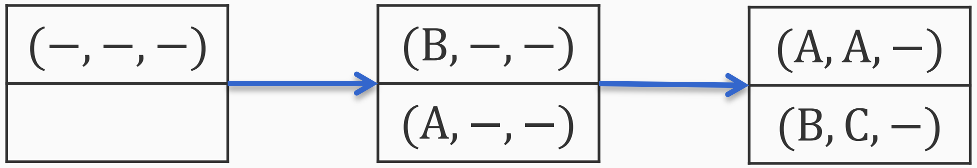
Score the newly created states

The top k new states form the new beam (sorted)

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2



$(-,-,-)$

$(B,-,-)$

$(A,-,-)$

$(A,A,-)$

$(B,C,-)$

Expand all the states in the beam

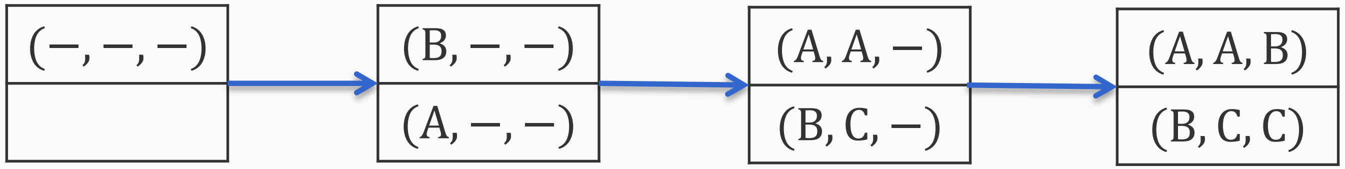
Score the newly created states

The top k new states form the new beam (sorted)

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2

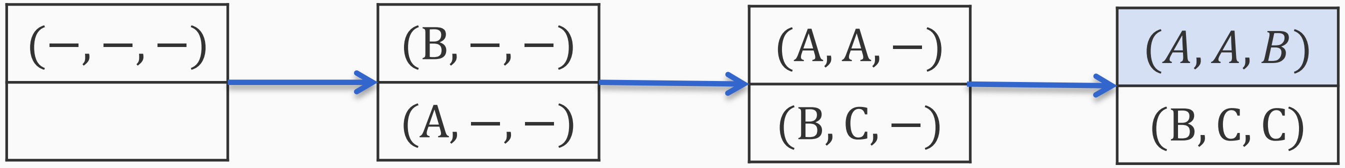


$(-,-,-)$	$(B,-,-)$	$(A,A,-)$	(A,A,B)
	$(A,-,-)$	$(B,C,-)$	(B,C,C)

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2



$(-,-,-)$

$(\mathrm{B},-,-)$ $(\mathrm{A},-,-)$

$(\mathrm{A},\mathrm{A},-)$ $(\mathrm{B},\mathrm{C},-)$

(A,A,B) $(\mathrm{B},\mathrm{C},\mathrm{C})$

Final answer: Top of the beam at the end of search

Beam Search

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Pros

- Explores more than greedy search. Greedy search is beam search with beam size 1
- In general, easy to implement, very popular
- We get a set of sequences that we can then re-order or use in other ways

Beam Search

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Pros

- Explores more than greedy search. Greedy search is beam search with beam size 1
- In general, easy to implement, very popular
- We get a set of sequences that we can then re-order or use in other ways

Cons

- A good state might fall out of the beam
- Can be still repetitive. Possible solution: add an n-gram penalty to penalize n-grams that get repeated
- Generated text may be *boring* for a reader

  Do we always choose the most probable next words? What makes a sequences of words interesting?

Different decoding strategies exist

Search based decoding

Deterministic approaches that involves searching the space of sequences to select one

- Greedy decoding
- Beam search

Sampling based decoding

Randomized approaches that involve sampling from the token conditional probability distribution

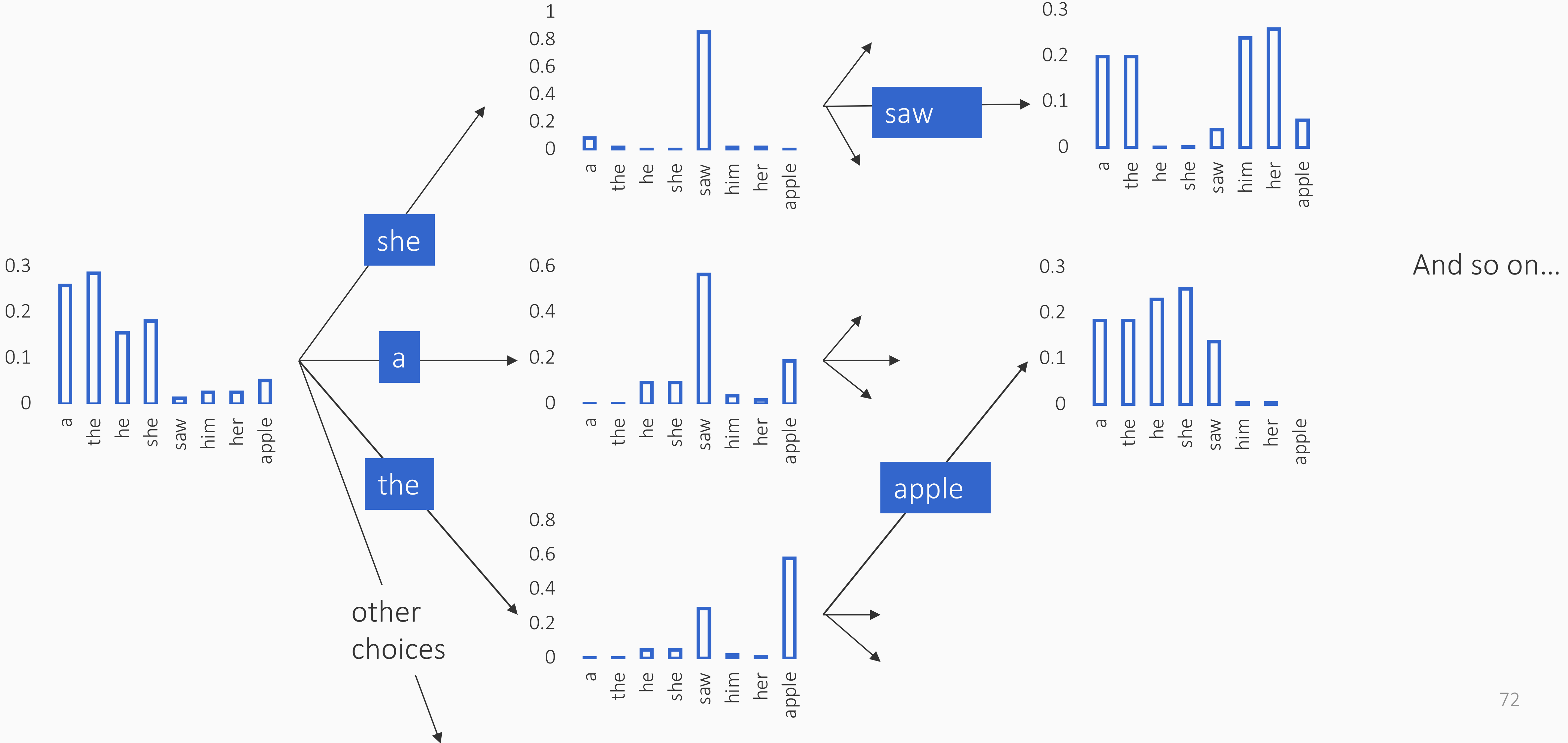
- Random sampling
- Top-K sampling
- Nucleus sampling

Sampling based approaches

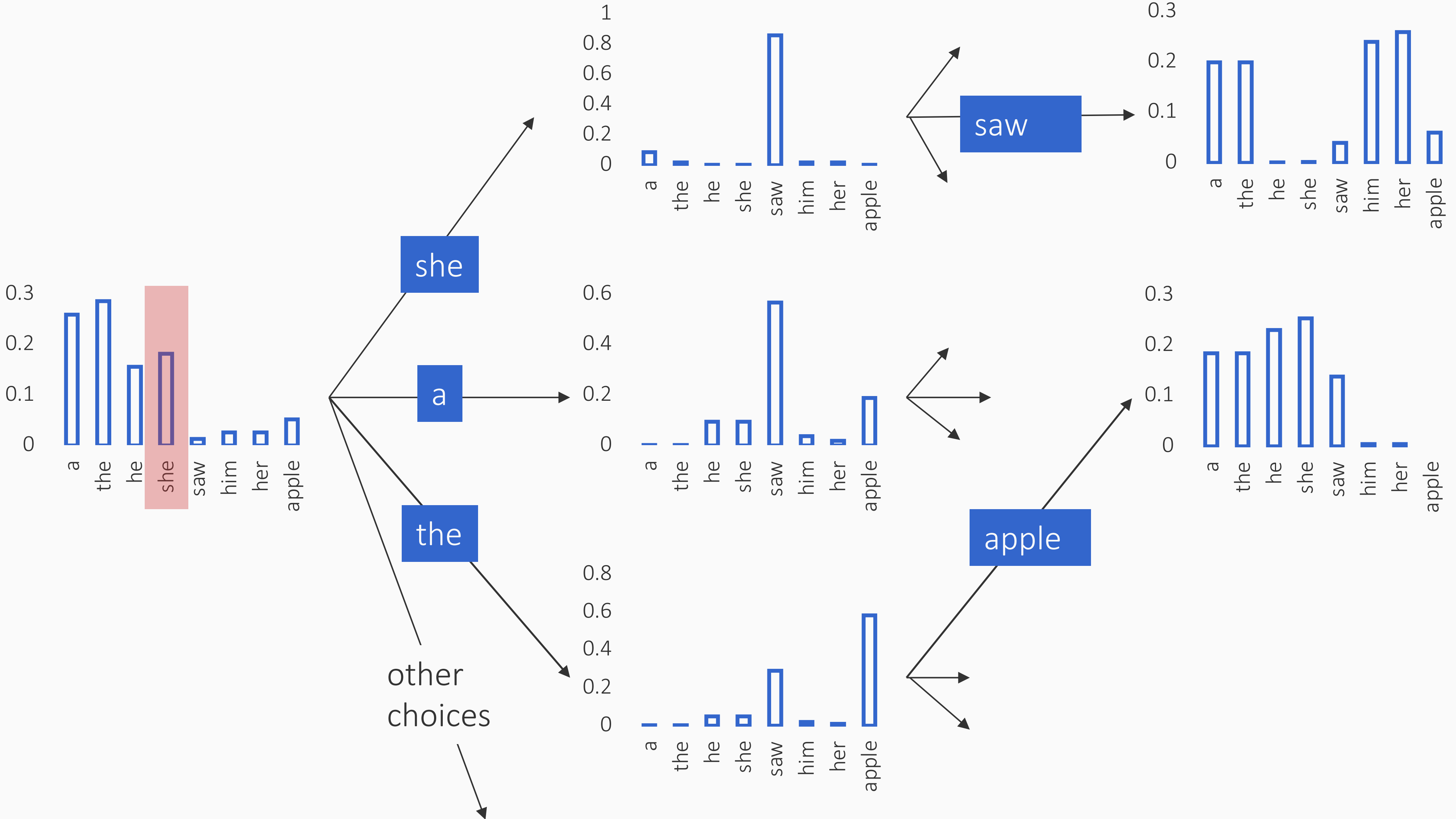
Rather than picking the most probable next token, randomly pick one using the next token distribution

$$w_n \sim P\left(v \mid w_0 w_1 \cdots w_{\{n-1\}} \right)$$

Random sampling in our toy setting

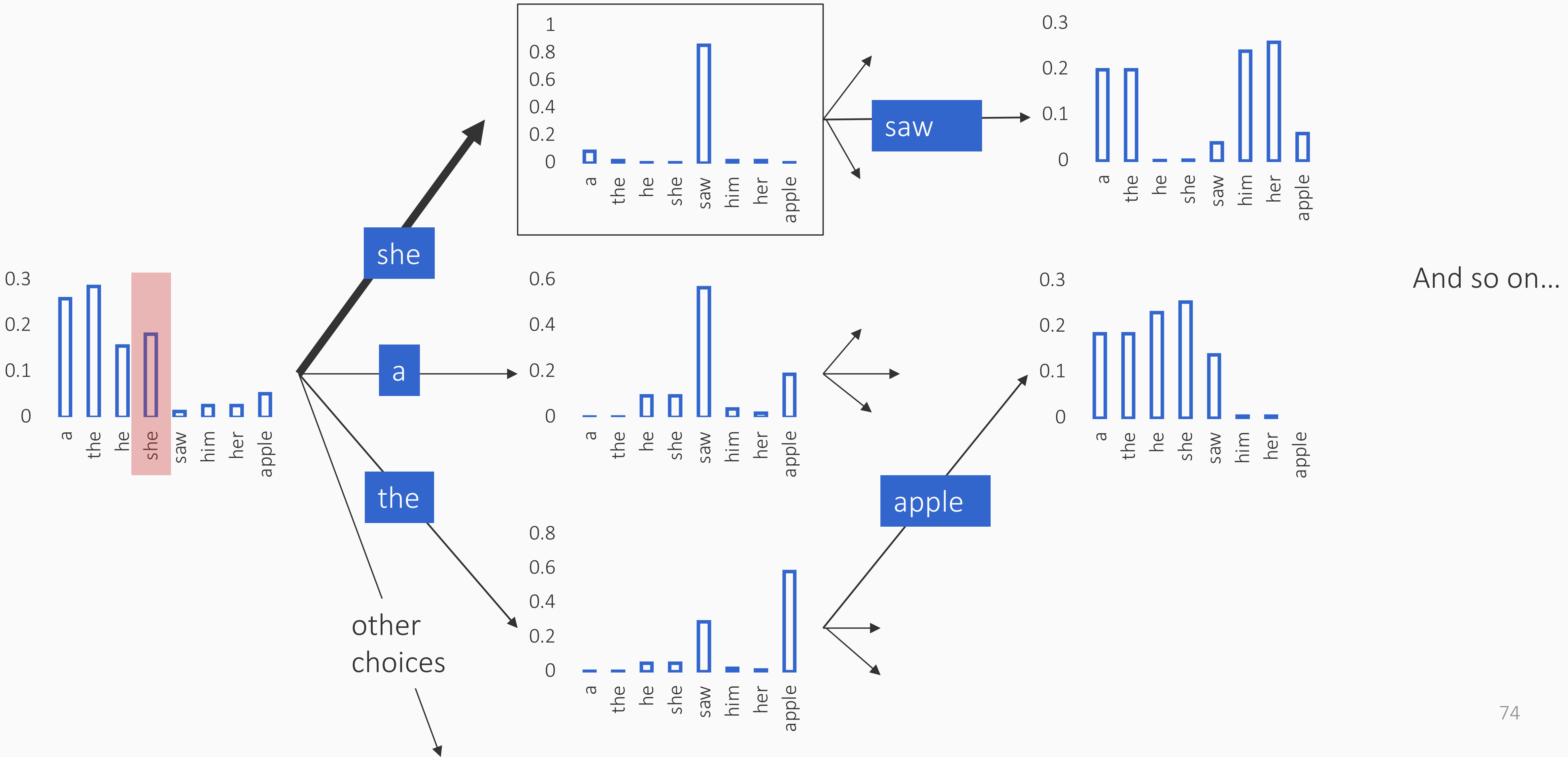


Random sampling in our toy setting

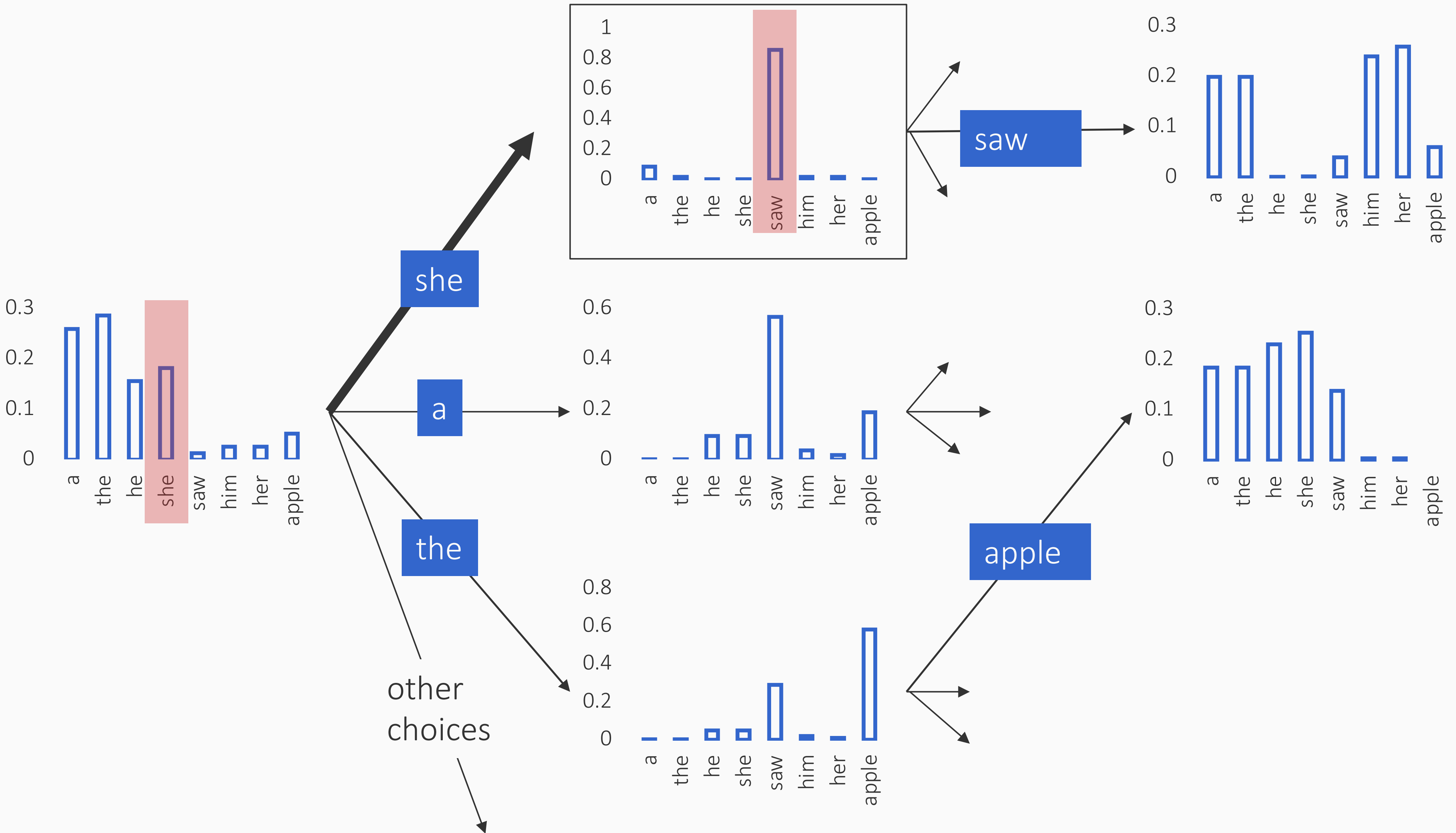


And so on…

Random sampling in our toy setting

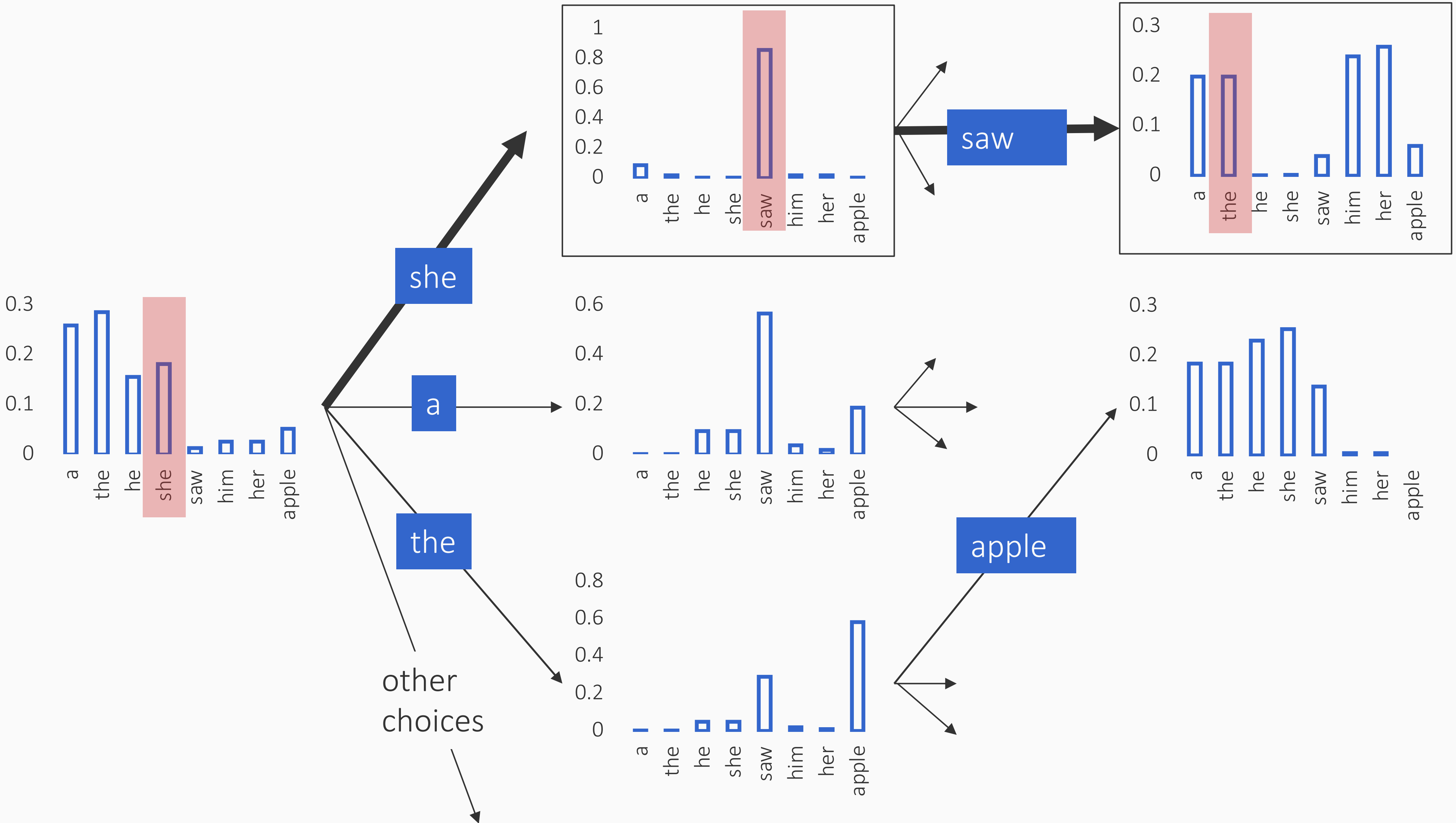


Random sampling in our toy setting



And so on…

Random sampling in our toy setting



And so on…

Sampling based approaches

Rather than picking the most probable next token, randomly pick one using the next token distribution

$$w_n \sim P\left(v \mid w_0 w_1 \cdots w_{\{n-1\}} \right)$$

Pros

- Produces more interesting text
- Diverse outputs

Cons

- Does not produce coherent outputs. Why?

Sampling based approaches

Rather than picking the most probable next token, randomly pick one using the next token distribution

$$w_n \sim P\left(v \mid w_0 w_1 \cdots w_{\{n-1\}} \right)$$

Pros

- Produces more interesting text
- Diverse outputs

Cons

- Does not produce coherent outputs. Why?
  A solution: Use a temperature term in the softmax to make the probabilities “peaky”

Sampling based approaches

Rather than picking the most probable next token, randomly pick one using the next token distribution

$$w_n \sim P\left(v \mid w_0 w_1 \cdots w_{\{n-1\}} \right)$$

Pros

- Produces more interesting text
- Diverse outputs

Cons

- Does not produce coherent outputs. Why?
  A solution: Use a temperature term in the softmax to make the probabilities “peaky”

Sampling based approaches

Rather than picking the most probable next token, randomly pick one using the next token distribution

$$w_n \sim P\left(v \mid w_0 w_1 \cdots w_{\{n-1\}} \right)$$

Pros

- Produces more interesting text
- Diverse outputs

Cons

- Does not produce coherent outputs. Why?
  A solution: Use a *temperature* term in the softmax to make the probabilities "peaky"

$$P(token_i|context) = \frac{\exp\left(\frac{s_i}{T}\right)}{\sum_j \exp\left(\frac{s_j}{T}\right)}$$

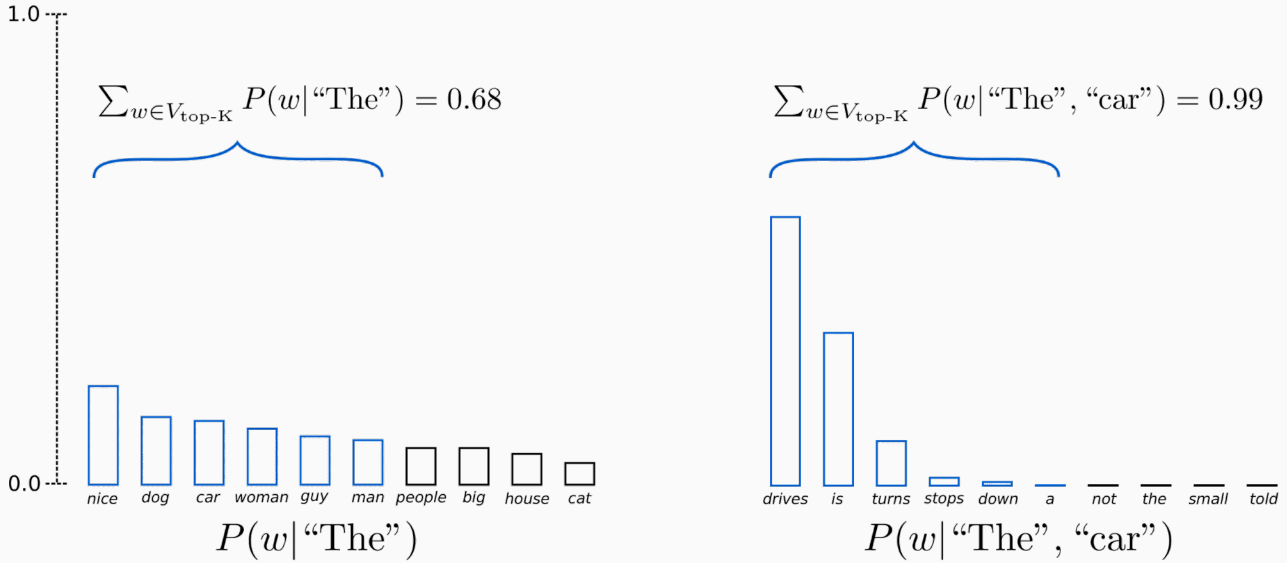
When T is lower than 1, probabilities get more sharp and lower probabilities get diminished

When T = 0, all the probability is placed on the token with the highest s_i → Greedy decoding

Other variants of sampling

Top-K sampling

- Instead of sampling from all the words, pick the K most probable words, and distribute the probability mass among them
- GPT2 used this



Other variants of sampling

Top-K sampling

- Instead of sampling from all the words, pick the K most probable words, and distribute the probability mass among them
- GPT2 used this

Nucleus sampling (sometimes called Top-p sampling)

- Instead of choosing a fixed K, choose as many words as necessary so that the total probability allocated to them is at least p

Both methods seem to produce more fluent text than greedy and beam search

This lecture: Decoding algorithms

How to predict a sequence

- The setup: An abstract auto-regressive model
- Decoding algorithms
  - Greedy decoding
  - Beam search
  - Random sampling
  - Top-K sampling
  - Nucleus sampling