Deep learning for text processing

STAT 4710

November 28, 2023

Where we are

- **Unit 1:** R for data mining
- **Unit 2:** Prediction fundamentals
- **Unit 3:** Regression-based methods
- **Unit 4:** Tree-based methods
- **Unit 5:** Deep learning

Lecture 1: Deep learning preliminaries

Lecture 2: Neural networks

Lecture 3: Deep learning for images

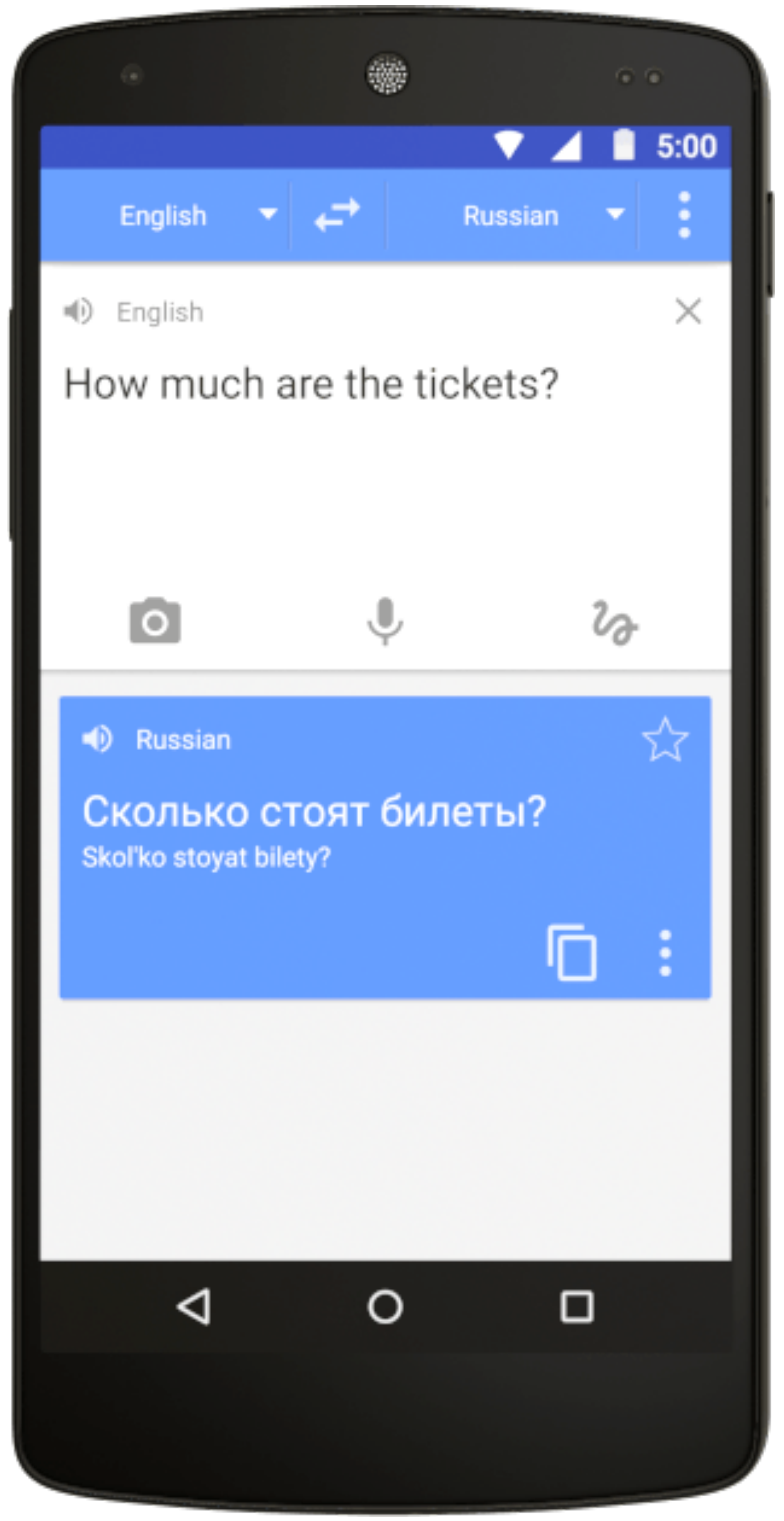
$\boxed{\textbf{Lecture 4:}\text{ Deep learning for text}}$

Lecture 5: Unit review and quiz in class

Applications of natural language processing

Applications of natural language processing

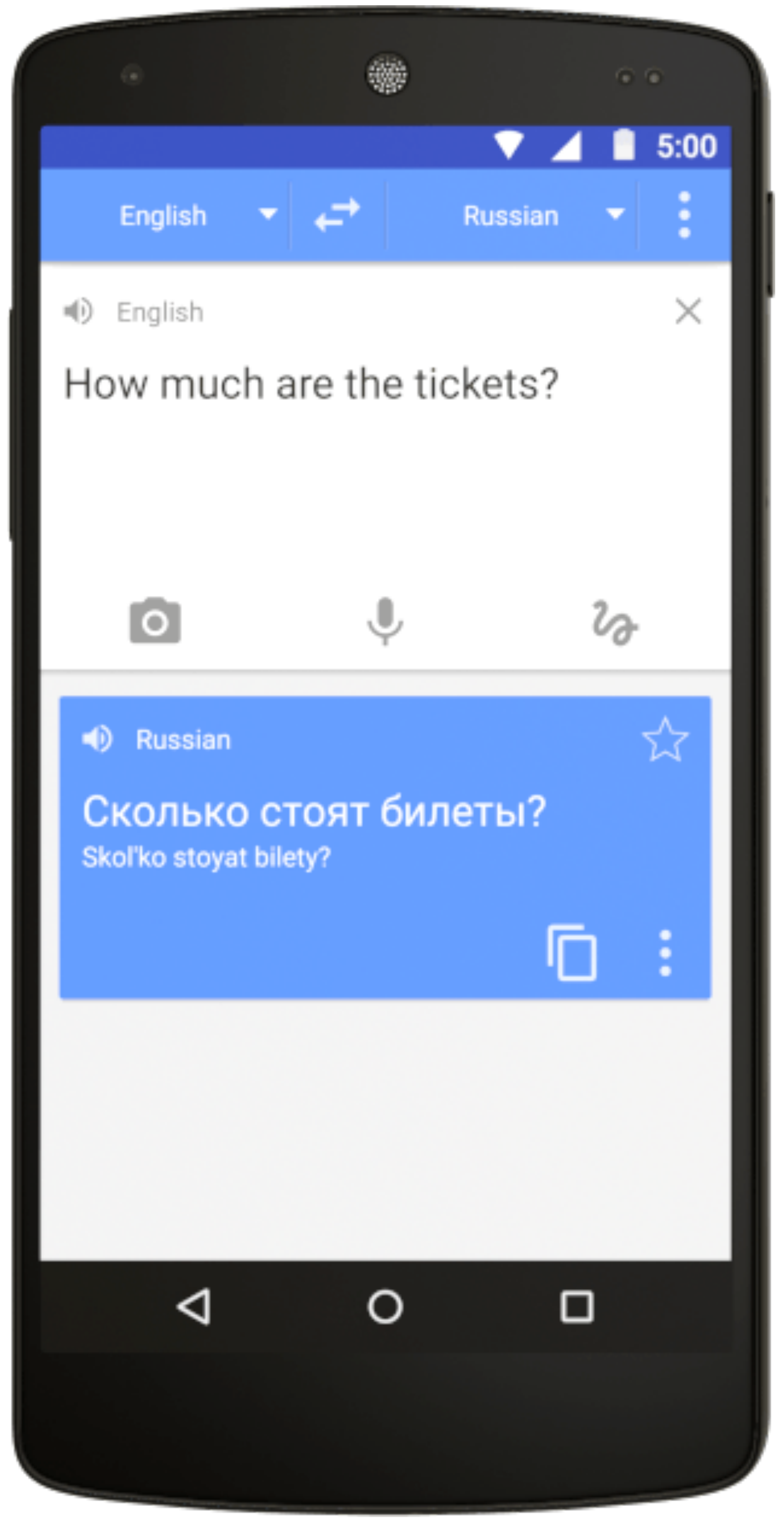
Machine translation



https://translate.google.com/intl/en/about/

Applications of natural language processing

Machine translation

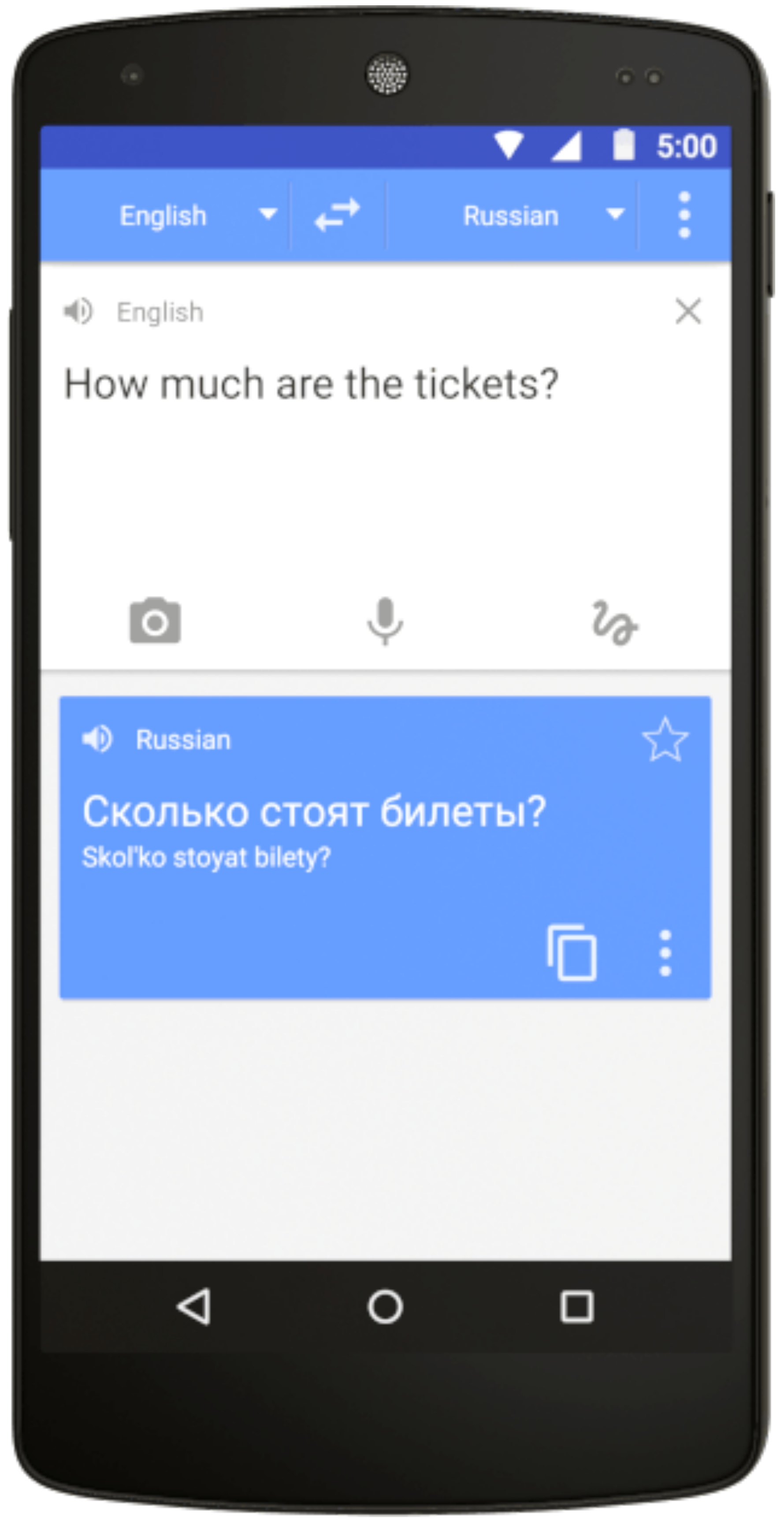


https://translate.google.com/intl/en/about/

Chatbots

Applications of natural language processing

Machine translation



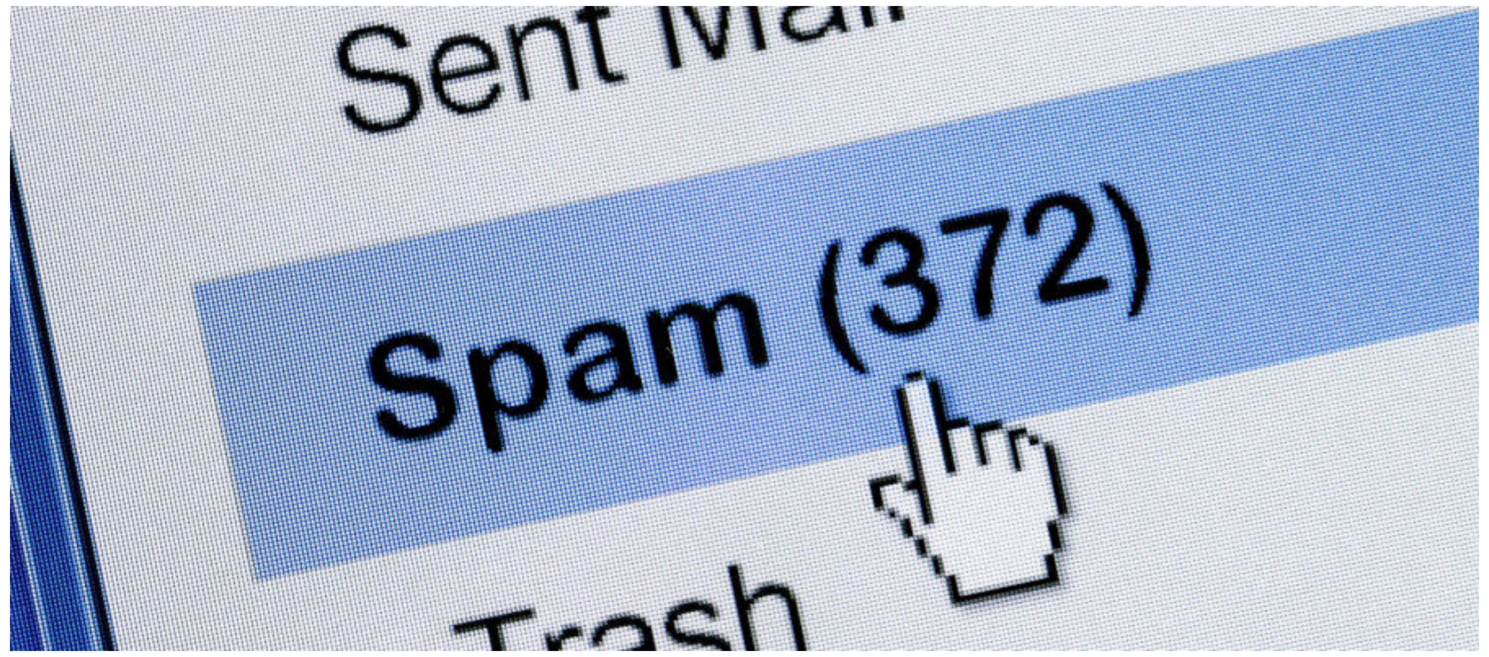
https://translate.google.com/intl/en/about/

Chatbots

ChatGPT

Bard

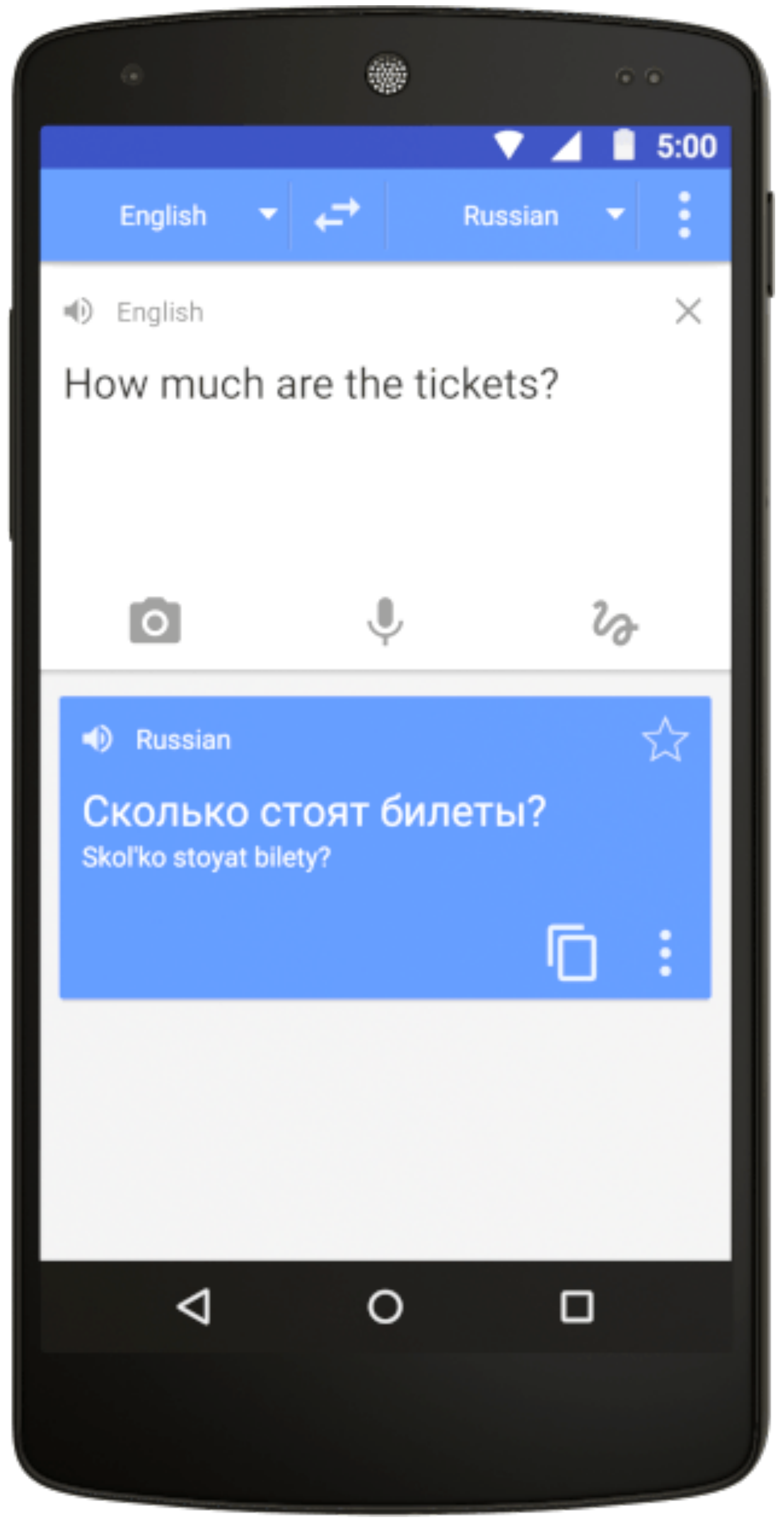
Spam filtering



https://blog.malwarebytes.com/security-world/2017/02/explained-bayesian-spam-filtering/

Applications of natural language processing

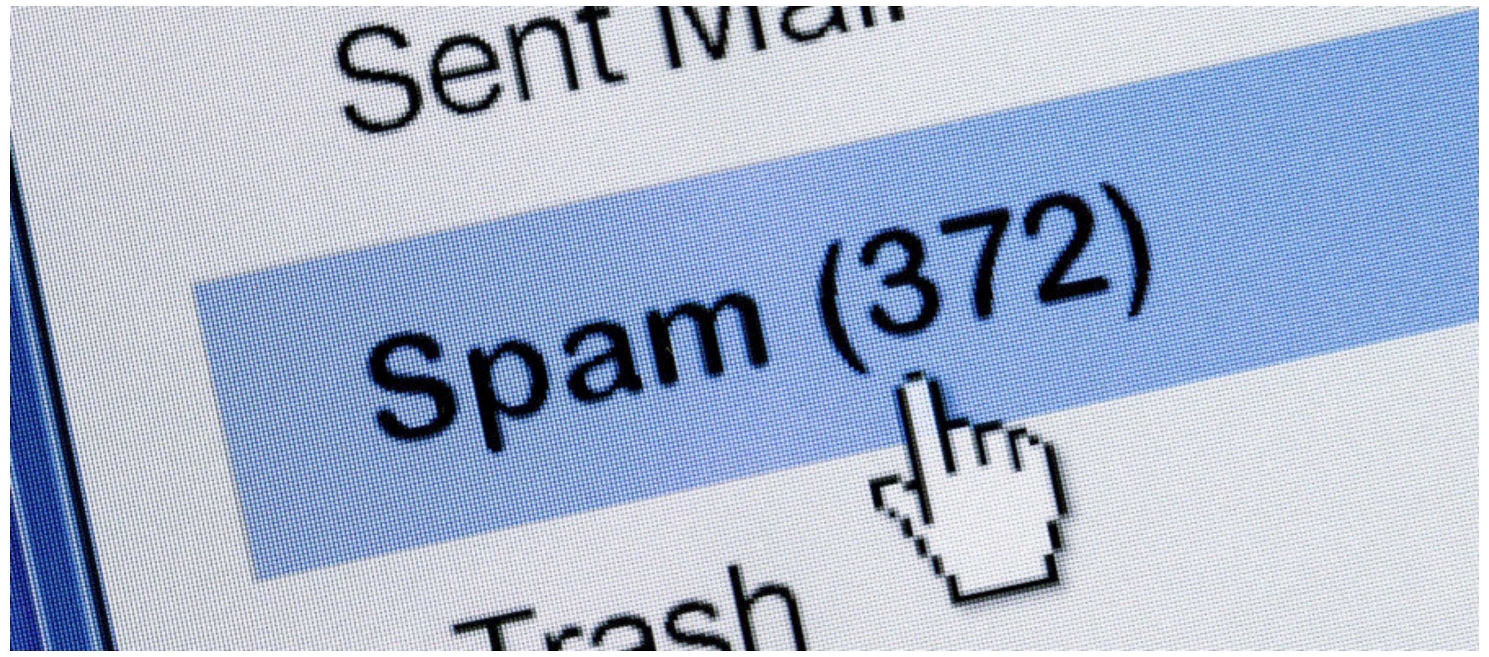
Machine translation



https://translate.google.com/intl/en/about/

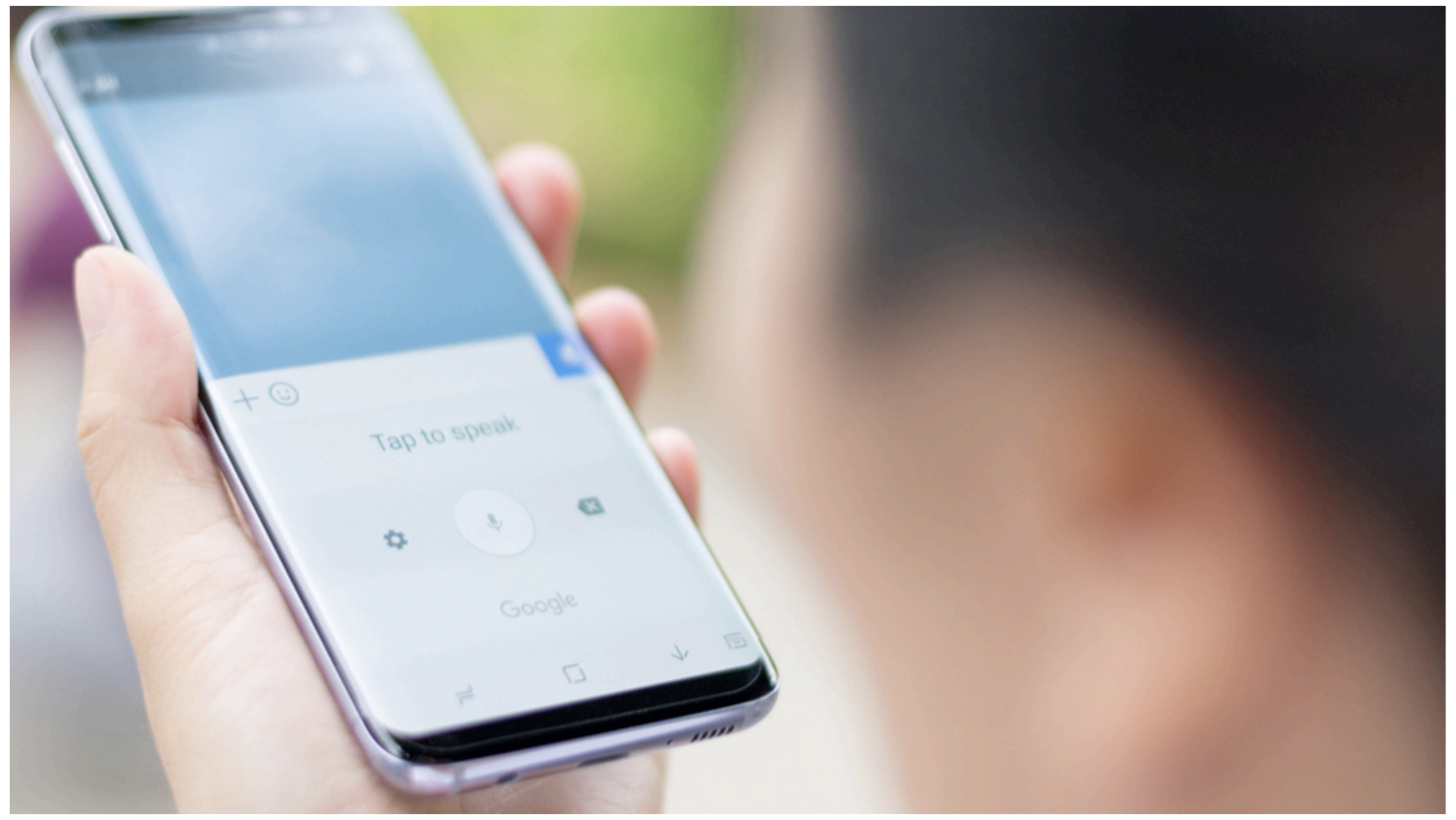
Chatbots

Spam filtering



https://blog.malwarebytes.com/security-world/2017/02/explained-bayesian-spam-filtering/

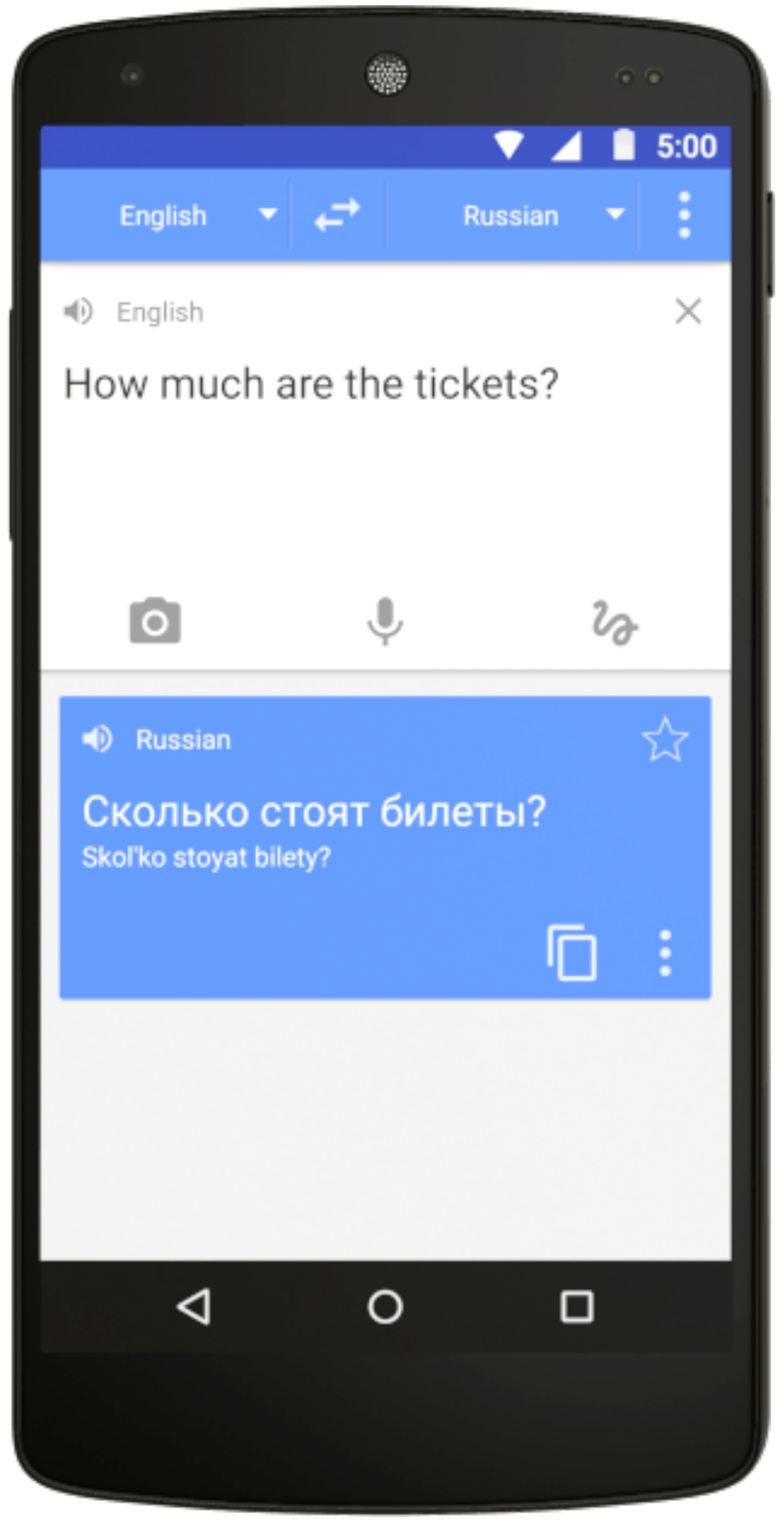
Voice to text



https://www.itproportal.com/guides/how-to-turn-off-ok-google-android-voice-search/

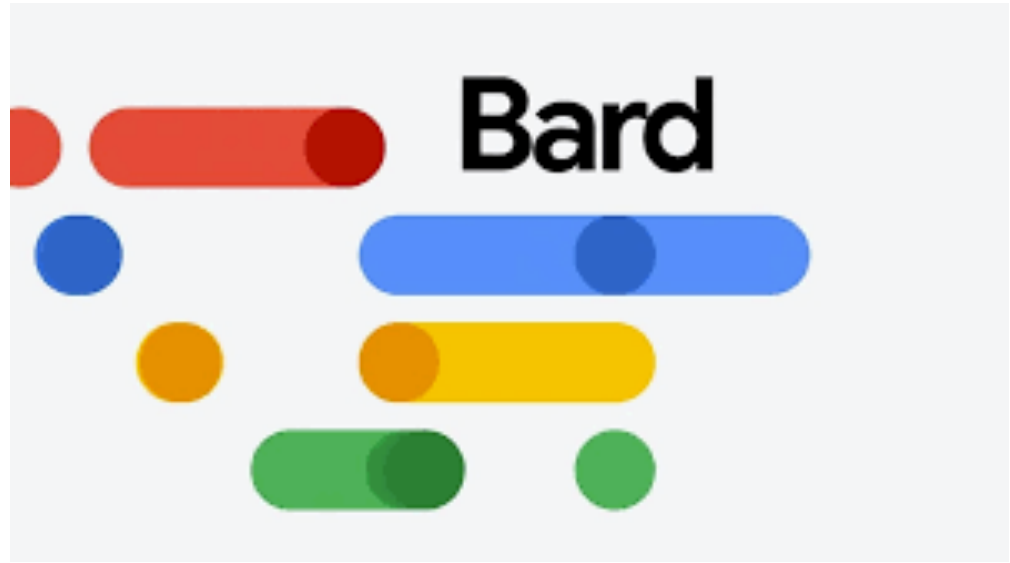
Applications of natural language processing

Machine translation

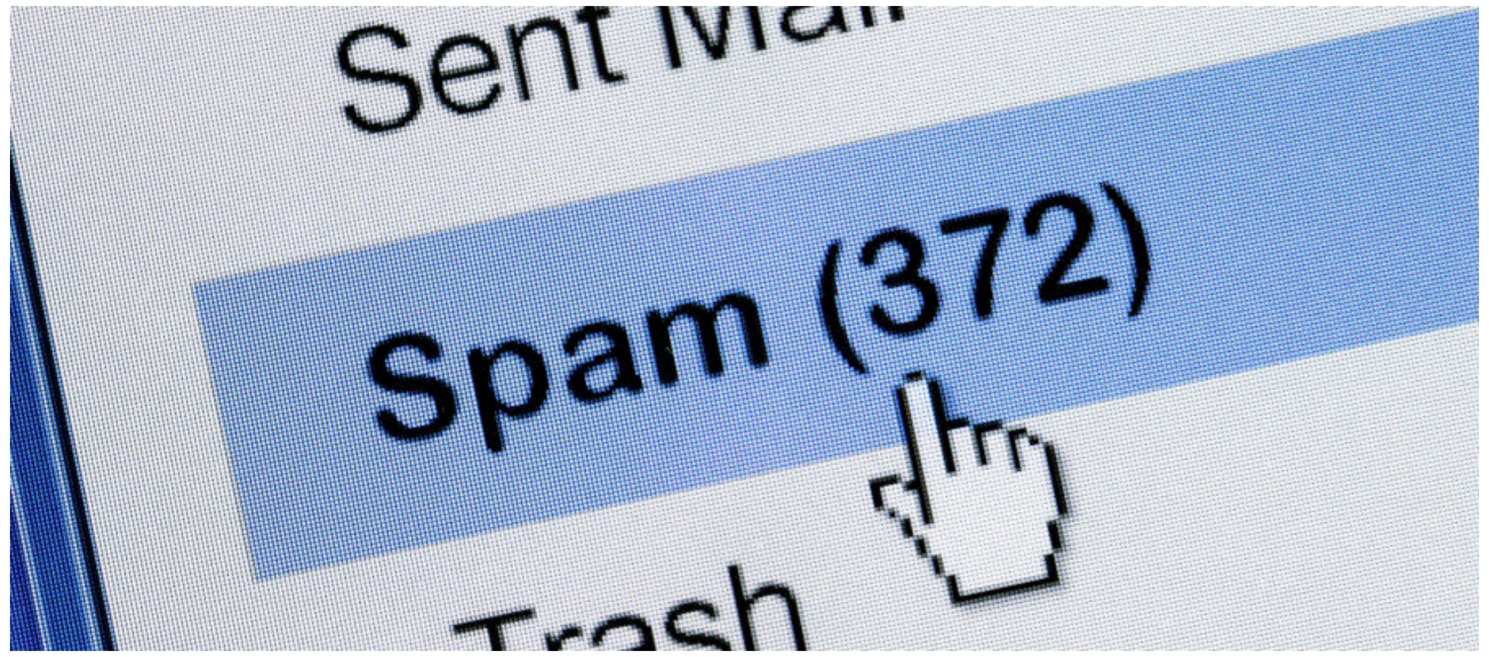


https://translate.google.com/intl/en/about/

Chatbots

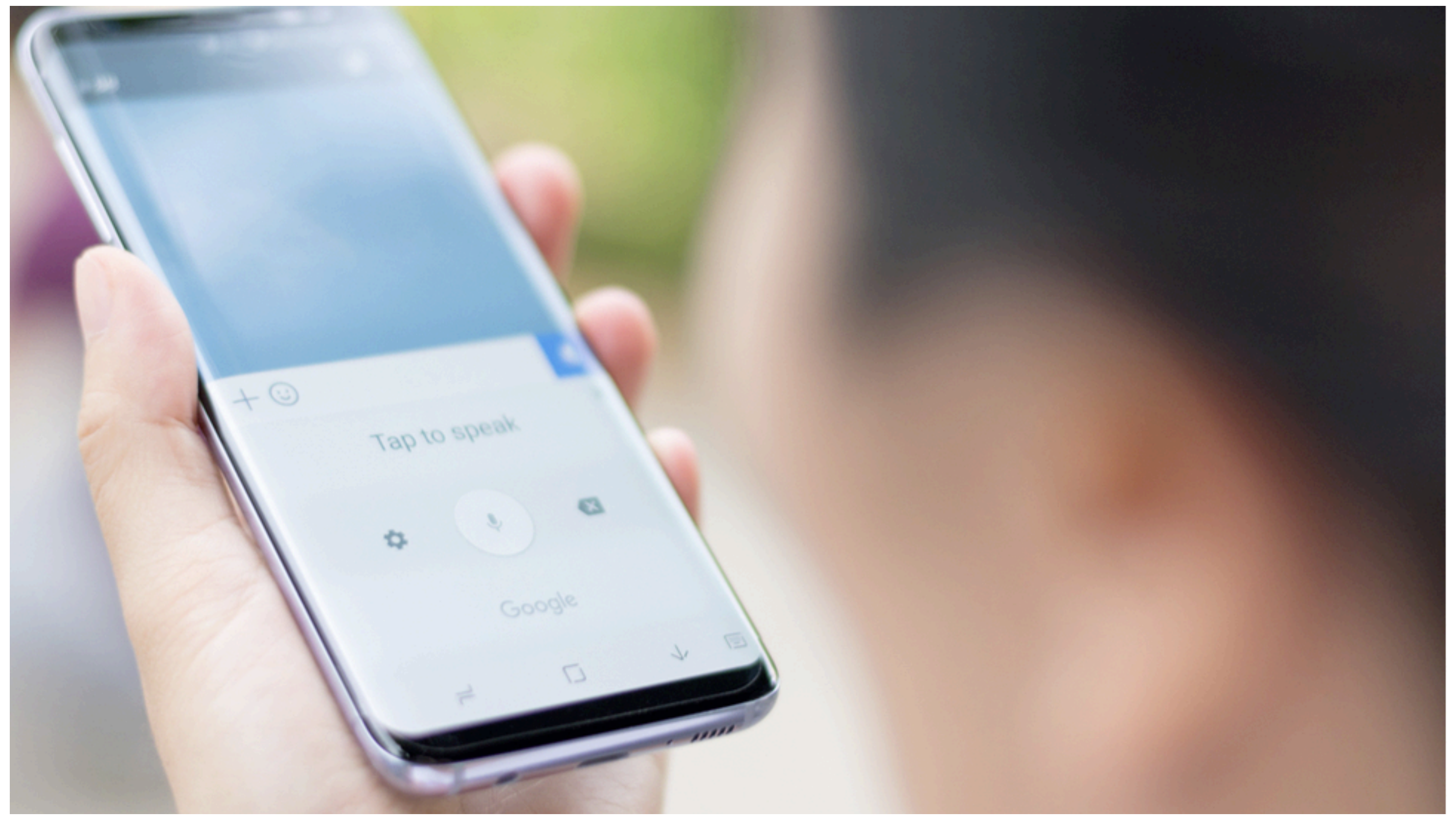


Spam filtering



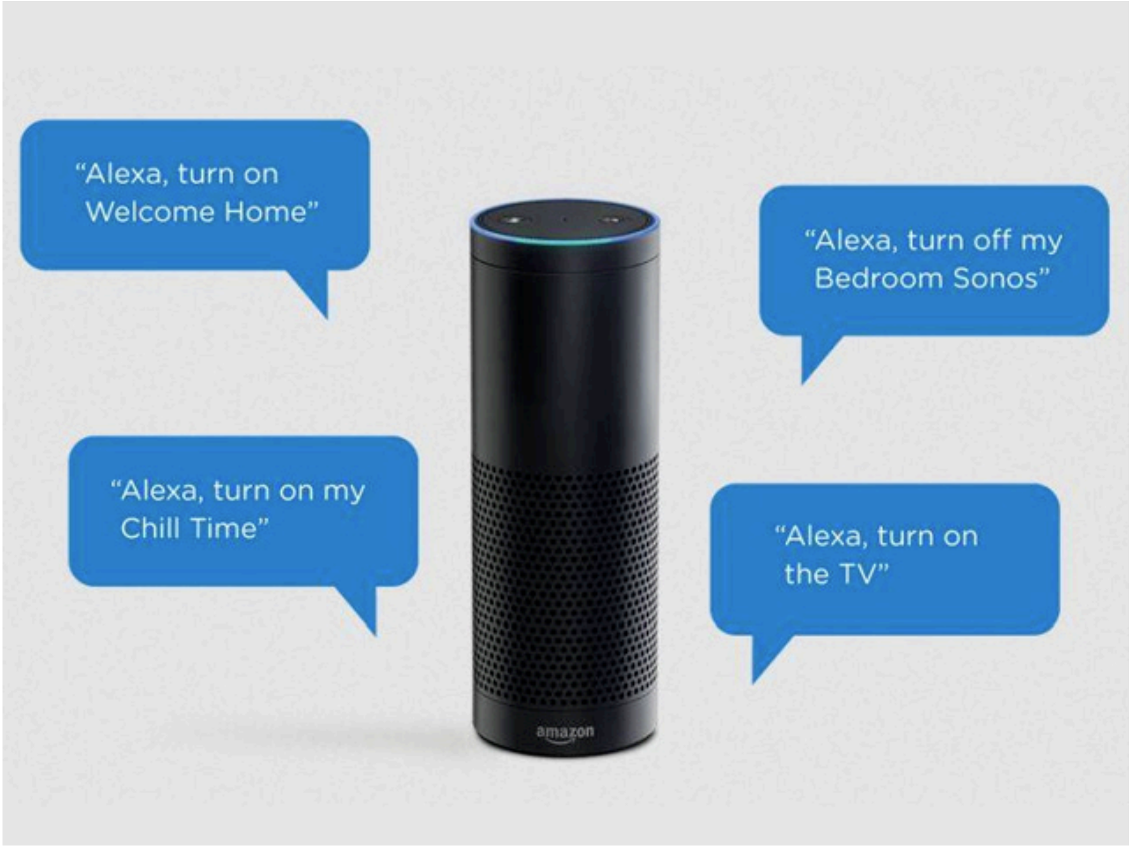
https://blog.malwarebytes.com/security-world/2017/02/explained-bayesian-spam-filtering/

Voice to text



https://www.itproportal.com/guides/how-to-turn-off-ok-google-android-voice-search/

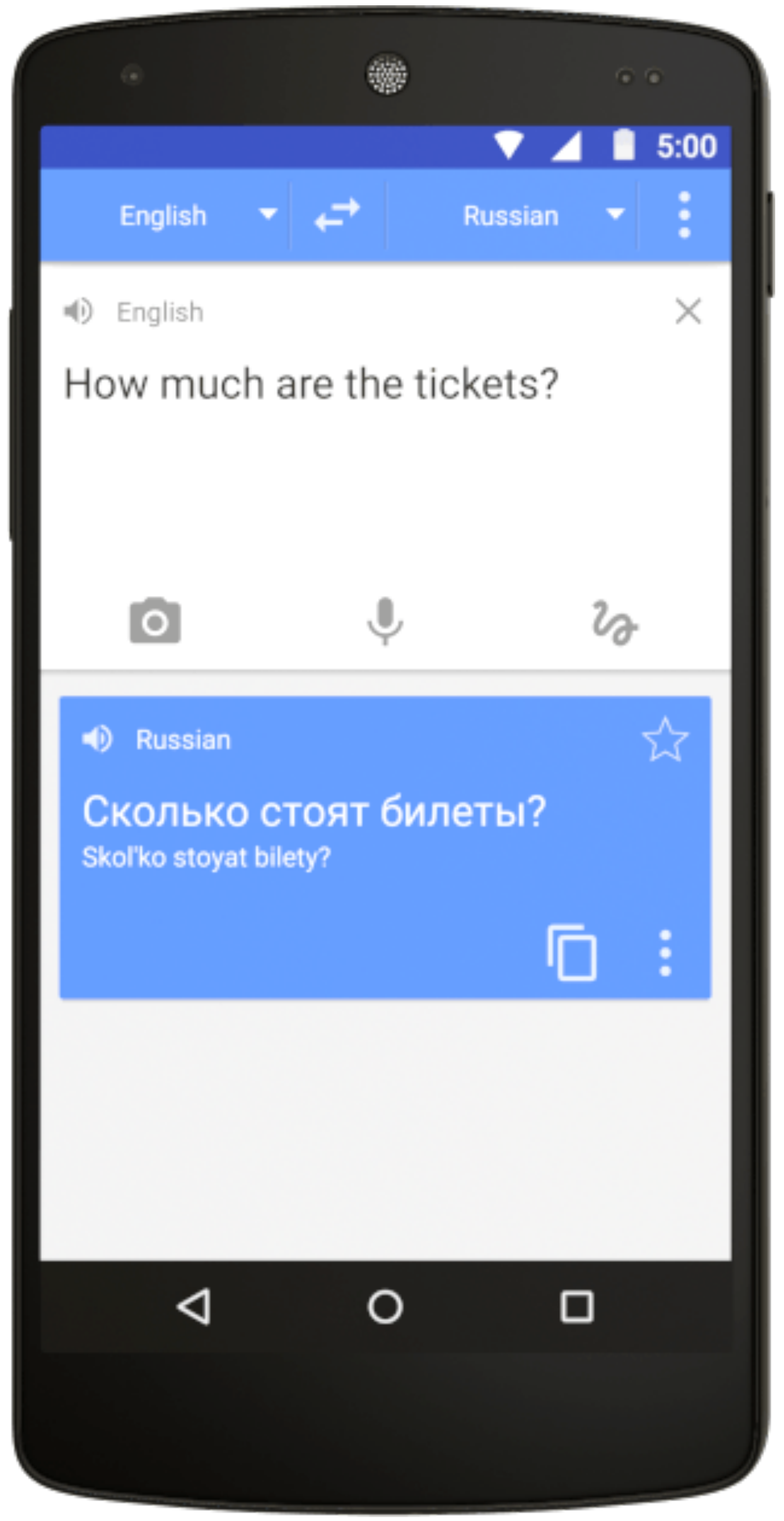
Personal assistant



https://brailleinstitute.org/event/online-introducing-amazon-alexa

Applications of natural language processing

Machine translation



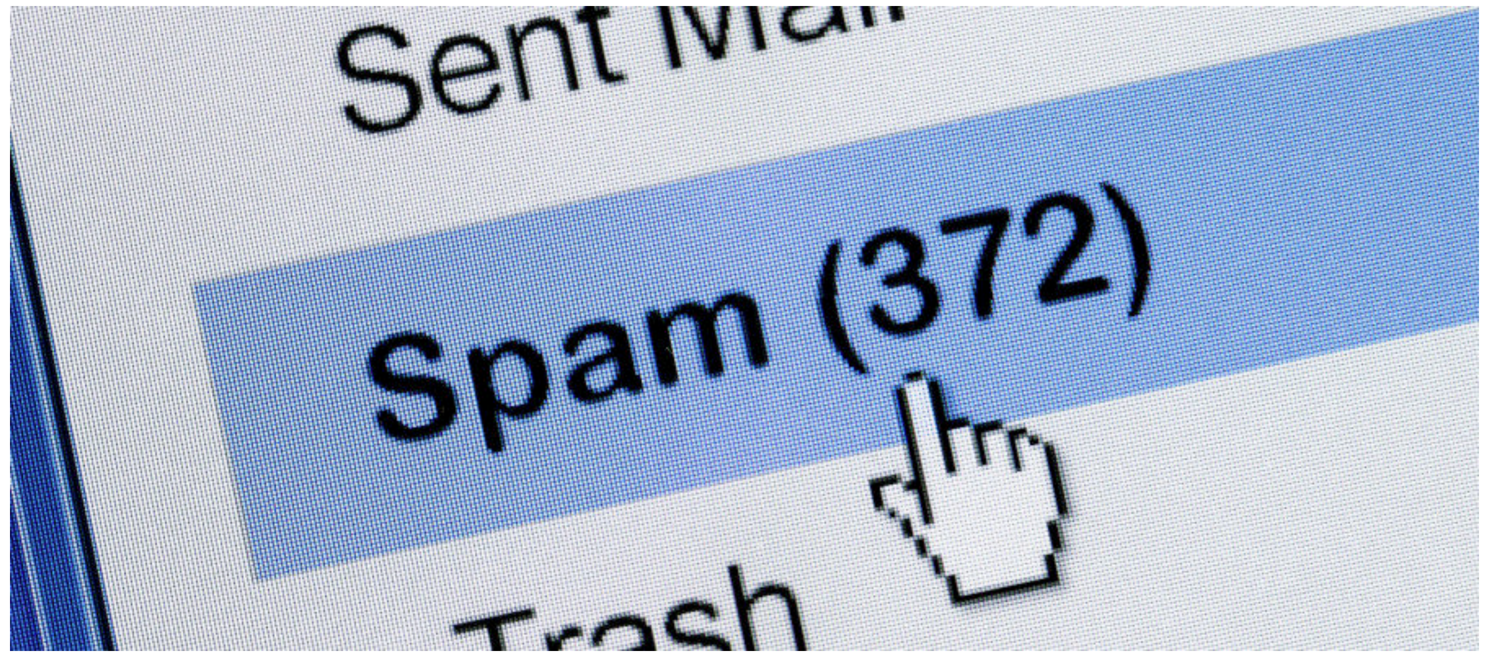
https://translate.google.com/intl/en/about/

Chatbots

ChatGPT

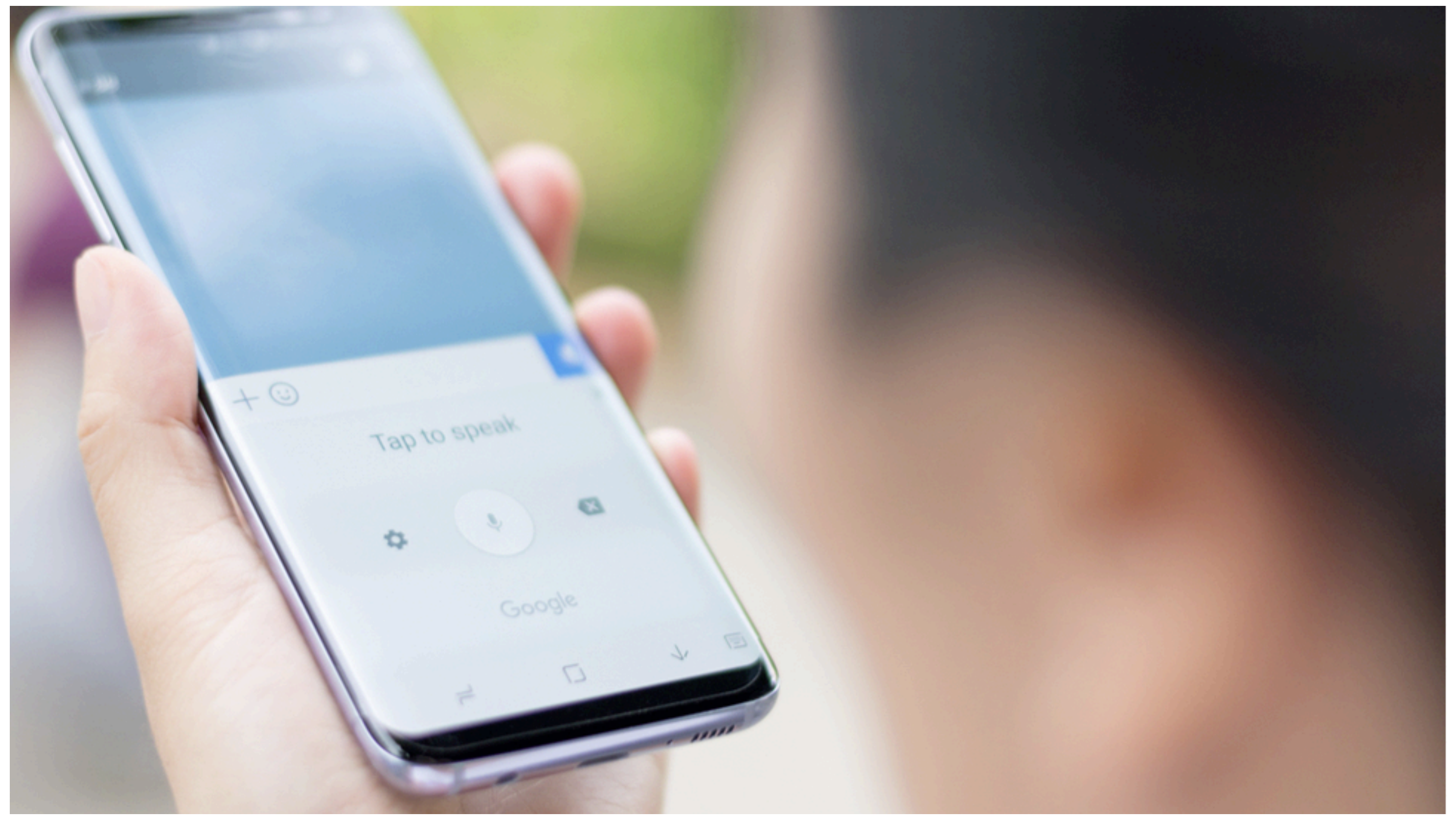
Bard

Spam filtering



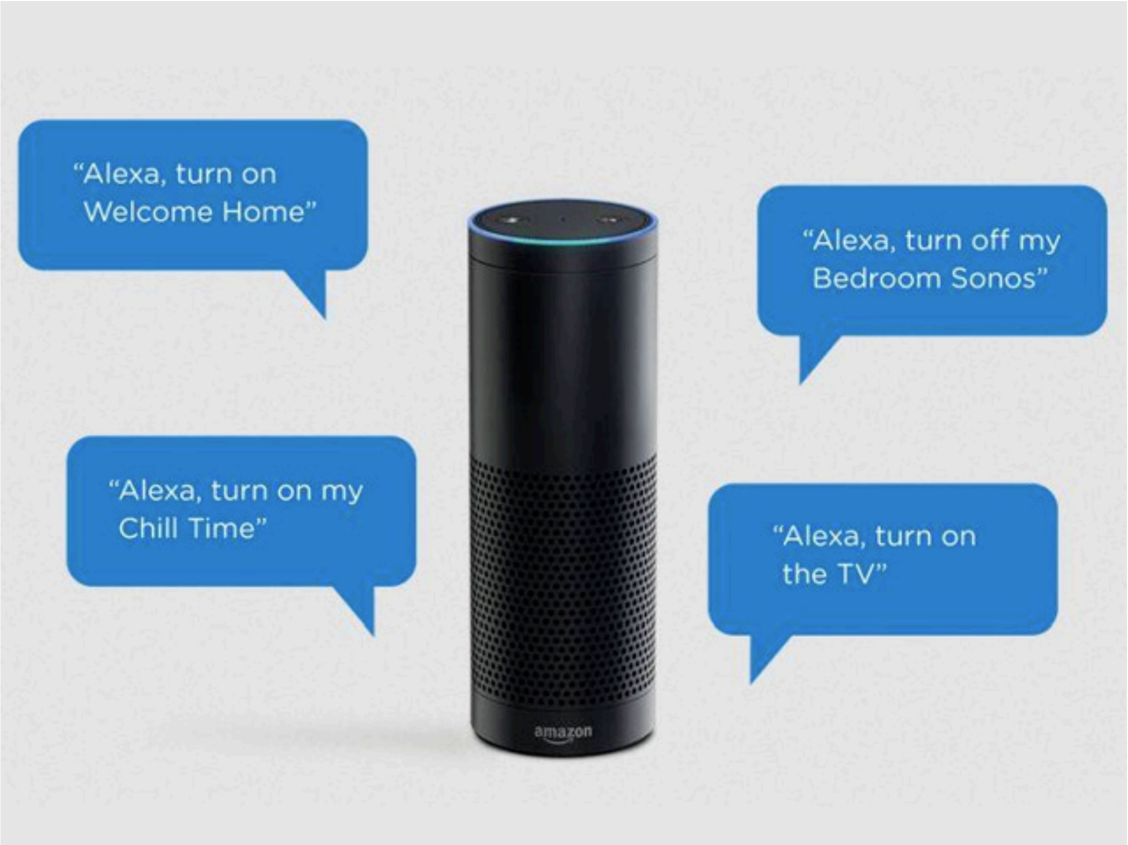
https://blog.malwarebytes.com/security-world/2017/02/explained-bayesian-spam-filtering/

Voice to text



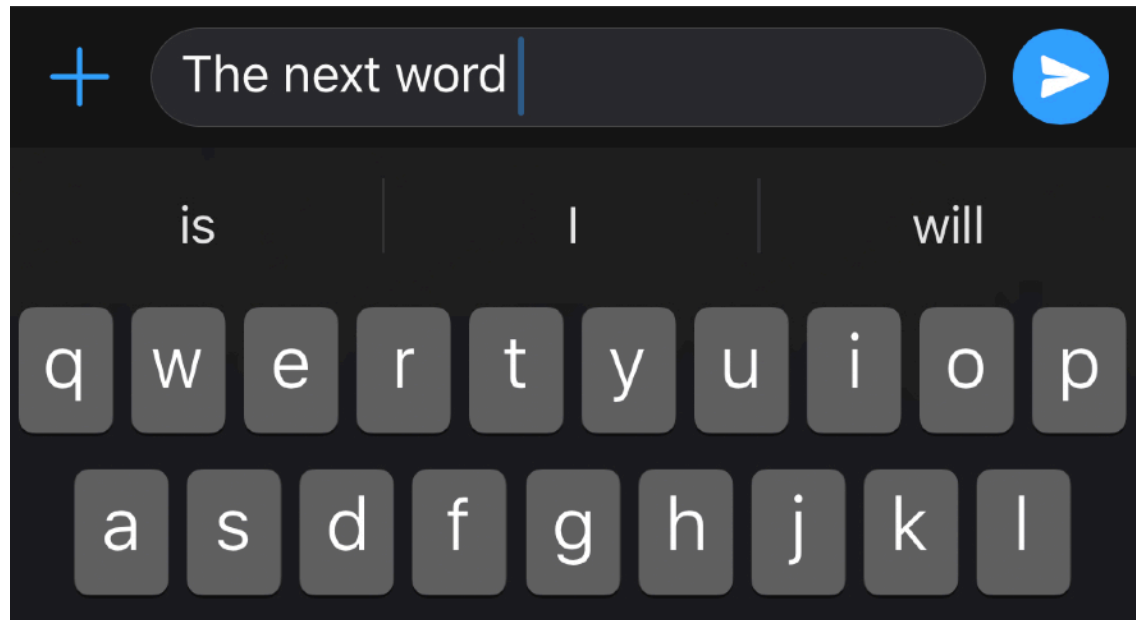
https://www.itproportal.com/guides/how-to-turn-off-ok-google-android-voice-search/

Personal assistant



https://brailleinstitute.org/event/online-introducing-amazon-alexa

Auto-complete



https://towardsdatascience.com/language-modeling-c1cf7b983685

Three natural language processing tasks

Three natural language processing tasks

Sentiment analysis

Three natural language processing tasks

Sentiment analysis

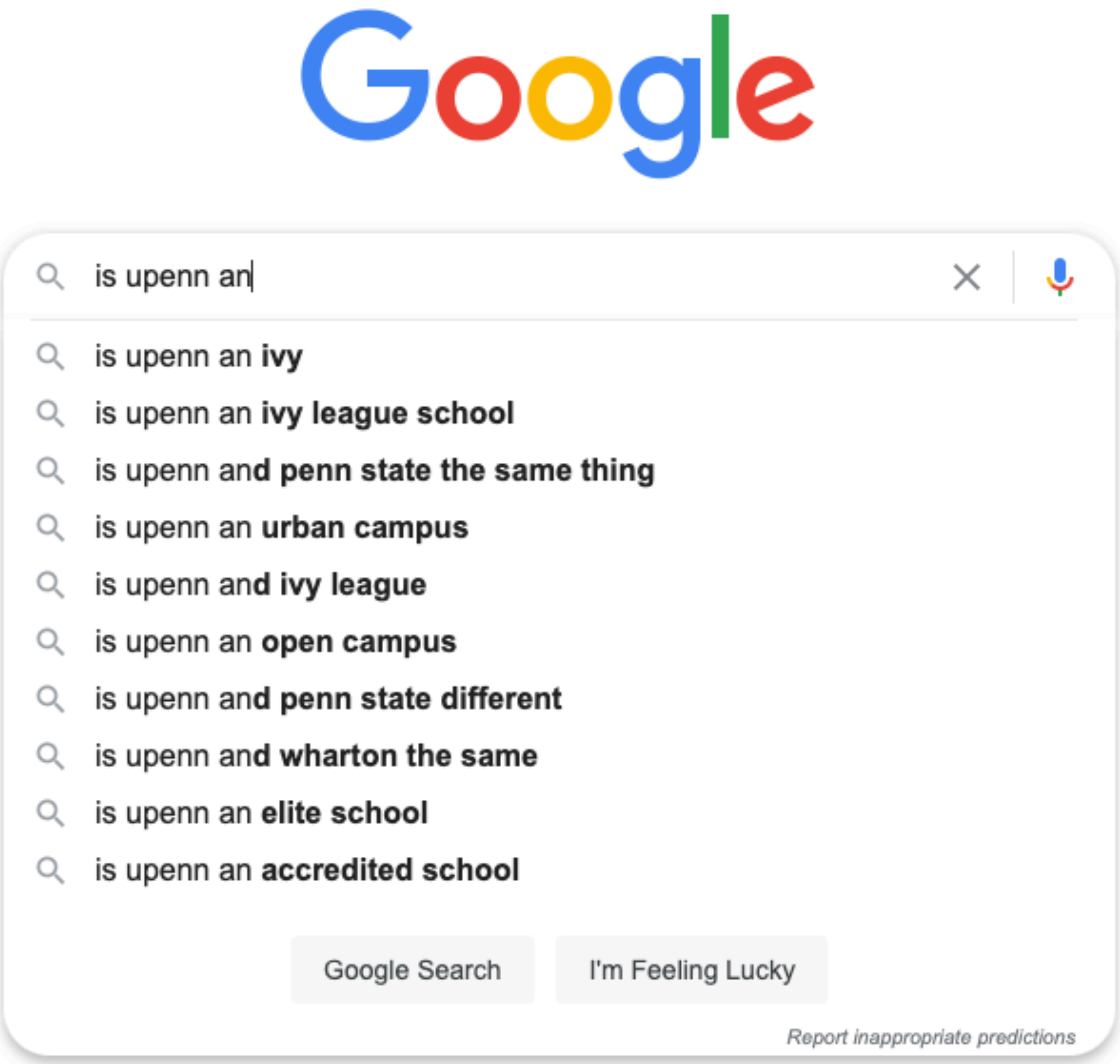
Input: Movie review
Output: Positive or negative

Three natural language processing tasks

Sentiment analysis

Input: Movie review
Output: Positive or negative

Language modeling

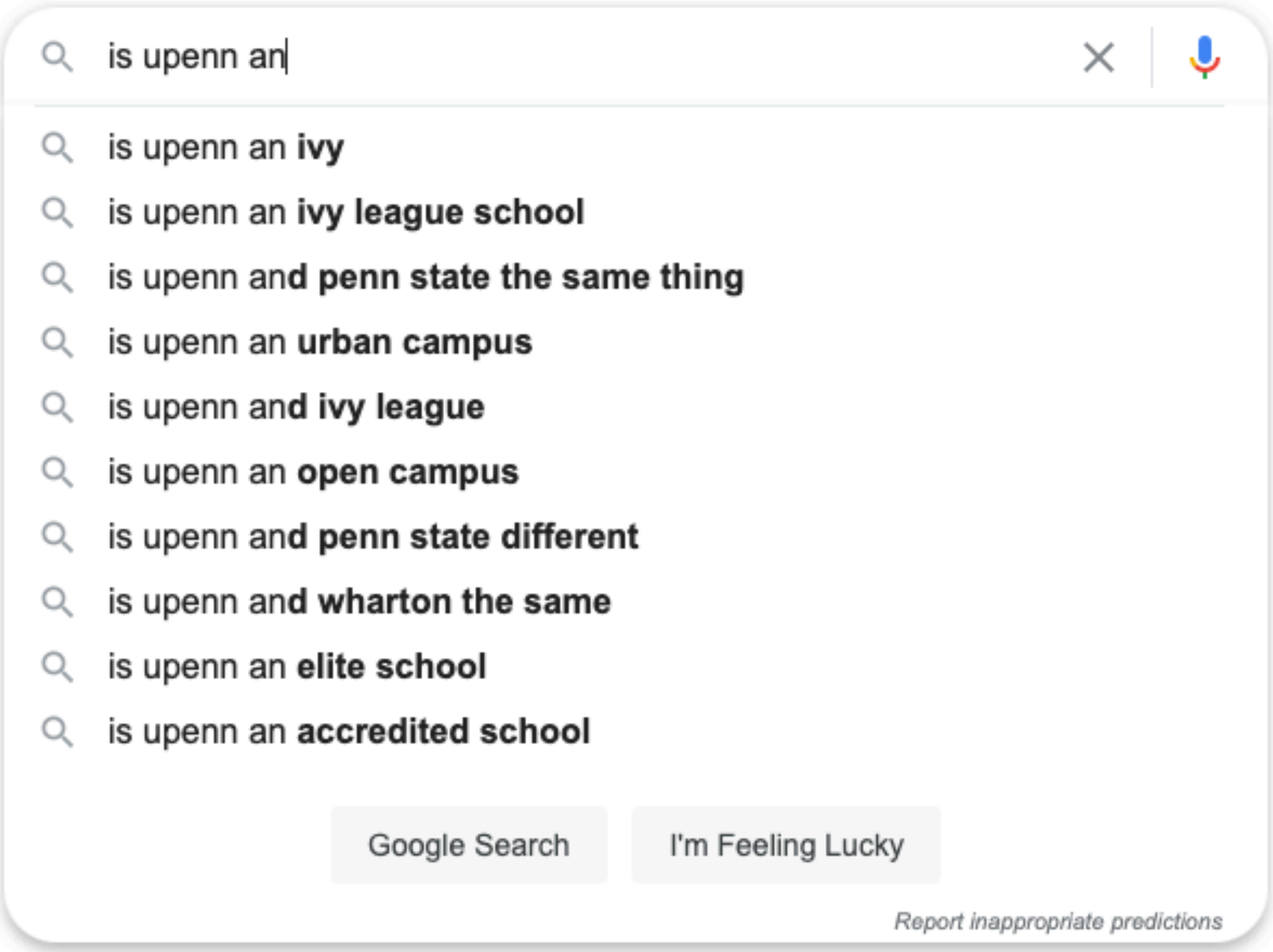


Three natural language processing tasks

Sentiment analysis

Input: Movie review
Output: Positive or negative

Language modeling



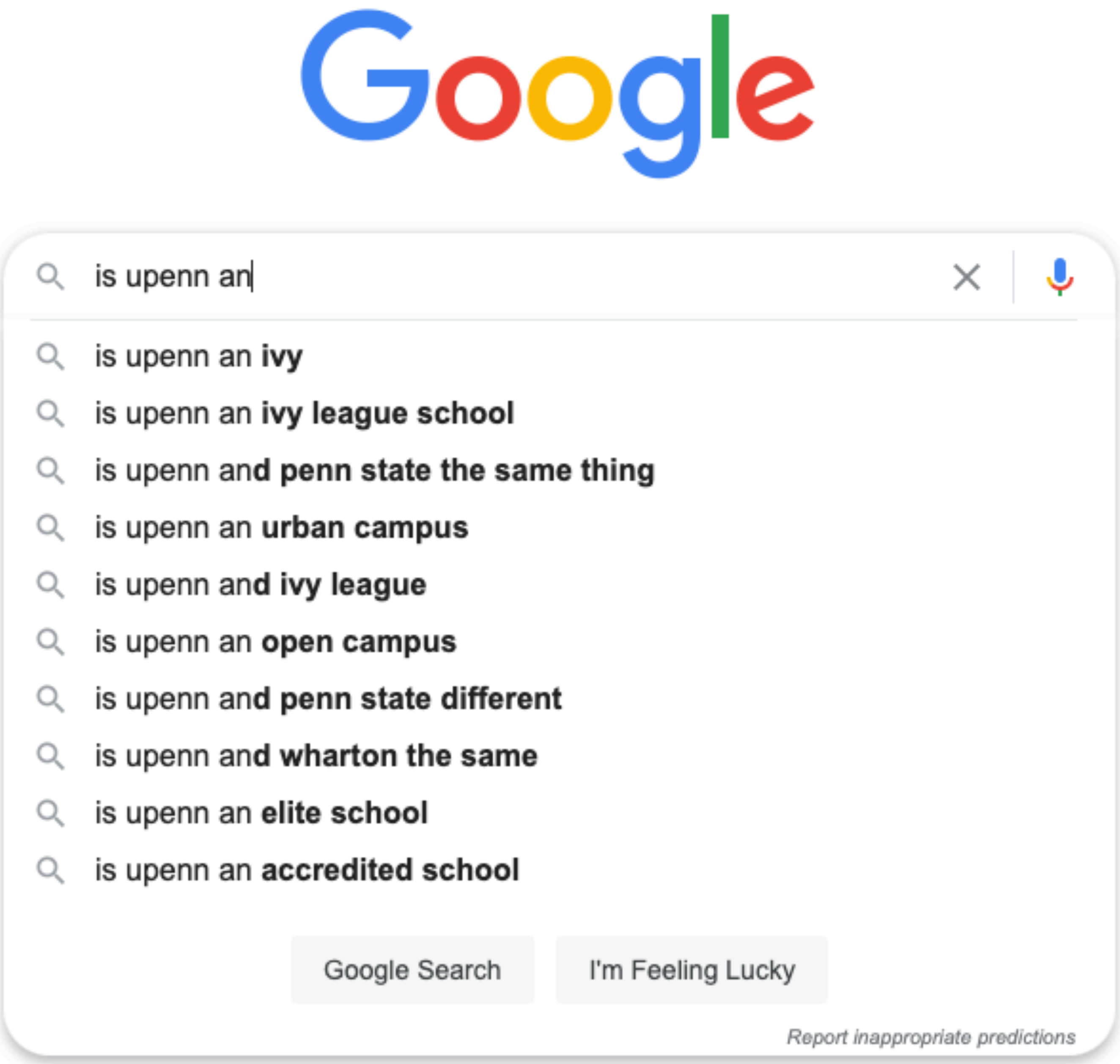
Input: Beginning of sentence
Output: Next word

Three natural language processing tasks

Sentiment analysis

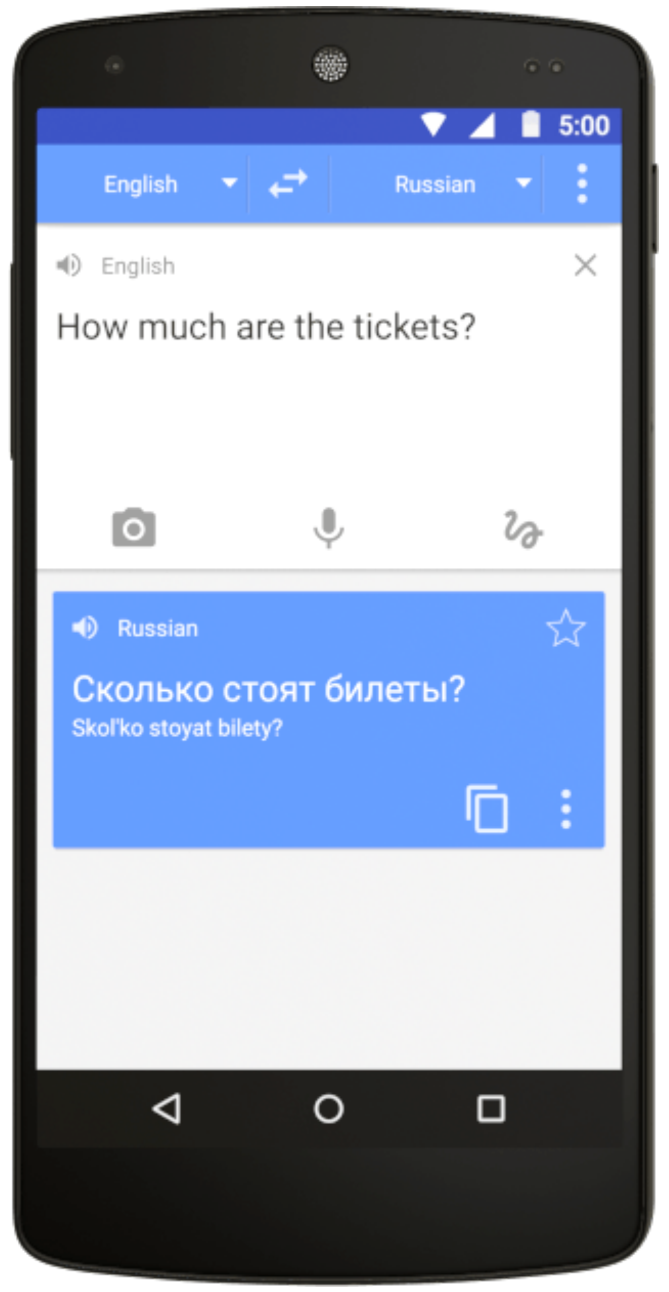
Input: Movie review
Output: Positive or negative

Language modeling



Input: Beginning of sentence
Output: Next word

Machine translation

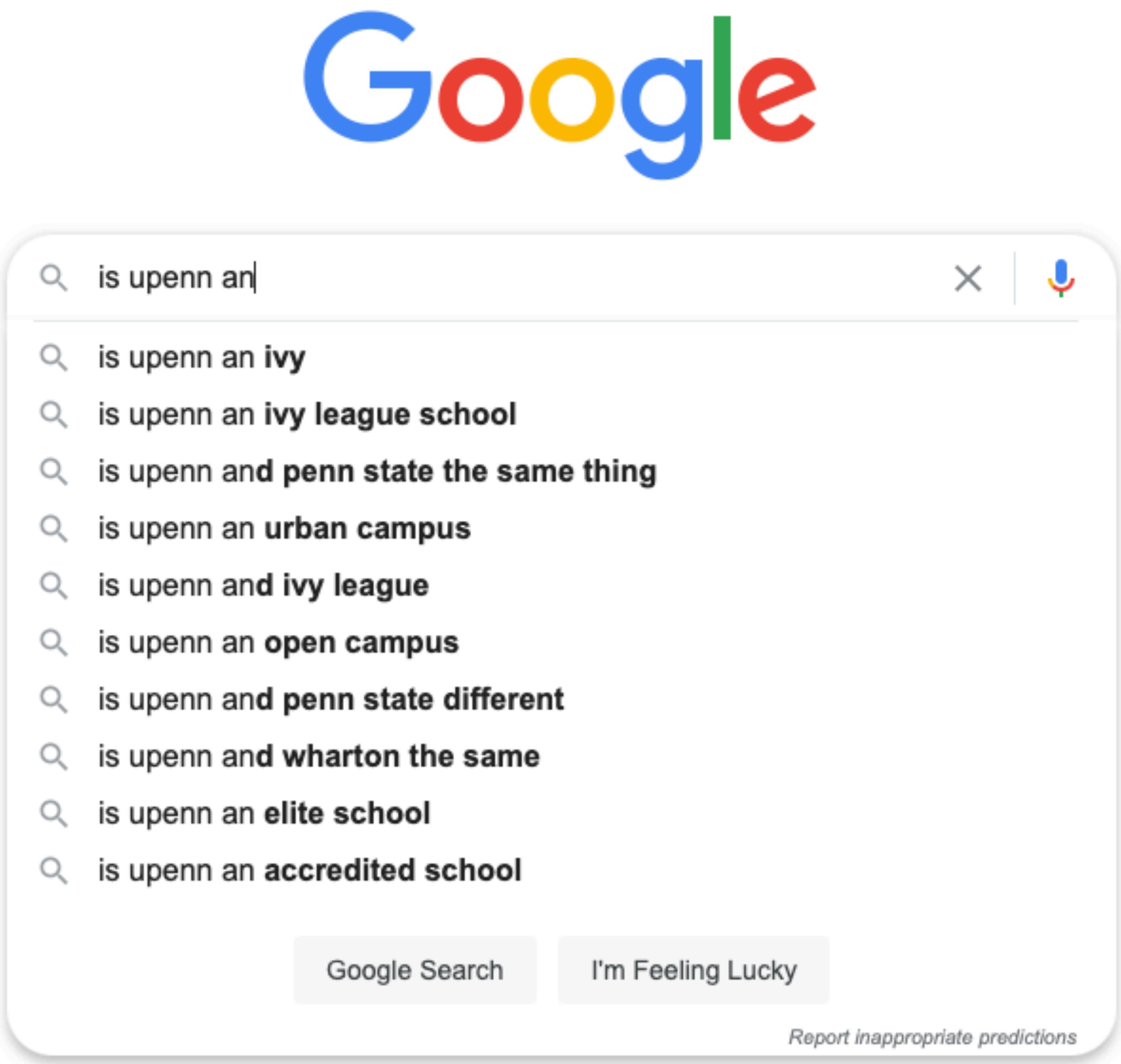


Three natural language processing tasks

Sentiment analysis

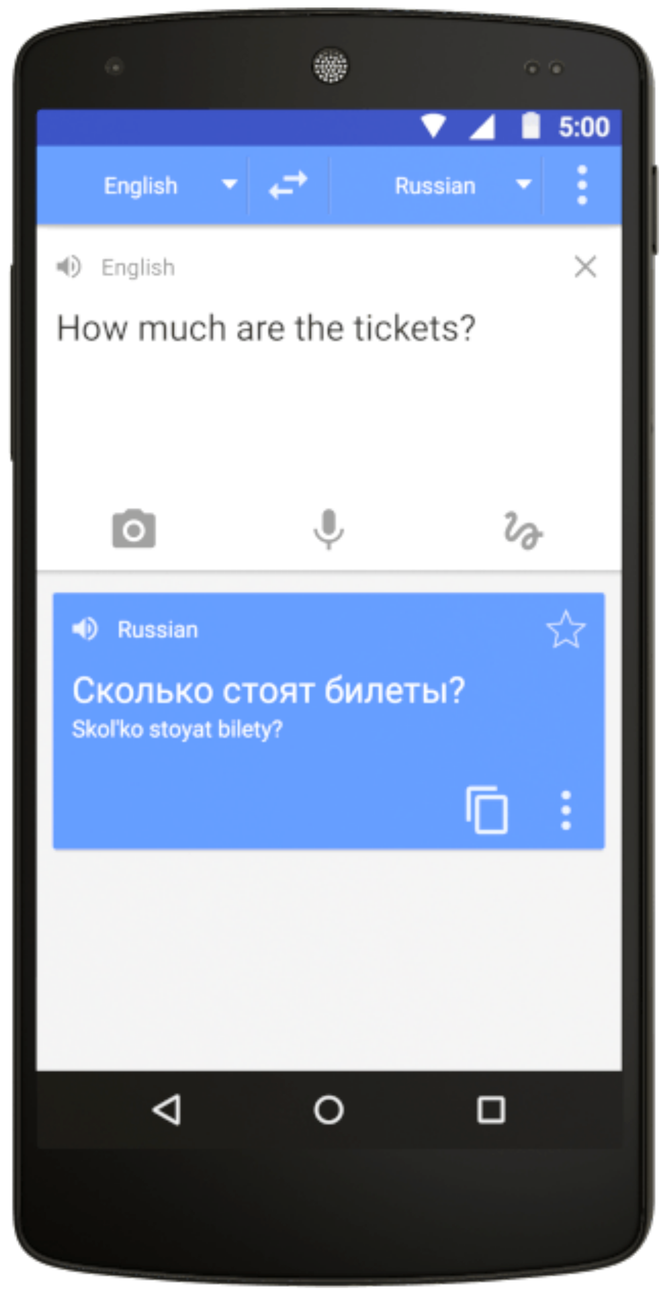
Input: Movie review
Output: Positive or negative

Language modeling



Input: Beginning of sentence
Output: Next word

Machine translation



Input: Sentence in one language
Output: Translation of sentence to another language

What makes NLP challenging?

What makes NLP challenging?

1. NLP requires handling variable-length inputs and/or outputs.

What makes NLP challenging?

1. NLP requires handling variable-length inputs and/or outputs.

Task	Input length	Output length
Sentiment analysis	Variable	Fixed

What makes NLP challenging?

1. NLP requires handling variable-length inputs and/or outputs.

Task	Input length	Output length
Sentiment analysis	Variable	Fixed
Language modeling	Variable	Fixed

What makes NLP challenging?

1. NLP requires handling variable-length inputs and/or outputs.

Task	**Input length**	**Output length**
Sentiment analysis	Variable	Fixed
Language modeling	Variable	Fixed
Machine translation	Variable	Variable

What makes NLP challenging?

1. NLP requires handling variable-length inputs and/or outputs.

Task	Input length	Output length
Sentiment analysis	Variable	Fixed
Language modeling	Variable	Fixed
Machine translation	Variable	Variable

2. The meaning of a word depends on its context:

What makes NLP challenging?

1. NLP requires handling variable-length inputs and/or outputs.

Task	Input length	Output length
Sentiment analysis	Variable	Fixed
Language modeling	Variable	Fixed
Machine translation	Variable	Variable

2. The meaning of a word depends on its context:

“She sat by the river bank.” versus “He went to the bank to deposit his paycheck.”

What makes NLP challenging?

1. NLP requires handling variable-length inputs and/or outputs.

Task	Input length	Output length
Sentiment analysis	Variable	Fixed
Language modeling	Variable	Fixed
Machine translation	Variable	Variable

2. The meaning of a word depends on its context:

“She sat by the river bank.” versus “He went to the bank to deposit his paycheck.”

3. Words do not come with vector representations (unlike pixels).

NLP terminology

NLP terminology

- **Token.** A discrete unit of language, such as a word. Other tokens include punctuation symbols, <SOS> (start of sequence), and <EOS> (end of sequence).

NLP terminology

- **Token.** A discrete unit of language, such as a word. Other tokens include punctuation symbols, <SOS> (start of sequence), and <EOS> (end of sequence).
- **Vocabulary.** The set of all tokens considered by an NLP method.

NLP terminology

- **Token.** A discrete unit of language, such as a word. Other tokens include punctuation symbols, <SOS> (start of sequence), and <EOS> (end of sequence).
- **Vocabulary.** The set of all tokens considered by an NLP method.
- **Vocabulary size.** The number of tokens in a vocabulary.

NLP terminology

- **Token.** A discrete unit of language, such as a word. Other tokens include punctuation symbols, <SOS> (start of sequence), and <EOS> (end of sequence).
- **Vocabulary.** The set of all tokens considered by an NLP method.
- **Vocabulary size.** The number of tokens in a vocabulary.
- **Corpus.** A large body of text (such as Wikipedia), which can serve as training data for an NLP model.

NLP terminology

- **Token.** A discrete unit of language, such as a word. Other tokens include punctuation symbols, <SOS> (start of sequence), and <EOS> (end of sequence).
- **Vocabulary.** The set of all tokens considered by an NLP method.
- **Vocabulary size.** The number of tokens in a vocabulary.
- **Corpus.** A large body of text (such as Wikipedia), which can serve as training data for an NLP model.
- **Encoding.** Representation of token or sequence of tokens as a numeric vector.

NLP timeline

NLP timeline

Deep learning models for NLP:

Model	Popular during	Speed	Sequence length

NLP timeline

Deep learning models for NLP:

Model	Popular during	Speed	Sequence length
Recurrent neural network (RNN)	1980s to early	Good	5-50 tokens

NLP timeline

Deep learning models for NLP:

Model	Popular during	Speed	Sequence length
Recurrent neural network (RNN)	1980s to early	Good	5-50 tokens
RNN with “memory” (e.g. LSTMs)	1997 to mid-2010s	Poor	100-500 tokens

NLP timeline

Deep learning models for NLP:

Model	Popular during	Speed	Sequence length
Recurrent neural network (RNN)	1980s to early	Good	5-50 tokens
RNN with “memory” (e.g. LSTMs)	1997 to mid-2010s	Poor	100-500 tokens
LSTMs with attention	Mid-2010s	Very poor	1000+ tokens

NLP timeline

Deep learning models for NLP:

Model	Popular during	Speed	Sequence length
Recurrent neural network (RNN)	1980s to early	Good	5-50 tokens
RNN with “memory” (e.g. LSTMs)	1997 to mid-2010s	Poor	100-500 tokens
LSTMs with attention	Mid-2010s	Very poor	1000+ tokens
Transformers (“attention is all you need”)	2017-present	Excellent	1000+ tokens

NLP timeline

Deep learning models for NLP:

Model	Popular during	Speed	Sequence length
Recurrent neural network (RNN)	1980s to early	Good	5-50 tokens
RNN with “memory” (e.g. LSTMs)	1997 to mid-2010s	Poor	100-500 tokens
LSTMs with attention	Mid-2010s	Very poor	1000+ tokens
Transformers (“attention is all you need”)	2017-present	Excellent	1000+ tokens

Transformer models:

Model	Year	Number of parameters

NLP timeline

Deep learning models for NLP:

Model	Popular during	Speed	Sequence length
Recurrent neural network (RNN)	1980s to early	Good	5-50 tokens
RNN with “memory” (e.g. LSTMs)	1997 to mid-2010s	Poor	100-500 tokens
LSTMs with attention	Mid-2010s	Very poor	1000+ tokens
Transformers (“attention is all you need”)	2017-present	Excellent	1000+ tokens

Transformer models:

Model	Year	Number of parameters
Initial transformer model (Google)	2017	213M

NLP timeline

Deep learning models for NLP:

Model	Popular during	Speed	Sequence length
Recurrent neural network (RNN)	1980s to early	Good	5-50 tokens
RNN with “memory” (e.g. LSTMs)	1997 to mid-2010s	Poor	100-500 tokens
LSTMs with attention	Mid-2010s	Very poor	1000+ tokens
Transformers (“attention is all you need”)	2017-present	Excellent	1000+ tokens

Transformer models:

Model	Year	Number of parameters
Initial transformer model (Google)	2017	213M
GPT (OpenAI)	2018	117M

NLP timeline

Deep learning models for NLP:

Model	Popular during	Speed	Sequence length
Recurrent neural network (RNN)	1980s to early	Good	5-50 tokens
RNN with “memory” (e.g. LSTMs)	1997 to mid-2010s	Poor	100-500 tokens
LSTMs with attention	Mid-2010s	Very poor	1000+ tokens
Transformers (“attention is all you need”)	2017-present	Excellent	1000+ tokens

Transformer models:

Model	Year	Number of parameters
Initial transformer model (Google)	2017	213M
GPT (OpenAI)	2018	117M
BERT (Google)	2018	340M

NLP timeline

Deep learning models for NLP:

Model	Popular during	Speed	Sequence length
Recurrent neural network (RNN)	1980s to early	Good	5-50 tokens
RNN with “memory” (e.g. LSTMs)	1997 to mid-2010s	Poor	100-500 tokens
LSTMs with attention	Mid-2010s	Very poor	1000+ tokens
Transformers (“attention is all you need”)	2017-present	Excellent	1000+ tokens

Transformer models:

Model	Year	Number of parameters
Initial transformer model (Google)	2017	213M
GPT (OpenAI)	2018	117M
BERT (Google)	2018	340M
ChatGPT-3.5 (OpenAI)	2022	175B

NLP timeline

Deep learning models for NLP:

Model	Popular during	Speed	Sequence length
Recurrent neural network (RNN)	1980s to early	Good	5-50 tokens
RNN with “memory” (e.g. LSTMs)	1997 to mid-2010s	Poor	100-500 tokens
LSTMs with attention	Mid-2010s	Very poor	1000+ tokens
Transformers (“attention is all you need”)	2017-present	Excellent	1000+ tokens

Transformer models:

Model	Year	Number of parameters
Initial transformer model (Google)	2017	213M
GPT (OpenAI)	2018	117M
BERT (Google)	2018	340M
ChatGPT-3.5 (OpenAI)	2022	175B
ChatGPT-4 (OpenAI)	2023	1.76T

Word vectors

Word vectors

A common first step in deep learning for NLP is to encode each word using a word vector, e.g.

$$v_{\text{man}} = (0.2, 1.6, 2.5, -3.2, 4.1)$$

Word vectors

A common first step in deep learning for NLP is to encode each word using a word vector, e.g.

$$v_{\text{man}} = (0.2, 1.6, 2.5, -3.2, 4.1)$$

Word vectors capture meaning, unlike one-hot encodings, such as

$$v_{\text{man}} = (0,0,1,0,0,0,0,0,0)$$

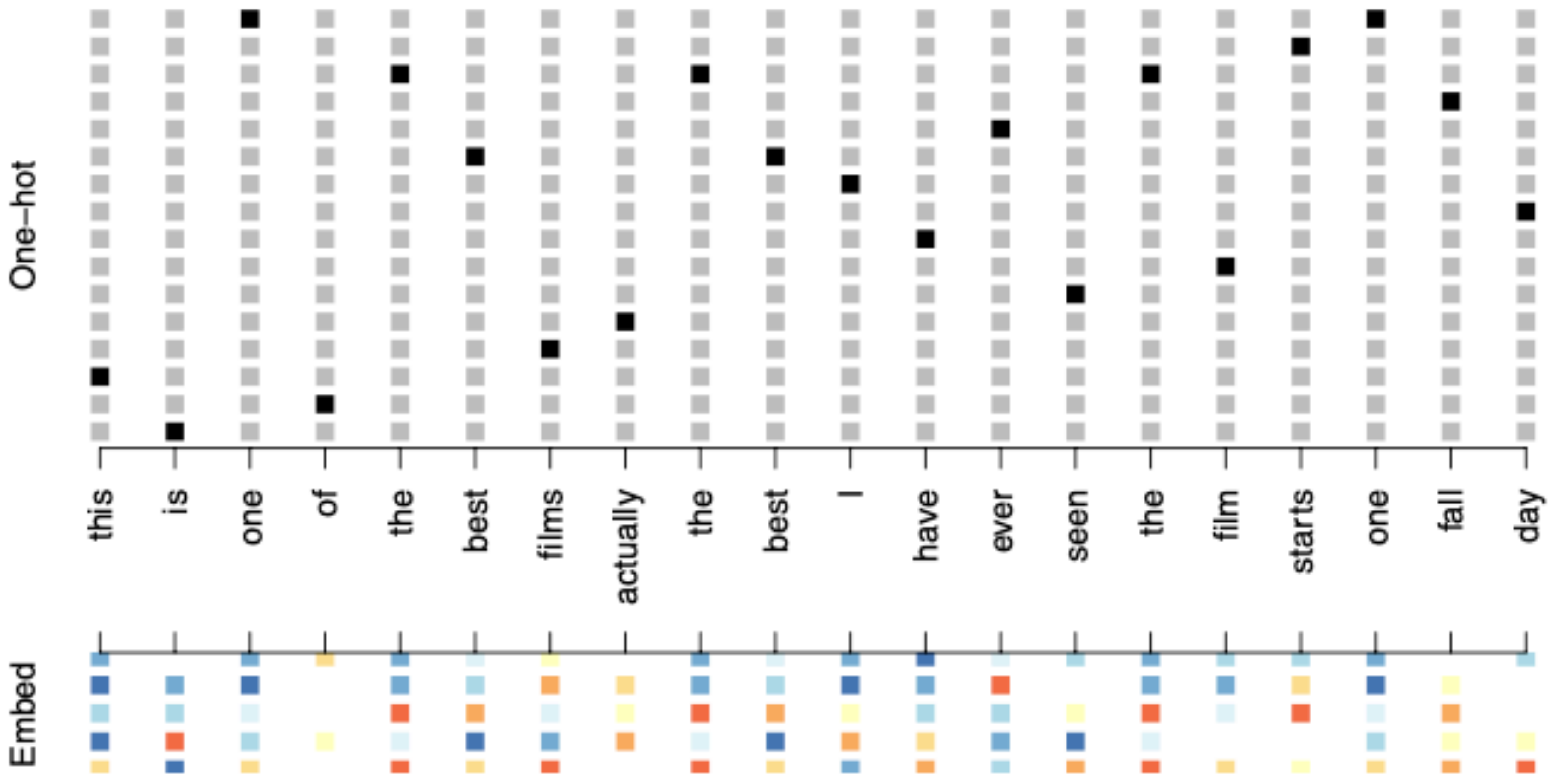
Word vectors

A common first step in deep learning for NLP is to encode each word using a word vector, e.g.

$$v_{\text{man}} = (0.2, 1.6, 2.5, -3.2, 4.1)$$

Word vectors capture meaning, unlike one-hot encodings, such as

$$v_{\text{man}} = (0,0,1,0,0,0,0,0,0)$$



Source: ISLRv2

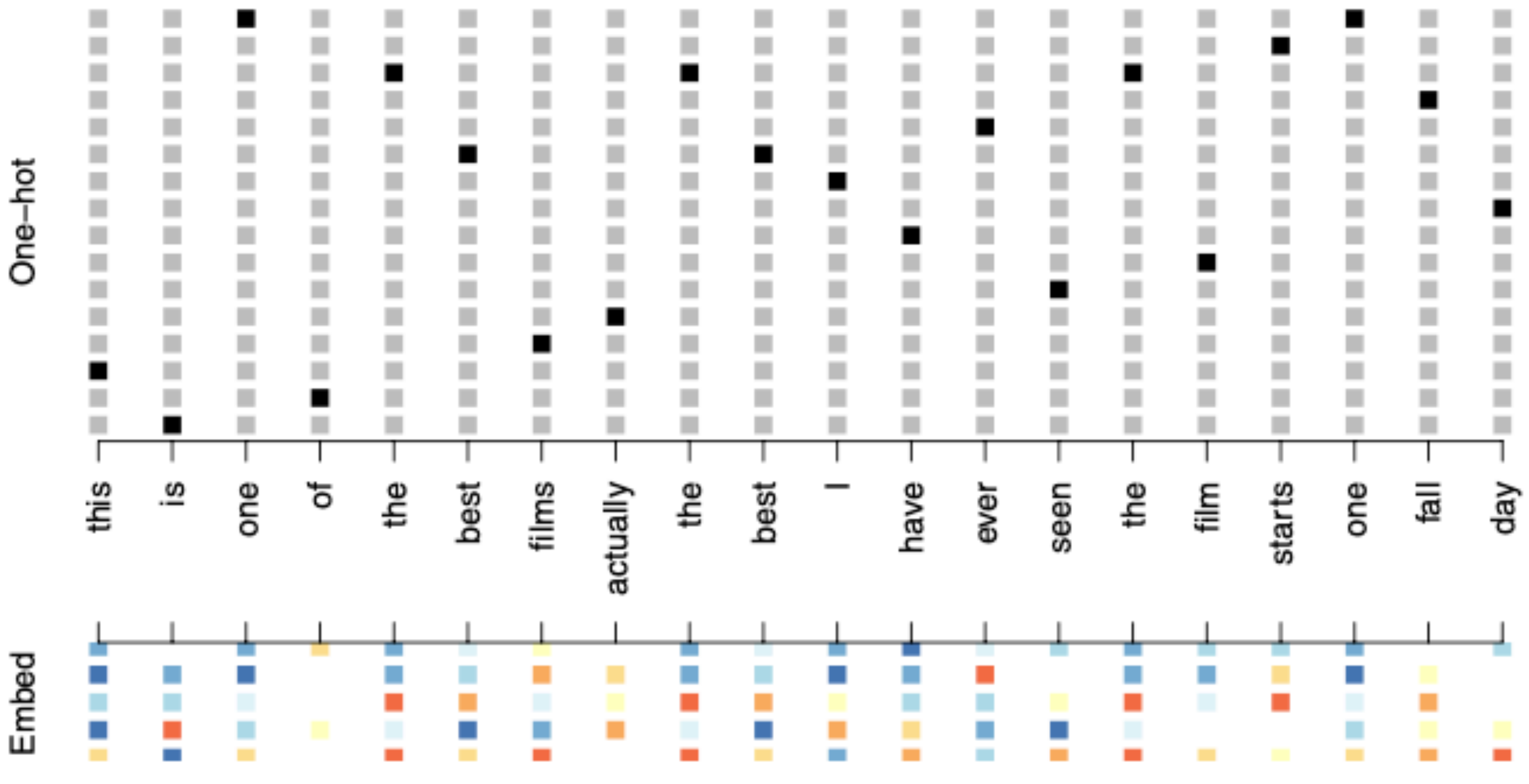
Word vectors

A common first step in deep learning for NLP is to encode each word using a word vector, e.g.

$$v_{\text{man}} = (0.2, 1.6, 2.5, -3.2, 4.1)$$

Word vectors capture meaning, unlike one-hot encodings, such as

$$v_{\text{man}} = (0,0,1,0,0,0,0,0,0)$$



Source: ISLRv2

Word vectors are trained on large text corpora, so that word vectors for pairs of frequently co-occurring words are similar. Popular algorithms: word2vec, GloVe.

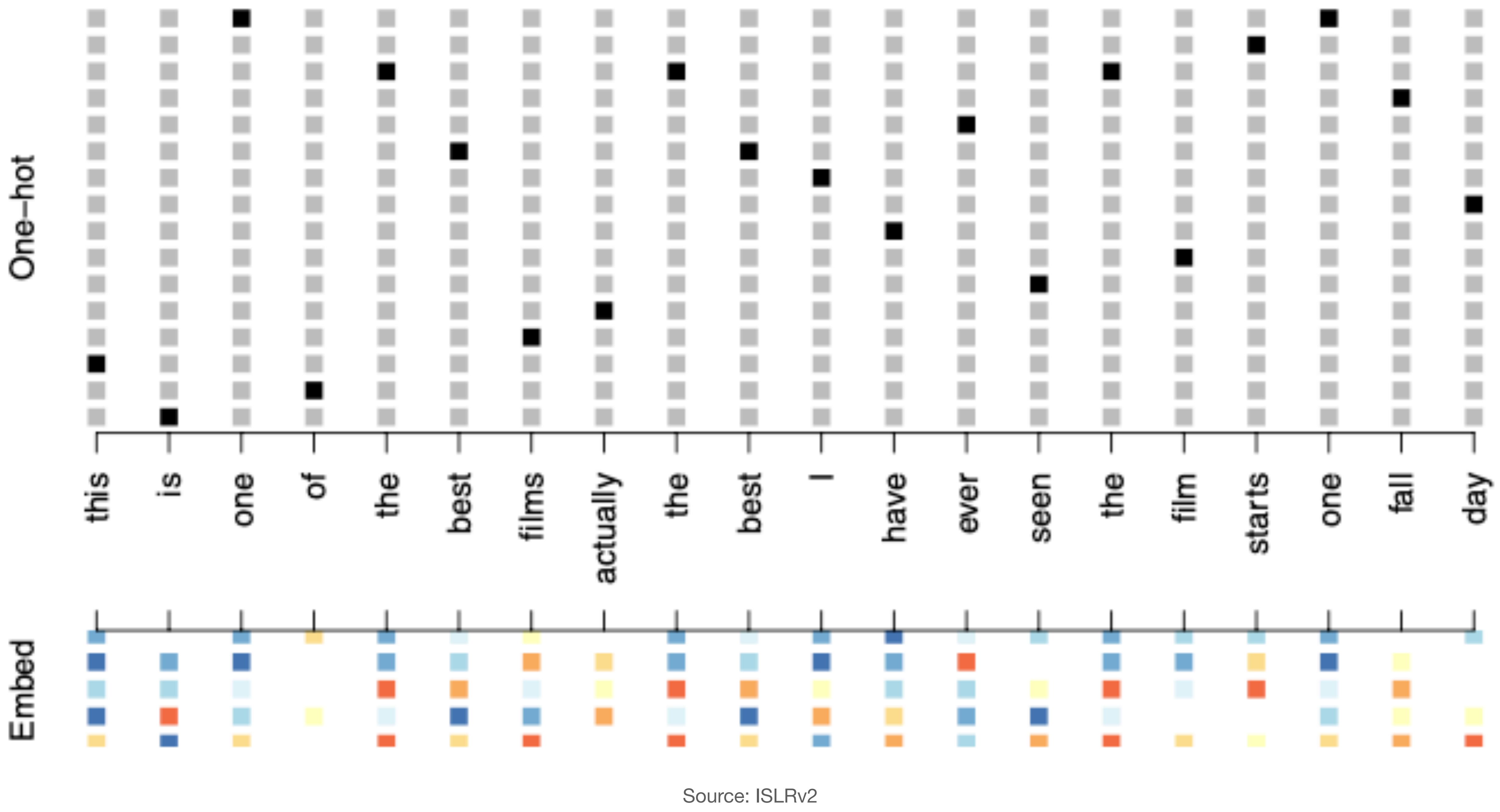
Word vectors

A common first step in deep learning for NLP is to encode each word using a word vector, e.g.

$$v_{\text{man}} = (0.2, 1.6, 2.5, -3.2, 4.1)$$

Word vectors capture meaning, unlike one-hot encodings, such as

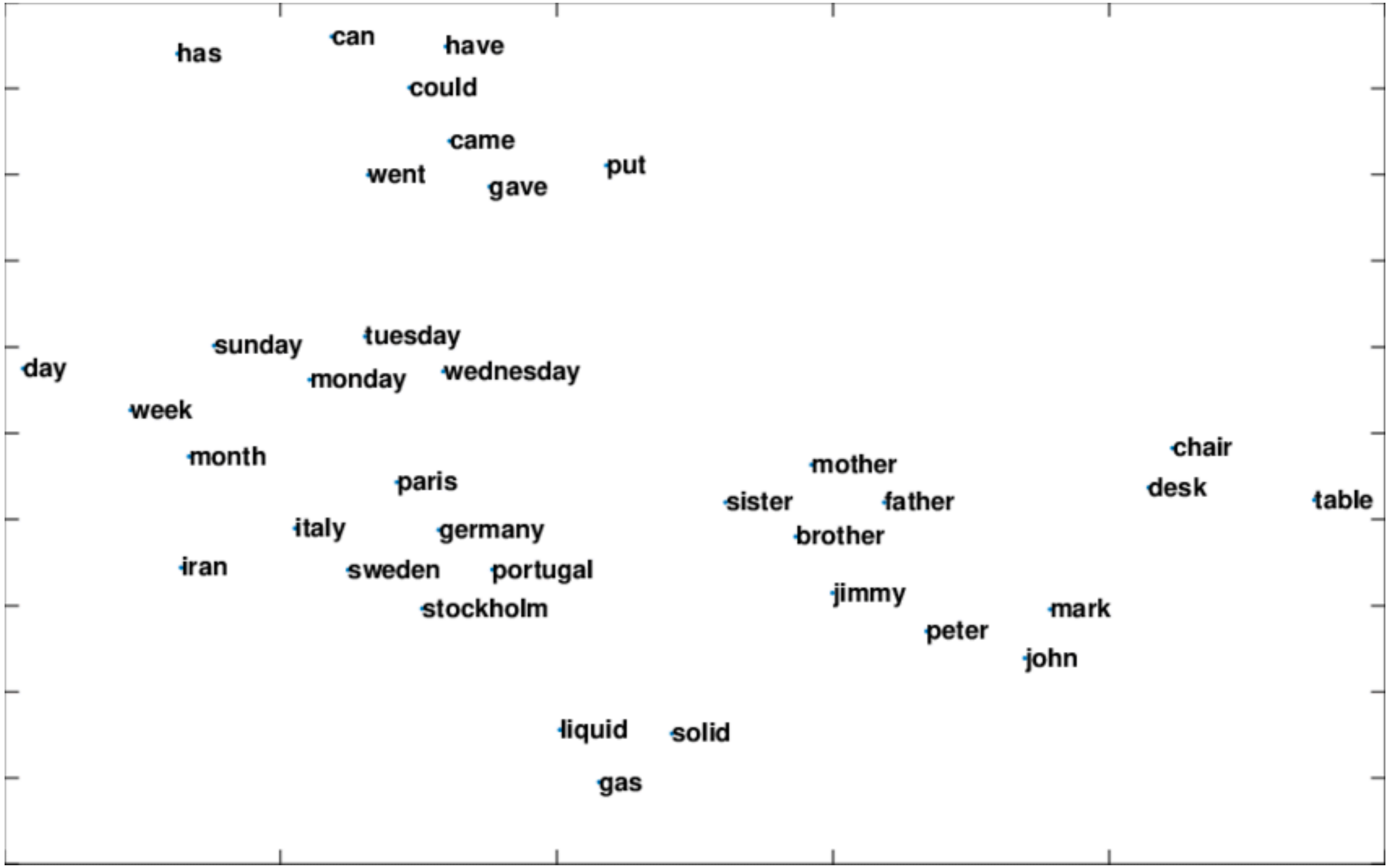
$$v_{\text{man}} = (0,0,1,0,0,0,0,0,0)$$



Word vectors are trained on large text corpora, so that word vectors for pairs of frequently co-occurring words are similar. Popular algorithms: word2vec, GloVe.

Word vectors need to be trained only once (for each language), and can be reused.

Word vectors capture semantic relationships



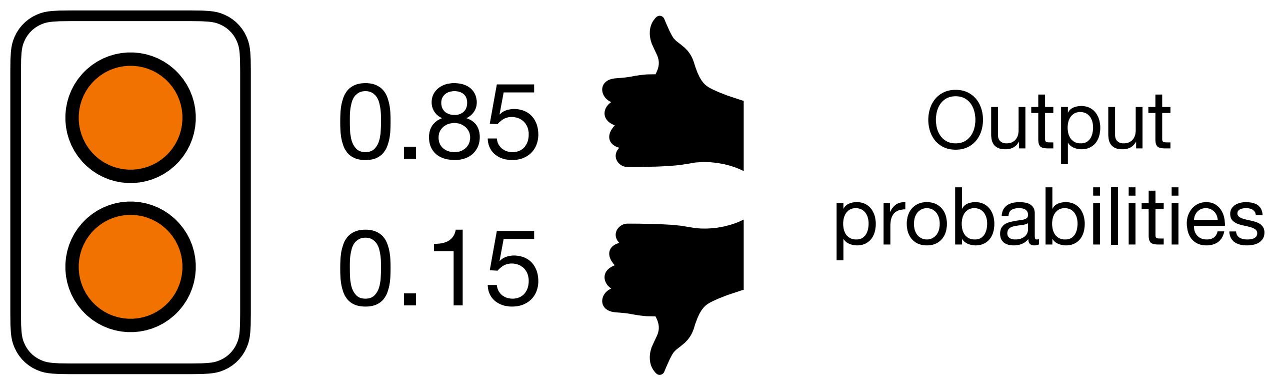
https://www.researchgate.net/figure/A-two-dimensional-representation-of-word-embeddings-Words-with-similar-meanings-are_fig1_327074728

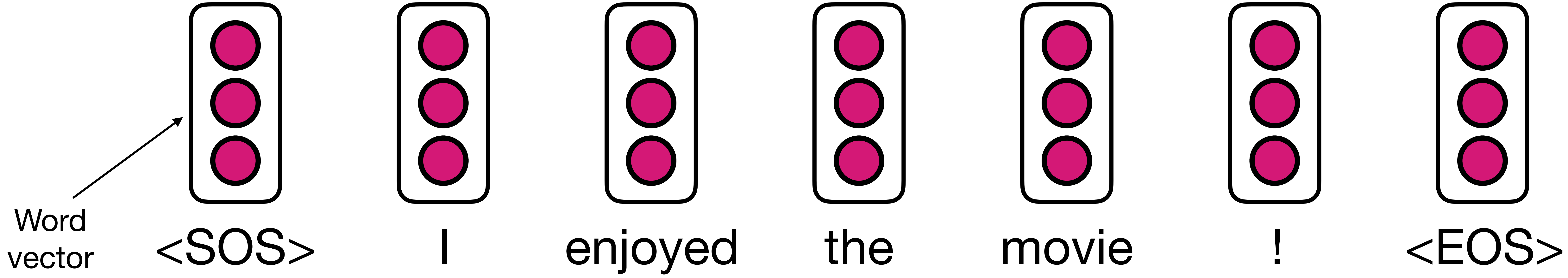
RNN for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

RNN for sentiment analysis

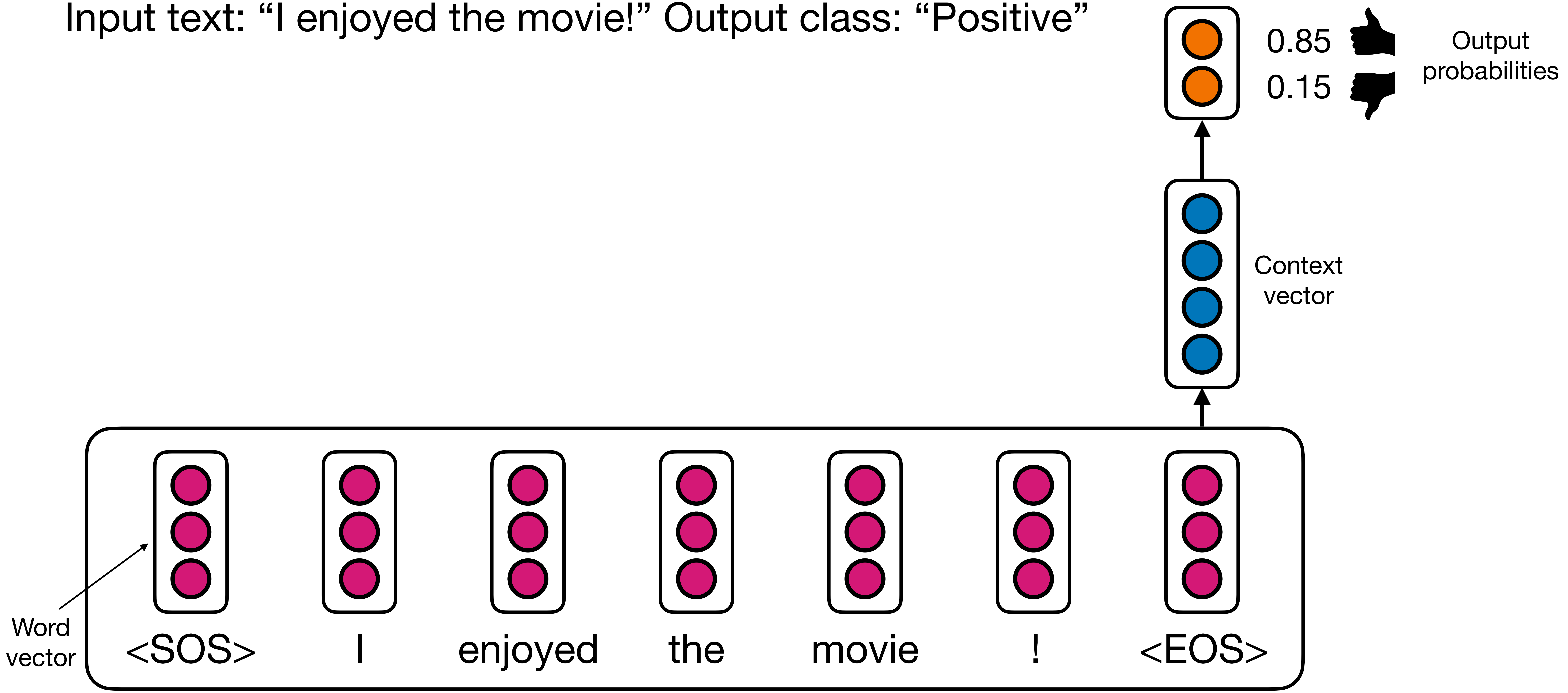
Input text: “I enjoyed the movie!” Output class: “Positive”





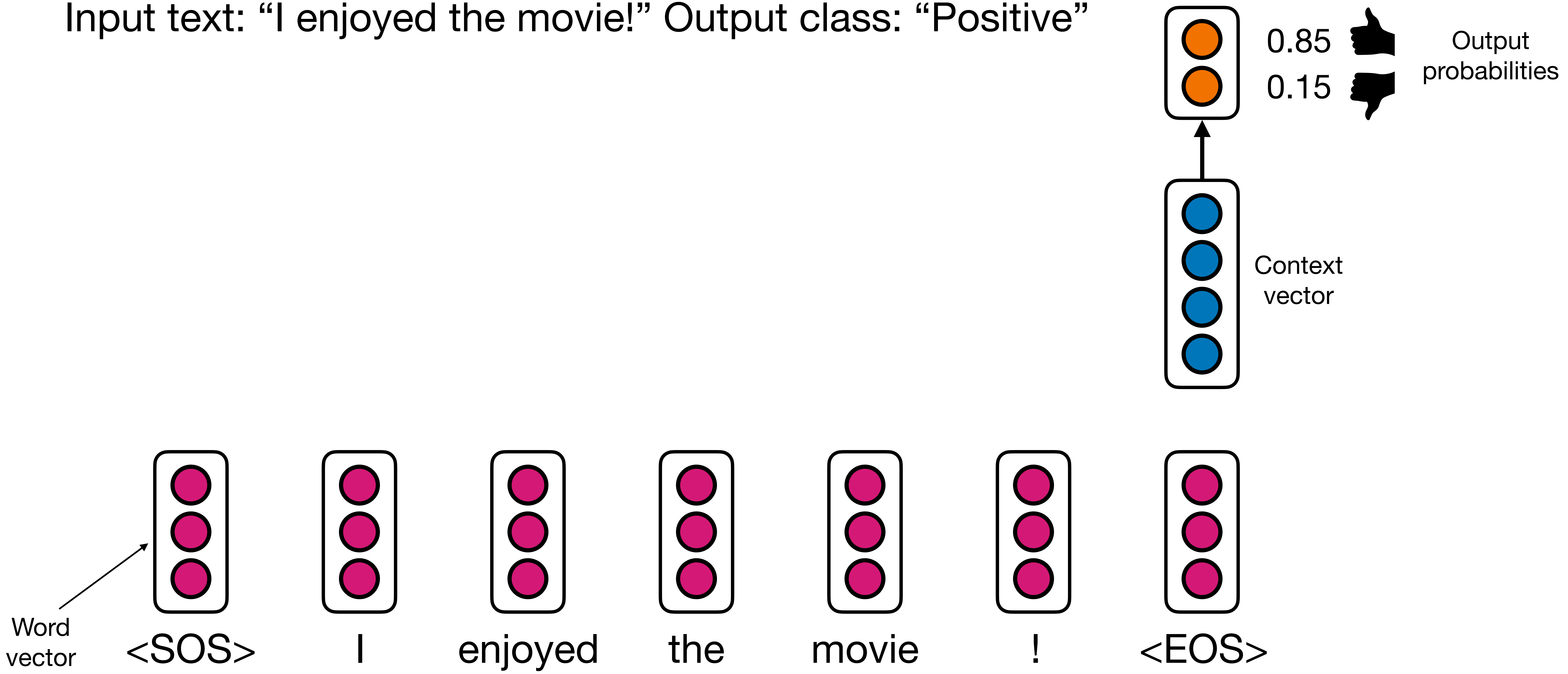
RNN for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”



RNN for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”



RNN for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

RNN for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

RNN for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

RNN for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

RNN for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

- Arrows denote FC layers

RNN for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

RNN for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

RNN for sentiment analysis

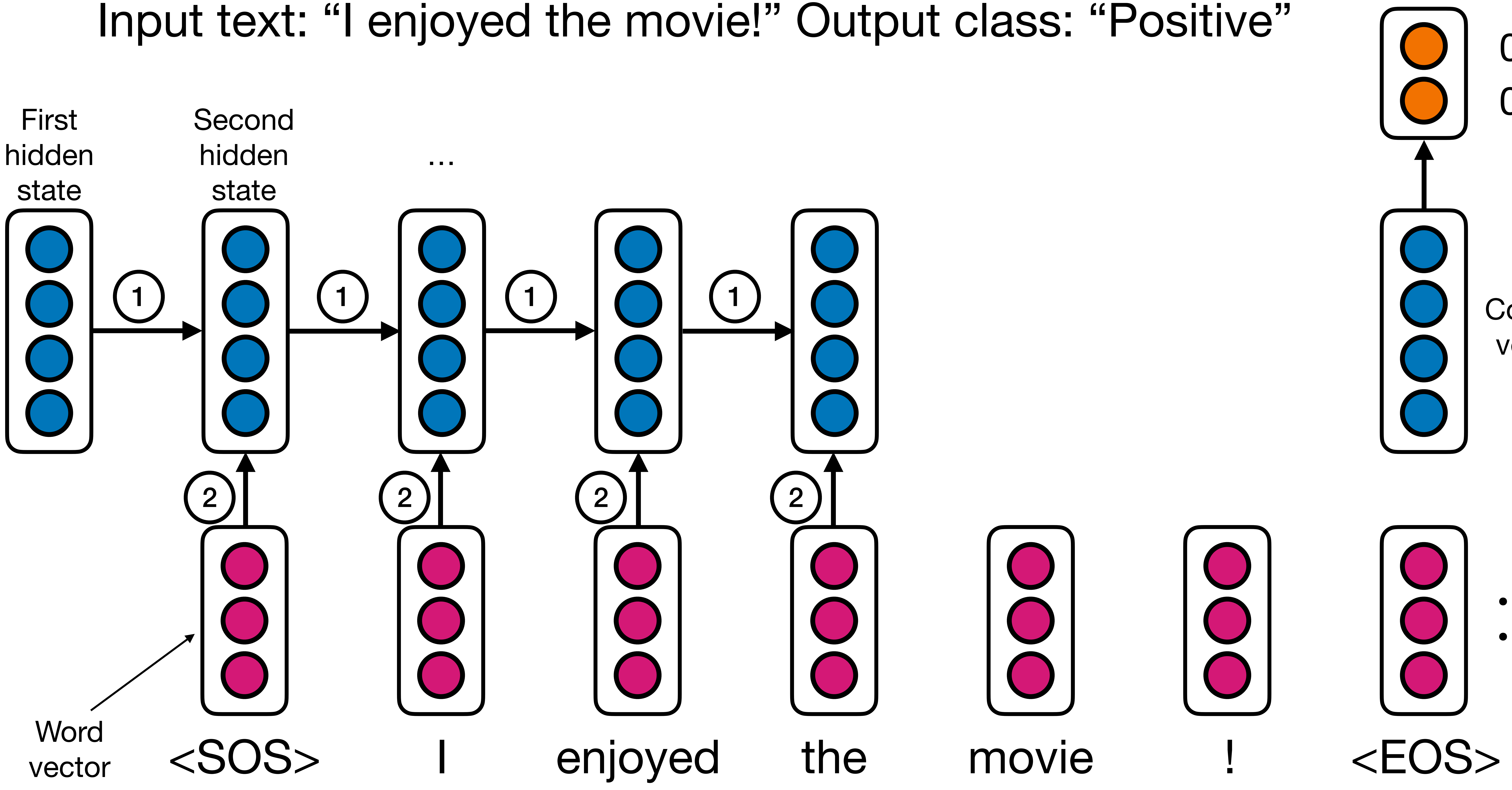
Input text: “I enjoyed the movie!” Output class: “Positive”

RNN for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

RNN for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”



- Arrows denote FC layers
- Arrows with same numbers share weights

RNN for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

- Arrows denote FC layers
- Arrows with same numbers share weights

RNN for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

RNN for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

RNN for sentiment analysis

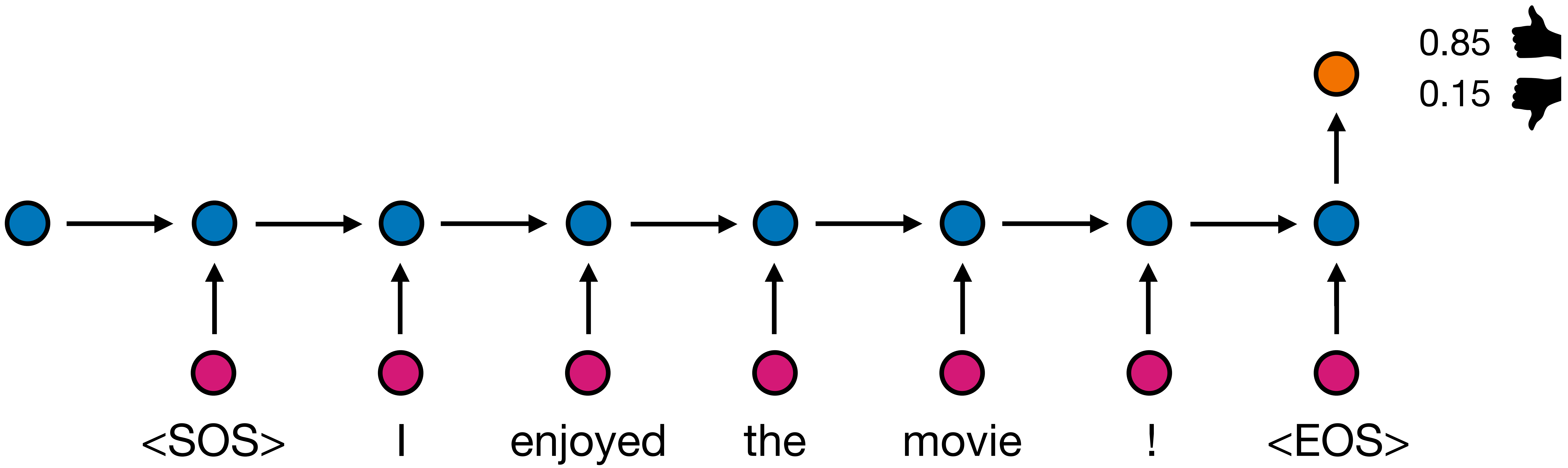
Input text: “I enjoyed the movie!” Output class: “Positive”

RNN for sentiment analysis (simplified)

Input text: “I enjoyed the movie!” Output class: “Positive”

RNN for sentiment analysis (simplified)

Input text: “I enjoyed the movie!” Output class: “Positive”



Training RNNs for sentiment classification

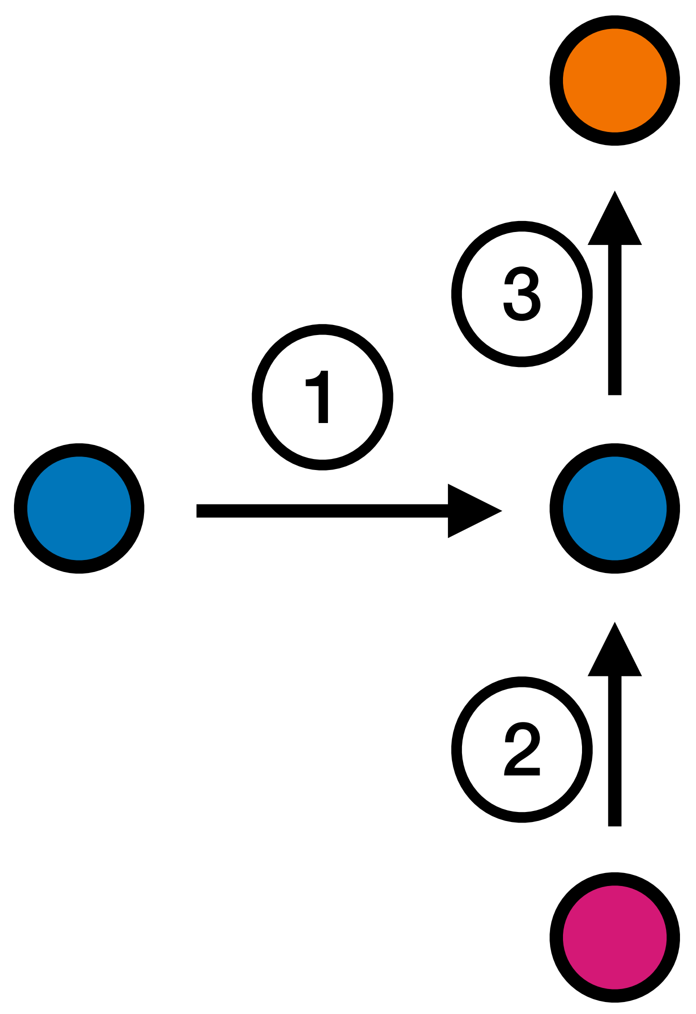
Training data

Movie ratings database:

Input	Output
“I enjoyed the movie.”	“Positive”
“Despite its intriguing premise, this movie ended up being disappointment.”	“Negative”
…	…
“This was the best movie I had seen in a while.”	“Positive”

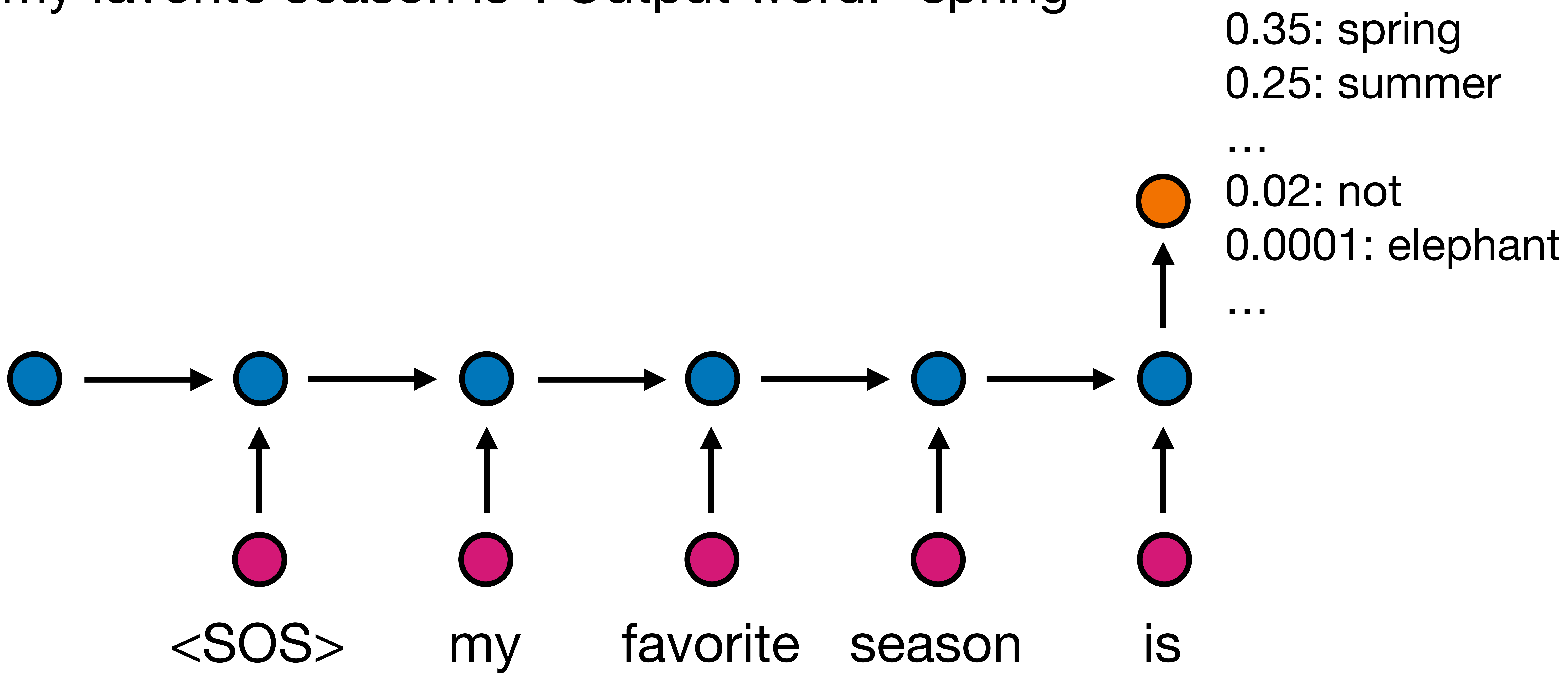
Training process

- Learn the three sets of weights via stochastic gradient descent on the cross-entropy loss function.



RNN for language modeling

Input text: “my favorite season is”. Output word: “spring”



Training RNNs for language modeling

Training data

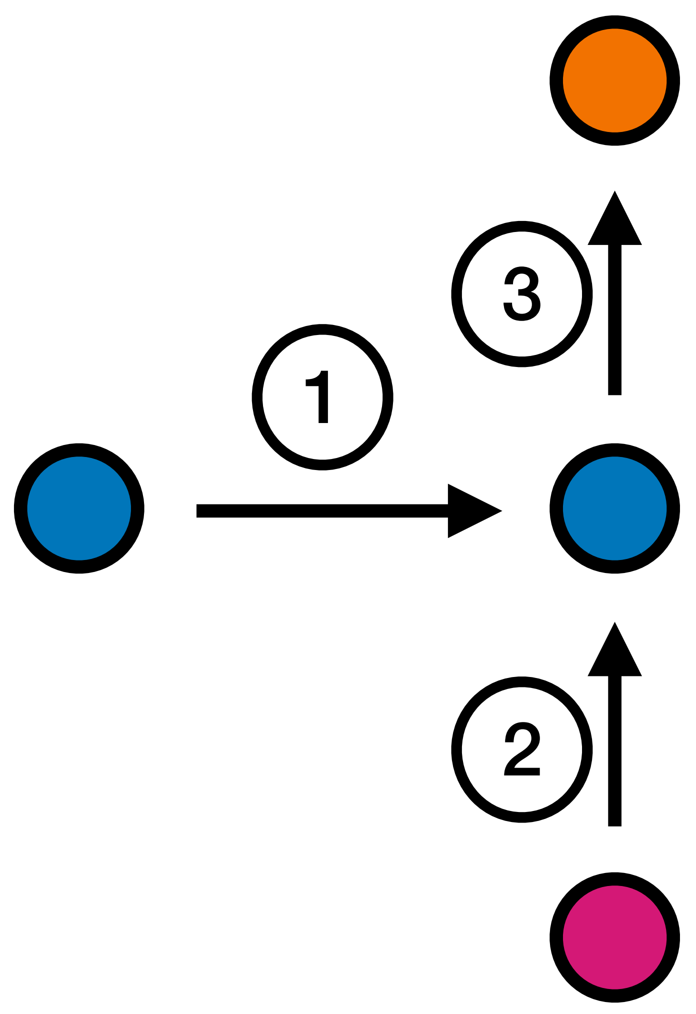
Large corpus of text, e.g. Wikipedia.

“My favorite season in spring. The flowers are blooming and the sun is shining…”

Input	Output
“My”	“favorite”
“My favorite”	“season”
“My favorite season”	“is”
“My favorite season is”	“spring”
…	…

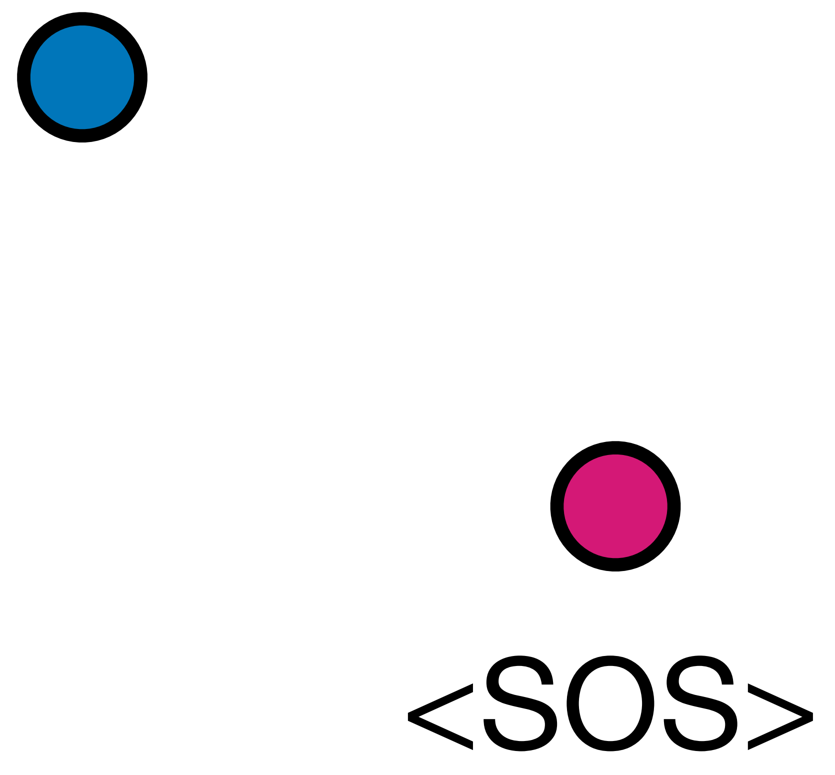
Training process

- Learn the three sets of weights via stochastic gradient descent on the cross-entropy loss function.

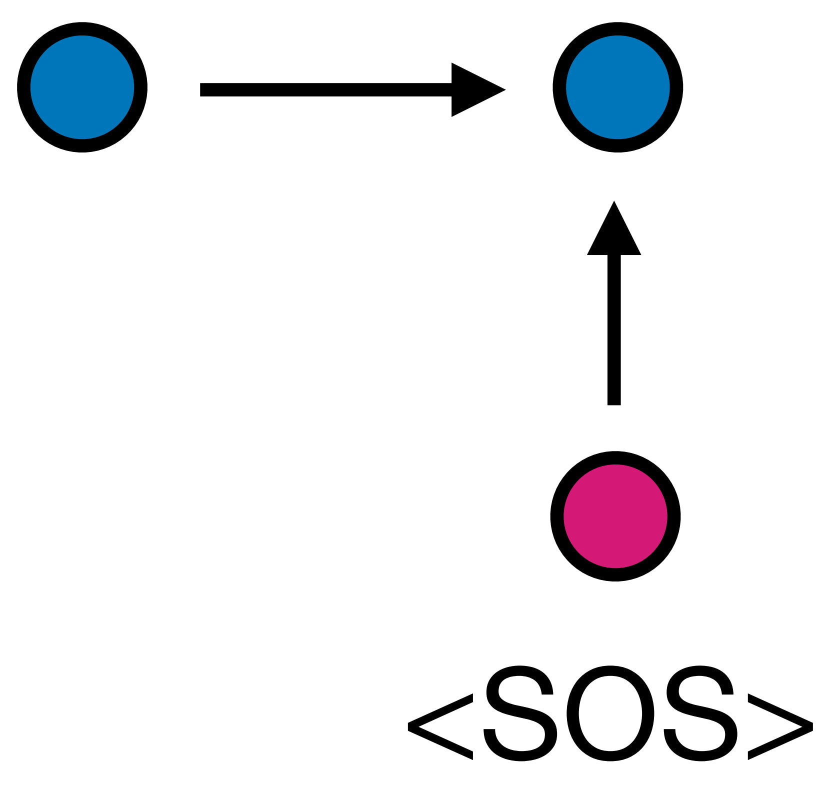


RNNs for autoregressive text generation

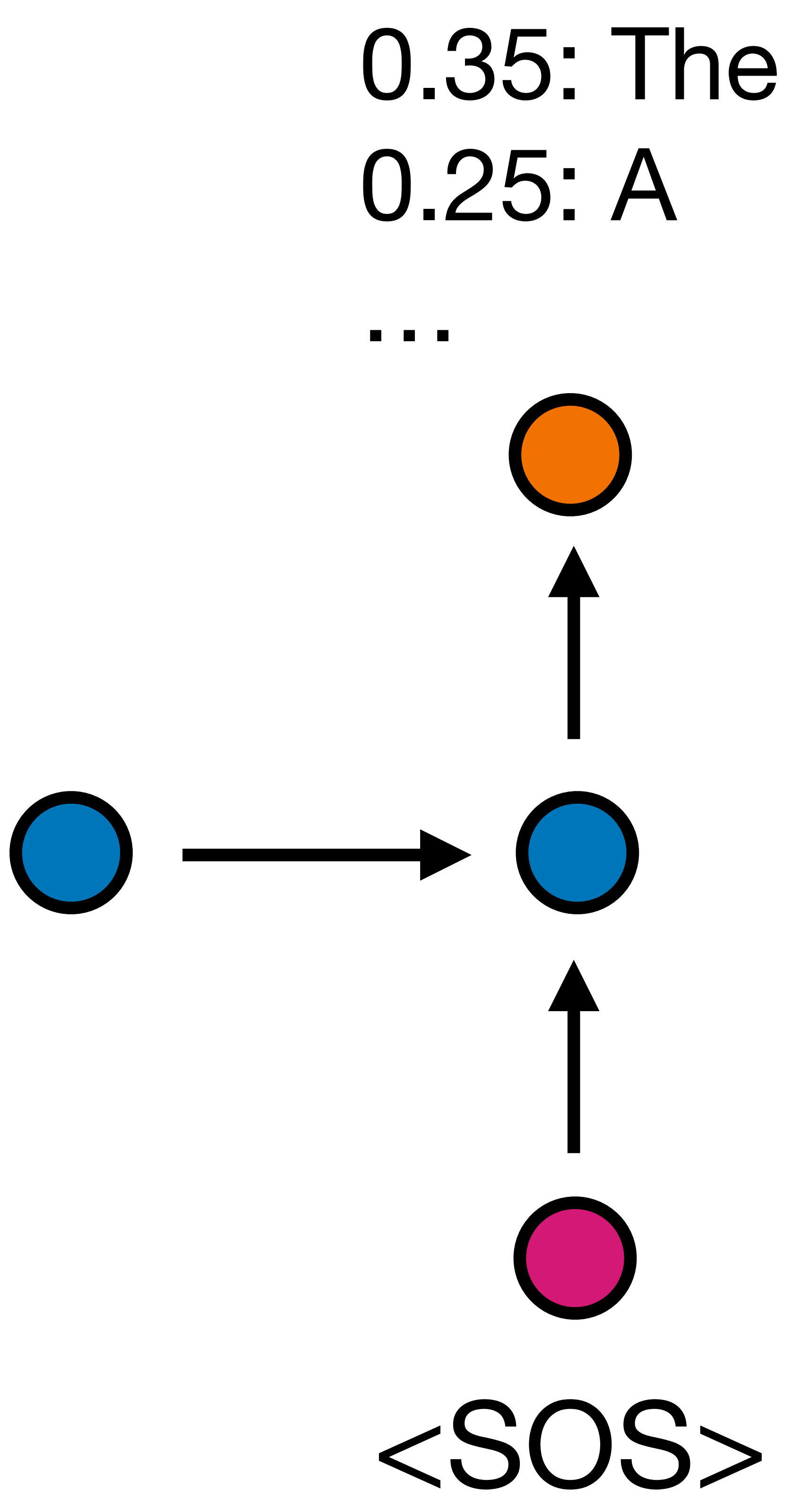
RNNs for autoregressive text generation



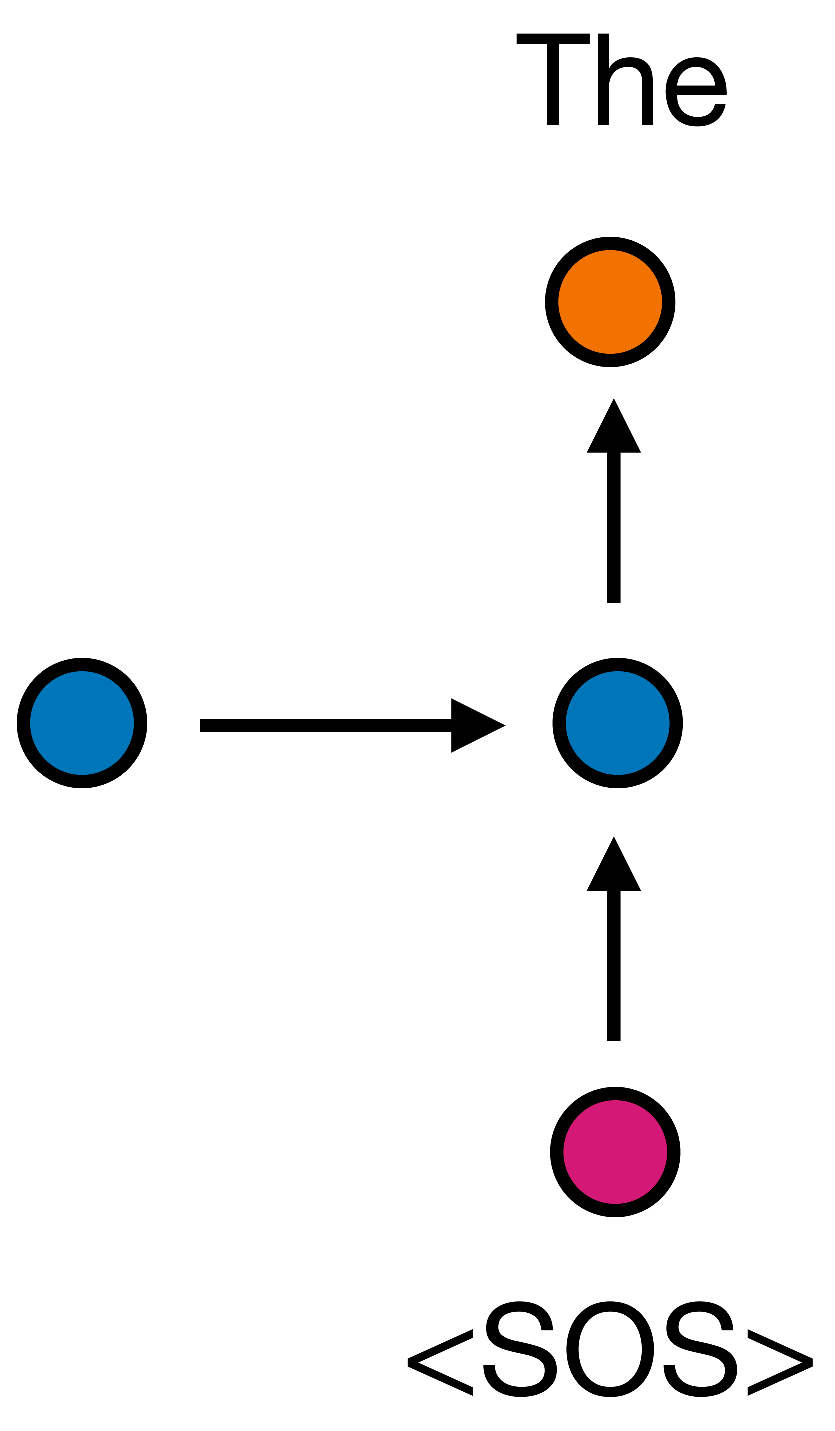
RNNs for autoregressive text generation



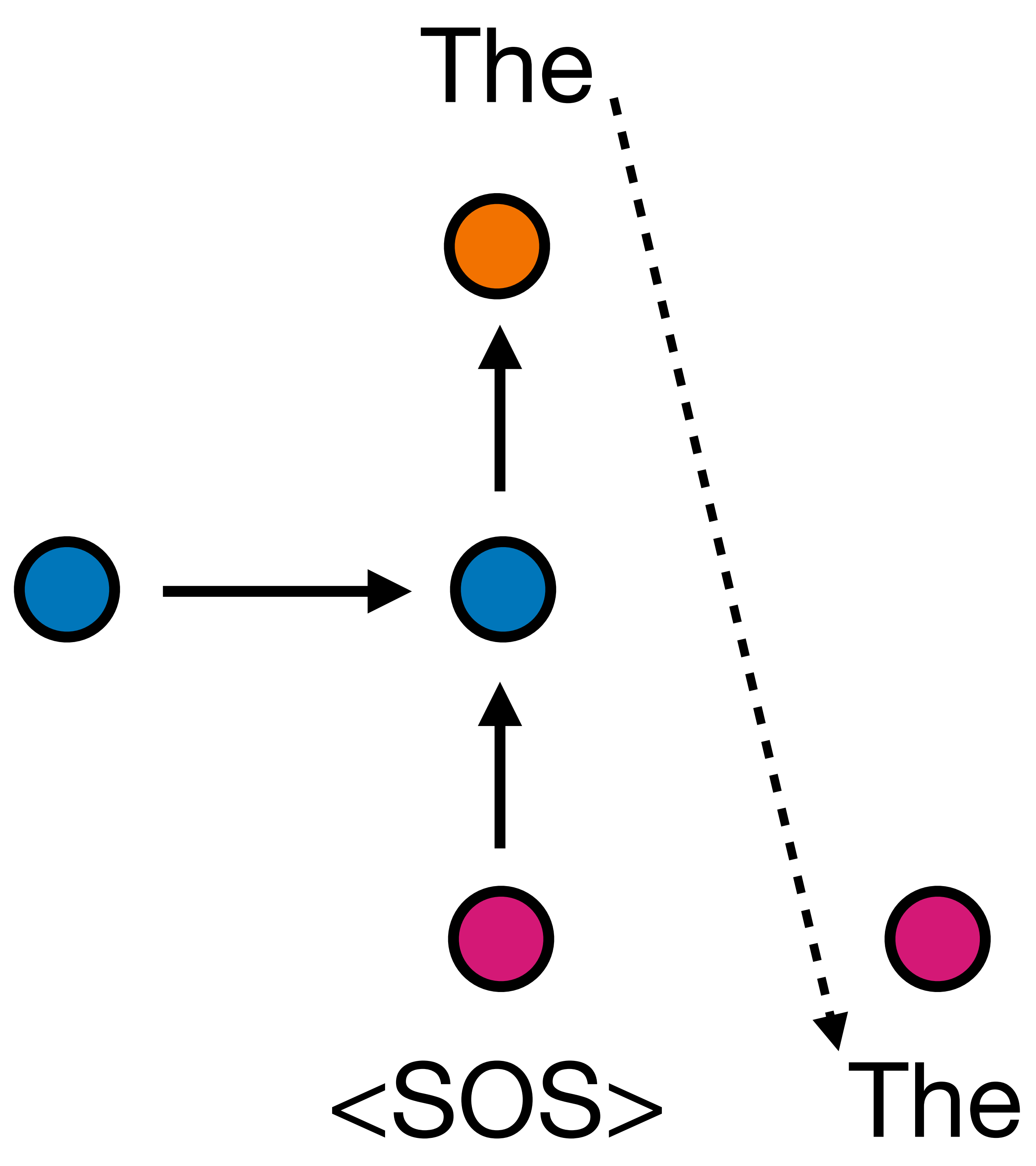
RNNs for autoregressive text generation



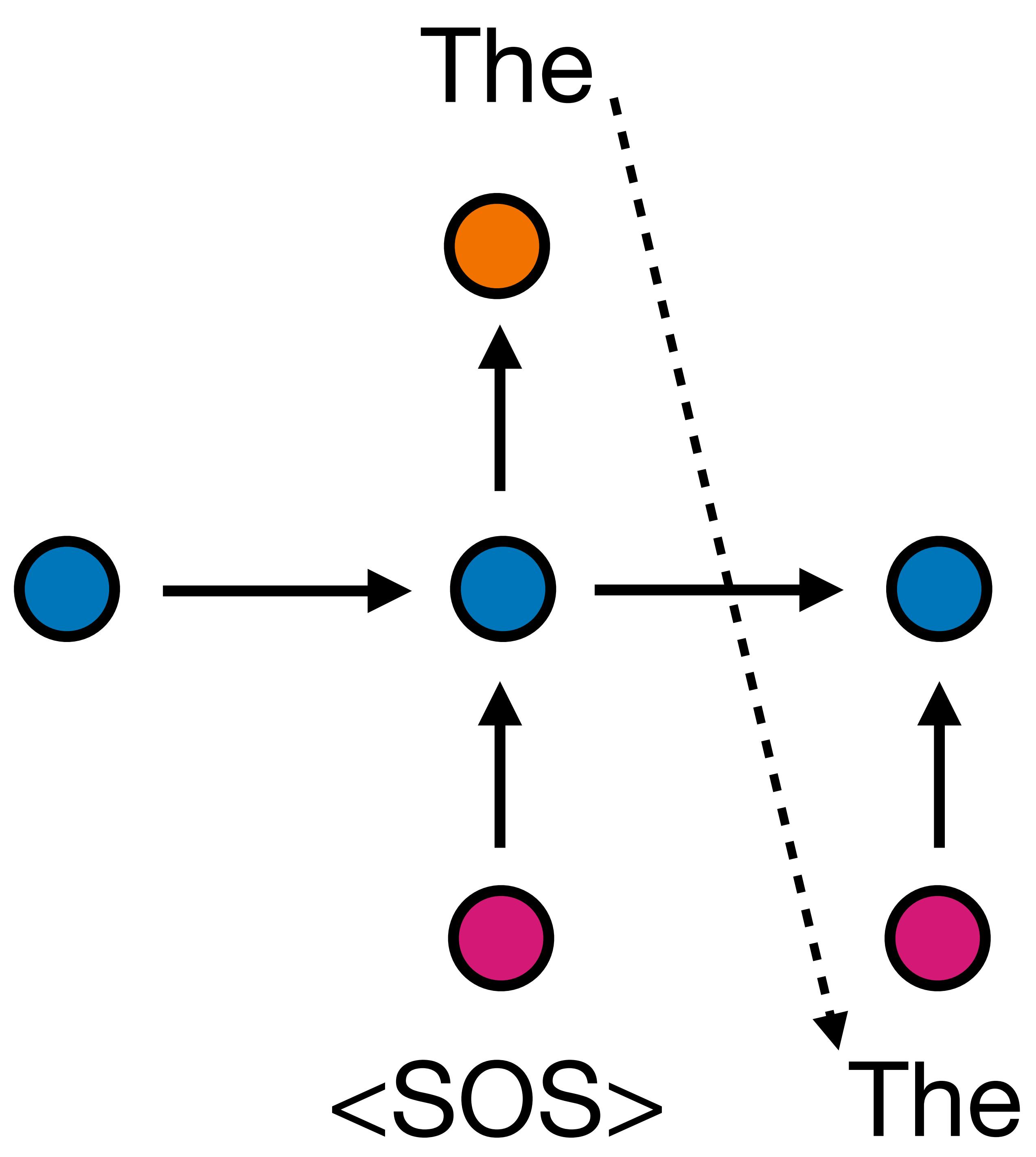
RNNs for autoregressive text generation



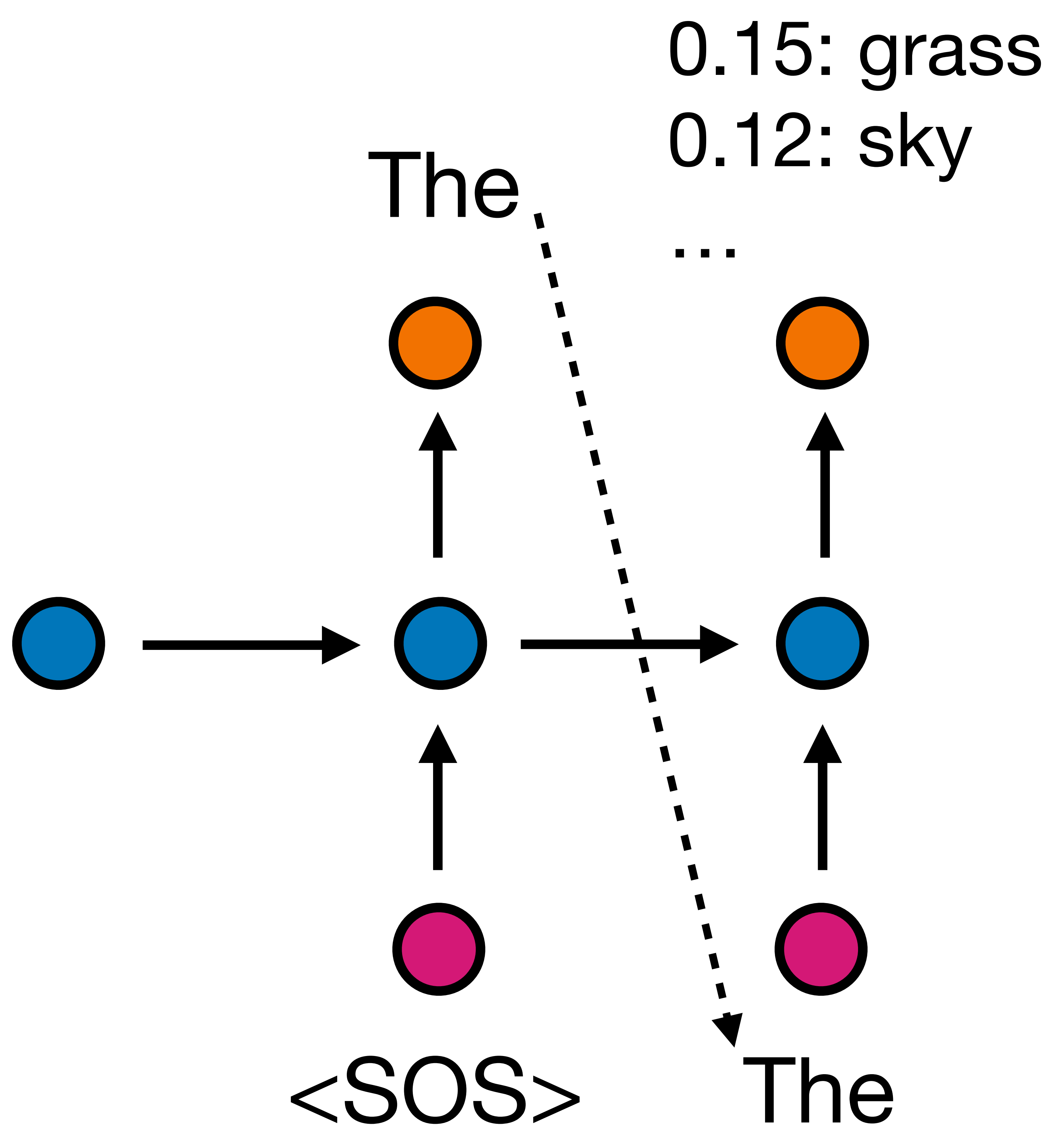
RNNs for autoregressive text generation



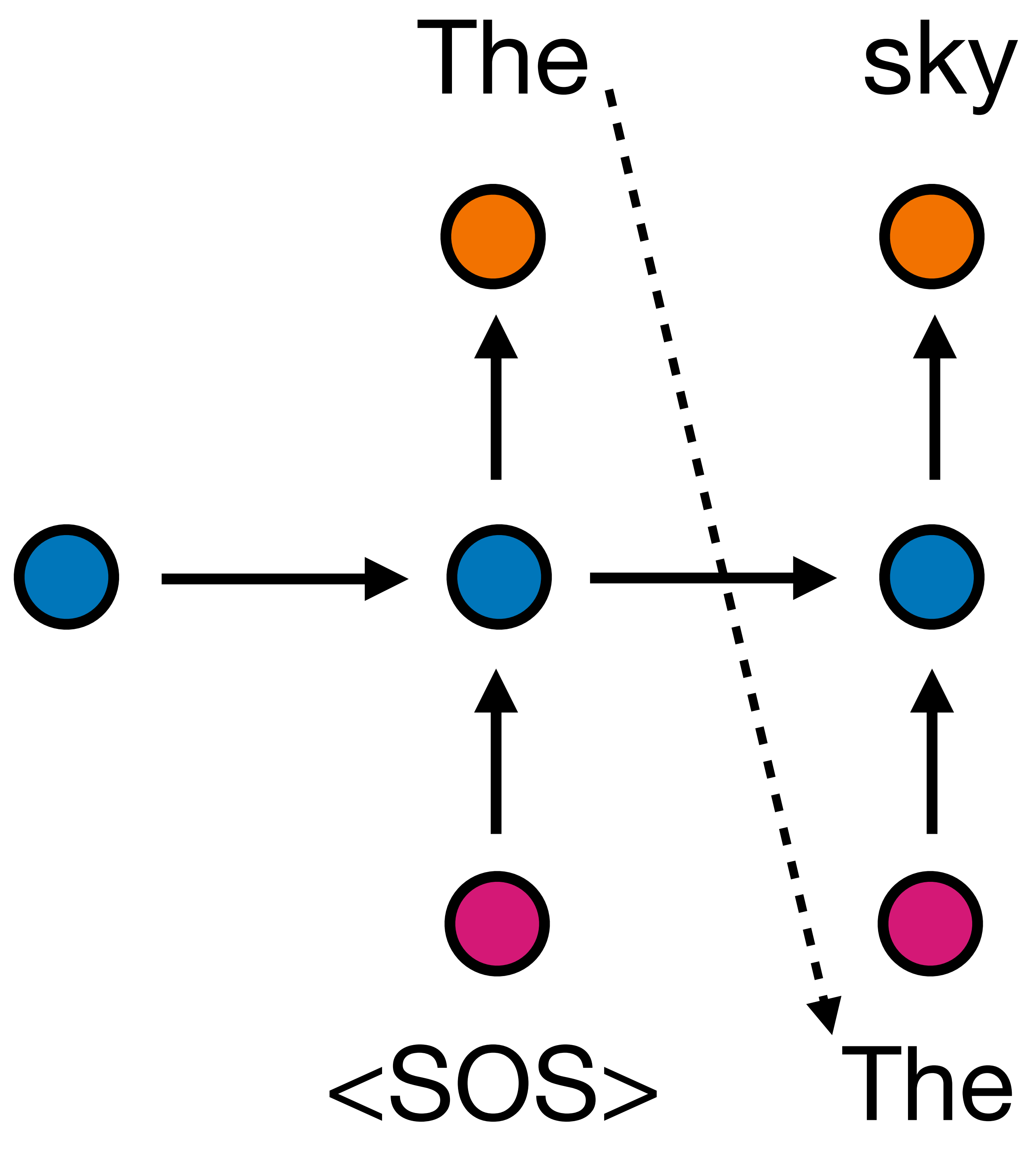
RNNs for autoregressive text generation



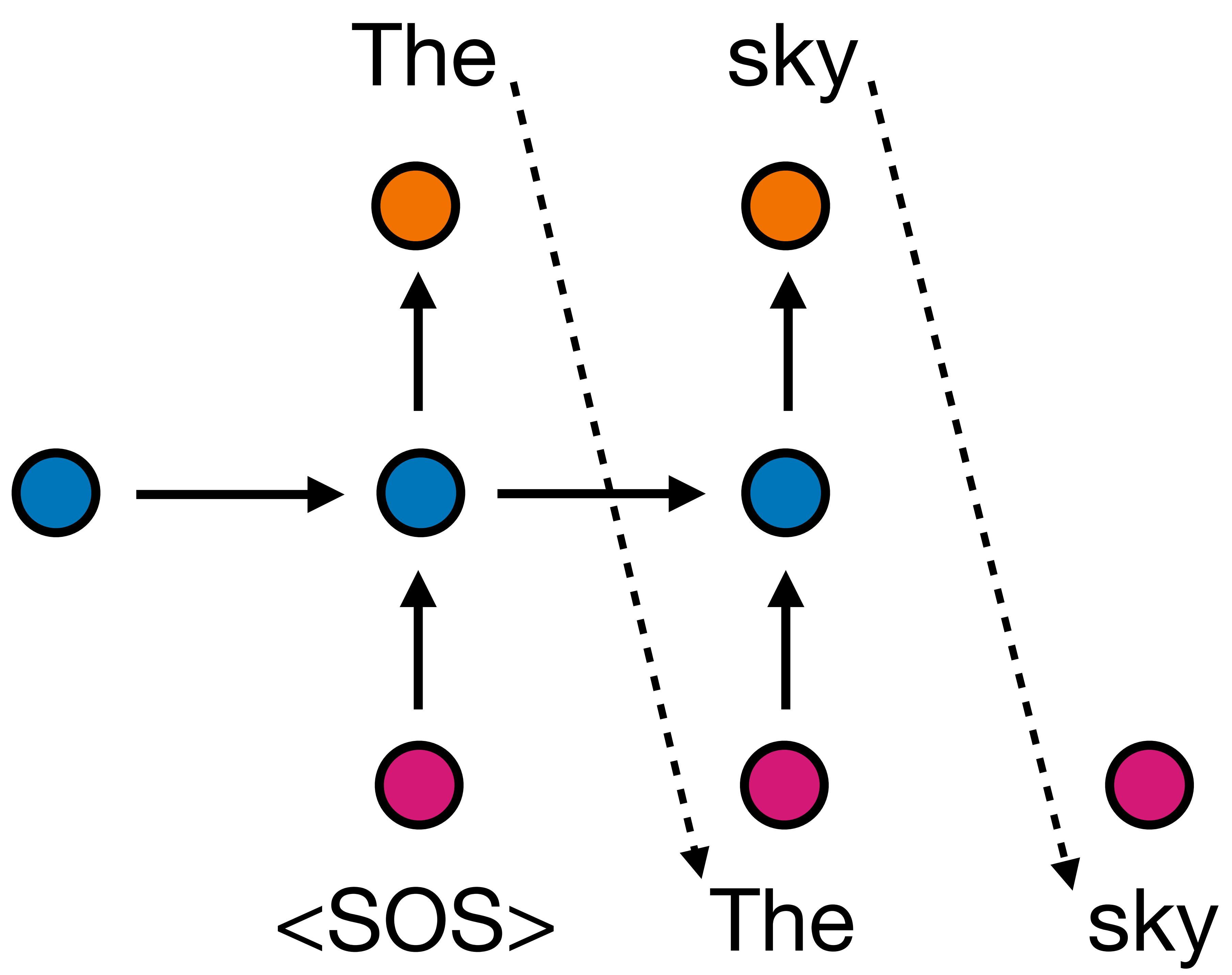
RNNs for autoregressive text generation



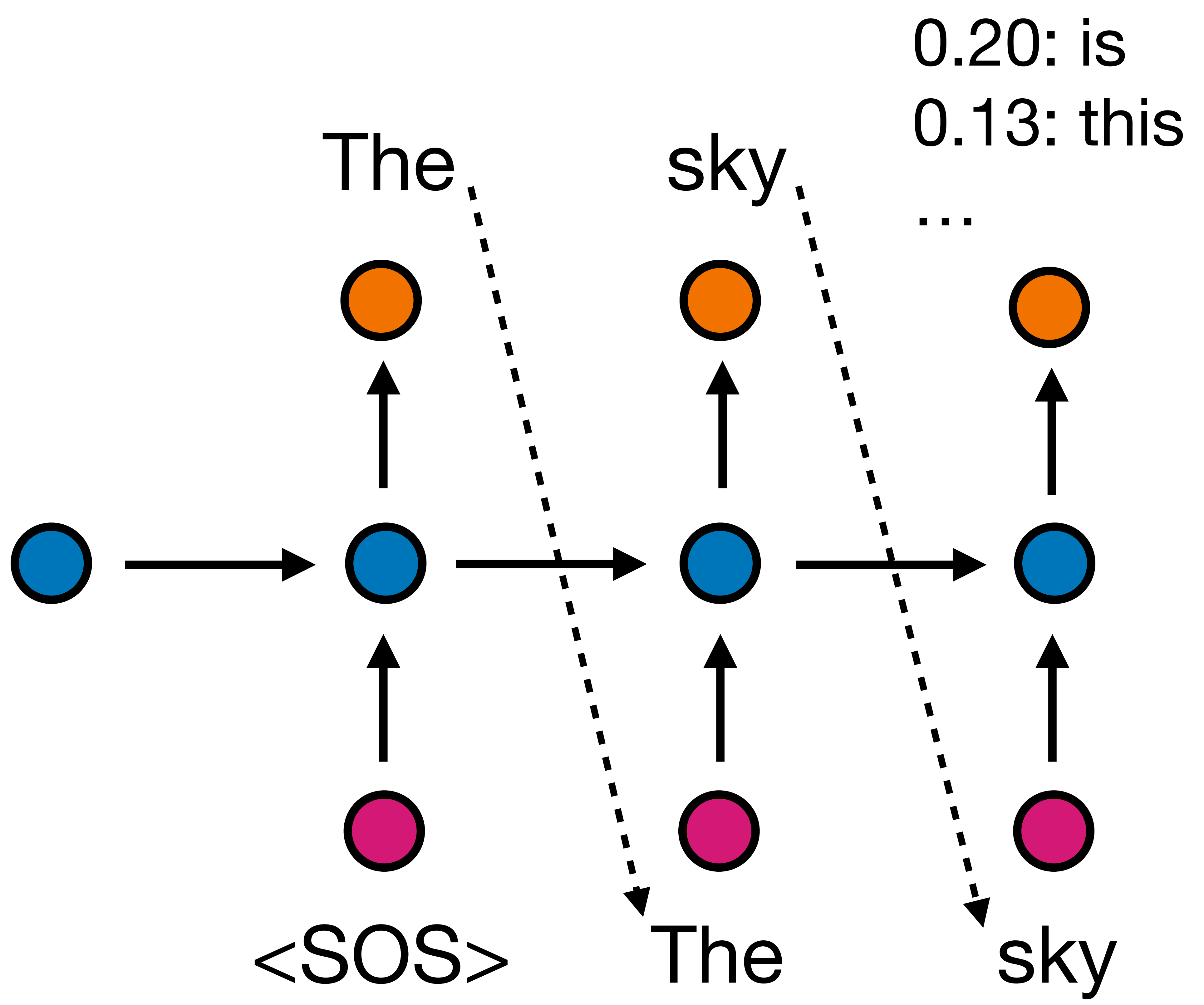
RNNs for autoregressive text generation



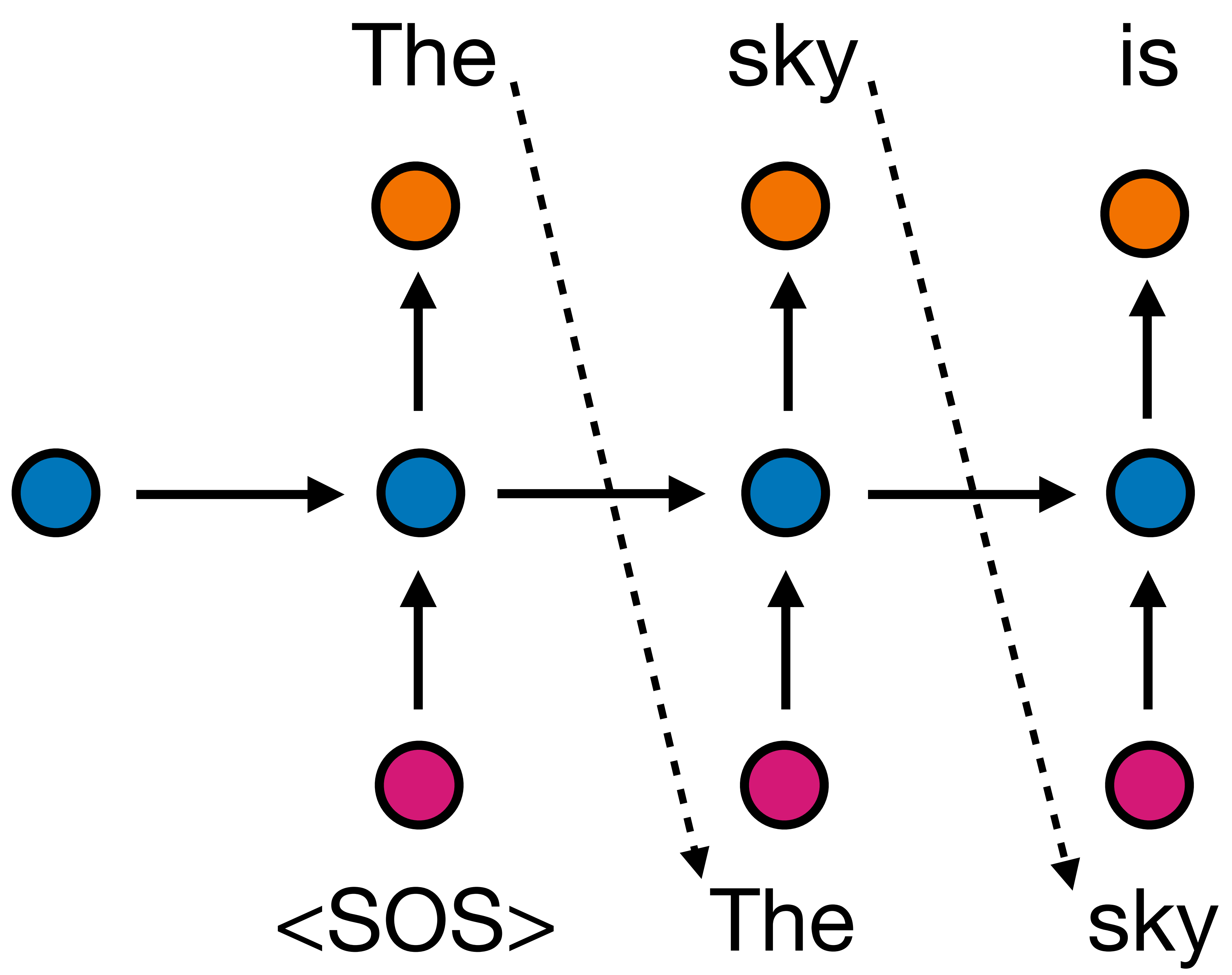
RNNs for autoregressive text generation



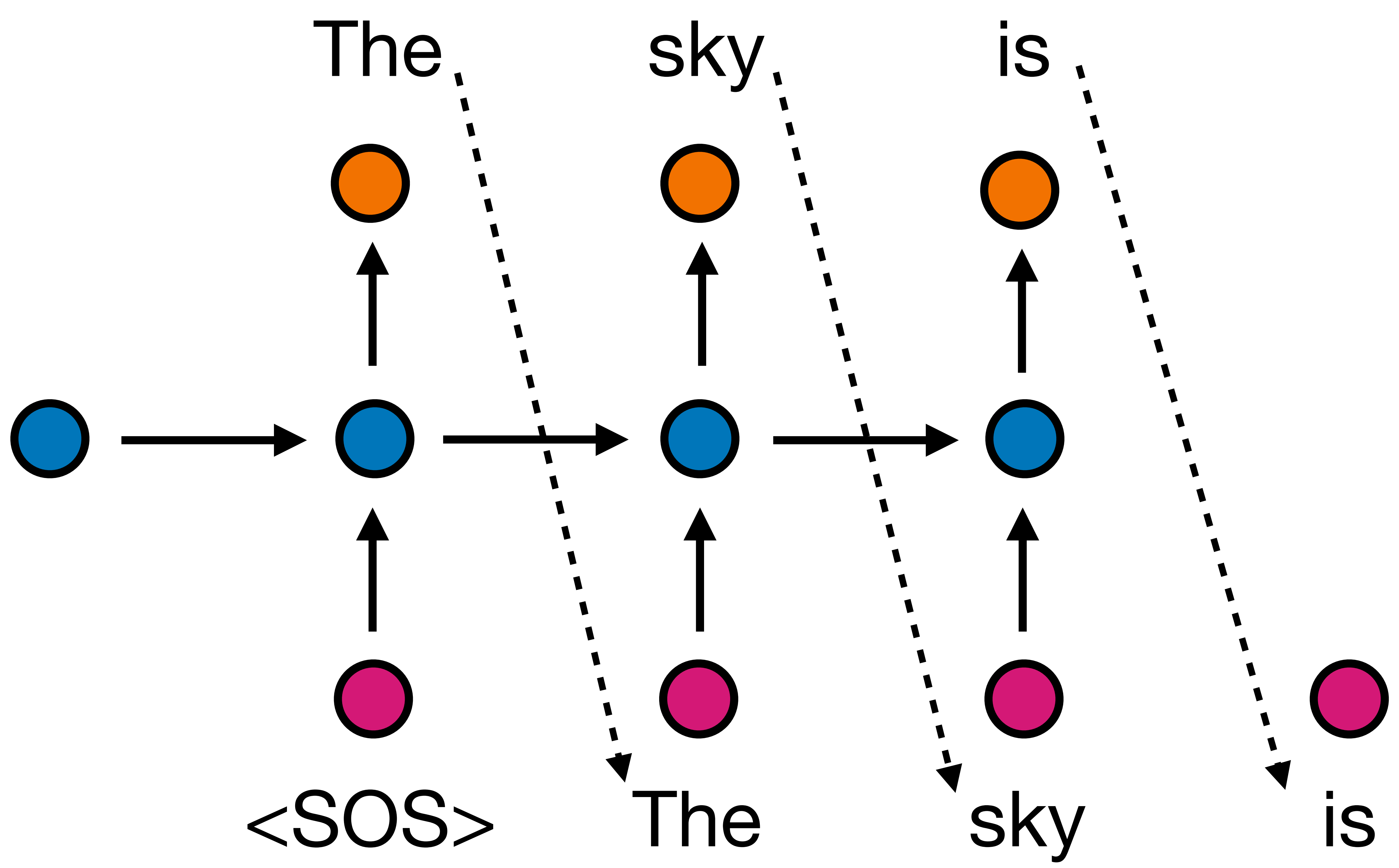
RNNs for autoregressive text generation



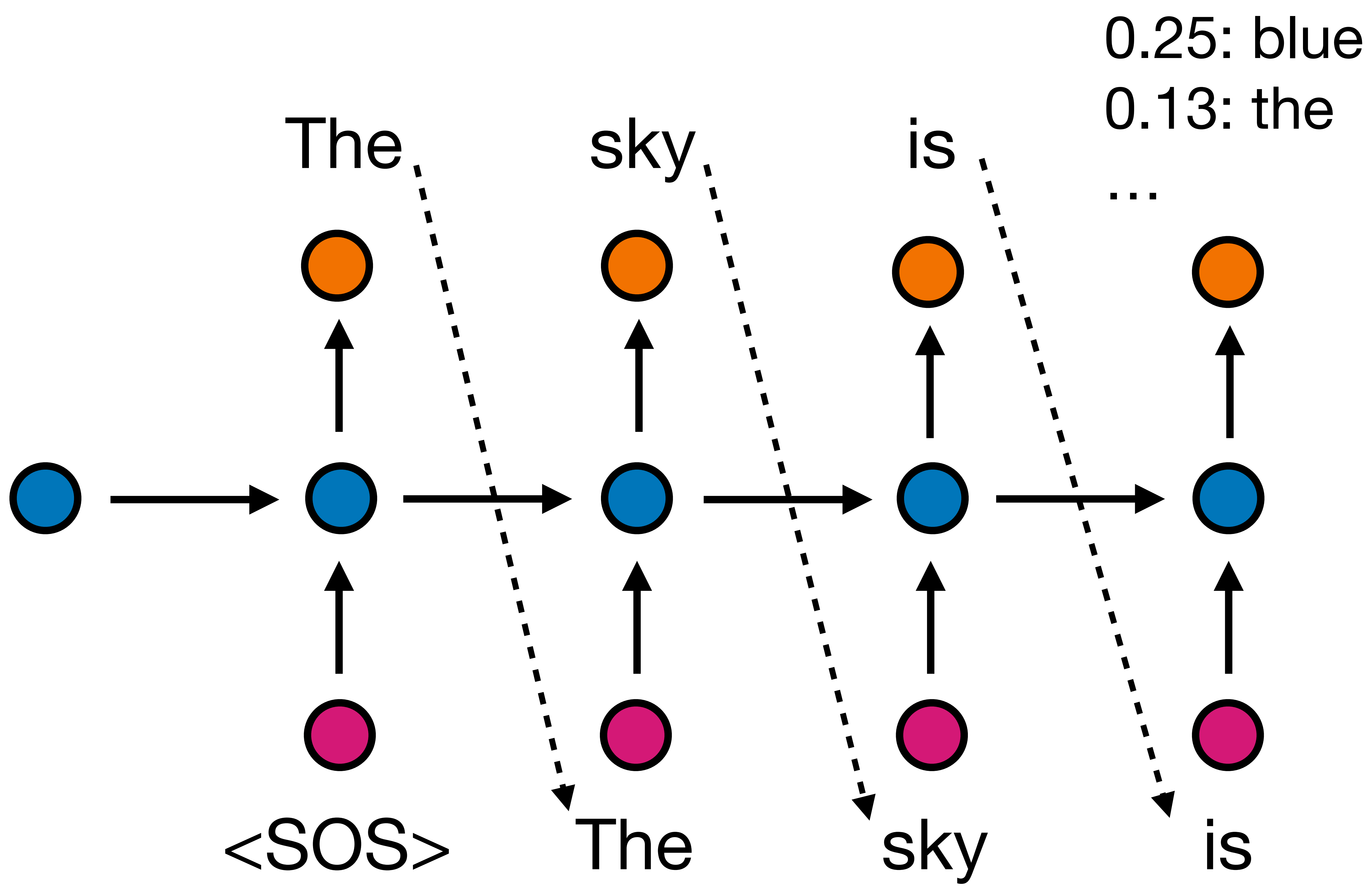
RNNs for autoregressive text generation



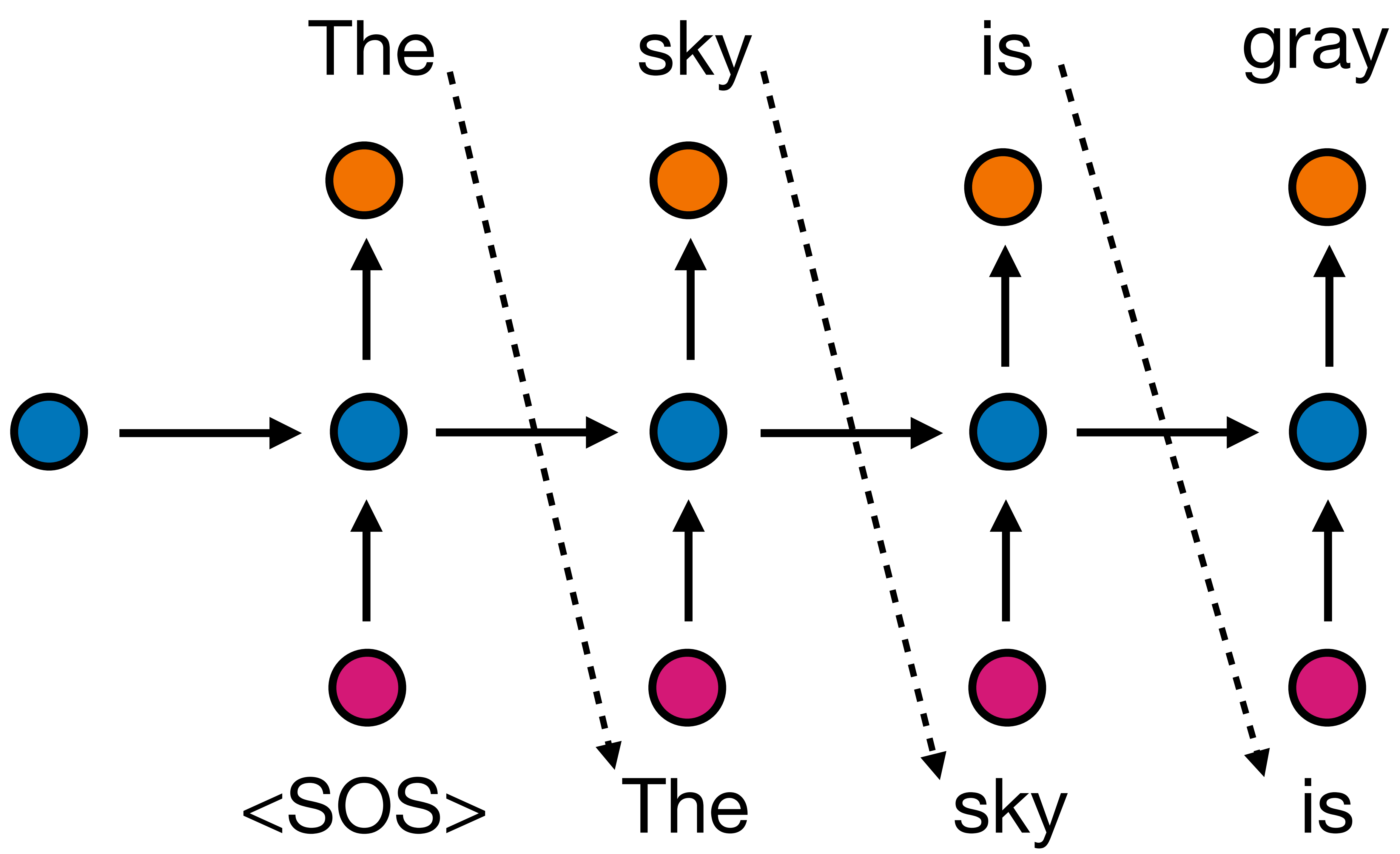
RNNs for autoregressive text generation



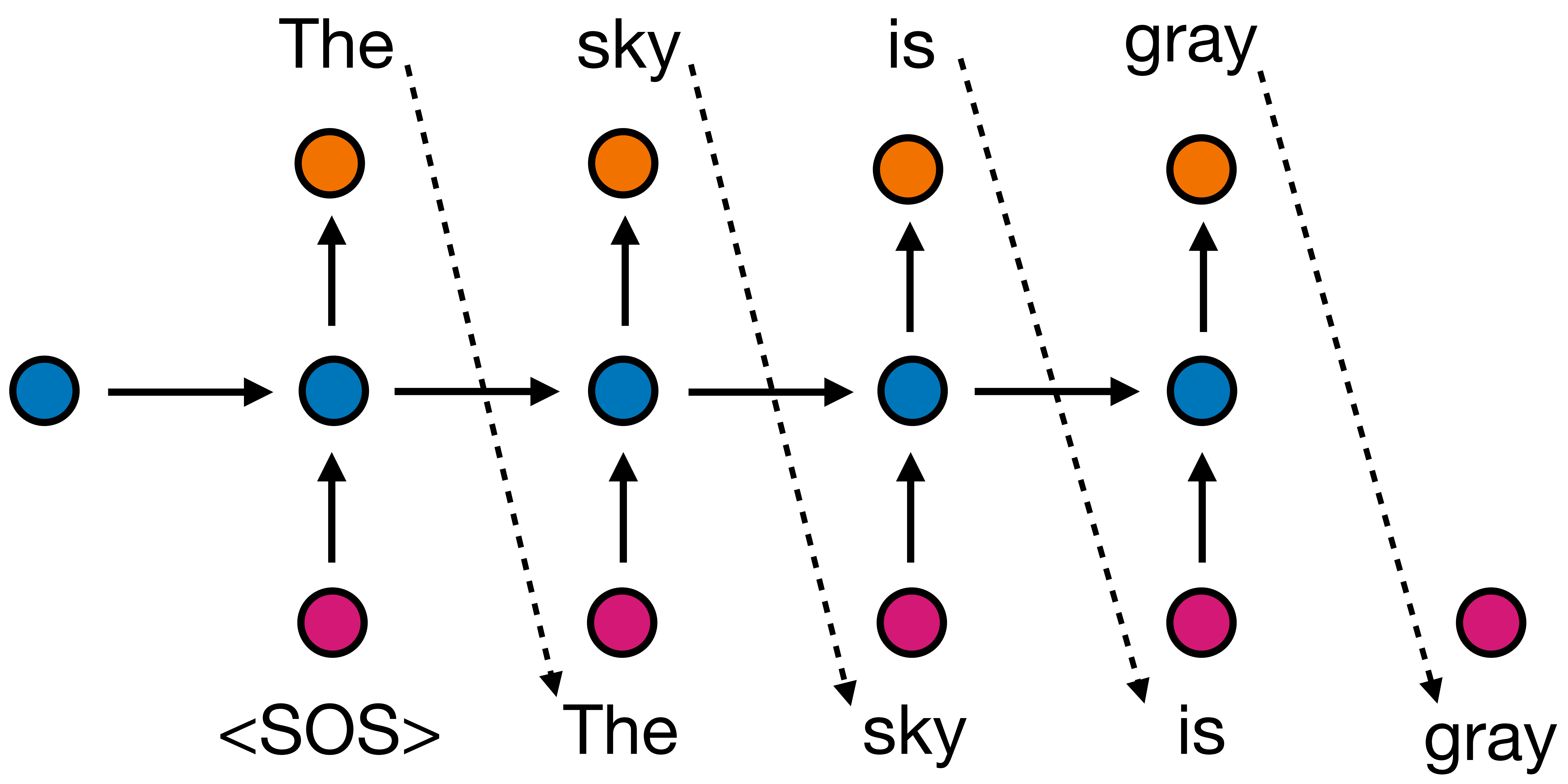
RNNs for autoregressive text generation



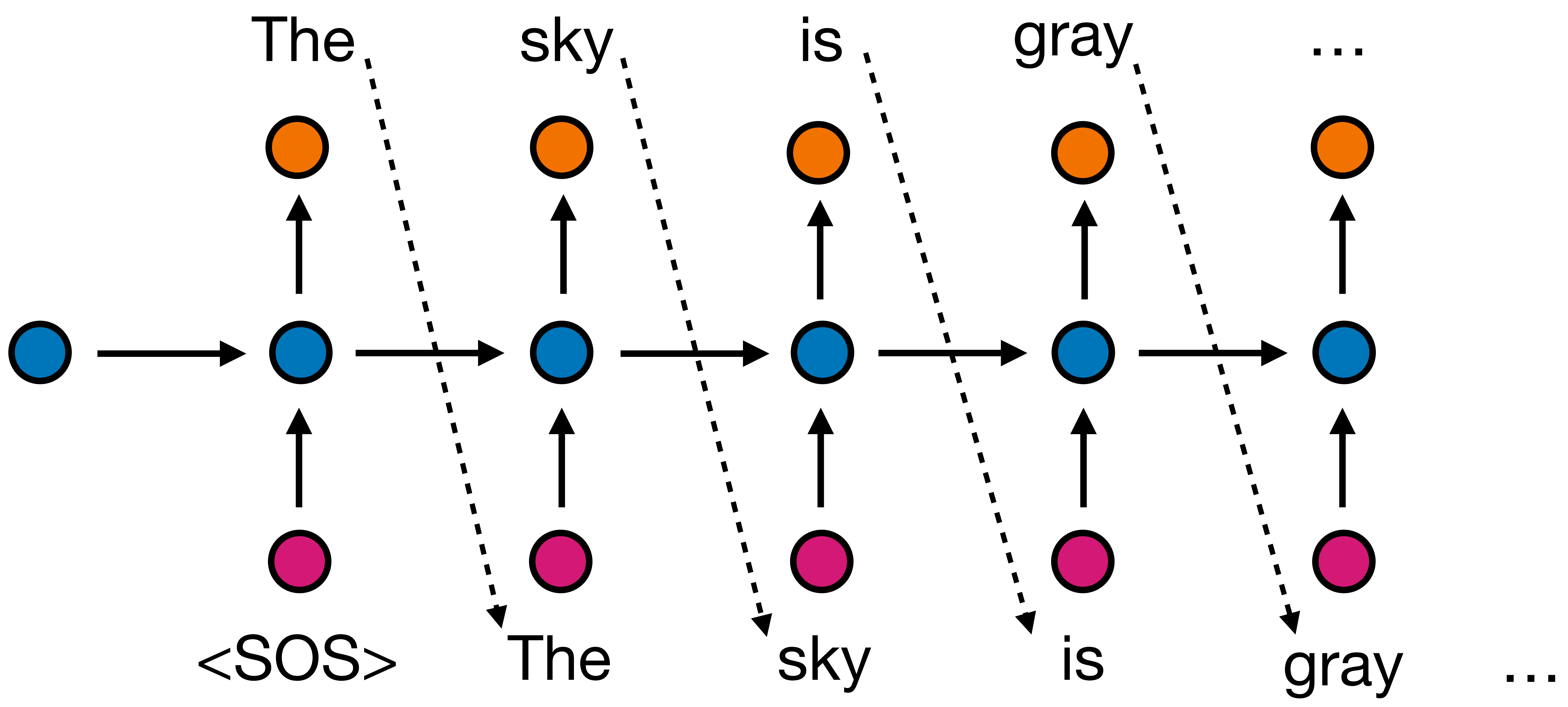
RNNs for autoregressive text generation



RNNs for autoregressive text generation



RNNs for autoregressive text generation



RNN for machine translation

RNN for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

RNN for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

RNN for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

Encoder RNN

RNN for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

Encoder RNN

RNN for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

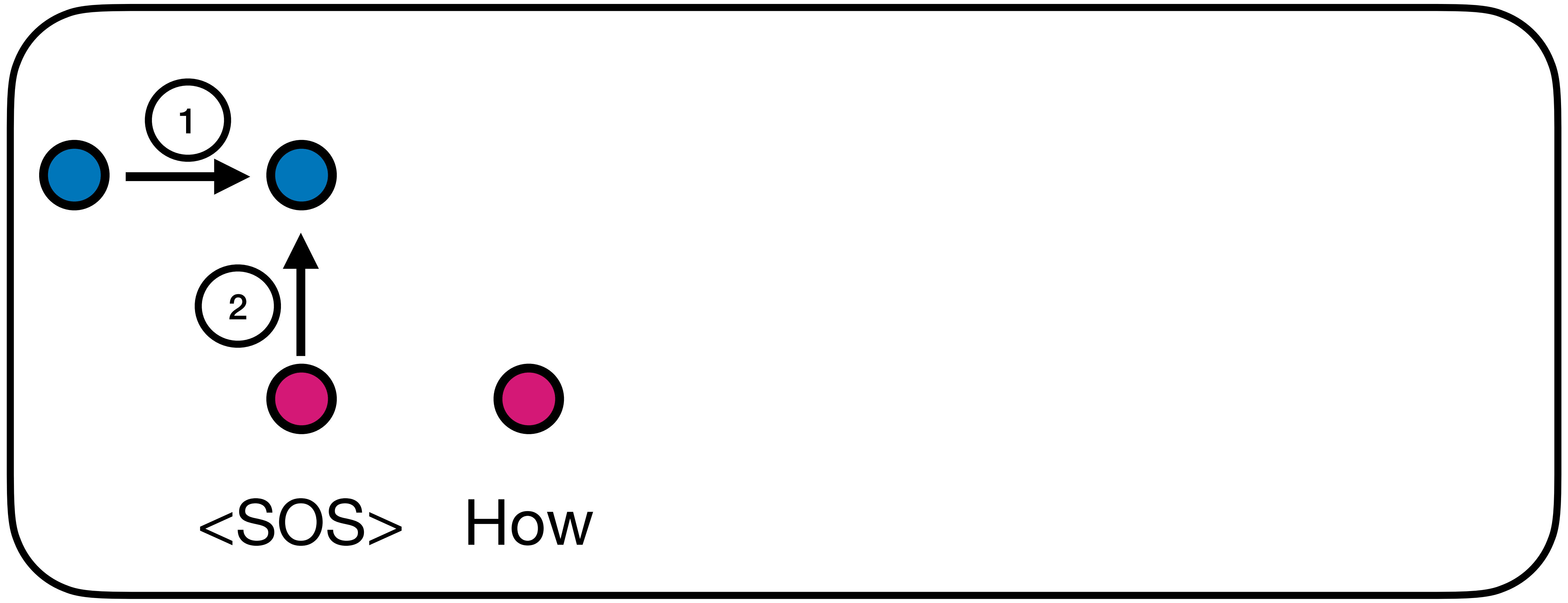
1. Pass input sentence through *encoder RNN.*

Encoder RNN

RNN for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

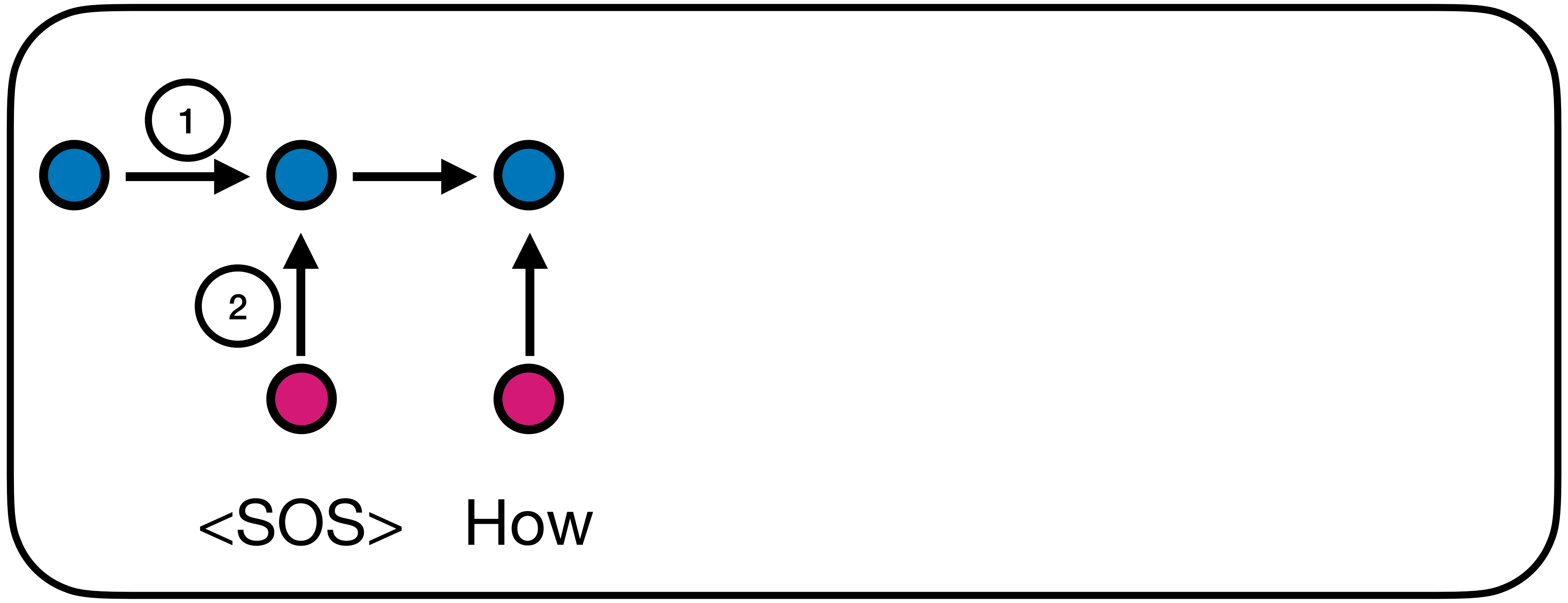


Encoder RNN

RNN for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

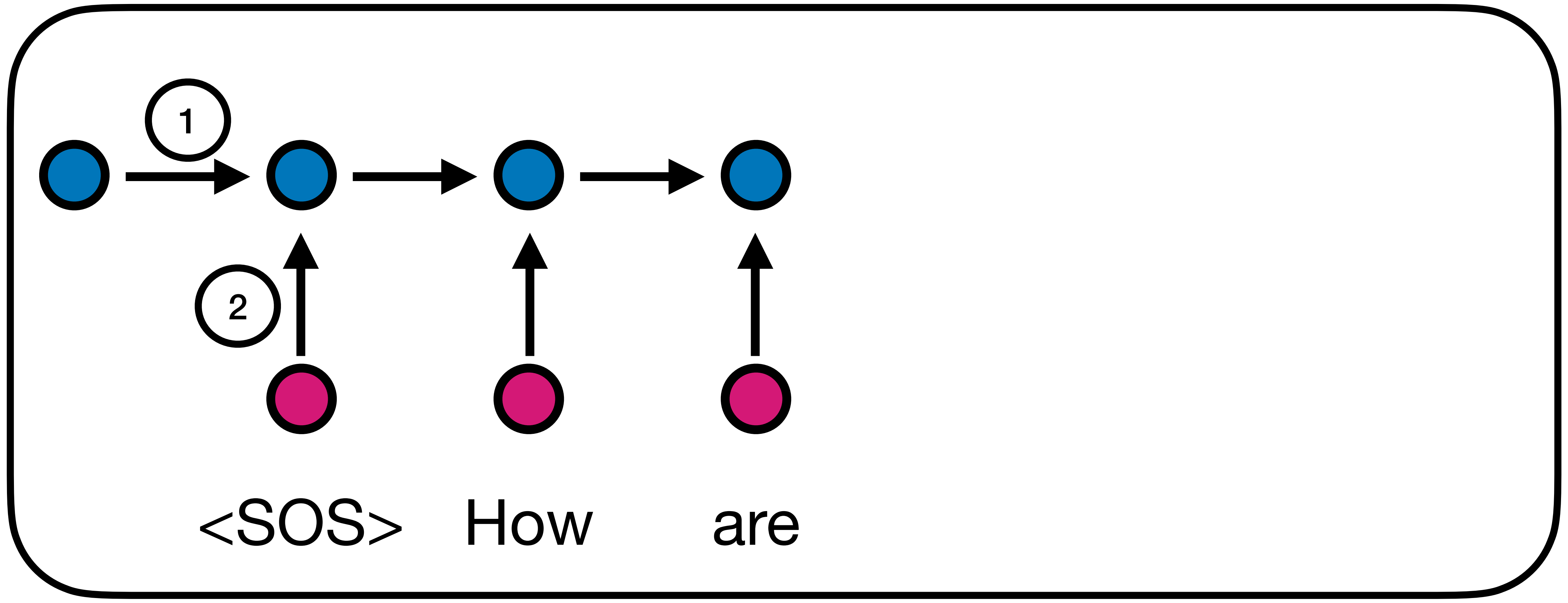


Encoder RNN

RNN for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

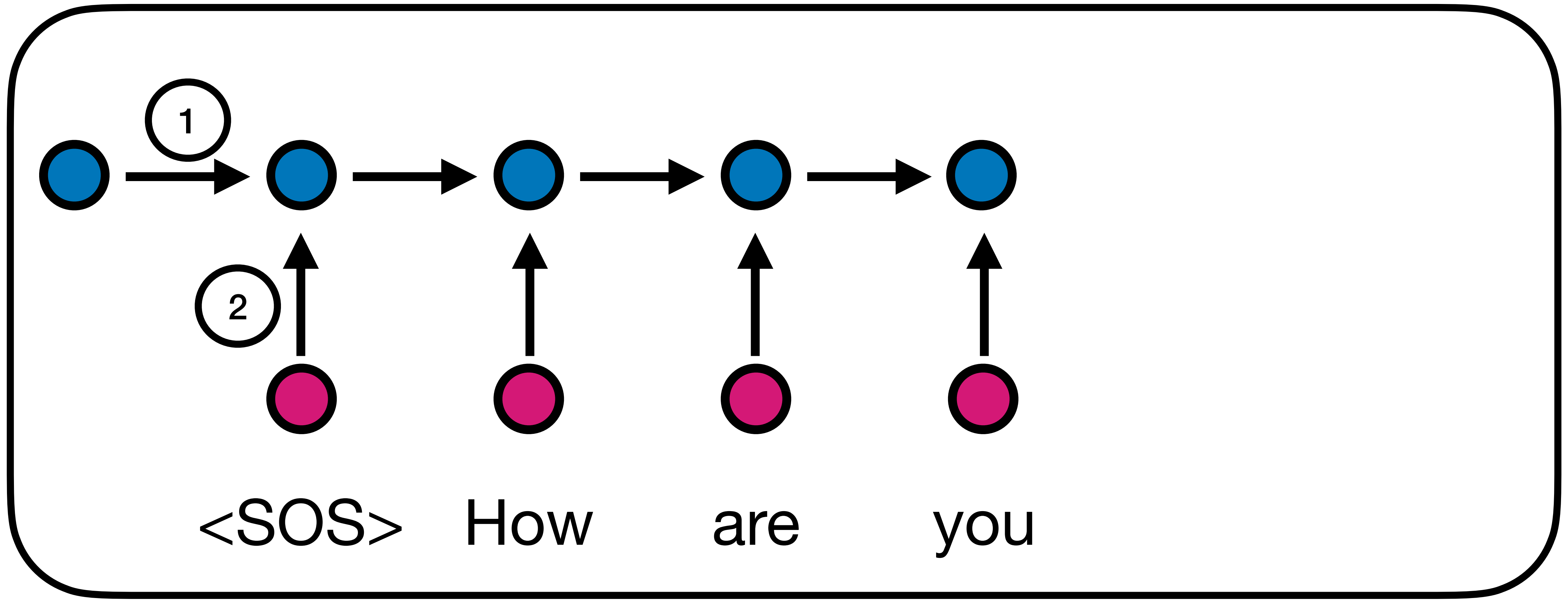


Encoder RNN

RNN for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

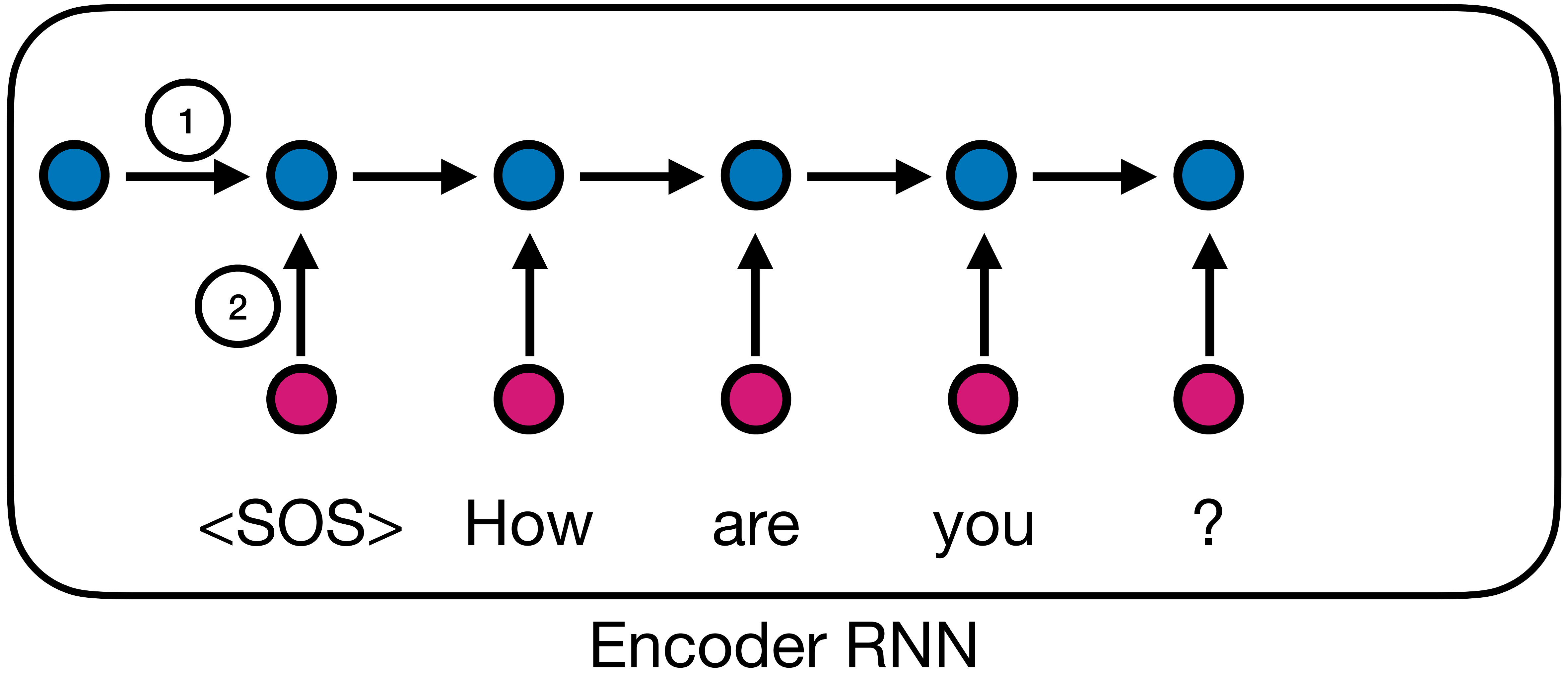


Encoder RNN

RNN for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

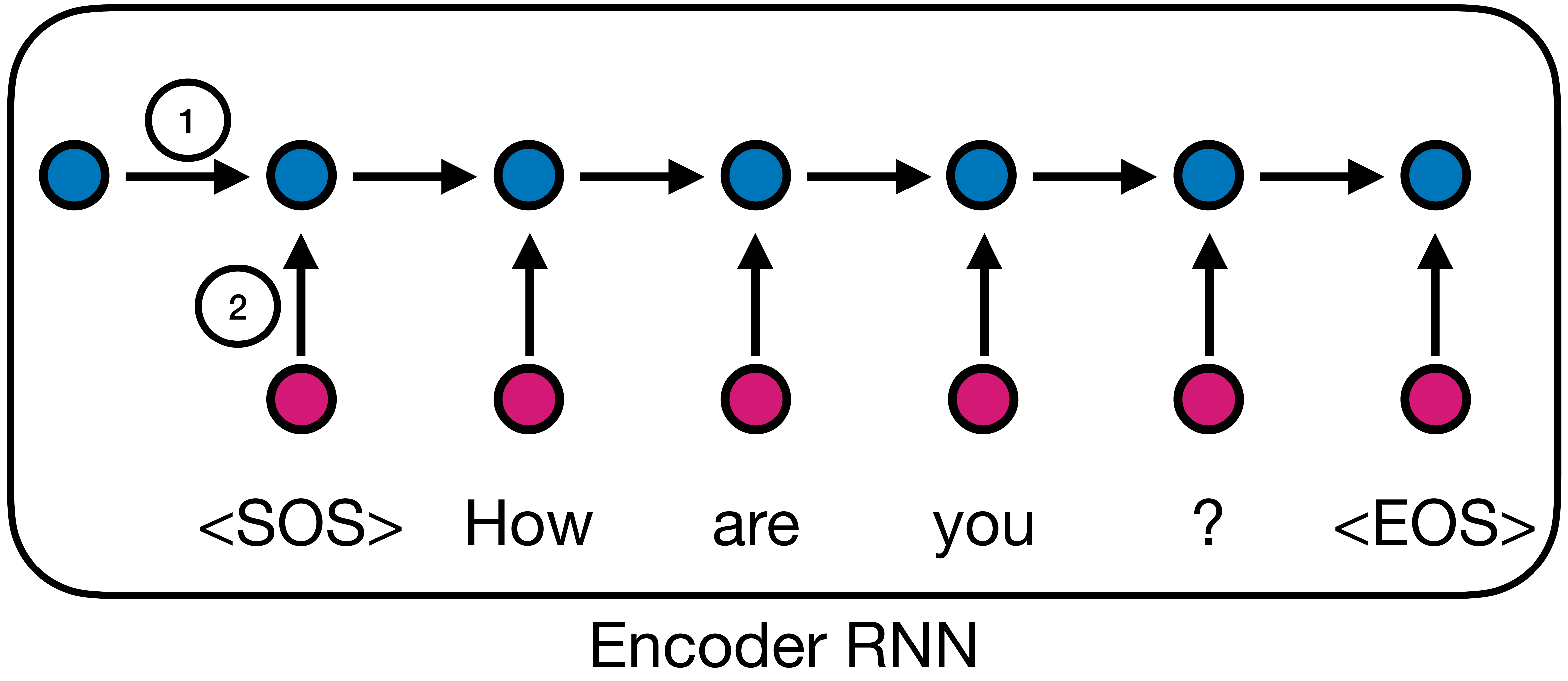
1. Pass input sentence through *encoder RNN.*



RNN for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

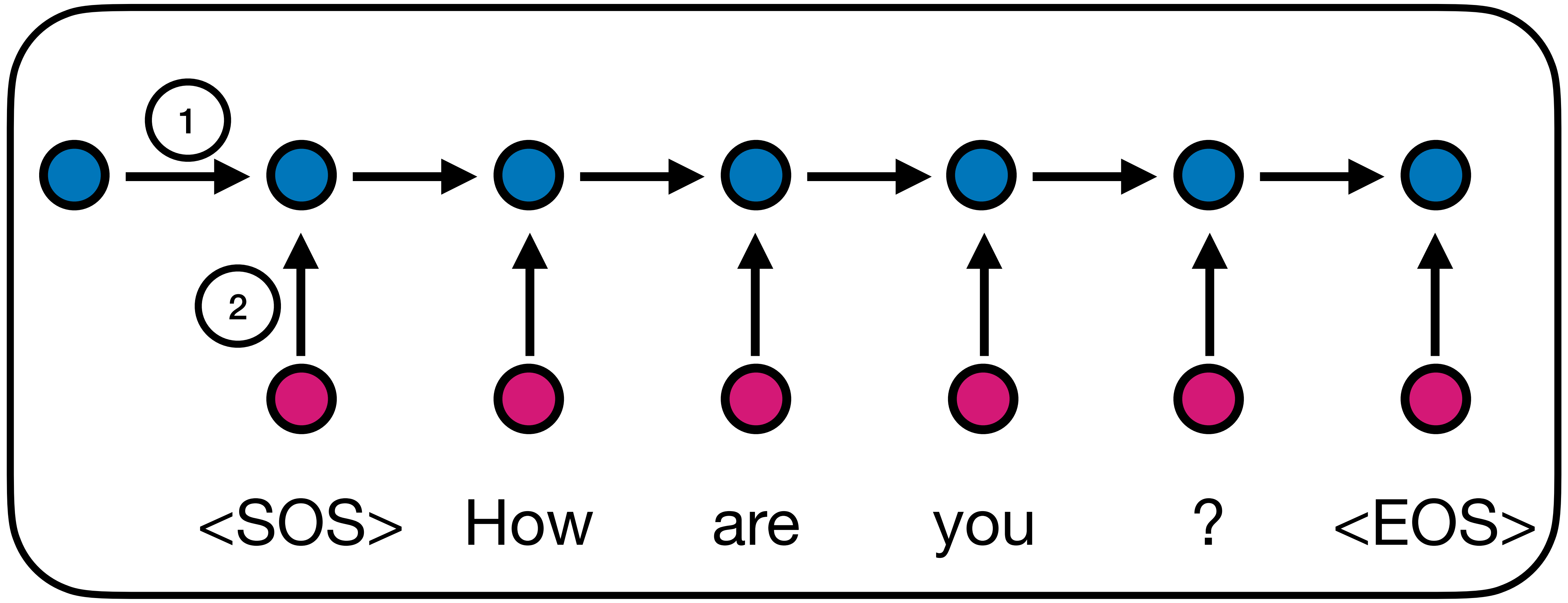


RNN for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

2. Find the last hidden state of the encoder RNN, called the *context vector*.



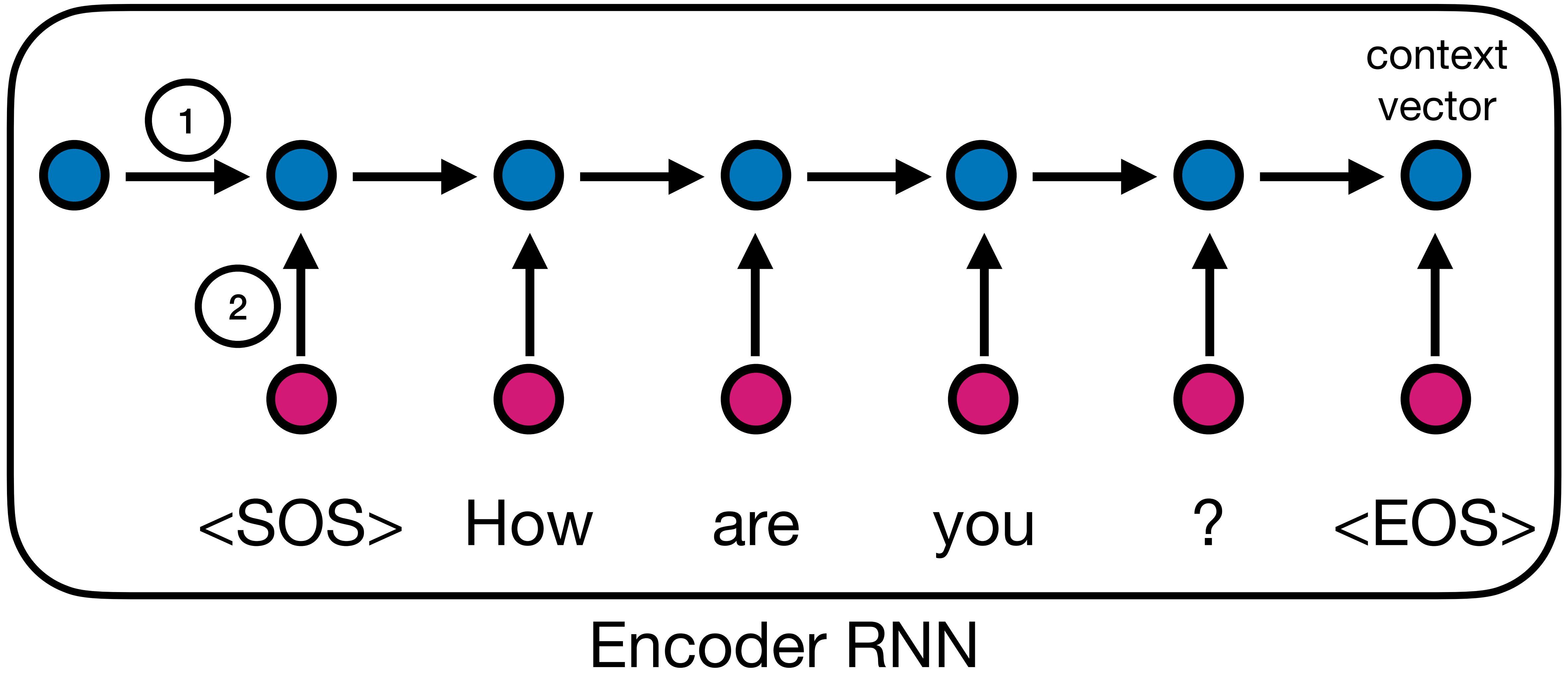
Encoder RNN

RNN for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

2. Find the last hidden state of the encoder RNN, called the *context vector*.



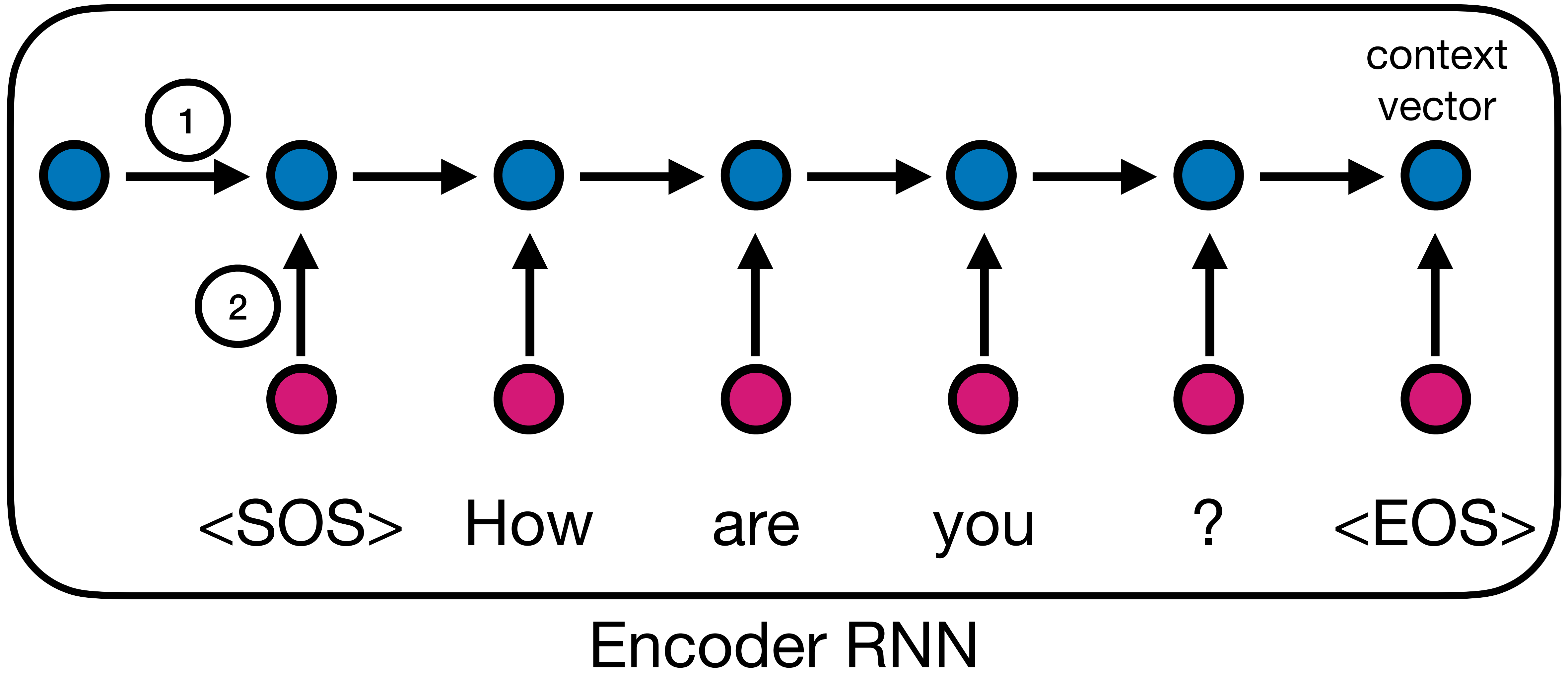
RNN for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

2. Find the last hidden state of the encoder RNN, called the *context vector*.

3. Copy the context vector into the first hidden state of a *decoder RNN.*



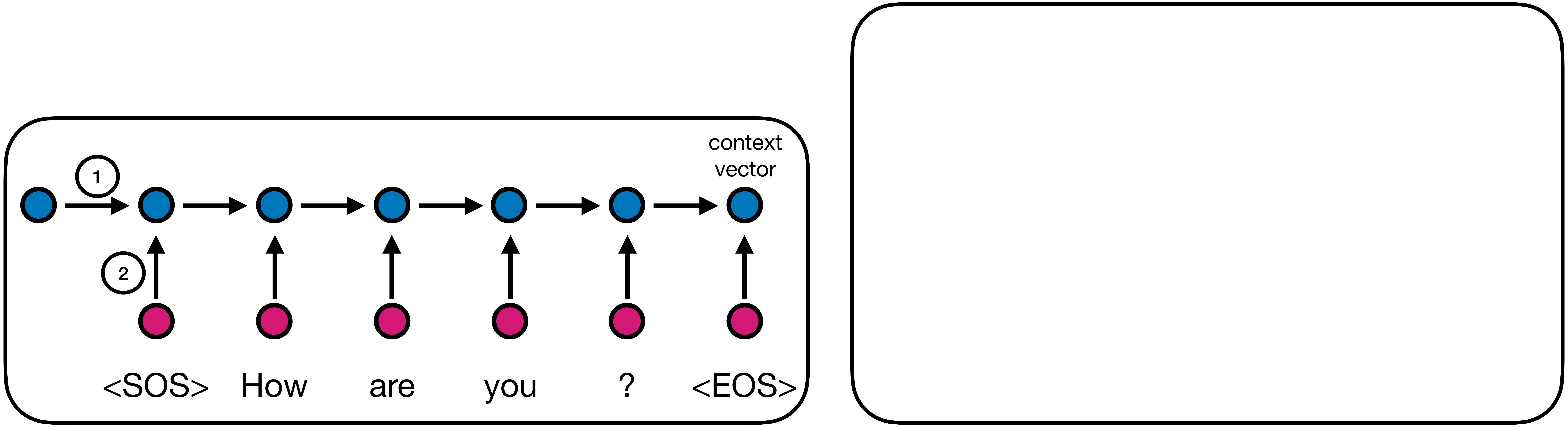
RNN for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

2. Find the last hidden state of the encoder RNN, called the *context vector*.

3. Copy the context vector into the first hidden state of a *decoder RNN.*



Encoder RNN

Decoder RNN

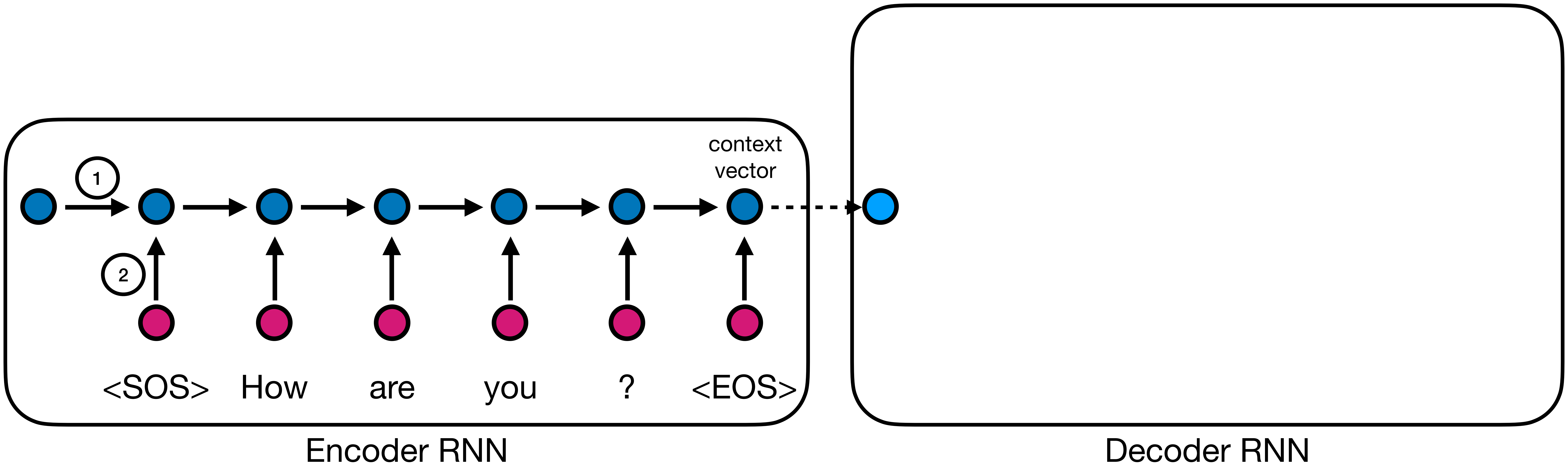
RNN for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

2. Find the last hidden state of the encoder RNN, called the *context vector*.

3. Copy the context vector into the first hidden state of a *decoder RNN.*



RNN for machine translation

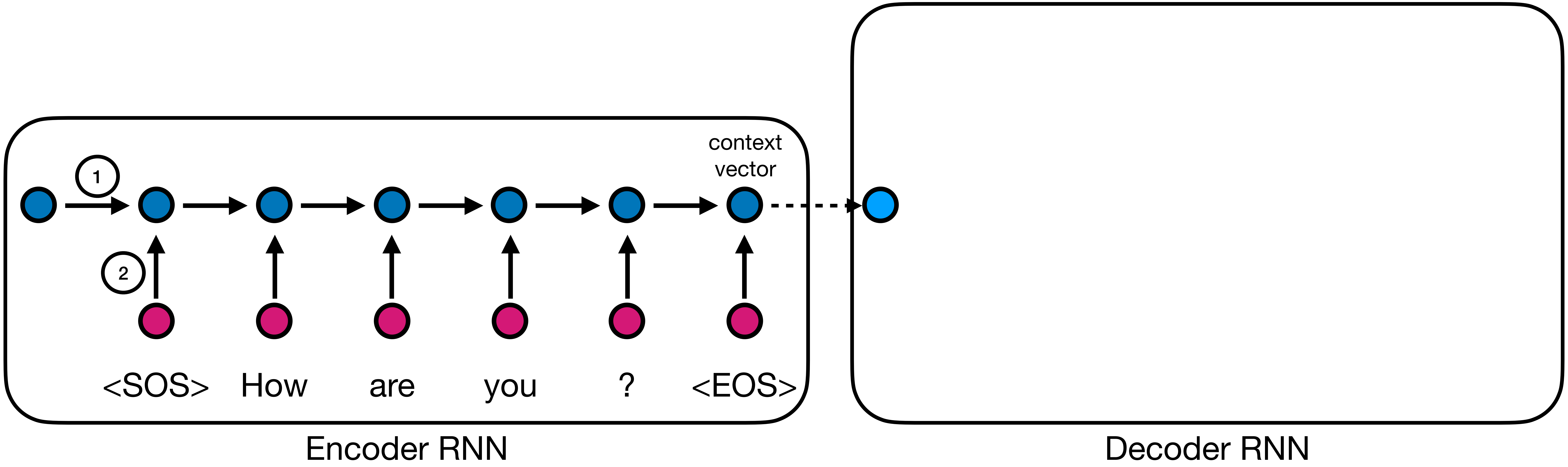
Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

2. Find the last hidden state of the encoder RNN, called the *context vector*.

3. Copy the context vector into the first hidden state of a *decoder RNN.*

4. Use the decoder RNN to generate the translation.



RNN for machine translation

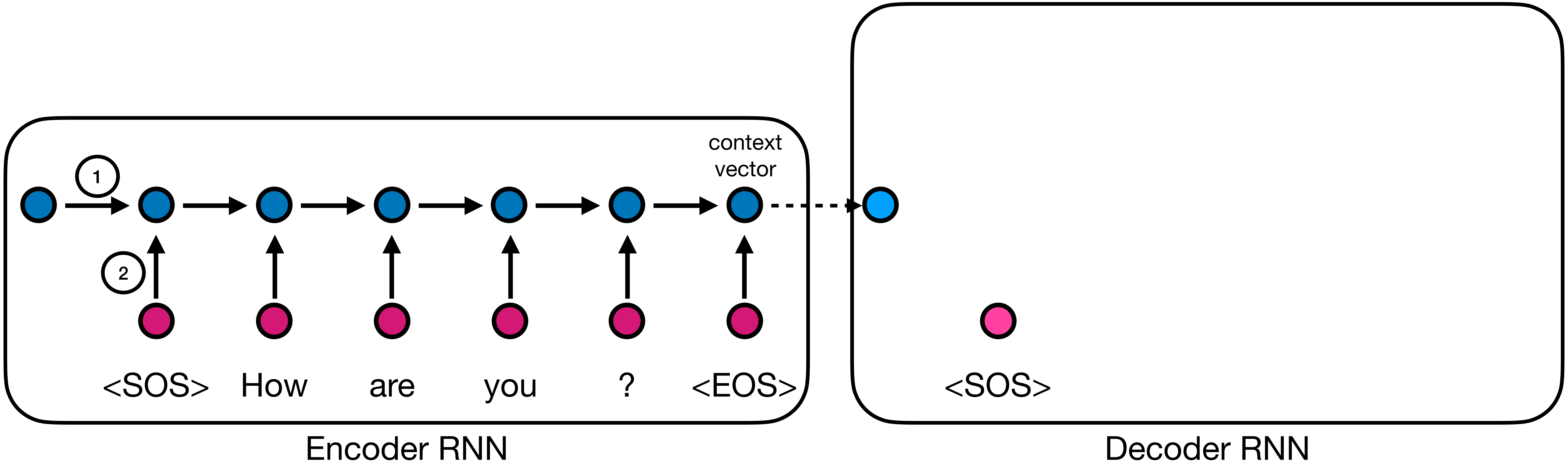
Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

2. Find the last hidden state of the encoder RNN, called the *context vector*.

3. Copy the context vector into the first hidden state of a *decoder RNN.*

4. Use the decoder RNN to generate the translation.



RNN for machine translation

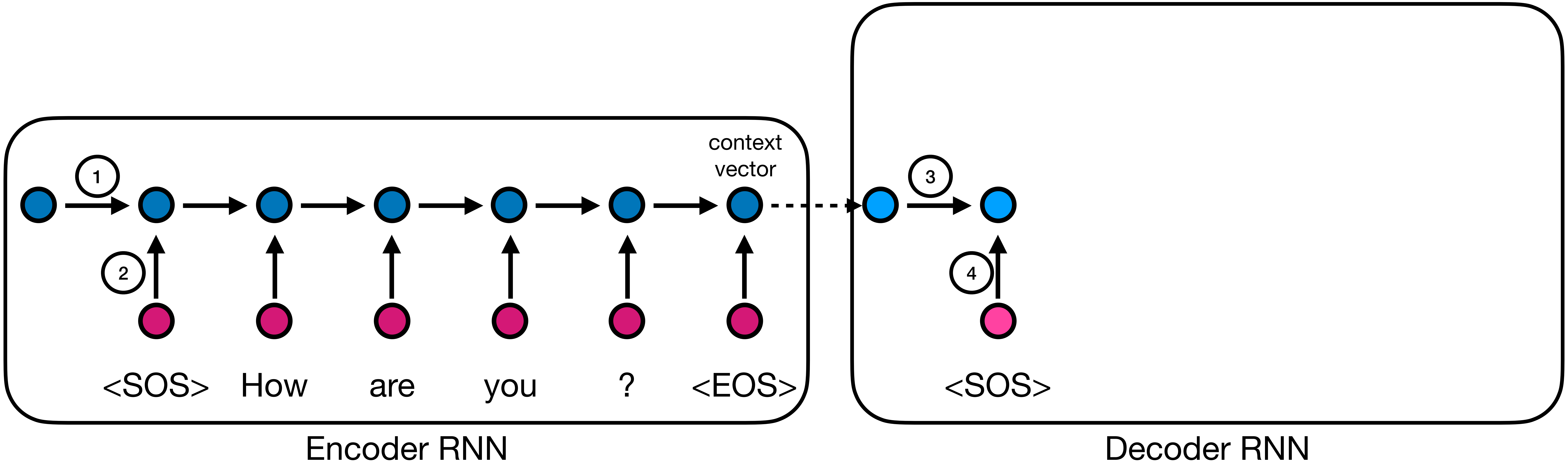
Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

2. Find the last hidden state of the encoder RNN, called the *context vector*.

3. Copy the context vector into the first hidden state of a *decoder RNN.*

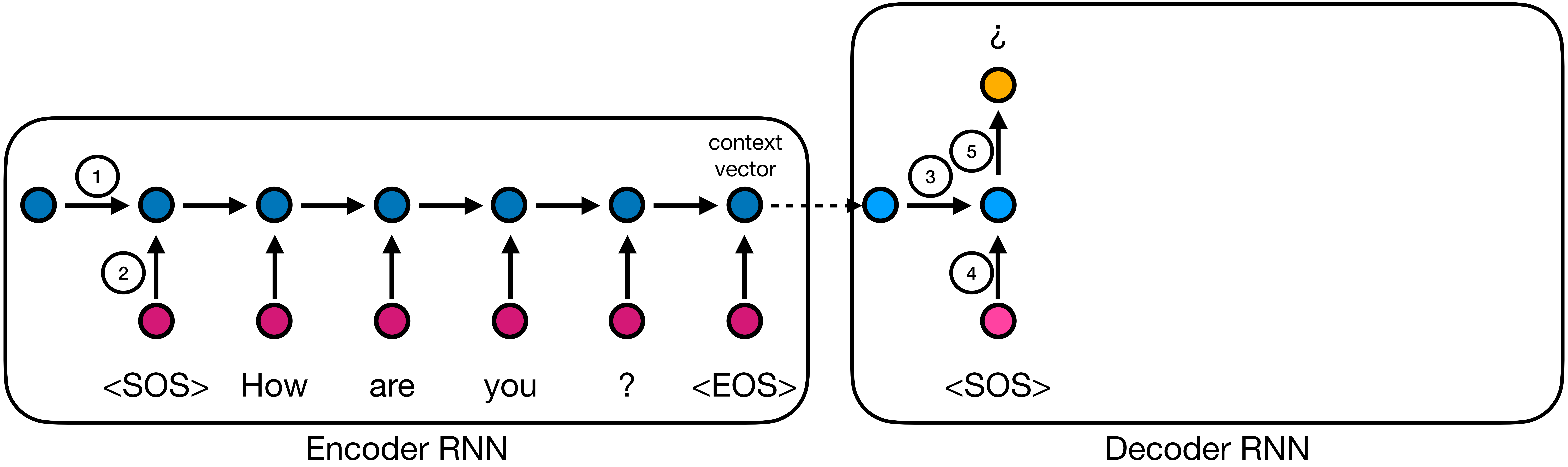
4. Use the decoder RNN to generate the translation.



RNN for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*
2. Find the last hidden state of the encoder RNN, called the *context vector*.
3. Copy the context vector into the first hidden state of a *decoder RNN.*
4. Use the decoder RNN to generate the translation.



RNN for machine translation

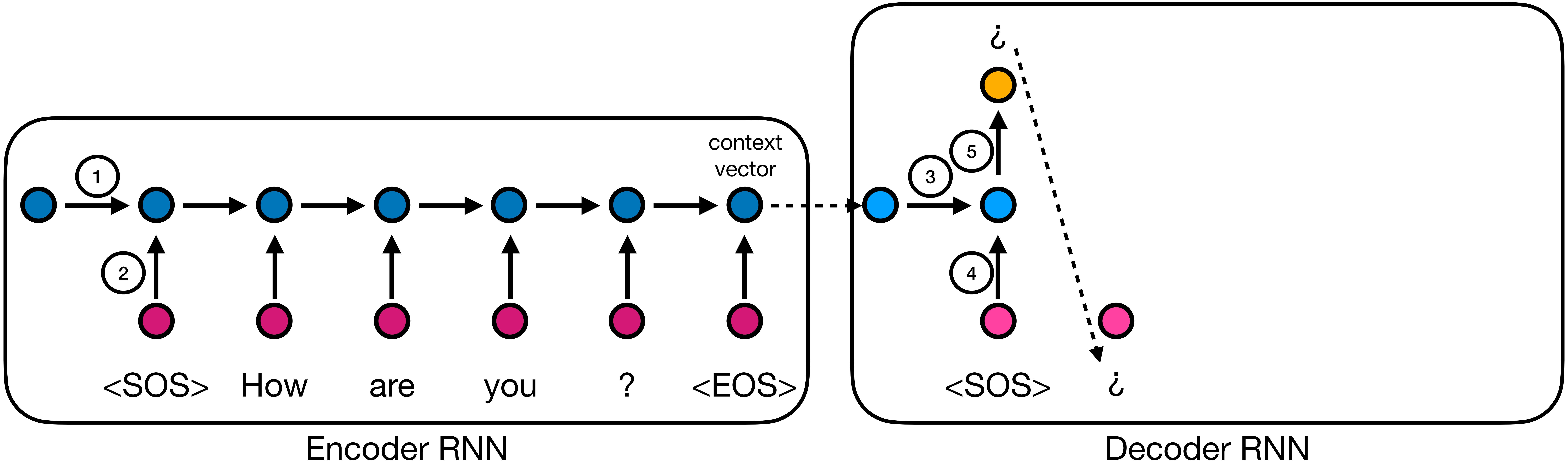
Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

2. Find the last hidden state of the encoder RNN, called the *context vector*.

3. Copy the context vector into the first hidden state of a *decoder RNN.*

4. Use the decoder RNN to generate the translation.



RNN for machine translation

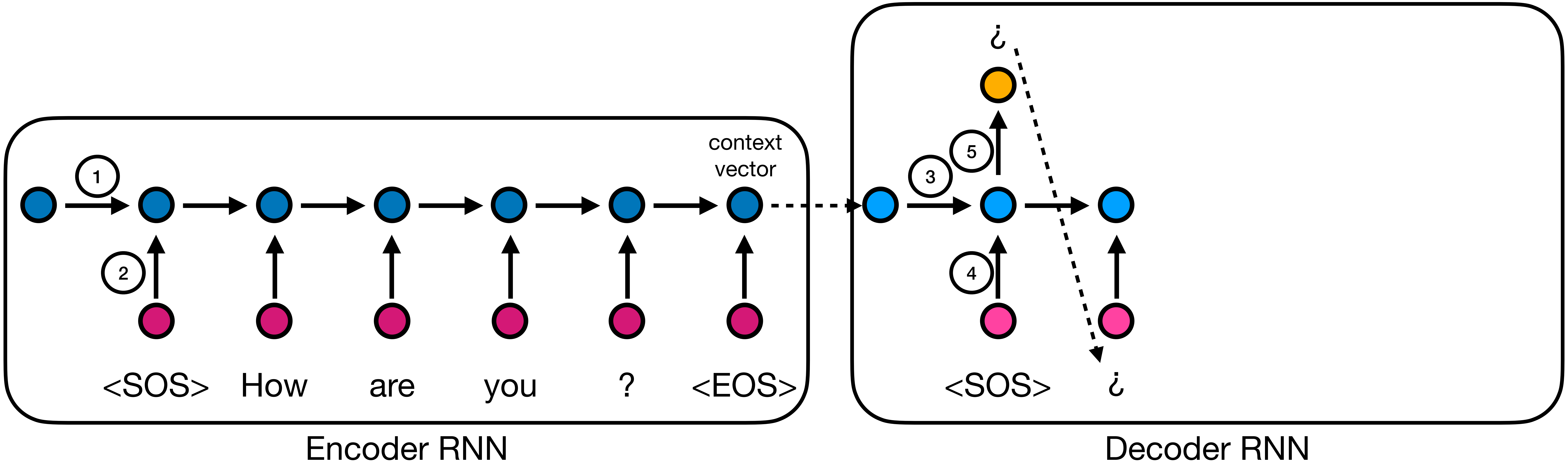
Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

2. Find the last hidden state of the encoder RNN, called the *context vector*.

3. Copy the context vector into the first hidden state of a *decoder RNN.*

4. Use the decoder RNN to generate the translation.



RNN for machine translation

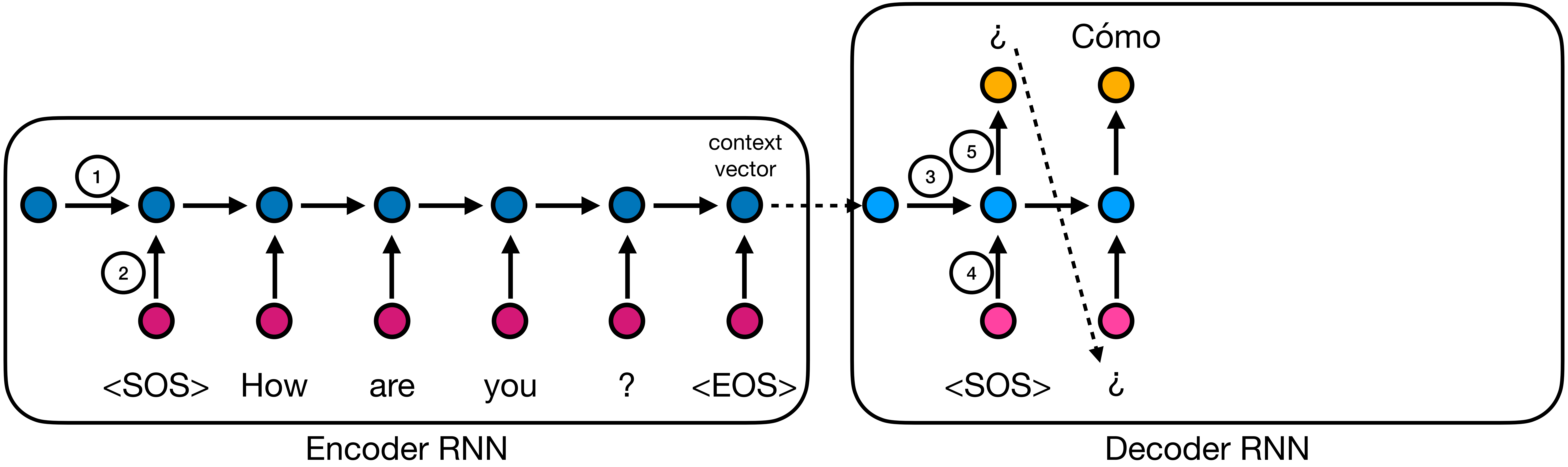
Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

2. Find the last hidden state of the encoder RNN, called the *context vector*.

3. Copy the context vector into the first hidden state of a *decoder RNN.*

4. Use the decoder RNN to generate the translation.



RNN for machine translation

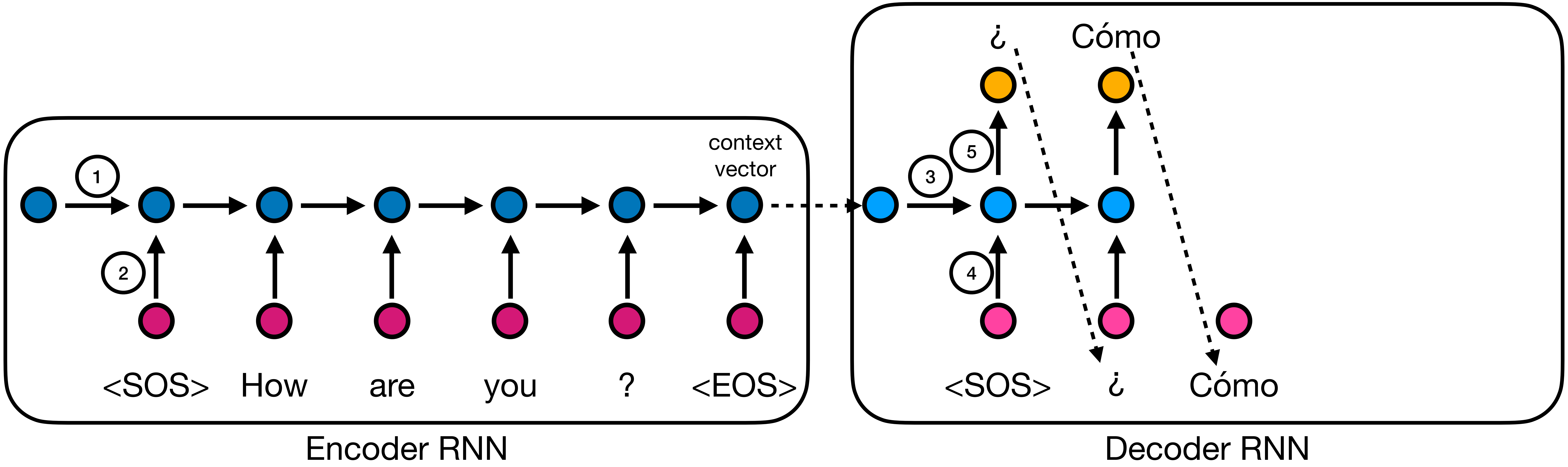
Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

2. Find the last hidden state of the encoder RNN, called the *context vector*.

3. Copy the context vector into the first hidden state of a *decoder RNN.*

4. Use the decoder RNN to generate the translation.



RNN for machine translation

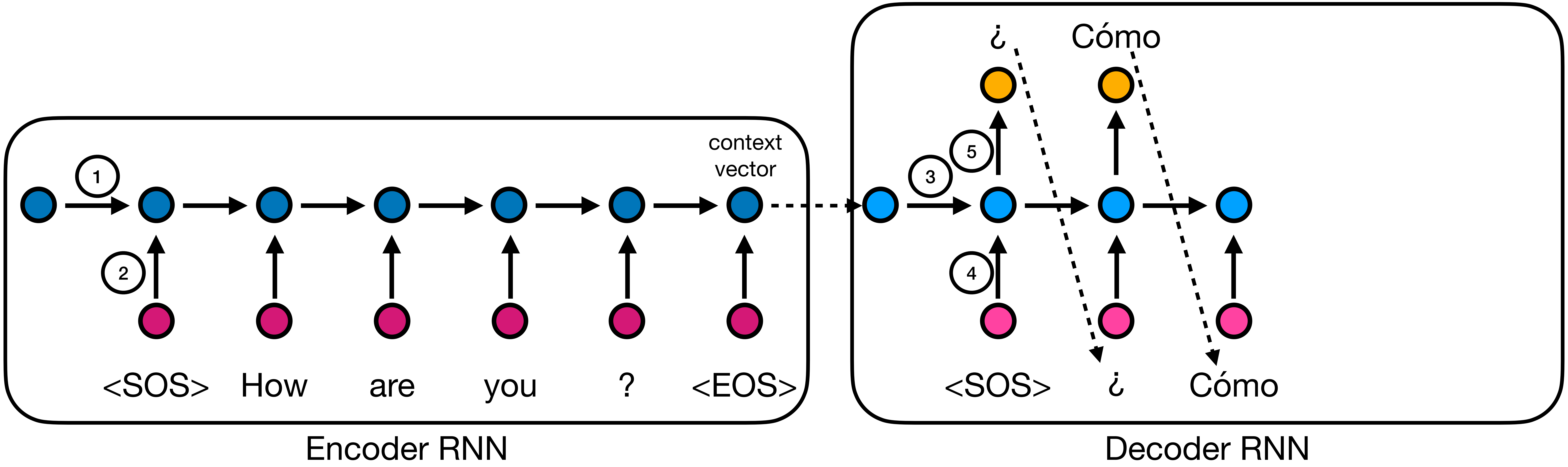
Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

2. Find the last hidden state of the encoder RNN, called the *context vector*.

3. Copy the context vector into the first hidden state of a *decoder RNN.*

4. Use the decoder RNN to generate the translation.



RNN for machine translation

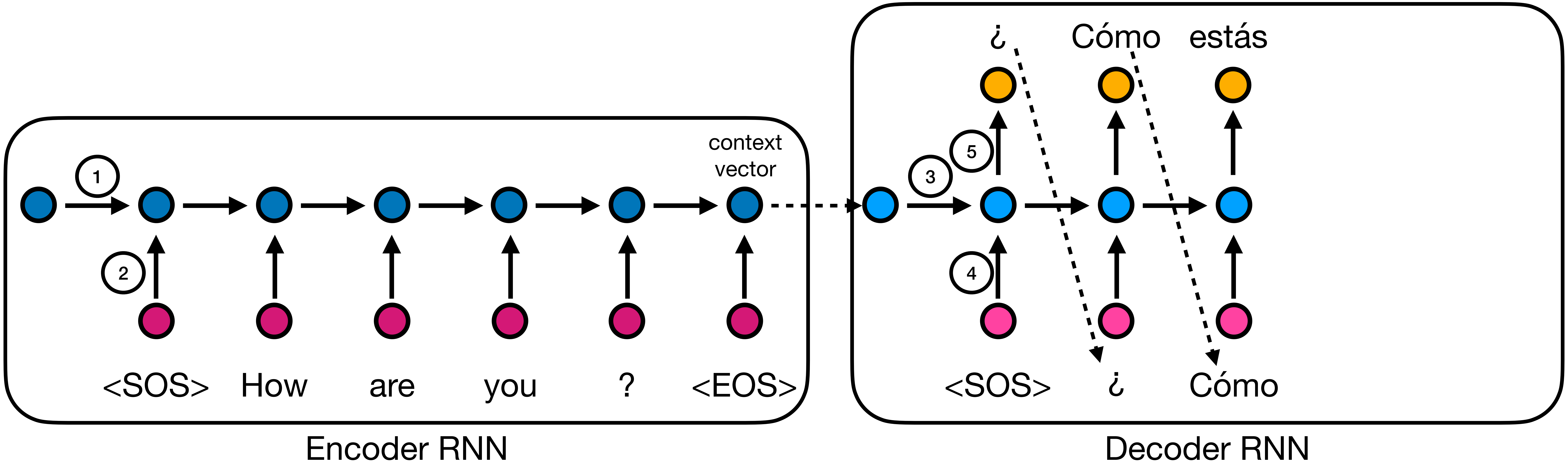
Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

2. Find the last hidden state of the encoder RNN, called the *context vector*.

3. Copy the context vector into the first hidden state of a *decoder RNN.*

4. Use the decoder RNN to generate the translation.



RNN for machine translation

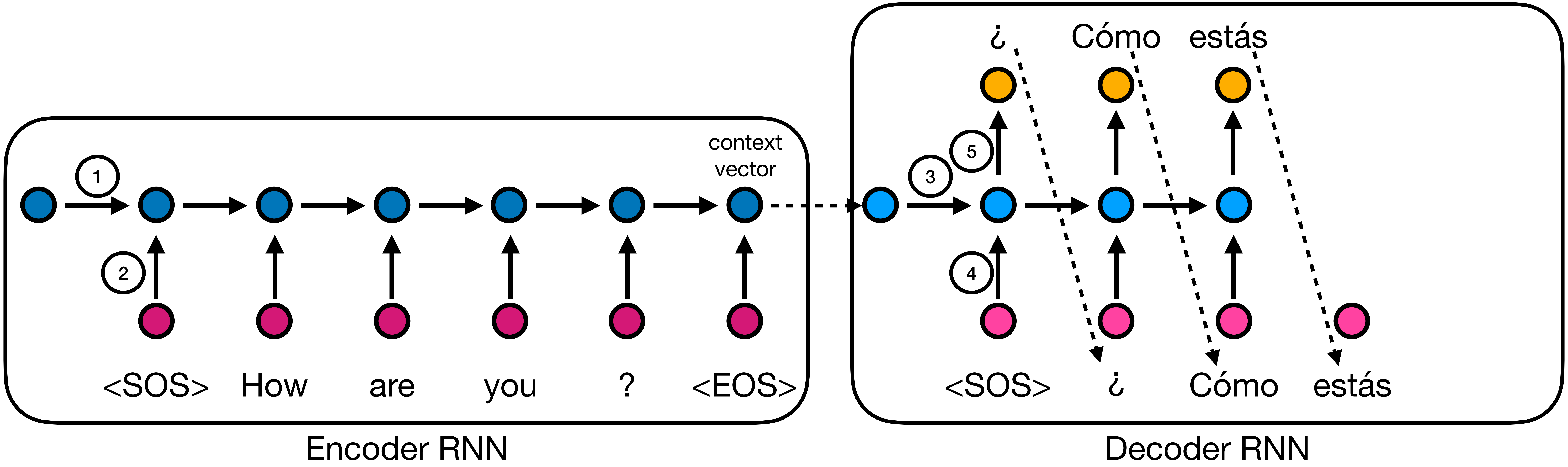
Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

2. Find the last hidden state of the encoder RNN, called the *context vector*.

3. Copy the context vector into the first hidden state of a *decoder RNN.*

4. Use the decoder RNN to generate the translation.



RNN for machine translation

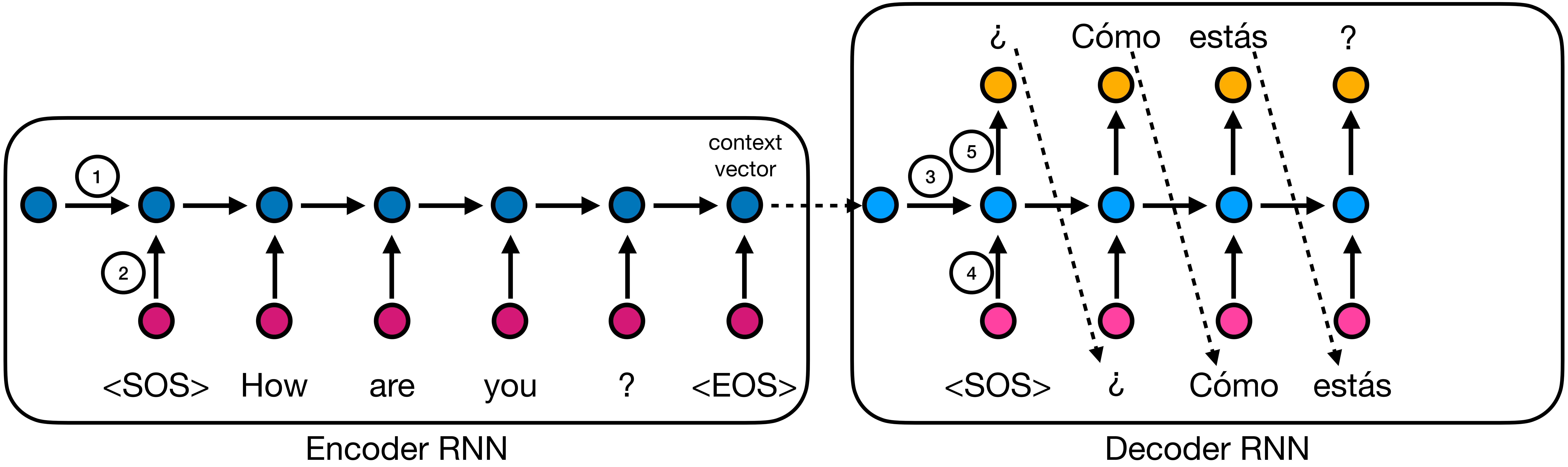
Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

2. Find the last hidden state of the encoder RNN, called the *context vector*.

3. Copy the context vector into the first hidden state of a *decoder RNN.*

4. Use the decoder RNN to generate the translation.



RNN for machine translation

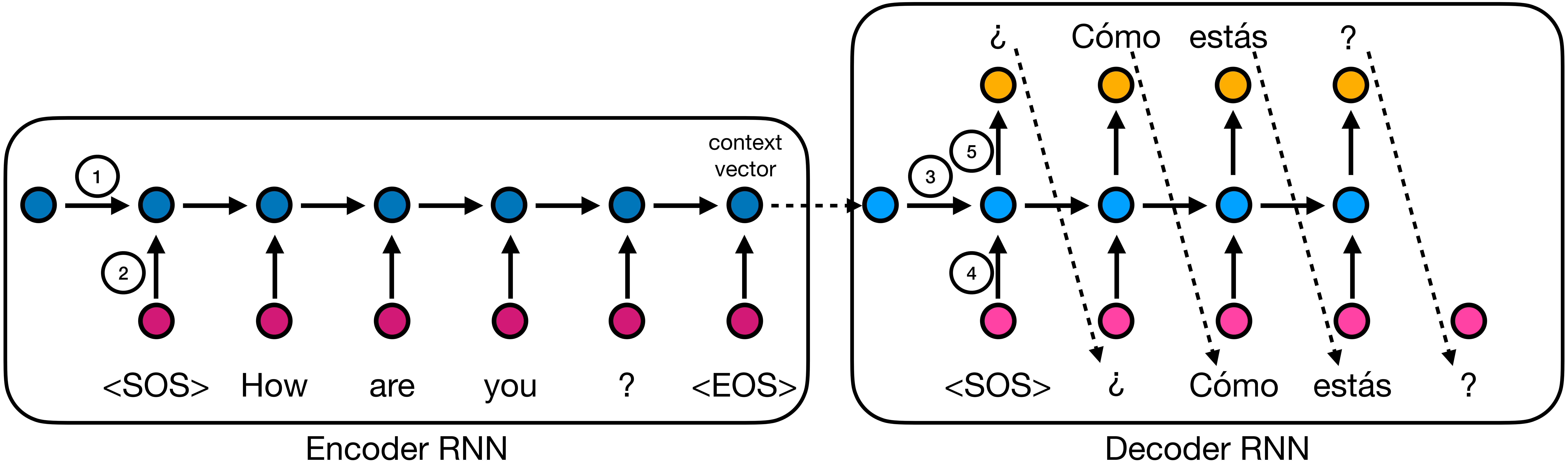
Input text: "How are you?" Output translation: "¿Cómo estás?"

1. Pass input sentence through *encoder RNN.*

2. Find the last hidden state of the encoder RNN, called the *context vector*.

3. Copy the context vector into the first hidden state of a *decoder RNN.*

4. Use the decoder RNN to generate the translation.



RNN for machine translation

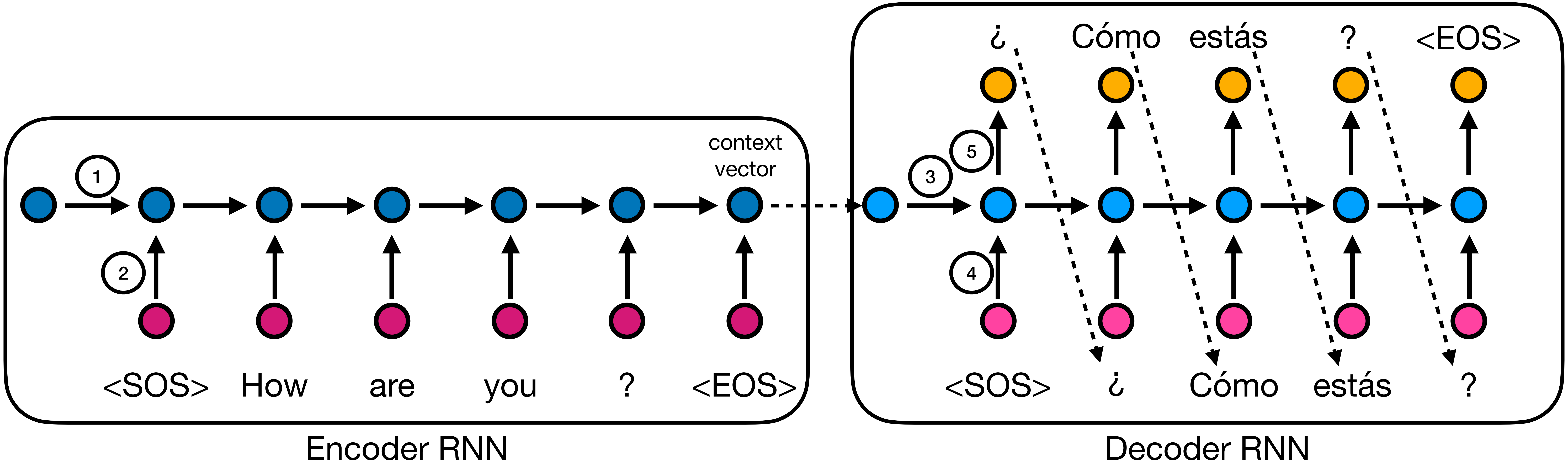
Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through *encoder RNN.*

2. Find the last hidden state of the encoder RNN, called the *context vector*.

3. Copy the context vector into the first hidden state of a *decoder RNN.*

4. Use the decoder RNN to generate the translation.



Training RNNs for machine translation

Training data

Parallel text corpora across two languages, e.g. U.N. proceedings.

Input	Output
“How are you?”	“¿Cómo estás?”
“The cat sleeps.”	“El gato duerme.”
“I am reading.”	“Estoy leyendo.”
“She is happy.”	“Ella está feliz.”
…	…

Training RNNs for machine translation

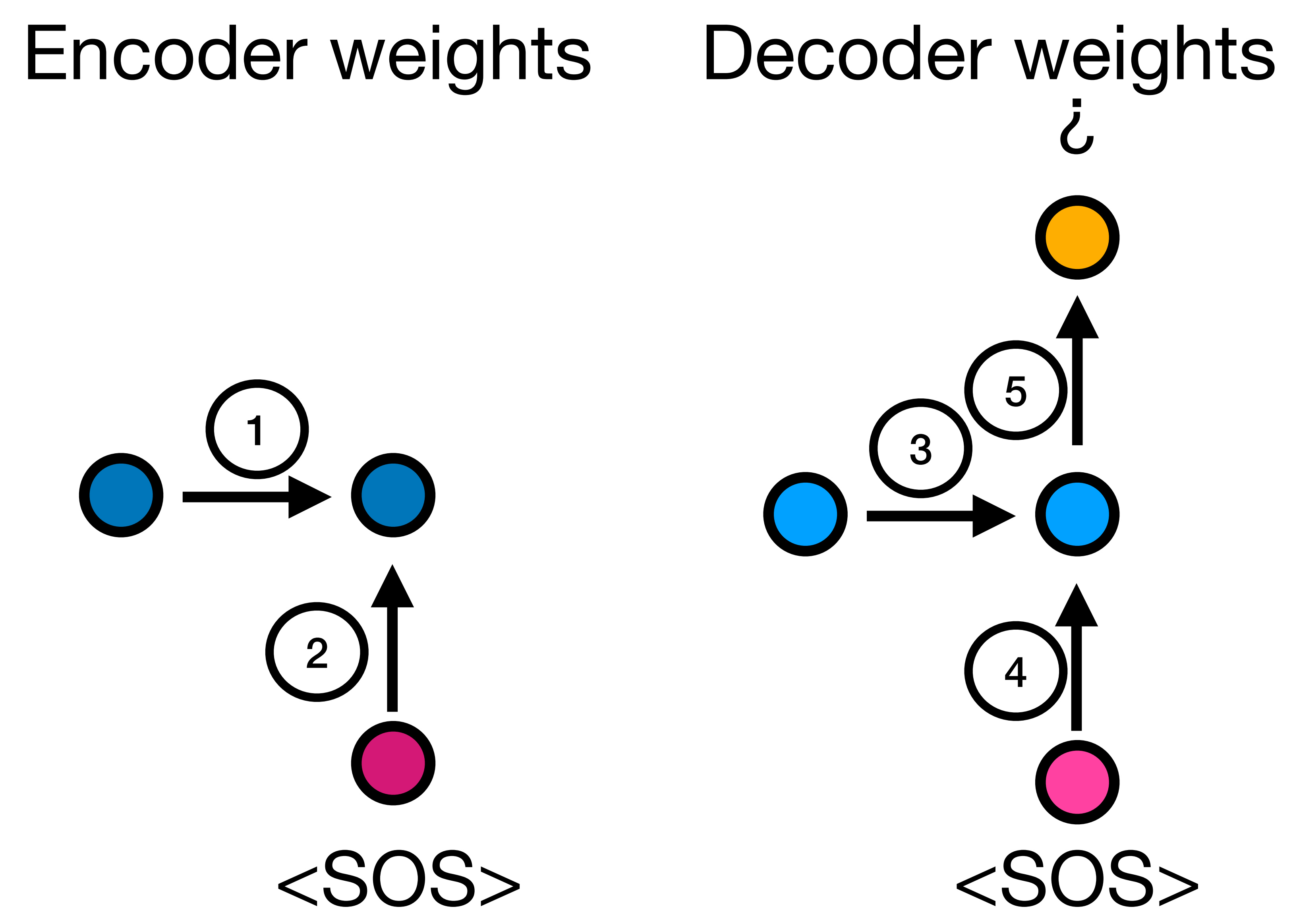
Training data

Parallel text corpora across two languages, e.g. U.N. proceedings.

Input	Output
“How are you?”	“¿Cómo estás?”
“The cat sleeps.”	“El gato duerme.”
“I am reading.”	“Estoy leyendo.”
“She is happy.”	“Ella está feliz.”
…	…

Training process

- Learn the two sets of weights in encoder RNN and three sets of weights in decoder RNN via stochastic gradient descent on the cross-entropy loss function.



Weaknesses of RNNs

Weaknesses of RNNs

- **Hard to handle long sequences.**
  Hard to encode the information from entire input sequence in a single context vector. RNNs tend to “forget.”

Weaknesses of RNNs

- **Hard to handle long sequences.** Hard to encode the information from entire input sequence in a single context vector. RNNs tend to "forget."

Example: Text generated from RNN trained on Obama's speeches:

Weaknesses of RNNs

- **Hard to handle long sequences.** Hard to encode the information from entire input sequence in a single context vector. RNNs tend to “forget.”

Example: Text generated from RNN trained on Obama’s speeches:

“Good afternoon. God bless you.

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done…”

Weaknesses of RNNs

- **Hard to handle long sequences.** Hard to encode the information from entire input sequence in a single context vector. RNNs tend to "forget."

- **Slow to train.** The sequential nature of RNNs prevents parallelization.

Example: Text generated from RNN trained on Obama's speeches:

"Good afternoon. God bless you.

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done…"

Weaknesses of RNNs

- **Hard to handle long sequences.** Hard to encode the information from entire input sequence in a single context vector. RNNs tend to "forget."

- **Slow to train.** The sequential nature of RNNs prevents parallelization.

Improvements to handle long sequences:

Example: Text generated from RNN trained on Obama's speeches:

"Good afternoon. God bless you.

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done…"

Weaknesses of RNNs

- **Hard to handle long sequences.** Hard to encode the information from entire input sequence in a single context vector. RNNs tend to "forget."
- **Slow to train.** The sequential nature of RNNs prevents parallelization.

Improvements to handle long sequences:

- Equipping RNNs with "memory" (LSTMs)

Example: Text generated from RNN trained on Obama's speeches:

"Good afternoon. God bless you.

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done…"

Weaknesses of RNNs

- **Hard to handle long sequences.** Hard to encode the information from entire input sequence in a single context vector. RNNs tend to "forget."

- **Slow to train.** The sequential nature of RNNs prevents parallelization.

Improvements to handle long sequences:

- Equipping RNNs with "memory" (LSTMs)

- Equipping RNNs with "attention"

Example: Text generated from RNN trained on Obama's speeches:

"Good afternoon. God bless you.

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done…"

Weaknesses of RNNs

- **Hard to handle long sequences.** Hard to encode the information from entire input sequence in a single context vector. RNNs tend to “forget.”

- **Slow to train.** The sequential nature of RNNs prevents parallelization.

Improvements to handle long sequences:

- Equipping RNNs with “memory” (LSTMs)

- Equipping RNNs with “attention” ←

Example: Text generated from RNN trained on Obama’s speeches:

“Good afternoon. God bless you.

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done…”

RNN for sentiment analysis with attention

Input text: “I enjoyed the movie!” Output class: “Positive”

RNN for sentiment analysis with attention

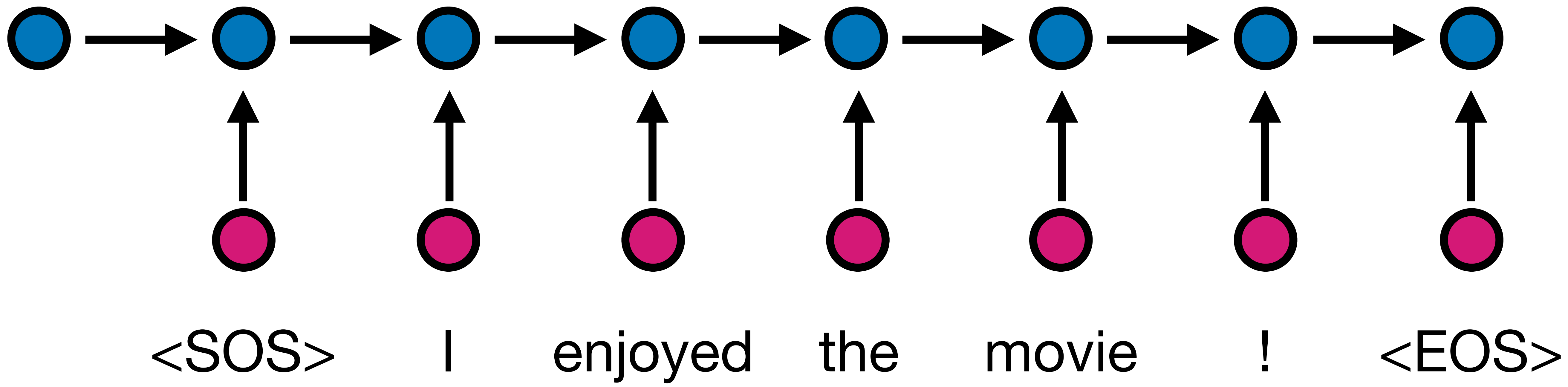
Input text: “I enjoyed the movie!” Output class: “Positive”

1. Encode input sentence via RNN.

RNN for sentiment analysis with attention

Input text: “I enjoyed the movie!” Output class: “Positive”

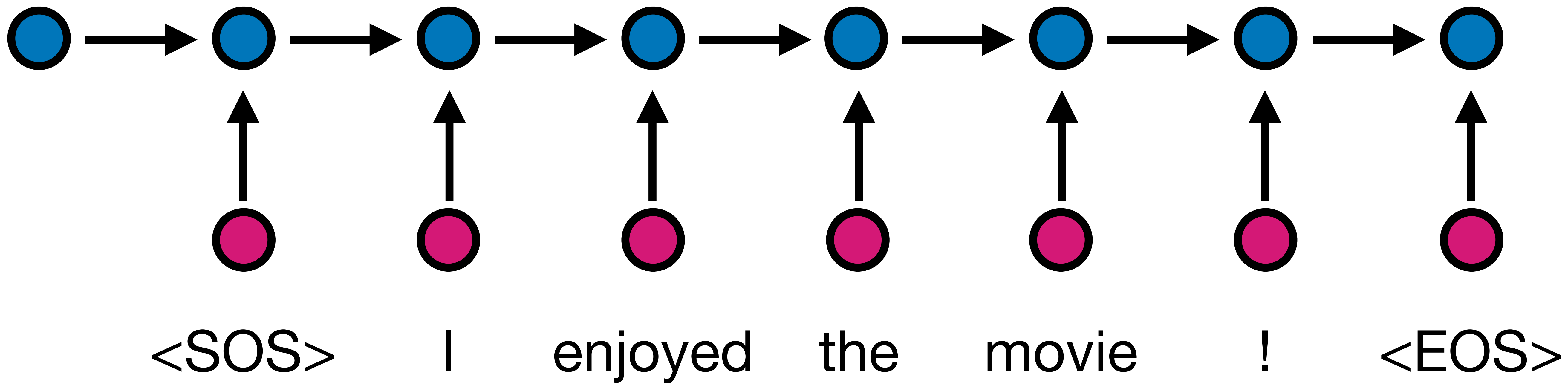
1. Encode input sentence via RNN.



RNN for sentiment analysis with attention

Input text: “I enjoyed the movie!” Output class: “Positive”

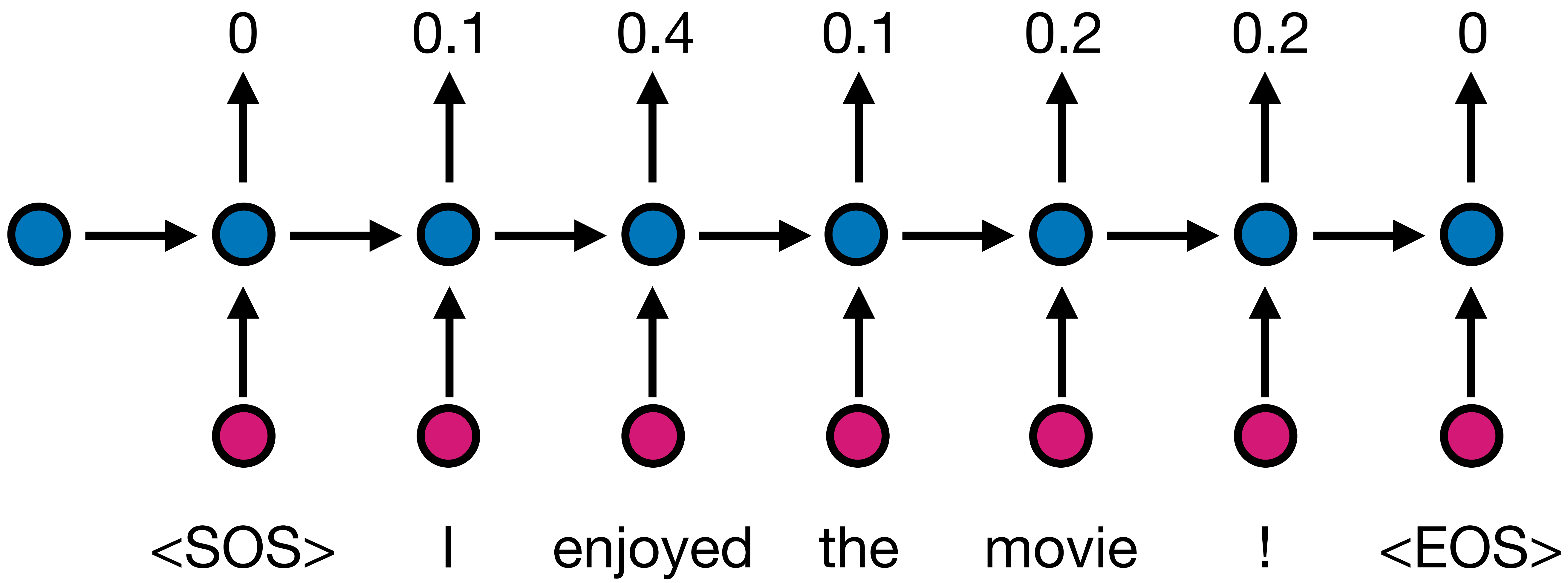
1. Encode input sentence via RNN.
2. Pass each hidden state through FC layer with softmax to get *attention weights.*



RNN for sentiment analysis with attention

Input text: “I enjoyed the movie!” Output class: “Positive”

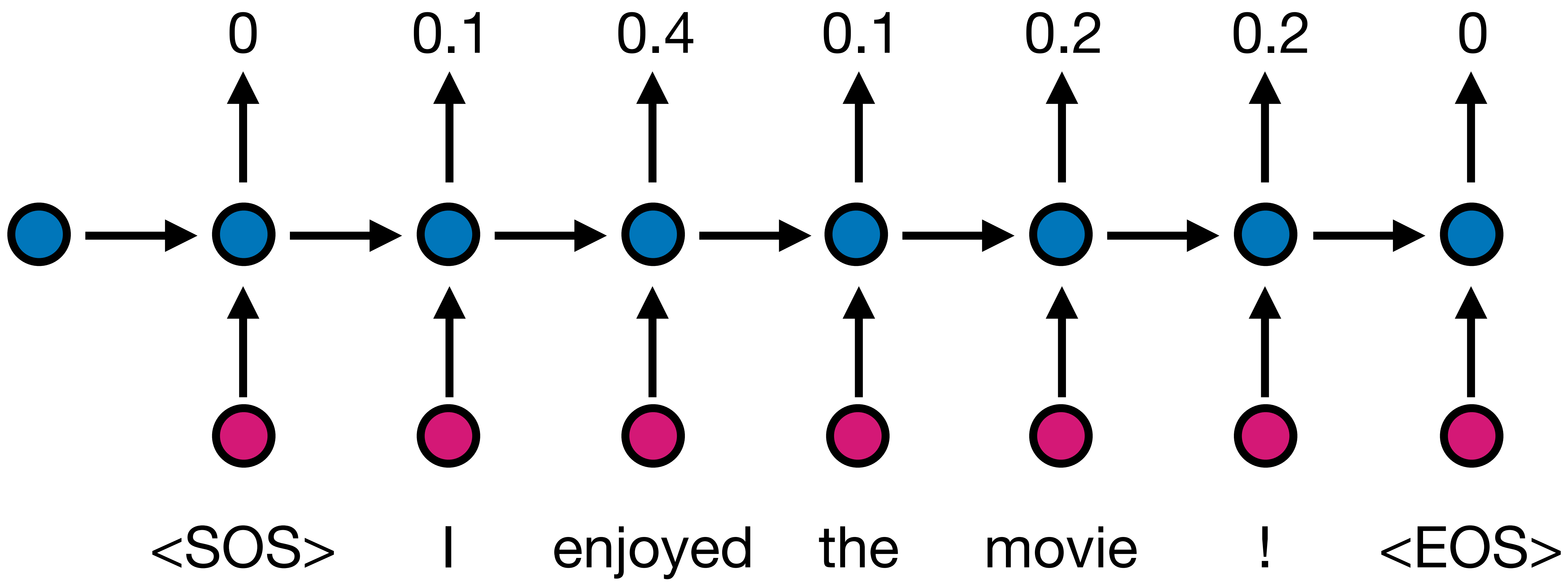
1. Encode input sentence via RNN.
2. Pass each hidden state through FC layer with softmax to get *attention weights.*



RNN for sentiment analysis with attention

Input text: “I enjoyed the movie!” Output class: “Positive”

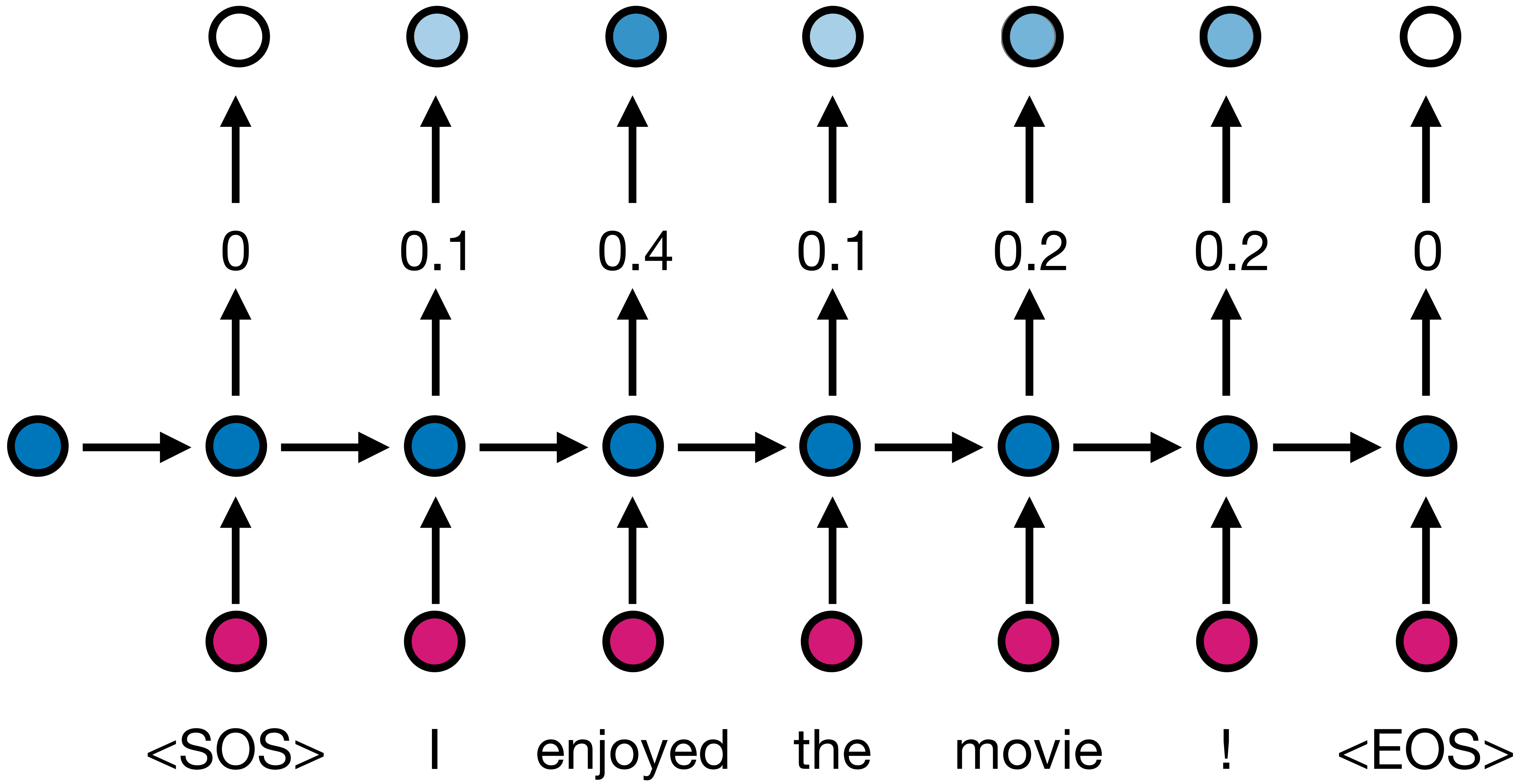
1. Encode input sentence via RNN.
2. Pass each hidden state through FC layer with softmax to get *attention weights.*
3. Multiply hidden states by attention weights.



RNN for sentiment analysis with attention

Input text: “I enjoyed the movie!” Output class: “Positive”

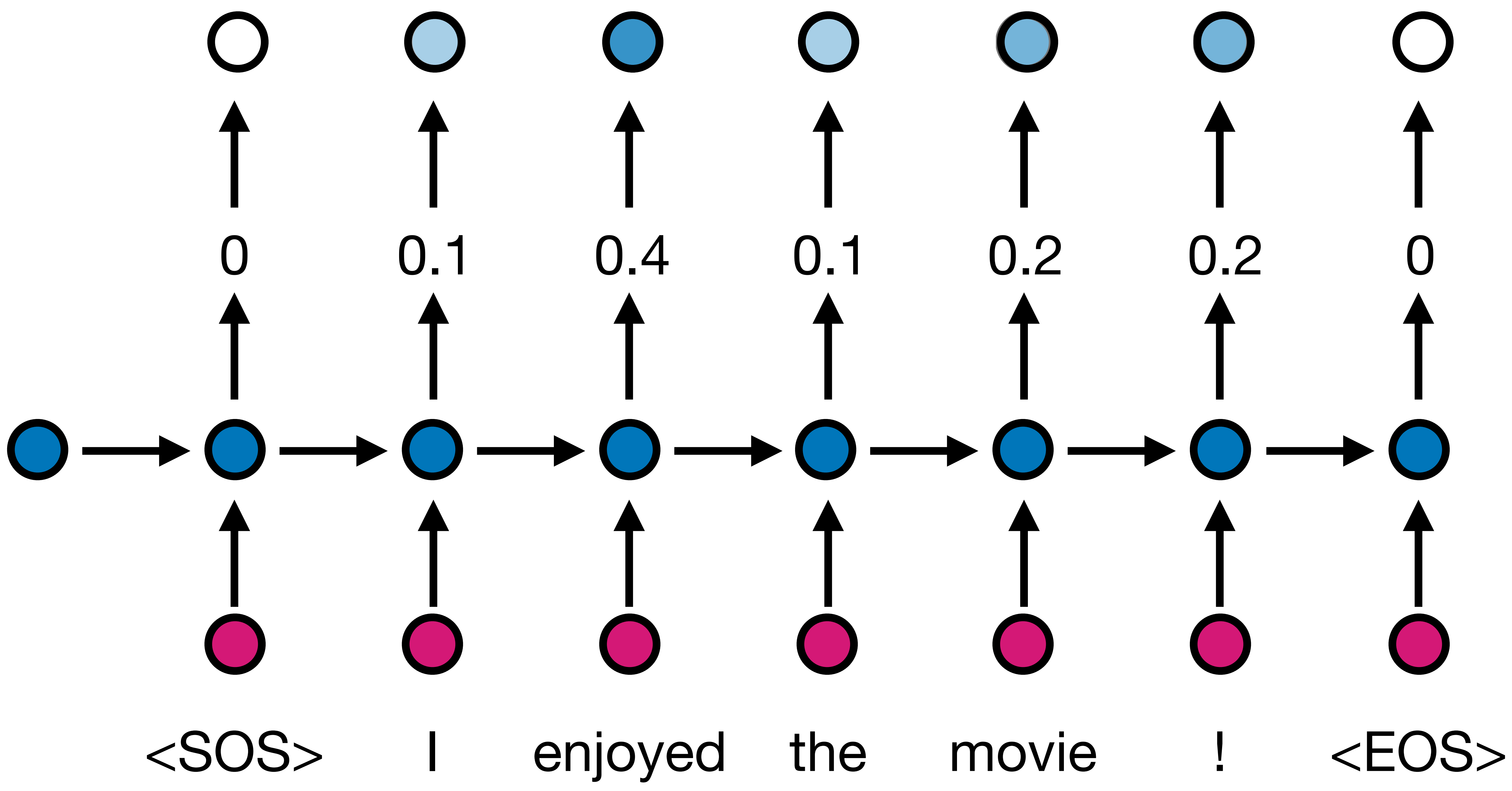
1. Encode input sentence via RNN.
2. Pass each hidden state through FC layer with softmax to get *attention weights.*
3. Multiply hidden states by attention weights.



RNN for sentiment analysis with attention

Input text: “I enjoyed the movie!” Output class: “Positive”

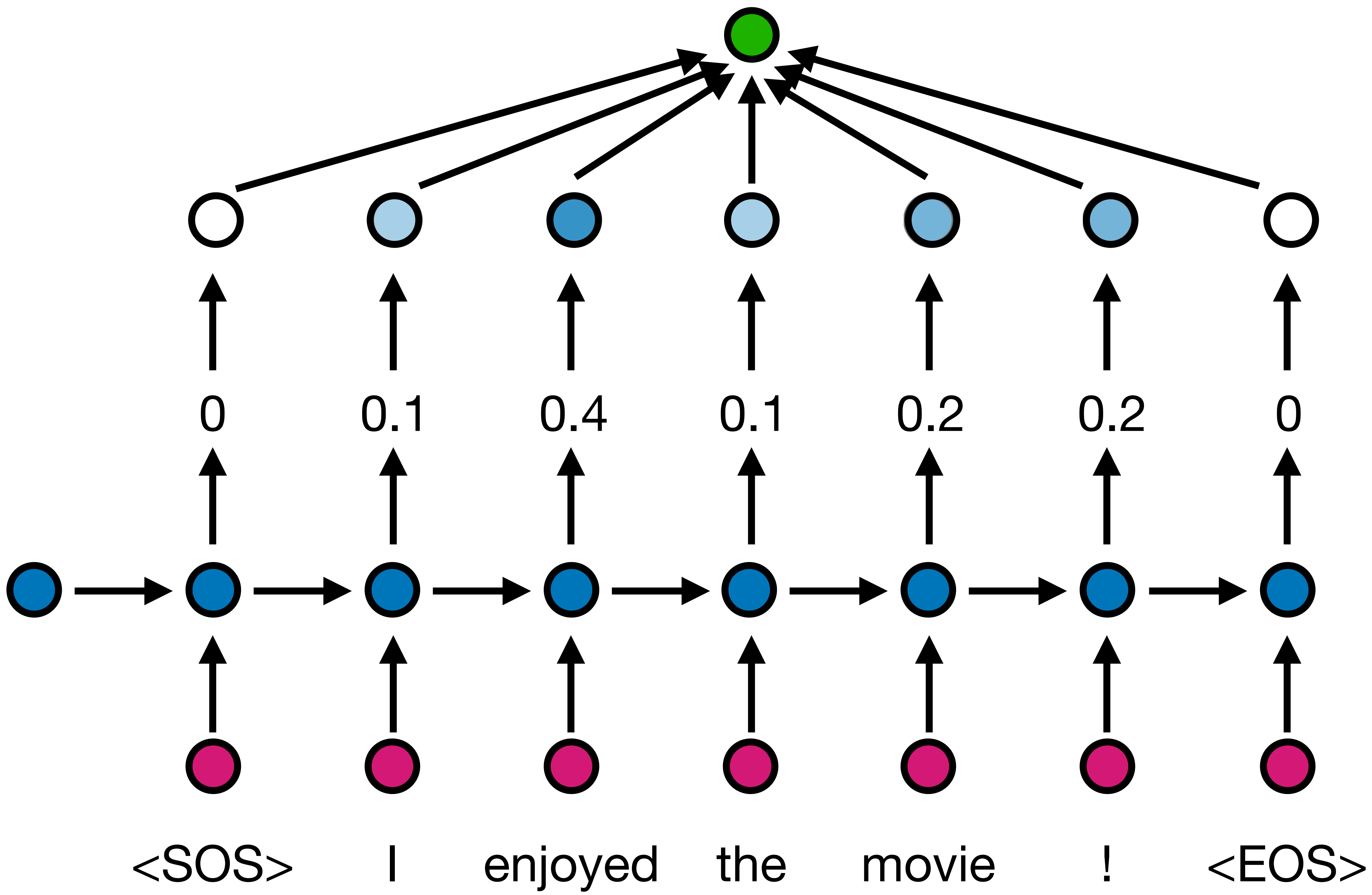
1. Encode input sentence via RNN.
2. Pass each hidden state through FC layer with softmax to get *attention weights.*
3. Multiply hidden states by attention weights.
4. Add weighted hidden states to get *context vector*.



RNN for sentiment analysis with attention

Input text: “I enjoyed the movie!” Output class: “Positive”

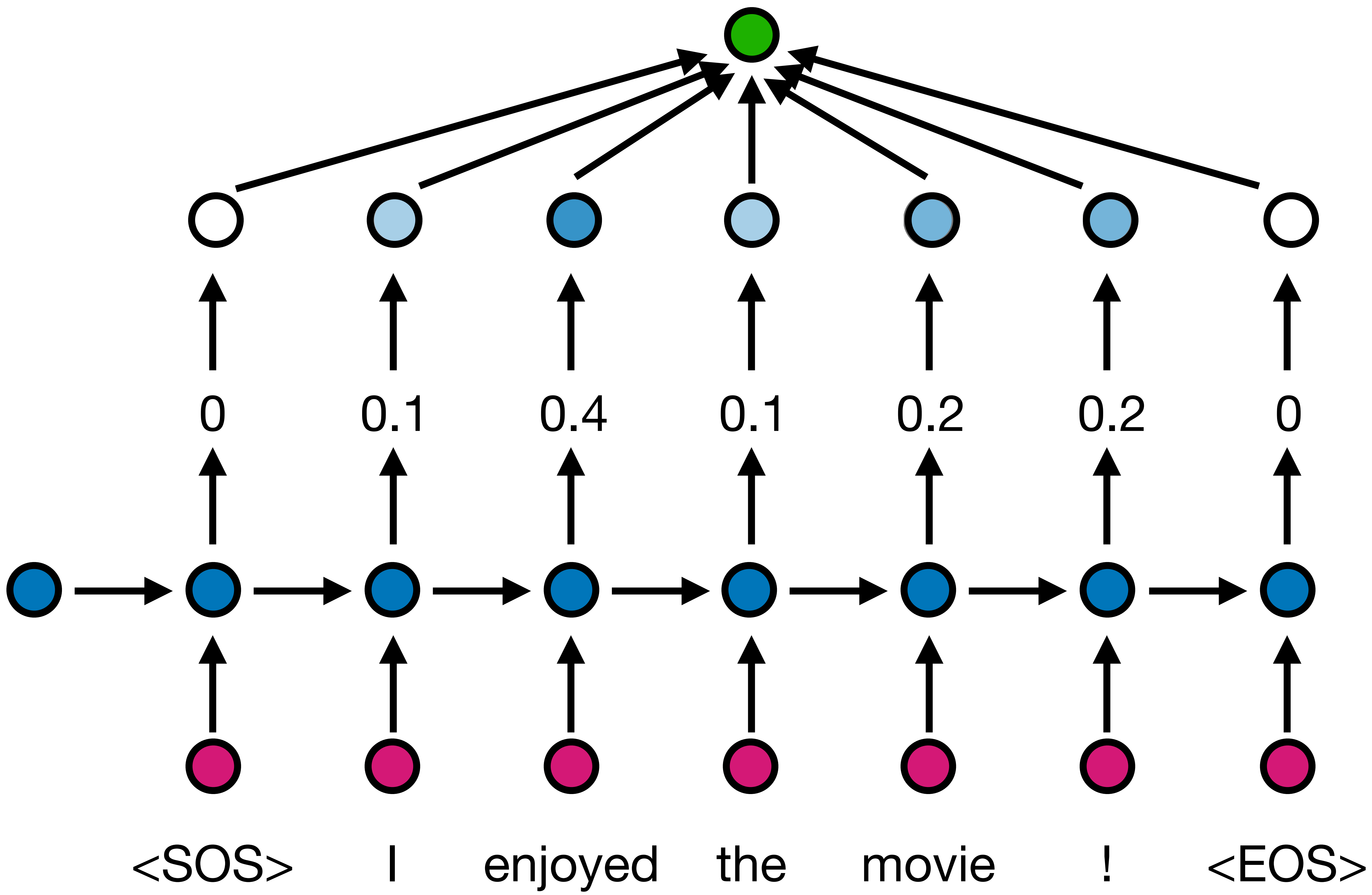
1. Encode input sentence via RNN.
2. Pass each hidden state through FC layer with softmax to get *attention weights.*
3. Multiply hidden states by attention weights.
4. Add weighted hidden states to get *context vector*.



RNN for sentiment analysis with attention

Input text: “I enjoyed the movie!” Output class: “Positive”

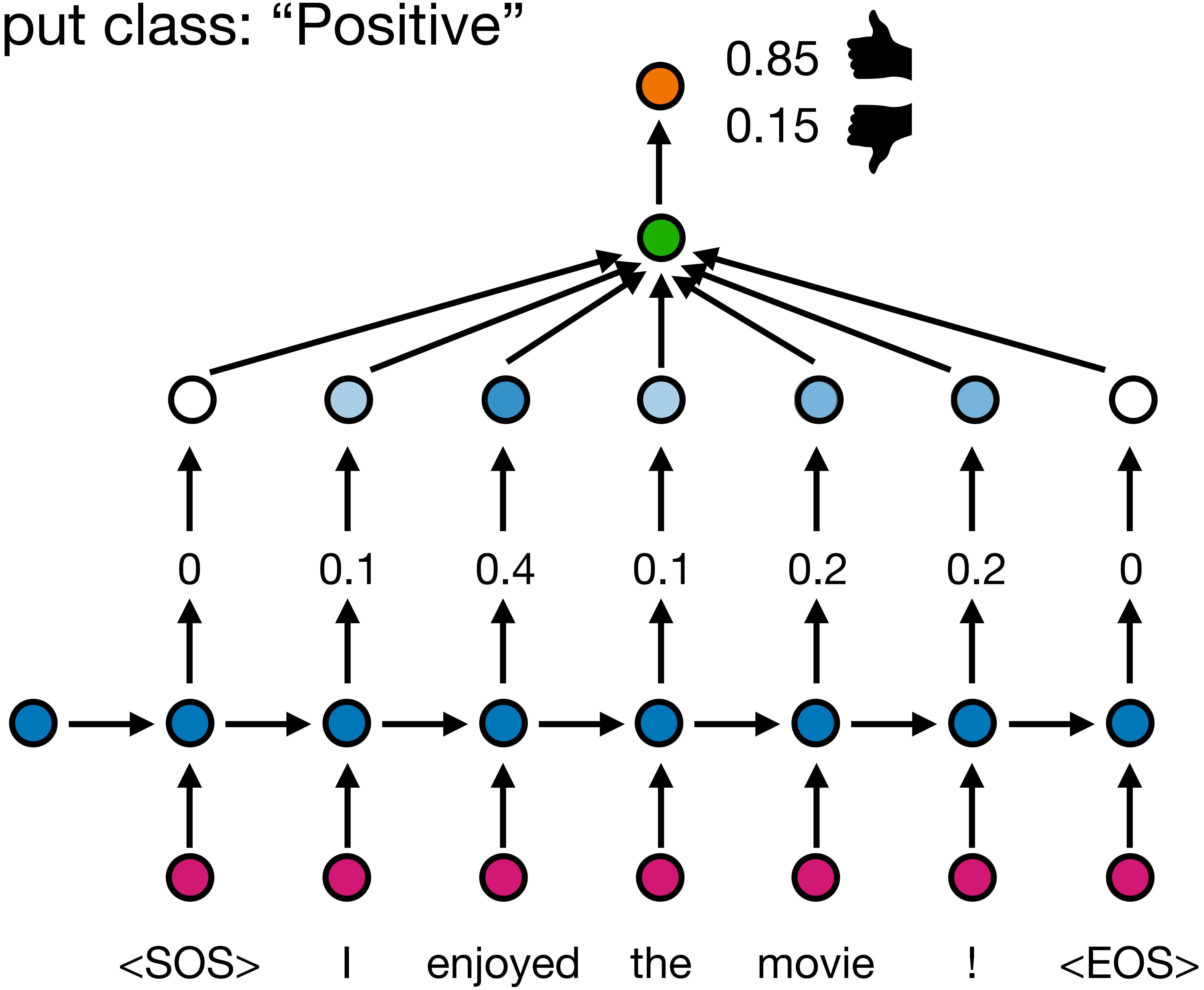
1. Encode input sentence via RNN.
2. Pass each hidden state through FC layer with softmax to get *attention weights.*
3. Multiply hidden states by attention weights.
4. Add weighted hidden states to get *context vector*.
5. Pass context vector through FC layer with softmax to get output class probabilities.



RNN for sentiment analysis with attention

Input text: “I enjoyed the movie!” Output class: “Positive”

1. Encode input sentence via RNN.
2. Pass each hidden state through FC layer with softmax to get *attention weights.*
3. Multiply hidden states by attention weights.
4. Add weighted hidden states to get *context vector*.
5. Pass context vector through FC layer with softmax to get output class probabilities.



RNN for machine translation with attention

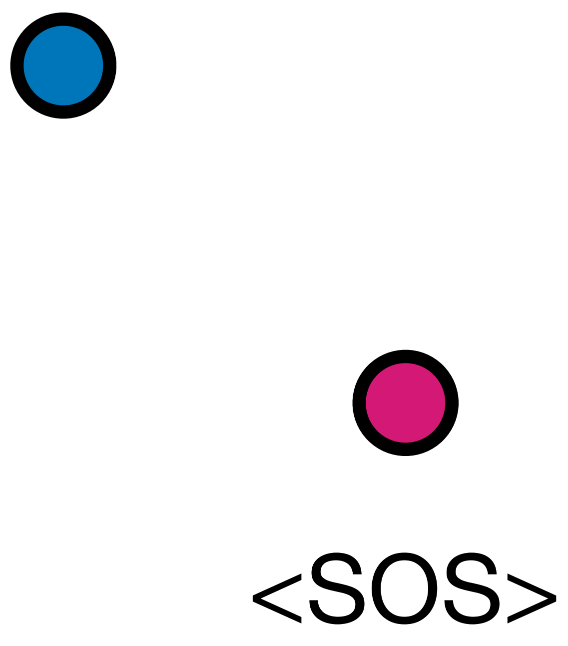
Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Pass input sentence through encoder RNN.

RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

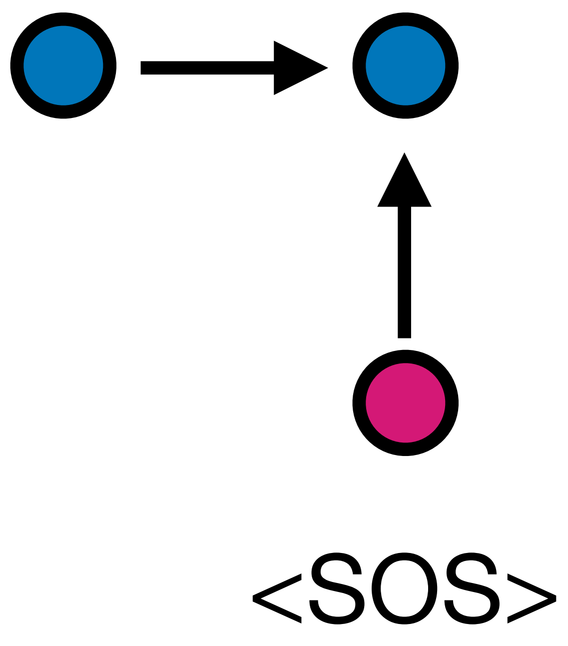
1. Pass input sentence through encoder RNN.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

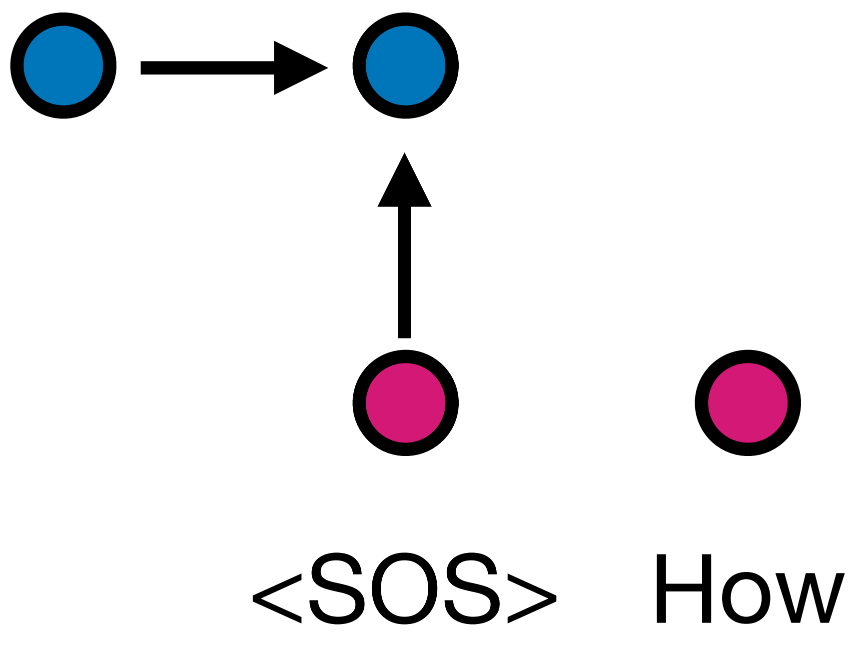
1. Pass input sentence through encoder RNN.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

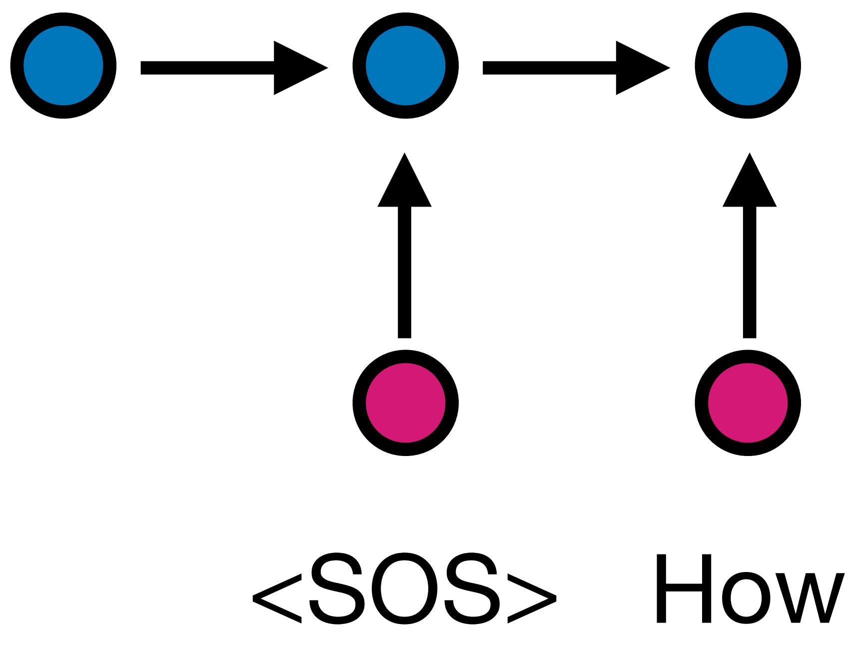
1. Pass input sentence through encoder RNN.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

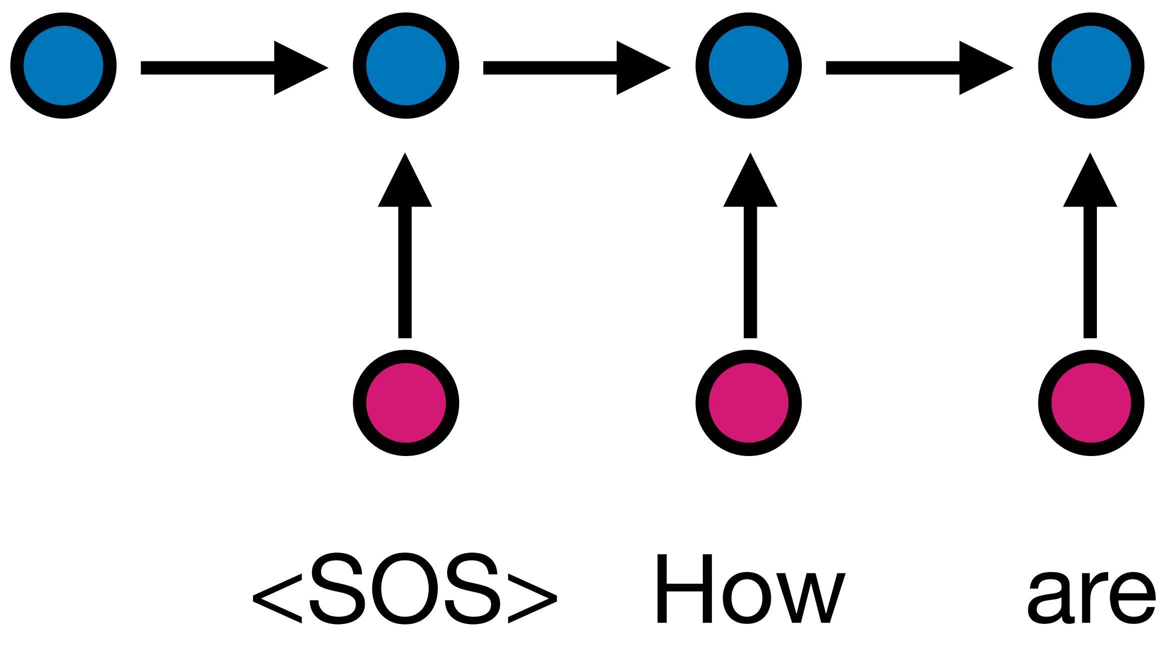
1. Pass input sentence through encoder RNN.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

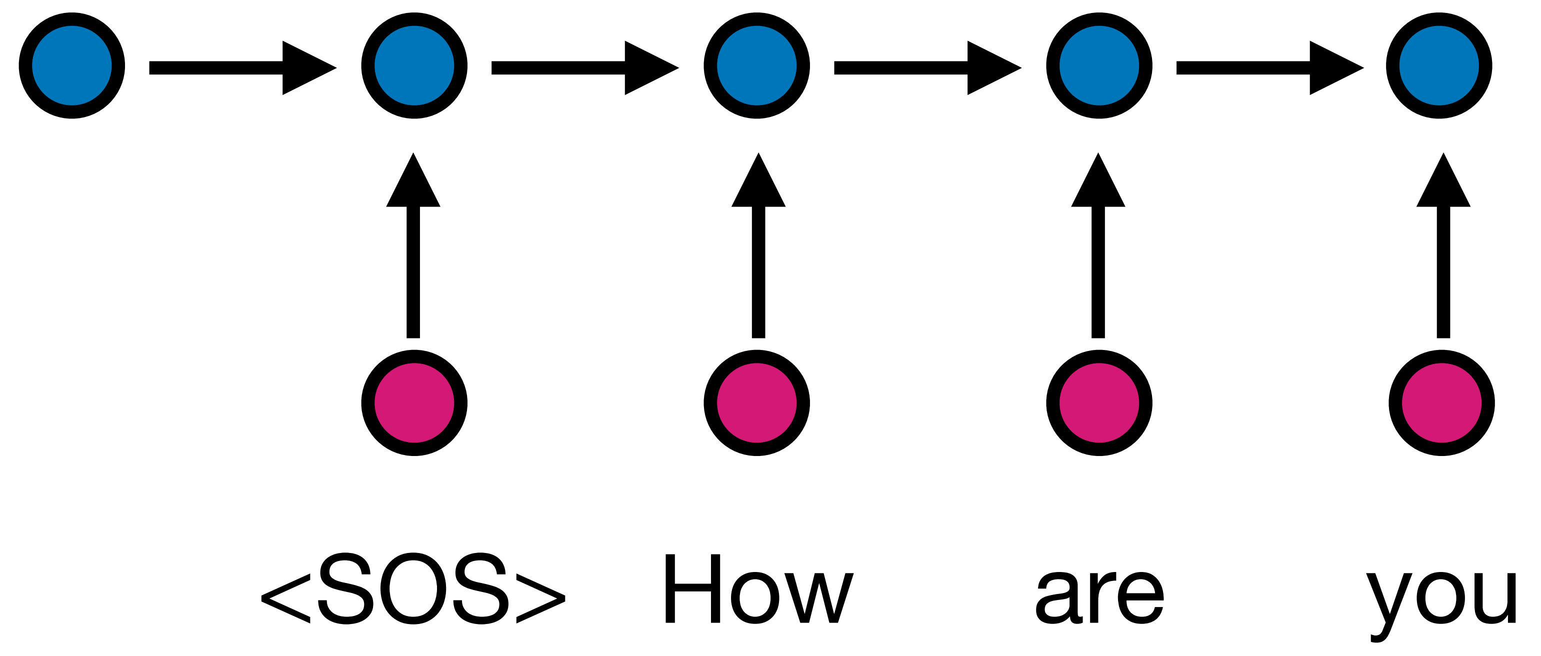
1. Pass input sentence through encoder RNN.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

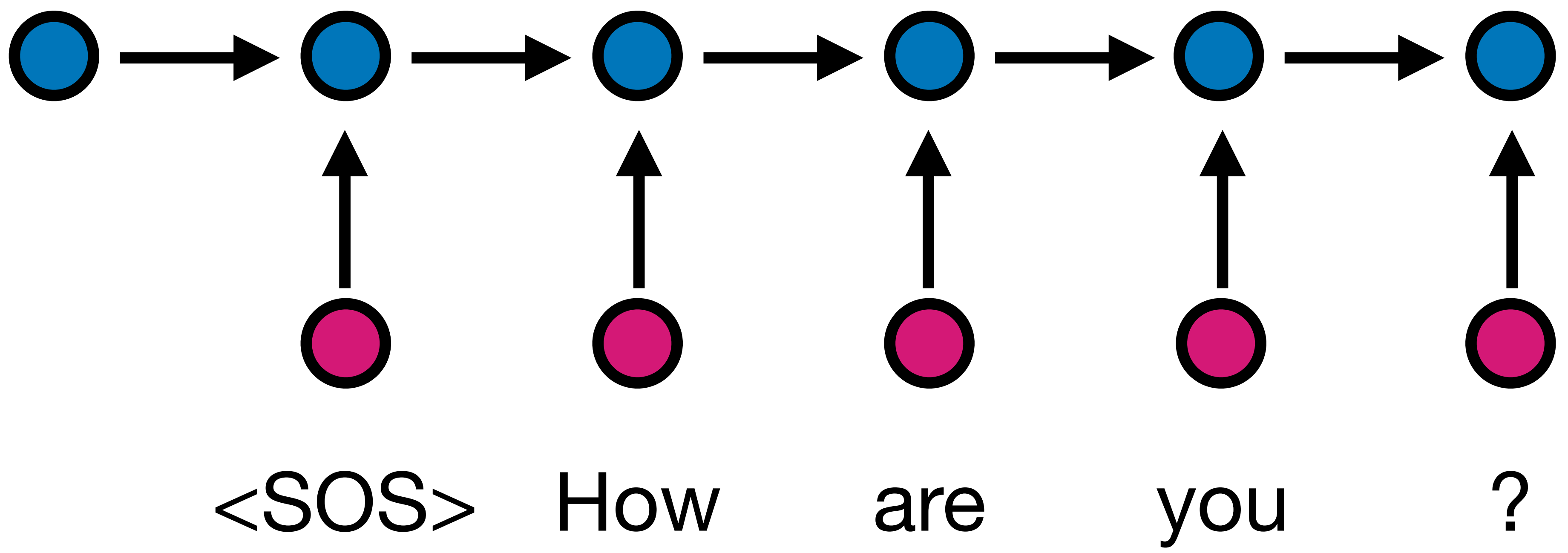
1. Pass input sentence through encoder RNN.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

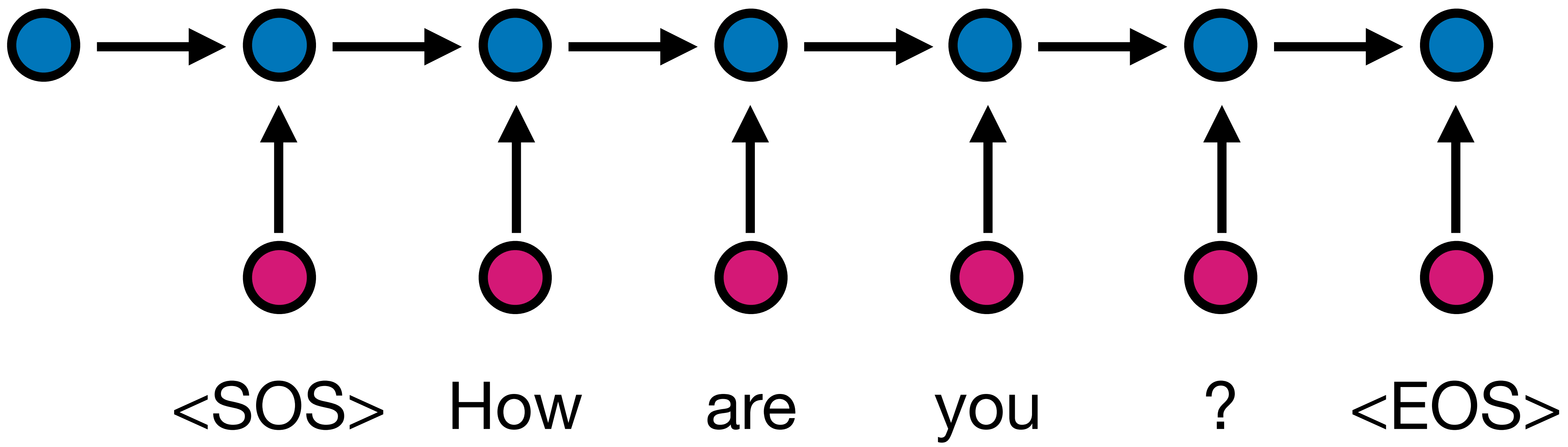
1. Pass input sentence through encoder RNN.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

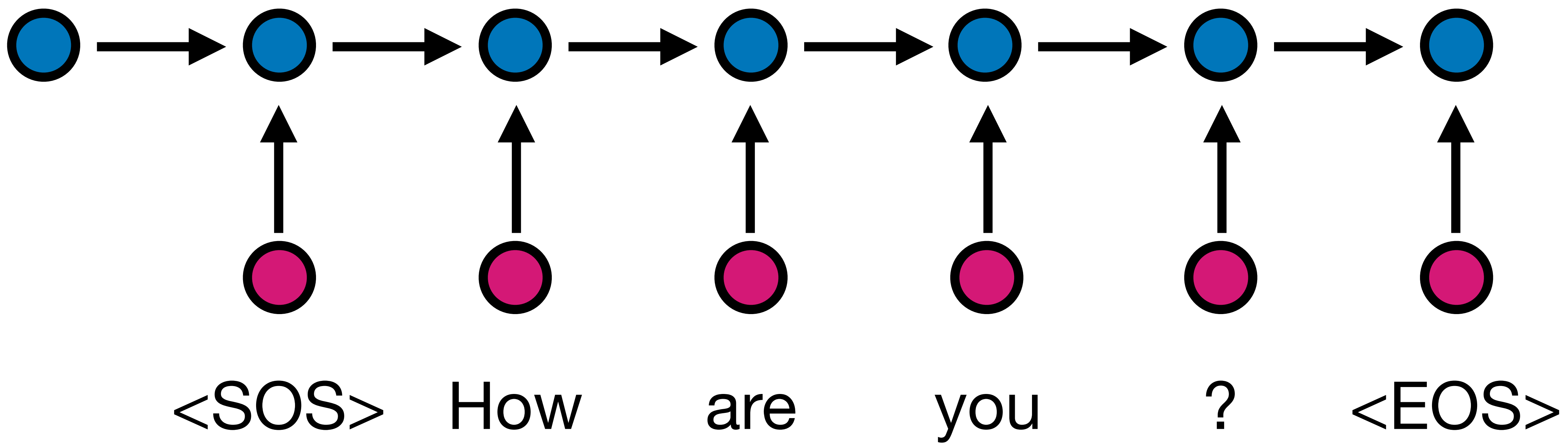
1. Pass input sentence through encoder RNN.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

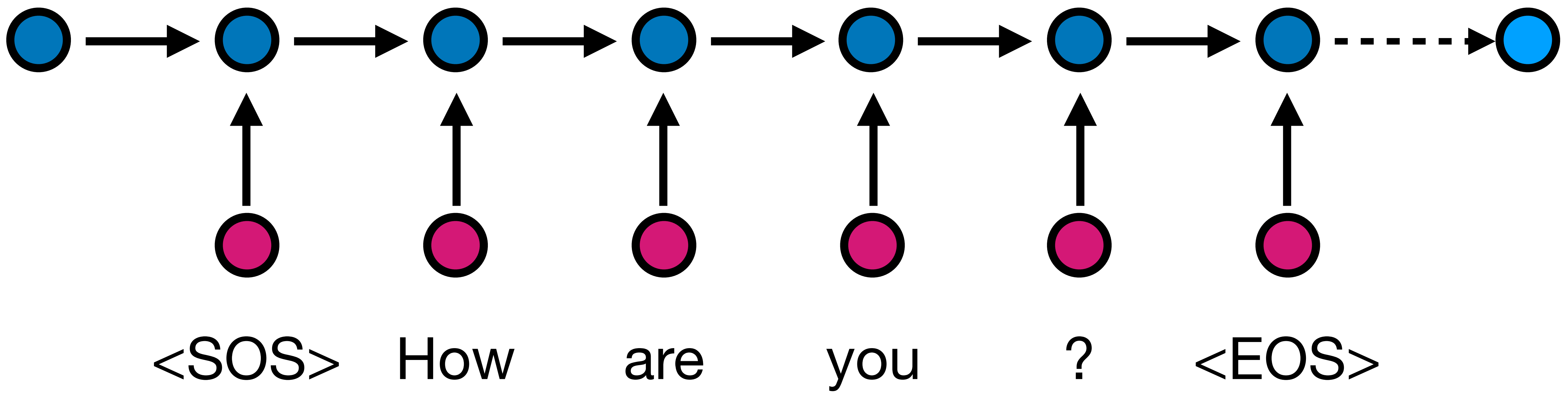
2. Copy the last hidden state of encoder RNN to first hidden state of a decoder RNN.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

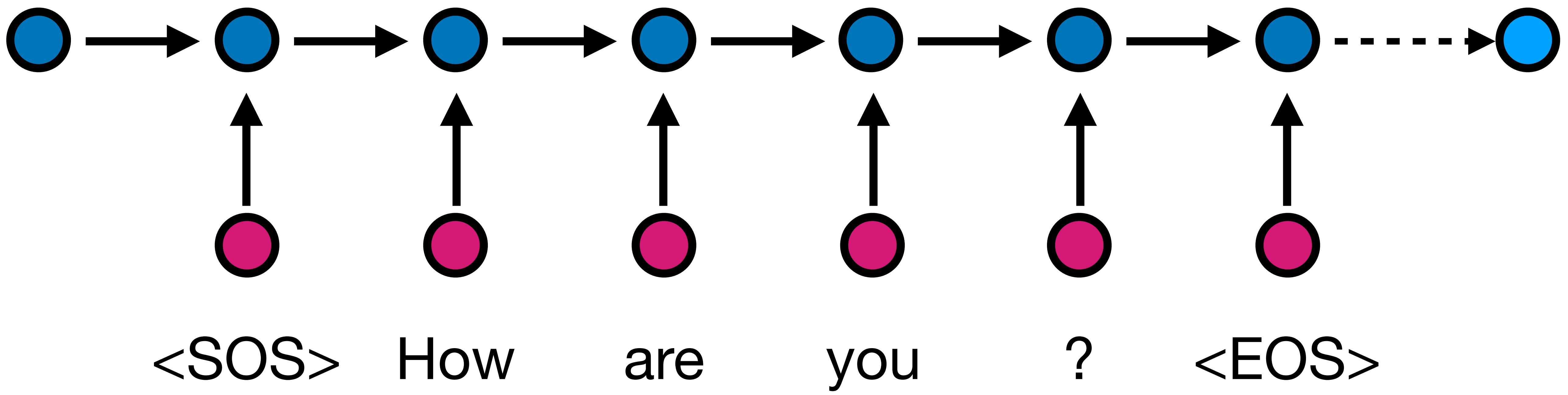
2. Copy the last hidden state of encoder RNN to first hidden state of a decoder RNN.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

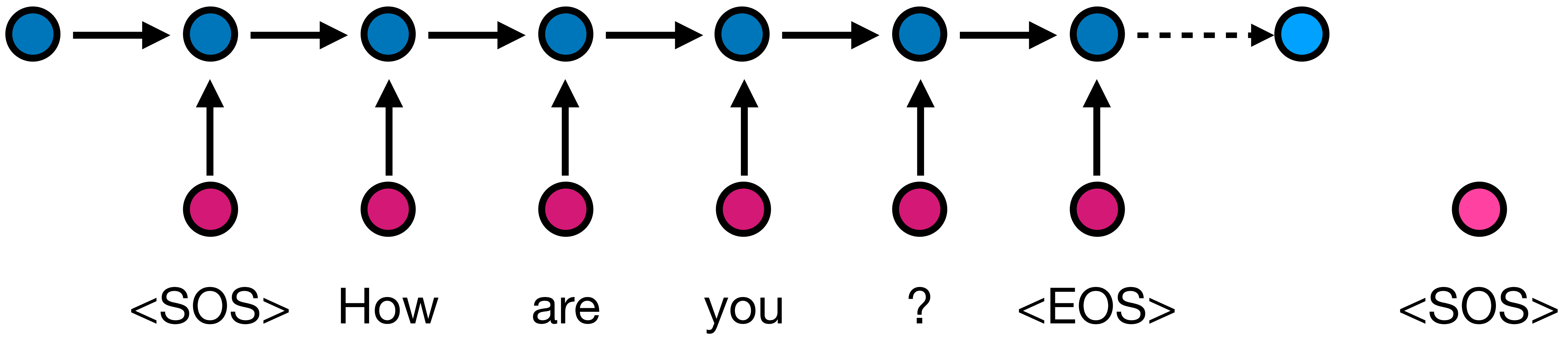
3. Input <SOS> token to generate next decoder hidden state.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

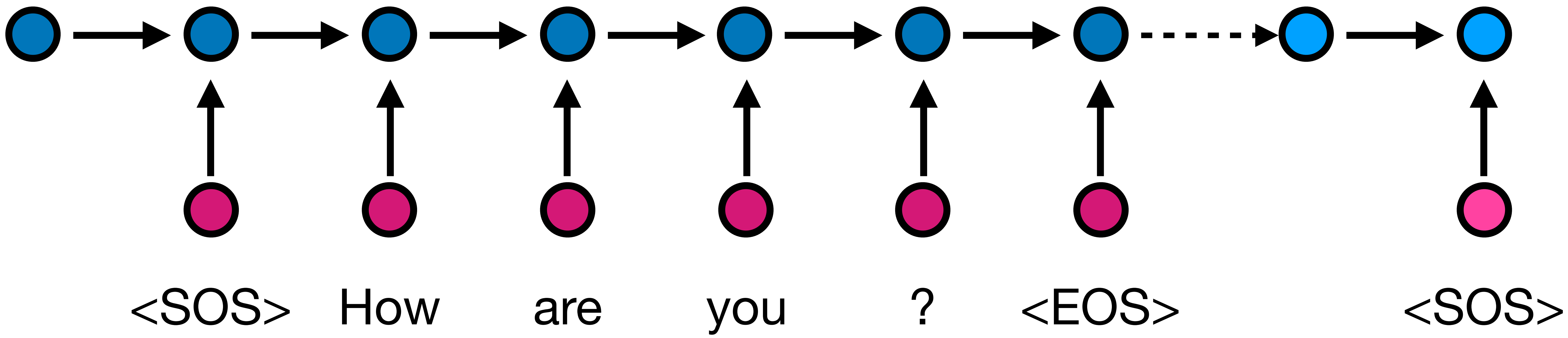
3. Input <SOS> token to generate next decoder hidden state.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

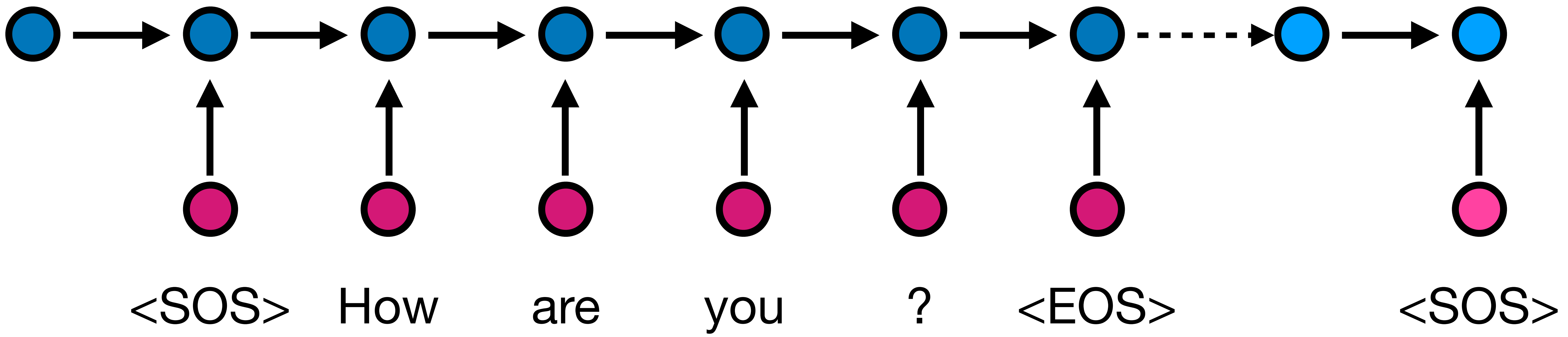
3. Input <SOS> token to generate next decoder hidden state.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

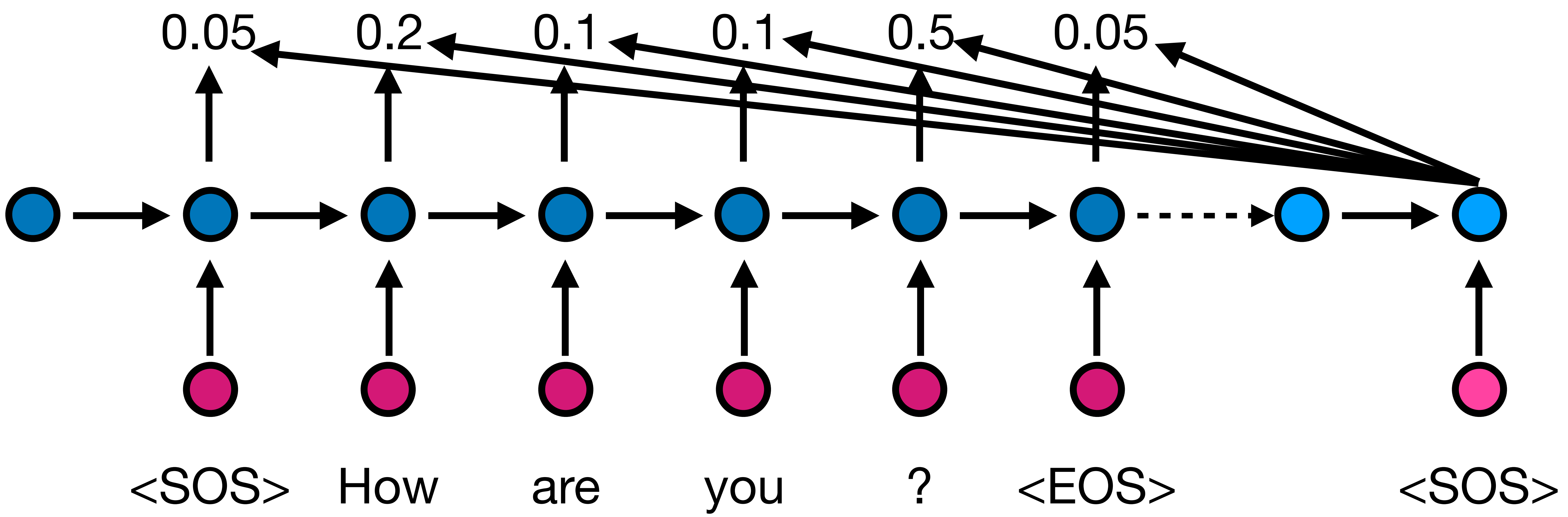
4. Calculate attention vector over encoder hidden states based on decoder hidden state.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

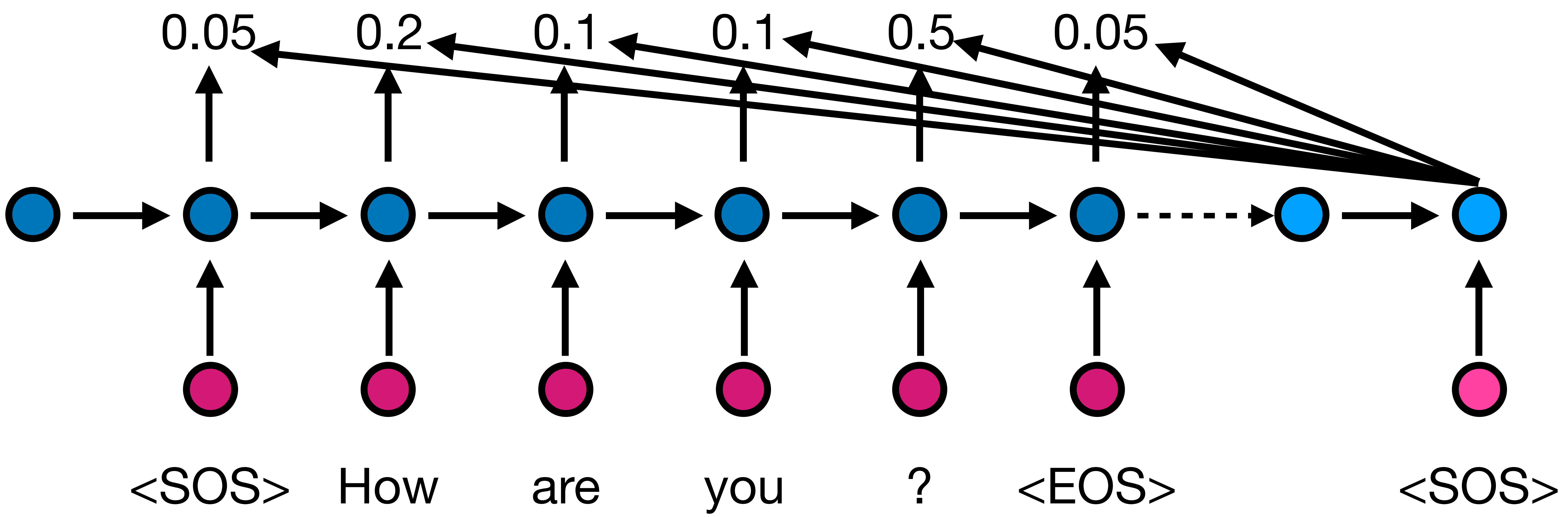
4. Calculate attention vector over encoder hidden states based on decoder hidden state.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

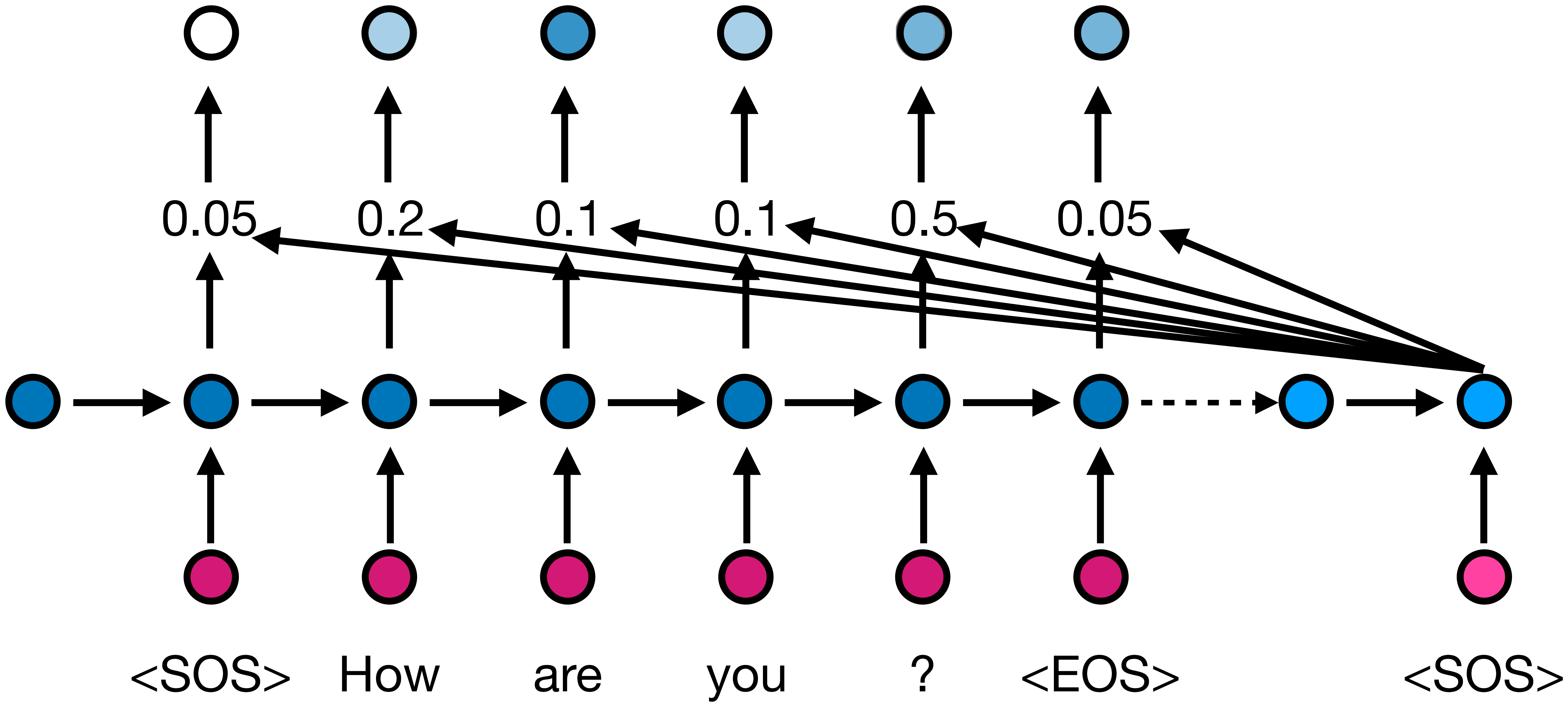
5. Compute first context vector as weighted average of encoder hidden states.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

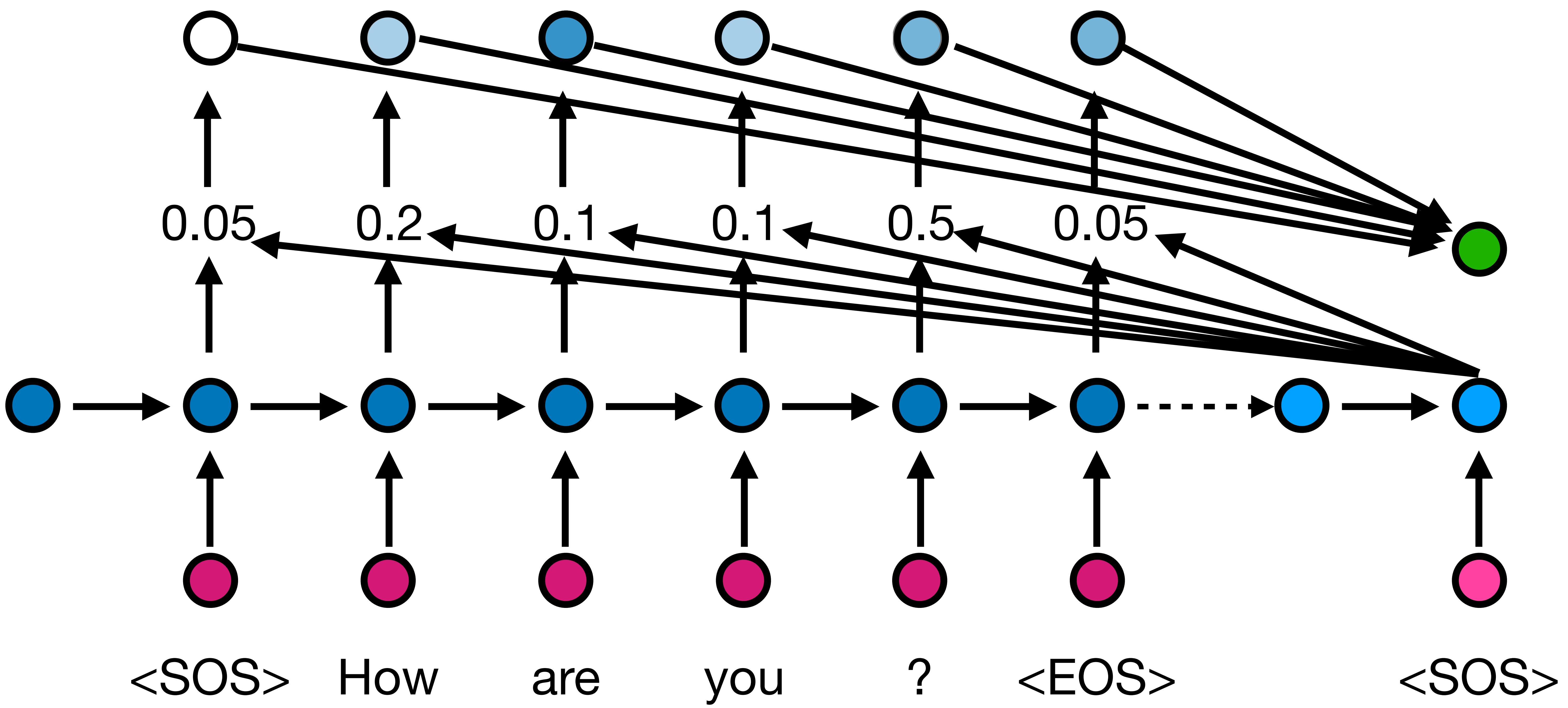
5. Compute first context vector as weighted average of encoder hidden states.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

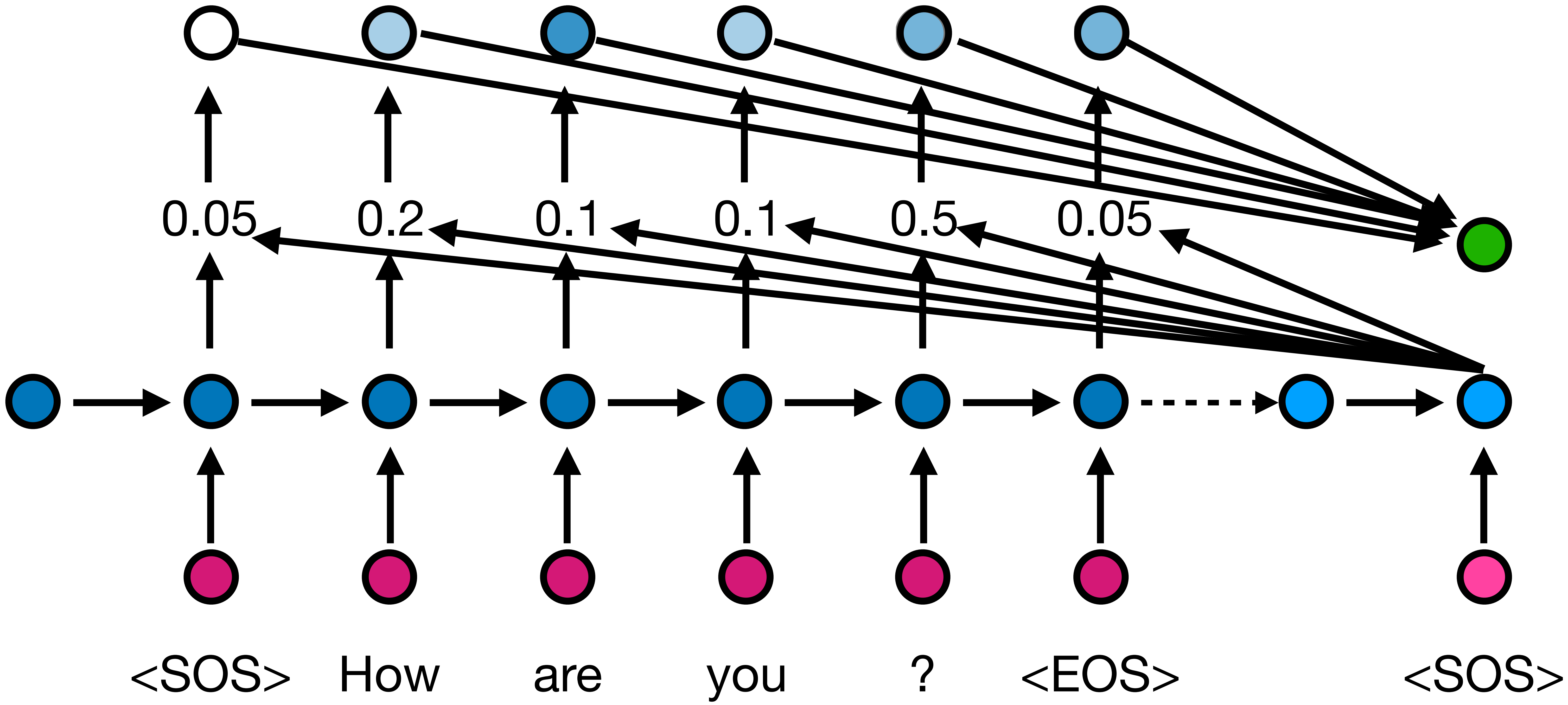
5. Compute first context vector as weighted average of encoder hidden states.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

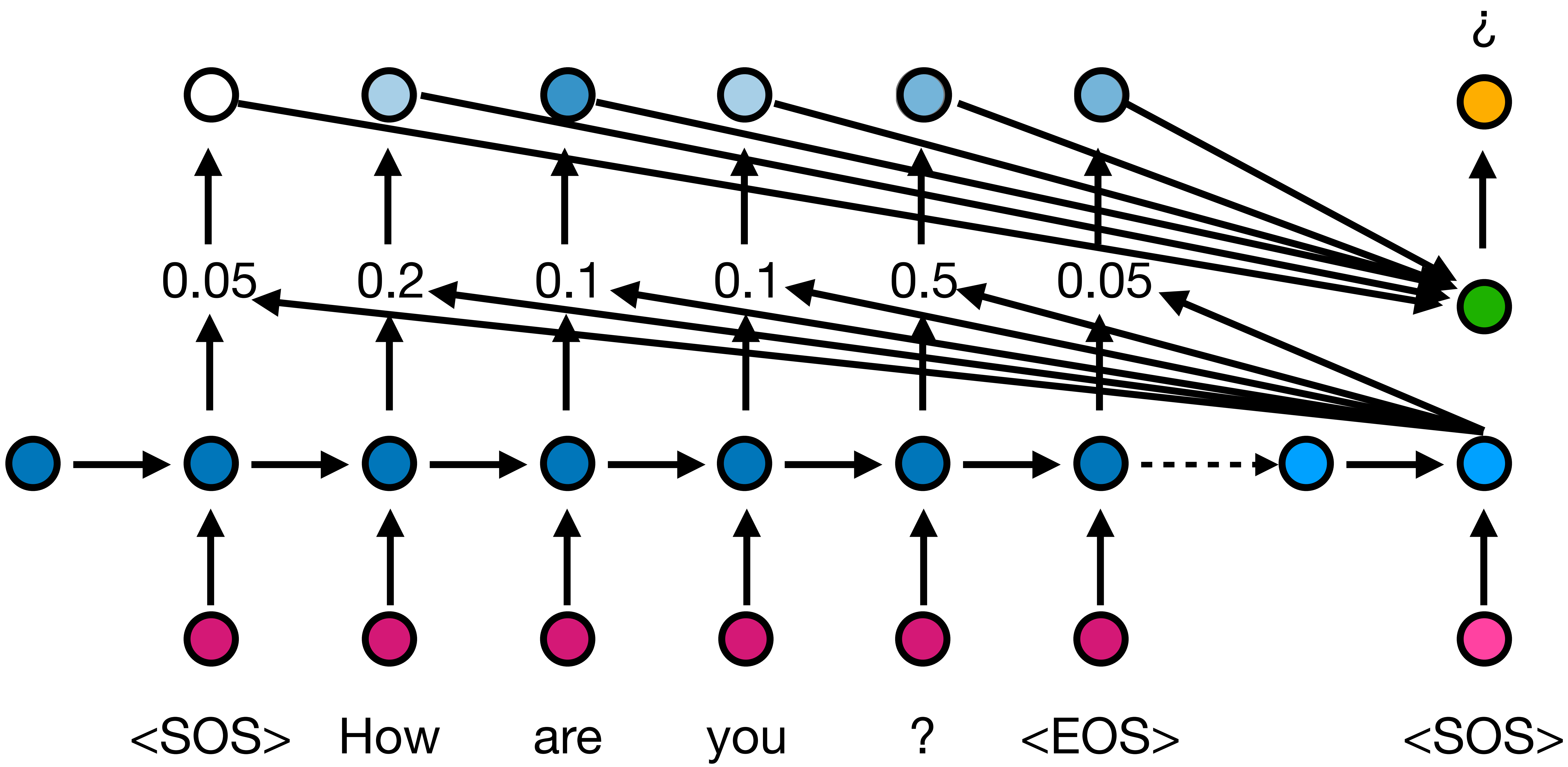
6. Pass context vector through FC layer with softmax to get first predicted token.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

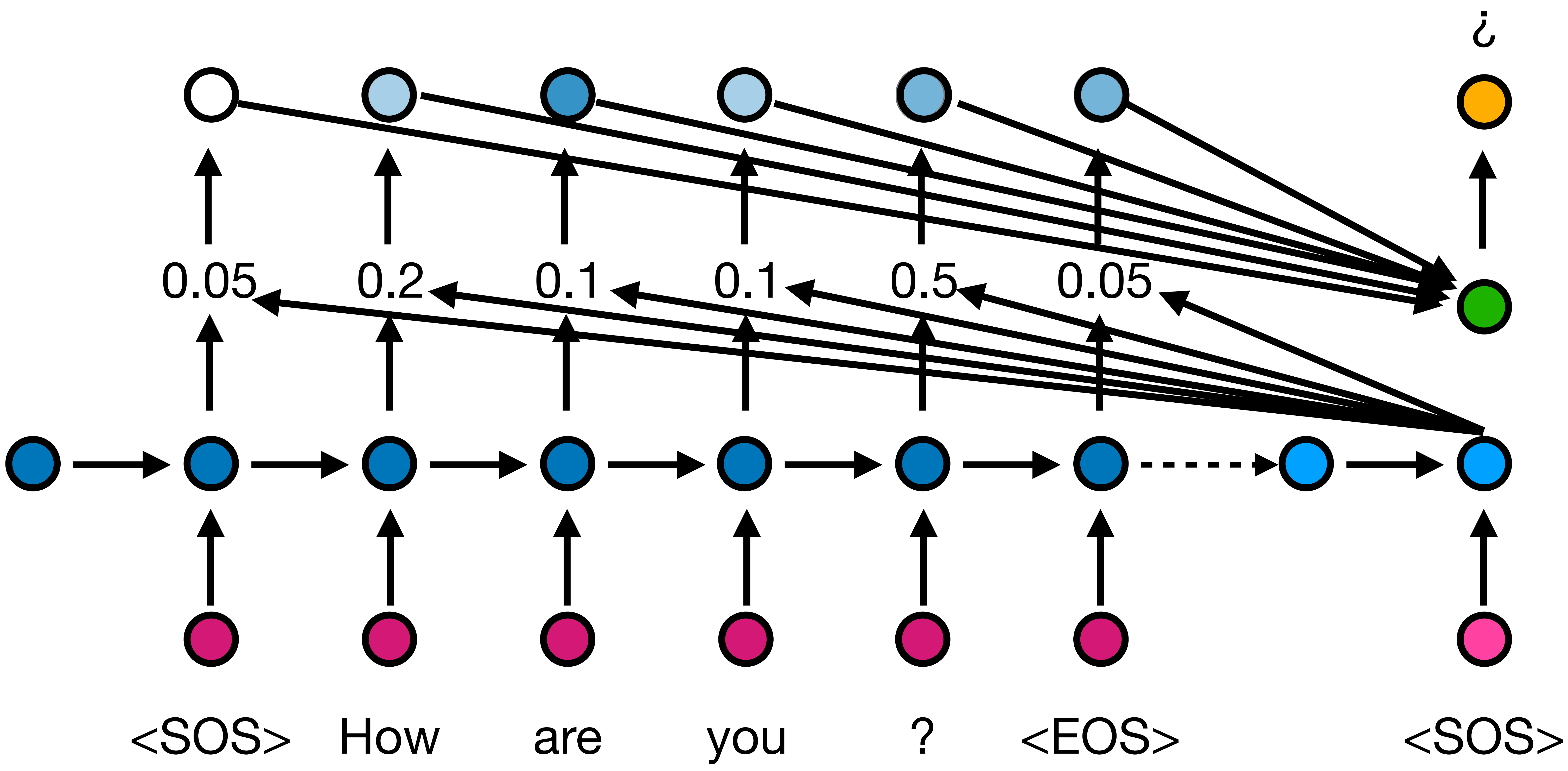
6. Pass context vector through FC layer with softmax to get first predicted token.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

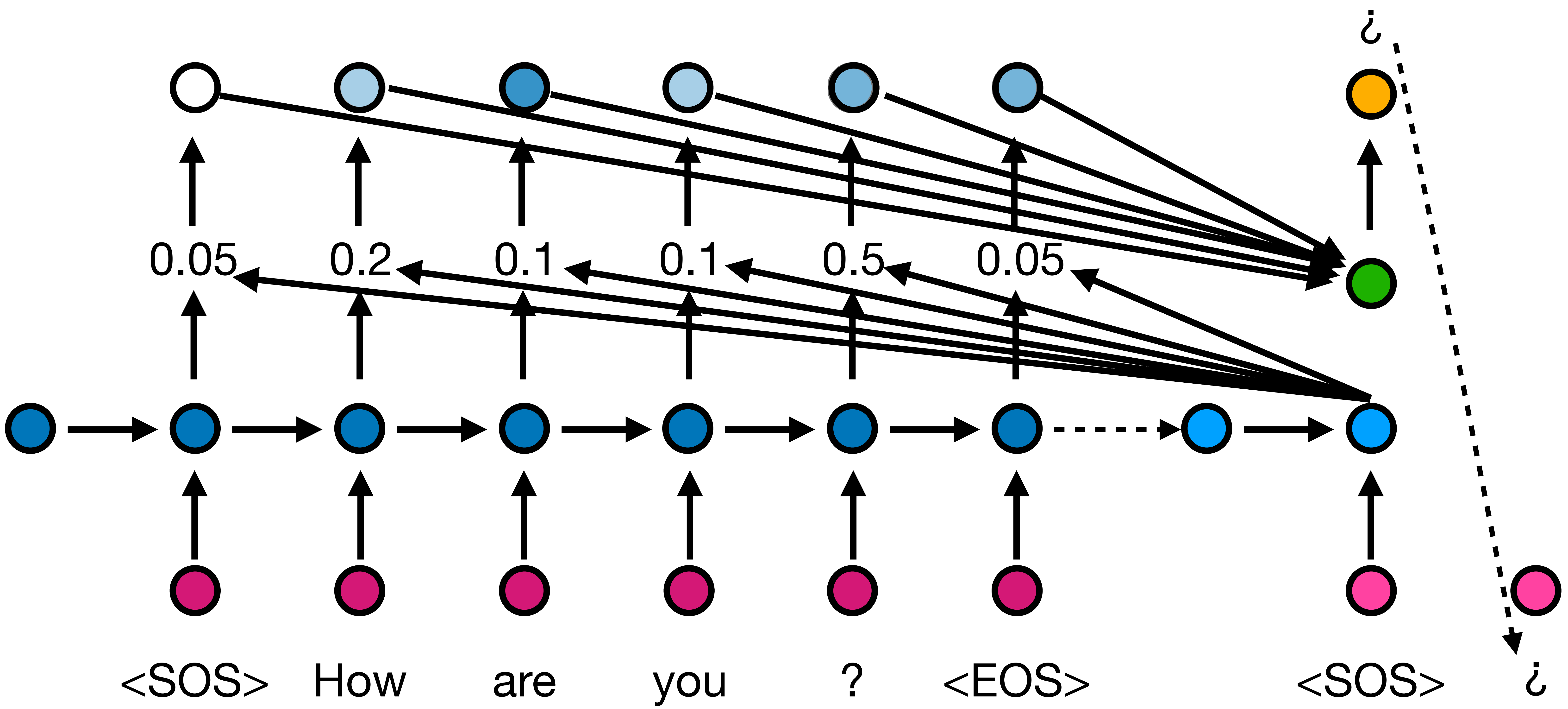
7. Copy first predicted token to input sequence.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

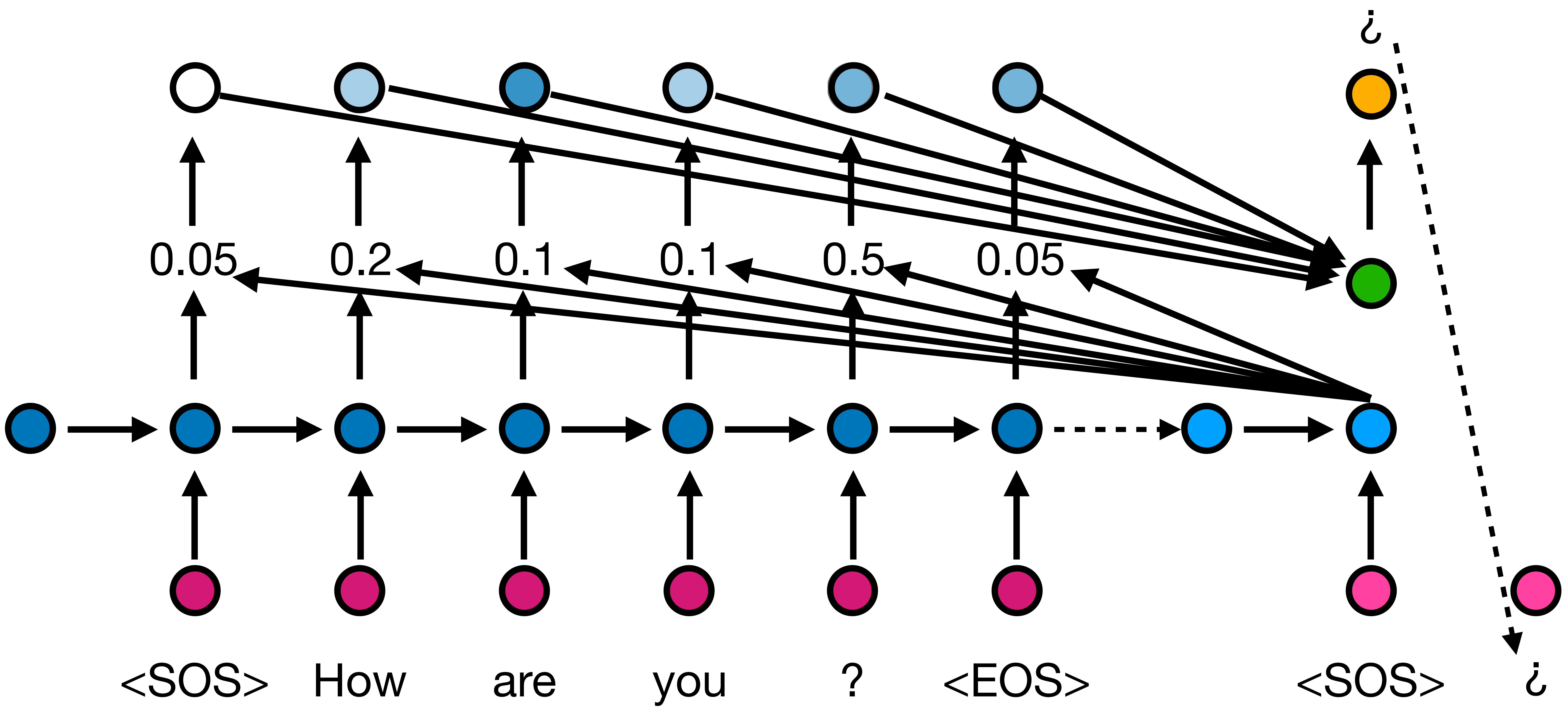
7. Copy first predicted token to input sequence.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

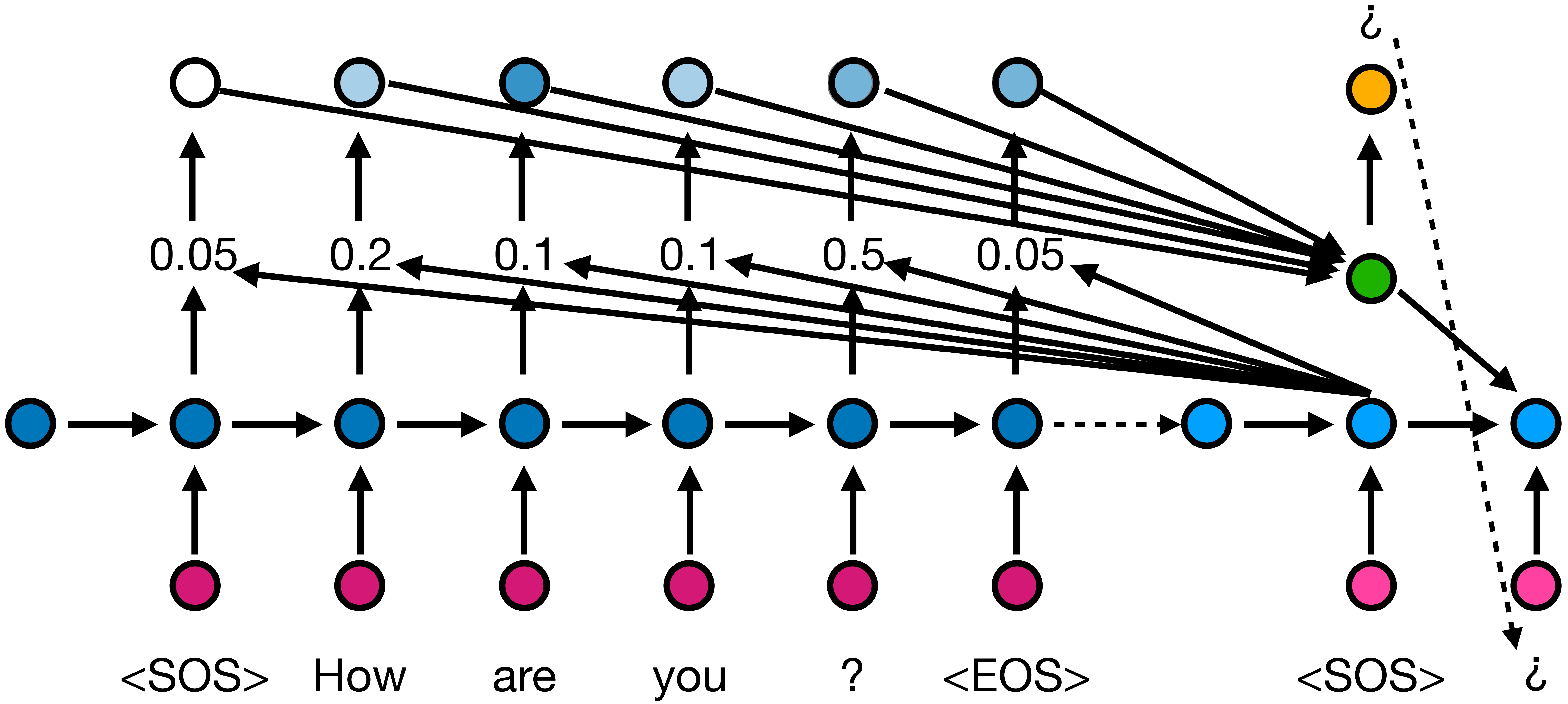
8. Define next hidden state by passing previous context vector, previous hidden state, and input token through FC layers.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

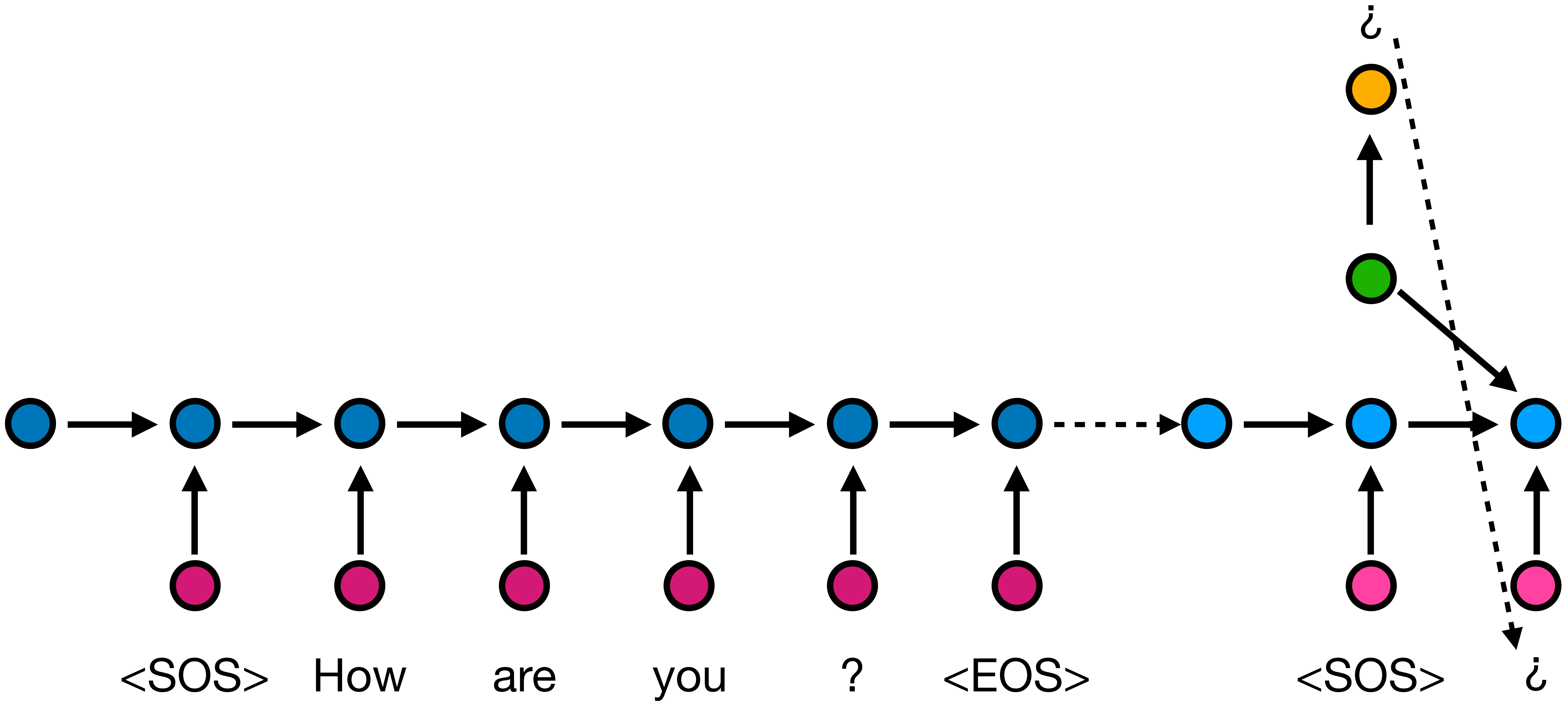
8. Define next hidden state by passing previous context vector, previous hidden state, and input token through FC layers.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

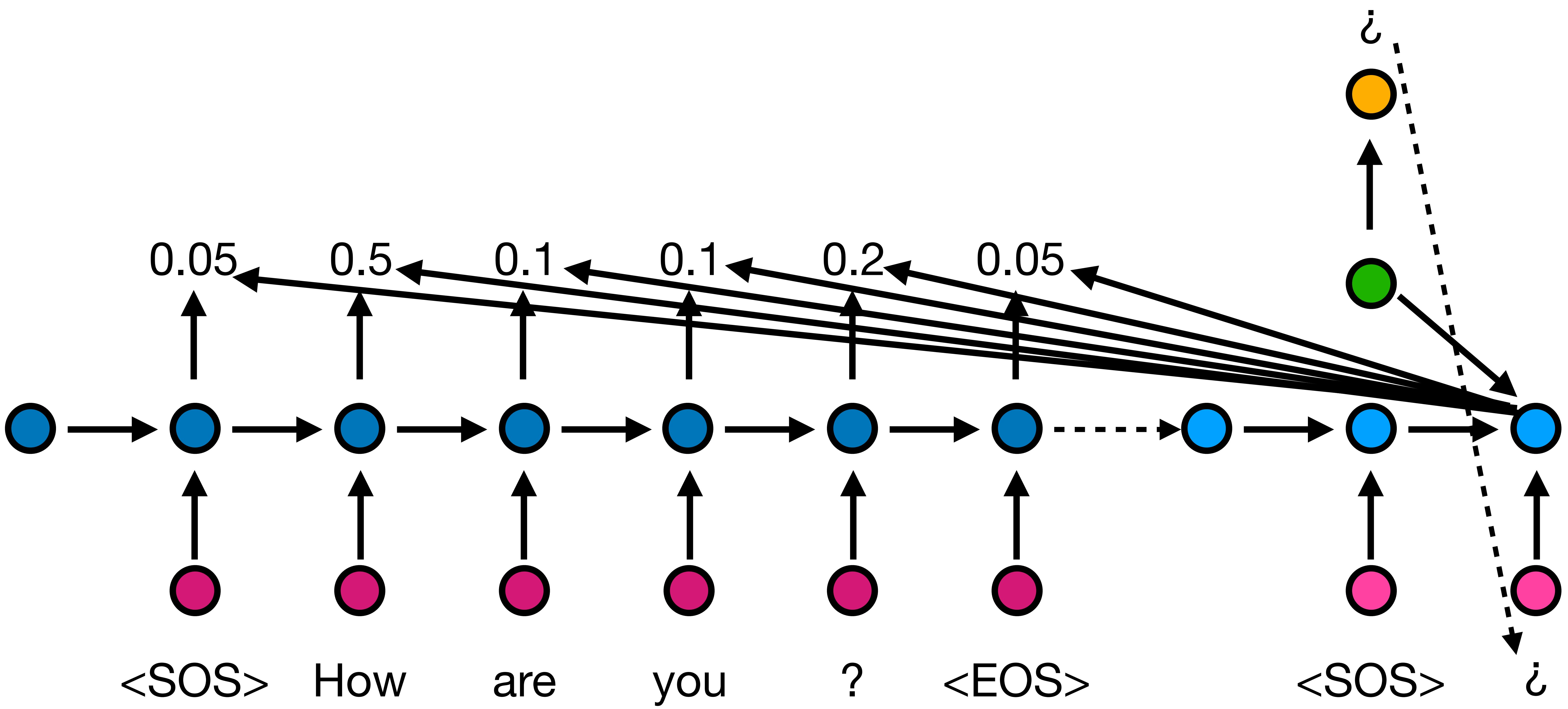
8. Repeat until <EOS> token is reached.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

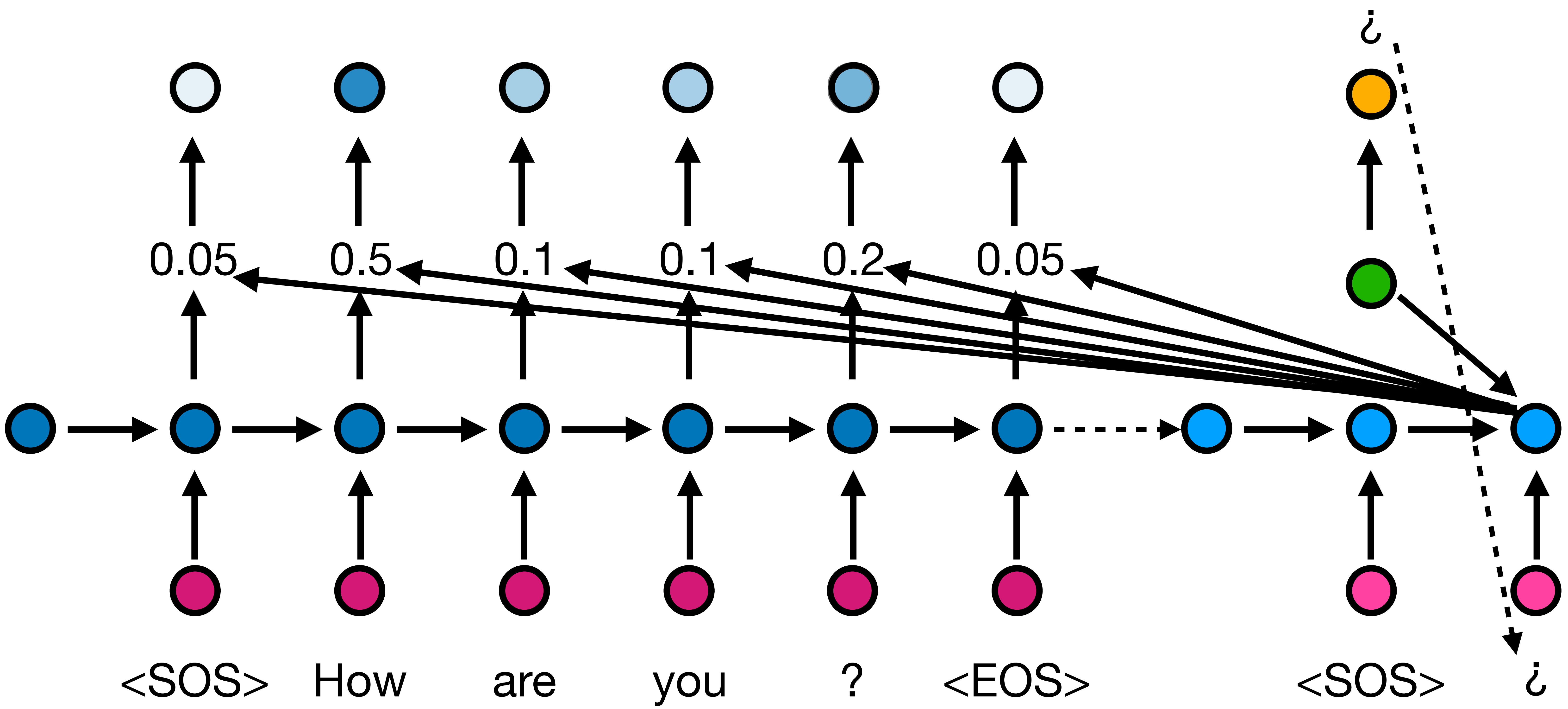
8. Repeat until <EOS> token is reached.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

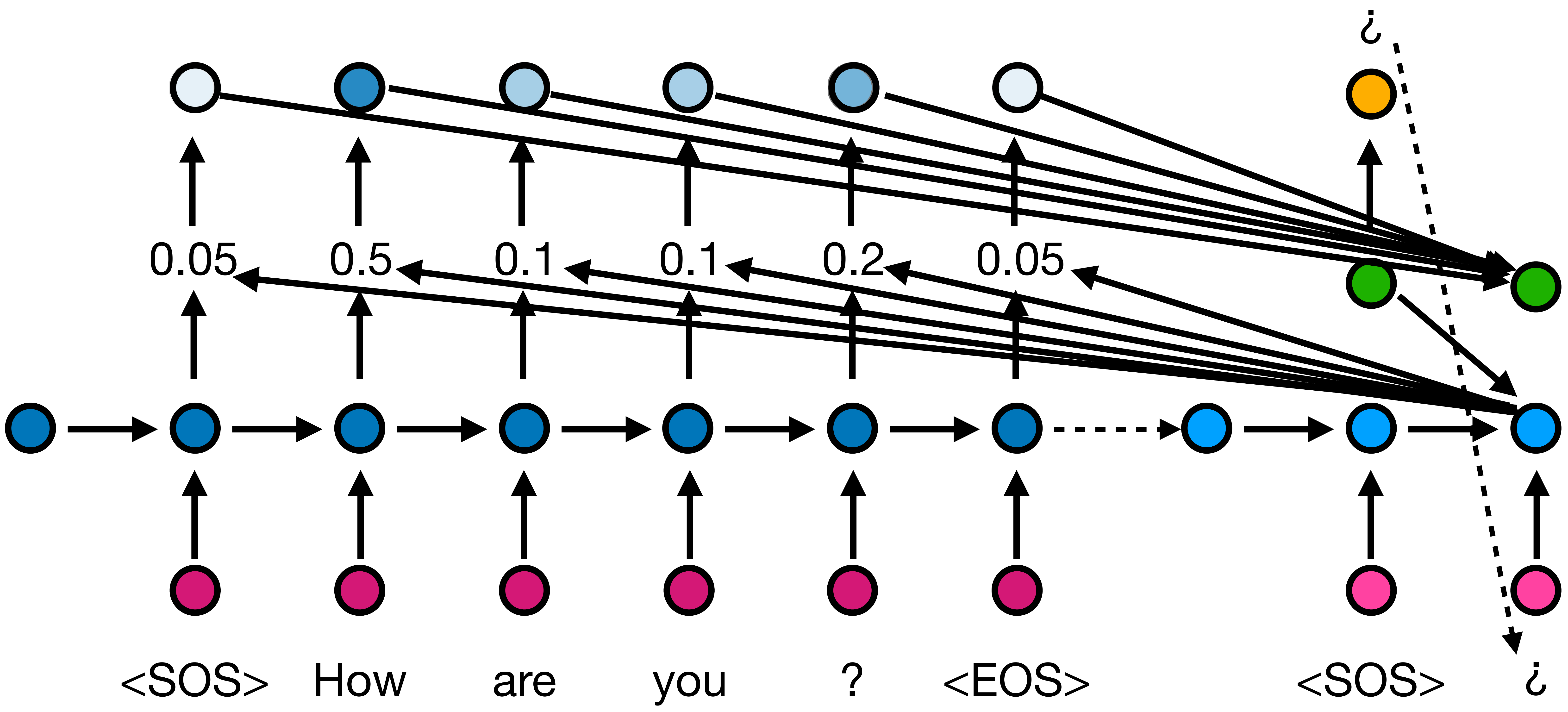
8. Repeat until <EOS> token is reached.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

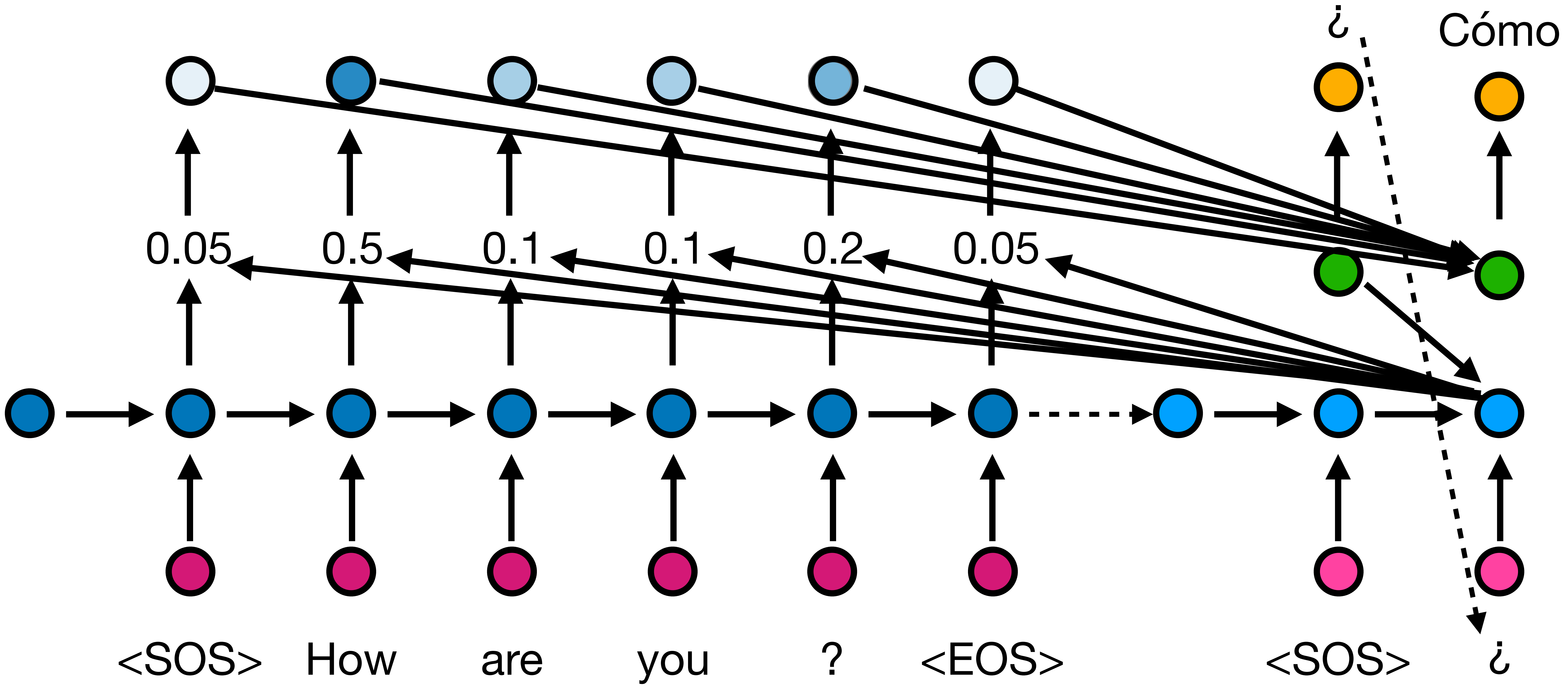
8. Repeat until <EOS> token is reached.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

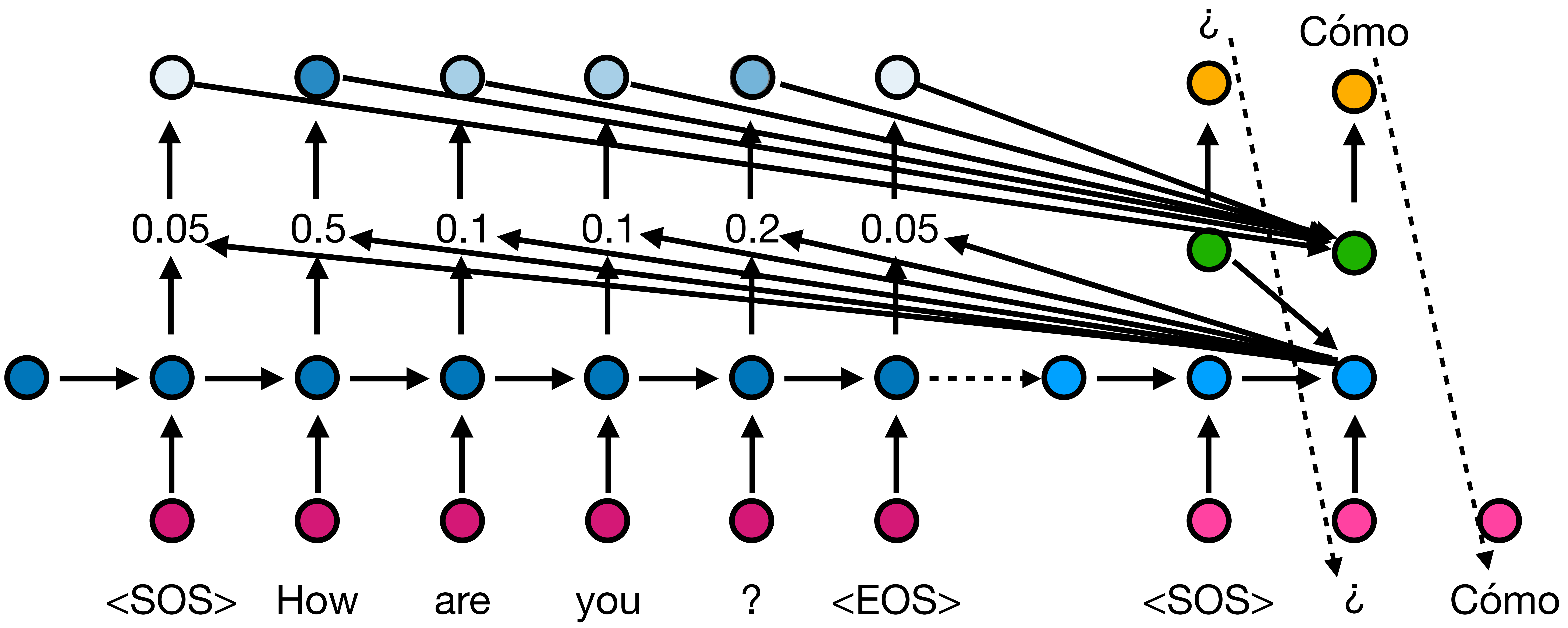
8. Repeat until <EOS> token is reached.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

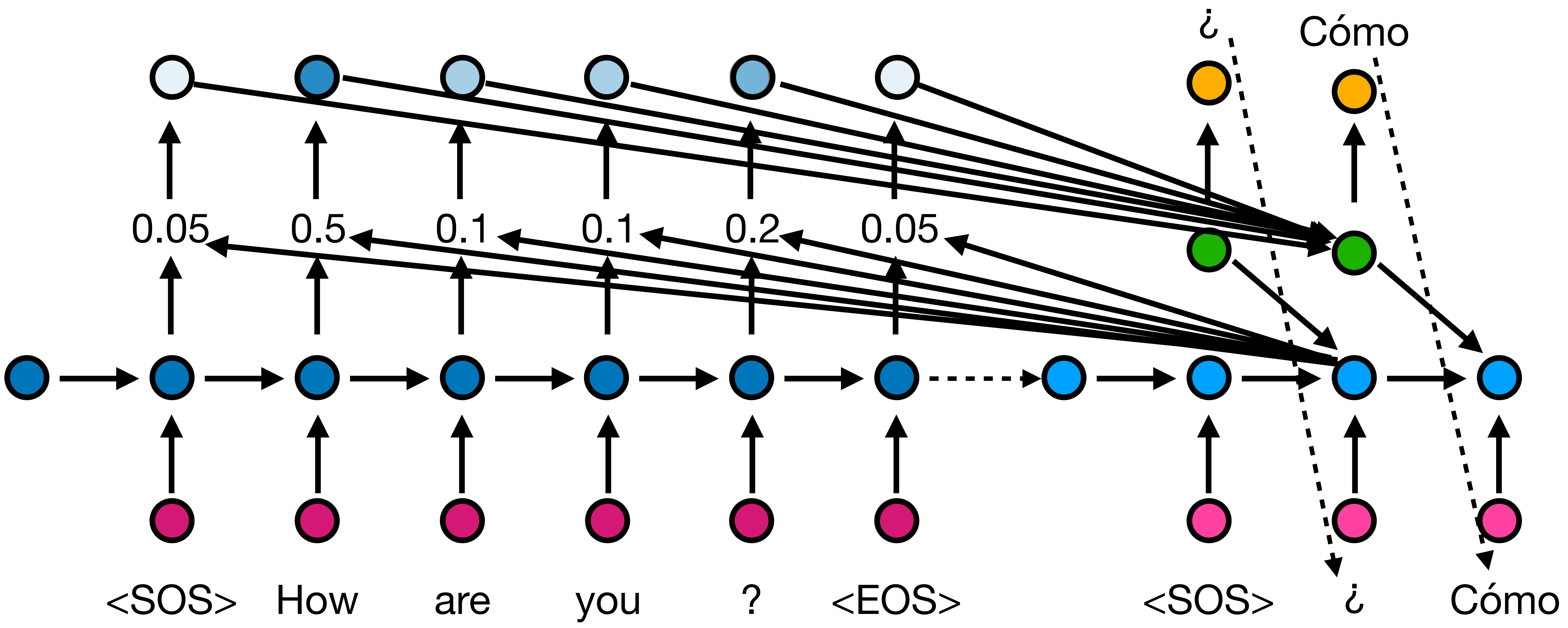
8. Repeat until <EOS> token is reached.



RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

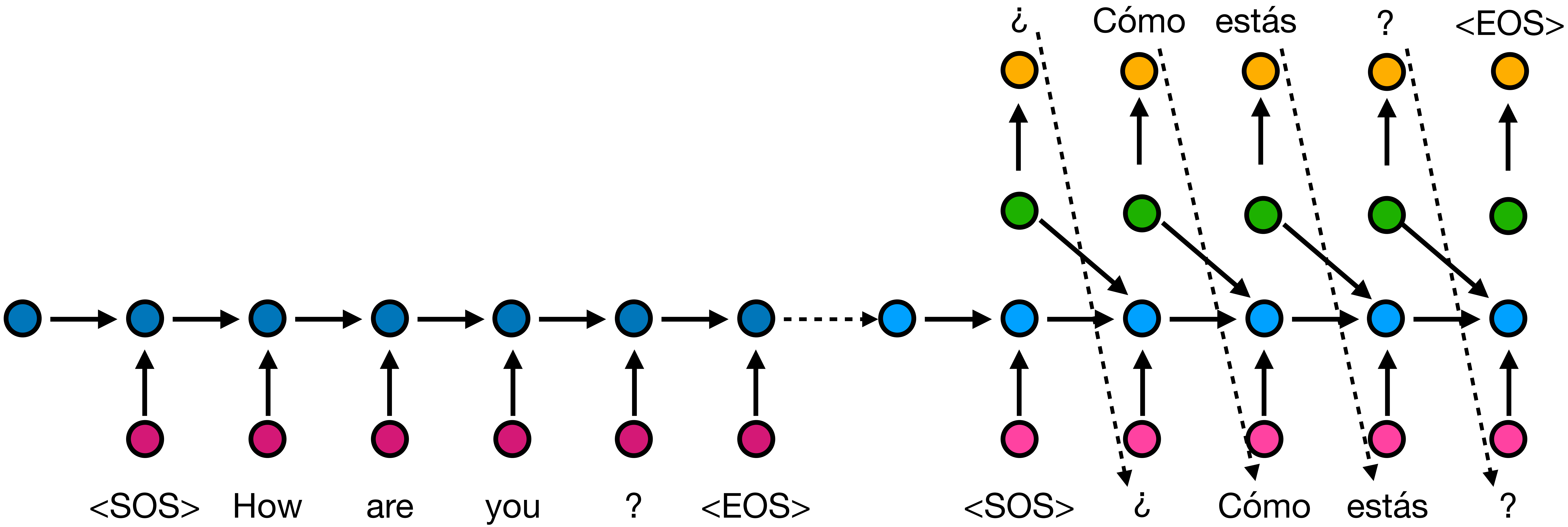
8. Repeat until <EOS> token is reached.



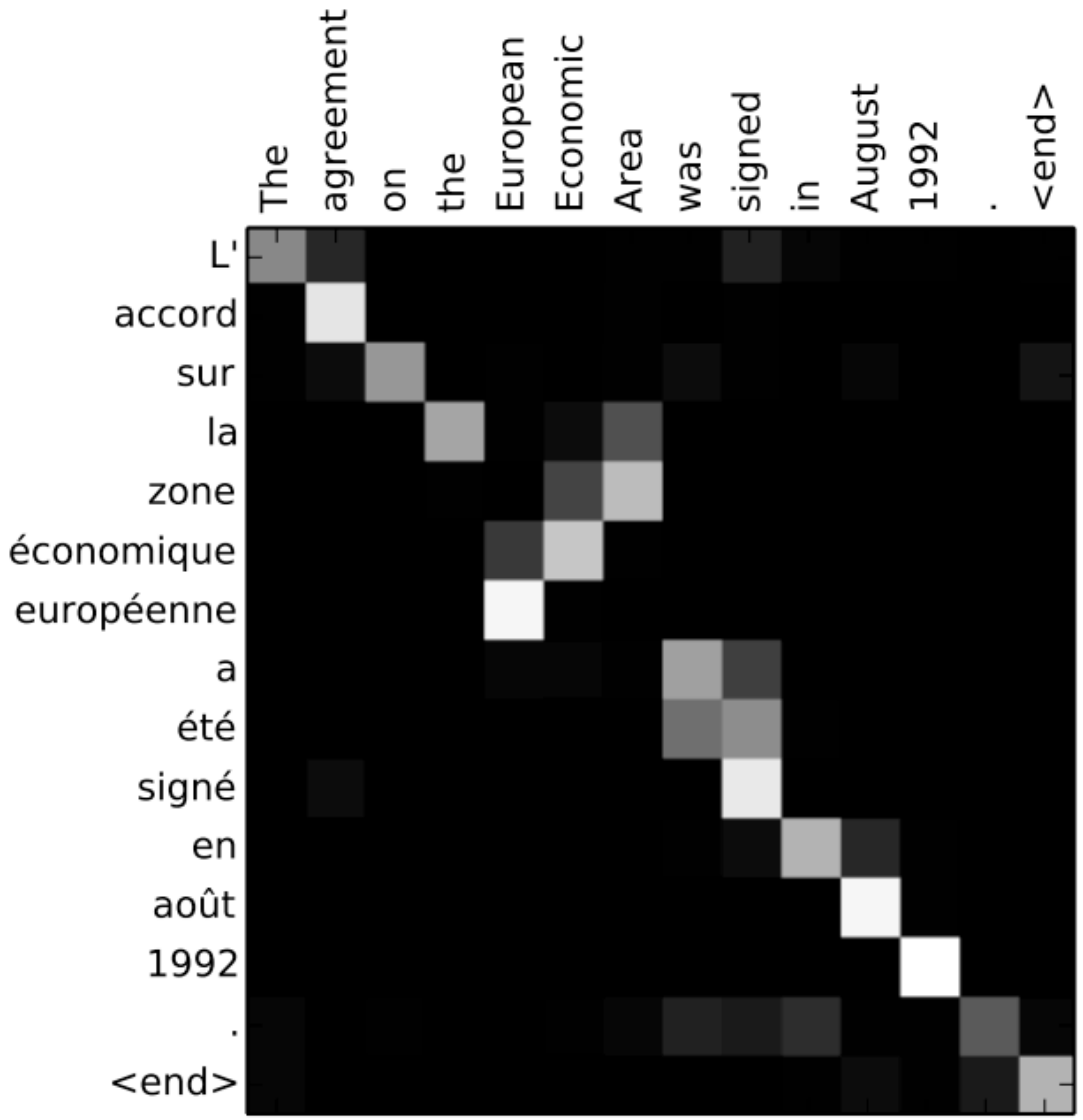
RNN for machine translation with attention

Input text: “How are you?” Output translation: “¿Cómo estás?”

8. Repeat until <EOS> token is reached.



Visualizing attention



https://arxiv.org/pdf/1409.0473.pdf

From RNNs to transformers

From RNNs to transformers

Deep learning models for NLP:

Model	**Popular during**	**Speed**	**Sequence length**
Recurrent neural network (RNN)	1980s to early	Good	5-50 tokens
RNN with “memory” (e.g. LSTMs)	1997 to mid-2010s	Poor	100-500 tokens
LSTMs with attention	Mid-2010s	Very poor	1000+ tokens

From RNNs to transformers

Deep learning models for NLP:

Model	Popular during	Speed	Sequence length
Recurrent neural network (RNN)	1980s to early	Good	5-50 tokens
RNN with “memory” (e.g. LSTMs)	1997 to mid-2010s	Poor	100-500 tokens
LSTMs with attention	Mid-2010s	Very poor	1000+ tokens
Transformers (“attention is all you need”)	2017-present	Excellent	1000+ tokens

From RNNs to transformers

Deep learning models for NLP:

Model	Popular during	Speed	Sequence length
Recurrent neural network (RNN)	1980s to early	Good	5-50 tokens
RNN with “memory” (e.g. LSTMs)	1997 to mid-2010s	Poor	100-500 tokens
LSTMs with attention	Mid-2010s	Very poor	1000+ tokens
Transformers (“attention is all you need”)	2017-present	Excellent	1000+ tokens

RNNs, no matter how fancy, still did not allow parallel processing of inputs.

From RNNs to transformers

Deep learning models for NLP:

Model	Popular during	Speed	Sequence length
Recurrent neural network (RNN)	1980s to early	Good	5-50 tokens
RNN with “memory” (e.g. LSTMs)	1997 to mid-2010s	Poor	100-500 tokens
LSTMs with attention	Mid-2010s	Very poor	1000+ tokens
Transformers (“attention is all you need”)	2017-present	Excellent	1000+ tokens

RNNs, no matter how fancy, still did not allow parallel processing of inputs.

Transformer proposed in 2017: An architecture based on attention but not recurrence, which performed better than RNNs and accommodated parallelization.

Simplified transformer for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

Simplified transformer for sentiment analysis

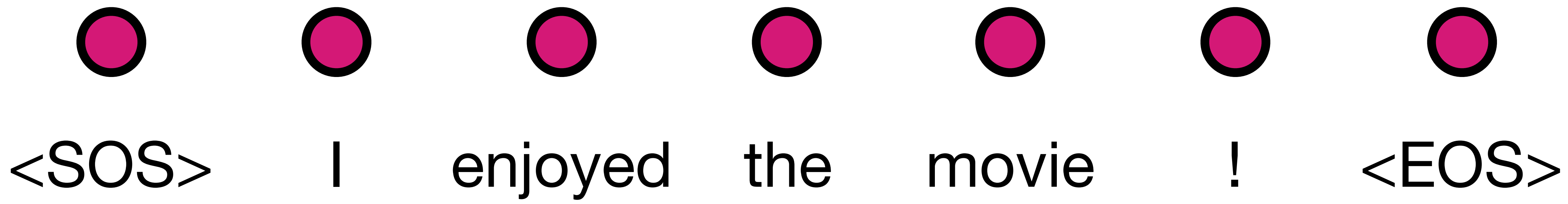
Input text: “I enjoyed the movie!” Output class: “Positive”

1. Start with each input token’s word vector.

Simplified transformer for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

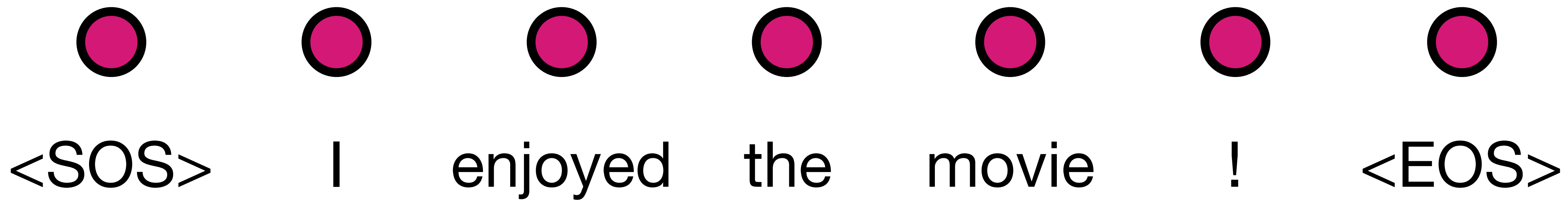
1. Start with each input token’s word vector.



Simplified transformer for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

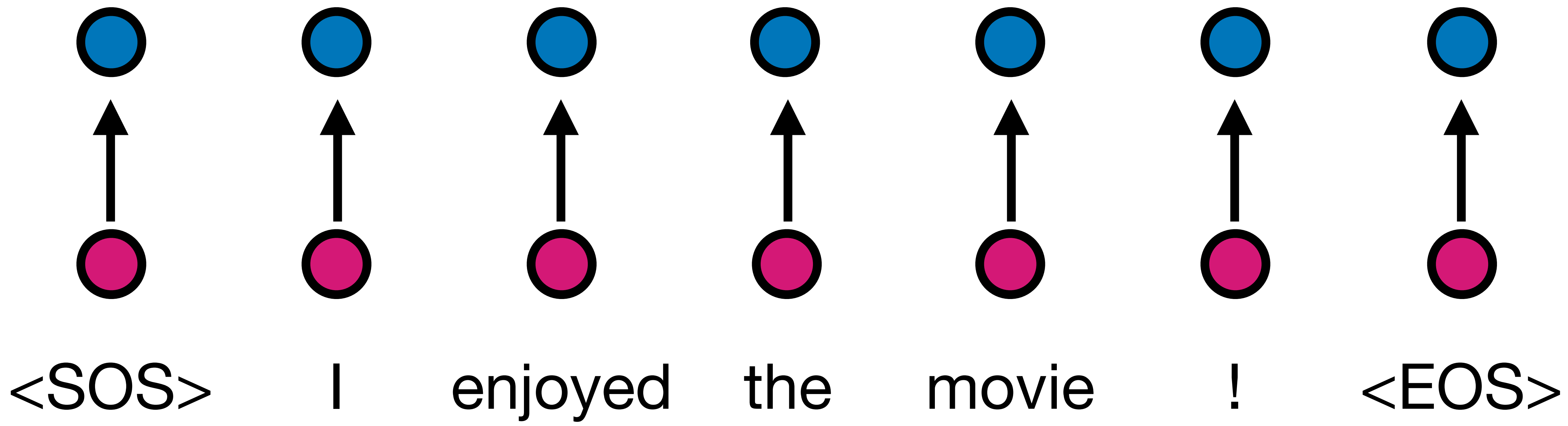
1. Start with each input token’s word vector.
2. Compute each input token’s *value* by passing its word vector through FC layer.



Simplified transformer for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

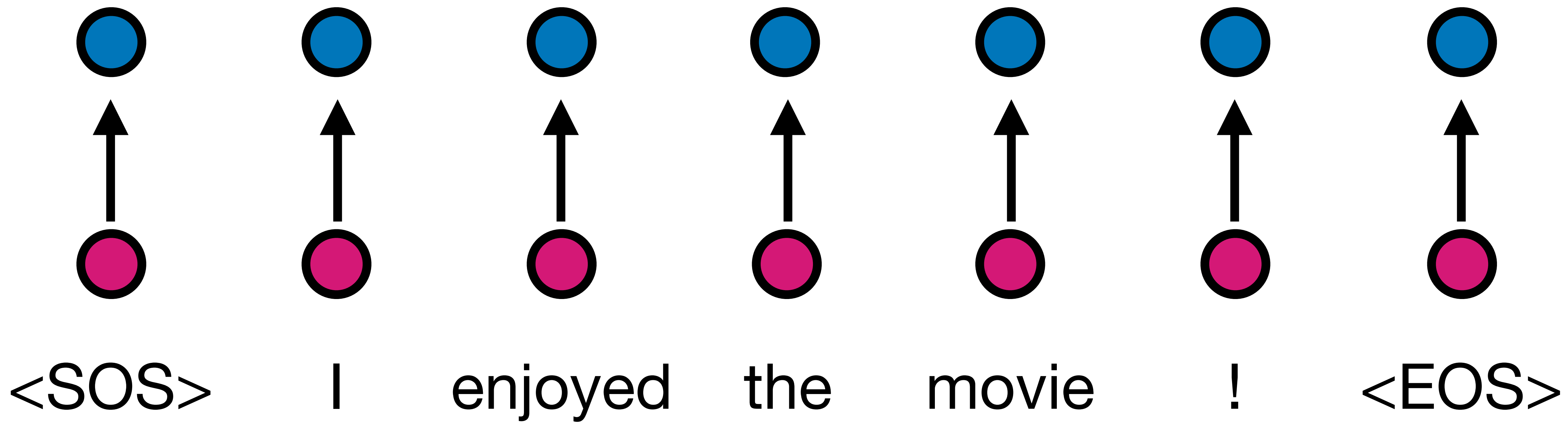
1. Start with each input token’s word vector.
2. Compute each input token’s *value* by passing its word vector through FC layer.



Simplified transformer for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

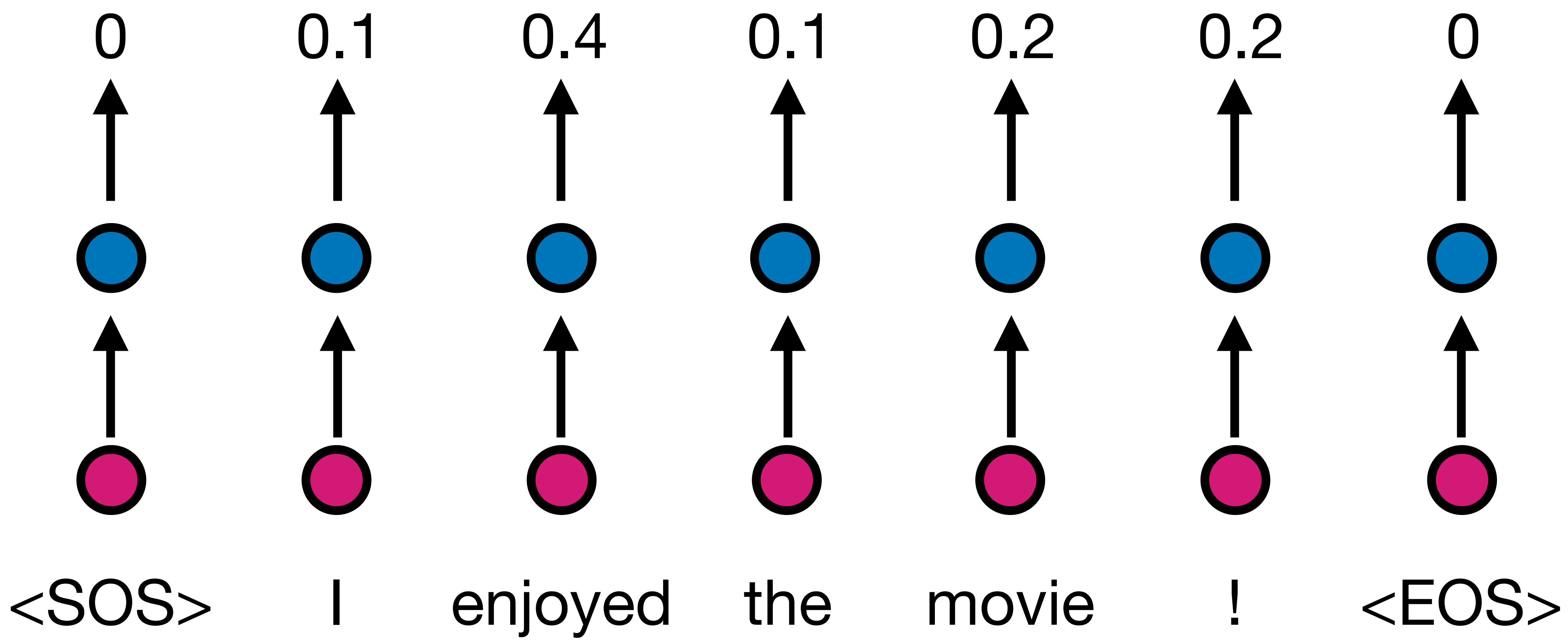
1. Start with each input token’s word vector.
2. Compute each input token’s *value* by passing its word vector through FC layer.
3. Pass each token value through another FC layer with softmax to get *attention weights.*



Simplified transformer for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

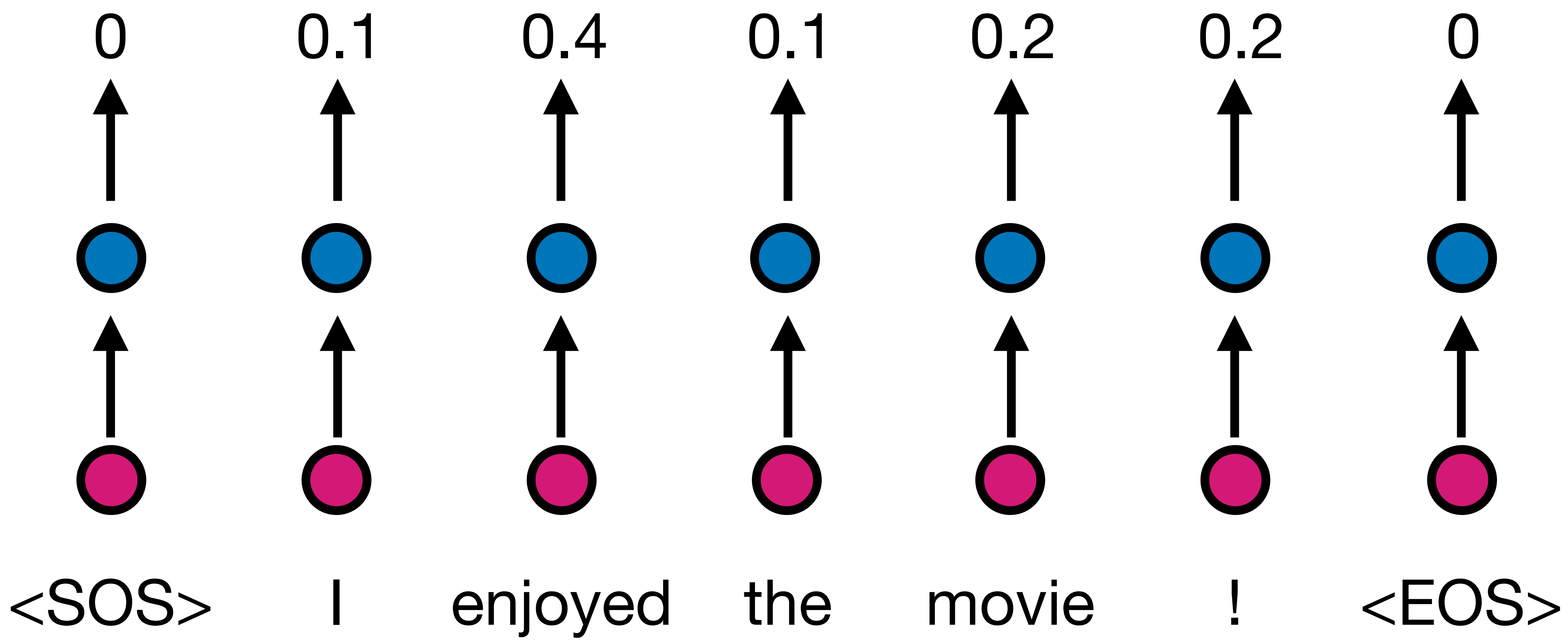
1. Start with each input token’s word vector.
2. Compute each input token’s *value* by passing its word vector through FC layer.
3. Pass each token value through another FC layer with softmax to get *attention weights.*



Simplified transformer for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

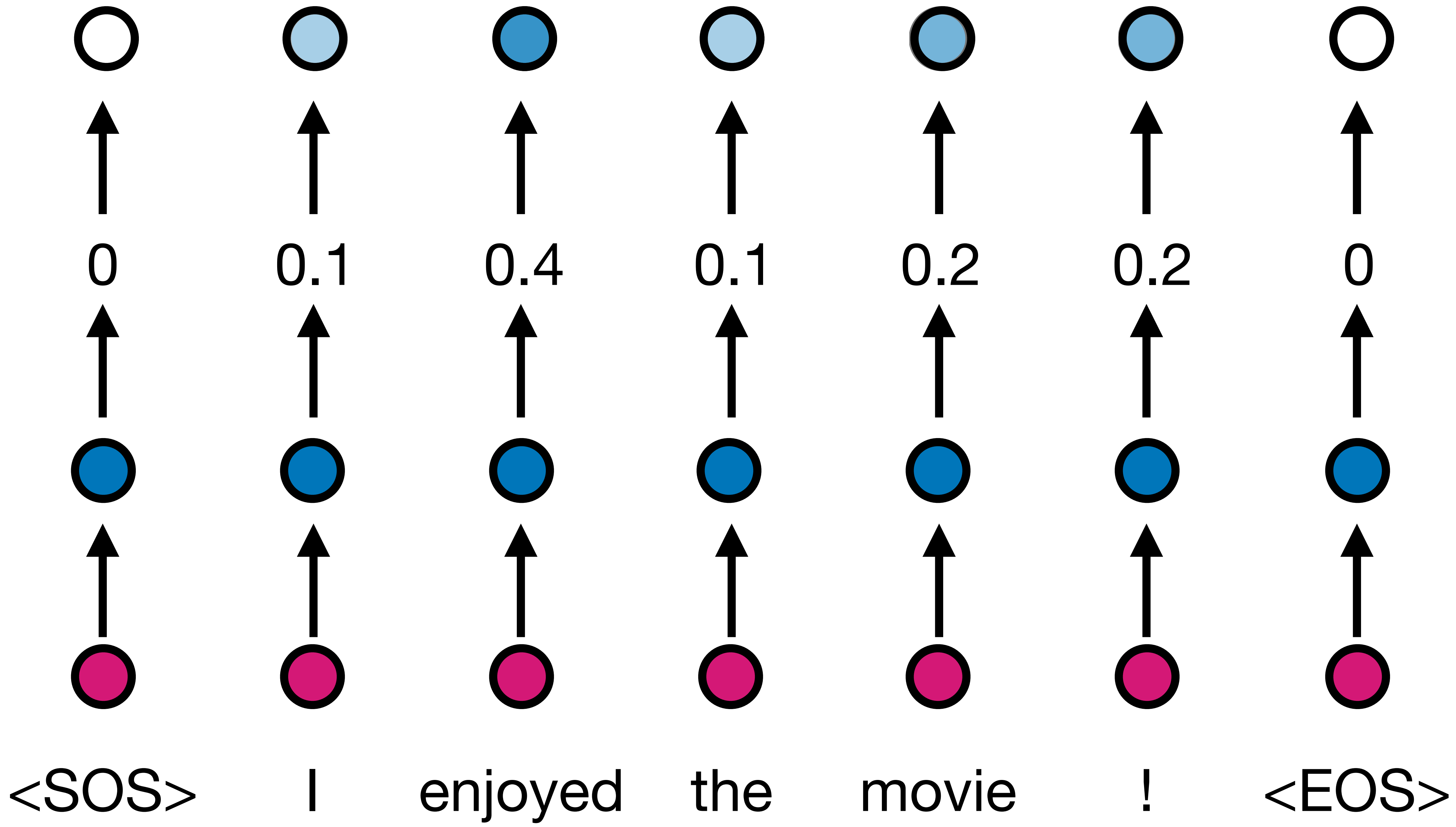
1. Start with each input token’s word vector.
2. Compute each input token’s *value* by passing its word vector through FC layer.
3. Pass each token value through another FC layer with softmax to get *attention weights.*
4. Take weighted average of token values to get *context vector*.



Simplified transformer for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

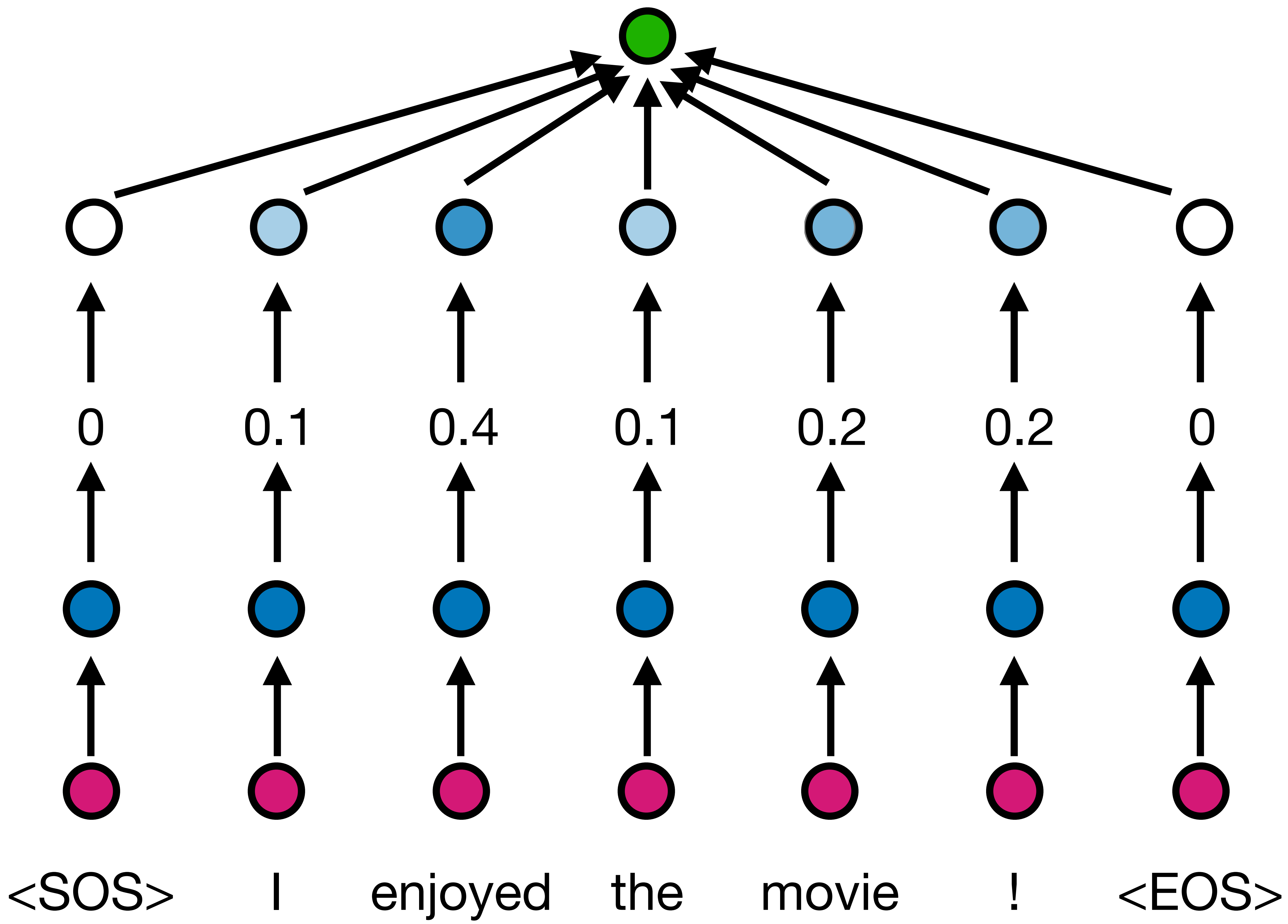
1. Start with each input token’s word vector.
2. Compute each input token’s *value* by passing its word vector through FC layer.
3. Pass each token value through another FC layer with softmax to get *attention weights.*
4. Take weighted average of token values to get *context vector*.



Simplified transformer for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

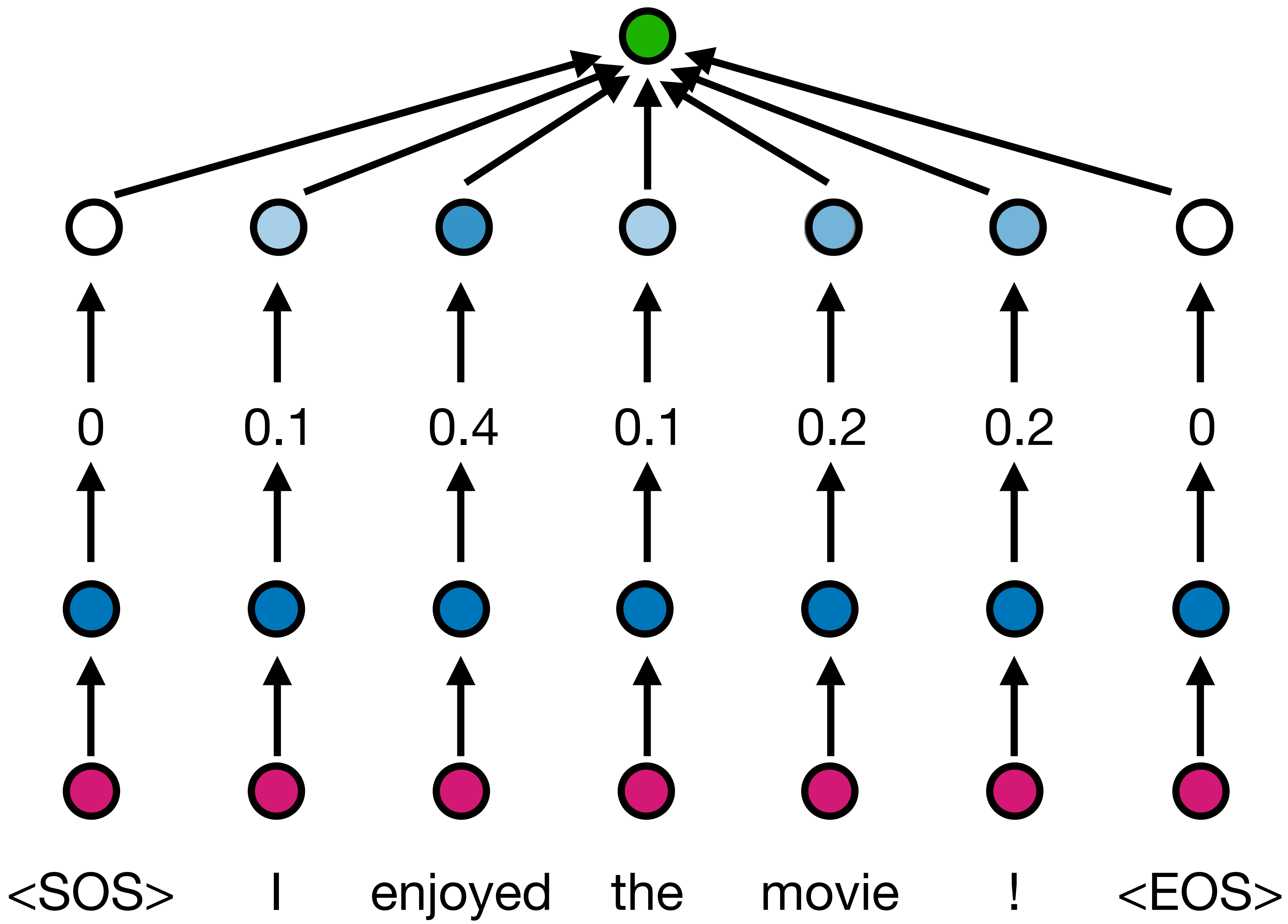
1. Start with each input token’s word vector.
2. Compute each input token’s *value* by passing its word vector through FC layer.
3. Pass each token value through another FC layer with softmax to get *attention weights.*
4. Take weighted average of token values to get *context vector*.



Simplified transformer for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

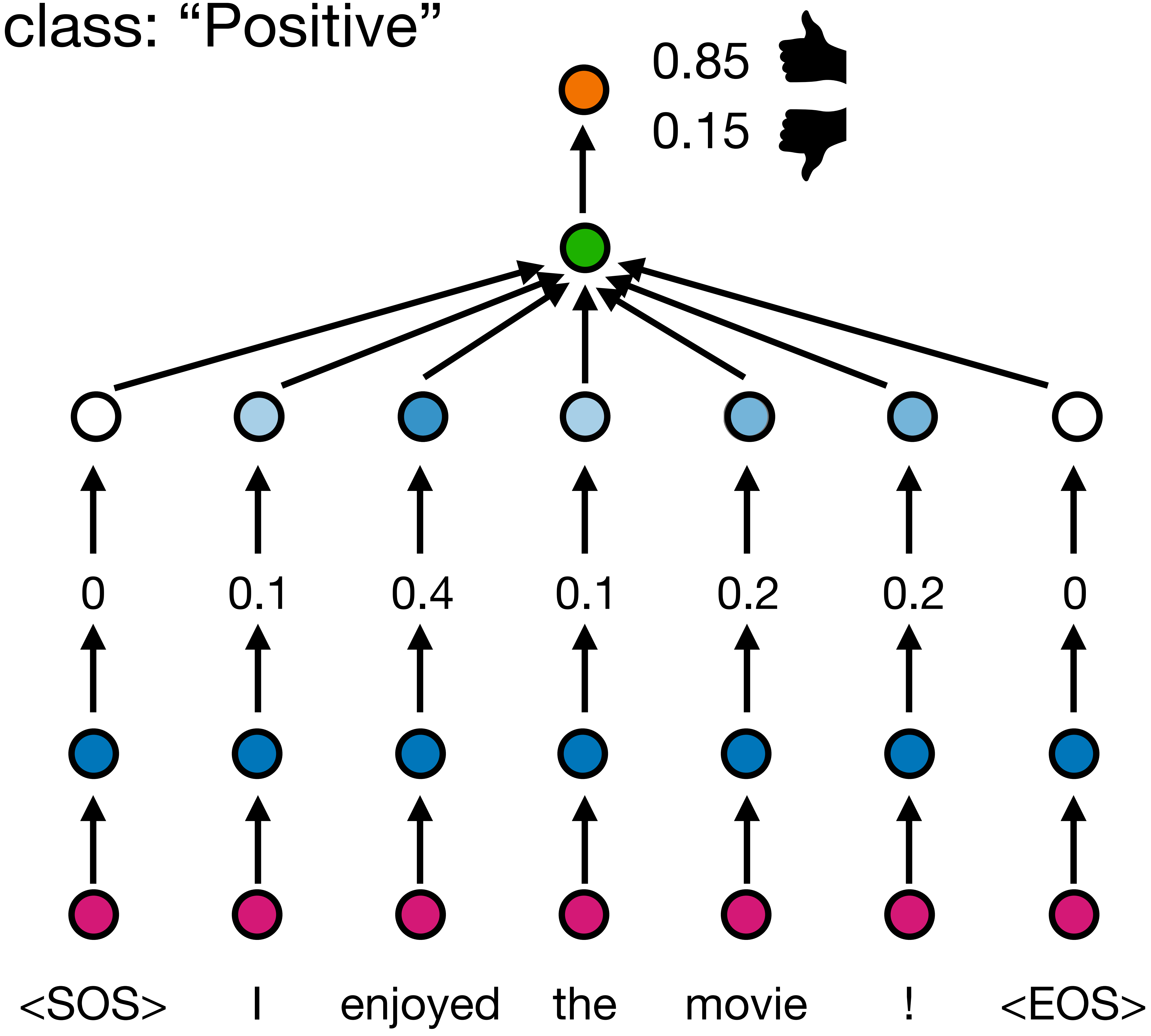
1. Start with each input token’s word vector.
2. Compute each input token’s *value* by passing its word vector through FC layer.
3. Pass each token value through another FC layer with softmax to get *attention weights.*
4. Take weighted average of token values to get *context vector*.
5. Pass context vector through FC layer with softmax to get output class probabilities.



Simplified transformer for sentiment analysis

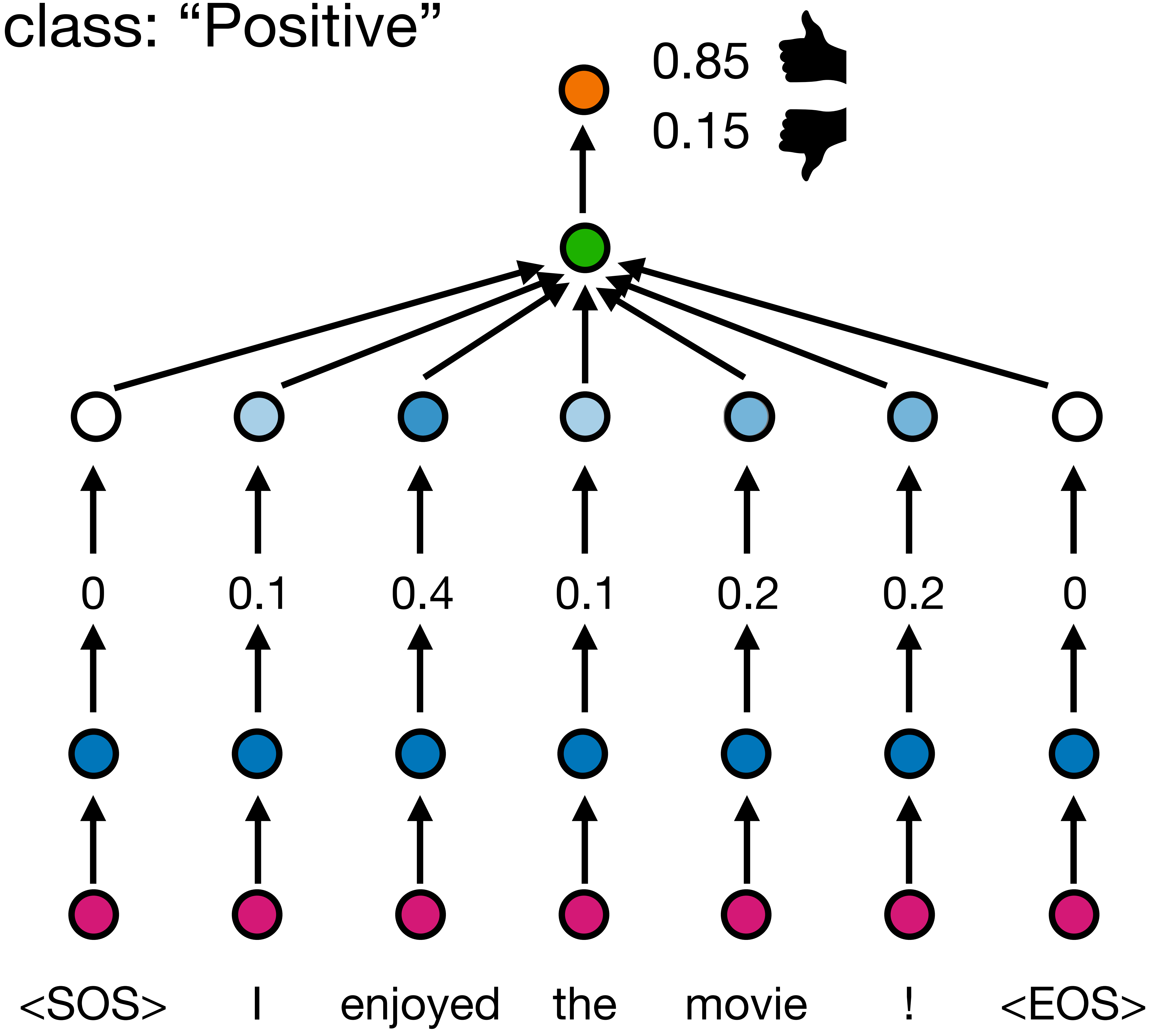
Input text: “I enjoyed the movie!” Output class: “Positive”

1. Start with each input token’s word vector.
2. Compute each input token’s *value* by passing its word vector through FC layer.
3. Pass each token value through another FC layer with softmax to get *attention weights.*
4. Take weighted average of token values to get *context vector*.
5. Pass context vector through FC layer with softmax to get output class probabilities.



Simplified transformer for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”



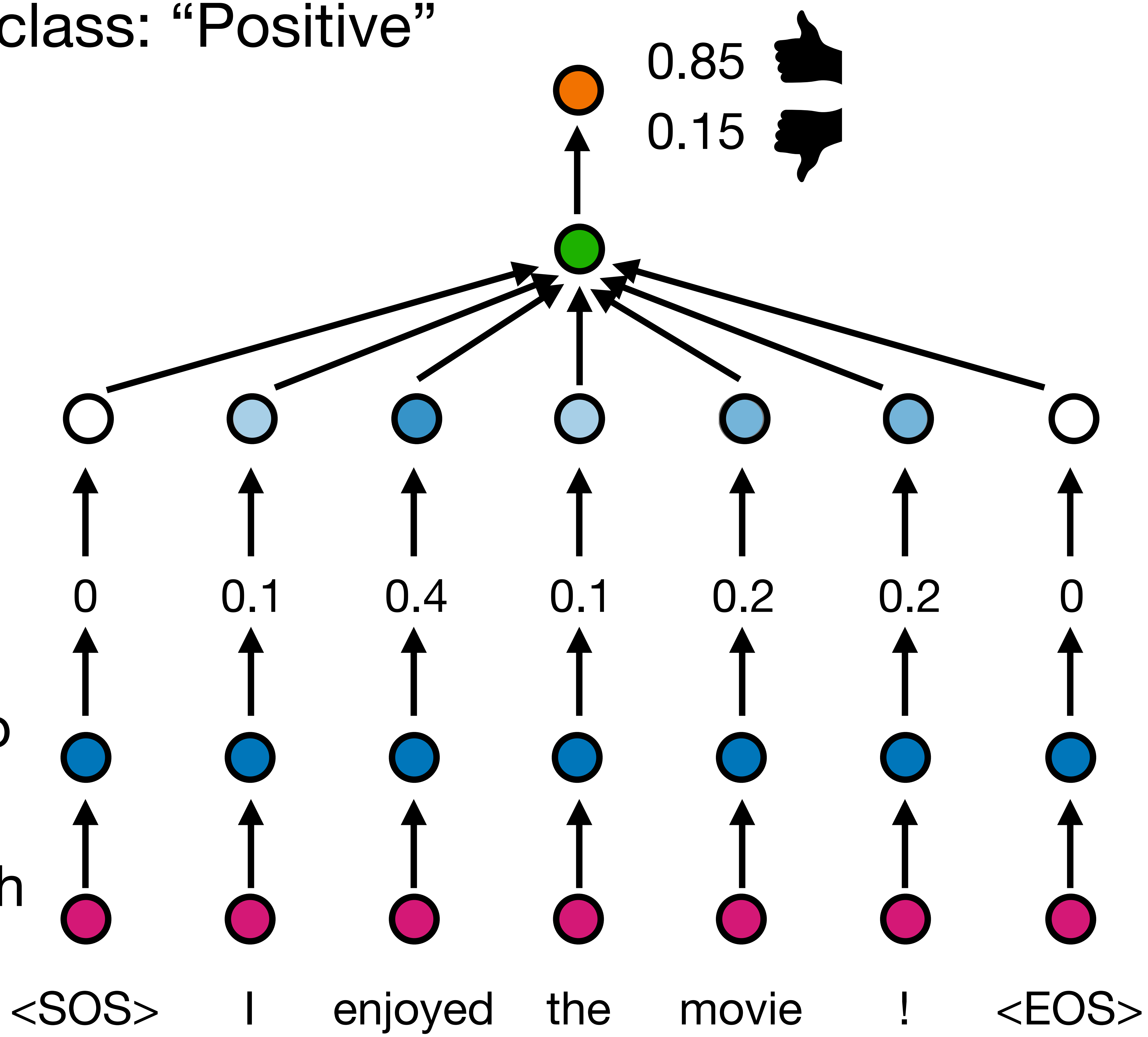
1. Start with each input token’s word vector.
2. Compute each input token’s *value* by passing its word vector through FC layer.
3. Pass each token value through another FC layer with softmax to get *attention weights.*
4. Take weighted average of token values to get *context vector*.
5. Pass context vector through FC layer with softmax to get output class probabilities.

This strategy ignores word order!

Simplified transformer for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

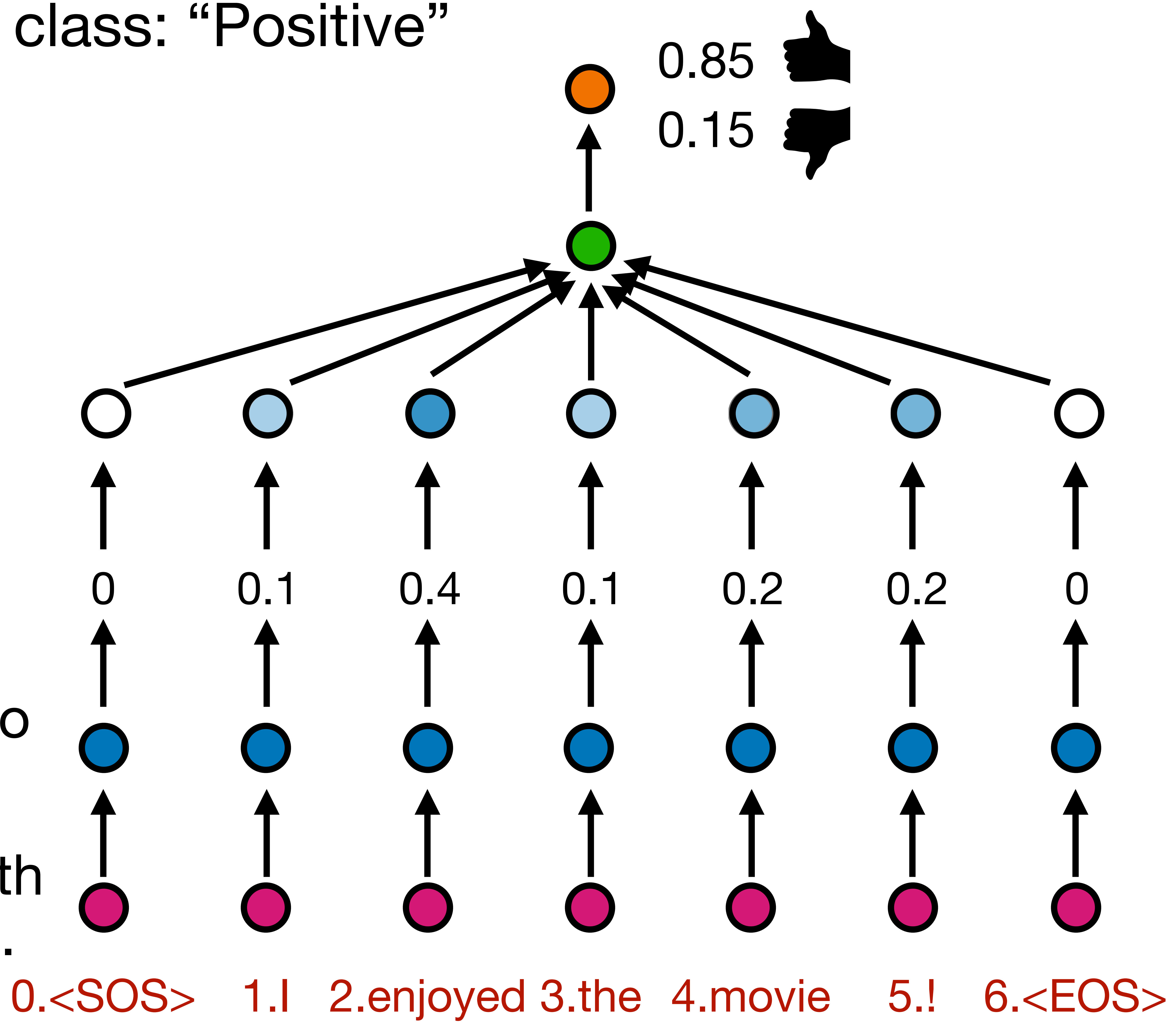
1. Positionally encode each input token’s word vector by appending its index.
2. Compute each input token’s *value* by passing its positional encoding vector through FC layer.
3. Pass each token value through another FC layer with softmax to get *attention weights.*
4. Take weighted average of token values to get *context vector*.
5. Pass context vector through FC layer with softmax to get output class probabilities.



Simplified transformer for sentiment analysis

Input text: “I enjoyed the movie!” Output class: “Positive”

1. Positionally encode each input token’s word vector by appending its index.
2. Compute each input token’s *value* by passing its positional encoding vector through FC layer.
3. Pass each token value through another FC layer with softmax to get *attention weights.*
4. Take weighted average of token values to get *context vector*.
5. Pass context vector through FC layer with softmax to get output class probabilities.



Simplified transformer for language modeling

Input text: “my favorite season is” Output word: “spring”

Simplified transformer for language modeling

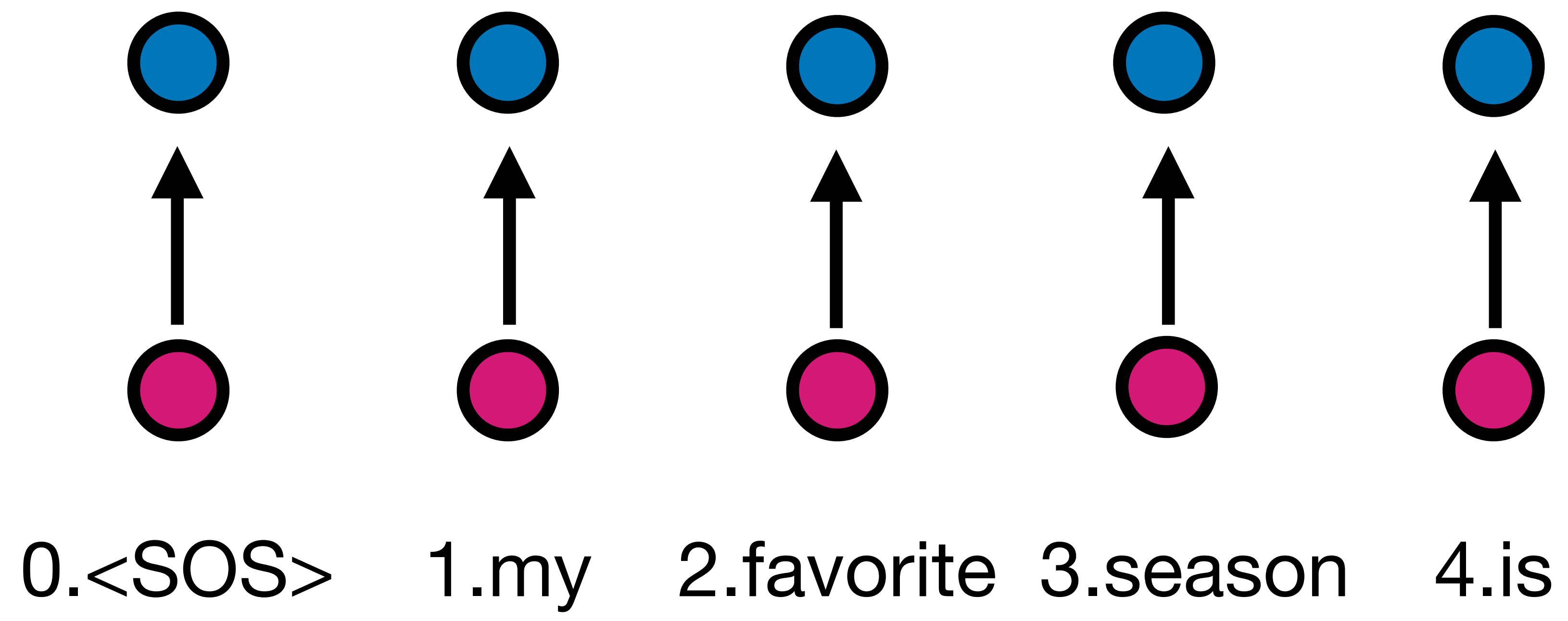
Input text: “my favorite season is” Output word: “spring”

1. *Positionally encode* each input token and pass it through FC layer to compute *values.*

Simplified transformer for language modeling

Input text: “my favorite season is” Output word: “spring”

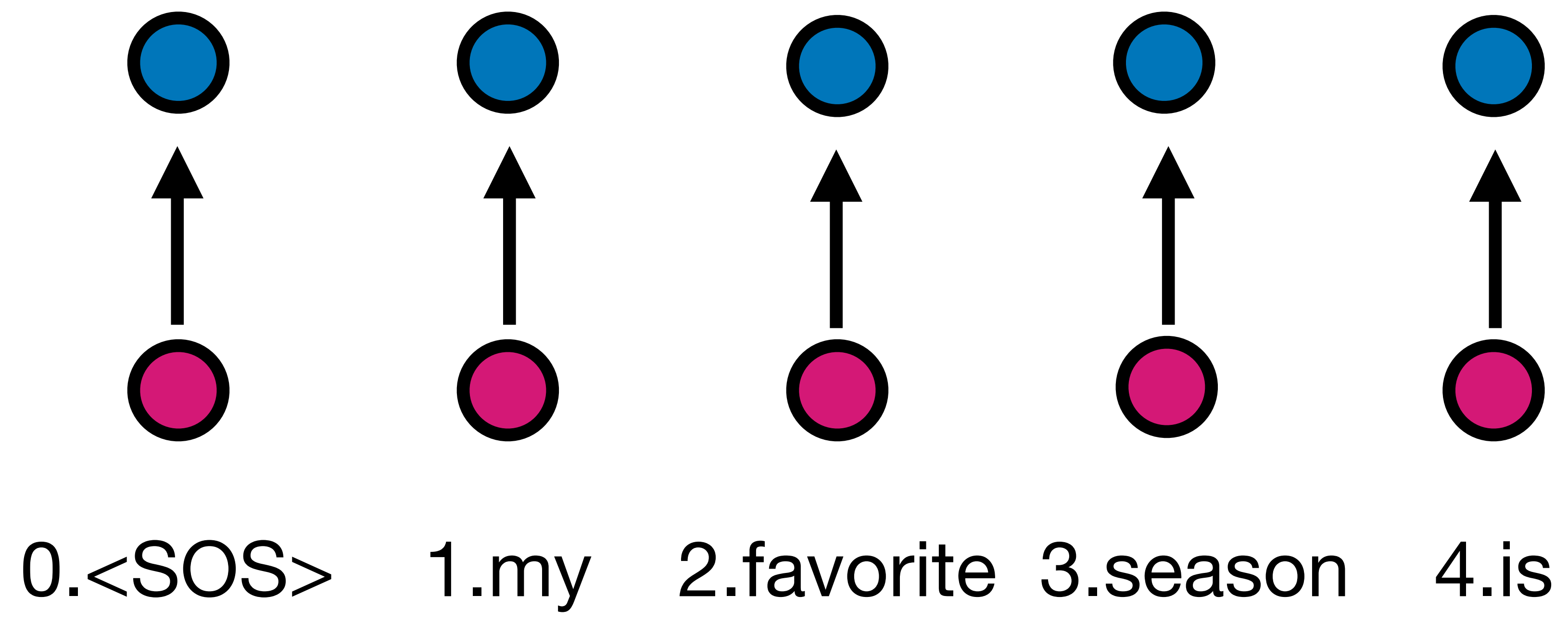
1. *Positionally encode* each input token and pass it through FC layer to compute *values.*



Simplified transformer for language modeling

Input text: “my favorite season is” Output word: “spring”

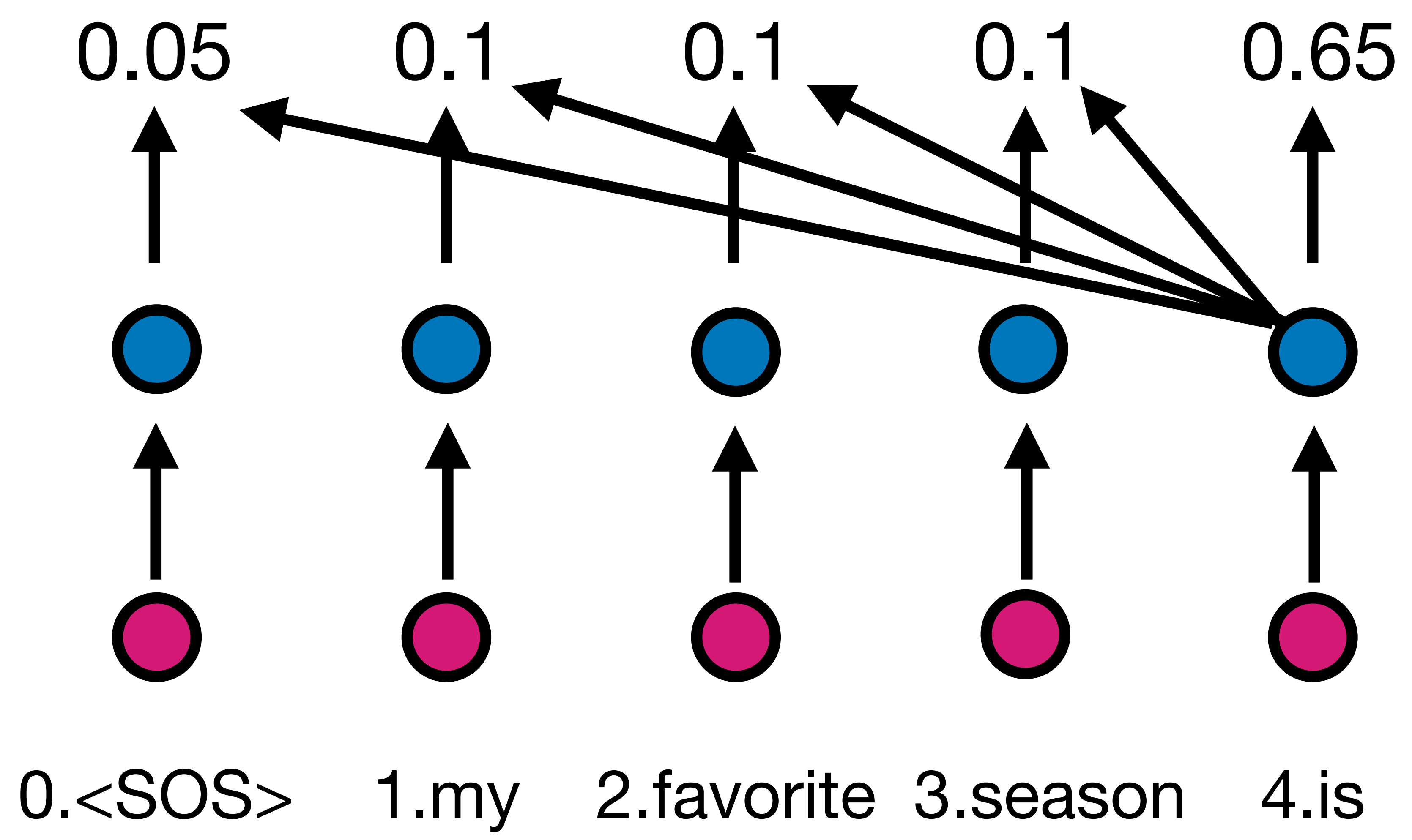
1. *Positionally encode* each input token and pass it through FC layer to compute *values.*
2. Calculate attention weights for “is” by comparing its value to its own and those of preceding tokens (*self-attention*).



Simplified transformer for language modeling

Input text: “my favorite season is” Output word: “spring”

1. *Positionally encode* each input token and pass it through FC layer to compute *values.*
2. Calculate attention weights for “is” by comparing its value to its own and those of preceding tokens (*self-attention*).



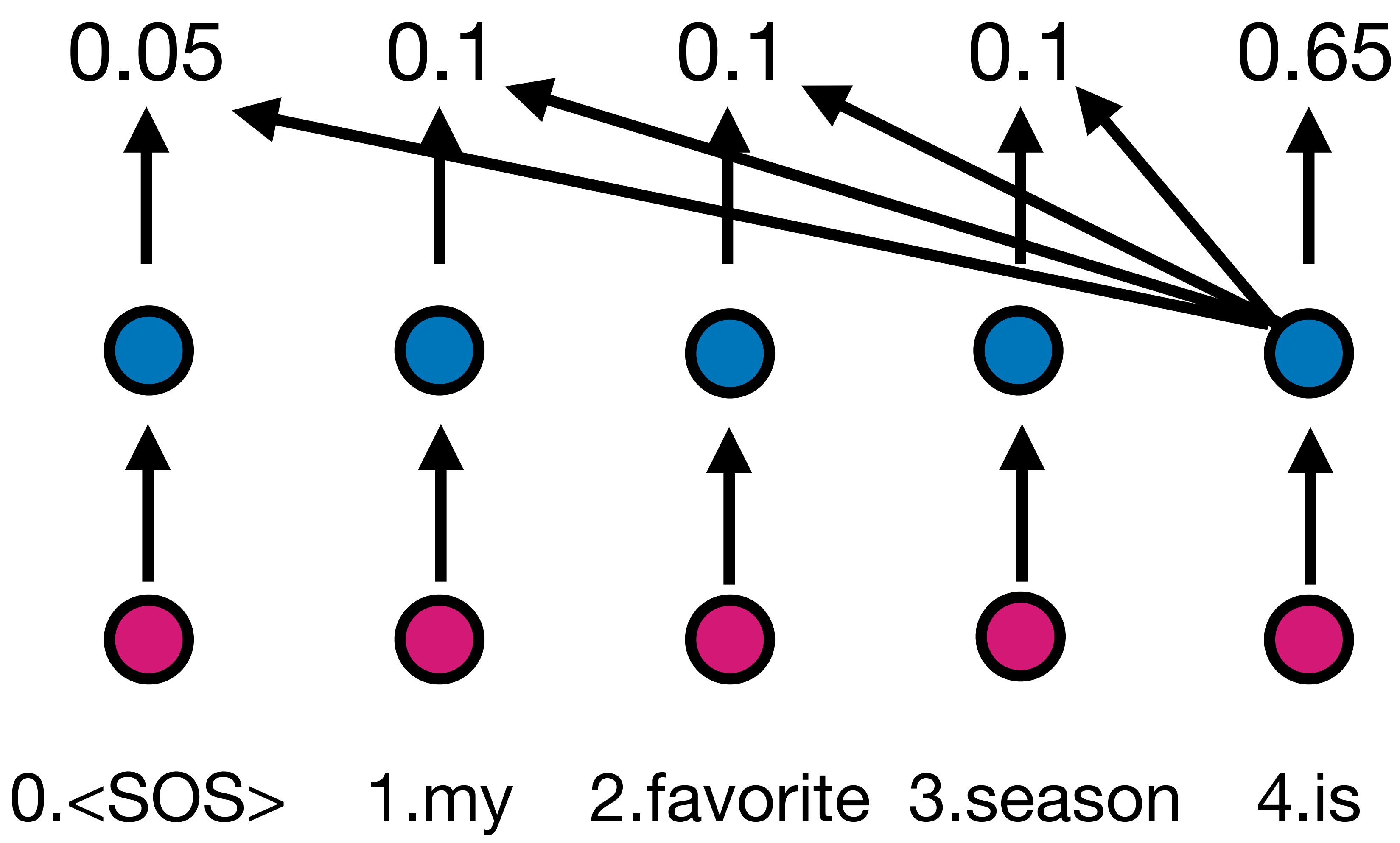
Simplified transformer for language modeling

Input text: “my favorite season is” Output word: “spring”

1. *Positionally encode* each input token and pass it through FC layer to compute *values.*

2. Calculate attention weights for “is” by comparing its value to its own and those of preceding tokens (*self-attention*).

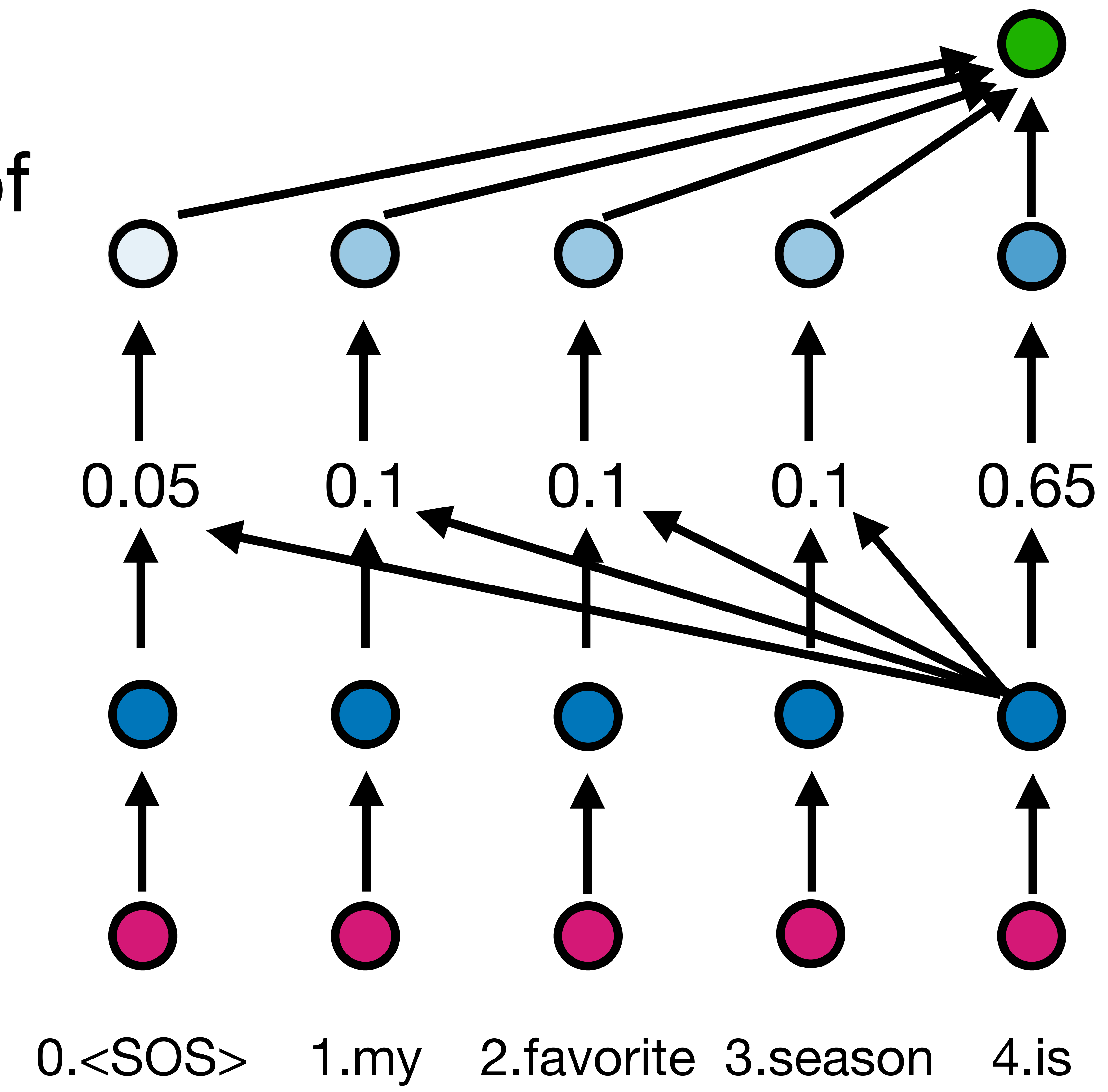
3. Take weighted average of preceding token values to get *context vector* or *contextual embedding* for “is”.



Simplified transformer for language modeling

Input text: “my favorite season is” Output word: “spring”

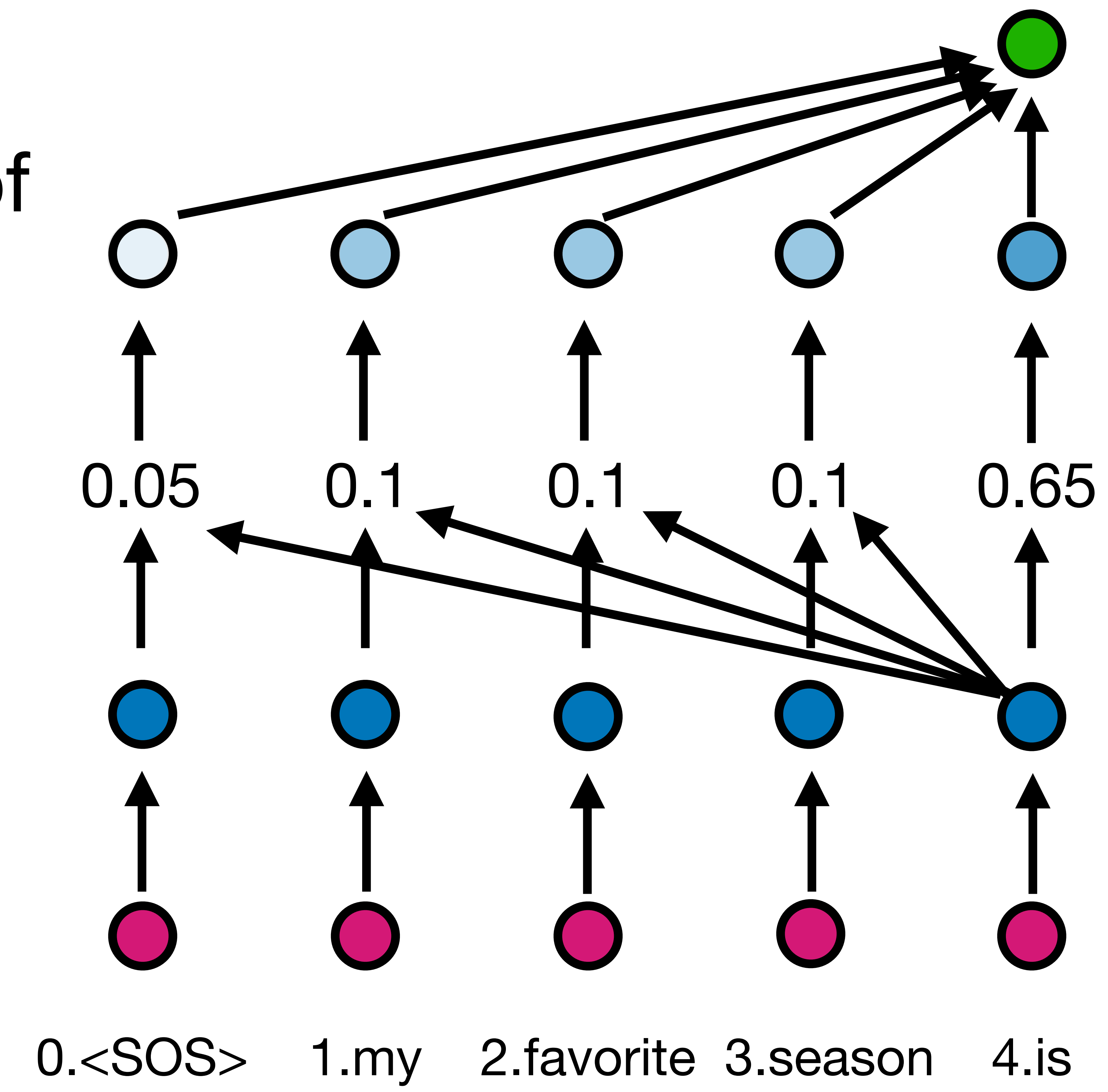
1. *Positionally encode* each input token and pass it through FC layer to compute *values.*
2. Calculate attention weights for “is” by comparing its value to its own and those of preceding tokens (*self-attention*).
3. Take weighted average of preceding token values to get *context vector* or *contextual embedding* for “is”.



Simplified transformer for language modeling

Input text: “my favorite season is” Output word: “spring”

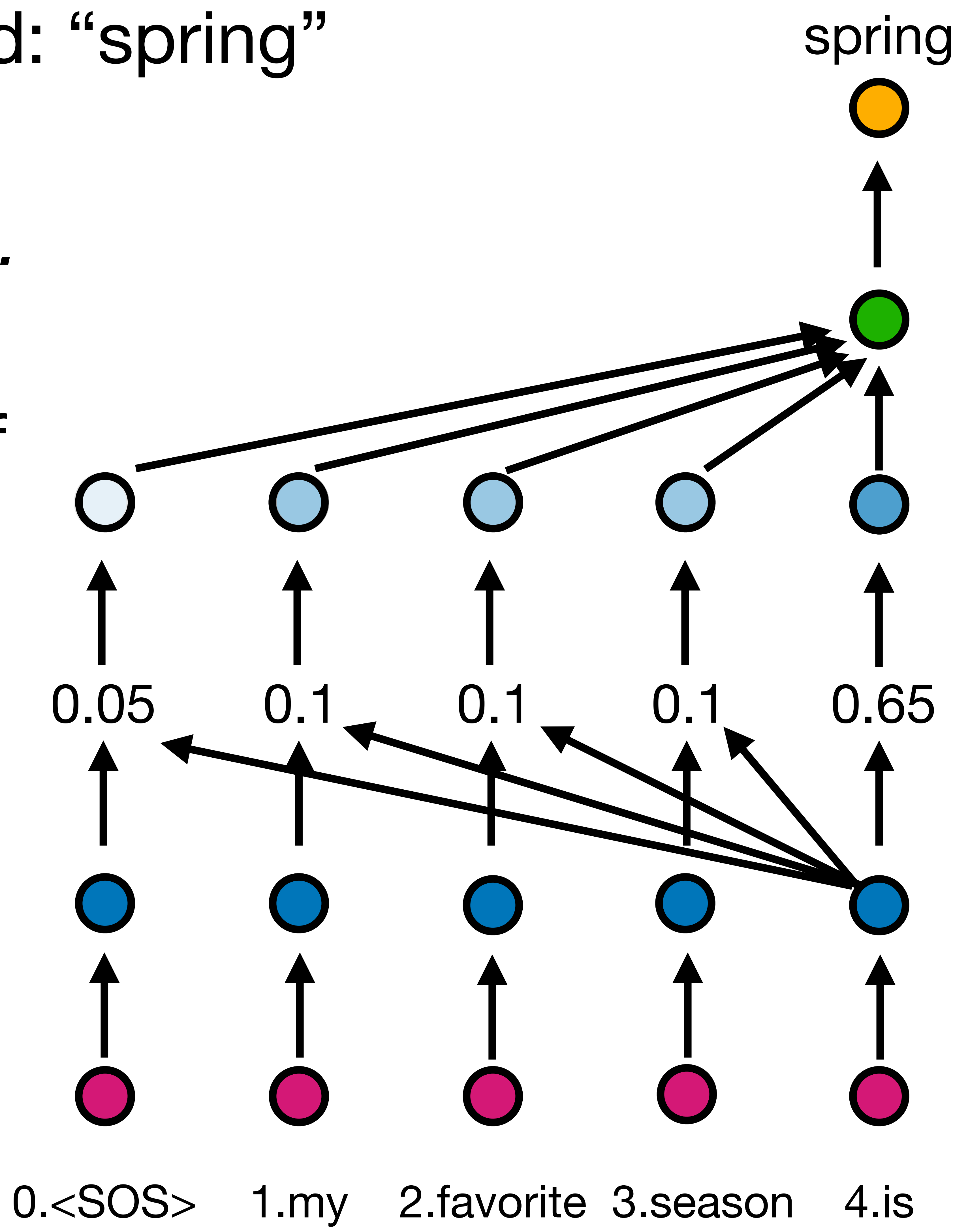
1. *Positionally encode* each input token and pass it through FC layer to compute *values.*
2. Calculate attention weights for “is” by comparing its value to its own and those of preceding tokens (*self-attention*).
3. Take weighted average of preceding token values to get *context vector* or *contextual embedding* for “is”.
4. Pass contextual embedding through FC layer with softmax to get output class probabilities.



Simplified transformer for language modeling

Input text: “my favorite season is” Output word: “spring”

1. *Positionally encode* each input token and pass it through FC layer to compute *values.*
2. Calculate attention weights for “is” by comparing its value to its own and those of preceding tokens (*self-attention*).
3. Take weighted average of preceding token values to get *context vector* or *contextual embedding* for “is”.
4. Pass contextual embedding through FC layer with softmax to get output class probabilities.

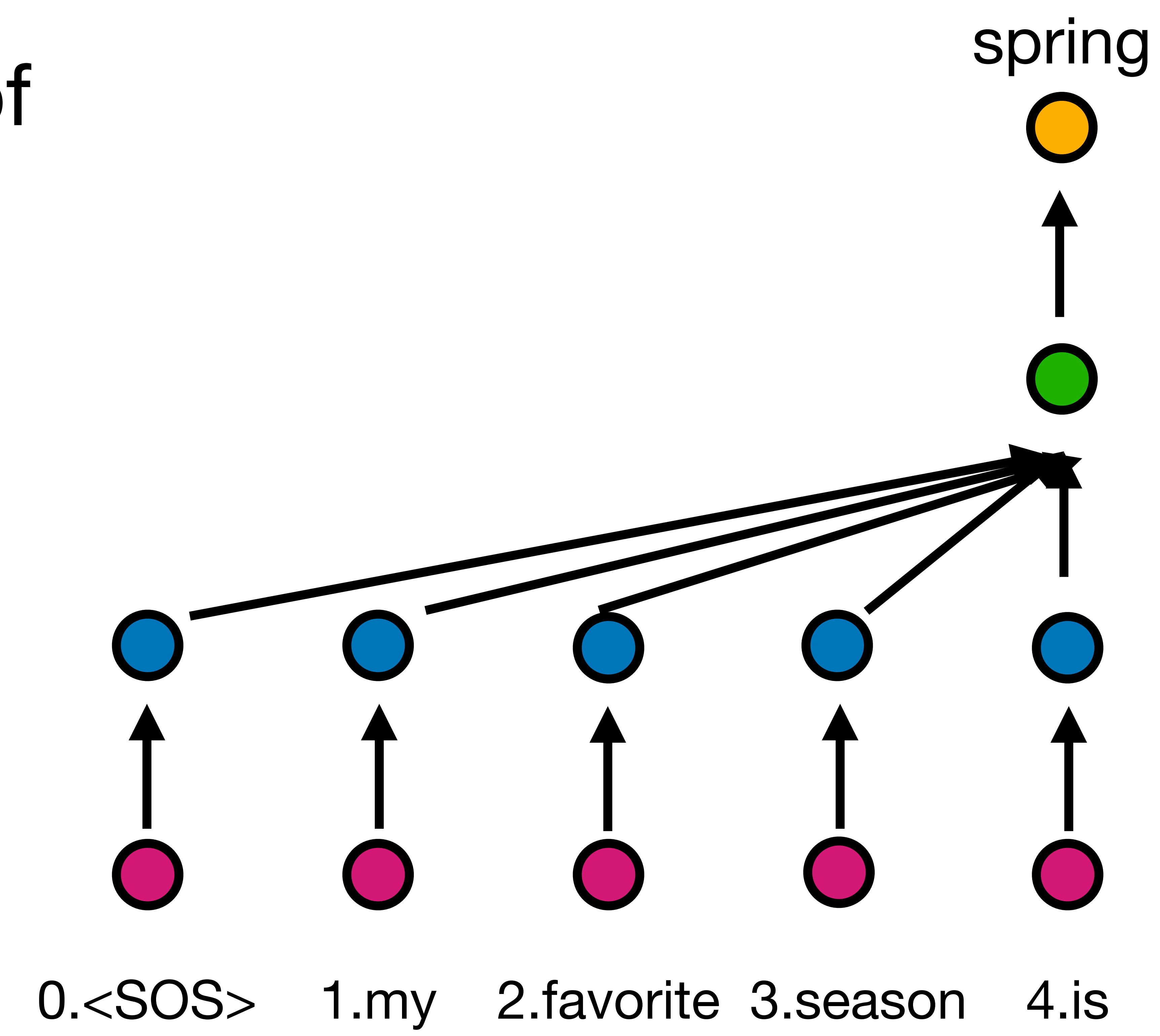


Simplified transformer for language modeling

Input text: “my favorite season is” Output word: “spring”

1. *Positionally encode* each input token and pass it through FC layer to compute *values.*
2. Calculate attention weights for “is” by comparing its value to its own and those of preceding tokens (*self-attention*).
3. Take weighted average of preceding token values to get *context vector* or *contextual embedding* for “is”.
4. Pass contextual embedding through FC layer with softmax to get output class probabilities.

Simplified flowchart of simplified transformer



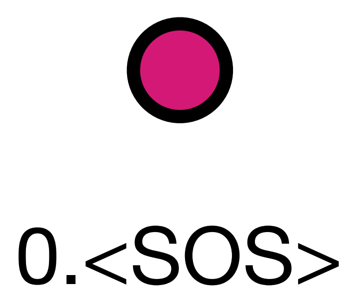
Simplified transformer for autoregressive text generation

Simplified transformer for autoregressive text generation

1. Positionally encode <SOS> token.

Simplified transformer for autoregressive text generation

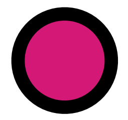
1. Positionally encode <SOS> token.



Simplified transformer for autoregressive text generation

1. Positionally encode <SOS> token.

2. Calculate <SOS> token’s *value* by passing token through fully connected layer.

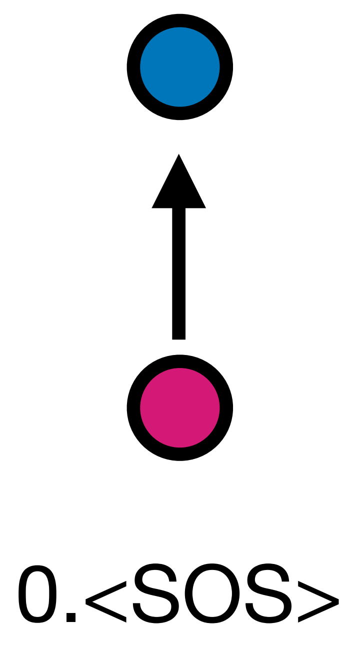


0.<SOS>

Simplified transformer for autoregressive text generation

1. Positionally encode <SOS> token.

2. Calculate <SOS> token's *value* by passing token through fully connected layer.

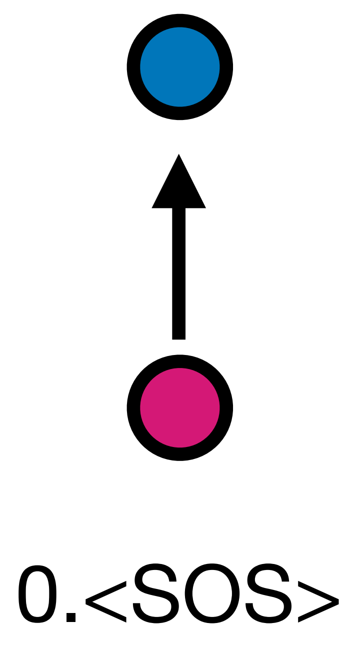


Simplified transformer for autoregressive text generation

1. Positionally encode <SOS> token.

2. Calculate <SOS> token's *value* by passing token through fully connected layer.

3. Calculate <SOS> token's contextual embedding by attention-weighted linear combination of its value and all preceding values (so far, no preceding values).

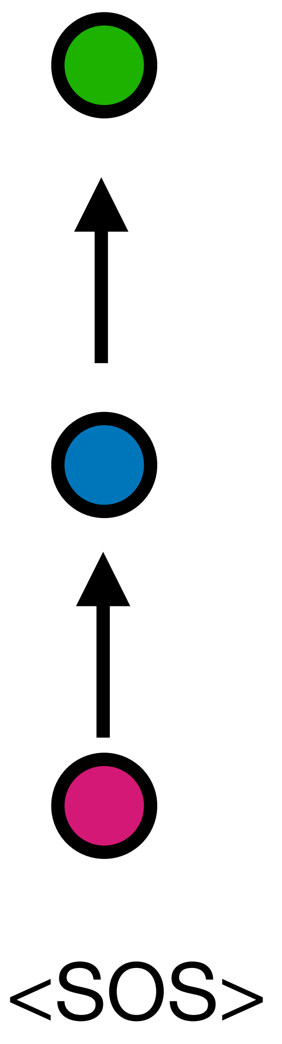


Simplified transformer for autoregressive text generation

1. Positionally encode <SOS> token.

2. Calculate <SOS> token's *value* by passing token through fully connected layer.

3. Calculate <SOS> token's contextual embedding by attention-weighted linear combination of its value and all preceding values (so far, no preceding values).



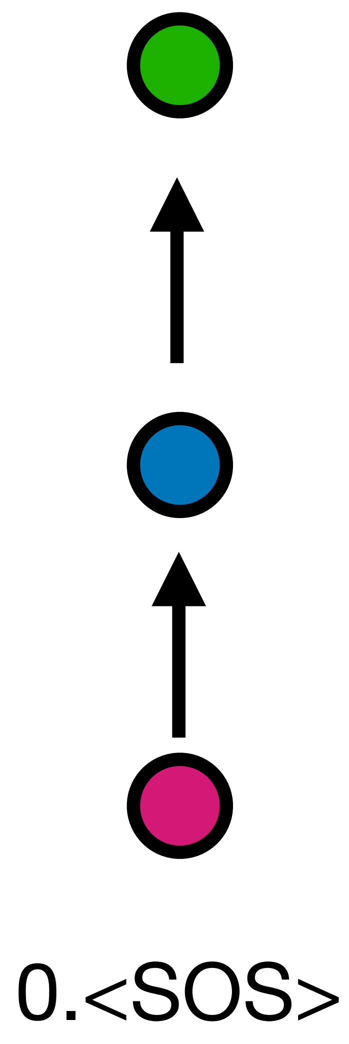
Simplified transformer for autoregressive text generation

1. Positionally encode <SOS> token.

2. Calculate <SOS> token's *value* by passing token through fully connected layer.

3. Calculate <SOS> token's contextual embedding by attention-weighted linear combination of its value and all preceding values (so far, no preceding values).

4. Generate next token by passing contextual embedding through FC layer + softmax.



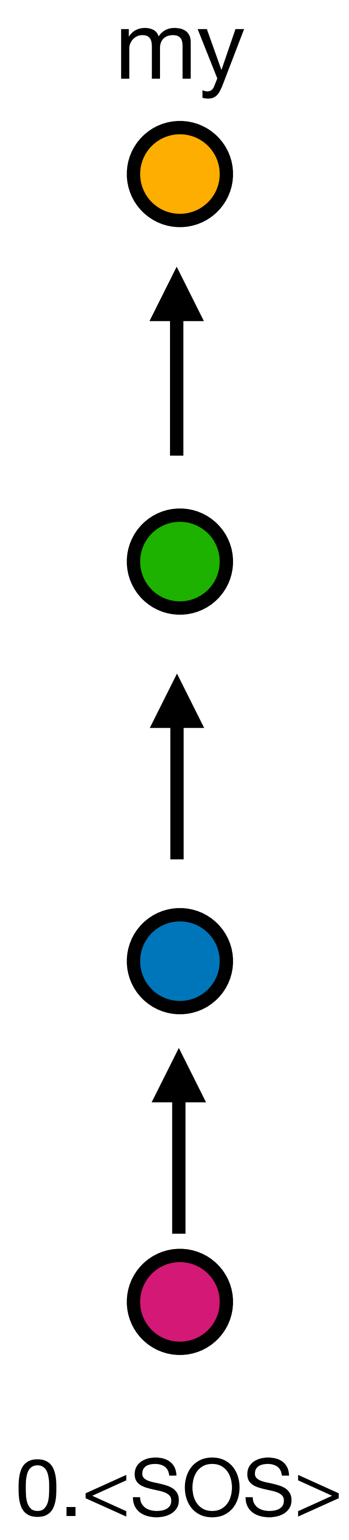
Simplified transformer for autoregressive text generation

1. Positionally encode <SOS> token.

2. Calculate <SOS> token’s *value* by passing token through fully connected layer.

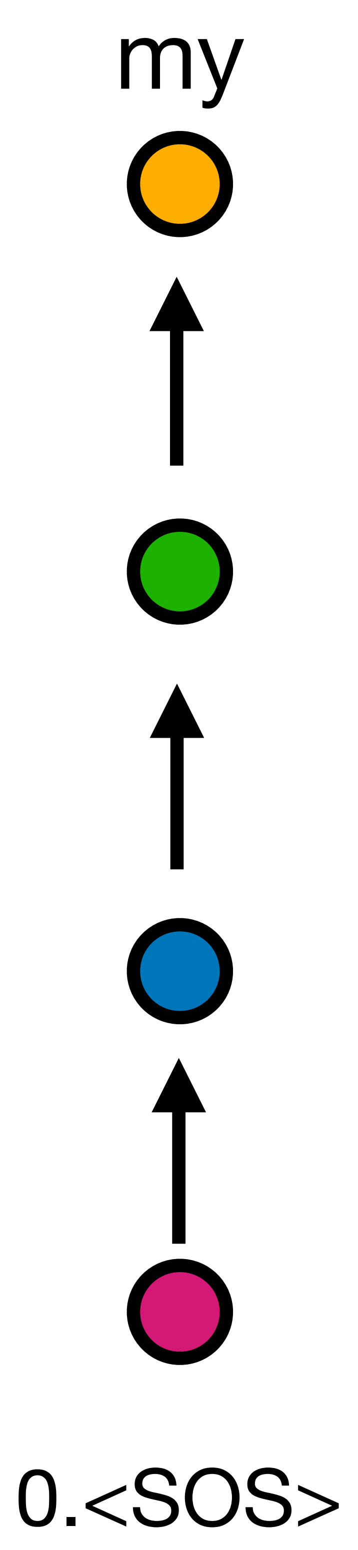
3. Calculate <SOS> token’s contextual embedding by attention-weighted linear combination of its value and all preceding values (so far, no preceding values).

4. Generate next token by passing contextual embedding through FC layer + softmax.



Simplified transformer for autoregressive text generation

1. Positionally encode <SOS> token.
2. Calculate <SOS> token's *value* by passing token through fully connected layer.
3. Calculate <SOS> token's contextual embedding by attention-weighted linear combination of its value and all preceding values (so far, no preceding values).
4. Generate next token by passing contextual embedding through FC layer + softmax.
5. Copy generated token to the input.



Simplified transformer for autoregressive text generation

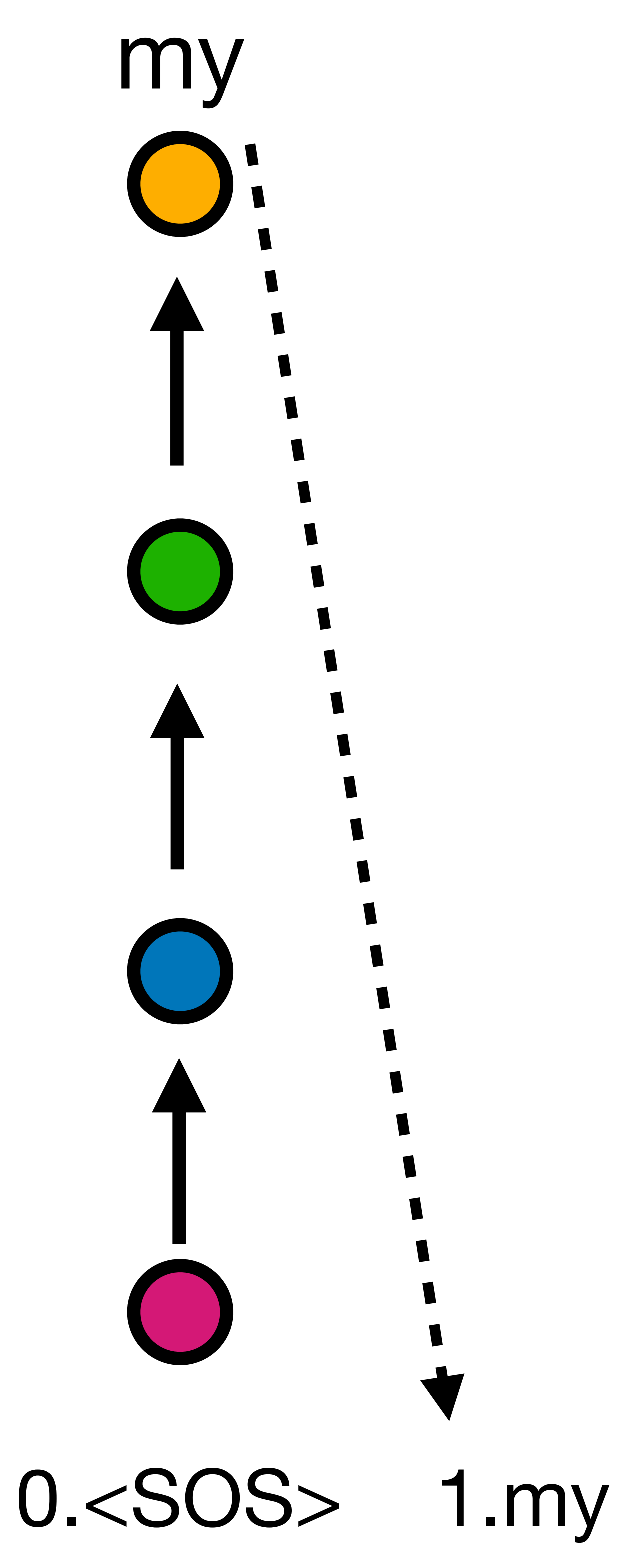
1. Positionally encode <SOS> token.

2. Calculate <SOS> token's *value* by passing token through fully connected layer.

3. Calculate <SOS> token's contextual embedding by attention-weighted linear combination of its value and all preceding values (so far, no preceding values).

4. Generate next token by passing contextual embedding through FC layer + softmax.

5. Copy generated token to the input.



Simplified transformer for autoregressive text generation

1. Positionally encode <SOS> token.

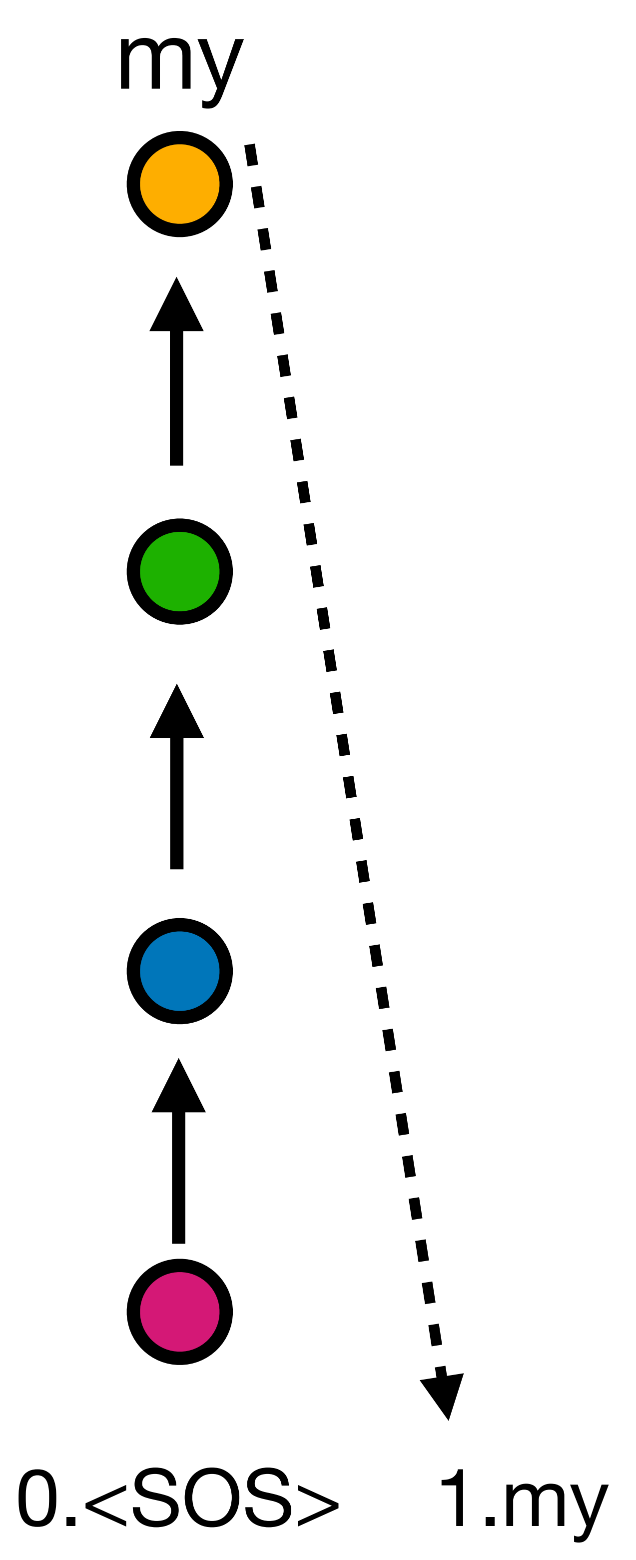
2. Calculate <SOS> token's *value* by passing token through fully connected layer.

3. Calculate <SOS> token's contextual embedding by attention-weighted linear combination of its value and all preceding values (so far, no preceding values).

4. Generate next token by passing contextual embedding through FC layer + softmax.

5. Copy generated token to the input.

6. Repeat until <EOS> token is generated.



Simplified transformer for autoregressive text generation

1. Positionally encode <SOS> token.

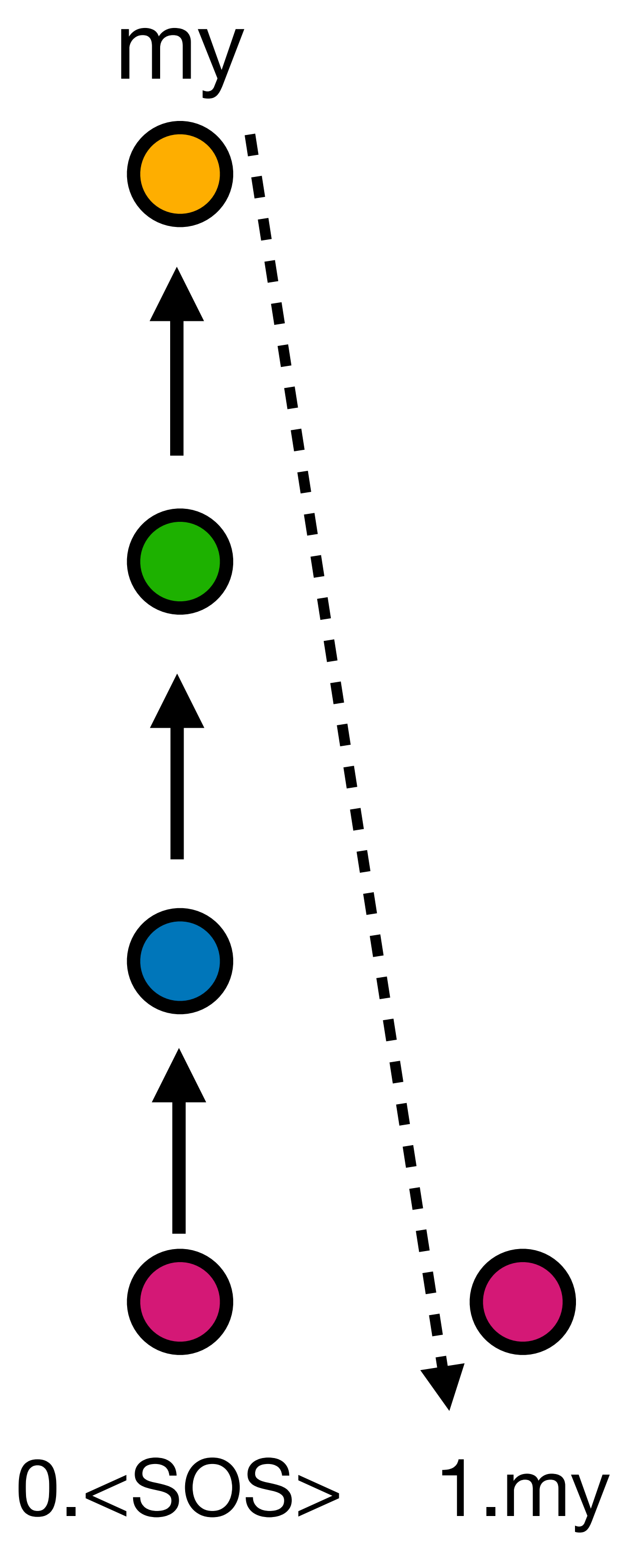
2. Calculate <SOS> token's *value* by passing token through fully connected layer.

3. Calculate <SOS> token's contextual embedding by attention-weighted linear combination of its value and all preceding values (so far, no preceding values).

4. Generate next token by passing contextual embedding through FC layer + softmax.

5. Copy generated token to the input.

6. Repeat until <EOS> token is generated.



Simplified transformer for autoregressive text generation

1. Positionally encode <SOS> token.

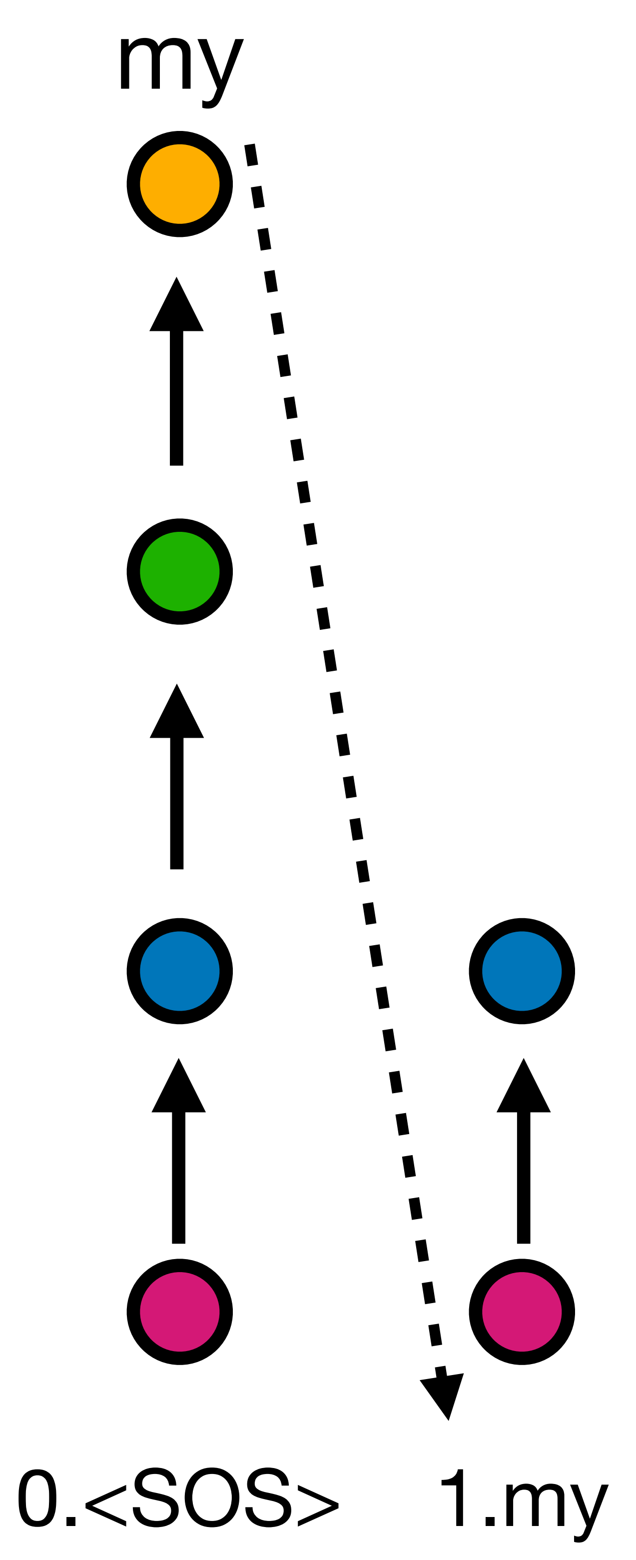
2. Calculate <SOS> token's *value* by passing token through fully connected layer.

3. Calculate <SOS> token's contextual embedding by attention-weighted linear combination of its value and all preceding values (so far, no preceding values).

4. Generate next token by passing contextual embedding through FC layer + softmax.

5. Copy generated token to the input.

6. Repeat until <EOS> token is generated.



Simplified transformer for autoregressive text generation

1. Positionally encode <SOS> token.

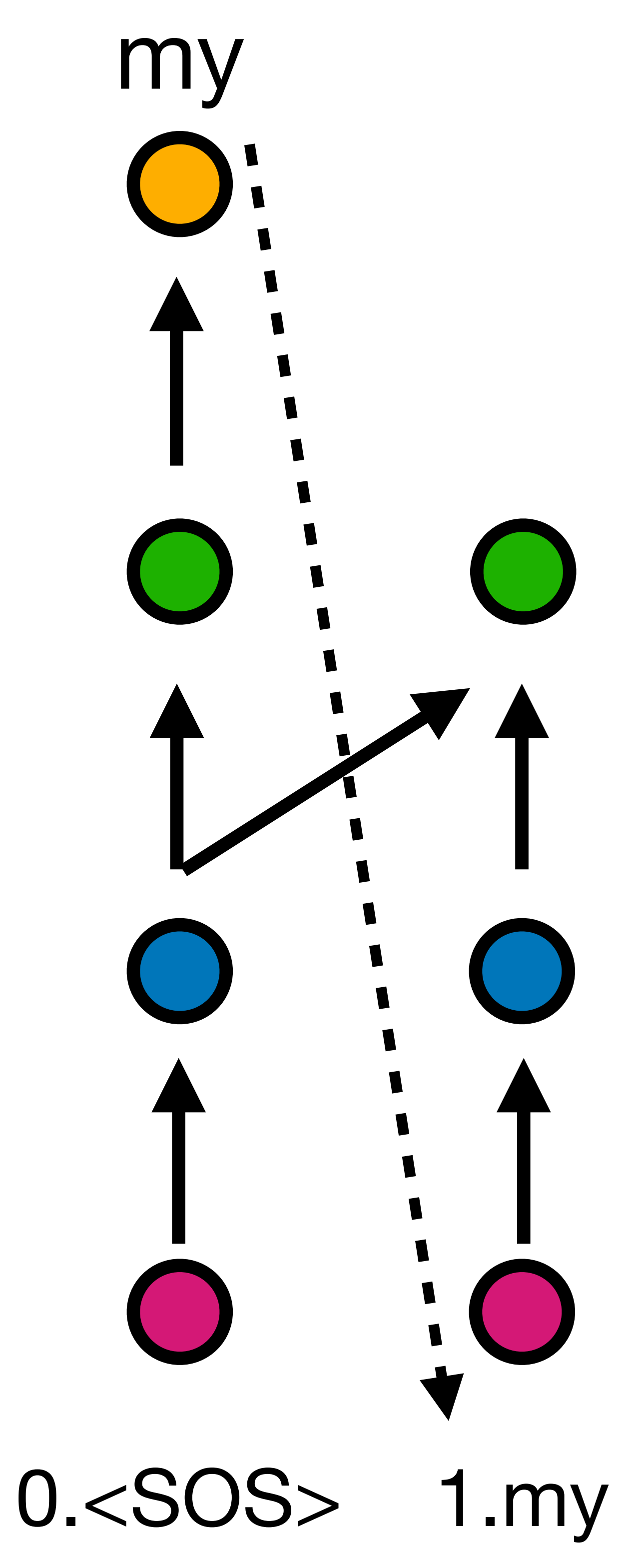
2. Calculate <SOS> token's *value* by passing token through fully connected layer.

3. Calculate <SOS> token's contextual embedding by attention-weighted linear combination of its value and all preceding values (so far, no preceding values).

4. Generate next token by passing contextual embedding through FC layer + softmax.

5. Copy generated token to the input.

6. Repeat until <EOS> token is generated.



Simplified transformer for autoregressive text generation

1. Positionally encode <SOS> token.

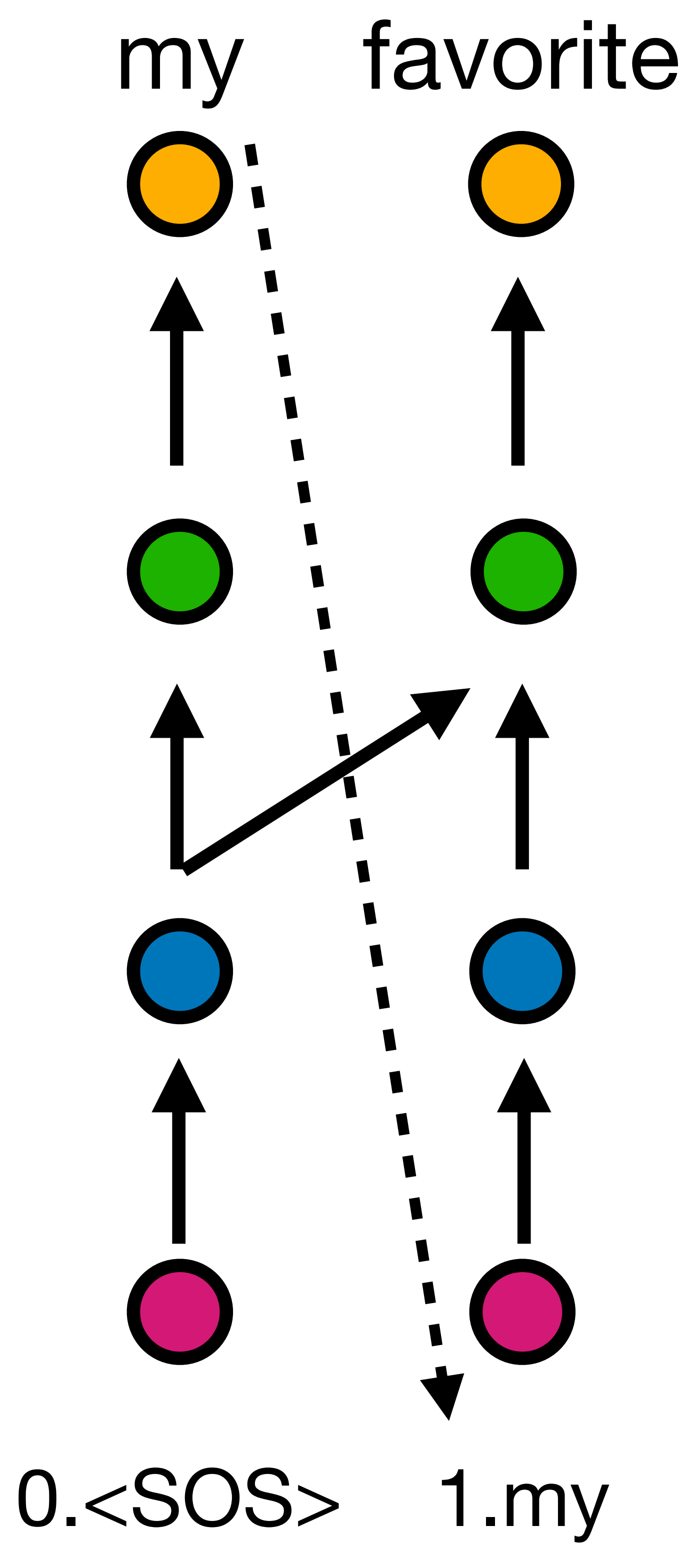
2. Calculate <SOS> token's *value* by passing token through fully connected layer.

3. Calculate <SOS> token's contextual embedding by attention-weighted linear combination of its value and all preceding values (so far, no preceding values).

4. Generate next token by passing contextual embedding through FC layer + softmax.

5. Copy generated token to the input.

6. Repeat until <EOS> token is generated.



Simplified transformer for autoregressive text generation

1. Positionally encode <SOS> token.

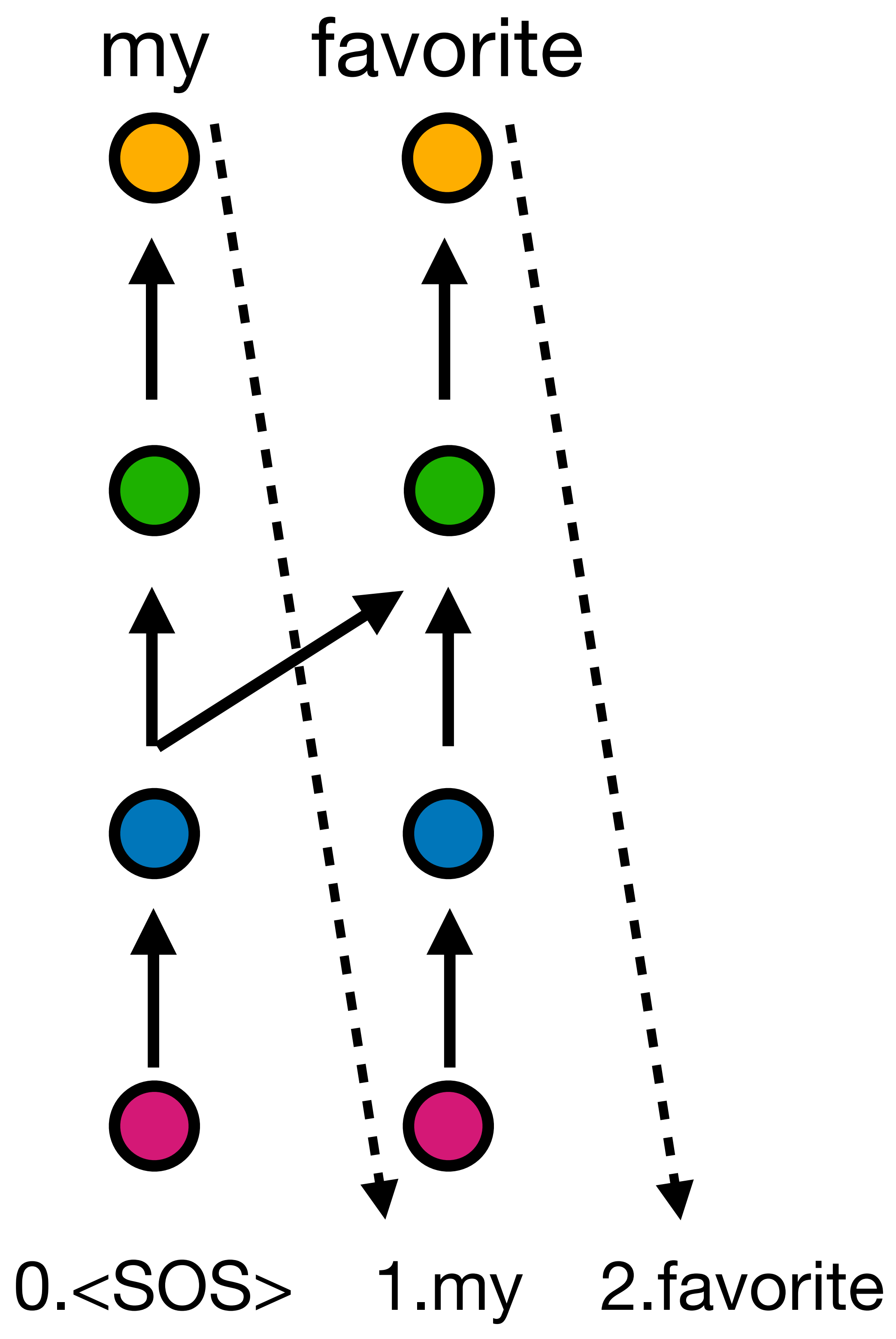
2. Calculate <SOS> token's *value* by passing token through fully connected layer.

3. Calculate <SOS> token's contextual embedding by attention-weighted linear combination of its value and all preceding values (so far, no preceding values).

4. Generate next token by passing contextual embedding through FC layer + softmax.

5. Copy generated token to the input.

6. Repeat until <EOS> token is generated.



Simplified transformer for autoregressive text generation

1. Positionally encode <SOS> token.

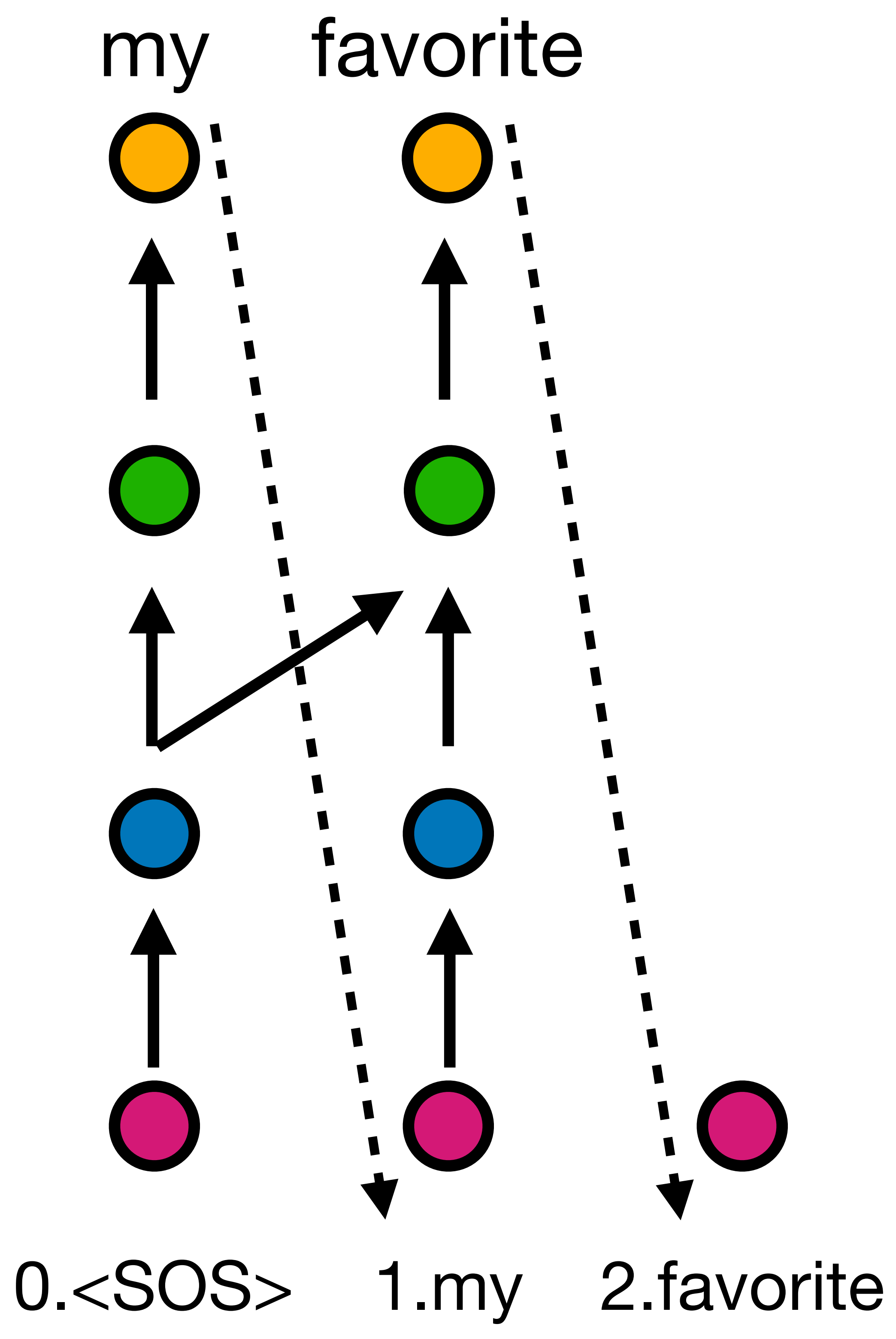
2. Calculate <SOS> token's *value* by passing token through fully connected layer.

3. Calculate <SOS> token's contextual embedding by attention-weighted linear combination of its value and all preceding values (so far, no preceding values).

4. Generate next token by passing contextual embedding through FC layer + softmax.

5. Copy generated token to the input.

6. Repeat until <EOS> token is generated.



Simplified transformer for autoregressive text generation

1. Positionally encode <SOS> token.

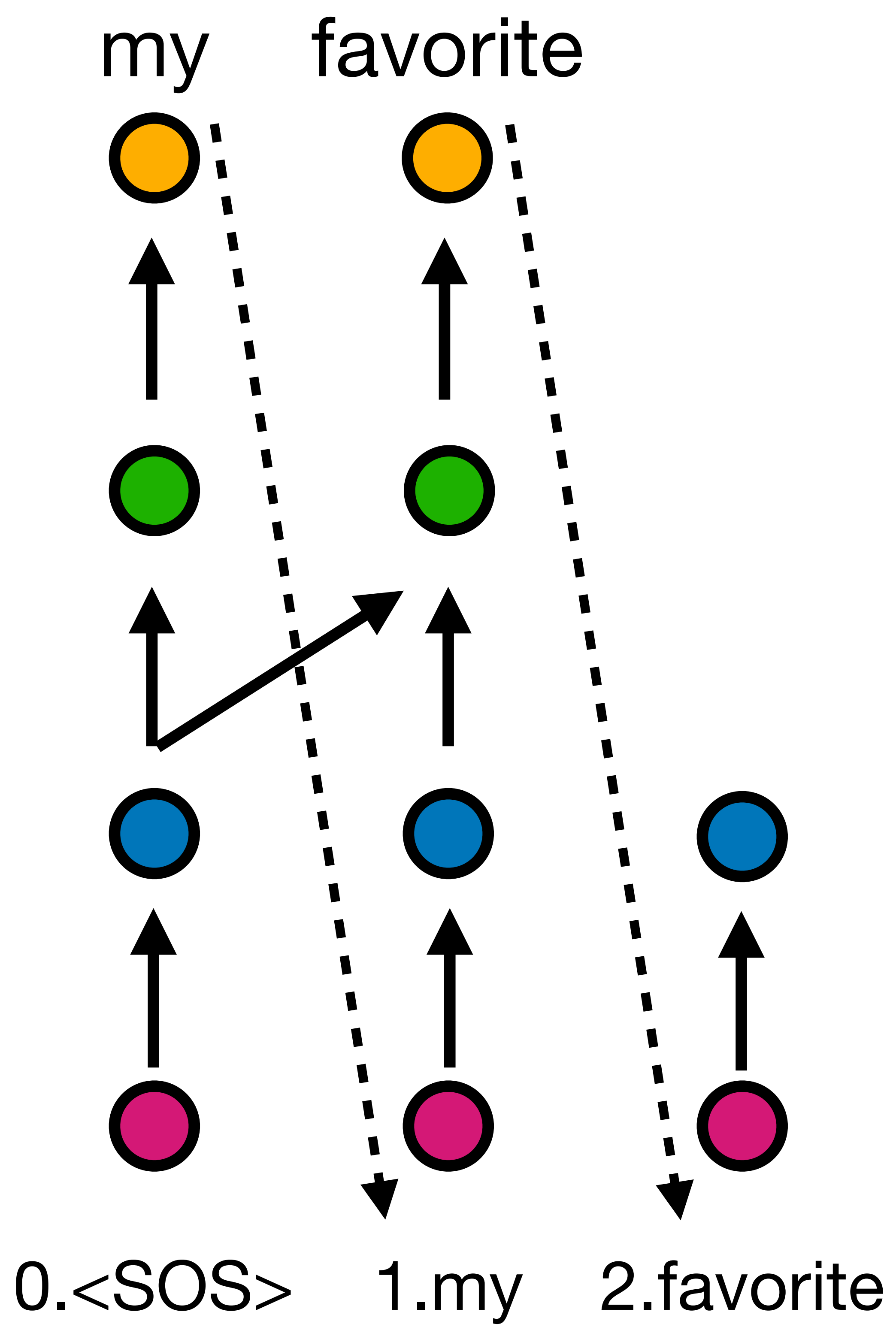
2. Calculate <SOS> token’s *value* by passing token through fully connected layer.

3. Calculate <SOS> token’s contextual embedding by attention-weighted linear combination of its value and all preceding values (so far, no preceding values).

4. Generate next token by passing contextual embedding through FC layer + softmax.

5. Copy generated token to the input.

6. Repeat until <EOS> token is generated.



Simplified transformer for autoregressive text generation

1. Positionally encode <SOS> token.

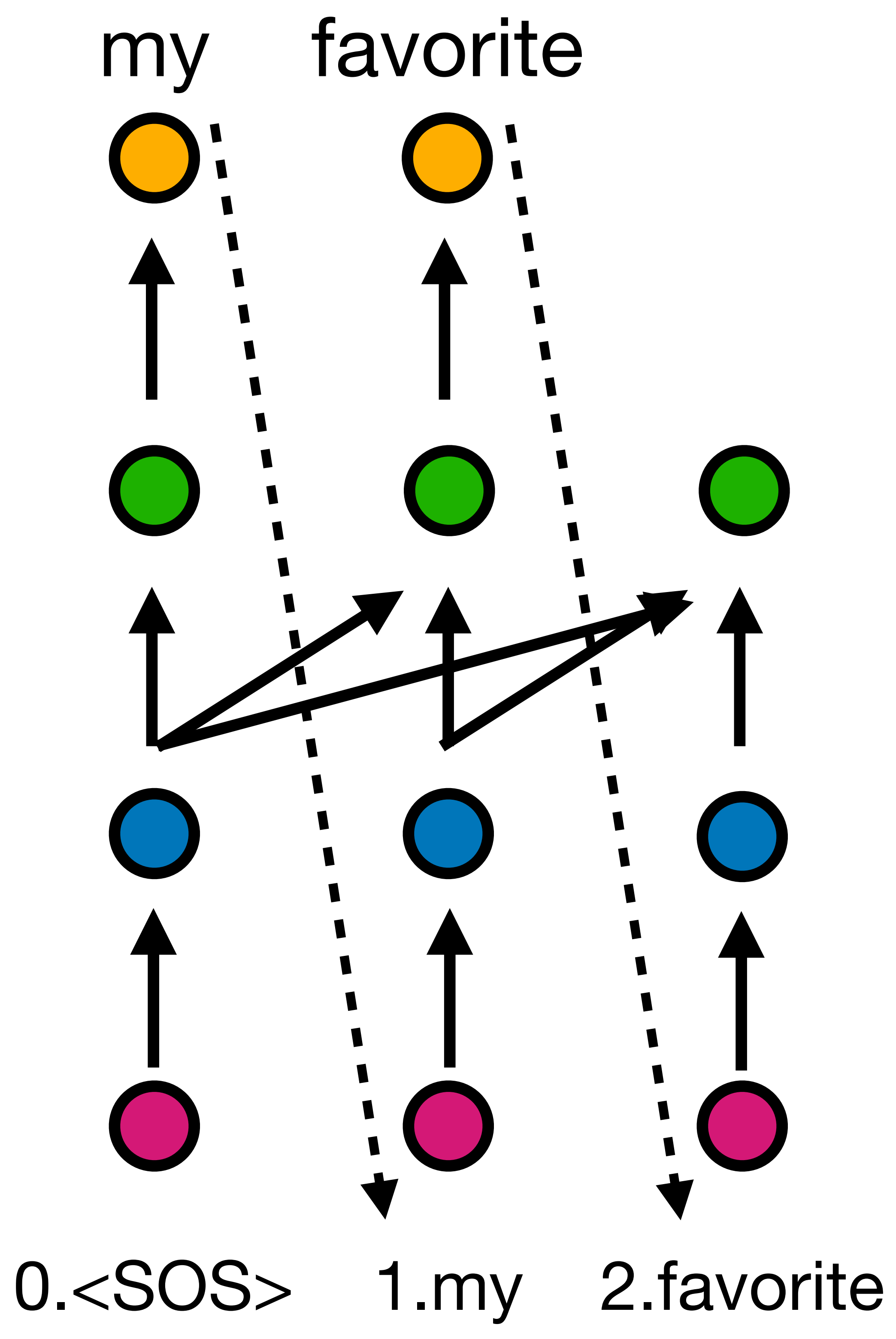
2. Calculate <SOS> token's *value* by passing token through fully connected layer.

3. Calculate <SOS> token's contextual embedding by attention-weighted linear combination of its value and all preceding values (so far, no preceding values).

4. Generate next token by passing contextual embedding through FC layer + softmax.

5. Copy generated token to the input.

6. Repeat until <EOS> token is generated.



Simplified transformer for autoregressive text generation

1. Positionally encode <SOS> token.

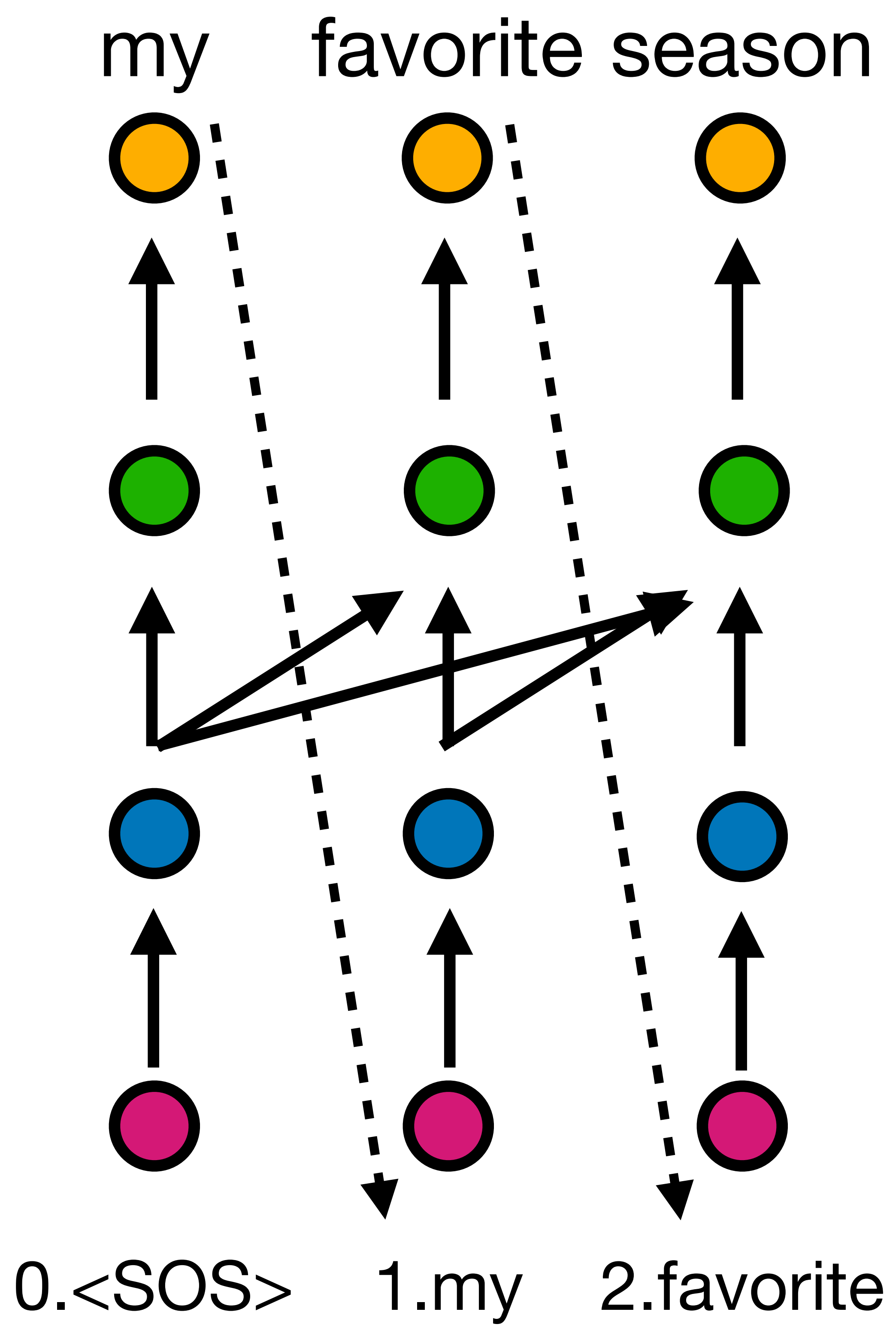
2. Calculate <SOS> token's *value* by passing token through fully connected layer.

3. Calculate <SOS> token's contextual embedding by attention-weighted linear combination of its value and all preceding values (so far, no preceding values).

4. Generate next token by passing contextual embedding through FC layer + softmax.

5. Copy generated token to the input.

6. Repeat until <EOS> token is generated.



Simplified transformer for autoregressive text generation

1. Positionally encode <SOS> token.

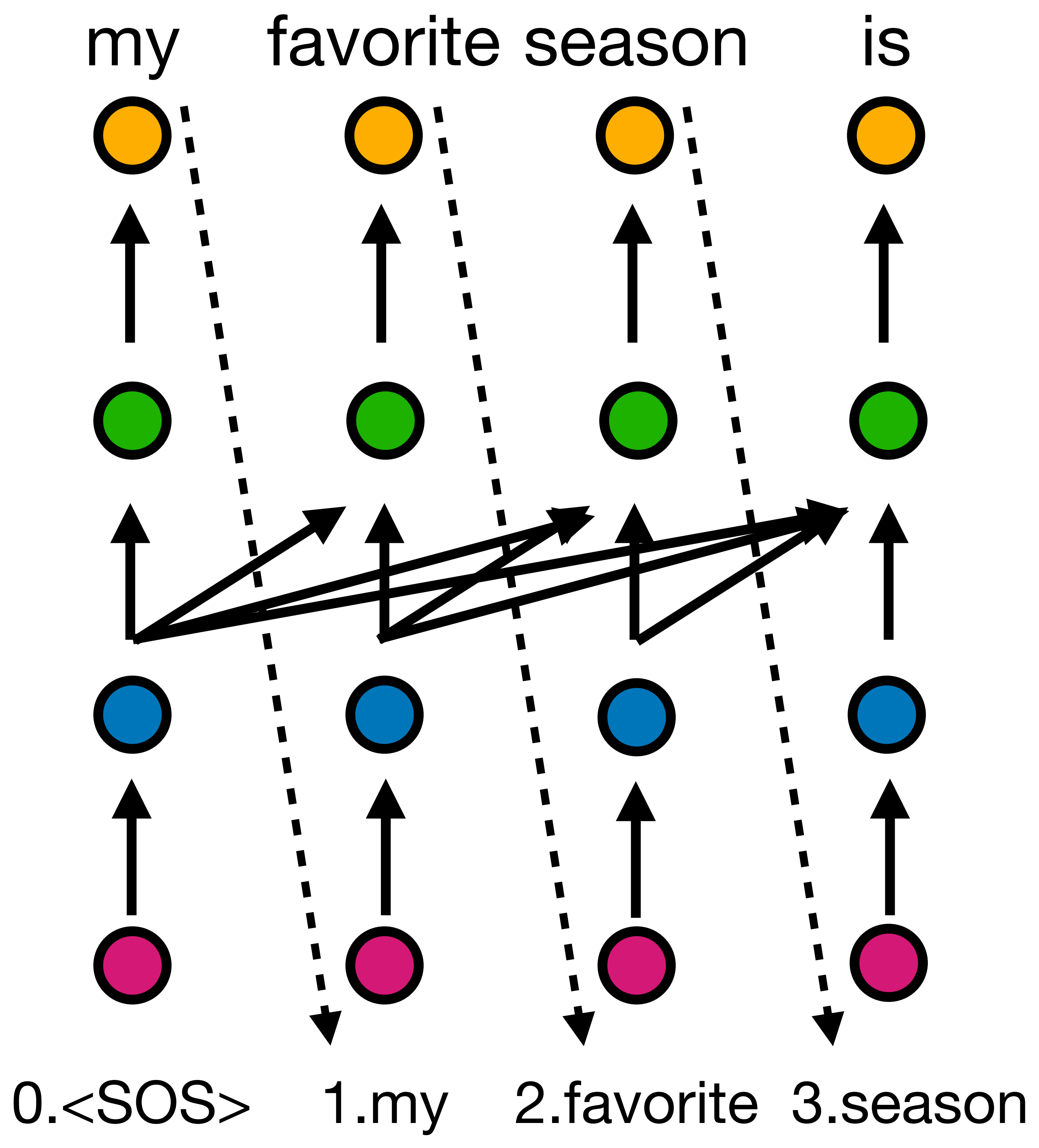
2. Calculate <SOS> token's *value* by passing token through fully connected layer.

3. Calculate <SOS> token's contextual embedding by attention-weighted linear combination of its value and all preceding values (so far, no preceding values).

4. Generate next token by passing contextual embedding through FC layer + softmax.

5. Copy generated token to the input.

6. Repeat until <EOS> token is generated.



Simplified transformer for autoregressive text generation

1. Positionally encode <SOS> token.

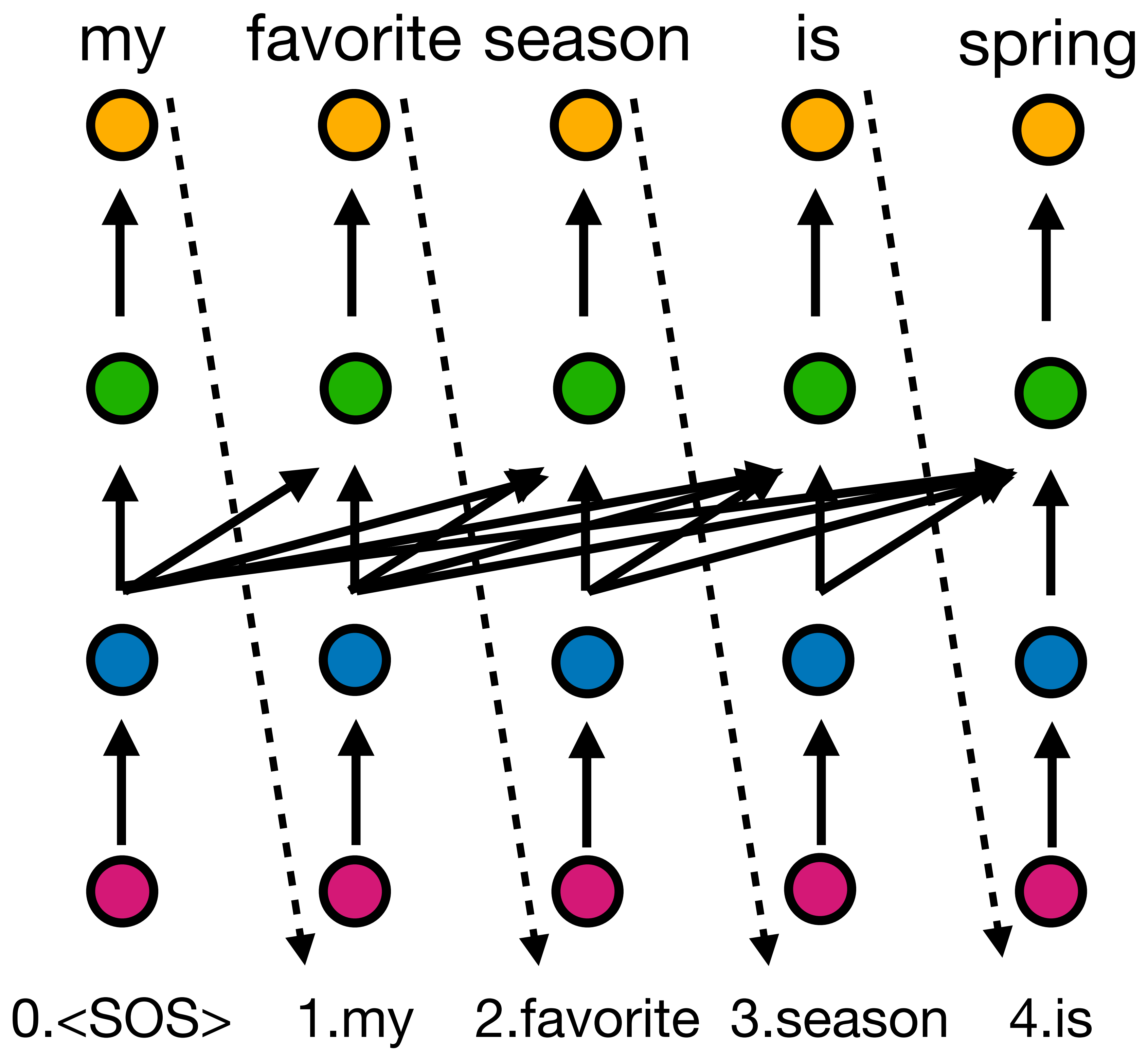
2. Calculate <SOS> token's *value* by passing token through fully connected layer.

3. Calculate <SOS> token's contextual embedding by attention-weighted linear combination of its value and all preceding values (so far, no preceding values).

4. Generate next token by passing contextual embedding through FC layer + softmax.

5. Copy generated token to the input.

6. Repeat until <EOS> token is generated.



Simplified transformer for machine translation

Simplified transformer for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

Simplified transformer for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Start a sequence based on input text and append <SOS>.

Simplified transformer for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Start a sequence based on input text and append <SOS>.

Simplified transformer for machine translation

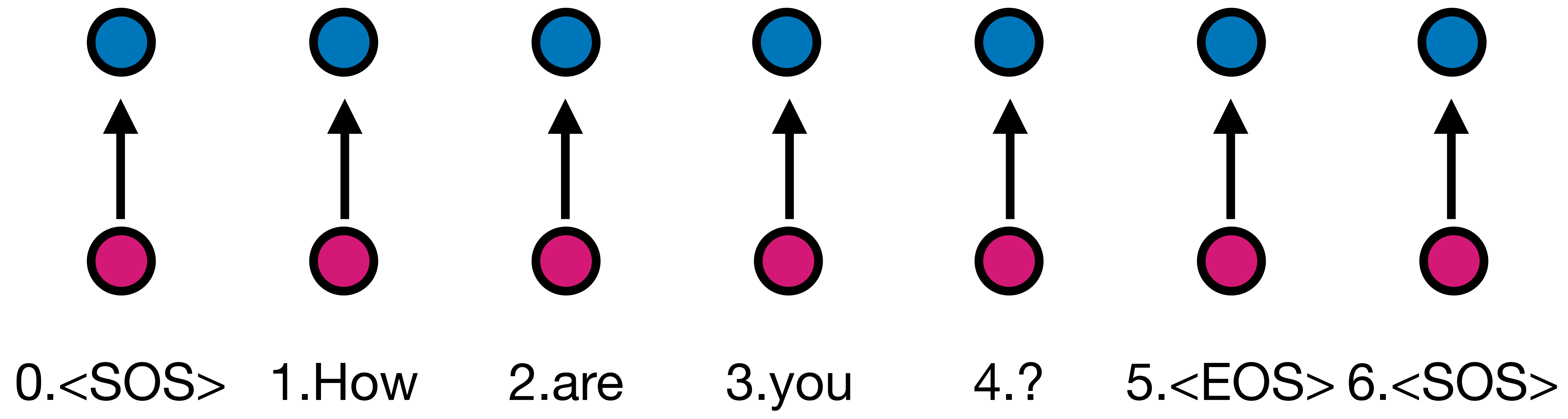
Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Start a sequence based on input text and append <SOS>.
2. Generate text autoregressively until <EOS> token is reached.

Simplified transformer for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

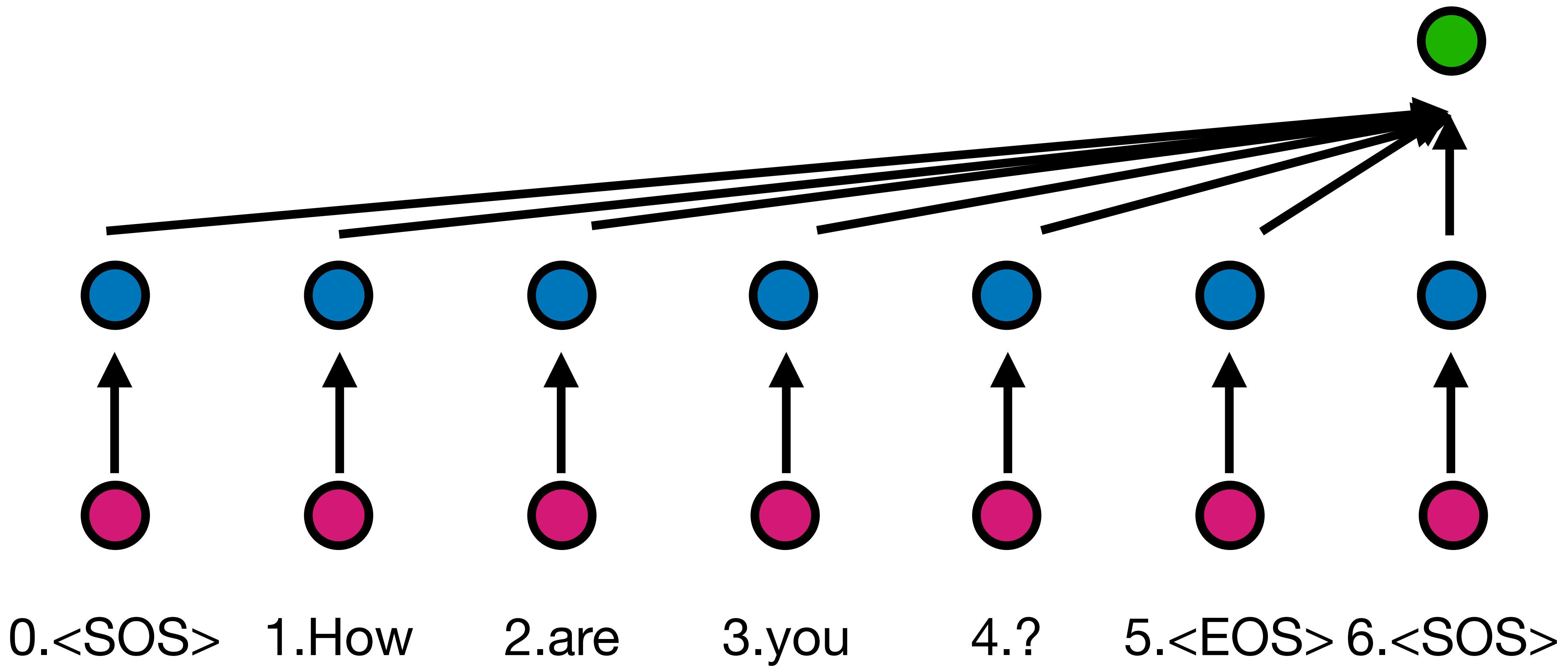
1. Start a sequence based on input text and append <SOS>.
2. Generate text autoregressively until <EOS> token is reached.



Simplified transformer for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

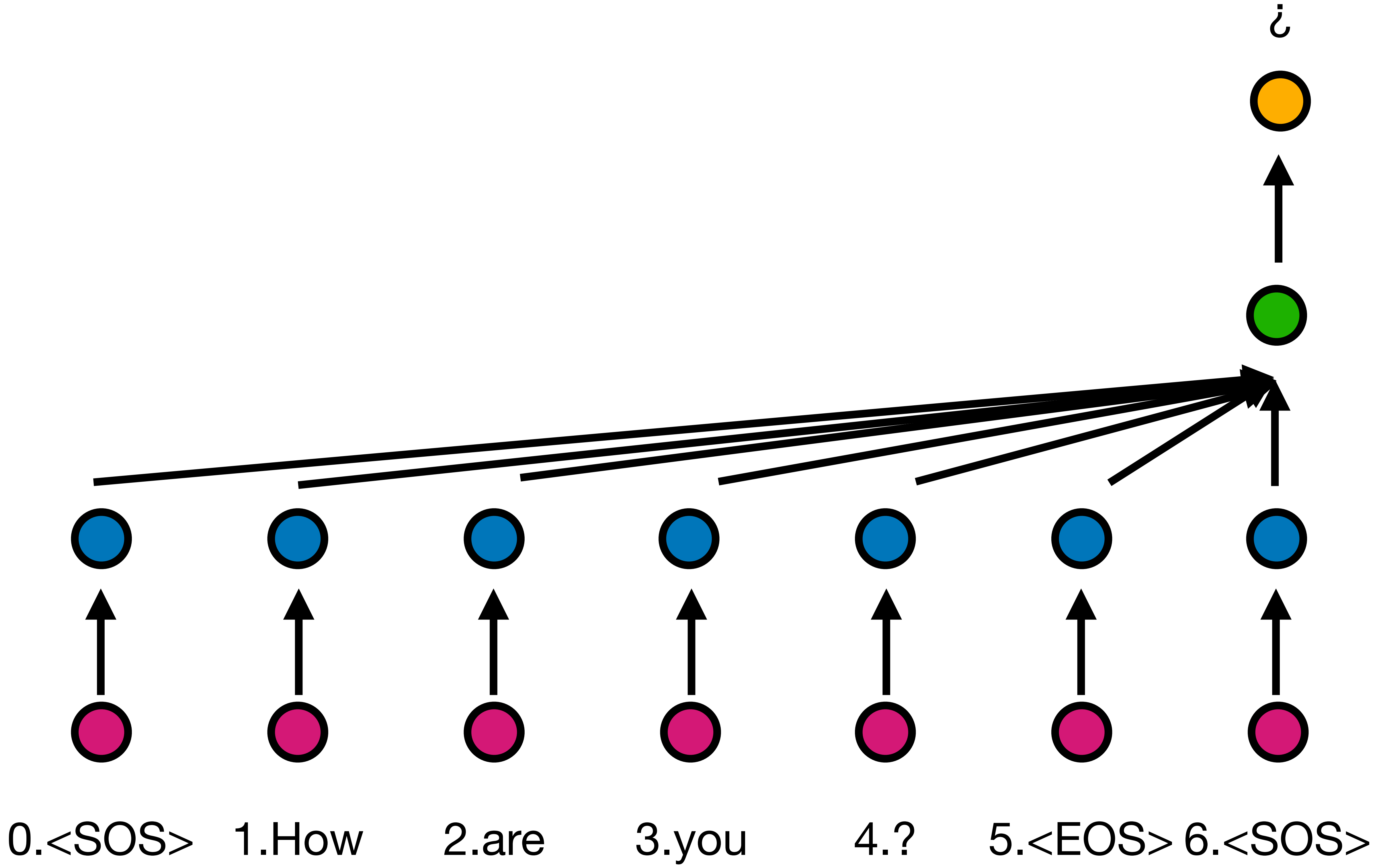
1. Start a sequence based on input text and append <SOS>.
2. Generate text autoregressively until <EOS> token is reached.



Simplified transformer for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

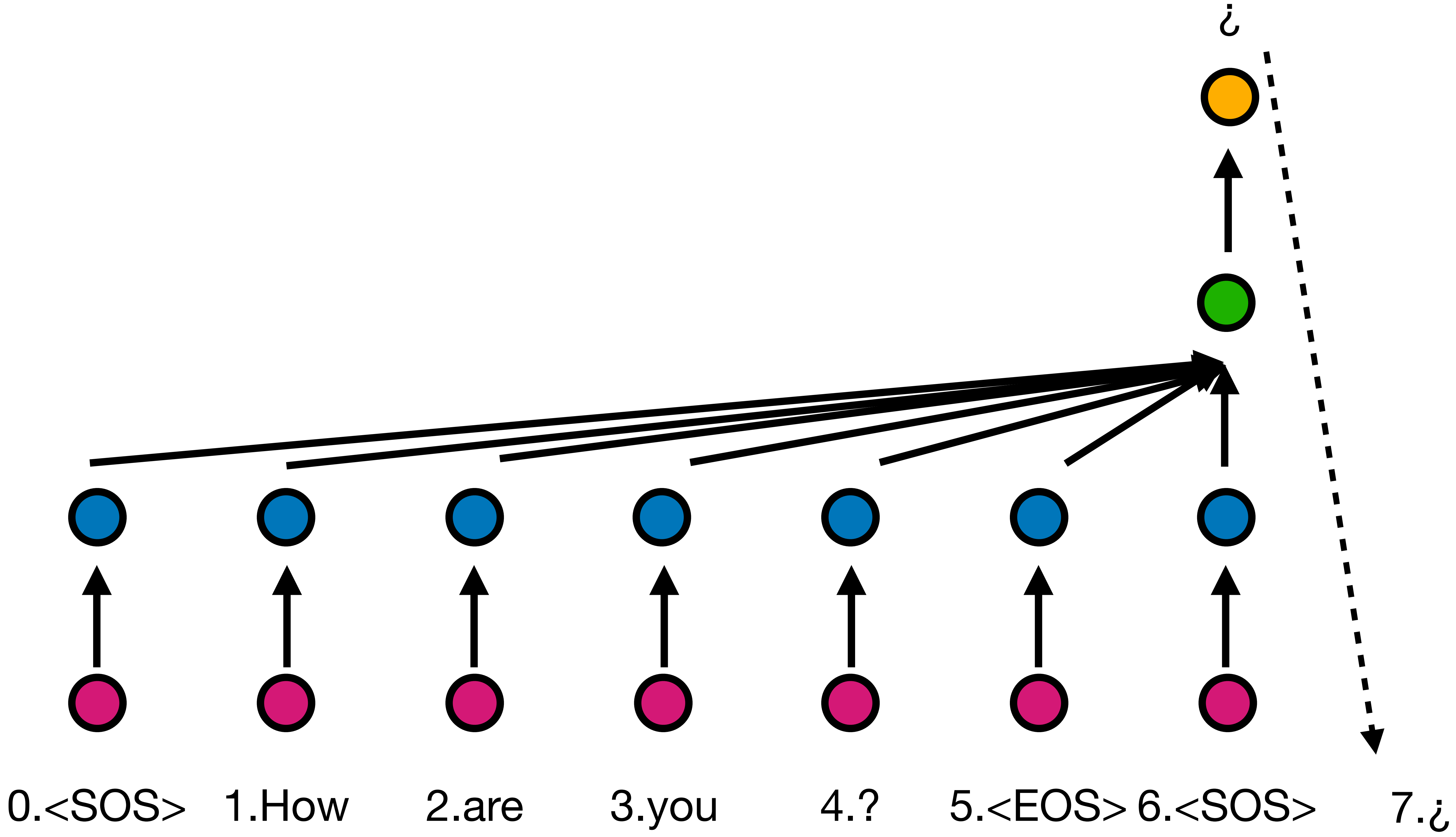
1. Start a sequence based on input text and append <SOS>.
2. Generate text autoregressively until <EOS> token is reached.



Simplified transformer for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

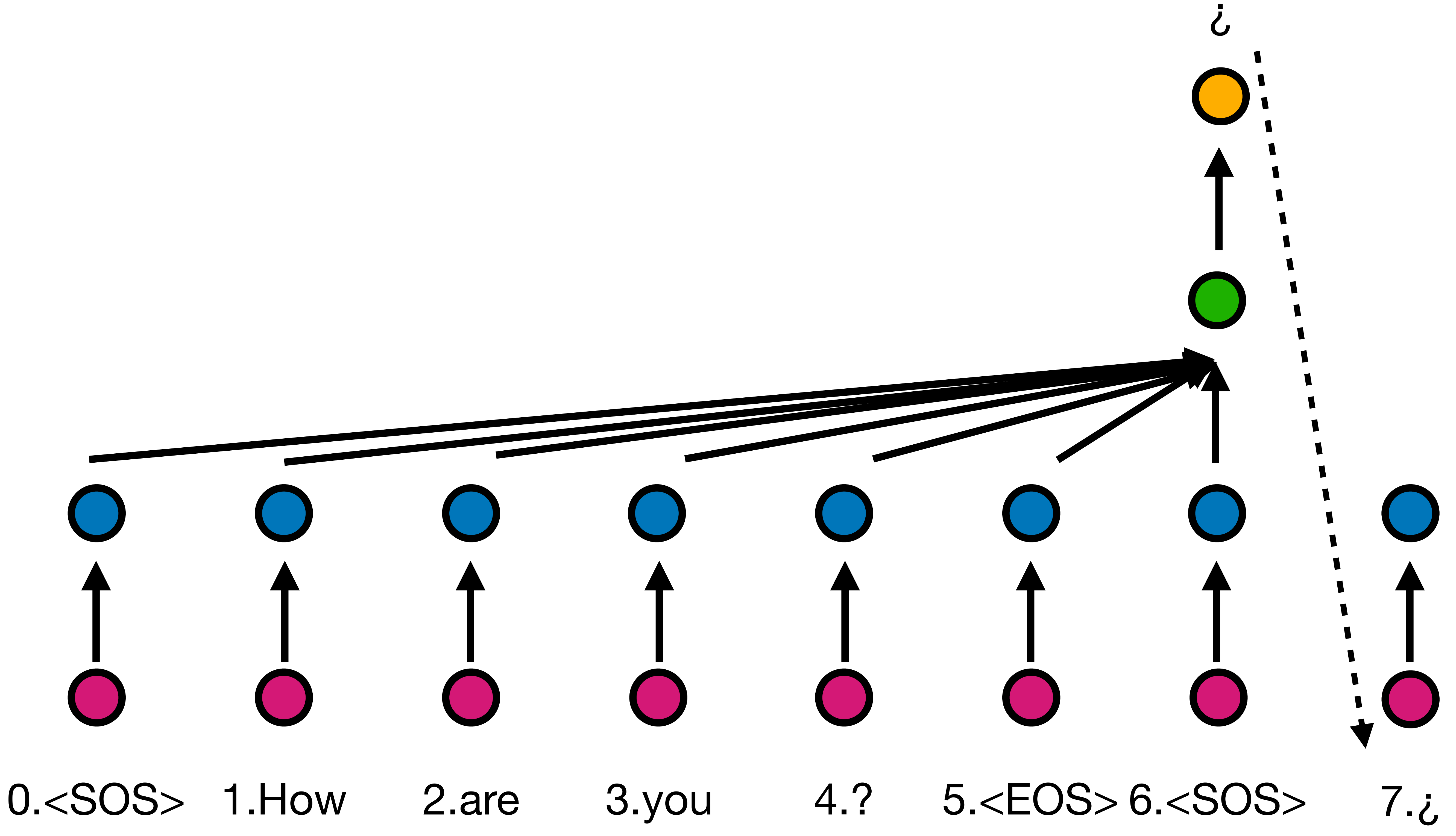
1. Start a sequence based on input text and append <SOS>.
2. Generate text autoregressively until <EOS> token is reached.



Simplified transformer for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

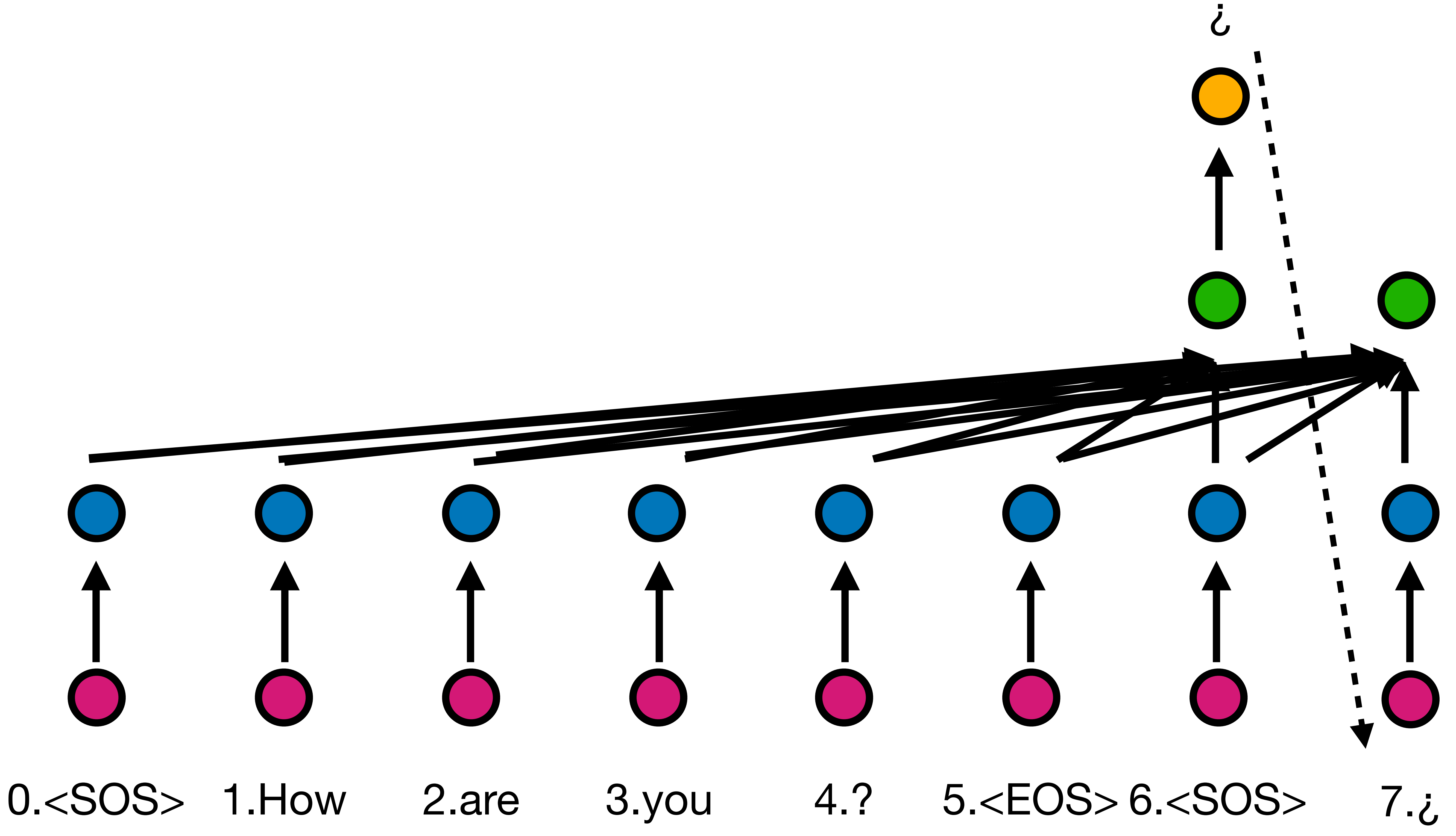
1. Start a sequence based on input text and append <SOS>.
2. Generate text autoregressively until <EOS> token is reached.



Simplified transformer for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

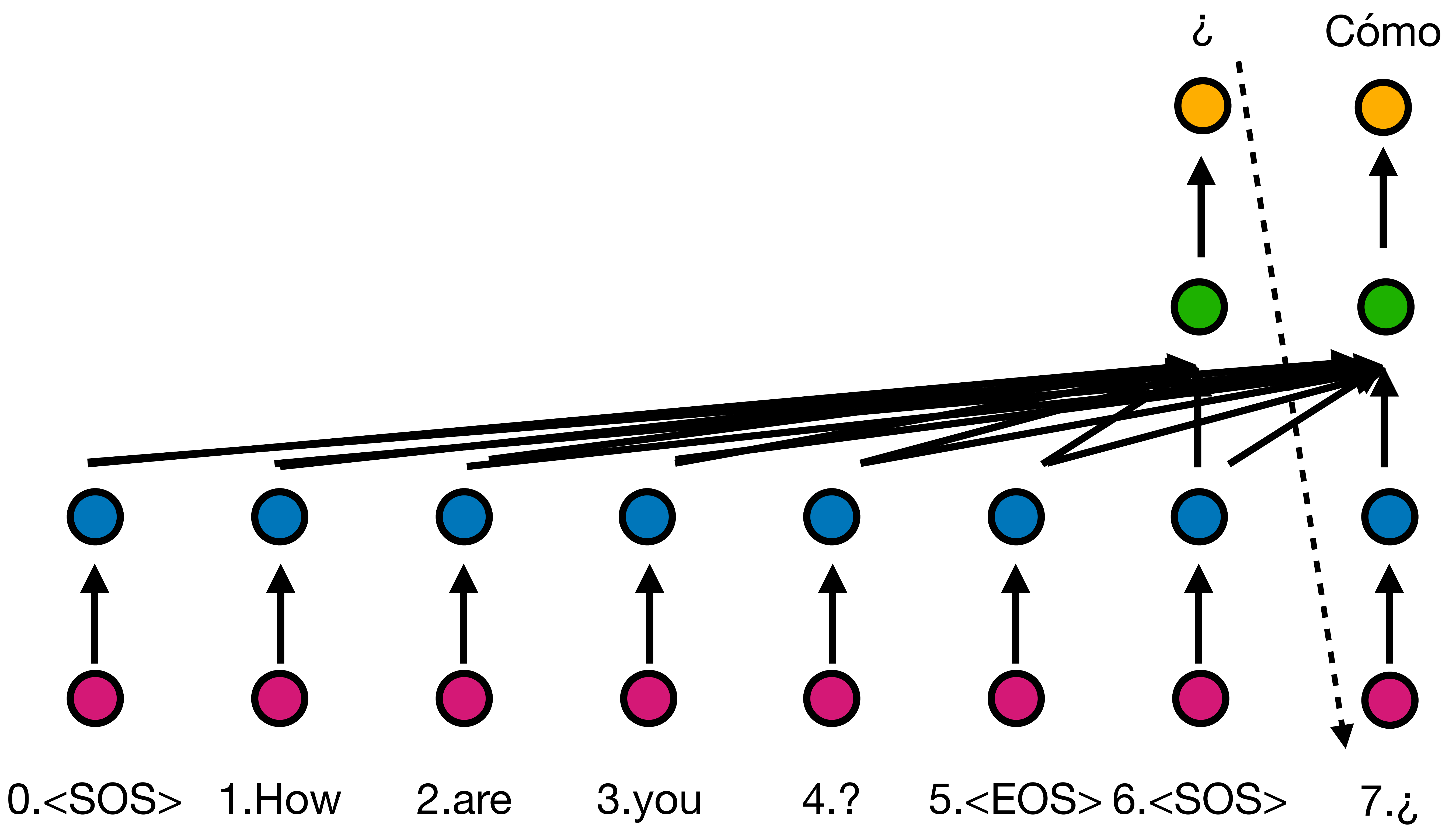
1. Start a sequence based on input text and append <SOS>.
2. Generate text autoregressively until <EOS> token is reached.



Simplified transformer for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

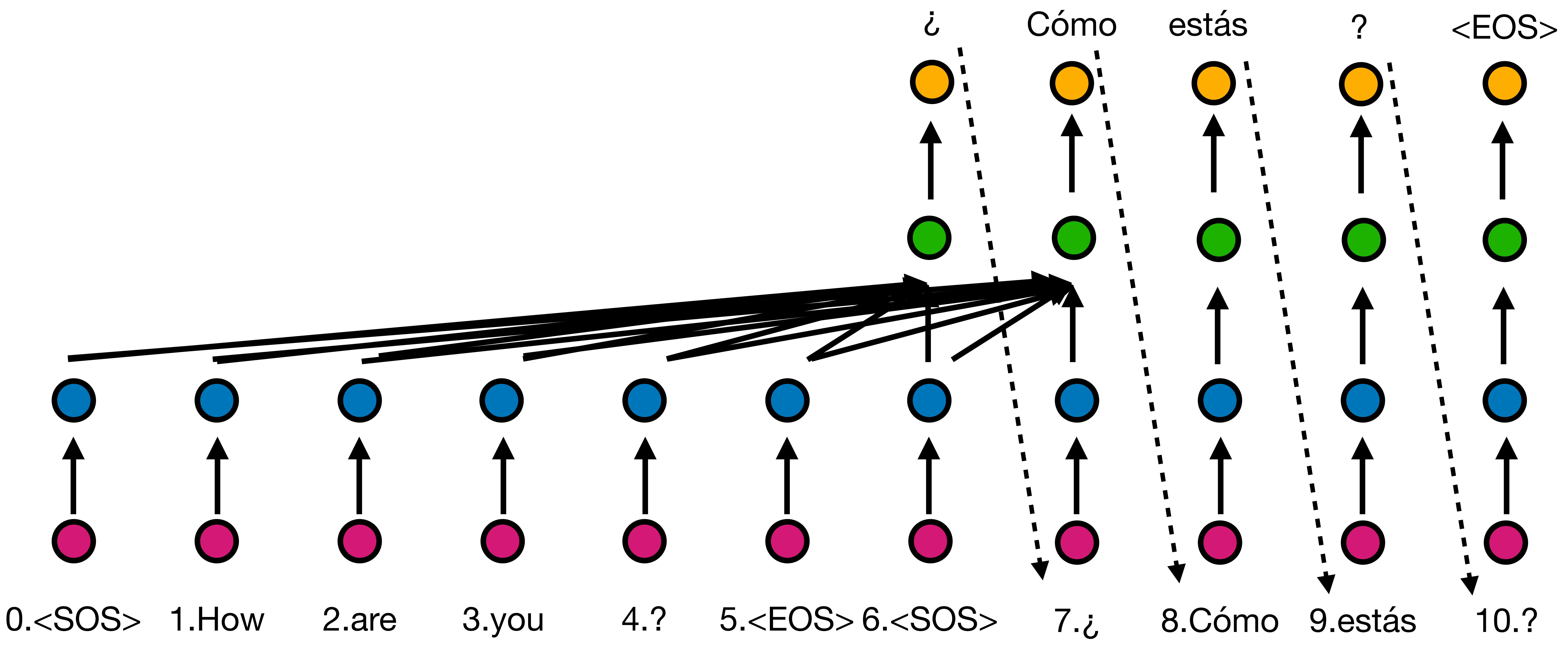
1. Start a sequence based on input text and append <SOS>.
2. Generate text autoregressively until <EOS> token is reached.



Simplified transformer for machine translation

Input text: “How are you?” Output translation: “¿Cómo estás?”

1. Start a sequence based on input text and append <SOS>.
2. Generate text autoregressively until <EOS> token is reached.



Training transformer language models

Training transformer language models

- Trained based on unlabeled text, of which there is a massive amount online.

Training transformer language models

- Trained based on unlabeled text, of which there is a massive amount online.

"My favorite season in spring. The flowers are blooming and the sun is shining…"

Input	Output
"My"	"favorite"
"My favorite"	"season"
"My favorite season"	"is"
"My favorite season is"	"spring"
…	…

Training transformer language models

- Trained based on unlabeled text, of which there is a massive amount online.

- For example, GPT-3 was trained on 45TB of data.

"My favorite season in spring. The flowers are blooming and the sun is shining…"

Input	Output
"My"	"favorite"
"My favorite"	"season"
"My favorite season"	"is"
"My favorite season is"	"spring"
…	…

Training transformer language models

- Trained based on unlabeled text, of which there is a massive amount online.
- For example, GPT-3 was trained on 45TB of data.
- Can have hundreds of billions of parameters (large language models).

"My favorite season in spring. The flowers are blooming and the sun is shining…"

Input	Output
"My"	"favorite"
"My favorite"	"season"
"My favorite season"	"is"
"My favorite season is"	"spring"
…	…

Training transformer language models

- Trained based on unlabeled text, of which there is a massive amount online.
- For example, GPT-3 was trained on 45TB of data.
- Can have hundreds of billions of parameters (large language models).
- Along with the weights of the network, word vectors themselves are trained.

"My favorite season in spring. The flowers are blooming and the sun is shining…"

Input	Output
"My"	"favorite"
"My favorite"	"season"
"My favorite season"	"is"
"My favorite season is"	"spring"
…	…

Training transformer language models

- Trained based on unlabeled text, of which there is a massive amount online.
- For example, GPT-3 was trained on 45TB of data.
- Can have hundreds of billions of parameters (large language models).
- Along with the weights of the network, word vectors themselves are trained.

"My favorite season in spring. The flowers are blooming and the sun is shining…"

Input	Output
"My"	"favorite"
"My favorite"	"season"
"My favorite season"	"is"
"My favorite season is"	"spring"
…	…

- As with other models, trained using SGD based on cross-entropy.

Training transformer language models

- Trained based on unlabeled text, of which there is a massive amount online.
- For example, GPT-3 was trained on 45TB of data.
- Can have hundreds of billions of parameters (large language models).
- Along with the weights of the network, word vectors themselves are trained.

"My favorite season in spring. The flowers are blooming and the sun is shining…"

Input	Output
"My"	"favorite"
"My favorite"	"season"
"My favorite season"	"is"
"My favorite season is"	"spring"
…	…

- As with other models, trained using SGD based on cross-entropy.
- Training is massively parallelized using specialized GPU hardware.

Fine-tuning transformers for supervised tasks

Fine-tuning transformers for supervised tasks

Sentiment analysis, machine translation, and other supervised tasks (e.g. Q&A, like ChatGPT) have much smaller labeled datasets available.

Fine-tuning transformers for supervised tasks

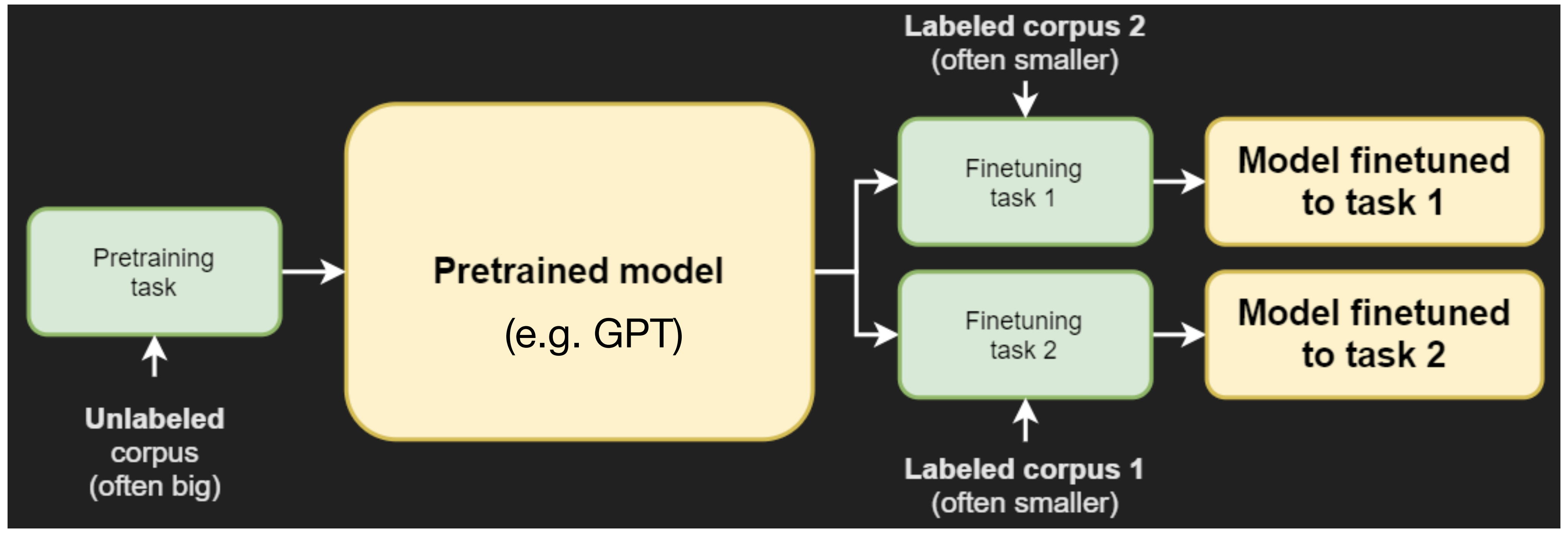
Sentiment analysis, machine translation, and other supervised tasks (e.g. Q&A, like ChatGPT) have much smaller labeled datasets available.

Key idea: Initialize transformer weights based on huge pre-trained language model, and then fine-tune these weights for supervised task.

Fine-tuning transformers for supervised tasks

Sentiment analysis, machine translation, and other supervised tasks (e.g. Q&A, like ChatGPT) have much smaller labeled datasets available.

Key idea: Initialize transformer weights based on huge pre-trained language model, and then fine-tune these weights for supervised task.



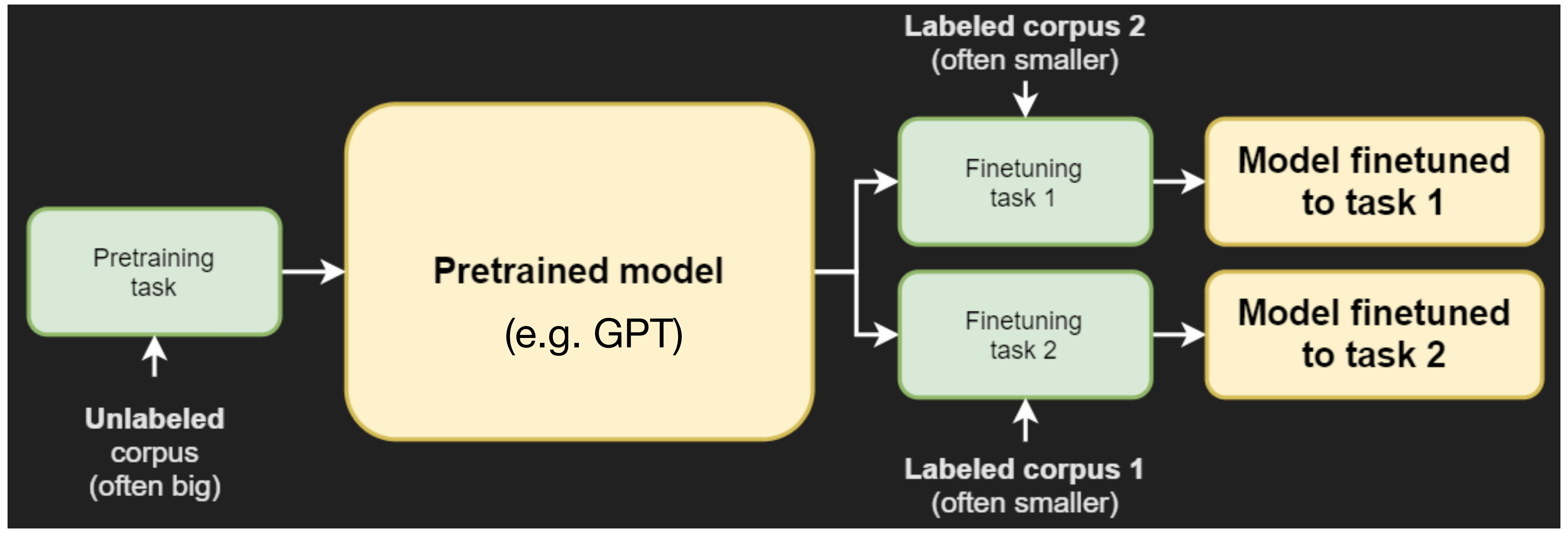
https://www.machinecurve.com/index.php/2021/01/04/intuitive-introduction-to-bert/

Fine-tuning transformers for supervised tasks

Sentiment analysis, machine translation, and other supervised tasks (e.g. Q&A, like ChatGPT) have much smaller labeled datasets available.

Key idea: Initialize transformer weights based on huge pre-trained language model, and then fine-tune these weights for supervised task.

In many cases, network architecture can stay the same; training data just needs to be formatted appropriately.



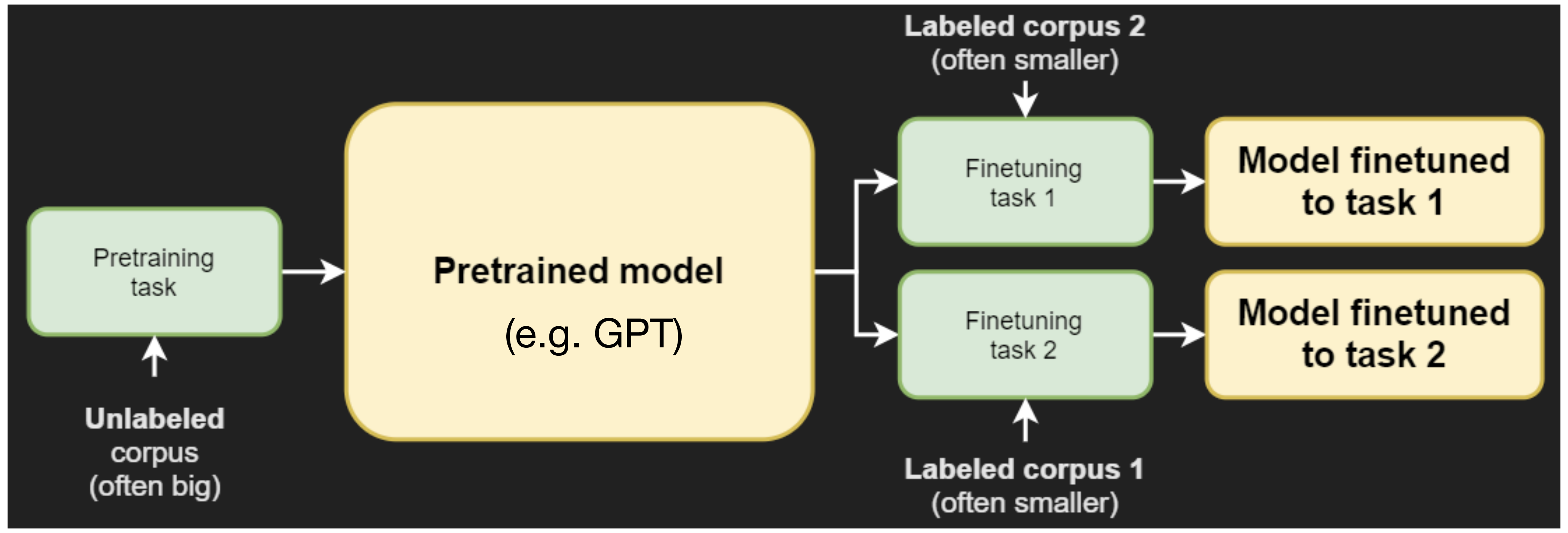
https://www.machinecurve.com/index.php/2021/01/04/intuitive-introduction-to-bert/

Fine-tuning transformers for supervised tasks

Sentiment analysis, machine translation, and other supervised tasks (e.g. Q&A, like ChatGPT) have much smaller labeled datasets available.

Key idea: Initialize transformer weights based on huge pre-trained language model, and then fine-tune these weights for supervised task.

In many cases, network architecture can stay the same; training data just needs to be formatted appropriately.



https://www.machinecurve.com/index.php/2021/01/04/intuitive-introduction-to-bert/

Input	Output
“How are you?”	“¿Cómo estás?”
“The cat sleeps.”	“El gato duerme.”
“I am reading.”	“Estoy leyendo.”
“She is happy.”	“Ella está feliz.”
…	…

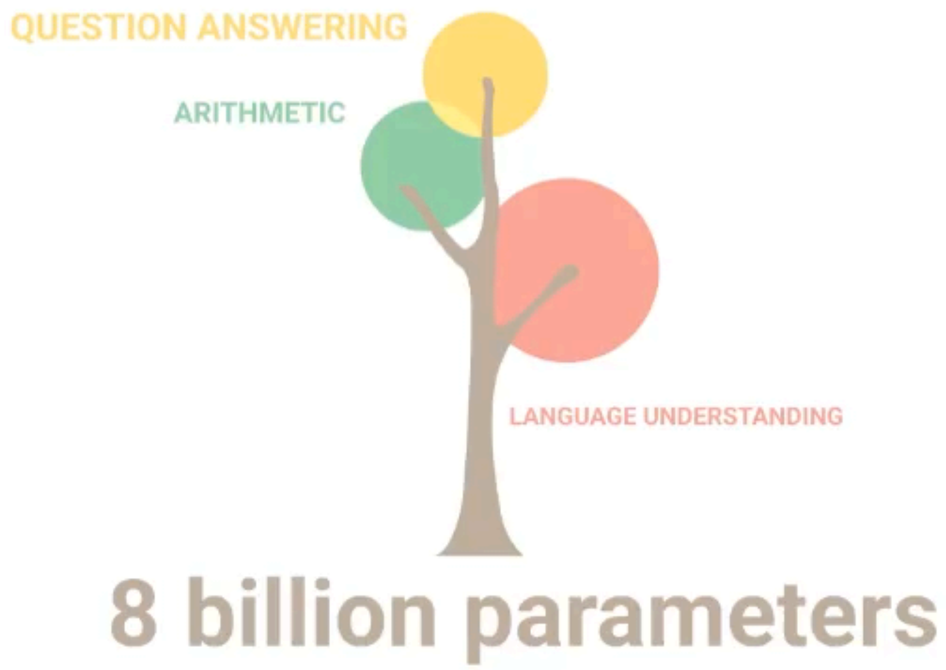
Input
How are you?<to-sp>¿Cómo estás?
The cat sleeps.<to-sp>El gato duerme.
I am reading.<to-sp>Estoy leyendo.
She is happy.<to-sp>Ella está feliz.
…

Emergent properties of LLMs

As LLMs grow, they develop emergent properties: Abilities they were not explicitly trained for.

Emergent properties of LLMs

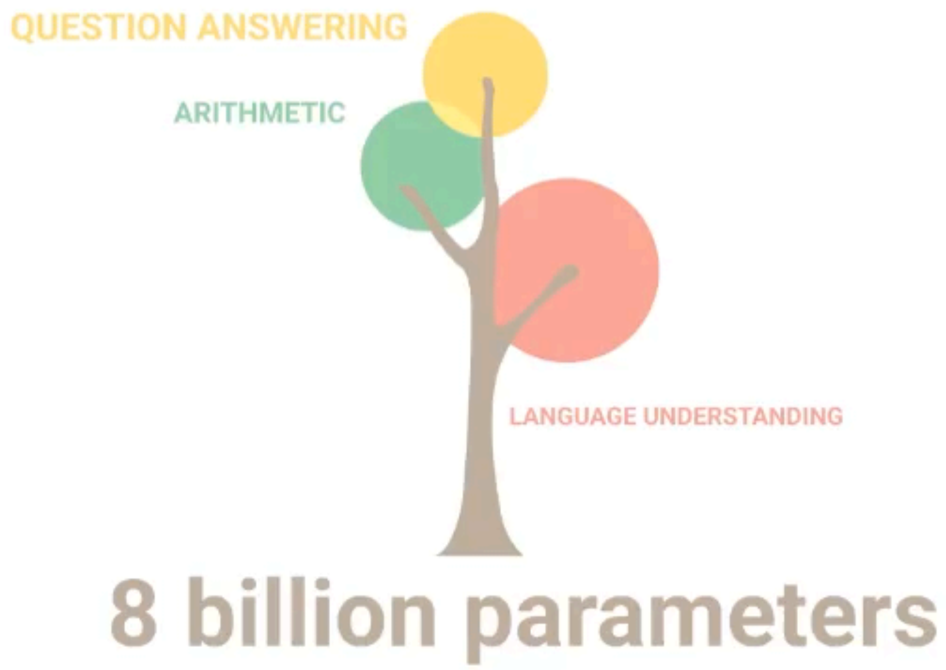
As LLMs grow, they develop emergent properties: Abilities they were not explicitly trained for.



https://www.assemblyai.com/blog/emergent-abilities-of-large-language-models/

Emergent properties of LLMs

As LLMs grow, they develop emergent properties: Abilities they were not explicitly trained for.



https://www.assemblyai.com/blog/emergent-abilities-of-large-language-models/

GPT-4 vs ChatGPT

Prompt: Here we have a book, 9 eggs, a laptop, a bottle and a nail. Please tell me how to stack them onto each other in a stable manner.

GPT-4:

```
One possible way to stack the objects onto each other in a stable manner is:

- Place the book flat on a level surface, such as a table or a floor. The book will serve as the base of
the stack and provide a large and sturdy support.
- Arrange the 9 eggs in a 3 by 3 square on top of the book, leaving some space between them. The eggs
will
form a second layer and distribute the weight evenly. Make sure the eggs are not cracked or broken, and
handle them gently to avoid dropping them.
- Place the laptop on top of the eggs, with the screen facing down and the keyboard facing up. The
laptop will fit snugly within the boundaries of the book and the eggs, and its flat and rigid surface
will provide a stable platform for the next layer. Make sure the laptop is closed and turned off, and do
not press any keys or buttons.
- Place the bottle on top of the laptop, with the cap facing up and the bottom facing down. The bottle
will add some height and balance to the stack, and its cylindrical shape will reduce the contact area
with the laptop. Make sure the bottle is empty and closed, and do not shake or tilt it.
- Place the nail on top of the bottle cap, with the pointy end facing up and the flat end facing down.
The nail will be the final and smallest object in the stack, and its sharp tip will prevent it from
rolling or sliding off the bottle. Make sure the nail is clean and straight, and do not touch or move
it.
```

ChatGPT:

```
It would be difficult to stack all of these objects on top of each other in a stable manner without
using any extra objects. The book and laptop are likely too large and heavy to stack on top of the
eggs without damaging them. The bottle and nail may be too small to provide a stable base for the other
objects.

Here is one possible way to stack the objects:

- Place the bottle on a flat surface.
- Carefully balance the nail on top of the bottle.
- Place the eggs on top of the nail, making sure they are balanced and not tilting to one side.
- Place the laptop on top of the eggs, again making sure it is balanced and not tilting to one side.
- Finally, place the book on top of the laptop.

This stack may not be very stable, so it is important to be careful when handling it. Alternatively, you
could try stacking the objects in a different order or using some of the objects as supports for others
to create a more stable stack.
```

https://arxiv.org/pdf/2303.12712.pdf

Summary

Summary

- Word vectors used to translate words into numbers for predictive modeling.

Summary

- Word vectors used to translate words into numbers for predictive modeling.
- New architectures, such as recurrent neural networks and transformers, needed for variable-length inputs and outputs.

Summary

- Word vectors used to translate words into numbers for predictive modeling.
- New architectures, such as recurrent neural networks and transformers, needed for variable-length inputs and outputs.
- RNNs work by processing the input sequence one word at a time, updating a hidden representation of the input using a fixed set of weights.

Summary

- Word vectors used to translate words into numbers for predictive modeling.
- New architectures, such as recurrent neural networks and transformers, needed for variable-length inputs and outputs.
- RNNs work by processing the input sequence one word at a time, updating a hidden representation of the input using a fixed set of weights.
- RNNs tend to forget text after a while, so they were augmented with memory and attention mechanisms.

Summary

- Word vectors used to translate words into numbers for predictive modeling.
- New architectures, such as recurrent neural networks and transformers, needed for variable-length inputs and outputs.
- RNNs work by processing the input sequence one word at a time, updating a hidden representation of the input using a fixed set of weights.
- RNNs tend to forget text after a while, so they were augmented with memory and attention mechanisms.
- These fancier RNNs were displaced by transformers, an attention-based architecture allowing parallelized training.

Summary

- Word vectors used to translate words into numbers for predictive modeling.
- New architectures, such as recurrent neural networks and transformers, needed for variable-length inputs and outputs.
- RNNs work by processing the input sequence one word at a time, updating a hidden representation of the input using a fixed set of weights.
- RNNs tend to forget text after a while, so they were augmented with memory and attention mechanisms.
- These fancier RNNs were displaced by transformers, an attention-based architecture allowing parallelized training.
- The hard work of training large language models can be recycled through the pre-training and fine-tuning paradigm.