Introduction to Transformers for NLP

where we are and how we got here

Olga Petrova
AI Product Manager

Preliminaries

- Who I am:
  - Product Manager for AI PaaS at Scaleway

    Scaleway: European cloud provider, originating from France

Preliminaries

- Who I am:
  - Product Manager for AI PaaS at Scaleway

    Scaleway: European cloud provider, originating from France

    Smart labeling: data annotation platform for computer vision (summer 2021) and NLP (2022)

Preliminaries

- Who I am:
  - Product Manager for AI PaaS at Scaleway
    Scaleway: European cloud provider, originating from France
    Smart labeling: data annotation platform for computer vision (summer 2021) and NLP (2022)
  - Machine Learning engineer at Scaleway

Preliminaries

- Who I am:
  - Product Manager for AI PaaS at Scaleway
    Scaleway: European cloud provider, originating from France
    Smart labeling: data annotation platform for computer vision (summer 2021) and NLP (2022)
  - Machine Learning engineer at Scaleway
  - Quantum physicist at École Normale Supérieure, the Max Planck Institute, Johns Hopkins University

Preliminaries

- What this talk is about:
    - Transformer neural networks for NLP

Preliminaries

- What this talk is about:
    - Transformer neural networks for NLP, but also:

Preliminaries

- What this talk is about:
    - Transformer neural networks for NLP, but also:
    - Recurrent Neural Networks
    - (Written) text pre-processing: tokenization, word embeddings (word2vec)

Preliminaries

- What this talk is about:
    - Transformer neural networks for NLP, but also:
    - Recurrent Neural Networks
    - (Written) text pre-processing: tokenization, word embeddings (word2vec)

    Common across most NLP tasks (text classification, language modeling, machine translation, etc)

Preliminaries

- What this talk is about:
  - Transformer neural networks for NLP, but also:
  - Recurrent Neural Networks
  - (Written) text pre-processing: tokenization, word embeddings (word2vec)

    Common across most NLP tasks (text classification, language modeling, machine translation, etc)
  - BERT: contextual word embeddings

Preliminaries

- What this talk is about:
  - Transformer neural networks for NLP, but also:
  - Recurrent Neural Networks
  - (Written) text pre-processing: tokenization, word embeddings (word2vec)

    Common across most NLP tasks (text classification, language modeling, machine translation, etc)
  - BERT: contextual word embeddings
- Who this talk is for:
  - Enough intro for newcomers to NLP

Preliminaries

- What this talk is about:
    - Transformer neural networks for NLP, but also:
    - Recurrent Neural Networks
    - (Written) text pre-processing: tokenization, word embeddings (word2vec)

      Common across most NLP tasks (text classification, language modeling, machine translation, etc)
    - BERT: contextual word embeddings
- Who this talk is for:
    - Enough intro for newcomers to NLP
    - Enough transformers for those who are used to RNNs

Outline

1. NLP 101: how do we represent words in machine learning
   🤖 Tokenization 🤖 Word embeddings
2. Order matters: Recurrent Neural Networks
   🤖 RNNs 🤖 Seq2Seq models
   🤖 What is wrong with RNNs
3. Attention is all you need!
   🤖 What is Attention 🤖 Attention, the linear algebra perspective
4. The Transformer architecture
   🤖 Mostly Attention
5. BERT: *contextualized* word embeddings

Outline

1. NLP 101: how do we represent words in machine learning

   🤖 Tokenization 🤖 Word embeddings

2. Order matters: Recurrent Neural Networks

   🤖 RNNs 🤖 Seq2Seq models

   🤖 What is wrong with RNNs

3. Attention is all you need!

   🤖 What is Attention 🤖 Attention, the linear algebra perspective

4. The Transformer architecture

   🤖 Mostly Attention

5. BERT: *contextualized* word embeddings

Representing words

Representing words

I play with my dog.

Representing words

I play with my dog.

- Machine learning: inputs are vectors / matrices

Representing words

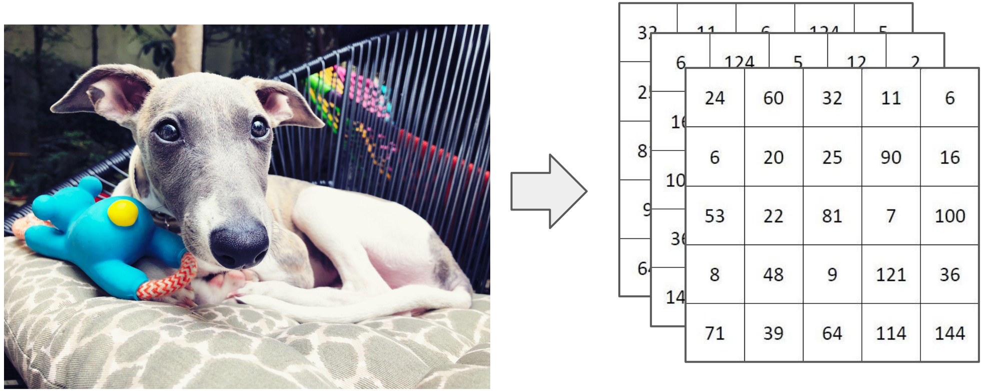
I play with my dog.

- Machine learning: inputs are vectors / matrices

Representing words

I play with my dog.

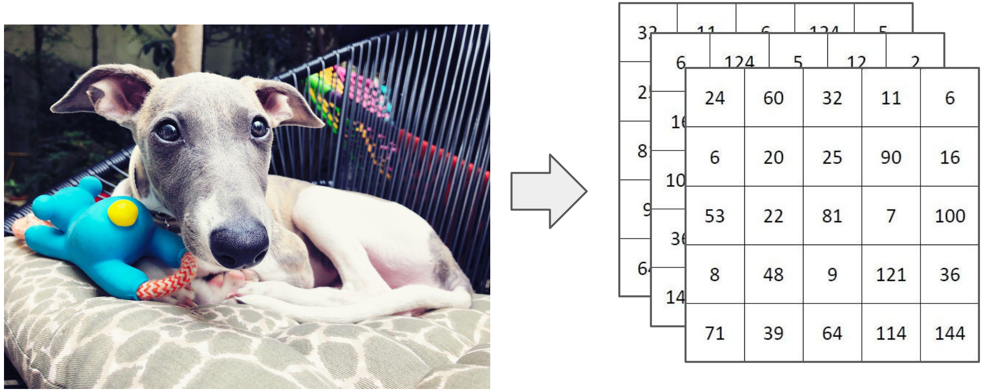
- Machine learning: inputs are vectors / matrices



Representing words

I play with my dog.

- Machine learning: inputs are vectors / matrices

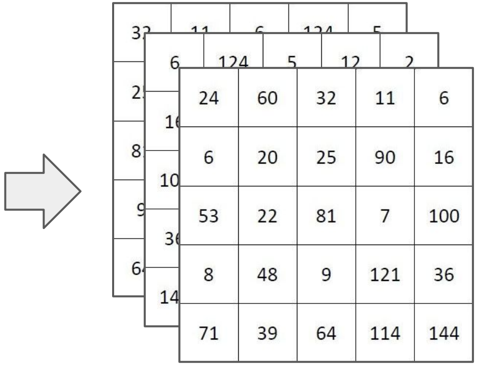


ID	Age	Gender	Weight	Diagnosis
264264	27	0	72	1
908696	61	1	80	0

Representing words

I play with my dog.

- Machine learning: inputs are vectors / matrices



ID	Age	Gender	Weight	Diagnosis
264264	27	0	72	1
908696	61	1	80	0

- How do we convert “I play with my dog” into numerical values?

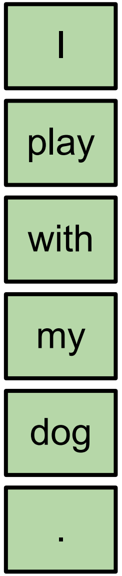
Representing words

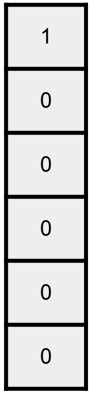
I play with my dog.

I
play
with
my
dog
.

Representing words

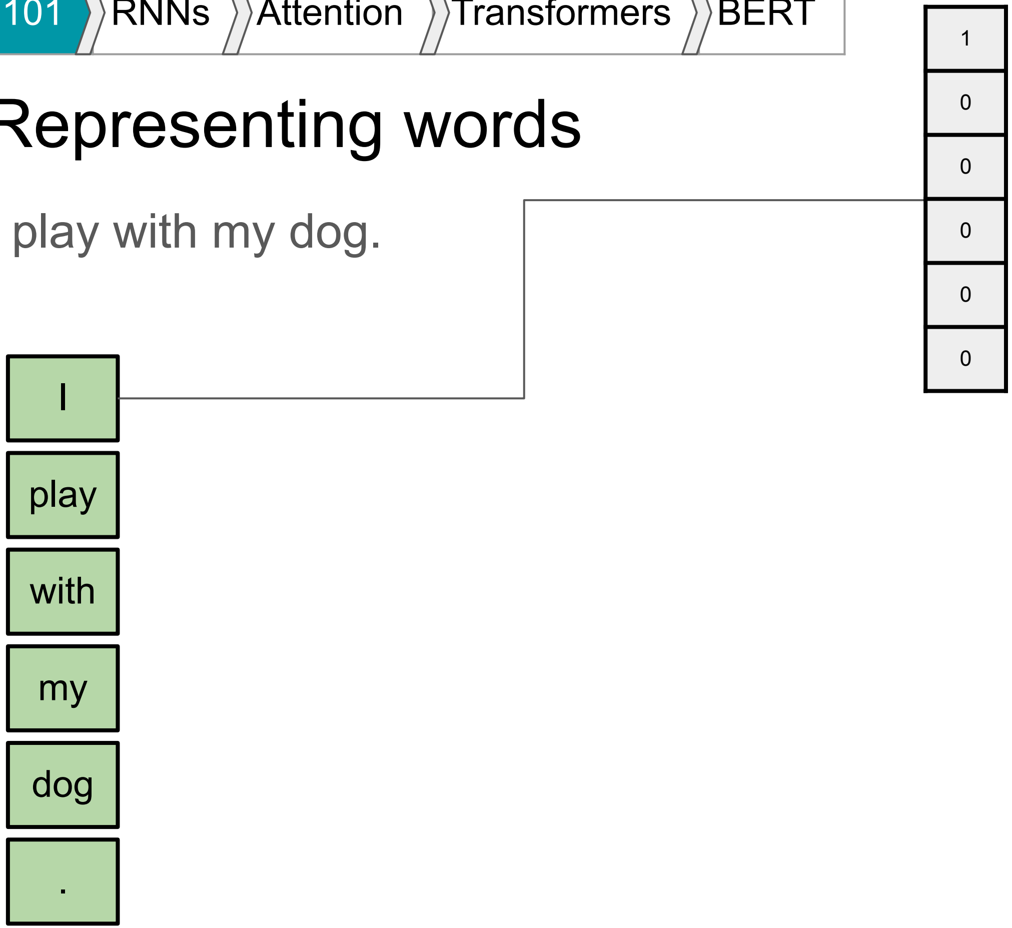
I play with my dog.





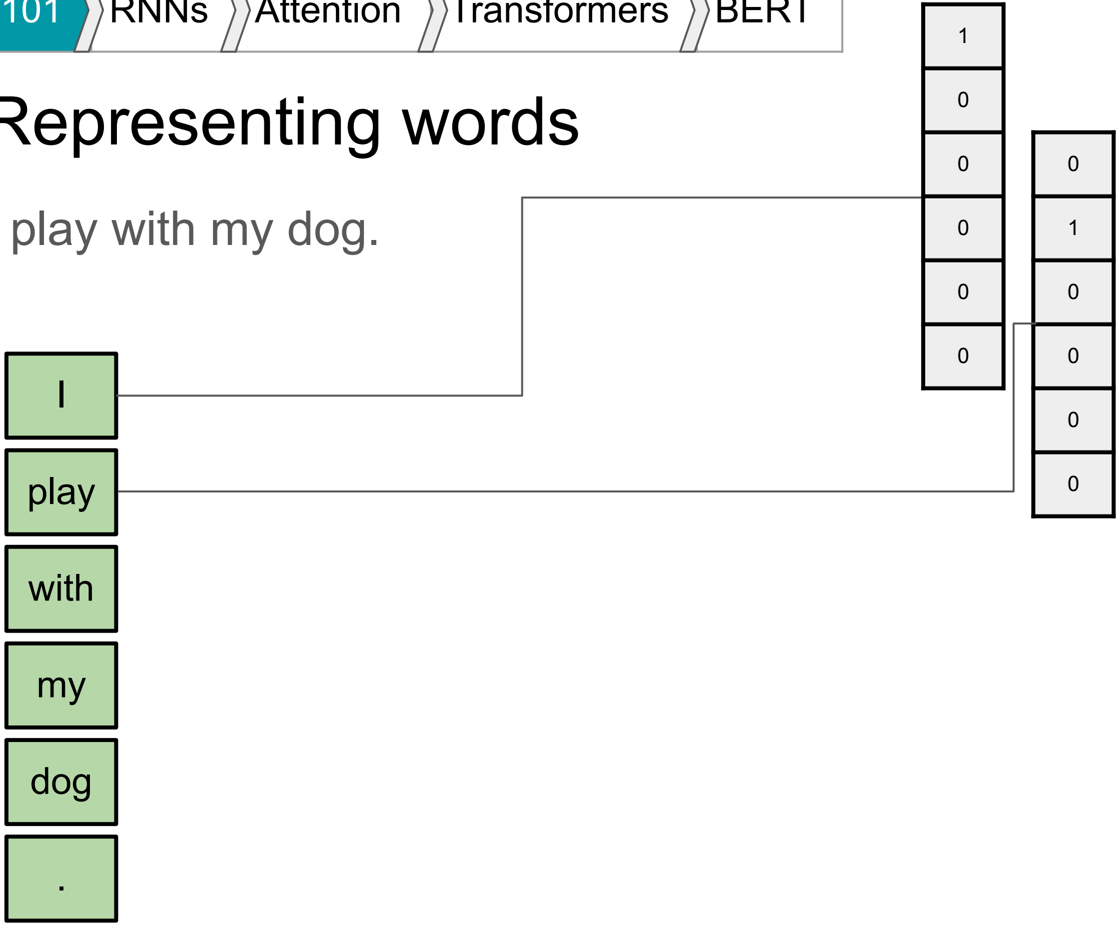
Representing words

I play with my dog.



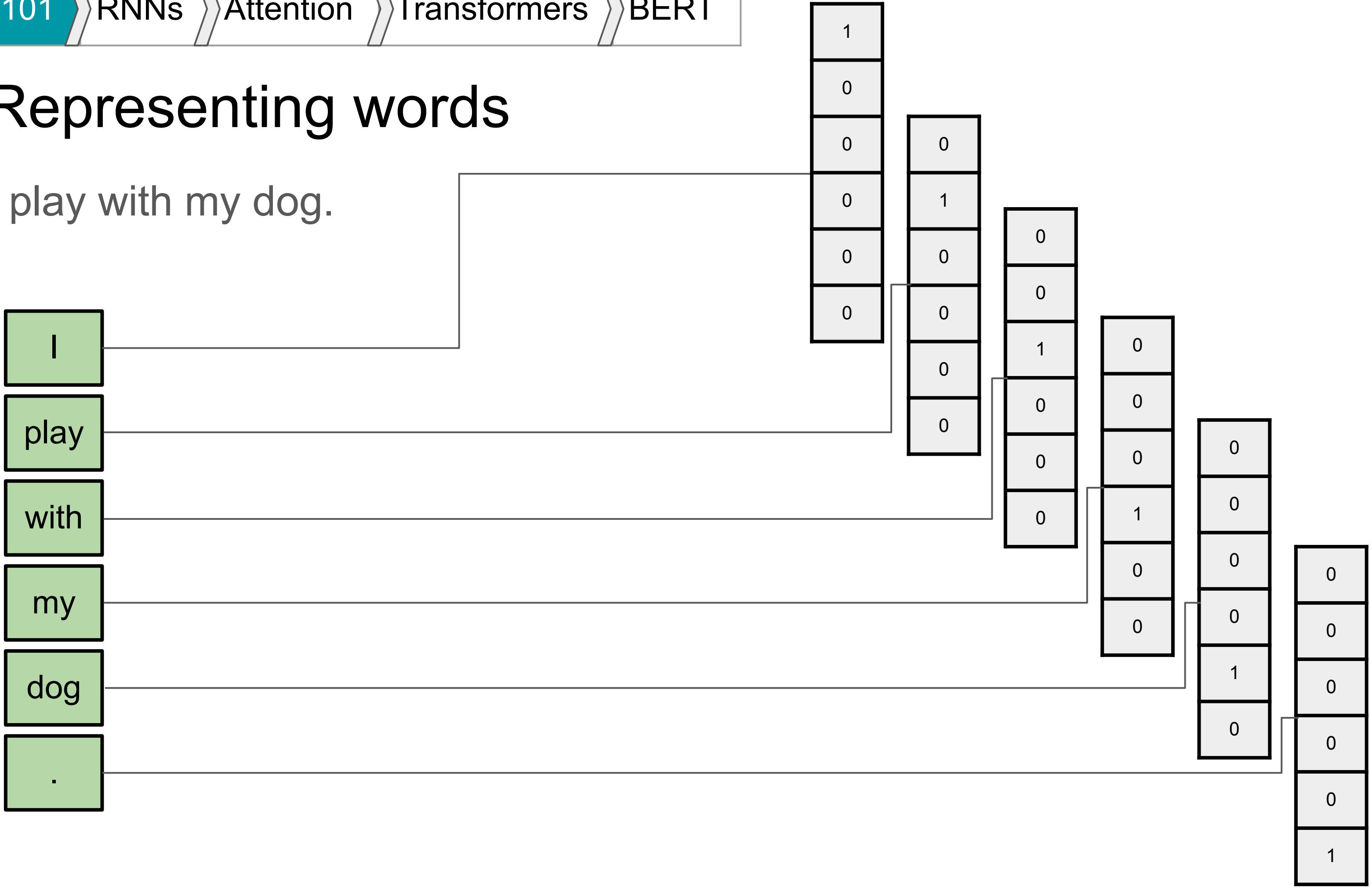
Representing words

I play with my dog.



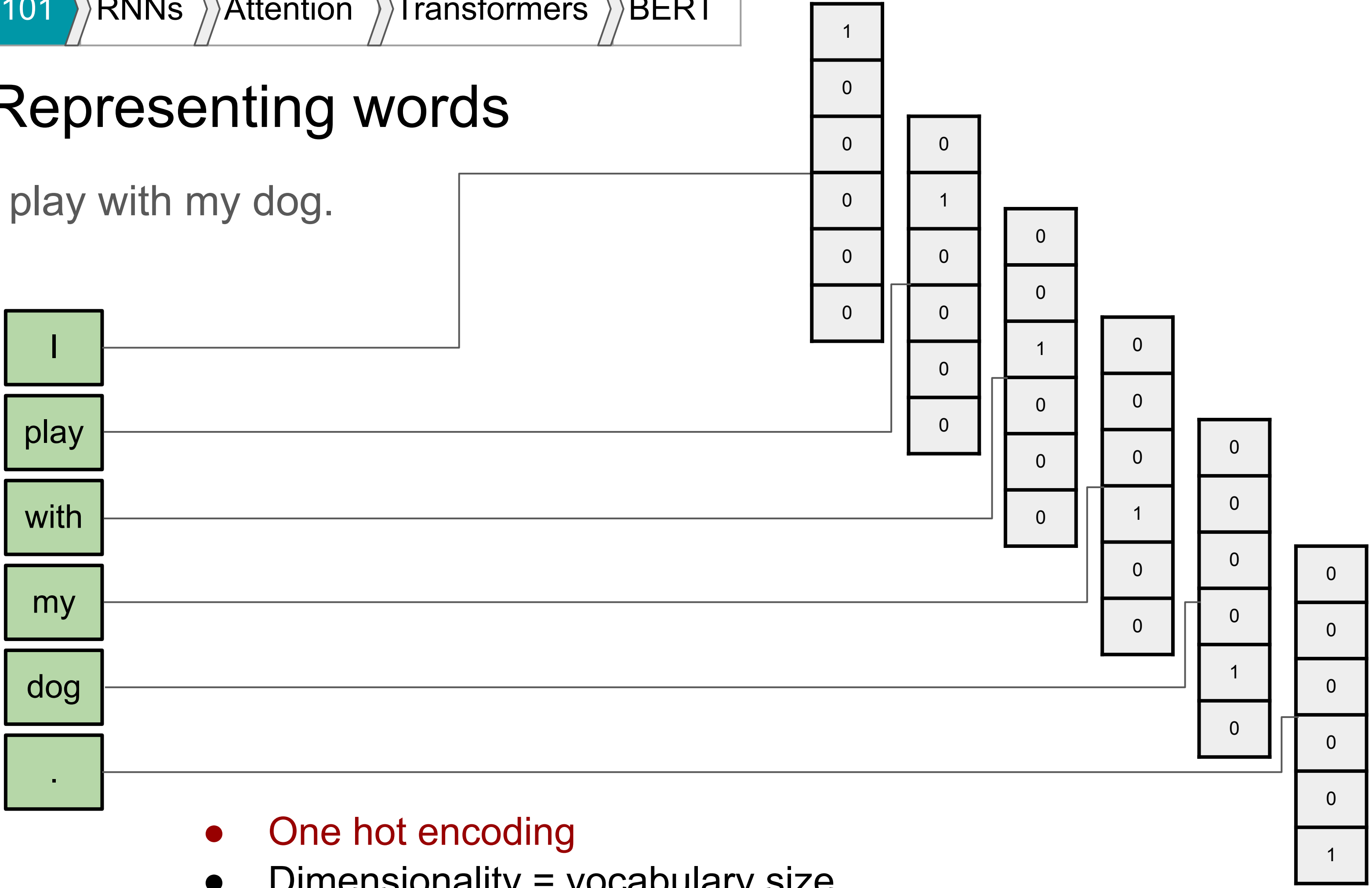
Representing words

I play with my dog.



Representing words

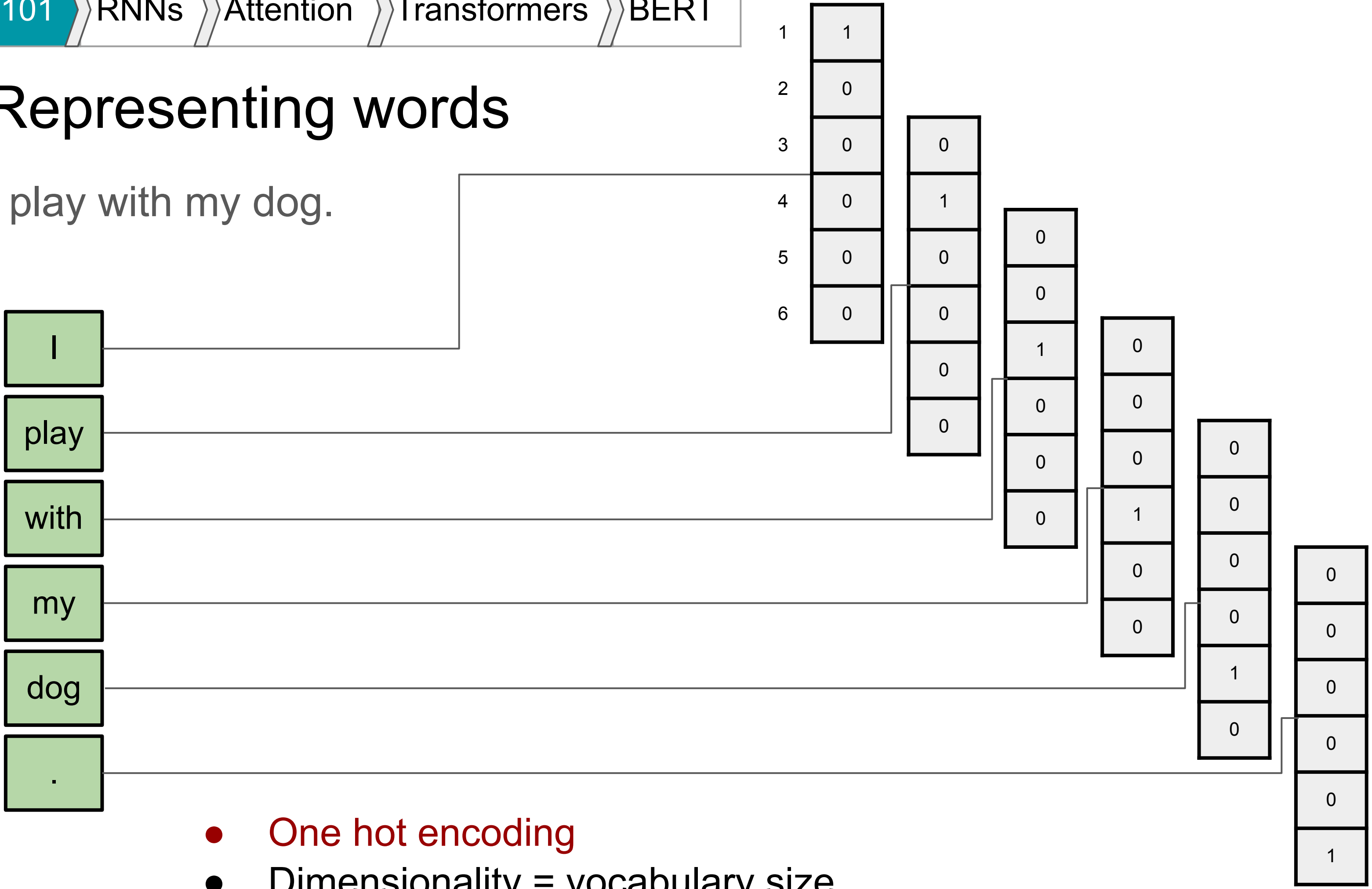
I play with my dog.



- One hot encoding
- Dimensionality = vocabulary size

Representing words

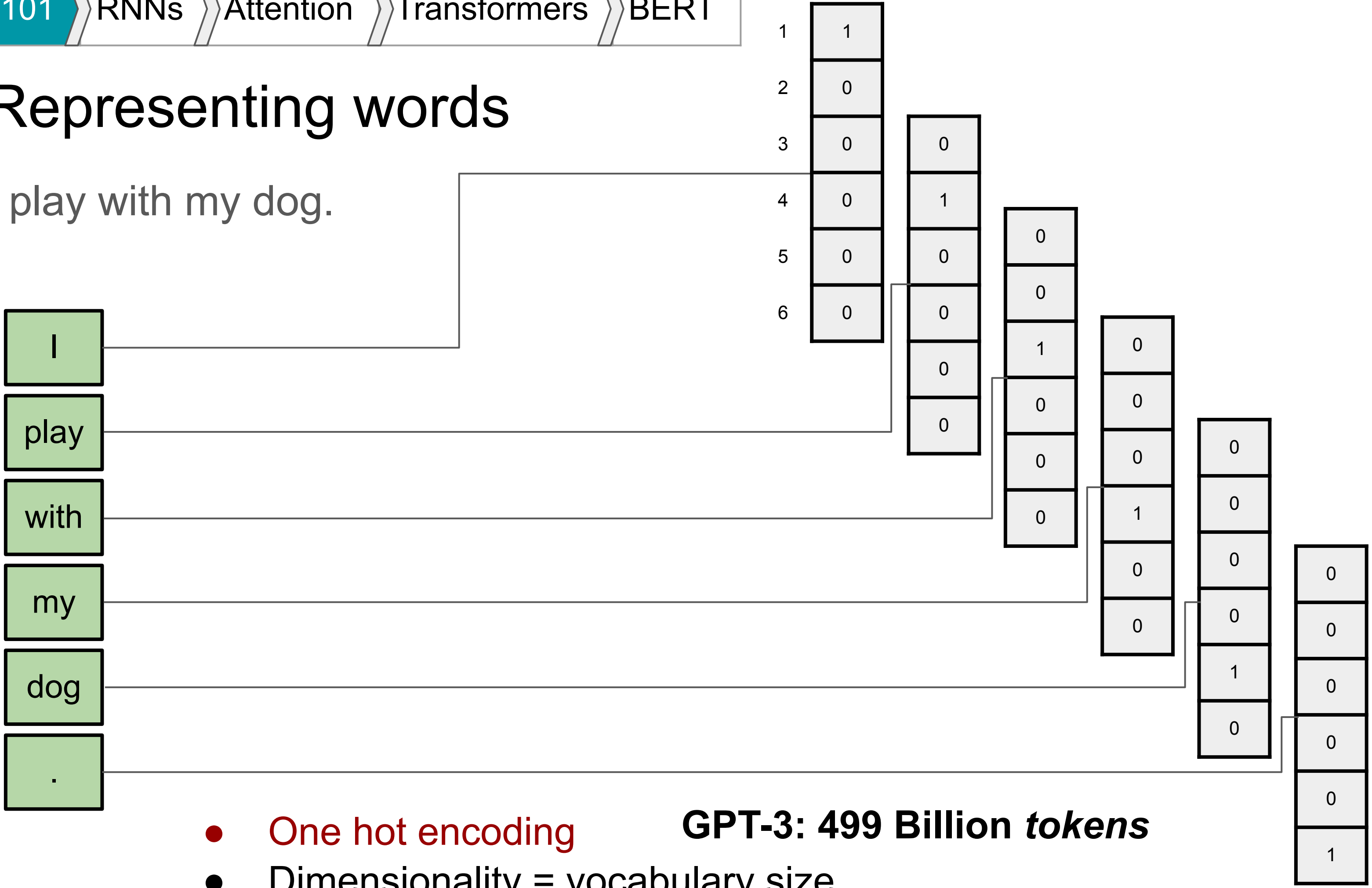
I play with my dog.



- One hot encoding
- Dimensionality = vocabulary size

Representing words

I play with my dog.



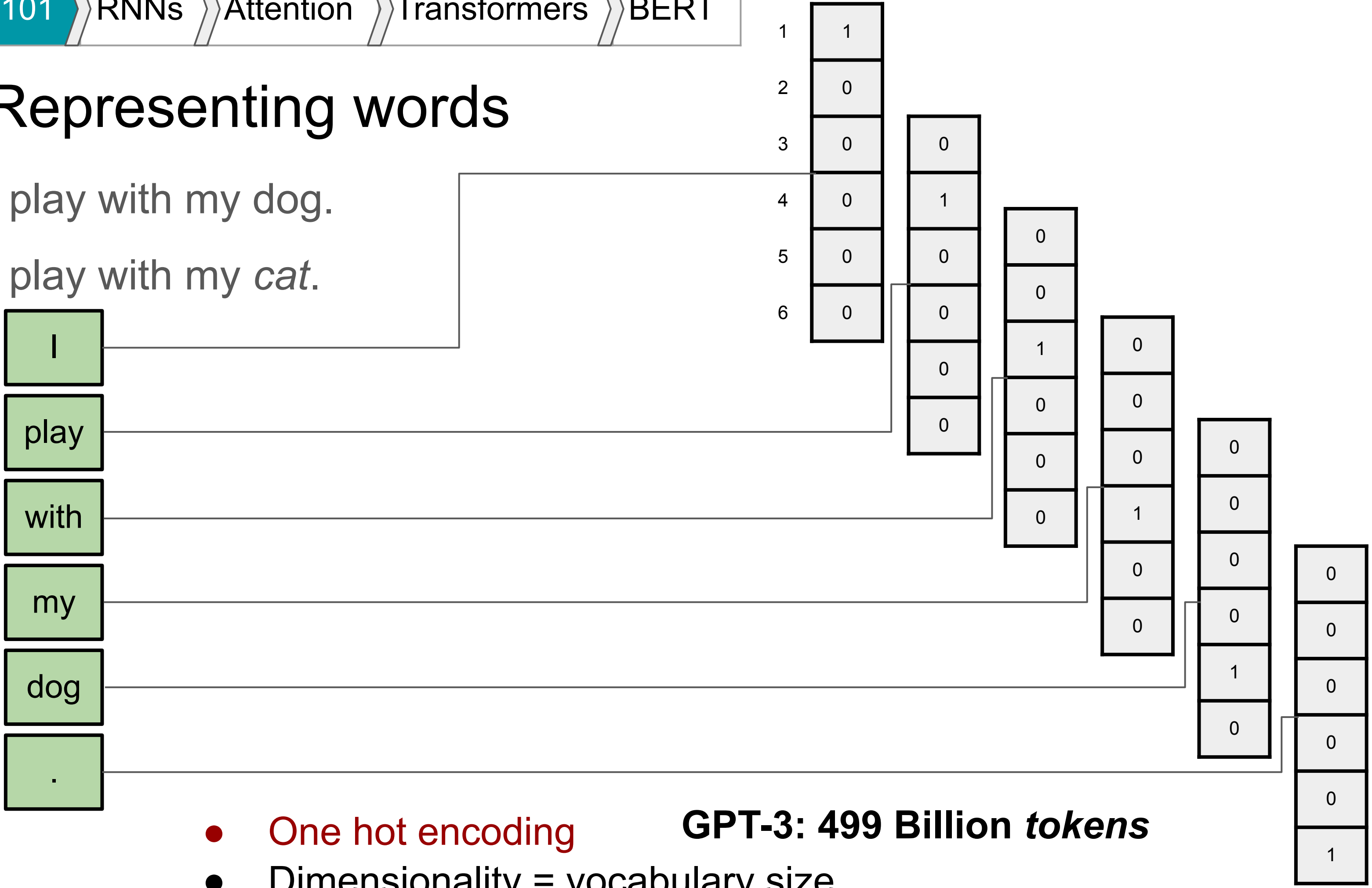
GPT-3: 499 Billion *tokens*

- One hot encoding
- Dimensionality = vocabulary size

Representing words

I play with my dog.

I play with my *cat*.



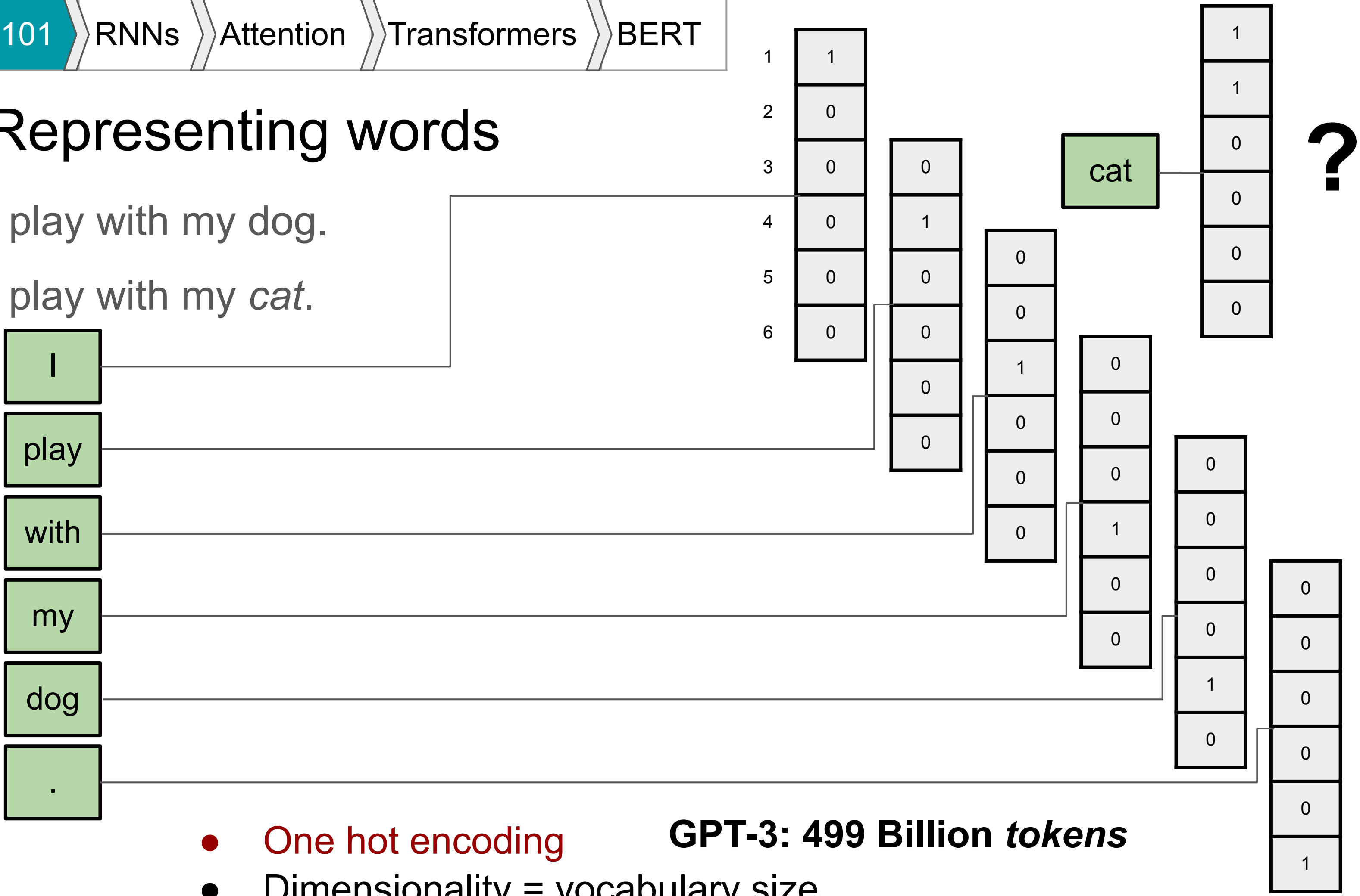
- One hot encoding
- Dimensionality = vocabulary size

GPT-3: 499 Billion *tokens*

Representing words

I play with my dog.

I play with my *cat*.



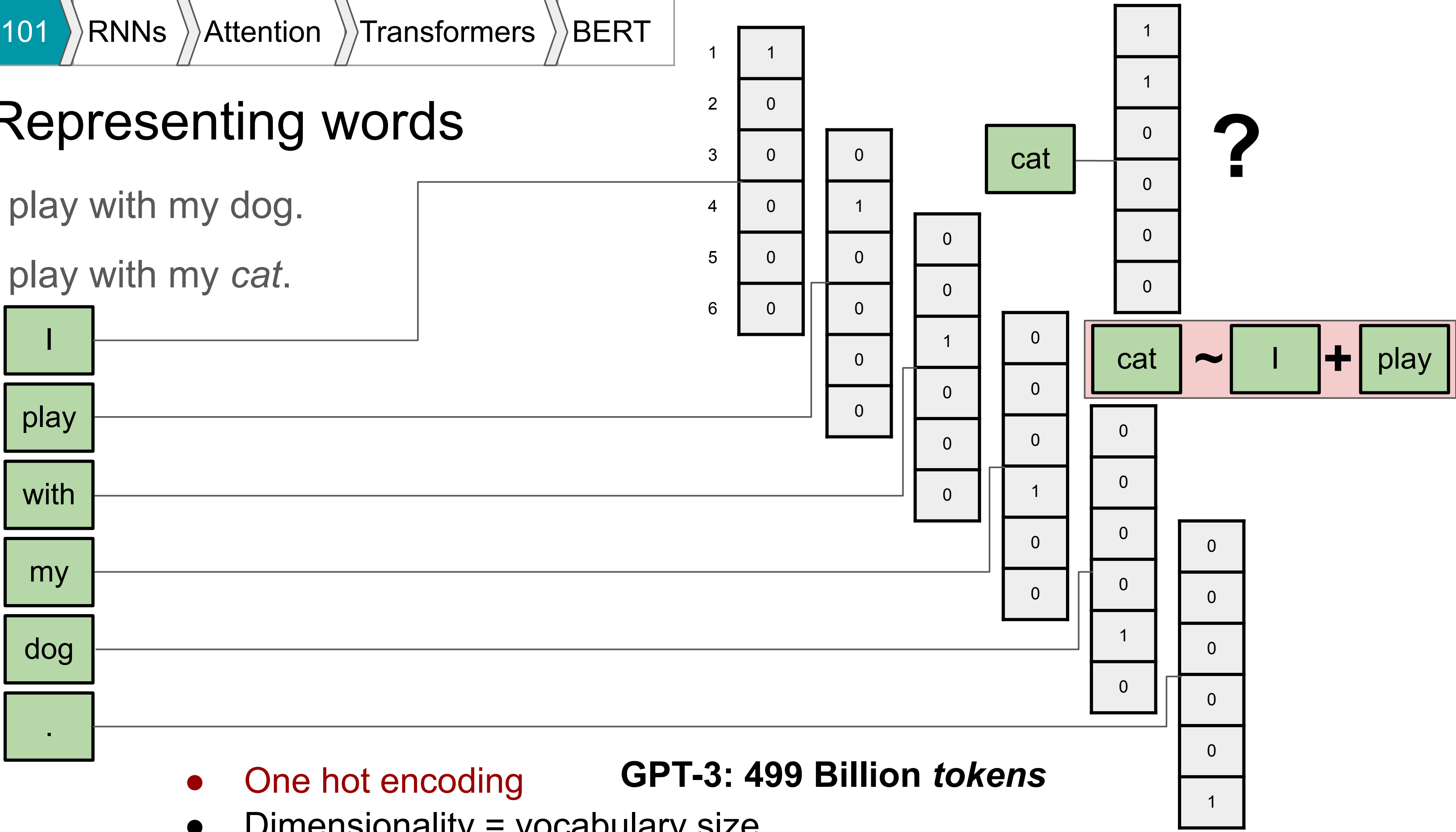
- One hot encoding
- Dimensionality = vocabulary size

GPT-3: 499 Billion *tokens*

Representing words

I play with my dog.

I play with my *cat*.



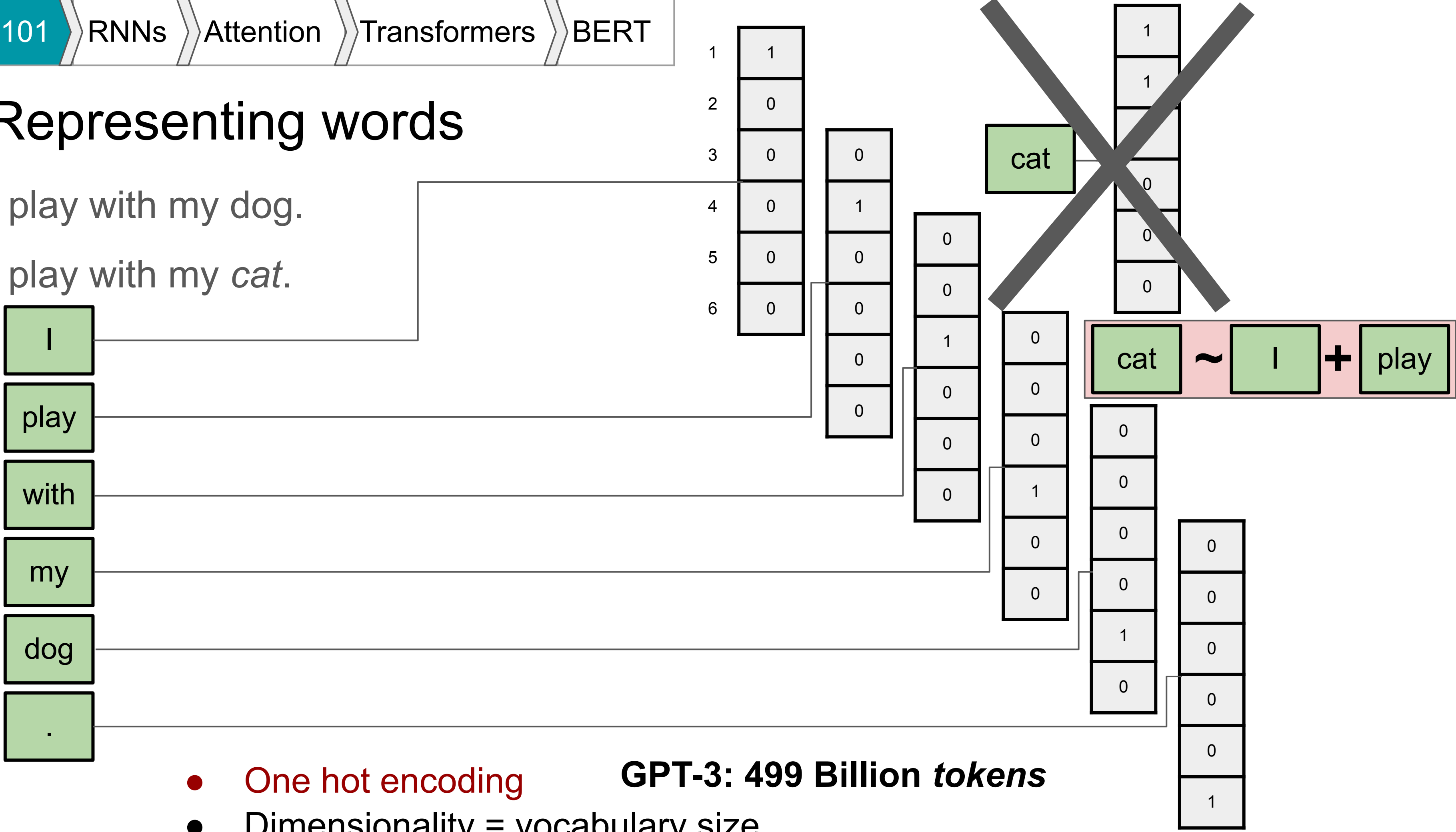
GPT-3: 499 Billion *tokens*

- One hot encoding
- Dimensionality = vocabulary size

Representing words

I play with my dog.

I play with my *cat*.



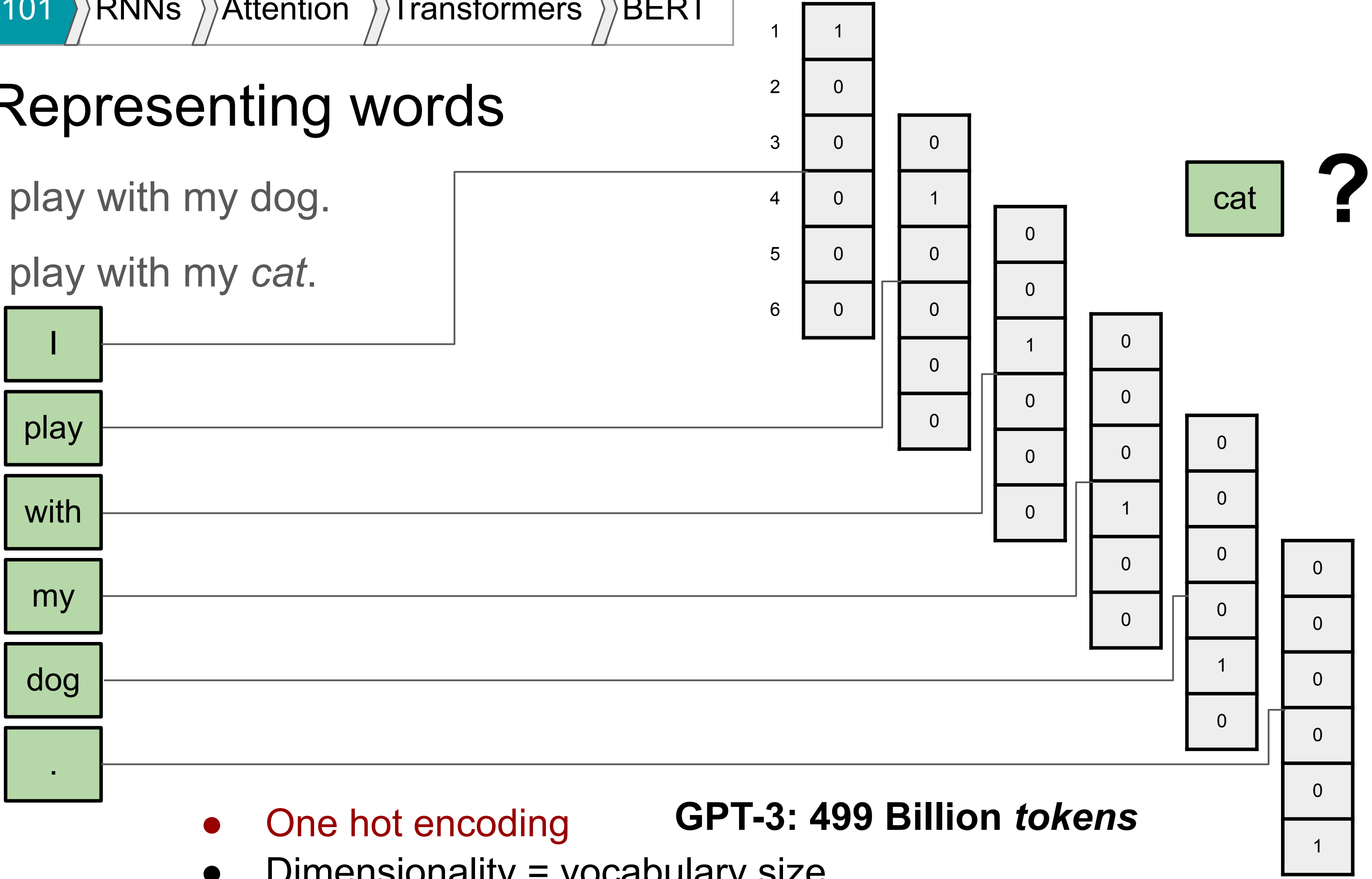
- One hot encoding
- Dimensionality = vocabulary size

GPT-3: 499 Billion *tokens*

Representing words

I play with my dog.

I play with my *cat*.



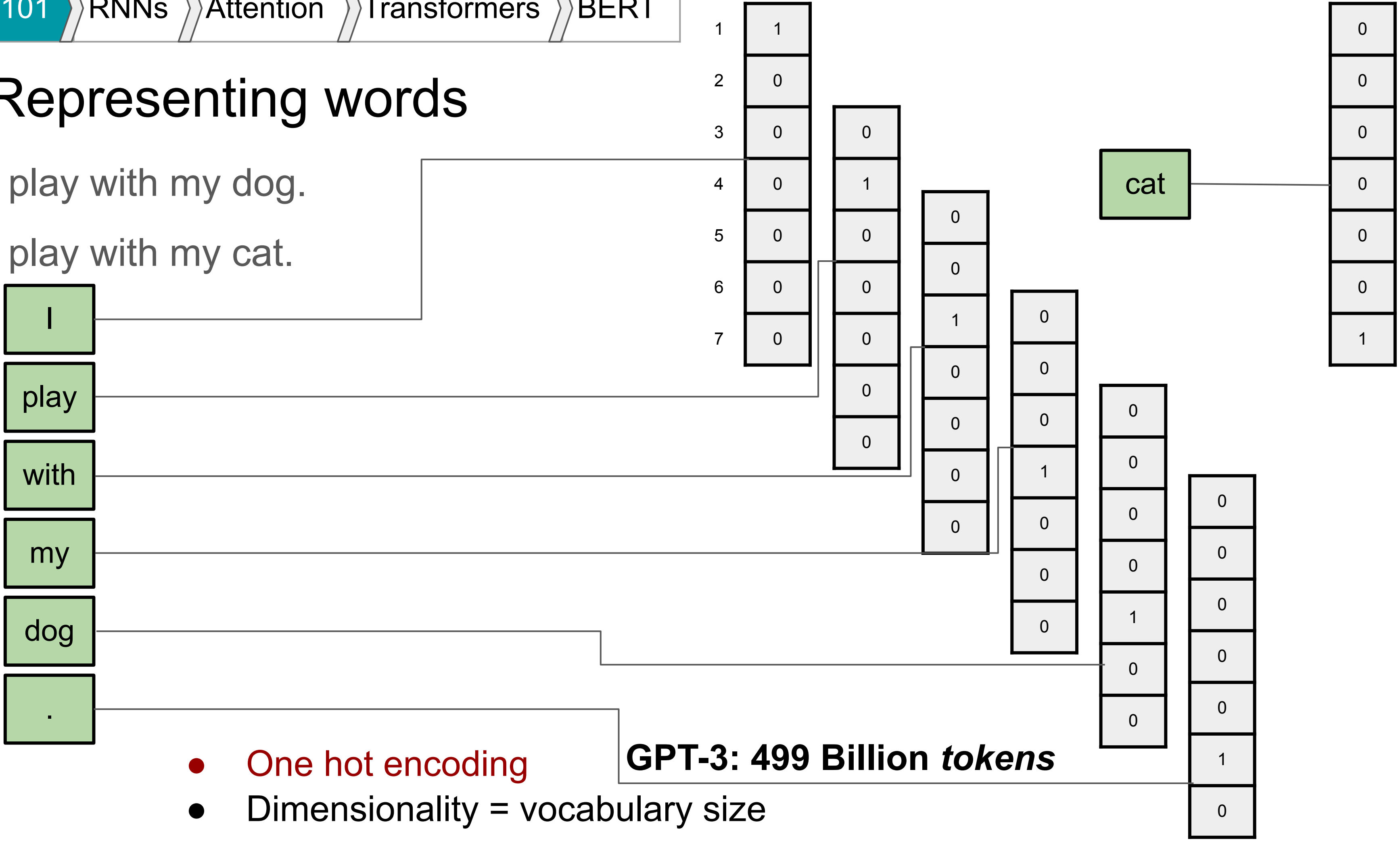
- One hot encoding
- Dimensionality = vocabulary size

GPT-3: 499 Billion *tokens*

Representing words

I play with my dog.

I play with my cat.



GPT-3: 499 Billion *tokens*

- One hot encoding
- Dimensionality = vocabulary size

High dimensional inputs

✅ One hot vectors are sparse

High dimensional inputs

✅ One hot vectors are sparse

⛔ However:

➢ Higher dimensional inputs → more trainable parameters to learn
→ more training data needed

High dimensional inputs

✅ One hot vectors are sparse

⛔ However:

➢ Higher dimensional inputs → more trainable parameters to learn
→ more training data needed

➢ Words are **not** linearly independent
→ want word representations to capture relationships

High dimensional inputs

✅ One hot vectors are sparse

⛔ However:

- Higher dimensional inputs → more trainable parameters to learn
  → more training data needed
- Words are **not** linearly independent
  → want word representations to capture relationships

👉 Tokenization, then word embeddings

Tokenization

I play / played / was playing with my dog.

Tokenization

I play / played / was playing with my dog.

Word = token: play, played, playing

Tokenization

I play / played / was playing with my dog.

Word = token: play, played, playing

Subword = token: play, #ed, #ing

Tokenization

I play / played / was playing with my dog.

Word = token: play , played , playing

Subword = token: play , #ed , #ing

E.g.: played = play ; #ed

Tokenization

I play / played / was playing with my dog.

Word = token: play, played, playing

Subword = token: play, #ed, #ing

E.g.: played = play ; #ed

Why tokenize at subword level?

Tokenization

- Cut down on the size of the vocabulary:

play | played | playing   ask | asked | asking   call | called | calling   burn | burned | burning

vs.

play | ask | call | burn | #ed | #ing

Tokenization

- Cut down on the size of the vocabulary:

play	played	playing	ask	asked	asking	call	called	calling	burn	burned	burning

vs.

play	ask	call	burn	#ed	#ing

- Fewer ***Unknown*** words, partial correction of misspellings

Tokenization

- Cut down on the size of the vocabulary:

play	played	playing	ask	asked	asking	call	called	calling	burn	burned	burning

vs.

play	ask	call	burn	#ed	#ing

- Fewer ***Unknown*** words, partial correction of misspellings
- Parts of related words (e.g. play / played) will be represented by the same vector

Tokenization

- Cut down on the size of the vocabulary:

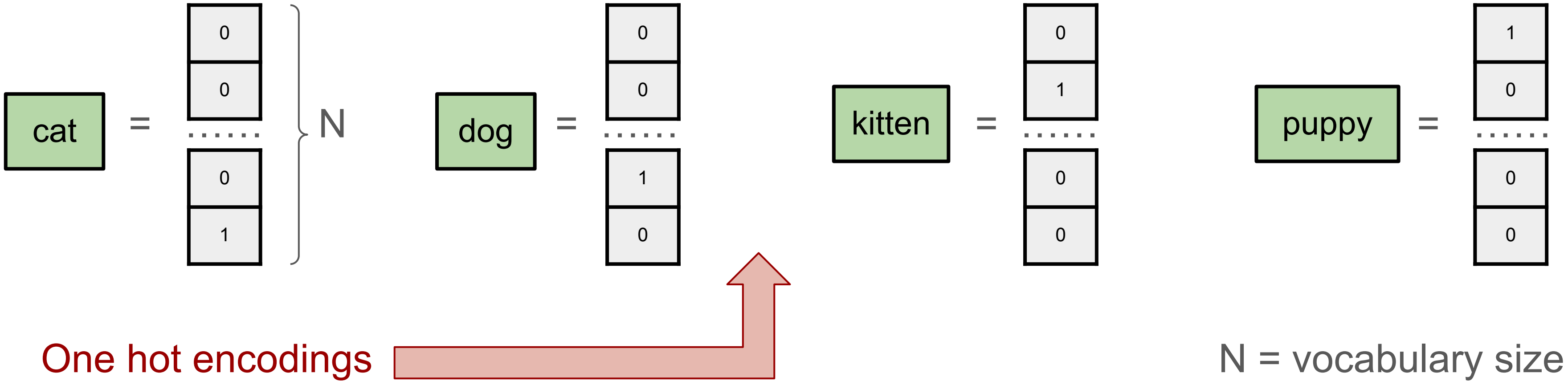
play	played	playing	ask	asked	asking	call	called	calling	burn	burned	burning

vs.

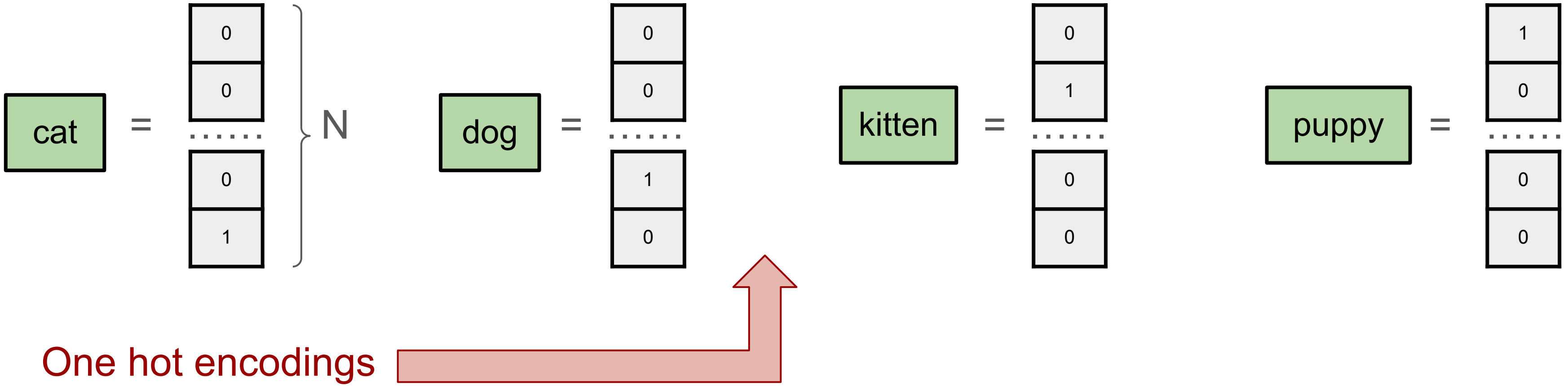
play	ask	call	burn	#ed	#ing

- Fewer ***Unknown*** words, partial correction of misspellings
- Parts of related words (e.g. play / played) will be represented by the same vector
- Is there a way to encode semantic relations between words, **and** decrease the dimensions of the inputs?

Word embeddings

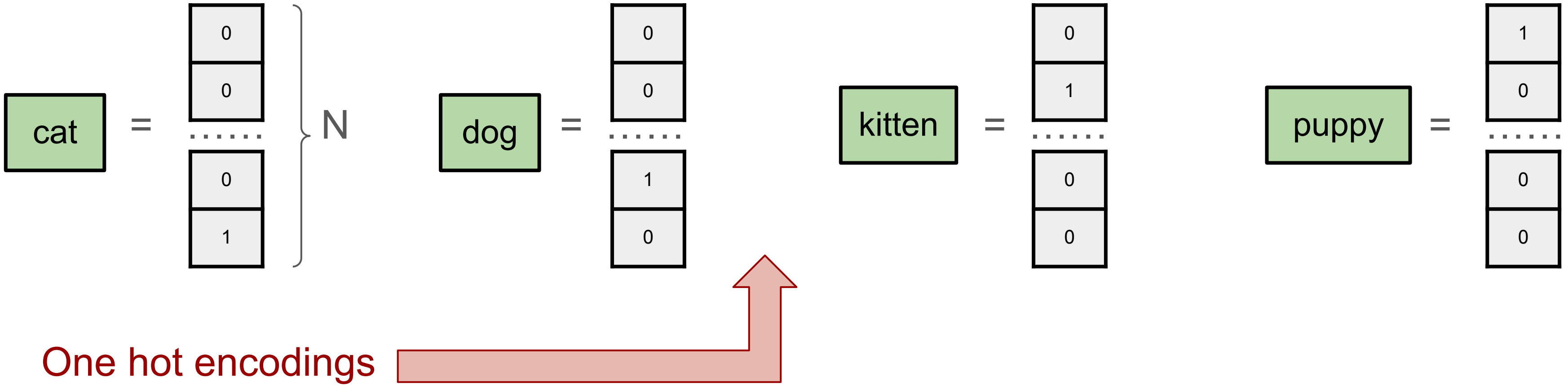


Word embeddings



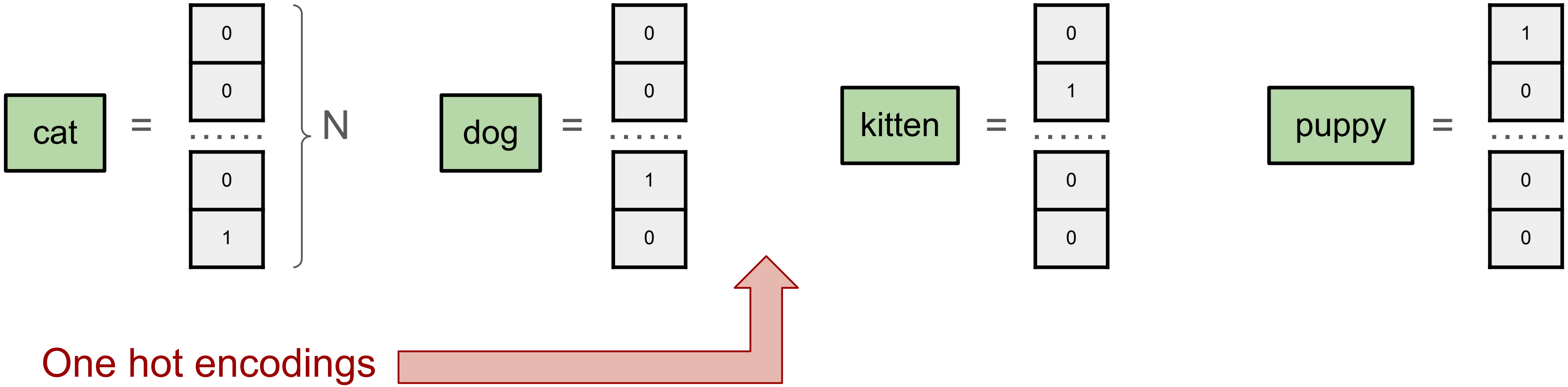
- dog : puppy = cat : kitten. How can we record this?

Word embeddings

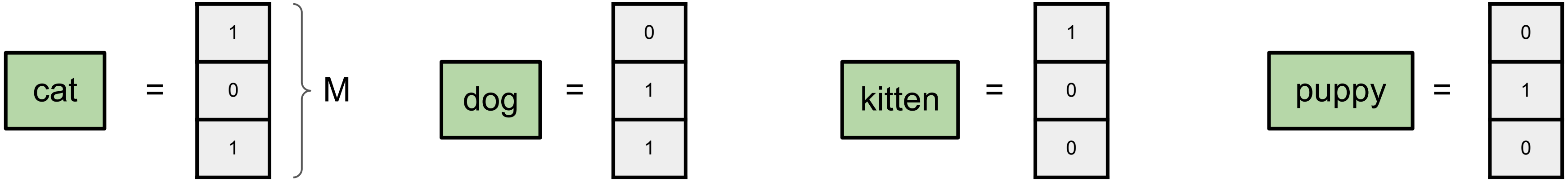


- dog : puppy = cat : kitten. How can we record this?
- Goal: a lower-dimensional representation of a word (M << N)

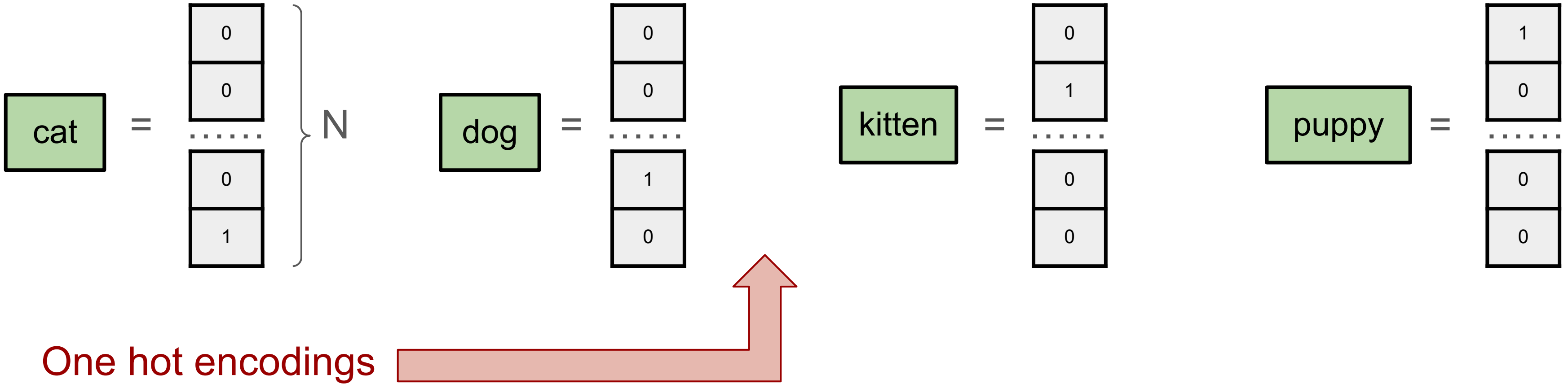
Word embeddings



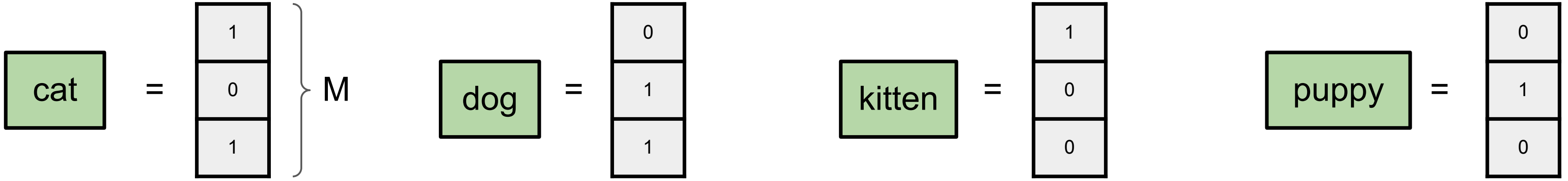
- dog : puppy = cat : kitten. How can we record this?
- Goal: a lower-dimensional representation of a word (M << N)



Word embeddings



- dog : puppy = cat : kitten. How can we record this?
- Goal: a lower-dimensional representation of a word (M << N)

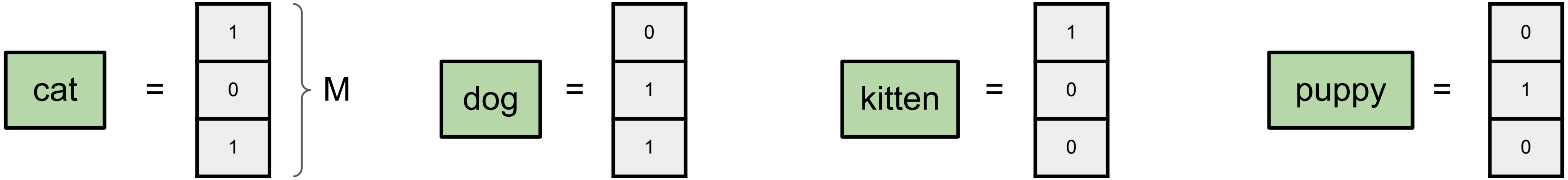


- Words are forced to “group together” in a meaningful way

Word embeddings

What do you mean, “meaningful way”?

- dog : puppy = cat : kitten. How can we record this?
- Goal: a lower-dimensional representation of a word (M << N)



- Words are forced to “group together” in a meaningful way

Word embeddings

What do you mean, “meaningful way”?

...	1 for feline
...	1 for canine
...	1 for adult / 0 for baby

- dog : puppy = cat : kitten. How can we record this?
- Goal: a lower-dimensional representation of a word (M << N)

$$\text{cat} = \left.\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}\right\} M \qquad \text{dog} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \qquad \text{kitten} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \qquad \text{puppy} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

- Words are forced to “group together” in a meaningful way

Word embeddings

What do you mean, “meaningful way”?

...	1 for feline
...	1 for canine
...	1 for adult / 0 for baby

dog - puppy = cat - kitten

- dog : puppy = cat : kitten. How can we record this?
- Goal: a lower-dimensional representation of a word (M << N)

$$\text{cat} = \left.\begin{bmatrix}1\\0\\1\end{bmatrix}\right\} M \qquad \text{dog} = \begin{bmatrix}0\\1\\1\end{bmatrix} \qquad \text{kitten} = \begin{bmatrix}1\\0\\0\end{bmatrix} \qquad \text{puppy} = \begin{bmatrix}0\\1\\0\end{bmatrix}$$

- Words are forced to “group together” in a meaningful way

Word embeddings

How do we arrive at this representation?

Word embeddings

How do we arrive at this representation?

- Pass the one-hot vector through a network with a bottleneck layer

Word embeddings

How do we arrive at this representation?

- Pass the one-hot vector through a network with a bottleneck layer
  = multiply the N-dim vector by NxM matrix to get an M-dim vector

$$\underbrace{\begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{pmatrix}}_{\text{Embedding matrix W}} \times \overset{\text{One-hot vector}}{\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}} = \underset{\text{Embedded vector}}{\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}}$$

Word embeddings

How do we arrive at this representation?

- Pass the one-hot vector through a network with a bottleneck layer
  = multiply the N-dim vector by NxM matrix to get an M-dim vector

One-hot vector

$$\underbrace{\begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{pmatrix}}_{\text{Embedding matrix W}} \times \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Embedded vector

- Some information gets lost in the process

Word embeddings

How do we arrive at this representation?

- Pass the one-hot vector through a network with a bottleneck layer = multiply the N-dim vector by NxM matrix to get an M-dim vector

$$\underbrace{\begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{pmatrix}}_{\text{Embedding matrix W}} \times \overset{\text{One-hot vector}}{\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}} = \underset{\text{Embedded vector}}{\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}}$$

- Some information gets lost in the process
- Embedding matrix: through machine learning!

Word embeddings

How do we arrive at this representation?

- Pass the one-hot vector through a network with a bottleneck layer
  = multiply the N-dim vector by NxM matrix to get an M-dim vector

One-hot vector

$$\underbrace{\begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{pmatrix}}_{\text{Embedding matrix W}} \times \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Embedded vector

In practice:

- Embedding can be part of your model that you train for a ML task
- Or: use pre-trained embeddings (e.g. Word2Vec, GloVe, etc)

- Some information gets lost in the process
- Embedding matrix: through machine learning!

Further reading

Tokenization:

blog.floydhub.com/tokenization-nlp/

Word embeddings:

jalammar.github.io/illustrated-word2vec/

Outline

1. NLP 101: how do we represent words in machine learning

   🤖 Tokenization 🤖 Word embeddings

2. Order matters: Recurrent Neural Networks

   🤖 RNNs 🤖 Seq2Seq models

   🤖 What is wrong with RNNs

3. Attention is all you need!

   🤖 What is Attention 🤖 Attention, the linear algebra perspective

4. The Transformer architecture

   🤖 Mostly Attention

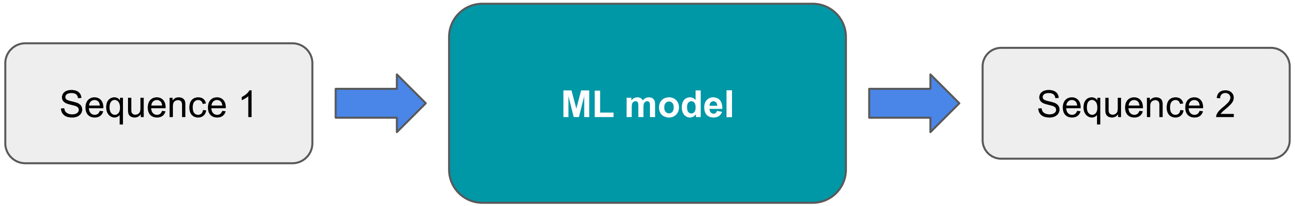
5. BERT: *contextualized* word embeddings

Sequence to Sequence models

- Not all seq2seq models are NLP
- Not all NLP models are seq2seq!
- But: they are common enough

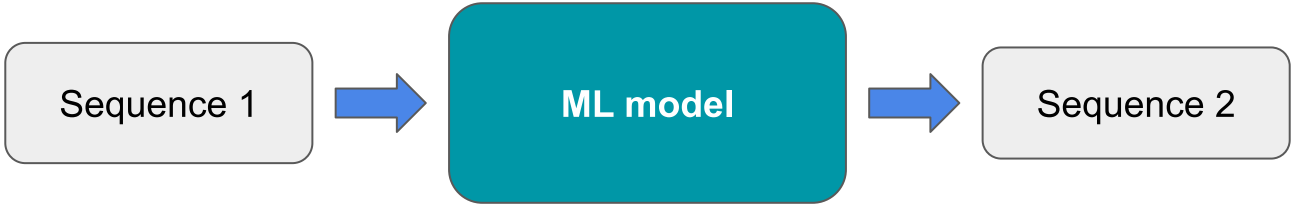
Sequence to Sequence models

- Not all seq2seq models are NLP
- Not all NLP models are seq2seq!
- But: they are common enough

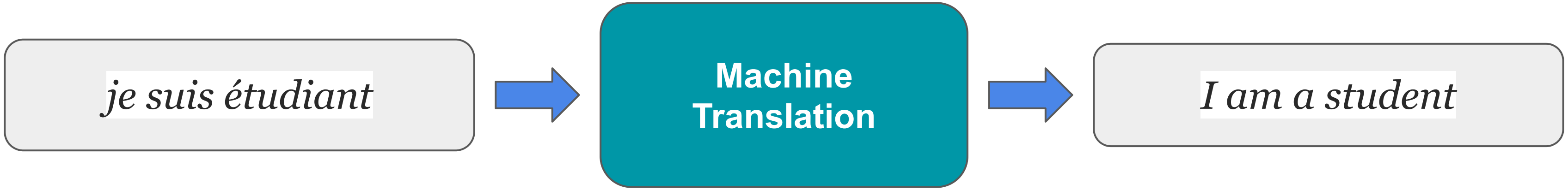


Sequence to Sequence models

- Not all seq2seq models are NLP
- Not all NLP models are seq2seq!
- But: they are common enough



- Language modeling, question answering, machine translation, etc



Sequence to Sequence models

- Not all seq2seq models are NLP
- Not all NLP models are seq2seq!
- But: they are common enough

Takeaways:

1. **Multiple elements in a sequence**
2. **Order of the elements matters**

Sequence 1 → **ML model** → Sequence 2

- Language modeling, question answering, machine translation, etc

je suis étudiant → **Machine Translation** → *I am a student*

RNNs: Recurrent Neural Networks

Recurrent neural network

From Wikipedia, the free encyclopedia

Not to be confused with recursive neural network.

A **recurrent neural network** (**RNN**) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Derived from

RNNs: Recurrent Neural Networks

Recurrent neural network

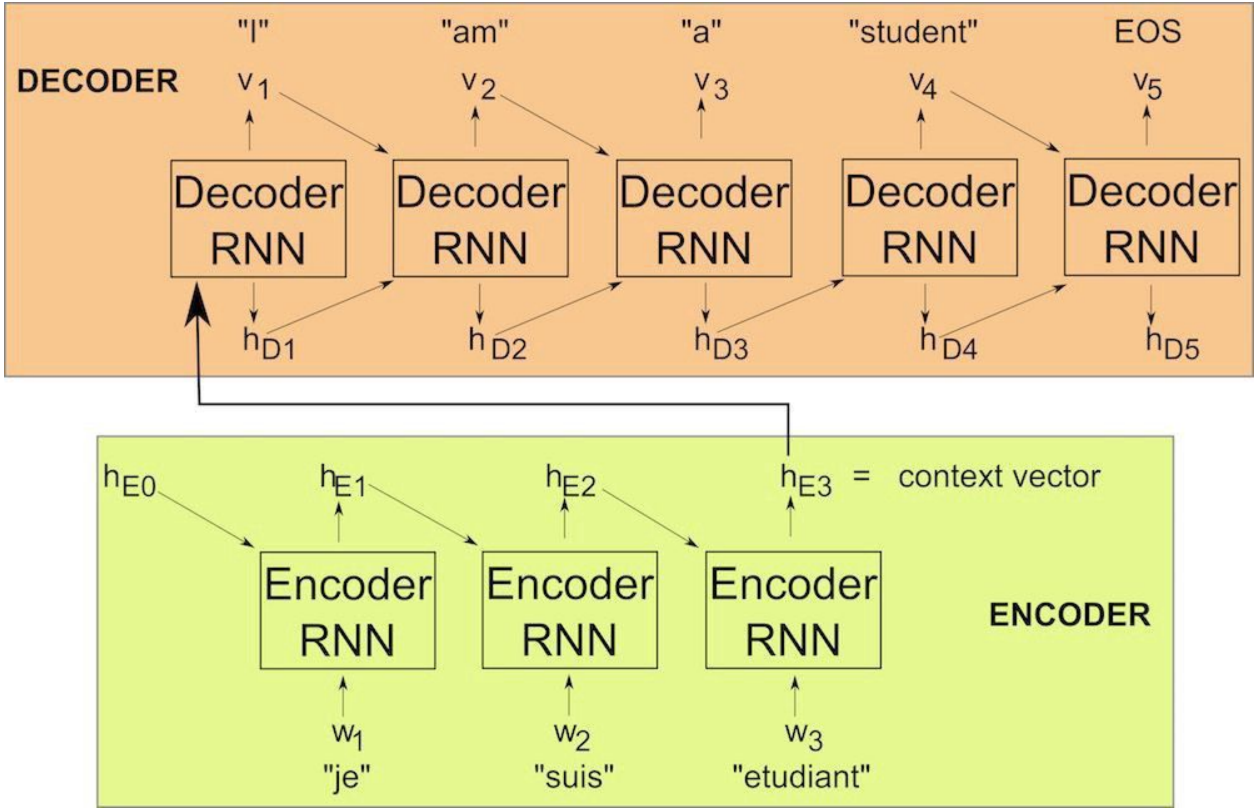
From Wikipedia, the free encyclopedia

Not to be confused with recursive neural network.

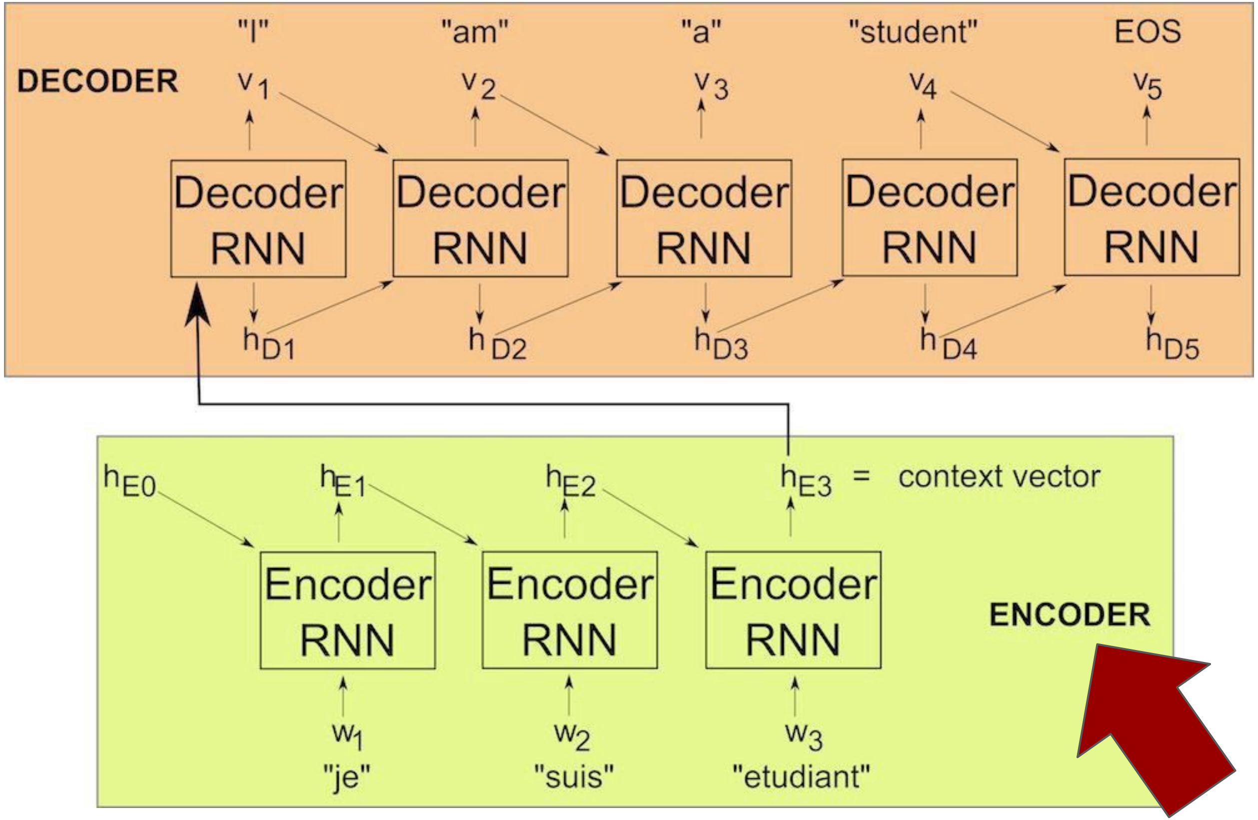
A **recurrent neural network** (**RNN**) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Derived from

What does this mean???

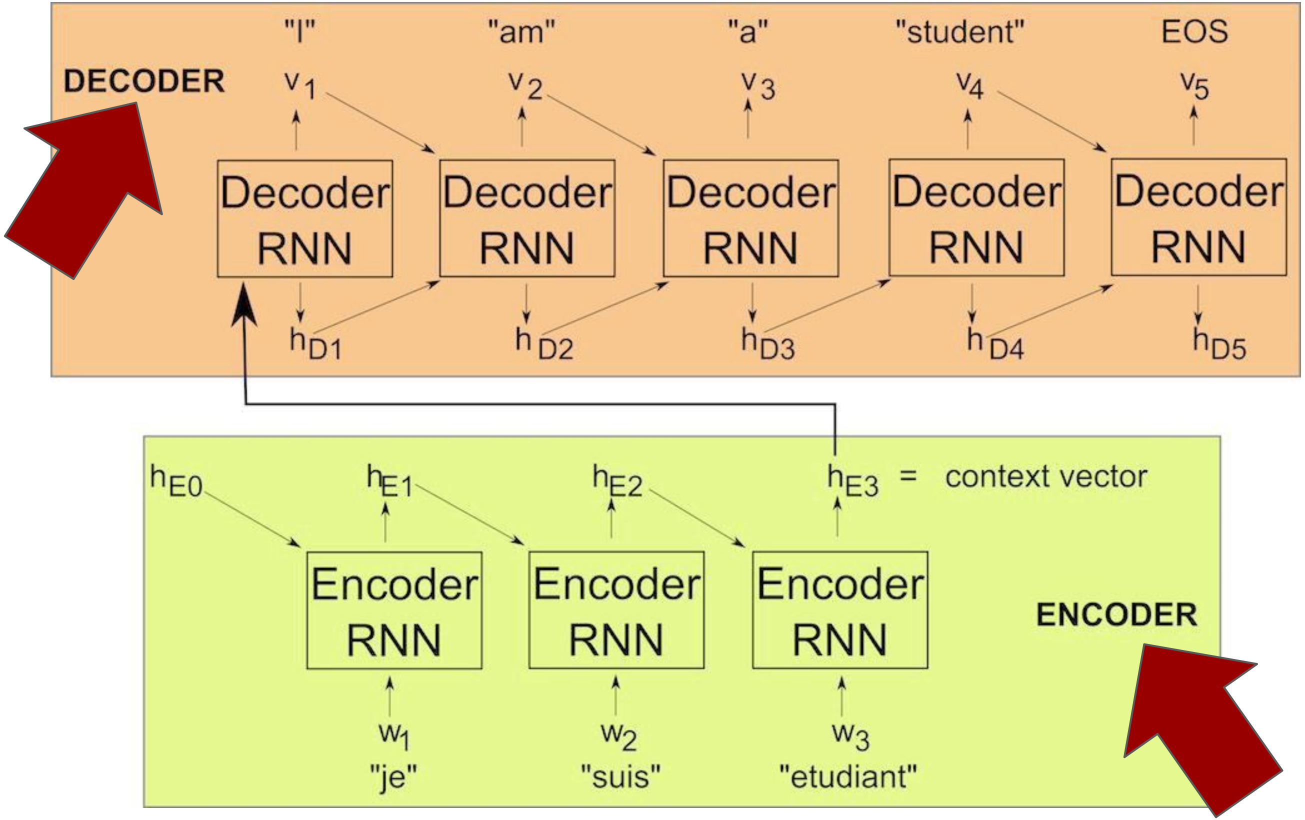
RNNs: the seq2seq example



RNNs: the seq2seq example

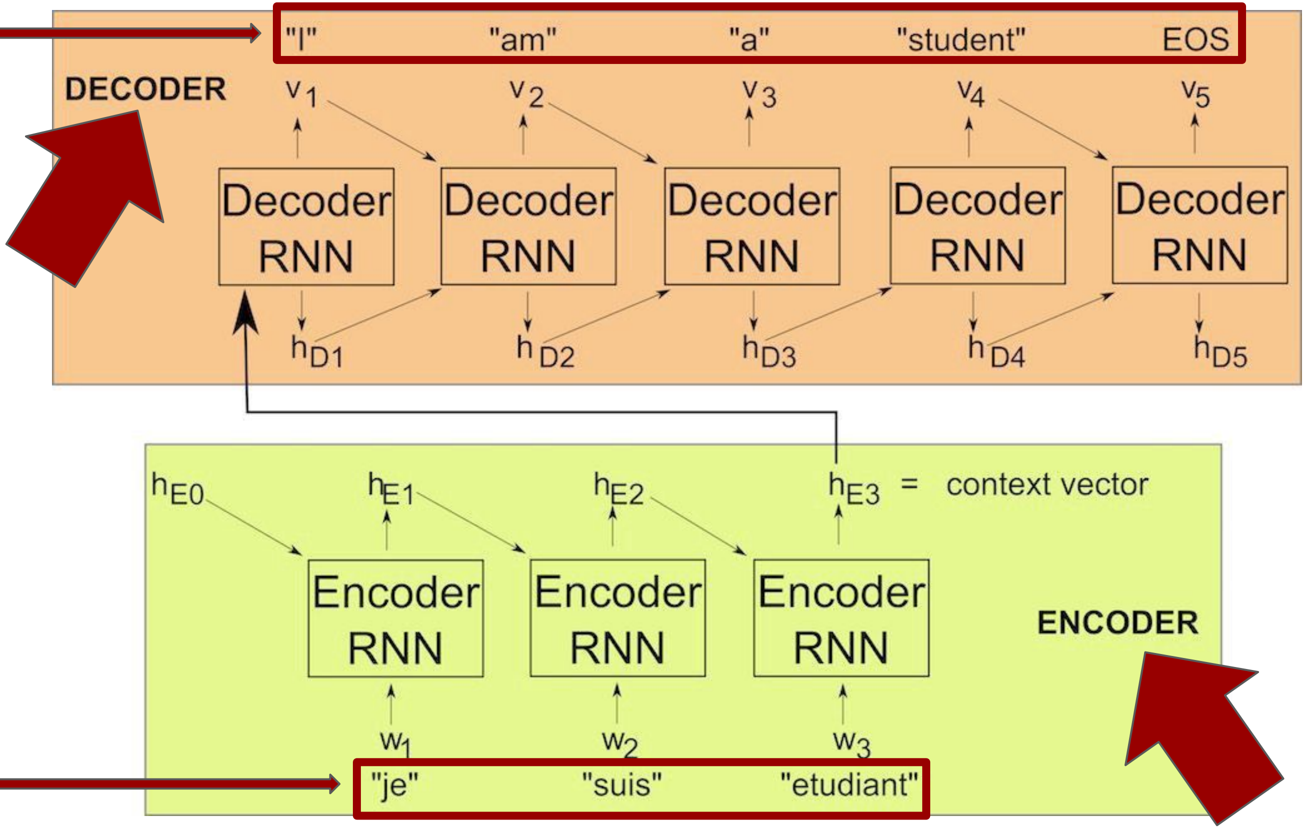


RNNs: the seq2seq example



RNNs: the seq2seq example

Output sequence

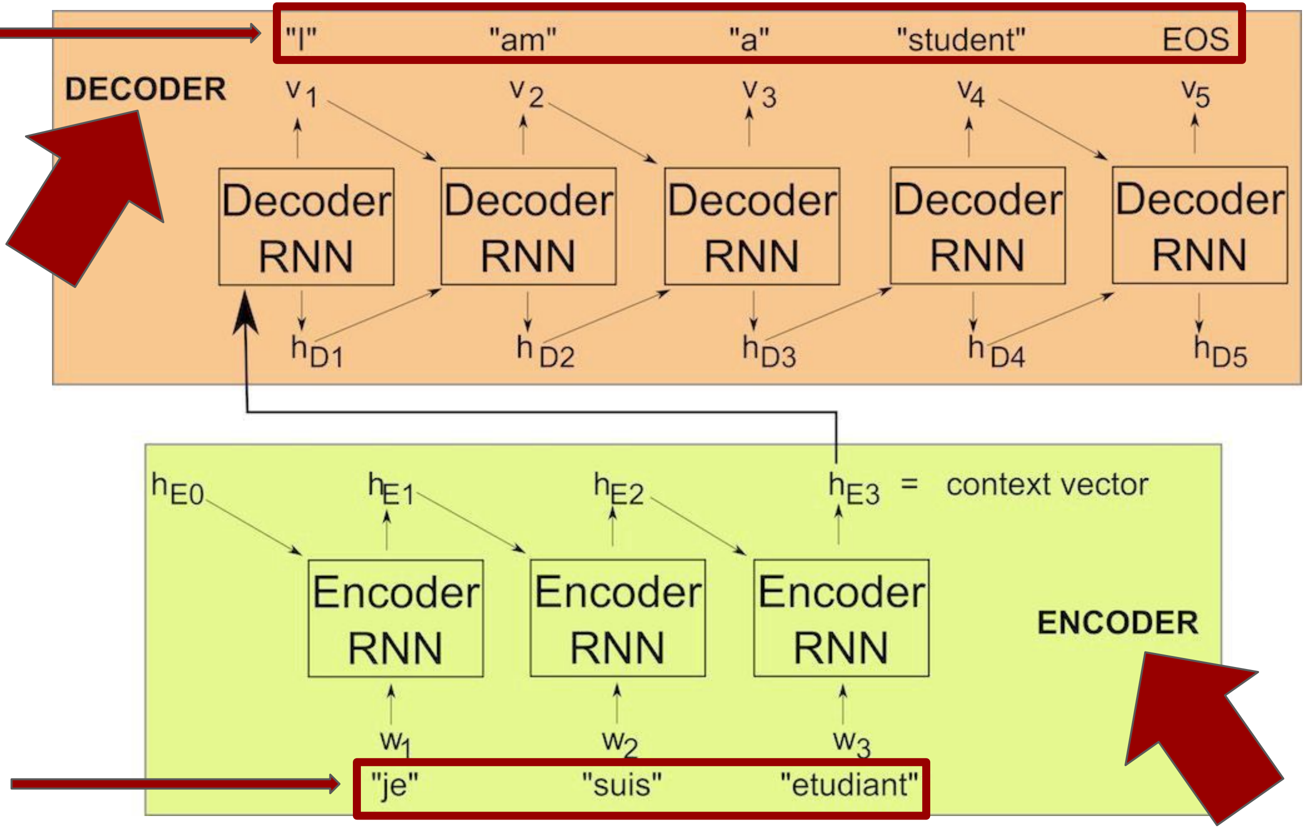


Input sequence

RNNs: the seq2seq example

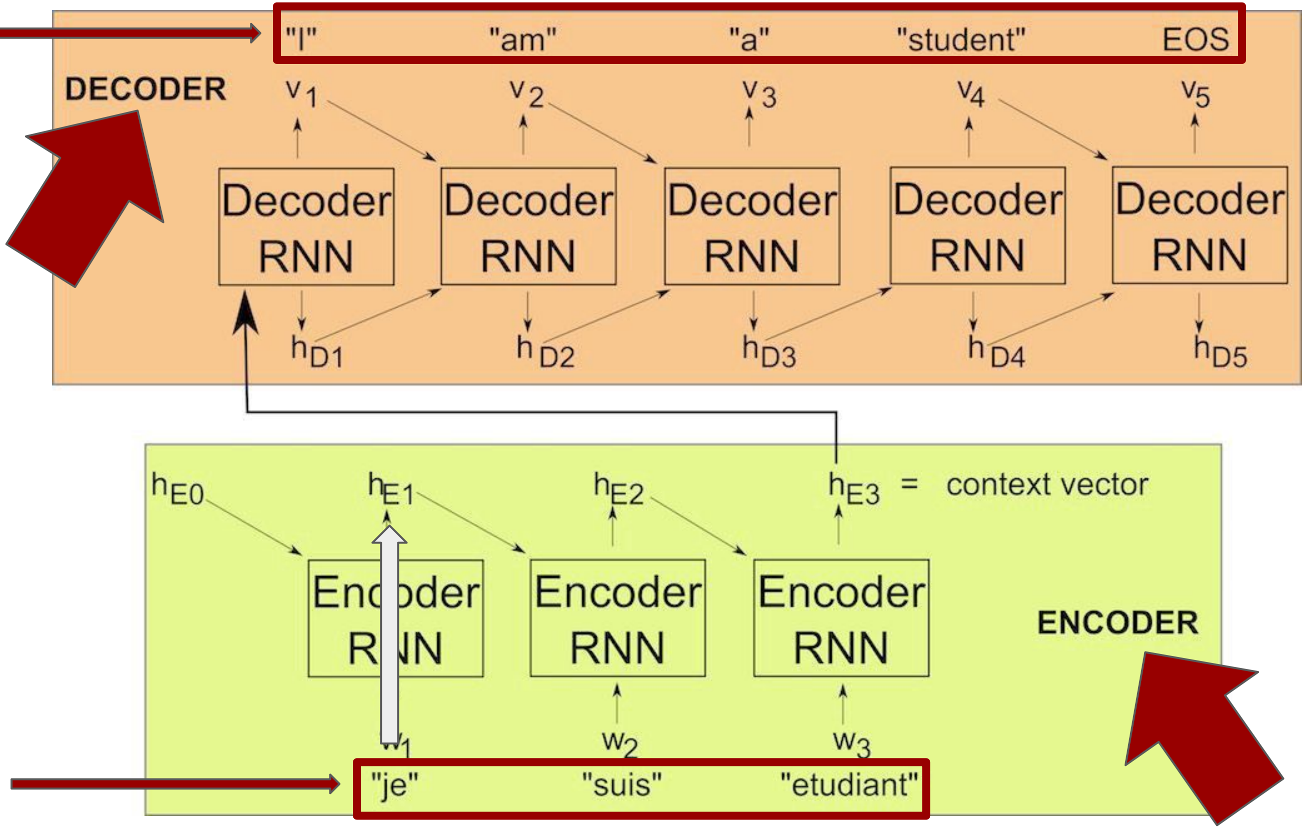
Outputs (one-hot)

Inputs (word embeddings)



RNNs: the seq2seq example

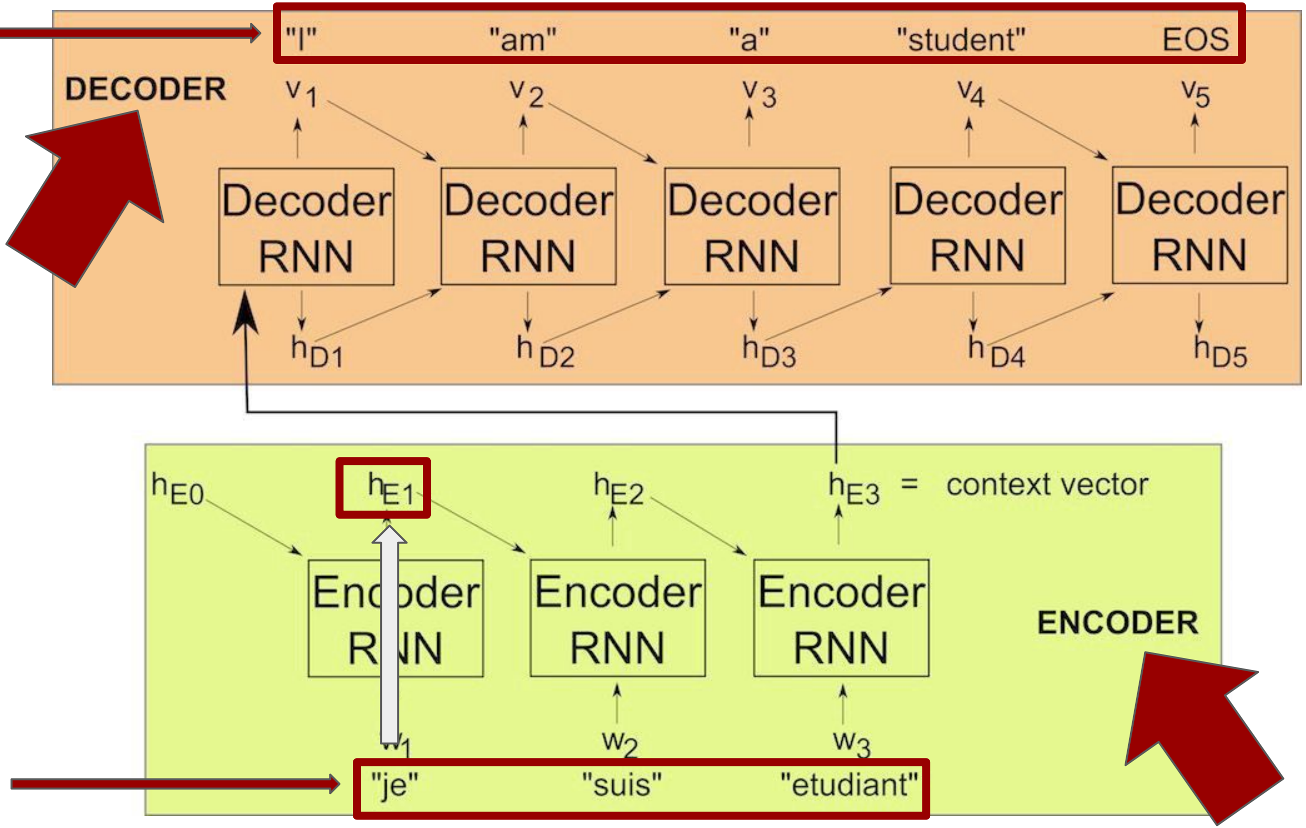
Outputs (one-hot)



Inputs (word embeddings)

RNNs: the seq2seq example

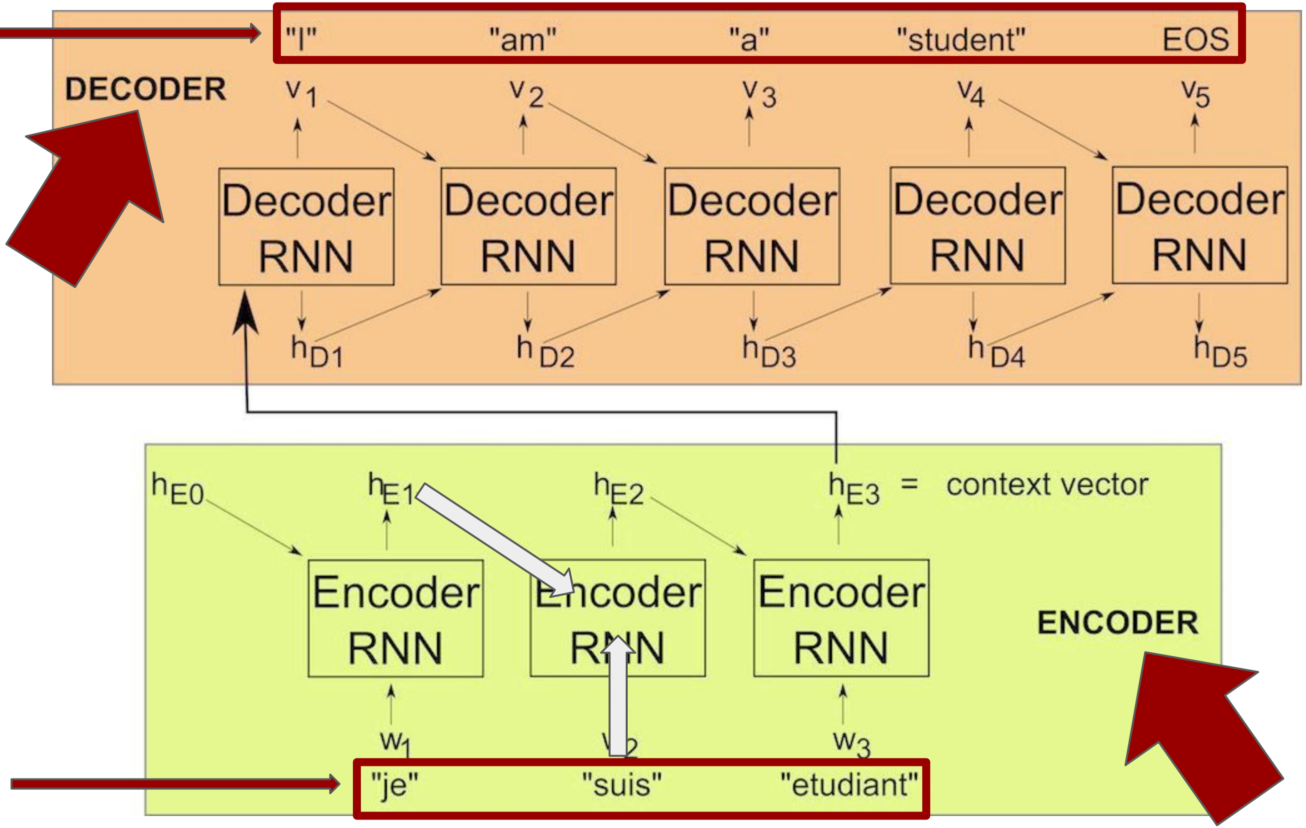
Outputs (one-hot)



Inputs (word embeddings)

RNNs: the seq2seq example

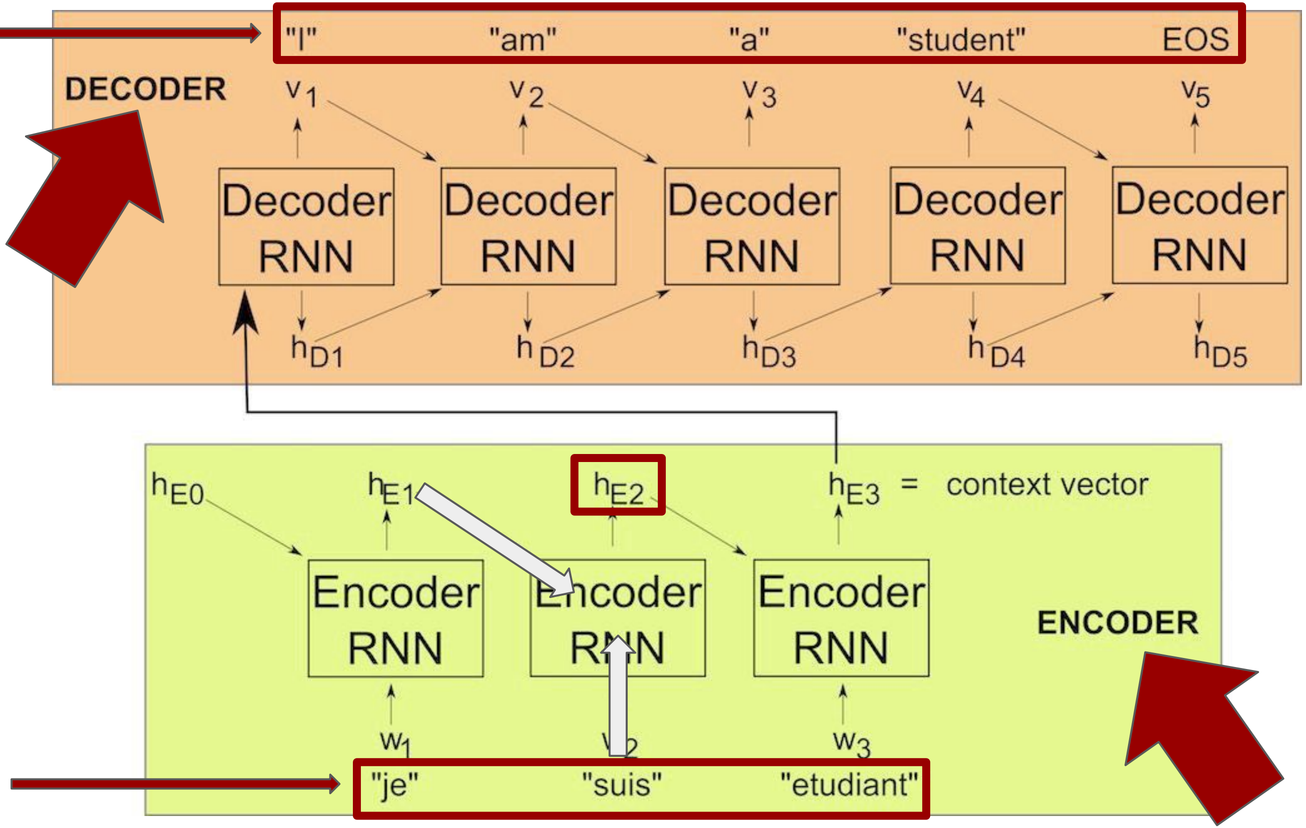
Outputs (one-hot)



Inputs (word embeddings)

RNNs: the seq2seq example

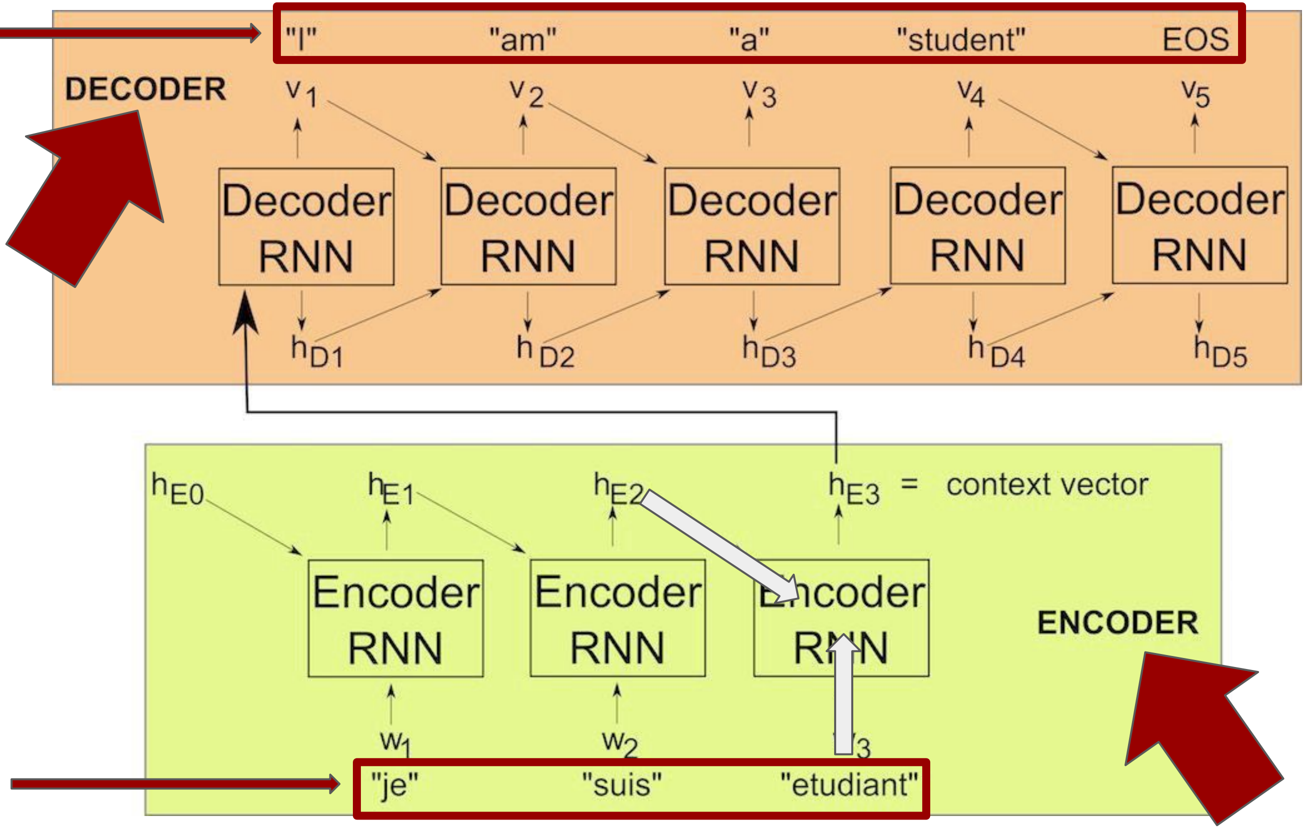
Outputs (one-hot)



Inputs (word embeddings)

RNNs: the seq2seq example

Outputs (one-hot)



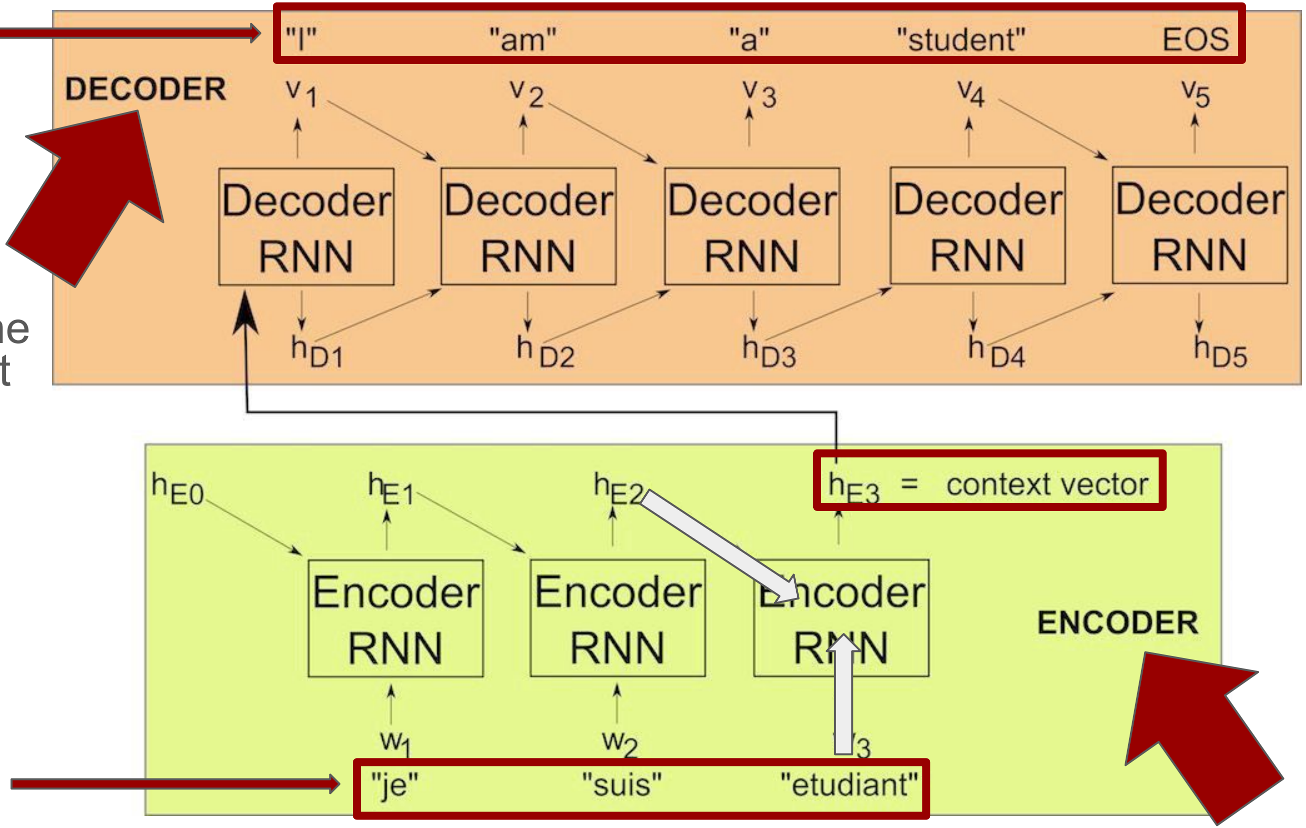
Inputs (word embeddings)

RNNs: the seq2seq example

Outputs (one-hot)

Context vector: contains the meaning of the whole input sequence

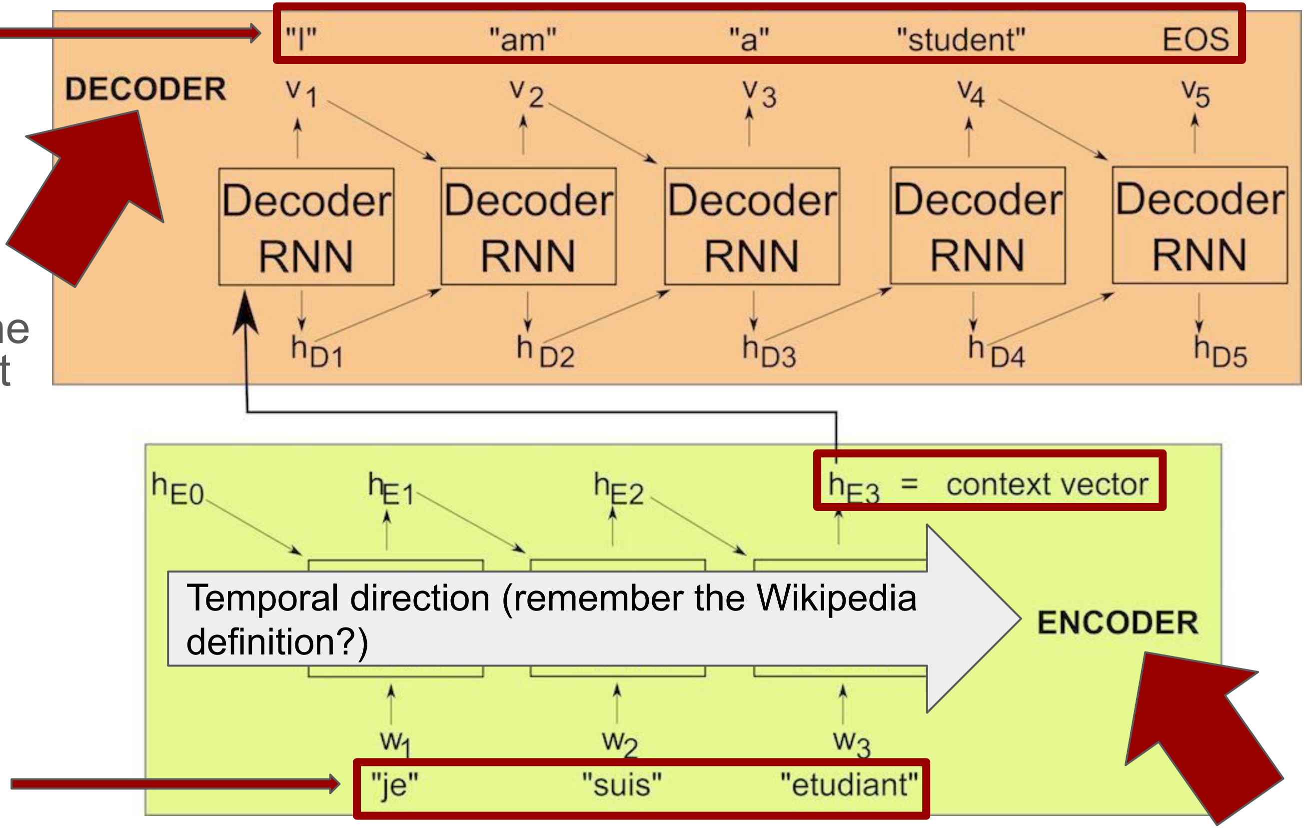
Inputs (word embeddings)



RNNs: the seq2seq example

Outputs (one-hot)

Context vector: contains the meaning of the whole input sequence



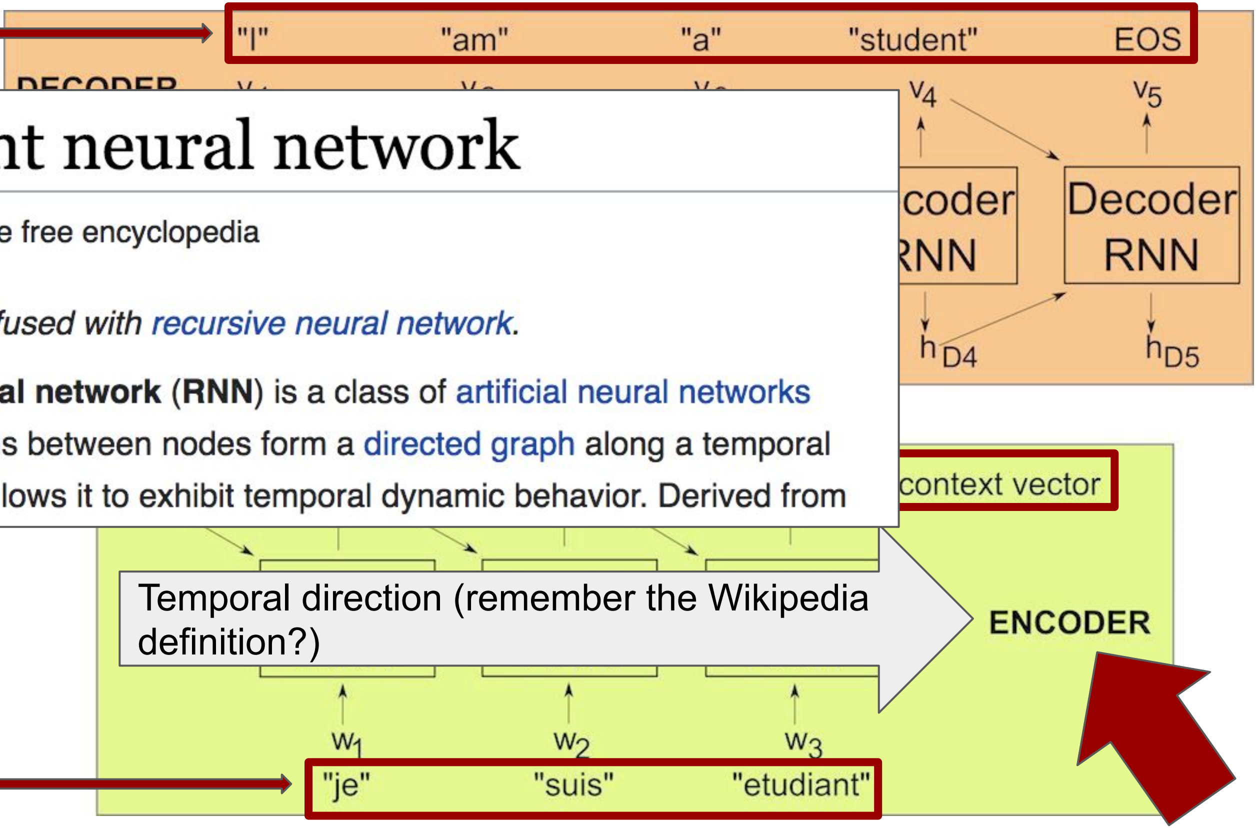
Inputs (word embeddings)

RNNs: the seq2seq example

Outputs (one-hot)

Context vecto
meaning of th
sequence

Inputs (word embeddings)



Recurrent neural network

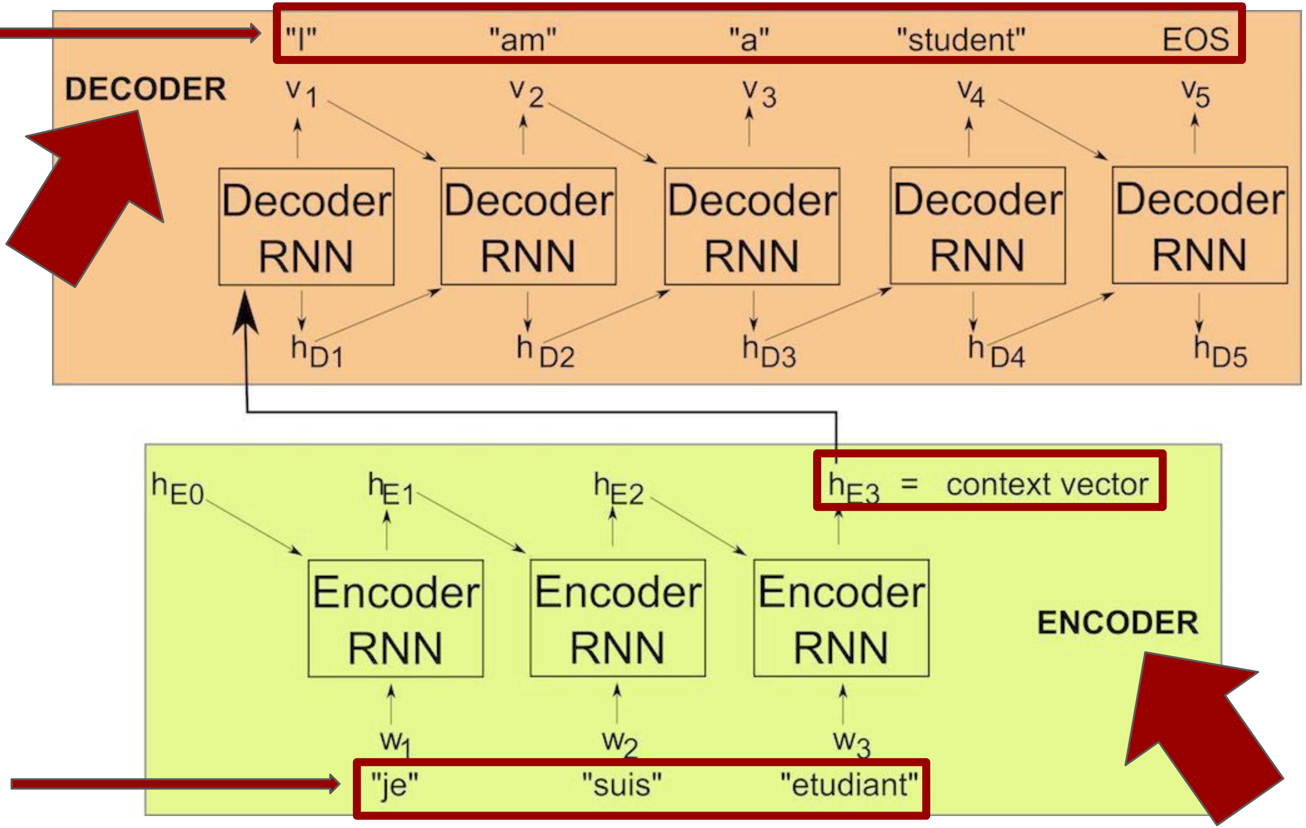
From Wikipedia, the free encyclopedia

Not to be confused with recursive neural network.

A **recurrent neural network** (**RNN**) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Derived from

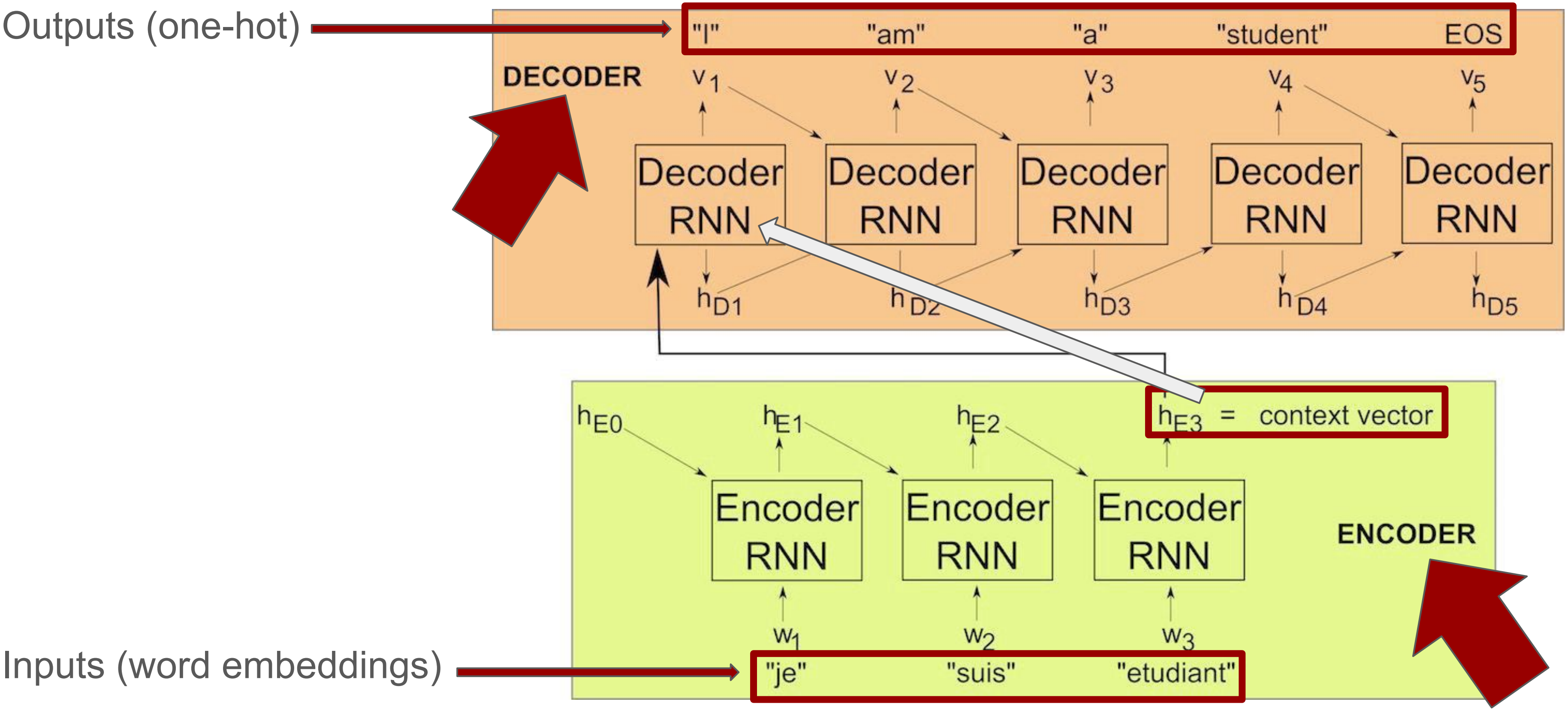
RNNs: the seq2seq example

Outputs (one-hot)



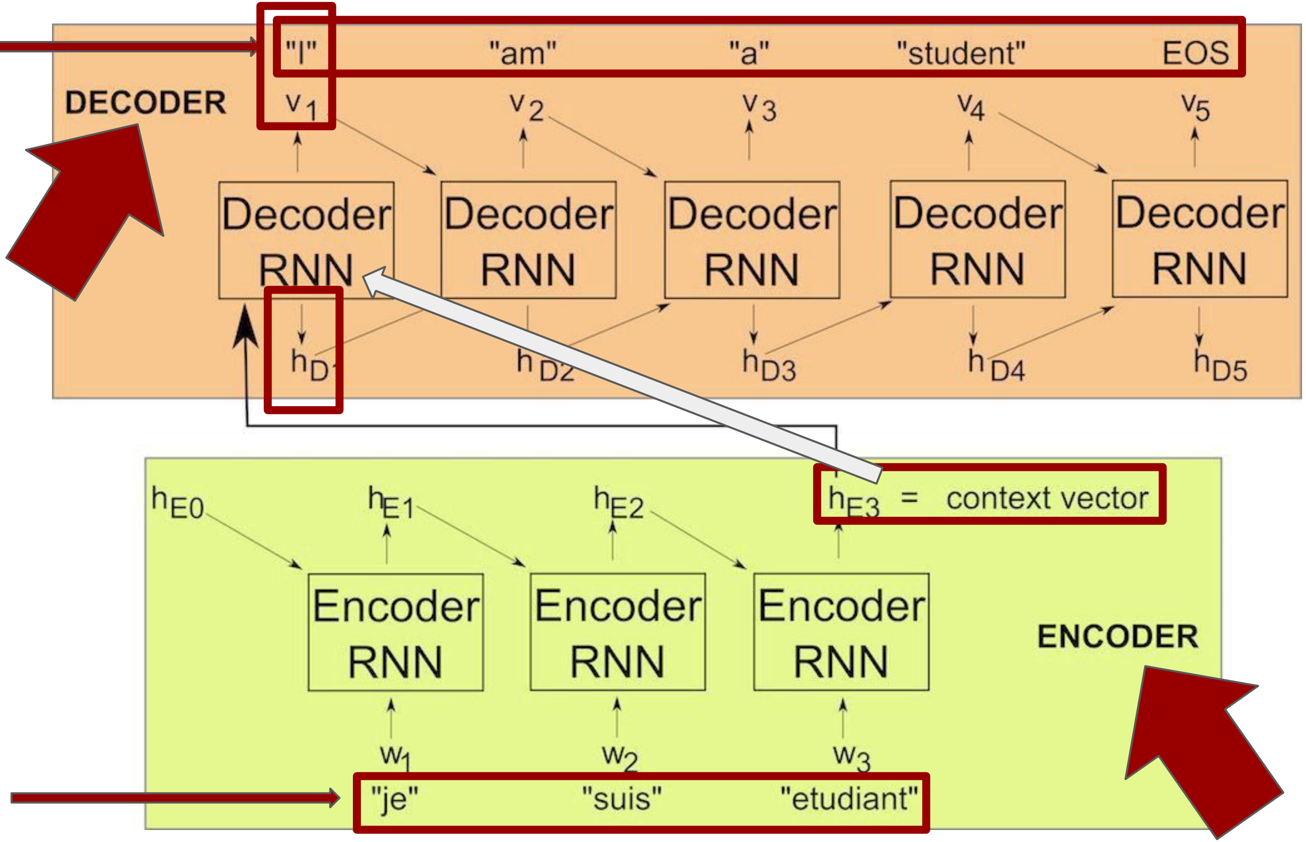
Inputs (word embeddings)

RNNs: the seq2seq example



RNNs: the seq2seq example

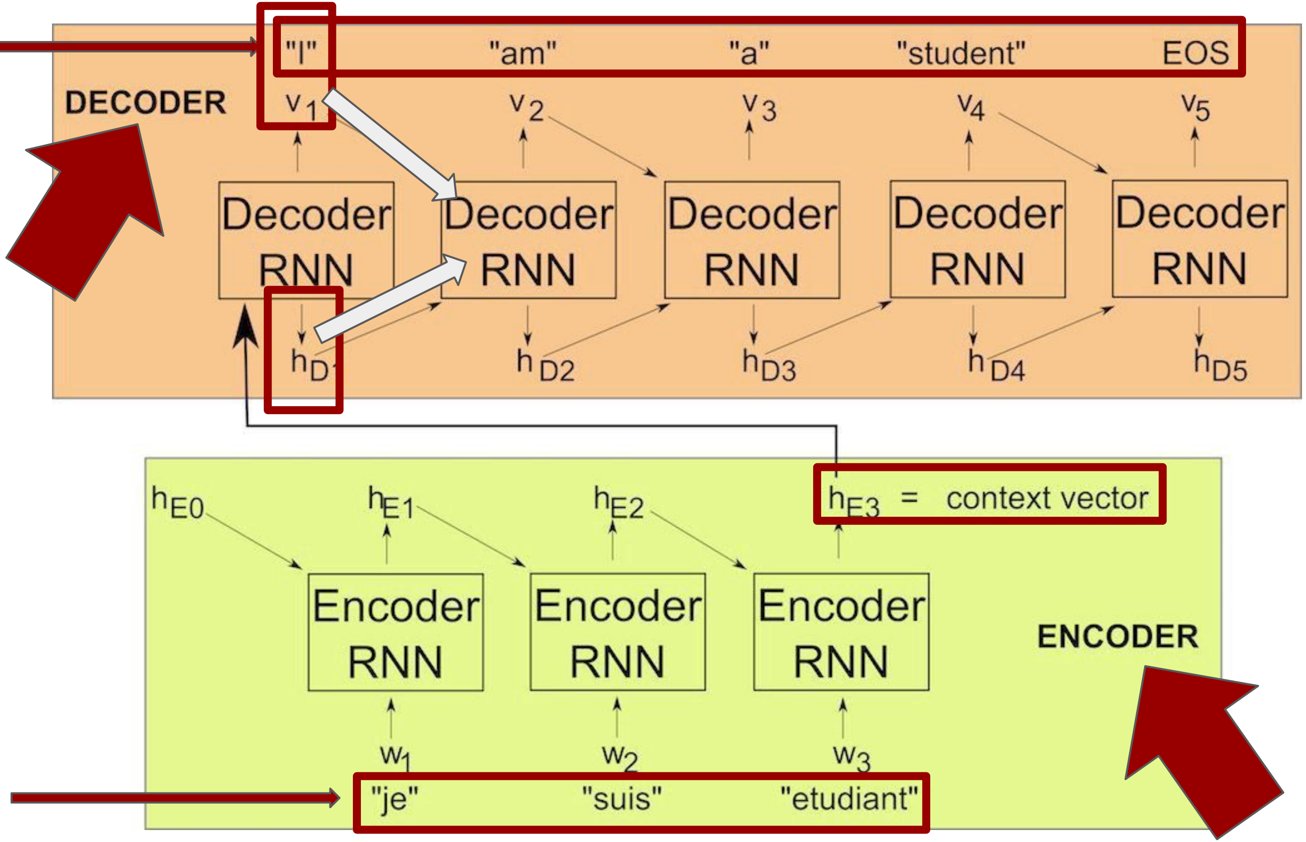
Outputs (one-hot)



Inputs (word embeddings)

RNNs: the seq2seq example

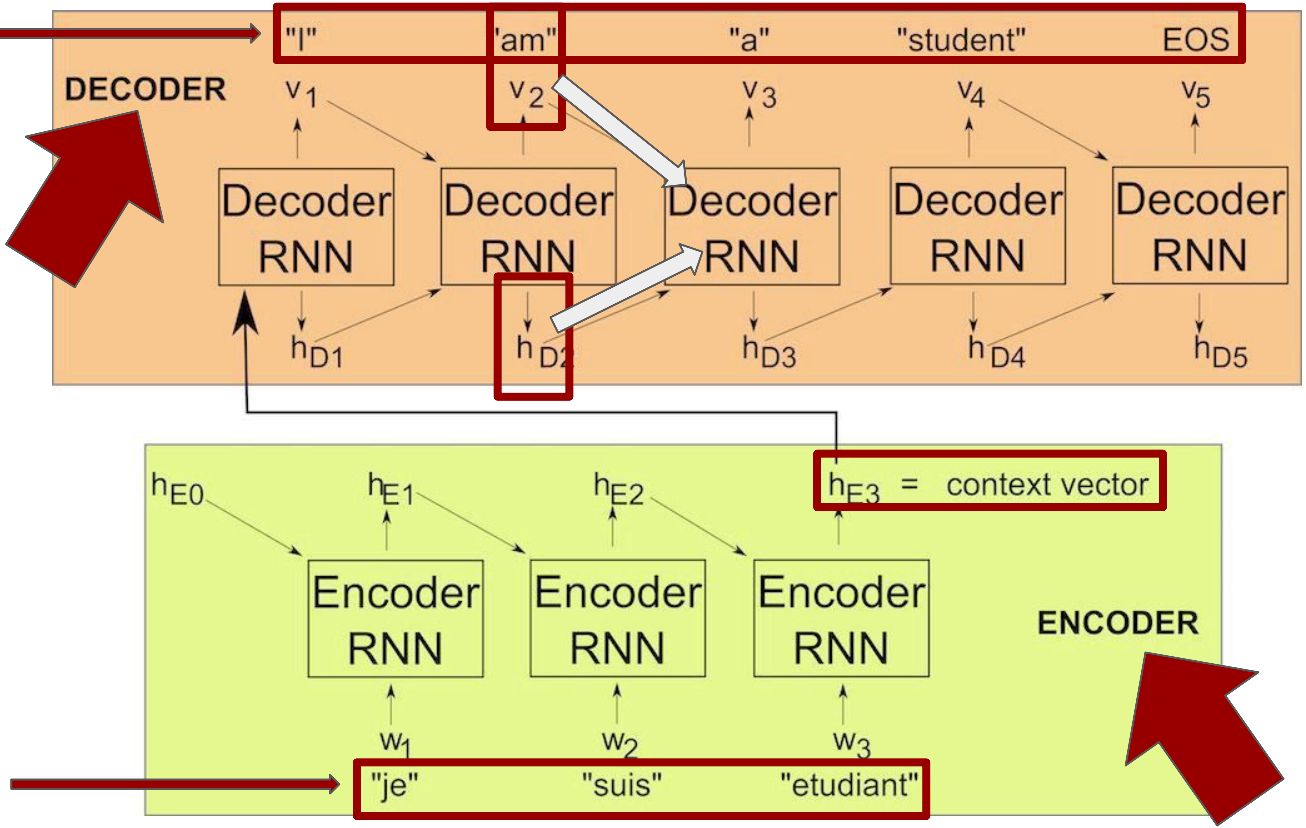
Outputs (one-hot)



Inputs (word embeddings)

RNNs: the seq2seq example

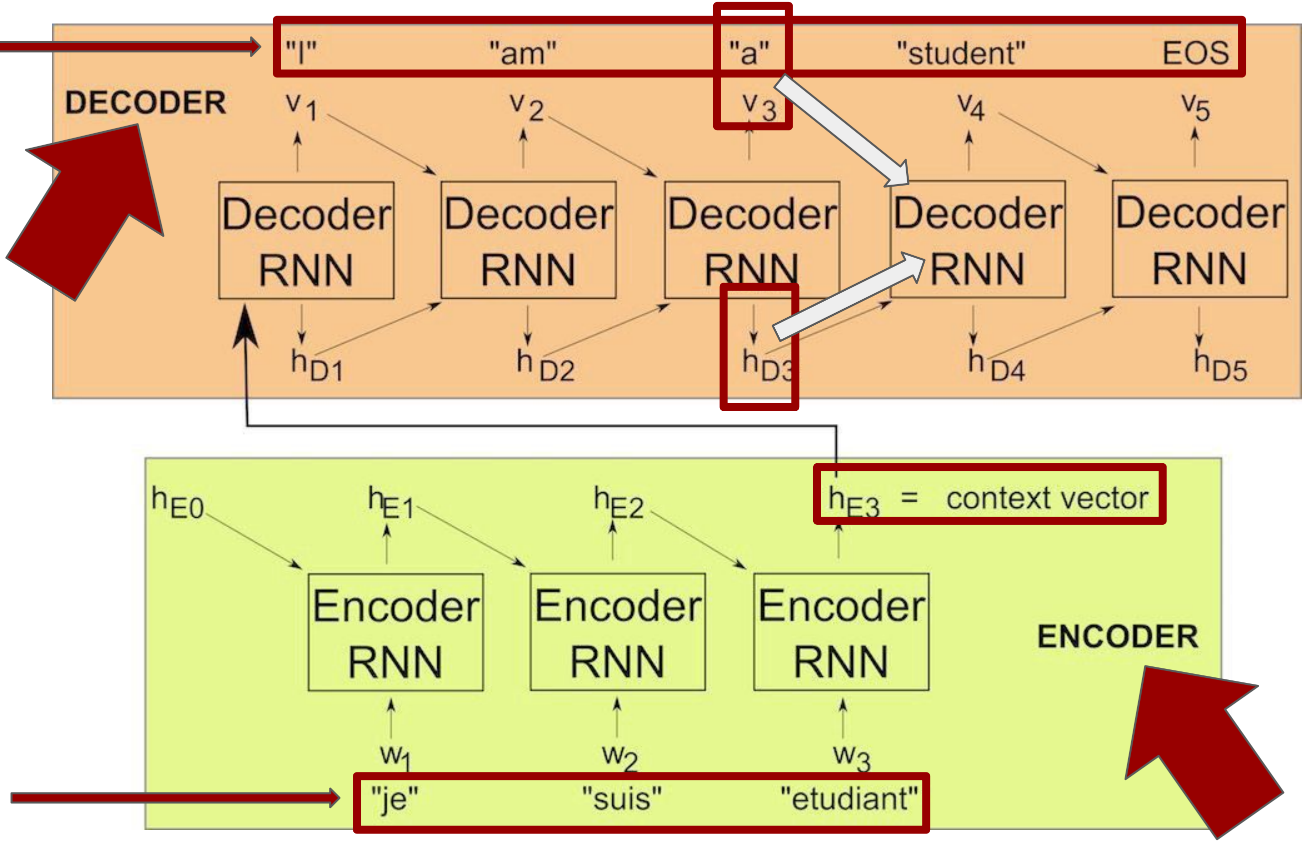
Outputs (one-hot)



Inputs (word embeddings)

RNNs: the seq2seq example

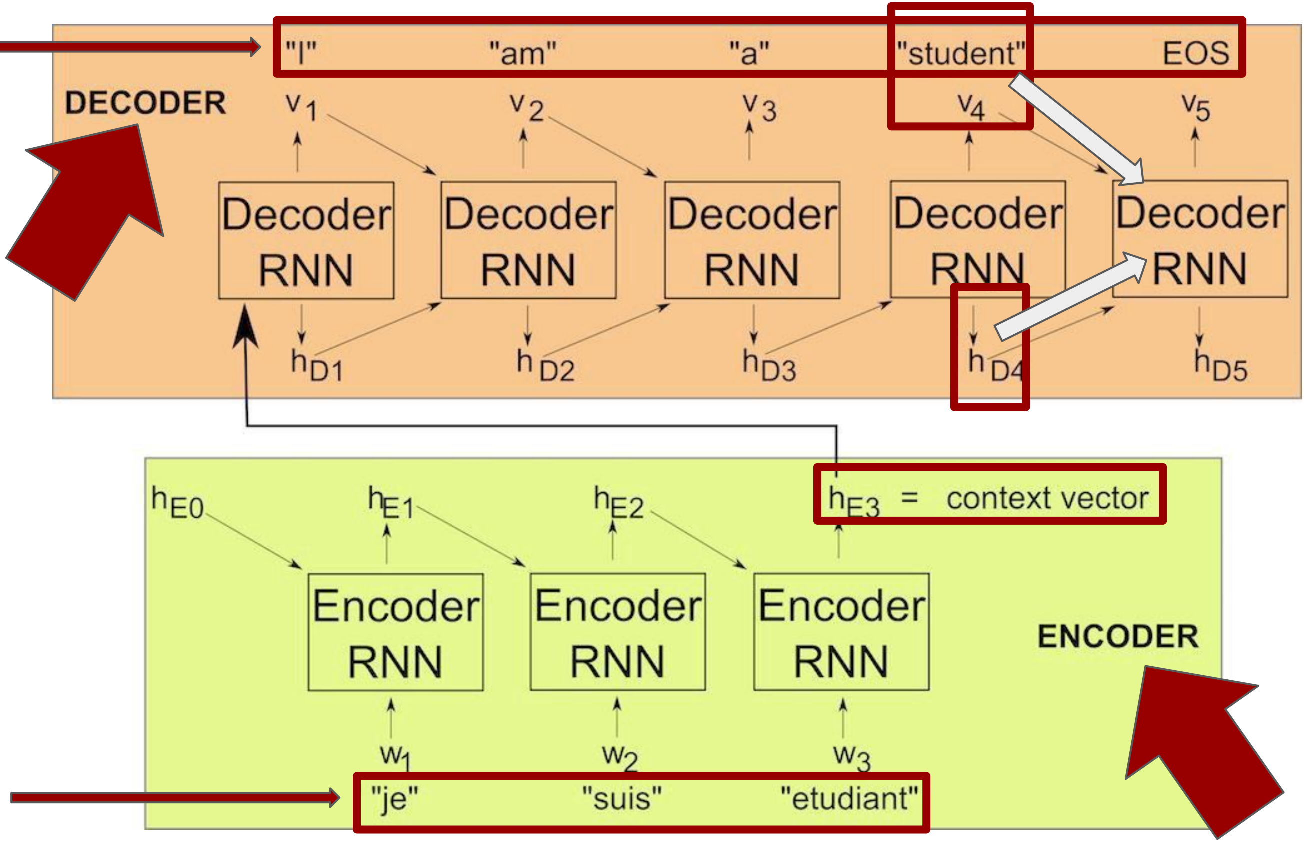
Outputs (one-hot)



Inputs (word embeddings)

RNNs: the seq2seq example

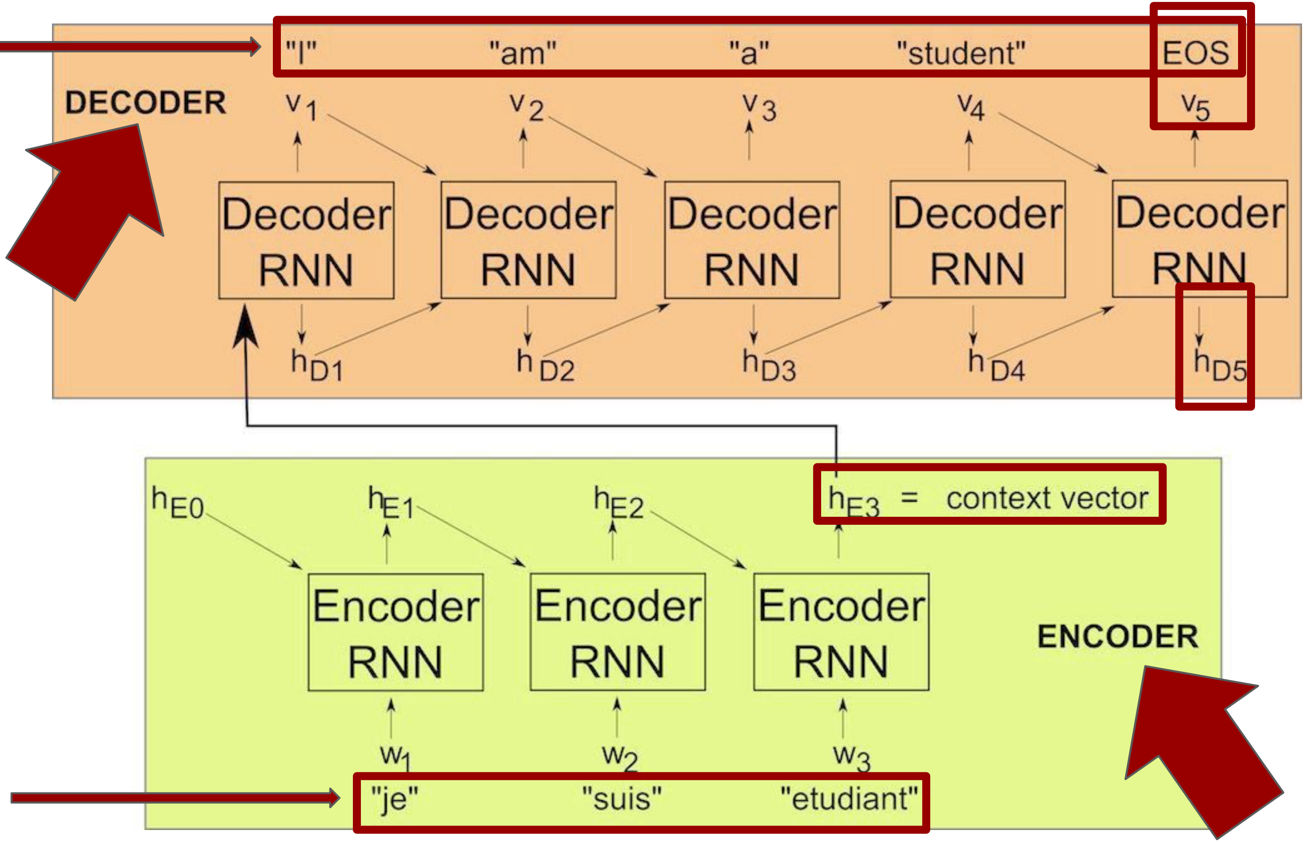
Outputs (one-hot)



Inputs (word embeddings)

RNNs: the seq2seq example

Outputs (one-hot)



Inputs (word embeddings)

Does this really work?

Does this really work?

Yes! (For the most part) Google Translate has been using these since ~ 2016

Does this really work?

Yes! (For the most part) Google Translate has been using these since ~ 2016

What is the catch?

Does this really work?

Yes! (For the most part) Google Translate has been using these since ~ 2016

What is the catch?

- Catch # 1: Recurrent = Non-parallelizable 🐌

Does this really work?

Yes! (For the most part) Google Translate has been using these since ~ 2016

What is the catch?

- Catch # 1: Recurrent = Non-parallelizable 🐌
- Catch # 2: Relying on a fixed-length context vector to capture the full input sequence. What can go wrong?

Does this really work?

Yes! (For the most part) Google Translate has been using these since ~ 2016

What is the catch?

- Catch # 1: Recurrent = Non-parallelizable 🐌
- Catch # 2: Relying on a fixed-length context vector to capture the full input sequence. What can go wrong?

“I am a student.”

Does this really work?

Yes! (For the most part) Google Translate has been using these since ~ 2016

What is the catch?

- Catch # 1: Recurrent = Non-parallelizable 🐌
- Catch # 2: Relying on a fixed-length context vector to capture the full input sequence. What can go wrong?

“I am a student.” ✅

Does this really work?

Yes! (For the most part) Google Translate has been using these since ~ 2016

What is the catch?

- Catch # 1: Recurrent = Non-parallelizable 🐌
- Catch # 2: Relying on a fixed-length context vector to capture the full input sequence. What can go wrong?

“I am a student.” ✅

“It was a wrong number that started it, the telephone ringing three times in the dead of night, and the voice on the other end asking for someone he was not.”

Does this really work?

Yes! (For the most part) Google Translate has been using these since ~ 2016

What is the catch?

- Catch # 1: Recurrent = Non-parallelizable 🐌
- Catch # 2: Relying on a fixed-length context vector to capture the full input sequence. What can go wrong?

“I am a student.” ✅

“It was a wrong number that started it, the telephone ringing three times in the dead of night, and the voice on the other end asking for someone he was not.” 😱

Outline

1. NLP 101: how do we represent words in machine learning

   🤖 Tokenization 🤖 Word embeddings

2. Order matters: Recurrent Neural Networks

   🤖 RNNs 🤖 Seq2Seq models

   🤖 What is wrong with RNNs

3. Attention is all you need!

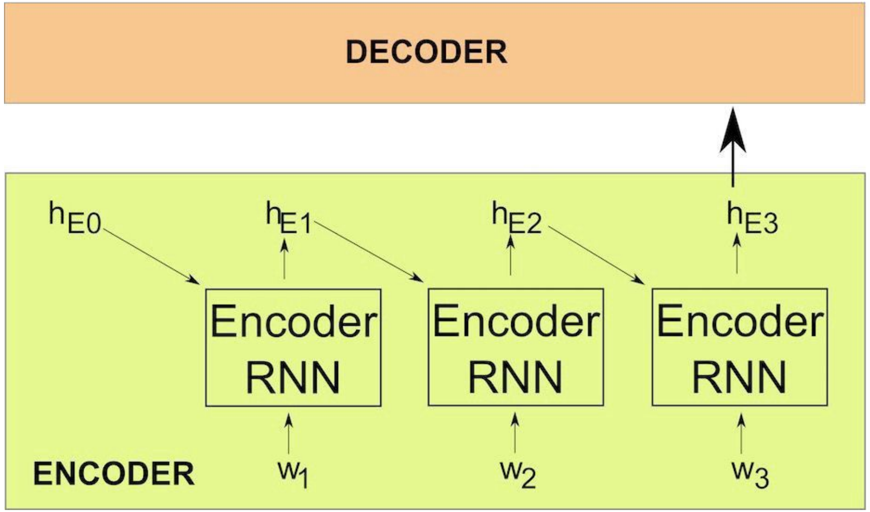
   🤖 What is Attention 🤖 Attention, the linear algebra perspective

4. The Transformer architecture

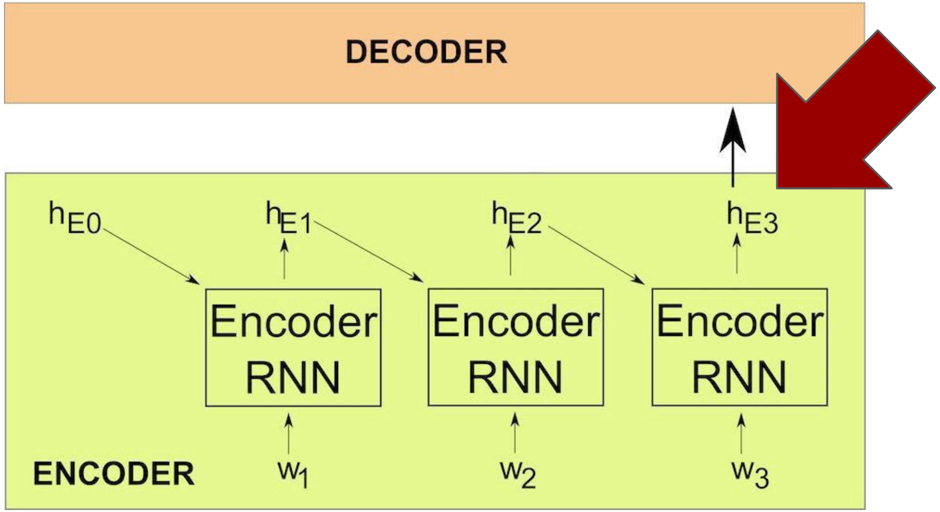
   🤖 Mostly Attention

5. BERT: *contextualized* word embeddings

The Attention Mechanism

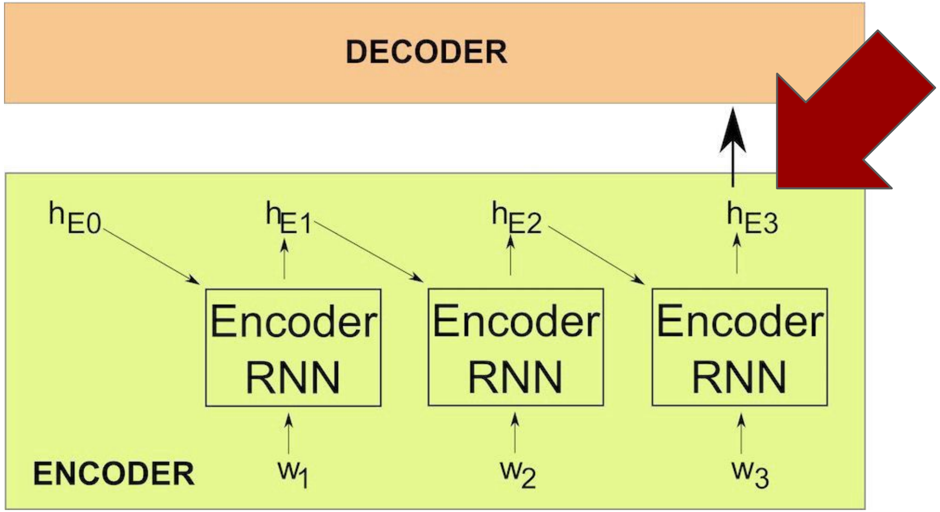


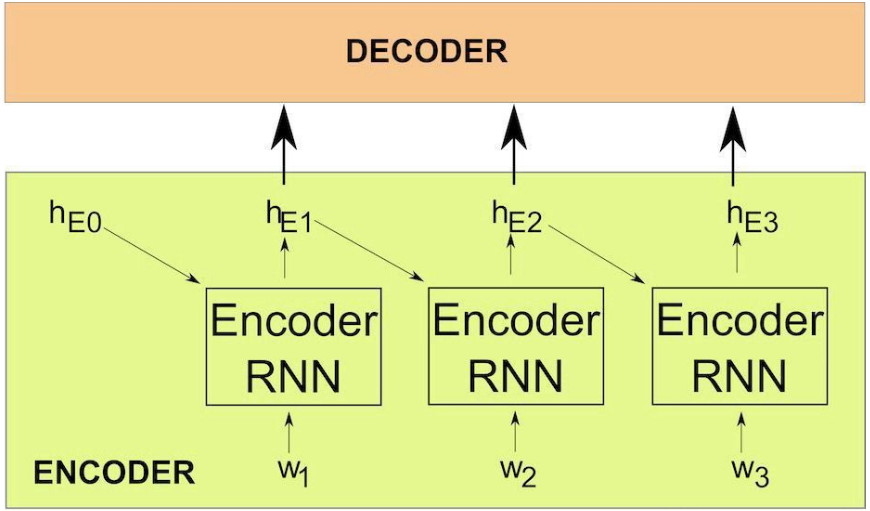
The Attention Mechanism



Recurrent NN from the previous section

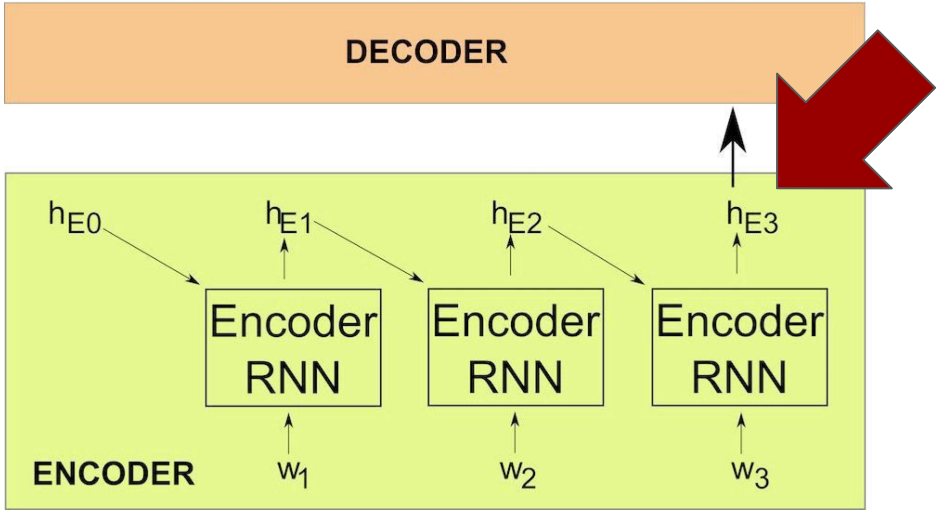
The Attention Mechanism



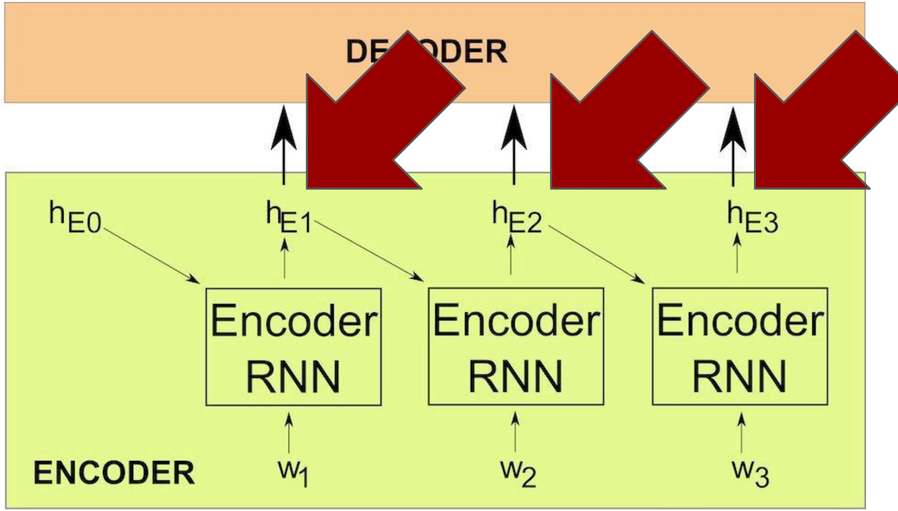


Recurrent NN from the previous section

The Attention Mechanism

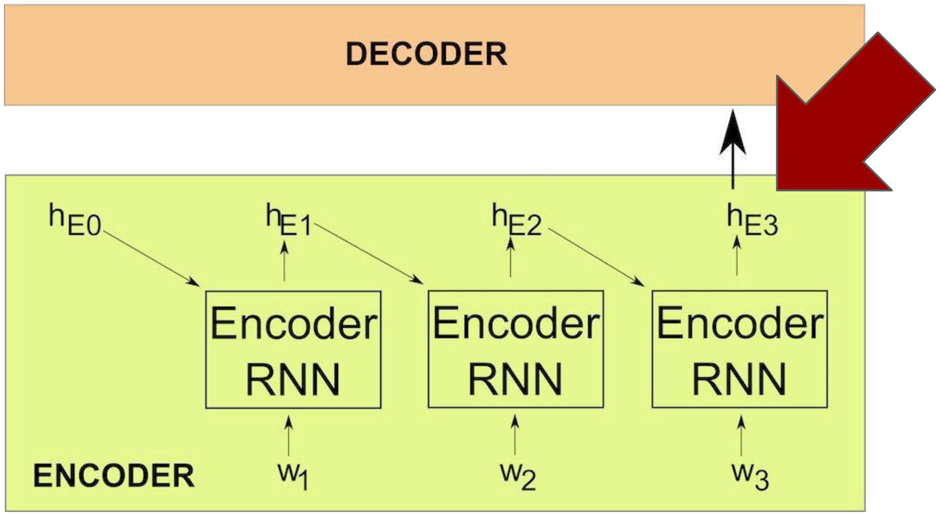


Recurrent NN from the previous section

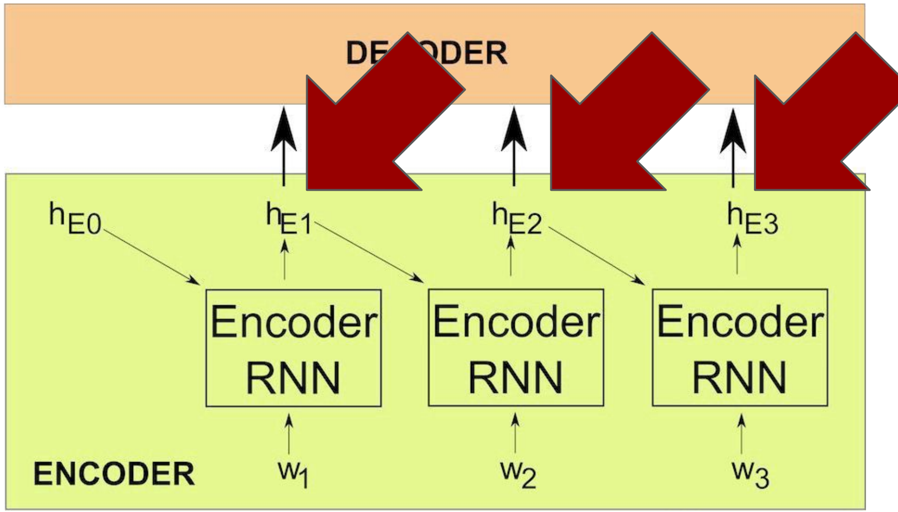


Send *all* of the hidden states to the Decoder, not just the last one

The Attention Mechanism



Recurrent NN from the previous section



Send *all* of the hidden states to the Decoder, not just the last one

The Decoder determines which input elements get *attended* to via a softmax layer

Further reading

jalammar.github.io/illustrated-transformer/

My blog post that this talked is based on + the linear algebra perspective on Attention:

towardsdatascience.com/natural-language-processing-the-age-of-transformers-a36c0265937d

Outline

1. NLP 101: how do we represent words in machine learning
   - 🤖 Tokenization
   - 🤖 Word embeddings
2. Order matters: Recurrent Neural Networks
   - 🤖 RNNs
   - 🤖 Seq2Seq models
   - 🤖 What is wrong with RNNs
3. Attention is all you need!
   - 🤖 What is Attention
   - 🤖 Attention, the linear algebra perspective
4. The Transformer architecture
   - 🤖 Mostly Attention
5. BERT: *contextualized* word embeddings

Attention is All You Need © Google, 2017

Attention is All You Need © Google, 2017

- Keep the Attention mechanism, throw away the sequential RNN
- Reminder: the basic idea of Attention was to pass all of the hidden states as inputs

Attention is All You Need © Google, 2017

- Keep the Attention mechanism, throw away the sequential RNN
- Reminder: the basic idea of Attention was to pass all of the hidden states as inputs
- Transformer:
    - Decoder and Encoder both have multiple layers

Attention is All You Need © Google, 2017

- Keep the Attention mechanism, throw away the sequential RNN
- Reminder: the basic idea of Attention was to pass all of the hidden states as inputs
- Transformer:
    - Decoder and Encoder both have multiple layers
    - The output hidden states of the Encoder are inputs to *all* the layers of the Decoder

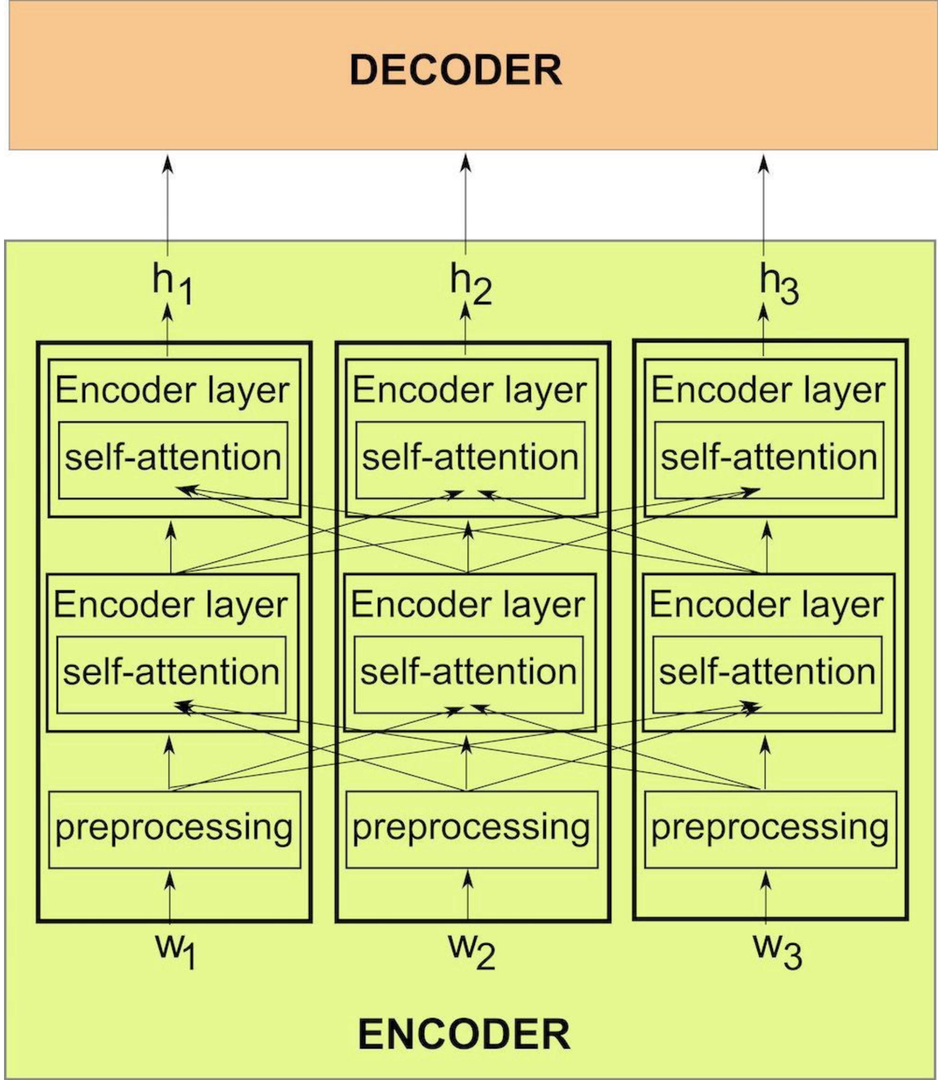
Attention is All You Need © Google, 2017

- Keep the Attention mechanism, throw away the sequential RNN
- Reminder: the basic idea of Attention was to pass all of the hidden states as inputs
- Transformer:
  - Decoder and Encoder both have multiple layers
  - The output hidden states of the Encoder are inputs to *all* the layers of the Decoder
  - The intermediate hidden states of the Encoder are also inputs… to other sequence elements' Encoder layers! **Self-Attention**

Attention is All You Need © Google, 2017

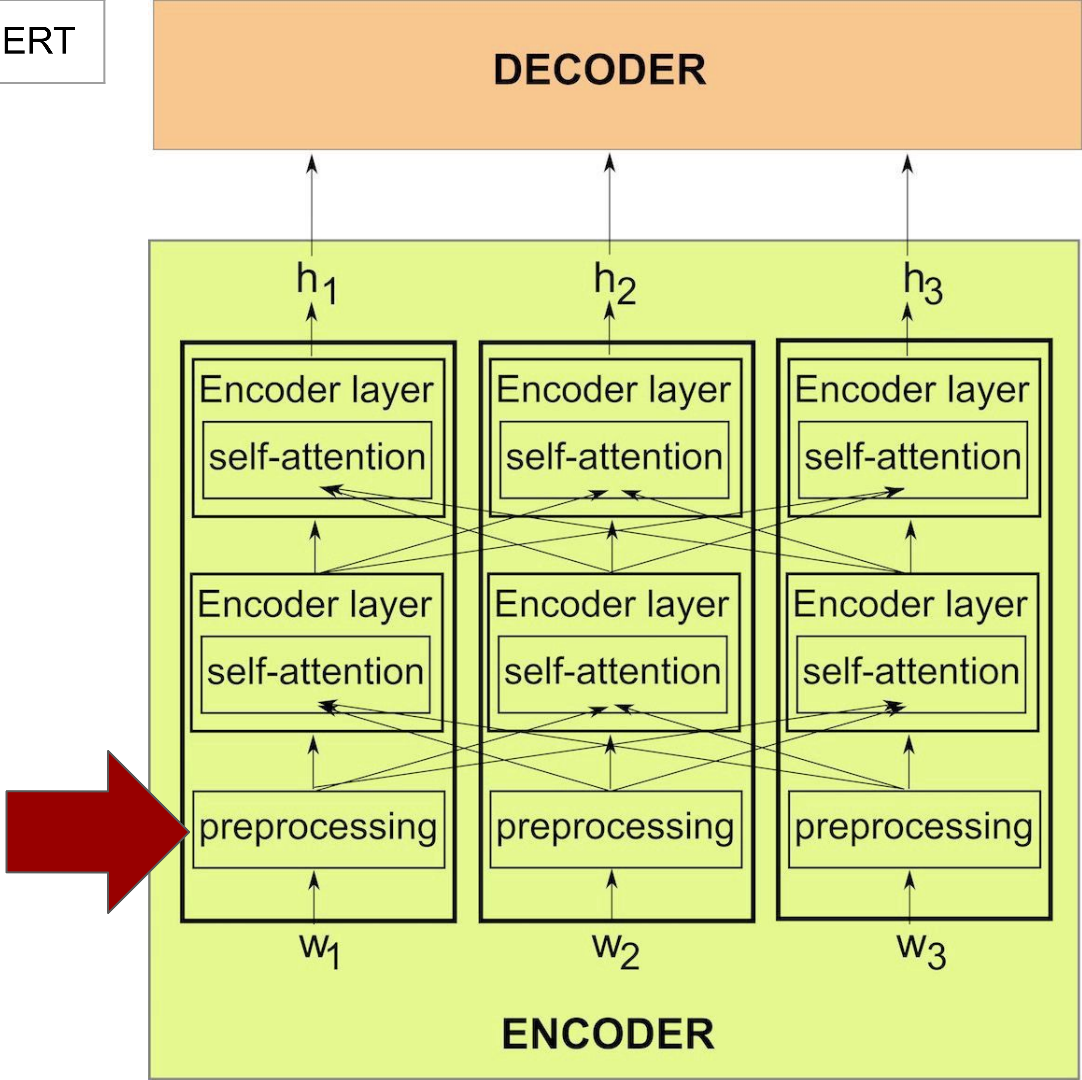
- Keep the Attention mechanism, throw away the sequential RNN
- Reminder: the basic idea of Attention was to pass all of the hidden states as inputs
- Transformer:
  - Decoder and Encoder both have multiple layers
  - The output hidden states of the Encoder are inputs to *all* the layers of the Decoder
  - The intermediate hidden states of the Encoder are also inputs… to other sequence elements' Encoder layers! **Self-Attention**
  - Also Self-Attention in the Decoder

Attention all around



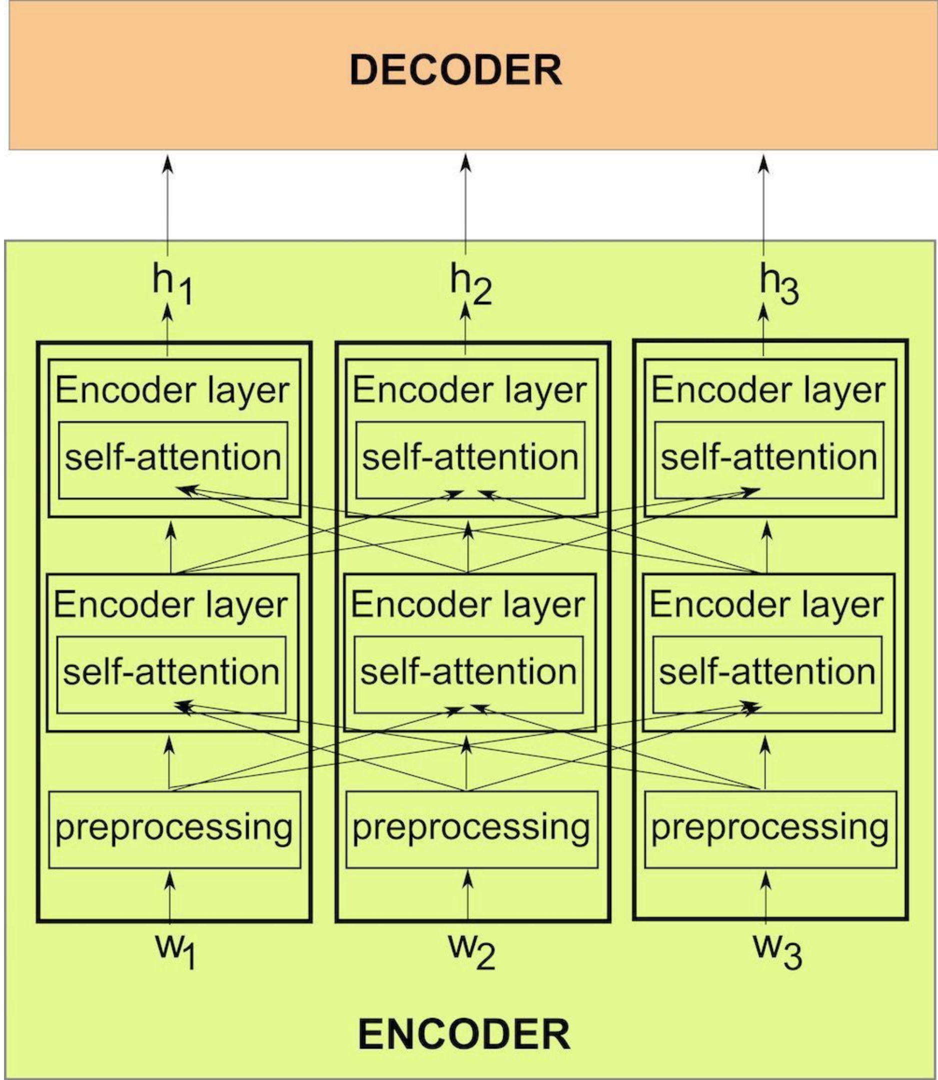
Attention all around

Word embedding + position encoding



Attention all around

- All of the sequence elements (w1, w2, w3) are being processed in parallel
- Each element's encoding gets information (through hidden states) from other elements
- Bidirectional by design 😍

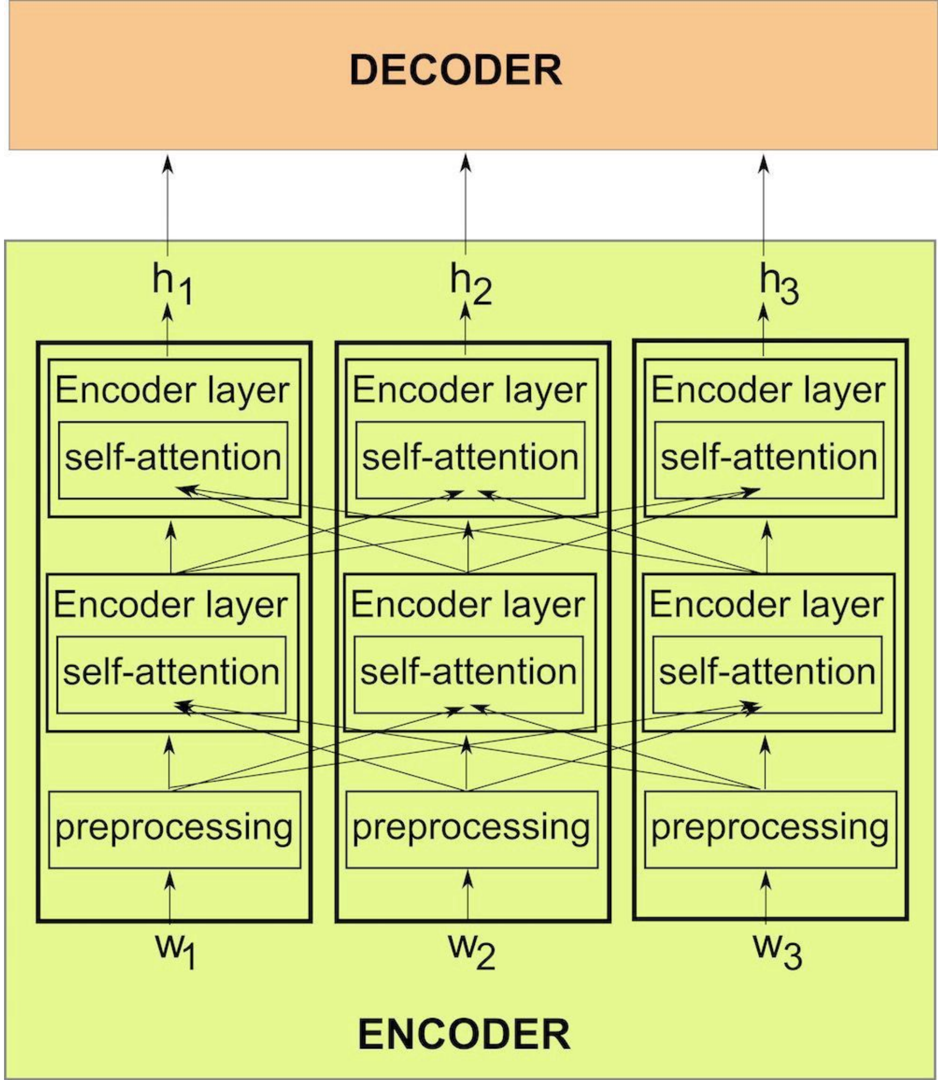


Why Self-Attention?

- “The **animal** did not cross the road because **it** was too tired.”

vs.

- “The animal did not cross the **road** because **it** was too wide.”



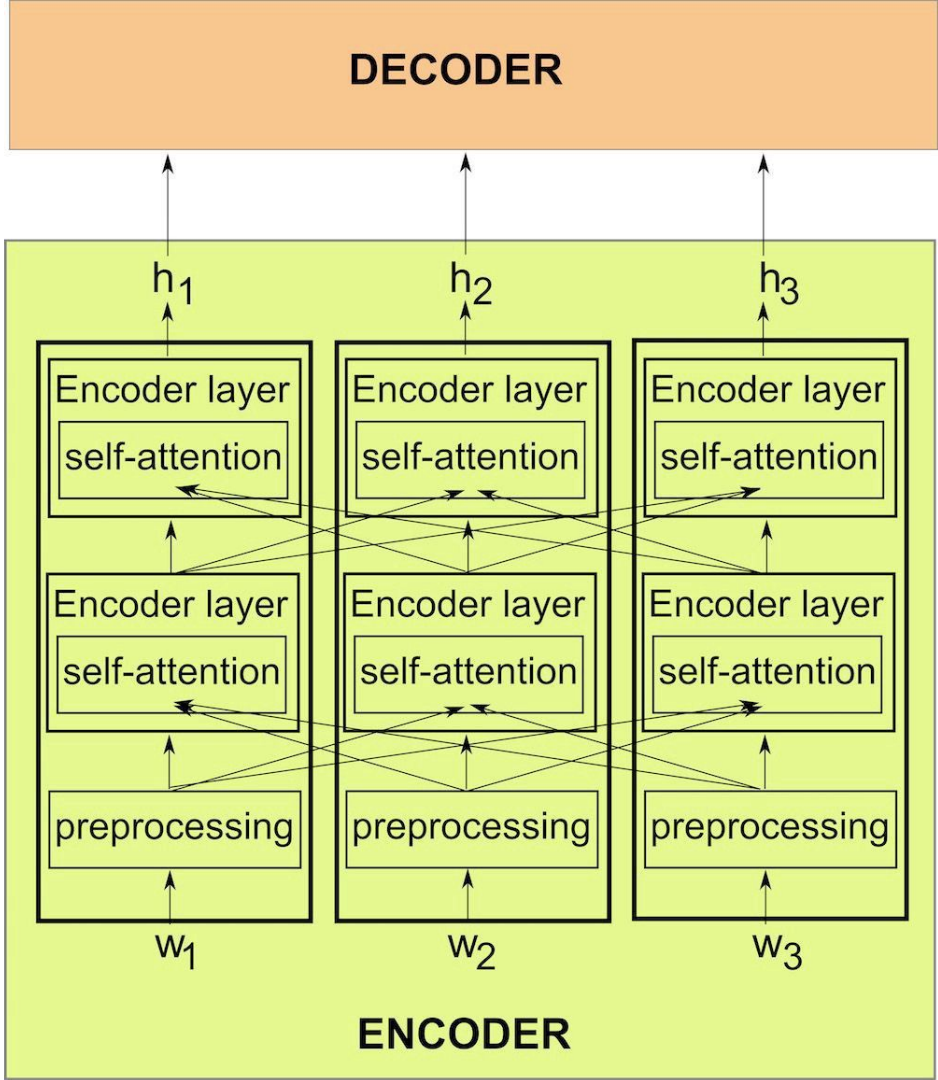
Why Self-Attention?

- “The **animal** did not cross the road because **it** was too *tired*.”

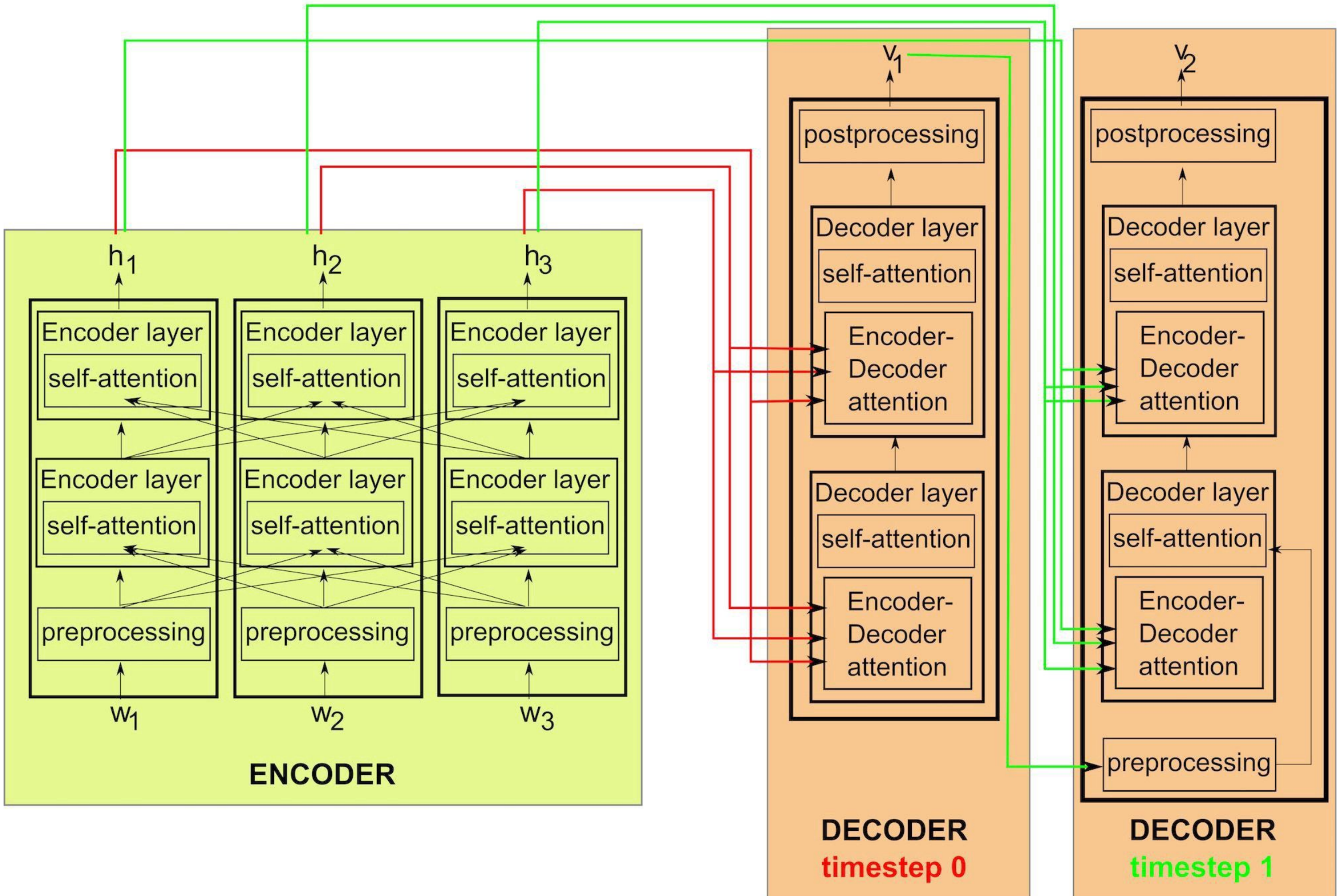
vs.

- “The animal did not cross the **road** because **it** was too *wide*.”

Uni-directional RNN would have missed this! Bi-directional is nice.



TRANSFORMER ARCHITECTURE



Outline

1. NLP 101: how do we represent words in machine learning
   - 🤖 Tokenization
   - 🤖 Word embeddings
2. Order matters: Recurrent Neural Networks
   - 🤖 RNNs
   - 🤖 Seq2Seq models
   - 🤖 What is wrong with RNNs
3. Attention is all you need!
   - 🤖 What is Attention
   - 🤖 Attention, the linear algebra perspective
4. The Transformer architecture
   - 🤖 Mostly Attention
5. BERT: *contextualized* word embeddings

(Contextual) word embeddings

- Dense (i.e. not sparse) vectors capturing the meaning of tokens/words

(Contextual) word embeddings

- Dense (i.e. not sparse) vectors capturing the meaning of tokens/words
- Meaning often depends on the context (surrounding words)

(Contextual) word embeddings

- Dense (i.e. not sparse) vectors capturing the meaning of tokens/words
- Meaning often depends on the context (surrounding words)
- Self-Attention in Transformers processes context when encoding a token

(Contextual) word embeddings

- Dense (i.e. not sparse) vectors capturing the meaning of tokens/words
- Meaning often depends on the context (surrounding words)
- Self-Attention in Transformers processes context when encoding a token
- Can we just take the Transformer Encoder and use it to embed words?

(Contextual) word embeddings

- Dense (i.e. not sparse) vectors capturing the meaning of tokens/words
- Meaning often depends on the context (surrounding words)
- Self-Attention in Transformers processes context when encoding a token
- Can we just take the Transformer Encoder and use it to embed words?

(Contextual) word embeddings

BERT: Bidirectional Encoder Representations from Transformers

(Contextual) word embeddings

BERT: Bidirectional Encoder Representations from Transformers

Instead of Word2Vec, use pre-trained BERT for your NLP tasks ;)

What was BERT pre-trained on?

Task # 1: Masked Language Modeling

- Take a text corpus, randomly mask 15% of words
- Put a softmax layer (dim = vocab size) on top of BERT encodings for the words that are present to predict which word is missing

What was BERT pre-trained on?

Task # 1: Masked Language Modeling

- Take a text corpus, randomly mask 15% of words
- Put a softmax layer (dim = vocab size) on top of BERT encodings for the words that are present to predict which word is missing

Task # 2: Next Sentence Prediction

- Input sequence: two sentences separated by a special token
- Binary classification: are these sentences successive or not?

Further reading and coding

My blog posts:

towardsdatascience.com/natural-language-processing-the-age-of-transformers-a36c0265937d

blog.scaleway.com/understanding-text-with-bert/

The HuggingFace 🤗 Transformers library: huggingface.co/