How do ‘Large Language Models’ work?

All of the history, none of the maths

Ali Knott

How we arrived at GPT—some landmarks

How we arrived at GPT—some landmarks

1986 Neural network learning method (backpropagation) (Rumelhart, Hinton, Williams)

How we arrived at GPT—some landmarks

1986 Neural network learning method (backpropagation) (Rumelhart, Hinton, Williams)

1991 Geoff Elman invents a network for processing word sequences

How we arrived at GPT—some landmarks

1986 Neural network learning method (backpropagation) (Rumelhart, Hinton, Williams)

1991 Geoff Elman invents a network for processing word sequences

2012 Deep networks are invented (Krizhevsky, Sutskever and Hinton)

How we arrived at GPT—some landmarks

1986 Neural network learning method (backpropagation) (Rumelhart, Hinton, Williams)

1991 Geoff Elman invents a network for processing word sequences

2012 Deep networks are invented (Krizhevsky, Sutskever and Hinton)

2013 New methods for representing individual words (influenced by Le Cun and Bengio)

How we arrived at GPT—some landmarks

1986 Neural network learning method (backpropagation) (Rumelhart, Hinton, Williams)

1991 Geoff Elman invents a network for processing word sequences

2012 Deep networks are invented (Krizhevsky, Sutskever and Hinton)

2013 New methods for representing individual words (influenced by Le Cun and Bengio)

2016 Google Translate adopts a neural network model

How we arrived at GPT—some landmarks

1986 Neural network learning method (backpropagation) (Rumelhart, Hinton, Williams)

1991 Geoff Elman invents a network for processing word sequences

2012 Deep networks are invented (Krizhevsky, Sutskever and Hinton)

2013 New methods for representing individual words (influenced by Le Cun and Bengio)

2016 Google Translate adopts a neural network model

2015 Improved word sequencing network, using 'attention' (Bahdanau, Cho and Bengio)

How we arrived at GPT—some landmarks

1986 Neural network learning method (backpropagation) (Rumelhart, Hinton, Williams)

1991 Geoff Elman invents a network for processing word sequences

2012 Deep networks are invented (Krizhevsky, Sutskever and Hinton)

2013 New methods for representing individual words (influenced by Le Cun and Bengio)

2016 Google Translate adopts a neural network model

2015 Improved word sequencing network, using ‘attention’ (Bahdanau, Cho and Bengio)

2018 Google invents the transformer: ‘attention is all you need’

How we arrived at GPT—some landmarks

1986 Neural network learning method (backpropagation) (Rumelhart, Hinton, Williams)

1991 Geoff Elman invents a network for processing word sequences

2012 Deep networks are invented (Krizhevsky, Sutskever and Hinton)

2013 New methods for representing individual words (influenced by Le Cun and Bengio)

2016 Google Translate adopts a neural network model

2015 Improved word sequencing network, using 'attention' (Bahdanau, Cho and Bengio)

2018 Google invents the transformer: 'attention is all you need'

2019 OpenAI release a transformer designed for general use: GPT-2.

How we arrived at GPT—some landmarks

1986 Neural network learning method (backpropagation) (Rumelhart, Hinton, Williams)

1991 Geoff Elman invents a network for processing word sequences

2012 Deep networks are invented (Krizhevsky, Sutskever and Hinton)

2013 New methods for representing individual words (influenced by Le Cun and Bengio)

2016 Google Translate adopts a neural network model

2015 Improved word sequencing network, using 'attention' (Bahdanau, Cho and Bengio)

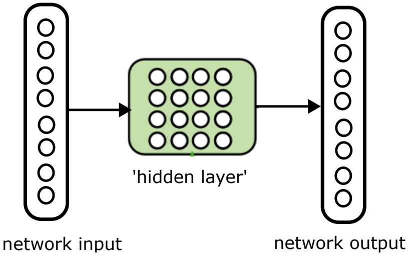
2018 Google invents the transformer: 'attention is all you need'

2019 OpenAI release a transformer designed for general use: GPT-2.

Nowadays, you can be a computational linguist without knowing anything that happened before 2012.

Recap: a basic network for supervised learning

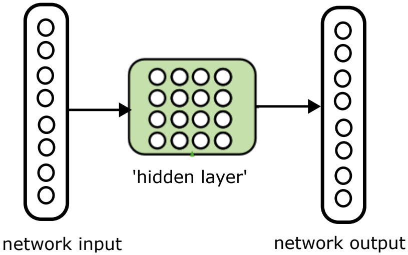
Here's the basic neural network architecture I introduced before.



Recap: a basic network for supervised learning

Here's the basic neural network architecture I introduced before.

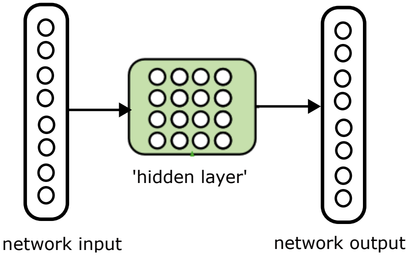
- It can be trained to map any 'input' representations onto any 'output' representations.



Recap: a basic network for supervised learning

Here's the basic neural network architecture I introduced before.

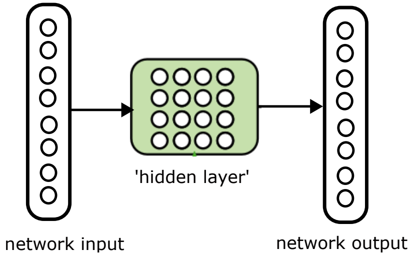
- It can be trained to map any 'input' representations onto any 'output' representations.
  E.g. images $\rightarrow$ object labels



Recap: a basic network for supervised learning

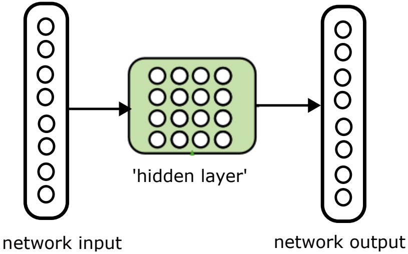
Here's the basic neural network architecture I introduced before.

- It can be trained to map any 'input' representations onto any 'output' representations.
  E.g. images $\rightarrow$ object labels
- Training happens through supervised learning. (Backpropagation, circa 1986.)



Elman's invention: a network for learning word sequences

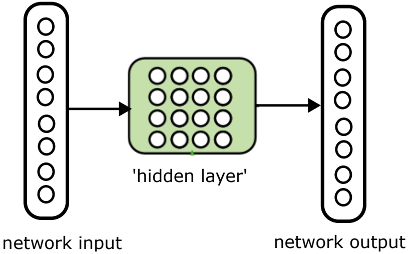
In 1991, Jeff Elman made a network designed to process language.



Elman's invention: a network for learning word sequences

In 1991, Jeff Elman made a network designed to process language.

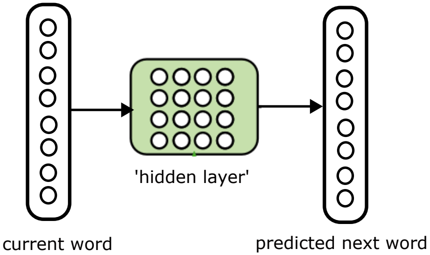
- The network was set up to process sequences.



Elman's invention: a network for learning word sequences

In 1991, Jeff Elman made a network designed to process language.

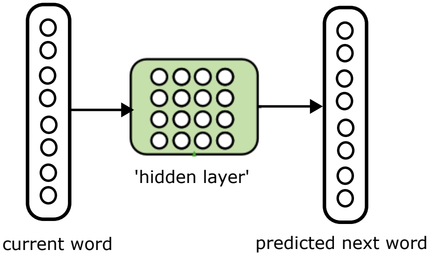
- The network was set up to process sequences.
- It took one word at a time, in its input layer, and learned to predict the next word.



Elman's invention: a network for learning word sequences

In 1991, Jeff Elman made a network designed to process language.

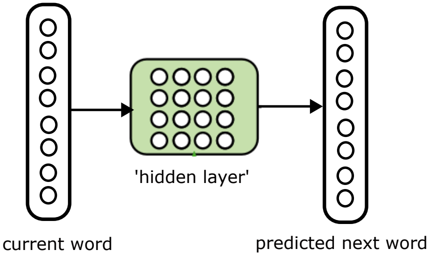
- The network was set up to process sequences.
- It took one word at a time, in its input layer, and learned to predict the next word.
- This network can learn from any sample of language, through 'self-supervision'.



Elman's invention: a network for learning word sequences

In 1991, Jeff Elman made a network designed to process language.

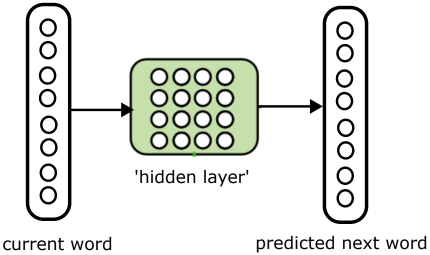
- Of course, the next word depends on the current word but also on *previous words*.



Elman's invention: a network for learning word sequences

In 1991, Jeff Elman made a network designed to process language.

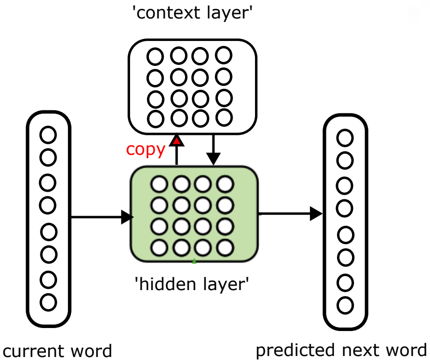
- Of course, the next word depends on the current word but also on *previous words*.
- Elman added a 'context layer', which *took a copy of the hidden layer* at each timepoint, and passed this as an extra ('recurrent') input *at the next timepoint*.



Elman's invention: a network for learning word sequences

In 1991, Jeff Elman made a network designed to process language.

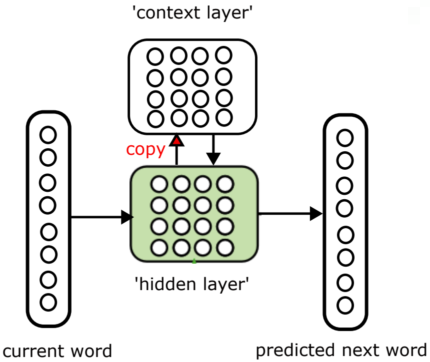
- Of course, the next word depends on the current word but also on *previous words*.
- Elman added a 'context layer', which *took a copy of the hidden layer* at each timepoint, and passed this as an extra ('recurrent') input *at the next timepoint*.



Elman's invention: a network for learning word sequences

In 1991, Jeff Elman made a network designed to process language.

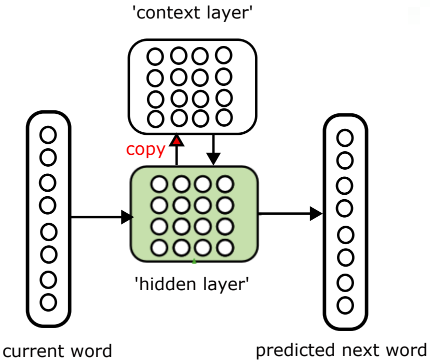
- The context layer functions as a *memory* for the recent words.



Elman's invention: a network for learning word sequences

In 1991, Jeff Elman made a network designed to process language.

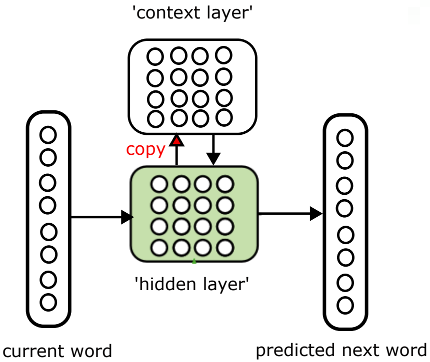
- The context layer functions as a *memory* for the recent words.
- It holds a representation of 'the word sequence up to here'.



Elman's invention: a network for learning word sequences

In 1991, Jeff Elman made a network designed to process language.

- The context layer functions as a *memory* for the recent words.
- It holds a representation of 'the word sequence up to here'.
- But. . . this representation is heavily skewed towards *the most recent words*.



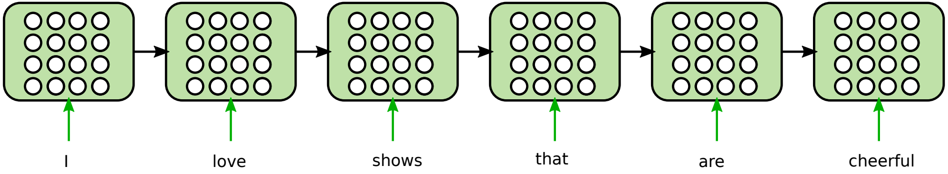
Elman networks for representing ‘the meaning of a text’

Elman networks for representing 'the meaning of a text'

Imagine the hidden layer of an Elman network as it processes each word in a sentence, one by one.

Elman networks for representing 'the meaning of a text'

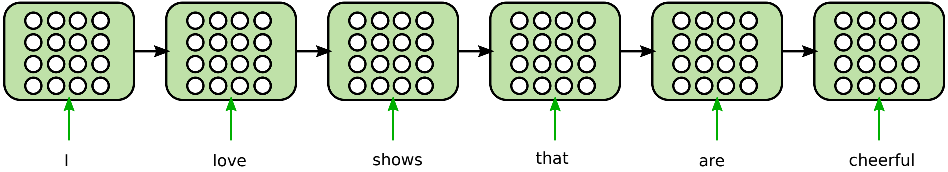
Imagine the hidden layer of an Elman network as it processes each word in a sentence, one by one.



Elman networks for representing 'the meaning of a text'

Imagine the hidden layer of an Elman network as it processes each word in a sentence, one by one.

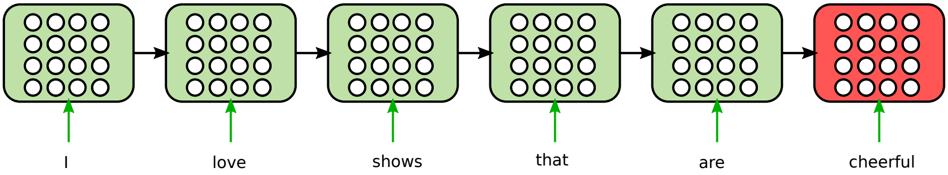
- The activity in the hidden layer after the *last word* in the sentence kind of represents 'the whole sentence' (as a word sequence).



Elman networks for representing 'the meaning of a text'

Imagine the hidden layer of an Elman network as it processes each word in a sentence, one by one.

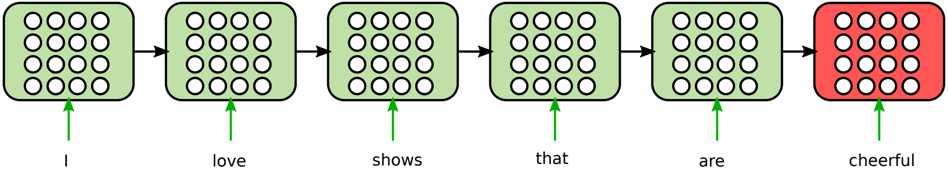
- The activity in the hidden layer after the *last word* in the sentence kind of represents 'the whole sentence' (as a word sequence).



Elman networks for representing ‘the meaning of a text’

Imagine the hidden layer of an Elman network as it processes each word in a sentence, one by one.

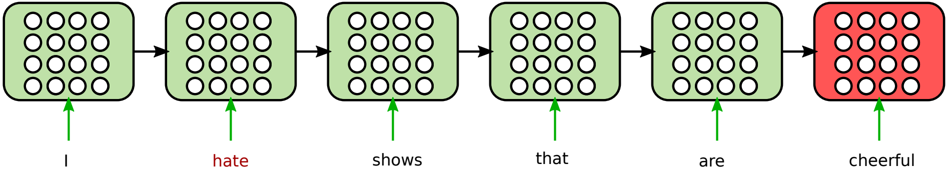
- The activity in the hidden layer after the *last word* in the sentence kind of represents ‘the whole sentence’ (as a word sequence).
- But the representation is biased towards words at the end of the sentence.



Elman networks for representing 'the meaning of a text'

Imagine the hidden layer of an Elman network as it processes each word in a sentence, one by one.

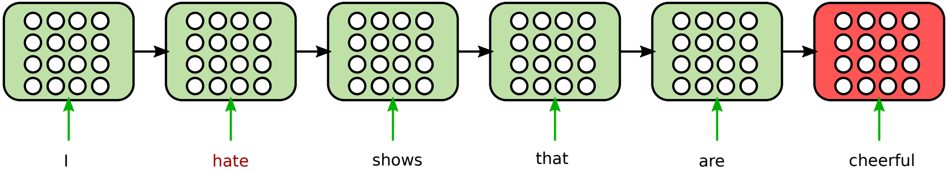
- The activity in the hidden layer after the *last word* in the sentence kind of represents 'the whole sentence' (as a word sequence).
- But the representation is biased towards words at the end of the sentence.



Elman networks for representing 'the meaning of a text'

Imagine the hidden layer of an Elman network as it processes each word in a sentence, one by one.

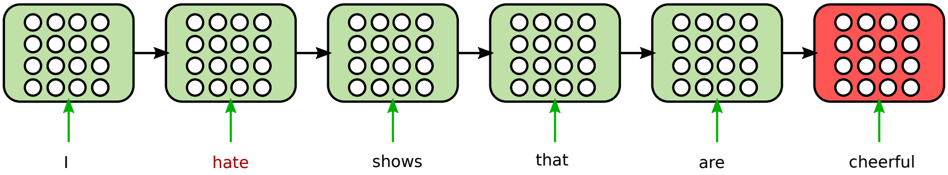
- The activity in the hidden layer after the *last word* in the sentence kind of represents 'the whole sentence' (as a word sequence).
- But the representation is biased towards words at the end of the sentence.
- A few improvements happened in the 90s...



Elman networks for representing 'the meaning of a text'

Imagine the hidden layer of an Elman network as it processes each word in a sentence, one by one.

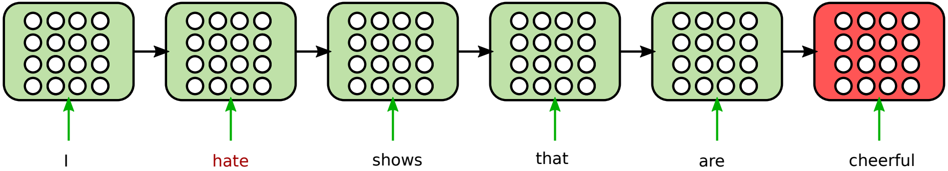
- The activity in the hidden layer after the *last word* in the sentence kind of represents 'the whole sentence' (as a word sequence).
- But the representation is biased towards words at the end of the sentence.
- A few improvements happened in the 90s... but networks still used a 'recurrent' hidden layer updated after each word, losing information about earlier words.



Elman networks for representing 'the meaning of a text'

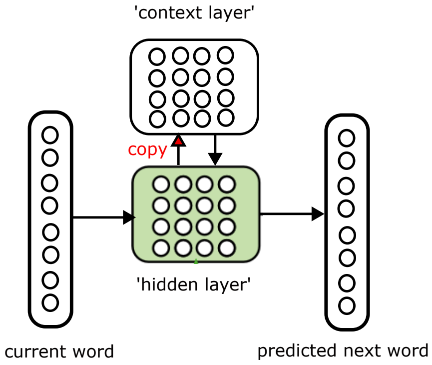
Imagine the hidden layer of an Elman network as it processes each word in a sentence, one by one.

- The activity in the hidden layer after the *last word* in the sentence kind of represents 'the whole sentence' (as a word sequence).
- But the representation is biased towards words at the end of the sentence.
- A few improvements happened in the 90s. . . but networks still used a 'recurrent' hidden layer updated after each word, losing information about earlier words.
- A recurrent hidden layer creates a 'bottleneck problem'.



Representations of words in neural networks

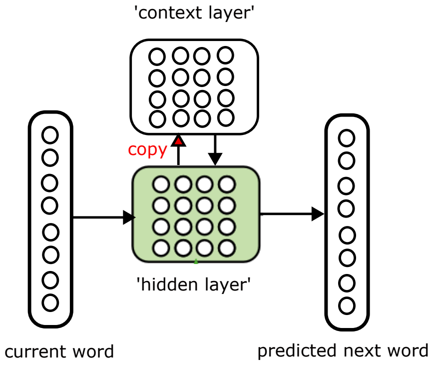
From 95 on, people were also improving how networks represented *individual words*.



Representations of words in neural networks

From 95 on, people were also improving how networks represented *individual words*.

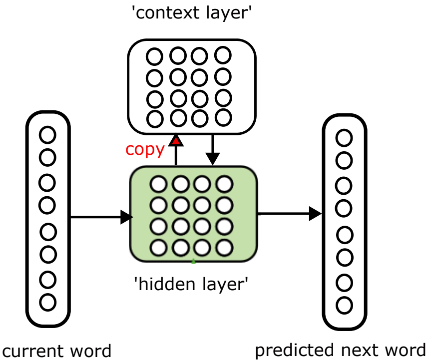
- A simple way is to represent each word in its own unit ('one-hot' encoding).



Representations of words in neural networks

From 95 on, people were also improving how networks represented *individual words*.

- A simple way is to represent each word in its own unit ('one-hot' encoding).

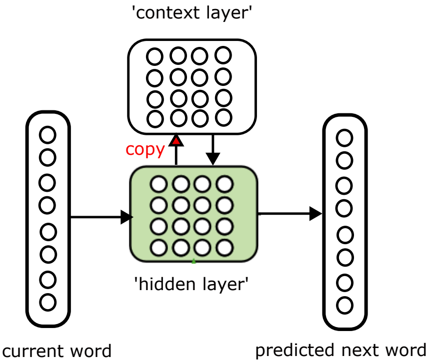


Word	encoding
dog	1000. . .
cat	0100. . .
apple	0010. . .
banana	0001. . .
. . .	

Representations of words in neural networks

From 95 on, people were also improving how networks represented *individual words*.

- A simple way is to represent each word in its own unit ('one-hot' encoding).
- But this encoding ignores similarities between words—we want our network to have *similar responses to similar words*.

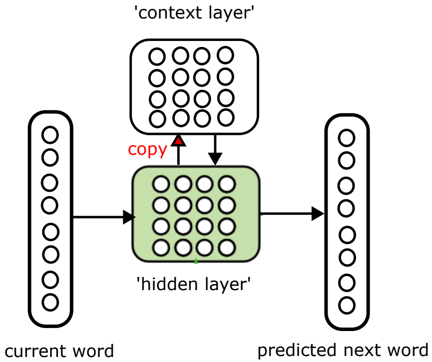


Word	encoding
dog	1000. . .
cat	0100. . .
apple	0010. . .
banana	0001. . .
. . .	

Representations of words in neural networks

From 95 on, people were also improving how networks represented *individual words*.

- Actually, one-hot encodings of words are very good for the *output* layer.

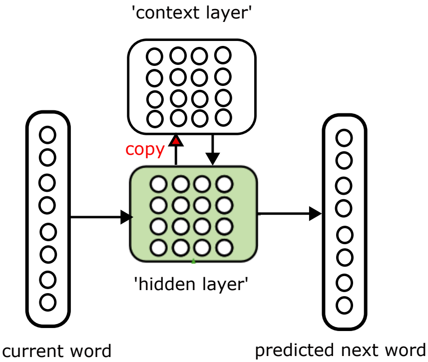


Word	encoding
dog	1000. . .
cat	0100. . .
apple	0010. . .
banana	0001. . .
. . .	

Representations of words in neural networks

From 95 on, people were also improving how networks represented *individual words*.

- Actually, one-hot encodings of words are very good for the *output* layer.
  - Since words are independent, the pattern of activity in the output layer can be interpreted as a *probability distribution* over words.

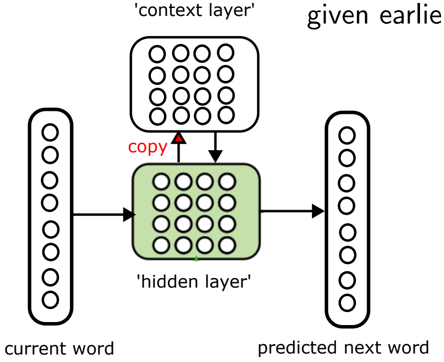


Word	encoding
dog	1000...
cat	0100...
apple	0010...
banana	0001...
...	

Representations of words in neural networks

From 95 on, people were also improving how networks represented *individual words*.

- Actually, one-hot encodings of words are very good for the *output* layer.
  - Since words are independent, the pattern of activity in the output layer can be interpreted as a *probability distribution* over words.
  - An Elman network estimates the (conditional) probability of the next word, given earlier words.

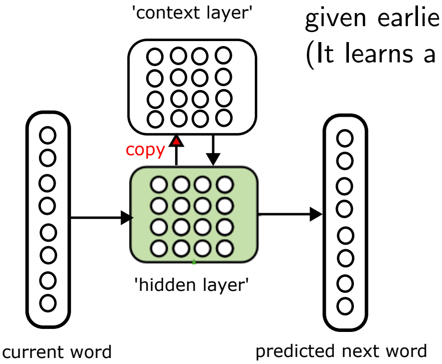


Word	encoding
dog	1000. . .
cat	0100. . .
apple	0010. . .
banana	0001. . .
. . .	

Representations of words in neural networks

From 95 on, people were also improving how networks represented *individual words*.

- Actually, one-hot encodings of words are very good for the *output* layer.
  - Since words are independent, the pattern of activity in the output layer can be interpreted as a *probability distribution* over words.
  - An Elman network estimates the (conditional) probability of the next word, given earlier words.
    (It learns a probability model of its training text!)

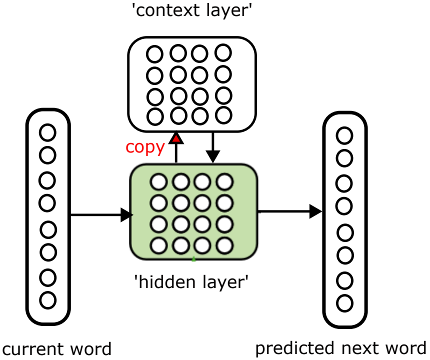


Word	encoding
dog	1000. . .
cat	0100. . .
apple	0010. . .
banana	0001. . .
. . .	

Representations of words in neural networks

From 95 on, people were also improving how networks represented *individual words*.

- But for words in the *input* layer, we want similar words to have similar patterns.

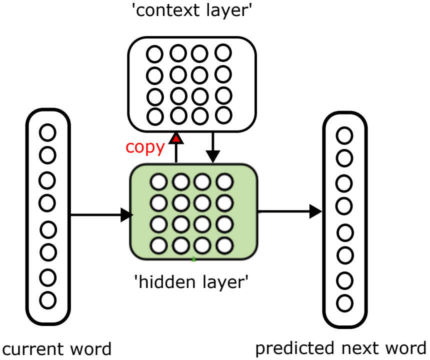


Word	encoding
dog	1000. . .
cat	0100. . .
apple	0010. . .
banana	0001. . .
. . .	

Representations of words in neural networks

From 95 on, people were also improving how networks represented *individual words*.

- But for words in the *input* layer, we want similar words to have similar patterns.
- How can a network learn word representations that encode similarity like that?

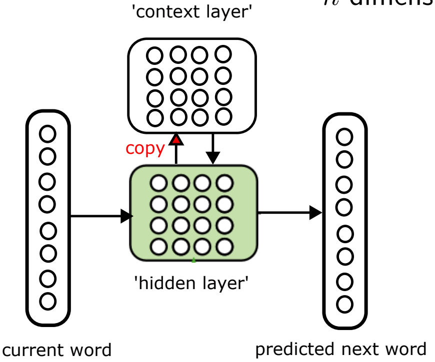


Word	encoding
dog	1000. . .
cat	0100. . .
apple	0010. . .
banana	0001. . .
. . .	

Representations of words in neural networks

From 95 on, people were also improving how networks represented *individual words*.

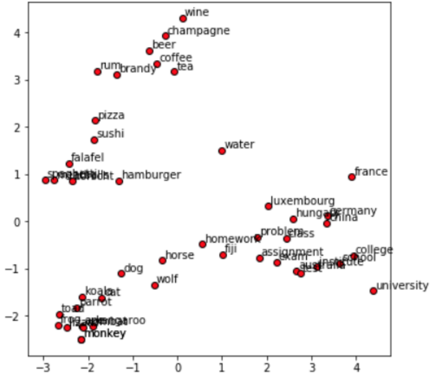
- But for words in the *input* layer, we want similar words to have similar patterns.
- How can a network learn word representations that encode similarity like that?
- The answer is to think of the n units in the input layer as holding an 'n-dimensional space' of possible word meanings.



Word	encoding
dog	1000. . .
cat	0100. . .
apple	0010. . .
banana	0001. . .
. . .	

An n-dimensional space for word representations

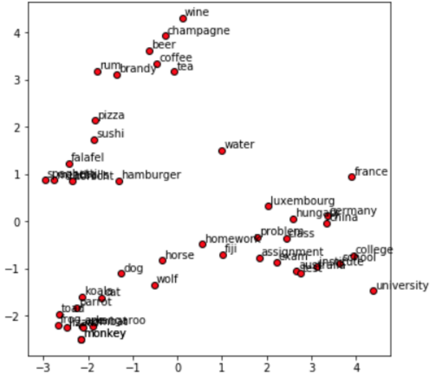
Let's visualise this in 2 dimensions, because that's the best our brains can do :-)



An n-dimensional space for word representations

Let's visualise this in 2 dimensions, because that's the best our brains can do :-)

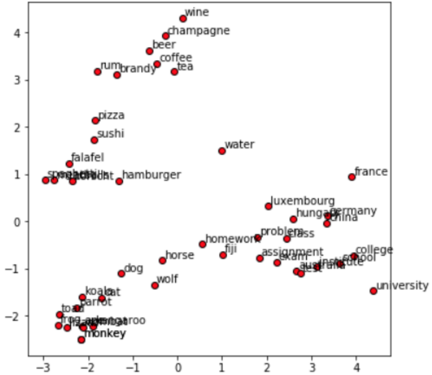
- So how can we learn representations like this?



An n-dimensional space for word representations

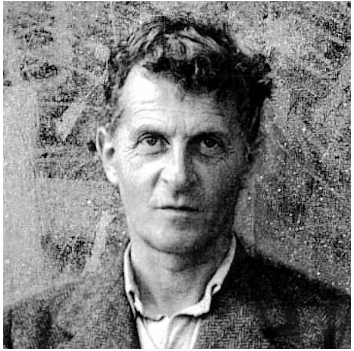
Let's visualise this in 2 dimensions, because that's the best our brains can do :-)

- So how can we learn representations like this?
- Again, we want to learn from actual text, using self-supervision. . .



Learning vector-based word representations

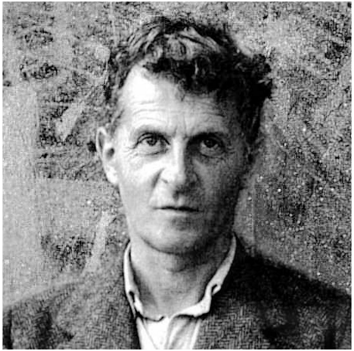
The learning methods all trade on Wittgenstein's idea that the meaning of a word consists in its 'use'.



Learning vector-based word representations

The learning methods all trade on Wittgenstein's idea that the meaning of a word consists in its 'use'.

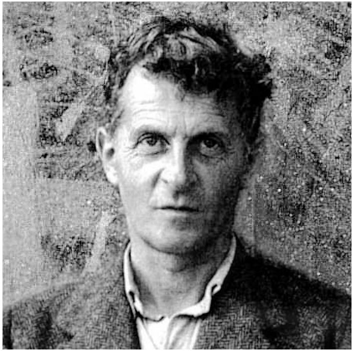
- The rules that govern how words are used are *loose*, like the rules that guide participation in other social practices.



Learning vector-based word representations

The learning methods all trade on Wittgenstein's idea that the meaning of a word consists in its 'use'.

- The rules that govern how words are used are *loose*, like the rules that guide participation in other social practices.
- Different uses of a given word *loosely resemble one another*.



‘Word use’ in a language corpus

Aligning uses of words in a corpus makes the point well.

‘Word use’ in a language corpus

Aligning uses of words in a corpus makes the point well.

```
 pierced, elongated into a CUP, and terminated by the phospl
one morning while taking a CUP of chocolate in a cafe, Rouss
. warming his hands on the CUP, although the room was heavy
f powdered coffee into his CUP and then filled it with hot w
 of powdered coffee in his CUP and filled it with hot water,
They all three had another CUP of coffee. Eugene was in his
f self-esteem. Offer her a CUP of tea and she would say, "Wh
ith everything included, a CUP of coffee and the fastest se
 go". Maude swooped up the CUP and hiked up her top hoop as
drinkable- especially that CUP! She was deeply, horribly su
ethal dose of opium in the CUP. So suppose somebody only wis
le's murder and opium in a CUP of coffee. She started back i
 to the kitchen. She had a CUP of something steaming, coffee
ne, sit down. Put down the CUP of coffee. Tell me what this
s heated, poured himself a CUP and went up to the chartroom.
ogan said. "Sitting with a CUP of coffee now. It shouldn't k
or "Kohi futotsu"! for one CUP of coffee. Two other Japanese
t you come in then, have a CUP of coffee- or something? ...
sually stop, make myself a CUP of coffee, and contemplate wh
```

‘Word use’ in a language corpus

Aligning uses of words in a corpus makes the point well.

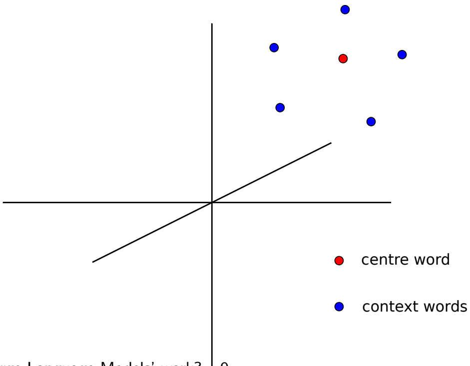
- J R Firth (1956): ‘You shall know a word by the company it keeps’.

```
 pierced, elongated into a CUP, and terminated by the phosph
one morning while taking a CUP of chocolate in a cafe, Rouss
. warming his hands on the CUP, although the room was heavy
f powdered coffee into his CUP and then filled it with hot w
 of powdered coffee in his CUP and filled it with hot water,
They all three had another CUP of coffee. Eugene was in his
f self-esteem. Offer her a CUP of tea and she would say, "Wh
ith everything included, a CUP of coffee and the fastest se
 go". Maude swooped up the CUP and hiked up her top hoop as
drinkable- especially that CUP! She was deeply, horribly su
ethal dose of opium in the CUP. So suppose somebody only wis
le's murder and opium in a CUP of coffee. She started back f
 to the kitchen. She had a CUP of something steaming, coffee
ne, sit down. Put down the CUP of coffee. Tell me what this
s heated, poured himself a CUP and went up to the chartroom.
ogan said. "Sitting with a CUP of coffee now. It shouldn't b
or "Kohi futotsu"! for one CUP of coffee. Two other Japanese
t you come in then, have a CUP of coffee- or something? ...
sually stop, make myself a CUP of coffee, and contemplate wh
```

Learning vector-based word representations

Start off by placing each word at a random point in the n-dimensional space.

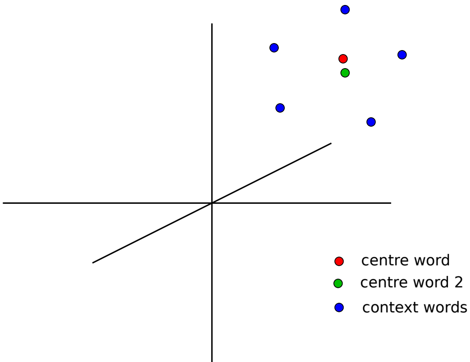
- Then chug through a corpus, a word at a time, moving each word towards all the words it's close to.



Learning vector-based word representations

Start off by placing each word at a random point in the n-dimensional space.

- Then chug through a corpus, a word at a time, moving each word towards all the words it's close to.
- After this, words that appear *in similar contexts* wil be close to each other.



Putting language models to work

Putting language models to work

Many language tasks involve mapping an ‘input text’ onto an *output text*.

Putting language models to work

Many language tasks involve mapping an ‘input text’ onto an *output text*.

Examples:

- Question-answering. (*What’s your name? My name is Ali.*)

Putting language models to work

Many language tasks involve mapping an ‘input text’ onto an *output text*.

Examples:

- Question-answering. (*What’s your name? My name is Ali.*)
- Text summarisation. (*[Long match report] Spurs played Arsenal. Spurs won 1-0.*)

Putting language models to work

Many language tasks involve mapping an 'input text' onto an *output text*.

Examples:

- Question-answering. (*What's your name? My name is Ali.*)
- Text summarisation. (*[Long match report] Spurs played Arsenal. Spurs won 1-0.*)
- Machine translation. (*My name is Ali. Ko Ali taku ingoa.*)

Putting language models to work

Many language tasks involve mapping an ‘input text’ onto an *output text*.

Examples:

- Question-answering. (*What’s your name? My name is Ali.*)
- Text summarisation. (*[Long match report] Spurs played Arsenal. Spurs won 1-0.*)
- Machine translation. (*My name is Ali. Ko Ali taku ingoa.*)

We can use our network-based language models to learn these tasks.

Sequence-to-Sequence language models

A sequence-to-sequence (Seq2Seq) language model is trained on a corpus of input/output text pairs.

Input text	Output text
My name is Ali	Ko Ali taku ingoa
Where do you live	Ko wai tō kainga inaianei
I am from Dunedin	Nō Otepoti ahau
The dog chased the cat	I whai te kurī i te ngeru

Sequence-to-Sequence language models

A sequence-to-sequence (Seq2Seq) language model is trained on a corpus of input/output text pairs.

Input text	Output text
My name is Ali	Ko Ali taku ingoa
Where do you live	Ko wai tō kainga inaianei
I am from Dunedin	Nō Otepoti ahau
The dog chased the cat	I whai te kurī i te ngeru

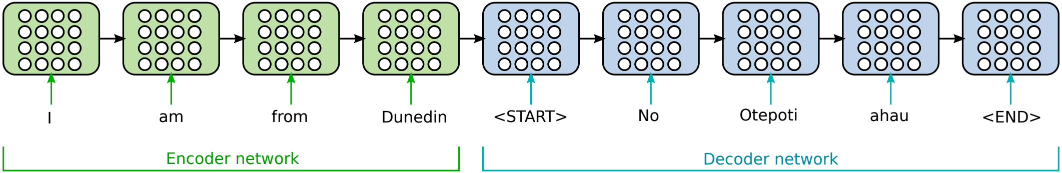
Sequence-to-Sequence language models

A sequence-to-sequence (Seq2Seq) language model is trained on a corpus of input/output text pairs.

Input text	Output text
My name is Ali	Ko Ali taku ingoa
Where do you live	Ko wai tō kainga inaianei
I am from Dunedin	Nō Otepoti ahau
The dog chased the cat	I whai te kurī i te ngeru

Sequence-to-Sequence language models

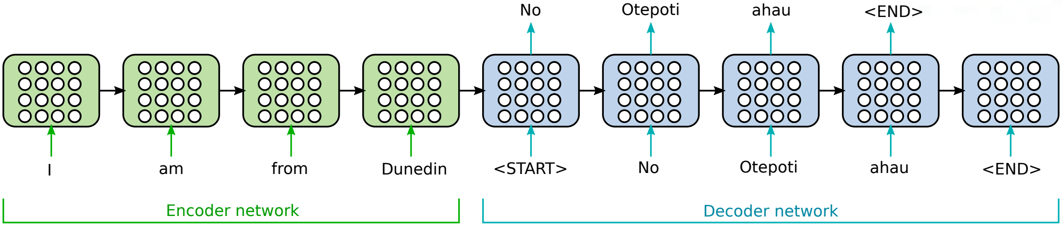
We present each input/output pair to 2 networks: an encoder, and a decoder.



Sequence-to-Sequence language models

We present each input/output pair to 2 networks: an encoder, and a decoder.

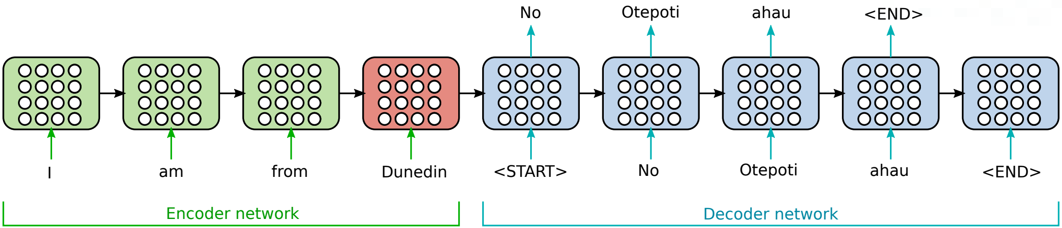
- The encoder network processes the input text, word by word.
  The decoder network iteratively *predicts the next word* of the output text.



Sequence-to-Sequence language models

We present each input/output pair to 2 networks: an encoder, and a decoder.

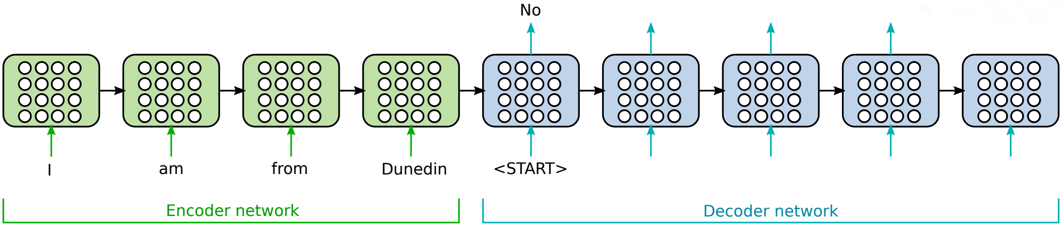
- The encoder network processes the input text, word by word.
  The decoder network iteratively *predicts the next word* of the output text.
- The key idea is that the decoder network takes the final state of the encoder network as its initial internal state.



Sequence-to-Sequence language models

We present each input/output pair to 2 networks: an encoder, and a decoder.

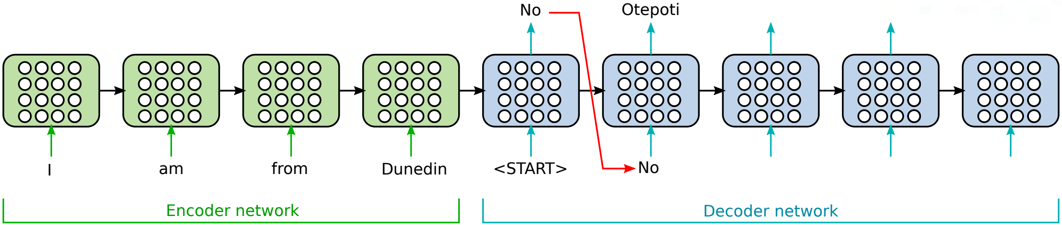
- The encoder network processes the input text, word by word.
  The decoder network iteratively *predicts the next word* of the output text.
- The key idea is that the decoder network takes the final state of the encoder network as its initial internal state.
- After training, we use the decoder *to generate* the output text.



Sequence-to-Sequence language models

We present each input/output pair to 2 networks: an encoder, and a decoder.

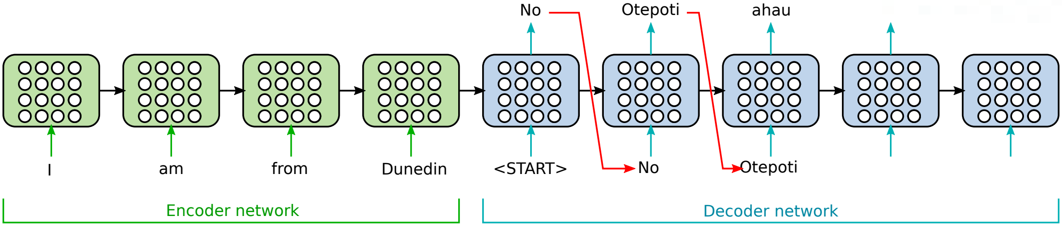
- The encoder network processes the input text, word by word.
  The decoder network iteratively *predicts the next word* of the output text.
- The key idea is that the decoder network takes the final state of the encoder network as its initial internal state.
- After training, we use the decoder *to generate* the output text.



Sequence-to-Sequence language models

We present each input/output pair to 2 networks: an encoder, and a decoder.

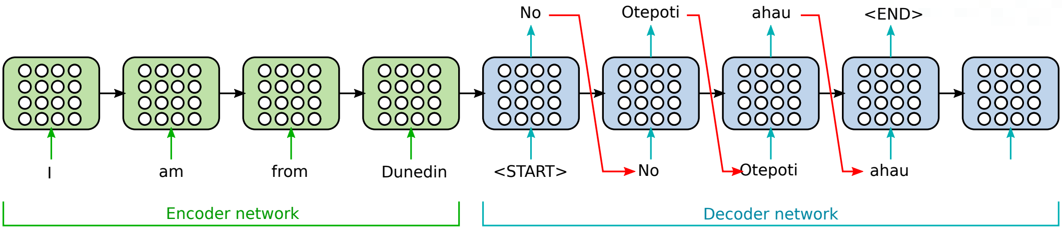
- The encoder network processes the input text, word by word.
  The decoder network iteratively *predicts the next word* of the output text.
- The key idea is that the decoder network takes the final state of the encoder network as its initial internal state.
- After training, we use the decoder *to generate* the output text.



Sequence-to-Sequence language models

We present each input/output pair to 2 networks: an encoder, and a decoder.

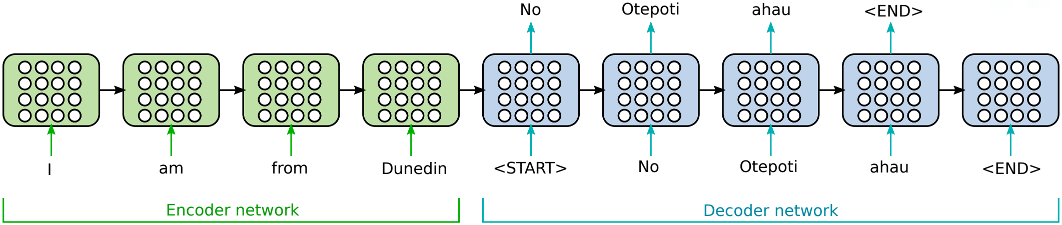
- The encoder network processes the input text, word by word.
  The decoder network iteratively *predicts the next word* of the output text.
- The key idea is that the decoder network takes the final state of the encoder network as its initial internal state.
- After training, we use the decoder *to generate* the output text.



Sequence-to-Sequence language models

We present each input/output pair to 2 networks: an encoder, and a decoder.

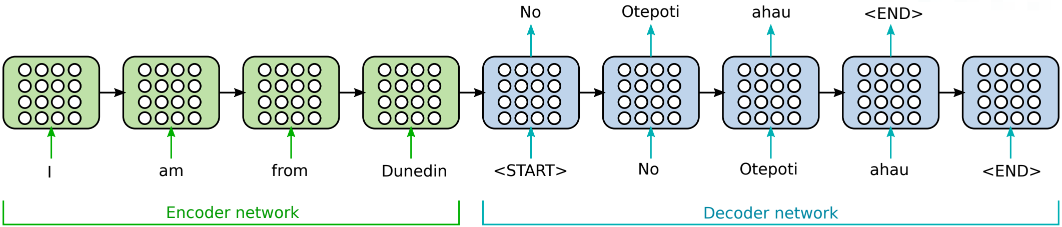
- The encoder network processes the input text, word by word.
  The decoder network iteratively *predicts the next word* of the output text.
- The key idea is that the decoder network takes the final state of the encoder network as its initial internal state.
- After training, we use the decoder *to generate* the output text.



Sequence-to-Sequence language models

A regular RNN network learns a language model:

$$p(w_{i+1}|w_{i-D} \ldots w_D)$$



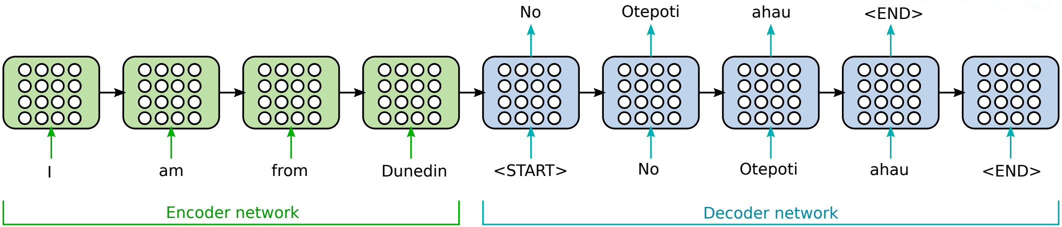
Sequence-to-Sequence language models

A regular RNN network learns a language model:

$$p(w_{i+1}|w_{i-D}\dots w_D)$$

A Seq2Seq network learns a conditional language model:

$$p(y_{i+1}|y_{i-D}\dots y_D, x)$$



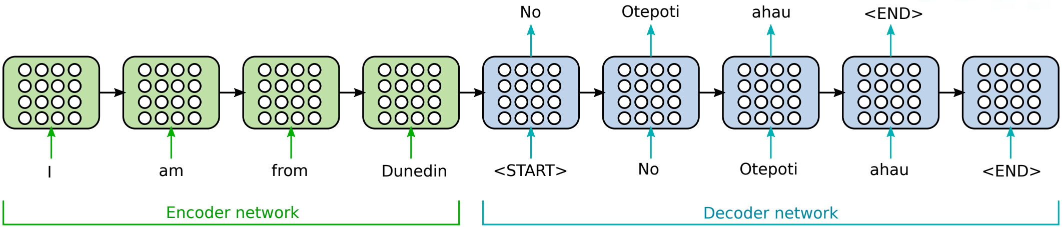
Sequence-to-Sequence language models

A regular RNN network learns a language model:

$$p(w_{i+1}|w_{i-D}\dots w_D)$$

A Seq2Seq network learns a conditional language model:

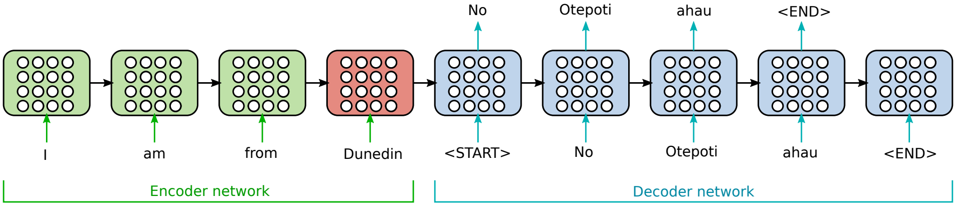
$$p(y_{i+1}|y_{i-D}\dots y_D, x) \qquad \leftarrow x \text{ is the input sentence.}$$



Attention (Bahdanau, Cho and Bengio, 2014)

Attention

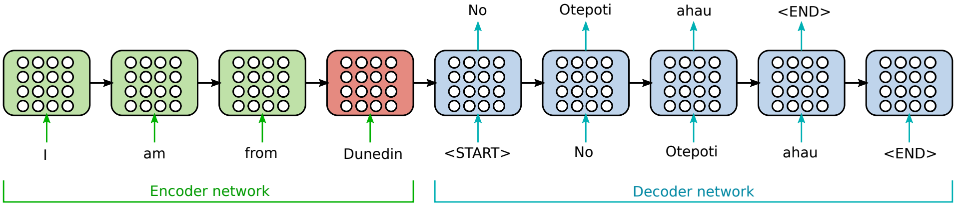
The Seq2Seq model just described still suffers from the ‘bottleneck problem’:



Attention

The Seq2Seq model just described still suffers from the 'bottleneck problem':

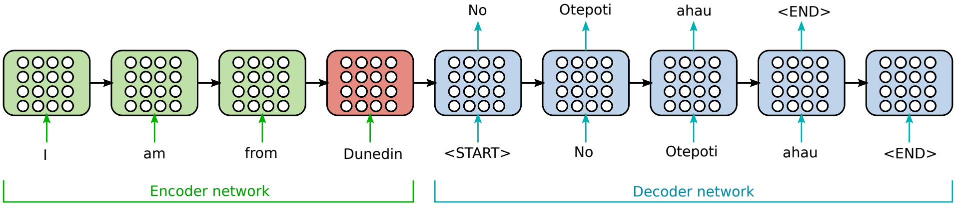
- The decoder network gets all its information about the input sentence from the final state of the encoder network.



Attention

The Seq2Seq model just described still suffers from the ‘bottleneck problem’:

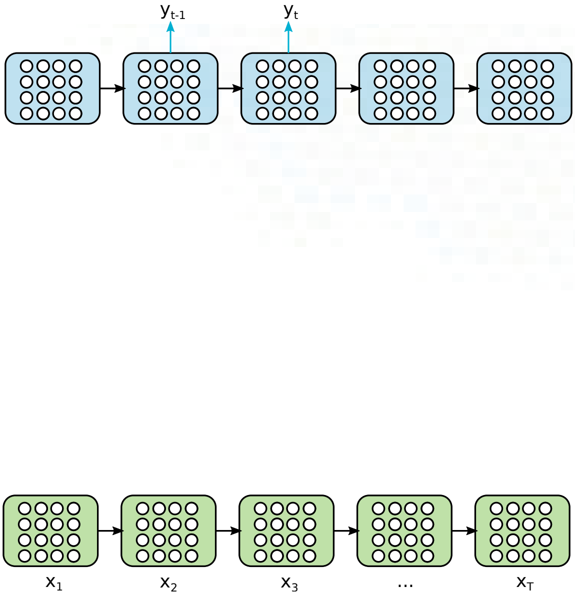
- The decoder network gets all its information about the input sentence from the final state of the encoder network.
- The attention mechanism radically changed that.



Attention

In a Seq2Seq model with attention, when the decoder network is producing an output word y_t, it uses:

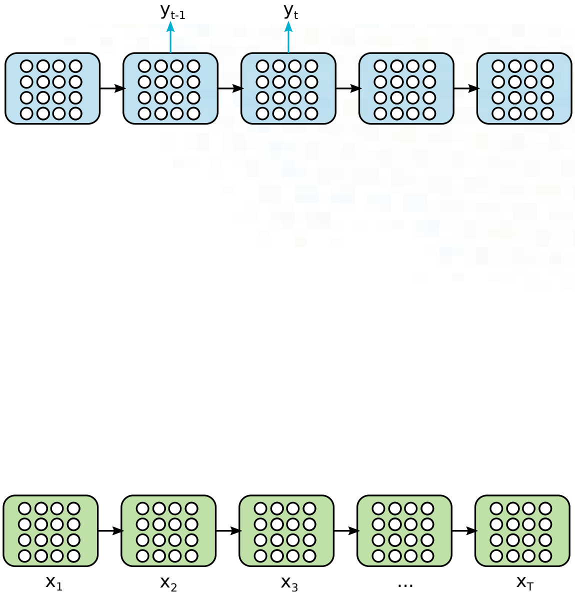
- Its own previous internal state;



Attention

In a Seq2Seq model with attention, when the decoder network is producing an output word y_t, it uses:

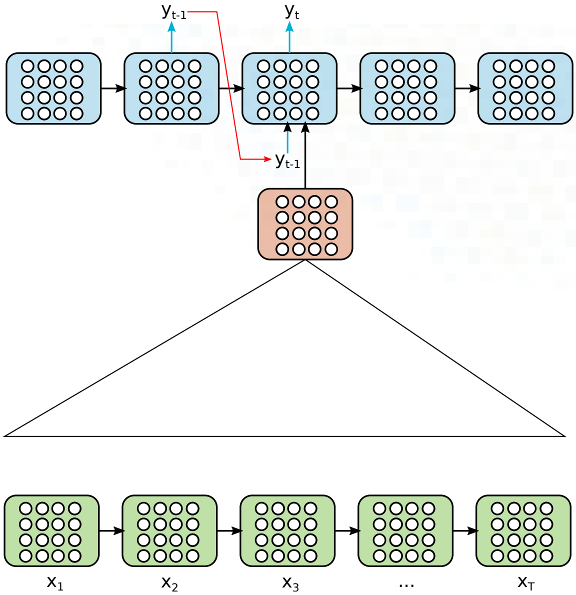
- Its own previous internal state;
- The last word it generated;



Attention

In a Seq2Seq model with attention, when the decoder network is producing an output word y_t, it uses:

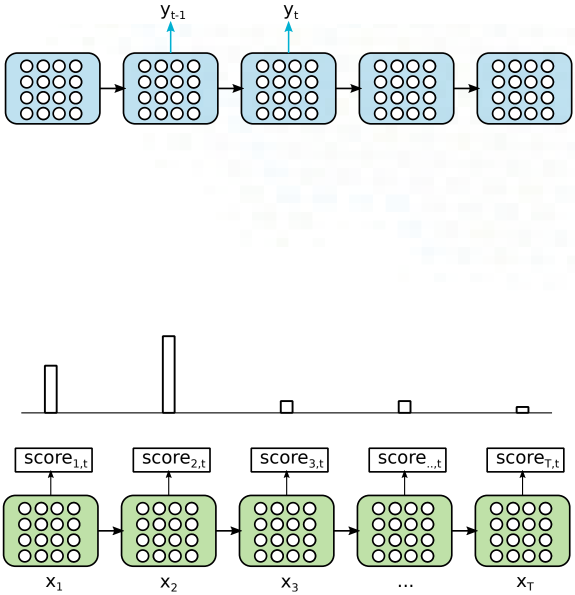
- Its own previous internal state;
- The last word it generated;
- A *weighted sum* of the internal states of the encoder network.



Attention

In a Seq2Seq model with attention, when the decoder network is producing an output word y_t, it uses:

- Its own previous internal state;
- The last word it generated;
- A *weighted sum* of the internal states of the encoder network.

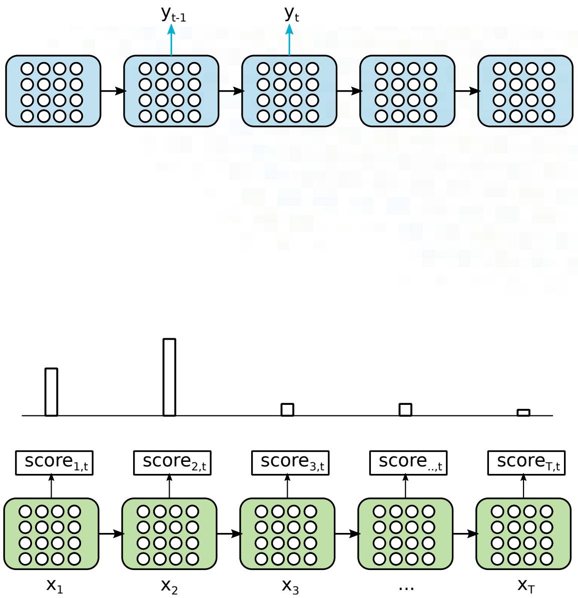


Attention

In a Seq2Seq model with attention, when the decoder network is producing an output word y_t, it uses:

- Its own previous internal state;
- The last word it generated;
- A *weighted sum* of the internal states of the encoder network.

The decoder can now 'pay attention' to different parts of the encoded text at different times.

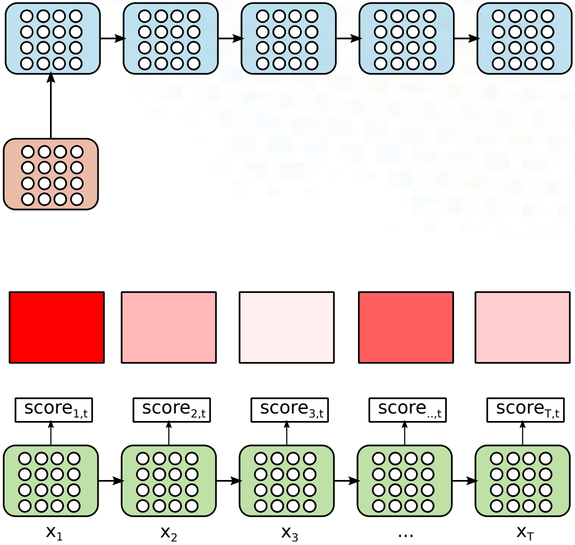


Attention

In a Seq2Seq model with attention, when the decoder network is producing an output word y_t, it uses:

- Its own previous internal state;
- The last word it generated;
- A *weighted sum* of the internal states of the encoder network.

The decoder can now ‘pay attention’ to different parts of the encoded text at different times.

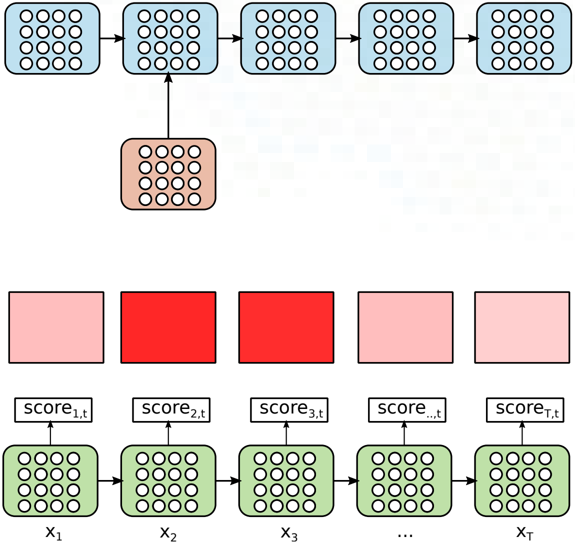


Attention

In a Seq2Seq model with attention, when the decoder network is producing an output word y_t, it uses:

- Its own previous internal state;
- The last word it generated;
- A *weighted sum* of the internal states of the encoder network.

The decoder can now 'pay attention' to different parts of the encoded text at different times.

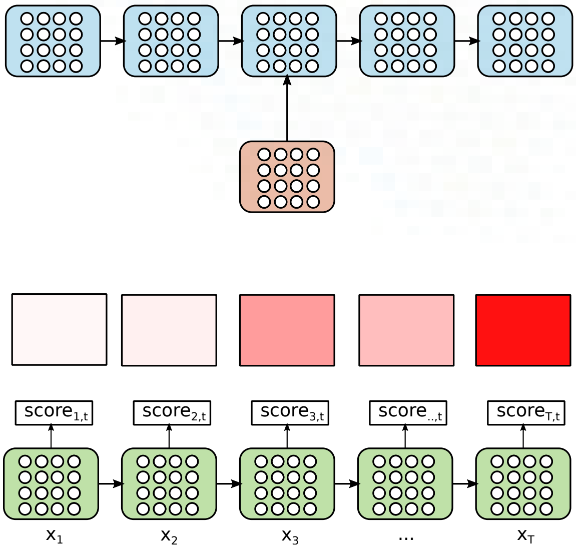


Attention

In a Seq2Seq model with attention, when the decoder network is producing an output word y_t, it uses:

- Its own previous internal state;
- The last word it generated;
- A *weighted sum* of the internal states of the encoder network.

The decoder can now ‘pay attention’ to different parts of the encoded text at different times.

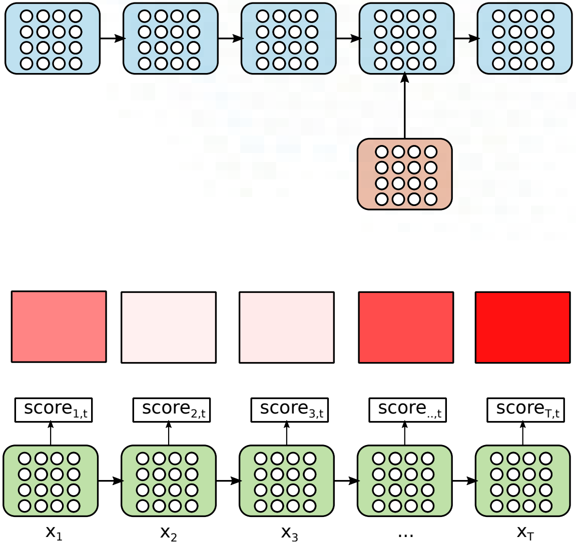


Attention

In a Seq2Seq model with attention, when the decoder network is producing an output word y_t, it uses:

- Its own previous internal state;
- The last word it generated;
- A *weighted sum* of the internal states of the encoder network.

The decoder can now 'pay attention' to different parts of the encoded text at different times.

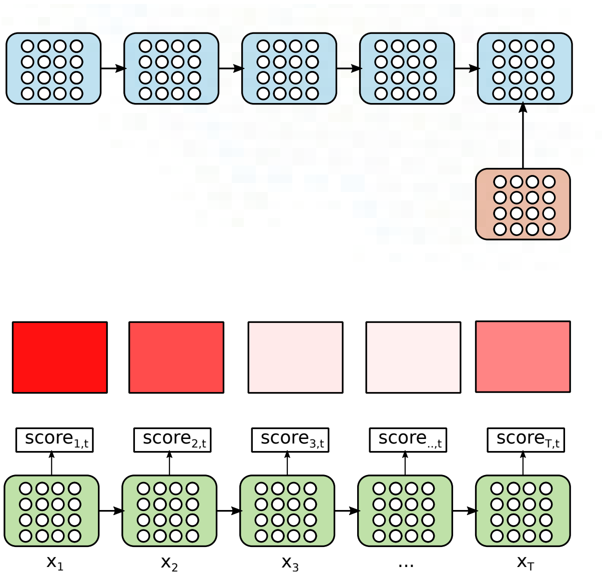


Attention

In a Seq2Seq model with attention, when the decoder network is producing an output word y_t, it uses:

- Its own previous internal state;
- The last word it generated;
- A *weighted sum* of the internal states of the encoder network.

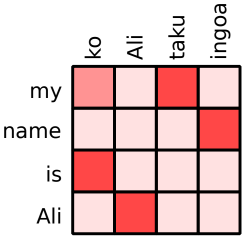
The decoder can now 'pay attention' to different parts of the encoded text at different times.



Attention and word alignment

In machine translation, a key thing is to figure out which words in the input sentence map onto which words in the output sentence.

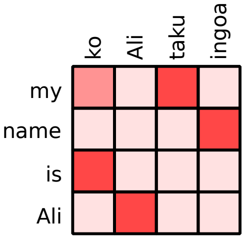
We often use an alignment chart to visualise this.



Attention and word alignment

In machine translation, a key thing is to figure out which words in the input sentence map onto which words in the output sentence.

We often use an alignment chart to visualise this.



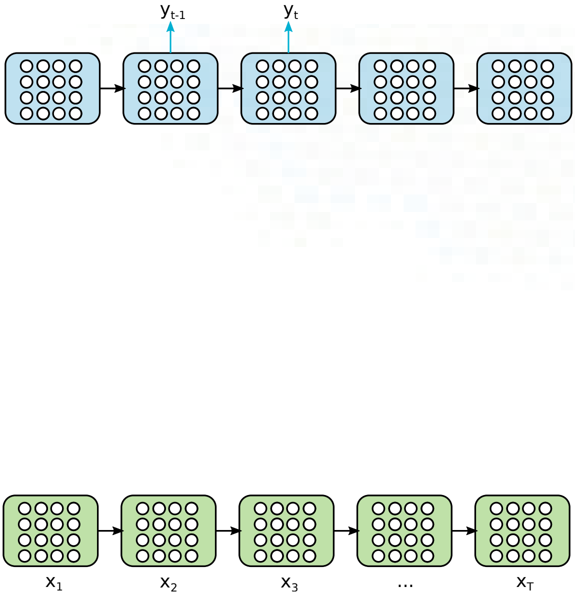
A Seq2Seq model using attention learns alignments by itself.

Transformers

Transformers

Attention lets the decoder access all the encoder words at once, when predicting its next word.

- It avoids *one* bottleneck problem.

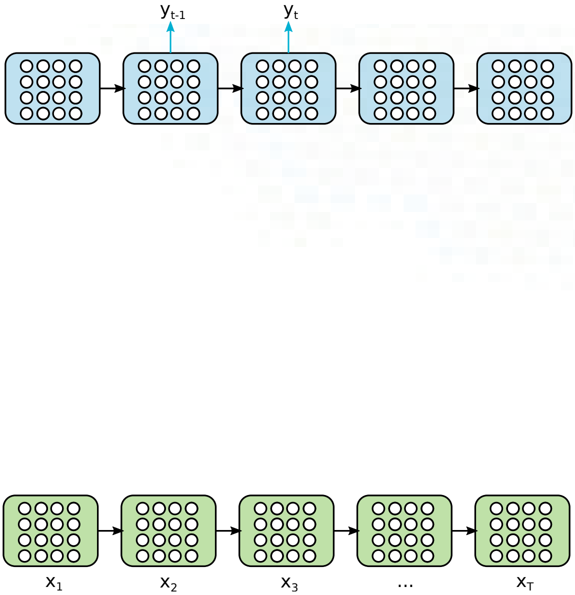


Transformers

Attention lets the decoder access all the encoder words at once, when predicting its next word.

- It avoids *one* bottleneck problem.

But there are two other bottlenecks in our Seq2Seq network: one in the encoder, one in the decoder.



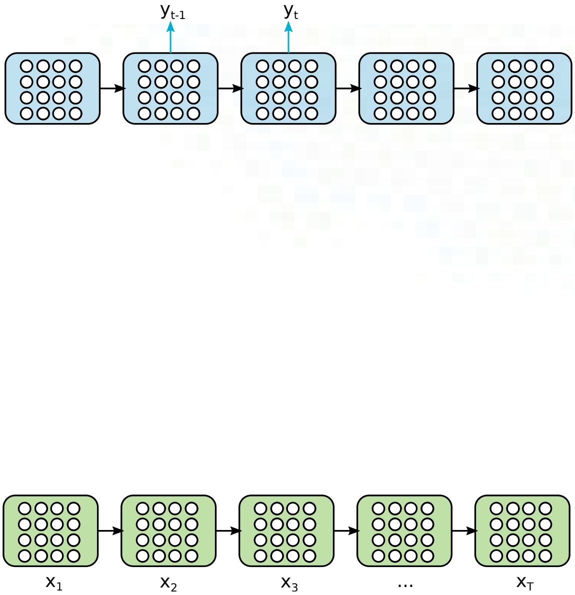
Transformers

Attention lets the decoder access all the encoder words at once, when predicting its next word.

- It avoids *one* bottleneck problem.

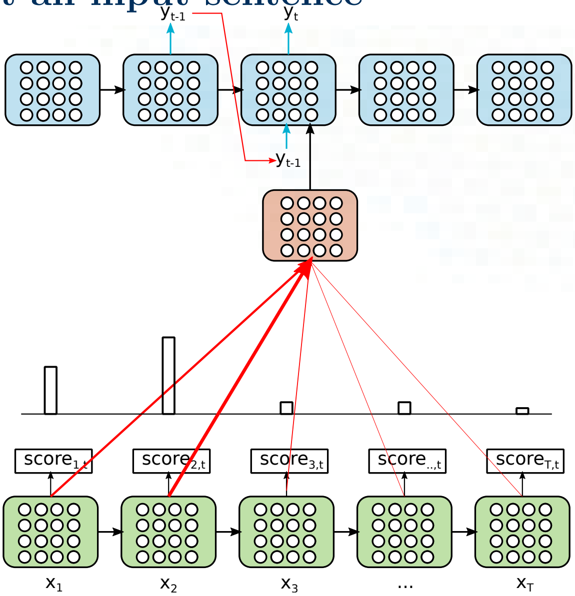
But there are two other bottlenecks in our Seq2Seq network: one in the encoder, one in the decoder.

- Google's 'transformer' model replaced each of these with a self-attention mechanism.



Using attention to represent an input sentence

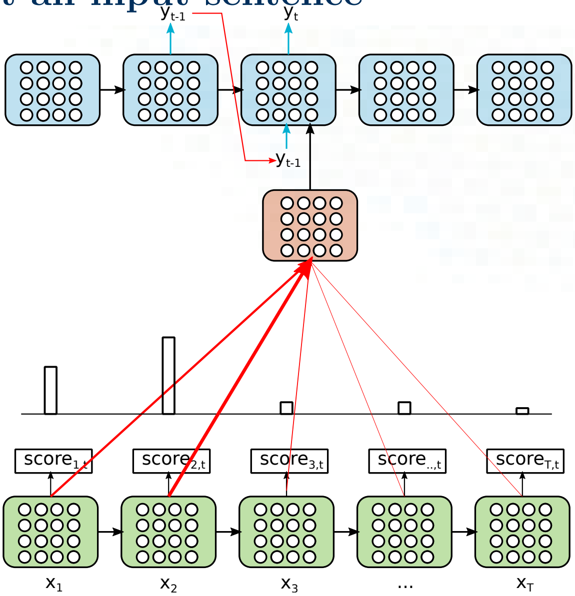
In a transformer network, we begin by using attention *to encode the input sentence*.



Using attention to represent an input sentence

In a transformer network, we begin by using attention *to encode the input sentence*.

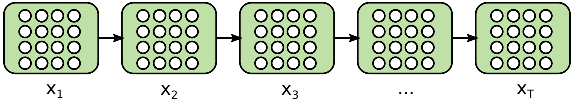
- We do this by building an attention network that learns to map the input sentence *onto itself*.



Using attention to represent an input sentence

In a transformer network, we begin by using attention *to encode the input sentence*.

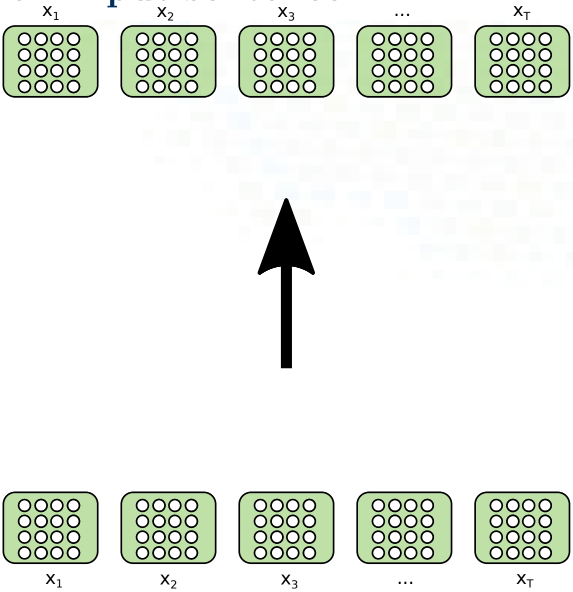
- We do this by building an attention network that learns to map the input sentence *onto itself*.



Using attention to represent an input sentence

In a transformer network, we begin by using attention *to encode the input sentence*.

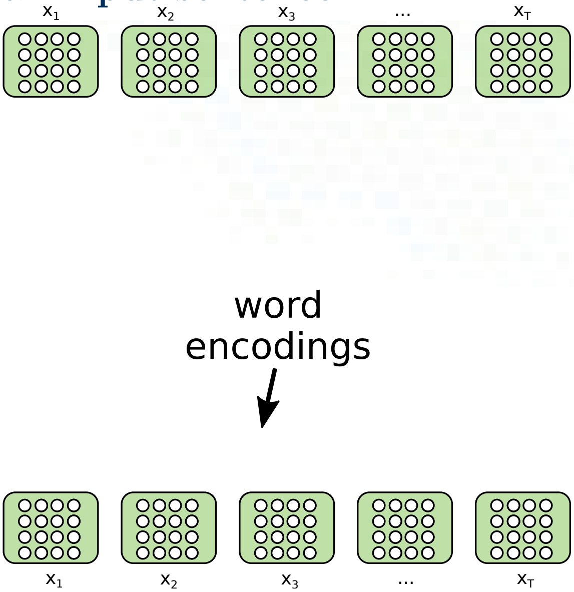
- We do this by building an attention network that learns to map the input sentence *onto itself*.



Using attention to represent an input sentence

In a transformer network, we begin by using attention *to encode the input sentence*.

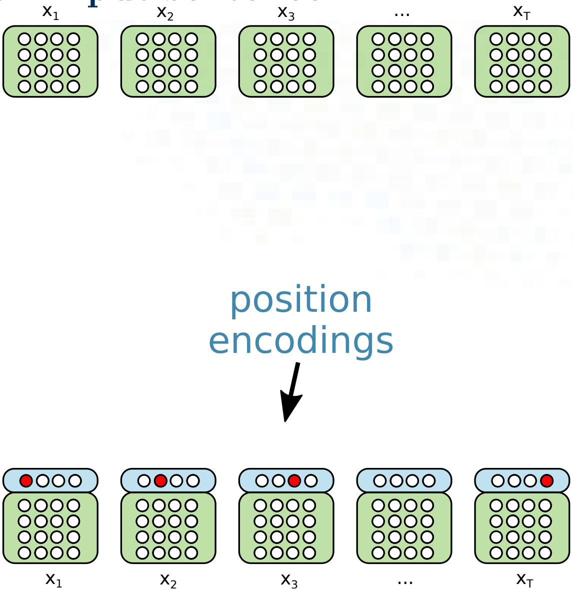
- We do this by building an attention network that learns to map the input sentence *onto itself*.
- There's no longer a RNN on the input: we just use word encodings.



Using attention to represent an input sentence

In a transformer network, we begin by using attention *to encode the input sentence*.

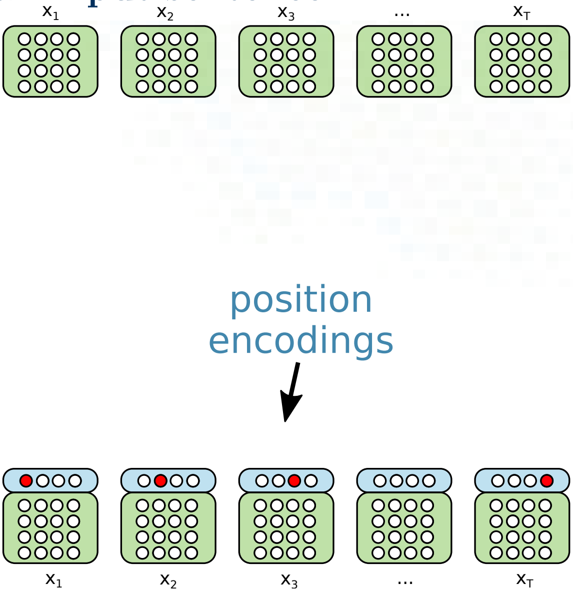
- We do this by building an attention network that learns to map the input sentence *onto itself*.
- There's no longer a RNN on the input: we just use word encodings.
- To compensate, we have to encode the position of input words explicitly.



Using attention to represent an input sentence

In a transformer network, we begin by using attention *to encode the input sentence*.

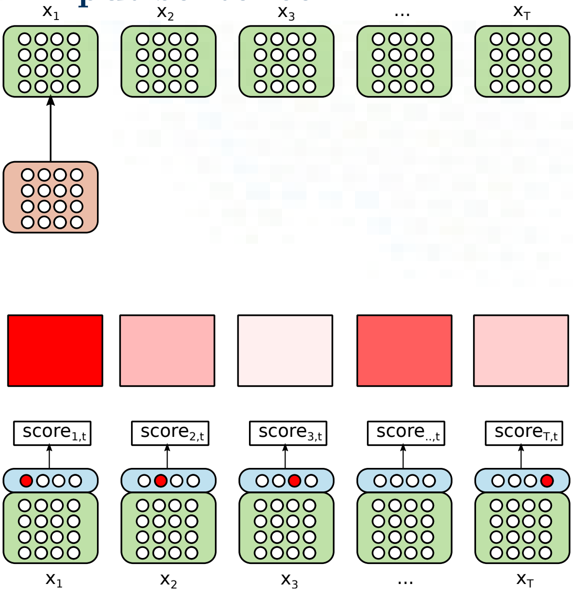
- We do this by building an attention network that learns to map the input sentence *onto itself*.
- There's no longer a RNN on the input: we just use word encodings.
- To compensate, we have to encode the position of input words explicitly.
- The encoder network now learns a weighted sum of word encodings for each word in the sentence.



Using attention to represent an input sentence

In a transformer network, we begin by using attention *to encode the input sentence*.

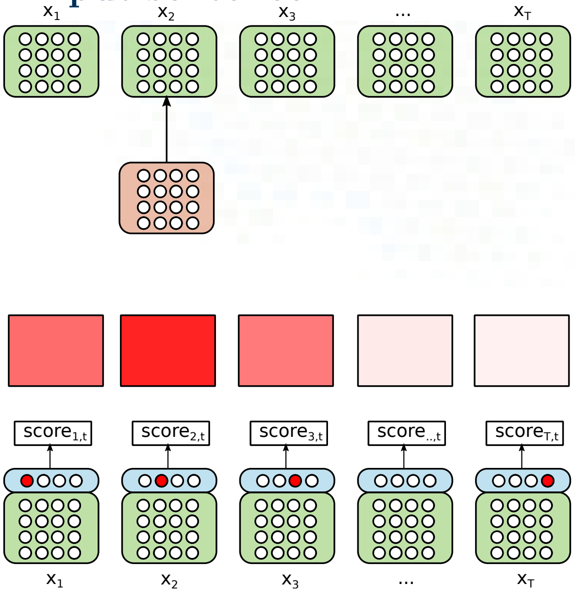
- We do this by building an attention network that learns to map the input sentence *onto itself*.
- There's no longer a RNN on the input: we just use word encodings.
- To compensate, we have to encode the position of input words explicitly.
- The encoder network now learns a weighted sum of word encodings for each word in the sentence.



Using attention to represent an input sentence

In a transformer network, we begin by using attention *to encode the input sentence*.

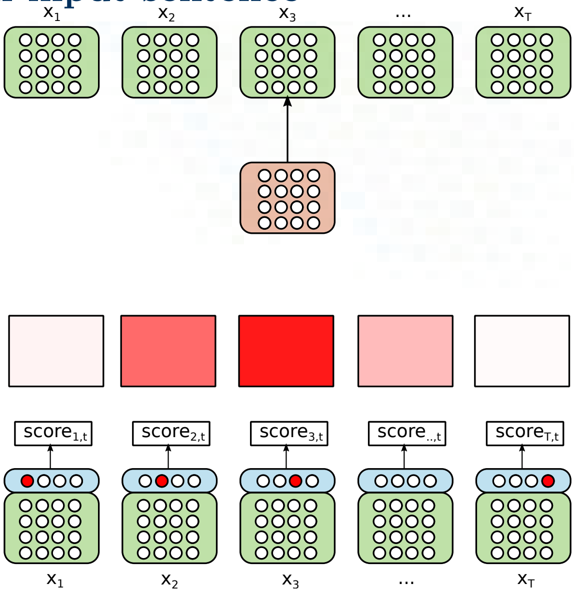
- We do this by building an attention network that learns to map the input sentence *onto itself*.
- There's no longer a RNN on the input: we just use word encodings.
- To compensate, we have to encode the position of input words explicitly.
- The encoder network now learns a weighted sum of word encodings for each word in the sentence.



Using attention to represent an input sentence

In a transformer network, we begin by using attention *to encode the input sentence*.

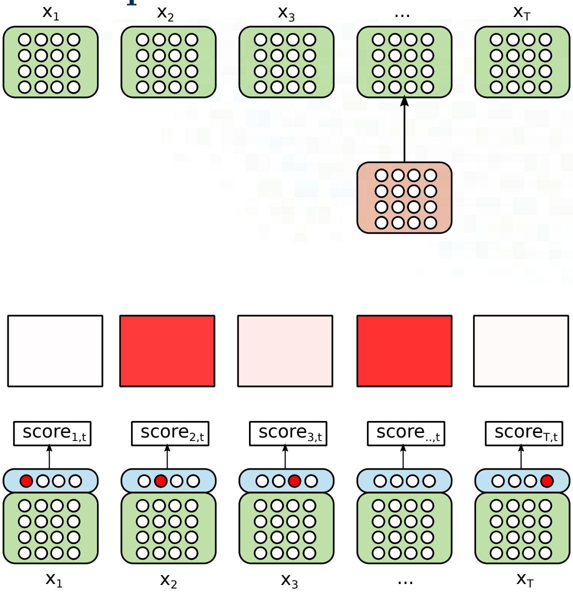
- We do this by building an attention network that learns to map the input sentence *onto itself*.
- There's no longer a RNN on the input: we just use word encodings.
- To compensate, we have to encode the position of input words explicitly.
- The encoder network now learns a weighted sum of word encodings for each word in the sentence.



Using attention to represent an input sentence

In a transformer network, we begin by using attention *to encode the input sentence*.

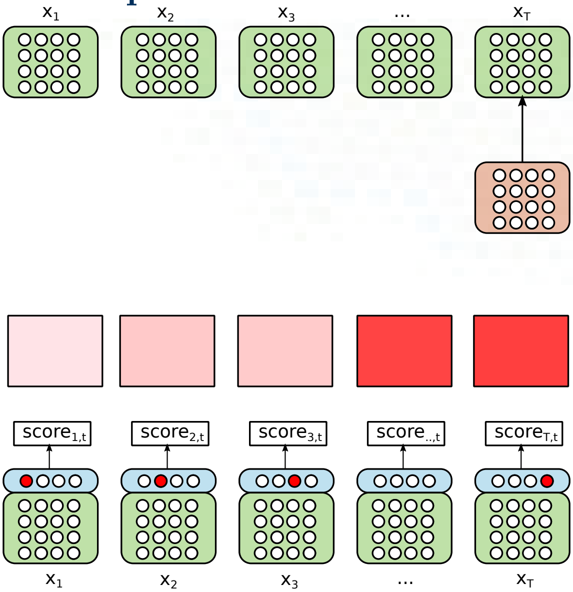
- We do this by building an attention network that learns to map the input sentence *onto itself*.
- There's no longer a RNN on the input: we just use word encodings.
- To compensate, we have to encode the position of input words explicitly.
- The encoder network now learns a weighted sum of word encodings for each word in the sentence.



Using attention to represent an input sentence

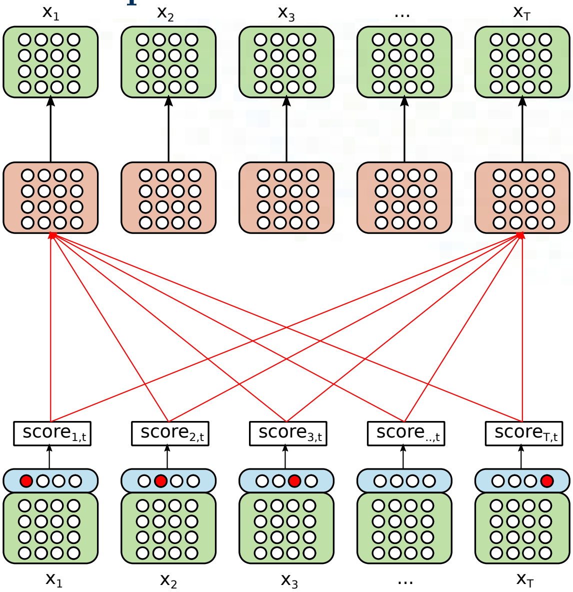
In a transformer network, we begin by using attention *to encode the input sentence*.

- We do this by building an attention network that learns to map the input sentence *onto itself*.
- There's no longer a RNN on the input: we just use word encodings.
- To compensate, we have to encode the position of input words explicitly.
- The encoder network now learns a weighted sum of word encodings for each word in the sentence.



Using attention to represent an input sentence

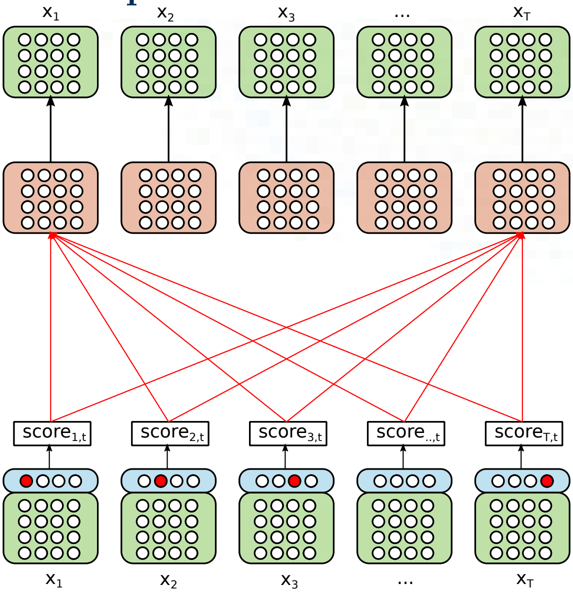
A big advantage of using attention is that learning can be done *in parallel* for each input word.



Using attention to represent an input sentence

A big advantage of using attention is that learning can be done *in parallel* for each input word.

For a RNN, we have to do word x_1, <u>then</u> x_2, <u>then</u> x_3...

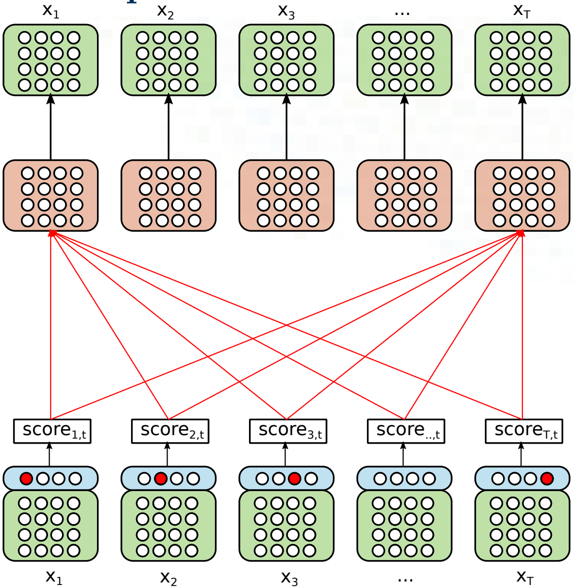


Using attention to represent an input sentence

A big advantage of using attention is that learning can be done *in parallel* for each input word.

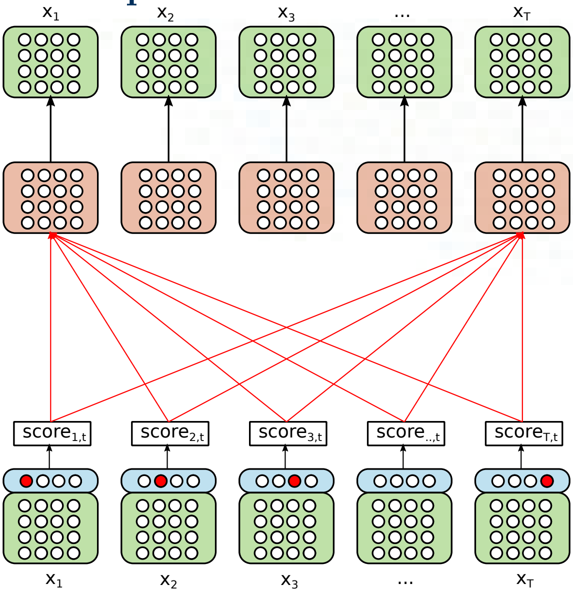
For a RNN, we have to do word x_1, <u>then</u> x_2, <u>then</u> $x_3 \ldots$

The network is basically an *autoencoder*—but without recurrent connections.



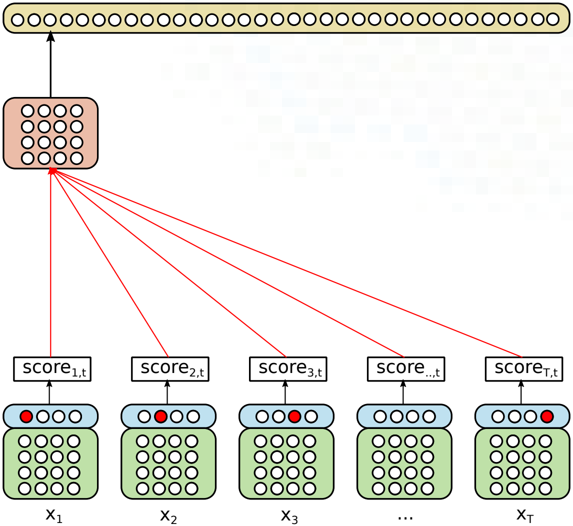
Using attention to represent an input sentence

The new representation of each input word is then passed through a feed-forward layer, to allow the network to learn richer representations.



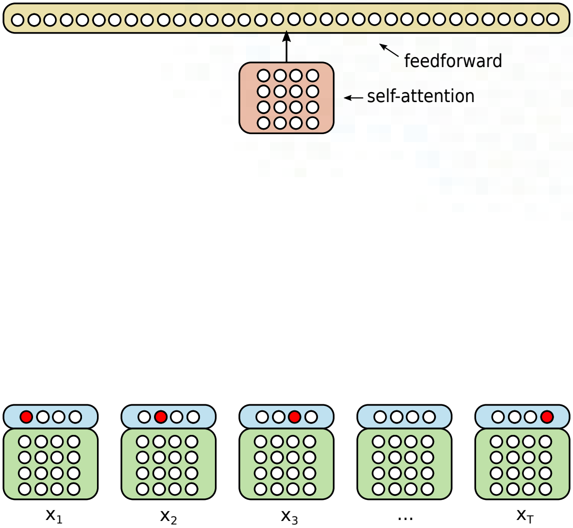
Using attention to represent an input sentence

The new representation of each input word is then passed through a feed-forward layer, to allow the network to learn richer representations.



Using attention to represent an input sentence

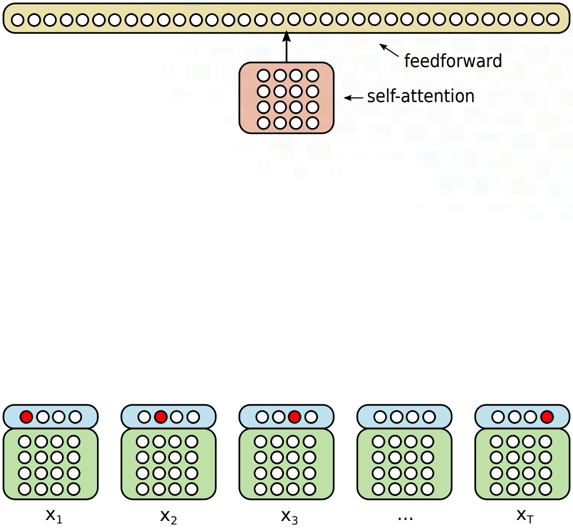
The new representation of each input word is then passed through a feed-forward layer, to allow the network to learn richer representations.



Using attention to represent an input sentence

The new representation of each input word is then passed through a feed-forward layer, to allow the network to learn richer representations.

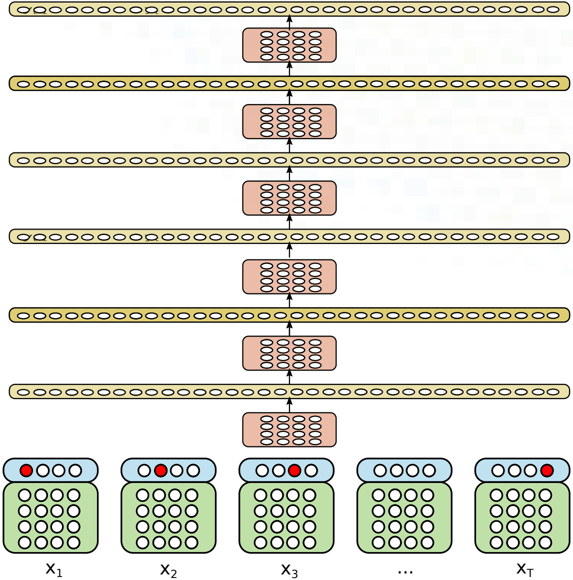
And then this pattern of self-attention and feed-forward is repeated 6 times.



Using attention to represent an input sentence

The new representation of each input word is then passed through a feed-forward layer, to allow the network to learn richer representations.

And then this pattern of self-attention and feed-forward is repeated 6 times.

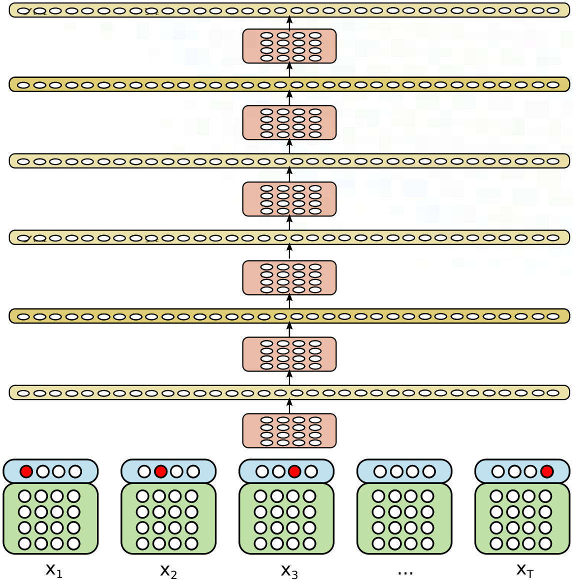


Using attention to represent an input sentence

The new representation of each input word is then passed through a feed-forward layer, to allow the network to learn richer representations.

And then this pattern of self-attention and feed-forward is repeated 6 times.

On top of that, there are *multiple* whole attentional systems.



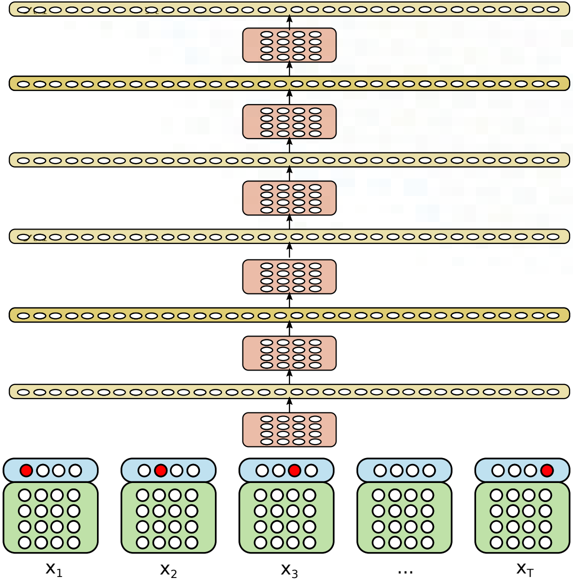
Using attention to represent an input sentence

The new representation of each input word is then passed through a feed-forward layer, to allow the network to learn richer representations.

And then this pattern of self-attention and feed-forward is repeated 6 times.

On top of that, there are *multiple* whole attentional systems.

- We want to be able to learn multiple *independent features* of any given word.



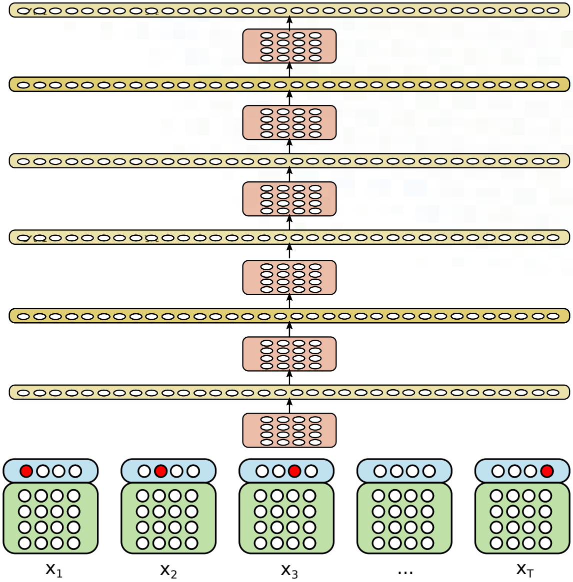
Using attention to represent an input sentence

The new representation of each input word is then passed through a feed-forward layer, to allow the network to learn richer representations.

And then this pattern of self-attention and feed-forward is repeated 6 times.

On top of that, there are *multiple* whole attentional systems.

- We want to be able to learn multiple *independent features* of any given word.
- A single attentional network just computes a weighted sum: it can't do that.



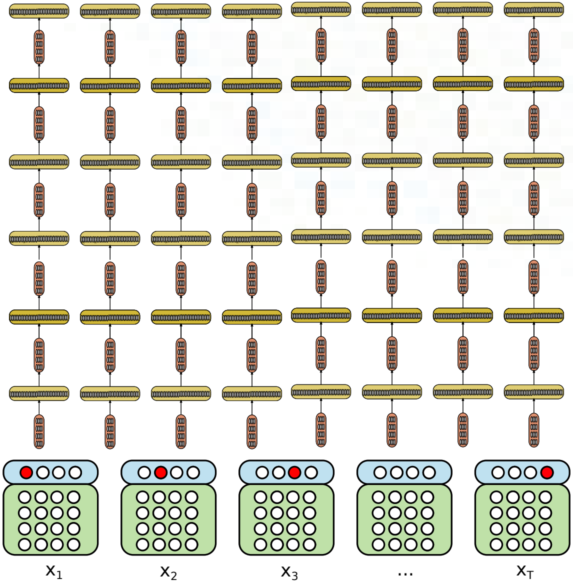
Using attention to represent an input sentence

The new representation of each input word is then passed through a feed-forward layer, to allow the network to learn richer representations.

And then this pattern of self-attention and feed-forward is repeated 6 times.

On top of that, there are *multiple* whole attentional systems.

- We want to be able to learn multiple *independent features* of any given word.
- A single attentional network just computes a weighted sum: it can't do that.



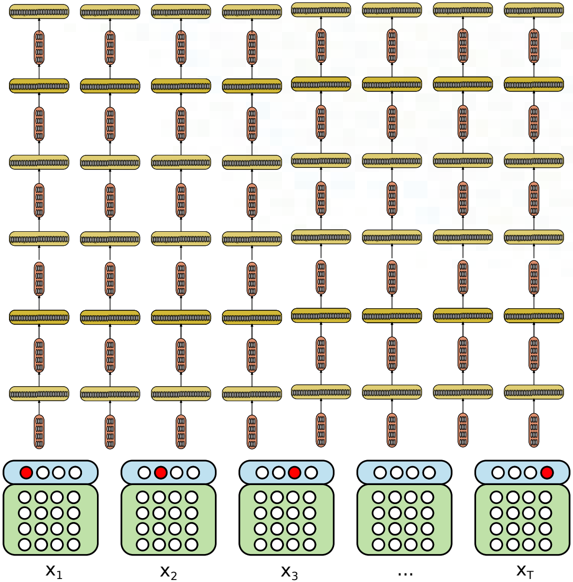
Multi-head attention

The new representation of each input word is then passed through a feed-forward layer, to allow the network to learn richer representations.

And then this pattern of self-attention and feed-forward is repeated 6 times.

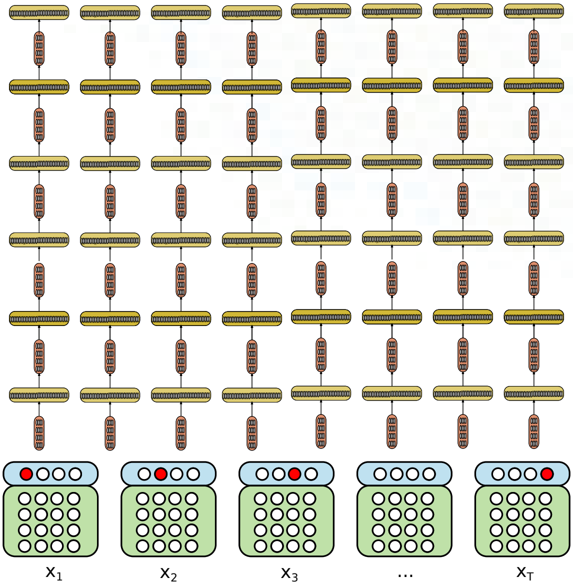
On top of that, there are *multiple* whole attentional systems.

- We want to be able to learn multiple *independent features* of any given word.
- A single attentional network just computes a weighted sum: it can't do that.



Multi-head attention

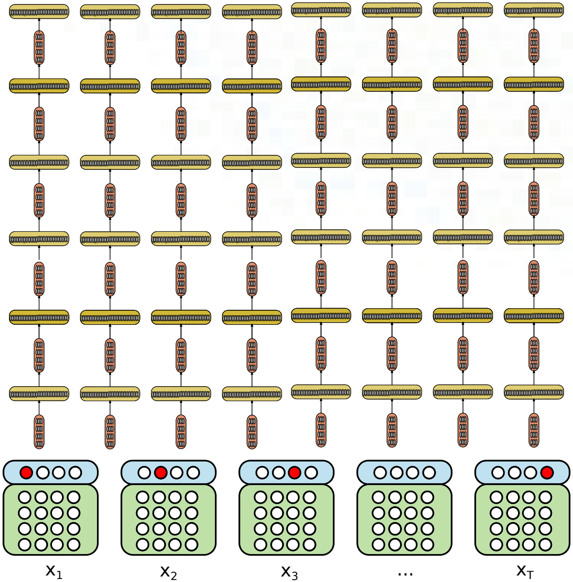
Each attentional system is associated with its own ‘key’.



Multi-head attention

Each attentional system is associated with its own 'key'.

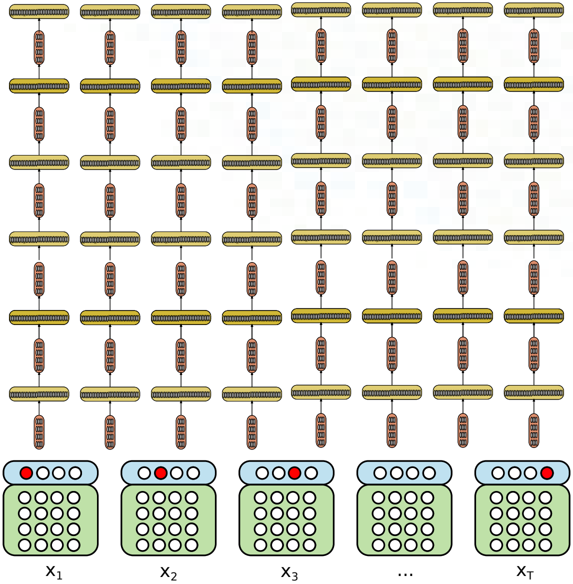
- Recall the 'query'/'key'/'value' notation mentioned earlier...
- A key is a vector (initialised randomly), that is multiplied by each word encoding to create the 'query' posed to the set of input word vectors.



Multi-head attention

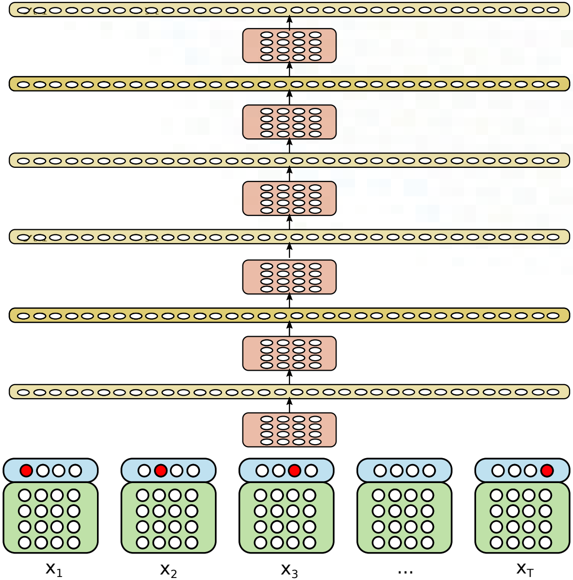
Each attentional system is associated with its own 'key'.

- Recall the 'query'/'key'/'value' notation mentioned earlier...
- A key is a vector (initialised randomly), that is multiplied by each word encoding to create the 'query' posed to the set of input word vectors.
- Each feedforward layer acts like a dedicated 'key' for the next layer. (Whose value is learned through backpropagation.)



Let's have just one head!

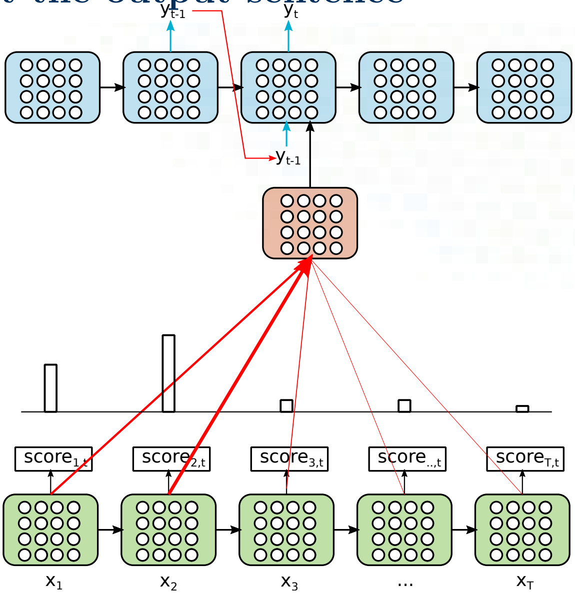
...to keep things manageable!



Using attention to represent the output sentence

We can use a self-attention mechanism to represent the output sentence too.

- That is, the *decoder* can use self-attention.

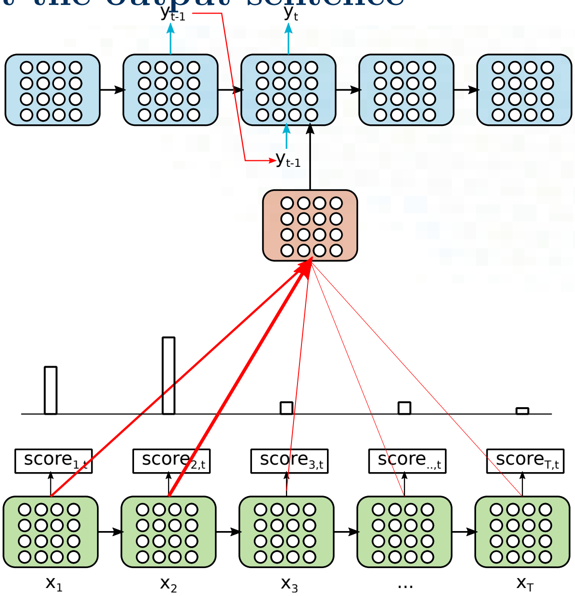


Using attention to represent the output sentence

We can use a self-attention mechanism to represent the output sentence too.

- That is, the *decoder* can use self-attention.

The decoder must predict the next word, given the words produced so far.



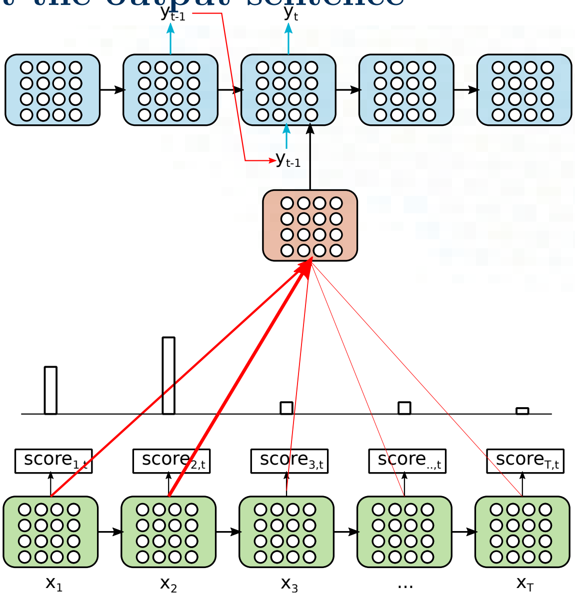
Using attention to represent the output sentence

We can use a self-attention mechanism to represent the output sentence too.

- That is, the *decoder* can use self-attention.

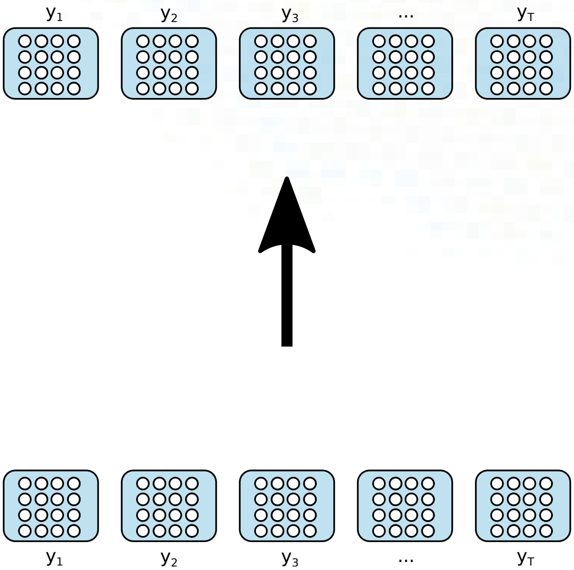
The decoder must predict the next word, given the words produced so far.

We can use a self-attention mechanism to represent the words produced so far (and their importance).



Using attention to represent the output sentence

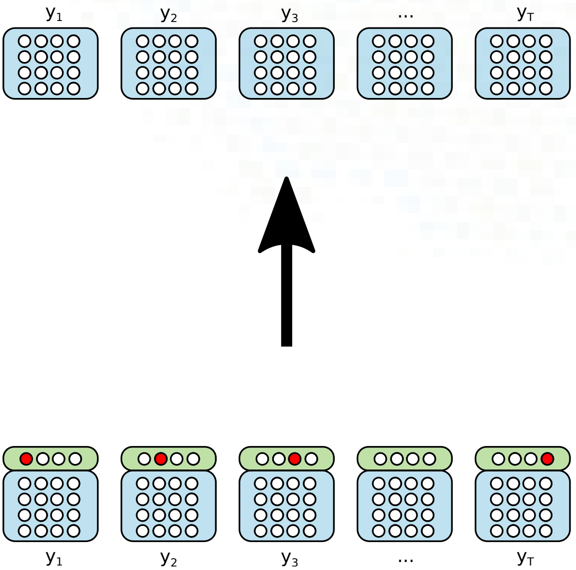
The new decoder learns to represent each word in the output sentence as a weighted sum of word encodings.



Using attention to represent the output sentence

The new decoder learns to represent each word in the output sentence as a weighted sum of word encodings.

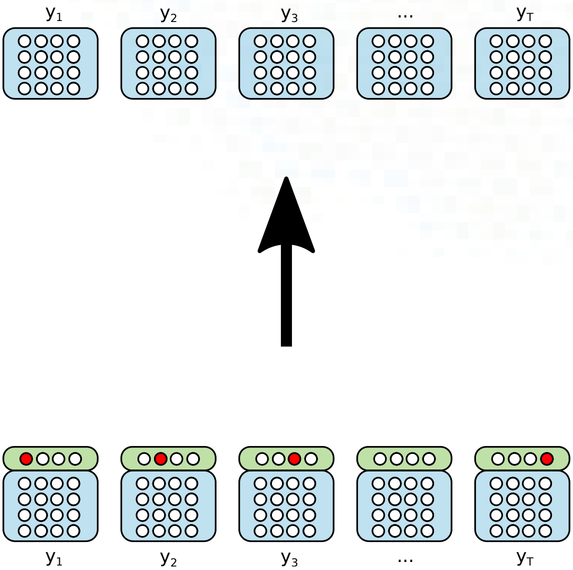
- Again, we need a positional encoding. . .



Using attention to represent the output sentence

The new decoder learns to represent each word in the output sentence as a weighted sum of word encodings.

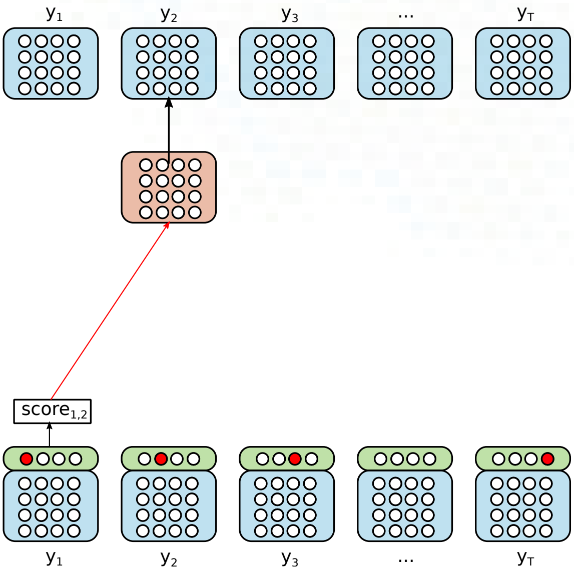
- Again, we need a positional encoding. . .
- Importantly, we can only use the words that have already been produced.



Using attention to represent the output sentence

The new decoder learns to represent each word in the output sentence as a weighted sum of word encodings.

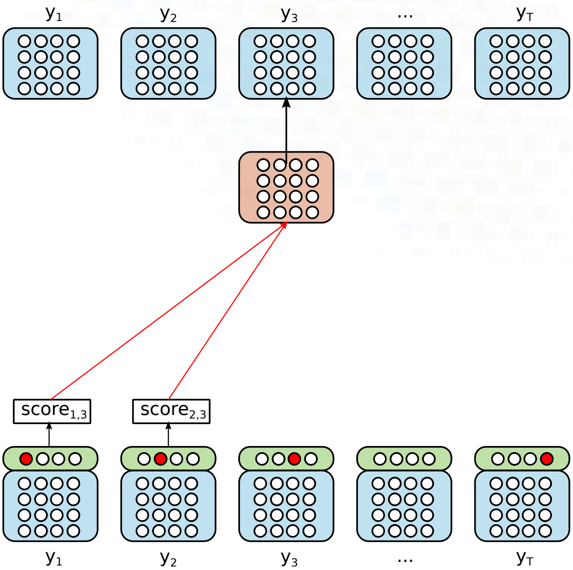
- Again, we need a positional encoding...
- Importantly, we can only use the words that have already been produced.



Using attention to represent the output sentence

The new decoder learns to represent each word in the output sentence as a weighted sum of word encodings.

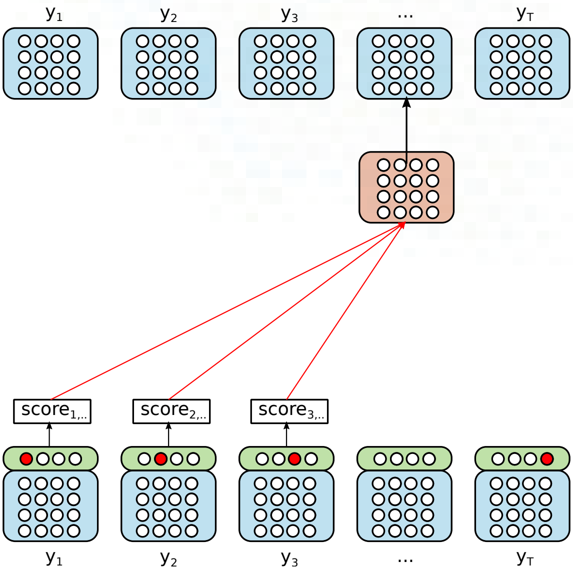
- Again, we need a positional encoding. . .
- Importantly, we can only use the words that have already been produced.



Using attention to represent the output sentence

The new decoder learns to represent each word in the output sentence as a weighted sum of word encodings.

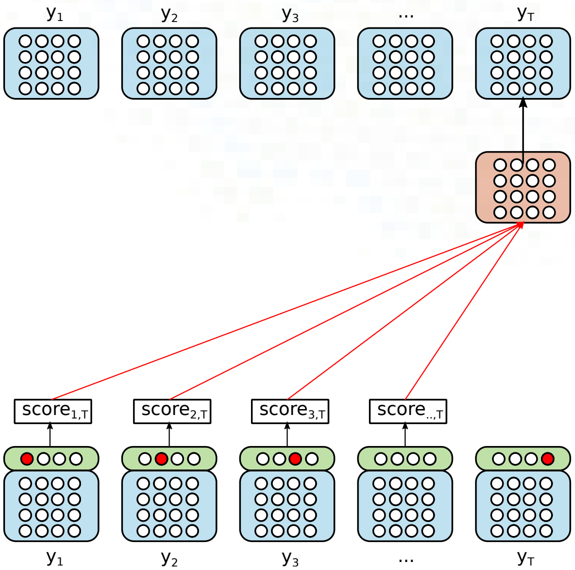
- Again, we need a positional encoding...
- Importantly, we can only use the words that have already been produced.



Using attention to represent the output sentence

The new decoder learns to represent each word in the output sentence as a weighted sum of word encodings.

- Again, we need a positional encoding...
- Importantly, we can only use the words that have already been produced.



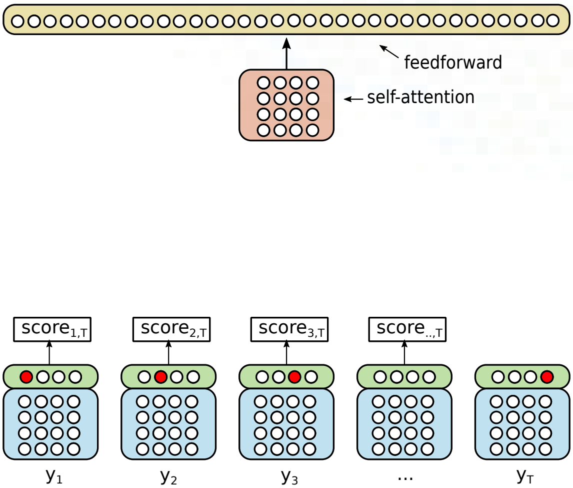
Using attention to represent the output sentence

The new decoder learns to represent each word in the output sentence as a weighted sum of word encodings.

- Again, we need a positional encoding. . .
- Importantly, we can only use the words that have already been produced.

As with the encoder,

- we add a feedforward layer. . .



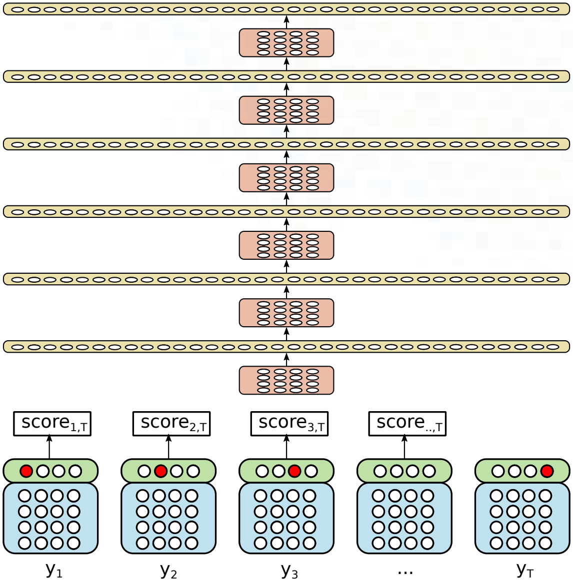
Using attention to represent the output sentence

The new decoder learns to represent each word in the output sentence as a weighted sum of word encodings.

- Again, we need a positional encoding. . .
- Importantly, we can only use the words that have already been produced.

As with the encoder,

- we add a feedforward layer. . .
- and iterate six times. . .



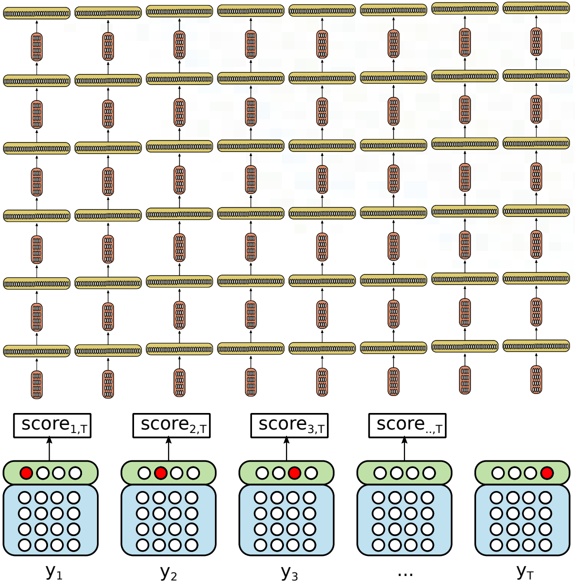
Using attention to represent the output sentence

The new decoder learns to represent each word in the output sentence as a weighted sum of word encodings.

- Again, we need a positional encoding...
- Importantly, we can only use the words that have already been produced.

As with the encoder,

- we add a feedforward layer...
- and iterate six times...
- And implement multi-headed attention.



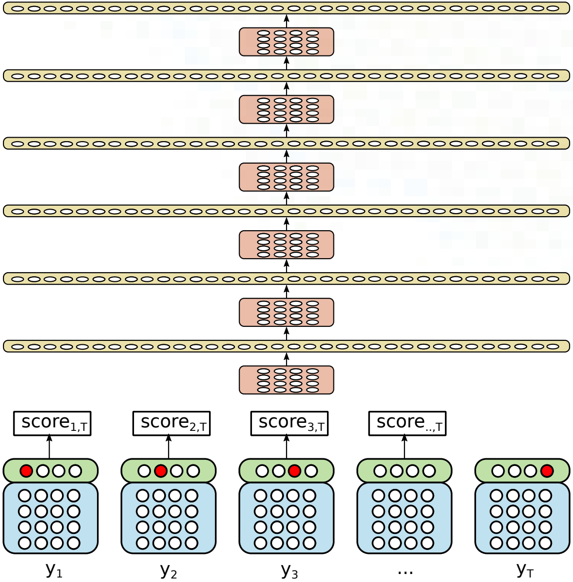
Using attention to represent the output sentence

The new decoder learns to represent each word in the output sentence as a weighted sum of word encodings.

- Again, we need a positional encoding...
- Importantly, we can only use the words that have already been produced.

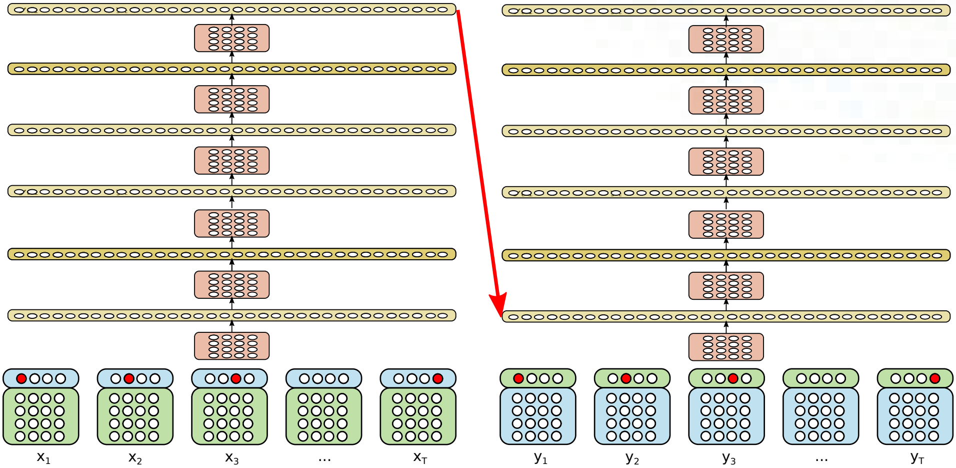
As with the encoder,

- we add a feedforward layer...
- and iterate six times...
- And implement multi-headed attention.
- But let's ignore that!



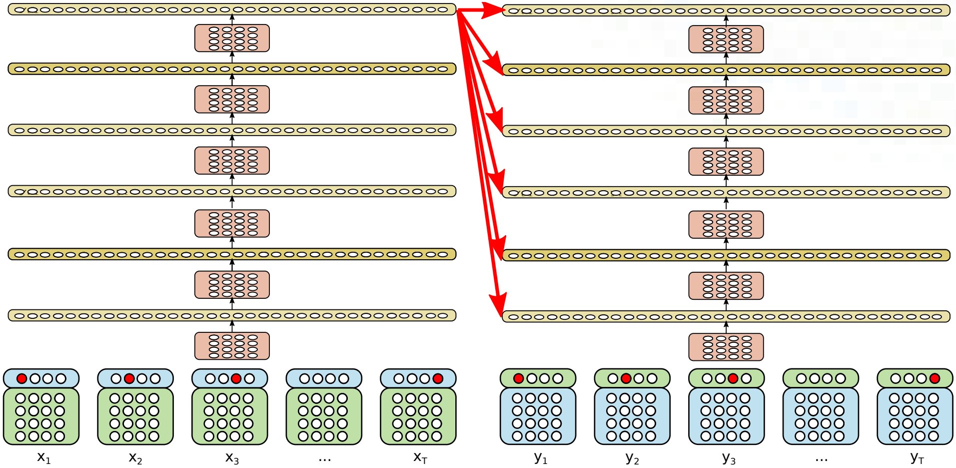
The complete transformer

The encoder and decoder networks are linked with an attentional mechanism, as before.



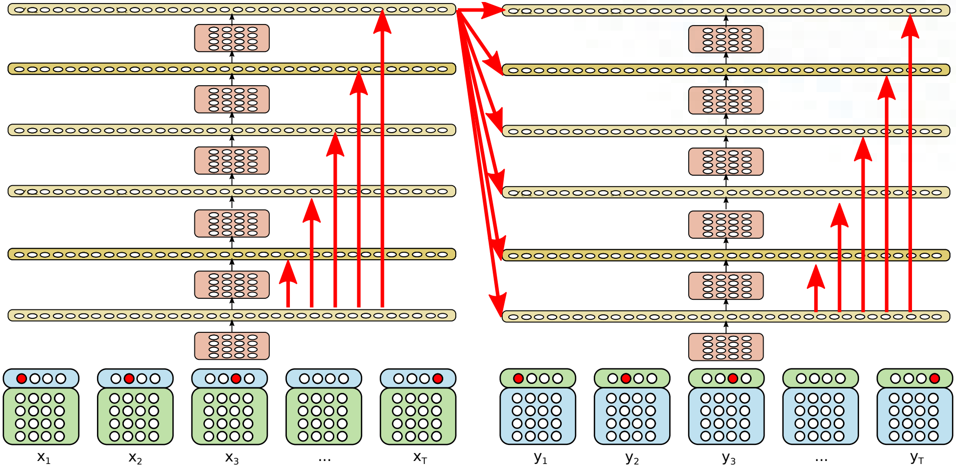
The complete transformer

Actually, this link uses residual connections ('skip connections') to each decoder layer.



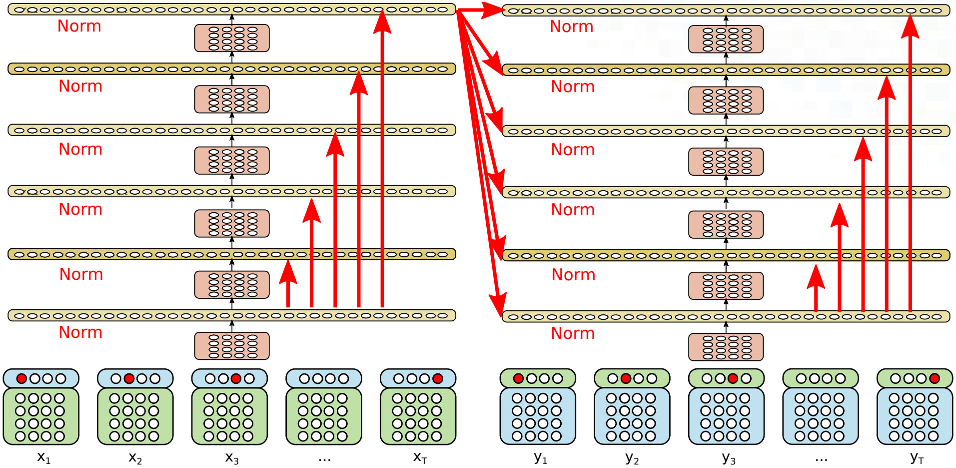
The complete transformer

Actually, residual connections are used within each encoder and decoder layer too.



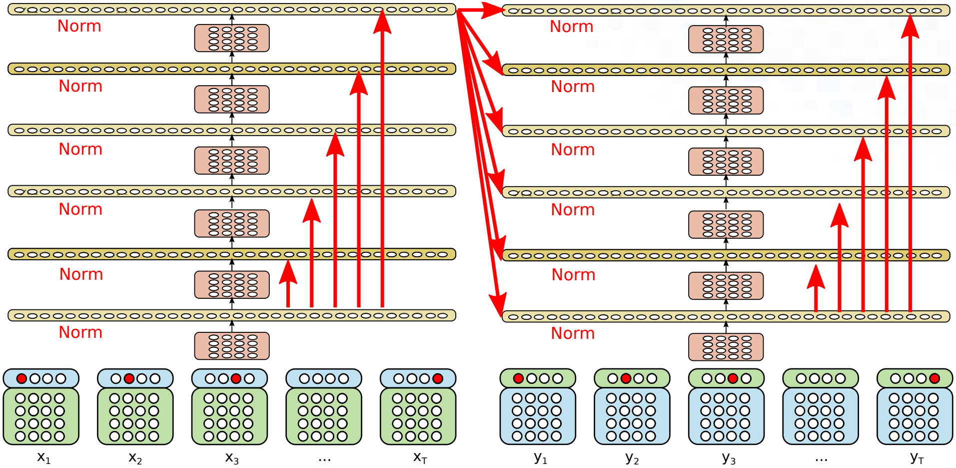
The complete transformer

And activity normalisation happens in each feedforward layer.



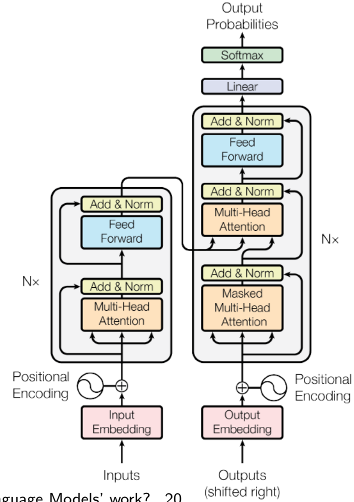
‘Attention is all you need’!

And activity normalisation happens in each feedforward layer.



‘Attention is all you need’

Here’s the diagram from the orginal transformer paper (Vaswani *et al.*, 2017).



Transformers and pre-trained language models

Transformers and pre-trained language models

Language models that use transformer-like methods (multi-headed attention in multiple layers) turned out to support pre-training—only possible for vision networks, hitherto.

Transformers and pre-trained language models

Language models that use transformer-like methods (multi-headed attention in multiple layers) turned out to support pre-training—only possible for vision networks, hitherto.

- Very large models can be trained on very large datasets. . .

Transformers and pre-trained language models

Language models that use transformer-like methods (multi-headed attention in multiple layers) turned out to support pre-training—only possible for vision networks, hitherto.

- Very large models can be trained on very large datasets. . .
- And then fine-tuned (much more quickly) for a specific task.

Transformers and pre-trained language models

Language models that use transformer-like methods (multi-headed attention in multiple layers) turned out to support pre-training—only possible for vision networks, hitherto.

- Very large models can be trained on very large datasets. . .
- And then fine-tuned (much more quickly) for a specific task.
  E.g. by adjusting the last n layers of weights.

Transformers and pre-trained language models

Language models that use transformer-like methods (multi-headed attention in multiple layers) turned out to support pre-training—only possible for vision networks, hitherto.

- Very large models can be trained on very large datasets...
- And then fine-tuned (much more quickly) for a specific task.
  E.g. by adjusting the last n layers of weights.
- This kind of fine-tuning is one variety of transfer learning.
  (There are many others.)

Transformers and pre-trained language models

Language models that use transformer-like methods (multi-headed attention in multiple layers) turned out to support pre-training—only possible for vision networks, hitherto.

- Very large models can be trained on very large datasets...
- And then fine-tuned (much more quickly) for a specific task.
  E.g. by adjusting the last n layers of weights.
- This kind of fine-tuning is one variety of transfer learning.
  (There are many others.)

After the success of transformers, a number of 'large, fine-tunable, general-purpose' language models were developed, that have since been very influential.

The idea of pre-training transformer-based language models

The idea of pre-training transformer-based language models

The practicality of pre-training for transformer-based models was first shown by researchers at OpenAI.

The idea of pre-training transformer-based language models

The practicality of pre-training for transformer-based models was first shown by researchers at OpenAI.

- They developed a model called GPT ('Generative Pre-Training'). Radford *et al.* (2018).

The idea of pre-training transformer-based language models

The practicality of pre-training for transformer-based models was first shown by researchers at OpenAI.

- They developed a model called GPT ('Generative Pre-Training').
  Radford *et al.* (2018).

The key insight in this paper:

- 'Natural language understanding comprises a wide range of diverse tasks such as textual entailment, question answering, semantic similarity assessment, and document classification.'

The idea of pre-training transformer-based language models

The practicality of pre-training for transformer-based models was first shown by researchers at OpenAI.

- They developed a model called GPT ('Generative Pre-Training'). Radford *et al.* (2018).

The key insight in this paper:

- 'Natural language understanding comprises a wide range of diverse tasks such as textual entailment, question answering, semantic similarity assessment, and document classification.'
- 'We demonstrate that large gains on these tasks can be realized by generative pre-training of a language model on a diverse corpus of unlabeled text, followed by discriminative fine-tuning on each specific task.'

GPT's architecture

GPT's architecture

The GPT model wasn't given any particular task: it was just trained to 'predict the next word'.

GPT's architecture

The GPT model wasn't given any particular task: it was just trained to 'predict the next word'.

- In its architecture, it's just the *decoder* portion of Vaswani *et al.*'s transformer.

GPT's architecture

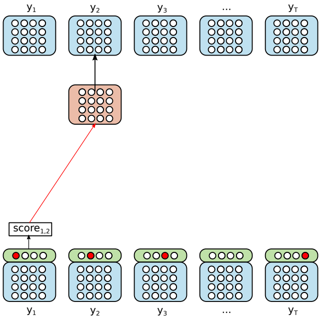
The GPT model wasn't given any particular task: it was just trained to 'predict the next word'.

- In its architecture, it's just the *decoder* portion of Vaswani *et al.*'s transformer. Recall: this model learns to predict the next word using masked self-attention.

GPT's architecture

The GPT model wasn't given any particular task: it was just trained to 'predict the next word'.

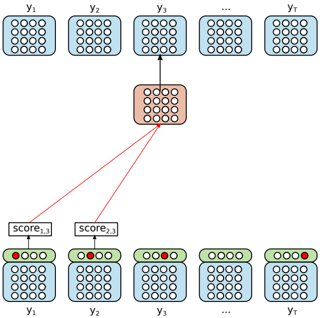
- In its architecture, it's just the *decoder* portion of Vaswani *et al.*'s transformer. Recall: this model learns to predict the next word using masked self-attention.



GPT's architecture

The GPT model wasn't given any particular task: it was just trained to 'predict the next word'.

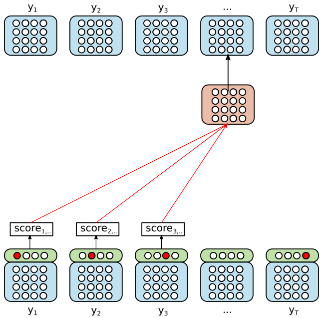
- In its architecture, it's just the *decoder* portion of Vaswani *et al.*'s transformer. Recall: this model learns to predict the next word using masked self-attention.



GPT's architecture

The GPT model wasn't given any particular task: it was just trained to 'predict the next word'.

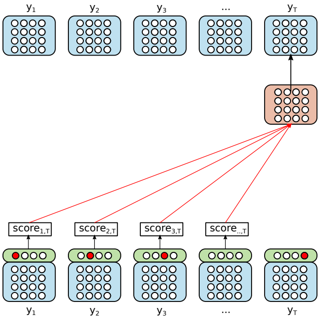
- In its architecture, it's just the *decoder* portion of Vaswani *et al.*'s transformer. Recall: this model learns to predict the next word using masked self-attention.



GPT's architecture

The GPT model wasn't given any particular task: it was just trained to 'predict the next word'.

- In its architecture, it's just the *decoder* portion of Vaswani *et al.*'s transformer. Recall: this model learns to predict the next word using masked self-attention.



GPT's 'prompt'

GPT's 'prompt'

GPT's prompt is basically a (long) input to a decoder network, that predicts the next word.

GPT's 'prompt'

GPT's prompt is basically a (long) input to a decoder network, that predicts the next word.

- So it can produce a natural continuation of the prompt.

GPT's 'prompt'

GPT's prompt is basically a (long) input to a decoder network, that predicts the next word.

- So it can produce a natural continuation of the prompt.

When you give it a prompt, you're placing the network at a point within the huge space of 'possible texts' that it learned during training.

GPT's 'prompt'

GPT's prompt is basically a (long) input to a decoder network, that predicts the next word.

- So it can produce a natural continuation of the prompt.

When you give it a prompt, you're placing the network at a point within the huge space of 'possible texts' that it learned during training.

- The longer the prompt is, the more precisely you position GPT within this space.

GPT's 'prompt'

GPT's prompt is basically a (long) input to a decoder network, that predicts the next word.

- So it can produce a natural continuation of the prompt.

When you give it a prompt, you're placing the network at a point within the huge space of 'possible texts' that it learned during training.

- The longer the prompt is, the more precisely you position GPT within this space. N.B. No learning happens through the prompt, though it may seem otherwise!

GPT's 'prompt'

GPT's prompt is basically a (long) input to a decoder network, that predicts the next word.

- So it can produce a natural continuation of the prompt.

When you give it a prompt, you're placing the network at a point within the huge space of 'possible texts' that it learned during training.

- The longer the prompt is, the more precisely you position GPT within this space. N.B. No learning happens through the prompt, though it may seem otherwise!

Next seminar, Simon will be doing a practical course on 'prompt engineering'!