Data Science Day 2023

When? **October 19 starting at 17:00**
Where? **The science library**

Neural networks

IN4080

Natural Language Processing

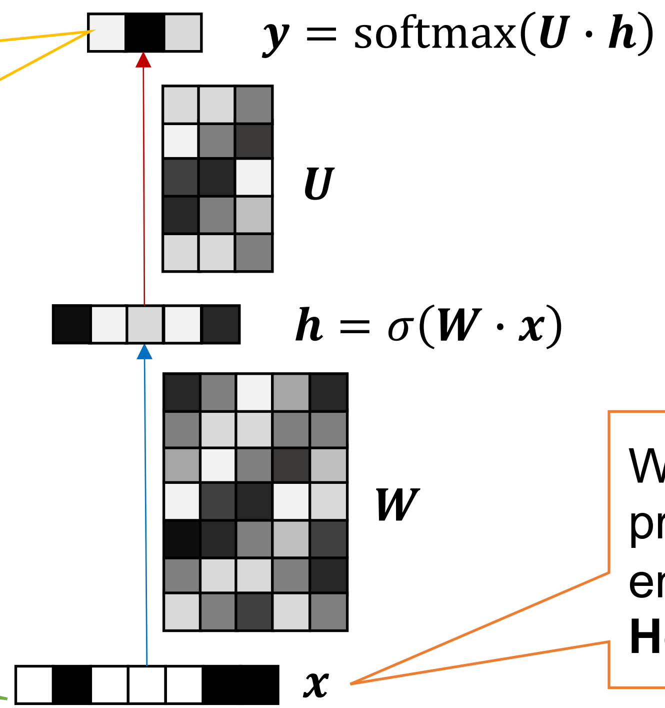
Yves Scherrer

Neural networks

- 1-layer feed-forward network = logistic regression model
- LR-like models can be stacked to create deeper networks
- We know how to get word vectors:
  - Counting + dimensionality reduction
  - Skip-gram with negative sampling

Neural document classification

The size of the prediction vector is defined by the number of classes.



The size of the input vector is defined by the number of distinct words in the training corpus (bag of words).

We can use pretrained word embeddings here. **How?**

Neural document classification

- Document classification:
  - One vector per document
- Word embeddings:
  - One vector per word

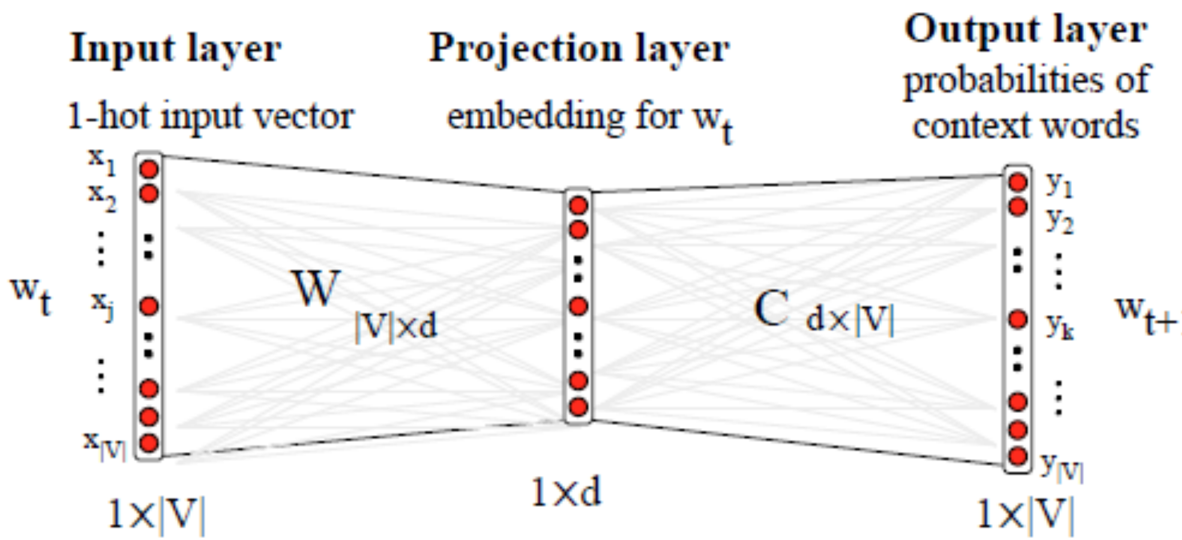
Problem: texts come in different lengths…

Pooling:

- Combine all word vectors into a document vector
  - Concatenate all word vectors
    - Set vector values of “unused” words to zero.
    - This yields a very large vector
  - Take the mean of all the word vectors
  - Take the element-wise maximum of all vectors

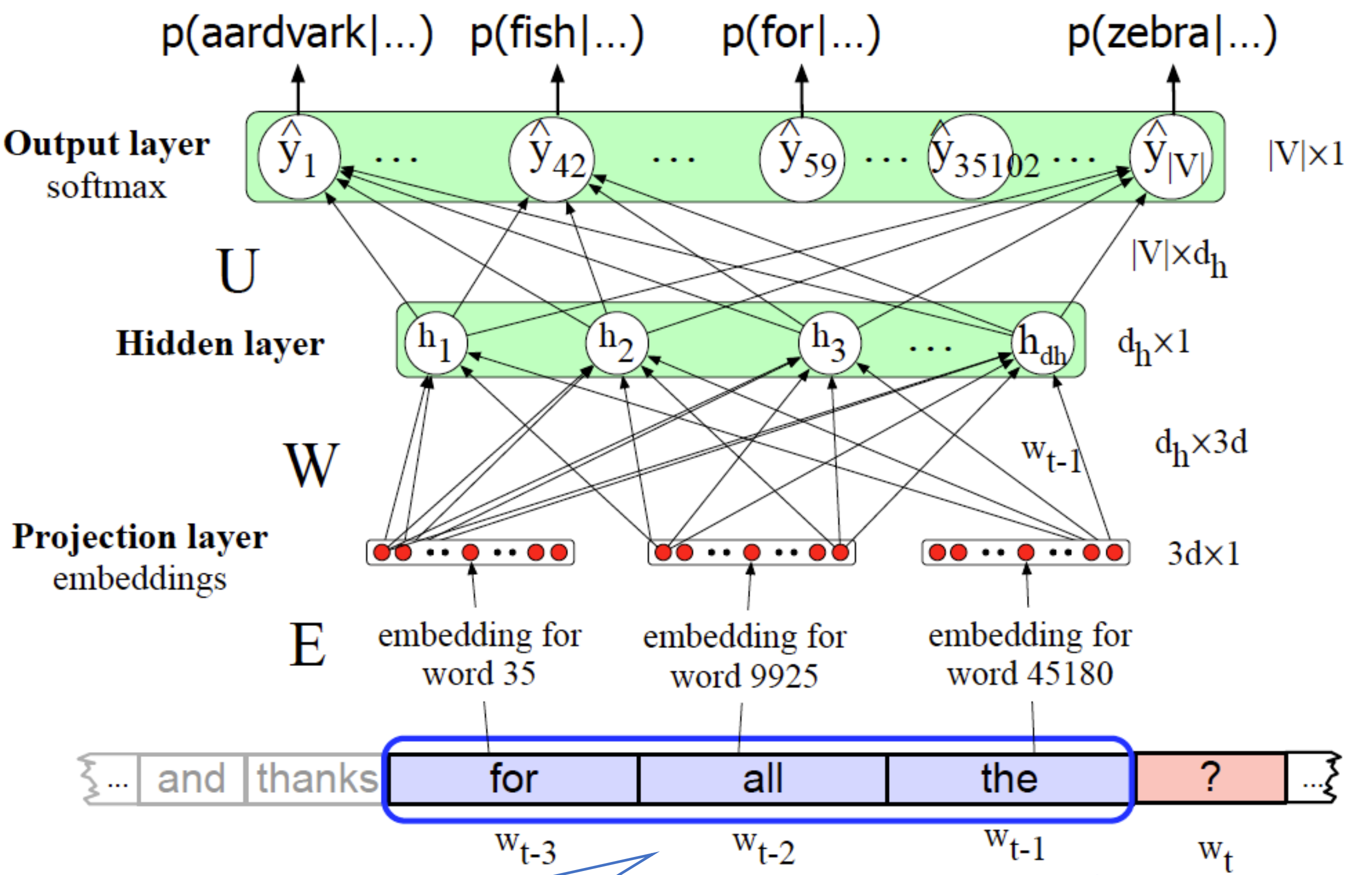
Neural language models

Recall our first attempt:



- This was for producing word embeddings. Now we assume that we have them.
- We can use more than one input word.

Neural language models



Sliding window (of length 3)

Neural language models

- This model has serious drawbacks:
  - Not efficient, need to run the same embeddings several times through the network.
  - Context is limited to window size.
- But neural LMs still work better than probabilistic ones.
  - Why?

Neural language models

Example:

- Training data:
  - Seen: I have to make sure that the cat gets fed.
  - Not seen: dog gets fed
- Test data:
  - I forgot to make sure that the dog gets ___
- A probabilistic (trigram) LM can't predict fed.
- A neural LM can use the similarity of cat and dog embeddings to generalize and predict fed after dog.

The Transformer: Dealing with word sequences

Types of Transformer models

- Sequence encoders with self-attention
  - BERT
  - Contextualized word embeddings, document classification, sequence labeling
- Sequence decoders with self-attention
  - GPT
  - Language modeling, text generation
- Encoder-decoder model with cross-attention
  - The original Transformer
  - Machine translation

Types of Transformer models

- Sequence encoders with self-attention
  - BERT
  - Contextualized word embeddings, document classification, sequence labeling
- Sequence decoders with self-attention
  - GPT
  - Language modeling, text generation
- Encoder-decoder model with cross-attention
  - The original Transformer
  - Machine translation

Encoder models

For SGNS, we used one target word and one context word to predict their similarity.

- The model doesn’t know about the sentence in which the target word is used.
  - Homographs cannot be distinguished.
- The model doesn’t distinguish between right and left contexts, and between close and far contexts.
- Similarity prediction is practical because it enables a self-supervised setup (no data annotation needed).

Masked language modeling

Let us use another setup:

- Take a sentence, replace 12% of tokens by a blank
- In addition, replace 1.5% of tokens by another randomly chosen token
- Train a model to “fill the blanks”
- 1 sentence = 1 training instance
- Self-supervised

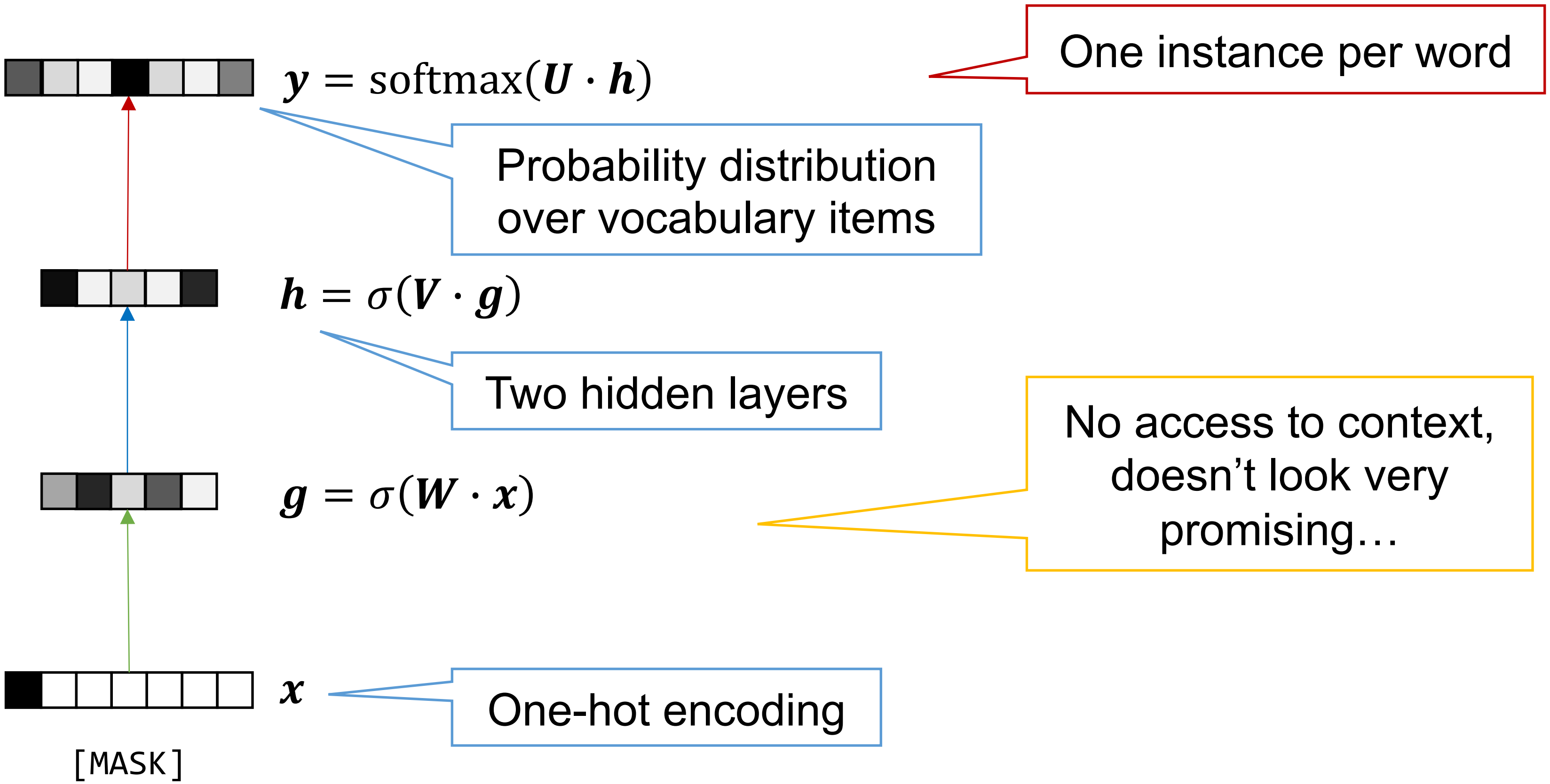
Example:

```
So long   and thanks for all the     fish

So [MASK] and [MASK] for all apricot fish
```

What kind of neural network?

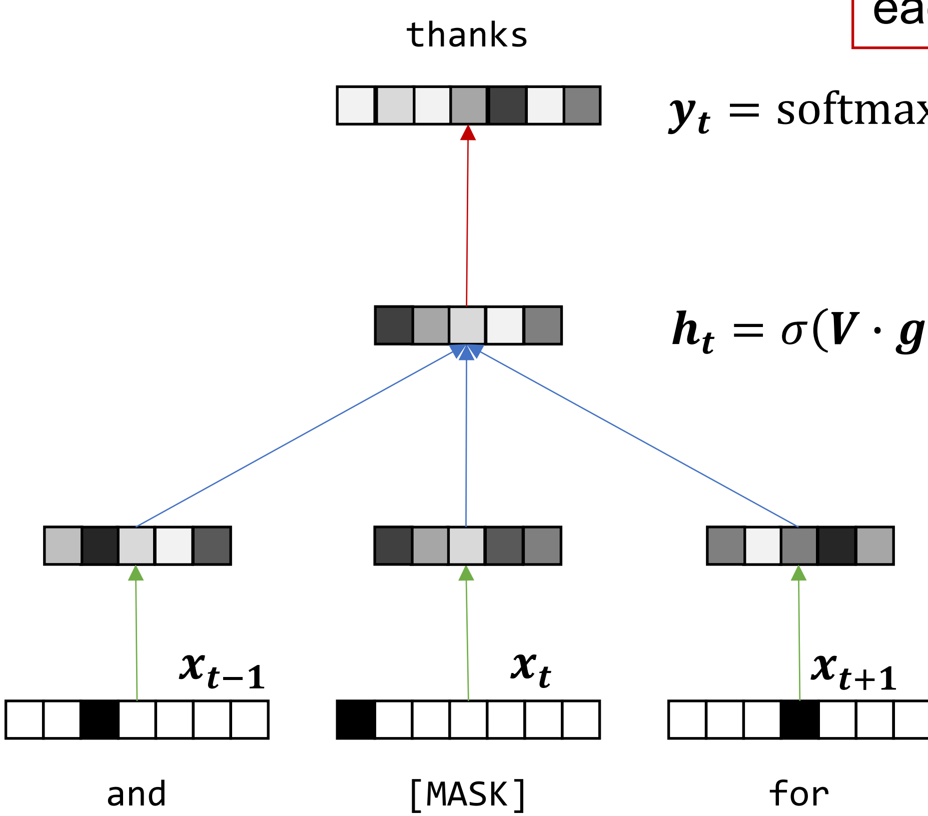
A simple feed-forward network:



What kind of neural network?

Include context words:

One instance per center word, each word is present in 3 instances



$$y_t = \mathrm{softmax}(\boldsymbol{U} \cdot \boldsymbol{h}_t)$$

$$\boldsymbol{h}_t = \sigma(\boldsymbol{V} \cdot \boldsymbol{g}_{t-1} + \boldsymbol{V} \cdot \boldsymbol{g}_t + \boldsymbol{V} \cdot \boldsymbol{g}_{t+1})$$

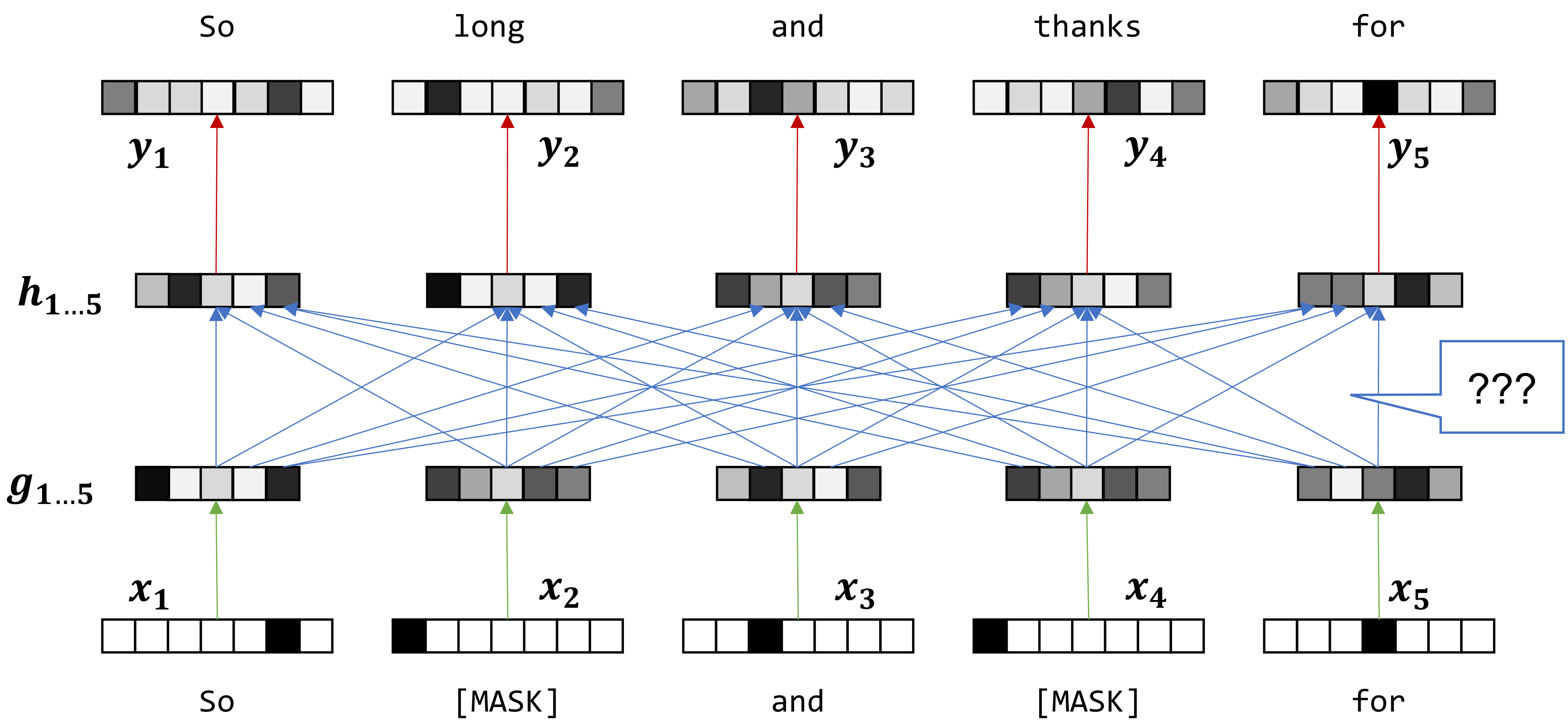
All 3 words contribute equally to the sum

$$\boldsymbol{g}_i = \sigma(\boldsymbol{W} \cdot \boldsymbol{x}_i)$$

Limited access to context: window size 1

What kind of neural network?

One training instance per sentence,
As many output words as input words



Self-attention

How to compute the $\boldsymbol{h}$ layer?

- Simple average:
  $$\boldsymbol{h_t} = \sigma\left(\frac{1}{n} \cdot \boldsymbol{V} \cdot \boldsymbol{g_1} + \cdots + \frac{1}{n} \cdot \boldsymbol{V} \cdot \boldsymbol{g_t} + \cdots + \frac{1}{n} \cdot \boldsymbol{V} \cdot \boldsymbol{g_n}\right)$$
- Some words are more important than others. Let's use a weighted average:
  $$\boldsymbol{h_t} = \sigma\left(\alpha_{t,1} \cdot \boldsymbol{V} \cdot \boldsymbol{g_1} + \cdots + \alpha_{t,t} \cdot \boldsymbol{V} \cdot \boldsymbol{g_t} + \cdots + \alpha_{t,n} \cdot \boldsymbol{V} \cdot \boldsymbol{g_n}\right)$$
- How do we know the α values?
  - Maybe depending on the distance to t? Maybe not?
  - Anyway, a neural network should be able to learn these values automatically…
- Idea: depending on similarity: $\alpha_{t,c} = \boldsymbol{g_t} \cdot \boldsymbol{g_c}$
  - Recall: the dot product is a simple measure of similarity

Self-attention

How to compute the $\boldsymbol{h}$ layer?

- The more similar the vectors $\boldsymbol{g_c}$ and $\boldsymbol{g_t}$ are, the more important $\boldsymbol{g_c}$ is for computing $\boldsymbol{h_t}$:
$$\alpha_{t,c} = \boldsymbol{g_t} \cdot \boldsymbol{g_c}$$
- For a weighted average, all αs should sum up to 1. Let's use softmax:
$$\alpha_{t,c} = \mathrm{softmax}(\boldsymbol{g_t} \cdot \boldsymbol{g_c})$$
- Putting everything together:
$$\boldsymbol{h_t} = \sigma\left(\sum_{i=1}^{n} \alpha_{t,i} \cdot \boldsymbol{V} \cdot \boldsymbol{g_i}\right)$$
- That's a simple type of **self-attention**.

The "real" thing is still more complicated…

Self-attention, continued

- The $\boldsymbol{g}$ vectors now occur in 3 places and roles:
  - As target word for computing α: $\mathrm{softmax}(\boldsymbol{g_t} \cdot \boldsymbol{g_c})$
    - We call this role **query**
  - As context word for computing α: $\mathrm{softmax}(\boldsymbol{g_t} \cdot \boldsymbol{g_c})$
    - We call this role **key**
  - As a factor of the final product: $\alpha_{t,i} \cdot \boldsymbol{V} \cdot \boldsymbol{g_i}$
    - We call this role **value**
- The 3 roles are different, and the $\boldsymbol{g}$ vectors are not equally well suited for all of them.
  - Let's create 3 different vectors tailored to the different roles!

Self-attention, continued

- Let's create 3 different vectors:
  - Query vector: $\boldsymbol{q_i} = \boldsymbol{W^Q} \cdot \boldsymbol{g_i}$
  - Key vector: $\boldsymbol{k_i} = \boldsymbol{W^K} \cdot \boldsymbol{g_i}$
  - Value vector: $\boldsymbol{v_i} = \boldsymbol{V} \cdot \boldsymbol{g_i}$ (we already have this one)
- What about our weight values α?

$$\alpha_{t,c} = \mathrm{softmax}(\boldsymbol{q_t} \cdot \boldsymbol{k_c})$$

- It turns out that the dot product needs to be scaled before passing it to the softmax:

$$\alpha_{t,c} = \mathrm{softmax}\left(\frac{\boldsymbol{q_t} \cdot \boldsymbol{k_c}}{\sqrt{d_k}}\right)$$

d_k refers to the dimensionality of $\boldsymbol{k}$ (and also of $\boldsymbol{q}$)

Self-attention, continued

Putting things together again:

$$\boldsymbol{h_t} = \sigma\left(\sum_{i=1}^{n} \underbrace{\mathrm{softmax}\left(\frac{\boldsymbol{q_t} \cdot \boldsymbol{k_i}}{\sqrt{d_k}}\right)}_{\alpha_{t,i}} \cdot \underbrace{\boldsymbol{v_i}}_{\boldsymbol{V} \cdot \boldsymbol{g_i}}\right)$$

α is a square matrix that shows how important a context word is at a given position:

q1•k1	q1•k2	q1•k3	q1•k4	q1•k5
q2•k1	q2•k2	q2•k3	q2•k4	q2•k5
q3•k1	q3•k2	q3•k3	q3•k4	q3•k5
q4•k1	q4•k2	q4•k3	q4•k4	q4•k5
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

Multi-head attention

- We may want to have several α matrices to represent different types of attention:
  - One for syntactic relatedness
  - One for semantic relatedness
  - One for coreference, etc.

- Let's start over, with an extra index h (head):
  - Query vector: $\boldsymbol{q_i^h} = \boldsymbol{W^{Q,h}} \cdot \boldsymbol{g_i}$
  - Key vector: $\boldsymbol{k_i^h} = \boldsymbol{W^{K,h}} \cdot \boldsymbol{g_i}$
  - Value vector: $\boldsymbol{v_i^h} = \boldsymbol{V^h} \cdot \boldsymbol{g_i}$

Multi-head attention

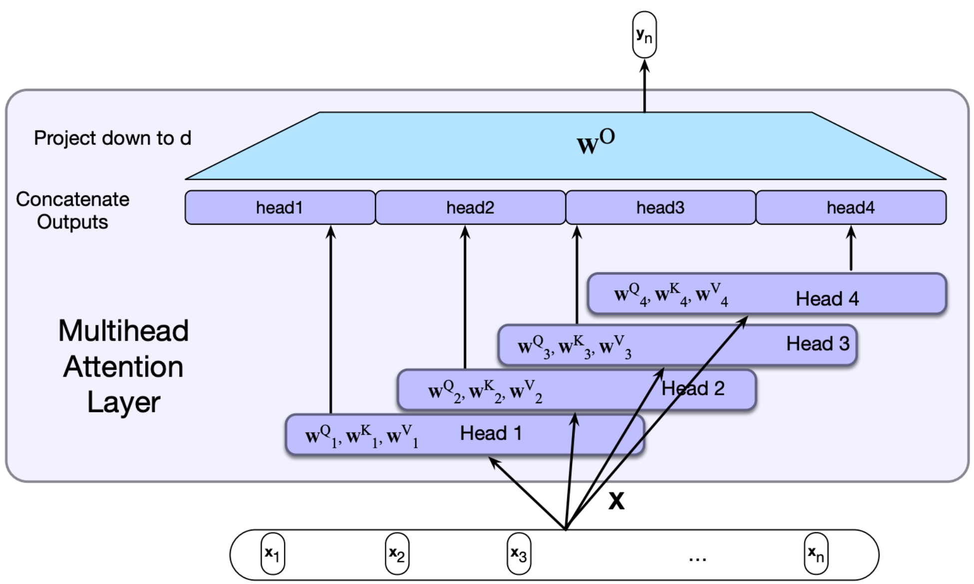
We have removed the sigmoid here.

- Compute the vector for one head:

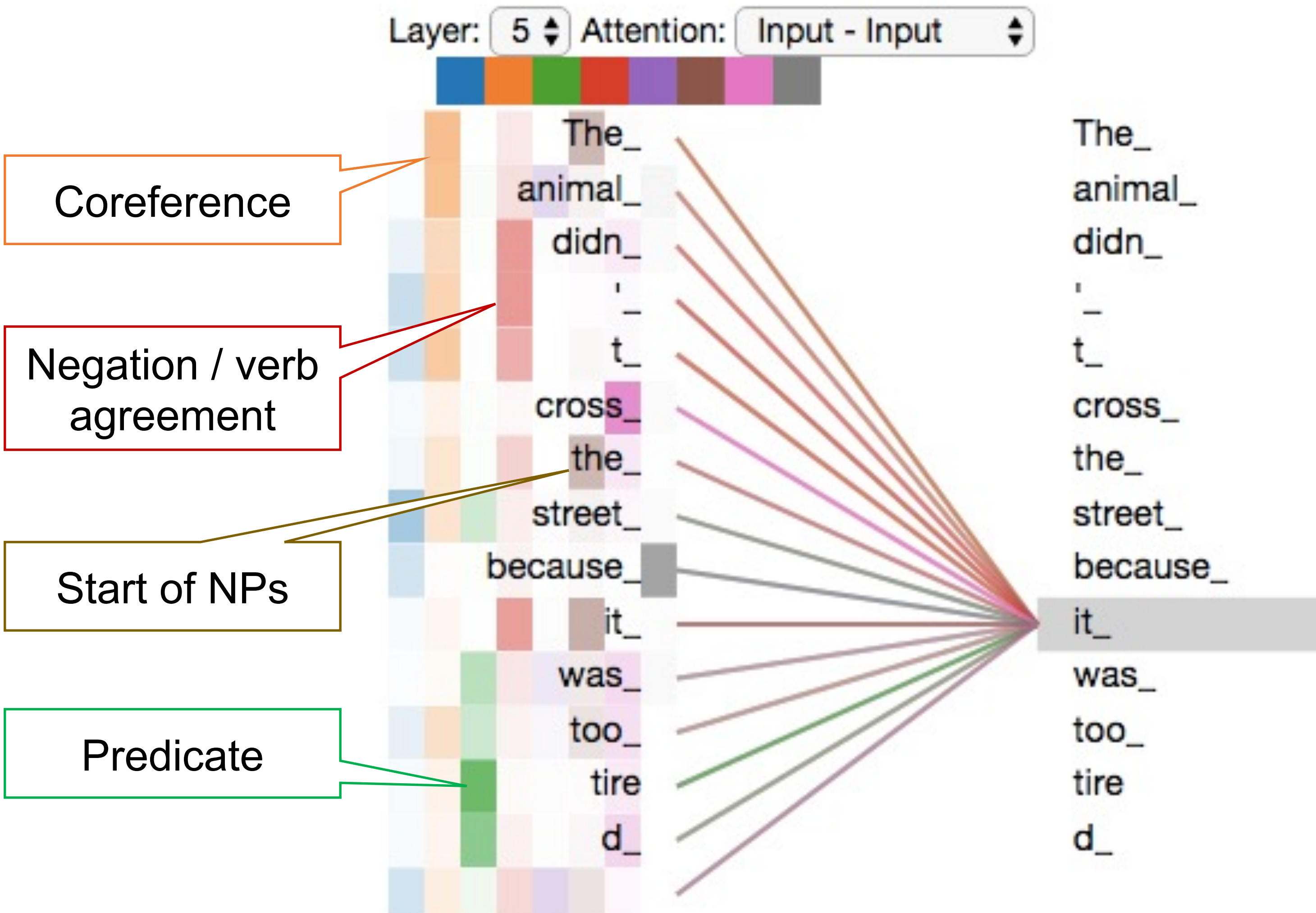
$$\mathbf{head}_t^h = \sum_{i=1}^{n} \operatorname{softmax}\left(\frac{\boldsymbol{q}_t^h \cdot \boldsymbol{k}_i^h}{\sqrt{d_k}}\right) \cdot \boldsymbol{v}_i^h$$

- Then concatenate all head vectors and project them:

$$\boldsymbol{h}_t = \left(\mathbf{head}_t^1 \oplus \cdots \oplus \mathbf{head}_t^m\right) \cdot \boldsymbol{W}^O$$



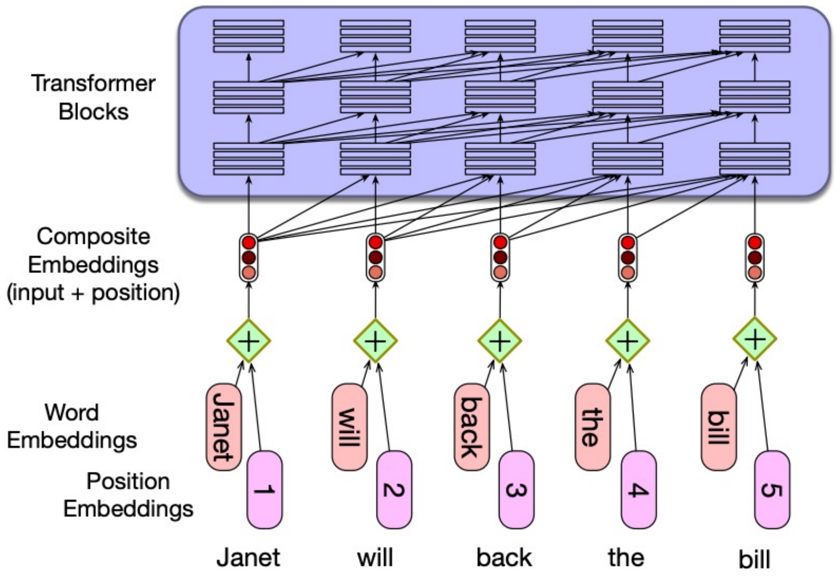
Multi-head attention



http://jalammar.github.io/illustrated-transformer/

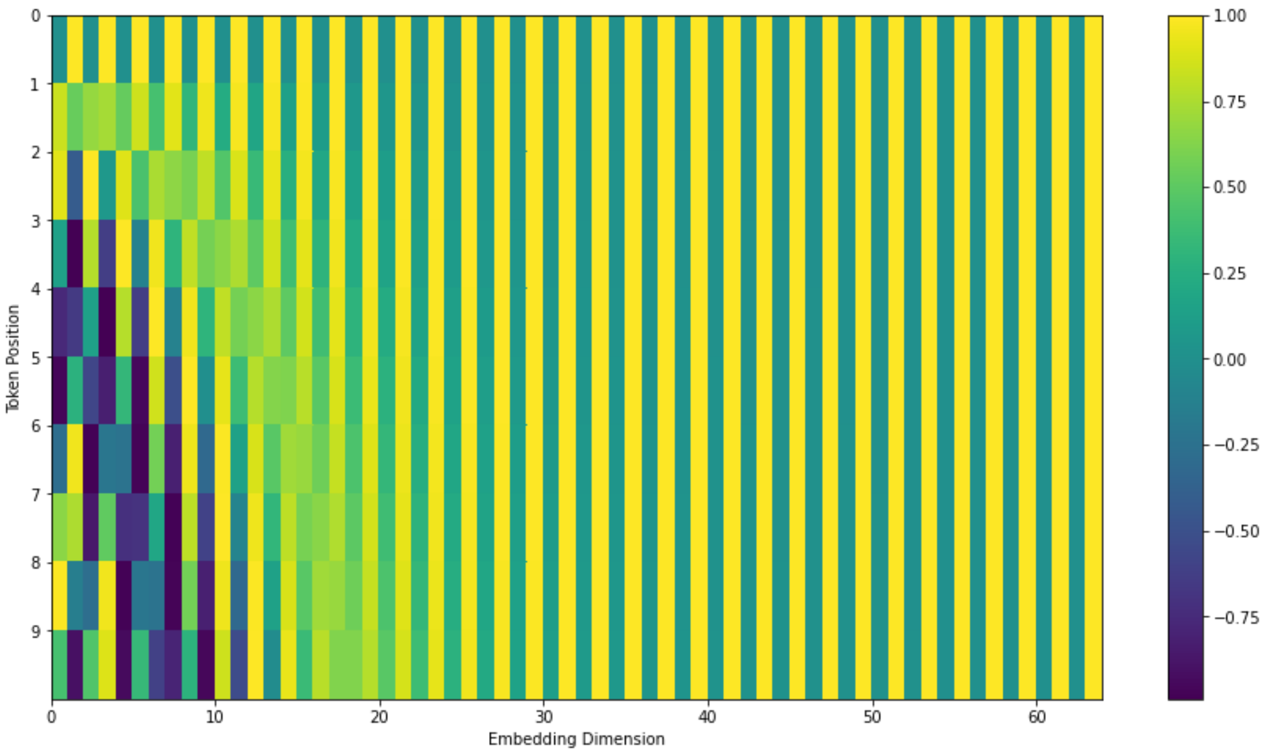
Position embeddings

- Self-attention does not have any notion of word ordering
  - As of now, the attention weights are only chosen based on semantic similarity of the words
  - But we probably should take simple proximity into account…
- Simple solution:
  - Concatenate semantic word embedding with absolute position embedding



Position embeddings

- Absolute numbers are not very efficient.
- The “real thing” uses several overlaid sine functions:

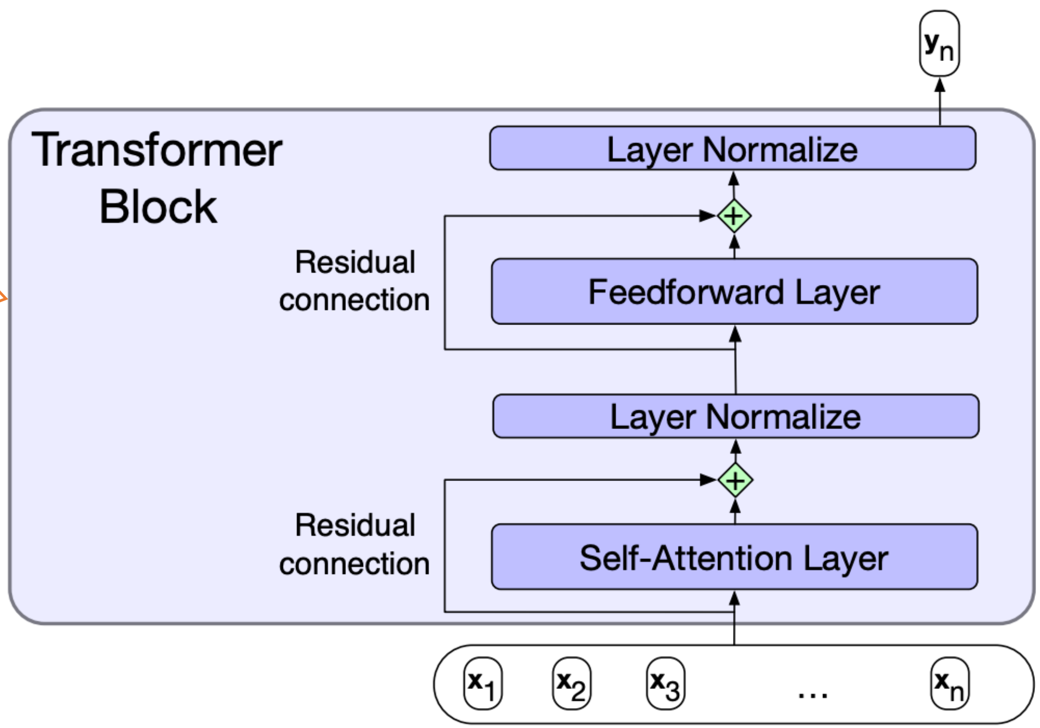


Transformer blocks

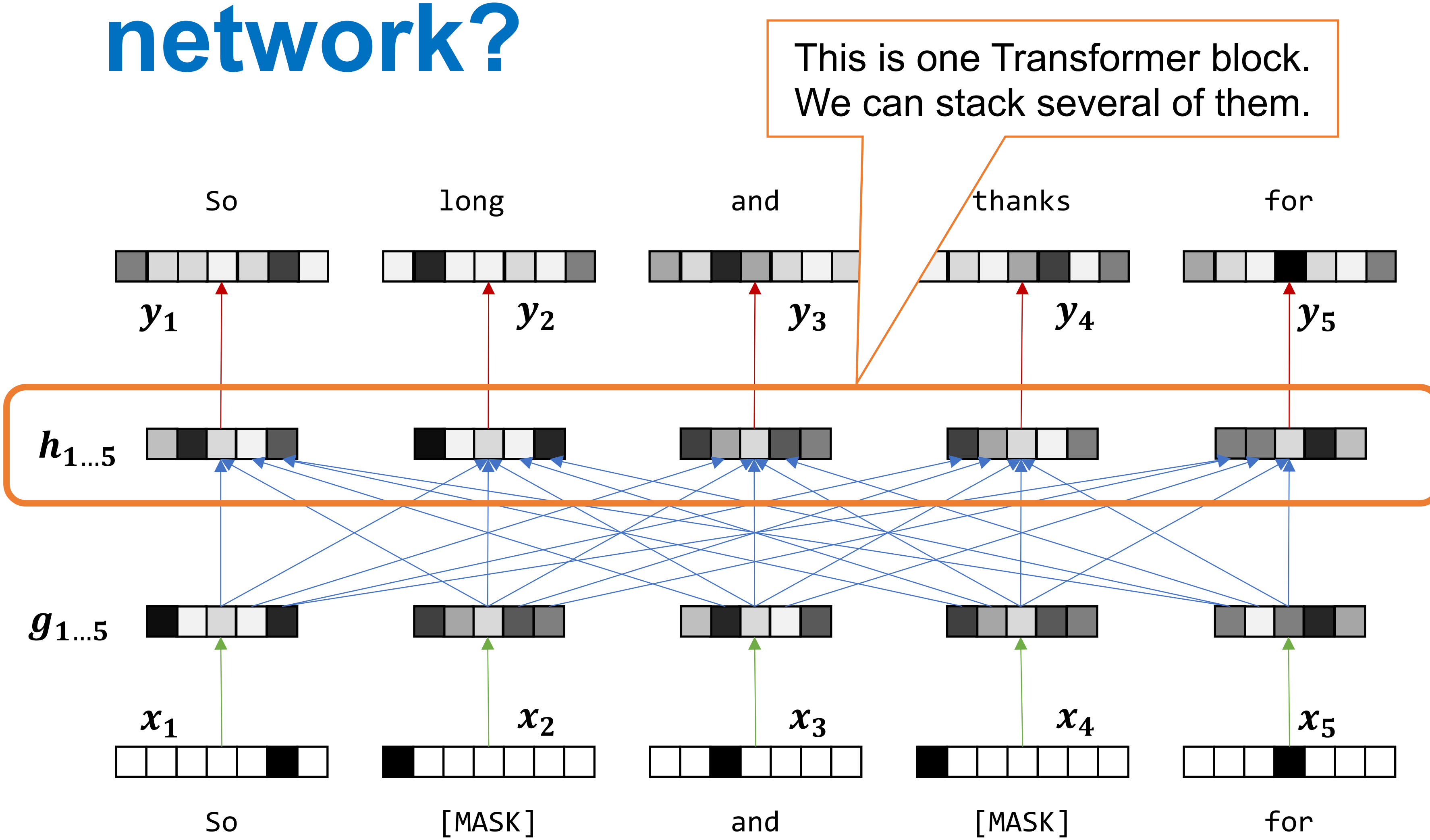
Multi-head self-attention isn't quite sufficient.

We need a few extra things and package everything up into so-called Transformer blocks:

And of course, we have to make sure that the whole thing remains differentiable…



What kind of neural network?

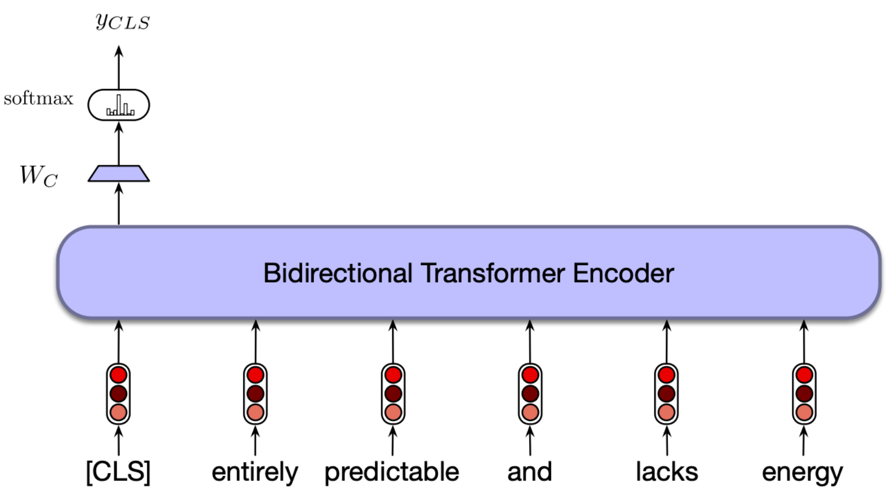


What can we do with sequence encoders?

- Get contextualized word embeddings
  - Train a model on the MLM task
  - Pass a sentence through the network and extract the h vector of each word as its embedding
  - Alternative: average the vectors of several layers

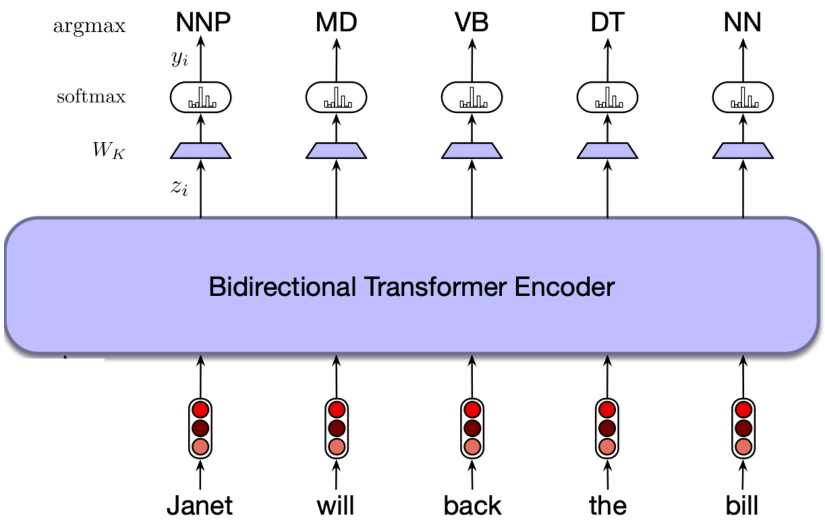
What can we do with sequence encoders?

- Text classification (one label per sentence
  - Train a model on the MLM task, adding a `[CLS]` token in front of every sentence
  - Throw away the output layer, create a new one
  - **Fine-tune** the model to predict the label at the `[CLS]` position



What can we do with sequence encoders?

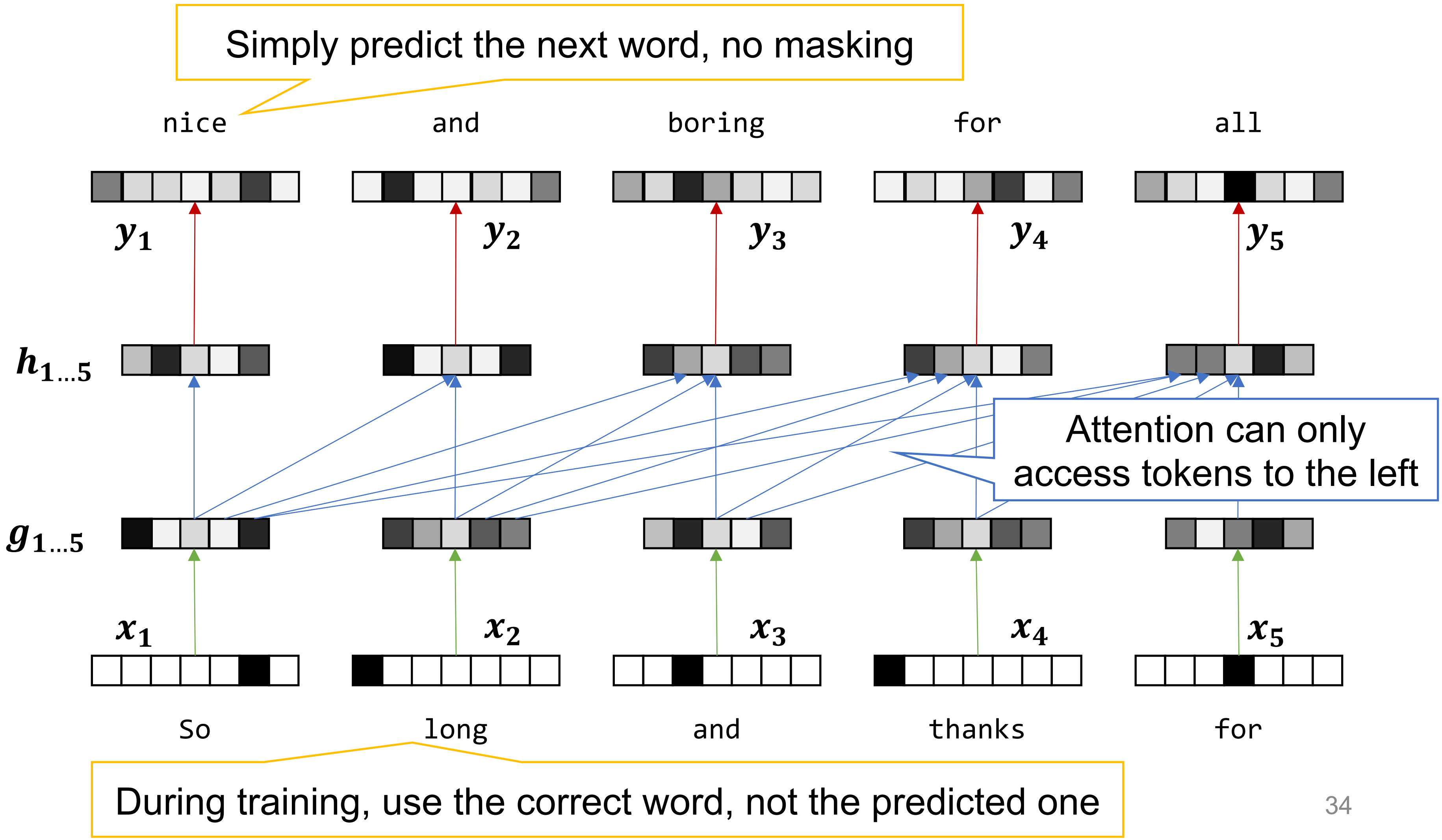
- Sequence labeling (e.g. POS tagging)
  - Train a model on the MLM task
  - Throw away the output layer, create a new one
  - **Fine-tune** the model on POS-annotated data



Types of Transformer models

- Sequence encoders with self-attention
  - BERT
  - Contextualized word embeddings, document classification, sequence labeling
- Sequence decoders with self-attention
  - GPT
  - Language modeling, text generation
- Encoder-decoder model with cross-attention
  - The original Transformer
  - Machine translation

Language modeling with self-attention



Language modeling with self-attention

Probabilistic language model:

$$P(w_1, \ldots, w_n)$$
$$= P(w_1) \cdot P(w_2 |\, w_1) \cdot P(w_3 | w_1, w_2) \cdot \cdots$$
$$\cdot\, P(w_n | w_1, \ldots, w_{n-1})$$

- For n-gram models, we could not condition on all preceding tokens
- Thanks to attention, we can do that now
- But typically, far-away tokens get less attention

Language modeling with self-attention

For simplicity, let's assume a single head:

$$\alpha_{t,c} = \text{softmax}\left(\begin{cases} -\infty, & c > t \\ \dfrac{\boldsymbol{q_t} \cdot \boldsymbol{k_c}}{\sqrt{d_k}}, & c \le t \end{cases}\right)$$

q1•k1	−∞	−∞	−∞	−∞
q2•k1	q2•k2	−∞	−∞	−∞
q3•k1	q3•k2	q3•k3	−∞	−∞
q4•k1	q4•k2	q4•k3	q4•k4	−∞
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

$$\boldsymbol{h_t} = \sigma\left(\sum_{i=1}^{n} \alpha_{t,i} \cdot \boldsymbol{v_t}\right)$$

Language modeling with self-attention

- In terms of model architecture, that’s about all you need to know in order to build your own GPT model ☺
- Of course, you’ll still need the right amount (and quality) of training data…

Types of Transformer models

- Sequence encoders with self-attention
  - BERT
  - Contextualized word embeddings, document classification, sequence labeling
- Sequence decoders with self-attention
  - GPT
  - Language modeling, text generation
- Encoder-decoder model with cross-attention
  - The original Transformer
  - Machine translation

Machine translation

Some (fairly obvious) facts about MT:

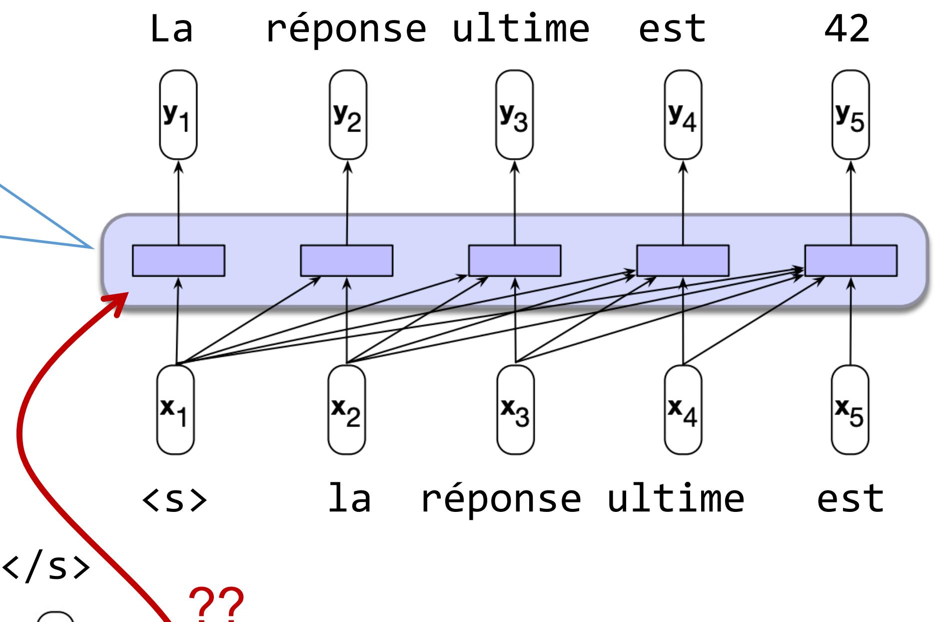
- Input: a sentence in the source language
- Output: a sentence in the target language
    - Hopefully grammatical
    - Hopefully with the same meaning
- Source and target sentence most often do not have the same number of words nor the same word order

Main idea:

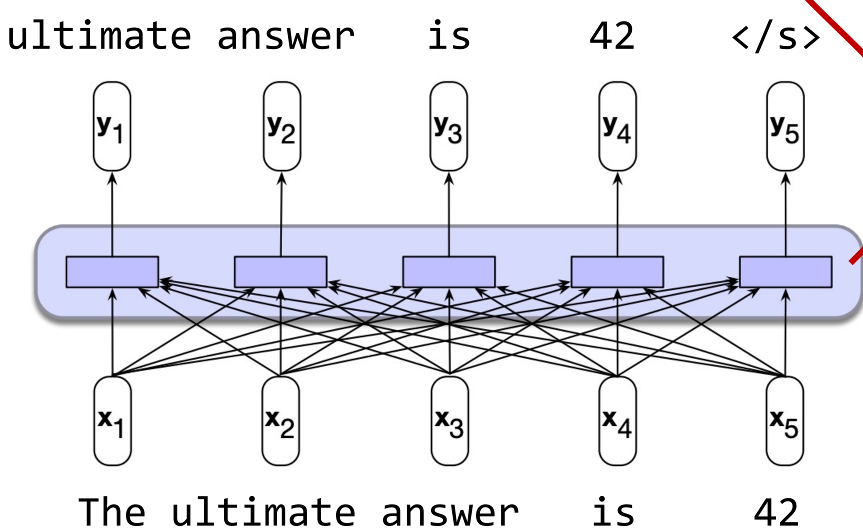
- Train an encoder for the source language
- Train a decoder for the target language
- Connect the two

Everything can be trained together thanks to backpropagation

Connecting encoders and decoders



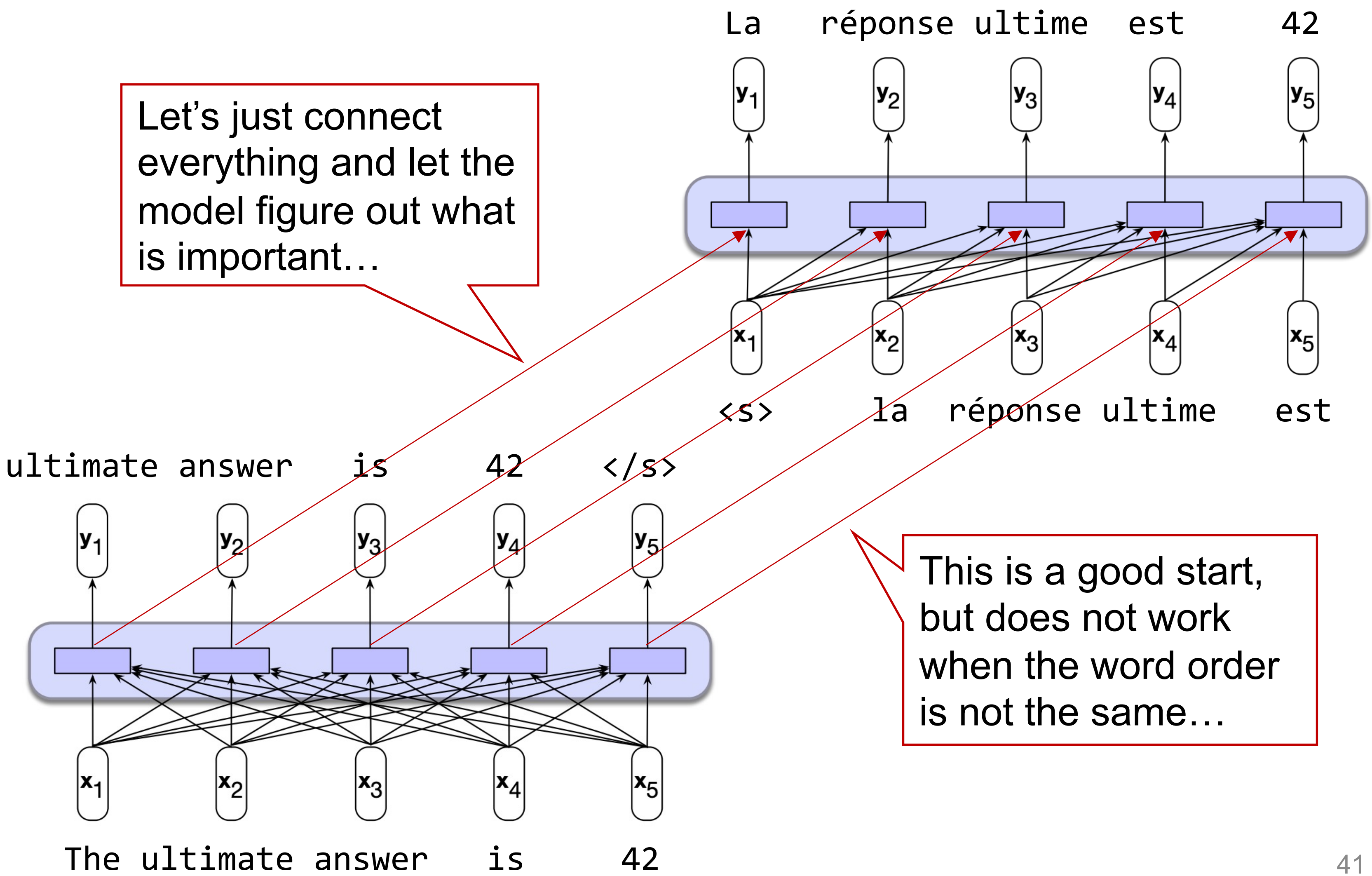
Decoder:
- Trained on next word prediction task
- Only has access to the left context



Encoder:
- Trained on next word prediction task
- Has access to left and right context

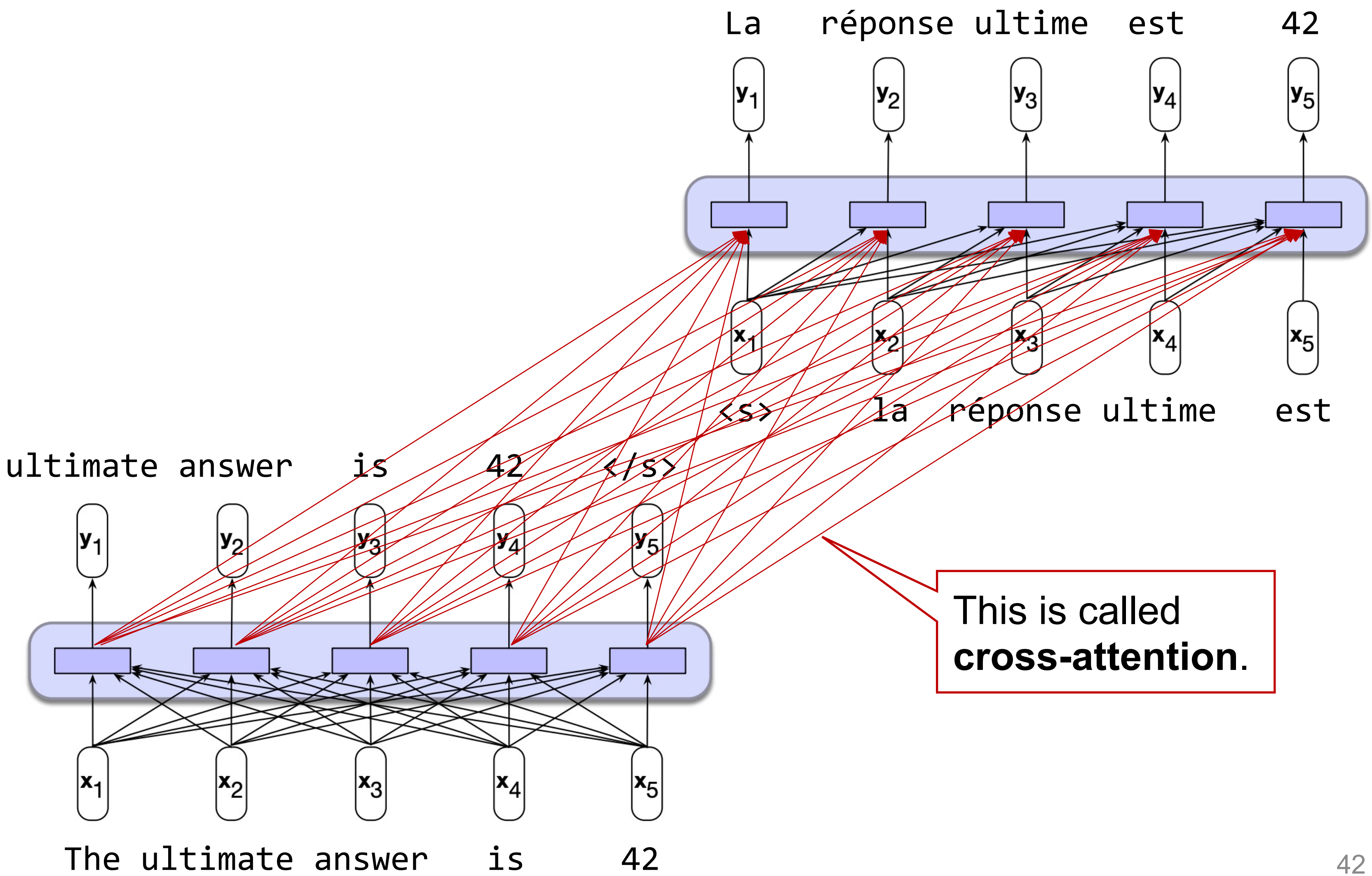
Connecting encoders and decoders

Let's just connect everything and let the model figure out what is important…



This is a good start, but does not work when the word order is not the same…

Connecting encoders and decoders



This is called **cross-attention**.

Cross-attention

- Dot-product attention:

$$\alpha_{i,j} = \text{softmax}\left(\boldsymbol{h}_{\boldsymbol{j}}^{\boldsymbol{enc}} \cdot \boldsymbol{h}_{\boldsymbol{i-1}}^{\boldsymbol{dec}}\right)$$

  - Determines the importance of the jth word of the encoder for the ith word of the encoder.
  - There are other attention types, and one can again use multiple heads.
- Context vector:

$$\boldsymbol{c}_{\boldsymbol{i}} = \sum_{j} \alpha_{i,j} \cdot \boldsymbol{h}_{\boldsymbol{j}}^{\boldsymbol{enc}}$$

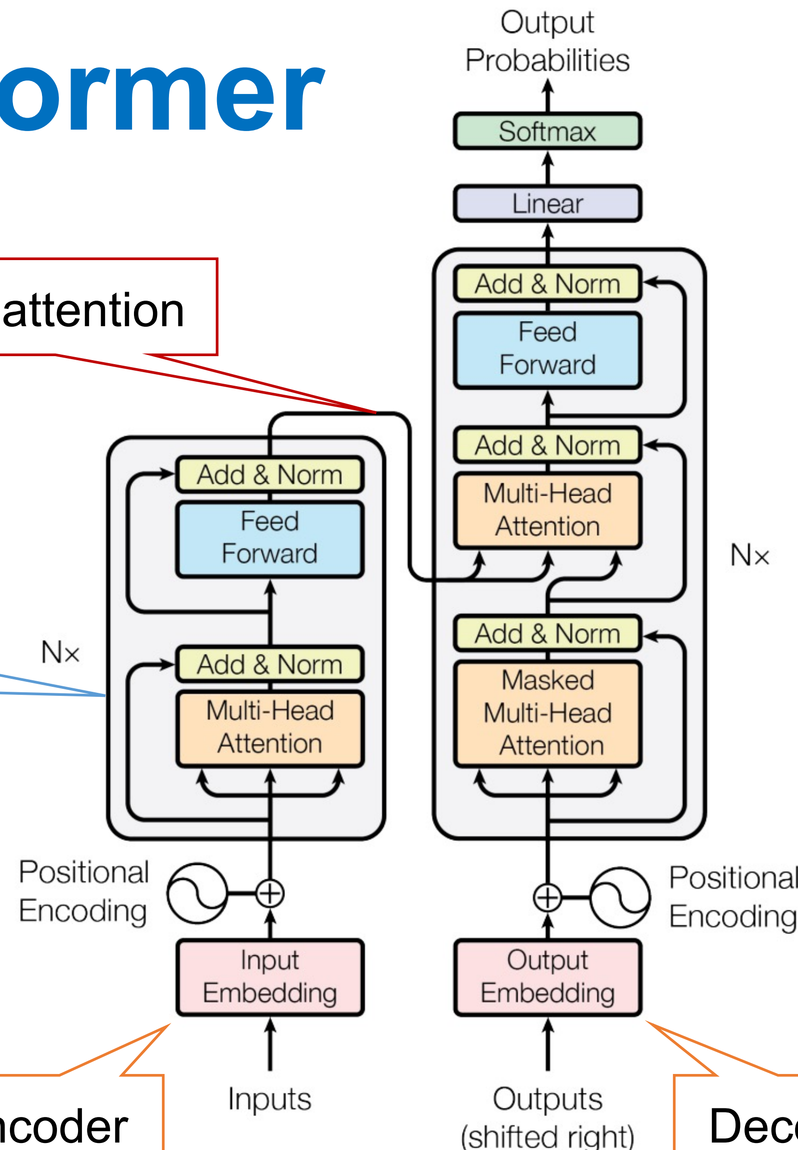
  - This vector is then combined with the vector produced by the decoder self-attention.

The full Transformer

Historically, this is what everything started with…

Cross-attention

A Transformer block with self-attention + the rest



Encoder

Decoder

https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Readings

- Sequence encoders with self-attention
  - Jurafsky & Martin, chapter 11
- Sequence decoders with self-attention
  - Jurafsky & Martin, chapter 10
- Encoder-decoder model with cross-attention
  - Jurafsky & Martin, chapter 9.8 + 13.3
  - Note: chapter 13 (“machine translation”) is numbered 10 in the current draft