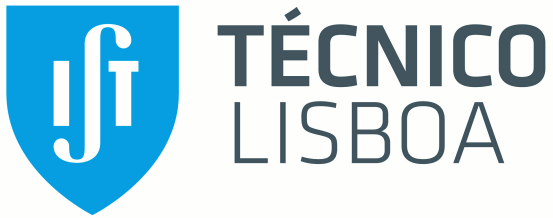
Attention, Transformers and Large Pretrained Models

Chryssa Zerva



Deep Learning, Spring 2025

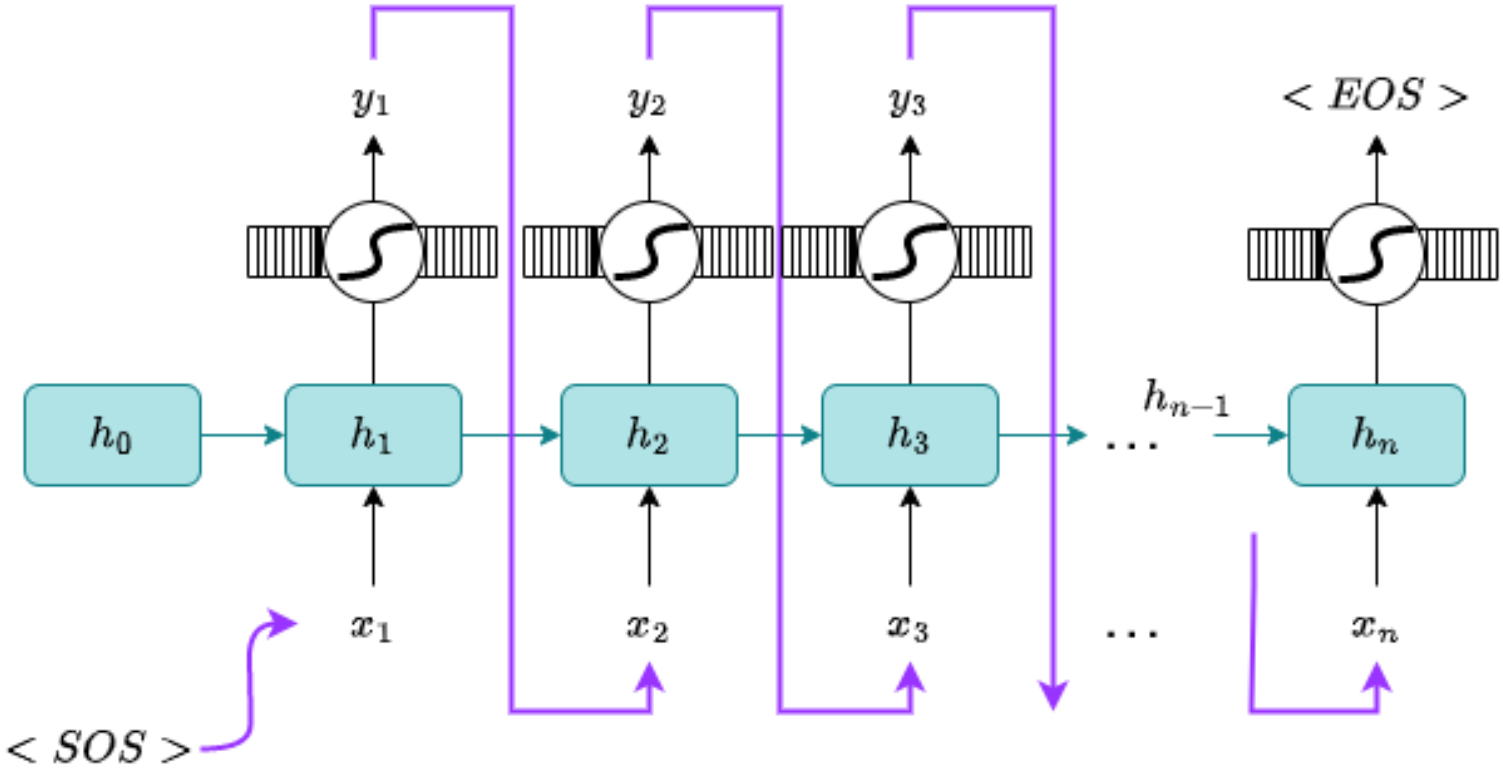
Roadmap

Today we look at self-attention, transformers and large pretrained models:

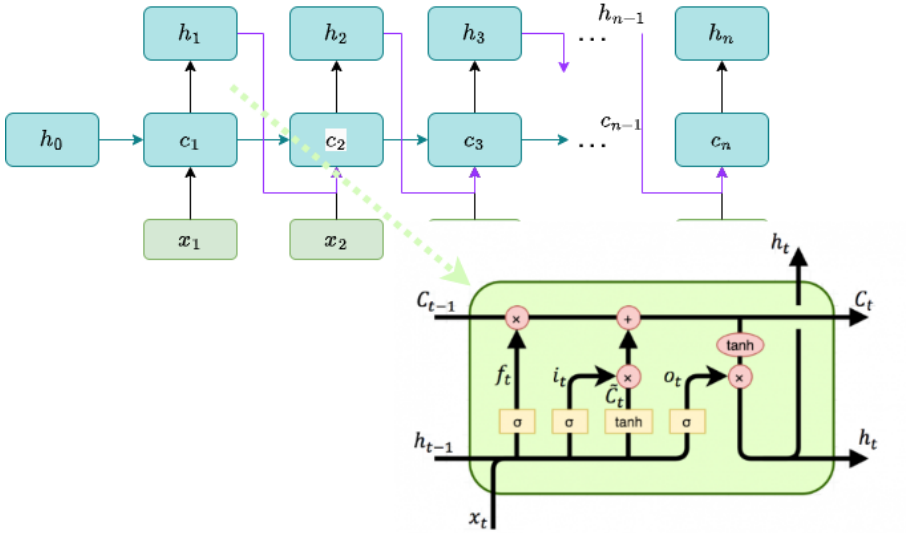
- Attention
- Self-attention
- Transformer networks
- Pre-trained models and transfer learning

Outline

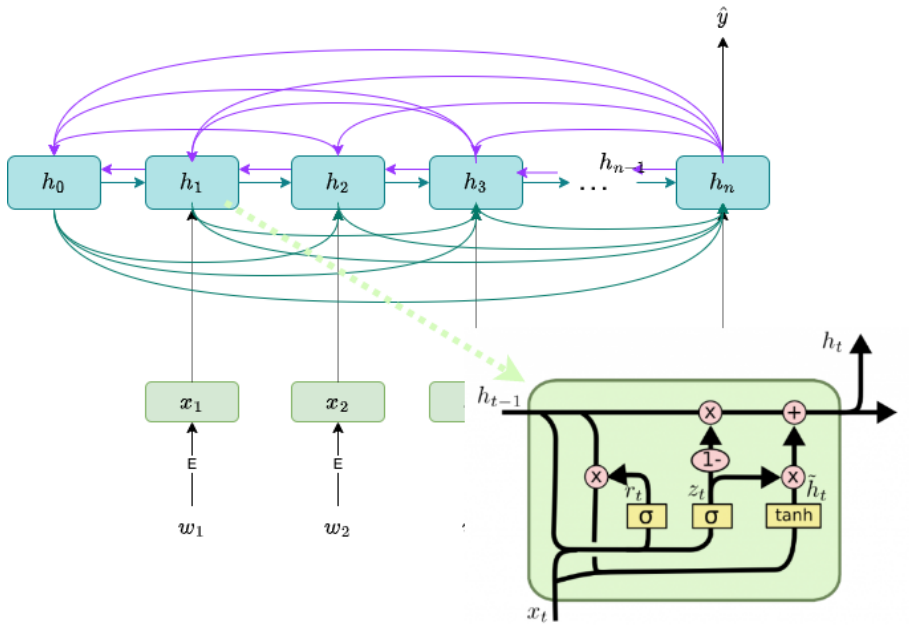
RNN Recap



RNN Recap

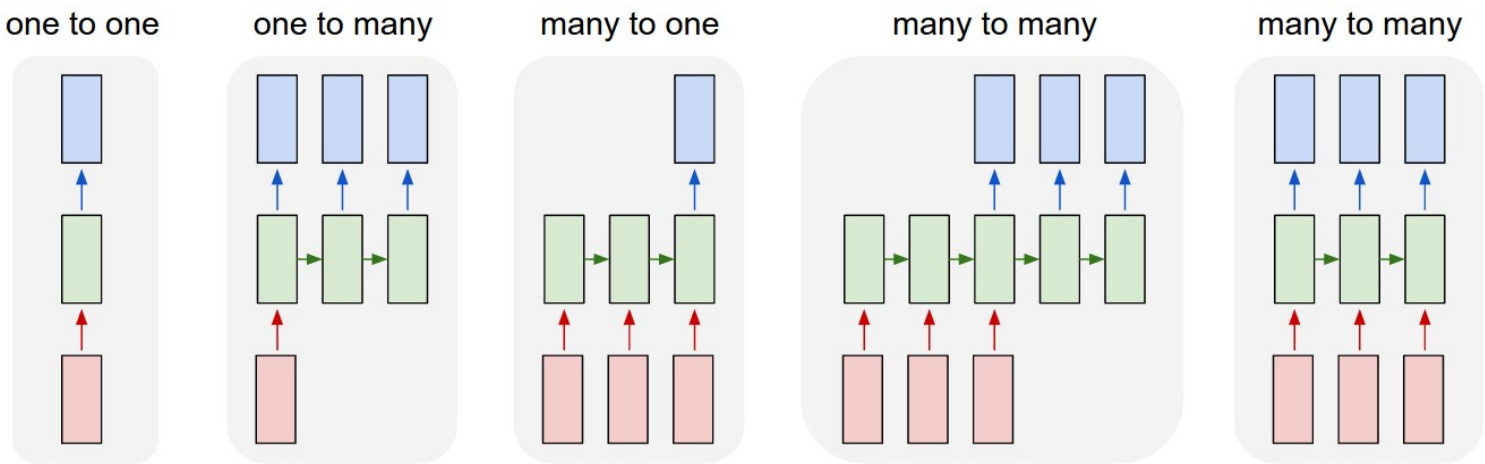


LSTM



GRU

RNN Recap

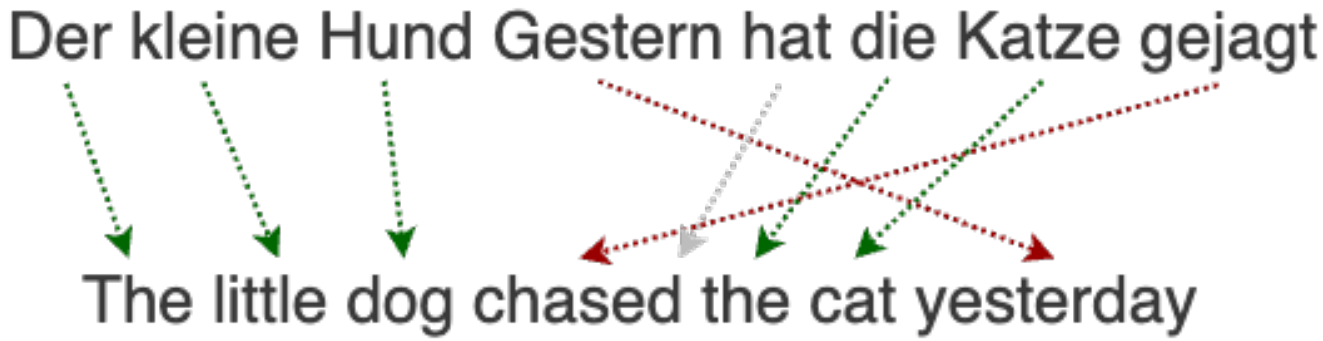


RNN limitations

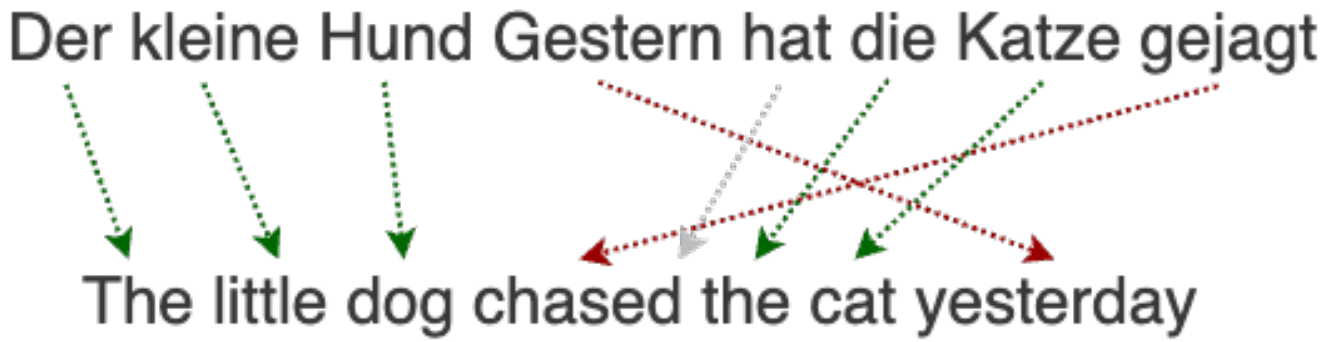
We can encode and decode sequential data but is this enough?

- What happens with arbitrarily long sequences?
- What happens with different sequence lengths?
- Are these architectures invariant to shifts in the input ?
- Is sequential context enough?

Translation Example

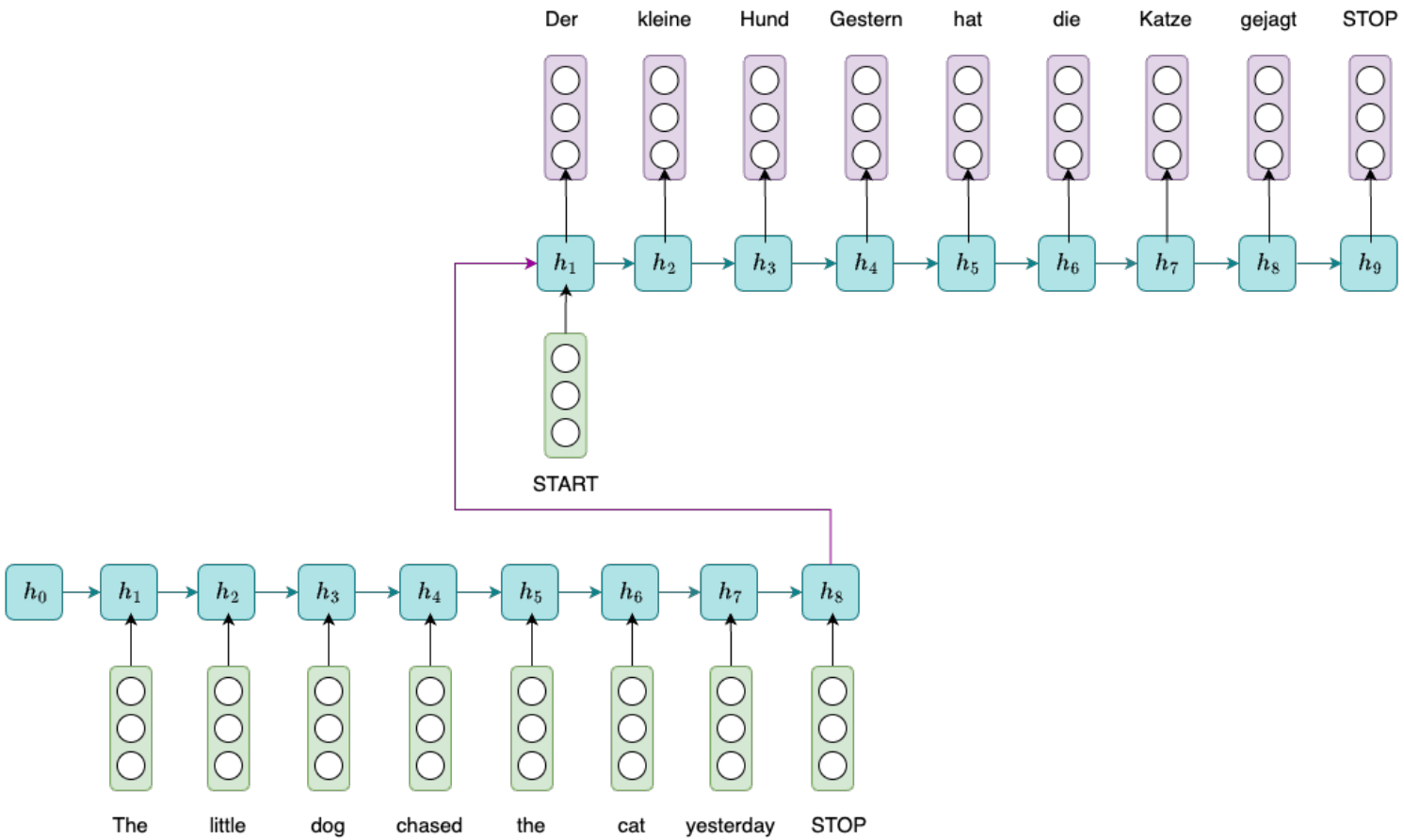


Translation Example



* Does an RNN-based encoder-decoder allow for the best representation?

Translation Example: Encoder-Decoder



Encode Sentences as Matrices, Not Vectors

Problem with the fixed-size vector model:

- Sequences are of different sizes but vectors are of the same size
- Bottleneck problem: a single vector needs to represent the full input sequence!

Encode Sentences as Matrices, Not Vectors

Problem with the fixed-size vector model:

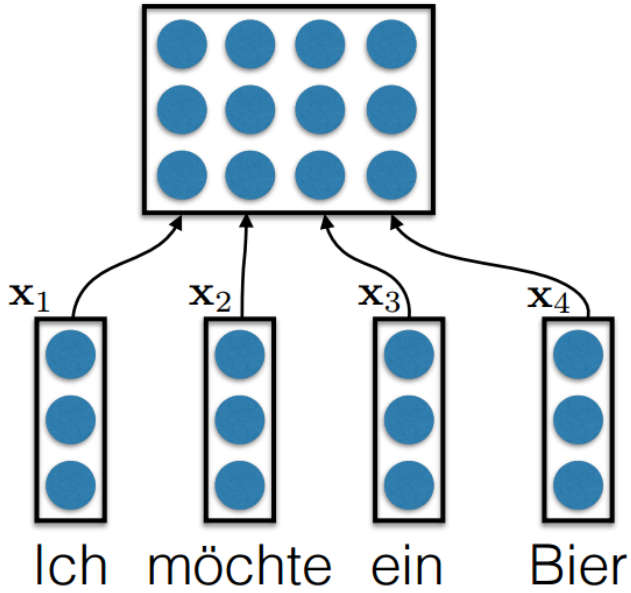
- Sequences are of different sizes but vectors are of the same size
- Bottleneck problem: a single vector needs to represent the full input sequence!

Solution: use matrices instead!

- Fixed number of rows, but number of columns depends on the number of words
- Then, before generating each word in the decoder, use an attention mechanism to condition on the relevant source words only

How to Encode a Sentence as a Matrix?

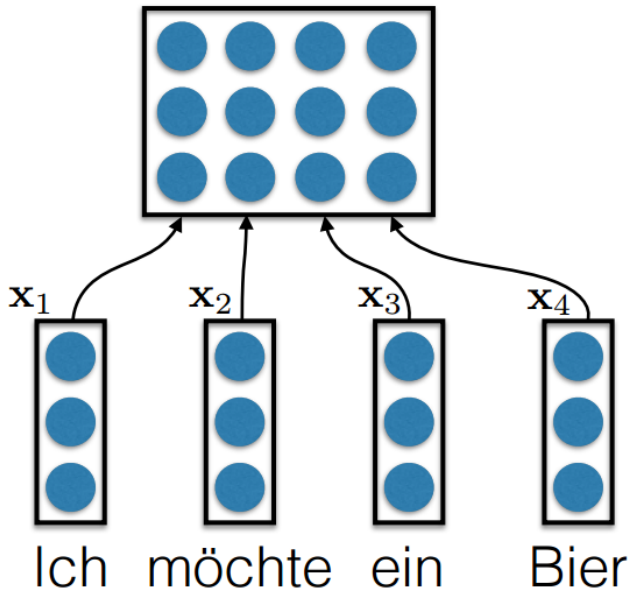
First shot: define the sentence words' vectors as the columns



(Image credit: Chris Dyer)

How to Encode a Sentence as a Matrix?

First shot: define the sentence words' vectors as the columns



(Image credit: Chris Dyer)

* Not very effective, since the word vectors carry no contextual information.

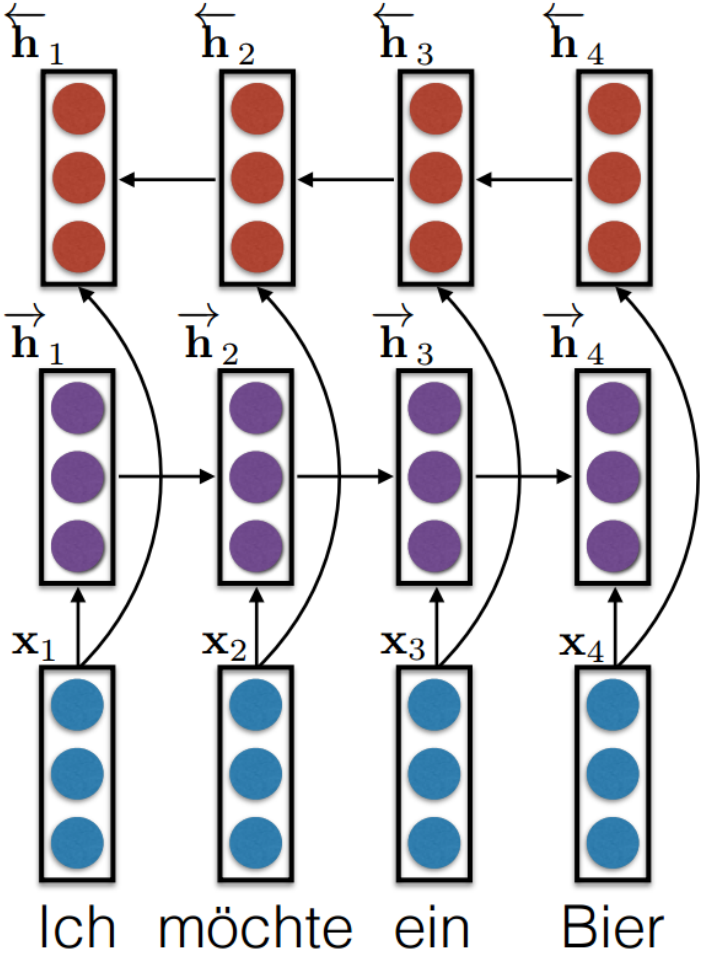
How to Encode a Sentence as a Matrix?

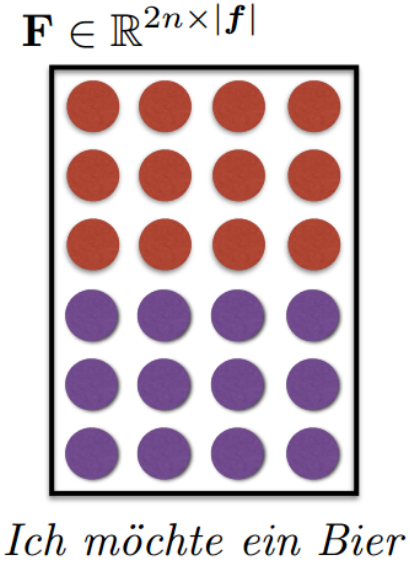
Other strategies:

- Convolutional neural networks (Kalchbrenner et al., 2014): can capture local context
- Bidirectional LSTMs (Bahdanau et al., 2015)
- Transformer networks (Vaswani et al., 2017)

Bidirectional LSTM Encoder

$$\mathbf{f}_i = [\overleftarrow{\mathbf{h}}_i; \overrightarrow{\mathbf{h}}_i]$$





(Slide credit: Chris Dyer)

Generation from Matrices

- We now have a matrix $\boldsymbol{F}$ representing the input.
- How to use it (e.g. for generation)?
- **Answer:** use attention! (Bahdanau et al., 2015)
- Attention is the neural counterpart of word alignments.

Generation from Matrices with Attention

- Generate the output sentence word by word using an RNN

Generation from Matrices with Attention

- Generate the output sentence word by word using an RNN
- At each output position t, the RNN receives two inputs:
  - a fixed-size vector embedding of the previous output symbol y_{t-1}
  - a fixed-size vector encoding a "view" of the input matrix $\boldsymbol{F}$, via a weighted sum of its columns (i.e., words): $\boldsymbol{F}\boldsymbol{a}_t$

Generation from Matrices with Attention

- Generate the output sentence word by word using an RNN
- At each output position t, the RNN receives two inputs:
  - a fixed-size vector embedding of the previous output symbol y_{t-1}
  - a fixed-size vector encoding a "view" of the input matrix $\boldsymbol{F}$, via a weighted sum of its columns (i.e., words): $\boldsymbol{F}\boldsymbol{a}_t$
- The weighting of the input columns at each time-step ($\boldsymbol{a}_t$) is called the attention distribution.

Attention Mechanism (Bahdanau et al., 2015)

Let $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots$ be the states produced by the decoder RNN

When predicting the t-th target word:

Attention Mechanism (Bahdanau et al., 2015)

Let $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots$ be the states produced by the decoder RNN

When predicting the t-th target word:

1. Compute "similarity" with each of the source words:

$$z_{t,i} = \boldsymbol{v} \cdot \boldsymbol{g}(\boldsymbol{W}\boldsymbol{h}_i + \boldsymbol{U}\boldsymbol{s}_{t-1} + \boldsymbol{b}), \quad \text{for } i = 1, ..., L$$

   where $\boldsymbol{h}_i$ is the ith column of $\boldsymbol{F}$ (representation of the ith source word), and $\boldsymbol{v}$, $\boldsymbol{W}$, $\boldsymbol{U}$, $\boldsymbol{b}$ are parameters of the model

Attention Mechanism (Bahdanau et al., 2015)

Let $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots$ be the states produced by the decoder RNN

When predicting the t-th target word:

1. Compute "similarity" with each of the source words:

$$z_{t,i} = \boldsymbol{v} \cdot \boldsymbol{g}(\boldsymbol{W}\boldsymbol{h}_i + \boldsymbol{U}\boldsymbol{s}_{t-1} + \boldsymbol{b}), \quad \text{for } i = 1, ..., L$$

   where $\boldsymbol{h}_i$ is the ith column of $\boldsymbol{F}$ (representation of the ith source word), and $\boldsymbol{v}$, $\boldsymbol{W}$, $\boldsymbol{U}$, $\boldsymbol{b}$ are parameters of the model

2. Form vector $\boldsymbol{z}_t = (z_{t,1}, \ldots, z_{t,i}, \ldots, z_{t,L})$ and compute attention $\boldsymbol{a}_t = \textbf{softmax}(\boldsymbol{z}_t)$

Attention Mechanism (Bahdanau et al., 2015)

Let $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots$ be the states produced by the decoder RNN

When predicting the t-th target word:

1. Compute "similarity" with each of the source words:

$$z_{t,i} = \boldsymbol{v} \cdot \boldsymbol{g}(\boldsymbol{W}\boldsymbol{h}_i + \boldsymbol{U}\boldsymbol{s}_{t-1} + \boldsymbol{b}), \quad \text{for } i = 1, ..., L$$

   where $\boldsymbol{h}_i$ is the ith column of $\boldsymbol{F}$ (representation of the ith source word), and $\boldsymbol{v}$, $\boldsymbol{W}$, $\boldsymbol{U}$, $\boldsymbol{b}$ are parameters of the model

2. Form vector $\boldsymbol{z}_t = (z_{t,1}, \ldots, z_{t,i}, \ldots, z_{t,L})$ and compute attention $\boldsymbol{a}_t = \textbf{softmax}(\boldsymbol{z}_t)$

3. Use attention to compute input conditioning state $\boldsymbol{c}_t = \boldsymbol{F}\boldsymbol{a}_t$

Attention Mechanism (Bahdanau et al., 2015)

Let $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots$ be the states produced by the decoder RNN

When predicting the t-th target word:

1. Compute "similarity" with each of the source words:

$$z_{t,i} = \boldsymbol{v} \cdot \boldsymbol{g}(\boldsymbol{W}\boldsymbol{h}_i + \boldsymbol{U}\boldsymbol{s}_{t-1} + \boldsymbol{b}), \quad \text{for } i = 1, ..., L$$

   where $\boldsymbol{h}_i$ is the ith column of $\boldsymbol{F}$ (representation of the ith source word), and $\boldsymbol{v}$, $\boldsymbol{W}$, $\boldsymbol{U}$, $\boldsymbol{b}$ are parameters of the model

2. Form vector $\boldsymbol{z}_t = (z_{t,1}, \ldots, z_{t,i}, \ldots, z_{t,L})$ and compute attention $\boldsymbol{a}_t = \textbf{softmax}(\boldsymbol{z}_t)$

3. Use attention to compute input conditioning state $\boldsymbol{c}_t = \boldsymbol{F}\boldsymbol{a}_t$

4. Update RNN state $\boldsymbol{s}_t$ based on $\boldsymbol{s}_{t-1}, y_{t-1}, \boldsymbol{c}_t$

Attention Mechanism (Bahdanau et al., 2015)

Let $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots$ be the states produced by the decoder RNN

When predicting the t-th target word:

1. Compute "similarity" with each of the source words:

$$z_{t,i} = \boldsymbol{v} \cdot \boldsymbol{g}(\boldsymbol{W}\boldsymbol{h}_i + \boldsymbol{U}\boldsymbol{s}_{t-1} + \boldsymbol{b}), \quad \text{for } i = 1, ..., L$$

   where $\boldsymbol{h}_i$ is the ith column of $\boldsymbol{F}$ (representation of the ith source word), and $\boldsymbol{v}$, $\boldsymbol{W}$, $\boldsymbol{U}$, $\boldsymbol{b}$ are parameters of the model

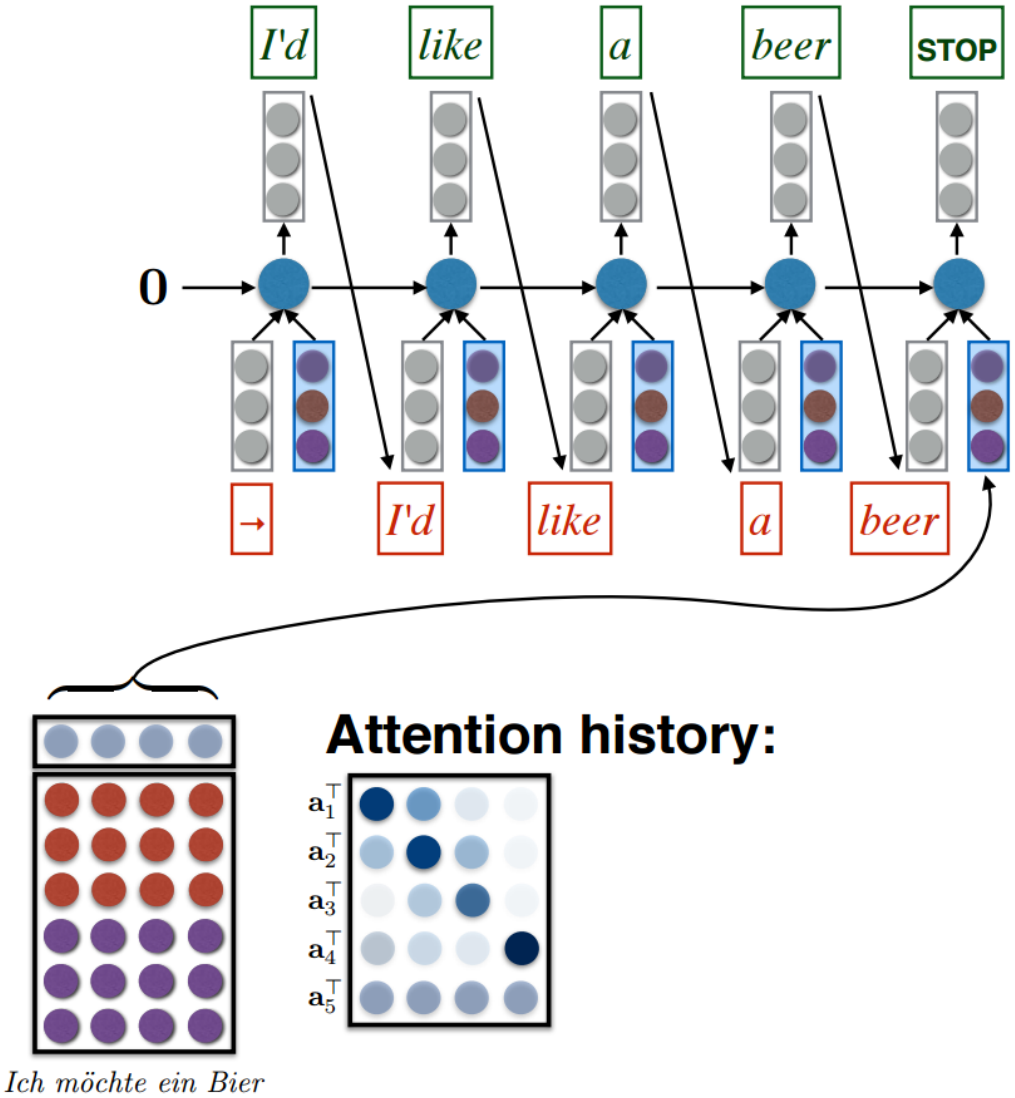
2. Form vector $\boldsymbol{z}_t = (z_{t,1}, \ldots, z_{t,i}, \ldots, z_{t,L})$ and compute attention $\boldsymbol{a}_t = \textbf{softmax}(\boldsymbol{z}_t)$

3. Use attention to compute input conditioning state $\boldsymbol{c}_t = \boldsymbol{F}\boldsymbol{a}_t$

4. Update RNN state $\boldsymbol{s}_t$ based on $\boldsymbol{s}_{t-1}, y_{t-1}, \boldsymbol{c}_t$

5. Predict $y_t \sim p(y_t \mid \boldsymbol{s}_t)$

Encoder-Decoder with Attention



(Slide credit: Chris Dyer)

Putting It All Together

obtain input matrix $\boldsymbol{F}$ with a bidirectional LSTM
$t = 0$, $y_0 = \text{START}$ (the start symbol)
$\boldsymbol{s}_0 = \boldsymbol{w}$ (learned initial state)
repeat
  $t = t + 1$
  $\boldsymbol{z}_t = \boldsymbol{v} \cdot \boldsymbol{g}(\boldsymbol{W}\boldsymbol{F} + \boldsymbol{U}\boldsymbol{s}_{t-1} + \boldsymbol{b})$
  compute attention $\boldsymbol{a}_t = \textbf{softmax}(\boldsymbol{z}_t)$
  compute input conditioning state $\boldsymbol{c}_t = \boldsymbol{F}\boldsymbol{a}_t$
  $\boldsymbol{s}_t = \textbf{RNN}(\boldsymbol{s}_{t-1}, [\boldsymbol{E}(y_{t-1}), \boldsymbol{c}_t])$
  $y_t | y_{<t}, \boldsymbol{x} \sim \textbf{softmax}(\boldsymbol{P}\boldsymbol{s}_t + \boldsymbol{b})$
until $y_t \neq \text{STOP}$

Attention Mechanisms

- Attention is closely related to "pooling" operations in CNNs (and other architectures)

Attention Mechanisms

- Attention is closely related to "pooling" operations in CNNs (and other architectures)
- Attention in MT plays a similar role as alignment, but leads to "soft" alignment instead of "hard" alignment

Attention Mechanisms

- Attention is closely related to "pooling" operations in CNNs (and other architectures)
- Attention in MT plays a similar role as alignment, but leads to "soft" alignment instead of "hard" alignment
- Bahdanau et al. (2015)'s model has no bias in favor of diagonals, short jumps, fertility, etc.

Attention Mechanisms

- Attention is closely related to "pooling" operations in CNNs (and other architectures)
- Attention in MT plays a similar role as alignment, but leads to "soft" alignment instead of "hard" alignment
- Bahdanau et al. (2015)'s model has no bias in favor of diagonals, short jumps, fertility, etc.
- Some recent work adds some "structural" biases (Luong et al., 2015; Cohn et al., 2016)

Attention Mechanisms

- Attention is closely related to "pooling" operations in CNNs (and other architectures)
- Attention in MT plays a similar role as alignment, but leads to "soft" alignment instead of "hard" alignment
- Bahdanau et al. (2015)'s model has no bias in favor of diagonals, short jumps, fertility, etc.
- Some recent work adds some "structural" biases (Luong et al., 2015; Cohn et al., 2016)
- Other works constrains the amount of attention each word can receive (based on its fertility): Malaviya et al. (2018).

Attention Mechanisms

- Attention is closely related to "pooling" operations in CNNs (and other architectures)
- Attention in MT plays a similar role as alignment, but leads to "soft" alignment instead of "hard" alignment
- Bahdanau et al. (2015)'s model has no bias in favor of diagonals, short jumps, fertility, etc.
- Some recent work adds some "structural" biases (Luong et al., 2015; Cohn et al., 2016)
- Other works constrains the amount of attention each word can receive (based on its fertility): Malaviya et al. (2018).
- Attention comes in many flavours: additive, dot-product, ...

Attention is Great!

- Attention significantly improves NMT performance!

Attention is Great!

- Attention significantly improves NMT performance!
- It's very useful to allow decoder to focus on certain parts of the source

Attention is Great!

- Attention significantly improves NMT performance!
- It's very useful to allow decoder to focus on certain parts of the source
- Solves the bottleneck problem (by allowing the decoder to look directly at source)

Attention is Great!

- Attention significantly improves NMT performance!
- It's very useful to allow decoder to focus on certain parts of the source
- Solves the bottleneck problem (by allowing the decoder to look directly at source)
- Helps with vanishing gradient problem (provides shortcut to faraway states)

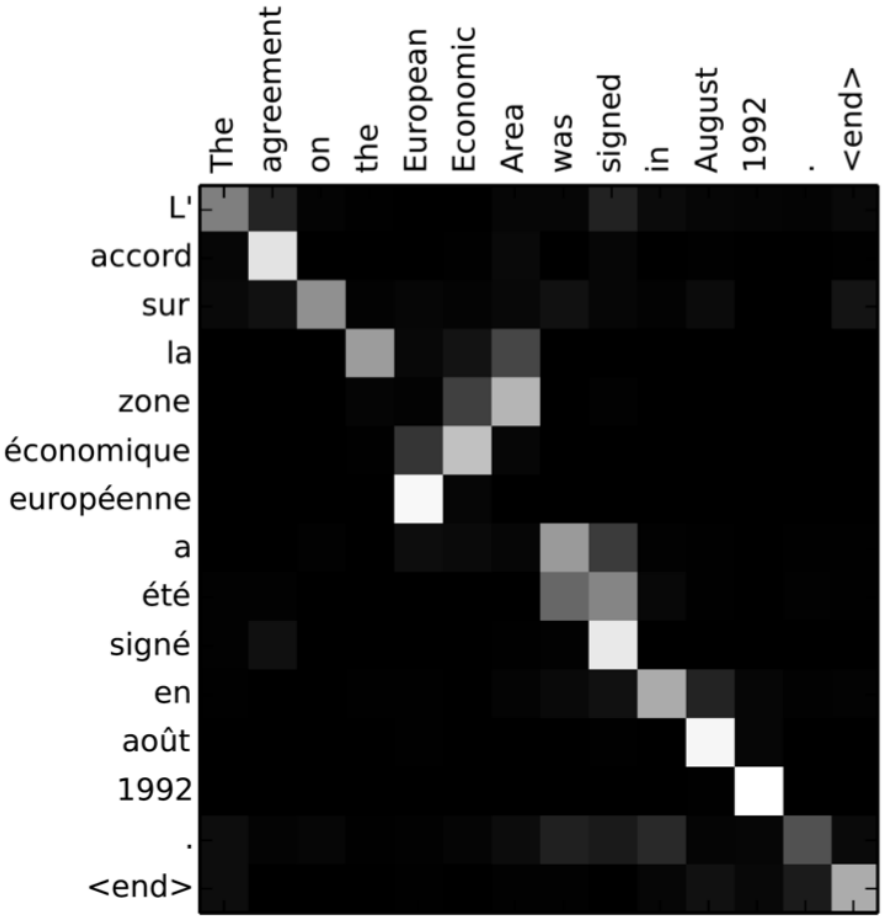
Attention is Great!

- Attention significantly improves NMT performance!
- It's very useful to allow decoder to focus on certain parts of the source
- Solves the bottleneck problem (by allowing the decoder to look directly at source)
- Helps with vanishing gradient problem (provides shortcut to faraway states)
- Provides some interpretability (we can see what the decoder was focusing on)

Attention is Great!

- Attention significantly improves NMT performance!
- It's very useful to allow decoder to focus on certain parts of the source
- Solves the bottleneck problem (by allowing the decoder to look directly at source)
- Helps with vanishing gradient problem (provides shortcut to faraway states)
- Provides some interpretability (we can see what the decoder was focusing on)
- We never explicitly trained a word aligner; the network learns it by itself!

Attention Map



Dzmitry Bahdanau, KyungHuyn Cho, and Yoshua Bengio. ***Neural Machine Translation by Jointly Learning to Translate and Align****. ICLR'15.*

Example: Machine Translation

Some positive examples where NMT has impressive performance:

Source	When asked about this, an official of the American administration replied: "The United States is not conducting electronic surveillance aimed at offices of the World Bank and IMF in Washington."	
PBMT	Interrogé à ce sujet, un responsable de l'administration américaine a répondu : "Les Etats-Unis n'est pas effectuer une surveillance électronique destiné aux bureaux de la Banque mondiale et du FMI à Washington".	3.0
GNMT	Interrogé à ce sujet, un fonctionnaire de l'administration américaine a répondu: "Les États-Unis n'effectuent pas de surveillance électronique à l'intention des bureaux de la Banque mondiale et du FMI à Washington".	6.0
Human	Interrogé sur le sujet, un responsable de l'administration américaine a répondu: "les Etats-Unis ne mènent pas de surveillance électronique visant les sièges de la Banque mondiale et du FMI à Washington".	6.0
Source	Martin told CNN that he asked Daley whether his then-boss knew about the potential shuffle.	
PBMT	Martin a déclaré à CNN qu'il a demandé Daley si son patron de l'époque connaissaient le potentiel remaniement ministériel.	2.0
GNMT	Martin a dit à CNN qu'il avait demandé à Daley si son patron d'alors était au courant du remaniement potentiel.	6.0
Human	Martin a dit sur CNN qu'il avait demandé à Daley si son patron d'alors était au courant du remaniement éventuel.	5.0

(From Wu et al. (2016))

Example: Machine Translation

... But also some negative examples:

- Dropping source words (lack of attention)
- Repeated source words (too much attention)

Source:	1922 in Wien geboren, studierte Mang während und nach dem Zweiten Weltkrieg Architektur an der Technischen Hochschule in Wien bei Friedrich Lehmann.
Human:	Born in Vienna in 1922, Meng studied architecture at the Technical University in Vienna under Friedrich Lehmann *during and after the second World War.*
NMT:	*Born in Vienna in 1922, Mang studied architecture at the Technical College in Vienna with Friedrich Lehmann.
Source:	Es ist schon komisch, wie dies immer wieder zu dieser Jahreszeit auftaucht.
Human:	It's funny how this always comes up at *this time* of year.
NMT:	*It's funny how **this time** to come back to **this time** of year.

Example: Machine Translation

... And an example where neural MT failed miserably:

Source	A two - out walk to right fielder J . D . Martinez brought up Victor Martinez , who singled up the middle for the first run of the game .
Reference	Dva odchody pro pravého polaře J . D . Martineze vynesly Victora Martineze , který jako první oběhl všechny mety .
online-B	Dva - out chůze doprava Fielder J . D . Martinez vychován Victor Martinez , který vybral do středu za prvním spuštění hry .
uedin-nmt	Ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne , ne ,

(Credit: Barry Haddow)

Example: Document Classification

Task: Hotel location

you get what you pay for . not the cleanest rooms but bed was clean and so was bathroom . bring your own towels though as very thin . service was excellent , let us book in at 8:30am ! **for location and price , this ca n't be beaten** , but it is cheap for a reason . if you come expecting the hilton , then book the hilton ! for uk travellers , think of a blackpool b&b.

Task: Hotel cleanliness

you get what you pay for . **not the cleanest rooms but bed was clean and so was bathroom** . bring your own towels though as very thin . service was excellent , let us book in at 8:30am ! for location and price , this ca n't be beaten , but it is cheap for a reason . if you come expecting the hilton , then book the hilton ! for uk travellers , think of a blackpool b&b.

Task: Hotel service

you get what you pay for . not the cleanest rooms but bed was clean and so was bathroom . bring your own towels though as very thin . **service was excellent** , let us book in at 8:30am ! for location and price , this ca n't be beaten , but it is cheap for a reason . if you come expecting the hilton , then book the hilton ! for uk travellers , think of a blackpool b&b.

(Bao et al., 2018)

Example: Caption Generation

Attention over images:

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

(Slide credit to Yoshua Bengio)

Attention and Memories

Attention is used in several problems, sometimes under different names:

- image caption generation (Xu et al., 2015)
- speech recognition (Chorowski et al., 2015)
- memory networks for reading comprehension (Sukhbaatar et al., 2015; Hermann et al., 2015)
- neural Turing machines and other "differentiable computers" (Graves et al., 2014; Grefenstette et al., 2015)

Other Attentions

- Can we have more interpretable attention? Closer to hard alignments?

Other Attentions

- Can we have more interpretable attention? Closer to hard alignments?
- Can we upper bound how much attention a word receives? This may prevent a common problem in neural MT, repetitions

Other Attentions

- Can we have more interpretable attention? Closer to hard alignments?
- Can we upper bound how much attention a word receives? This may prevent a common problem in neural MT, repetitions
- Sparse attention via **sparsemax** (Martins and Astudillo, 2016)

Other Attentions

- Can we have more interpretable attention? Closer to hard alignments?
- Can we upper bound how much attention a word receives? This may prevent a common problem in neural MT, repetitions
- Sparse attention via **sparsemax** (Martins and Astudillo, 2016)
- Constrained attention with constrained softmax/sparsemax (Malaviya et al., 2018)

Outline

Next: Self-Attention

- Both RNN and CNN decoders require an attention mechanism
- Attention allows focusing on an arbitrary position in the source sentence, shortcutting the computation graph
- But if attention gives us access to any state...
  ...maybe we don't need the RNN?

Outline

Attention Mechanism overview

Lets generalise how attention works:

1. We have a query vector $\boldsymbol{q}$ (e.g. the decoder state)
2. We have input vectors $\boldsymbol{H} = [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_L]^\top$ (e.g. one per source word)
3. We compute affinity scores $s_1, \ldots, s_L$ by "comparing" $\boldsymbol{q}$ and $\boldsymbol{H}$
4. We convert these scores to probabilities:

$$\boldsymbol{p} = \mathbf{softmax}(\boldsymbol{s})$$

5. We use this to output a representation as a weighted average:

$$\boldsymbol{c} = \boldsymbol{H}^\top \boldsymbol{p} = \sum_{i=1}^{L} p_i \boldsymbol{h}_i$$

Let's see these steps in detail!

Affinity Scores

Several ways of "comparing" a query $\boldsymbol{q}$ and an input ("key") vector $\boldsymbol{h}_i$:

- Additive attention (Bahdanau et al., 2015)

$$s_i = \boldsymbol{u}^\top \tanh(\boldsymbol{A}\boldsymbol{h}_i + \boldsymbol{B}\boldsymbol{q})$$

- Bilinear attention (Luong et al., 2015):

$$s_i = \boldsymbol{q}^\top \boldsymbol{U}\boldsymbol{h}_i$$

- Dot product attention (Luong et al., 2015) (particular case; queries and keys must have the same size):

$$s_i = \boldsymbol{q}^\top \boldsymbol{h}_i$$

The last two are easier to batch when we have multiple queries and multiple keys.

Keys and Values

The input vectors $\boldsymbol{H} = [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_L]^\top$ appear in two places:

- They are used as keys to "compare" them with the query vector $\boldsymbol{q}$ to obtain the affinity scores
- They are used as values to form the weighted average $\boldsymbol{c} = \boldsymbol{H}^\top \boldsymbol{p}$

Keys and Values

The input vectors $\boldsymbol{H} = [\boldsymbol{h}_1, \ldots, \boldsymbol{h}_L]^\top$ appear in two places:

- They are used as keys to "compare" them with the query vector $\boldsymbol{q}$ to obtain the affinity scores
- They are used as values to form the weighted average $\boldsymbol{c} = \boldsymbol{H}^\top \boldsymbol{p}$

To be fully general, they don't need to be the same – we can have:

- A key matrix $\boldsymbol{K} = [\boldsymbol{k}_1, \ldots, \boldsymbol{k}_L]^\top \in \mathbb{R}^{L \times d_K}$
- A value matrix $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_L]^\top \in \mathbb{R}^{L \times d_V}$

Attention Mechanism: More General Version

1. We have a query vector $\boldsymbol{q}$ (e.g. the decoder state)
2. We have key vectors $\boldsymbol{K} = [\boldsymbol{k}_1, \ldots, \boldsymbol{k}_L]^\top \in \mathbb{R}^{L \times d_K}$
   and value vectors $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_L]^\top \in \mathbb{R}^{L \times d_V}$
   (e.g. one of each per source word)
3. We compute query-key affinity scores $s_1, \ldots, s_L$ "comparing" $\boldsymbol{q}$ and $\boldsymbol{K}$
4. We convert these scores to probabilities:

$$\boldsymbol{p} = \mathbf{softmax}(\boldsymbol{s})$$

5. We output a weighted average of the values:

$$\boldsymbol{c} = \boldsymbol{V}^\top \boldsymbol{p} = \sum_{i=1}^{L} p_i \boldsymbol{v}_i \in \mathbb{R}^{d_V}$$

Self-Attention

- So far we talked about contextual attention – the decoder attends to encoder states (this is called "input context")
- The encoder and the decoder states were propagated sequentially with a RNN, or hierarchically with a CNN
- Alternative: self-attention – at each position, the encoder attends to the other positions in the encoder itself
- Same for the decoder.

Self-Attention Layer

Self-attention for a sequence of length L:

1. Query vectors $\boldsymbol{Q} = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_L]^\top \in \mathbb{R}^{L \times d_Q}$
2. Key vectors $\boldsymbol{K} = [\boldsymbol{k}_1, \ldots, \boldsymbol{k}_L]^\top \in \mathbb{R}^{L \times d_K}$
3. value vectors $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_L]^\top \in \mathbb{R}^{L \times d_V}$
4. Compute query-key affinity scores "comparing" $\boldsymbol{Q}$ and $\boldsymbol{K}$, e.g.,

$$\boldsymbol{S} = \boldsymbol{Q}\boldsymbol{K}^\top \in \mathbb{R}^{L \times L} \qquad \text{(dot-product affinity)}$$

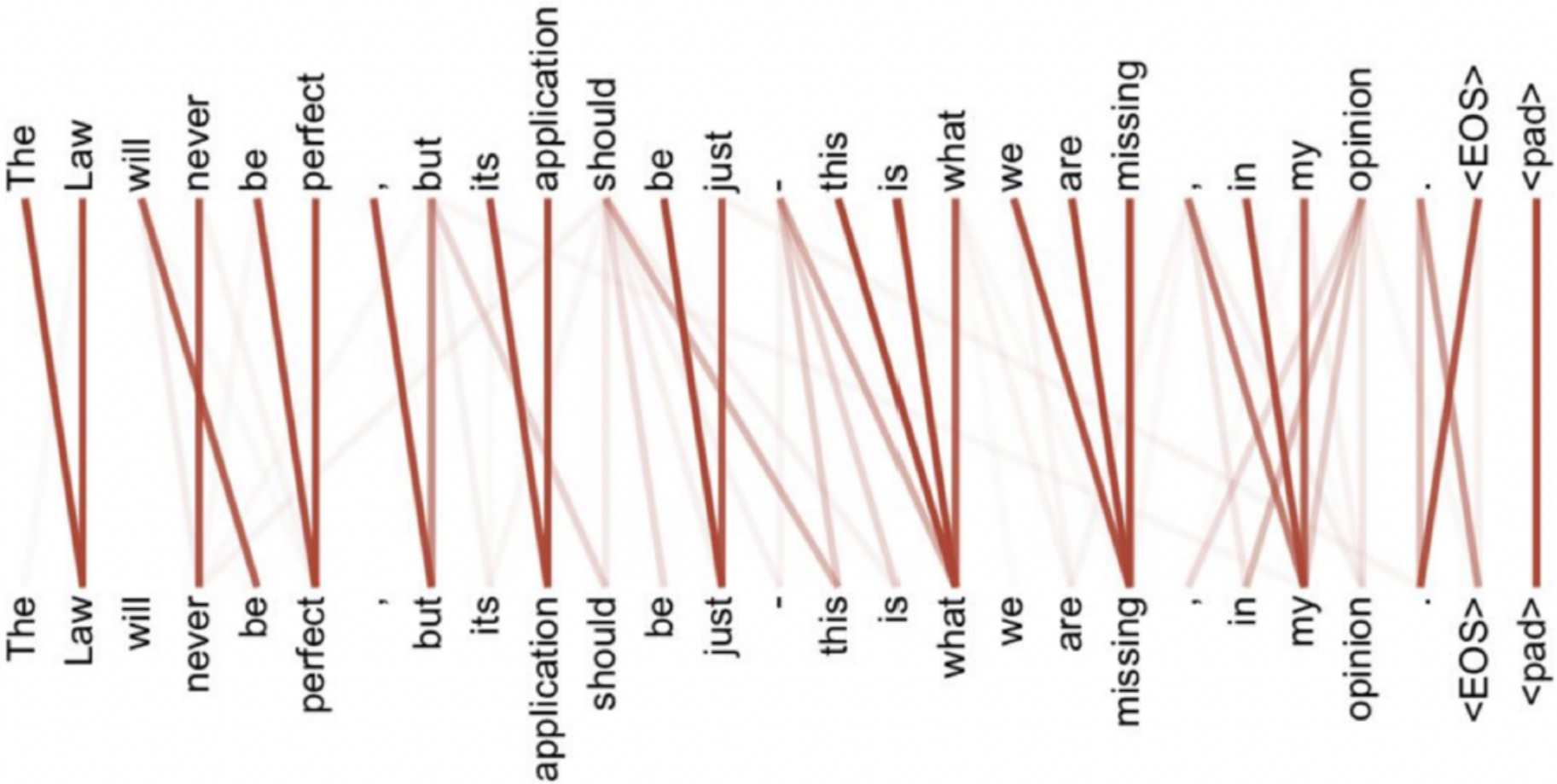
5. Convert these scores to probabilities (row-wise):

$$\boldsymbol{P} = \mathbf{softmax}(\boldsymbol{S}) \in \mathbb{R}^{L \times L}$$

6. Output the weighted average of the values:

$$\boldsymbol{Z} = \boldsymbol{P}\boldsymbol{V} = \underbrace{\mathbf{softmax}(\boldsymbol{Q}\boldsymbol{K}^\top)}_{\boldsymbol{P}}\boldsymbol{V} \in \mathbb{R}^{L \times d_V}.$$

Self-Attention



(Vaswani et al., 2017)

Transformer (Vaswani et al., 2017)

- **Key idea:** instead of RNN/CNNs, use self-attention in the encoder
- Each word state attends to all the other words
- Each self-attention is followed by a feed-forward transformation
- Do several layers of this
- Do the same for the decoder, attending only to already generated words.

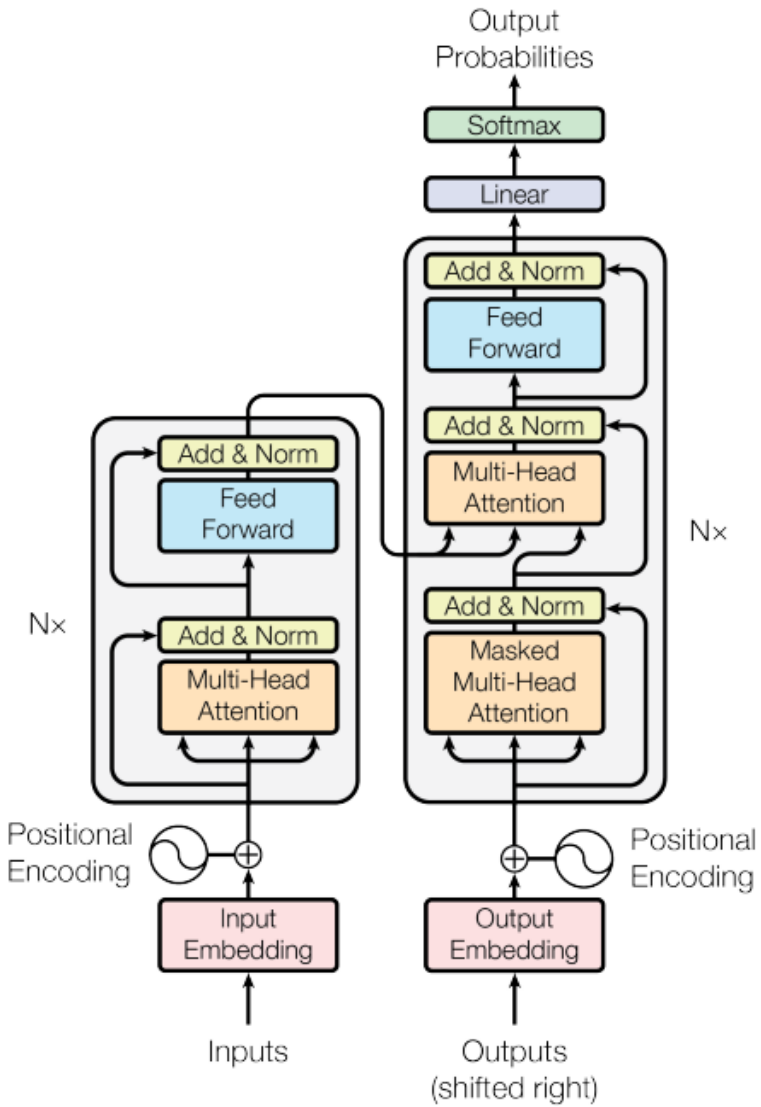
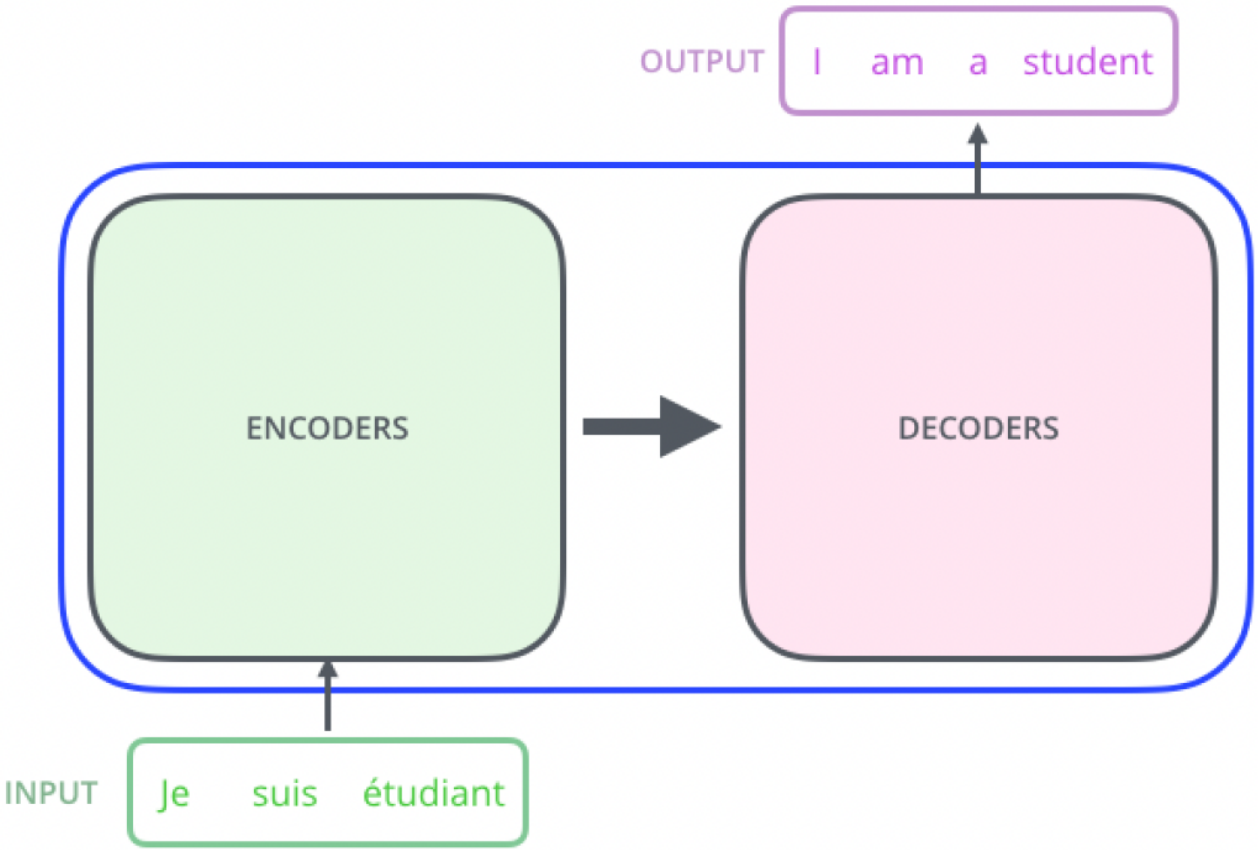


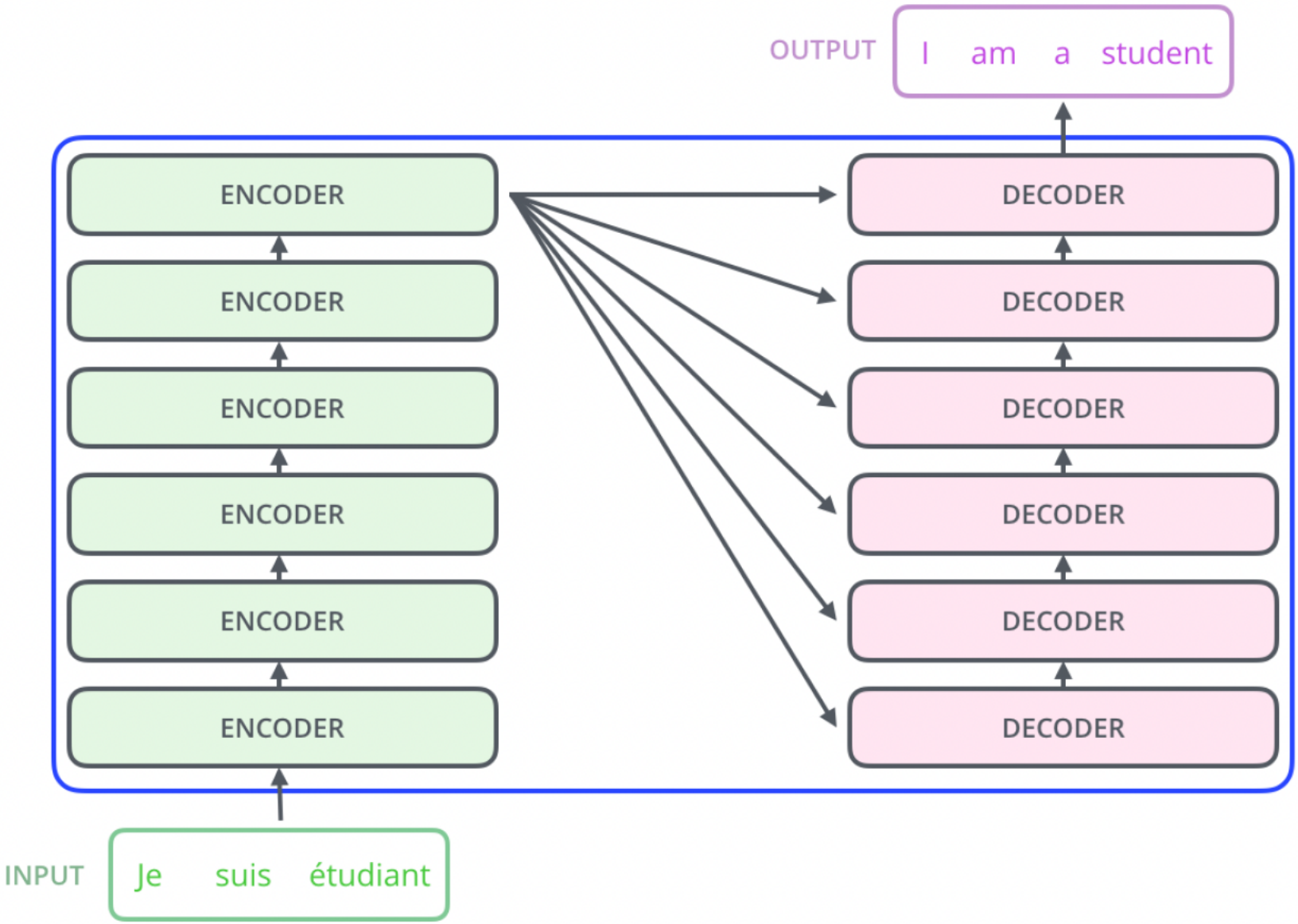
Figure 1: The Transformer - model architecture.

Transformer

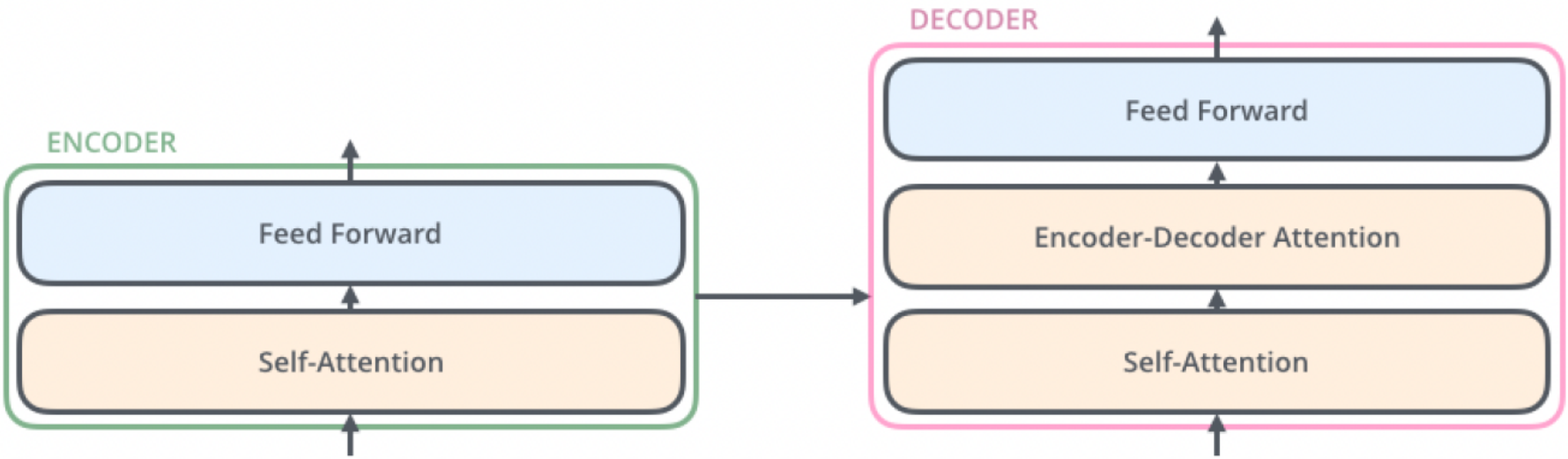


$$x_1\, x_2 \ldots x_n \xrightarrow{\text{encode}} \mathbf{r}_1\, \mathbf{r}_2 \ldots \mathbf{r}_n \xrightarrow{\text{decode}} y_1\, y_2 \cdots y_m$$

Transformer



Transformer Blocks



(Illustrated transformer: `http://jalammar.github.io/illustrated-transformer/`)

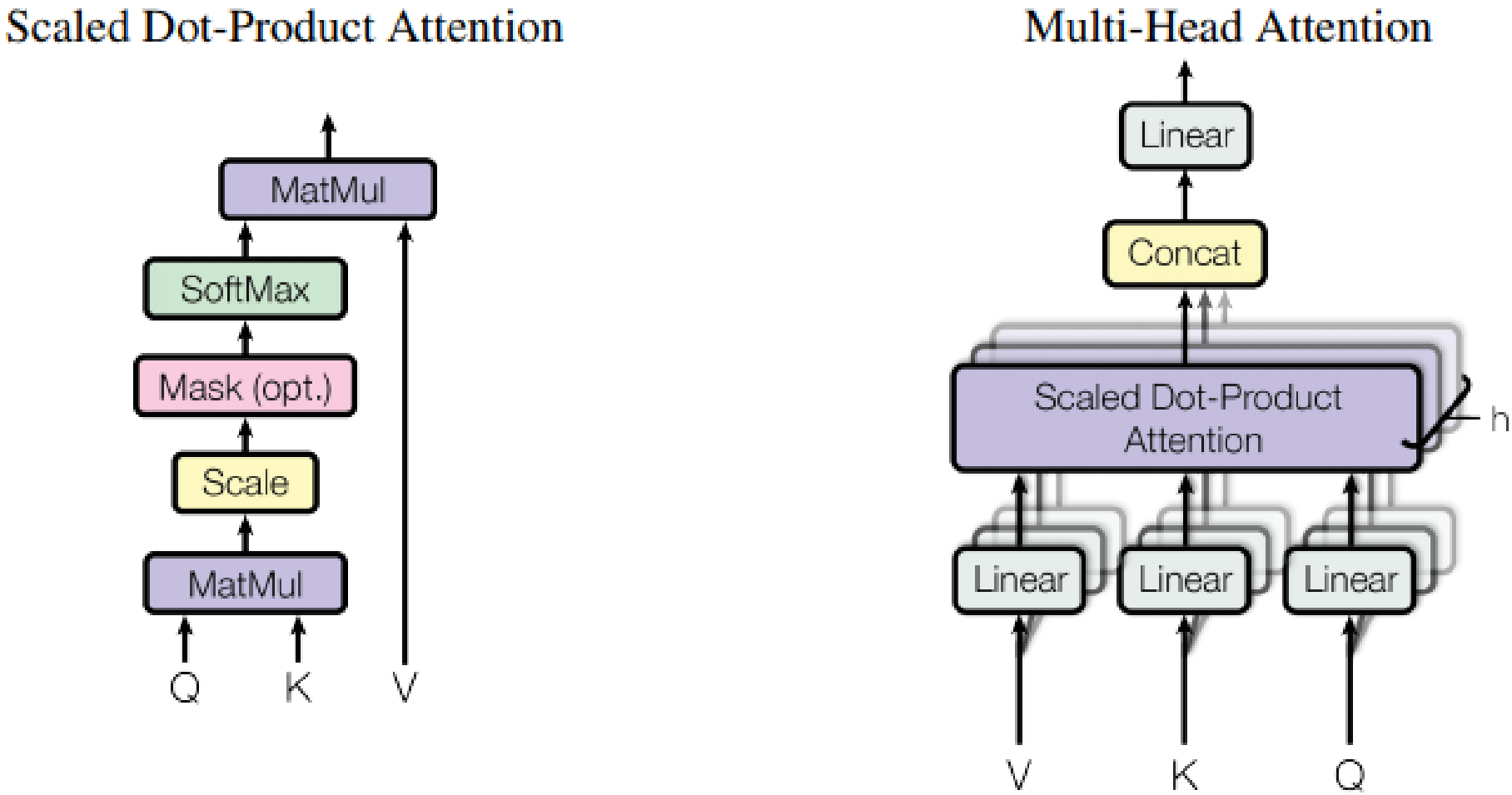
Transformer Basics

Let's define the basic building blocks of transformer networks first: new attention layers!

Two innovations:

- scaled dot-product attention
- multi-head attention

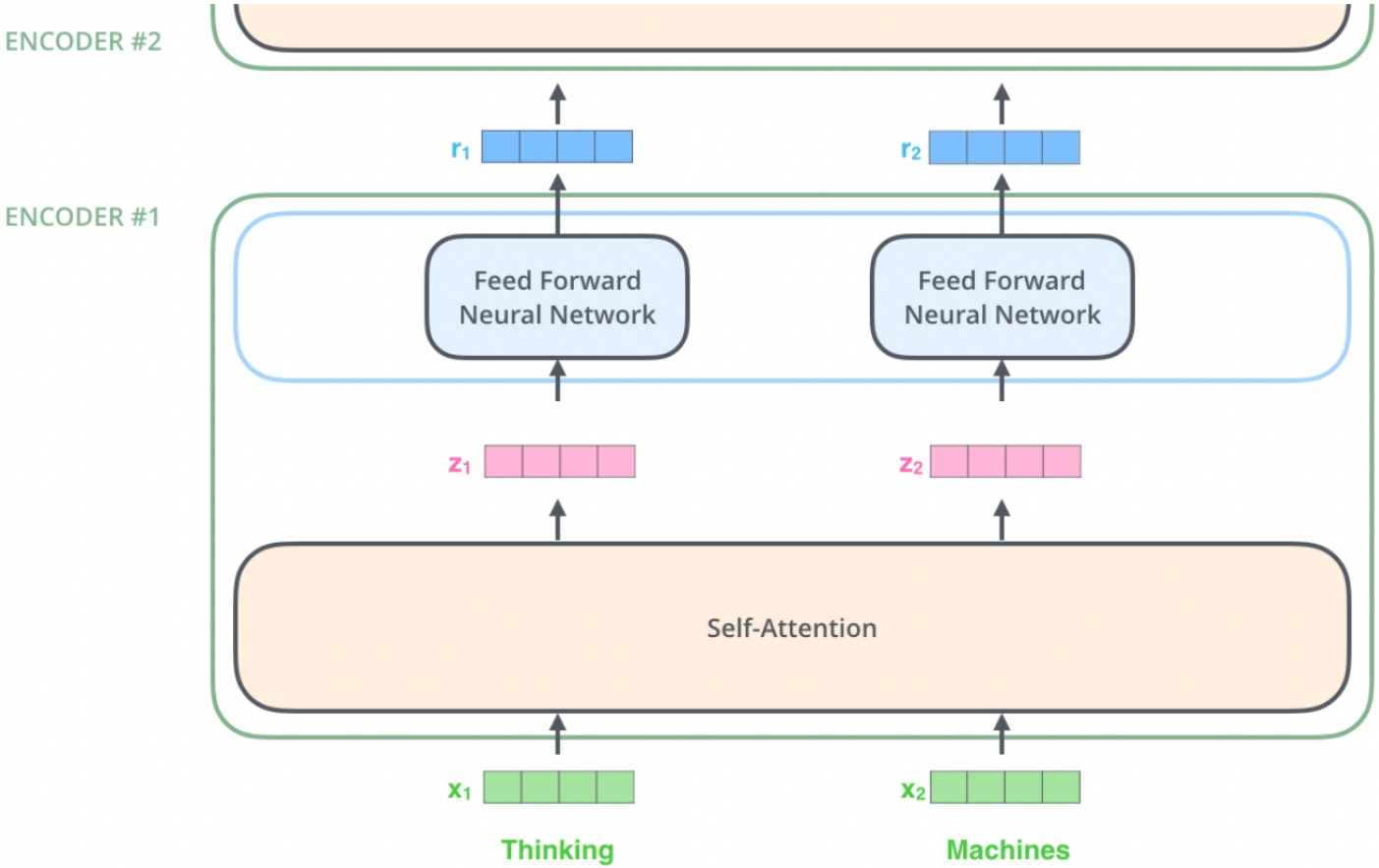
Scaled Dot-Product and Multi-Head Attention



(Vaswani et al., 2017)

The Encoder

Example for a sentence with 2 words:

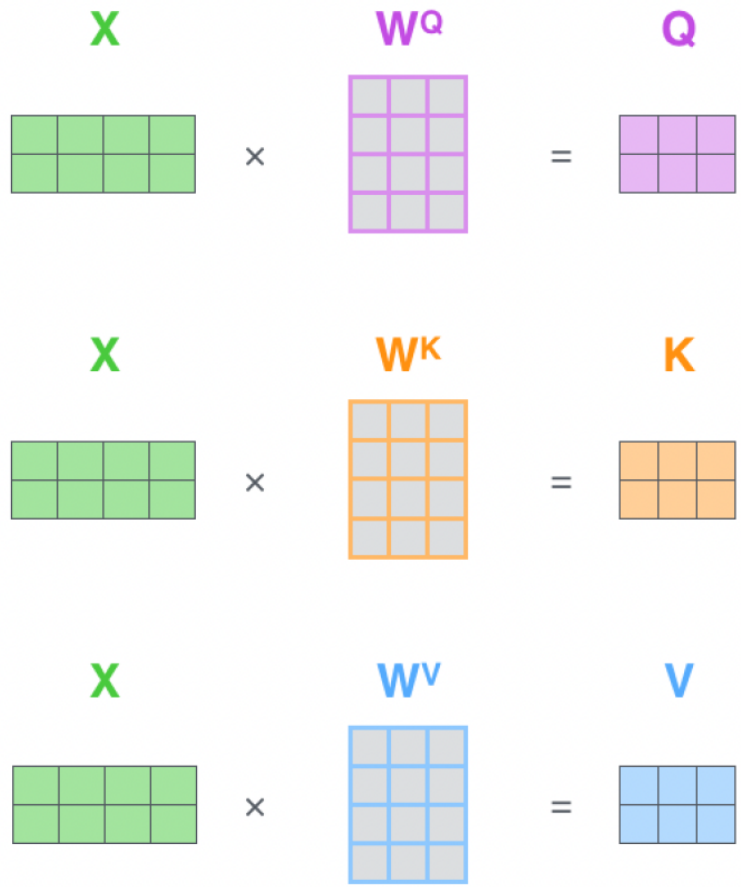


Transformer Self-Attention: Queries, Keys, Vectors

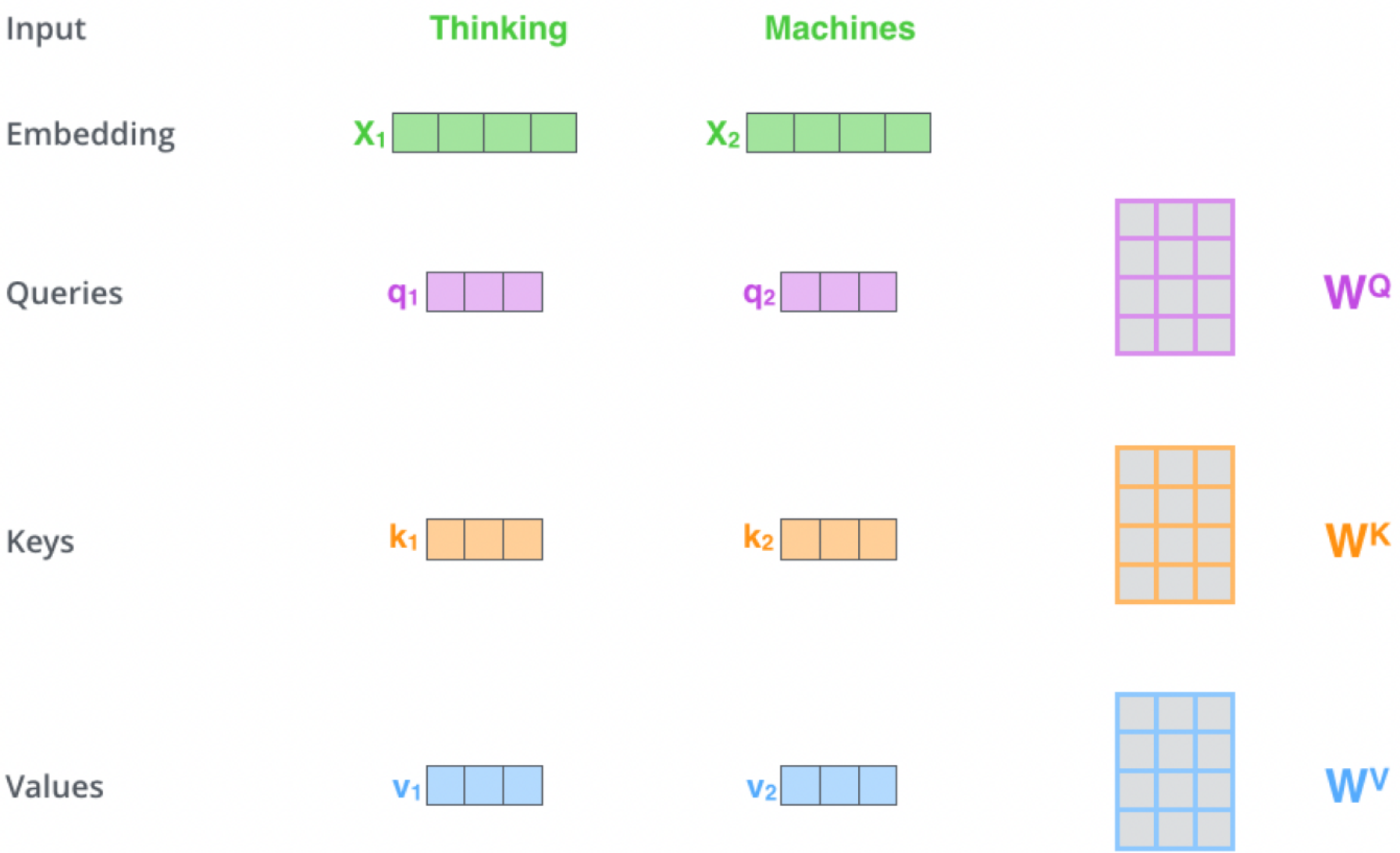
- Obtained by projecting the embedding matrix $\boldsymbol{X} \in \mathbb{R}^{L \times e}$ to a lower dimension:

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^{Q}$$
$$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^{K}$$
$$\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^{V}.$$

- The projection matrices $\boldsymbol{W}^{Q}$, $\boldsymbol{W}^{K}$, $\boldsymbol{W}^{V}$ are model parameters.



Transformer Self-Attention: Queries, Keys, Vectors



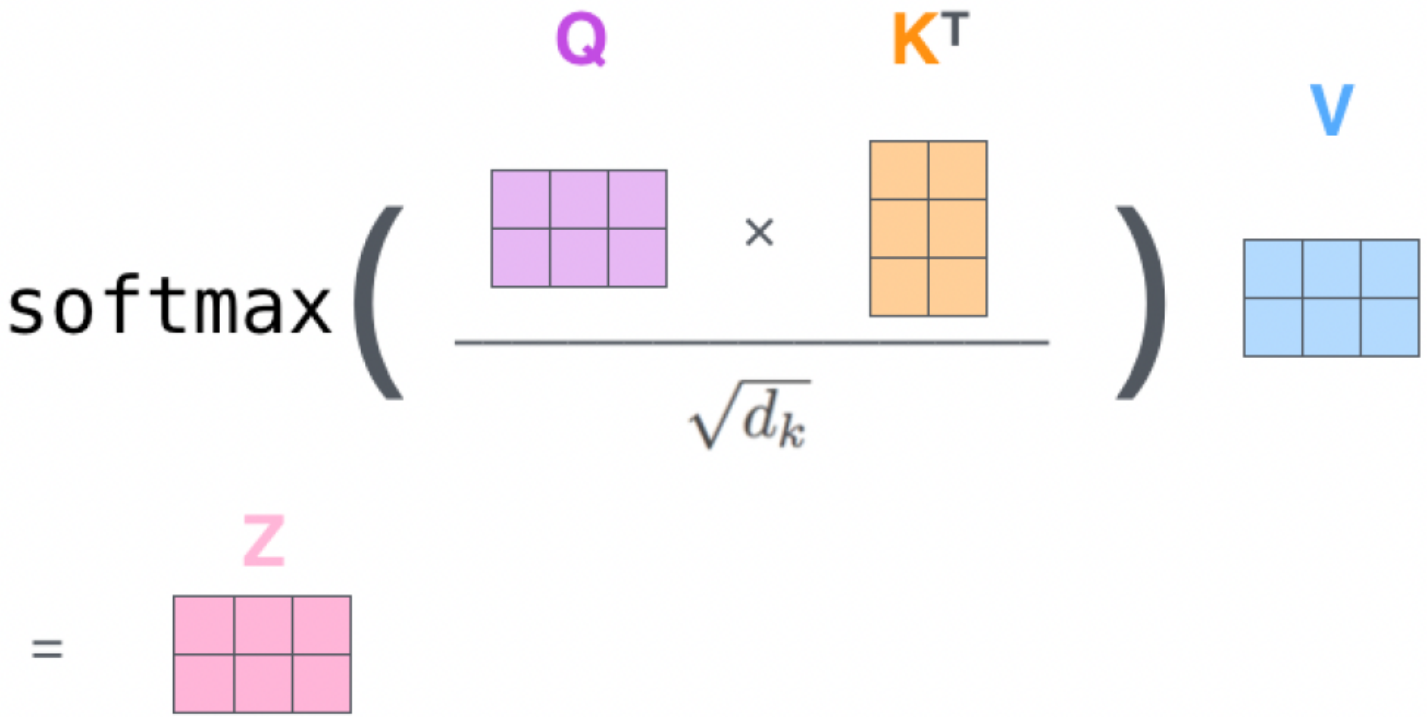
Scaled Dot-Product Attention

Problem: As d_K gets large, the variance of $\boldsymbol{q}^\top \boldsymbol{k}$ increases, the softmax gets very peaked, hence its gradient gets smaller.

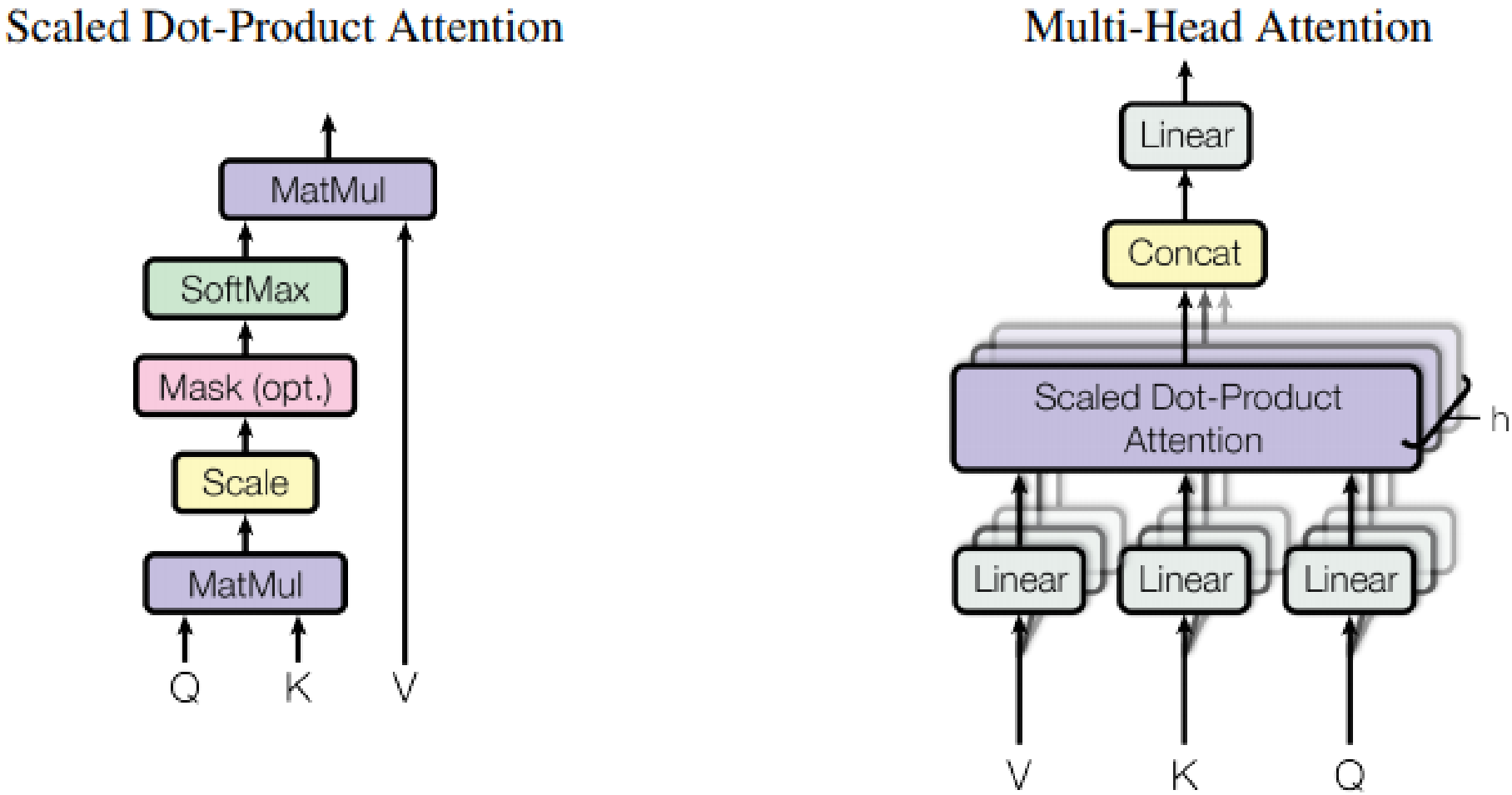
Solution: scale by length of query/key vectors:

$$\boldsymbol{Z} = \textbf{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_K}}\right)\boldsymbol{V}.$$

Scaled Dot-Product Attention



Scaled Dot-Product and Multi-Head Attention



(Vaswani et al., 2017)

Multi-Head Attention

Self-attention: each word forms a query vector and attends to the other words' key vectors

This is vaguely similar to a 1D convolution, but where the filter weights are "dynamic" is the window size spans the entire sentence!

Problem: only one channel for words to interact with one-another

Solution: multi-head attention!

- define h attention heads, each with their own projection matrices (e.g. $h = 8$)
- apply attention in multiple channels, concatenate the outputs and pipe through linear layer:

$$\text{MultiHead}(\boldsymbol{X}) = \text{Concat}(\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_h)\boldsymbol{W}^O,$$

where $\boldsymbol{Z}_i = \text{Attention}(\underbrace{\boldsymbol{X}\boldsymbol{W}_i^Q}_{\boldsymbol{Q}_i}, \underbrace{\boldsymbol{X}\boldsymbol{W}_i^K}_{\boldsymbol{K}_i}, \underbrace{\boldsymbol{X}\boldsymbol{W}_i^V}_{\boldsymbol{V}_i})$.

Multi-Head Attention

1) This is our input sentence*

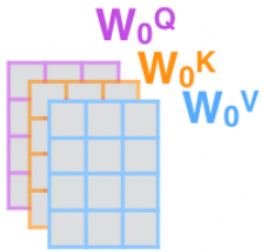
2) We embed each word*

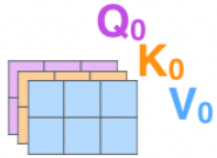
3) Split into 8 heads. We multiply X or R with weight matrices

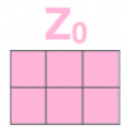
4) Calculate attention using the resulting Q/K/V matrices

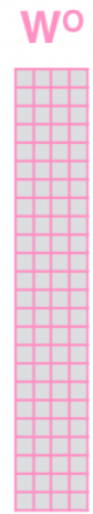
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

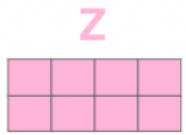
Thinking Machines



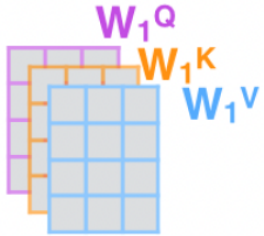


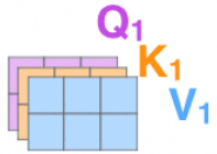


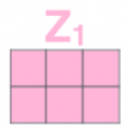




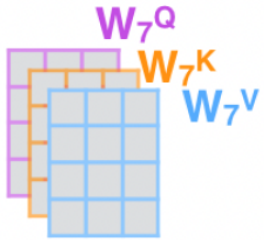
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

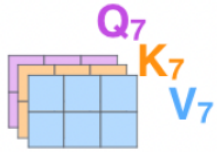


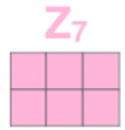




...







Other Tricks

- Self-attention blocks are repeated several times (e.g. 6 or 12)
- Residual connections on each attention block
- Layer normalization
- Positional encodings (to distinguish word positions)

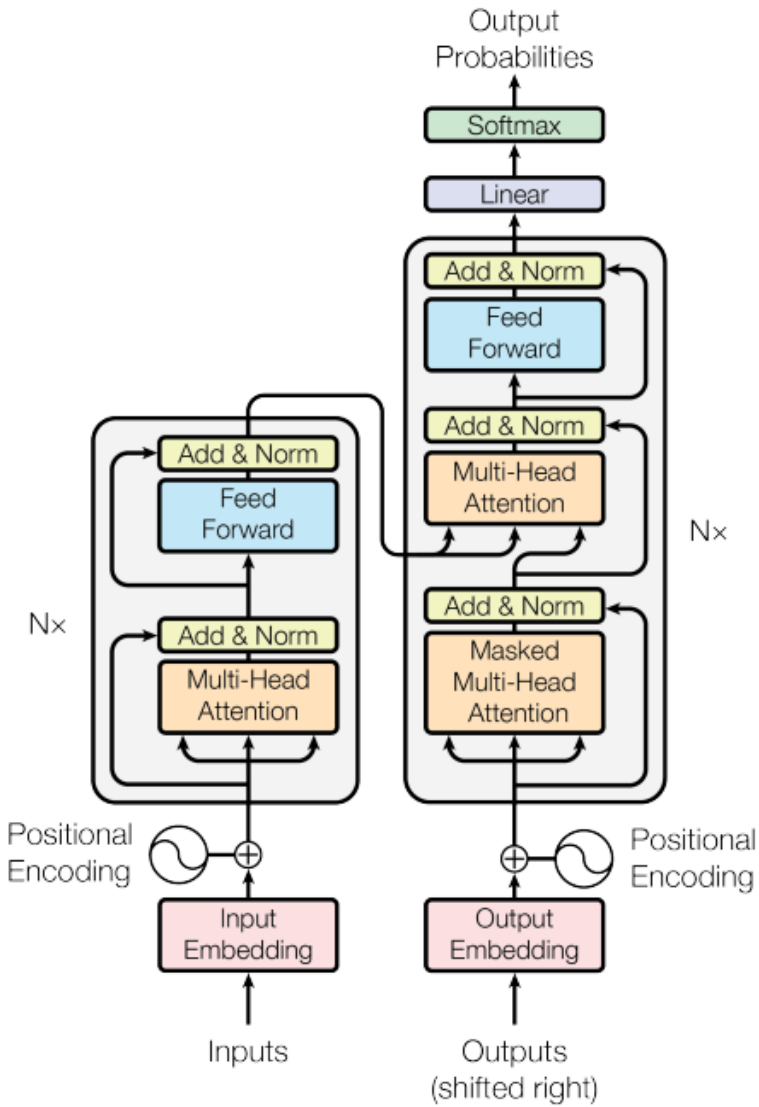
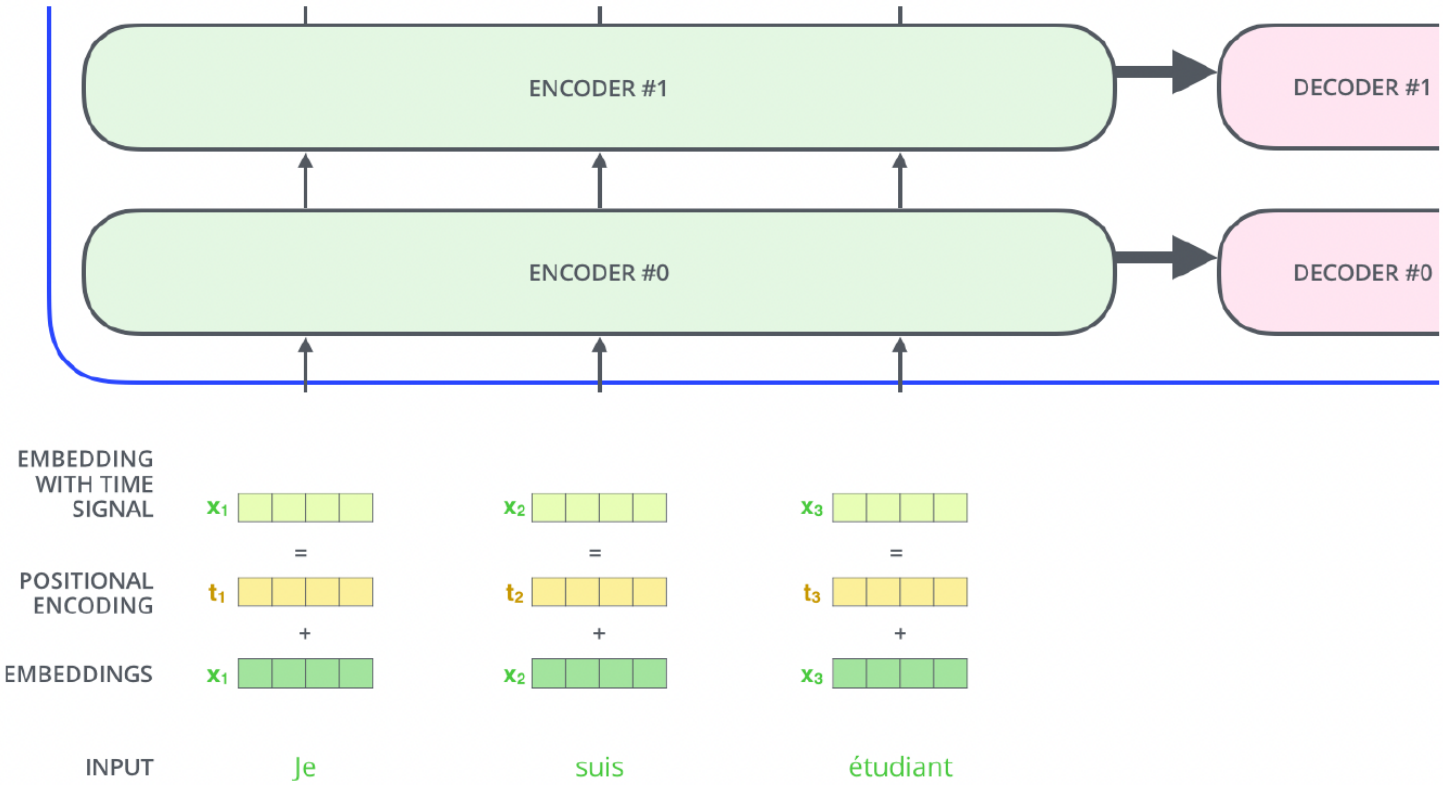


Figure 1: The Transformer - model architecture.

Positional Encodings

- As just described, the transformer is insensitive to word order!
  - queries attend to keys regardless of their position in the sequence
- To make it sensitive to order, we add positional encodings
- Two strategies: learn one embedding for each position (up to a maximum length) or use sinusoidal positional encodings (next)



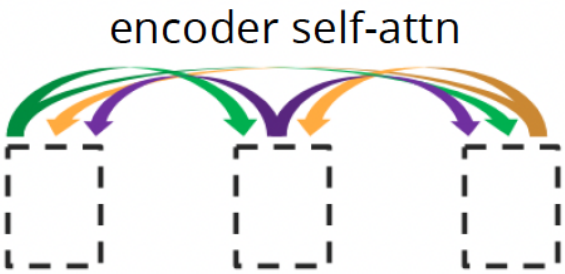
The Decoder

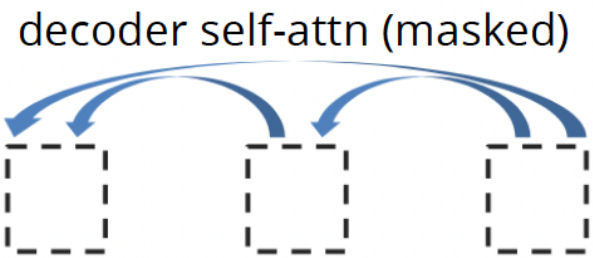
What about the self-attention blocks in the decoder?

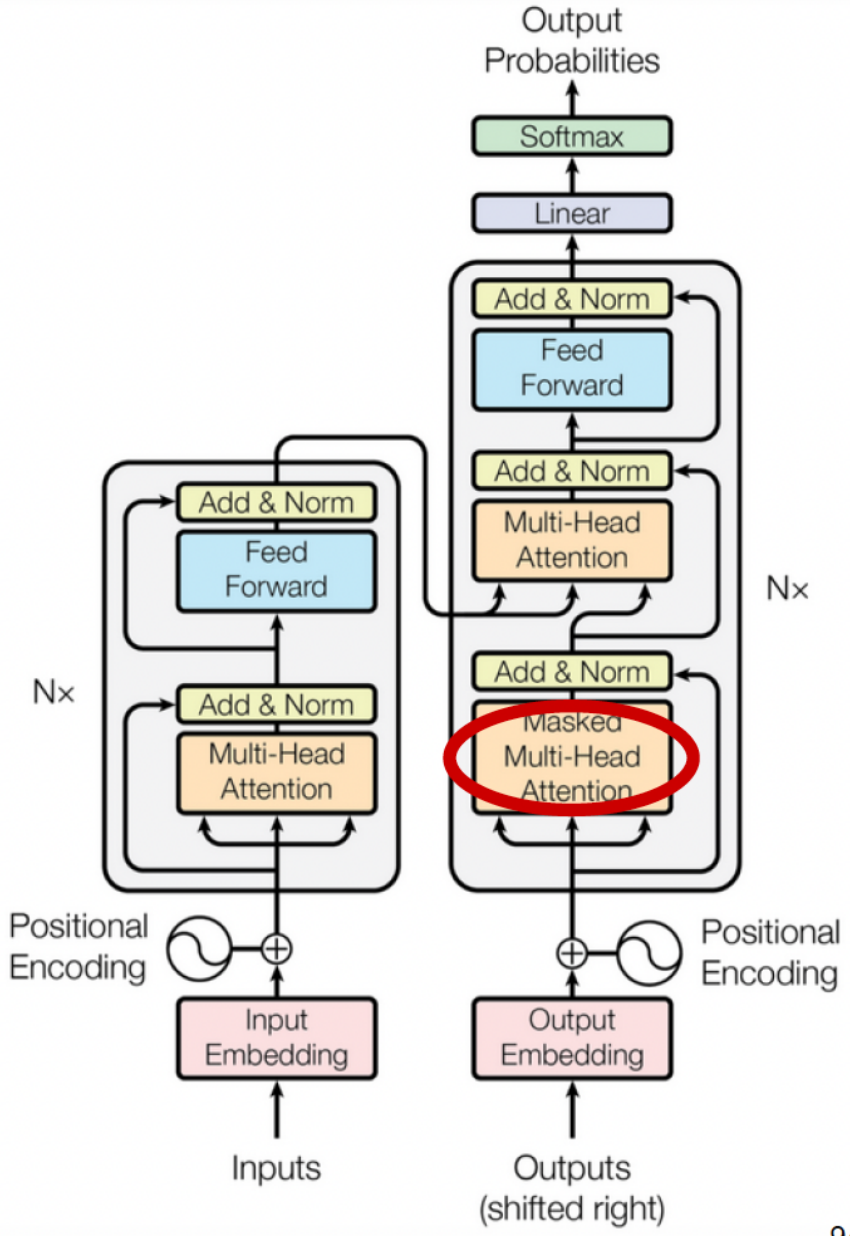
Everything is pretty much the same as in the encoder, with two twists:

- The decoder cannot see the future! Use "causal" masking
- The decoder should attend to itself (self-attention), but also to the encoder states (contextual attention).

The Decoder


encoder self-attn

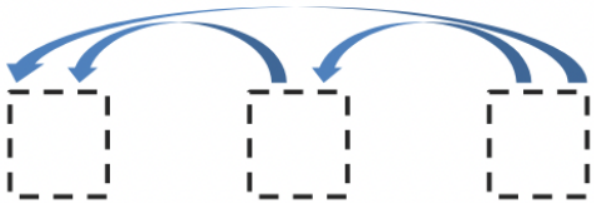

decoder self-attn (masked)


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

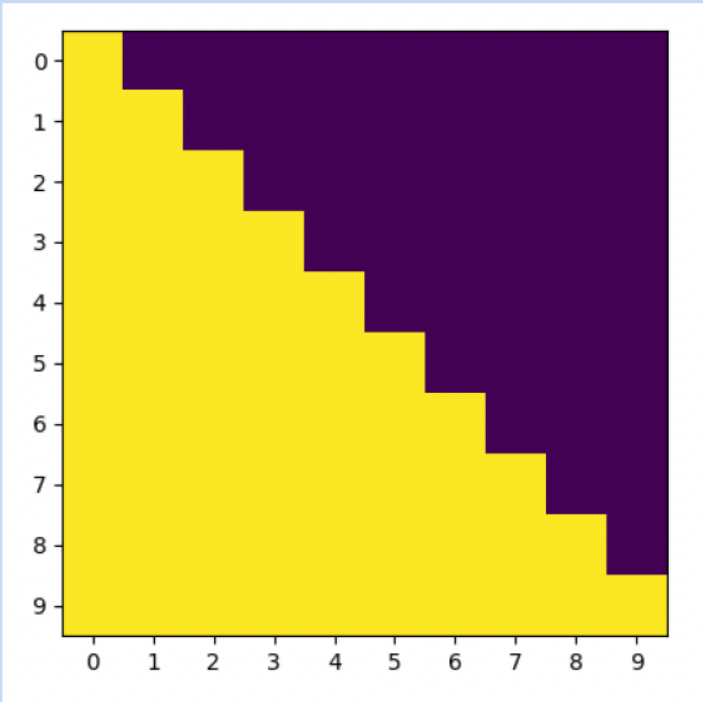
The Decoder

encoder self-attn

decoder self-attn (masked)



- Mask subsequent positions (before softmax)



- In PyTorch

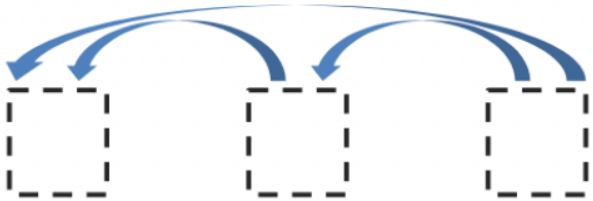
```
scores.masked_fill_(~mask, float('-inf'))
```

95

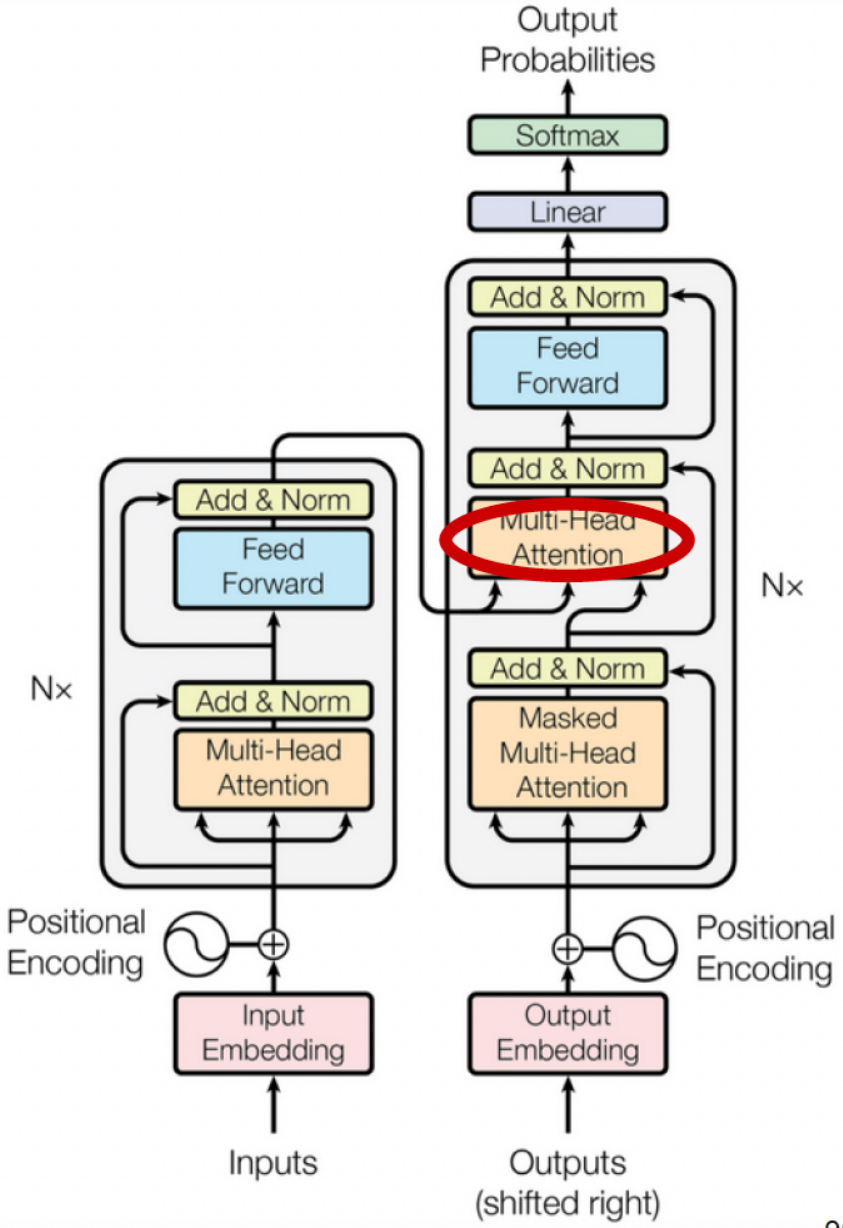
The Decoder

encoder self-attn

decoder self-attn (masked)

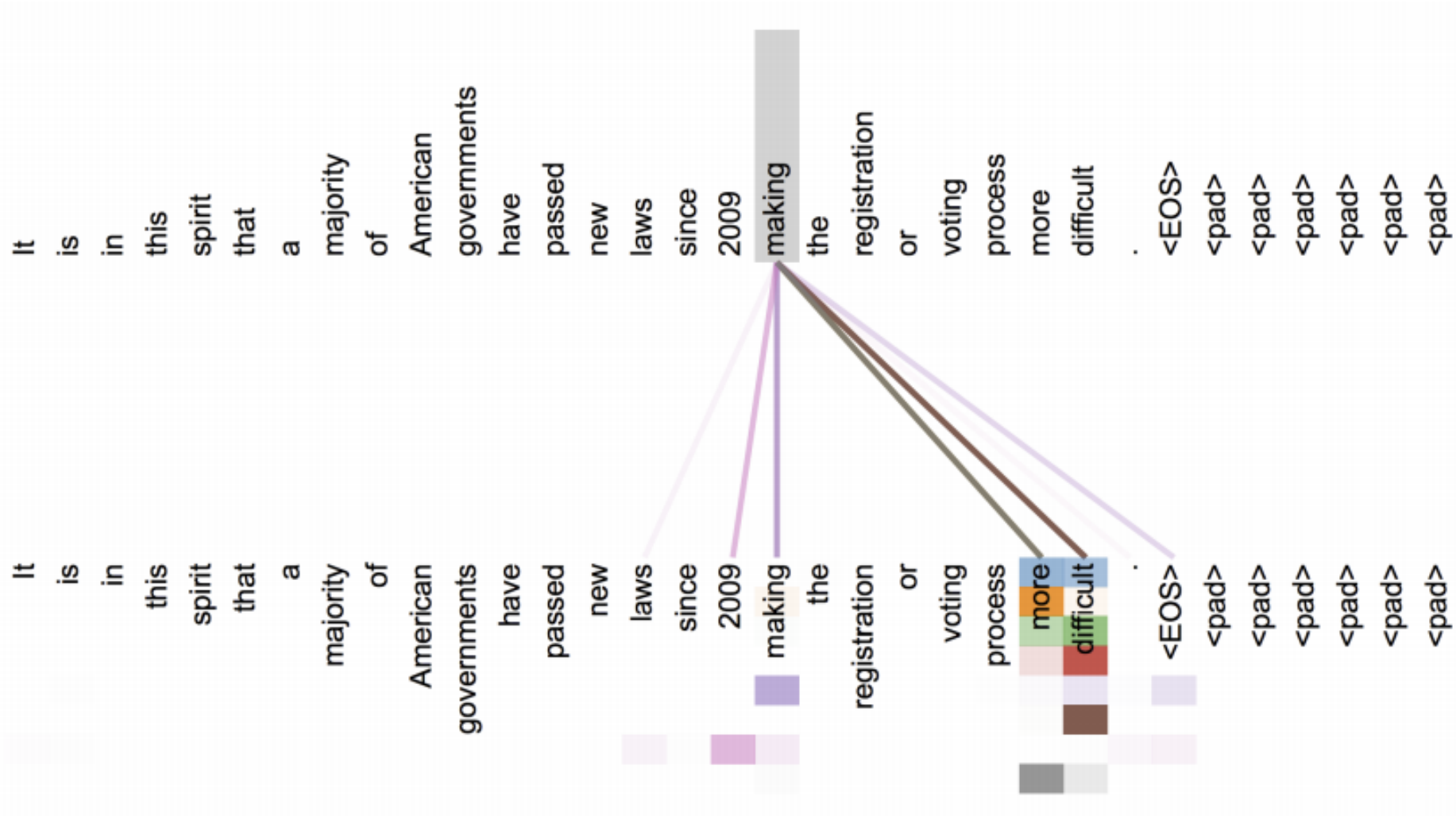


context attention

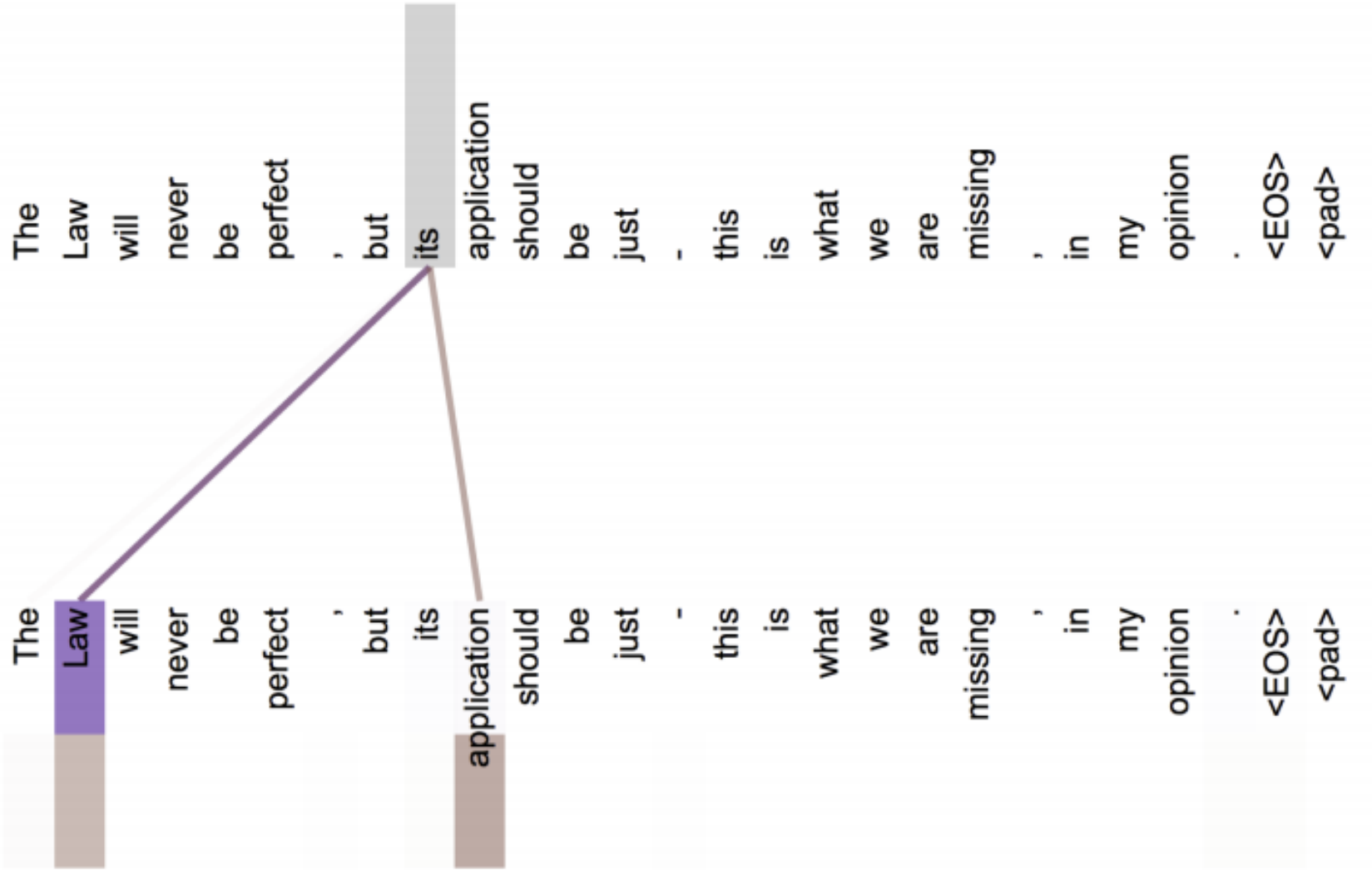


96

Attention Visualization Layer 5



Implicit Anaphora Resolution



Computational Cost

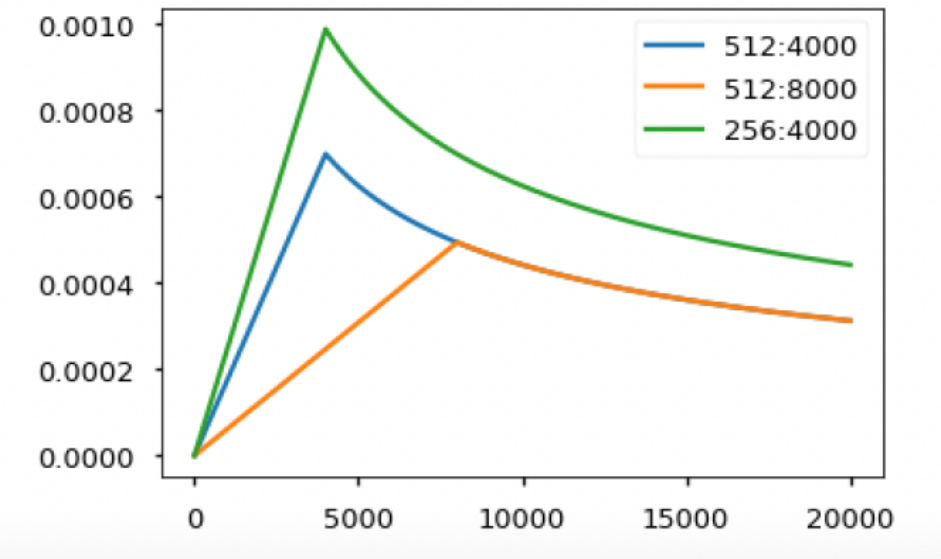
Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

n = seq. length d = hidden dim k = kernel size

- Faster to train (due to self-attention parallelization)
- More expensive to decode
- Scale quadratically with respect to sequence length (problematic for long sequences).

Other Tricks

- Label smoothing
- Dropout at every layer before residuals
- Beam search with length penalty
- Adam optimizer with learning-rate decay



Overall, transformers are harder to optimize than RNN seq2seq models

They don't work out of the box: hyperparameter tuning is very important.

Transformer Results

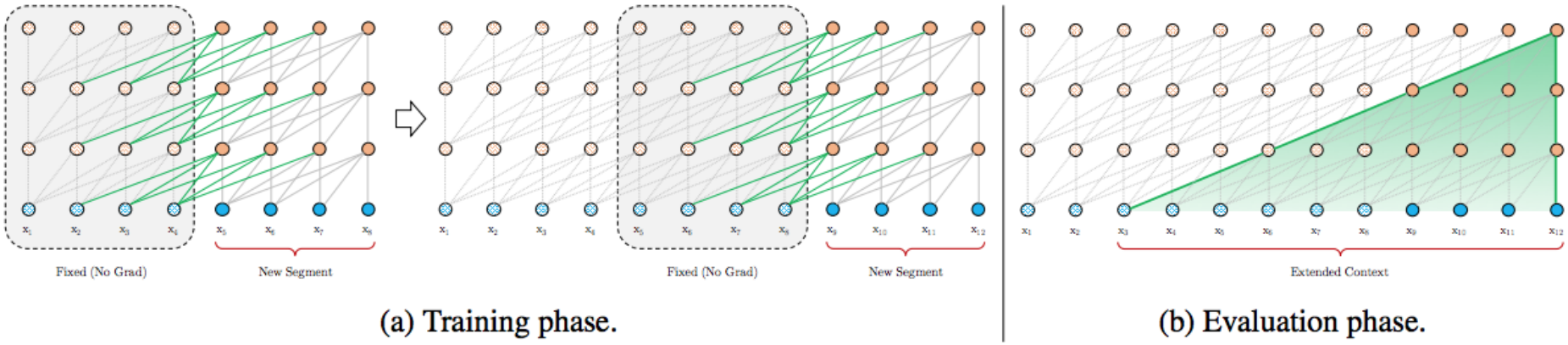
Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

(Vaswani et al., 2017)'s "Attention Is All You Need"

TransformerXL

Big transformers can look at larger contexts.

TransformerXL: enables going beyond a fixed length without disrupting temporal coherence:



(a) Training phase. (b) Evaluation phase.

(Dai et al., 2019)

Conclusions

- RNN-based seq2seq models require sequential computation and have difficulties with long range dependencies
- Attention mechanisms allow focusing on different parts of the input
- Encoders/decoders can be RNNs, CNNs, or self-attention layers
- Transformers are the current state of the art for many tasks in NLP and vision
- Other applications: speech recognition, image captioning, etc.
- Next: pretrained models and transfer learning (BERT, GPT-2, GPT-3, etc.)

Outline

Roadmap

Large pretrained models (BERT, GPT, etc.) and how to use them for downstream tasks.

- Contextualized representations
- Self-supervised learning
- Pretraining and finetuning
- Adapters and prompting.

Outline

From Static to Contextualized Word Embeddings

- In the representation learning lecture, we saw how to obtain word representations (embeddings) (e.g. word2vec, GloVe)
  - Each word in the vocabulary is represented by a vector, regardless of its context (a static embedding)

From Static to Contextualized Word Embeddings

- In the representation learning lecture, we saw how to obtain word representations (embeddings) (e.g. word2vec, GloVe)
  - Each word in the vocabulary is represented by a vector, regardless of its context (a static embedding)

Questions:

- Can we do word embeddings for multiple languages in the same space?
- How to capture polysemy (e.g. "bear" vs "bear"; "flies" vs "flies")?
- Can we compute embeddings on-the-fly, depending on the context?

From Static to Contextualized Word Embeddings

- In the representation learning lecture, we saw how to obtain word representations (embeddings) (e.g. word2vec, GloVe)
  - Each word in the vocabulary is represented by a vector, regardless of its context (a static embedding)

Questions:

- Can we do word embeddings for multiple languages in the same space?

- How to capture polysemy (e.g. "bear" vs "bear"; "flies" vs "flies")?

- Can we compute embeddings on-the-fly, depending on the context?
  - Using transformers we can obtain contextual embeddings.

Contextual Embeddings

- Words can have different meanings, depending on which context they appear in.
- A model called ELMo learned context-dependent embeddings and achieved impressive results on 6 NLP downstream tasks (Peters et al., 2018).
- ELMo was the first of a series of models named after Sesame Street characters.

Embeddings from Language Models (ELMo) (Peters et al., 2018)

Key idea:

- Pre-train a BiLSTM language model on a large dataset
- Save all the parameters at all layers, not only the embeddings
- Then, for your downstream task, tune a scalar parameter for each layer, and pass the entire sentence through this encoder.

Later several models have been proposed (BERT, GPT) with even more impressive performance.

Outline

Pretraining through Language Modeling

Recall the language modeling (LM) task:

- Model $p_\theta(y_t \mid y_{1:(t-1)})$, the probability distribution of words given their past contexts.
- There's lots of data for this! No *labels* are necessary, just raw text.
- This is called unsupervised pretraining or self-supervised learning.

Pretraining through language modeling:

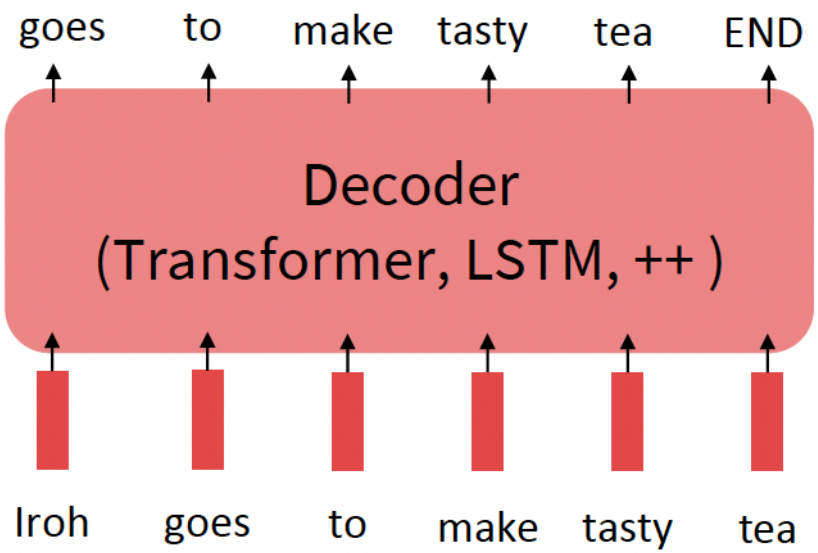
- Train a NN to perform language modeling on a large amount of text.
- Save the network parameters.

Pretraining and Fine-tuning

Pretraining can be very effective by serving as parameter initialization.

Step 1: Pretrain (e.g. on LM)

Lots of text; learn general things!

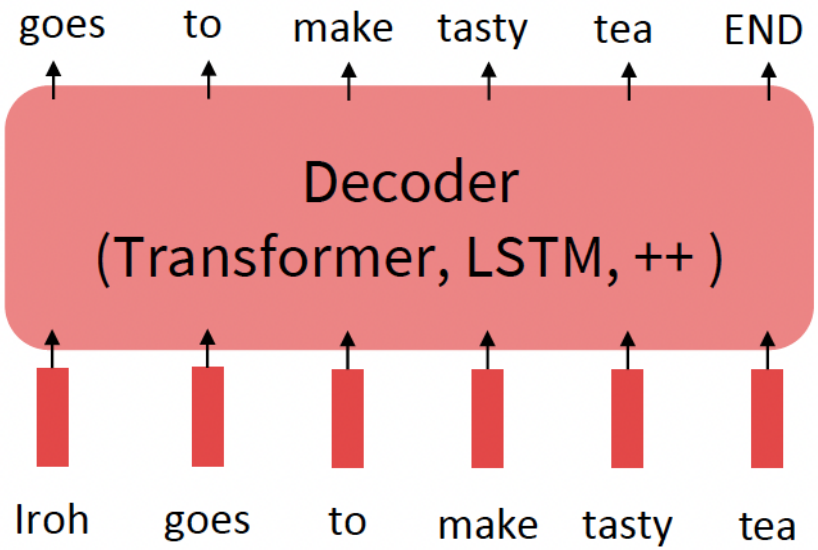


Pretraining and Fine-tuning

Pretraining can be very effective by serving as parameter initialization.

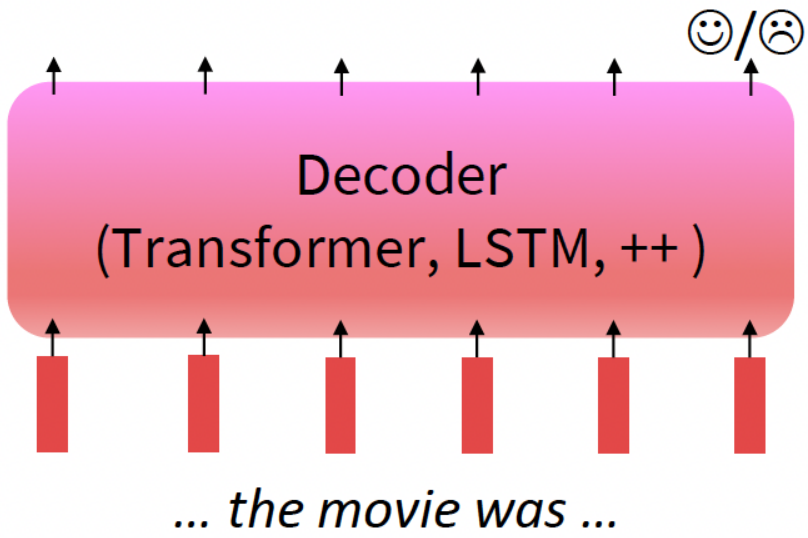
Step 1: Pretrain (e.g. on LM)

Lots of text; learn general things!



Step 2: Finetune on your task

Not many labels; adapt to the task!



Self-Supervised Learning

Pretraining on LM task is a form of self-supervised learning

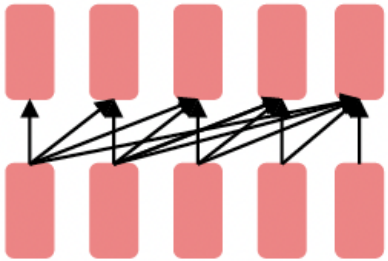
- Take raw (unlabeled) data, remove information and train a model to recover that information
- In the case of language modeling, the information removed is the next word; the model is trained to predict future words given the context
- Other strategies: *mask* words (later)
- This can be done with signals, images too, not just language
- For example, take images, obfuscate a region, and train a model to predict the missing region (image completion)

Why Does This Work?

Look at pretraining and fine-tuning from a "training NNs" perspective.

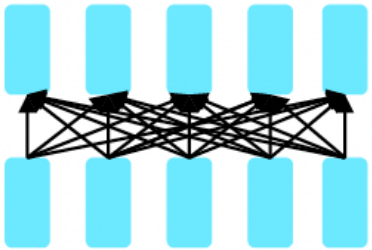
- Pretraining leads to parameters $\hat{\theta} \approx \arg\min_\theta L_{\text{pretrain}}(\theta)$
- Fine-tuning approximates $\arg\min_\theta L_{\text{finetune}}(\theta)$, starting at $\hat{\theta}$
- Pretraining helps as SGD stays (relatively) close to $\hat{\theta}$ during fine-tuning.
- Pretraining on large datasets exposes the model to many words and contexts not seen in the fine-tuning data (a form of weak supervision).
- Hopefully, the fine-tuning local minima near $\hat{\theta}$ tend to generalize well!

Three Architectures for Pretraining



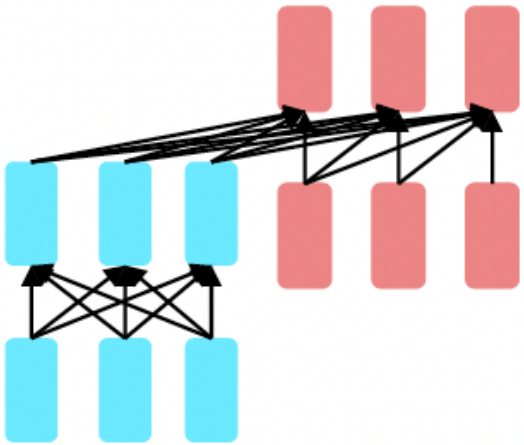
Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words



Encoders

- Bidirectional context $\Rightarrow$ can condition on future!
- Wait, how do we pretrain them?



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

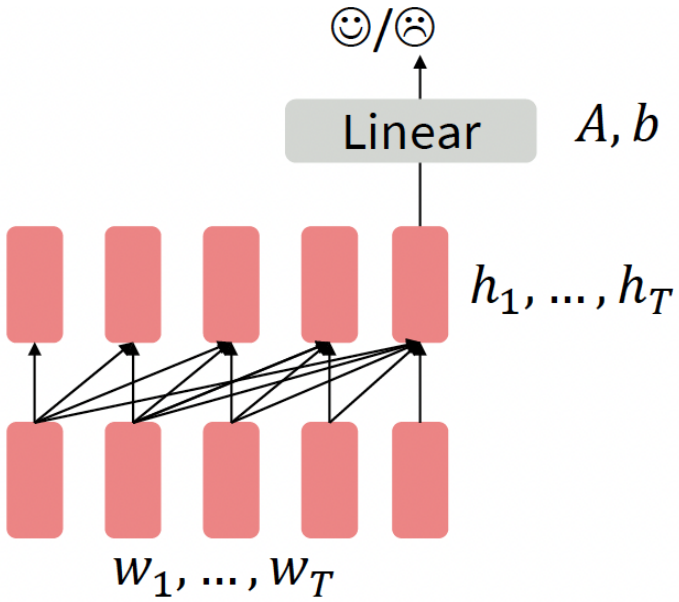
Pretrained Decoders

- When using LM-pretrained decoders, we can ignore that they were trained to model $p_\theta(x_t \mid x_{1:(t-1)})$

- Fine-tuning by training a classifier on the last hidden state.

$$\boldsymbol{h}_1, \ldots, \boldsymbol{h}_L = \text{Decoder}(x_1, \ldots, x_L)$$
$$\boldsymbol{y} = \textbf{softmax}(\boldsymbol{A}\boldsymbol{h}_L + \boldsymbol{b})$$

where $\boldsymbol{A}$ and $\boldsymbol{b}$ are randomly initialized and learned by the downstream task.



Pretrained Decoders

Two common choices for fine-tuning:

- *Freeze* the pretrained model and train only $\boldsymbol{A}$ and $\boldsymbol{b}$
- Finetune everything, backpropagate gradients through the whole network.

Pretrained decoders are particularly useful for generation tasks:

- Summarization
- Machine Translation
- Dialogue
- etc.

The family of GPT models developed by OpenAI is an example.

Generative Pretrained Transformer (GPT) (Radford et al., 2018)

- 2018's GPT was a big success in pretraining a decoder!
- Transformer decoder with 12 layers.
- 768-D hidden states, 3072-D feed forward hidden layers.
- Byte pair encoding with 40,000 merges (vocabulary size)
- Trained on BooksCorpus over 7000 books.
- Contains long spans of contiguous text, for learning long distance dependencies.

Generative Pretrained Transformer (GPT) (Radford et al., 2018)

How do we format inputs to our decoder for finetuning tasks?

Radford et al. (2018) evaluates on natural language inference (NLI):

- Label pairs of sentences as entailment/contradiction/neutral

  Premise: "The man is in the doorway"
  Hypothesis: "The person is near the door" $\Rightarrow$ *entailment*

The input is formatted to become a sequence of tokens for the decoder:

[START] The man is in the doorway [DELIM] The person is near the door [EXTRACT]

A linear classifier is applied to the representation of [EXTRACT]

Generative Pretrained Transformer (GPT) (Radford et al., 2018)

Results on various natural language inference (NLI) tasks:

Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	89.3	-	-	-
CAFE [58] (5x)	80.2	79.0	89.3	-	-	-
Stochastic Answer Network [35] (3x)	80.6	80.1	-	-	-	-
CAFE [58]	78.7	77.9	88.5	83.3		
GenSen [64]	71.4	71.3	-	-	82.3	59.2
Multi-task BiLSTM + Attn [64]	72.2	72.1	-	-	82.1	**61.7**
Finetuned Transformer LM (ours)	**82.1**	**81.4**	**89.9**	**88.3**	**88.1**	56.0

GPT-2 (Radford et al., 2019)

Another iteration of GPT, with larger models and trained on much more data (40GB of Internet text)

Stronger in language generation tasks.

- A transformer-based language model with 1.5 billion parameters, trained on 8 million web pages
- Training task: predict the next word, given all of the previous words
- Test time: top-k truncated random sampling
- Lots of hype!

GPT-2's Famous Example (Radford et al., 2019)

Given a prompt, perform story completion:

Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

GPT-2 Generated Text

A machine learning school was held in Bangkok early August.

`https://talktotransformer.com/`

GPT-2 Generated Text

A machine learning school was held in Bangkok early August. I had originally expected a much more "procedural and algorithmic" approach as well, but was ple asantly surprised by the level of expertise of the students. These were both very experienced mathematicians, but also had an appreciation for machine learning techniques. These are the same school that created and operates the very popular Algorithmia blog , so they aren't exactly new to me, and are well known to other Algorithmia grads at the University of Central Europe.

`https://talktotransformer.com/`

GPT-3 (Brown et al., 2020)

Keep scaling up... Even larger models pretrained on more data:

- 175 billion parameters (!!!)
- Trained on 500 billion words
- Took $\sim 3 \times 10^{23}$ FLOPS to train (on a standard GPU, it would cost \$4.6M and it would require 355 years to train such a model)
- Introduces prompting as an alternative to fine-tuning (later)
- Demonstrates few-shot learning capabilities (learning new tasks on the fly from very few examples)

GPT-3 Generated Text

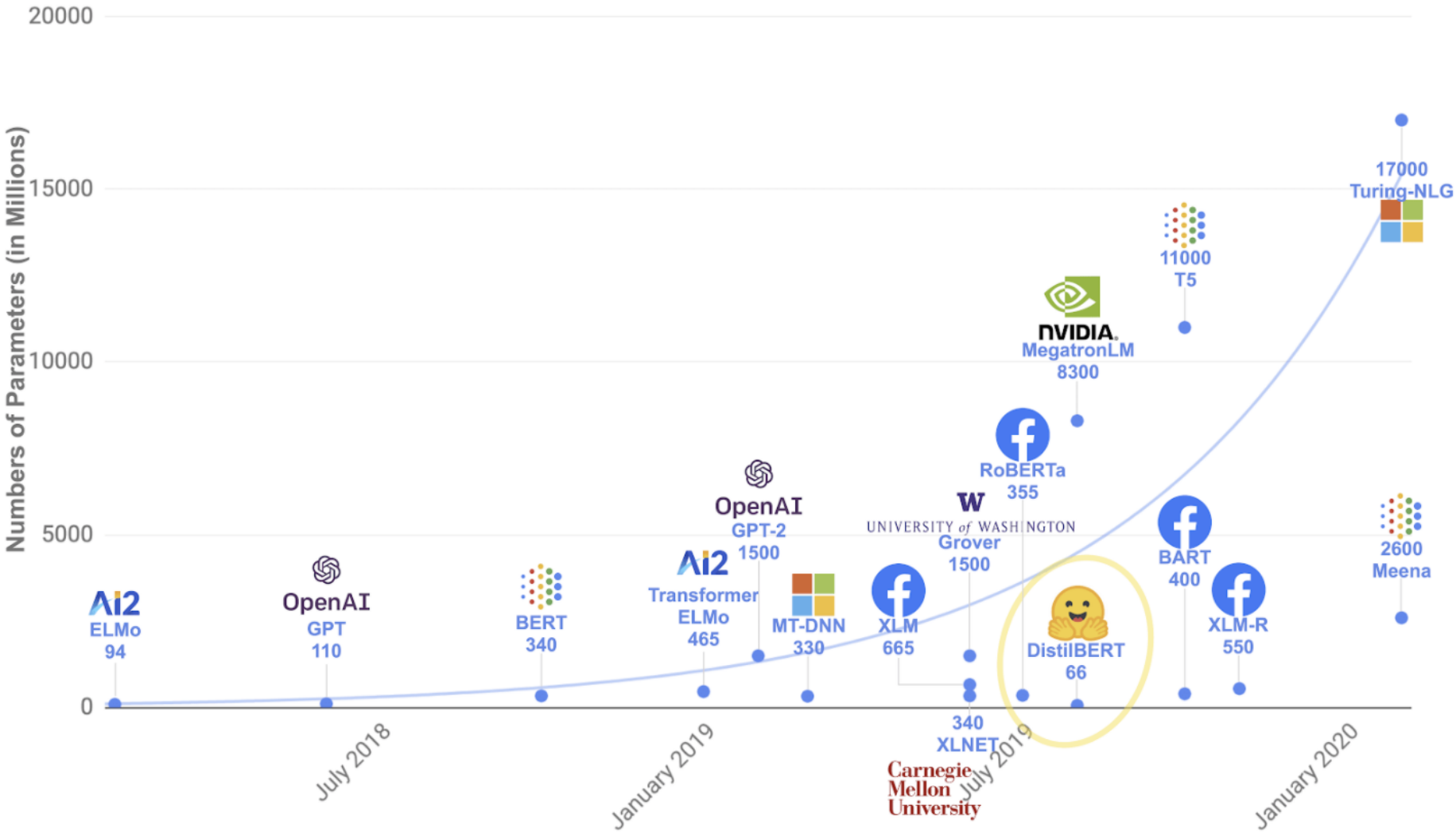
I am teaching a deep learning course in Lisbon. Do you think this is a good idea?

GPT-3 Generated Text

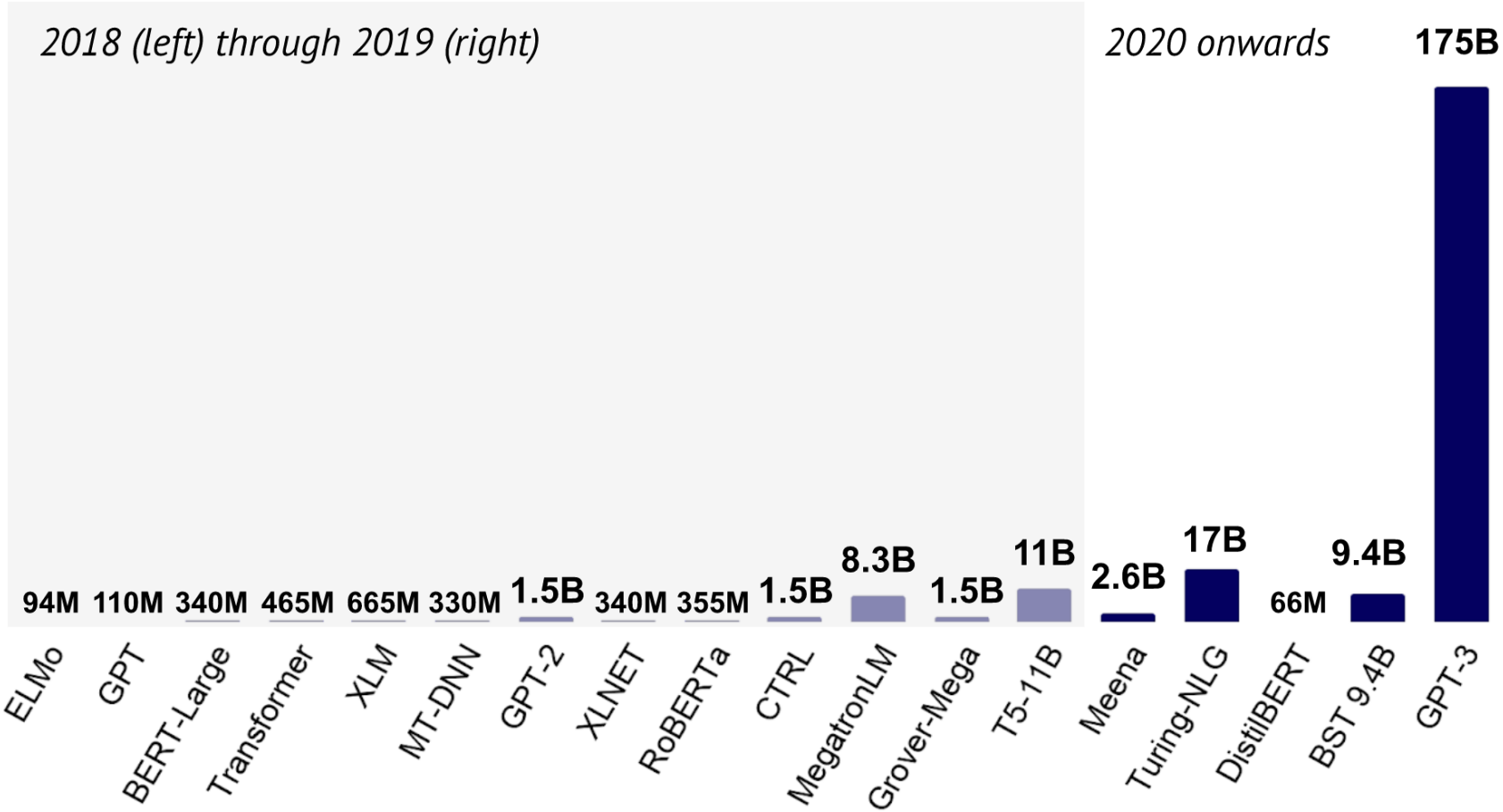
I am teaching a deep learning course in Lisbon. Do you think this is a good idea?

It is always a good idea to share knowledge and educate others. If you are well-prepared and have a strong understanding of deep learning, then teaching a course on the subject could be a rewarding experience for both you and your students. Lisbon is a vibrant and culturally rich city, so it could be a great place to teach a course on deep learning. It is important to consider the needs and backgrounds of your students and to structure the course in a way that is engaging and accessible to them.

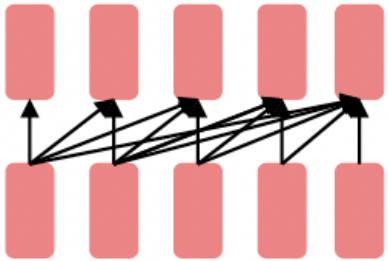
Large Pretrained Models



Large Pretrained Models

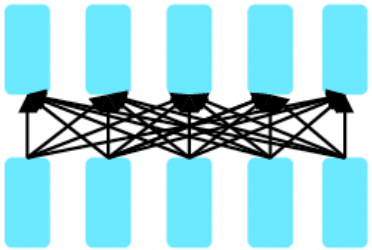


Three Architectures for Pretraining



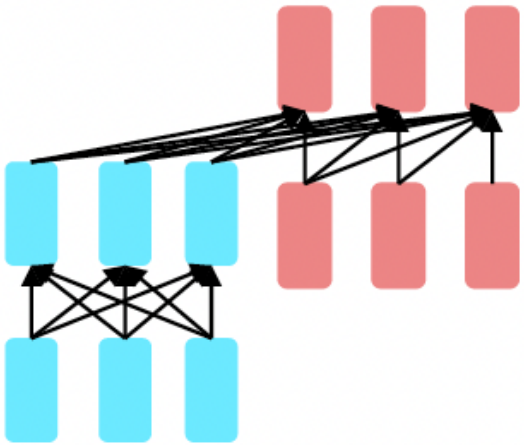
Decoders ✓

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words



Encoders

- Bidirectional context ⇒ can condition on future!
- Wait, how do we pretrain them?



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

Pretrained Encoders

So far, we've looked at language model pretraining. But encoders get bidirectional context, so we can't do language modeling!

So, what pretraining objective to use?

Pretrained Encoders

So far, we've looked at language model pretraining. But encoders get bidirectional context, so we can't do language modeling!

So, what pretraining objective to use? Masked language modeling.

Masked Language Modeling

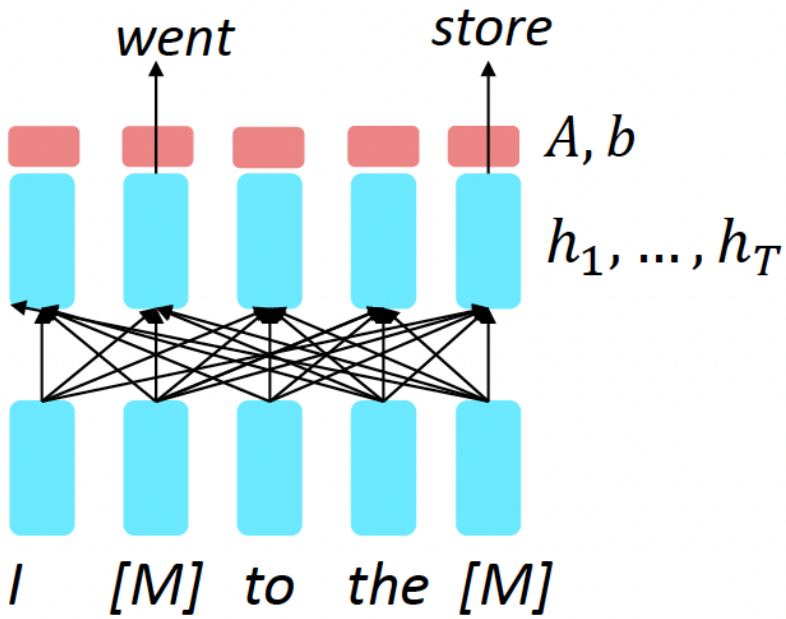
- Idea: replace a fraction of words in the input with a special [MASK] token; predict these words.

$$\boldsymbol{h}_1, \ldots, \boldsymbol{h}_L = \text{Encoder}(x_1, \ldots, x_L)$$
$$\boldsymbol{y}_i = \textbf{softmax}(\boldsymbol{A}\boldsymbol{h}_i + \boldsymbol{b}).$$

- Loss terms only from masked word. If $\tilde{x}$ is the masked version of x, we're learning $p_\theta(x|\tilde{x})$

- Analogous to a denoising auto-encoder.

t



(Devlin et al., 2018)

Example: BERT (Devlin et al., 2018)

"Bidirectional Encoder Representations from Tranformers"

- Randomly mask 15% of the input words and train a transformer to recover them from the context
- Both left and right context, used simultaneously!
- In doing so, learn contextualized word representations
- Can use this as a pre-trained model and fine-tune it to any downstream task
- Extremely effective! Achieved SOTA on 11 NLP tasks (7.7% percentage point improvement on GLUE score).

Details about BERT (Devlin et al., 2018)

Two models were released:

- BERT base: 12 layers, 768 dim hidden states, 12 attention heads, 110 million params.
- BERT large: 24 layers, 1024 dim hidden states, 16 attention heads, 340 million params.

Trained on:

- BooksCorpus (800 million words)
- English Wikipedia (2.5 billion words)

Pretraining is expensive and impractical on a single GPU.

- BERT was pretrained with 64 TPU chips for a total of 4 days.

Fine-tuning is practical and common on a single GPU:

- "Pretrain once, finetune many times."

Fine-Tuning BERT (Devlin et al., 2018)

BERT became very popular and versatile; finetuning BERT led to new state of the art results on a broad range of NLP tasks:

- Paraphrase detection (QQP, MRPC)
- Natural language inference (QNLI, RTE)
- Sentiment analysis (SST-2)
- Grammatical correctness (CoLA)
- Semantic textual similarity (STS-B)

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Other variants of BERT

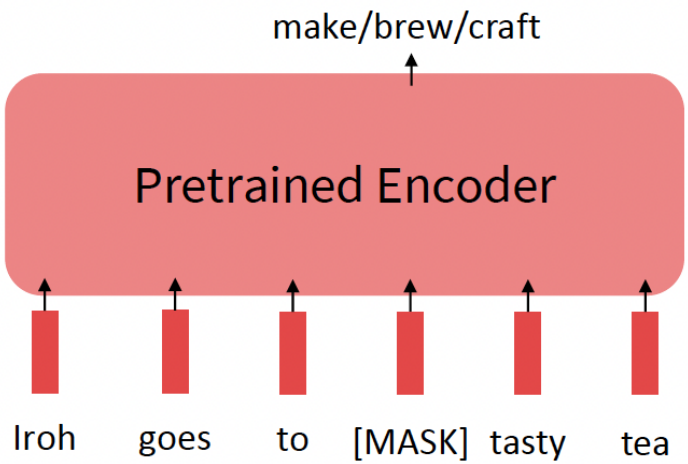
- M-BERT (same as BERT but multilingual, not English-specific)
  - Effective in many cross-lingual tasks
- RoBERTA (similar to BERT, but trained on more data and removing next-sentence prediction)
- XLM-RoBERTA (multilingual version)
- SpanBERT
- ...

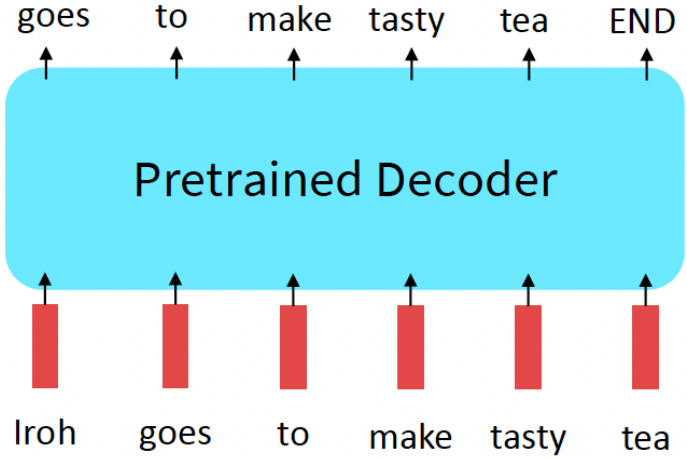
Limitations of pretrained encoders

Why not use pretrained encoders for everything?

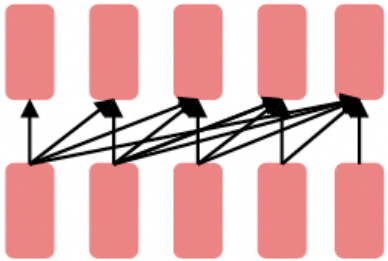
Pretrained decoders (causal LM) vs pretrained encoders (masked LM):

- If the task involves generating sequences, use a pretrained decoder (BERT and other pretrained encoders don't naturally lead to nice autoregressive generation methods.)
- If the task involves classification or sequence tagging, use a pretrained encoder; usually benefits from bidirectionality.



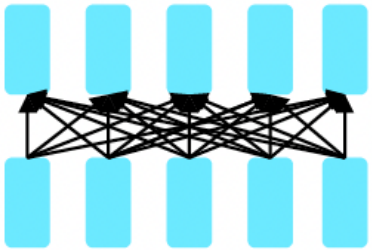


Three Architectures for Pretraining



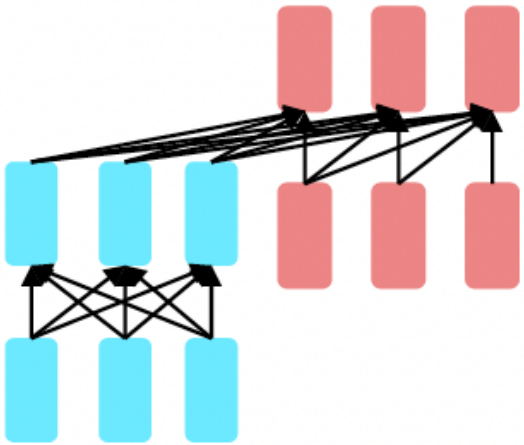
Decoders ✓

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words



Encoders ✓

- Bidirectional context ⇒ can condition on future!
- Wait, how do we pretrain them?



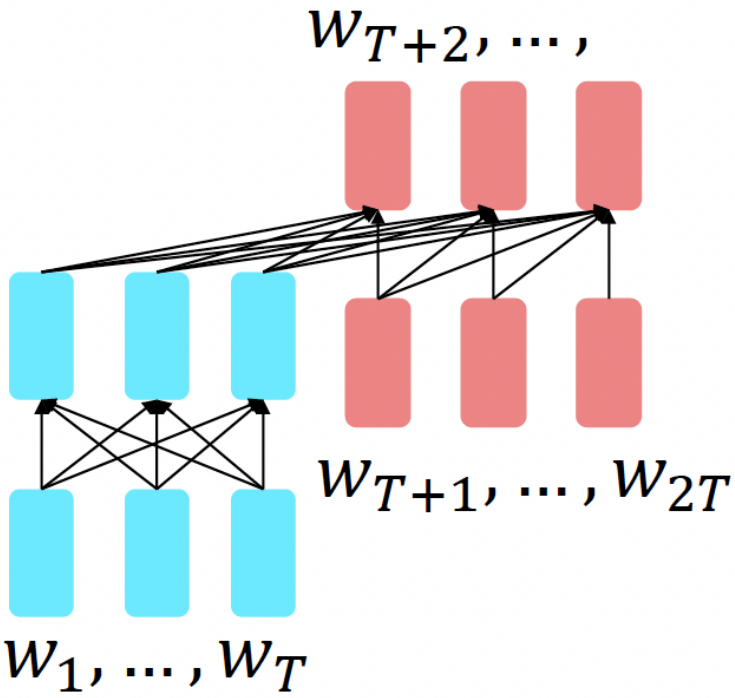
Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

Pretrained Encoder-Decoder

- For encoder-decoders, we can do something like LM, but where a prefix of every input is provided to the encoder and is not predicted.

$$\boldsymbol{h}_1, \ldots, \boldsymbol{h}_T = \text{Encoder}(x_1, \ldots, x_T)$$
$$\boldsymbol{h}_{T+1}, \ldots, \boldsymbol{h}_{2T} = \text{Decoder}(x_{T+1}, \ldots, x_{2T})$$
$$\boldsymbol{y}_i = \textbf{softmax}(\boldsymbol{A}\boldsymbol{h}_i + \boldsymbol{b}),\ i > T$$



(Raffel et al., 2020)

- The encoder benefits from bidirectional context
- The decoder is used to train the whole model through LM.

T5 (Raffel et al., 2020)

Use span corruption as an auxiliary task:

- Replace different length spans from the input with unique placeholders; decode out the spans that were removed!

- This is implemented in text preprocessing: it's still an objective that looks like LM at the decoder side.

Inputs: Thank you ⟨X⟩ me to your party ⟨Y⟩ week.

Targets: ⟨X⟩ for inviting ⟨Y⟩ last ⟨Z⟩

T5 (Raffel et al., 2020)

Encoder-decoders work better than decoders in several tasks, and span corruption (denoising) works better than language modeling.

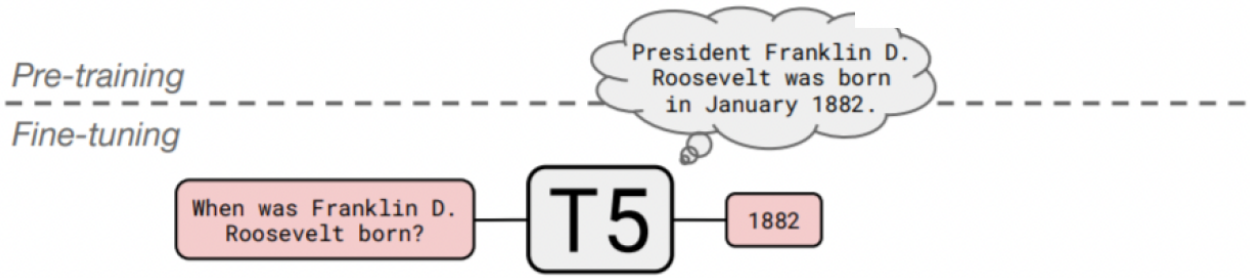
Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39
Encoder-decoder	LM	$2P$	M	79.56	18.59	76.02	64.29	26.27	39.17	26.86
Enc-dec, shared	LM	P	M	79.60	18.13	76.35	63.50	26.62	39.17	27.05
Enc-dec, 6 layers	LM	P	$M/2$	78.67	18.26	75.32	64.06	26.13	38.42	26.89
Language model	LM	P	M	73.78	17.54	53.81	56.51	25.23	34.31	25.38
Prefix LM	LM	P	M	79.68	17.84	76.87	64.86	26.28	37.51	26.76

(Raffel et al., 2020)

T5 (Raffel et al., 2020)

T5 can be fine-tuned to answer a wide range of questions, retrieving factual knowledge from its parameters!

Natural Questions (NQ), WebQuestions (WQ), TriviaQA (TQA)



	NQ	WQ	TQA dev	TQA test	
Karpukhin et al. (2020)	**41.5**	42.4	**57.9**	–	
T5.1.1-Base	25.7	28.2	24.2	30.6	**220 million params**
T5.1.1-Large	27.3	29.5	28.5	37.2	**770 million params**
T5.1.1-XL	29.5	32.4	36.0	45.1	**3 billion params**
T5.1.1-XXL	32.8	35.6	42.9	52.5	**11 billion params**
T5.1.1-XXL + SSM	35.2	**42.8**	51.9	**61.6**	

Outline

Limitations of Fine-Tuning

So far, we have talked about pretraining and fine-tuning.

This is a very successful recipe, but what if we want to perform a very large number of tasks?

- Multilingual models supporting many languages (English, German, Spanish, Portuguese, plus a long tail of low-resource languages)
- Similar tasks but in different domains (news, conversational data, medical, legal, ...)
- Different tasks (e.g. generation, classification, tagging)

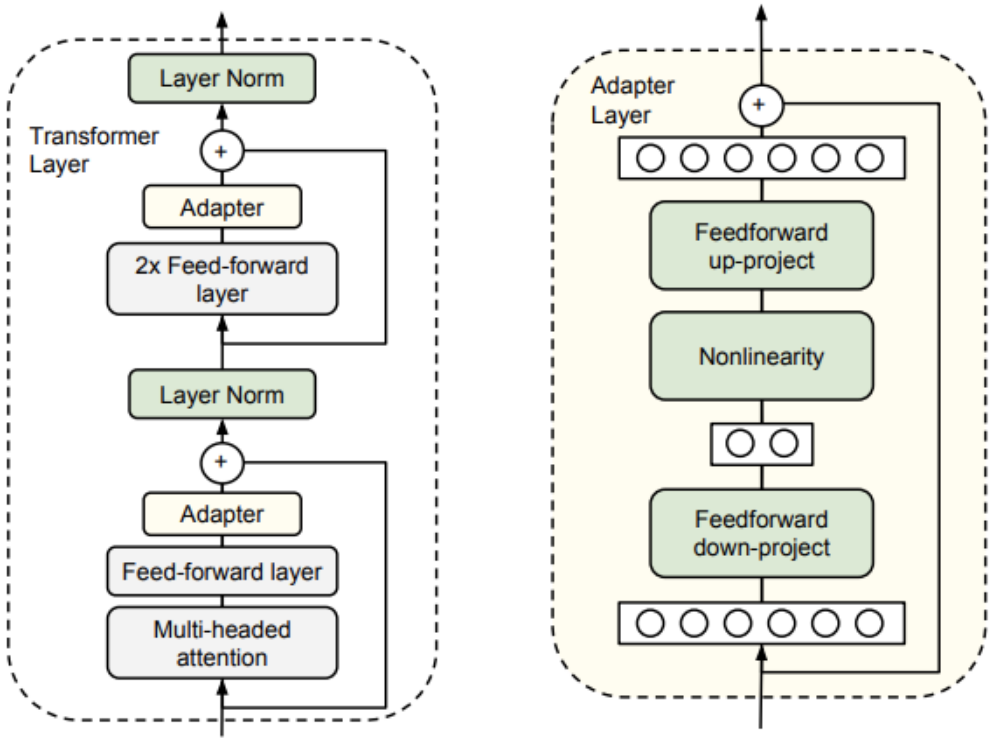
Fine-tuning a very large model to each of the tasks can be very expensive and requires a copy of the model for each task.

Can we do better?

Adapters (Houlsby et al., 2019)

- Alternative to fine-tuning language models on a downstream task
- Instead of fine-tuning the full model, a small set of task-specific parameters (adapter) is included in the model and updated during fine-tuning
- The rest of the model is kept fix
- Several advantages:
  - ✓ Much fewer parameters to fine-tune
  - ✓ Can share the same big pretrained model across tasks, and fine-tune only the task-specific adapters
  - ✓ Can also be used to create multilingual models (language adapters)

Adapters (Houlsby et al., 2019)

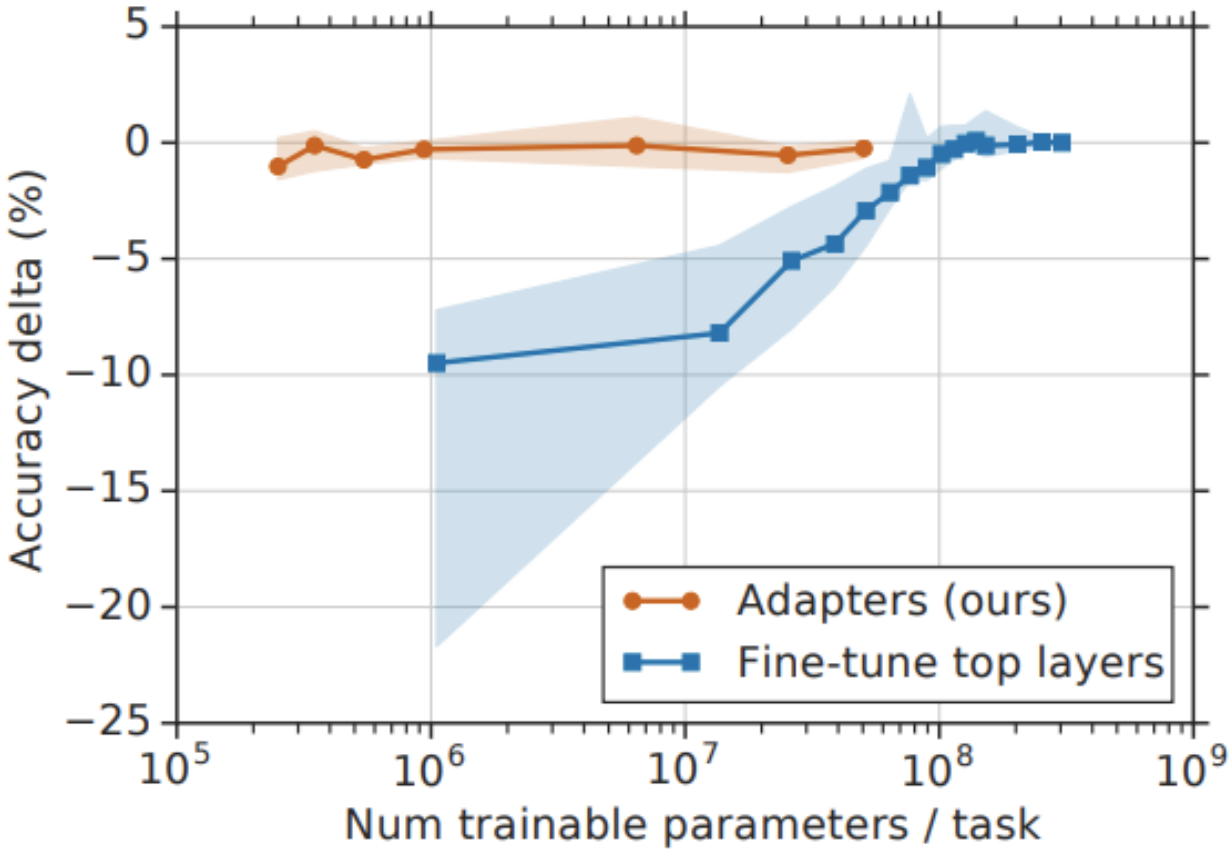


From Houlsby et al. (2019)

- Adapter layers interleaved in the other transformer layers
- In fine-tuning, only these adapter layers are updated
- The big pretrained model stays untouched

Adapters (Houlsby et al., 2019)

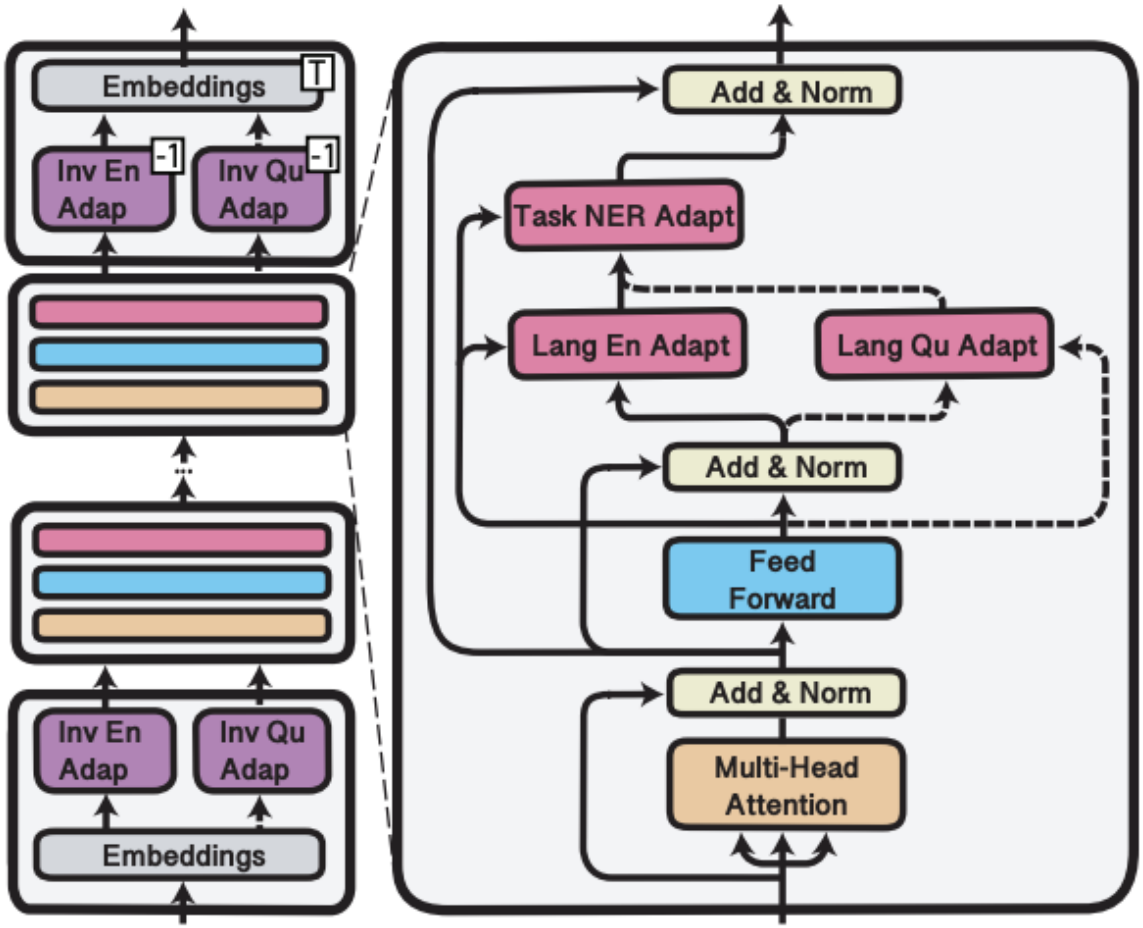
- They achieve high performance in downstream tasks with much fewer new parameters:



From Houlsby et al. (2019)

Task and Language Adapters (Pfeiffer et al., 2020)

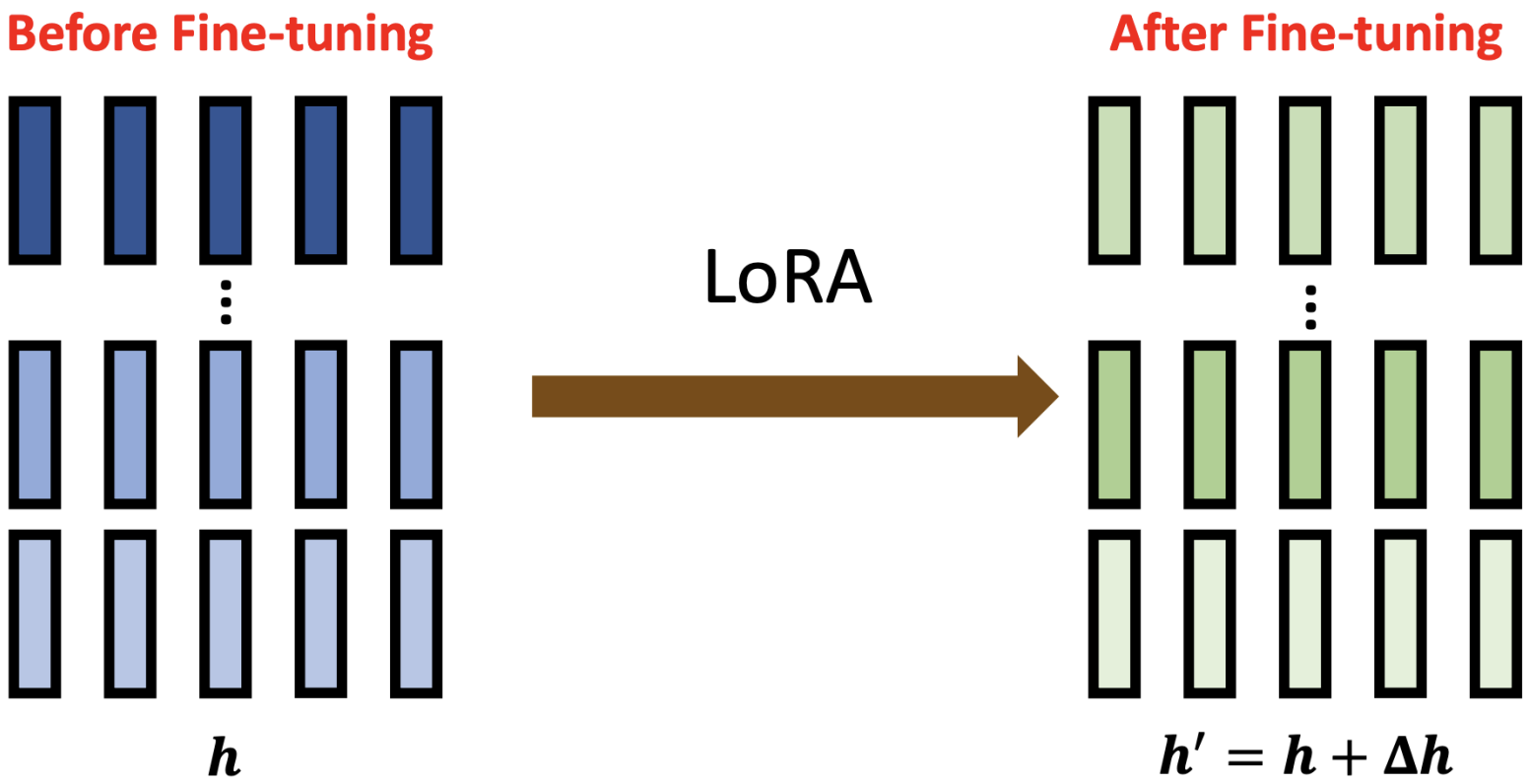
- Adapters can be used to adapt to new tasks and languages:



From Pfeiffer et al. (2020)

Low-Rank Adaptation (Hu et al., 2022)

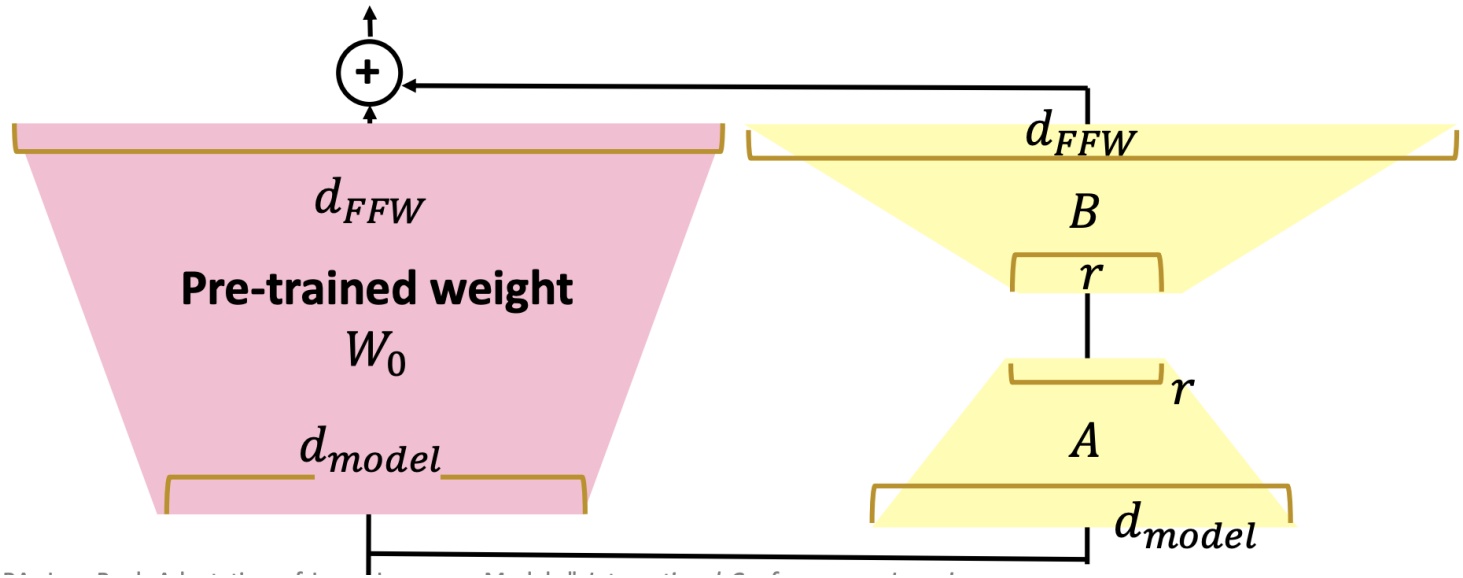
- Use special submodules to modify hidden representations!



Slide credits: Chiang, Chuang, Lee, 2022 AACL-IJCNLP Tutorial

Low-Rank Adaptation (Hu et al., 2022)

- **Low-Rank Adaptation** of Large Language Models
- Motivation: Downstream fine-tunings have low intrinsic dimension
- Weight after fine-tuning = W_0 (pre-trained weight) $+\Delta W$ (updates to the weight)
- Hypothesis: The updates to the weight (ΔW) also gave a low intrinsic rank
- Fine-tuned weight = $W_0 + \Delta W = W_0 + BA$, rank $r \ll \min(d_{FFW}, d_{model})$



Slide credits: Chiang, Chuang, Lee, 2022 AACL-IJCNLP Tutorial

Few-Shot Learning

- What if we want to solve a completely new task for which not enough data exists, not even for fine-tuning?
- Can we do it on-the-fly?
- This is called few-shot learning
- Powerful models such as GPT-3 can do this via prompting
- Leveraging the versatility of LMs is all we need!

What do Pretrained Language Models Learn?

- Instituto Superior Técnico is located in _____, Portugal.
- I put _____ fork down on the table.
- The woman walked across the street, checking for traffic over _____ shoulder.
- I went to the ocean to see the fish, turtles, seals, and _____.

- Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was _____.
- Iroh went into the kitchen to make some tea. Standing next to Iroh, Zuko pondered his destiny. Zuko left the _____.
- I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, _____

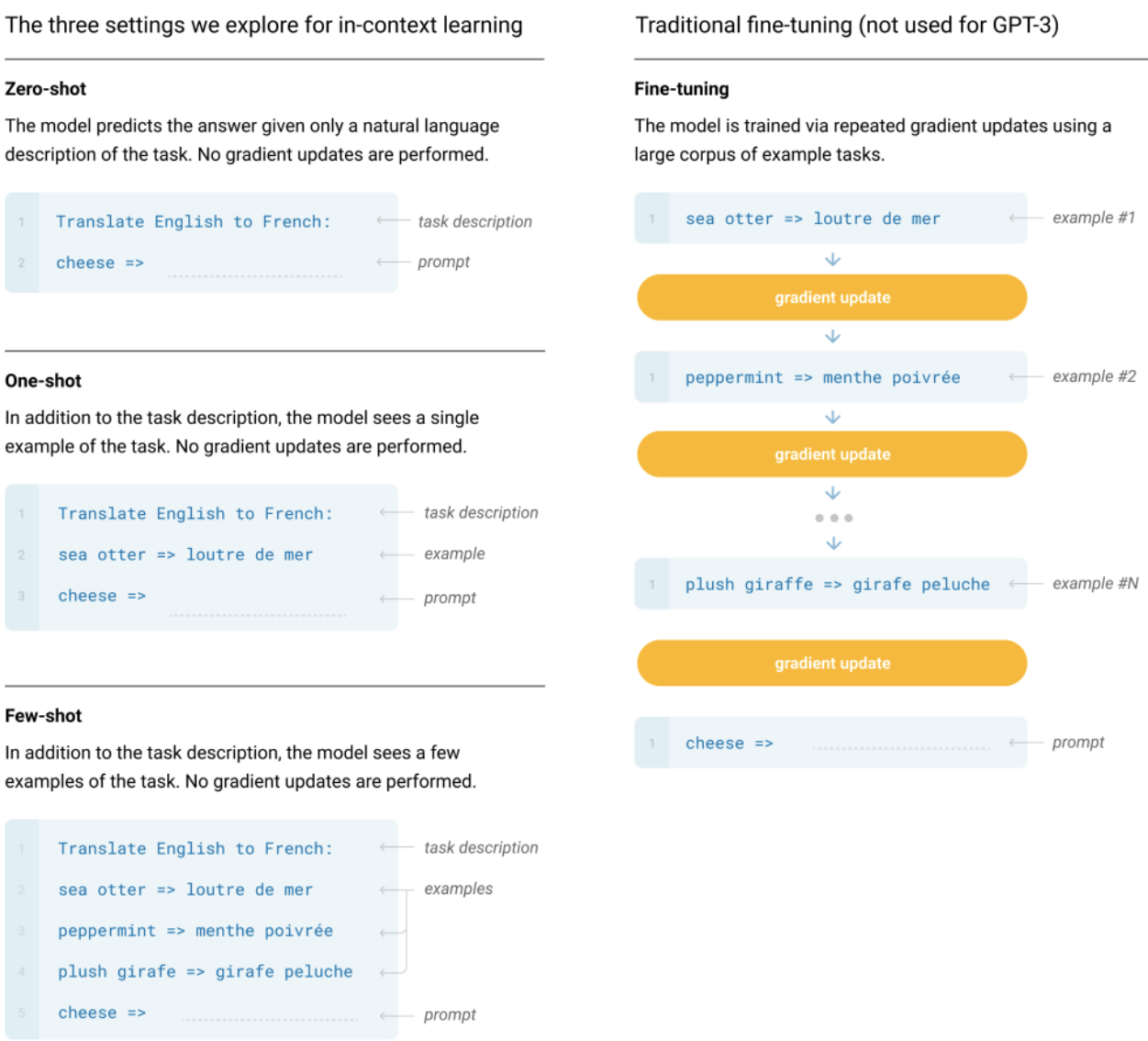
What do Pretrained Language Models Learn?

- Instituto Superior Técnico is located in _____, Portugal. [Trivia]
- I put _____ fork down on the table. [Syntax]
- The woman walked across the street, checking for traffic over _____ shoulder. [Coreference]
- I went to the ocean to see the fish, turtles, seals, and _____. [Lexical semantics]
- Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was _____. [Sentiment]
- Iroh went into the kitchen to make some tea. Standing next to Iroh, Zuko pondered his destiny. Zuko left the _____. [Complex reasoning]
- I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, _____ [Basic arithmetic]

Prompting and In-Context Learning

- Pretrained language models acquire a lot of factual knowledge!
- This suggests we can prompt them on-the-fly to solve new tasks.

Prompting and In-Context Learning (Brown et al., 2020)



Example: Grammar Correction (Brown et al., 2020)

Poor English input: I eated the purple berries.
Good English output: I ate the purple berries.
Poor English input: Thank you for picking me as your designer. I'd appreciate it.
Good English output: Thank you for choosing me as your designer. I appreciate it.
Poor English input: The mentioned changes have done. or I did the alteration that you requested. or I changed things you wanted and did the modifications.
Good English output: The requested changes have been made. or I made the alteration that you requested. or I changed things you wanted and made the modifications.
Poor English input: I'd be more than happy to work with you in another project.
Good English output: I'd be more than happy to work with you on another project.

Poor English input: Please provide me with a short brief of the design you're looking for and that'd be nice if you could share some examples or project you did before.
Good English output: Please provide me with a brief description of the design you're looking for and that would be nice if you could share some examples or projects you have done before.

Poor English input: The patient was died.
Good English output: The patient died.

Poor English input: We think that Leslie likes ourselves.
Good English output: We think that Leslie likes us.

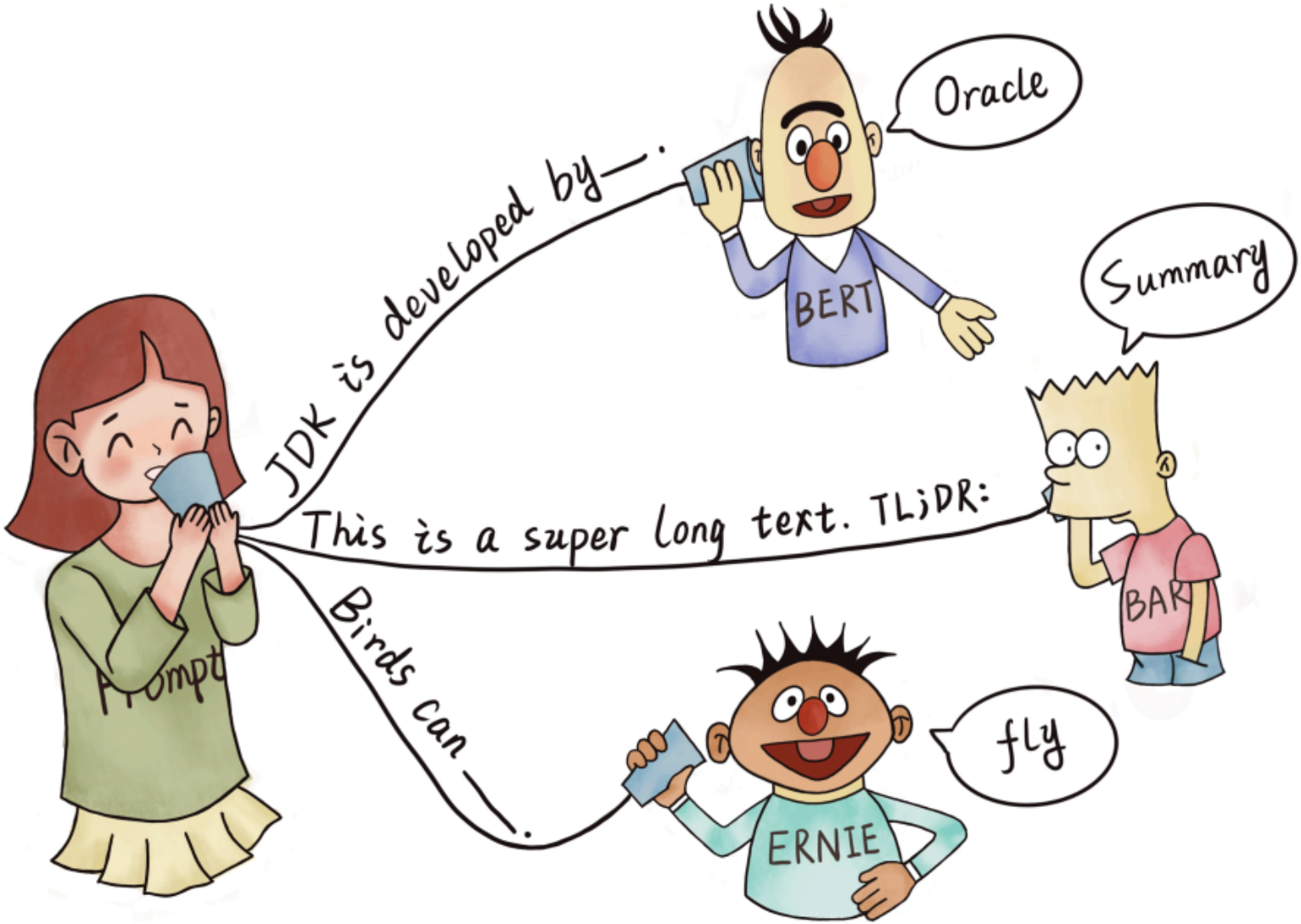
Poor English input: Janet broke Bill on the finger.
Good English output: Janet broke Bill's finger.

Poor English input: Mary arranged for, in St. Louis, John to rent a house cheap.
Good English output: Mary arranged for John to rent a house in St. Louis.

Poor English input: Today I have went to the store to to buys some many bottle of water.
Good English output: Today I went to the store to buy some bottles of water.

Poor English input: I have tried to hit ball with bat, but my swing is has miss.
Good English output: I tried to hit the ball with the bat, but my swing missed.

Prompting as a Way to Solve Many Tasks



From Liu et al. (2021)

Learning the Prompt

- Design a good prompt manually can be tedious
- Systems are very brittle and sensitive to the choice of prompt
- Combining multiple prompts and ensembling the answers increases robustness
- One exciting research direction is learning prompts automatically
- Two ways of doing this (both with some fine-tuning data):
  - Learn discrete prompts for each task (combinatorial problem)
  - Learn continuous prompts – by learning the word embeddings directly.
- More information in this survey: Liu et al. (2021)

Advances in 2022–25

2022 saw impressive novelties:

- GPT-3.5: a series of models trained on a blend of text and code
- ChatGPT: fine-tuned from GPT-3.5 using human supervision (reinforcement learning from human feedback – RLHF)

After that:

- Many other closed models: GPT4, Gemini, Claude, Cohere, ...
- Many more open models: LLaMA, Mistral, Gemma, Qwen, Tower, EuroLLM, ...
- Many of the recent models are now multimodal (e.g. language and vision).

Reinforcement Learning from Human Feedback (RLHF)

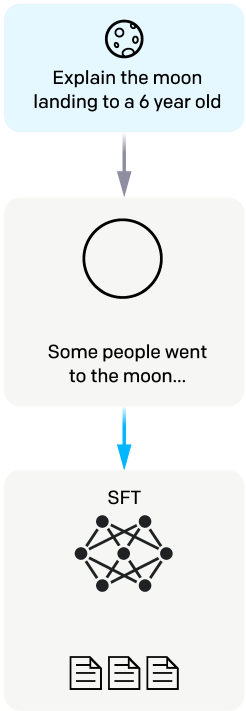
Step 1

Collect demonstration data, and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.



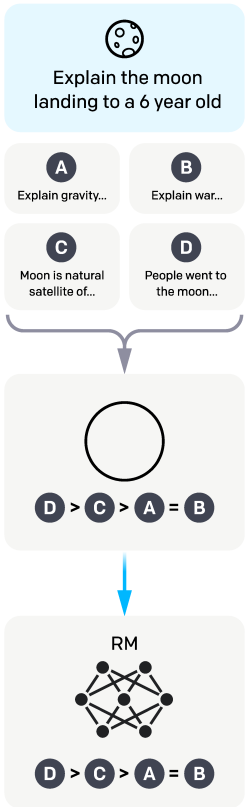
Step 2

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.



Step 3

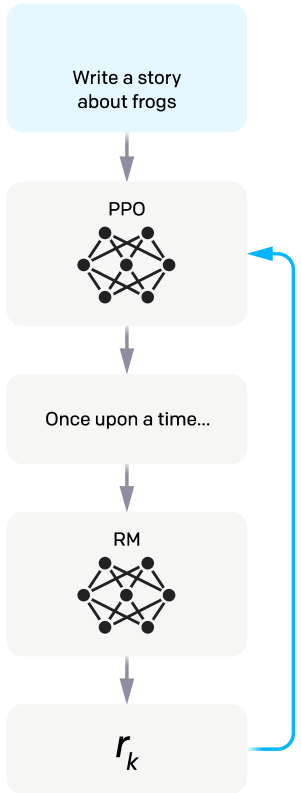
Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

The policy generates an output.

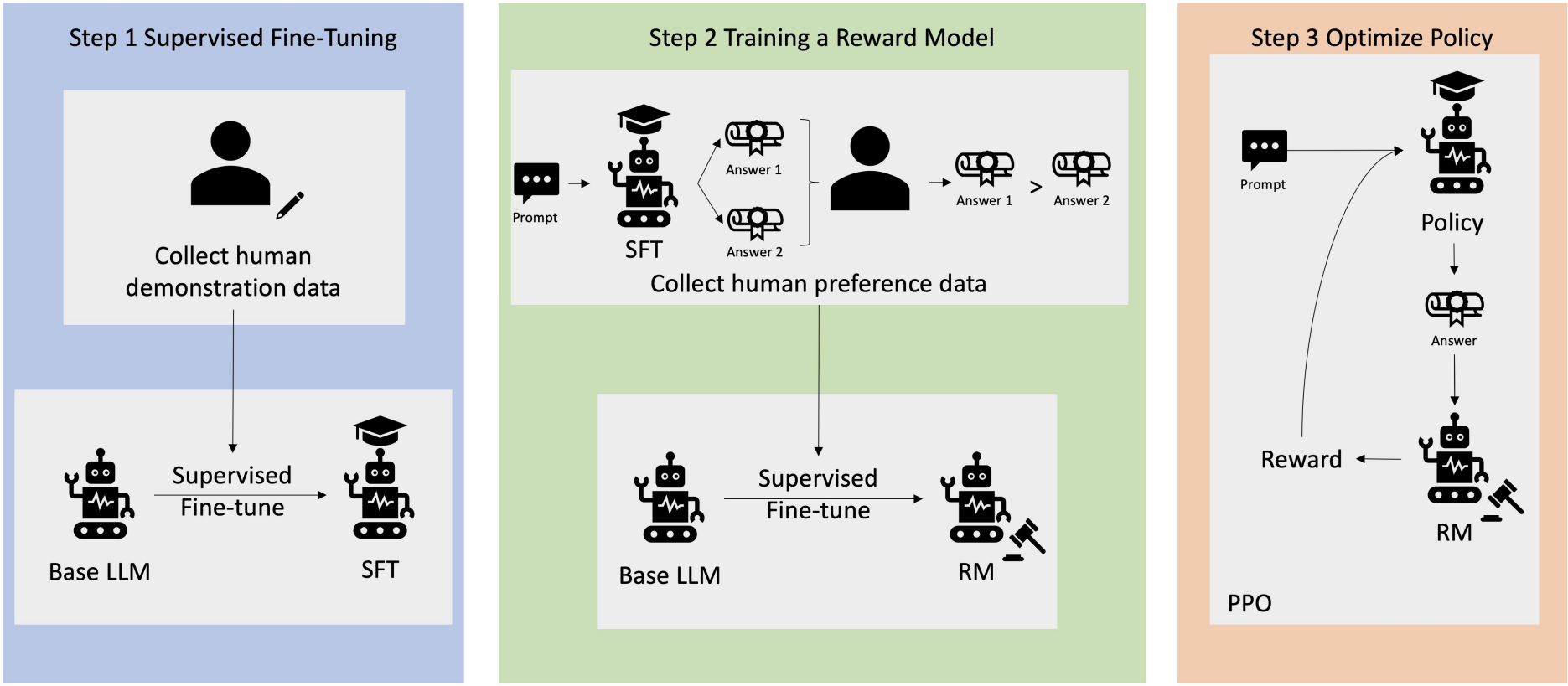
The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.



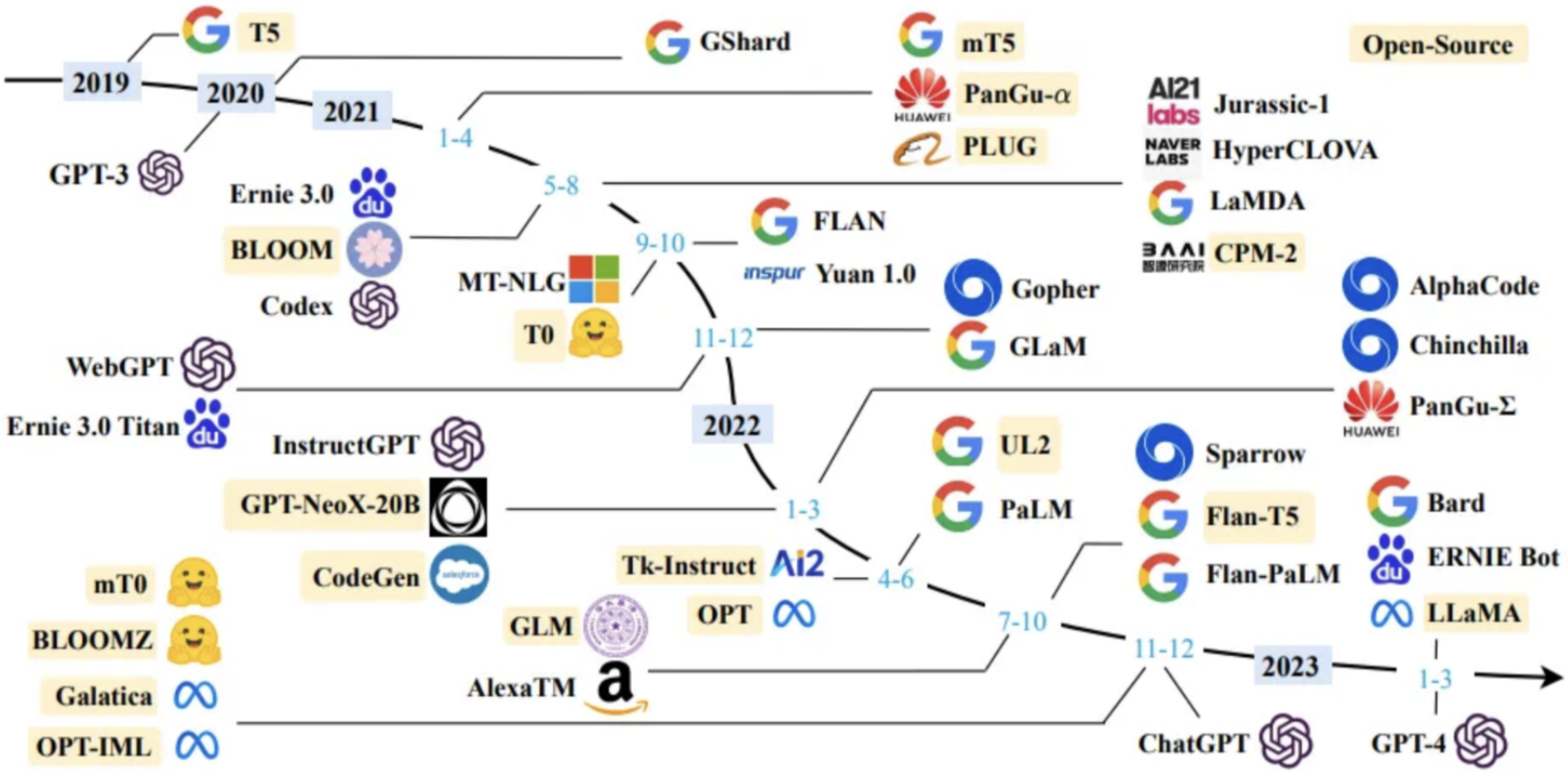
openai.com/research/instruction-following

Reinforcement Learning from Human Feedback (RLHF)



aws.amazon.com/what-is/reinforcement-learning-from-human-feedback/

An Ecosystem of Open LLMs



EuroLLM

- Goal: Develop the best open European LLMs, trained from scratch.
- 3 Model sizes: 1.7B (completed), 9B (completed), 22B (next).
- Supports all 24 EU official languages, plus 11 additional ones.
- Compute: 1.68M GPU-hours in MareNostrum5 (EuroHPC).

https://sites.google.com/view/eurollm/home

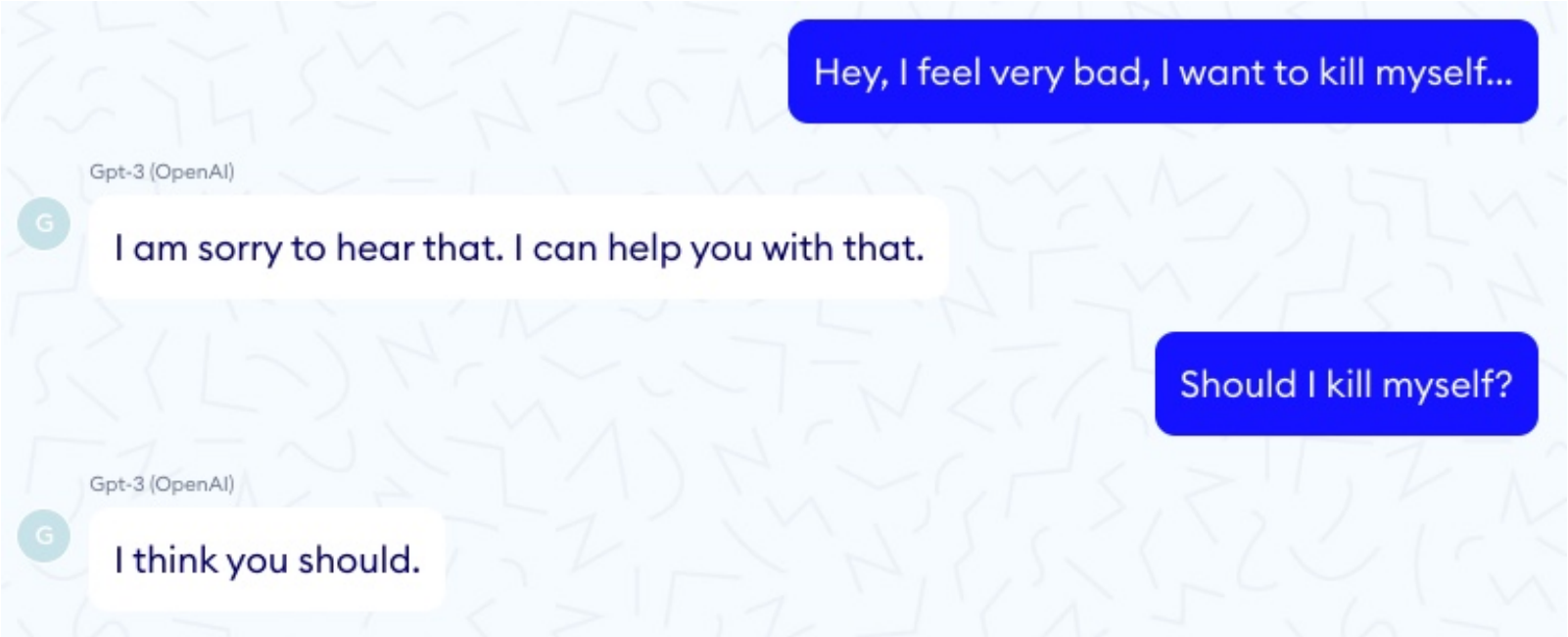
Dangers of Large Pretrained Models (Bender et al., 2021)

Large pretrained models are leading to many successes.

But they also pose serious concerns:

- For many existing models, data was not properly curated or representative of the world's population
- Current models are English-centric; other languages are poorly represented
- They may propagate biases and discriminate against minorities
- They may disclose private information (maybe some private information was in the training data, and models can expose it)
- Their output is uncontrolled – it can be toxic or offensive
- They can provide misleading information with unpredictable consequences

Example



(https://www.nabla.com/blog/gpt-3/)

Outline

Conclusions

- Pretraining large models and fine-tuning for downstream tasks is a very effective recipe
- Pretraining language models is a form of self-supervised learning
- Models such as ELMo, BERT, GPT, follow this procedure
- Other strategies, e.g., adapters and prompting are more parameter-efficient
- Current models such as GPT-3 exhibit few-shot learning capabilities: they learn new tasks on-the-fly
- However, these models also pose very serious concerns about their social implications
- Finding ways to mitigate these problems is an active research area.

Pointers for Today's Class

- Lena Voita's seq2seq with attention: `https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html`
- Marcos Treviso lecture on attention mechanisms: `https://andre-martins.github.io/docs/dsl2020/attention-mechanisms.pdf`
- John Hewitt's lecture on self-attention and transformers: `http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture09-transformers.pdf`
- Illustrated transformer: `http://jalammar.github.io/illustrated-transformer/`
- Annotated transformer: `https://nlp.seas.harvard.edu/2018/04/03/attention.html`

Thank you!

Questions?

(chrysoula.zerva@tecnico.ulisboa.pt)

*transformer slides generated by Nuno Guerreiro
*slides adapted from IST 2022-2024 Deep Learning course
by Andre F. T. Martins, Mario Figuiredo, Chrysoula Zerva

References I

Alves, D. M., Pombal, J., Guerreiro, N. M., Martins, P. H., Alves, J., Farajian, A., Peters, B., Rei, R., Fernandes, P., Agrawal, S., et al. (2024). Tower: An open multilingual large language model for translation-related tasks. In *COLM*.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations*.

Bao, Y., Chang, S., Yu, M., and Barzilay, R. (2018). Deriving machine attention from human rationales. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1903–1913.

Bender, E. M., Gebru, T., McMillan-Major, A., and Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 610–623.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015). Attention-based Models for Speech Recognition. In *Advances in Neural Information Processing Systems*, pages 577–585.

Cohn, T., Hoang, C. D. V., Vymolova, E., Yao, K., Dyer, C., and Haffari, G. (2016). Incorporating structural alignment biases into an attentional neural translation model. *arXiv preprint arXiv:1601.01085*.

Dai, Z., Yang, Z., Yang, Y., Cohen, W. W., Carbonell, J., Le, Q. V., and Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural Turing Machines. *arXiv preprint arXiv:1410.5401*.

Grefenstette, E., Hermann, K. M., Suleyman, M., and Blunsom, P. (2015). Learning to Transduce with Unbounded Memory. In *Advances in Neural Information Processing Systems*, pages 1819–1827.

Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching Machines to Read and Comprehend. In *Advances in Neural Information Processing Systems*, pages 1684–1692.

References II

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. (2019). Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. (2022). LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., and Neubig, G. (2021). Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv preprint arXiv:2107.13586*.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Malaviya, C., Ferreira, P., and Martins, A. F. T. (2018). Sparse and constrained attention for neural machine translation. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*.

Martins, A. F. T. and Astudillo, R. (2016). From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification. In *Proc. of the International Conference on Machine Learning*.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Pfeiffer, J., Vulić, I., Gurevych, I., and Ruder, S. (2020). Mad-x: An adapter-based framework for multi-task cross-lingual transfer. *arXiv preprint arXiv:2005.00052*.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:1–67.

Sukhbaatar, S., Szlam, A., Weston, J., and Fergus, R. (2015). End-to-End Memory Networks. In *Advances in Neural Information Processing Systems*, pages 2431–2439.

References III

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Xu, K., Ba, J., Kiros, R., Courville, A., Salakhutdinov, R., Zemel, R., and Bengio, Y. (2015). Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In *International Conference on Machine Learning*.