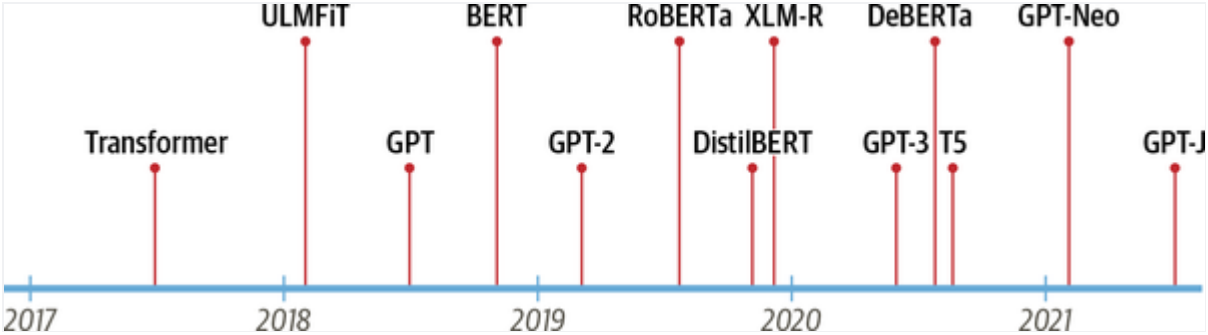
[Chapter1] NLP with Transformers

Links and references:

Tools:
models: https://huggingface.co/models
tokenizers: https://github.com/huggingface/tokenizers
datasets: https://huggingface.co/datasets

Other Learning Resources:
Huggingface Official Course: https://huggingface.co/course/chapter1
HuggingFace Live Sessions: https://www.youtube.com/watch?v=aV4wfnIakSQ&list=PLo2EIpI_JMQuQ8StH9RwKXwJVqLTDxwwy
Lucidrain Transformers implementation: https://github.com/lucidrains
Attention is all you need: https://arxiv.org/abs/1706.03762
Illustrated Transformers: https://jalammar.github.io/illustrated-transformer/
DALL-E: https://openai.com/blog/dall-e/

Chapter 1: Hello Transformers

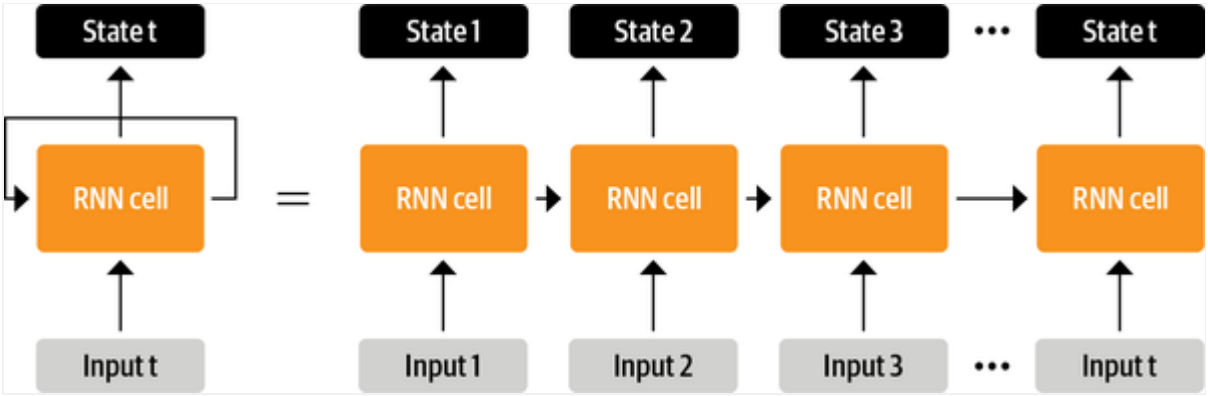


1.1 Categories:

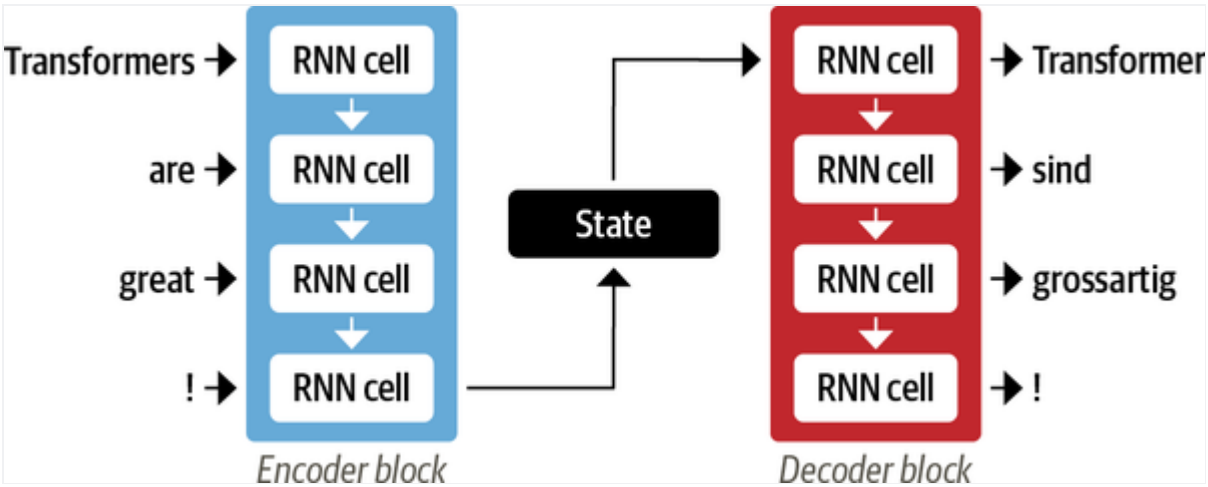
- **GPT-Like:** auto-regressive/decoder Transformers; Good for generating texts
- **BERT-like:** auto-encoding/encoder Transformers; Good for classification, extractive questions; ALBERT, DistilBERT, ELECTRA, RoBERTa
- **BART/T5:** Seq2seq Transformers: Good for summarization, language translation

1.2 Components of Transformers:

1.2.1 The The encoder-decoder framework



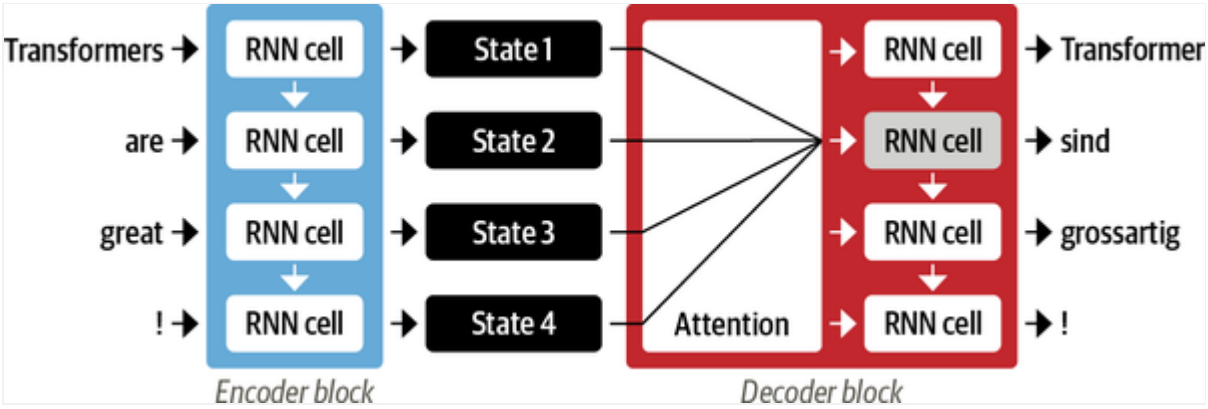
RNN as encoder-decoder (seq2seq): Prior to transformers, recurrent architectures such as LSTMs were the state of the art in NLP. These architectures contain a feedback loop in the network connections that allows information to propagate from one step to another, making them ideal for modeling sequential data like text.



Weakness of RNNs:

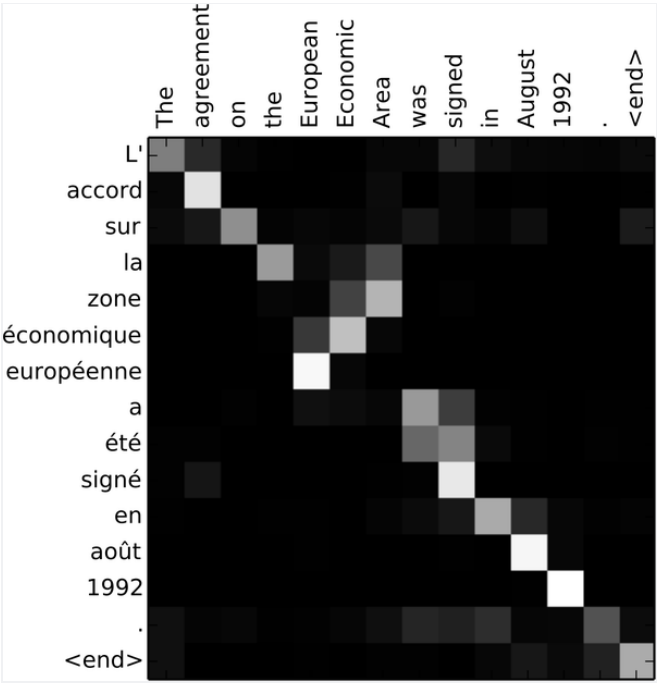
- One weakness of this architecture is that the final hidden state of the encoder creates an *information bottleneck*: it has to represent the meaning of the whole input sequence because this is all the decoder has access to when generating the output.
- This is especially challenging for long sequences, where information at the start of the sequence might be lost in the process of compressing everything to a single, fixed representation.
- A major shortcoming with using recurrent models for the encoder and decoder: the computations are inherently sequential and cannot be parallelized across the input sequence.

1.2.2 Attention mechanisms

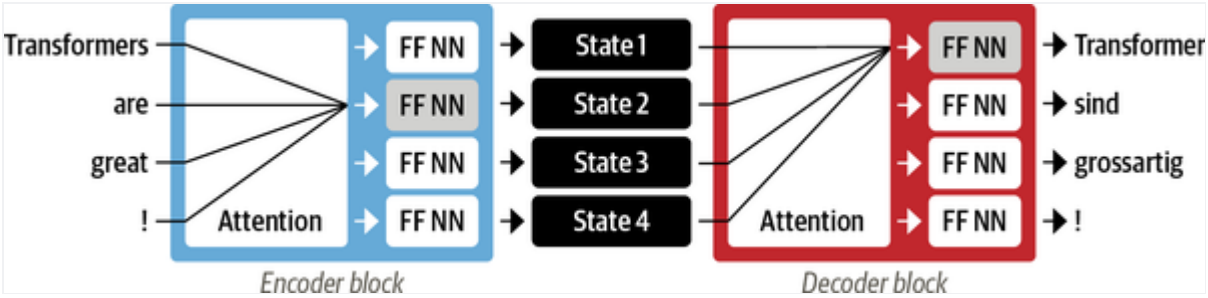


Attention Mechanism: Using all the states of the encoder at the same time would create a huge input for the decoder, so some mechanism is needed to prioritize which states to use. This is where attention comes in: it lets the decoder assign a different amount of weight, or “attention,” to each of the encoder states at every decoding timestep.

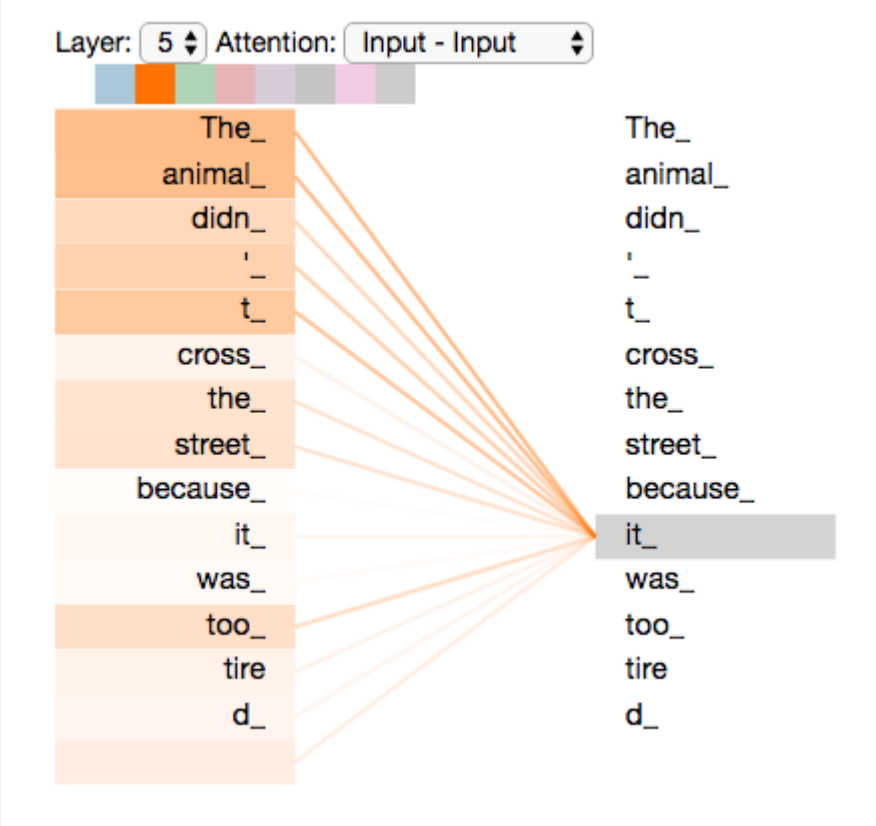
Look at all of the tokens at the same time



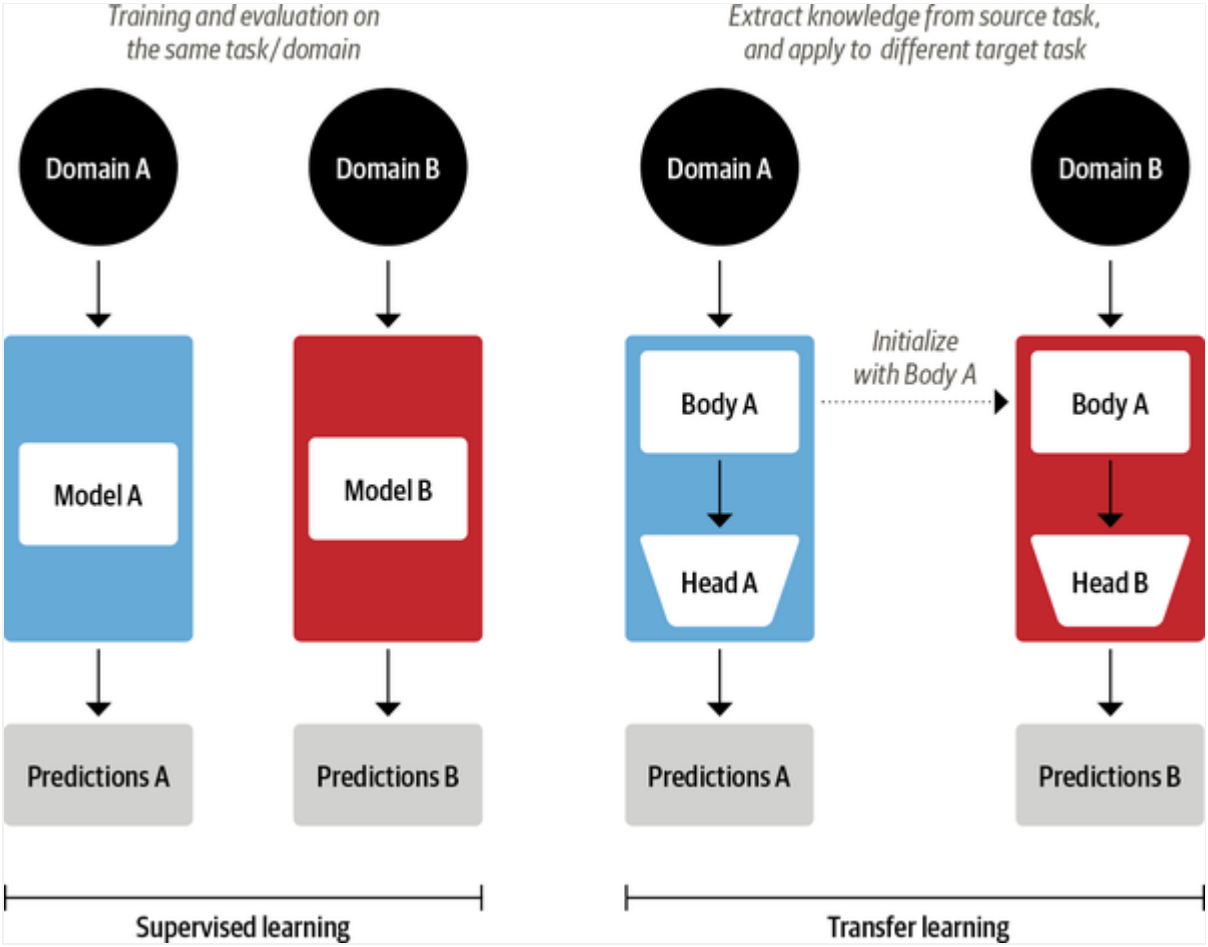
By focusing on which input tokens are most relevant at each timestep, these attention-based models are able to learn nontrivial alignments between the words in a generated translation and those in a source sentence.



Self-attention: The basic idea is to allow attention to operate on all the states in the *same layer* of the neural network.
In the figure, both the encoder and the decoder have their own self-attention mechanisms, whose outputs are fed to **feed-forward neural networks** (**FF NNs**). This architecture can be trained much faster than recurrent models and paved the way for many of the recent breakthroughs in NLP.

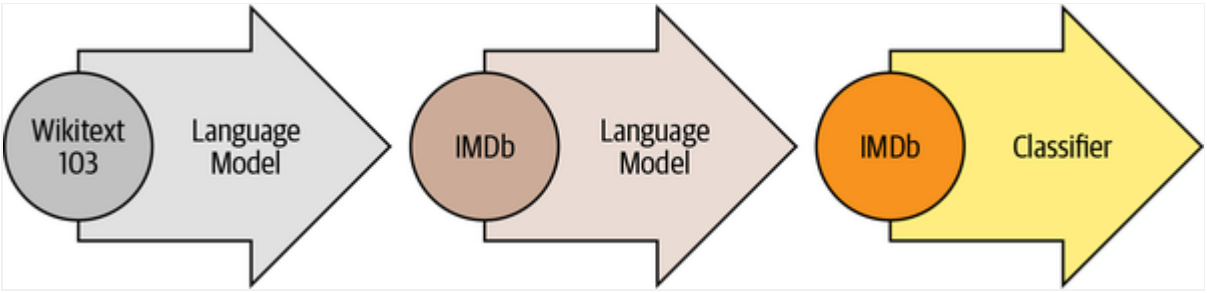


1.2.3 Transfer learning



- It is nowadays common practice in computer vision to use transfer learning to train a convolutional neural network like ResNet on one task, and then adapt it to or *fine-tune* it on a new task.
- This allows the network to make use of the knowledge learned from the original task. Architecturally, this involves splitting the model into of a *body* and a *head*, where the head is a task-specific network.
- During training, the weights of the body learn broad features of the source domain, and these weights are used to initialize a new model for the new task.
- Compared to traditional supervised learning, this approach typically produces high-quality models that can be trained much more efficiently on a variety of downstream tasks, and with much less labeled data.
-

1.2.3.1 ULMFiT (2018)



3 Steps:

Pretraining: The initial training objective is quite simple: predict the next word based on the previous words. This task is referred to as *language modeling*.

Domain Adaptation: Adapt it to the in-domain corpus. This stage still uses language modeling, but now the model has to predict the next word in the target corpus.

Fine-tuning: In this step, the language model is fine-tuned with a classification layer for the target task

1.2.3.2 GPT:

Uses only the decoder part of the Transformer architecture, and the same language modeling approach as ULMFiT.

1.2.3.3 BERT:

(Bidirectional) Uses the encoder part of the Transformer architecture, and a special form of language modeling called *masked language modeling*. The objective of masked language modeling is to predict randomly masked words in a text.

1.3 Transformer Applications:

1.3.1 Text Classification

```
from transformers import pipeline
import pandas as pd

classifier = pipeline("text-classification")

text = """Dear Amazon, last week I ordered an Optimus Prime action figure
from your online store in Germany. Unfortunately, when I opened the package,
I discovered to my horror that I had been sent an action figure of Megatron
instead! As a lifelong enemy of the Decepticons, I hope you can understand my
dilemma. To resolve the issue, I demand an exchange of Megatron for the
```

```
Optimus Prime figure I ordered. Enclosed are copies of my records concerning
this purchase. I expect to hear from you soon. Sincerely, Bumblebee."""


outputs = classifier(text)
pd.DataFrame(outputs)
```

label	score
NEGATIVE	0.901546

1.3.2 Named Entity Recognition (NER)

In NLP, real-world objects like products, places, and people are called *named entities*, and extracting them from text is called *named entity recognition* (NER).

```
ner_tagger = pipeline("ner", aggregation_strategy="simple")
outputs = ner_tagger(text)
pd.DataFrame(outputs)
```

entity_group	score	word	start	end
ORG	0.879010	Amazon	5	11
MISC	0.990859	Optimus Prime	36	49
LOC	0.999755	Germany	90	97
MISC	0.556569	Mega	208	212
PER	0.590256	##tron	212	216
ORG	0.669692	Decept	253	259
MISC	0.498350	##icons	259	264
MISC	0.775361	Megatron	350	358
MISC	0.987854	Optimus Prime	367	380
PER	0.812096	Bumblebee	502	511

1.3.3 Question Answering

In question answering, we provide the model with a passage of text called the *context*, along with a question whose answer we'd like to extract. The model then returns the span of text corresponding to the answer.

```
reader = pipeline("question-answering")
question = "What does the customer want?"
```

```
outputs = reader(question=question, context=text)
pd.DataFrame([outputs])
```

score	start	end	answer
0.631291	335	358	an exchange of Megatron

- Extractive Question Answering
- Generative Question Answering

1.3.4 Summarization

The goal of text summarization is to take a long text as input and generate a short version with all the relevant facts.

```
summarizer = pipeline("summarization")
outputs = summarizer(text, max_length=45, clean_up_tokenization_spaces=True)
print(outputs[0]['summary_text'])
```

Output:

```
Bumblebee ordered an Optimus Prime action figure from your online store in
Germany. Unfortunately, when I opened the package, I discovered to my horror
that I had been sent an action figure of Megatron instead.
```

1.3.5 Translation

Like summarization, translation is a task where the output consists of generated text.

```
translator = pipeline("translation_en_to_de",
                      model="Helsinki-NLP/opus-mt-en-de")
outputs = translator(text, clean_up_tokenization_spaces=True, min_length=100)
  print(outputs[0]['translation_text'])
```

Output:

```
Sehr geehrter Amazon, letzte Woche habe ich eine Optimus Prime Action Figur aus
Ihrem Online-Shop in Deutschland bestellt. Leider, als ich das Paket öffnete,
entdeckte ich zu meinem Entsetzen, dass ich stattdessen eine Action Figur von
Megatron geschickt worden war! Als lebenslanger Feind der Decepticons, Ich
hoffe, Sie können mein Dilemma verstehen. Um das Problem zu lösen, Ich fordere
einen Austausch von Megatron für die Optimus Prime Figur habe ich bestellt.
```

```
Anbei sind Kopien meiner Aufzeichnungen über diesen Kauf. Ich erwarte, bald von
Ihnen zu hören. Aufrichtig, Bumblebee.
```

1.3.6 Text Generation

With a text generation model you can:

```
generator = pipeline("text-generation")
response = "Dear Bumblebee, I am sorry to hear that your order was mixed up."
prompt = text + "\n\nCustomer service response:\n" + response
outputs = generator(prompt, max_length=200)
print(outputs[0]['generated_text'])
```

Output:

```
Dear Amazon, last week I ordered an Optimus Prime action figure from your online
store in Germany. Unfortunately, when I opened the package, I discovered to my
horror that I had been sent an action figure of Megatron instead! As a lifelong
enemy of the Decepticons, I hope you can understand my dilemma. To resolve the
issue, I demand an exchange of Megatron for the Optimus Prime figure I ordered.
Enclosed are copies of my records concerning this purchase. I expect to hear
from you soon. Sincerely, Bumblebee.

Customer service response:
Dear Bumblebee, I am sorry to hear that your order was mixed up. The order was
completely mislabeled, which is very common in our online store, but I can
appreciate it because it was my understanding from this site and our customer
service of the previous day that your order was not made correct in our mind and
that we are in a process of resolving this matter. We can assure you that your
order
```

1.4 Challenges with Transformers

- Language
- Data availability
- Working with long documents
- Opacity
- Bias