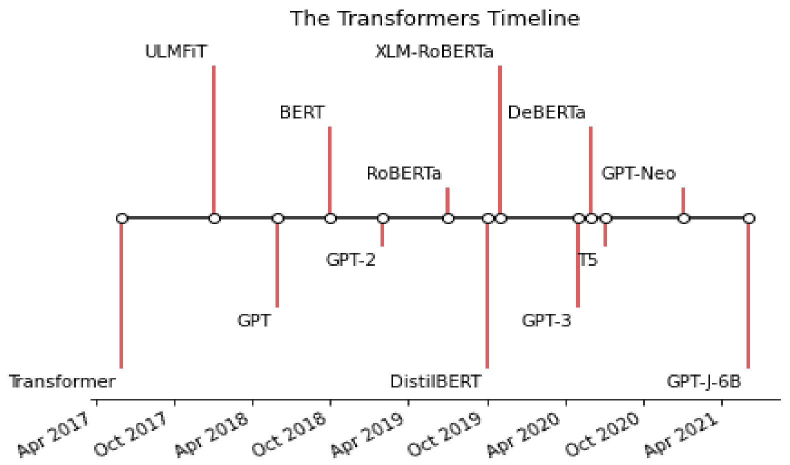
Transformers

Transformers Origin

- In 2017 researchers at Google published a paper which proposed a novel neural network architecture for sequence modeling.
- Transformer, this architecture outperformed recurrent neural networks (RNNs) on machine translation tasks, both in terms of translation quality and training cost.
- In parallel, an effective transfer learning method called ULMFiT showed that pretraining Long-Short Term Memory (LSTM) networks with a language modeling objective on a very large and diverse corpus, and then fine-tuning on a target task could produce robust text classifiers with little labeled data.
- These advances were the catalysts for two of the most well-known transformers today: GPT and BERT

Transfer Learning

- Transfer learning (TL) is a machine learning (ML) technique where a model pre-trained on one task is fine-tuned for a new, related task. Training a new ML model is a time-consuming and intensive process that requires a large amount of data, computing power, and several iterations before it is ready for production.



But we are getting ahead of ourselves. To understand what is novel about this approach combining very large datasets and a novel architecture:

1. The encoder-decoder framework
2. Attention mechanisms
3. Transfer learning

The encoder-decoder architecture

- Prior to transformers, LSTMs were the state-of-the-art in NLP. These architectures contain a cycle or feedback loop in the network connections that allows information to propagate from one step to another, making them ideal for modeling sequential data like language.

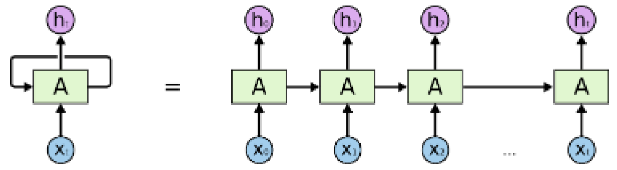
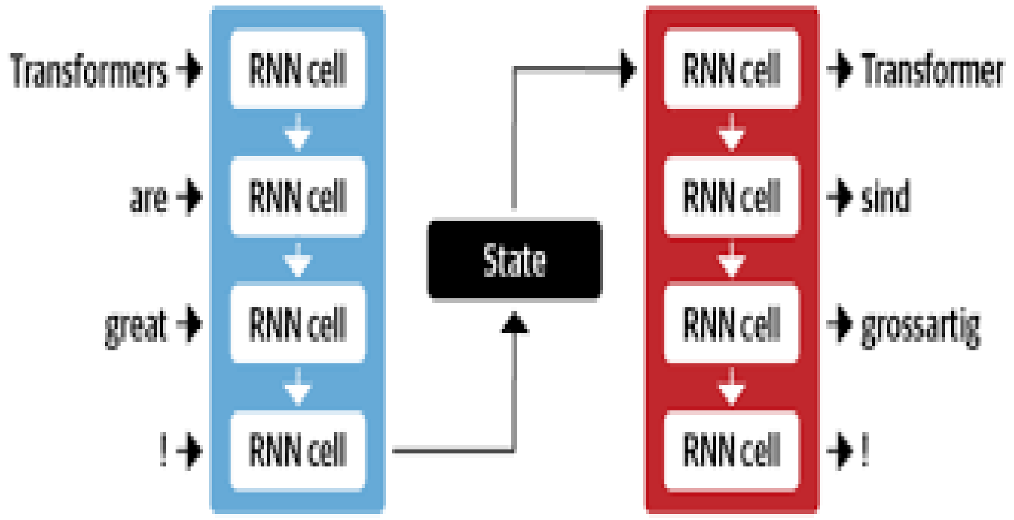


Figure 1-3. Unrolling a RNN in time.

As the name suggests, the job of the encoder is to encode the information from the input sequence into a numerical representation that is often called the last hidden state. This state is then passed to the decoder, which generates the output sequence.

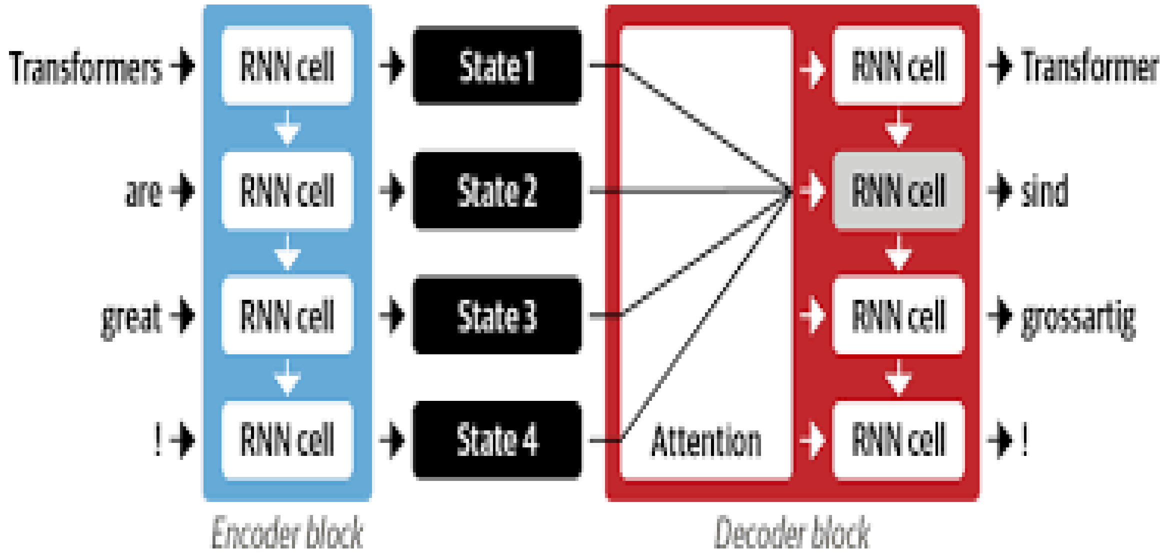
In general, the encoder and decoder components can be any kind of neural network architecture that is suited for modeling sequences.



- one weakness with this architecture is that the final hidden state of the encoder creates an information bottleneck: it has to capture 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 creating a single, fixed representation.
- Fortunately, there is a way out of this bottleneck by allowing the decoder to have access to all of the encoder’s hidden states. The general mechanism for this is called attention and is a key component in many modern neural network architectures

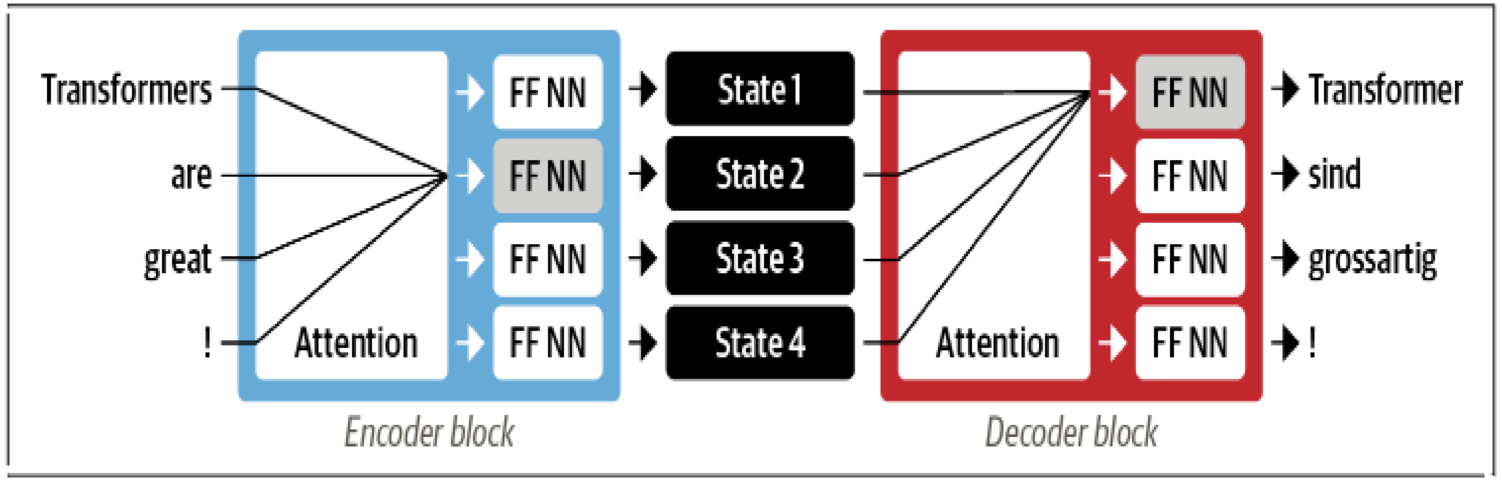
Attention Mechanism

- The main idea behind attention is that instead of producing a single hidden state for the input sequence, the encoder outputs a hidden state at each step which the decoder can access.
- However, using all states at the same time creates 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 weight or “pay attention” to the specific states in the past (and the context length can be very long - several thousands words in the past for recent models like GPT or reformers) which are most relevant for producing the next element in the output sequence.


Transformers
are
great
!
RNN cell
RNN cell
RNN cell
RNN cell
Encoder block
State 1
State 2
State 3
State 4
Attention
RNN cell
RNN cell
RNN cell
RNN cell
Decoder block
Transformer
sind
grossartig
!

- The Transformer architecture took this this idea several steps further and replaced the recurrent units inside the encoder and decoder entirely with self-attention layers and simple feed-forward networks.
- Moving from a sequential processing to a fully parallel processing unlocked strong computational efficiency gains allowing to train on orders of magnitude larger corpora for the same computational cost.
- At the same time, removing the sequential processing bottleneck of information makes the transformer architecture more efficient on several task that requires aggregating information over long time spans

Encoder-decoder architecture of the original Transformer

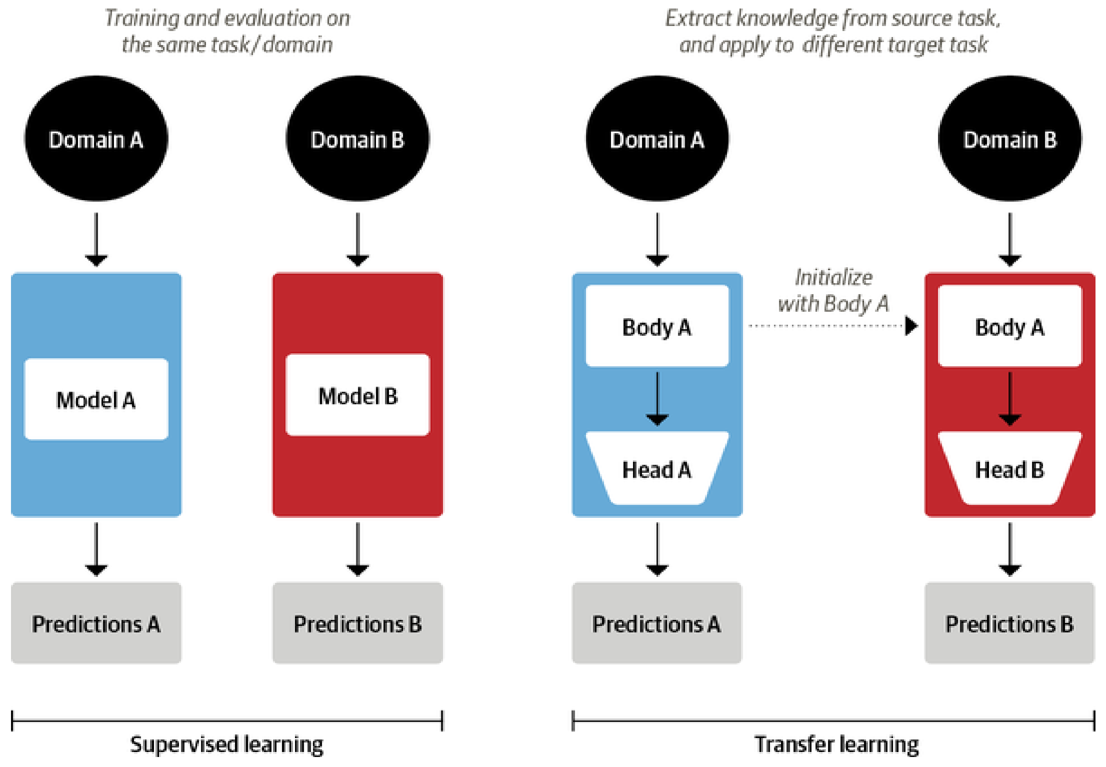


Transfer Learning

- ***Transfer Learning*** is crucial in Natural Language Processing (NLP) due to its ability to leverage knowledge learned from one task or domain and apply it to another, typically related, task or domain.

- Data Efficiency
- Resource savings
- Performance improvement
- Domain adaption
- Continual learning

Transfer learning



ULMFiT

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. The elegance of this approach lies in the fact that no labeled data is required, and one can make use of abundantly available text from sources such as Wikipedia.1

Domain adaptation

Once the language model is pretrained on a large-scale corpus, the next step is to 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

Machine Learning Architecture

- Implement the model architecture in code, typically based on PyTorch or TensorFlow.
- Load the pretrained weights (if available) from a server.
- Preprocess the inputs, pass them through the model, and apply some task-specific postprocessing.
- Implement data loaders and define loss functions and optimizers to train the model.

Each of these steps requires custom logic for each model and task.

Traditionally (but not always!), when research groups publish a new article, they will also release the code along with the model weights. However, this code is rarely standardized and often requires days of engineering to adapt to new use cases.

Transformer Applications

- Text Classification
- Named Entity Recognition
- Question Answering
- Summarization
- Translation
- Text Geneartion

Text Classification

- Transformers has a layered API that allows you to interact with the library at various levels of abstraction.
- In Transformers, we instantiate a pipeline by calling the pipeline() function and providing the name of the task we are interested in:

```
from transformers import pipeline
classifier = pipeline("text-classification")
```

The first time you run this code you’ll see a few progress bars appear because the pipeline automatically downloads the model weights from the Hugging Face Hub.

The second time you instantiate the pipeline, the library will notice that you’ve already downloaded the weights and will use the cached version instead. By default, the text classification pipeline uses a model that’s designed for sentiment analysis, but it also supports multiclass and multilabel classification.

Named Entity Recognition

- Predicting the sentiment of customer feedback is a good first step, but you often want to know if the feedback was about a particular item or service.
- 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). We can apply NER by loading the corresponding pipeline and feeding our customer review to it:

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

Question Answering

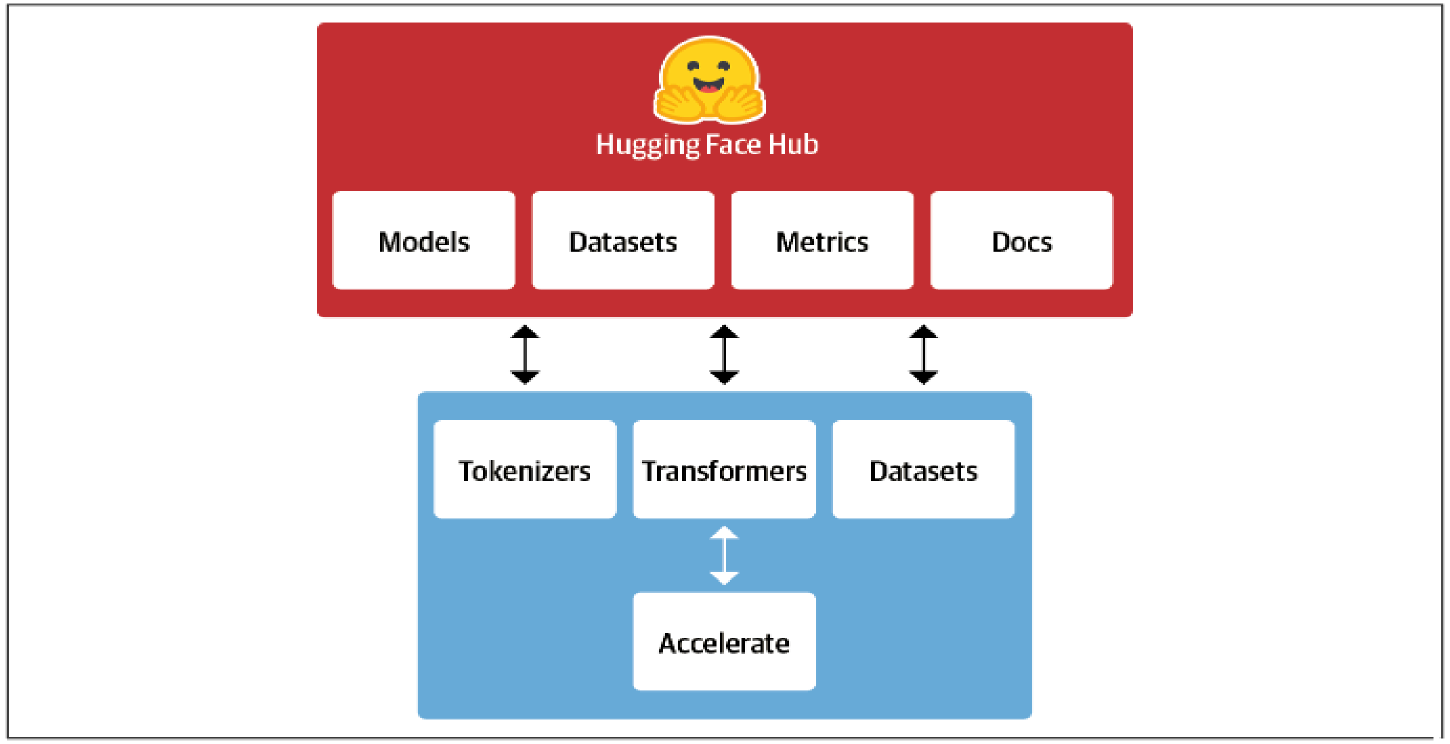
```
reader = pipeline("question-answering")
question = "What does the customer want?"
outputs = reader(question=question, context=text)
pd.DataFrame([outputs])
```

Summarization

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

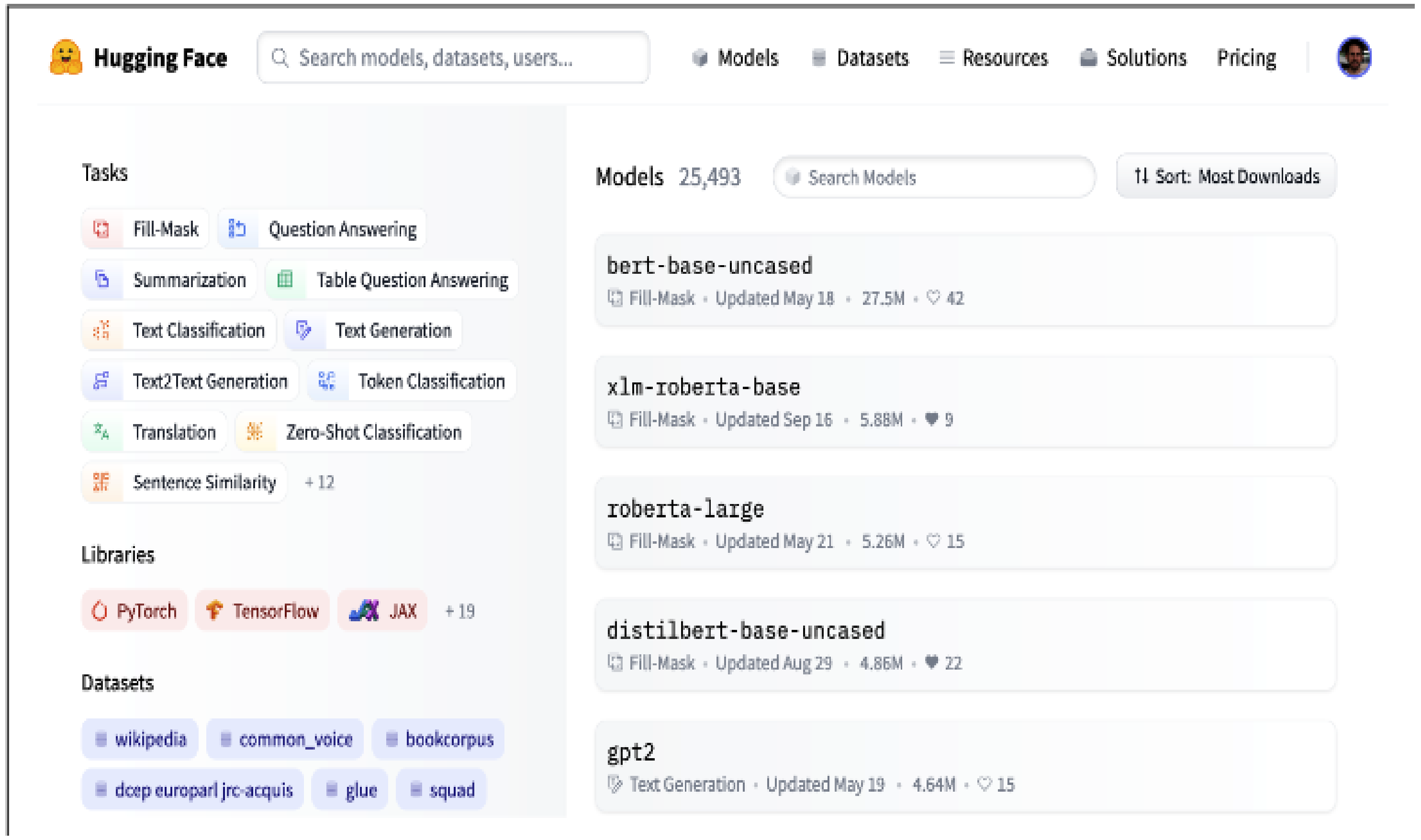
The Hugging Face Ecosystem

The Hugging Face ecosystem consists of mainly two parts: a family of libraries and the Hub



The Hugging Face Hub

- Transfer learning is one of the key factors driving the success of transformers because it makes it possible to reuse pretrained models for new tasks. Consequently, it is crucial to be able to load pretrained models quickly and run experiments with them.
- The Hugging Face Hub hosts over 20,000 freely available models.
- In addition to model weights, the Hub also hosts datasets and scripts for computing metrics, which let you reproduce published results or leverage additional data for your application.
- The Hub also provides model and dataset cards to document the contents of models and datasets and help you make an informed decision about whether they’re the right ones for you.


Hugging Face
Search models, datasets, users...
Models Datasets Resources Solutions Pricing

Tasks

Fill-Mask
Question Answering
Summarization
Table Question Answering
Text Classification
Text Generation
Text2Text Generation
Token Classification
Translation
Zero-Shot Classification
Sentence Similarity
+ 12

Libraries

PyTorch
TensorFlow
JAX
+ 19

Datasets

wikipedia
common_voice
bookcorpus
dcep europarl jrc-acquis
glue
squad

Models 25,493
Search Models
Sort: Most Downloads

bert-base-uncased
Fill-Mask · Updated May 18 · 27.5M · ♡ 42

xlm-roberta-base
Fill-Mask · Updated Sep 16 · 5.88M · ♥ 9

roberta-large
Fill-Mask · Updated May 21 · 5.26M · ♡ 15

distilbert-base-uncased
Fill-Mask · Updated Aug 29 · 4.86M · ♥ 22

gpt2
Text Generation · Updated May 19 · 4.64M · ♡ 15

Hugging Face Tokenizers

- Transformer models are trained on numerical representations of these tokens, so getting this step right is pretty important for the whole NLP project!
- Tokenizers provides many tokenization strategies and is extremely fast at tokenizing text thanks to its Rust backend.12 It also takes care of all the pre- and postprocessing steps, such as normalizing the inputs and transforming the model outputs to the required format.

Hugging Face Datasets

- Loading, processing, and storing datasets can be a cumbersome process, especially when the datasets get too large to fit in your laptop’s RAM. In addition, you usually need to implement various scripts to download the data and transform it into a standard format.
- It also provides smart caching (so you don’t have to redo your preprocessing each time you run your code) and avoids RAM limitations by leveraging a special mechanism called memory mapping that stores the contents of a file in virtual memory and enables multiple processes to modify a file more efficiently.

Hugging Face Accelerate

- If you’ve ever had to write your own training script in PyTorch, chances are that you’ve had some headaches when trying to port the code that runs on your laptop to the code that runs on your organization’s cluster.
- Accelerate adds a layer of abstraction to your normal training loops that takes care of all the custom logic necessary for the training infrastructure. This literally accelerates your workflow by simplifying the change of infrastructure when necessary

Main Challenges with Transformers

1. Language

NLP research is dominated by the English language. There are several models for other languages, but it is harder to find pretrained models for rare or low-resource languages.

2. Data availability

Although we can use transfer learning to dramatically reduce the amount of labeled training data our models need ; it is still a lot compared to how much a human needs to perform the task.

3. Working with long documents

Self-attention works extremely well on paragraph-long texts, but it becomes very expensive when we move to longer texts like whole documents.

4. Opacity

As with other deep learning models, transformers are to a large extent opaque. It is hard or impossible to unravel "why" a model made a certain prediction. This is an especially hard challenge when these models are deployed to make critical decisions.

5. Bias

Transformer models are predominantly pretrained on text data from the internet. This imprints all the biases that are present in the data into the models.