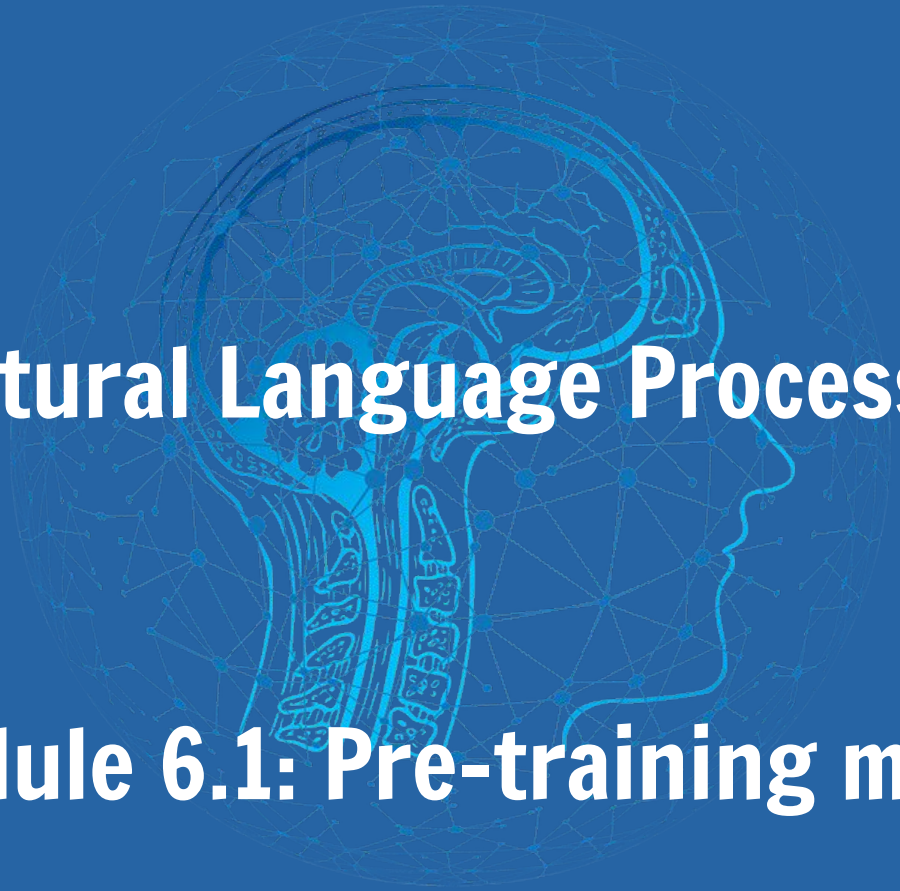
Imperial College London



Natural Language Processing

Module 6.1: Pre-training models

Today

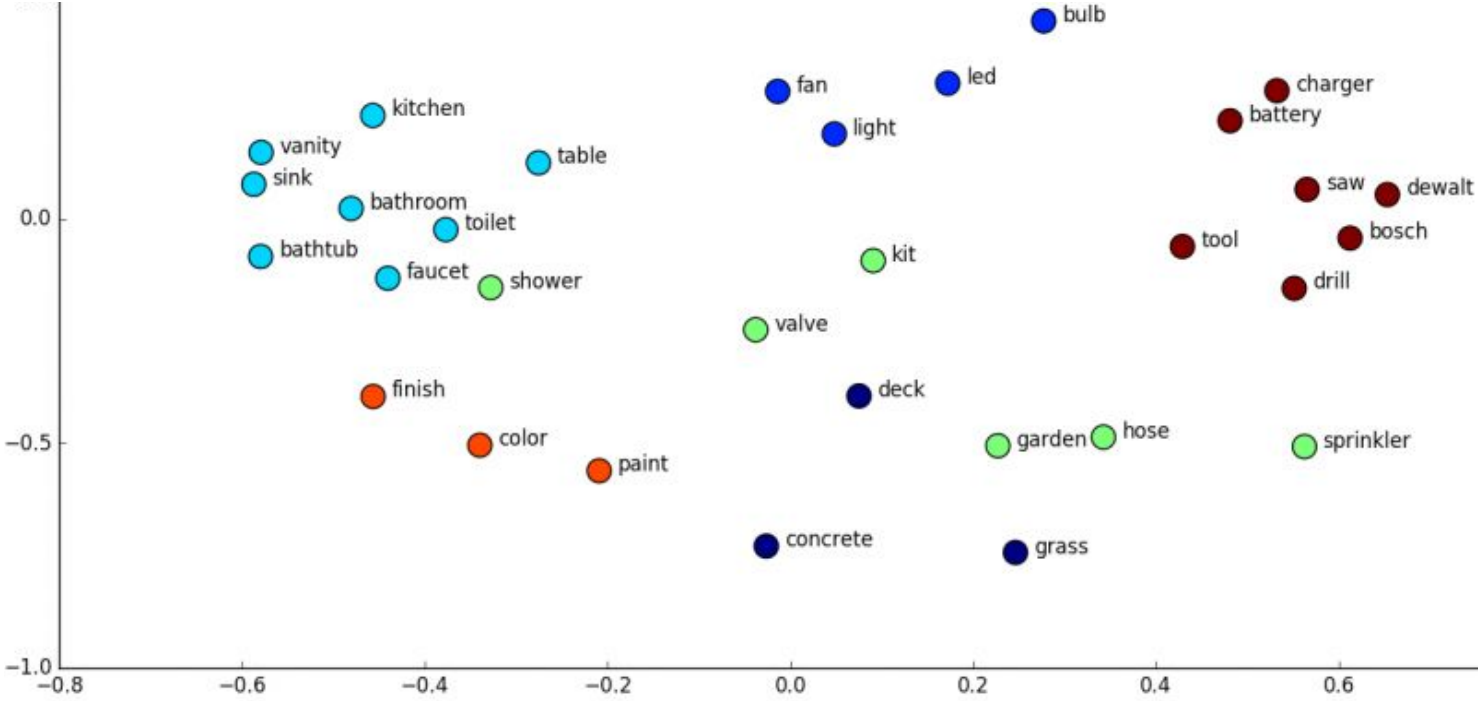
1. Contextual word representations
2. Pre-training encoder models
3. BERT and relatives
4. Pre-training encoder-decoder models

On Thursday: Pre-training decoder models. GPT and other models.

Contextual word representations

Word embeddings

We can learn a vector for each word, such that similar words have similar vectors.



Contextual word embeddings

The meaning of a word can depend on its context

I deposited some money in the **bank**

I was camping on the east **bank** of the river

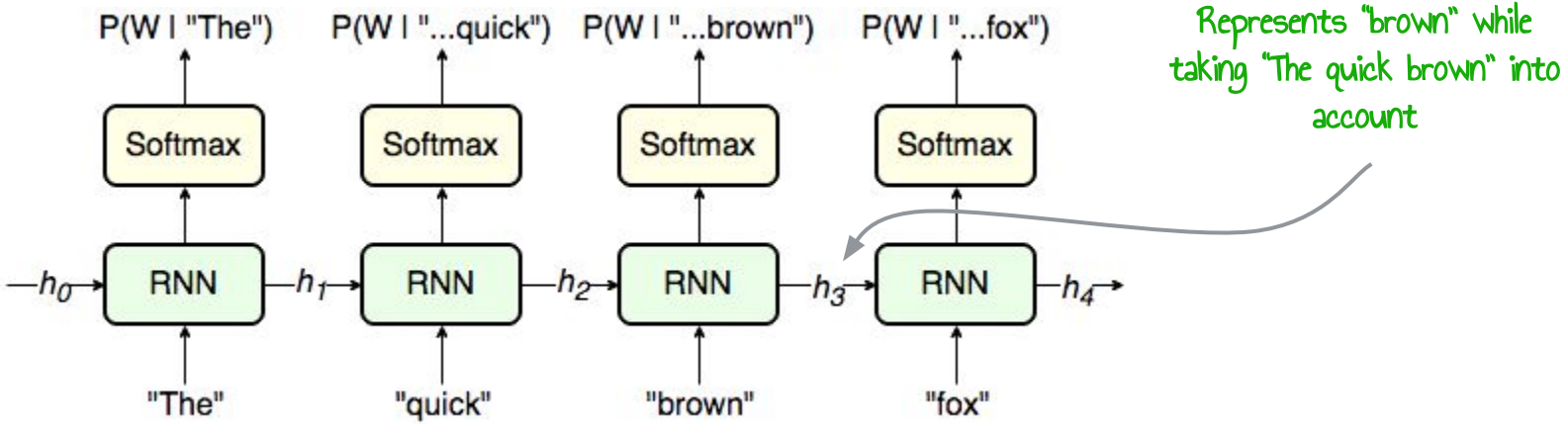
Having a single vector for every word doesn't cut it.

Need to take the context into account when constructing word representations.

Contextual word embeddings

We can train a recurrent neural network to predict the next word in the sequence, based on the previous words in the context.

Internally it will learn to encode any given context into a vector.



https://medium.com/@florijan.stamenkovic_99541/rnn-language-modelling-with-pytorch-packed-batching-and-tied-weights-9d8952db35a9

ELMo: Embeddings from Language Models

Take a large corpus of plain text and train two language models:

1. One recurrent language model going forward, left-to-right.

$$p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_1, t_2, \ldots, t_{k-1})$$

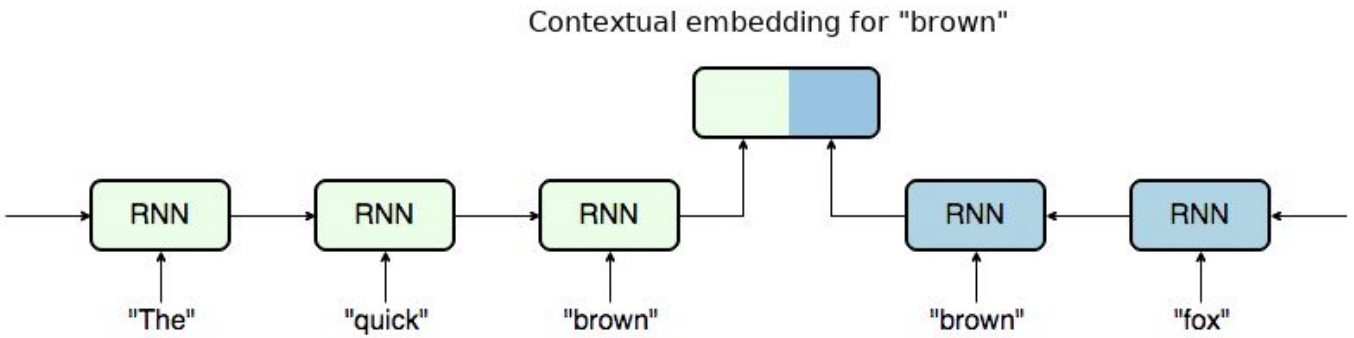
2. A second recurrent language model going backward, right-to-left.

$$p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_{k+1}, t_{k+2}, \ldots, t_N)$$

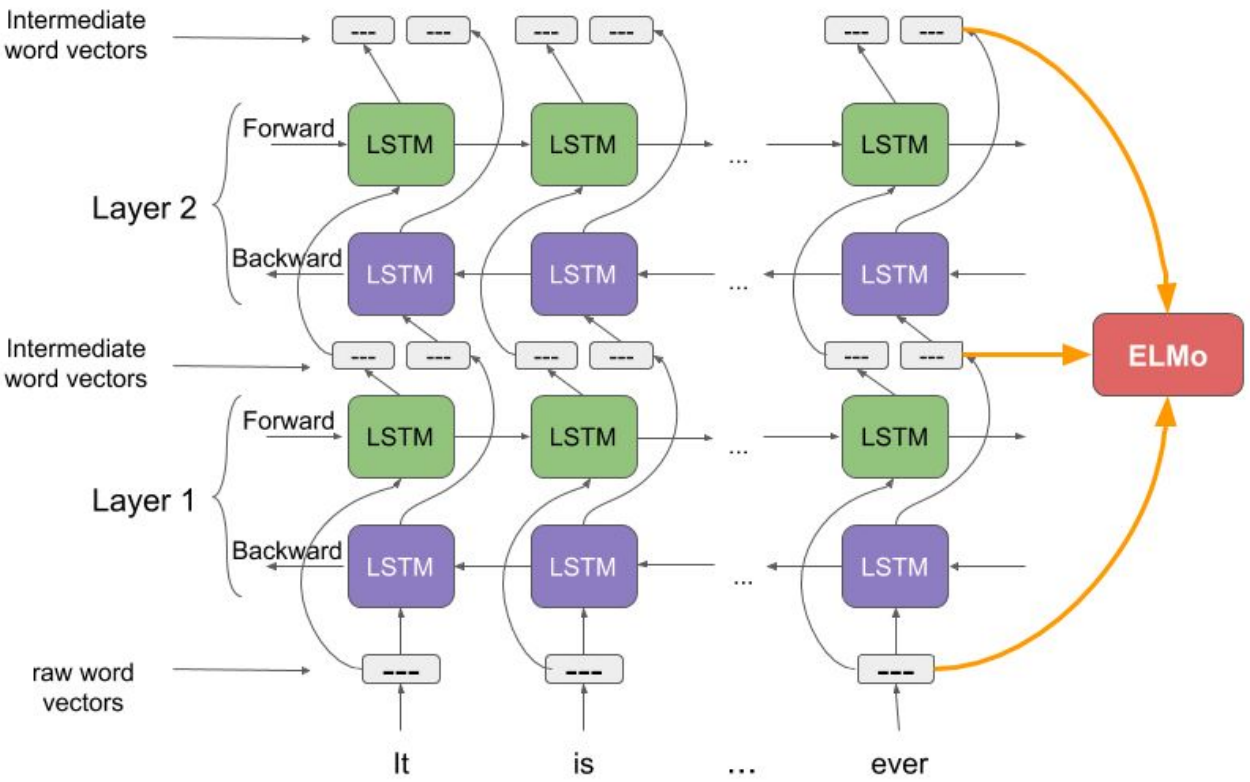
Peters et al. 2018. Deep contextualized word representations.

Contextual word embeddings

When we need a vector for a word, combine the representations from both directions.



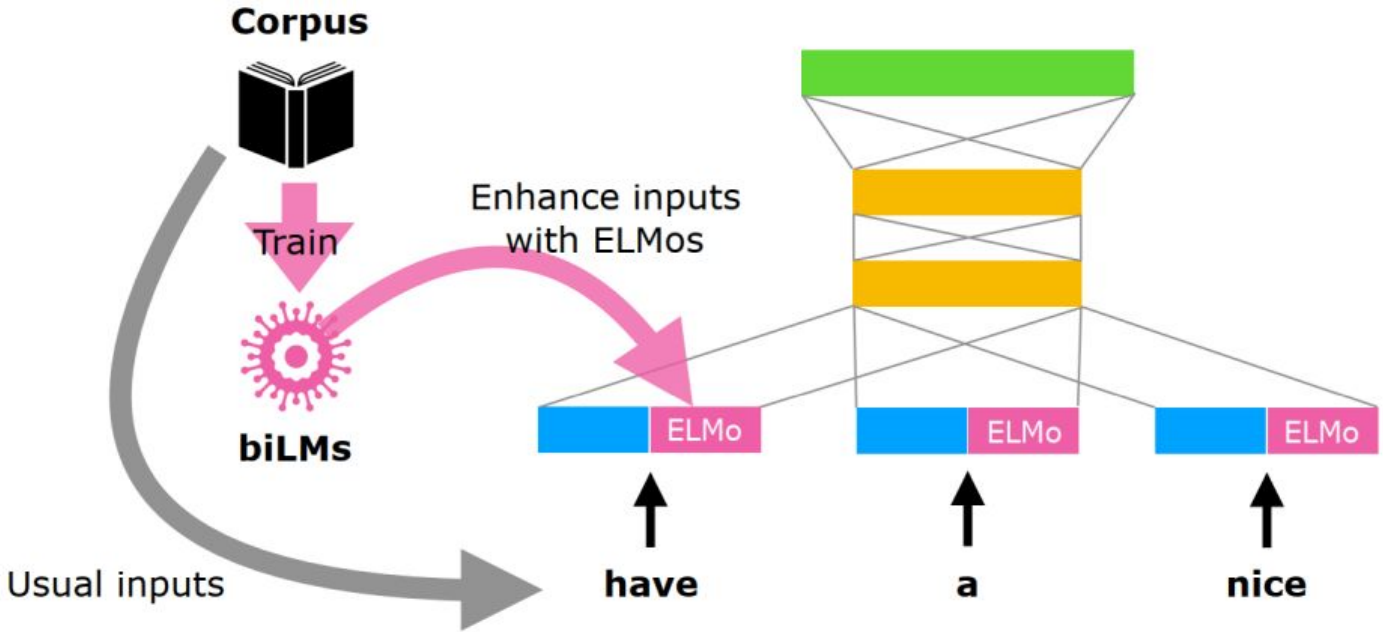
Contextual word embeddings



https://www.analyticsvidhya.com/blog/2019/03/learn-to-use-elmo-to-extract-features-from-text/

Contextual word embeddings

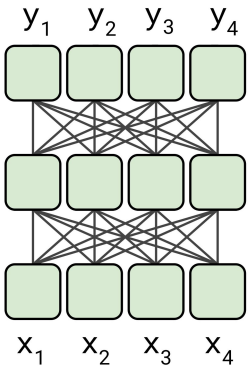
ELMo could be integrated into almost all neural NLP tasks with a simple concatenation to the embedding layer.

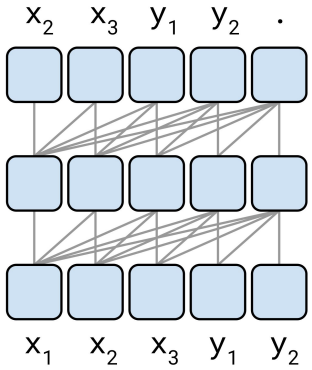


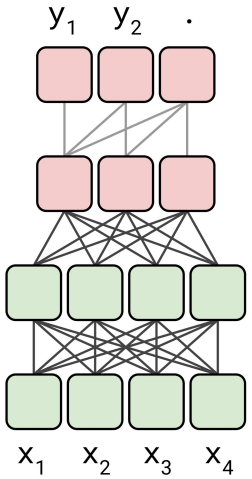
https://www.slideshare.net/shuntaroy/a-review-of-deep-contextualized-word-representations-peters-2018

Pre-training encoders

Encoder-decoder models







Encoders: Are able to access the whole sequence, using context on both sides of each token.

Decoders: Are able to access context on the left. Language models, good for generating text.

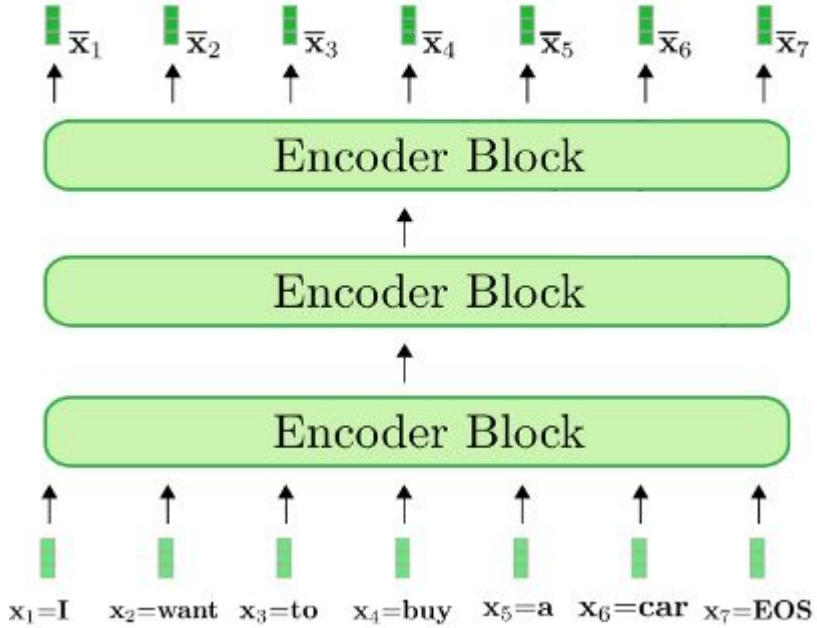
Encoder-decoder: Input is processed using an encoder, then output is generated using a decoder.

BERT

Bidirectional Encoder Representations from Transformers (Devlin et al., 2019).

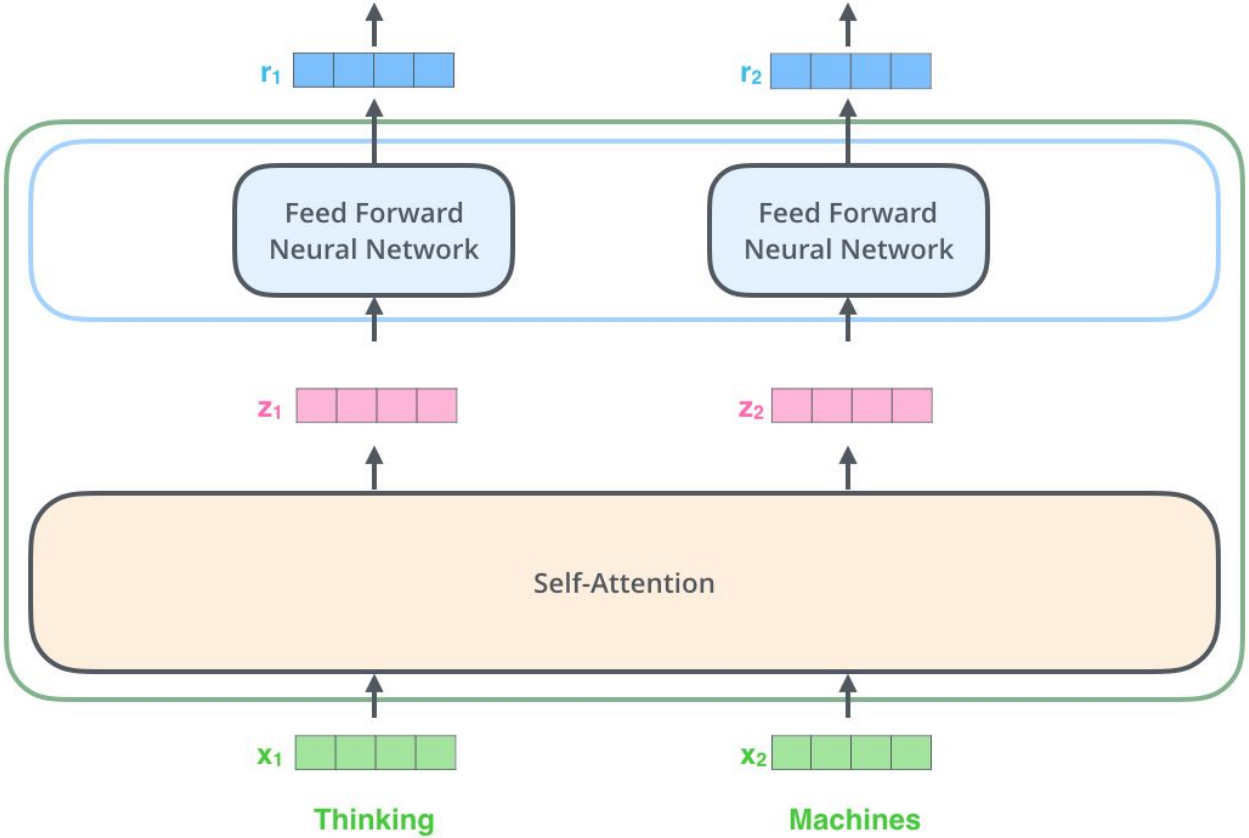
Takes in a sequence of tokens, gives as output a vector for each token.

Builds on previous work (ELMo and others), combines some good ideas and scales it up further.



https://medium.com/@gabell/encoder-decoder-models-and-transformers-5c1500c22c22

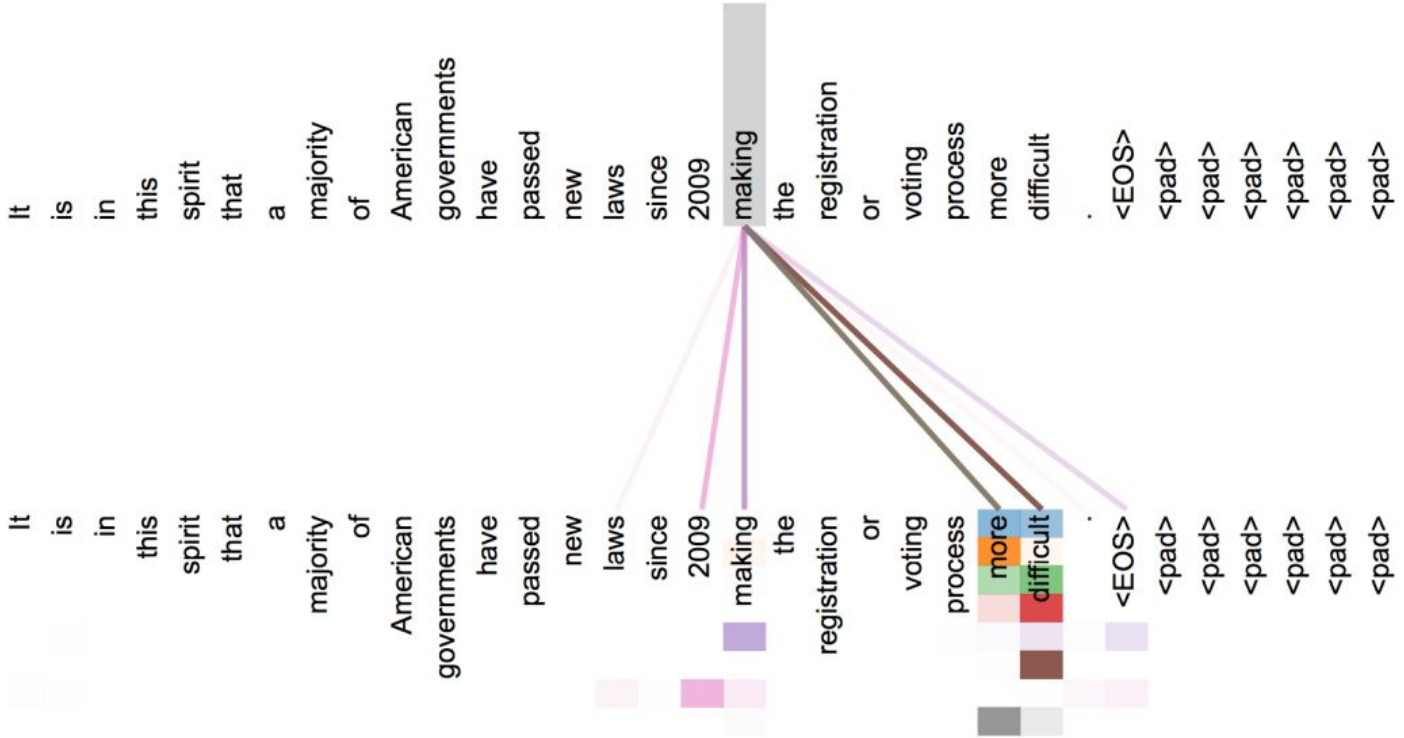
What's inside the transformer encoder



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

Self-attention

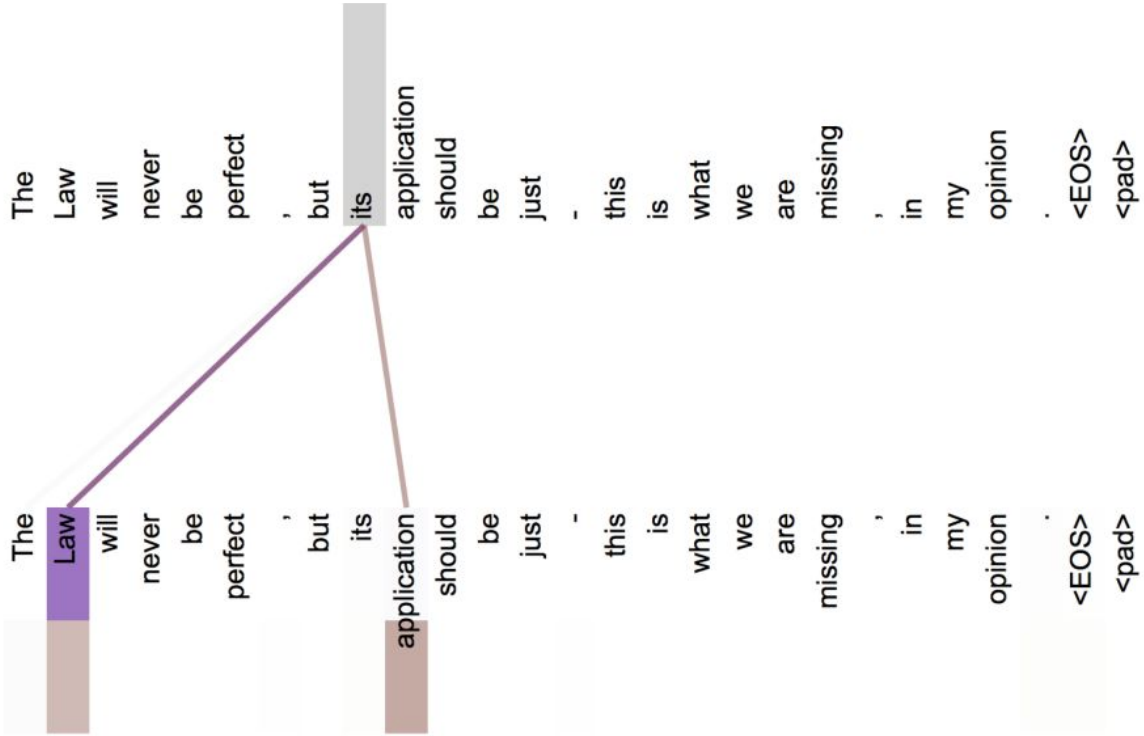
The new representation of each word is calculated based on all the other words.



https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/slides/cs224n-2019-lecture13-contextual-representations.pdf

Self-attention

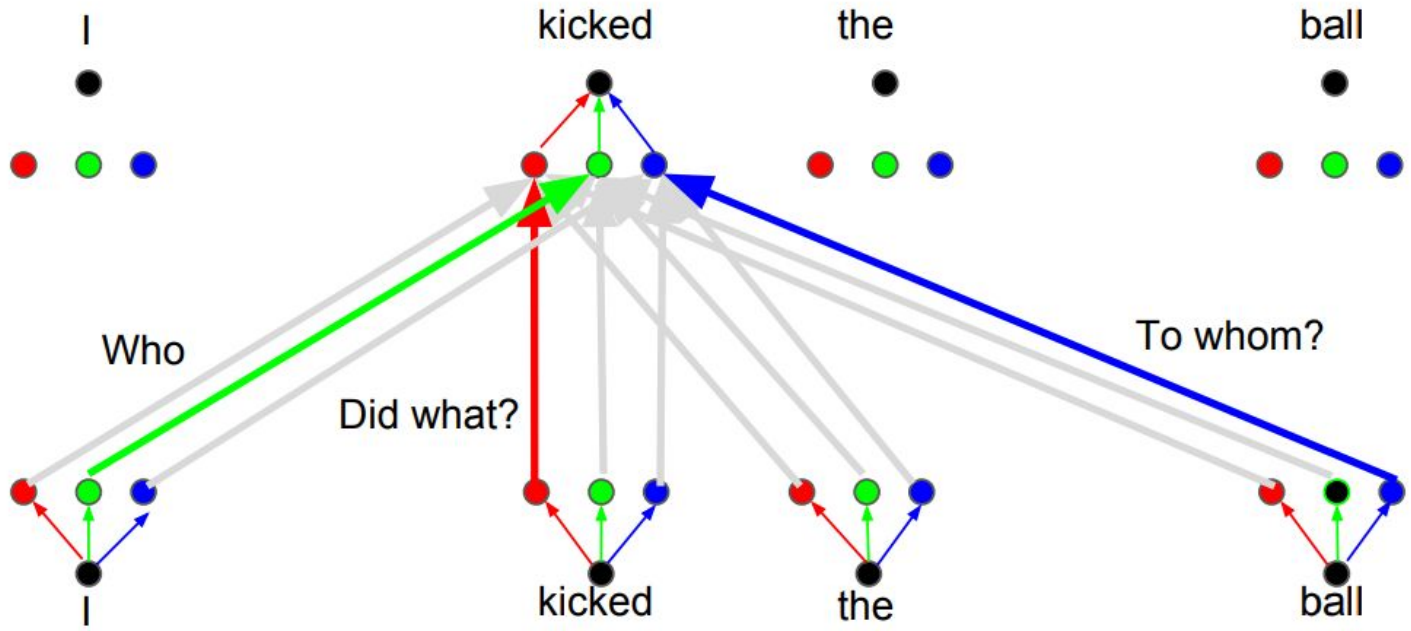
Unlike RNNs and LSTMs, every word is just one step away from every other word



https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/slides/cs224n-2019-lecture13-contextual-representations.pdf

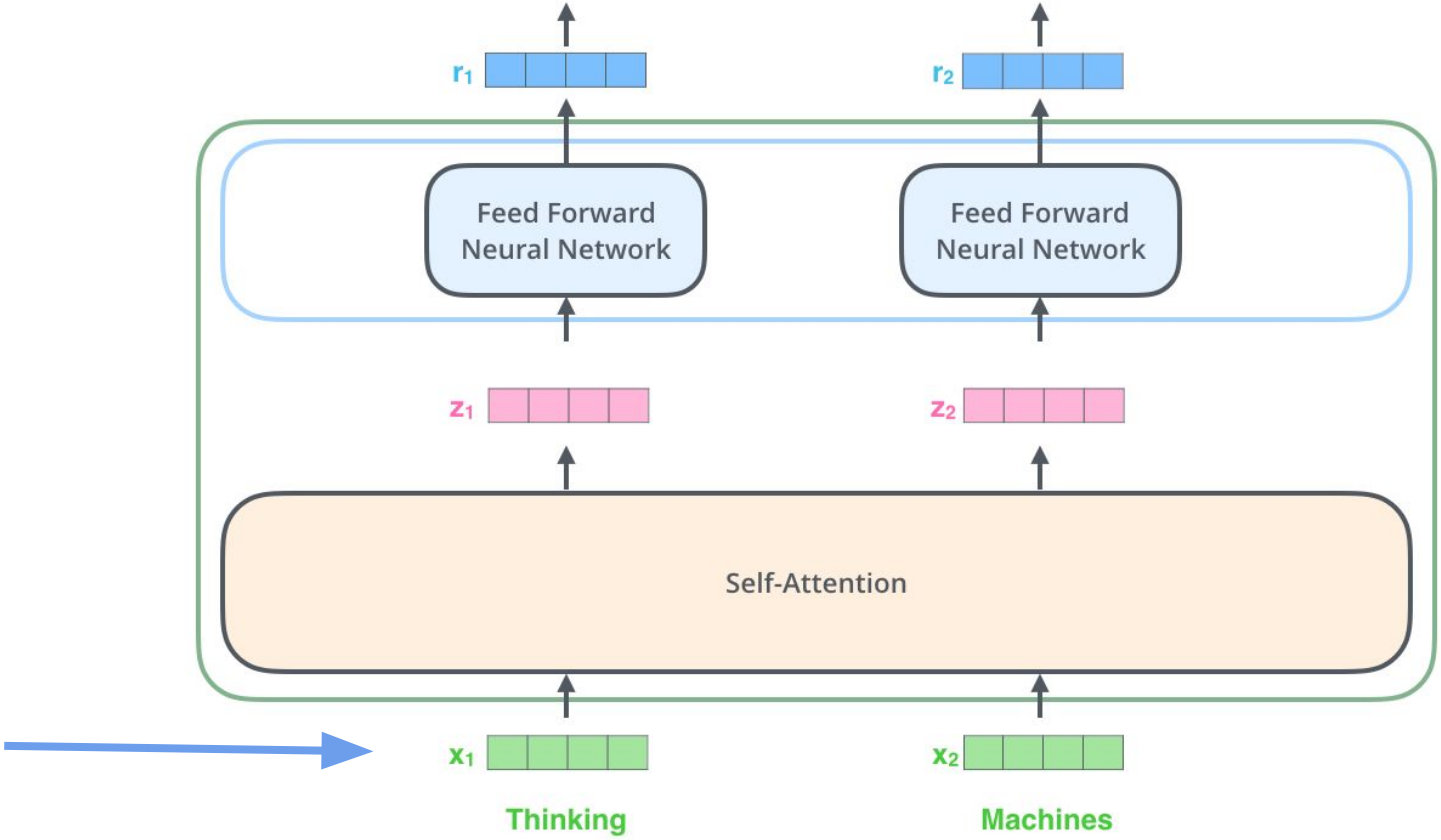
Multi-head self-attention

Each head learns to focus on different type of information.



http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture14-transformers.pdf

What's inside the encoder

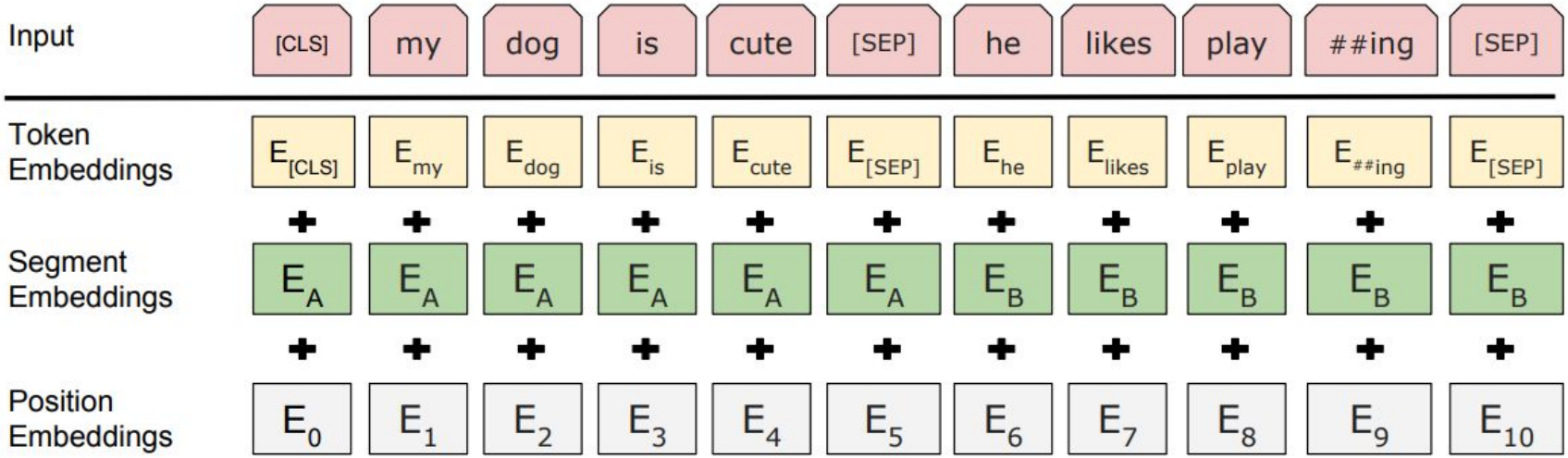


http://jalammar.github.io/illustrated-transformer/

Input embeddings

Transformers have no concept of sequence order.

Positional embeddings allow it to capture the order of the input words.



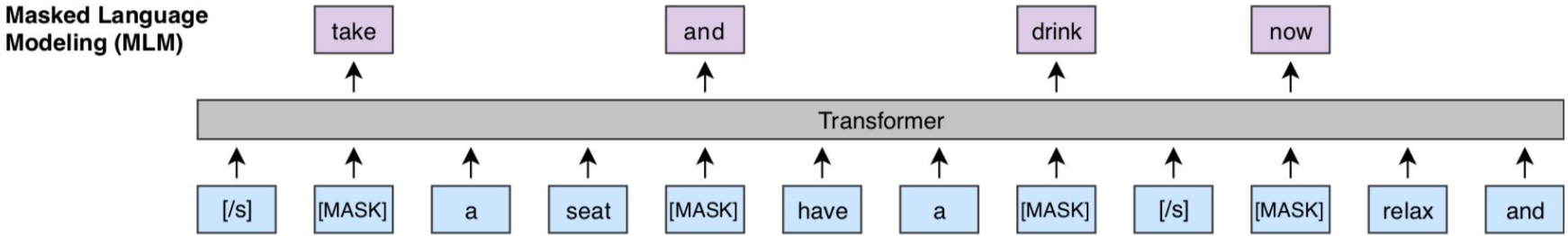
Devlin et al. (2018)

Masked Language Modeling

We want to train this model on huge amounts of plain text, so it learns general-purpose language understanding.

Use masked language modeling as the training task - it requires no labeled data!

Hide k% of the input words behind a mask, train the model to predict them.



Masked Language Modeling

I went for a walk in the park and stopped to eat the tastiest ham and _____ sandwich ever.

Imperial College is a public research university in _____, United Kingdom.

The value I got from the two hours watching it was the sum total of the popcorn and the drink. This movie was _____.

Masked Language Modeling

Too little masking: too expensive to train
Too much masking: not enough context to make predictions

Actual strategy used:
Pick 15% of the input words as training targets

- 80% of those are replaced with the [MASK] token
  `went to the store -> went to the [MASK]`
- 10% are replaced with a random word
  `went to the store -> went to the running`
- 10% are the left as the original word
  `went to the store -> went to the store`

If we masked all the chosen words, the model wouldn't necessarily learn to construct good representations for non-masked words.

Next sentence prediction

BERT used a second pre-training objective.

Given two sentences, predict whether they appeared in the original order in the source text.

Sentence A = The man went to the store.
Sentence B = He bought a gallon of milk.
Label = IsNextSentence

Sentence A = The man went to the store.
Sentence B = Penguins are flightless.
Label = NotNextSentence

Later work found that this did not provide much additional benefit, so it has not been used in newer versions.

BERT details

Two model variants:

- BERT-base
  - 12 layers
  - 768-dimensional hidden states
  - 12 attention heads
  - 110 million parameters
- BERT-large
  - 24 layers
  - 1024-dimensional hidden states
  - 16 attention heads
  - 340 million parameters

Trained on:

- BooksCorpus (800 million words)
- English Wikipedia (2,500 million words)

Pre-trained with 64 TPU chips for a total of 4 days

Putting pre-trained models to work

How do we use these models?

BERT-like models give us a representation vector for every input token.
Just have to chop off the masked LM head, it is no longer needed.

We can use those vectors to represent individual tokens or full sentences.

Option 1:

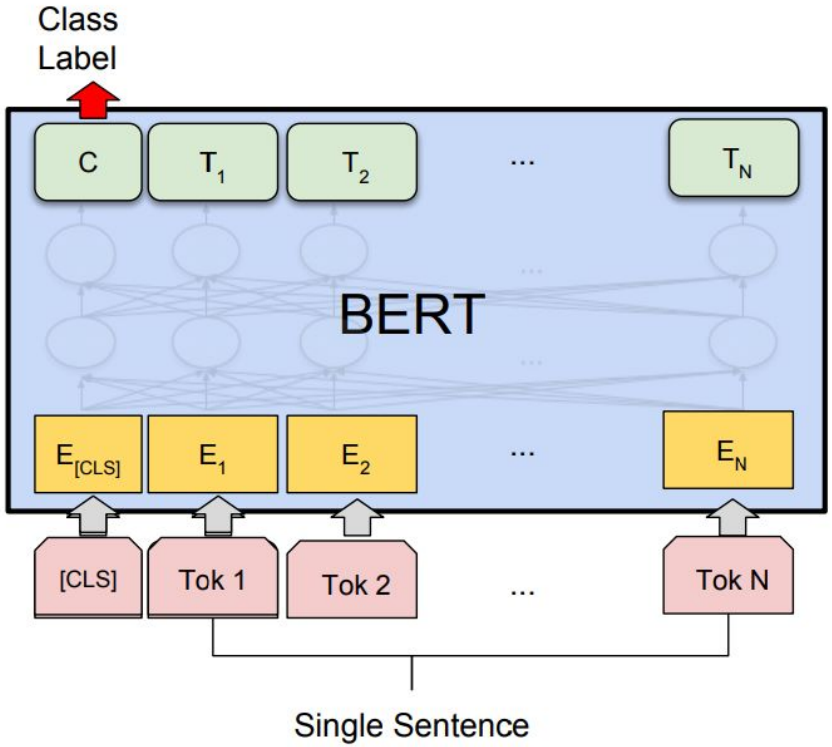
Freeze BERT, use it to calculate informative representation vectors.
Train another ML model that uses these vectors as input.

Option 2 (more common these days):

Put a minimal neural architecture on top of BERT (e.g. a single output layer)
Train the whole thing end-to-end (called *fine-tuning*).

Sentence classification

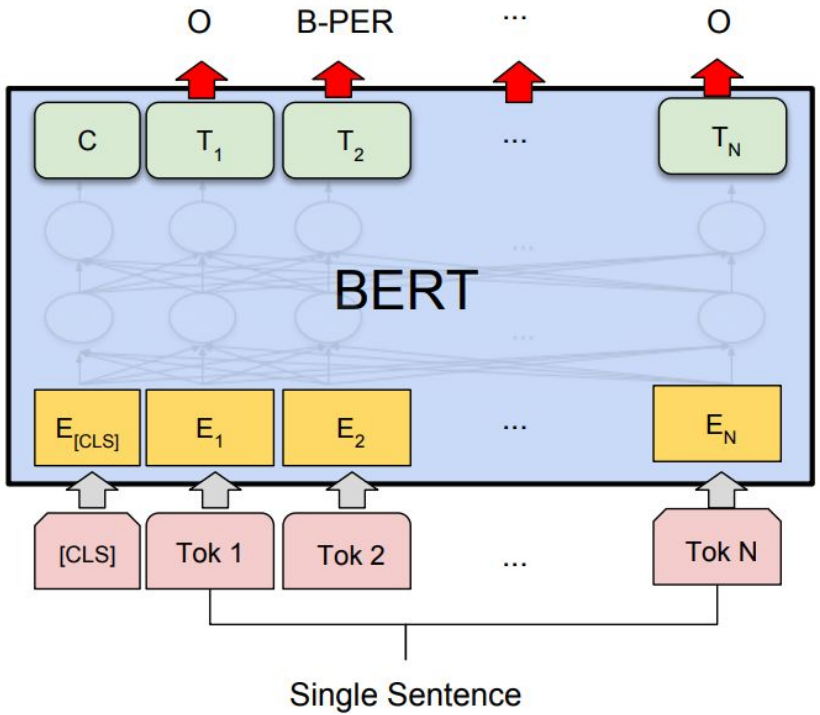
We can add a special token in the input that represents the whole sentence



Devlin et al. (2018)

Token labeling

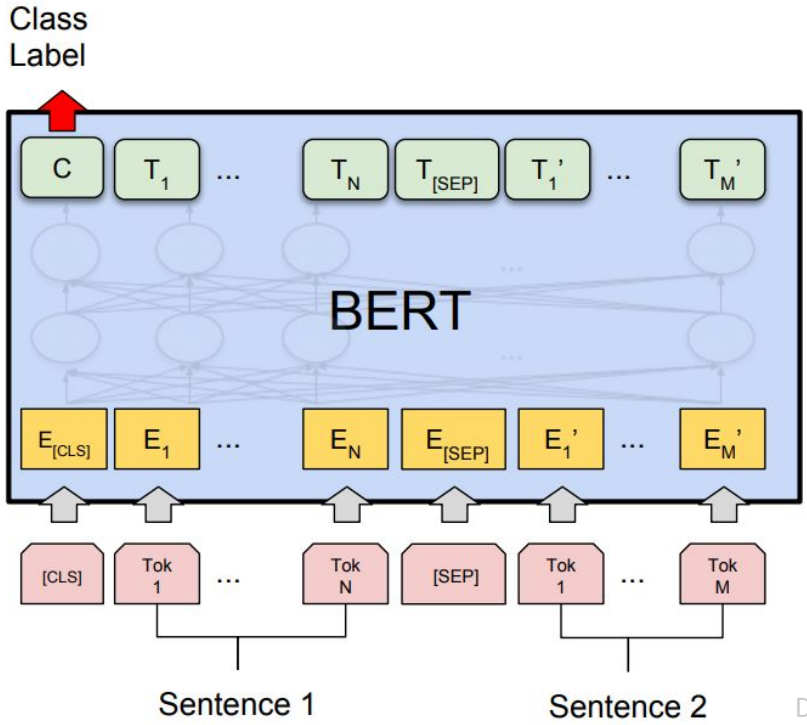
Putting a classification layer on top of token vectors



Devlin et al. (2018)

Sentence pair classification

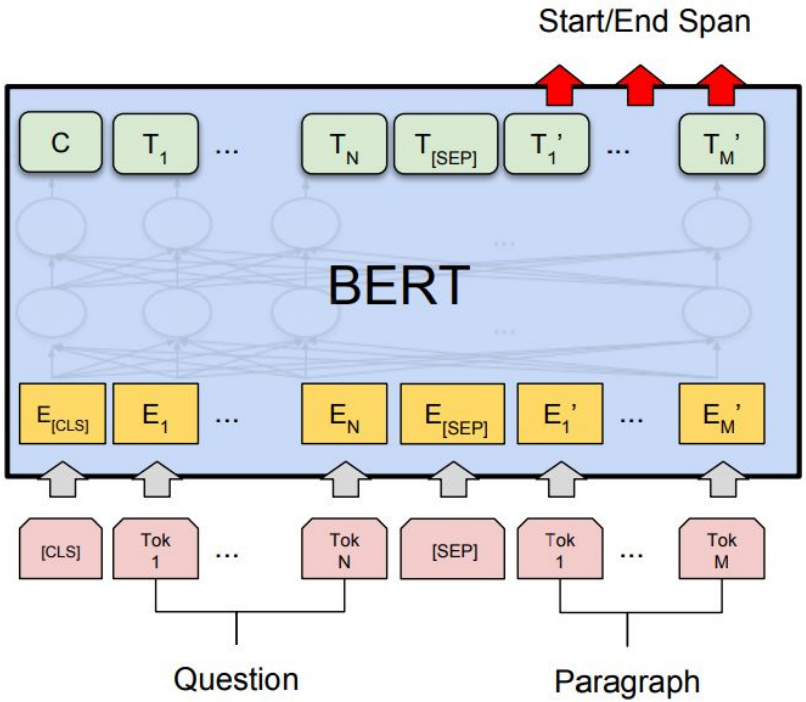
Giving multiple sentences as input



Devlin et al. (2018)

Question answering

Labeling the tokens in the candidate answer span



Devlin et al. (2018)

Performance improvements

Convincingly helps on pretty much every natural language task

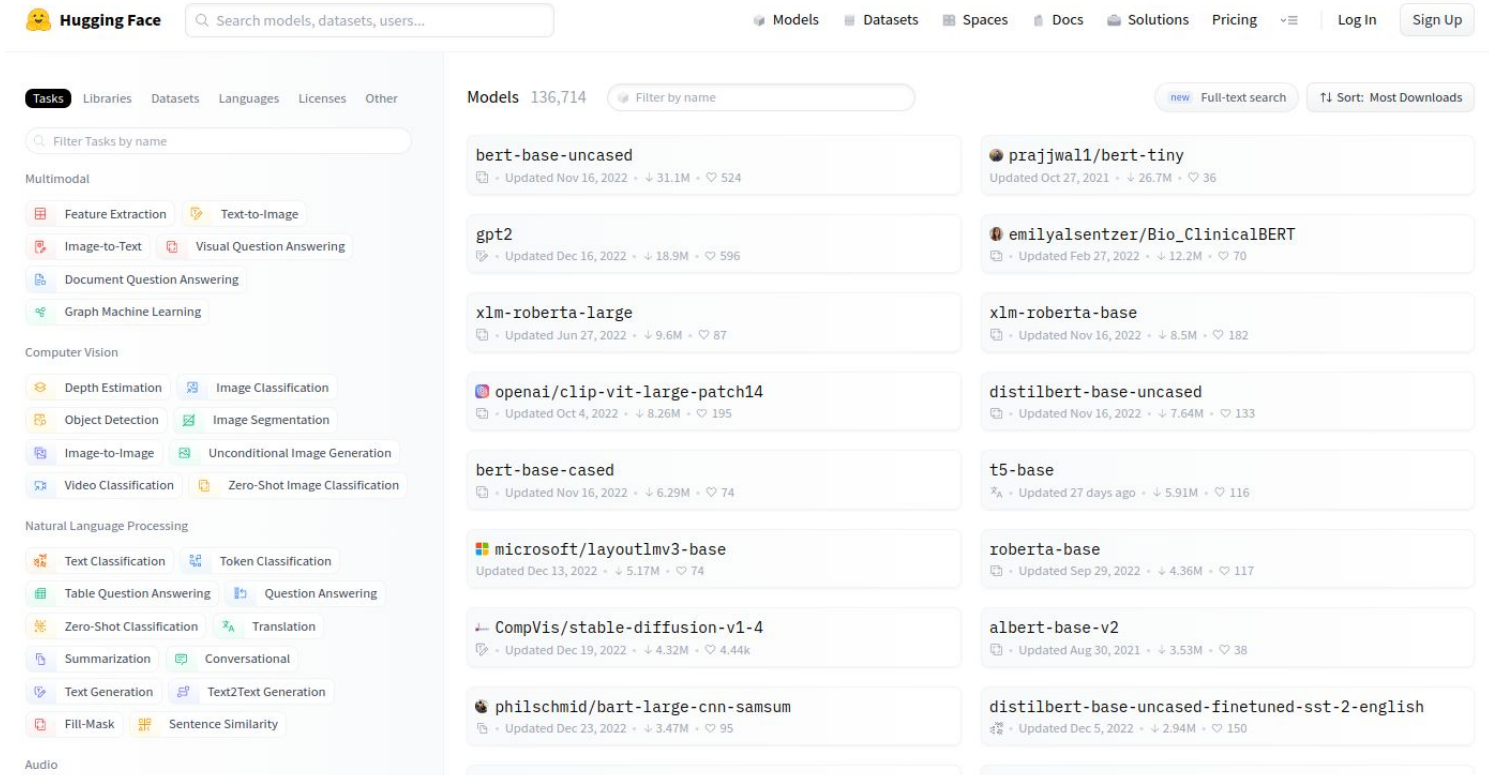
System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	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**

QQP, **STS-B**, **MRPC**: Detecting similar text and paraphrases
QNLI, **RTE**: Detecting entailment (natural language inference)
SST-2: Sentiment analysis (positive / negative / neutral)
CoLA: Detecting grammaticality of text

Text classification with BERT in practice

huggingface.co

A huge library of pre-trained models and a python framework for accessing them.



Python code example

https://github.com/marekrei/bert_text_classification_example

Contextual word embeddings

```
# Text classification example with BERT
# Created by Marek Rei
# Based on https://colab.research.google.com/github/huggingface/notebooks/blob/master/cours
# Training the model for binary sentiment detection, using the SST2 dataset.

# Some settings
# Which pre-trained model to use.
# See https://huggingface.co/models for options.
checkpoint = "bert-base-uncased"
```

Identifies the model to load from Huggingface

```
# Loading the pretrained model
model = AutoModelForSequenceClassification.from_pretrained(checkpoint, num_labels=2)
model = model.to(device)
```

Loading the model

```
# Load the data
raw_datasets = load_dataset("glue", "sst2")
raw_datasets.cleanup_cache_files()
```

Loading the dataset

```
# Perform tokenization
tokenizer = AutoTokenizer.from_pretrained(checkpoint)

def tokenize_function(example):
    return tokenizer(example["sentence"], truncation=True)

tokenized_datasets = raw_datasets.map(tokenize_function, batched=True)
```

Tokenizer directly from the Huggingface model

Contextual word embeddings

```
# Setting up model training for fine-tuning
optimizer = AdamW(model.parameters(), lr=5e-5)

num_epochs = 3
num_training_steps = num_epochs * len(train_dataloader)
lr_scheduler = get_scheduler(
    "linear",
    optimizer=optimizer,
    num_warmup_steps=0,
    num_training_steps=num_training_steps,
)
```

Setting up the optimizer

```
# Setting the model to training mode
model.train()

# Running the training
progress_bar = tqdm(range(num_training_steps))
for epoch in range(num_epochs):
    for batch in train_dataloader:
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        loss = outputs.loss
        loss.backward()

        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()
        progress_bar.update(1)
```

Activating training mode (activates dropout, batch norm, etc)

Training loop

Contextual word embeddings

```
# Setting the model to evaluation mode
model.eval()

# Running evaluation
metric = evaluate.load("glue", "sst2")
for batch in eval_dataloader:
    batch = {k: v.to(device) for k, v in batch.items()}
    with torch.no_grad():
        outputs = model(**batch)

    logits = outputs.logits
    predictions = torch.argmax(logits, dim=-1)
    metric.add_batch(predictions=predictions, references=batch["labels"])

print(metric.compute())
```

Setting the model to eval mode (turning off dropout, batchnorm, etc)

Printing out evaluation metric

```
Downloading builder script: 100% 5.75k/5.75k [00:00<00:00, 316kB/s]
{'accuracy': 0.8910550458715596}
```

```
# Getting predictions for the example sentence again, now that we have trained the model
print_example_predictions(example_sentence, model)
```

Print predictions for an example sentence

```
[[4.224632e-04 9.995776e-01]]
Example sentence:  this was by far the best movie of the year
Predicted logits:  [[-3.7340453  4.0349402]]
Predicted probabilities:  [[4.224632e-04 9.995776e-01]]
Prediction:  positive
```

A whole family of pre-trained encoders

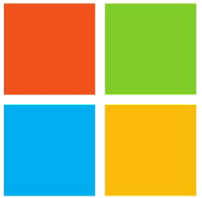
Many spin-off models

BERT

- Trained with masked language modelling and next sentence prediction
- First large pre-trained transformer model
- Outperformed everything before it

RoBERTa

- Got rid of next sentence prediction, optimized hyperparameters
- Trained on much more data than BERT
- Outperformed BERT



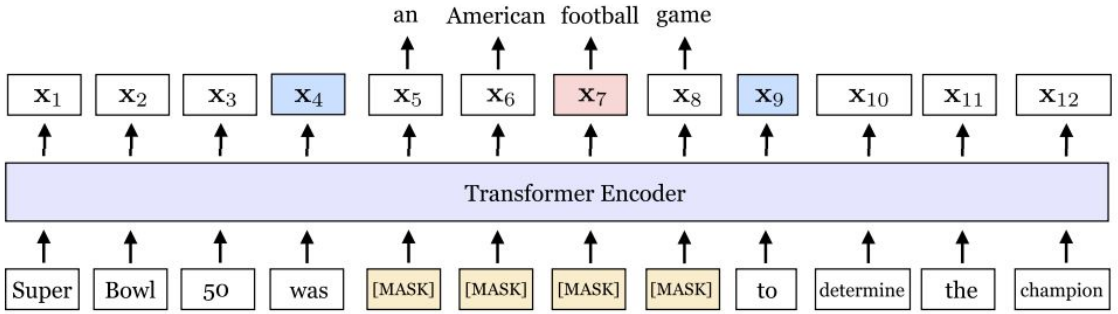
DeBERTa

- Focused on improvements to positional encodings
- Trained on much more data than RoBERTa
- Outperformed RoBERTa

SpanBERT

Instead of random tokens, mask contiguous tokens and predict what is behind the masked span.

Makes the task harder and the resulting model performs better.



	NewsQA	TriviaQA	SearchQA	HotpotQA	Natural Questions	Avg.
Google BERT	68.8	77.5	81.7	78.3	79.9	77.3
Our BERT	71.0	79.0	81.8	80.5	80.5	78.6
Our BERT-1seq	71.9	80.4	84.0	80.3	81.8	79.7
SpanBERT	**73.6**	**83.6**	**84.8**	**83.0**	**82.5**	**81.5**

Table 2: Performance (F1) on the five MRQA extractive question answering tasks.

Joshi, Mandar, et al. "Spanbert: Improving pre-training by representing and predicting spans." (2020)

DistilBERT and ALBERT

For some applications, we need to create smaller and faster models. For example, using distillation: training a small model to behave similarly to the bigger version, while keeping most of the benefit.

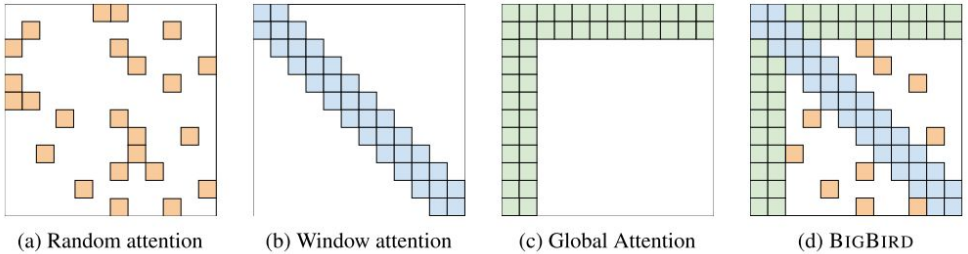
Table 1: **DistilBERT retains 97% of BERT performance.** Comparison on the dev sets of the GLUE benchmark. ELMo results as reported by the authors. BERT and DistilBERT results are the medians of 5 runs with different seeds.

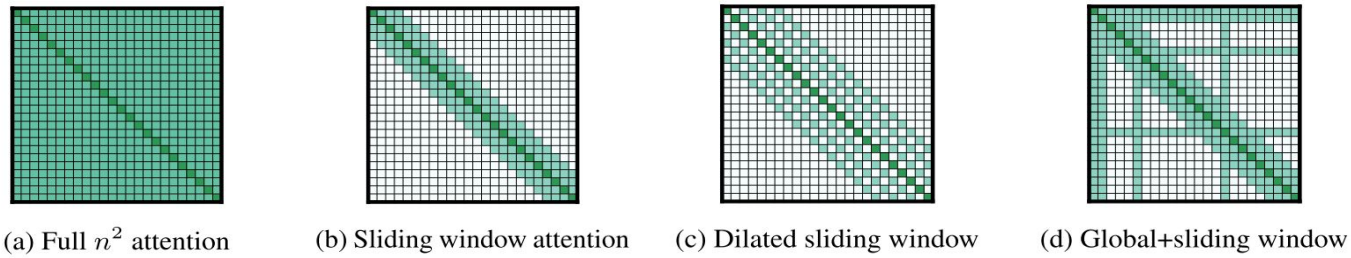
Model	**Score**	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI
ELMo	68.7	44.1	68.6	76.6	71.1	86.2	53.4	91.5	70.4	56.3
BERT-base	79.5	56.3	86.7	88.6	91.8	89.6	69.3	92.7	89.0	53.5
DistilBERT	77.0	51.3	82.2	87.5	89.2	88.5	59.9	91.3	86.9	56.3

Sanh, Victor, et al. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter." (2019)
ALBERT: A Lite BERT for Self-supervised Learning of Language Representations (2019)

Big Bird and LongFormer

Regular self-attention has $O(N^2)$ complexity, so doesn't scale very well. Models with sparse attention mechanisms have been proposed to extend the input length.





Zaheer, Manzil, et al. "Big bird: Transformers for longer sequences." (2020)
Beltagy, Iz, Matthew E. Peters, and Arman Cohan. "Longformer: The long-document transformer." (2020)

ClinicalBert, MedBert, PubMedBert, BEHRT

There are BERT-like models trained on data from specific domains or designed for specific applications. For example, models for the medical domain.

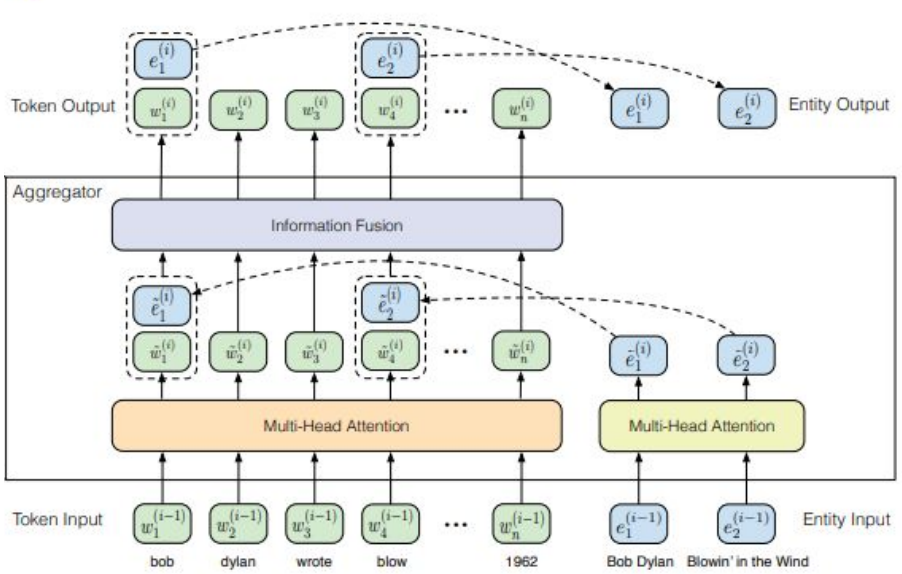
Huang, Kexin, J Altosaar, and R Ranganath. "Clinicalbert: Modeling clinical notes and predicting hospital readmission." (2019)
Li, Yikuan, et al. "BEHRT: transformer for electronic health records." (2020)

ERNIE

We have existing knowledge graphs and could take advantage of the information that they contain.

Given that entities are detected in text, ERNIE adds special entity embeddings into the transformer for additional information.

A separate multi-headed attention operates over the entity embeddings and then the information is combined with the main model in a fusion layer.



Zhang, Zhengyan, et al. "ERNIE: Enhanced language representation with informative entities." (2019)

Something for every language

There are versions of BERT trained for different languages.

German BERT - for German applications

CamemBERT - for French applications

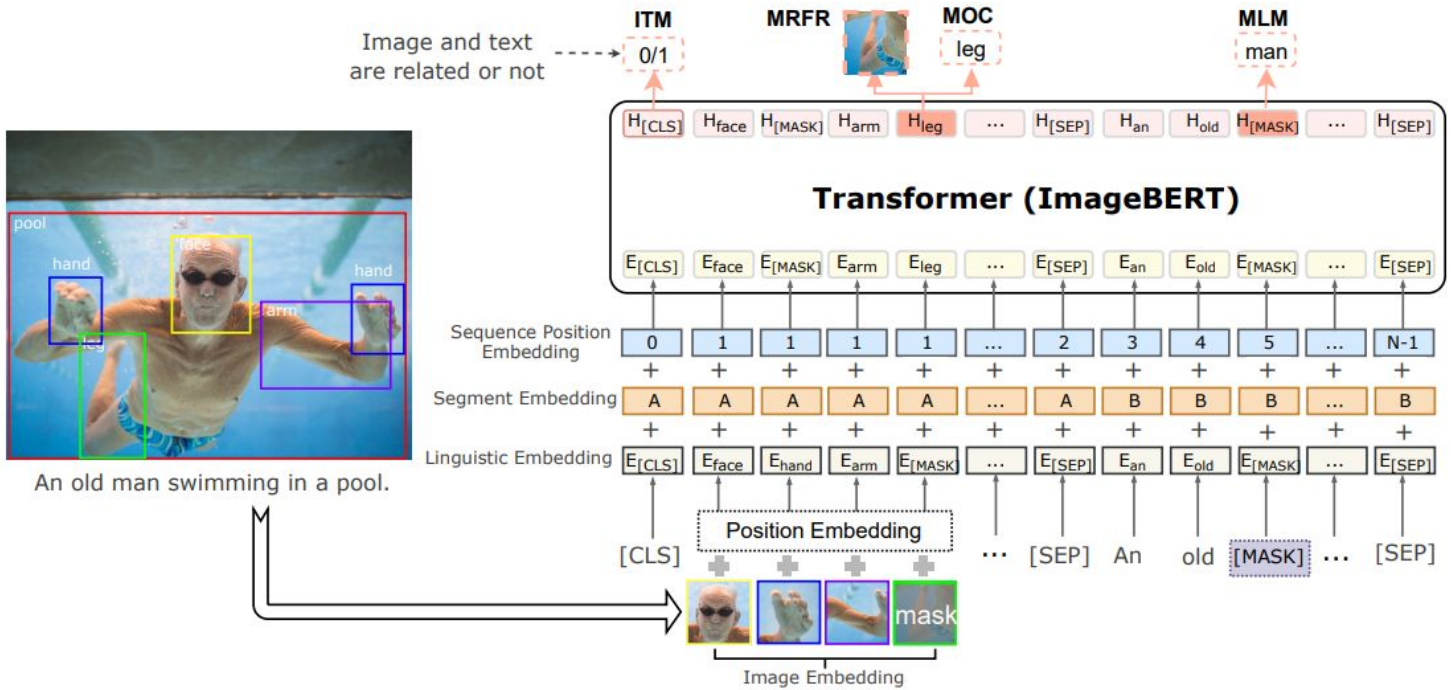
EstBERT - for Estonian applications

…

M-BERT - trained jointly on 104 different languages
https://github.com/google-research/bert/blob/master/multilingual.md

Multimodal models

Combining visual and textual information into the same transformer.
LXMERT, VisualBERT, ImageBERT, VilBERT



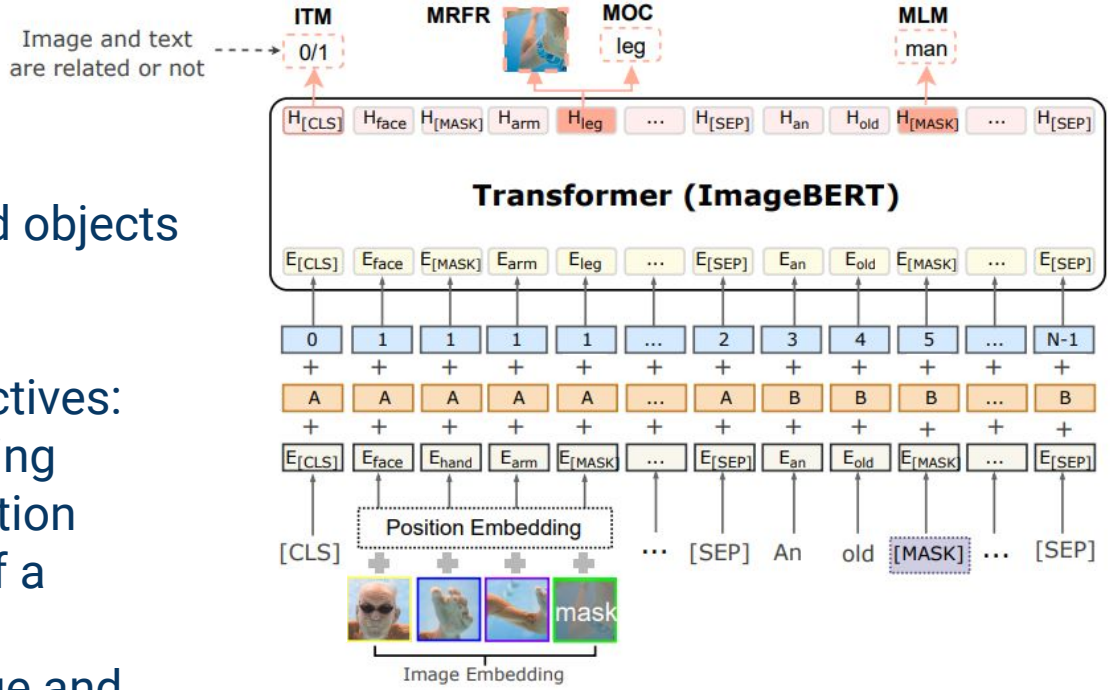
Qi et al. (2020) ImageBERT

Multimodal models: ImageBERT

ImageBERT encodes detected objects as additional tokens.

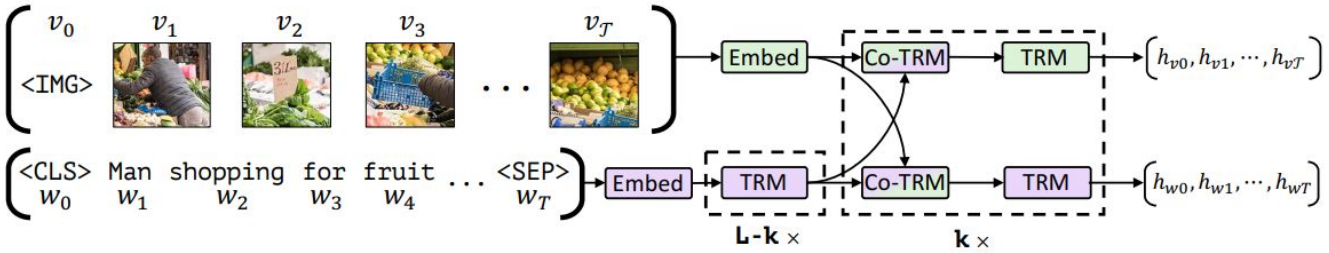
Uses 4 different training objectives:

- Masked language modeling
- Masked object classification
- Predicting the features of a masked object
- Classifying whether image and text are related



Qi et al. (2020) ImageBERT

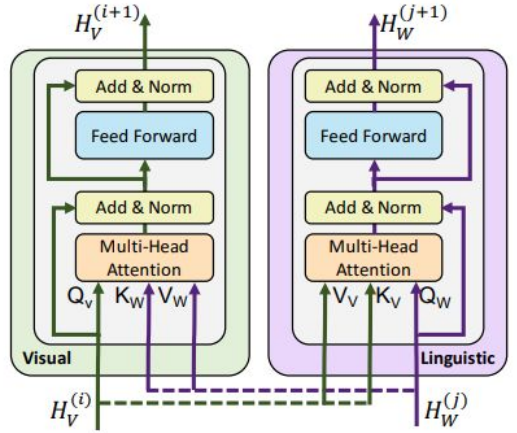
Multimodal models: ViLBERT



ViLBERT uses two parallel BERT-like encoders.

These interact using a co-attention module. Key & value matrices are received from the other modality.

Image features are extracted using pre-trained Faster R-CNN (object detection model)

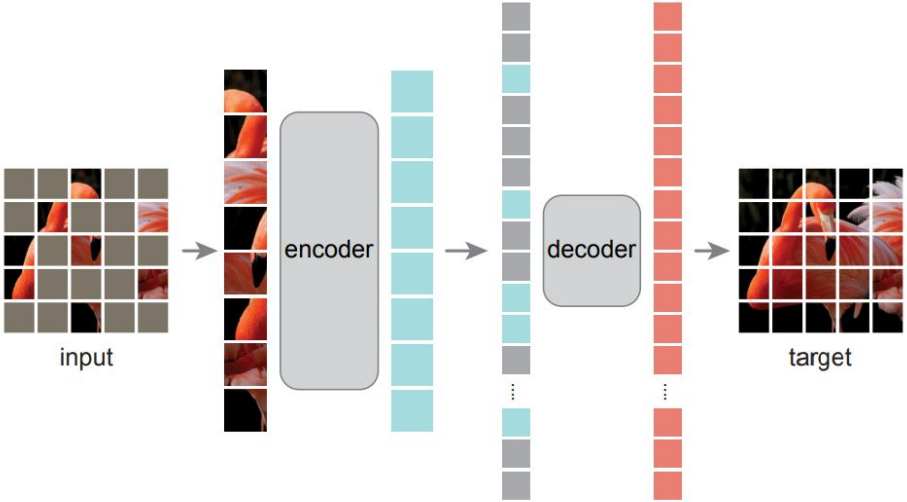


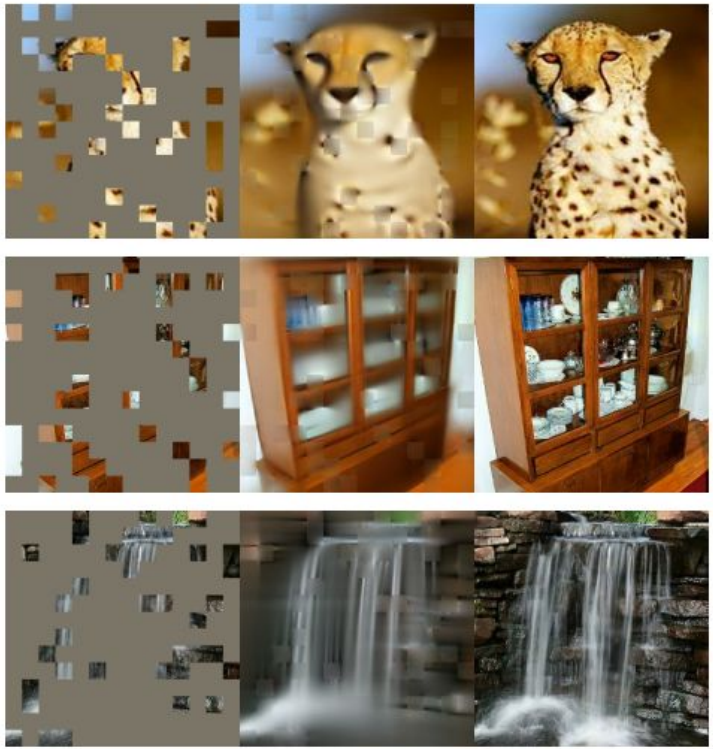
(b) Our co-attention transformer layer

Lu, Jiasen, et al. (2019). Vilbert

Masked image modelling

The idea of masked language modelling is being applied on other types of data as well.



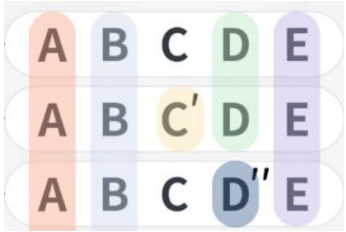


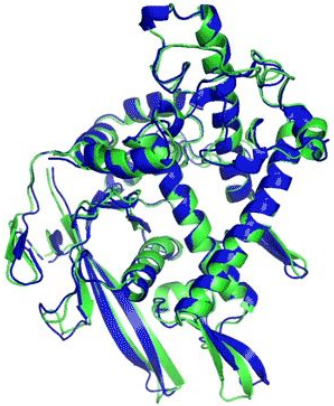
https://arxiv.org/pdf/2111.06377.pdf

Masked protein modelling

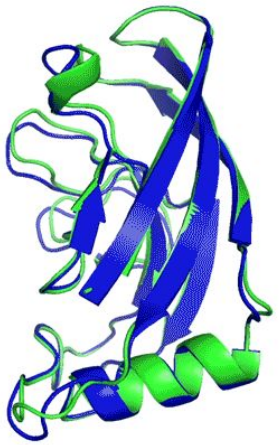
AlphaFold is a system that predicts the 3D structure of proteins based on their amino acid sequence.

As far as external researchers can tell, it uses a large pre-trained “language model” to represent the proteins, taking MSA (Multiple Sequence Alignment) as input.

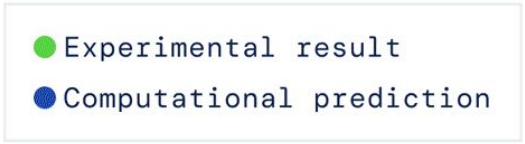




T1037 / 6vr4
90.7 GDT
(RNA polymerase domain)



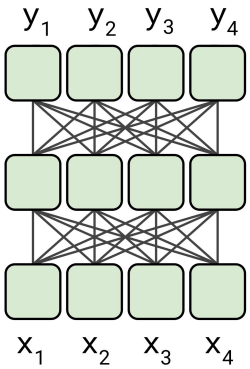
T1049 / 6y4f
93.3 GDT
(adhesin tip)

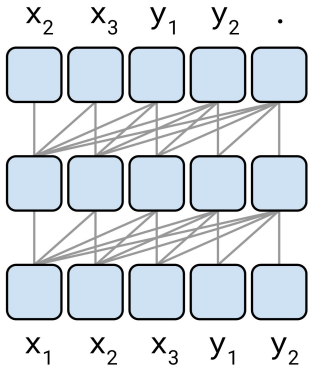


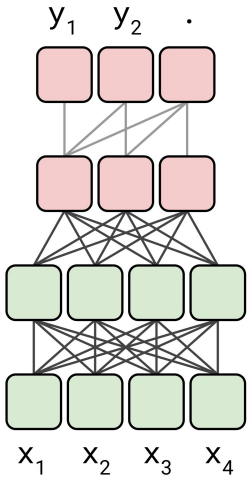
https://oecd.ai/fr/wonk/deepminds-alphafold-a-solution-to-challenge-of-protein-folding

Pre-training encoder-decoder models

Encoder-decoder models







Encoders: Are able to access the whole sequence, using context on both sides of each token.

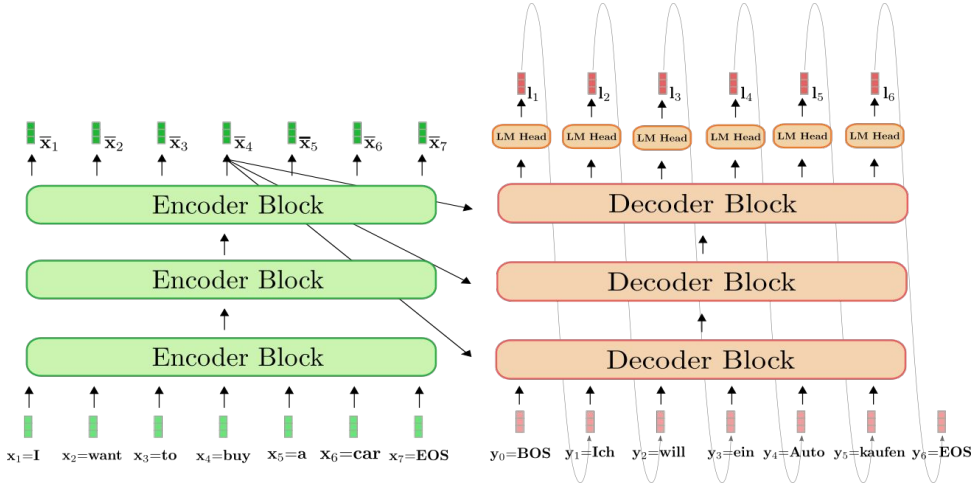
Decoders: Are able to access context on the left. Language models, good for generating text.

Encoder-decoder: Input is processed using an encoder, then output is generated using a decoder.

Encoder-decoder models

Initial input sequence can be fully attended over using the encoder, getting the maximum information.

Then, a dedicated decoder is used for generating the output sequence.



Particularly popular for machine translation: encoder and decoder can have separate vocabularies and focus on different languages.

https://huggingface.co/blog/encoder-decoder

Pre-training encoder-decoder models

How to pre-train? Can't really do Masked Language Modelling the same way any more, there isn't a direct correspondence between input and output tokens.

Some ideas:

Prefix language modeling

```
 Input: Thank you for inviting me
Target: to your party last week .
```

Sentence permutation deshuffling

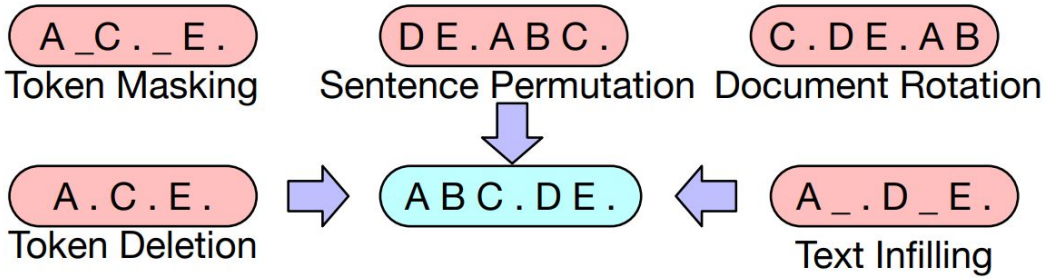
```
 Input: party me for your to . last fun you inviting week Thank
Target: Thank you for inviting me to your party last week .
```

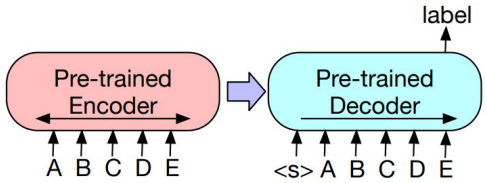
BERT-style token masking

```
 Input: Thank you MASK inviting MASK to your party apple week .
Target: Thank you for inviting me to your party last week .
```

Pre-training encoder-decoder models

We can corrupt the original sentence in various different ways, then optimise the model to reconstruct the original sentences. Good for generation tasks.
For example, the BART model:





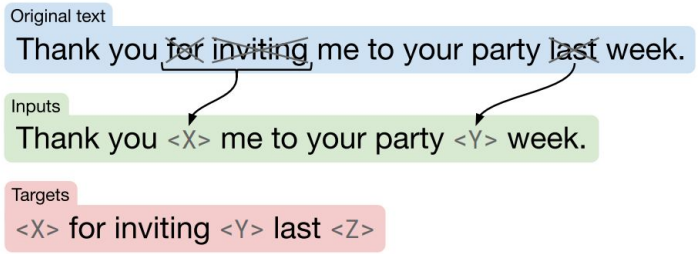
Encoder-decoder models can also be used for classification, by constructing a new output layer and connecting it to the last hidden state from the decoder.

Lewis, Mike, et al. "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension." (2019)

Pre-training encoder-decoder models

Replace corrupted spans

SpanBERT-like objective, adapted for decoding



Objective	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
Prefix language modeling	80.69	18.94	77.99	65.27	26.86	39.73	27.49
BERT-style	82.96	19.17	80.65	69.85	26.78	**40.03**	27.41
Deshuffling	73.17	18.59	67.61	58.47	26.11	39.30	25.62
Replace corrupted spans	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	39.82	**27.65**

Raffel, Colin, et al. "Exploring the limits of transfer learning with a unified text-to-text transformer." (2020)

Pre-training encoder-decoder models

Raffel et al. (2018) ran a thorough comparison of different hyperparameter choices. They found encoder-decoder models trained with span corruption to work better than regular decoders (just language models), at least for their tasks.

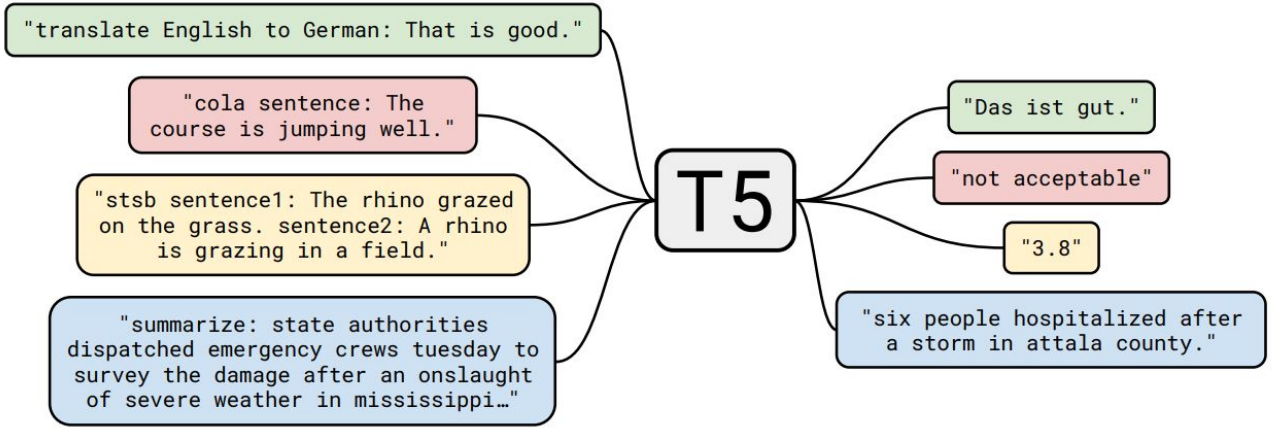
Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	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, Colin, et al. "Exploring the limits of transfer learning with a unified text-to-text transformer." (2020)

Instructional training

Encoder-decoder models can be pre-trained trained on different supervised tasks, by stating in the input what task the model should perform.

For example, T5 (Text-To-Text Transfer Transformer) is trained using the span corruption unsupervised objective, along with a number of different supervised tasks.

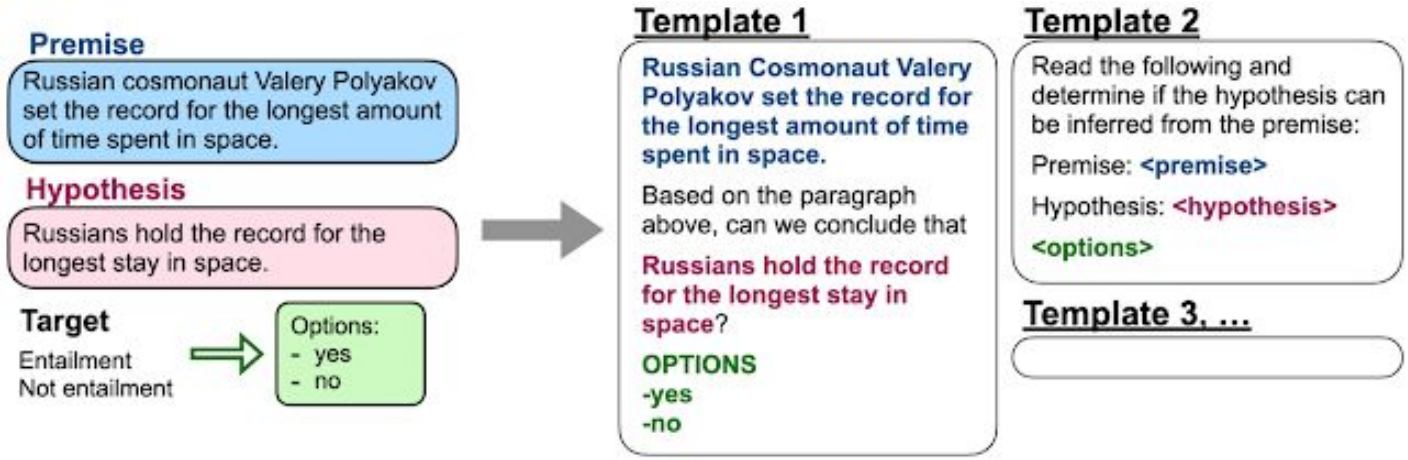


Raffel, Colin, et al. "Exploring the limits of transfer learning with a unified text-to-text transformer." (2020)

Instructional training

Prefixing the input with a particular string for a given task isn't very natural.

Instead existing datasets can be converted into more conversational-sounding instructions using templates.

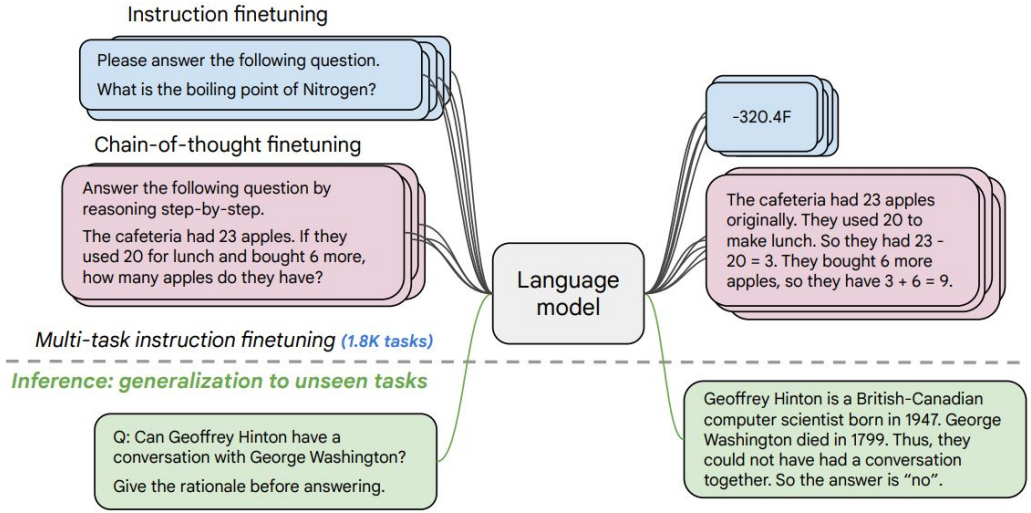


https://ai.googleblog.com/2021/10/introducing-flan-more-generalizable.html

Instructional training

We can then train with natural language instructions as inputs, and annotated answers as target outputs.

FLAN-T5 (Fine-tuned LAnguage Net) is the same size as T5 but trained on much more data, more languages and 1.8K tasks phrased as instructions.



Chung, Hyung Won, et al. "Scaling instruction-finetuned language models." (2022)

Instructional training

FLAN-T5 is freely available on Huggingface.

You can load it on Colab and use for generating text based on your instructions.

```
model = AutoModelForSeq2SeqLM.from_pretrained("google/flan-t5-large")
tokenizer = AutoTokenizer.from_pretrained("google/flan-t5-large")

inputs = tokenizer("A step by step recipe to make bolognese pasta:", return_tensors="pt")
outputs = model.generate(**inputs,
                         min_length=256,
                         max_length=512,
                         length_penalty=2,
                         num_beams=16,
                         no_repeat_ngram_size=2,
                         early_stopping=True)
print(tokenizer.batch_decode(outputs, skip_special_tokens=True))
```

https://github.com/marekrei/flan-t5_text_generation_example

Instructional training

FLAN-T5 is freely available on Huggingface.

You can load it on Colab and use for generating text based on your instructions.

Input: A step by step recipe to make bolognese pasta:

Output: Melt the butter in a large skillet over medium-high heat. Add the sausage and cook, stirring occasionally, until browned on all sides, about 5 minutes. Stir in the bolognese sauce and salt and pepper to taste. Pour the sauce over the pasta and toss to coat. Cover the skillet with cling wrap and refrigerate until ready to use, at least 1 hour. Serve topped with Parmesan shavings and parsley sprigs, if desired, and drizzle with extra-virgin olive oil and balsamic vinegar. Garnish with grated Parmigiano cheese and roasted red pepper flakes. Makes 1 1/2 lb (450 g) servings. Note: If you'd like, you can add 1 cup (240 ml) of ricotta cheese for an additional 2 tbsp. of cheese. You can also add 2 tablespoons of chopped chives, 1 teaspoon each of oregano and basil, 2 teaspoons thyme, 1/4 teaspoon kosher salt, 1/2 teaspoon freshly ground black pepper and 1/8 teaspoon ground nutmeg to the mixture and stir to combine.

https://github.com/marekrei/flan-t5_text_generation_example

Parameter-efficient fine-tuning

Parameter-efficient fine-tuning

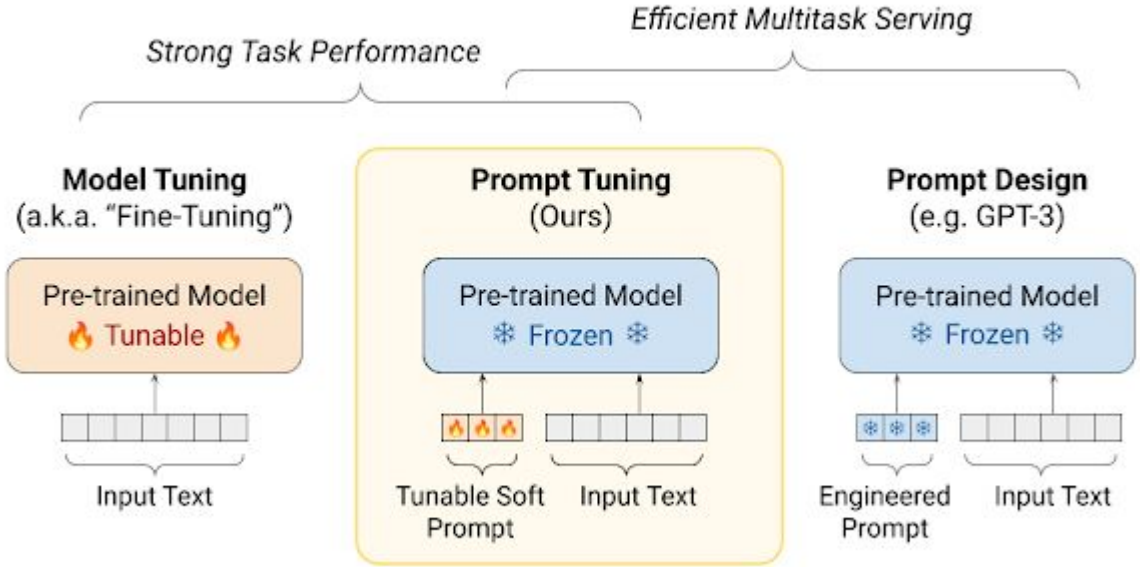
Models fine-tuned for one task are usually better at that particular task, compared to models trained to do many different tasks.

We don’t want to have thousands of different copies of huge models, each one trained for a slightly different task.

Instead: let’s keep most of the parameters the same (frozen) and fine-tune only some of them to be task-specific.

Prompt tuning

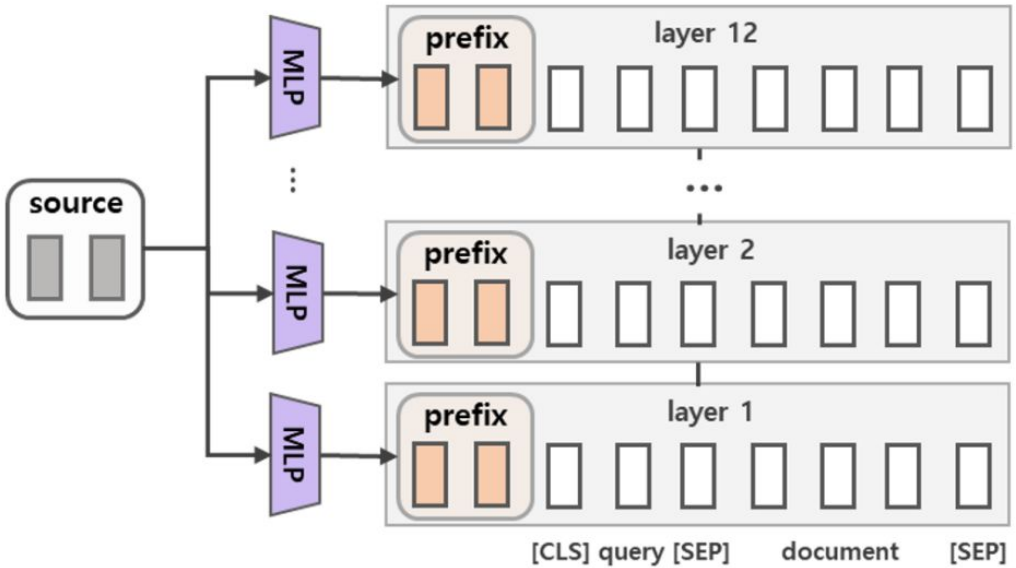
Include additional task-specific “tokens” in the input, then fine-tune only their embeddings for that particular task, while keeping the rest of the model frozen.



https://ai.googleblog.com/2022/02/guiding-frozen-language-models-with.html

Prefix tuning

In addition to the input, can include these trainable task-specific “tokens” into all layers of the transformer.



Li, Xiang Lisa, and Percy Liang. "Prefix-tuning: Optimizing continuous prompts for generation." (2021)
Image from https://arxiv.org/pdf/2110.14943.pdf

Control Prefixes

Training different prefixes for each property/attribute you want the output to have. For example, the domain or desired length of the text.

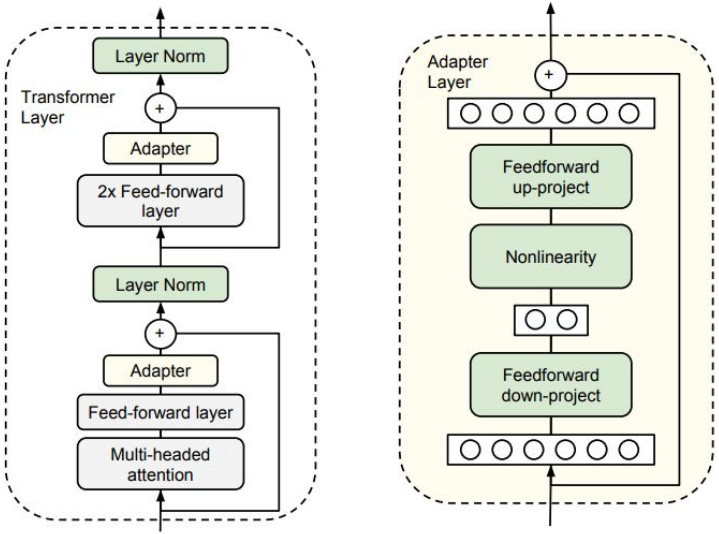
This was an MSc project at Imperial.

	ϕ%	DART			ϕ%	WebNLG			ϕ%	E2E Clean	
		BLEU	METEOR	TER ↓		S	U	A		BLEU	METEOR
T5-large fine-tuned	100	50.66	40	43	100	64.89	54.01	59.95	100	41.83	38.1
SOTA	100	50.66	40	43	100	65.82	56.01	61.44	100	43.6	39
Prefix-tuning	1.0	51.20	40.62	43.13	1.0	67.05	55.37	61.78	1.0	43.04	38.7
CONTROL PREFIXES (A_2)	1.1	**51.95**	**41.07**	**42.75**	1.0	66.99	55.56	61.83	1.0	**44.15**	**39.2**
CONTROL PREFIXES (A_1,A_2)	-	-	-	-	1.4	67.15	**56.41**	**62.27**	-	-	-

Clive, Jordan, Kris Cao, and Marek Rei. "Control prefixes for parameter-efficient text generation." 2022

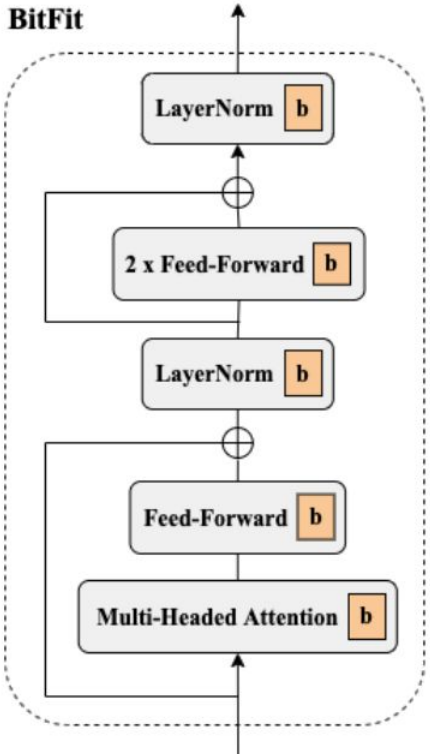
Adapters

Inserting specific trainable modules into different points of the transformer, while keeping the rest of the model frozen.



Houlsby, Neil, et al. "Parameter-efficient transfer learning for NLP.", 2019

BitFit



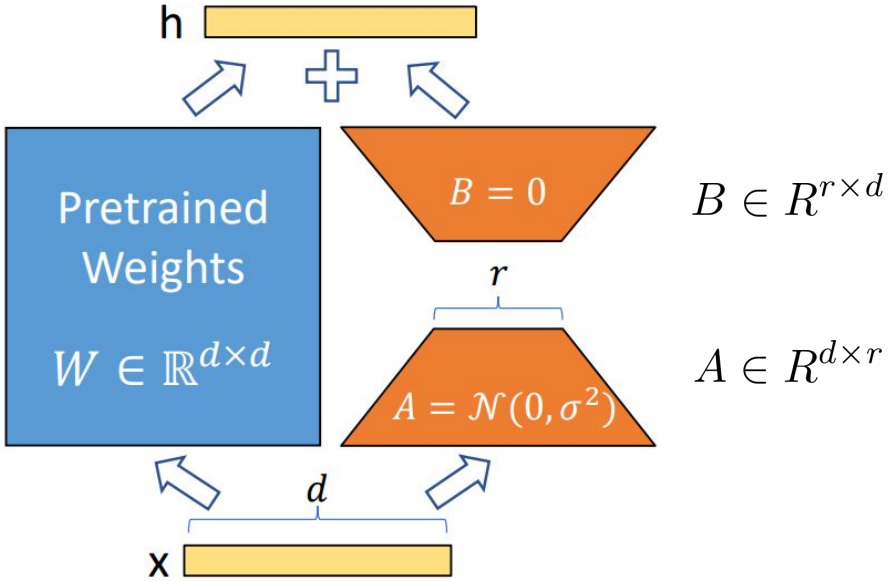
Keep most of the model parameters frozen, fine-tune only the biases.

Works surprisingly well, considering it only affects 0.08% of parameters.

Zaken, Elad Ben, Shauli Ravfogel, and Yoav Goldberg. "Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models." (2021)
Image from https://arxiv.org/pdf/2205.00305.pdf

Low-rank adaptation

Keep the main weights frozen but fine-tune two smaller matrices A and B such that the new weights are going to be W' = W + AB



Hu, Edward J., et al. "Lora: Low-rank adaptation of large language models." (2021)

The keys to good models of language

Transformer uprising

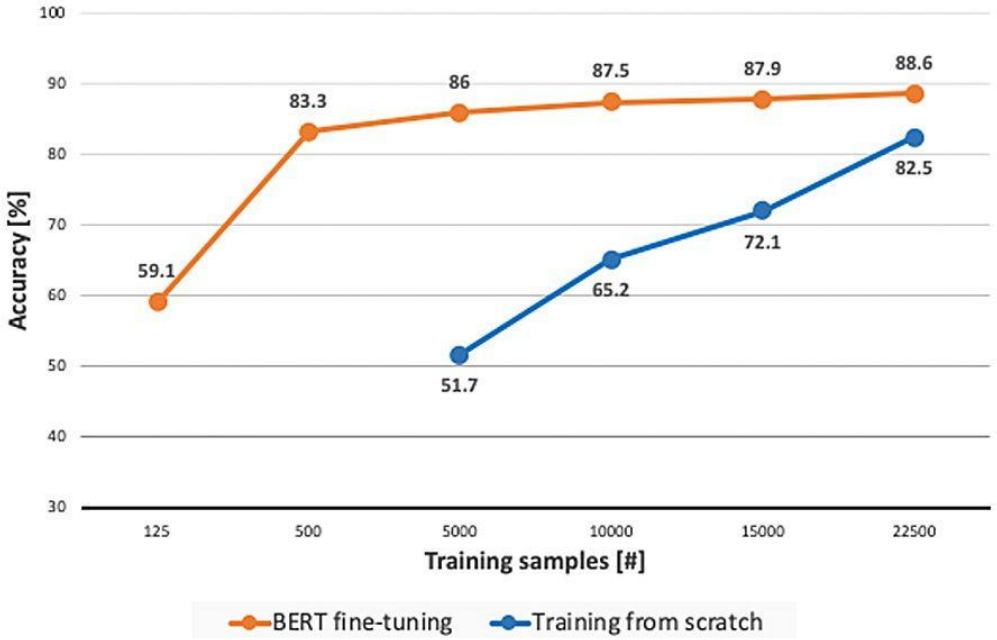
Pre-trained transformer models have been a revolution in NLP.

After only a couple of years it is difficult to find any model that doesn't use one of the pre-trained transformer models.

This is thanks to a few properties that can be applied to other ML tasks as well, beyond language.

1. Transfer learning (model pre-training)

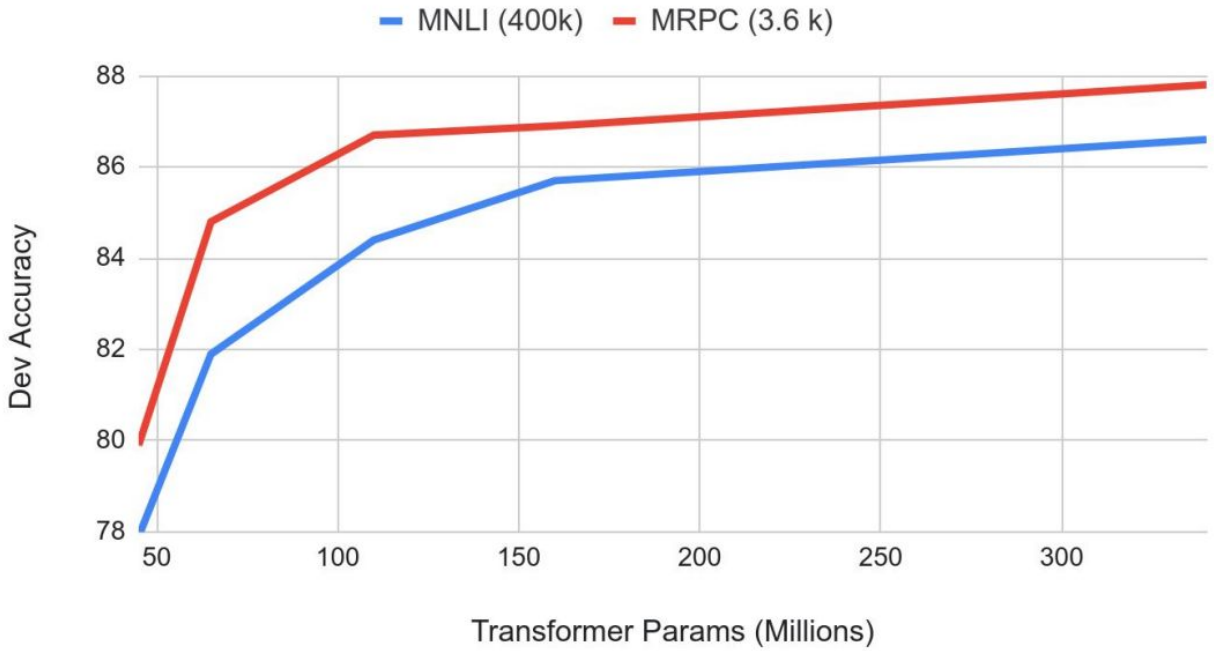
Pre-training the model gives us better performance even with fewer downstream training examples.



https://www.kdnuggets.com/2020/03/bert-fails-commercial-environments.html

2. Very large models

Training bigger models is giving better performance, although diminishing returns.

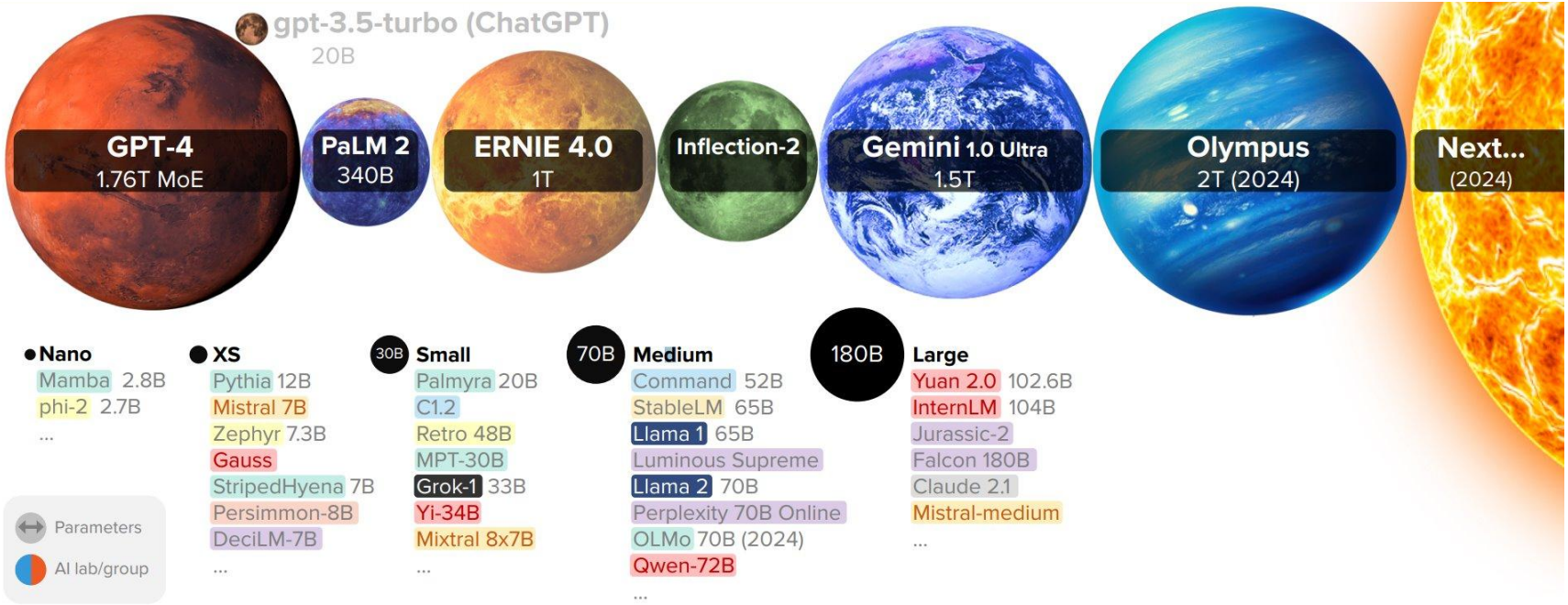


https://nlp.stanford.edu/seminar/details/jdevlin.pdf

2. Very large models

Still growing. GPT-4 is estimated to have 1.76 trillion parameters (OpenAI hasn't confirmed or denied).

For comparison, a human brain has ~100 trillion synapses.



Sizes linear to scale. Selected highlights only. All models are available. All models are Chinchilla-aligned (20:1 tokens:parameters) https://lifearchitect.ai/chinchilla/ All 200+ models: https://lifearchitect.ai/models-table/ Alan D. Thompson. 2023.

3. Loads of data

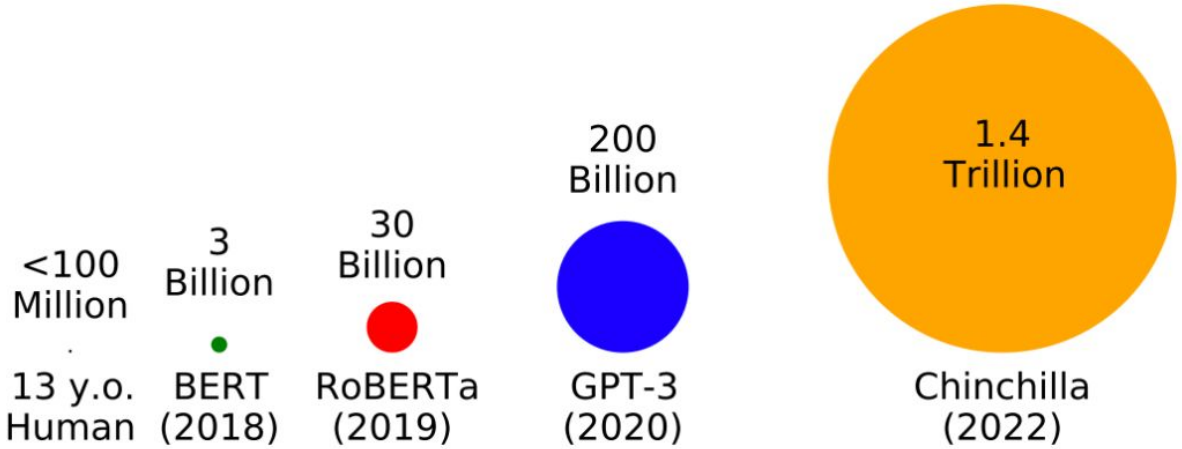
Huge amounts of training data.

Model name	Amount of training data
ELMo	800M tokens
BERT	3.3B words, 16GB
RoBERTa	160 GB, ~33B words
DeBERTa	78GB
T5	750GB
GPT-3	45TB

Only plausible using unsupervised learning objectives, with unlabeled data.

3. Loads of data

And the model training sets keep growing



Number of words seen during training

https://babylm.github.io/

4. Fast computation

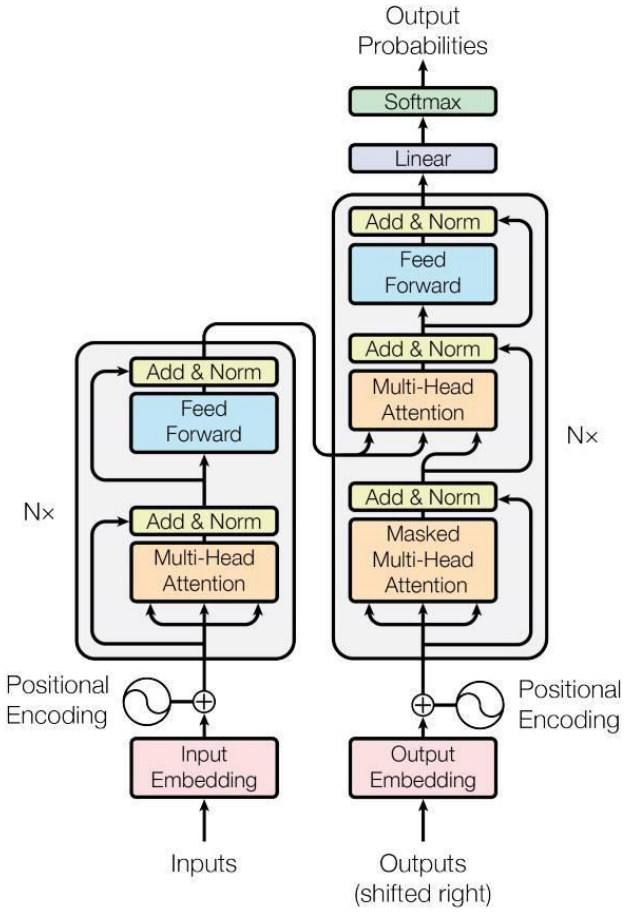
In order to process huge amounts of data, the models need to be fast.

Transformers are not particularly fast.

But they are fast for their size.

Representations for all the tokens in a sentence can be calculated in parallel. Particularly good for running on GPUs!

In contrast, RNNs and LSTMs would process each word in sequence.



https://pytorch.org/tutorials/beginner/transformer_tutorial.html

5. A difficult learning task

The prediction of missing words (MLM, span-based, etc) is a very difficult task:

- Tens of thousands of possible options to choose from.
  ~0.002% chance of guessing the correct word by accident.
- The model needs to take all types of language information into account
- The task is so difficult that even humans would have trouble

The model can’t just memorise the correct answers.