

DEPARTMENT OF COMPUTING
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

Pre-training Models

Includes topics such as contextual word representations, pre-training encoder models, BERT and relatives and pre-training encoder-decoder models.

Author: Anton Zhitomirsky

Architecture	Pre-trained model	Fine-tuned model	AI product
RNN	BERT	Sentiment classifier	ChatGPT
LSTM	RoBERTa	En-De translation system	Grammarly
ConvNet	GPT-3	Grammatical error detection system	Google Translate
Transformer	T5		Predictive keyboards

Contents

1 Contextual Word Representations	3
1.1 RNN	3
1.2 ELMo	3
1.2.1 Architecture	4
2 BERT	4
2.1 What is inside the transformer encoder?	5
2.1.1 Self-attention	5
2.1.2 Multi-head self-attention	6
2.1.3 Input embeddings	6
2.2 Training	6
2.2.1 Masked Language Modelling	6
2.2.2 Next sentence prediction	7
2.3 Variants	7
3 Putting pre-trained models to work	7
3.1 Performance	8
4 Text classification with BERT in practice	9
4.1 Worked Example	9

5 BERT Variations	11
5.1 BERT RoBERTa DeBERTa	11
5.2 SpanBERT	11
5.3 DistilBERT ALBERT	11
5.4 BigBird LongFormer	12
5.5 ClinicalBert MedBert PubMedBert BEHRT	12
5.6 ERNIE	12
5.7 Multi-Lingual Models	12
5.8 Multi-Modal Models	13
5.8.1 ImageBERT	13
5.8.2 ViLBERT	13
5.9 Masked image modelling	13
5.10 Masked protein modelling	14
6 Pre-training encoder-decoder models	14
6.1 Pipeline	14
6.2 Pre-training models	14
6.3 Instructional training	15
6.3.1 Example	15
7 Parameter-efficient fine-tuning	16
8 Pre-training decoder models	16
8.1 Learning Methods	16
8.1.1 Fine tuning	17
8.1.2 Zero-shot One-shot	17
8.1.3 Few-shot	18
8.2 Large language models for code	19
8.3 Large language models for dialogue	19
9 Advanced prompting and learning from human feedback	19
9.1 Chain Of Thought	19
9.1.1 Zero-shot chain of thought	20
9.2 Retrieval-based language models	20
9.3 Limitations of instruction fine-tuning	21
9.3.1 Reinforcement learning from human feedback	21
9.3.2 Training Pipeline	21

1 Contextual Word Representations

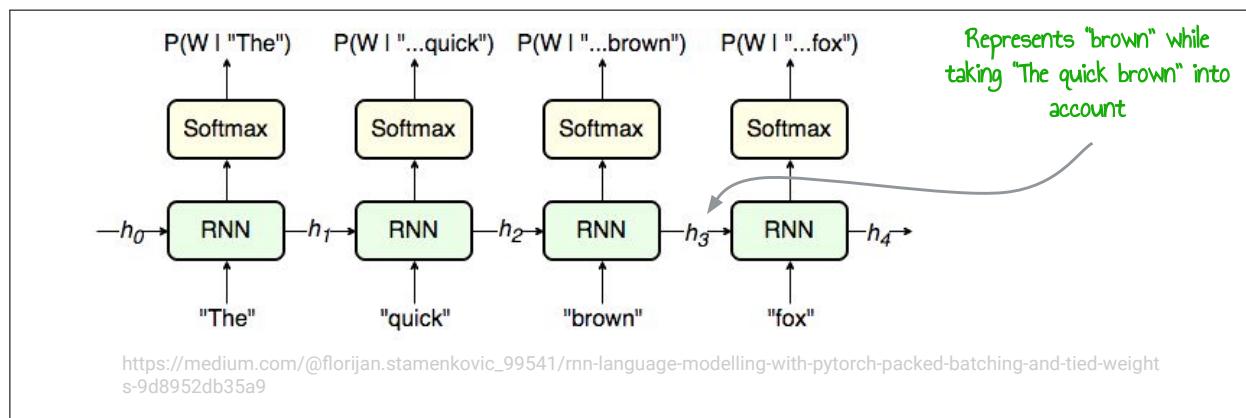
Word Embeddings allow us to learn vector representations for individual words, useful for applications such as representing similar words with similar vectors, synonym replacement, word classification and so on. However, their shortcoming is that each word has only one embedding, and often times the meaning of a word depends on the context.

For instance, an example:

I deposited some money in the *bank* VS I was camping on the east *bank* of the river.

Here, having a single vector for every word doesn't cut it since we need to take the context into account when constructing word representations.

1.1 RNN



One way, is to use a model like an RNN; by going through the text in sequence and building up representations of the text, you can use them for word representations as well. Internally, it will learn to encode any given context into a vector.

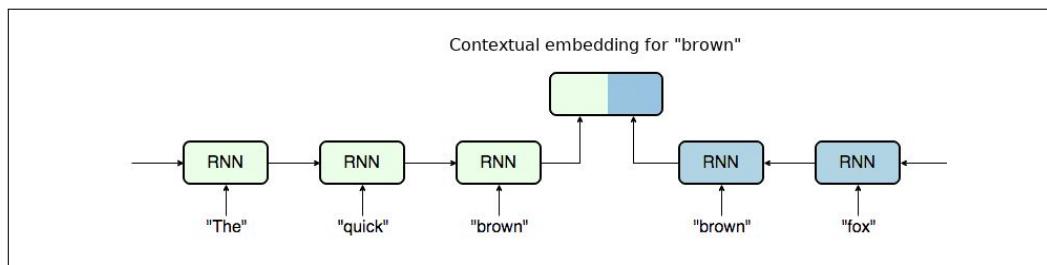
1.2 ELMo

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

1. One recurrent language model going forward, left-to-right.
- $$p(t_1, t_2, \dots, t_N) = \prod_{k=1}^N p(t_k | t_1, t_2, \dots, t_{k-1})$$
2. A second recurrent language model going backward, right-to-left.
- $$p(t_1, t_2, \dots, t_N) = \prod_{k=1}^N p(t_k | t_{k+1}, t_{k+2}, \dots, t_N)$$

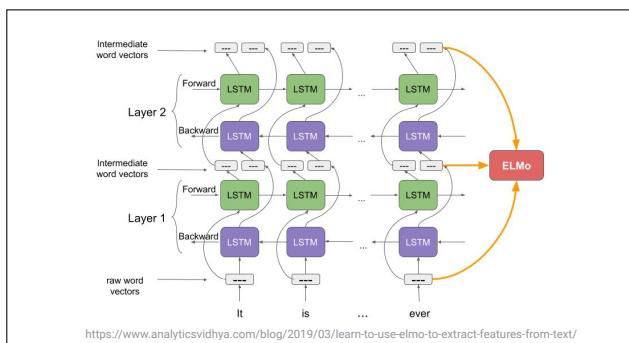
Embeddings from Language Models:

- One model tries to predict the next word and another tries to predict the previous word in the sentence.
- The two are trained on large volumes of text.



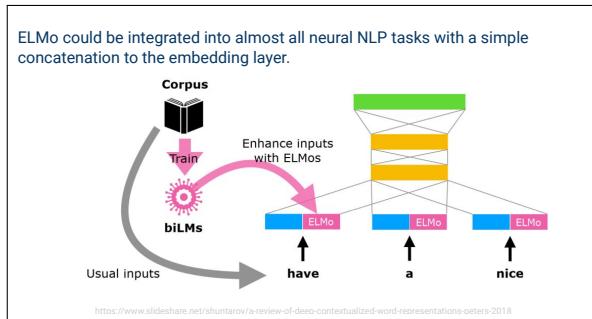
The idea is to take the embeddings from both directions and combine them into one vector representation

1.2.1 Architecture

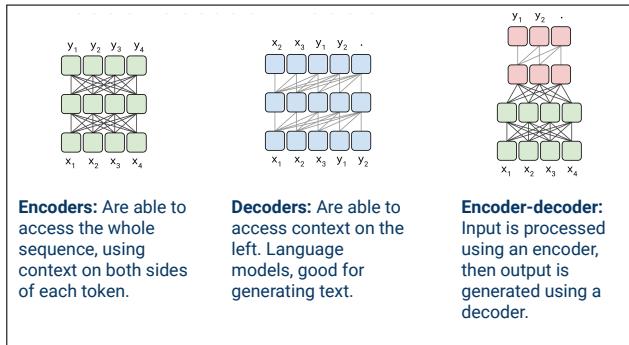


- ELMo uses LSTMs as opposed to regular rnns.
- They have multiple layers of LSTMs
- If a sentence is given, it is embedded and fed into a bidirectional LSTM, which then gets fed into the next layer again.
- The final vector gets combined from all the layers.

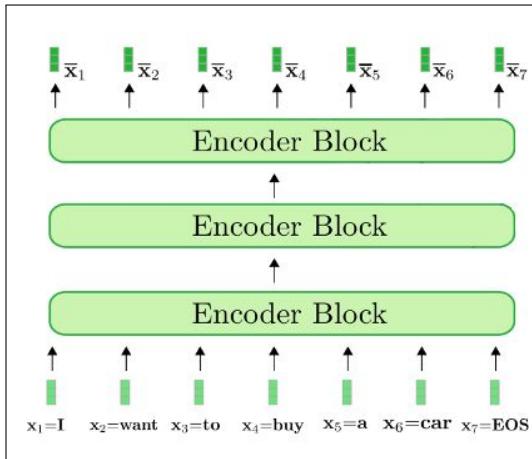
Having multiple layers allows for more complex ‘second-level’ reasoning about words.



2 BERT



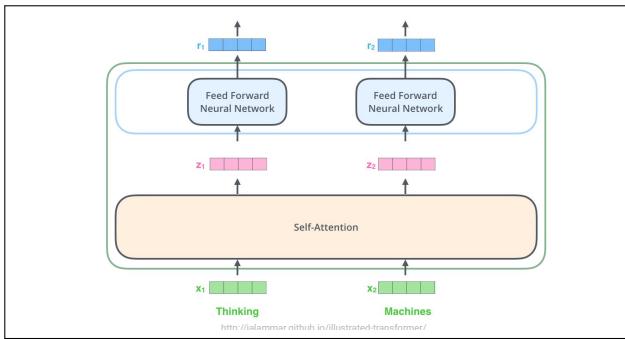
- ELMo was the first important model of this type
- Nowadays, we have three different types of models.
- Decoders are efficient: the bounding of reading to the left makes it more technically efficient than an encoder; each time you generate a new word you don't want to recalculate the context representations for the whole context. Instead, you attend only to the left.



Bidirectional Encoder Representations from Transformers

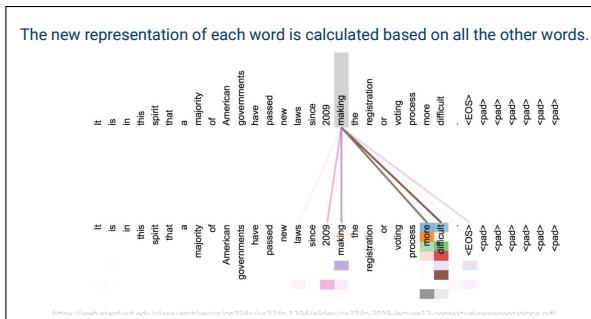
- Takes in a sequence of tokens, gives as output a vector for each token. At the output it produces the same number of representations as tokens in the input.
- Builds on previous word (ELMo) and combines some good ideas and scales it up further.

2.1 What is inside the transformer encoder?

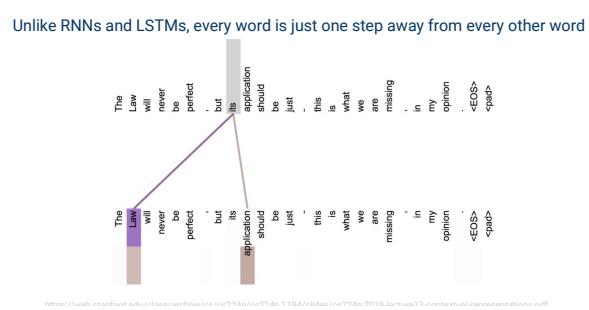


- A transformer cell/layer takes words as input through the bottom.
- This cell is repeated as many times as we like.

2.1.1 Self-attention



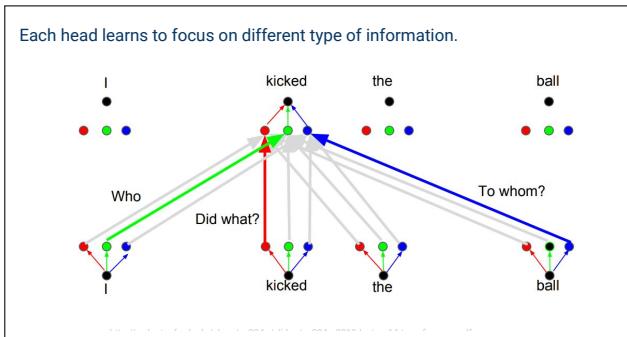
(j) Bert uses self attention — every word in the sequence (k) Advantage from self-attention is that every word is calculates new representations based on all the other just one hop away from every other word.



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

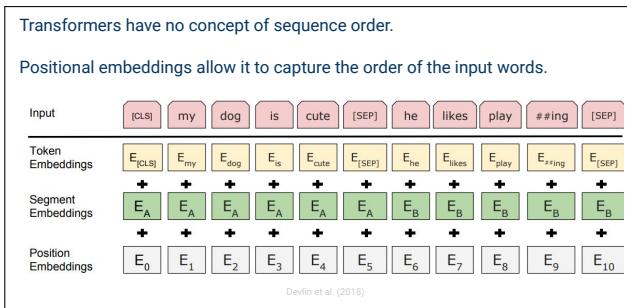
- (j) Bert uses self attention — every word in the sequence (k) Advantage from self-attention is that every word is calculates new representations based on all the other just one hop away from every other word. In LSTMs or RNNs, if we wanted to get information from one word to another, we'd have to step through every word in the sequence and keep information in memory. Unlike Self-attention which combines directly information from every other word.

2.1.2 Multi-head self-attention



- Each head is able to specialize to a particular task
- e.g. one head might be doing who, another, did what, another to whom. (we don't know, we let it learn whatever it needs to)
- Then, all this is combined into one representation.

2.1.3 Input embeddings

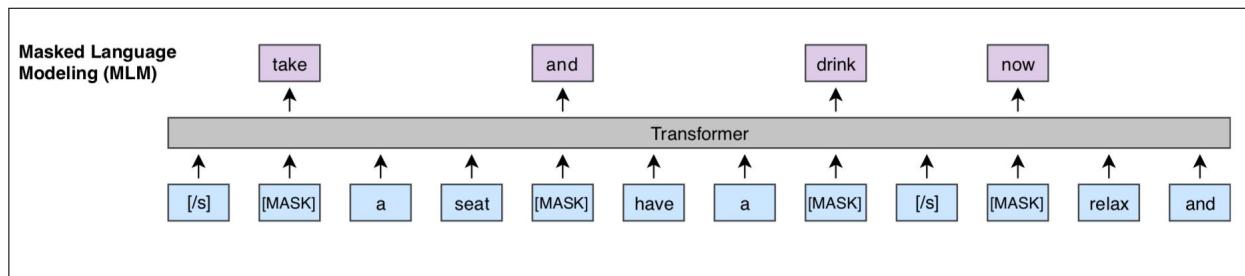


- We tokenize an input sentence
- **Token Embeddings:** normal learnable embeddings learnt during the training of the BERT model
- **Segment Embeddings:** shows the model where in the input a word is, specifically, which section.
- **Position Embeddings:** positional predefined sinusoidal embeddings or learnable embeddings.

2.2 Training

2.2.1 Masked Language Modelling

How do we train our model? We want it to have as much training data as possible so that it learns as much as it can. For that, we would want to use just plain text because annotated data is very expensive to obtain. Furthermore, there is not that much available versus all the data on the internet that you can crawl.



Here, the language model learns to predict missing words. We hide $k\%$ of the input words behind a mask and train the model to predict them. Since BERT returns a representation for each token, we only train those parts for which a word is absent.

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 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.

- When we train these models we want the correct amount of masking.
- When we have too little masking, the model needs to consider the entire scope of the document to make a prediction.
- if we have too much masking, then we may just lack enough information to fill in the gap correctly.

The last two bullet points are curious. We replace 10% with random words because in basic training it won't ever have any incorrect words. This trains the model to calculate correct representations regardless of the context which has varying truth. Then, 10% is left as the original but also predicted because if we train the model to only predict correct words where there is a mask token, then the model will learn to only predict in those places; in practice we don't have any masks.

2.2.2 Next sentence prediction

BERT used a second pre-training objective. It is given two sentences, and it tries to predict whether they appeared in the original order in the source text. However, this didn't add much benefit so it hasn't been used in newer versions.

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

2.3 Variants

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)

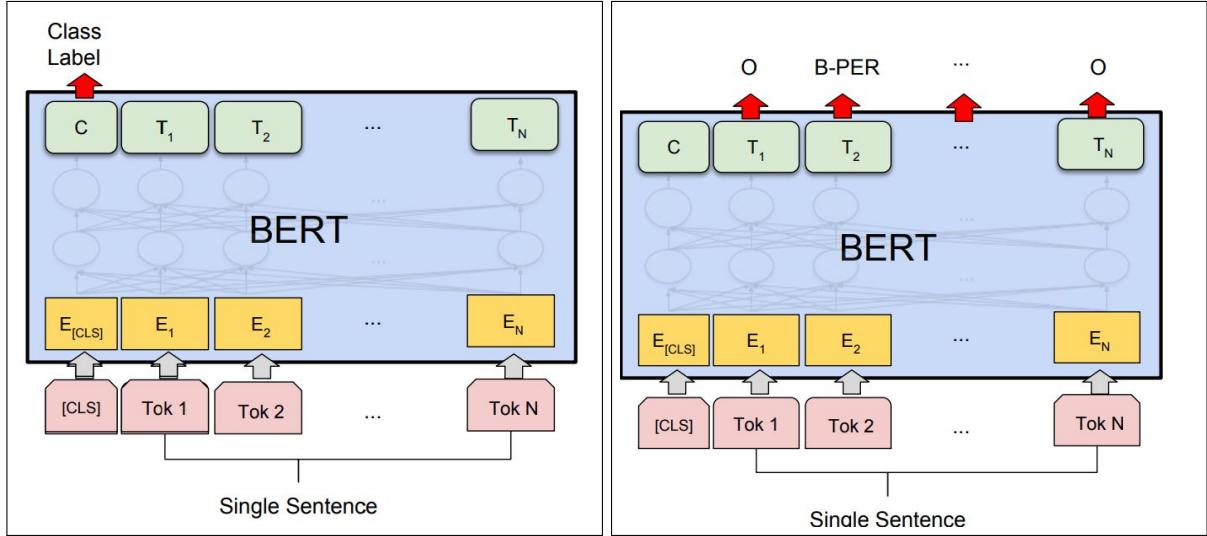
Pre-trained with 64 TPU chips for a total of 4 days

3 Putting pre-trained models to work

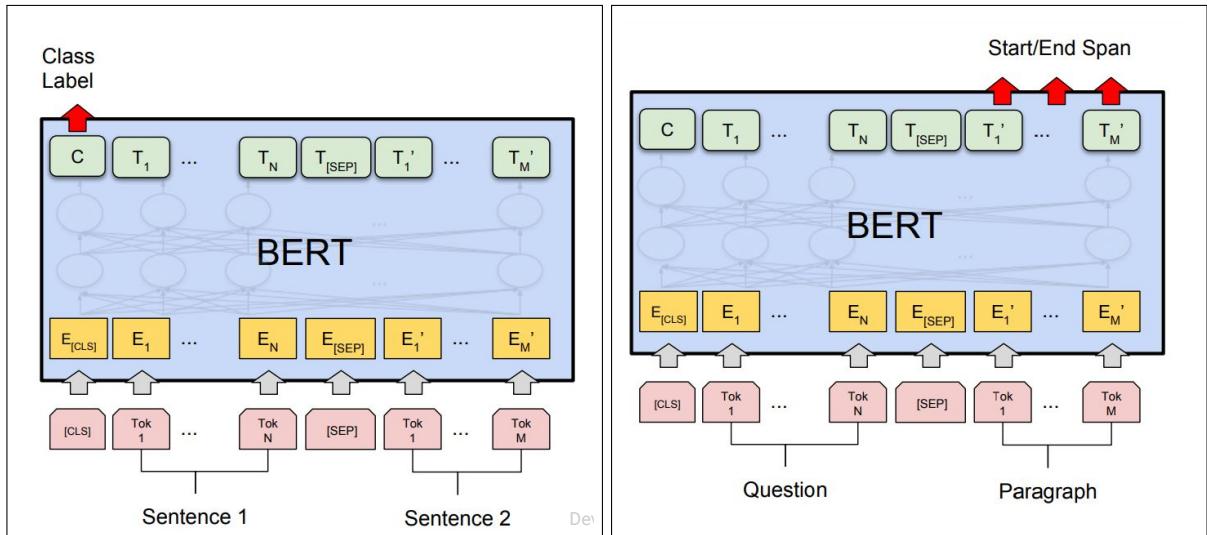
BERT-like models give us a representation vector for every input token. We just have to chop off the masked LM head, it is no longer needed.

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

1. Freeze BERT, use it to calculate informative representation vectors. Train another ML model that uses these vectors as input.
2. (more commonly) 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)



(r) For **sentence classification**, give the language to (s) **Token Labelling**: assigning labels to individual tokens, do the same, just put classifications pairs on each token. If BERT produces tokens for every token, we attach our token layer to the CLS token so that the model produces good text representations as that token's representation; **Add a special token in the input that represents the whole sentence**



(t) **Pair Classification**: we separate two sentences with a token. We put a classification head on top of the class token and train the model to classify ‘does sentence 1 entail sentence 2 or vice versa?’

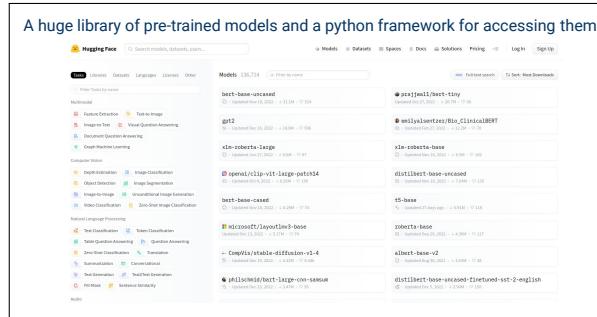
(u) **Question Answering**: We can structure the task with an input, and a paragraph of text which may contain some answers. Then train the model to lable individual tokens to indicate which tokens are the answer, then simply extract these.

3.1 Performance

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
BERT _{BASE}	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT _{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

4 Text classification with BERT in practice



hugging face is a website with many pre-trained models

4.1 Worked Example

This example lives on [Marke's Github](#)

```

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

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 the model to training mode
model.train() ← Activating training mode (activates dropout, batch norm, etc)

# 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)

```

```

# Setting the model to evaluation mode
model.eval() ← Setting the model to eval mode (turning off dropout, batchnorm, etc)

# 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()) ← 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

```

```

# Text classification example with BERT
# Created by Marek Rei
# Based on https://colab.research.google.com/github/huggingface/notebooks/blob/master/courses/transfer-learning/text-classification.ipynb
# 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) ← Tokenizer directly from the Huggingface model

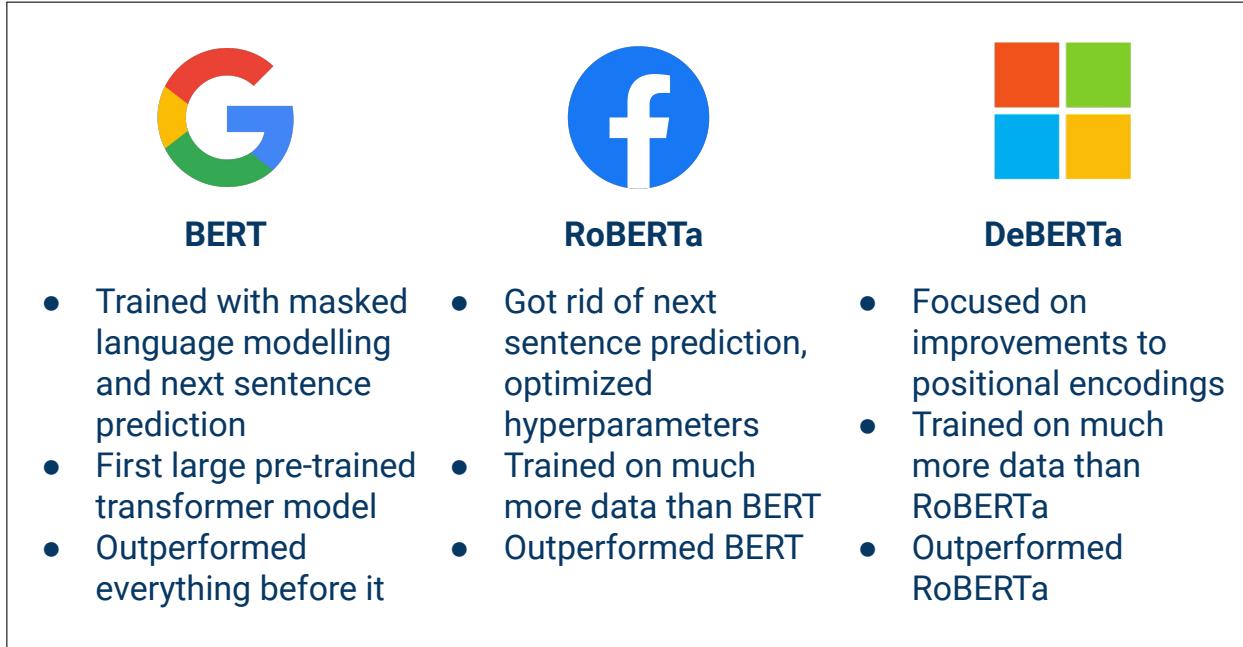
def tokenize_function(example):
    return tokenizer(example["sentence"], truncation=True)

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

```

5 BERT Variations

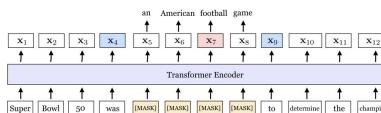
5.1 BERT | RoBERTa | DeBERTa



5.2 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.



Transformer Encoder

	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)

- They make the task more difficult, and mask out sequences of words. Gives models more incentive to learn more detailed representations.

5.3 DistilBERT | 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.

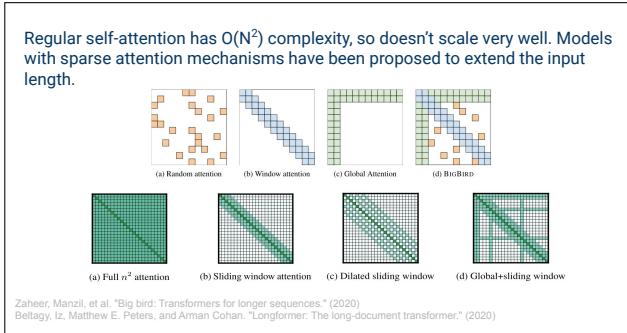
Table 1: DistilBERT retains 97% of BERT performance. Comparison on the dev sets of the GLUE benchmark. ELMo results as reported by the authors. BERT and DistilBERT results are the medians of 5 runs with different seeds.

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)

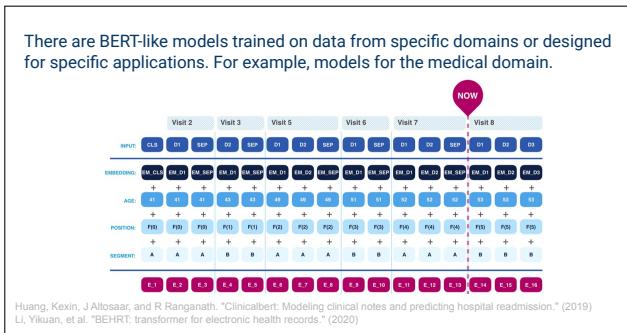
- Sometime models require a lot of resources, and these models return results quickly. E.g. DistilBERT trains a smaller transformer model on the output of the bigger model.

5.4 BigBird | LongFormer



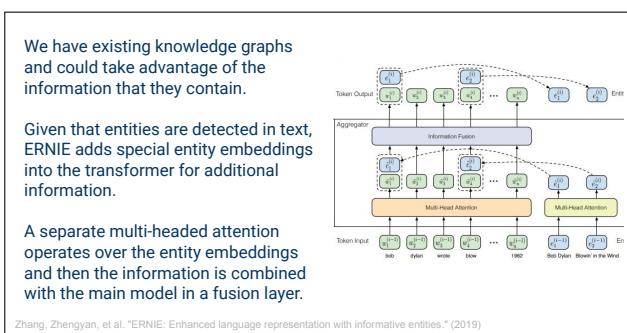
- Sometimes we need more context. These encode longer text.
 - This model attends to some words based on a pattern, e.g. randomly, nearby, or global (some words attend to all, and some don't)

5.5 ClinicalBert | MedBert | PubMedBert | BEHRT



- Domain specific BERTs also exist.
 - There is a specific embedding for the age of the patient, and the segment embeddings for hospital visits.

5.6 ERNIE



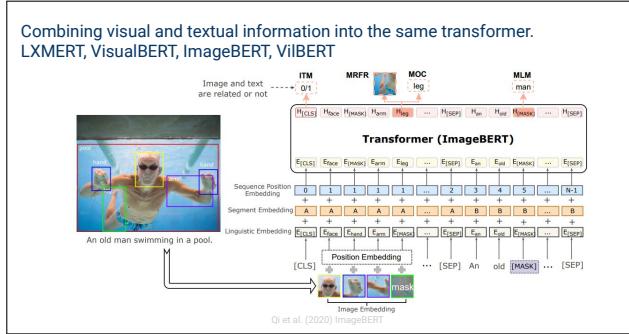
- “We can have databases and knowledge graphs that contain a lot of factual information and it would be useful to include those in language models as well. So there are models that have been proposed to include this information in BERT by retrieving relevant information from a knowledge graph, embedding it, and then connecting it directly into the embeddings of BERT.”

5.7 Multi-Lingual Models

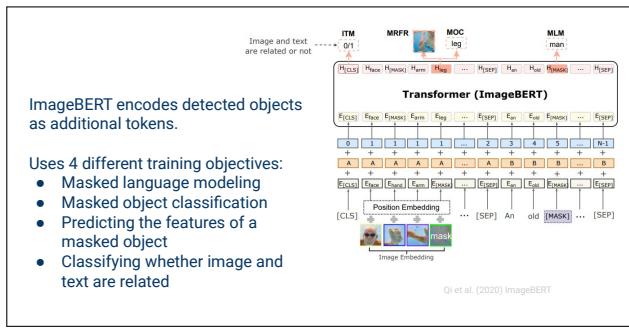


- There are also BERTs for every different language
 - M-BERT is multilingual

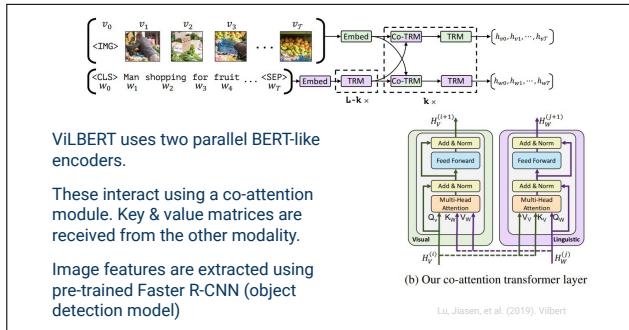
5.8 Multi-Modal Models



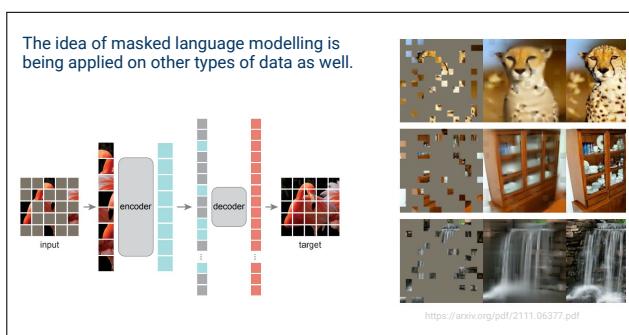
5.8.1 ImageBERT



5.8.2 ViLBERT



5.9 Masked image modelling



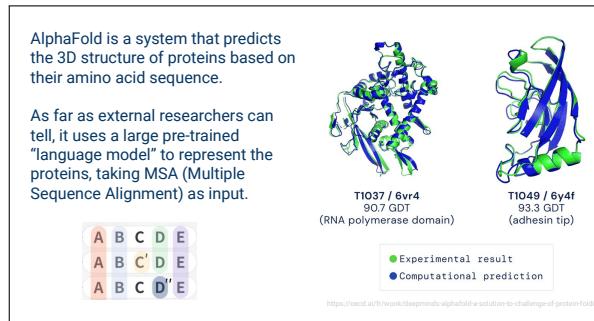
- Where the model can process information both from the visual and textual domain.

- It gets the input text and image, and extracts these little areas from the image that represent different objects. Then encodes these into vectors and maps them into the same space as the textual tokens.
- Then we can have multiple different objectives for this task for training.

- Contains cross attention models

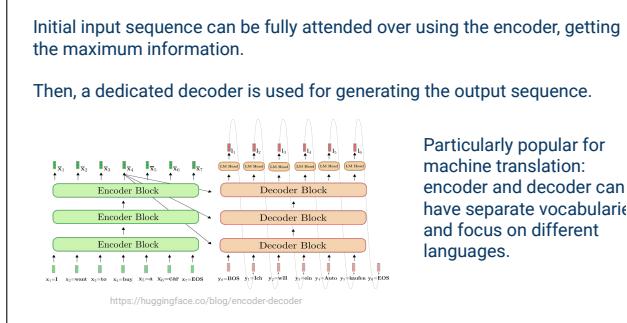
- mask different parts of the image and apply mask fixes onto the image.

5.10 Masked protein modelling



6 Pre-training encoder-decoder models

6.1 Pipeline



- We have an encoded part and decoded part
- For the encoder, we have the familiar BERT strategy of giving the whole text as input with self-attention over all tokens in all possible directions
- We then get these vector representations out for each token which are passed to the decoder block and the decoder block generates words one word at a time
- It then puts that word back into its own input and encodes that as the context and tries to predict the next word in the sequence.

This architecture is particularly useful in machine translation. Another advantage is that encoder and decoder can have separate vocabularies which abstracts the machine translation task better.

6.2 Pre-training 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 .

Replace corrupted spans
SpanBERT-like objective, adapted for decoding

Original text: Thank you for inviting me to your party last week.

Input: Thank you <X> me to your party <Y> week.

Target: <X> for inviting <Y> last <Z>

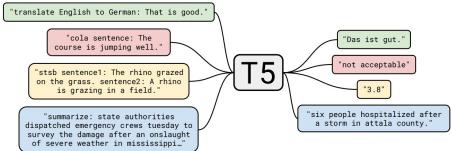
Objective	GLUE	CNNDM	SQuAD	SGLU	EnDe	EnFr	EnRo
Prefix language modeling	80.69	18.94	77.99	65.27	26.86	39.73	27.49
BERT-style	82.96	19.17	80.65	69.85	26.78	40.03	27.41
Deshuffling	73.17	18.59	67.61	58.47	26.11	39.30	25.62
Replace corrupted spans	83.28	19.24	80.88	71.36	26.98	39.82	27.65

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

6.3 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)

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

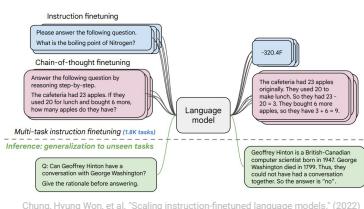
Instead existing datasets can be converted into more conversational-sounding instructions using templates.



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

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)

6.3.1 Example

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

- Take annotated datasets and phrase it to look like a sequence to sequence translation task.

- The follow-up work creates more human sounding templates.

- Trained on more data and more tasks.

7 Parameter-efficient fine-tuning

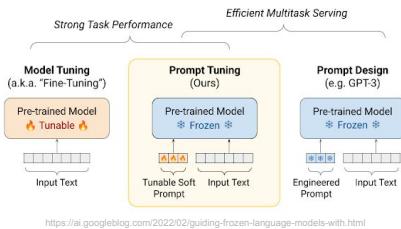
Models fine-tuned for one task are usually better at that particular task, compared to models trained to do many different tasks.

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.

- Many pre-trained models are large. If we want to get the best performance on a task we have to pre-train it.
- If we have many different tasks, we don't want 1000s of distinct models solving one specific task.

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.



THE REST IS NOT IN THE EXAM

8 Pre-training decoder models

Language Models

- They don't have an explicit encoder; Everything happens together in the decoder.
- Using efficient attention for generating one word at a time.
- We can train on unlabeled text, optimizing

$$p_{\theta}(w_t | w_{1:t-1}) \quad (8.0.1)$$

- Great for tasks where the output has the same vocabulary as the pre-training data.

For example: dialogue systems, summarization, simplification, etc

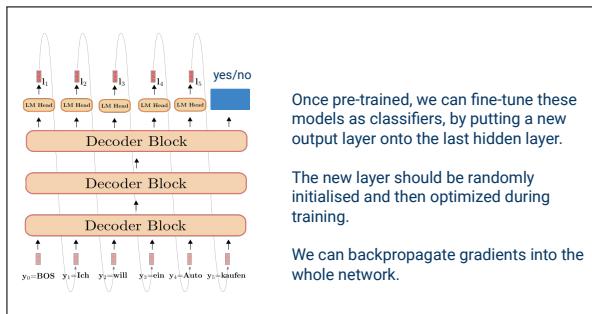
8.1 Learning Methods

There are alternative ways of using pre-trained decoders

8.1.1 Fine tuning



- optimize model parameters
- Train the model on an input output pair (our dataset) and perform gradient descent, or cut out the top layer and put a specialized layer on top.



The original GPT (Radford et al. 2018) performed generative pre-training of the decoder but then was fine-tuned as a discriminative classifier.

Natural Language Inference:
Label pairs of sentences as entailing/contradictory/neutral

Premise: The man is in the doorway
Hypothesis: The person is near the door

This input is formatted as a sequence of tokens for the decoder:

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

The linear classifier is applied to the representation of the [EXTRACT] token.

<https://web.stanford.edu/class/cs224n/2023-lecture9-pretraining.pdf>

GPT was a big step in the area of pretrained decoders

- Transformer decoder with 12 layers, 117M parameters.
- 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
- Byte-pair encoding with 40,000 merges
- Trained on BooksCorpus: over 7000 unique books.
- Contains long spans of contiguous text, for learning long-distance dependencies.

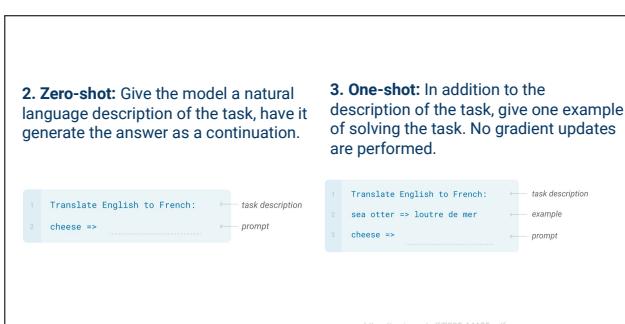
This pre-training, followed by fine-tuning of the decoder provided large improvements on tasks such as NLI and question answering.

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

https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf

With BERT we can look at the text in every direction, that's why we have CLS at beginning. In decoder, we want to do classification on the last node because it works on left data.

8.1.2 Zero-shot | One-shot



Zero:

- once being told what to do it will
- The instruction is in the context, and still behaves as a language model

One-shot

- In addition to instruction, we give examples of how to solve it manually.

Zero-Shot Learning

8.1 Learning Methods

8 PRE-TRAINING DECODER MODELS

<p>A key emergent ability in large language models is zero-shot learning: the ability to do many tasks with no examples, and no gradient updates. If we structure the problem in a creative way, LMs can be used for many tasks.</p> <ul style="list-style-type: none"> Question answering. "Where was Tom Brady born?" Coreference resolution The cat couldn't fit into the hat because it was too big. Does it = "the cat" or it="the hat"? 	<p>GPT-2 is a bigger version of the pre-trained transformer decoder.</p> <ul style="list-style-type: none"> 117 million → 1.5 billion parameters 4GB → 40GB of training data from the web Scraped Reddit links that had at least 3 upvotes (as a proxy of quality) <p>Demonstrated that advanced language models can perform complex tasks (such as QA) without any additional fine-tuning.</p>
---	---

<https://uash.stanford.edu/courses/cs224n/slides/cs224n-2019%20lecture11-nmmlmns-rhfl.pdf>

<https://dl4mtufrankwucloudfront.net/heather.lanuana-mrcmodelslanuana-mrcmodels.arxiv.unsupervised.multitask.learners.pdf>

<p>You can get interesting zero-shot behavior if you're creative enough with how you specify your task!</p> <ul style="list-style-type: none"> Summarization on CNN/DailyMail dataset [See et al., 2017]: SAN FRANCISCO, California (CNN) - A magnitude 4.2 earthquake shook the San Francisco ... overturn unstable objects. TL;DR: ← Too Long, Didn't Read 	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>LAMBADA (PPL)</th><th>LAMBADA (ACC)</th><th>CBT-CN (ACC)</th><th>CBT-NE (ACC)</th><th>WikiText2 (PPL)</th><th>PTB (PPL)</th><th>enwik8 (BPC)</th><th>text8 (BPC)</th><th>WikiText103 (PPL)</th><th>IWB (PPL)</th></tr> </thead> <tbody> <tr> <td>SOTA</td><td>99.8</td><td>59.23</td><td>85.7</td><td>82.3</td><td>39.14</td><td>46.54</td><td>0.99</td><td>1.08</td><td>18.3</td><td>21.8</td></tr> <tr> <td>117M</td><td>35.13</td><td>45.99</td><td>87.65</td><td>83.4</td><td>29.41</td><td>65.85</td><td>1.16</td><td>1.17</td><td>37.50</td><td>75.20</td></tr> <tr> <td>345M</td><td>15.60</td><td>55.48</td><td>92.35</td><td>87.1</td><td>22.76</td><td>47.33</td><td>1.01</td><td>1.06</td><td>26.37</td><td>55.77</td></tr> <tr> <td>762M</td><td>10.87</td><td>60.12</td><td>93.45</td><td>88.0</td><td>19.93</td><td>40.31</td><td>0.97</td><td>1.02</td><td>22.05</td><td>44.57</td></tr> <tr> <td>1542M</td><td>8.63</td><td>63.24</td><td>93.30</td><td>89.05</td><td>18.34</td><td>35.76</td><td>0.93</td><td>0.98</td><td>17.48</td><td>42.16</td></tr> </tbody> </table>	LAMBADA (PPL)	LAMBADA (ACC)	CBT-CN (ACC)	CBT-NE (ACC)	WikiText2 (PPL)	PTB (PPL)	enwik8 (BPC)	text8 (BPC)	WikiText103 (PPL)	IWB (PPL)	SOTA	99.8	59.23	85.7	82.3	39.14	46.54	0.99	1.08	18.3	21.8	117M	35.13	45.99	87.65	83.4	29.41	65.85	1.16	1.17	37.50	75.20	345M	15.60	55.48	92.35	87.1	22.76	47.33	1.01	1.06	26.37	55.77	762M	10.87	60.12	93.45	88.0	19.93	40.31	0.97	1.02	22.05	44.57	1542M	8.63	63.24	93.30	89.05	18.34	35.76	0.93	0.98	17.48	42.16
LAMBADA (PPL)	LAMBADA (ACC)	CBT-CN (ACC)	CBT-NE (ACC)	WikiText2 (PPL)	PTB (PPL)	enwik8 (BPC)	text8 (BPC)	WikiText103 (PPL)	IWB (PPL)																																																									
SOTA	99.8	59.23	85.7	82.3	39.14	46.54	0.99	1.08	18.3	21.8																																																								
117M	35.13	45.99	87.65	83.4	29.41	65.85	1.16	1.17	37.50	75.20																																																								
345M	15.60	55.48	92.35	87.1	22.76	47.33	1.01	1.06	26.37	55.77																																																								
762M	10.87	60.12	93.45	88.0	19.93	40.31	0.97	1.02	22.05	44.57																																																								
1542M	8.63	63.24	93.30	89.05	18.34	35.76	0.93	0.98	17.48	42.16																																																								

https://d4mucfpkaywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

<https://uash.stanford.edu/courses/cs224n/slides/cs224n-2019%20lecture11-nmmlmns-rhfl.pdf>

Give the model context. Or coreference resolution with probability

Few-Shot Learning

<p>Large language models are also able to learn from examples.</p> <p>Simply give it some examples of doing a task and have it continue the generation.</p> <p>For best results, this is combined with the task description.</p> <p>Requires no gradient updates! The model stays frozen and only learns from the input that is given.</p>	<p>GPT-3 is another increase in decoder size.</p> <p>1.5 billion → 175 billion parameters 40GB → 600GB of training data</p> <p>Good performance on many tasks, such as QA. Seeing additional examples helps.</p> <p>Also good at generating text given any imaginable prompt.</p>
--	---

Setting	NaturalQ8	WebQ8	TriviaQA
RAG (Fine-tuned, Open-Domain) [LPP+20]	44.5	45.5	68.0
T5-11B-SSM (Fine-tuned, Closed-Book) [RRS20]	36.6	44.7	60.5
T5-11B-SM (Fine-tuned, Closed-Book)	44.5	37.4	50.1
GPT3-Zero-Shot	14.6	14.4	64.3
GPT3-One-Shot	23.0	25.3	68.0
GPT3-Few-Shot	29.9	41.5	71.2

<p>Here's a short story about two mice playing chess during a storm.</p> <p>Once upon a time there were two mice who enjoyed playing chess together. Every morning they would set up the board and play.</p> <p>One day a storm blew in and the house began to shake. The wind was so strong that it sent the chess pieces flying around the room. The two mice were determined to let the storm pass before game, so they used whatever they could find to replace the pieces.</p> <p>The mice used buttons for pawns, safety pins for knights, bottle caps for bishops, and a wooden block for the king. The board was made of sticks and paper, and they used a feather to move the pieces.</p> <p>The two mice played on right, taking turns holding under a stack of books to protect themselves from the wind. In the end, the mice passed the time more than once to play chess during a hurricane.</p>
--

<https://arxiv.org/pdf/1905.14185.pdf>

<https://ml4nlp.sailorai.ml/ml4nlp/univ.html>

Give an instruction to the system, and a paragraph and ask questions about the paragraph. This establishes patterns about answering questions about a paragraph

8.1.3 Few-shot

<p>I. Few-shot: In addition to the task description, give a few examples of the task is input. No gradient updates are performed.</p>

- Multiple examples as input

8.2 Large language models for code

We can train language models on other types of data, not just text. GitHub Copilot was trained on code from GitHub repositories.

```

1 #!/usr/bin/env ts-node
2
3 import { fetch } from "fetch-h2";
4
5 // Determine whether the sentiment of text is positive
6 // Use a web service
7 async function isPositive(text: string): Promise<boolean> {
8   const response = await fetch(`http://text-processing.com/api/sentiment/`, {
9     method: "POST",
10    body: `text=${text}`,
11    headers: {
12      "Content-Type": "application/x-www-form-urlencoded",
13    },
14  });
15  const json = await response.json();
16  return json.label === "pos"
17}

```

https://github.com/microsoft/terminal#languagedetection

8.3 Large language models for dialogue

Large language models are being adapted into dialogue systems. For example: ChatGPT, Bing and Google Bard/Gemini.

There are still plenty of issues. The model can just invent plausible-sounding claims.

Grant Tremblay (@stetoprog) Not to be a buzzkill, but the recently-parked JWST did not return the very first image of a planet outside our solar system. The first image was instead done by Chauvin et al. (2004) the VLT/NACO using adaptive optics.

Google shares drop \$100 billion after its new chatbot makes a mistake

Grant Tremblay (@stetoprog) Not to be a buzzkill, but the recently-parked JWST did not return the very first image of a planet outside our solar system. The first image was instead done by Chauvin et al. (2004) the VLT/NACO using adaptive optics.

These models pick up any biases that appear in the data, some of them harmful.

Companies are working hard to apply guardrails. And users keep finding new ways of breaking them.

<https://twitter.com/spiantade/status/1599462375887114240>

9 Advanced prompting and learning from human feedback

9.1 Chain Of Thought

LLMs are able to do some reasoning if we show them examples of reasoning.

Standard Prompting

Model Input: Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: The answer is 11.

Model Output: A: The answer is 27. ✗

Chain-of-Thought Prompting

Model Input: Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Model Input: Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?
A: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have? The answer is 9. ✓

In addition to the natural language chain-of-thought, the model can be instructed to output the reasoning steps in code.

The final answer would then be generated by executing this code in a Python interpreter.

Input: Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: Roger started with 5 tennis balls.
2 cans of 3 tennis balls each is
bought_balls = 2 * 3
balls_left = 5 + bought_balls
The answer is 11.

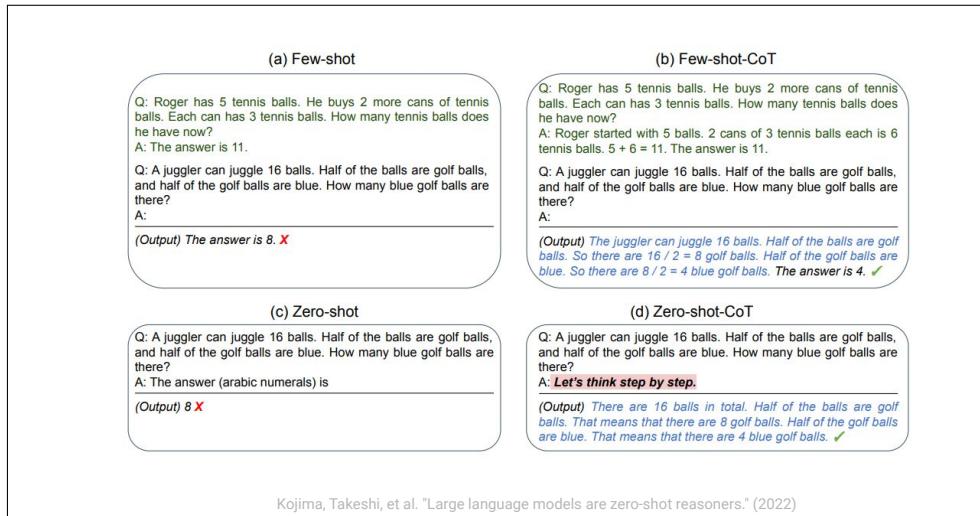
Model Output: A: The bakes started with 200 leaves. They sold 93 in the morning and 39 in the afternoon. So they sold 93 + 39 = 132 leaves. They also returned 6 leaves. So they had 200 - 132 - 6 = 62 leaves left.
The answer is 62. ✗

Input: Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: Roger started with 5 tennis balls.
2 cans of 3 tennis balls each is
bought_balls = 2 * 3
balls_left = 5 + bought_balls
The answer is 11.

Model Output: A: The bakes started with 200 leaves. They sold 93 in the morning and 39 in the afternoon. So they sold 93 + 39 = 132 leaves. They also returned 6 leaves. So they had 200 - 132 - 6 = 62 leaves left.
The answer is 62. ✗

Gao, Luyu, et al. "PAL: Program-aided Language Models." (2022)

9.1.1 Zero-shot chain of thought

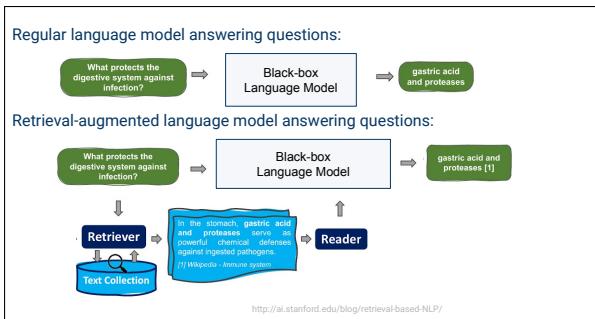
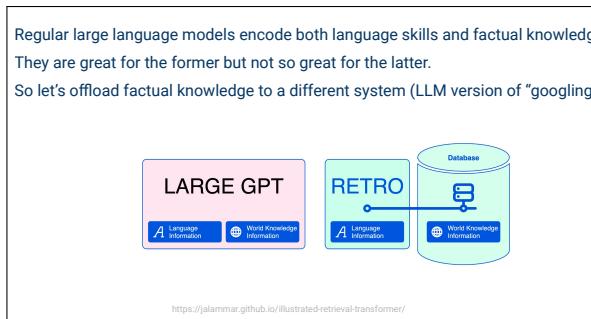


Additional reasoning examples help, but zero-shot chain-of-thought greatly outperforms showing the model only examples of solving the task.

	MultiArith	GSM8K
Zero-Shot	17.7	10.4
Few-Shot (2 samples)	33.7	15.6
Few-Shot (8 samples)	33.8	15.6
Zero-Shot-CoT	78.7	40.7
Few-Shot-CoT (2 samples)	84.8	41.3
Few-Shot-CoT (4 samples : First) (*1)	89.2	-
Few-Shot-CoT (4 samples : Second) (*1)	90.5	-
Few-Shot-CoT (8 samples)	93.0	48.7

Kojima, Takeshi, et al. "Large language models are zero-shot reasoners." (2022)

9.2 Retrieval-based language models



Idea is that in a normal language model we have the same language model weights storing both linguistic and language information along with the factual information. NNs aren't a great place to store factual information because everything is distributed and smooth embeddings. Thus the language model acts as the controller, and the factual retrieval model finds relevant texts from any databases you have. *This allows for citations*

9.3 Limitations of instruction fine-tuning

Language models are being trained with instruction fine-tuning, using manually created ground truth data that follows instructions. This data is expensive to collect.

There are also other, subtler limitations.

- Problem 1: tasks like open-ended creative generation have no right answer.
Write me a story about a dog and her pet grasshopper.
- Problem 2: language modeling penalizes all token-level mistakes equally, but some errors are worse than others.

Can we explicitly attempt to satisfy human preferences?

<https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture11-prompting-rlf.pdf>

9.3.1 Reinforcement learning from human feedback

Let's say we are training a language model on some task (e.g. summarization).

For each LM sample s , imagine we had a way to obtain a human reward of that summary: $R_s \in \mathbb{R}$, higher is better.

SAN FRANCISCO, California (CNN) -- A magnitude 4.2 earthquake shock the San Francisco ... overturn unstable objects.	An earthquake hit San Francisco. There was minor property damage, but no injuries.	The Bay Area has good weather but is prone to earthquakes and wildfires.
s_1	s_2	
$R(s_1) = 8.0$	$R(s_2) = 1.2$	

Now we can optimize our language model to maximize this expected reward, using reinforcement learning (outside the scope of this course).

<https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture11-prompting-rlf.pdf>

- there are some answers that are still correct

- Have language models decide amongst a few answers and with reinforcement learning improve the answer.

Problem: Human-in-the-loop is expensive!
Solution: Instead of directly asking humans for preferences, model their preference as a separate (NLP) problem.
Train another LM to predict the human score for a given text.

Problem: Human judgments are noisy and miscalibrated!
Solution: Instead of asking for direct ratings, ask for pairwise comparisons, which can be more reliable.

An earthquake hit San Francisco.	A 4.2 magnitude earthquake hit San Francisco, > resulting in massive damage.	The Bay Area has good weather but is prone to earthquakes and wildfires.
There was minor property damage, but no injuries.	>	

<https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture11-prompting-rlf.pdf>

- Humans are expensive, so train a model to choose one!
- Human choices are noisy, so instead of asking for direct scores, ask for comparisons.

9.3.2 Training Pipeline

