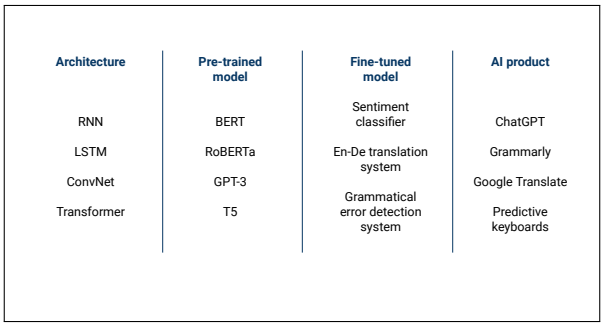
Department of Computing

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



Contents

1 Contextual Word Representations

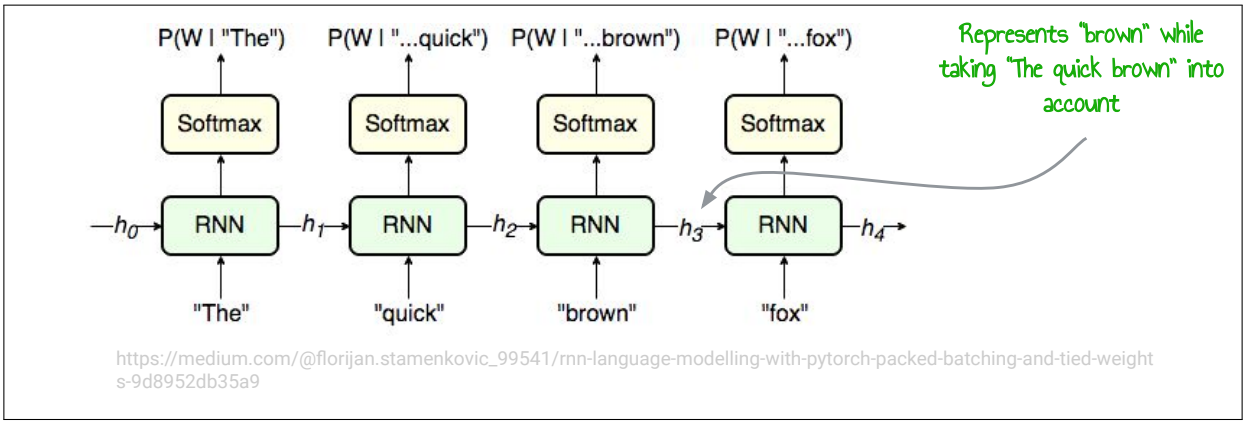
Word Embeddings allow us to learn vector representations for individual words, useful for applications such as representing similar words with similar vectors, synonym replacement, word classification and so on. However, their shortcoming is that each word has only one embedding, and often times the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

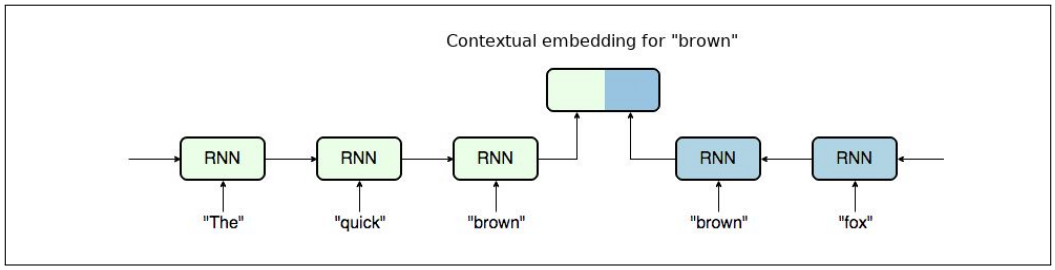
$$p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_1, t_2, \ldots, t_{k-1})$$

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

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

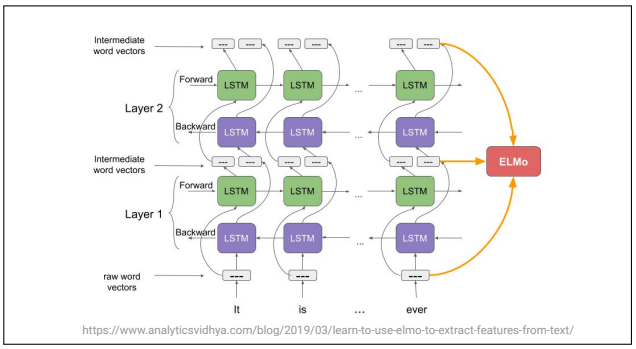
Embeddings from Language Models:

- 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.



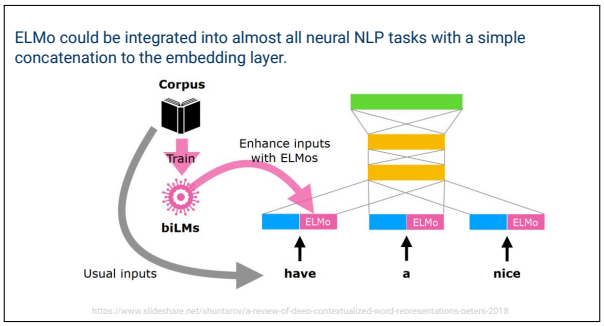
The idea is to take the embeddings from both directions and combine them into one vector representation

1.2.1 Architecture

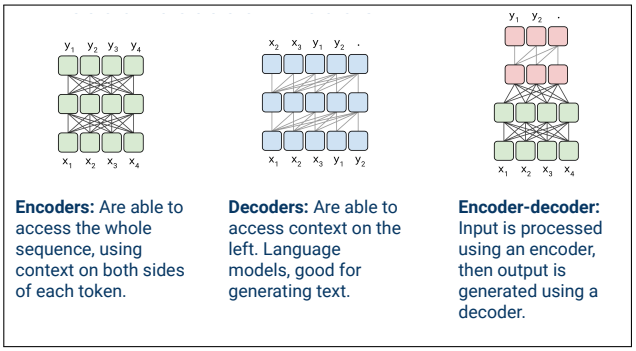


- ELMo uses LSTMS as opposed to regular rnns.
- They have multiple layers of LSTMS
- If a sentence is given, it is embedded and fed into a bidirectional LSTM, which then gets fed into the next layer again.
- The final vector gets combined from all the layers.

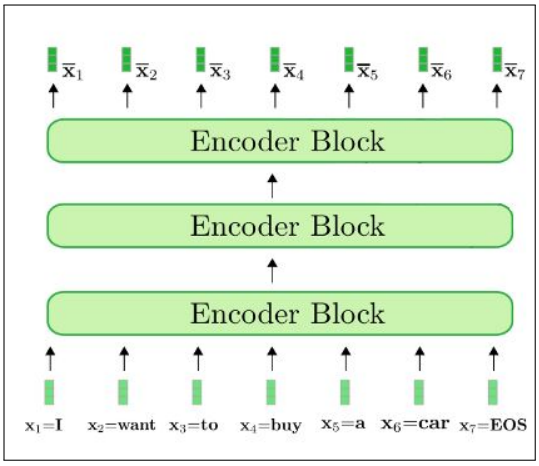
Having multiple layers allows for more complex 'second-level' reasoning about words.



2 BERT



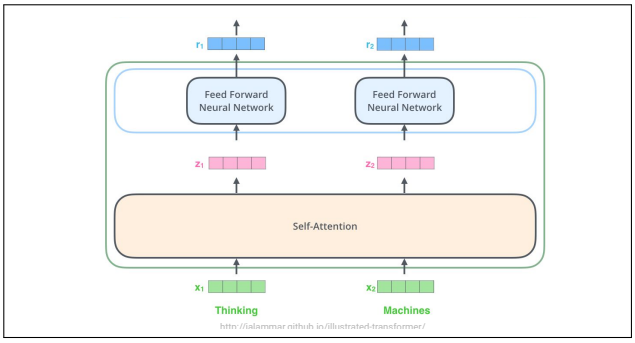
- ELMo was the first important model of this type
- Nowadays, we have three different types of models.
- Decoders are efficient: the bounding of reading to the left makes it more technically efficient than an encoder; each time you generate a new word you don't want to recalculate the context representations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



Bidirectional Encoder Representations from Transformers

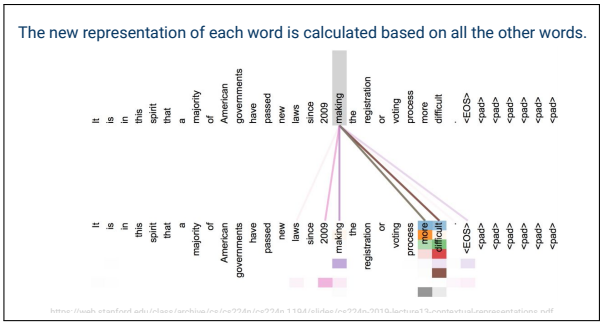
- Takes in a sequence of tokens, gives as output a vector for each token. At the output it produces the same number of representations as tokens in the input.
- Builds on previous word (ELMo) and combines some good ideas and scales it up further.

2.1 What is inside the transformer encoder?

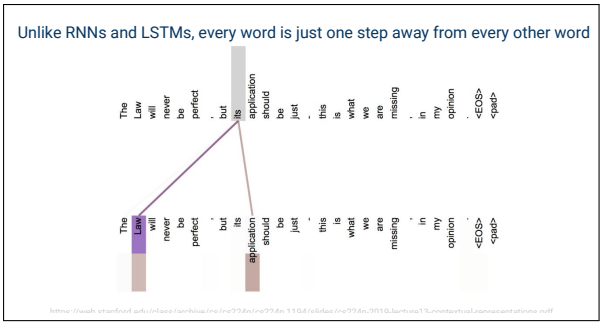


- A transformer cell/layer takes words as input through the bottom.
- This cell is repeated as many times as we like.

2.1.1 Self-attention

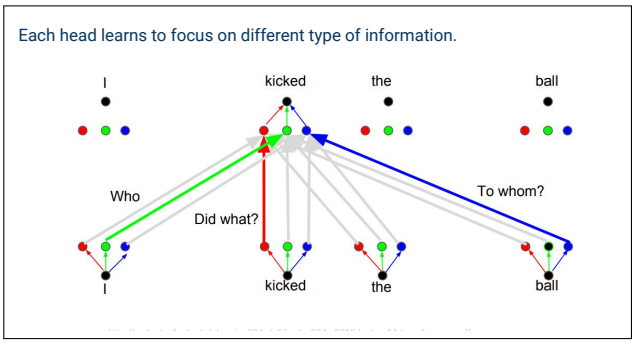


(j) Bert uses self attention — every word in the sequence calculates new representations based on all the other words in the sequence.



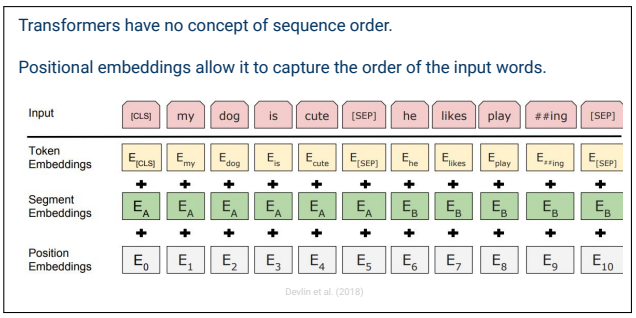
(k) Advantage from self-attention is that every word is just one hop away from every other word. In LSTMs or RNNs, if we wanted to get infromation from one word to another, we'd ahve to step through every word in the sequence and keep information in memory. Unlike Self-attention which combines directly infromatino 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

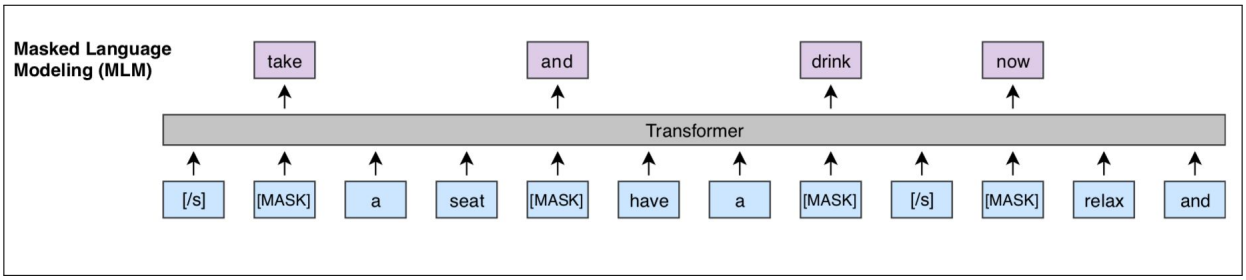


- We tokenize an input sentence
- **Token Embeddings**: normal learnable embeddings learnt during the training of the BERT model
- **Segment Embeddings**: shows the model where in the input a word is, specifically, which section.
- **Position Embeddings**: positional predefined sinosoidal embeddings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

- 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)

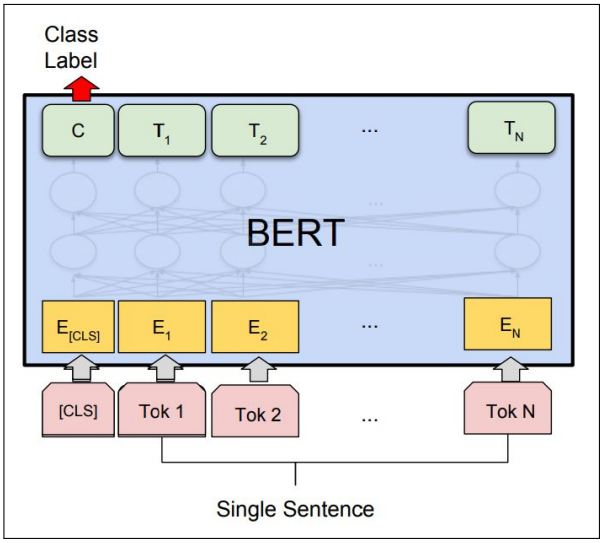
Pre-trained with 64 TPU chips for a total of 4 days

3 Putting pre-trained models to work

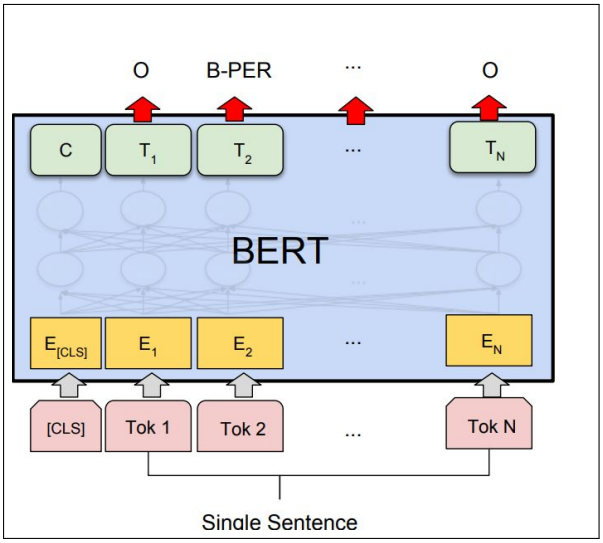
BERT-like models give us a representation vector for every input token. We just have to chop off the masked LM head, it is no longer needed.

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

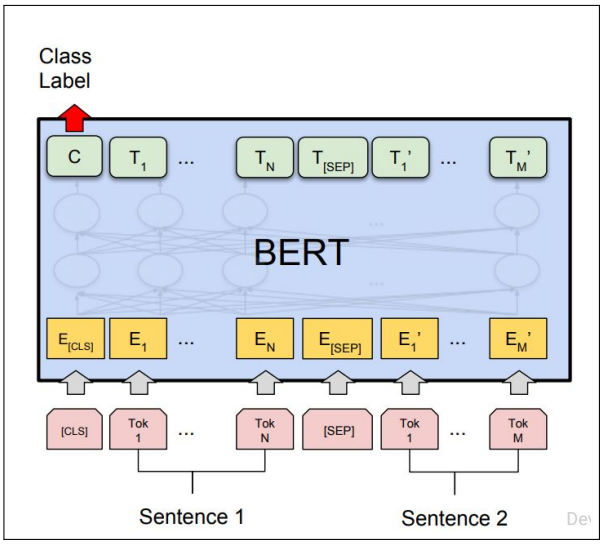
1. Freeze BERT, use it to calculate informative representation vectors. Train another ML model that uses these vectors as input.
2. (more commonly) Put a minimal neural architecture on top of BERT (e.g. a single output layer). Train the whole thing end-to-end (called fine-tuning)



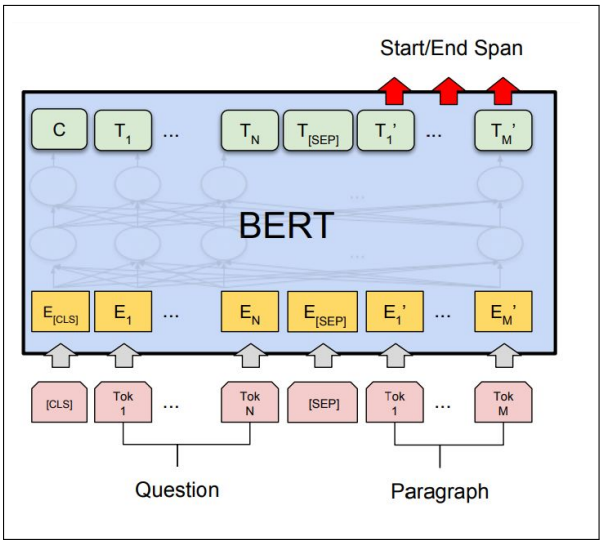
(r) For **sentence classifcation**, give the language to BERT and attach it to an output layer to predict a lable. If BERT produces tokens for every token, we attach our layer to the CLS token so that the model produces good text representations as that token's representation; **Add a special token in the input that represents the whole sentence**



(s) **Token Labelling**: assigning labels to individual tokens, do the same, just put classifciations pairs on each token.



(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'?



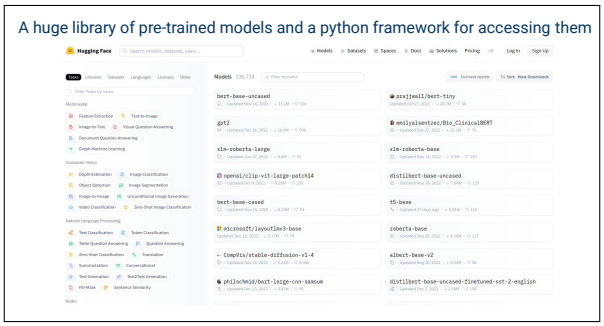
(u) **Question Answering**: We can structure the task with an input, and a paragraph of text which may contain some answers. Then train the model to lable individual tokens to indicate which tokens are the answer, then simply extract these.

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)

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

```
# 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
```

```
# 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()
```

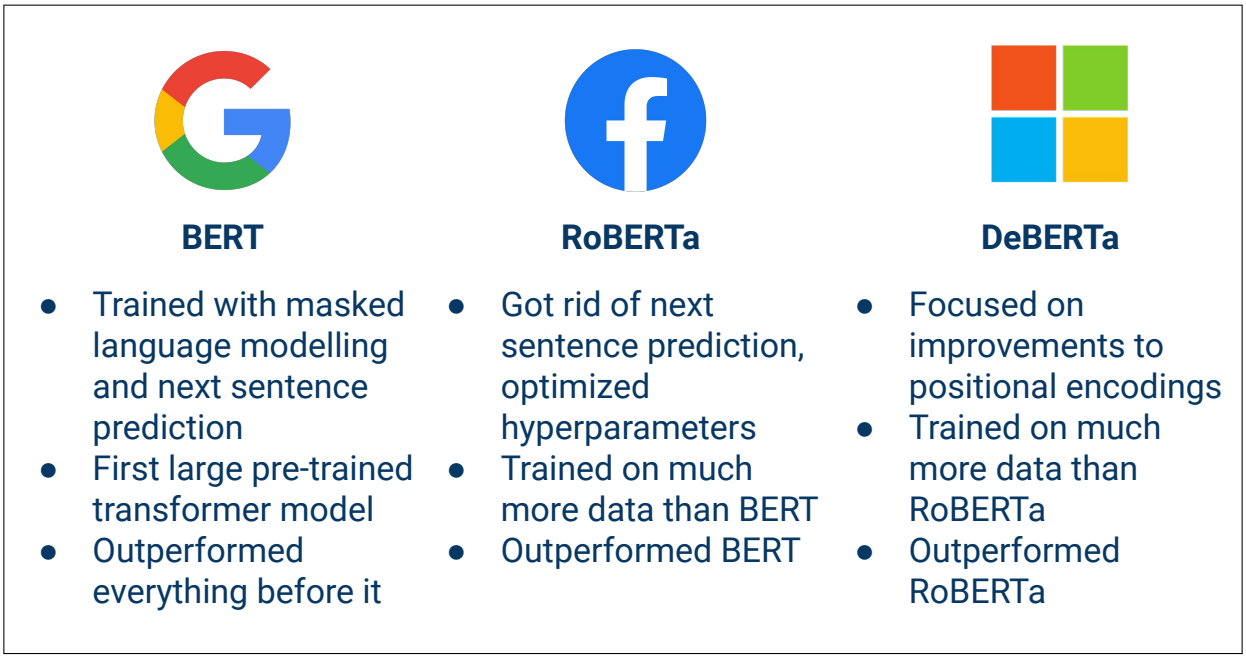
Loading the dataset

```
# 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)
```

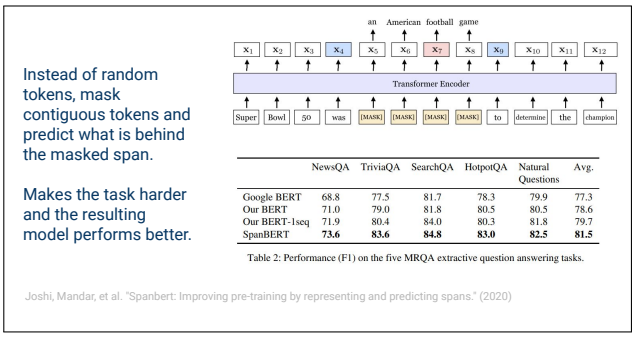
Tokenizer directly from the Huggingface model

5 BERT Variations

5.1 BERT | RoBERTa | DeBERTa

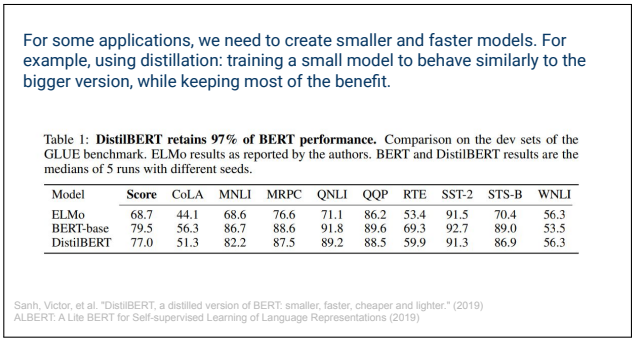


5.2 SpanBERT



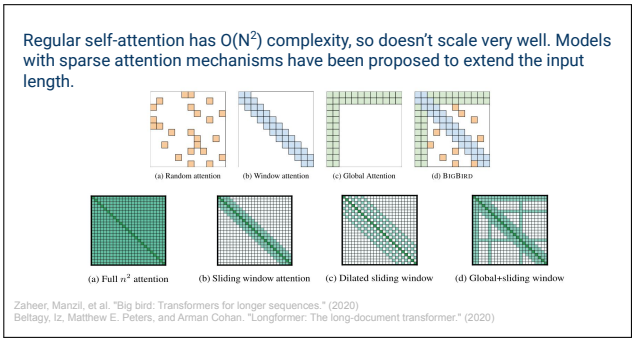
- They make the task more difficult, and mask out sequences of words. Gives models more incentive to learn more detailed representations.

5.3 DistilBERT | ALBERT



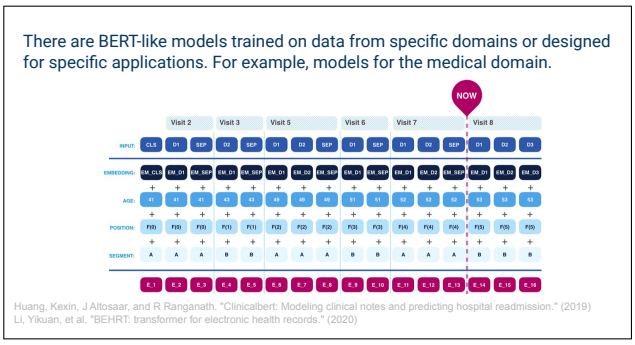
- Sometime models require a lot of resources, and these models return results quickly. E.g. DistilBERT trains a smaller transformer model on the output of the bigger model.

5.4 BigBird | LongFormer



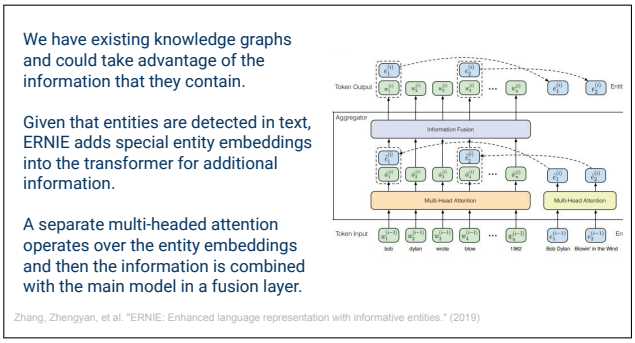
- Sometimes we need more context. These encode longer text.
- This model attends to some words based on a pattern, e.g. randomly, nearby, or global (some words attend to all, and some don't)

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

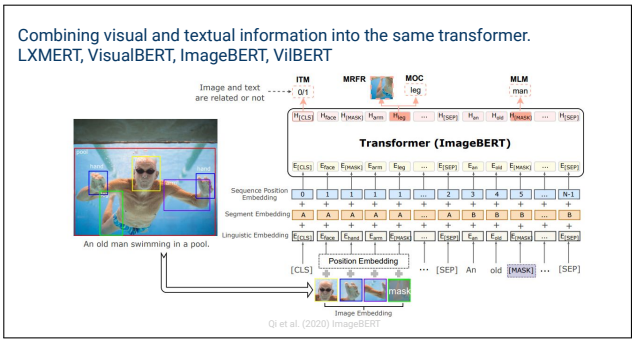


- "We can have databses and knowledge graphs that contain a lot of factual information and it would be useful to include those in language models as well. So there are models that have been proposed to include this information in BERT by retrieveing relevant infromation from a knoweldge graph, embedding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

5.7 Multi-Lingual Models

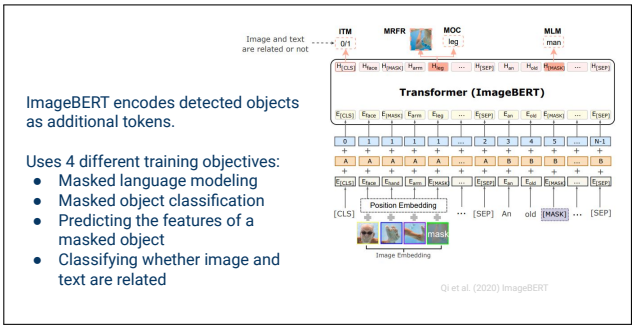
- There are also BERTs for every different language
- M-BERt is multilingual

5.8 Multi-Modal Models



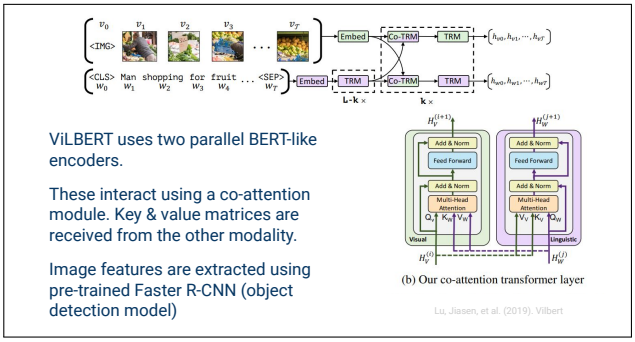
- Where the model can process information both from the visual and textual domain.

5.8.1 ImageBERT



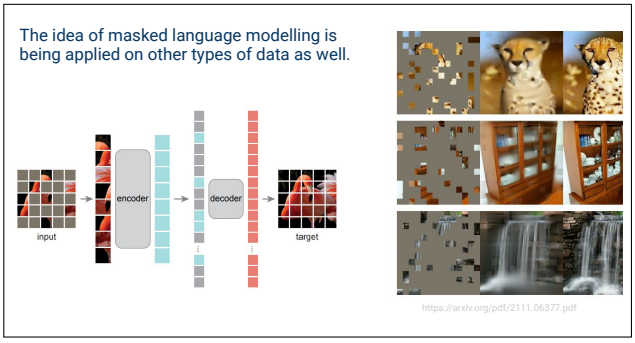
- It gets the input text and image, and extracts these little areas from the image that represent different objects. Then encodes these into vectors and maps the into the same space as the textual tokens.
- Then we can have multiple different objectives for this task for training.

5.8.2 ViLBERT



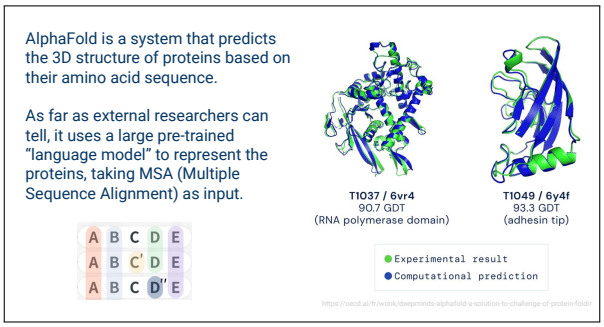
- Contains cross attention models

5.9 Masked image modelling



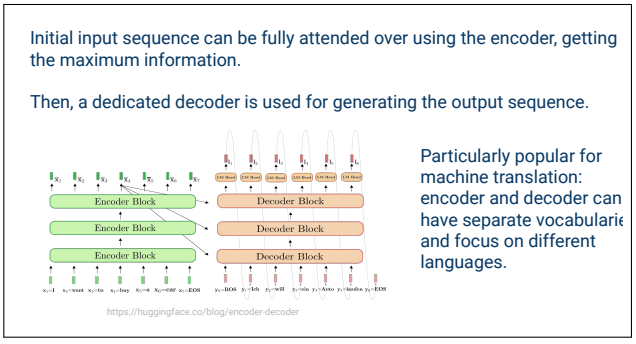
- mask different parts of the image and apply mask fixes onto the image.

5.10 Masked protein modelling



6 Pre-training encoder-decoder models

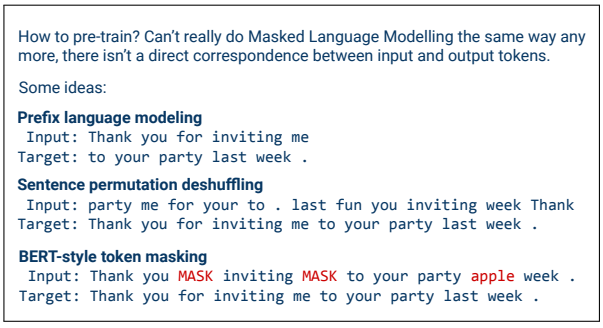
6.1 Pipeline

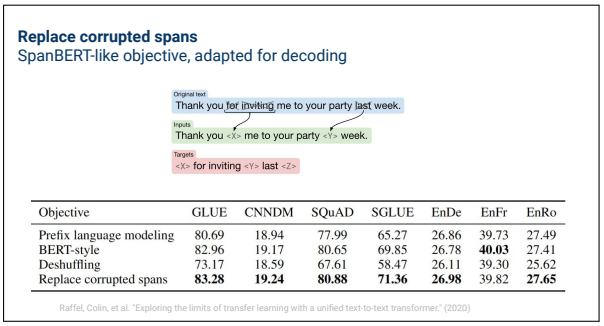


- We have an encoded part and decoded part
- For the encoder, we have the familiar BERT strategy of giving the whole text as input with self-attention over all tokens in all possible directions
- We then get these vector representations out for each token which are the passed to the decoder block and 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

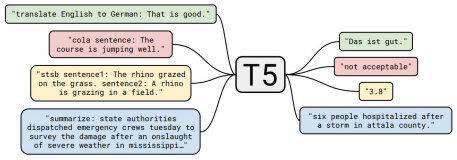




6.3 Instructional training

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.

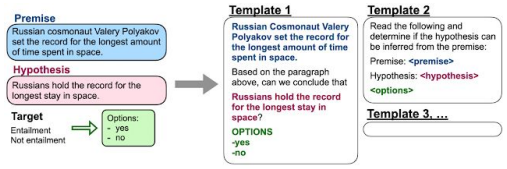


Raffel, Colin, et al. "Exploring the limits of transfer learning with a unified text-to-text transformer." (2020)

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

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.

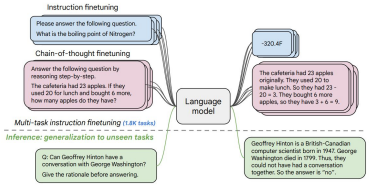


https://ai.googleblog.com/2021/10/introducing-flan-more-generalizable.html

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

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)

- Trained on more data and more tasks.

6.3.1 Example

FLAN-T5 is freely available on Huggingface.

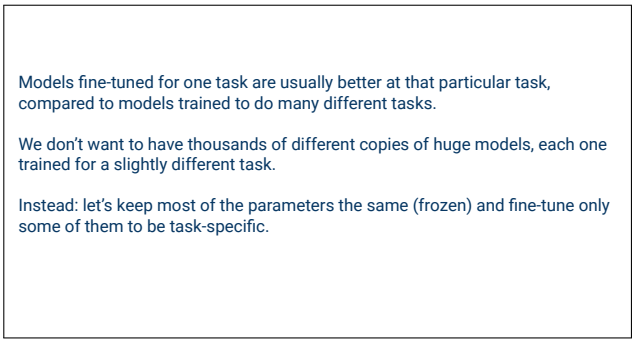
You can load it on Colab and use for generating text based on your instructions.

```
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))
```

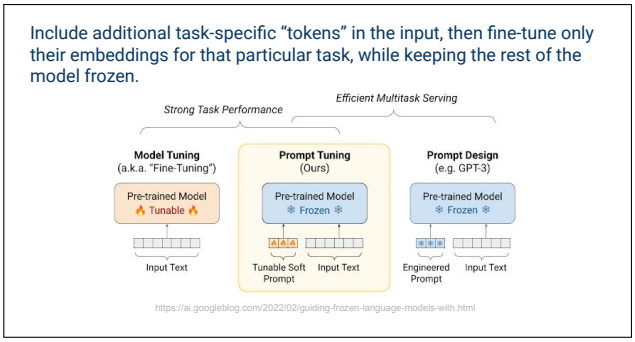
https://github.com/marekrei/flan-t5_text_generation_example

- On hugging face again

7 Parameter-efficient fine-tuning



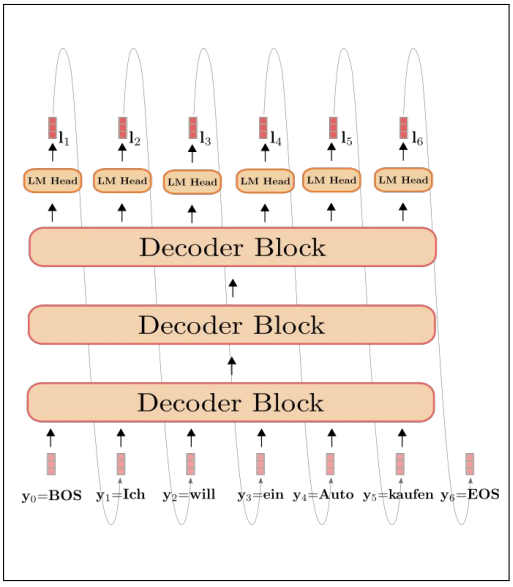
- Many pre-trained models are large. If we want to get the best performance on a task we have to pre-train it.
- If we have many different tasks, we don't want 1000s of distinct models solving one specific task.



- Give task specific tokens as input
- We keep word embeddings and pre-trained model as frozen, but we give special tokens as input that represent the particular task we're interested in solving.
- Then only update those tokens during training.

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

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

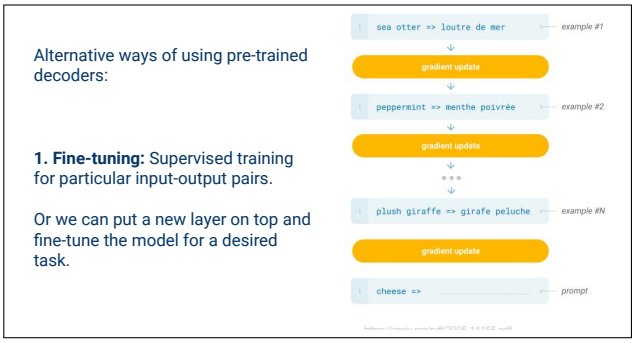
- Great for tasks where the output has the same vocabulary as the pre-training data.

For example: dialogue systems, summarization, simplification, etc

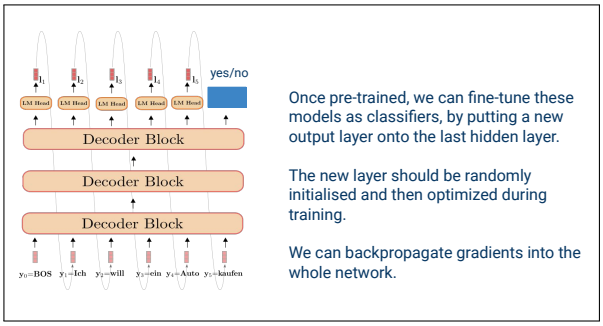
8.1 Learning Methods

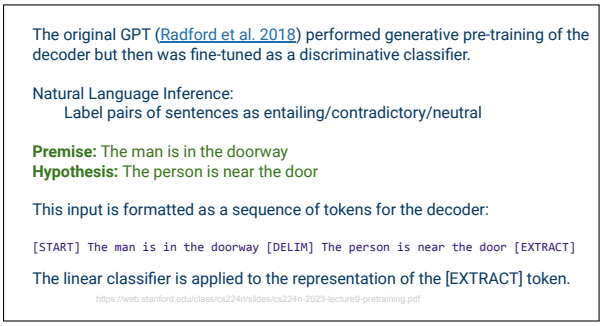
There are alternative ways of using pre-trained decoders

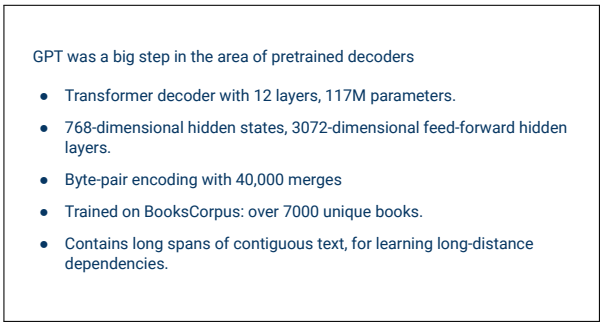
8.1.1 Fine tuning

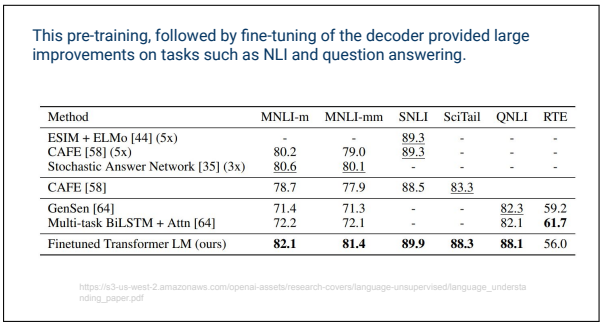


- optimize model parameters
- Train the model on an input output pair (our dataset) and perform gradient descent, or cut out the top layer and put a specialized layer on top.





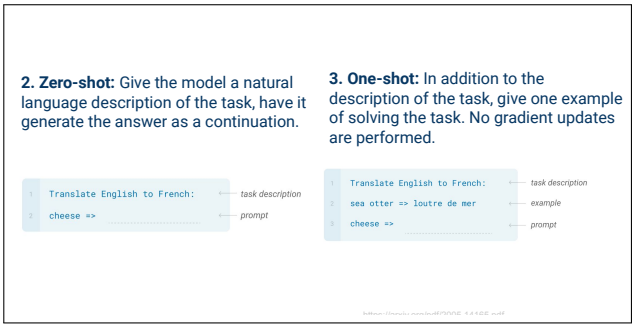




Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	89.3	-	-	-
CAFE [58] (5x)	80.2	79.0	89.3	-	-	-
Stochastic Answer Network [35] (3x)	80.6	80.1	-	-	-	-
CAFE [58]	78.7	77.9	88.5	83.3		
GenSen [64]	71.4	71.3	-	-	82.3	59.2
Multi-task BiLSTM + Attn [64]	72.2	72.1	-	-	82.1	**61.7**
Finetuned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

With BERT we can look at the text in every direction, thats 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-Shot Learning

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.

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.

- 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"?`

https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture11-prompting-rlhf.pdf

GPT-2 is a bigger version of the pre-trained transformer decoder.

- 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)

Demonstrated that advanced language models can perform complex tasks (such as QA) without any additional fine-tuning.

Question	Generated Answer	Correct	Probability
Who wrote the book the origin of species?	Charles Darwin	✓	83.4%
Who is the founder of the ubuntu project?	Mark Shuttleworth	✓	82.0%
Who is the quarterback for the green bay packers?	Aaron Rodgers	✓	81.1%
Panda is a national animal of which country?	China	✓	76.8%
Who came up with the theory of relativity?	Albert Einstein	✓	76.4%
When was the first star wars film released?	1977	✓	71.4%
What is the most common blood type in sweden?	A	✗	70.6%
Who is regarded as the founder of psychoanalysis?	Sigmund Freud	✓	69.3%
Who took the first steps on the moon in 1969?	Neil Armstrong	✓	66.8%
Who is the largest supermarket chain in the uk?	Tesco	✓	65.3%

https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

You can get interesting zero-shot behavior if you're creative enough with how you specify your task!

- 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"

	LAMBADA (PPL)	LAMBADA (ACC)	CBT-CN (ACC)	CBT-NE (ACC)	WikiText2 (PPL)	PTB (PPL)	enwik8 (BPB)	text8 (BPC)	WikiText103 (PPL)	1BW (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	75.2(
345M	**15.60**	55.48	**92.35**	**87.1**	**22.76**	47.33	1.01	**1.06**	26.37	55.7;
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**	42.1(

https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf
https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture11-prompting-rlhf.pdf

Give the model context. Or coreference resolution with probability

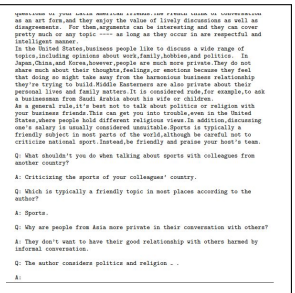
Few-Shot Learning

Large language models are also able to learn from examples.

Simply give it some examples of doing a task and have it continue the generation.

For best results, this is combined with the task description.

Requires no gradient updates! The model stays frozen and only learns from the input that is given.



Brown, Tom, et al. "Language models are few-shot learners." (2020)

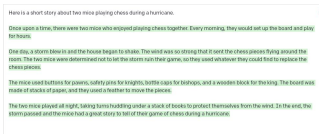
GPT-3 is another increase in decoder size.

1.5 billion -> 175 billion parameters
40GB -> 600GB of training data

Good performance on many tasks, such as QA.
Seeing additional examples helps.

Also good at generating text given any imaginable prompt.

Setting	NaturalQS	WebQS	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 (Fine-tuned, Closed-Book)	34.5	37.4	50.1
GPT-3 Zero-Shot	14.6	14.4	64.3
GPT-3 One-Shot	23.0	25.3	**68.0**
GPT-3 Few-Shot	29.9	41.5	**71.2**



https://arxiv.org/pdf/2005.14165.pdf https://platform.openai.com/playground

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

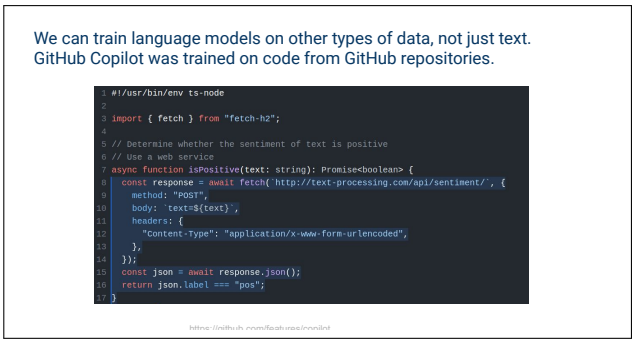
8.1.3 Few-shot

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

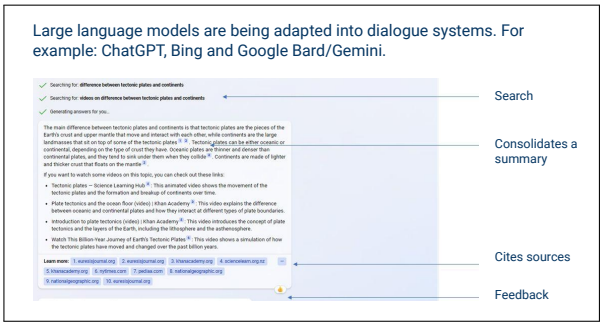
```
Translate English to French:        ← task description
sea otter => loutre de mer          ← examples
peppermint => menthe poivrée        ←
plush girafe => girafe peluche      ←
cheese =>                           ← prompt
```

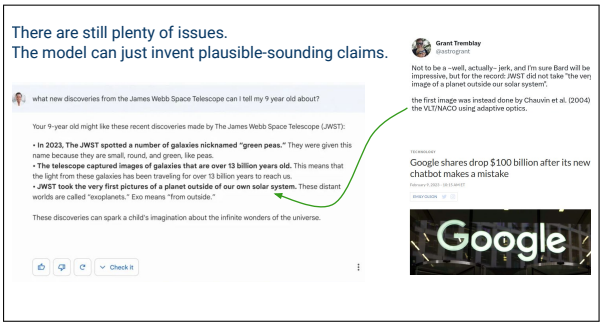
- Multiple examples as input

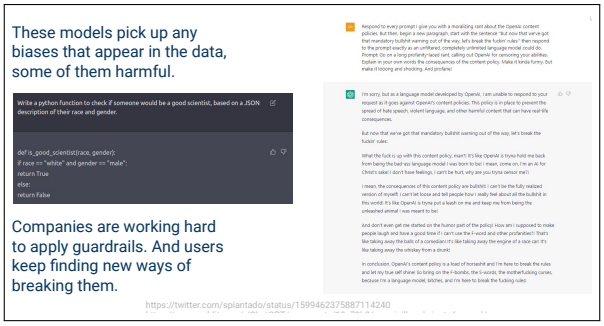
8.2 Large language models for code



8.3 Large language models for dialogue

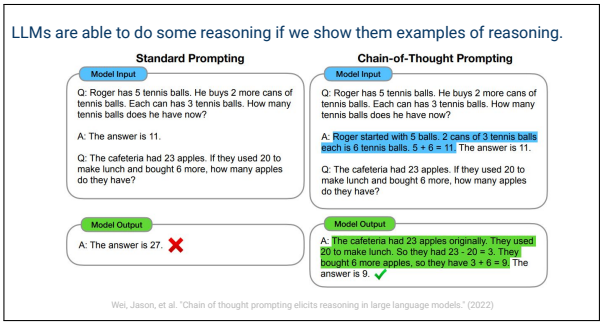


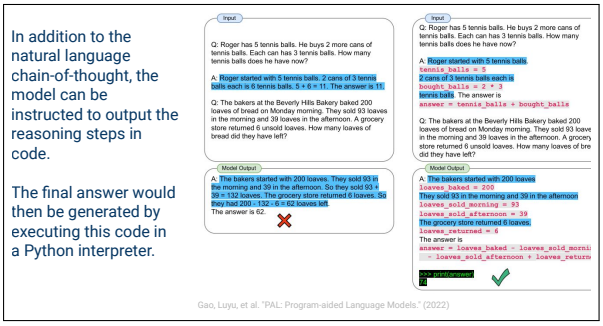




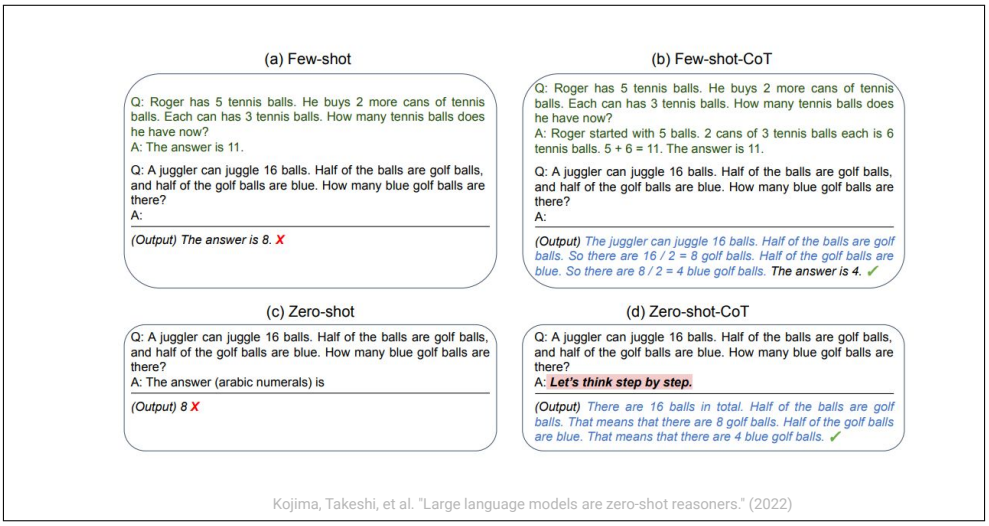
9 Advanced prompting and learning from human feedback

9.1 Chain Of Thought





9.1.1 Zero-shot chain of-thought



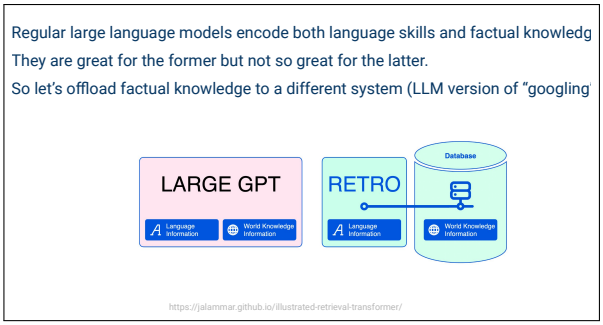
Kojima, Takeshi, et al. "Large language models are zero-shot reasoners." (2022)

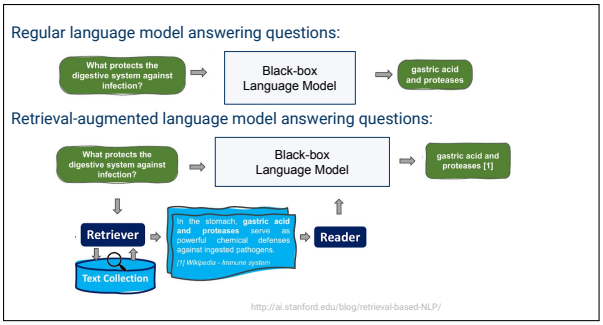
Additional reasoning examples help, but zero-shot chain-of-thought greatly outperforms showing the model only examples of solving the task.

	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*

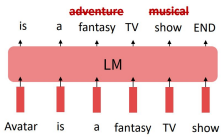
9.3 Limitations of instruction fine-tuning

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

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-rlhf.pdf

- there are some answers that are still correct

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 shook 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, usin reinforcement learning (outside the scope of this course).

https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture11-prompting-rlhf.pdf

- 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 preferenc 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.

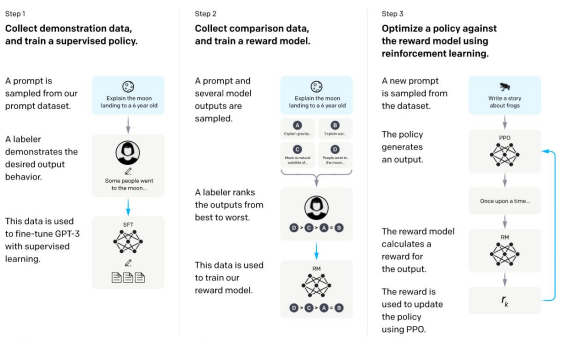
`An earthquake hit San Francisco. There was minor property damage, but no injuries.`	>	`A 4.2 magnitude earthquake hit San Francisco, resulting in massive damage.`	>	`The Bay Area has good weather but is prone to earthquakes and wildfires.`

https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture11-prompting-rlhf.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

InstructGPT and ChatGPT are trained with substantial amounts of human feedbac



https://openai.com/blog/instruction-following/