Natural Language Processing Contextualized Embeddings, Pre-Training, Fine-Tuning and Large Language Models

Felipe Bravo-Marquez

June 7, 2023

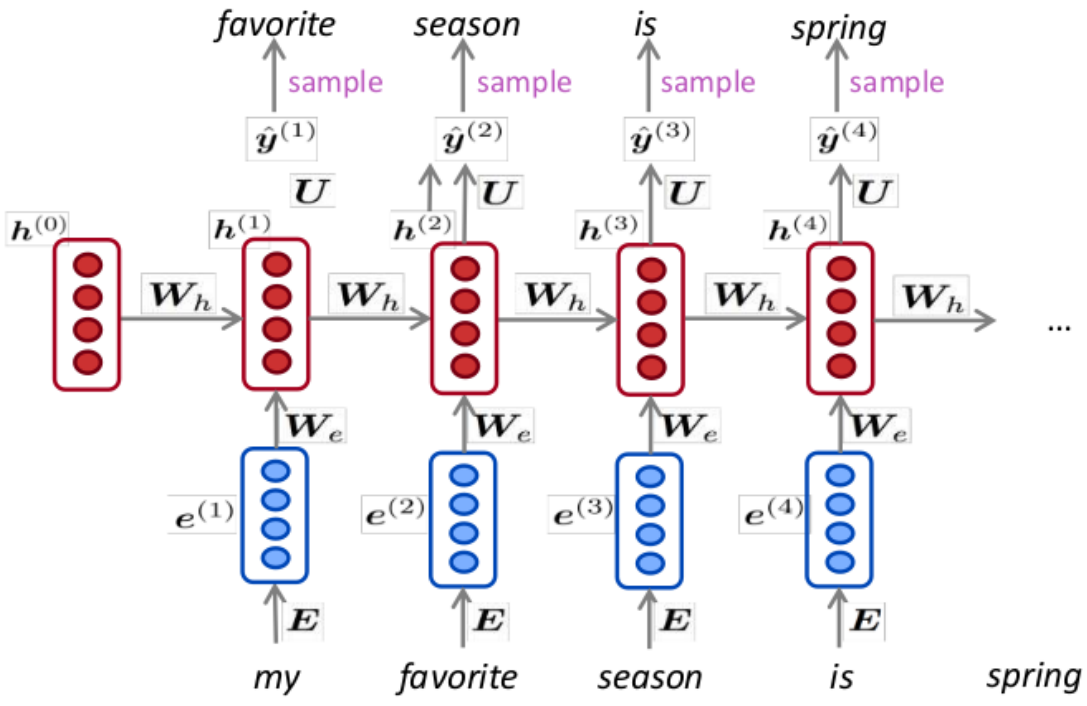
Representations for a word

- So far, we've basically had one representation of words, the word embeddings we've already learned: Word2vec, GloVe, fastText.[1].
- These embeddings have a useful semi-supervised quality, as they can be learned from unlabeled corpora and used in our downstream task-oriented architectures (LSTM, CNN, Transformer).
- However, they exhibit two problems.
- Problem 1: They always produce the same representation for a word type regardless of the context in which a word token occurs
- We might want very fine-grained word sense disambiguation
- Problem 2: We just have one representation for a word, but words have different aspects, including semantics, syntactic behavior, and register/connotations

[1] These slides are based on the Stanford CS224N: Natural Language Processing with Deep Learning course:
`http://web.stanford.edu/class/cs224n/`

Neural Language Models can produce Contextualized Embeddings

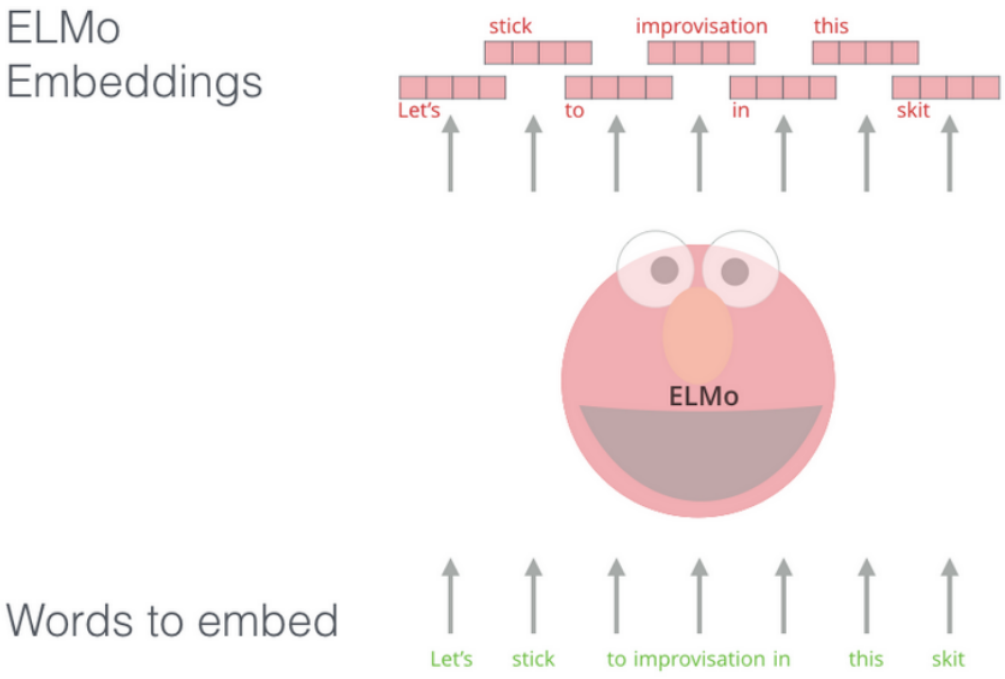
- In, a Neural Language Model (NLM), we immediately stuck word vectors (perhaps only trained on the corpus) through LSTM layers
- Those LSTM layers are trained to predict the next word.
- But these language models produce context-specific word representations in the hidden states of each position.



ELMo: Embeddings from Language Models

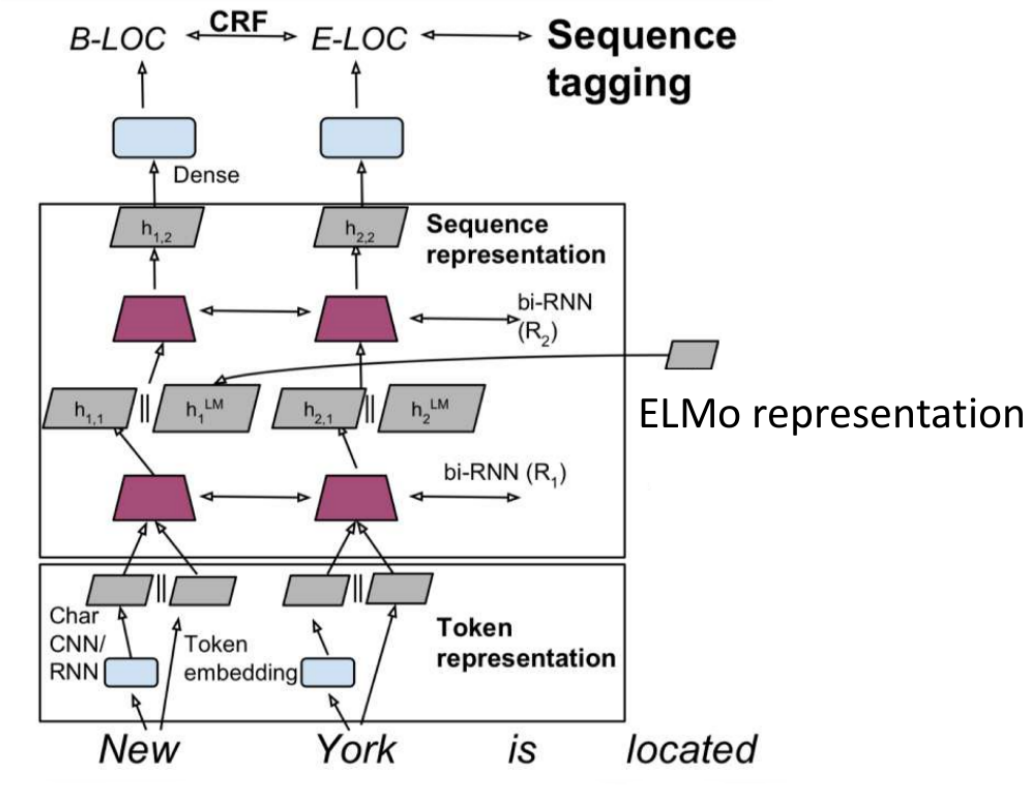
- Idea: train a large language model (LM) with a recurrent neural network and use its hidden states as “contextualized word embeddings” [Peters et al., 2018].
- ELMO is bidirectional LM with 2 biLSTM layers and around 100 million parameters.
- Uses character CNN to build initial word representation (only)
- 2048 char n-gram filters and 2 highway layers, 512 dim projection
- User 4096 dim hidden/cell LSTM states with 512 dim projections to next input
- Uses a residual connection
- Parameters of token input and output (softmax) are tied.

ELMo: Embeddings from Language Models



ELMo: Use with a task

- First run biLM to get representations for each word.
- Then let (whatever) end-task model use them.
- Freeze weights of ELMo for purposes of supervised model.
- Concatenate ELMo weights into task-specific model.



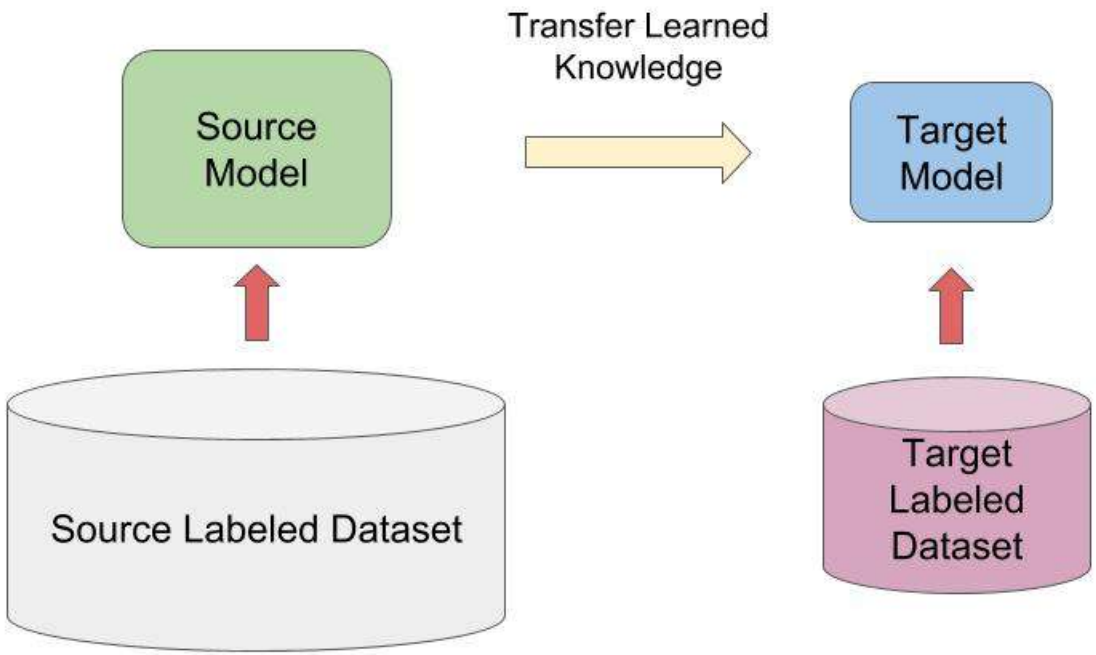
$$\mathbf{h}_{k,1} = [\overrightarrow{\mathbf{h}}_{k,1}; \overleftarrow{\mathbf{h}}_{k,1}; \mathbf{h}_k^{LM}]$$

ELMo: Results

Name	Description	Year	F1
ELMo	ELMo in BiLSTM	2018	92.22
TagLM Peters	LSTM BiLM in BiLSTM tagger	2017	91.93
Ma + Hovy	BiLSTM + char CNN + CRF layer	2016	91.21
Tagger Peters	BiLSTM + char CNN + CRF layer	2017	90.87
Ratinov + Roth	Categorical CRF+Wikipeda+word cls	2009	90.80
Finkel et al.	Categorical feature CRF	2005	86.86
IBM Florian	Linear/softmax/TBL/HMM ensemble, gazettes++	2003	88.76
Stanford	MEMM softmax markov model	2003	86.07

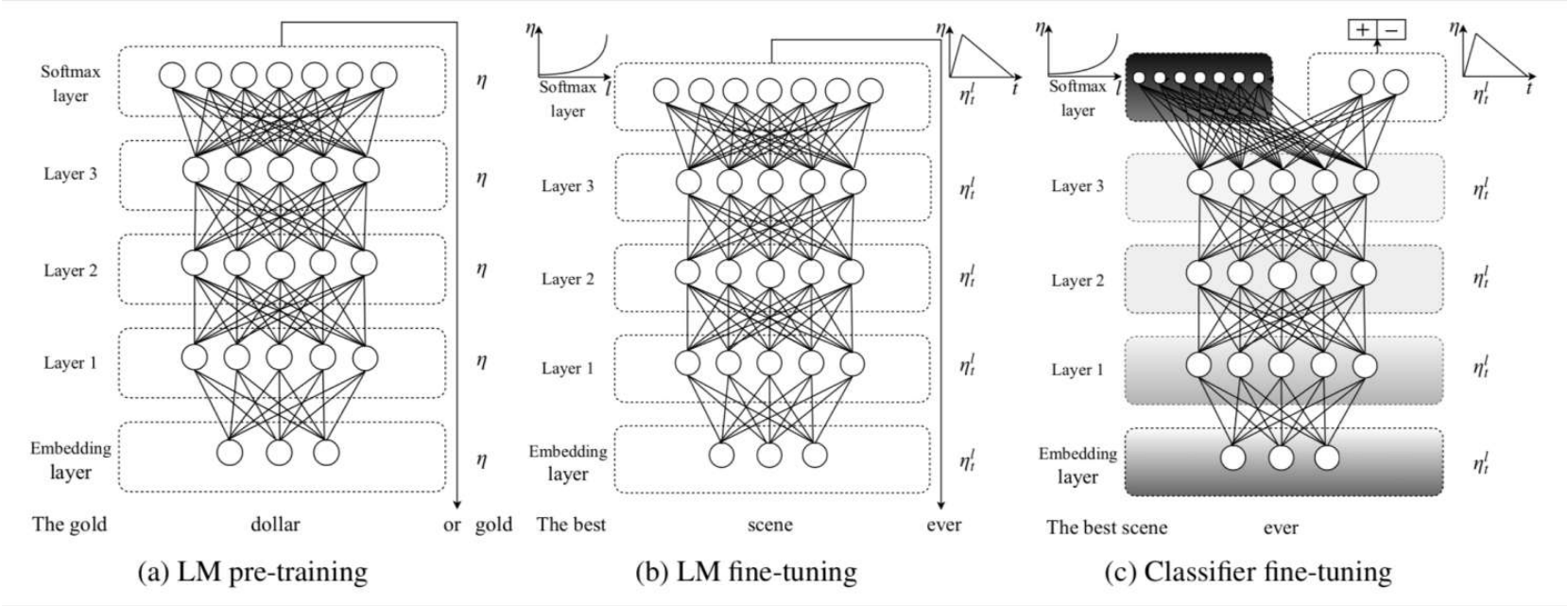
ULMfit

- Howard and Ruder (2018) Universal Language Model Fine-tuning for Text Classification [Howard and Ruder, 2018].
- Same general idea of transferring NLM knowledge
- Here applied to text classification



ULMfit

- Train LM on big general domain corpus (use biLM)
- Tune LM on target task data
- Fine-tune as classifier on target task



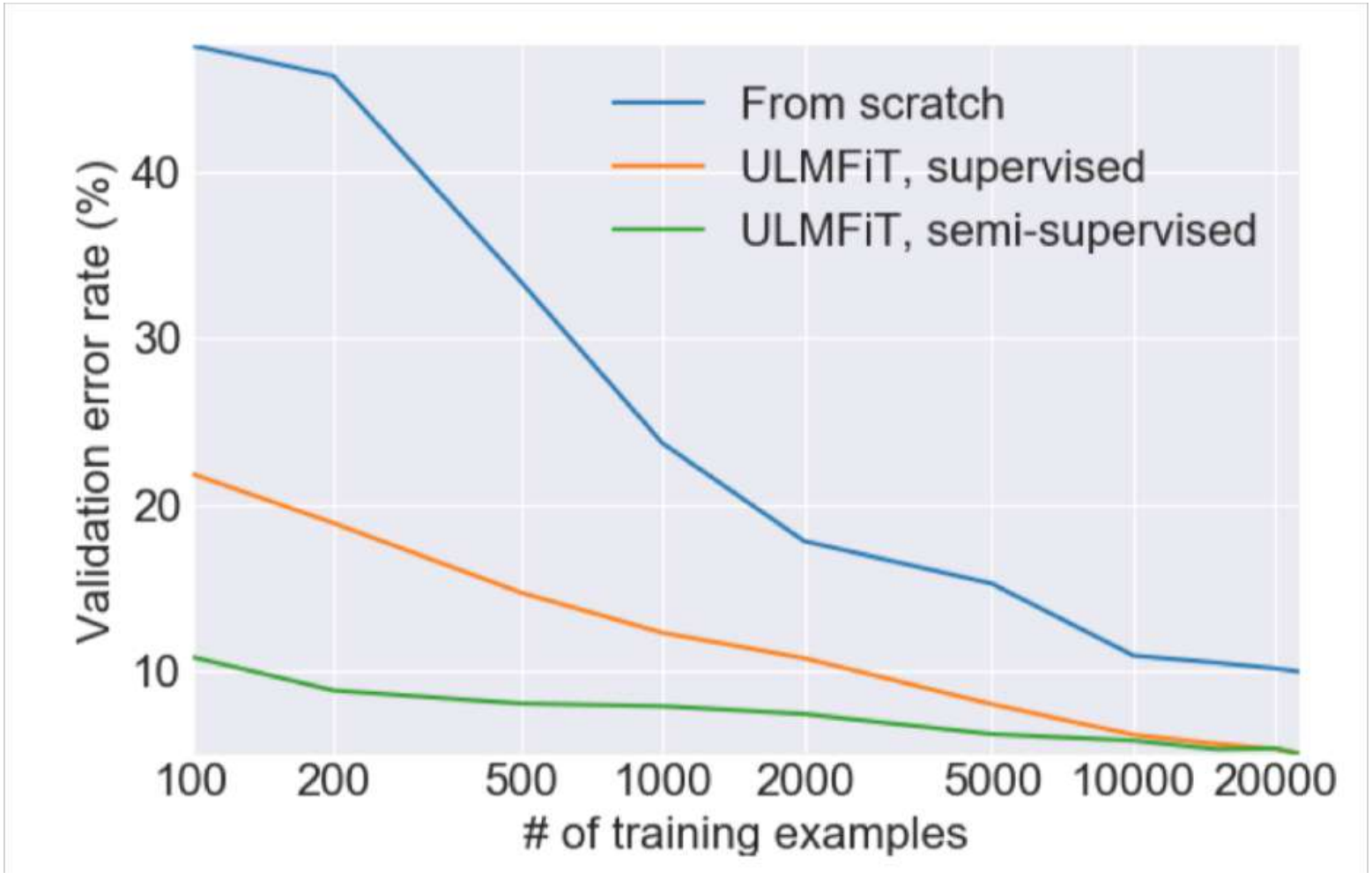
ULMfit emphases

- Use reasonable-size “1 GPU” language model not really huge one
- A lot of care in LM fine-tuning
- Different per-layer learning rates
- Slanted triangular learning rate (STLR) schedule
- Gradual layer unfreezing and STLR when learning classifier
- Classify using concatenation [h_T,maxpool(h),meanpool(h)]

	Model	Test		Model	Test
IMDb	CoVe (McCann et al., 2017)	8.2	TREC-6	CoVe (McCann et al., 2017)	4.2
	oh-LSTM (Johnson and Zhang, 2016)	5.9		TBCNN (Mou et al., 2015)	4.0
	Virtual (Miyato et al., 2016)	5.9		LSTM-CNN (Zhou et al., 2016)	3.9
	ULMFiT (ours)	**4.6**		ULMFiT (ours)	**3.6**

Text classifier error rates

ULMfit transfer learning



Let's scale it up!

ULMfit
Jan 2018
Training:
1 GPU day

GPT
June 2018
Training
240 GPU days

BERT
Oct 2018
Training
256 TPU days
~320–560
GPU days

GPT-2
Feb 2019
Training
~2048 TPU v3
days according to
a reddit thread

Transformer models

All of these models are Transformer architecture models … so maybe we had better learn about Transformers?

ULMfit

Jan 2018

Training:

1 GPU day

GPT

June 2018

Training

240 GPU days

BERT

Oct 2018

Training

256 TPU days

~320–560 GPU days

GPT-2

Feb 2019

Training

~2048 TPU v3 days according to a reddit thread

fast.ai

OpenAI

Google AI

OpenAI

BERT (Bidirectional Encoder Representations from Transformers)

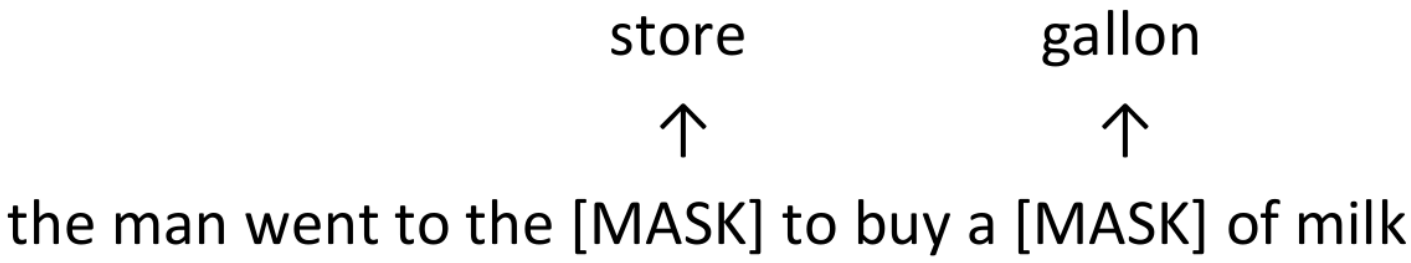
- Idea: combine ideas from ELMO, ULMFit and the Transformer [Kenton and Toutanova, 2019].
- How: Train a large model (335 million parameters) from a large unlabeled corpus using a Transformer encoder and then fine-tune it for other downstream tasks.
- The parallelizable properties of the Transformer (unlike RNNs, which must be processed sequentially) allow the model to scale to more parameters.
- This model is related but a little bit different from a standard Language Model.

BERT (Bidirectional Encoder Representations from Transformers)

- BERT doesn't predict the next word in a sentence like a traditional language model, but rather learns utilizes a "**masked language modeling**" (MLM) objective during pre-training.
- In MLM, random words in a sentence are masked and the model is trained to predict those masked words based on the surrounding context.
- BERT also incorporates a "**next sentence prediction**" task, where pairs of sentences are fed to the model, and it learns to predict whether the second sentence follows the first in the original text.
- **Fine-tuning** BERT involves adding a task-specific layer on top of the pre-trained model and training it on a labeled dataset for the target task.
- BERT achieved state-of-the-art results at the time of its release on NLP tasks, including sentence classification, named entity recognition, question answering, and more.

Masked Language Modeling and Next Sentence Prediction

- MLM: Mask out k% of the input words, and then predict the masked words
- They always use k = 15%.



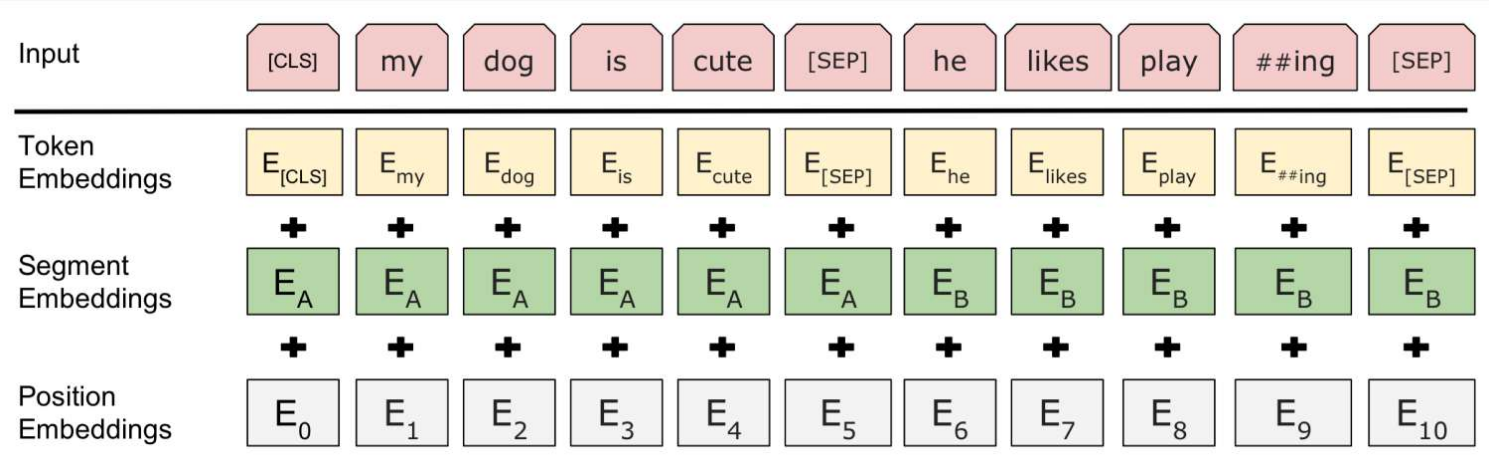
- Too little masking: Too expensive to train
- Too much masking: Not enough context

Masked Language Modeling and Next Sentence Prediction

- Next sentence prediction: To learn relationships between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence

BERT sentence pair encoding

- Token embeddings: Words are divided into smaller units called word pieces, and each word piece is assigned a token embedding.
- BERT learns a segmented embedding [SEP] to differentiate between the two sentences in a pair.
- BERT utilizes positional embeddings to capture the position of each word within the sentence.

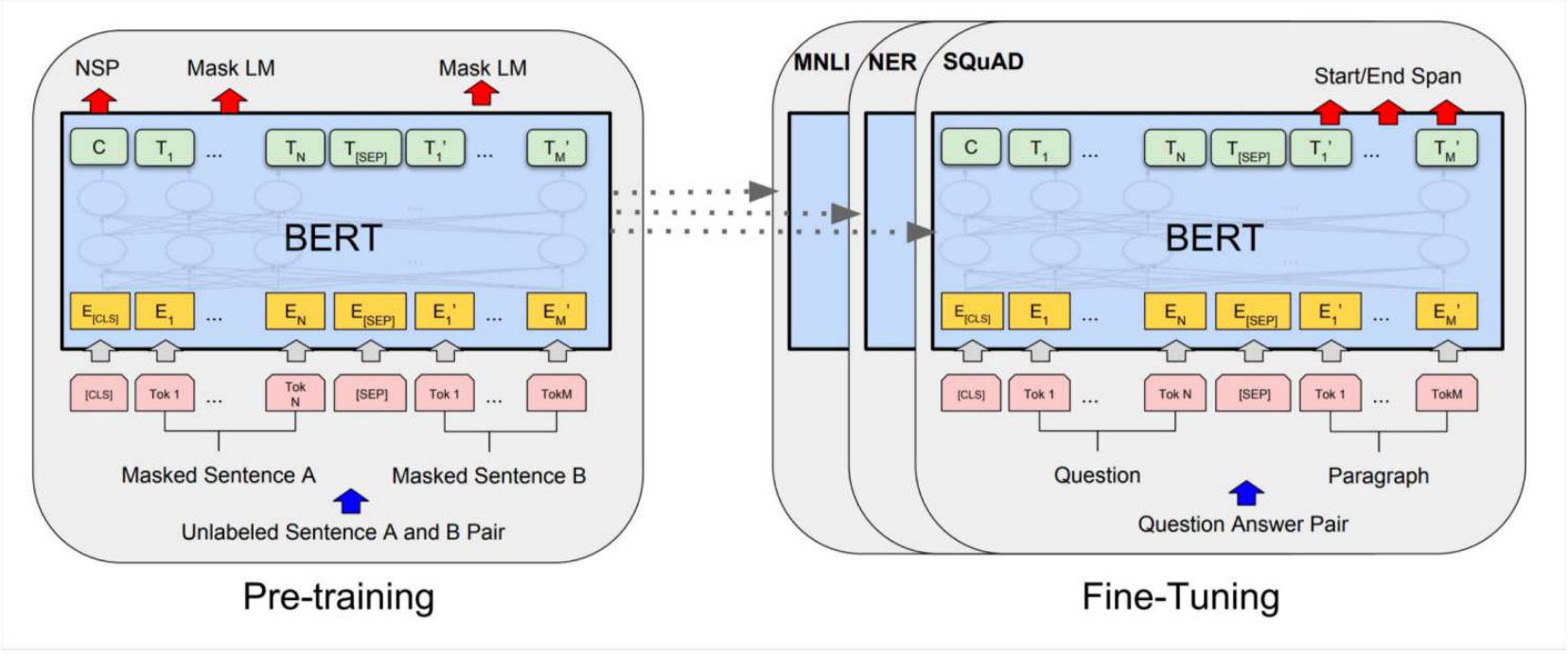


BERT Model Architecture and Training

- BERT is based on the Transformer encoder.
- The multi-headed self-attention block of the Transformer allows BERT to consider long-distance context effectively.
- The use of self-attention also enables efficient computations on GPU/TPU, with only a single multiplication per layer.
- BERT was trained on a large amount of unlabeled text data from Wikipedia and BookCorpus.
- Two different model sizes were trained:
  1. BERT-Base: 12 layers, 768 hidden units, and 12 attention heads.
  2. BERT-Large: 24 layers, 1024 hidden units, and 16 attention heads.
- The training process involved utilizing 4x4 or 8x8 TPU (Tensor Processing Unit) configurations for faster computation.
- Training BERT models took approximately 4 days to complete.

BERT model fine tuning

- Fine-tuning involves customizing the pre-trained BERT model for specific tasks.
- To fine-tune BERT, we add a task-specific layer on top of the pre-trained BERT model.
- The task-specific layer can vary depending on the task at hand, such as sequence labeling or sentence classification.
- We train the entire model, including the pre-trained BERT and the added task-specific layer, for the specific task.



BERT results on GLUE tasks

- BERT was massively popular and hugely versatile; finetuning BERT led to new state-of- the-art results on a broad range of tasks.
- BERT's performance was assessed using the GLUE benchmark, a collection of diverse NLP tasks.
- The GLUE benchmark primarily consists of natural language inference tasks, but also includes sentence similarity and sentiment analysis tasks.

Example Task: MultiNLI (Natural Language Inference)

- Premise: "Hills and mountains are especially sanctified in Jainism."
- Hypothesis: "Jainism hates nature."
- Label: Contradiction

Example Task: CoLa

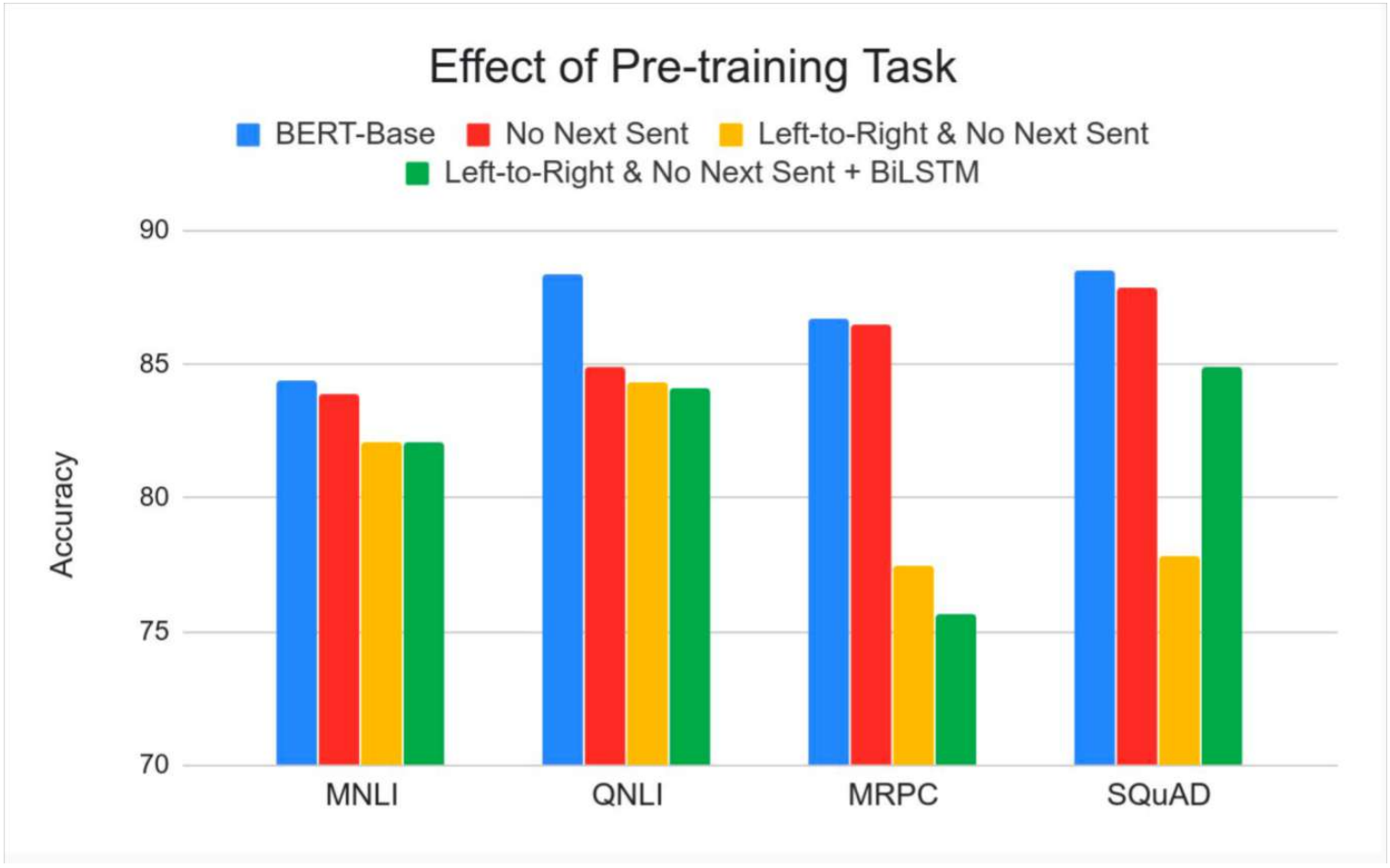
- Sentence: "The wagon rumbled down the road."
- Label: Acceptable
- Sentence: "The car honked down the road."
- Label: Unacceptable

BERT results on GLUE tasks

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	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

- QQP: Quora Question Pairs (detect paraphrase questions)
- QNLI: natural language inference over question answering data
- SST-2: sentiment analysis
- CoLA: corpus of linguistic acceptability (detect whether sentences are grammatical.)
- STS-B: semantic textual similarity
- MRPC: microsoft paraphrase corpus
- RTE: a small natural language inference corpus

BERT Effect of pre-training task



Pre-training decoders GPT and GPT-2

- Contemporary to BERT, OpenAI introduced an alternative approach called Generative Pretrained Transformer (GPT) [Radford et al.,].
- The idea behind GPT is to train a large standard language model using the generative part of the Transformer, specifically the decoder.
- GPT is a Transformer decoder with 12 layers and 117 million parameters.
- It has 768-dimensional hidden states and 3072-dimensional feed-forward hidden layers.
- GPT utilizes byte-pair encoding with 40,000 merges to handle subword units.
- GPT was trained on BooksCorpus, which consists of over 7,000 unique books.
- OpenAI later introduced GPT-2, a larger version with 1.5 billion parameters, trained on even more data.
- GPT-2 has been shown to generate relatively convincing samples of natural language.

GPT-2 language model (cherry-picked) output

Human provided prompt:
In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.
Model Completition:
The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.
Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.
Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

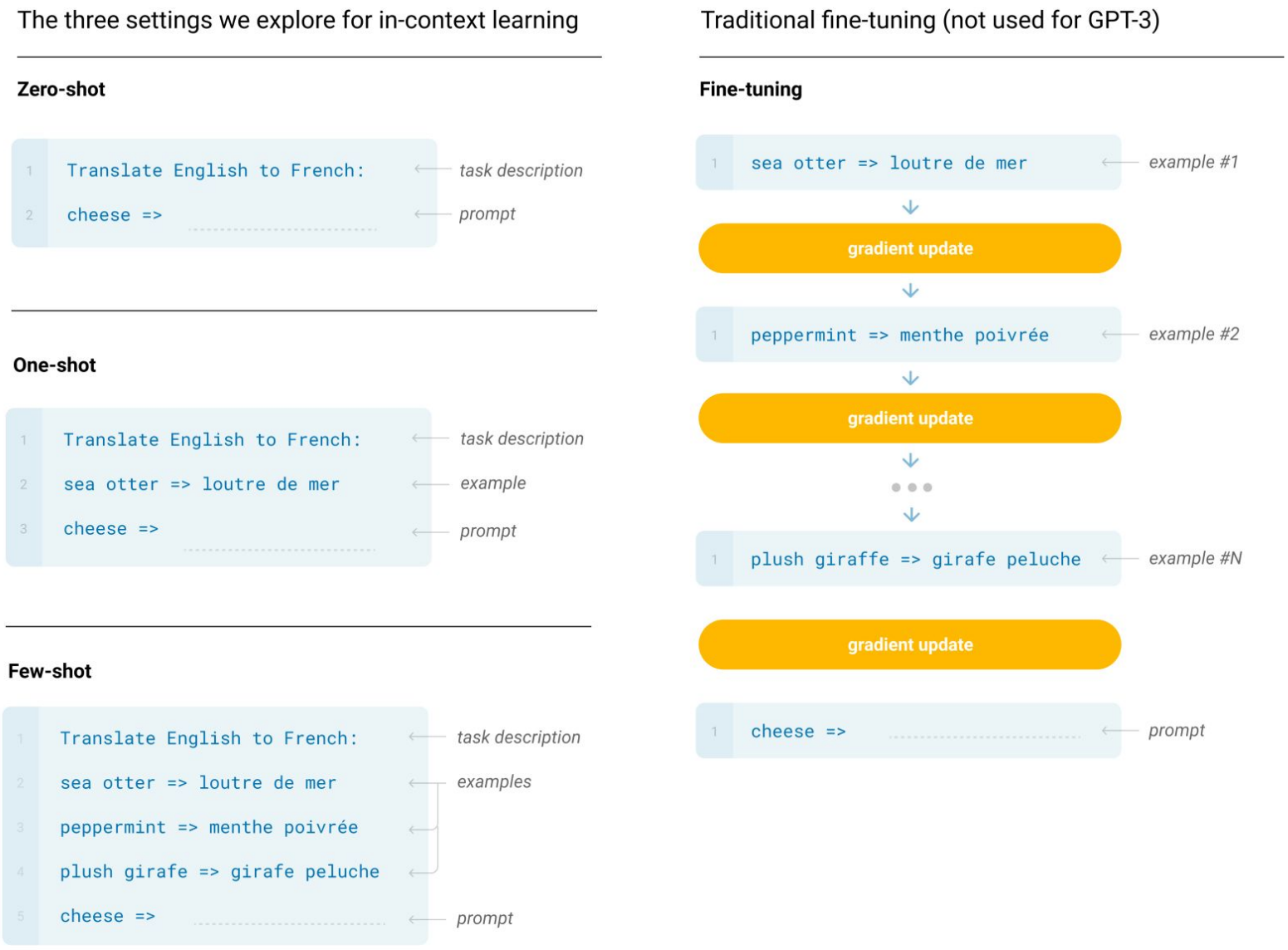
Phase Change: GPT-3 (2020)

- GPT-3 is another Transformer-based Language Model (LM) that pushed the boundaries with nearly 200 billion parameters, making it the largest model at the time [**?**].
- It was trained on a massive corpus consisting of nearly 500 billion words.
- **In-context learning**: GPT-3 demonstrated the ability to solve various natural language processing (NLP) tasks using **zero-shot**, **one-shot** and **few-shot learning**.
- The key to this capability lies in the prompt or context provided to GPT-3.
- GPT-3 demonstrated the ability to solve various tasks without performing gradient updates to the base model.

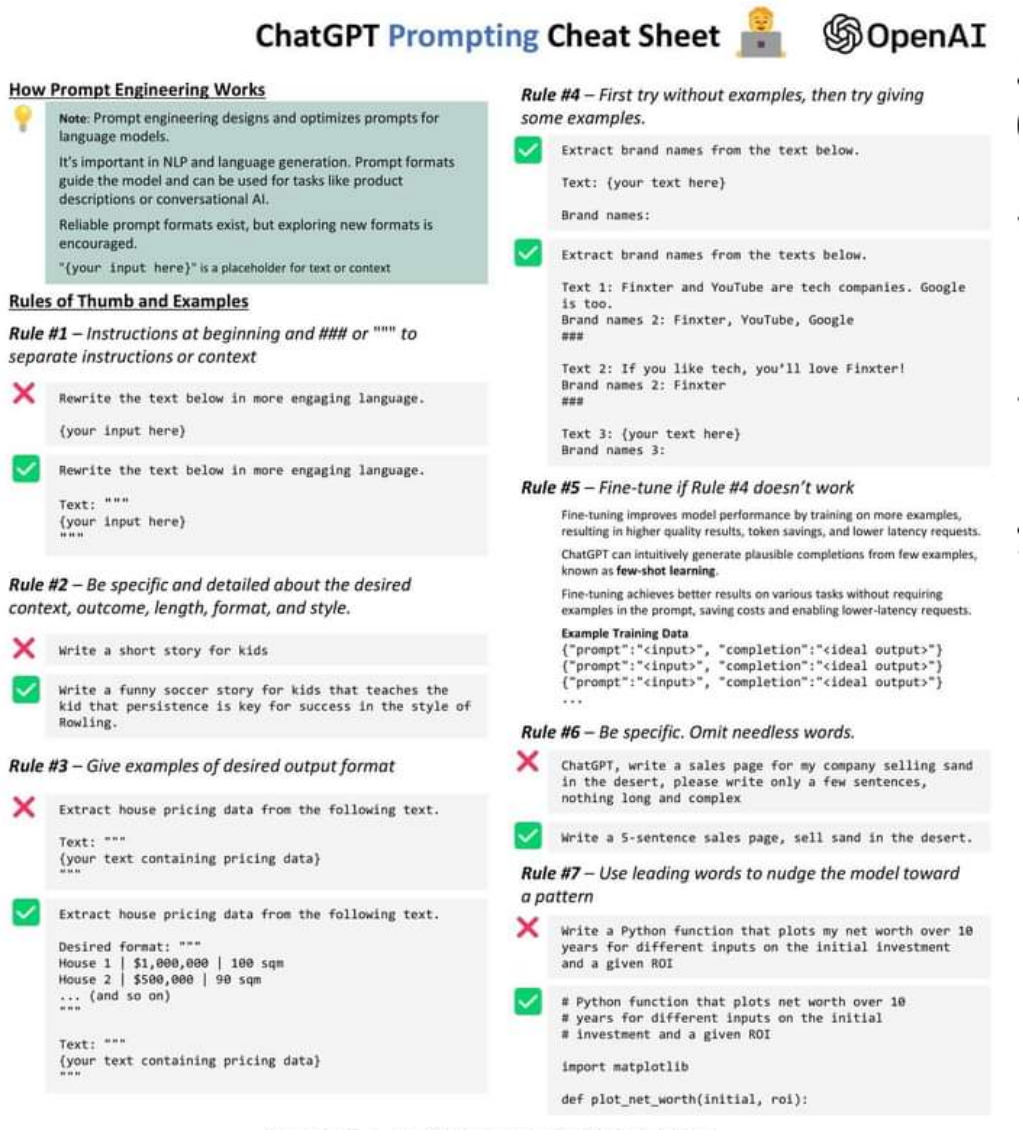
Zero-shot, One-shot, and Few-shot Learning with GPT-3

- GPT-3 showcased remarkable capabilities in NLP tasks through different learning paradigms: zero-shot, one-shot, and few-shot learning.
- **Zero-shot learning**: With zero-shot learning, GPT-3 can tackle tasks it has never seen before without any specific training. It achieves this by providing a prompt or instruction to guide its generation process. For example, by providing GPT-3 with a prompt like, ”Translate this English sentence to French,” it can generate the translated sentence without any explicit training for translation tasks.
- **One-shot learning**: In one-shot learning, GPT-3 can perform a task by adding a single input-output pair to the instruction.
- **Few-shot learning**: providing a limited number input-outpur pairs after the instruction in the prompt.

Zero-shot, One-shot, and Few-shot Learning with GPT-3



Prompt Engineering



ChatGPT Prompting Cheat Sheet

OpenAI

How Prompt Engineering Works

Note: Prompt engineering designs and optimizes prompts for language models.

It's important in NLP and language generation. Prompt formats guide the model and can be used for tasks like product descriptions or conversational AI.

Reliable prompt formats exist, but exploring new formats is encouraged.

"{your input here}" is a placeholder for text or context

Rules of Thumb and Examples

***Rule #1** – Instructions at beginning and ### or """ to separate instructions or context*

✗
```
Rewrite the text below in more engaging language.

{your input here}
```

✓
```
Rewrite the text below in more engaging language.

Text: """
{your input here}
"""
```

***Rule #2** – Be specific and detailed about the desired context, outcome, length, format, and style.*

✗
```
Write a short story for kids
```

✓
```
Write a funny soccer story for kids that teaches the
kid that persistence is key for success in the style of
Rowling.
```

***Rule #3** – Give examples of desired output format*

✗
```
Extract house pricing data from the following text.

Text: """
{your text containing pricing data}
"""
```

✓
```
Extract house pricing data from the following text.

Desired format: """
House 1 | $1,000,000 | 100 sqm
House 2 | $500,000 | 90 sqm
... (and so on)
"""

Text: """
{your text containing pricing data}
"""
```

***Rule #4** – First try without examples, then try giving some examples.*

✓
```
Extract brand names from the text below.

Text: {your text here}

Brand names:
```

✓
```
Extract brand names from the texts below.

Text 1: Finxter and YouTube are tech companies. Google
is too.
Brand names 2: Finxter, YouTube, Google
###

Text 2: If you like tech, you'll love Finxter!
Brand names 2: Finxter
###

Text 3: {your text here}
Brand names 3:
```

***Rule #5** – Fine-tune if Rule #4 doesn't work*

Fine-tuning improves model performance by training on more examples, resulting in higher quality results, token savings, and lower latency requests.

ChatGPT can intuitively generate plausible completions from few examples, known as **few-shot learning**.

Fine-tuning achieves better results on various tasks without requiring examples in the prompt, saving costs and enabling lower-latency requests.

Example Training Data
```
{"prompt":"<input>", "completion":"<ideal output>"}
{"prompt":"<input>", "completion":"<ideal output>"}
{"prompt":"<input>", "completion":"<ideal output>"}
...
```

***Rule #6** – Be specific. Omit needless words.*

✗
```
ChatGPT, write a sales page for my company selling sand
in the desert, please write only a few sentences,
nothing long and complex
```

✓
```
Write a 5-sentence sales page, sell sand in the desert.
```

***Rule #7** – Use leading words to nudge the model toward a pattern*

✗
```
Write a Python function that plots my net worth over 10
years for different inputs on the initial investment
and a given ROI
```

✓
```
# Python function that plots net worth over 10
# years for different inputs on the initial
# investment and a given ROI

import matplotlib

def plot_net_worth(initial, roi):
```

Questions?

Thanks for your Attention!

References I


Howard, J. and Ruder, S. (2018).
Universal language model fine-tuning for text classification.
In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia. Association for Computational Linguistics.


Kenton, J. D. M.-W. C. and Toutanova, L. K. (2019).
Bert: Pre-training of deep bidirectional transformers for language understanding.
In *Proceedings of NAACL-HLT*, pages 4171–4186.


Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018).
Deep contextualized word representations.
In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.


Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al.
Improving language understanding by generative pre-training.