GPT-3

Adaptation:

- a natural language description of the task
- a set of training instances (input-output pairs)

## Two Ways

Supervised Training:

- Probing: create a new model that uses the LM as features
- Fine-turning: Starting with the language model and updating it based on the training instances

## Prompting (in-context learning):

Construct a prompt (a string based on the description and training instances) or a set of prompts, feed those into a language model to obtain completions

## Language Modeling:

probability distribution over sequences of tokens

$$p(x_{1:L}) = \prod_{i=1}^{L} p(x_i \mid x_{1:i-1})$$

## Perplexity 困惑度

geometric average

code length

$$\text{perplexity}_p(x_{1:L}) = e^{\frac{1}{L}\sum_{i=1}^{L} \log\left(\frac{1}{p(x_i \mid x_{1:i-1})}\right)}$$

average "branching factor" per token

**Tale of two errors**. There are two types of errors a language model can make, and perplexity treats them asymmetrically:

- **Recall error**: The language model fails to place probability mass on some token. Perplexity has no mercy:

$$p(\text{ate} \mid \text{the}, \text{mouse}) \to 0 \quad \Rightarrow \quad \text{perplexity}_p(\text{the}, \text{mouse}, \text{ate}, \text{the}, \text{cheese}) \to \infty.$$

- **Precision error**: The language model places extra probability mass on some bad sequences. Perplexity provides a slap on the wrist. Given a language model $p$, suppose we mix in some garbage distribution $r$ with probability $\epsilon$:

$$q(x_i \mid x_{1:i-1}) = (1 - \epsilon)p(x_i \mid x_{1:i-1}) + \epsilon r(x_i \mid x_{1:i-1}).$$

Then we can compute the perplexity of $x_{1:L}$ under $q$:

$$\text{perplexity}_q(x_{1:L}) \le \frac{1}{1 - \epsilon}\text{perplexity}_p(x_{1:L}) \approxeq (1 + \epsilon)\text{perplexity}_p(x_{1:L}),$$

where the last approximate equality holds for small values of $\epsilon$. If we mix in 5% junk, then perplexity only by 5%. Note that the resulting language is horrible for generation, since every 20 tokens on average it's just going to generate a gibberish token.

Now let's get on with evaluating perplexity on an actual dataset.

Perplexity:

### Penn Tree Bank

The Penn Tree Bank is a classic dataset in NLP, originally annotated for syntactic parsing. Beginning with Emami and Jelinek (2004) and Mikolov and Zweig (2012), a version of the dataset that only contained Wall Street Journal articles was used as a language modeling evaluation. Note that the PTB language modeling benchmark involved some significant preprocessing of the original dataset (h/t to John Hewitt for pointing this out).

**Adaptation**. Feed the entire text as a prompt into GPT-3 and evaluate the perplexity (demo):

> *Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29. Mr. Vinken is chairman of Elsevier N.V., the Dutch publishing group.*

**Results**. GPT-3 vastly outperforms the existing state-of-the-art:

| Model | Perplexity |
| --- | --- |
| GPT-3 | **20.5** |
| BERT-Large-CAs1 | 31.3 |

See the leaderboard for the latest results.

**Train/test leakage**. The authors did not evaluate on some datasets such as WikiText-103 because GPT-3 was trained on Wikipedia. PTB had the advance of predating the Internet, and is only available through a paid license. This is another complication with large datasets: it is difficult to check that your test data did not appear in your training data and was memorized.

# LAMBADA (2016)

LAMBADA ([Paperno et al. 2016](#))

- Task: predict the last word of a sentence.
- Motivation: Solving the task requires modeling **long-range dependencies**.

**Adaptation**.

- LAMBADA is natively already a language modeling task, so we could just ask a language model to complete the final word of the sentence.
- Problem: language model doesn't know it should be producing the final word of the sentence.
- Solution: frame it more explicitly as a input-output mapping and use in-context learning with additional examples (demo):

> *Fill in blank:*
>
> *Alice was friends with Bob. Alice went to visit her friend ___. -> Bob*
>
> *She held the torch in front of her.*
> *She caught her breath.*
> *"Chris? There's a step."*
> *"What?"*
> *"A step. Cut in the rock. About fifty feet ahead." She moved faster. They both moved faster. "In fact," she said, raising the torch higher, "there's more than a ___. -> step*

**Results**. GPT-3 does *much better* on this task than the previous state-of-the-art (based on GPT-2):

| Model | Perplexity |
|---|---|
| GPT-3 (few-shot) | **1.92** |
| SOTA | 8.63 |

See the leaderboard for the latest results.

# HellaSwag (2019)

- Motivation: evaluate a model's ability to perform commonsense reasoning
- Task: choose the most appropriate completion for a sentence from a list of choices

**Adaptation**. This is a **multiple-choice task**, so the most natural thing to do is to **score** each candidate answer with the language model and predict the "best" one (demo):

> *Making a cake: Several cake pops are shown on a display. A woman and girl are shown making the cake pops in a kitchen. They ${answer}*

where ${answer} is one of:

1. *bake them, then frost and decorate.*
2. *taste them as they place them on plates.*
3. *put the frosting on the cake as they pan it.*
4. *come out and begin decorating the cake as well.*

How do you score a candidate answer $\mathrm{y}$ given a question $\mathrm{x}$? There's no principled answer, but here are some **heuristics**:

1. Unnormalized probability: $\mathrm{score}(\mathrm{x}, \mathrm{y}) = p(\mathrm{x}, \mathrm{y})$. The problem with the unnormalized probability is that it has a bias towards short answers (demo).
2. Length-normalized probability: $\mathrm{score}(\mathrm{x}, \mathrm{y}) = \frac{p(\mathrm{x},\mathrm{y})}{\text{num-tokens}(\mathrm{y})}$. This fixes the length bias. However, given two answers of the same length, the model still might prefer the more popular entity.
3. Frequency-normalized probability: $\mathrm{score}(\mathrm{x}, \mathrm{y}) = \frac{p(\mathrm{y} \mid \mathrm{x})}{p(\mathrm{y} \mid \mathrm{x}_0)}$, where $\mathrm{x}_0$ is a neutral string like Answer:. This lowers the score for answers that happen to just be common (e.g., \nl{John}). Compare demo versus demo.

**Results**. GPT-3 got close but did not exceed the state-of-the-art:

| Model | Accuracy |
|---|---|
| SOTA | **85.6** |
| GPT-3 | 79.3 |

However, the SOTA used fine-tuning on the HellaSwag training set, so it is pretty impressive that GPT-3 can get close without any task-specific training data!

Question Answering:

## Question answering

Now we consider (closed-book) question answering, where the input is a question and the output is an answer. The **language model has to somehow "know" the answer** without looking up information in a database or a set of documents (we'll consider reading comprehension later, where the information is provided).

> Input: What school did burne hogarth establish?
> Output: School of Visual Arts

### TriviaQA (Joshi et al. 2017)

- Task: given a trivia question, generate the answer
- The original dataset was collected from trivial enthusiasts and was presented as a challenge used for (open book) reading comprehension, but we use it for (closed-book) question answering.

**Adaptation**. We define a prompt based on the training instances (if any) and the question, and take the completion as the predicted answer (demo):

> *Q: 'Nude Descending A Staircase' is perhaps the most famous painting by which 20th century artist?*
> *A: Marcel Duchamp*

**Results**.

| Model | Accuracy |
|---|---|
| RAG | 68.0 |
| GPT-3 (zero-shot) | 64.3 |
| GPT-3 (few-shot) | **71.2** |

"in-context learning" ("Meta-Learning"
unsupervised pretraining)

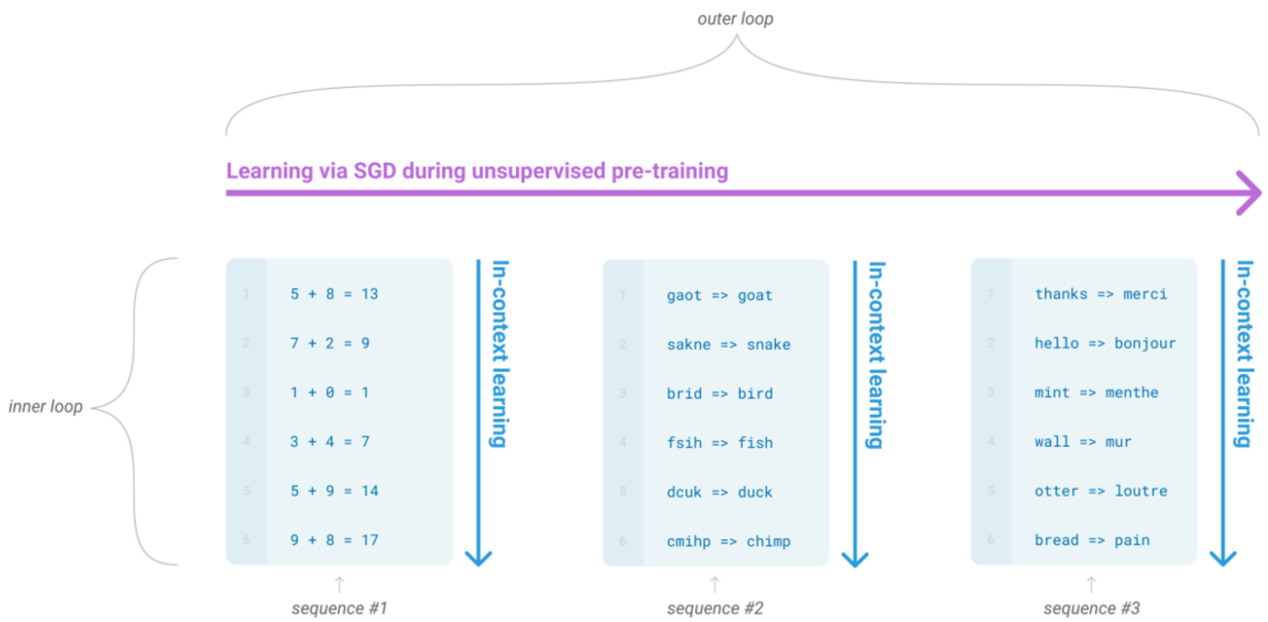


**Figure 1.1: Language model meta-learning.** During unsupervised pre-training, a language model develops a broad set of skills and pattern recognition abilities. It then uses these abilities at inference time to rapidly adapt to or recognize the desired task. We use the term "in-context learning" to describe the inner loop of this process, which occurs within the forward-pass upon each sequence. The sequences in this diagram are not intended to be representative of the data a model would see during pre-training, but are intended to show that there are sometimes repeated sub-tasks embedded within a single sequence.

| Model Name | $n_{\text{params}}$ | $n_{\text{layers}}$ | $d_{\text{model}}$ | $n_{\text{heads}}$ | $d_{\text{head}}$ | Batch Size | Learning Rate |
|---|---|---|---|---|---|---|---|
| GPT-3 Small | 125M | 12 | 768 | 12 | 64 | 0.5M | $6.0 \times 10^{-4}$ |
| GPT-3 Medium | 350M | 24 | 1024 | 16 | 64 | 0.5M | $3.0 \times 10^{-4}$ |
| GPT-3 Large | 760M | 24 | 1536 | 16 | 96 | 0.5M | $2.5 \times 10^{-4}$ |
| GPT-3 XL | 1.3B | 24 | 2048 | 24 | 128 | 1M | $2.0 \times 10^{-4}$ |
| GPT-3 2.7B | 2.7B | 32 | 2560 | 32 | 80 | 1M | $1.6 \times 10^{-4}$ |
| GPT-3 6.7B | 6.7B | 32 | 4096 | 32 | 128 | 2M | $1.2 \times 10^{-4}$ |
| GPT-3 13B | 13.0B | 40 | 5140 | 40 | 128 | 2M | $1.0 \times 10^{-4}$ |
| GPT-3 175B or "GPT-3" | 175.0B | 96 | 12288 | 96 | 128 | 3.2M | $0.6 \times 10^{-4}$ |

| Dataset | Quantity (tokens) | Weight in training mix | Epochs elapsed when training for 300B tokens |
|---|---|---|---|
| Common Crawl (filtered) | 410 billion | 60% | 0.44 |
| WebText2 | 19 billion | 22% | 2.9 |
| Books1 | 12 billion | 8% | 1.9 |
| Books2 | 55 billion | 8% | 0.43 |
| Wikipedia | 3 billion | 3% | 3.4 |

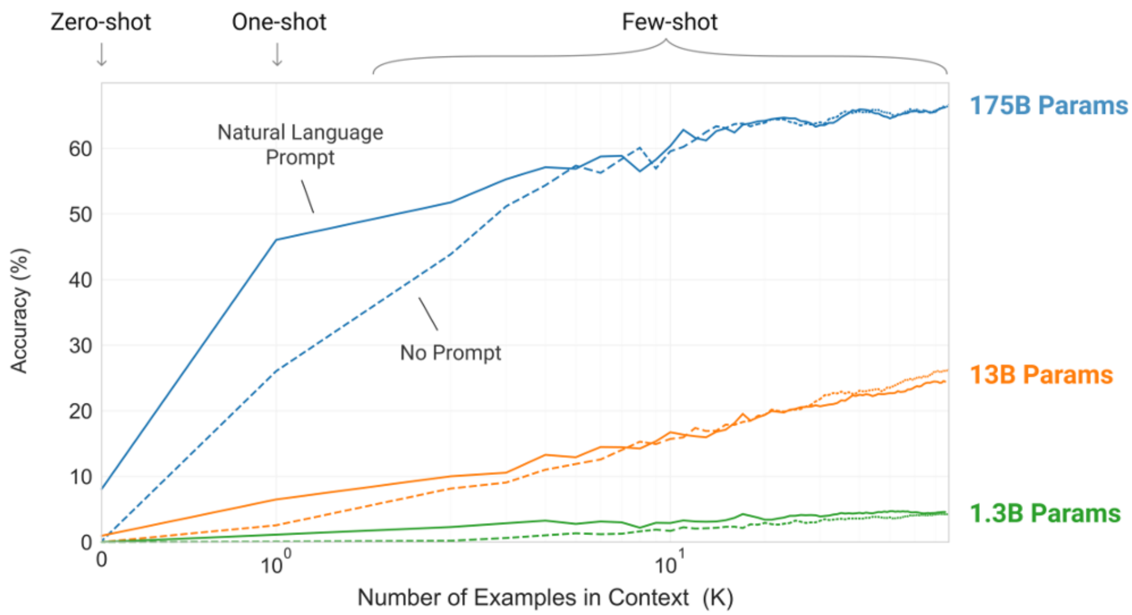


**Figure 1.2: Larger models make increasingly efficient use of in-context information.** We show in-context learning performance on a simple task requiring the model to remove random symbols from a word, both with and without a natural language task description (see Sec. 3.9.2). The steeper "in-context learning curves" for large models demonstrate improved ability to learn a task from contextual information. We see qualitatively similar behavior across a wide range of tasks.

"in-context information"
(contextual)

Emergence / Homogenization