

A Universal API: Language Modeling to solve many tasks

The Language Modeling objective seems simple, but appearances are deceiving.

We have shown that a Language Model can be adapted for new Target tasks

- via the Unsupervised Pre-Training + Supervised Fine-Tuning paradigm.

There is, perhaps, an alternative way to adapt to a new Target task

- turn each task *directly* into an instance of the Language Modeling objective.

Consider a new Target task: Entailment

- Input is a *pair* of text sequences [Premise, Hypothesis]
- Binary classification: Does the Hypothesis Logically follow from the Premise ?

Premise: The students are attending a lecture on Machine Learning

Hypothesis: The students are sitting in a class room

Label: Entails

Suppose we construct a sequence for the Language Model to extend

- consisting of the Premise, Hypothesis, and the string "Label:"

We call this sequence the *prompt* or the *context*

Premise: The students are attending a lecture on Machine Learning

Hypothesis: The students are sitting in a class room

Label:

We would hope that the Language Model extends the prompt by creating tokens that are the correct response for the Target task

Entail

since the Hypothesis logically follows from the Premise (other possible response: Not Entail)

We have turned Entailment into an instance of Language Modeling.

As another example: consider the Target task: Multiple Choice Question answering

- Input is
 - Context: a sentence or paragraph stating facts
 - Question
 - Answers: a set of possible answer sentences

We construct a prompt of the form

Context: It is December of 2022. Prof. Perry is teaching the second half of the Machine Learning Course.

Question: Where are the students ?

Answer 1: The beach

Answer 2: In a classroom in Brooklyn

Answer 3: Dreaming of being elsewhere.

Label:

and hope that the Language Model extends the prompt string with one of

{ Answer 1, Answer 2, Answer 3 }

- hopefully with probability near 100% for Answer 2 !

Power of text as the Universal API

The Universal API turns each task into an instance of the Language Modeling task.

What this means is that

- we might not need *task-specific* models
- just transform every task into instance of the "predict the next" word task

After converting our Source task into a LLM task

- we would ordinarily use Supervised Fine-Tuning
- on a dataset of (transformed) examples from the Target Task
- to Fine-Tune the LLM for the Target task

The transformed Target task training dataset would consist of examples like

$$\langle \mathbf{x}^{(i)}, \mathbf{y}^{(i)} \rangle = \langle \text{prompt}, \text{response} \rangle$$

That is

- the "features" turned into text ("prompt")
- the label ("response") is the continuation of the text

In-Context Learning

The Universal API also opens up an interesting possibility

- Can we use a Large Language Model
- To solve a new Target task
- *without* further training (i.e., Fine-Tuning)

On a pure syntax level, this is feasible

- all problems are instances of "predict the next"
- perhaps the LLM's text completion power *also* captures domain-specific knowledge
 - the completion is related to the domain-specific words of the prompt
 - for example, if the prompt is in French, the completion would be expected to be in French too

So the possibility is there.

But how does the Source LLM discern what the Target task is ?

The idea behind *In-Context Learning* is to

- condition the Pre-Trained Language Model (Source LLM)
- to complete the prompt of a new Target Task
- with the correct response for the Target Task
- **without** further training (Fine-Tuning)
- merely by providing the examples that demonstrate the behavior of the new Target task
- **as parts of the prompt**

The examples demonstrating the Target task are called *exemplars*.

- each exemplar is provided as a prompt and response pair, as per the Universal API

$$\langle \mathbf{x}^{(i)}, \mathbf{y}^{(i)} \rangle = \langle \text{prompt}, \text{response} \rangle$$

To predict the response for a new prompt \mathbf{x}

- the exemplars are concatenated together (the *Context C*)
- the Context is a demonstration of the Target Task's relationship between features and label

The new prompt \mathbf{x} is appended to the Context

- the Pre-Trained model is expected to complete the prompt
- by providing a response specific to the Target task and the prompt \mathbf{x}

For example, we can describe Translation between languages with the following Context
C

Translate English to French

sea otter => loutre de mer

peppermint => menthe poivree

plush giraffe => girafe peluche

The expectation is that when the user presents the prompt x

cheese =>

the model will respond with the French translation of cheese .

- the "next words" predicted by the Language Modeling

More formally:

- Let C ("context") denote the pre-prompt.
- Let \mathbf{x} denote the "query" (e.g., cheese =>)

The unconditional Language Modeling objective

$$p(\mathbf{y}|\mathbf{x})$$

is to create the sequence \mathbf{y} that follows the sequence of prompt \mathbf{x} .

Here, the pre-prompt conditions the model's objective

$$p(\mathbf{y}|C, \mathbf{x})$$

to create the sequence \mathbf{y} that follows from the exemplars C and prompt \mathbf{x} .

This is just a mechanical process

- create the sequence $\dot{\mathbf{x}}$
- by concatenating some number k of exemplars: $\langle \mathbf{x}^{(1)}, \mathbf{y}^{(1)} \rangle, \dots, \langle \mathbf{x}^{(k)}, \mathbf{y}^{(k)} \rangle$
- and prompt string \mathbf{x}
- delimiting elements by separator characters $\langle \text{SEP}_1 \rangle, \langle \text{SEP}_2 \rangle$

$$\begin{aligned} \dot{\mathbf{x}} = \text{concat}(& \mathbf{x}^{(1)}, \langle \text{SEP}_1 \rangle, \mathbf{y}^{(1)}, \langle \text{SEP}_2 \rangle, \\ & \vdots \\ & \mathbf{x}^{(k)}, \langle \text{SEP}_1 \rangle, \mathbf{y}^{(k)}, \langle \text{SEP}_2 \rangle, \\ & \mathbf{x} \\ &) \end{aligned}$$

The LLM then computes

$$p(\mathbf{y}|\dot{\mathbf{x}})$$

For convenience, we will just write this as the conditional probability

$$p(\mathbf{y}|\mathbf{x}, C)$$

In-Context learning: let's experiment

The [HuggingFace platform \(https://huggingface.co/\)](https://huggingface.co/) has libraries of pre-trained models for many tasks, including Language models.

There is a clean API for using these models in code (I recommend their on-line [course \(https://huggingface.co/\)](https://huggingface.co/) if you want to play with it).

But they also host many of their models for interactive use.

This is valuable not just for the obvious reason of ease of use

- some models are too big to load on the machines available to us

For fun, let's try using In-Context learning in order to get a Pre-Trained Language model to classify whether a short movie review is positive or negative.

Movie review sentiment: few shot learning GPT-2 (<https://huggingface.co/gpt2?text=this+movie+was+great%3A+positive%0A%0A+one+of+the+best+films+of+the+year>)

Movie review sentiment: few shot learning GPT-J 6B (<https://huggingface.co/EleutherAI/gpt-j-6B?text=this+movie+was+great%3A+positive%0A%0A+one+of+the+best+films+of+the+year>)

Movie review sentiment: few shot learning: gpt-neox-20b (<https://huggingface.co/EleutherAI/gpt-neox-20b?text=this+movie+was+great%3A+positive%0A%0A+one+of+the+best+films+of+the+year>)

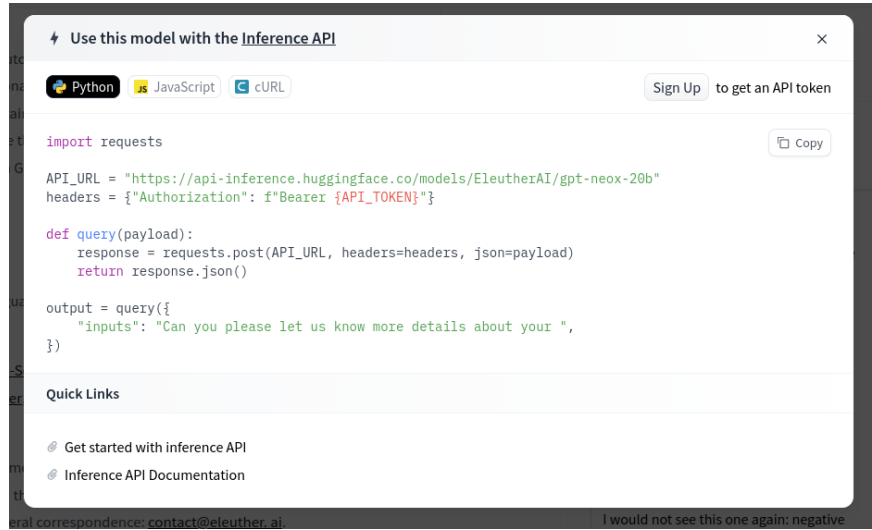
You can try cutting and pasting the prompt into the hosted inference instance of other models.

You can play with In-Context learning by going to the page of a model and typing into the *Hosted Inference API* text box.

But there is also an API that allows you to pass the input (context plus prompt) via a URL.

If you click on the `Deploy` button and choose the `Inference API` drop-down

- you will see Python code for querying the model programatically.



The screenshot shows a web interface for using the Inference API. At the top, there's a title "Use this model with the Inference API" and a close button. Below the title, there are three tabs: "Python" (selected), "JavaScript", and "cURL". To the right of the tabs is a "Sign Up" button with the text "to get an API token". The main area contains Python code for querying the API. The code imports the 'requests' library, sets the API_URL to "https://api-inference.huggingface.co/models/EleutherAI/gpt-neox-20b", and sets the headers to {"Authorization": f"Bearer {API_TOKEN}"}. A 'query' function is defined that takes a payload and returns the JSON response. The code then calls 'query' with a payload containing the input "Can you please let us know more details about your ". Below the code is a "Copy" button. At the bottom, there's a "Quick Links" section with two links: "Get started with inference API" and "Inference API Documentation".

```
import requests

API_URL = "https://api-inference.huggingface.co/models/EleutherAI/gpt-neox-20b"
headers = {"Authorization": f"Bearer {API_TOKEN}"}

def query(payload):
    response = requests.post(API_URL, headers=headers, json=payload)
    return response.json()

output = query({
    "inputs": "Can you please let us know more details about your ",
})
```

Quick Links

- Get started with inference API
- Inference API Documentation

Please note

- our toy example above used a *single* test example
- even if we manage to get a correct prediction on a single example
 - we don't have confidence that the new task was successfully learned !
 - we really should evaluate success on a larger number of text examples
- still: the fact that the exemplars taught the model the correct syntax for an answer is exciting

Learning to learn

Does In-Context learning really work ?

We can begin to answer this question by

- examining the behavior of a Pre-Trained LLM
- on a new task
- using k exemplars
 - varying k

Depending on k , we refer to the behavior of the LLM by slightly different names

- **Few shot learning:** $10 \leq k \leq 100$ typically
- **One shot learning:** $k = 1$
- **Zero shot learning** $k = 0$

A picture will help

The three settings we explore for in-context learning

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
1 Translate English to French: ← task description
2 cheese => ..... ← prompt
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
1 Translate English to French: ← task description
2 sea otter => loutre de mer ← example
3 cheese => ..... ← prompt
```

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

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

Traditional fine-tuning (not used for GPT-3)

Fine-tuning

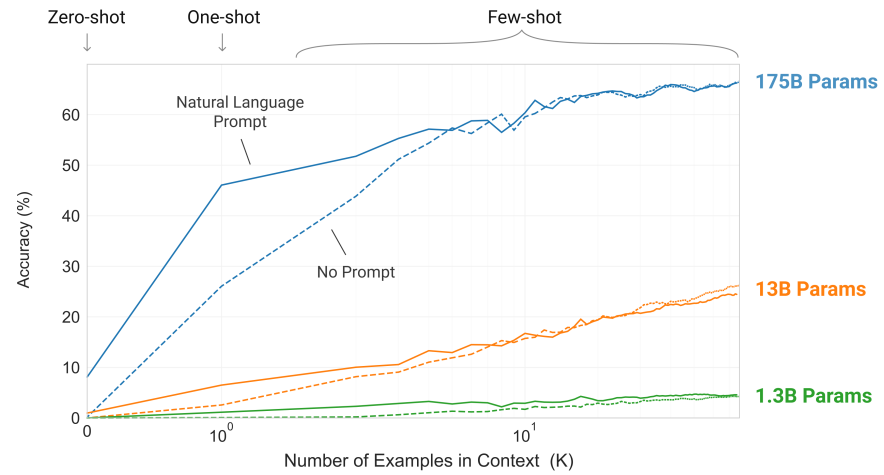
The model is trained via repeated gradient updates using a large corpus of example tasks.



Is this even possible ?! Learning a new task with **zero** exemplars ?

Let's look at the reported In-Context Learning results of 3 LLM's of varying size.

Few/One/Zero shot learning



Picture from: <https://arxiv.org/pdf/2005.14165.pdf>

A couple of observations

- As the size of the model grows: In-Context Learning behavior improves
 - compare the 175 Billion parameter model to the smaller models
 - we sometimes refer to this as behavior that "emerges" only when a model is sufficiently large
- More exemplars (greater k) helps
 - but not much for the smallest model
- Zero shot learning works !
 - but this is a behavior that only emerges for very large models

In [2]: `print("Done")`

Done

