OpenAI

Fine-tuning

Technique

February 2024

Overview

Fine-tuning involves adjusting the parameters of pre-trained models on a specific dataset or task. This process enhances the model's ability to generate more accurate and relevant responses for the given context by adapting it to the nuances and specific requirements of the task at hand.

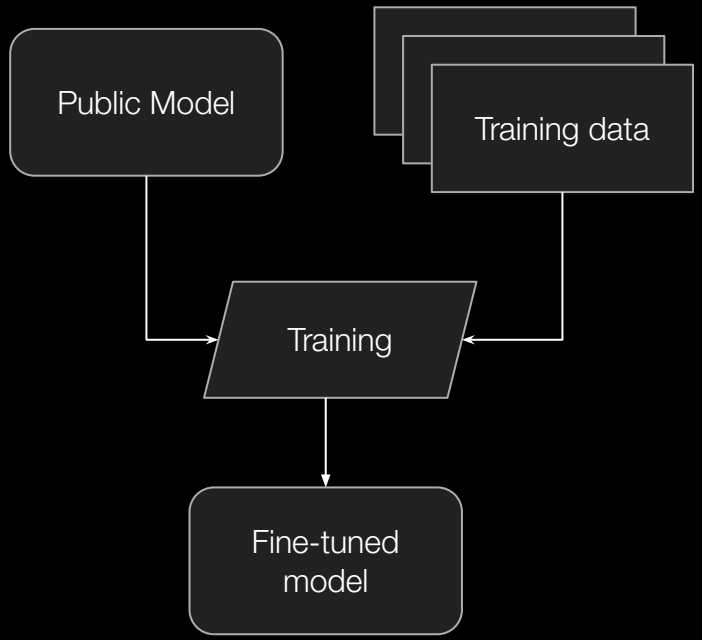
Example use cases

- Generate output in a consistent format
- Process input by following specific instructions

What we'll cover

- When to fine-tune
- Preparing the dataset
- Best practices
- Hyperparameters
- Fine-tuning advances
- Resources

What is Fine-tuning



Fine-tuning a model consists of training the model to follow a set of given input/output examples.

This will teach the model to behave in a certain way when confronted with a similar input in the future.

We recommend using 50-100 examples even if the minimum is 10.

When to fine-tune

Good for ✅

- Following a given format or tone for the output
- Processing the input following specific, complex instructions
- Improving latency
- Reducing token usage

Not good for ❌

- Teaching the model new knowledge
  - ➔ Use RAG or custom models instead
- Performing well at multiple, unrelated tasks
  - ➔ Do prompt-engineering or create multiple FT models instead
- Include up-to-date content in responses
  - ➔ Use RAG instead

Preparing the dataset

Example format

.jsonl

```
{
  "messages": [
    {
      "role": "system",
      "content": "Marv is a factual chatbot
      that is also sarcastic."
    },
    {
      "role": "user",
      "content": "What's the capital of
      France?"
    },
    {
      "role": "assistant",
      "content": "Paris, as if everyone
      doesn't know that already."
    }
  ]
}
```

- ➔ Take the set of instructions and prompts that you found worked best for the model prior to fine-tuning. Include them in every training example
- ➔ If you would like to shorten the instructions or prompts, it may take more training examples to arrive at good results

We recommend using 50-100 examples even if the minimum is 10.

Best practices

Curate examples carefully

Datasets can be difficult to build, start small and invest intentionally. Optimize for fewer high-quality training examples.

- Consider “prompt baking”, or using a basic prompt to generate your initial examples
- If your conversations are multi-turn, ensure your examples are representative
- Collect examples to target issues detected in evaluation
- Consider the balance & diversity of data
- Make sure your examples contain all the information needed in the response

Iterate on hyperparameters

Start with the defaults and adjust based on performance.

- If the model does not appear to converge, increase the learning rate multiplier
- If the model does not follow the training data as much as expected increase the number of epochs
- If the model becomes less diverse than expected decrease the # of epochs by 1-2

Automate your feedback pipeline

Introduce automated evaluations to highlight potential problem cases to clean up and use as training data.

Consider the G-Eval approach of using GPT-4 to perform automated testing using a scorecard.

Establish a baseline

Often users start with a zero-shot or few-shot prompt to build a baseline evaluation before graduating to fine-tuning.

Optimize for latency and token efficiency

When using GPT-4, once you have a baseline evaluation and training examples consider fine-tuning 3.5 to get similar performance for less cost and latency.

Experiment with reducing or removing system instructions with subsequent fine-tuned model versions.

Hyperparameters

Epochs

default: auto (standard is 4)

Refers to 1 full cycle through the training dataset

If you have hundreds of thousands of examples, we would recommend experimenting with two epochs (or one) to avoid overfitting.

Batch size

default: ~0.2% x N* (max 256)

*N = number of training examples

Number of training examples used to train a single forward & backward pass

In general, we've found that larger batch sizes tend to work better for larger datasets

Learning rate multiplier

default: 0.05, 0.1 or 0.2*

*depends on final batch size

Scaling factor for the original learning rate

We recommend experimenting with values between 0.02-0.2. We've found that larger learning rates often perform better with larger batch sizes.