Lecture 21, 22: In-Context Learning, Fine-Tuning, and Parameter Efficiency

Contents

1 Foundations: Pre-training and Inference

To understand fine-tuning and prompting strategies, we must first recall the fundamental architecture and training objective of Generative Pre-trained Transformers (GPT).

1.1 Pre-training Mechanism

The standard paradigm for training these models is **Next Token Prediction** (also known as Causal Language Modeling). Given a sequence of tokens $x = (x_1, x_2, \dots, x_T)$, the model maximizes the likelihood of the data by decomposing the joint probability:

$$P(x) = \prod_{t=1}^{T} P(x_t | x_{<t}; \theta) \tag{1}$$

where θ represents the model parameters. This is implemented via "Teacher Forcing," where the model receives the ground truth context $x_{<t}$ at every step during training. The loss function is the Cross-Entropy loss over the vocabulary.

1.2 Inference and Sampling

At inference time, the model operates in an autoregressive manner.

1. **Input processing:** The prompt is processed (forward pass) to generate the Key-Value (KV) cache.
2. **Generation:** The model outputs logits $z \in \mathbb{R}^{|V|}$ (where $|V|$ is vocabulary size) for the next token.
3. **Softmax with Temperature:** The logits are converted to probabilities using the softmax function. A hyperparameter T (temperature) controls the stochasticity:

$$P(x_i) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \tag{2}$$

   - As $T \to 0$, the distribution approaches an argmax (greedy decoding).
   - As $T \to \infty$, the distribution approaches a uniform distribution.
4. **Sampling:** A token is sampled from $P(x)$, appended to the context, and the process repeats until a `<stop>` token is generated.

2 In-Context Learning (ICL)

2.1 Historical Evolution

The capability of LLMs evolved through scaling:

- **GPT-1 (2018):** Established that decoder-only pre-training yields strong embeddings.
- **GPT-2 (2019):** Demonstrated that scaling up leads to natural prose generation and **Zero-shot learning** (eliciting facts via prompts like "The capital of France is...").
- **GPT-3 (2020):** Introduced **In-Context Learning (Few-Shot Learning)**. By providing examples within the context window, the model infers the task without weight updates.

2.2 Mechanism of ICL

In-Context Learning allows solving tasks by conditioning the model on examples $(x^{(i)}, y^{(i)})$ concatenated in the prompt:

$$\text{Prompt} = [x^{(1)}, y^{(1)}, x^{(2)}, y^{(2)}, \ldots, x^{(test)}] \tag{3}$$

The model predicts $y^{(test)}$ purely via forward pass dynamics (attention mechanisms) without gradient descent. This shifted the paradigm from "train a model for a task" to "design a prompt for a task."

3 Supervised Fine-Tuning (SFT)

While ICL is powerful, it is limited by context length and inference costs. To permanently instill behaviors (like instruction following), we utilize Supervised Fine-Tuning (SFT).

3.1 Instruction Following (The Alignment Problem)

Pre-trained base models are optimized for text completion, not instruction following. A prompt "Write a poem about a cat" might be autocompleted with "...and a dog" rather than the poem itself. SFT bridges this gap.

3.2 The SFT Training Setup

Given a dataset of instruction-response pairs (I, R), we perform a full fine-tune of the model parameters.

- **Input:** The concatenation of the Instruction and the Response.
- **Masking the Loss:** Crucially, the loss is calculated **only on the response tokens**, not the instruction tokens. We do not want the model to learn to predict the instruction; we want it to learn to predict the response conditional on the instruction.

Let the sequence be $x_{1:N}$ where $x_{1:k}$ is the instruction and $x_{k+1:N}$ is the response. The objective function becomes:

$$\mathcal{L}_{SFT}(\theta) = -\sum_{t=k+1}^{N} \log P(x_t | x_{1:t-1}; \theta) \tag{4}$$

Gradients flow back through the entire network (including the instruction processing path via attention), updating weights to better condition the response generation on the instruction.

3.3 The Alpaca Insight

Research (e.g., the Alpaca model, Dolly) demonstrated that the **quantity and style** of instruction-following data are critical. Even fitting a pre-trained model (like LLaMA) on a relatively small set (50k examples) of high-quality instruction-response pairs is sufficient to unlock instruction-following capabilities. It turns out the model possesses the knowledge; it primarily needs to learn the *format* of interacting as an assistant.

4 Prompt Optimization and Soft Prompting

When access to model weights is available (white-box access), we can use gradients to optimize the prompt itself rather than relying on manual "prompt engineering."

4.1 Soft Prompting

In standard prompting, we search for discrete tokens $x_{1:k}$ to prepend to our input. In **Soft Prompting**, we relax the constraint that the prompt must correspond to discrete vocabulary items.

We introduce a set of trainable vectors (a "soft prompt") $P \in \mathbb{R}^{L \times D}$, where L is the prompt length and D is the embedding dimension. These are treated as free parameters.

$$\text{Input Embeddings} = [P; E(x_{input})] \tag{5}$$

where $E(x_{input})$ are the fixed embeddings of the actual input text.

Training:

- Freeze the entire pre-trained model θ.
- Optimize only P via backpropagation: $\nabla_P \mathcal{L}$.

Parameter Efficiency: For a prompt length of $L = 100$ and dimension $D = 4096$:

$$N_{params} = 100 \times 4096 \approx 4 \times 10^5 \quad \text{(400k parameters)} \tag{6}$$

This is orders of magnitude smaller than an 8B parameter model, yet often achieves performance competitive with full fine-tuning for specific tasks.

4.2 Soft Prefixes (Prefix Tuning)

Soft prompting only modifies the input to the first layer. **Prefix Tuning** extends this by prepending trainable parameters to the Key (K) and Value (V) matrices at *every* layer of the transformer.

Let the model have M layers. For each layer, we introduce trainable prefixes $P_K, P_V \in \mathbb{R}^{L \times D}$.

- The gradients flow directly to these parameters at every layer, providing more expressivity than simple input soft prompts.
- Parameter count: $L \times D \times M \times 2$ (approx 26M parameters for typical sizes), still $\ll$ full model size.
- Interpretation: This effectively modulates the "activation memory" or state of the transformer at every step of computation.

5 Parameter-Efficient Fine-Tuning (PEFT): LoRA

While soft prompts are efficient, they can be difficult to optimize and harder to deploy (requiring modifying the input sequence). Low-Rank Adaptation (LoRA) provides a more general method for fine-tuning specific dense layers (typically Attention W_q, W_v or MLP weights) without updating the full model.

5.1 The Low-Rank Hypothesis

The core hypothesis of LoRA is that while weight matrices in pre-trained models are full rank, the change in weights ΔW required for adaptation to a specific downstream task has a low "intrinsic rank" r.

5.2 LoRA Formulation

Let $W_0 \in \mathbb{R}^{d \times k}$ be a frozen pre-trained weight matrix. We constrain the update ΔW by representing it as the product of two low-rank matrices B and A:

$$W = W_0 + \Delta W = W_0 + BA \tag{7}$$

where:

- $B \in \mathbb{R}^{d \times r}$
- $A \in \mathbb{R}^{r \times k}$
- Rank $r \ll \min(d, k)$ (e.g., $r = 4, 8, 16$).

During the forward pass, for an input x:

$$h = Wx = W_0x + B(Ax) \tag{8}$$

The term W_0x is computed using the frozen model, and BAx is computed in a parallel branch.

5.3 Initialization Strategy

Correct initialization is crucial for training stability.

1. **Matrix A:** Initialized with random Gaussian values (e.g., Kaiming/Xavier).
2. **Matrix B:** Initialized to **zeros**.

Why?

$$\Delta W_{init} = B_{init}A_{init} = 0 \times A_{init} = 0 \tag{9}$$

This ensures that at the start of training, the model behaves exactly like the pre-trained model.

- If both were 0: Gradients would be stuck at 0 (symmetry breaking issue).
- If both were Gaussian: ΔW would introduce random noise, destroying the pre-trained capabilities immediately ("shaking" the model).

5.4 Optimization and Scaling

The update is typically scaled by a factor α/r:

$$h = W_0x + \frac{\alpha}{r}BAx \tag{10}$$

This scaling allows the learning rate to be somewhat invariant to changes in r. Due to the vast difference in fan-in between A (fan-in k) and B (fan-in r), optimal learning rates for A and B may differ, though standard practice often uses a single tuned learning rate (usually higher than pre-training rates).

5.5 Inference with LoRA

A major advantage of LoRA is that it introduces **no inference latency**. Once training is complete, the matrices can be merged back into the original weights:

$$W_{new} = W_0 + (B \times A) \tag{11}$$

The model architecture remains unchanged, simply with updated numerical values in the weight matrices.