Advanced Human Language Technologies
Exercises on LLMs

Large Language Models

Exercise 1.

LLMs can be trained in two ways:

- *Pretraining* on a large corpus (self-supervised learning).
- *Fine-tuning* on a specific task (e.g., sentiment analysis, medical QA).

1. What types of knowledge does a model learn in *pretraining* ?
2. Why is fine-tuning necessary for domain-specific tasks?
3. What happens if an LLM is not fine-tuned and used for zero-shot tasks?

SOLUTION

1. During pretraining, the model learns:
    - **Linguistic Knowledge**: Syntax, grammar, and common sentence structures.
    - **Semantic Knowledge**: Word meanings, relationships, and contextual usage.
    - **World Knowledge**: Facts about the world, cultural references, and common-sense reasoning.
    - **Statistical Patterns**: Probabilistic associations between words and phrases.
2. Fine-tuning is necessary because:
    - General pretraining does not provide specialized knowledge (e.g., medical, legal, or technical domains).
    - Task-specific objectives (e.g., classification, summarization) require adaptation.
    - Pretrained models may contain biases that fine-tuning can mitigate.
    - Domain adaptation improves performance on specialized tasks by adjusting the model to the target data distribution.
3. If an LLM is used without fine-tuning:
    - It may generate reasonable responses but with lower accuracy on specialized topics.
    - It might lack precision in domain-specific terminology.
    - It could produce incorrect or misleading answers due to missing domain adaptation.
    - It may require prompt engineering to achieve better performance.

Exercise 2.

LLMs are sensitive to *how* we phrase a prompt. Consider the following queries given to an LLM:

- *What is the capital of France?*
- *Tell me a fun fact about Paris.*
- *Describe the geography of France.*

1. How do these prompts influence the model's response?
2. Why does a well-structured prompt improve accuracy?
3. How might using few-shot prompting (i.e. providing examples) change the behavior of the model?

SOLUTION

1. The phrasing of a prompt significantly affects the model's response:
    - **Direct factual questions** (e.g., *What is the capital of France?*) elicit short, precise answers.
    - **Open-ended prompts** (e.g., *Tell me a fun fact about Paris.*) lead to more varied and potentially unexpected responses.
    - **Broad descriptive prompts** (e.g., *Describe the geography of France.*) encourage the model to generate a longer, structured response with multiple details.
2. A well-structured prompt improves accuracy because:
    - It reduces ambiguity, helping the model understand the expected response format.
    - It provides context, guiding the model to retrieve the most relevant information.
    - It minimizes potential misinterpretations, leading to more precise and relevant answers.
    - It aligns better with the model's pretraining data, increasing the likelihood of correct responses.
3. Few-shot prompting influences the model by:
    - Demonstrating the desired response format, improving consistency.
    - Helping the model recognize patterns in the examples, leading to more relevant answers.
    - Reducing randomness in responses, especially for open-ended questions.
    - Enhancing performance on specialized or uncommon tasks by providing context-specific examples.

Exercise 3.

Many LLMs have a limited context window (e.g. 2048 tokens).

1. Why does this limit long conversations or documents?
2. If an LLM forgets early parts of a conversation, how can techniques like *retrieval-augmented generation (RAG)* help?
3. Why are newer models increasing their context size?

SOLUTION

1. The limited context window affects long conversations and documents because:
    - The model can only process a fixed number of tokens at a time, truncating older parts of the conversation.
    - In long interactions, important earlier details may be forgotten, leading to inconsistencies in responses.
    - Summarizing or analyzing long documents becomes difficult, as only a portion can be considered at once.
2. Retrieval-augmented generation (RAG) helps by:
    - Storing and retrieving relevant context from an external database or memory.
    - Dynamically incorporating past information into the current query, maintaining continuity.
    - Allowing the model to access relevant knowledge beyond its immediate context window.
3. Newer models are increasing their context size because:
    - A larger context window enables more coherent and context-aware responses in long conversations.
    - It improves performance on tasks requiring deep document understanding, such as summarization.
    - It reduces the need for external memory mechanisms, simplifying implementation.
    - It enhances reasoning over extended sequences, benefiting applications like legal and research analysis.

Exercise 4.

LLMs are trained on vast amounts of internet data, which can introduce biases.

1. How might an LLM generate biased responses in a job application review system?
2. What are some ways to mitigate bias in LLMs?
3. Why is explainability important when using LLMs for decision-making?

SOLUTION

1. An LLM might generate biased responses in a job application review system due to:
    - **Training Data Bias**: If historical hiring data reflects societal biases, the model may favor certain demographics.
    - **Implicit Stereotypes**: The model may associate job roles with specific genders, ethnicities, or backgrounds.
    - **Unequal Language Representation**: If resumes from different groups use different phrasing, the model might favor certain linguistic patterns.
2. Bias in LLMs can be mitigated by:
    - **Curating Training Data**: Using diverse and balanced datasets.
    - **Bias Auditing**: Analyzing model outputs for discriminatory patterns.
    - **Fairness Constraints**: Applying techniques like debiasing algorithms or adversarial training.
    - **Human Oversight**: Incorporating human review in critical decision-making processes.
3. Explainability is crucial because:

- It helps users understand why the model made a certain decision.
- It increases trust and accountability in AI-driven decisions.
- It enables bias detection and correction.
- It ensures compliance with ethical and legal standards.

Exercise 5.

Some LLMs are open-source (e.g., LLaMA, Mistral), while others are closed-source (e.g., GPT-4, Claude).

1. What are the advantages of open models?
2. Why do companies keep some models closed?
3. How does fine-tuning an open model differ from using an API to fine-tune a closed model?

SOLUTION

1. Open models offer several benefits:
   - **Transparency**: Researchers and developers can inspect and improve the model.
   - **Customization**: Users can fine-tune the model for specific applications.
   - **Cost Efficiency**: Avoids recurring API fees from proprietary providers.
   - **Community Contributions**: Open-source communities can improve and extend the model.
2. Companies keep models closed for:
   - **Competitive Advantage**: Preventing competitors from accessing proprietary technology.
   - **Monetization**: Charging for API access and enterprise services.
   - **Security and Safety**: Reducing risks of misuse or unintended consequences.
   - **Ethical and Legal Concerns**: Controlling the use of the model to comply with regulations.
3. The differences include:
   - **Full Control vs. Limited Access**: Open models allow direct access to weights, while closed models only offer API-based fine-tuning.
   - **Computational Resources**: Fine-tuning an open model requires substantial hardware, whereas API-based fine-tuning is managed by the provider.
   - **Customization**: Open models enable deeper modifications, while closed models often have constraints on fine-tuning depth.

Exercise 6.

A company is developing an AI-powered **customer support chatbot** and needs to decide between:

- **Fine-tuning an open-source LLM** (LLaMA-3.1-8B) and hosting it on cloud infrastructure.
- **Using a proprietary LLM API** (GPT-4o-mini) that allows fine-tuning for a fee per token.

The following costs apply:

- **Proprietary API Costs (GPT-4o-mini)**:
  - Fine-tuning: $\mathbf{3 \cdot 10^{-6}}$ \$/token
  - Inference: $\mathbf{0.5 \cdot 10^{-6}}$ \$/token
- **Open-Source LLM Costs (e.g., LLaMA-3.1-8B on AWS)**:

- Cloud machine for fine-tuning (3xRTX3090 GPUs): **400** $ (one-time cost, 3 days usage)
- Cloud machine for inference (1xRTX3090 GPU): **500** $/month.
  LLaMA-3.1 running in this machine can process **200** tokens per second.

The company expects:

- To fine-tune on **10** million tokens of training data, using **5** epochs.
- Each user to generate an average of $\mathbf{50,000}$ tokens per month during inference.
- The company to have $\mathbf{1,000}$ users.

1. Compute the total cost of fine-tuning for both options.
2. Compute the monthly inference cost for both options.
3. If our company grows and we have more users, will the open-source model become cheaper at some point ?
4. What other factors (besides cost) should the company consider when choosing between open-source and proprietary models?

SOLUTION

1. **Fine-tuning cost**
   - Proprietary API: $(10^7 \text{ tokens}) \times (3 \cdot 10^{-6} \text{ \$/token}) \times 5 \text{ epochs} = 150.00 \text{ \$}$
   - Open-source model: 400 \$ (one-time cloud server cost)
2. **Monthly Inference Cost**
   - Proprietary API: $(50,000 \text{ tokens}) \times (0.5 \cdot 10^{-6} \text{ \$/token}) \times (1,000 \text{ users}) = 25.00 \text{ \$/month}$
   - Open-source model:
     Our cloud machine can handle 200 tokens/sec $\rightarrow 200{*}60{*}60{*}24{*}30 = 518,400,000$ tokens/month
     So, with each server we can attend requests from:

     $$\left\lfloor \frac{518,400,000 \text{ tokens/month}}{50,000 \text{ tokens/user-month}} \right\rfloor = 10,368 \text{ users}$$

     To serve 1,000 users, we will need just 1 server, so the total monthly inference cost is 500 \$/month
3. To find out whether the open-source option becomes cheaper when having more users, we need to compute the monthly cost per user.
   - Proprietary API: $(50,000 \text{ tokens/user-month}) \times (0.5 \cdot 10^{-6} \text{ \$/token})) = 0.025 \text{ \$/user-month}$
   - Open-source model: If we use the server at maximum capacity, we have
     $\frac{500 \text{ \$/month}}{10,368 \text{ users}} \approx 0.05$ \$/user-month
     If we have more users, we'll need more servers, but the cost per user will be the same. If the servers are not at maximum capacity (i.e. less than 10,386 users per server) the cost per user will be higher.

   So, the proprietary model is cheaper, independently of the number of users.
4. Other considerations beyond cost
   - **Customization**: Fine-tuning an open-source model allows for more control over behavior.

- **Maintenance**: Self-hosting an open-source model requires technical expertise and maintenance time.
- **Performance**: Proprietary models may have higher performance for complex tasks.
- **Privacy**: Open-source models provide better control over sensitive data.
- **Independence**: Proprietary models may be modified or deprecated by the owners, or prices changed, with no regard for our needs.

Exercise 7.

Large Language Models (LLMs) can be used in different ways depending on the task:

- **Zero-shot learning**: Using the model as-is, without providing examples.
- **Few-shot learning**: Providing a few examples in the prompt to guide the model's response.
- **Fine-tuning**: Training the model on task-specific data to improve performance.

For each of the following tasks and scenarios, explain which approach is most suitable and why.

1. A company wants to deploy an AI-powered customer support chatbot. The chatbot must follow company policies and provide accurate responses to frequently asked questions. The company has access to a large dataset of past customer interactions.
2. A startup is developing an AI assistant that helps developers write Python code. The model should understand coding patterns, generate functions, and complete code snippets. The company does not have its own dataset but can provide a few examples in the prompt.
3. A business needs a model to analyze customer sentiment in *legal documents*, where words have different meanings than in general language. They have a labeled dataset with sentiment annotations.
4. A mobile app needs a lightweight model for *on-device* translation between rare language pairs. There is no labeled data for training, and responses must be fast.
5. A media company wants an AI system to generate *concise summaries* of news articles. The articles vary in topics and writing style. The company has a moderate budget and needs a scalable solution.

SOLUTION

1. Best approach: Fine-tuning
   - The chatbot must follow strict company policies and provide accurate responses.
   - Since a large dataset of past interactions is available, fine-tuning will help tailor the model to company-specific knowledge.
   - Few-shot prompting might work for simple queries, but fine-tuning ensures consistency and accuracy.
2. Best approach: Few-shot learning
   - The model already has general programming knowledge from pretraining.
   - Since the company does not have its own dataset, fine-tuning is not an option.
   - Few-shot prompting can guide the model with relevant examples of coding patterns.
3. Best approach: Fine-tuning
   - Legal terminology differs significantly from general language, requiring domain adaptation.
   - The company has labeled sentiment data, making fine-tuning feasible.
   - Zero-shot and few-shot learning may struggle with domain-specific sentiment interpretation.

4. Best approach: Zero-shot learning
    - No labeled data is available, making fine-tuning impractical.
    - The model needs to be fast and lightweight for on-device inference, which limits the complexity of prompts.
    - Zero-shot translation using a multilingual LLM is the most viable option.
5. Best approach: Few-shot learning
    - Articles vary in topics and style, making it hard to fine-tune on a single dataset.
    - Few-shot prompting with examples of well-structured summaries can improve output quality.
    - Fine-tuning could work, but it would require significant labeled data and may not generalize well.

Exercise 8.

Design the LLM interactions and prompts required to build an assistant that can recommend recent news articles to a user, based on their interest profile (already known to the system) and a user query.

The system has access to a news search API.

E.g. The system has a user profile where his known interest are: `['international politics', 'economy', 'football', 'cinema']`. The user asks the system: "What are the latest news about war in Gaza?". The system searches for news, ranks them according to user interests, and provides most relevant ones as a response.

SOLUTION

User $\longrightarrow$ **System**: "What are the latest news about war in gaza?"

System $\longrightarrow$ **LLM**: "Identify the intent in the user sentence 'What are the latest news about war in Gaza'?"

System $\longleftarrow$ **LLM***: {`'intent':'search_news'; 'topic':'war in Gaza'`}

System $\longrightarrow$ **NewsAPI**: `call_API_search('war in Gaza')`

System $\longleftarrow$ **NewsAPI**: $\langle$list of recent news$\rangle$

Retrieve user interests INT from the user profile in the database

For each news item N in $\langle$list of recent news$\rangle$:

- **System** $\longrightarrow$ **LLM**$^+$: "Provide a score from 0 to 5 for how much the following news item matches the user interests $\langle INT \rangle$. News item: $\langle N \rangle$"
- **System** $\longleftarrow$ **LLM**: $\langle$score$\rangle$
- Store $\langle$score$\rangle$ as value for item N.

Sort $\langle$list of recent news$\rangle$ by highest score and select top-k

User $\longleftarrow$ **System**: $\langle$top-k news items$\rangle$

*: This LLM should be fine-tuned to recognize valid intents for our assistant (either fine-tuned specifically for our application, or using an available model with function calling for usual APIs)

$^+$: This LLM may be a generic model, not necessarily fine tuned

Exercise 9.

Design a system using LLMs to analyze open-ended user feedback and categorize it, and produce some statistical results

The system accesses a DB with all collected user feedback.

Each user opinion may include several sentences, stating different aspects.

Each opinion must be broken down in separate sentences, each with a category in [observation, suggestion, question]. Observation sentences must be classified as positive/neutral/negative.

The system has access to the API of a Bussiness Intelligence suite, which can produce statistical analysis of data, chats, graphs, etc.

SOLUTION

Retrieve feedback entries from the database: ⟨LIST_OF_FEEDBACKS⟩

For each feedback entry F in ⟨LIST_OF_FEEDBACKS⟩:

System $\longrightarrow$ **LLM***: "Break the following user feedback into individual sentences. For each sentence, assign a category from [observation, suggestion, question]. If category is 'observation', also label it as [positive, neutral, negative]. Feedback: $\langle F \rangle$"

System $\longleftarrow$ **LLM**$^+$: `[{"sentence":"The app is very fast.","category":"observation", "sentiment":"positive"}, {"sentence":"I wish it had dark mode.","category":"suggestion"}, {"sentence":"Is there a desktop version?","category":"question"} ]`

For each sentence returned:

Store `(sentence, category, sentiment)` in analysis_table

System $\longrightarrow$ **API_BI** `call_make_stats(analysis_table)`

System $\longleftarrow$ **API_BI** ⟨(stats, charts, graphs)⟩

User $\longleftarrow$ **System**: ⟨(stats, charts, graphs)⟩

*: This LLM should be fine-tuned to produce the right json output, or at least, the prompt should include some few-shot examples of the expected output

$^+$: The shown output is just an example. Obviously, it would be different for each user feedback.