GENERATIVE AI WITH LLMs

WEEK 2

What's included?

Fine-Tuning LLMs with instruction

1. Instruction fine-tuning
2. Fine-tuning on a single task
3. Multi-task instruction fine-tuning
4. Model Evaluation
5. Benchmarks

Parameter efficient Fine-Tuning

1. Intro to PEFT
2. PEFT Techniques : LoRA
3. PEFT Techniques : Soft Prompts
4. Fine-tuning a gen AI model - lab

FINE-TUNING LLMs WITH INSTRUCTION

INSTRUCTION FINE-TUNING

IN-CONTEXT LEARNING - PROMPT ENGINEERING

1. One - shot
2. Zero - shot
3. Few - shot

Possible limitations
- May not work with smaller models
- May take up extra space

Possible Solution : FINE-TUNING → *This is different from pre-training
↳ uses vast amounts of unstructured data via supervised learning

1. Supervised learning where a dataset of labelled examples to update the weights of LLMs.
2. Labelled examples are prompt completion pairs

PROMPT [....], COMPLETION [...]
PROMPT [....], COMPLETION [....]

(Task-specific examples)

3. Finetuning process extends the training of the model to improve its ability to generate good completion for a specific task.

how?

4. Instruction finetuning process trains the model using examples that demonstrate how it should respond to a specific instruction.

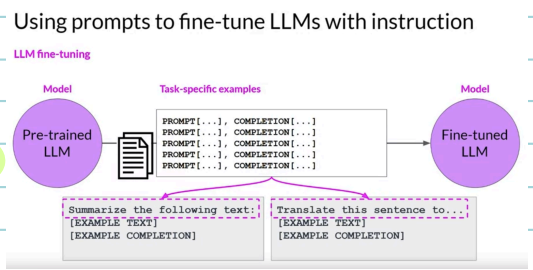
1. Classify this review: "I loved this DVD"
   Sentiment : "Positive"
2. Classify this review: "I don't like this chair"
   Sentiment : Negative.

(Each prompt/completion pair includes a specific "instruction" to the LLM)

↓

This can be extended to the type of task that is to be performed. Foreg. Summarization, translation etc

5. FULL FINETUNING : where all the model weights are updated → leads to a new model version
   ↓
   Process requires more memory & compute budget to store & process all gradients, optimizers & other components.
   ↓
   can be optimized using methods



How is instruction finetuning performed?

1. Prepare training data. For eg. Amazon Review Datasets
2. Prompt Template libraries q turned them into instruction prompt datasets for fine-tuning
3. These prompt templates can be used to prepare data for various tasks for eg classification, text generation q summarization

Sample prompt instruction templates

Classification / sentiment analysis

```
jinja: "Given the following review:\n{{review_body}}\npredict the associated rating\
  \ from the following choices (1 being lowest and 5 being highest)\n- {{ answer_choices\
  \ | join('\\n- ') }} \n|||\n{{answer_choices[star_rating-1]}}"
```

Text generation

```
jinja: Generate a {{star_rating}}-star review (1 being lowest and 5 being highest)
  about this product {{product_title}}.        |||        {{review_body}}
```

Text summarization

```
jinja: "Give a short sentence describing the following product review:\n{{review_body}}\
  \ \n|||\n{{review_headline}}"
```

{Review Body}:
- The original review
- This gets passed to the template
- Then gets inserted into the text that starts with an instruction like "Predict the associated rating, generate star review."
- The result: Contains a prompt that contains both an instruction q the example from the dataset

4. Once the instruction dataset is ready, it's split into training and testing split
5. During finetuning, prompts are selected from the training datasets and passed to the LLM, which then generates completions.
6. This generated completion is compared with the response specified in the training data
7. Output of LLM is probability distribution across tokens.
8. This distribution of the completion and that of the training label are compared, and standard cross-entropy function is used to calculate loss between the 2 token distribution in standard backpropogation.
9. This is repeated for many batches of prompt completion pairs over several epochs and weights are updated so that the model performance improves.
10. Evaluation: Done using the holdout validation set
    - Gives the validation accuracy
    Final performance evaluation - using the holdout test dataset - gives test accuracy

→ GIVES A NEW VERSION OF BASE MODEL: INSTRUCT MODEL

FINE-TUNING ON A SINGLE TASK

- Implemented when an application performs only a single task.
  eg. Summarization only
  500 – 1000 examples.

- Potential downside: CATASTROPHIC FORGETTING
  ↓
  full finetuning process modifies the weights of the LLM
  ↓
  leads to great performance on that single task
  but, degrades performance on other tasks.
  ↓
  If need to avoid this, train/finetune the model on
  50 – 100,000 examples of other tasks
  but, requires more data and compute
  ↓
  2nd option to avoid: PARAMETER EFFICIENT FINE-TUNING

MULTI-TASK INSTRUCTION FINE-TUNING

- Models are trained on specific tasks. for eg. Summarization, review rating, code translation, and entity recognition.
- Training dataset is comprised of example inputs & outputs for multiple tasks like above.
- This avoids catastrophic forgetting.
- Epochs, losses, used to update the weights of the models – instruction finetuned model.
  – (Can require a lot of data) ↓
- 50 – 100,000 examples required.
- FLAN → eg. of instruction

  FLANT5 – General purpose (samsum → dataset of msg like data.
  FLAN-PALM –

  dialogsum → Dataset to Summarise dialogues

MODEL EVALUATION

Traditional ML:

$$\text{Accuracy} = \frac{\text{Correct Prediction}}{\text{Total Prediction}}$$

LLMs (Non Deterministic & Language Based)

Scores relevant: **ROUGE** → Used for text Summarization
→ Compares summary to one or more reference summaries

BLEU SCORE → Used for text translation
→ Compares to human-generated translations

Examples:

unigram (1 word)

Reference: It is cold outside

n-gram (n words)

Generated: It is very cold outside

bigram (2 words)

Comparing Unigrams (1 word at a time)

$$\text{ROUGE-1 (Recall)} = \frac{\text{unigram matches}}{\text{unigram in reference}} = \frac{4}{4} = 1.0$$

$$\text{ROUGE-1 (Precision)} = \frac{\text{unigram matches}}{\text{unigrams in output}} = \frac{4}{5} = 0.8$$

$$\text{ROUGE-1 (F1)} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = 2 \times \frac{0.8}{1.8} = 0.89$$

It is / is very / very cold (like that)

$$\text{ROUGE-2 Recall} = \frac{\text{bigram matches}}{\text{bigrams in reference}} = \frac{2}{3} = 0.67$$

$$\text{ROUGE-2 Precision} = \frac{\text{bigram matches}}{\text{bigrams in output}} = \frac{2}{4} = 0.5$$

$$\text{ROUGE-2 F1} = 2 \times \frac{P \times R}{P + R} = 2 \times \frac{0.335}{1.17} = 0.57$$

Long common subsequences:

It is cold outside

It is very cold outside

= 2

$$\therefore \text{ROUGE-L (Recall)} = \frac{\text{LCS (Gen, Ref)}}{\text{unigrams in ref}} = \frac{2}{4} = 0.5$$

$$\text{ROUGE-L (Precision)} = \frac{\text{LCS (Gen, Ref)}}{\text{unigrams in output}} = \frac{2}{5} = 0.4$$

$$\text{ROUGE-L F1} = 2\frac{PXR}{P+R} = 2 \times \frac{0.2}{0.9} = 0.44$$

ROUGE scores are only relevant or useful when comparing same task. For eg. Summarization. Summarization & text translation can't be compared.

BLEU SCORE:

BLEU metric = Avg (precision across range of n-gram sizes)

BENCHMARKS

1) GLUE - General Language Understanding Evaluation
2) SuperGLUE - Updated GLUE
3) MMLU - Massive Multi-Task Language Understanding
4) BigBench - lite and BigBench
5) HELM - Holistic Evaluation of Language Models

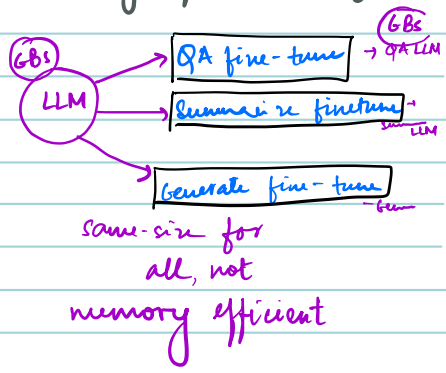
PARAMETER EFFICIENT FINE-TUNING

PEFT

Drawbacks of Full Finetuning:
↳ where every model weight is updated during supervised learning

1. Computationally intensive
   - Memory for trainable weights
   - " " optimizer states
   - " " Gradients
   - " " Forward activations
   - " " Temp Memory

} 12 × 20× weights



same size for all, not memory efficient

Solution: Parameter Efficient Finetuning

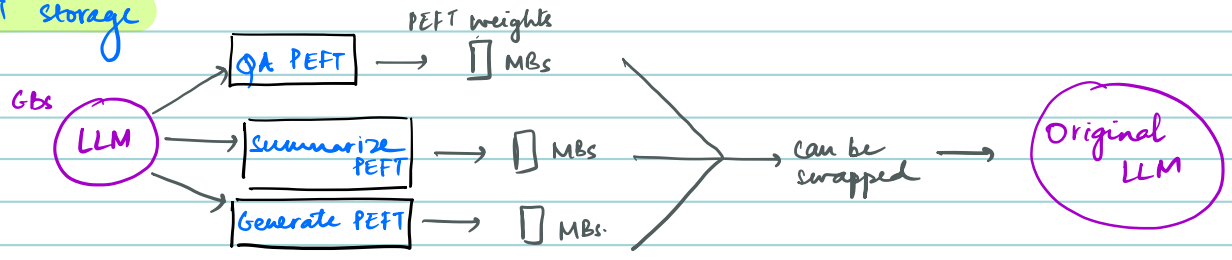
↳ Only updates a small subset of parameters

↳ Freeze most of the weights & focuses on only a subset of existing model parameters
eg. particular layers of components.

↳ Other related techniques don't touch the original weights at all, instead add a small no. of new parameters
∴ No. of trained parameters is much smaller than no. of params in LLM. In some case 15-20% parameters of original weights thereby, optimizing memory requirements.

↳ less prone to catastrophic forgetting problem since original model weights are frozen

PEFT Storage



PEFT Methods

↳ Selective: Select subset of initial LLM parameters to finetune. Performance mixed

↳ Reparameterization: Work with original parameters, but reduce the no. of parameters to train by creating new-low rank transformations of the original NN weights
LoRA

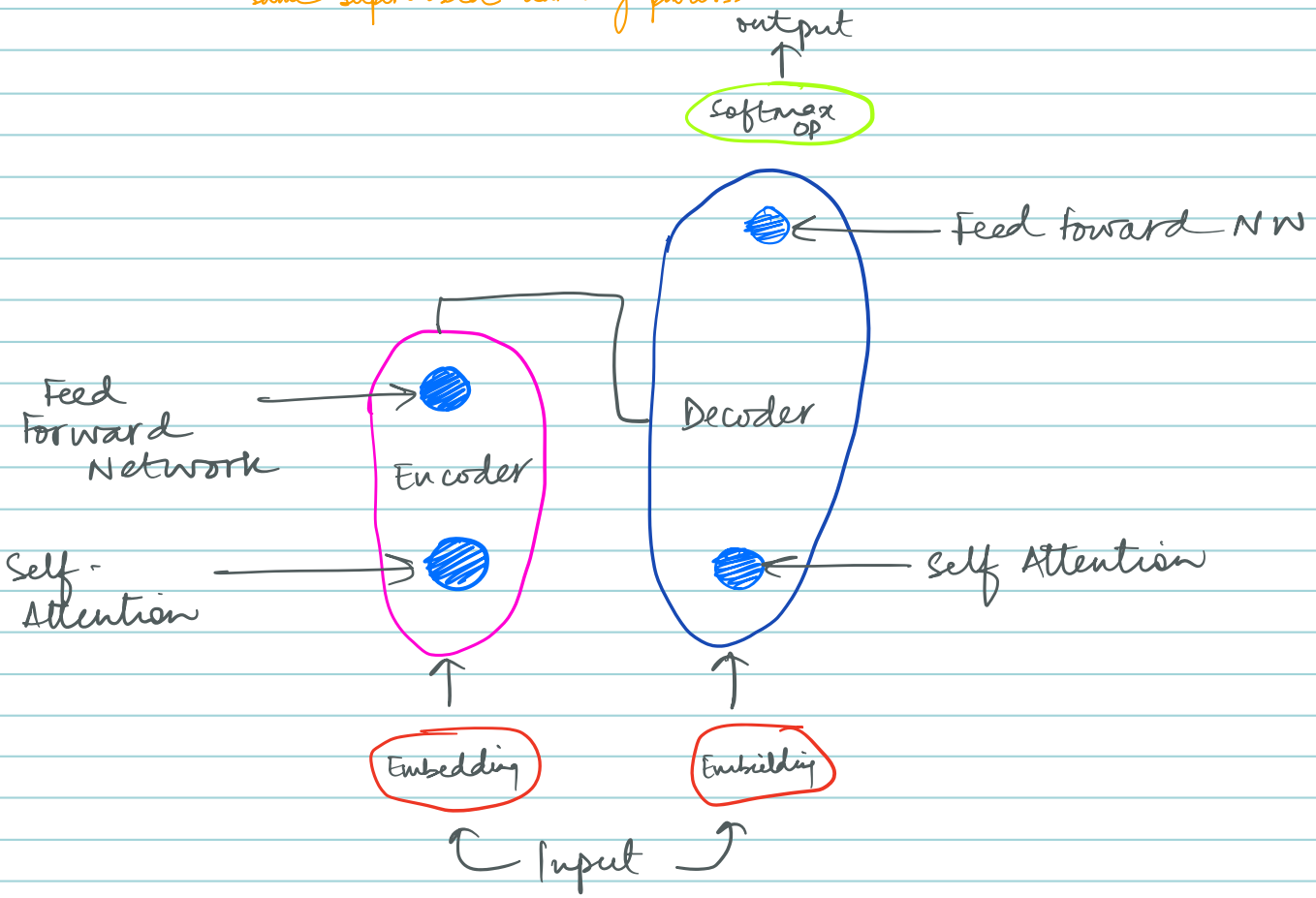
↳ Additive: All original weights frozen and introducing new trainable components
→ Adapter methods: New trainable layers, to the orig. arch, eg. encoder/decoder
→ Softprompts: Arch. fixed & frozen, manipulate input for performance. Add more trainable params to the prompt embeddings, retraining the embedding weights

PEFT Technique 1 : LoRA

Low Rank Adaptation of LLMs

- During full fine-tuning, all encoders' weights are updated
- But, LoRA freezes all the original model parameters & then injecting a pair of rank decomposition matrix alongside original weights.
- Smaller matrices : Set so that their product is a matrix with same dimension

  => Original weights are kept frozen and smaller matrices are trained using same supervised learning process.



1) Two kinds of neural networks are present in the transformer networks –
   - Feed Forward
   - Self-Attention

2) Weights of these 2 NNs are learned during pre-training.

3) Full-fine-tuning : Updates are the parameters in this layer.

4) LoRA

- Update the weights without having to update all the parameters.

, How? — By freezing all the original model parameters

— Injects a pair of rank decomposition matrices alongside the original weights.

— Dimensions of these small injected matrices:

1) B * A = (B X A) (smaller) → (keeps dimensions same as original)

2) Frozen + weight + (BXA) ← (+)

5) Problems solved by LoRA:

→ Preserves the no. of parameters
→ No impact of inference latency

6) Explained through examples:

- Transformer weight dimensions:

d x k = (512 X 64)

⇒ 512 x 64 ⇒ 32,768 Trainable parameters.

- LoRA finetuned:

rank r = 8

⇒ matrix A → dimensions: (r x k) = 8 X 64 = 512 parameters

matrix B → " " = 512 X 8 = 4,096 trainable params.

⇒ (86%) reduction in parameters to train.

Total: 4,096 + 512 = 4,608 parameters

(Single GPU) ← lot lesser than original

→ This method can be customized to different tasks.

7) Comparing LoRA finetuned vs Full fine-tuned. - Using ROUGE matrix.

FLAN T5 ← [Evaluating the performance of this model 4 specific task]

Dialog Summarization

LoRA fine-tuned
R1: 0.4081
R2: 0.1633
RL: 0.3251
RLsum: 0.3249

↑ - 3.20% but still significantly efficient!

Full fine-tuned model
R1: 0.4216
R2: 0.1804
RL: 0.3384
RLsum: 0.3384

↑ +80.63%

Baseline score
flan-T5 base
rouge1: 0.2334
rouge2: 0.0760
rougeL: 0.2014
rougeLsum: 0.2015

8) The lower the LoRA rank, lower is the no. of trainable parameters, bigger the savings on compute.

9) Optimizing the rank - still active area of research
- Explore latest developments

PEFT Techniques 2 : Softprompts

1) A.k.a Prompt Tuning (not prompt engineering)

2) How are these different?

PROMPT ENGG:
1) Manual
2) More effort to write
3) Extensive testing & combinations required.
4) limited by length of context window

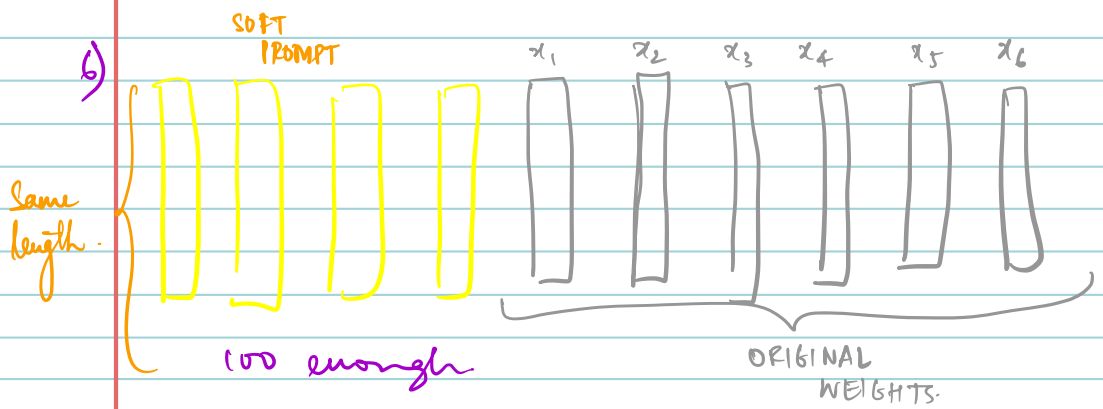
PROMPT TUNING
1) Add additional trainable tokens to the prompt
2) Have upto supervised learning process to determine optimal values.

3) Trainable tokens are called **soft prompts**

4) These gets prepended to the embedding vectors that represent the input text

5) length(soft prompt vectors) = length(embedding vectors of the tkns tokens)

6)



7) Virtual tokens that can take up any value in the multi-dimensional embedding space that maximize performance.

8) weights/embedding vectors get updated over time. for soft prompts

9) Performance

Superglue scores low for smaller LLMs but large for Big LLMs.

END OF WEEK 2.