Topics to study!

Language models

- n-gram [unigram, bigram, etc]
- perplexity

"simple" neural LMs

- fixed-window neural LM
  - concat $\boxed{\text{word embs}}$ → type/token
  - linear layers → $h = f(Wx)$
    - ↳ weight matrix
    - $f = \text{ReLU}$
  - softmax fn
    - used in output layer to get $p(w_i \mid w_{1 \ldots i-1})$
    - used in self-attn

Transformer LMs

- self attention
  - query / key / value
  - multi-head self-attn
  - KV caching
  - masking during training

- position embeddings
  - learned vs. fixed
  - absolute vs. relative
  - ALiBi
    - add linear bias to attn matrix
    - no additive embedding
  - RoPE
    - rotate Q, K
    - no additive embs
- Transformer configurations:
  - decoder vs. encoder vs. encoder-decoder

GPT, LLaMa, Mistral, etc.

- decoders predict the next word
  - masked self-attn
  - "prefix LM" where prompt is unmasked

BERT

- encoders compute representations of the entire input text
  - unmasked self-attn
  - cannot generate text

T5

- encoder/decoder models separate the input/prompt from generated output
  - cross-attn uses queries from decoder and K/V from encoder
  - residual connections needed in decoder to include both enc/dec value vectors

Training neural LMs:

- gradient descent
- backpropagation
  - chain rule + caching derivatives
- cross-entropy loss
- batching
- tokenization of inputs/outputs
  - BPE for subwords
- PyTorch implementation of Transformer and training loop (HW 2)

Transfer learning (pretraining ⇒ fine-tuning)

- fine-tuning (SFT)
  - needs labeled dataset for a downstream task (eg. sentiment)
    - much smaller than pretraining data
  - Same loss as pretraining: cross-entropy on target outputs
- Parameter-efficient adaptation
  - LoRA / prompt tuning
  - reduce number of params that are modified vs. SFT

Decoding from LMs:

- how to generate text at test-time
- greedy vs. beam search
- nucleus sampling vs. ancestral sampling
  - effect of "p" in nucleus sampling

Aligning LMs:

3 stages of RLHF:

1. instruction tuning
   - SFT on instruction-following data
     - FLAN
2. reward model training on human pref judgments
   - Bradley-Terry pref. model
3. objective:

$$\max_{\pi} \mathbb{E}_{x,y}\Big[\underbrace{r(x,y)}_{\text{reward of output } y \text{ given input } x} - \beta \underbrace{D_{KL}\big(\pi(y|x)\,\|\,\pi_{ref}(y|x)\big)}_{\text{KL penalty to prevent deviations from } \pi_{ref}}\Big]$$

- requires rollouts from policy model π
  - "rollouts" are generations

given an instruction x,
we sample a y using a
decoding algo

- optimize objective using PPO

- "Best-of-n" sampling
  - instead of stage 3, just sample multiple y's from the LM and rerank them w/ reward model
- DPO (direct pref optimization)
  - no explicit reward model
  - no rollouts

high-level derivation:

1. express reward model in terms of optimal policy π^*

$$r(x,y) = \beta \log \frac{\pi^*(y|x)}{\pi_{ref}(y|x)} + \beta \log Z$$

normalizer, intractable to compute

2. plug into Bradley-Terry model

$$p(y_w > y_l | x) = \frac{\exp(r(x, y_w))}{\exp(r(x,y_w)) + \exp(r(x,y_l))}$$

3. convert to loss fn

$$L_{DPO}(\pi, \pi_{ref}) = -\underset{x, y_w, y_l}{E} \log \sigma \left(\beta \log \frac{\pi(y_w|x)}{\pi_{ref}(y_w|x)} - \beta \log \frac{\pi(y_l|x)}{\pi_{ref}(y_l|x)} \right)$$

- allows us to simply fine-tune over human prefs with modified loss fn

- LLM-based feedback
  - constitutional AI

Prompting

- zero-shot vs. few-shot
  - few-shot uses "demonstrations" of x, y pairs
  - chain of thought
  - retrieval

Scaling

- Chinchilla
  - importance of data size vs. model size vs. total compute (FLOPS)
- small models don't exhibit properties such as few-shot learning / instruction following

Evaluation

- perplexity
- BLEU/ROUGE (word matching)
- BLEURT/COMET
  - finetune encoder on human judgments
- LLM-based eval
  - GPTEval, FactScore
- human eval (e.g. HW 1)