Introduction to Artificial Intelligence

Lecture 9: Language models II

October 2, 2025

Lecture plan

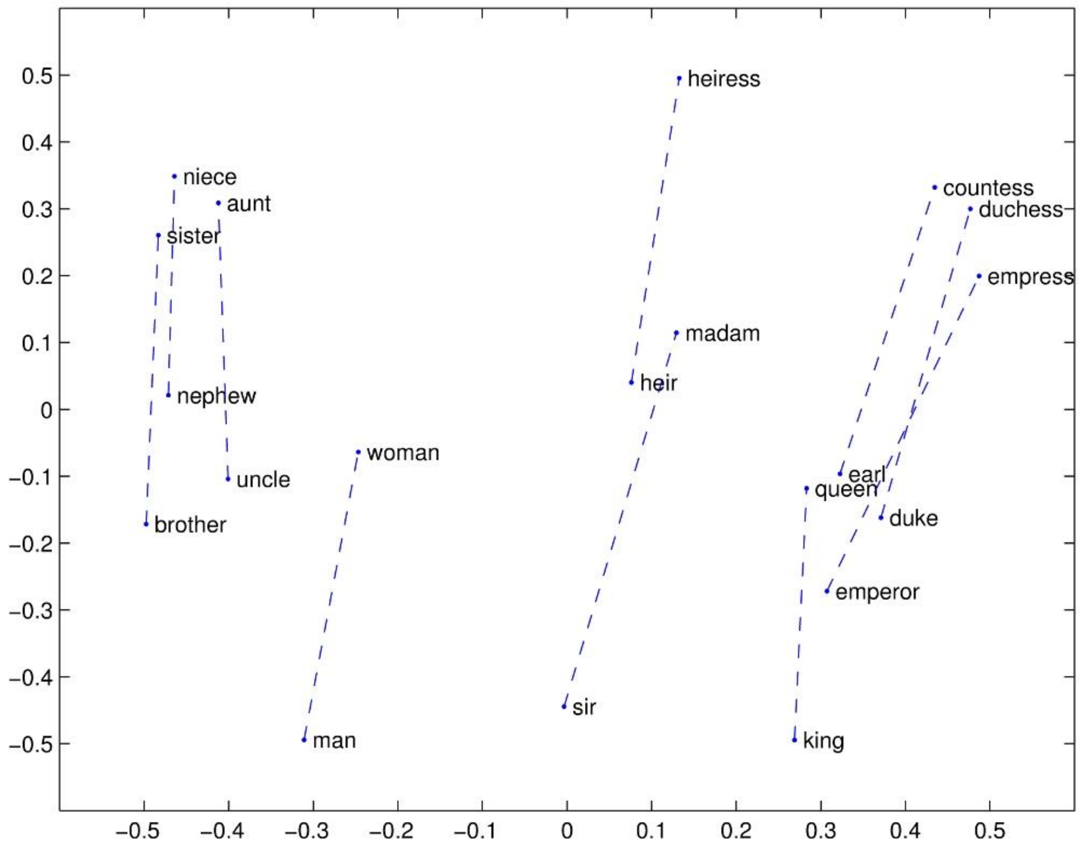
- Language models
  - What is a language model (last lecture)
  - Capabilities of a language model (last lecture)
  - **More specifics of modeling**
  - Training and adaptation

Tokenization

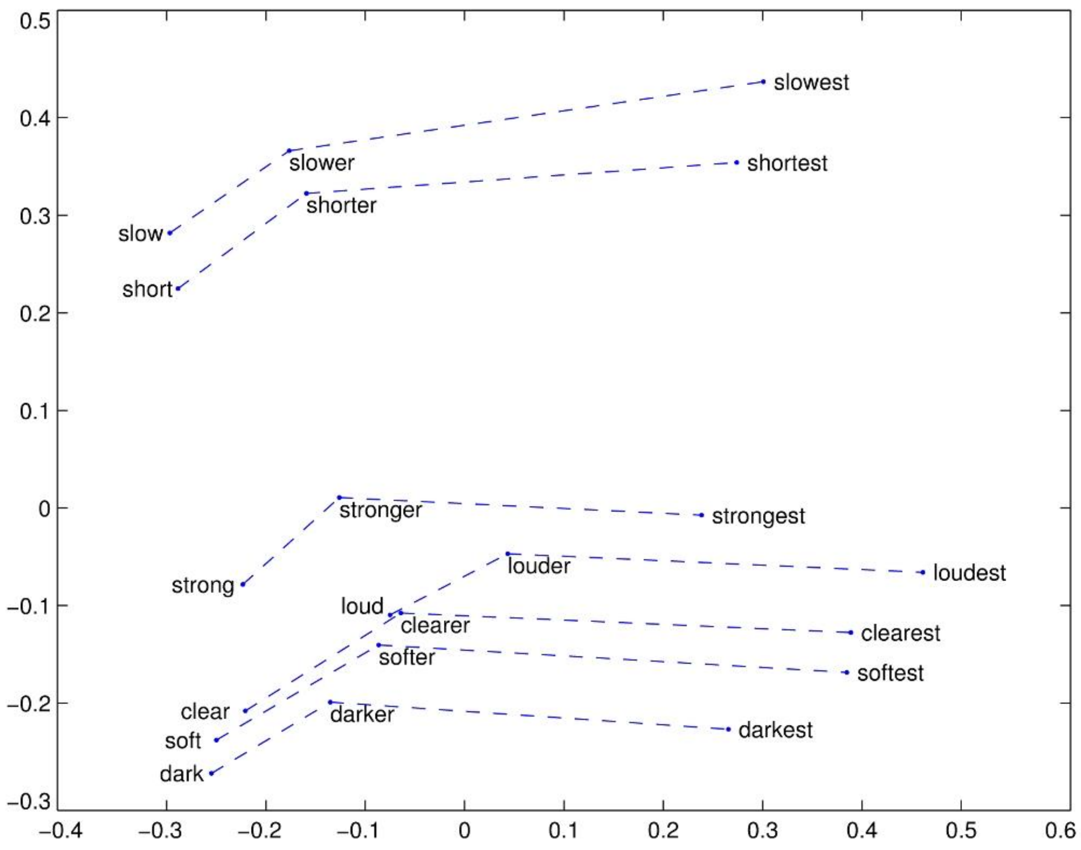
- Recall that a language model p is a probability distribution over a **sequence of tokens** where each token comes from some vocabulary, e.g., {the, mouse, ate, the, cheese}
- A **tokenizer** converts any string into a sequence of tokens
  - the mouse ate the cheese ⇒ [the, mouse, ate, the, cheese]
  - Replace each word with a word vector
- **Word embeddings**: vectors for word representation

Word embeddings can indicate linear substructures

- Male vs. female relationship



- Comparative - superlative



What makes a good tokenizer?

- Split by spaces: text.split(‘ ’)
  - However, this doesn’t work for Chinese
  - There are hyphenated words (e.g., fine-tuning) and contractions (e.g., don’t)
- **Learning the tokenizer**:
  - **Intuition**: start with each character as its own token and combine tokens that co-occur a lot
  - **Input**: a training corpus (sequence of characters)
  - **Procedure**
    - Find the pair of elements x, x' that co-occur the most number of times
    - Replace all occurrences of x, x' with a new symbol xx'
    - Repeat

Tokenizer example

- Example
  - [t, h, e, _, c, a, r], [t, h, e, _, c, a, t], [t, h, e, _, r, a, t]
  - [th, e, _, c, a, r], [th, e, _, c, a, t], [th, e, _, r, a, t] (th occurs 3x)
  - [the, _, c, a, r], [the, _, c, a, t], [the, _, r, a, t] (the occurs 3x)
  - [the, _, ca, r], [the, _, ca, t], [the, _, r, a, t] (ca occurs 2x)

Unigram model

- **Unigram model**: rather than just splitting by "frequency," a more "principled" approach is to define an objective function that captures what a good tokenization look like
  - Given a sequence $x_{1:L}$, a tokenization T is a set of
$$p(x_{1:L}) = \prod_{(i,j)\in T} p(x_{i:j})$$
  - **Example**
    - **Input**: ababc
    - **Tokenization**: $T = \{(1,2), (3,4), (5,5)\}$
    - **Likelihood**: $p(x_{1:L}) = \frac{2}{3} \cdot \frac{2}{3} \cdot \frac{1}{3}$
  - **Algorithm**: optimize $p(x_{1:L})$ and T with alternative update (expectation-maximization)

Encoder models

- These language models produce contextual embeddings but cannot be used directly to generate text

$$x_{1:L} \Rightarrow \varphi(x_{1:L})$$

- The contextual embeddings can be used for classification tasks
  - Example: **sentiment classification**

    [[CLS], the, movie, was, great] ⇒ positive
  - Example: **natural language inference**

    [[CLS], all, animals, breathe, [SEP], cats, breathe] ⇒ entailment
  - Contextual embeddings can depend **bidirectionally** on both left/right context, however, they cannot naturally **generate** completions

Decoder models

- These are the standard **autoregressive language models**, which given a prompt $x_{1:i}$ produces both contextual embeddings and a distribution over next tokens x_{i+1}

$$x_{1:i} \Rightarrow \varphi(x_{1:i}), p(x_{i+1}|x_i)$$

  - Example: **text autocomplete**

    [the, movie, was, [CLS]] ⇒ great

  - The embeddings can only depend on the left context, but it can naturally **generate** completions

Recurrent neural networks

- The basic form of an RNN simply computes a sequence of **hidden states** recursively
  - Process the sequence $x_1, \ldots, x_L$ left-to-right and recursively compute vectors $h_1, \ldots, h_L$
  - For $i = 1,2, \ldots, L$: compute $h_i = RNN(h_{i-1}, x_i)$
  - **RNN**: update hidden state h based on a new observation x
    - **SimpleRNN**: $\sigma(Uh + Vx + b)$
    - **LSTM** (long short-term memory) and **GRU** (gated recurrent unit)

Transformer networks

- **Attention mechanism**:
  - For a sequence $x_{1:L} = [x_1, \dots, x_L]$ and an arbitrary query y
  - A key matrix W_{key} and query matrix W_{query} to produce a score for every token

$$score_i = x_i^\top W_{key}^\top W_{query} y$$

  - Apply softmax to form a probability distribution over tokens

$$[\alpha_1, \dots, \alpha_L] = softmax(score_1, score_2, \dots, score_L)$$

  - Then the final output is a weighted combination over the values with a value matrix W_{value}

$$\sum_{i=1}^{L} \alpha_i (W_{value} x_i)$$

Self-attention and multi-head attention

- **Multi-head attention**:

$$[\text{Attention}(x_{1:L}, y), \text{Attention}(x_{1:L}, y), \ldots, \text{Attention}(x_{1:L}, y)]$$

- In a **self-attention layer**, we substitute x_i in for y as the query argument:

$$[\text{Attention}(x_{1:L}, x_1), \text{Attention}(x_{1:L}, x_2), \ldots, \text{Attention}(x_{1:L}, x_L)]$$

- **Final output**:

$$\sum_{i=1}^{h} W_O^i W_V^i X \cdot softmax\left(\frac{X^\top \left(W_K^i\right)^\top W_Q^i X}{\sqrt{d}}\right)$$

- Query W_Q, key W_K, and value W_V, output W_O

Improving training

- **Residual connection**: Instead of applying function f

$$f(x_{1:L})$$

Apply

$$x_{1:L} + f(x_{1:L})$$

- **Layer normalization**: Extract the mean and covariance of the input
- **Momentum:** An efficient, second-order optimization method
- **Adaptive gradient:** adjusts learning rate automatically during training
- **Positional embeddings:** For each position in the sequence, add an extra embedding vector for that position

Lecture plan

- Language models
  - What is a language model (last lecture)
  - Capabilities of a language model (last lecture)
  - More specifics of modeling
  - **Training and adaptation**

Training objectives (decoder models)

- Recall that an autoregressive language model defines a conditional distribution

$$p(x_i | x_{1:i-1})$$

- We first map the prefix sequence (prompt) to contextual embeddings $\varphi(x_{1:i-1})$
- Apply an embedding matrix E to obtain $E\varphi(x_{1:i-1})_{i-1}$
- Apply softmax to produce a distribution over x_i

$$p(x_i | x_{1:i-1}) = softmax(E\varphi(x_{1:i-1})_{i-1})$$

- Finally, apply maximum likelihood

$$\sum_{x_{1:L}} \sum_{i=1}^{L} -\log\left(p_\theta(x_i | x_{1:i-1})\right)$$

Encoder models

- Bidirectional transformer training objective:
  - **Masked language modeling**: Mask out a particular token, ask the model to predict the missing token
  - **Next sentence prediction**: BERT is trained on pairs of sentences concatenated. The goal of next sentence prediction is to predict whether the second sentence follows from the first or not

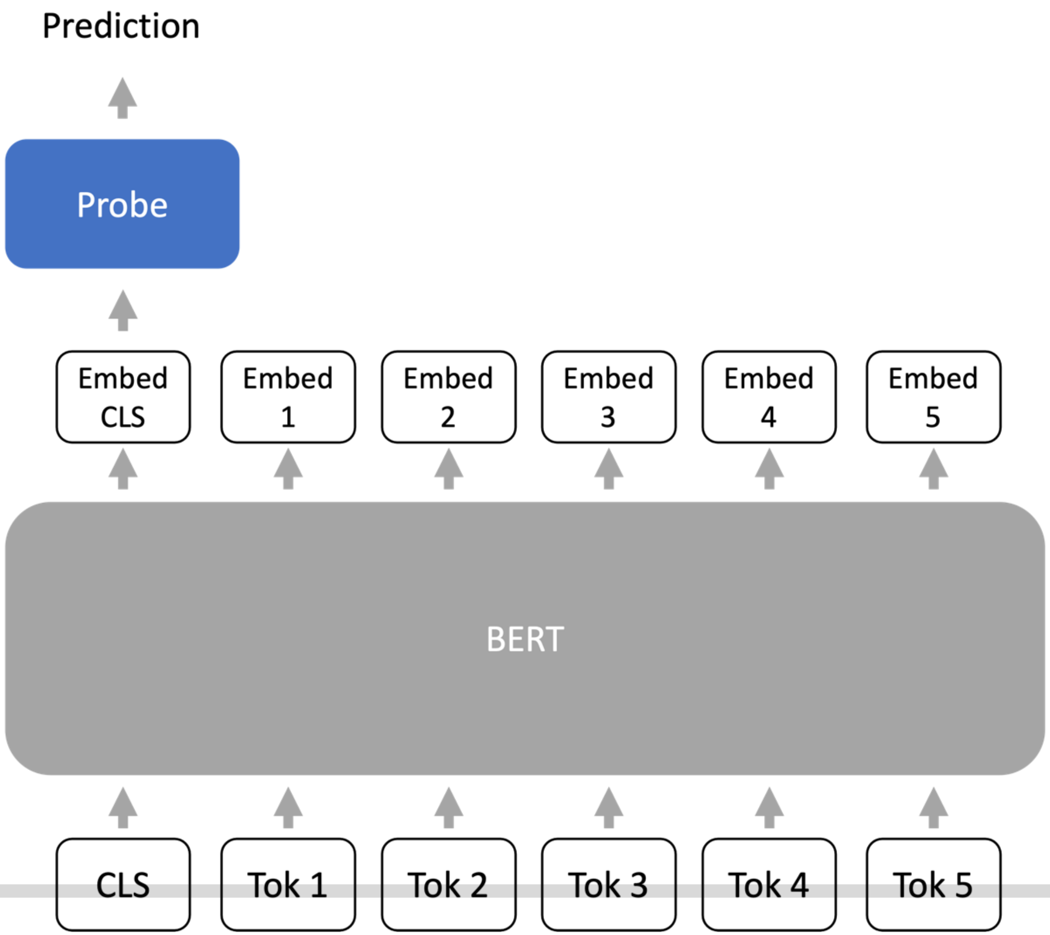
- Optimization algorithms: **Stochastic gradient**
  - Take a mini-batch of samples
  - Compute the gradient of the objective on the mini-batch, using backpropagation
  - Apply one gradient descent step with a learning rate parameter

Adapting a language model

- Language models are trained in a task-agnostic way
  - Downstream tasks can be very different from language modeling on the Pile
- Natural language inference (NLI)
  - **Premise**: I have never seen an apple that is not red
  - **Hypothesis**: I have never seen an apple
  - **Correct output**: Not entailment
- Ways downstream tasks can be different
  - **Topic shift**: the downstream task is focused on a new or specific topic
  - **Temporal shift**: the downstream task requires new knowledge that is unavailable during pre-training

Probing

- Train a probe (or prediction head) from the last layer representations of the language model to the output (e.g., class label)



Fine-tuning

- Using the entire language model parameters as the initialization or base model for optimization
- Usually, fine-tuning will produce a model that is fairly close to the base model
- Low-rank adapters and quantized adapters optimize the number of bits per performance

Transfer learning

- Transfer learning: use the information learned from one task to help learn another task
  - Example #1: building a face recognition system from open-source models plus a few hundred labeled examples
  - Example #2: fine-tuning a pre-trained language model for solving a downstream text prediction task
- Multitask learning: simultaneously train a multitask learning model on multiple objectives

LLM alignment

- Collect human-written demonstrations of desired behavior
- Perform supervised fine-tuning on the demonstrations
- On a set of instructions, sample outputs from the language model for each instruction, then gather human preferences for which sampled output is most preferred
- Fine-tune the model with a specialized objective to maximize preference reward