COMS 4771 Fall 2025
Language models

Language models

- (Large) Language Model: probabilistic model for discrete sequences
- Originally studied by Shannon (1948) in his theory of communication

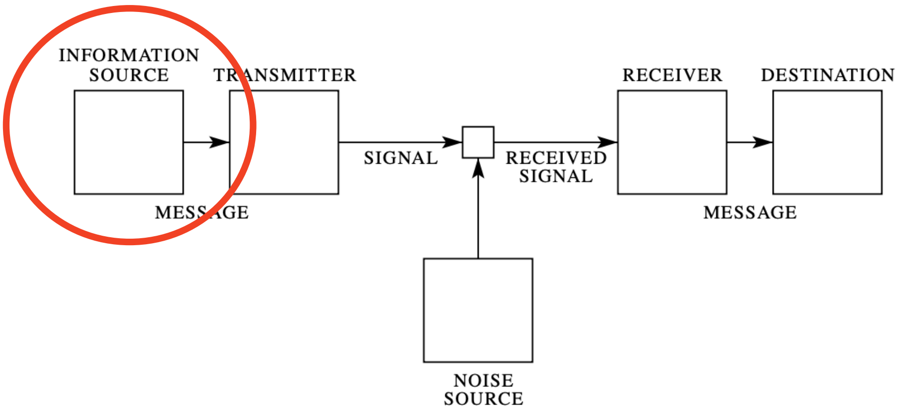


Fig. 1—Schematic diagram of a general communication system.

Probability of a sequence of tokens

- $X_{1:T} := (X_1, \dots, X_T)$: T random "tokens" with joint distribution P
  - Tokens could represent letters, words, "sub-words", etc.
  - Each X_t takes value in "alphabet" (a.k.a. "vocabulary") Σ
- Next token conditional distribution:

$$P(X_T = x_T | X_{1:T-1} = x_{1:T-1}) = \frac{P(X_{1:T} = x_{1:T})}{P(X_{1:T-1} = x_{1:T-1})}$$

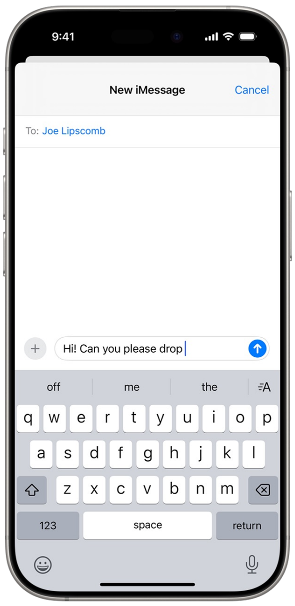
Application #1: Next-token prediction

Suppose you know joint distribution of $X_{1:T}$

- Q: What token is most likely to follow $x_{1:T-1} \in \Sigma^{T-1}$?
- A: Maximizer of next token (conditional) probability

$$\operatorname*{argmax}_{x \in \Sigma} P(X_T = x | X_{1:T-1} = x_{1:T-1})$$

- (Just like in multi-class prediction, with $|\Sigma|$ classes)



Application #2: Sequence generation

Sample random sequence according to joint distribution of $X_{1:T}$:

- First, draw $x_1 \sim P(X_1)$ [marginal distribution of X_1]
- Then, draw $x_2 \sim P(X_2 | X_1 = x_1)$ [conditional distribution of X_2 given $X_1 = x_1$]
- Then, draw $x_3 \sim P(X_3 | X_{1:2} = x_{1:2})$ […]
- Then, draw $x_4 \sim P(X_4 | X_{1:3} = x_{1:3})$ […]
- Etc.

Difficulties with language models

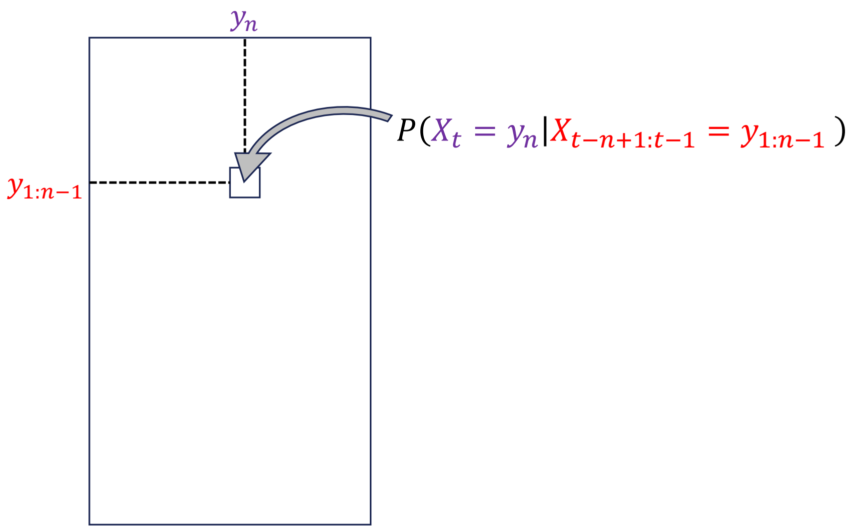
- $|\Sigma|^T$ many sequences of length T
- For large T, cannot write down all of their probabilities
- Need a more succinct parameterization

Shannon's *n*-gram models (*n*=2, *n*=3)

- Bigram model: distributions P satisfying, for all $t > 1$,
$$P(X_t = x_t | X_{1:t-1} = x_{1:t-1}) = P(X_t = x_t | X_{t-1} = x_{t-1})$$
- Parameters of bigram distribution (look-up tables):
  - $A_{x,y} \coloneqq P(X_t = y | X_{t-1} = x)$ for each $x, y \in \Sigma$
  - $\pi_x \coloneqq P(X_1 = x)$ for each $x \in \Sigma$
- Trigram model: distributions P satisfying, for all $t > 2$,
$$P(X_t = x_t | X_{1:t-1} = x_{1:t-1}) = P(X_t = x_t | X_{t-2:t-1} = x_{t-2:t-1})$$
- Parameters of trigram distribution (look-up tables):
  - $A_{x,y,z} \coloneqq P(X_t = z | X_{t-2} = x, X_{t-1} = y)$ for each $x, y, z \in \Sigma$
  - $\pi_{x,y} \coloneqq P(X_1 = x, X_2 = y)$ for each $x, y \in \Sigma$

Look-up tables

- Look-up table parameter A for *n*-gram model



$|\Sigma|^{n-1} \times |\Sigma|$ look-up table

Sequence generation with bigram model

Sample random sequence with bigram model for $X_{1:T}$:

- First, draw $x_1 \sim P(X_1)$
- Then, draw $x_2 \sim P(X_2 | X_1 = x_1)$
- Then, draw $x_3 \sim P(X_3 | X_2 = x_2)$
- Then, draw $x_4 \sim P(X_4 | X_3 = x_3)$
- Etc.

Application #2: Sequence generation

Sample random sequence according to joint distribution of $X_{1:T}$:

- First, draw $x_1 \sim P(X_1)$ [marginal distribution of X_1]
- Then, draw $x_2 \sim P(X_2 | X_1 = x_1)$ [conditional distribution of X_2 given $X_1 = x_1$]
- Then, draw $x_3 \sim P(X_3 | X_{1:2} = x_{1:2})$ […]
- Then, draw $x_4 \sim P(X_4 | X_{1:3} = x_{1:3})$ […]
- Etc.

Fitting *n*-gram models to data

- Many ways to do this, but simplest is to use empirical frequencies
- MLE for $P(X_t = y_n | X_{t-n+1:t-1} = y_{1:n-1})$:

$$\frac{\#\text{count}(y_{1:n-1}, y_n)}{\#\text{count}(y_{1:n-1})}$$

$\#\text{count}(z)$ is number of occurrences of string z in training data

- Variants: regularized counts (e.g., Laplace smoothing), …

Sequences generated by an *n*-gram model fit to data

Conditioning on initial tokens (a.k.a. prompt) $X_{1:28}$ =

`it is a truth universally ac`

n=1: [...] `mci w aeovmsne drsbwt elo oiwetrcao rne em ok hae lom`

n=2: [...] `o drto t bet it s f aree h at teshas rr l hasis popor`

n=3: [...] `es as pred cirse so tiought let of ant forrieng pled`

n=4: [...] `common of could ell his i foung laster are plage omin`

n=5: [...] `quaintance only can better he obliged it is the first`

Limitations of *n*-gram model

- Only uses last *n*−*1* tokens to predict next token
- Example (n = 5; Σ = English words):

as the proctor started the clock the students opened their ______

- $P(\text{books} \mid \text{the students opened their}) > P(\text{exams} \mid \text{the students opened their})$
- But with the entire context, "exams" is more likely

[Example from Chris Manning's CS224n Lecture 5]

Modern methods for fitting *n*-gram models to data

- Approaches based on look-up tables are typically limited to $n < 10$
- Today:
  - $n = 10^6$ or more
  - Conditional probabilities
    $$P(X_t = y_n | X_{t-n+1:t-1} = y_{1:n-1})$$
    computed by a neural net rather than using look-up table
  - Training: Fit parameters Θ of neural net by (approximately) minimizing
    $$\sum_{t=n}^{T} - \log P_\Theta(X_t = x_t | X_{t-n+1:t-1} = x_{t-n+1:t-1}) \qquad \text{(Logarithmic loss)}$$
    where $x_{1:T}$ is training data (e.g., all together as one long sequence)

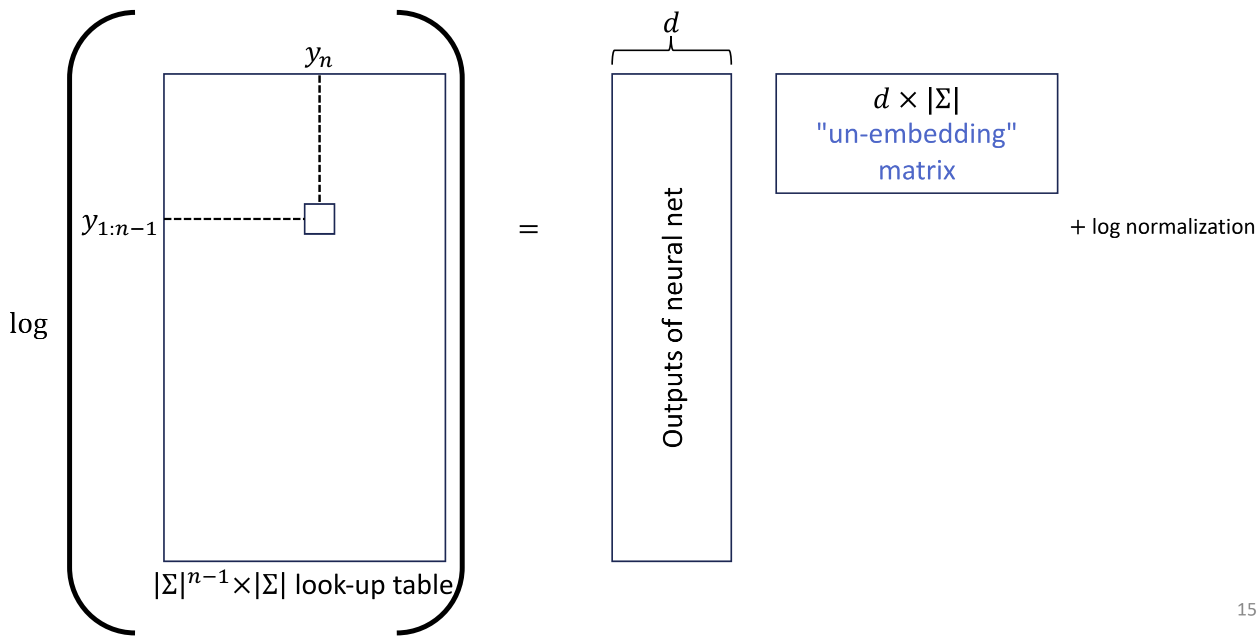
What kind of neural net?

Three issues:

1. Output of the neural net
2. Input to the neural net
3. Internal computation by the neural net

What kind of neural net? [Outputs]

- Typically use large alphabet/vocabulary Σ



What kind of neural net? [Inputs]

- Neural nets typically operate on real vectors in $\mathbb{R}^d$
- Token embedding matrix:

$d \times |\Sigma|$
"embedding"
matrix

  - Map sequence of tokens $x_{1:n-1} \in \Sigma^{n-1}$ to sequence of vectors (looked-up from embedding matrix)

- Embedding + un-embedding matrices related to word embeddings (à la Latent Semantic Analysis)
  - These will also be "trained" alongside neural net parameters

What kind of neural net? [Internal computation]

- Function computed by neural net
  - Input: N vectors from $\mathbb{R}^d$ (for some $N \leq n - 1$)
  - Output: vector from $\mathbb{R}^d$
- Many options:
  - Averaging
  - Convolutional net
  - Recurrent neural net
  - Long Short-Term Memory
  - Transformer
  - …

N input vectors from $\mathbb{R}^d$ $\mapsto$ output

- Challenge: effective + efficient processing of long sequences

Example: averaging

- Input: $\vec{x}_1, \ldots, \vec{x}_N \in \mathbb{R}^d$
- Output: uniform average

$$\frac{1}{N} \sum_{i=1}^{N} \vec{x}_i$$

- Linear transformation of a "Bag-of-Words" representation
  - Very efficient to compute; effective for some simple problems
  - But ineffective for other problems because critical information is lost

Example: attention (basic building block in transformers)

- Parameters: $W_Q, W_K, W_V \in \mathbb{R}^{d \times d}$
- Input: $\vec{x}_1, \dots, \vec{x}_N \in \mathbb{R}^d$
- Output: weighted average

"value" vectors (linear transforms of $\vec{x}_1, \dots, \vec{x}_N$)

"key" vectors (linear transforms of $\vec{x}_1, \dots, \vec{x}_N$)

"query" vector (linear transform of $\vec{x}_N$)

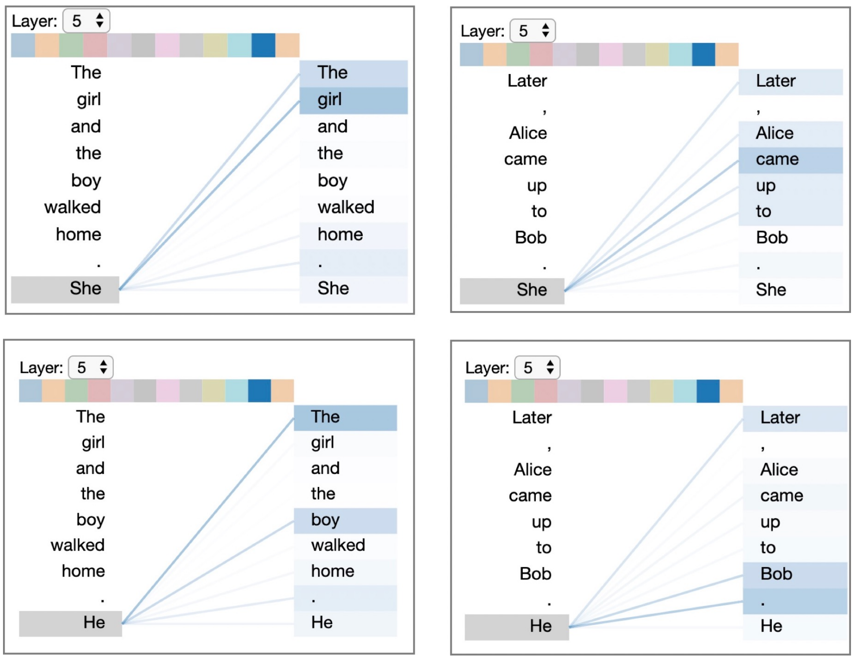
$$\sum_{i=1}^{N} \alpha_i (W_V^\top \vec{x}_i)$$

where

$$(\alpha_1, \dots, \alpha_N) = \mathrm{softmax}\Big(\langle W_Q^\top \vec{x}_N, W_K^\top \vec{x}_1 \rangle, \dots, \langle W_Q^\top \vec{x}_N, W_K^\top \vec{x}_N \rangle\Big)$$

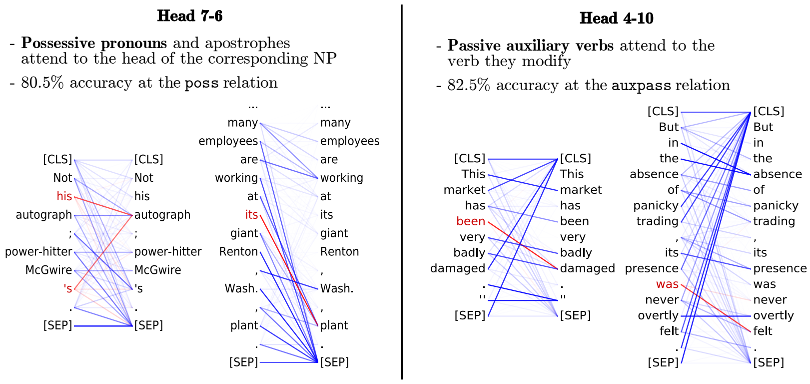
- Averaging weights determined by (softmax of) inner products between query vector and key vectors

Examples of attention patterns in GPT-2 [Vig, 2019]



Use of language models beyond next-token prediction

- Ability to compute accurate next-token predictions seems to involve interesting forms of "reasoning" (= algorithmic process)
- How do we know this? Neuroscience for LLMs [e.g., Clark et al, 2019]
  - Discovered some basic "algorithms" implemented by the LLMs (e.g., for rudimentary linguistic analysis and statistical inference)



Summary

- Modern language models: *n*-gram models with succinct neural network parameterizations
- Larger *n* → more "context" available to predict next-token
- Training: minimize sum of logarithmic losses on training data
- Why are large language models so powerful?
  - Accurate next-token predictions → "reasoning"-like computations

Course summary

- Statistical framework for ML
  - Risk / loss
  - IID assumption
  - Role of test data
  - Model selection / cross validation
  - Calibration
  - Reweighting training data
  - Equalizing error rates
- Algorithmic paradigms
  - Nearest neighbor
  - Maximum likelihood estimation
  - Greedy algorithms
  - Model averaging / bagging
  - Gradient descent
  - Autodiff
  - Boosting
- Some modeling techniques
  - Normal linear regression model
  - Logistic regression
  - Normal generative model
  - Distance functions
  - Trees
  - Linear models
  - Feature maps
  - Kernels
  - Neural nets
  - Regularization
  - Best fitting subspace
  - PCA