CS 784: Computational Linguistics
Lecture 11: Masked Language Models and Sequence Labeling

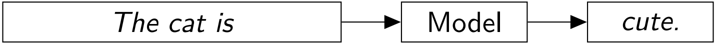
Freda Shi

School of Computer Science, University of Waterloo
fhs@uwaterloo.ca

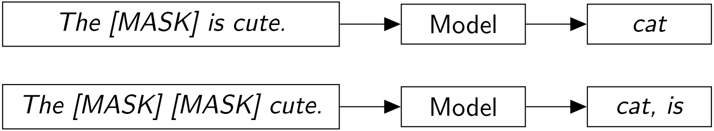
February 26, 2025

Recap: Language Models

Autoregressive language modeling (e.g., GPT, Radford et al., 2018):

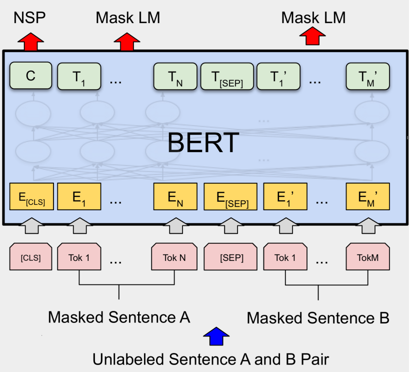


Masked language modeling (BERT, Devlin et al., 2019):



Masked Language Models

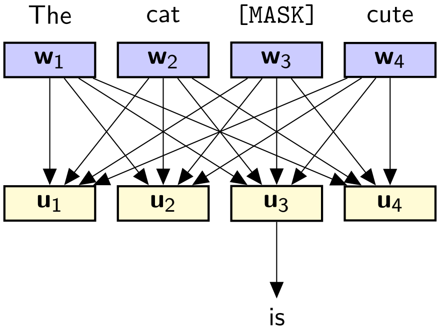
Bidirectional Encoder Representations from Transformers (BERT, Devlin et al., 2019):



- Two (random) sentences.
- Two objectives:
  - Masked LM.
  - Next sentence prediction (NSP).
- Two special tokens:
  - `[CLS]`: classification token.
  - `[SEP]`: separator token.

Pretraining Objectives

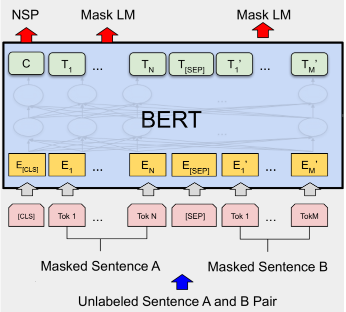
- **Masked LM**: given a sentence, mask some tokens and predict them.
  - A portion (15%) of tokens are replaced with `[MASK]`.
  - Predict masked tokens using the output of the model.



$\mathbf{w}_1, \ldots, \mathbf{w}_4 \rightarrow \mathbf{K}, \mathbf{Q}, \mathbf{V} \qquad \mathbf{U} = \mathbf{V}\mathtt{softmax}\left(\frac{\mathbf{K}^T\mathbf{Q}}{\sqrt{d}}\right) \qquad \mathbf{u}_3 \rightarrow \mathtt{is}$

Pretraining Objectives

- **Next sentence prediction (NSP)**: given two sentences, predict if they are consecutive or not.
  Underlying hypothesis: understanding of sentence relations makes better general-purpose sentence representation.
  Approach: **binary classification** with the `[CLS]` token representation.



RoBERTa (Liu et al., 2020): no NSP, larger batch size, more data, more training steps.
Works better than BERT.

The Special Tokens: [CLS] and [SEP]

- [SEP]: the separator token indicating sentence boundaries.
- [CLS]: the classification token.
  - The output of the [CLS] token is used for next-sentence prediction.

These tokens can be renamed with whatever you like.

There is no specific reason why [CLS] is at the beginning.

After Pretraining

- **Feature-based transfer learning**: instead of manually designed features, use a pre-trained model as feature extractor.
  Train another model with the extracted features.
  All layers of the pre-trained model are **frozen**.
- **Fine-tuning**: Keep the model architecture and weights, but continue training on a new task.
  The model weights can be updated during fine-tuning.

Practical convention: use the `[CLS]` token output as text representation for classification tasks.

I strongly encourage you to try out the BERT model in the Hugging Face Transformers library if you haven't done so! `https://huggingface.co/docs/transformers/en/model_doc/bert`

Sequence Labeling: The Task

Input:	The	cat	is	cute
Output:	DT	NN	VBZ	JJ

Sequence labeling: assign a label to each token in a sequence.

Taking part-of-speech (POS) tagging as an example:

$$\text{classify}(s) = \arg\max_{y} \mathit{score}(s, y; \boldsymbol{\Theta})$$

$$\text{POS-Tag}(s) = \arg\max_{\mathbf{y}} \text{score}(s, \mathbf{y}; \boldsymbol{\Theta})$$

Key difference from classification: the output is a sequence, not a single label.

Recap: Independence and Conditional Independence

Two random variables X and Y are independent if for all x and y,

$$P(X = x, Y = y) = P(X = x)P(Y = y).$$

Two random variables X and Y are conditionally independent given Z if for all x, y, and z,

$$P(X = x, Y = y \mid Z = z) = P(X = x \mid Z = z)P(Y = y \mid Z = z)$$

We write this as $X \perp Y \mid Z$.

Example: height and vocabulary size are (or at least should be) conditionally independent given age.

Markov Assumption and Markov Chain

Recap: the Markov assumption in n-gram language models implies an (n-1)-th order Markov assumption.

$$P(w_i \mid w_1, \ldots, w_{i-1}) = P(w_i \mid w_{i-n+1}, \ldots, w_{i-1})$$

First-order Markov assumption:

$$P(w_i \mid w_1, \ldots, w_{i-1}) = P(w_i \mid w_{i-1})$$

A **Markov chain** is a sequence of random variables $X_1, X_2, \ldots, X_n$ satisfies the Markov assumption: $X_t \perp X_{t-2}, \ldots, X_1 \mid X_{t-1}$.

Hidden Markov Models (HMMs) extend the Markov assumption to a set of **hidden states**.

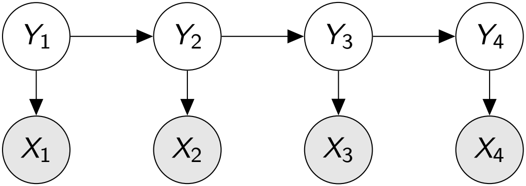
Note: the *hidden states* here is not the same as the *hidden layer/stats* in neural networks.

A good starting point of learning probabilistic graphical models.

Hidden Markov Models (HMMs)

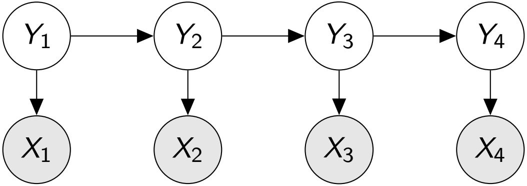
Modeling the joint probability of the **observed sequence of variables** $X_1, \ldots, X_n$ and the **hidden sequence of variables** $Y_1, \ldots, Y_n$:

$$P(X_1, \ldots, X_n, Y_1, \ldots, Y_n) = P(Y_1) \prod_{i=2}^{n} P(Y_i \mid Y_{i-1}) \prod_{i=1}^{n} P(X_i \mid Y_i)$$



An instantiation of **Bayesian networks**: representing conditional dependency with a directed acyclic graph (DAG).

Conditional Independence in HMMs



Intuitive interpretation: if the given variable Z is removed from the graph, two variables X and Y are conditionally independent if they are disconnected.

$$Y_t \perp Y_{t-2}, \ldots, Y_1, X_{t-1}, X_{t-2}, \ldots, X_1 \mid Y_{t-1}$$
$$X_t \perp Y_n, \ldots, Y_{t+1}, Y_{t-1}, \ldots, Y_1, X_n, \ldots, X_{t+1}, X_{t-1}, \ldots, X_1 \mid Y_t$$

More Background: Bayesian Networks

In a Bayesian network, the direction of arcs does not necessarily have specific meanings.

These two Bayesian networks represents the following, respectively.

$$P(X, Y) = P(X)P(Y \mid X) \qquad P(X, Y) = P(Y)P(X \mid Y)$$

However, it's always intuitive to construct Bayesian networks with **causal relationships** in mind.

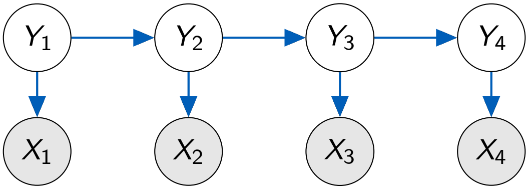
Problem Formulation

Suppose with are given a pretrained HMM with

- The **transition probabilities** $P(Y_i \mid Y_{i-1})$, shared across time steps
- The **emission probabilities** $P(X_i \mid Y_i)$, shared across time steps
- The **initial state distribution** $P(Y_1)$
- The observation sequence $X_1, \ldots, X_n$

What is the most likely sequence of hidden states $Y_1, \ldots, Y_n$?

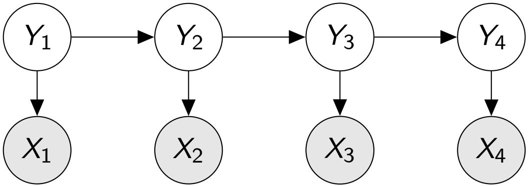
$$\arg\max_{Y_1,\ldots,Y_n} P(Y_1, \ldots, Y_n \mid X_1, \ldots, X_n)$$



Inference with HMMs

Goal : $\arg\max_{Y_1,\ldots,Y_n} P(Y_1,\ldots,Y_n \mid X_1,\ldots,X_n)$

$$\arg\max_{Y_1,\ldots,Y_n} P(Y_1,\ldots,Y_n, X_1,\ldots,X_n)$$



Bruce-force solution: enumerate all possible sequences of hidden states and compute the joint probability.

The Viterbi algorithm: compute it efficiently with dynamic programming.

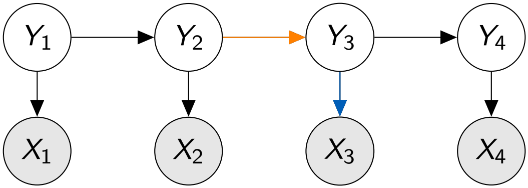
The Viterbi Algorithm

$$\textbf{Goal}: \arg\max_{Y_1,\dots,Y_n} P(Y_1,\dots,Y_n,X_1,\dots,X_n)$$

$$\text{For all} \quad i = 1,\dots,n \quad \text{and} \quad j = 1,\dots,k$$

$$F[i,j] = \max_{y_1,\dots,y_{i-1}} P(y_1,\dots,y_{i-1},Y_i = y_j, X_1,\dots,X_i)$$

$$= \max_{y_\ell} F[i-1,\ell]\, P(Y_i = y_j \mid Y_{i-1} = y_\ell)\, P(X_i \mid Y_i = y_j)$$



The Viterbi Algorithm (cont.)

This **dynamic programming** algorithm depends on the conditional independence.

$$\begin{aligned} F[i,j] &= \max_{y_1,\ldots,y_{i-1}} P(y_1,\ldots,y_{i-1},Y_i=y_j,X_1,\ldots,X_i) \\ &= \max_{y_\ell} P(y_1,\ldots,y_{i-1}=y_\ell,X_1,\ldots,X_{i-1}) \\ &\qquad P(Y_i=y_j \mid y_1,\ldots,Y_{i-1}=y_\ell,X_1,\ldots,X_{i-1}) \\ &\qquad P(X_i \mid y_1,\ldots,Y_{i-1}=y_\ell,Y_i=y_j,X_1,\ldots,X_{i-1}) \\ &= \max_{y_\ell} F[i-1,\ell]P(Y_i=y_j \mid Y_{i-1}=y_\ell)P(X_i \mid Y_i=y_j) \end{aligned}$$

Training HMMs with Supervised Data

Suppose we have a set of training data $\{(x_{1,1}, y_{1,1}), (x_{1,2}, y_{1,2}), \ldots,$ $(x_{1,n_1}, y_{1,n_1}), \ldots (x_{m,1}, y_{m,1}), \ldots, (x_{m,n_m}, y_{m,n_m})\}$.

m: number of sequences $\qquad n_i$: length of the i-th sequence.

We can directly estimate the HMM parameters (i.e., transition, emission and start probabilities) from the data by counting.

$$P(Y_i = y_j \mid Y_{i-1} = y_\ell) = \frac{\text{count}(y_\ell, y_j)}{\text{count}(y_\ell)}$$

$$P(X_i = x_j \mid Y_i = y_\ell) = \frac{\text{count}(x_j, y_\ell)}{\text{count}(y_\ell)}$$

$$P(Y_i = y_j) = \frac{\text{count}(\widehat{y}_j)}{m}$$

HMM Induction

What if the training data is not labeled?

We have a set of training input only $\{x_{1,1}, x_{1,2}, \ldots, x_{1,n_1}, \ldots, x_{m,1}, \ldots, x_{m,n_m}\}$.

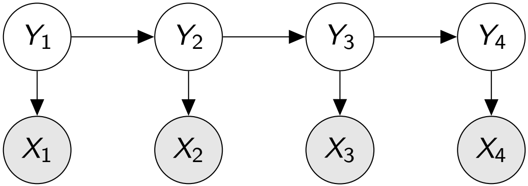
We can still assume the underlying model is an HMM and use the **Expectation-Maximization (EM) algorithm** to estimate the parameters.

- **Expectation**: compute the probability of the hidden states given the observed data.
- **Maximization**: update the model parameters based on the expected counts.

Also known as the **Baum-Welch algorithm**.

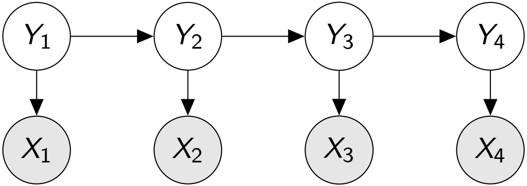
Forward Probability

$$
\begin{aligned}
\alpha_i(j) &= P(X_1 = x_1, \ldots, X_i = x_i, Y_i = y_j) \\
&= \sum_{j'=1}^{k} P(X_1 = x_1, \ldots, X_i = x_i, Y_{i-1} = y_{j'}, Y_i = y_j) \\
&= \sum_{j'=1}^{k} \alpha_{i-1}(j') P(X_i = x_i, Y_i = y_j \mid Y_{i-1} = y_{j'}, X_1, \ldots, X_{i-1}) \\
&= \sum_{j'=1}^{k} \alpha_{i-1}(j') P(X_i = x_i \mid Y_i = y_j) P(Y_i = y_j \mid Y_{i-1} = y_{j'})
\end{aligned}
$$



Backward Probability

$$\beta_{i}(j) = P(X_{i+1} = x_{i+1}, \ldots, X_n = x_n \mid Y_{i} = y_{j})$$
$$= \sum_{j'=1}^{k} \beta_{i+1}(j') P(X_{i+1} = x_{i+1} \mid Y_{i+1} = y_{j'}) P(Y_{i+1} = y_{j'} \mid Y_i = y_j)$$



Forward-Backward Probability

Given α and β, we can compute the **forward-backward probability** (i.e., soft count):

$$\begin{aligned}\alpha_i(j)\beta_i(j) &= P(X_{1:i}, Y_i = y_j)P(X_{i+1:n} \mid Y_i = y_j) \\ &= P(X_{1:n}, Y_i = y_j) \\ &\propto P(Y_i = y_j \mid X_{1:n}) = \gamma_i(j)\end{aligned}$$

And also the **soft transition count**:

$$\xi_i(j, j') = P(Y_i = y_j, Y_{i+1} = y_{j'} \mid X_{1:n})$$

Estimated Soft Transition Count

$$\xi_i(j, j') = P(Y_i = y_j, Y_{i+1} = y_{j'} \mid X_{1:n})$$

$$\alpha_i(j) = P(X_{1:i}, Y_i = y_j)$$

$$\beta_{i+1}(j') = P(X_{i+2:n} \mid Y_{i+1} = y_{j'})$$

$$\alpha_i(j)\beta_{i+1}(j') = P(X_{1:i}, Y_i = y_j)P(X_{i+2:n} \mid Y_{i+1} = y_{j'})$$

What is missing to combine the above into a joint probability distribution?

$$P(Y_{i+1} = y_{j'} \mid Y_i = y_j, X_{1:n}) = P(Y_{i+1} = y_{j'} \mid Y_i = y_j)$$

$$P(X_{i+1} \mid Y_{i+1} = y_{j'}, Y_i = y_j, X_{1:n}) = P(X_{i+1} \mid Y_{i+1} = y_{j'})$$

$$\xi_i(j, j') = \frac{\alpha_i(j)\beta_{i+1}(j')P(Y_{i+1} = y_{j'} \mid Y_i = y_j)P(X_{i+1} \mid Y_{i+1} = y_{j'})}{P(X_{1:n})}$$

Training HMMs with EM

- **E-step**: compute the forward-backward probability γ and the soft transition probability ξ.
- **M-step**: update the model parameters based on the expected counts.

$$P(Y_1 = y_j) = \frac{\sum_{i=1}^{m} \gamma_1^{(i)}(j)}{m}$$

$$P(Y_i = y_j \mid Y_{i-1} = y_\ell) = \frac{\sum_{i=1}^{m} \sum_{t=1}^{n_i - 1} \xi_t^{(i)}(\ell, j)}{\sum_{i=1}^{m} \sum_{t=1}^{n_i - 1} \gamma_t^{(i)}(\ell)}$$

$$P(X_i = x_j \mid Y_i = y_\ell) = \frac{\sum_{i=1}^{m} \sum_{t=1}^{n_i} \gamma_t^{(i)}(\ell) \mathbb{I}(X_t = x_j)}{\sum_{i=1}^{m} \sum_{t=1}^{n_i} \gamma_t^{(i)}(\ell)}$$

The EM algorithm is guaranteed to converge to a **local maximum** of the likelihood function.

The Baum-Welch algorithm is a special case of the EM algorithms.

Semi-Supervised Learning of HMMs

If we have a small amount of labeled data and a large amount of unlabeled data, we can use the **semi-supervised learning** approach.

Estimate the model parameters with the labeled data, then use the EM algorithm to estimate the model parameters with the unlabeled data.

Complexity Analysis

- The Viterbi algorithm: time complexity $O(nk^2)$ and space complexity $O(nk)$.

$$F[i,j] = \max_{y_\ell} F[i-1,\ell] P(Y_i = y_j \mid Y_{i-1} = y_\ell) P(X_i \mid Y_i = y_j)$$

- The forward-backward algorithm: time complexity $O(nk^2)$ and space complexity $O(nk)$.

$$\alpha_i(j) = \sum_{j'=1}^{k} \alpha_{i-1}(j') P(X_i = x_i \mid Y_i = y_j) P(Y_i = y_j \mid Y_{i-1} = y_{j'})$$

$$\beta_i(j) = \sum_{j'=1}^{k} \beta_{i+1}(j') P(X_{i+1} = x_{i+1} \mid Y_{i+1} = y_{j'}) P(Y_{i+1} = y_{j'} \mid Y_i = y_j)$$

Next

Conditional Random Fields

Sequence Labeling with Neural Networks