Language Models: a probability distribution over a sequence of tokens.

What is a language model?

The classic definition of a language model (LM) is a **probability distribution over sequences of tokens**. Suppose we have a **vocabulary** $\square$ of a set of tokens. A language model p assigns each sequence of tokens $x_1, \dots, x_L \in \square$ a probability (a number between 0 and 1):

$$p(x_1, \dots, x_L).$$

The probability intuitively tells us how "good" a sequence of tokens is. For example, if the vocabulary is $\square = \{\text{ate}, \text{ball}, \text{cheese}, \text{mouse}, \text{the}\}$, the language model might assign (demo):

$$p(\text{the}, \text{mouse}, \text{ate}, \text{the}, \text{cheese}) = 0.02,$$

$$p(\text{the}, \text{cheese}, \text{ate}, \text{the}, \text{mouse}) = 0.01,$$

$$p(\text{mouse}, \text{the}, \text{the}, \text{cheese}, \text{ate}) = 0.0001.$$

Mathematically, a language model is a very simple and beautiful object. But the simplicity is deceiving: the ability to assign (meaningful) probabilities to all sequences requires extraordinary (but *implicit*) linguistic abilities and world knowledge.

For example, the LM should assign $\text{mouse the the cheese ate}$ a very low probability implicitly because it's ungrammatical (**syntactic knowledge**). The LM should assign $\text{the mouse ate the cheese}$ higher probability than $\text{the cheese ate the mouse}$ implicitly because of **world knowledge**: both sentences are the same syntactically, but they differ in semantic plausibility.

Generation. As defined, a language model p takes a sequence and returns a probability to assess its goodness. We can also generate a sequence given a language model. The purest way to do this is to sample a sequence $x_{1:L}$ from the language model p with probability equal to $p(x_{1:L})$, denoted:

$$x_{1:L} \sim p.$$

How to do this computationally efficiently depends on the form of the language model p. In practice, we do not generally sample directly from a language model both because of limitations of real language models and because we sometimes wish to obtain not an "average" sequence but something closer to the "best" sequence.

Autoregressive language models

A common way to write the joint distribution $p(x_{1:L})$ of a sequence $x_{1:L}$ is using the **chain rule of probability**:

$$p(x_{1:L}) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \cdots p(x_L \mid x_{1:L-1}) = \prod_{i=1}^{L} p(x_i \mid x_{1:i-1}).$$

For example (demo):

$$\begin{aligned} p(\text{the}, \text{mouse}, \text{ate}, \text{the}, \text{cheese}) = \; & p(\text{the}) \\ & p(\text{mouse} \mid \text{the}) \\ & p(\text{ate} \mid \text{the}, \text{mouse}) \\ & p(\text{the} \mid \text{the}, \text{mouse}, \text{ate}) \\ & p(\text{cheese} \mid \text{the}, \text{mouse}, \text{ate}, \text{the}). \end{aligned}$$

In particular, $p(x_i \mid x_{1:i-1})$ is a **conditional probability distribution** of the next token x_i given the previous tokens $x_{1:i-1}$.

Contextual Embeddings:

Contextual embeddings. As a prerequisite, the main key development is to associate a sequence of tokens with a corresponding sequence of contextual embeddings:

$$[\text{the}, \text{mouse}, \text{ate}, \text{the}, \text{cheese}] \stackrel{\varphi}{\Rightarrow} \left[\binom{1}{0.1}, \binom{0}{1}, \binom{1}{1}, \binom{1}{-0.1}, \binom{0}{-1}\right].$$

- As the name suggests, the contextual embedding of a token depends on its context (surrounding words); for example, consider the.
- Notation: We will $\varphi : \square^L \to \mathbb{R}^{d \times L}$ to be the embedding function (analogous to a feature map for sequences).
- For a token sequence $x_{1:L} = [x_1, \ldots, x_L]$, φ produces contextual embeddings $\varphi(x_{1:L})$.

Encoder-only:

Encoder-only (BERT, RoBERTa, etc.). These language models produce contextual embeddings but cannot be used directly to generate text.

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

These contextual embeddings are generally used for classification tasks (sometimes boldly called natural language understanding tasks).

- Example: sentiment classification

$$[[\text{CLS}], \text{the}, \text{movie}, \text{was}, \text{great}] \Rightarrow \text{positive}.$$

- Example: natural language inference

$$[[\text{CLS}], \text{all}, \text{animals}, \text{breathe}, [\text{SEP}], \text{cats}, \text{breathe}] \Rightarrow \text{entailment}.$$

- Pro: contextual embedding for x_i can depend **bidirectionally** on both the left context ($x_{1:i-1}$) and the right context ($x_{i+1:L}$).
- Con: cannot naturally **generate** completions.
- Con: requires more **ad-hoc training** objectives (masked language modeling).

1. Decoder-only (e.g., GPT-3): compute unidirectional contextual embeddings, generate one token at a time
2. Encoder-only (e.g., BERT): compute bidirectional contextual embeddings
3. Encoder-decoder (e.g., T5): encode input, decode output

Decoder-only:

Decoder-only (GPT-2, GPT-3, etc.). These are our standard autoregressive language models, which given a prompt $x_{1:i}$ produces both contextual embeddings and a distribution over next tokens x_{i+1} (and recursively, over the entire completion $x_{i+1:L}$).

$$x_{1:i} \Rightarrow \phi(x_{1:i}), p(x_{i+1} \mid x_{1:i}).$$

- Example: text autocomplete

$$[[\text{CLS}], \text{the}, \text{movie}, \text{was}] \Rightarrow \text{great}$$

- Con: contextual embedding for x_i can only depend **unidirectionally** on both the left context ($x_{1:i-1}$).
- Pro: can naturally **generate** completions.
- Pro: **simple training** objective (maximum likelihood).

Encoder-Decoder:

Encoder-decoder (BART, T5, etc.). These models in some ways can the best of both worlds: they can use bidirectional contextual embeddings for the input $x_{1:L}$ and can generate the output $y_{1:L}$.

$$x_{1:L} \Rightarrow \phi(x_{1:L}), p(y_{1:L} \mid \phi(x_{1:L})).$$

- Example: table-to-text generation

$$[\text{name}, :, \text{Clowns}, |, \text{eatType}, :, \text{coffee}, \text{shop}] \Rightarrow [\text{Clowns}, \text{is}, \text{a}, \text{coffee}, \text{shop}].$$

- Pro: contextual embedding for x_i can depend **bidirectionally** on both the left context ($x_{1:i-1}$) and the right context ($x_{i+1:L}$).
- Pro: can naturally **generate** outputs.
- Con: requires more **ad-hoc training** objectives.

Decoder-only architectures:

- Language Models are Unsupervised Multitask Learners. *Alec Radford, Jeff Wu, R. Child, D. Luan, Dario Amodei, Ilya Sutskever*. 2019. Introduces **GPT-2** from OpenAI.
- Language Models are Few-Shot Learners. *Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, J. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. Henighan, R. Child, A. Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, Dario Amodei*. NeurIPS 2020. Introduces **GPT-3** from OpenAI.
- Scaling Language Models: Methods, Analysis&Insights from Training Gopher. *Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, J. Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, G. V. D. Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John F. J. Mellor, I. Higgins, Antonia Creswell, Nathan McAleese, Amy Wu, Erich Elsen, Siddhant M. Jayakumar, Elena Buchatskaya, D. Budden, Esme Sutherland, K. Simonyan, Michela Paganini, L. Sifre, Lena Martens, Xiang Lorraine Li, A. Kuncoro, Aida Nematzadeh, E. Gribovskaya, Domenic Donato, Angeliki Lazaridou, A. Mensch, J. Lespiau, Maria Tsimpoukelli, N. Grigorev, Doug Fritz, Thibault Sottiaux, Mantas Pajarskas, Tobias Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d'Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, I. Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew Johnson, Blake A. Hechtman, Laura Weidinger, Iason Gabriel, William S. Isaac, Edward Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem W. Ayoub, Jeff Stanway, L. Bennett, D. Hassabis, K. Kavukcuoglu, Geoffrey Irving*. 2021. Introduces **Gopher** from DeepMind.
- Jurassic-1: Technical details and evaluation. *Opher Lieber, Or Sharir, Barak Lenz, Yoav Shoham*. 2021. Introduces **Jurassic** from AI21 Labs.

Encoder-only architectures:

- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova*. NAACL 2019. Introduces **BERT** from Google.
- RoBERTa: A Robustly Optimized BERT Pretraining Approach. *Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, M. Lewis, Luke Zettlemoyer, Veselin Stoyanov*. 2019. Introduces **RoBERTa** from Facebook.

Encoder-decoder architectures:

- BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. *M. Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, Luke Zettlemoyer*. ACL 2019. Introduces **BART** from Facebook.
- Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Colin Raffel, Noam M. Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, W. Li, Peter J. Liu*. J. Mach. Learn. Res. 2019. Introduces **T5** from Google.

Decoder-only:

Decoder-only models

Recall that an autoregressive language model defines a conditional distribution:

$$p(x_i \mid x_{1:i-1}).$$

We define it as follows:

- Map $x_{1:i-1}$ to contextual embeddings $\varphi(x_{1:i-1})$.
- Apply an embedding matrix $E \in \mathbb{R}^{V \times d}$ to obtain scores for each token $E\varphi(x_{1:i-1})_{i-1}$.
- Exponentiate and normalize it to produce the distribution over x_i.

Succinctly:

$$p(x_{i+1} \mid x_{1:i}) = \text{softmax}(E\varphi(x_{1:i})_i).$$

Maximum likelihood. Let θ be all the parameters of large language models.

Let □ be the **training data** consisting of a set of sequences. We can then follow the maximum likelihood principle and define the following negative log-likelihood objective function:

$$\square(\theta) = \sum_{x_{1:L} \in \square} -\log p_\theta(x_{1:L}) = \sum_{x_{1:L} \in \square} \sum_{i=1}^{L} -\log p_\theta(x_i \mid x_{1:i-1}).$$

There's more to say about how to efficiently optimize this function, but that's all there is for the objective.

Encoder-only:

Encoder-only models

Unidirectional to bidirectional. A decoder-only model trained using maximum likelihood above also produces (unidirectional) contextual embeddings, but we can provide stronger bidirectional contextual embeddings given that we don't need to generate.

BERT. We will first present the BERT objective function, which contains two terms:

1. Masked language modeling
2. Next sentence prediction

Take the example sequence for natural language inference (predict entailment, contradiction, or neutral):

$$x_{1:L} = [\text{[CLS]}, \text{all}, \text{animals}, \text{breathe}, \text{[SEP]}, \text{cats}, \text{breathe}].$$

There are two special tokens:

- [CLS]: contains the embedding used to drive classification tasks
- [SEP]: used to tell the model where the first (e.g., premise) versus second sequence (e.g., hypothesis) are.

Using our notation from the previous lecture, the BERT model is defined as:

$$\text{BERT}(x_{1:L}) = \text{TransformerBlock}^{24}(\text{EmbedTokenWithPosition}(x_{1:L}) + \text{SentenceEmbedding}(x_{1:L})) \in \mathbb{R}^{d \times L},$$

where $\text{SentenceEmbedding}(x_{1:L})$ returns one of 2 vectors depending on the sequence:

- $e_A \in \mathbb{R}^d$ for tokens **left** of [SEP], and
- $e_B \in \mathbb{R}^d$ for tokens **right** of [SEP].