

- Language models assign probabilities to sequences
- The probability of sequence is estimated based on probability of each item (word) in the sequence
- Probability of each word in the sequence is predicted based on its context
- Language models can be trained with unlabeled text
- Language models have been traditionally an important part of some NLP applications (translation, ASR)
- Recently, they are used for (almost) any NLP task

N-gram language models

- We use probabilities of parts of the sentence (words) to calculate the probability of the whole sentence

$$P(w_1, w_2, \dots, w_m) = P(w_2 | w_1) \times P(w_3 | w_1, w_2) \times \dots \times P(w_m | w_1, w_2, \dots, w_{m-1})$$

- Making a conditional independence assumption, we can simplify the model

$$P(w_1, w_2, \dots, w_m) = P(w_2 | w_1) \times P(w_3 | w_1) \times \dots \times P(w_m | w_{m-1})$$

Issues with n-gram language models

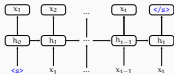
- Words are symbolic units. No notion of word similarity
- Morphologically complex languages: different inflections of the word
- Difficult to capture long-range dependencies
- No information from the following words

Feed-forward neural models



- Main idea is the same as n-gram models: predict the next word from a limited context
- The first layer is typically embeddings
- Continuous representations allow modeling similarities
- We can include right context, too

RNN language models

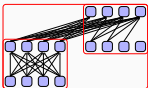


- RNNs can trivially be trained as language models
- Hidden representations provide contextual embeddings
- Can potentially handle long-range dependencies

Shortcomings of RNN language models

- RNNs solve many of the issues with n-gram (and feed-forward) language models
- Although RNN language models can model dependencies across arbitrary distances in theory, the memory is generally short even for gated RNNs
- RNN processing is inherently sequential to calculation of representations at each step require all earlier steps to be done

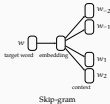
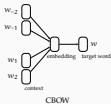
Transformer language models



- The decoder of the original transformer is simply a language model: it predicts the next word based on earlier words
- Encoder-decoder models can be used as language models if trained using autoencoder (or similar) objectives
- Encoder side of the Transformer can also be used as a language model with masked language model (MLM) objective

Short detour to word2vec

Is word2vec a language model?



A real-world RNN language model: ELMo

- ELMo is the first popular pre-trained language model providing contextualized representations
- ELMo is simply a (stacked/deep) LSTM language model trained on a large corpus (30 million sentences)
- Each layer in ELMo builds contextual representations for words
- ELMo is bidirectional: forward and backward representations are concatenated
- Similar to static word embeddings, ELMo representations can be used for downstream NLP tasks
- Note that unlike the word embeddings, the whole model needs to be distributed

Back to Transformers: a recap

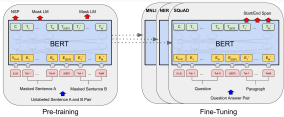


- The first layer is an *embedding* layer: no information from context information
- Subsequent layers use attention followed by a non-linear transformation (feed-forward layer)
- Feed-forward layer is an up-projection followed by projection back to input/output dimensions
- Input and output dimensions to each Transformer block is the same
- Layer normalization is after (sometimes before) the attention and feed-forward calculations

Encoder only transformers: masked language models

- Masked language models replace some of the words in the input with a special symbol [MASK]
- The task of the model is to predict the masked words
- The idea is similar to 'fill in the blanks' questions (cloze tests)
- It is also similar to 'noisy' autoencoding, but we do not reconstruct the full sentence, but only the masked tokens
- In the process, the model learns contextual representations that are useful for other NLP tasks

BERT: architecture



BERT: pretraining

- BERT uses two training objectives:
 - MLM: masked language modeling
 - NSP: next sentence prediction
- Input to BERT is pairs of sentences with [SEP] between them
- MLM typically predict the masked tokens, but some tokens are replaced with arbitrary words
- NSP is a binary classification task trying to predict whether the second sentence follows the first one
- Later models (e.g., RoBERTa) typically drop the NSP objective

How to use encoder-only LMs in downstream applications?

- For downstream tasks, we typically *finetune* BERT with a supervised objective
- For sequence labeling task, we replace the NSP 'head' with a classification layer
- For sequence labeling we attach a classifier to every step in the sequence
- The new 'heads' are typically randomly initialized
- Finetuning procedure updates all the weights (including the language model weights trained during pretraining)

A note on representations from BERT

- Embeddings produced by BERT-like models are 'contextualized': they assign different representations for different senses of words
- Representations learned are more useful for downstream (classification) tasks than static embeddings (e.g., word2vec)
- It is also often claimed that representations from different layers learn different representations (with mixed results)
 - Earlier layers learning morphology and syntax
 - Later layers semantics, world knowledge
- BERT representations are *anisotropic*: distances and similarities are typically not very meaningful
- Subword tokenization may also complicate obtaining representations for words

Encoder-only models: a few examples

- BERT: the first encoder-only language model
- RoBERTa: the same architecture, trained longer with more data, some improvements to training procedure
- XLNet: multilingual version of RoBERTa supporting 100 languages
- ModernBERT: longer context, applying some of the lessons learned from other architectures
- Monolingual models for many languages exist
- There are also domain-specific architectures, e.g., for legal or medical texts

Encoder-decoder architectures

- The original transformer architecture without modification can also serve as pretrained language models
- It is particularly suitable for generation tasks (machine translation, summarization, questions answering)
- Encoder-decoder models can also be used for classification (and less commonly regression) tasks: model is finetuned to produce class label, given text input(s)
- This is a relatively less-common approach
- Well-known models include BART and T5

Decoder-only models

- It is relatively trivial to train the decoder side of the Transformer as a language model
- The attention mask is set up to attend only to preceding input: task becomes next token prediction
- Most well-known large language models are decoder-only models, e.g., GPT family, Llama, DeepSeek, ...
- They are also known as *causal* LMs, or simply generative LMs
- These models are typically trained with much larger data, and tend to learn much more about language (and the world)
- Modern LLMs are not only trained with language modeling objective, they go through further training after LM pretraining

How to use generative models

- LLMs are next word predictors, using them they do classification, or interact as chat agents require some additional work
- By default, one can construct special 'prompts' to use LLMs for certain tasks
 - The sentiment of the sentence "Not worth the time" is _____
 - We can either let the model predict the next word
 - Or decide based on P(positive|context) and P(negative|context)
- Similar prompts can be built for other tasks
- More commonly, the LLMs go through additional training to interact with people the way we expect them to

Decoding from LLMs

- Decoding is the tasks of producing new tokens given the context:
 - Start with the context (or prompt)
 - Get the highest probability token given the context
 - Add the token to the context, and repeat until we sample end-of-sequence symbol
- Greedy decoding often leads to 'boring' text without much variation
- Instead we sample a random word, based on the softmax probabilities

Sampling with temperature

- One way to encourage further diversity is *temperature*.
- Instead of sampling based on softmax(x), we use softmax(x/T)
- $T = 1$ it is equal to normal sampling
- As T gets closer to 0, we approach greedy decoding: probability of most likely word tends to 1
- With high values for T, probabilities become smoother, allowing sampling less likely tokens

Post-training in LLMs

- Pretrained LLMs are useful, but for their typical use they generally go through a 'post-training'
 - Training on interactive prompts to adjust to typical human interaction, and increase their task performance: typically with supervised methods
 - Aligning with human preferences: typically through reinforcement learning

Finetuning LLMs

- The LLMs are typically very big, finetuning them require substantial resources
- They are typically used through zero-shot or few-shot prompting (so-called 'in-context learning')
- When needed, *parameter-efficient finetuning* is more common
 - Adapters: keep LM weights frozen, add new trainable parameters
 - Prefix-tuning: only update some input parameters
 - LoRA: Use low-rank approximation for parameter updates

Computational complexity of Transformers

- What is the computational complexity of Transformers in the sequence length n ?
 - For each time step at each layer, we need to calculate attention over all previous time steps
 - This results in a $O(n^2)$ complexity at each layer

- We want our sequences to be short
- Also remember: we also want to keep vocabulary size short (to avoid expensive softmax, among other problems)

Subword tokenization: BPE

- Byte-pair encoding (BPE) is an algorithm to segment a set of words into sub-words
- The general idea is:
 - Start with a vocabulary with bytes (or characters)
 - Iteratively add most common pair to the vocabulary
 - Stop when vocabulary size increases to a pre-defined number
- Many current models use a version of BPE algorithm for tokenization with some alternations
- The vocabulary size differ: BERT: 30K, RoBERTa: 50K, XLM-R (large): 250K
LLama3: 128K

Summary

- There are multiple neural architectures that can be used for language modeling
- The state-of-the-art architectures are based on Transformer, and can be:
 - Encoder-only (e.g., BERT family)
 - Decoder-only (e.g., GPT family)
 - Encoder-decoder (e.g., T5)
- Reading: Jurafsky and Martin, 2026, Chapter 11

ext

- More on Transformer language models
- Reading: Jurafsky and Martin, 2026, Chapter 10