Foundations of machine learning

Large Language Models and Transformers

Maximilian Kasy

Department of Economics, University of Oxford

Winter 2026

Outline

- Natural language model as a prediction problem.
- Self-supervised learning.
- Self-attention.
- Transformer models.
- Generative AI and beam search.

Takeaways for this part of class

- Natural language models are joint probability distributions for sequences of tokens $(X_1, X_2, \ldots)$.
- Tasks such as translation or question answering are based on conditional probability distributions of sequences $(Y_1, Y_2, \ldots)$ given $(X_1, X_2, \ldots)$.
- Self-supervised learning predicts tokens X_t given their context $\ldots, X_{t-1}, X_{t+1}, \ldots$.
- A successful class of models uses self-attention, stacked into multiple layers. Such models are called Transformers.
- Generative AI is based on sequential prediction of X_t given $(X_1, \ldots, X_{t-1})$. Finding high-probability predicted sequences often uses beam-search.

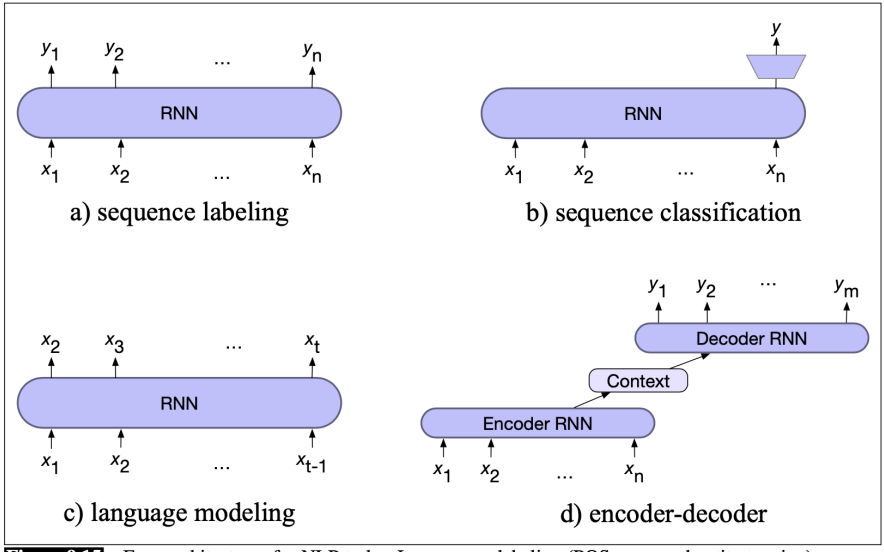

Figure 9.15 Four architectures for NLP tasks. In sequence labeling (POS or named entity tagging) we map each input token x_i to an output token y_i. In sequence classification we map the entire input sequence to a single class. In language modeling we output the next token conditioned on previous tokens. In the encoder model we have two separate RNN models, one of which maps from an input sequence $\mathbf{x}$ to an intermediate representation we call the **context**, and a second of which maps from the context to an output sequence $\mathbf{y}$.

Prediction tasks in language processing

- Suppose the data consist of pairs of sequences of "tokens:" $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_m)$.
- Various tasks in language processing require to estimate models $\hat{P}$ for

$$P(Y|X)$$

- Typical loss function for an observation (x, y): Negative log likelihood,

$$-\log \hat{P}(Y = y|X = x).$$

Examples:

1. Machine translation:
   x is a sentence in the source language.
   y is a sentence in the target language.

2. Question answering:

Self-supervised learning

- These prediction problems require specific data – pairs of x and y.
- There is much greater availability of data of "unlabeled" sequences x.
  E.g., all the text on the internet (Wikipedia, Arxiv, Github, ...).
- Self-supervised learning fits models for the distribution of such sequences.

Leading cases:

1. Autoregressive models:
   Model $P(x_i|x_1,\ldots,x_{i-1})$, for all i.
2. Masking:
   Model $P(x_i|x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n)$, for all i.

Masking

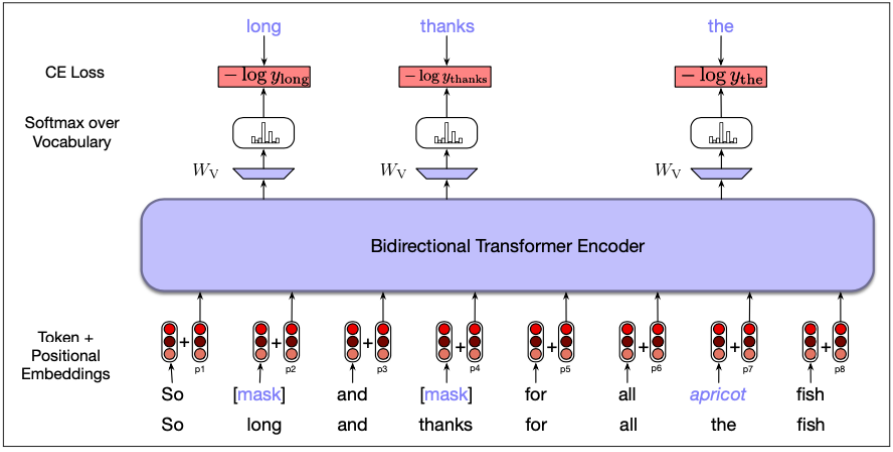

Figure 11.5 Masked language model training. In this example, three of the input tokens are selected, two of which are masked and the third is replaced with an unrelated word. The probabilities assigned by the model to these three items are used as the training loss. (In this and subsequent figures we display the input as words rather than subword tokens; the reader should keep in mind that BERT and similar models actually use subword tokens instead.)

Embeddings and pre-training

- Many language models are trained in two steps:
  1. *Self-supervised learning* on a large corpus of sequences x, using masking.
     This yields an embedding (representation) of the source data x.
  2. *Fine-tuning* on a task-specific corpus:
     Using the embeddings from 1. as predictors for y.
- This is also known as *transfer learning*.
  It yields much better results than simply training on the task-specific corpus.

The transformer architecture

- How do we get an embedding for a sequence of tokens?
- What functional form should we choose?
- Leading answer: *Transformers*.
- Transformers consist of multiple transformer blocks.
- Each of which includes self-attention layers.

Self-attention layers

- Take as given a sequence of input vectors $x_1, \ldots, x_n$,
- We want to *transform it*,
  to produce a sequence of output vectors $y_1, \ldots, y_n$
  of the same dimension.
- y_j is supposed to encode the meaning of x_i *in the context* of the other x_j.
- First step: Take a linear tranformation of the x_i.

$$v_i = W^v \cdot x_i.$$

- Second step: Take a weighted average of the v_i to get the output y_i.

$$y_i = \sum_j \alpha_{ij} v_j.$$

Self-attention layers continued

- The weights α_{ij} capture the importance of x_j as context for x_i.
- But where do the weights come from? *Self-attention*!

$$\alpha_{ij} = \frac{\exp(score_{ij})}{\sum_{j'} \exp(score_{ij'})}.$$

  Normalizing sum of weights to 1 (aka softmax / multinomial logit).

- $score_{ij}$: Relevance of x_j as context for x_i.

$$\begin{aligned} score_{ij} &= \langle q_i, k_j \rangle && \text{inner product} \\ q_i &= W^q \cdot x_i && \text{query} \\ k_j &= W^k \cdot x_j && \text{key} \end{aligned}$$

- Contrast to time series models: Weights depend only on $|i-j|$.
  $\Rightarrow$ Would not recognize importance of far-away sentence parts for context.

Backward looking and bi-directional self-attention

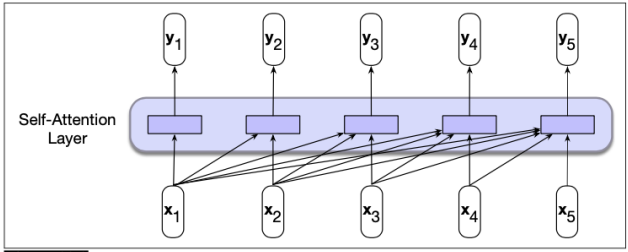

Figure 11.1 A causal, backward looking, transformer model like Chapter 10. Each output is computed independently of the others using only information seen earlier in the context.

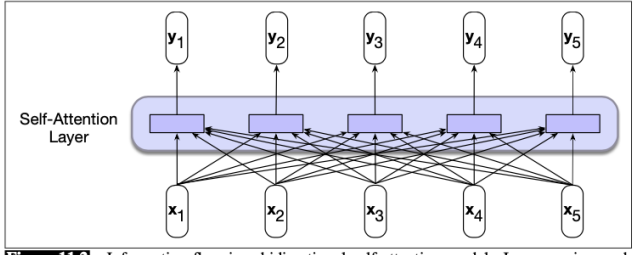

Figure 11.2 Information flow in a bidirectional self-attention model. In processing each element of the sequence, the model attends to all inputs, both before and after the current one.

Transformer blocks

- Self-attention layers are packaged with some additional transformations as follows:

$$z = LayerNorm(x + SelfAttention(x))$$
$$y = LayerNorm(z + FFN(z))$$

- $LayerNorm(x)$ normalizes $x = (x_1, \ldots, x_n)$
  by subtracting the mean and dividing by the standard deviation.

- The addition of x to $SelfAttention(x)$ is called "residual connection."
  This keeps raw information from previous input.

- $FFN(z)$ is a standard feed-forward neural network.

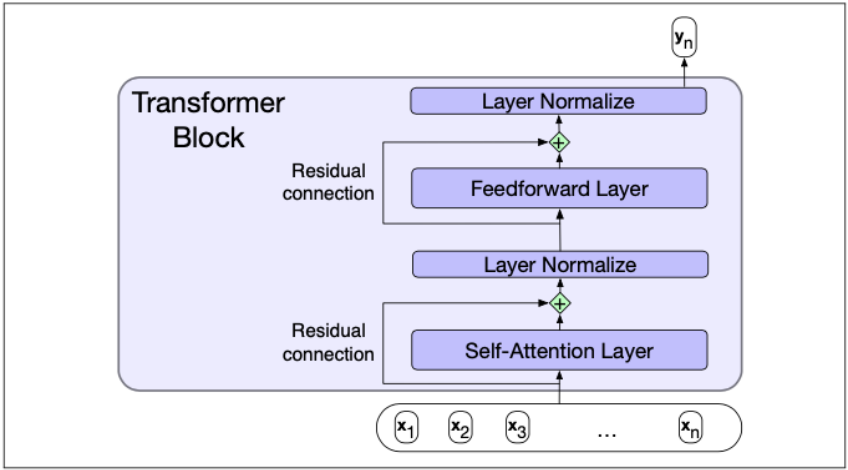

Figure 10.4 A transformer block showing all the layers.

Multi-head attention

- Tweak on the transformer block:
  Replace the single self-attention layer
  by several parallel versions, indexed by b.

- Thus:

$$y_i^b = \sum_j \alpha_{ij} \cdot \left[W^{v,b} \cdot x_i \right], \qquad \alpha_{ij}^b = \frac{\exp(score_{ij}^b)}{\sum_{j'} \exp(score_{ij'}^b)},$$
$$score_{ij}^b = \left\langle \left[W^{q,b} \cdot x_i \right], \left[W^{k,b} \cdot x_j \right] \right\rangle.$$

- The rest of the transformer block stays the same.

- Motivation: Context matters in various ways.

Generative AI

- Suppose you have fit an autoregressive model, which gives

$$\hat{P}(y_{i+1}|x, y_1, \ldots, y_{i-1}).$$

- Suppose you would like to generate a prediction of y, given an input x.

- That is you would like to find

$$\hat{y} = \underset{y}{\text{argmax}}\ \hat{P}(y|x) = \underset{y}{\text{argmax}} \prod_i \hat{P}(y_i|x, y_1, \ldots, y_{i-1}).$$

- Such forecasting of autoregressive models is at the heart of "generative AI."

Greedy sampling

- Naive idea: Sequentially find the highest probability prediction, one step at a time:

$$\hat{y}_i = \text{argmax } \hat{P}(y_{i+1}|x, y_1, \ldots, y_{i-1}).$$

- This is known as greedy search.

- Problem:
  This does not take into account the impact of the choice of $\hat{y}_i$
  on the availability of high probability choices later.

- Dynamic programming problem!

Beam search

- Exhaustive search of the tree of possible sequences is too costly.
- Compromise: Beam search.
  1. Start with the k highest-probability choices for $\hat{y}_1$.
  2. For each of these choices separately,
     find the k highest probability choices for $\hat{y}_2$.
  3. Keep the k sequences of $\hat{y}_1, \hat{y}_2$ with the highest probability,
     discard the rest.
  4. Iterate.

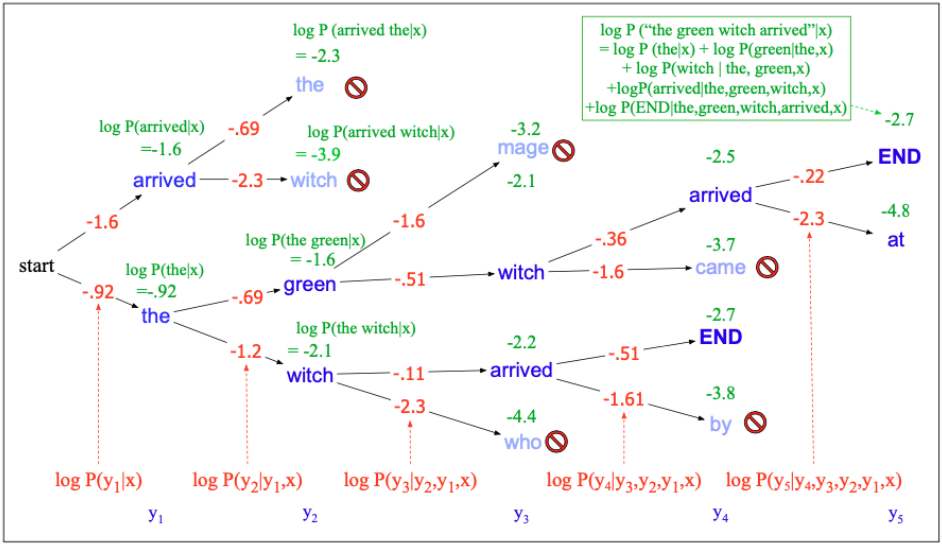

Figure 10.10 Scoring for beam search decoding with a beam width of $k = 2$. We maintain the log probability of each hypothesis in the beam by incrementally adding the logprob of generating each next token. Only the top k paths are extended to the next step.

References

Speech and Language Processing,
Dan Jurafsky and James H. Martin, 2023,
chapters 10-11.