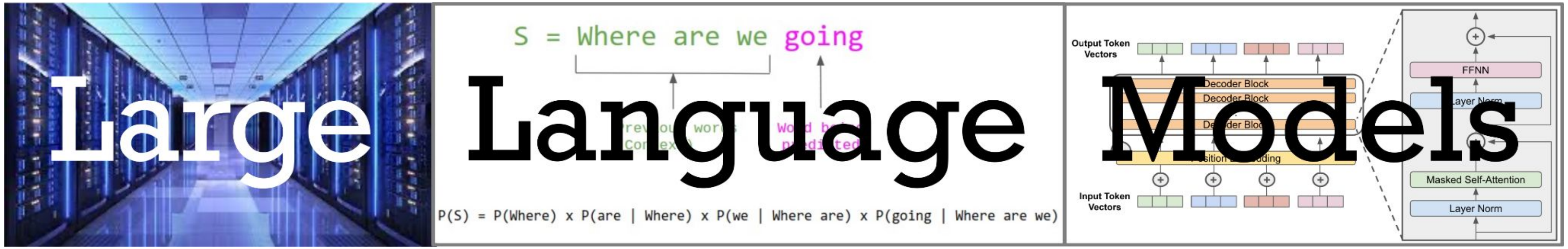


CIS 7000 - Fall 2024

The Transformer Architecture: Part II

Professor Mayur Naik

Slides adapted in part from Stanford's CS224N: Natural Language Processing with Deep Learning (Spring'24) and Chapters 9, 10, 11, 13 of Jurafsky/Martin's book "Speech and Language Processing" (3rd ed).

Recap Of Last Lecture

- Impact of Transformers
- From Recurrence (RNNs) to Attention-Based Models
- The Transformer Block

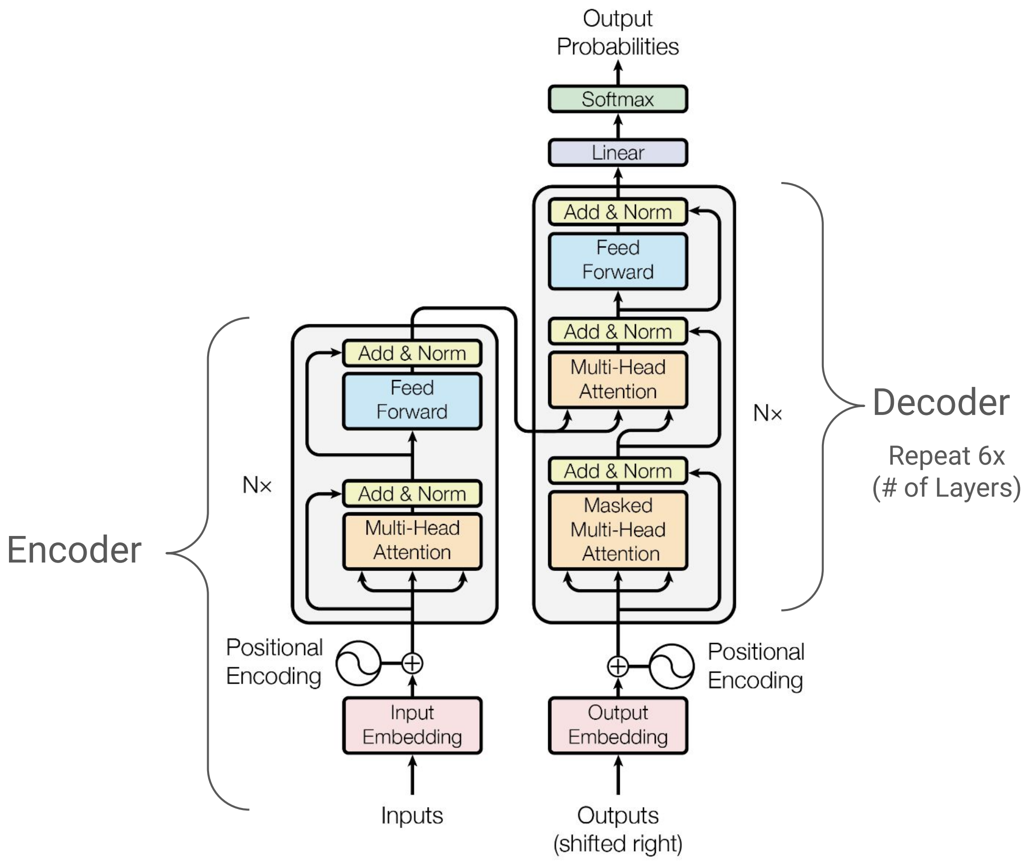
Today's Agenda

- The Overall Transformer Model
- Inference and Training
- State-of-the-Art Transformer Case Studies
- Drawbacks of Transformers

The Transformer Architecture

Last lecture, we completed the Encoder.

This lecture we will look at the Decoder...

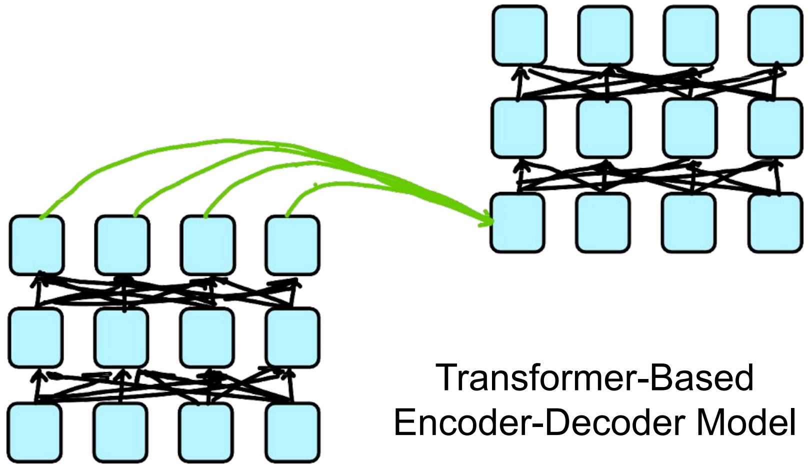


Decoder: Masked Self-Attention

Problem: How do we keep the decoder from “cheating”? If we have a language modeling objective, can't the network just look ahead and "see" the answer?

Solution: Masked Multi-Head Attention.

At a high-level, we hide (mask) information about future tokens from the model.


Transformer-Based Encoder-Decoder Model

Masking the Future in Self-Attention

To use self-attention in decoders, we need to ensure we can't peek at the future.

At every timestep, we could change the set of keys and queries to include only past words. (Inefficient!)

To enable parallelization, we mask out attention to future words by setting attention scores to $-\infty$.

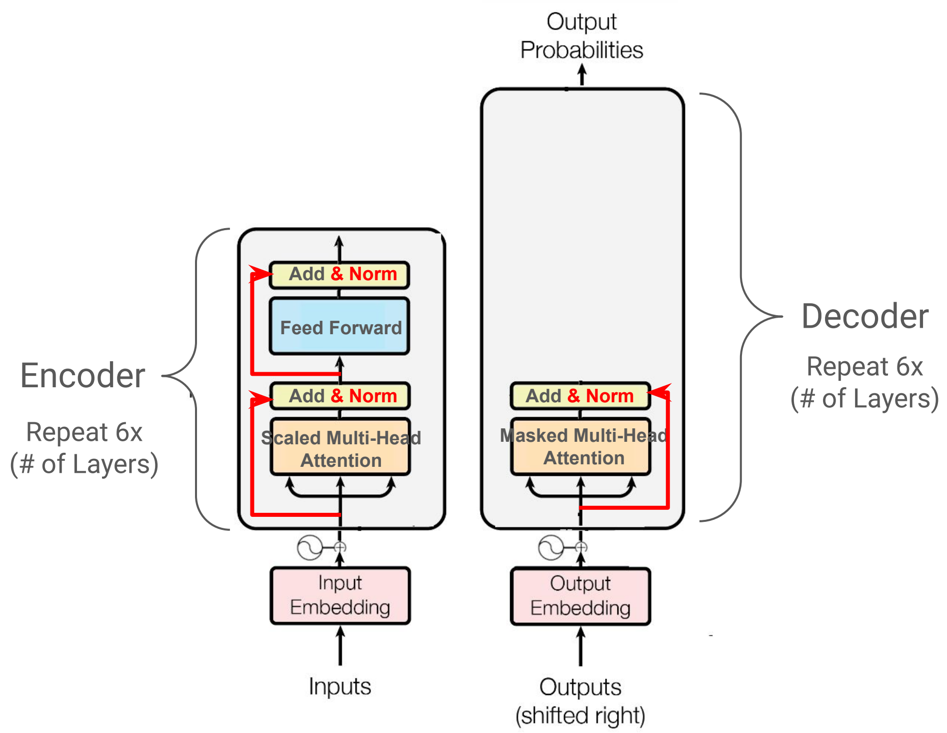
$$e_{ij} = \begin{cases} q_i^\top k_j, j < i \\ -\infty, j \geq i \end{cases}$$

We can look at these (not greyed out) words

For encoding these words

$$\begin{array}{c|cccc} & \text{[START]} & \text{The} & \text{chef} & \text{who} \\ \hline \text{[START]} & -\infty & -\infty & -\infty & -\infty \\ \text{The} & & -\infty & -\infty & -\infty \\ \text{chef} & & & -\infty & -\infty \\ \text{who} & & & & -\infty \end{array}$$

Decoder: Masked Multi-Head Self-Attention



Encoder-Decoder Attention

How does the decoder focus on appropriate places in the input sequence? Using encoder-decoder / cross attention!

Let $\mathbf{h}_1, \dots, \mathbf{h}_N$ be output vectors from the encoder

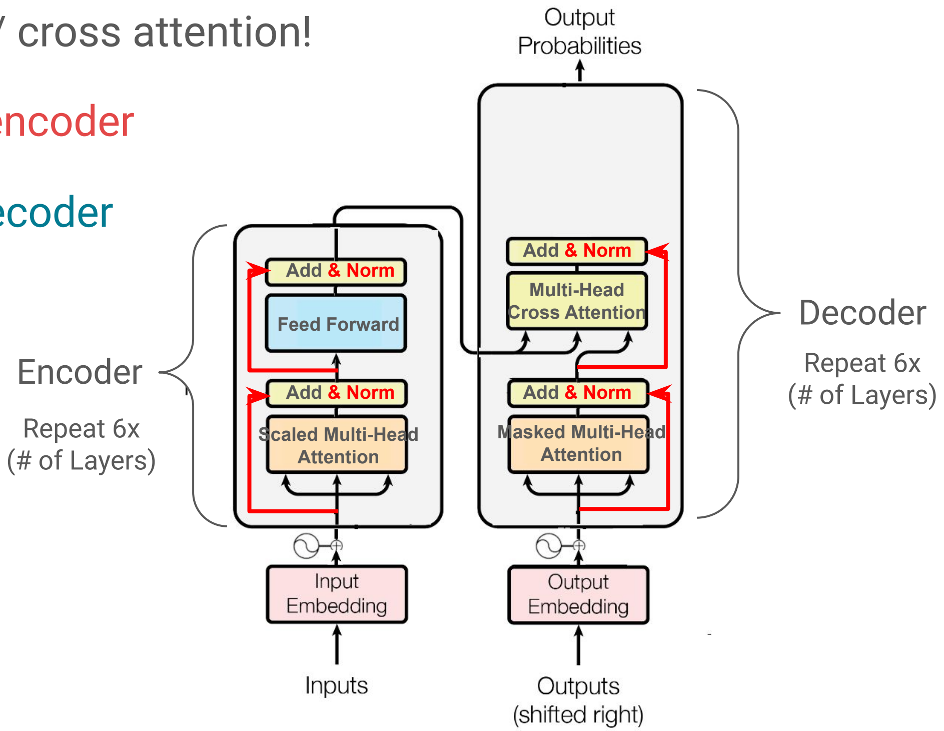
Let $\mathbf{z}_1, \dots, \mathbf{z}_N$ be input vectors from the decoder

Then **keys** and **values** are drawn from the encoder (like a memory):

$$\mathbf{k}_i = \mathbf{W}^K \mathbf{h}_i \text{ and } \mathbf{v}_i = \mathbf{W}^V \mathbf{h}_i$$

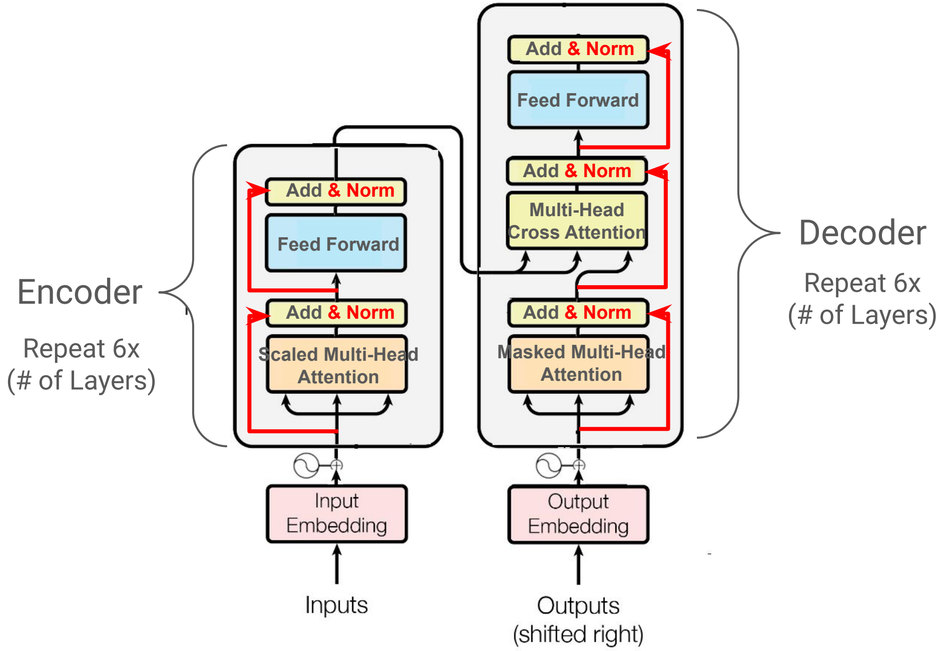
whereas the **queries** are drawn from the decoder:

$$\mathbf{q}_i = \mathbf{W}^Q \mathbf{z}_i$$



Decoder: Finishing Touches!

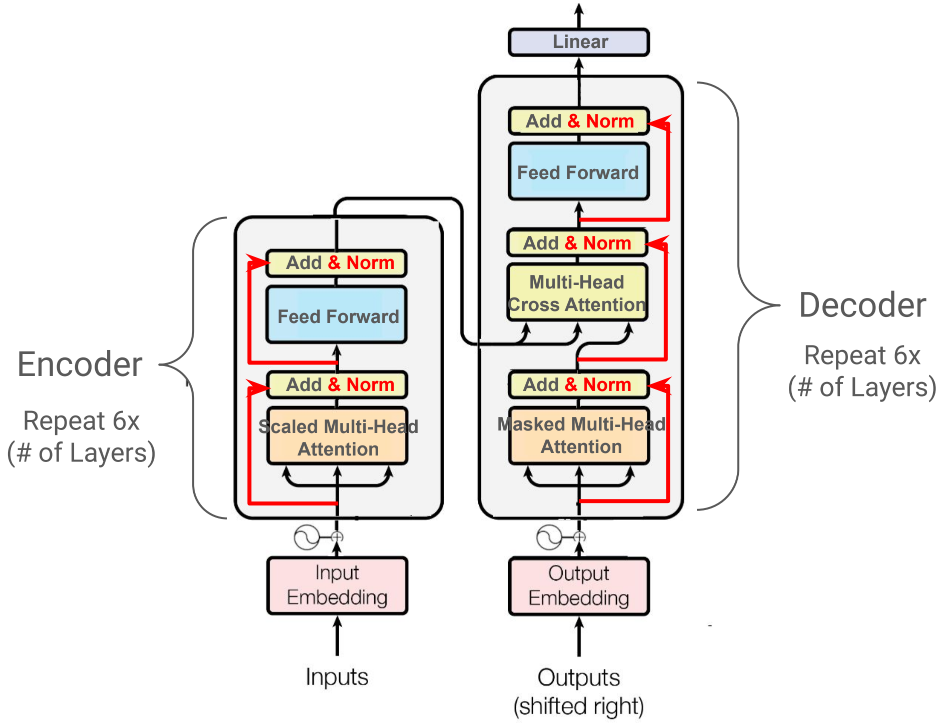
Add a feed forward layer (with residual connections and layer norm).



Decoder: Finishing Touches!

Add a feed forward layer (with residual connections and layer norm).

Add a final linear layer to project the embeddings into a much longer vector of length vocab size (logits).

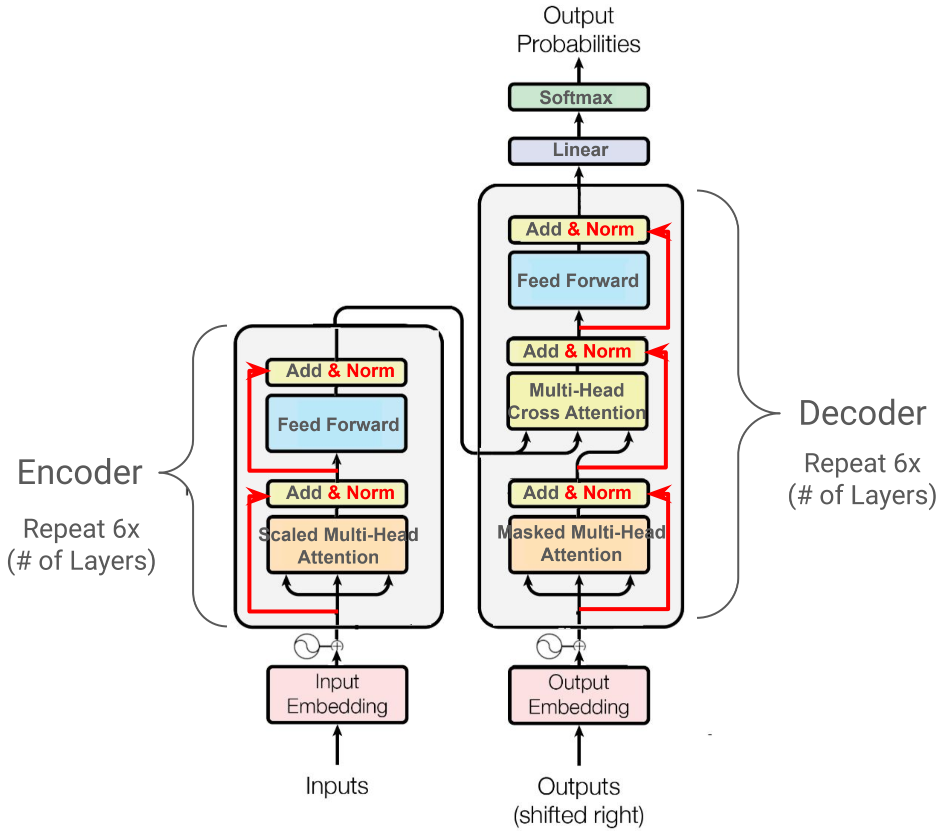


Decoder: Finishing Touches!

Add a feed forward layer (with residual connections and layer norm).

Add a final linear layer to project the embeddings into a much longer vector of length vocab size (logits).

Add a final softmax to generate a probability distribution of possible next words!



Differences in Attention Mechanism of RNN vs. Transformer

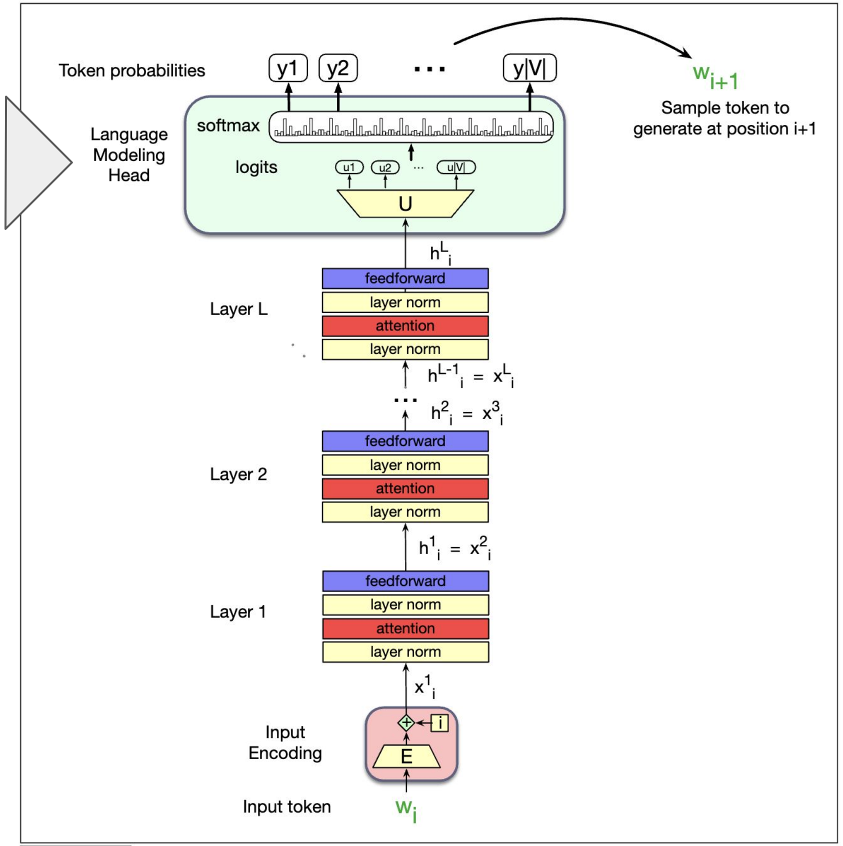
Feature	RNN with Attention (Bahdanau et al. 2015)	Transformer
Attention Type	Additive (Bahdanau) Attention	Scaled Dot Product Attention
Alignment	Based on decoder hidden state and encoder hidden states	Based on dot-product of query and keys (global attention)
Efficiency	Processes sequences step-by-step	Parallel processing of all positions
Context	Weighted sum of encoder hidden states at each step	Attends to all encoder positions for every output
Self-Attention	Not used	Self-attention in both encoder and decoder

Terminology Note

A transformer used as a causal language model is called a **decoder-only model** (GPT is an example). This is because it constitutes roughly half of the **encoder-decoder model** for transformers.

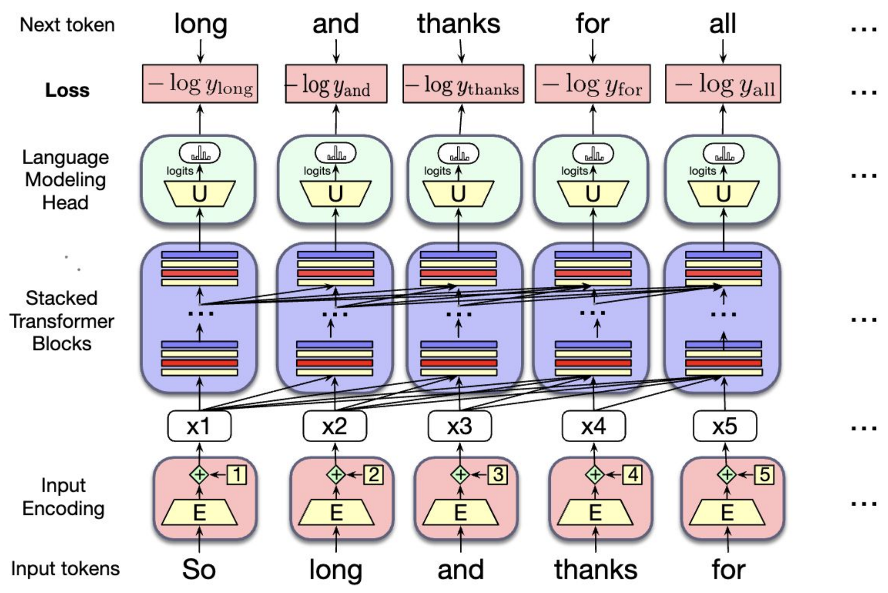
The original introduction of the transformer [Vaswani et al. 2017] had an encoder-decoder architecture (T5 is an example). It was only later that the standard paradigm for causal language model was defined by using only the decoder part of this architecture.

Later, we will also see the paradigm of masked language model which uses an **encoder-only** model (BERT is an example).



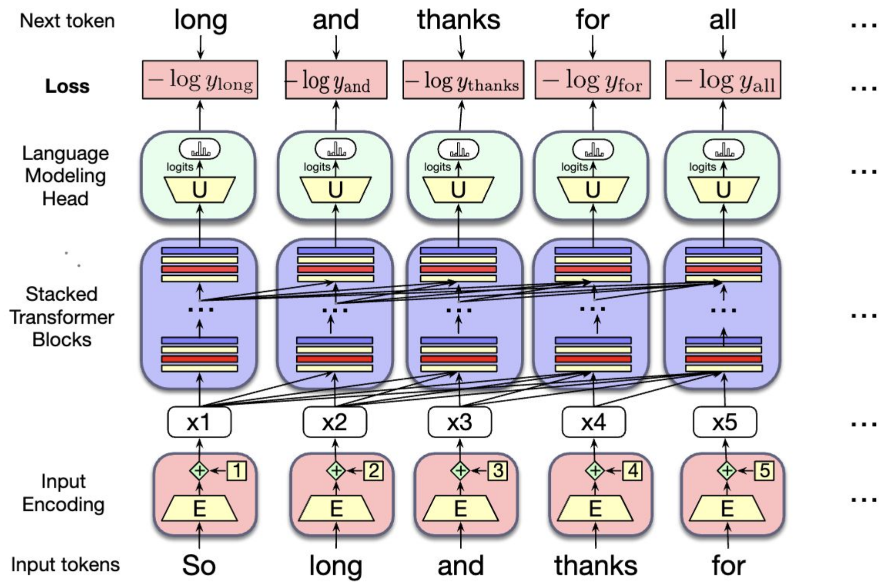
Training and Inference

Transformer Training

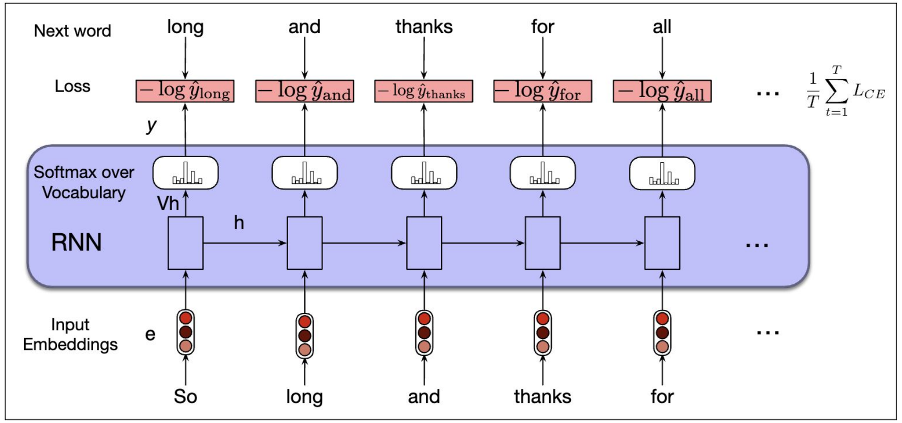


$$= \frac{1}{T}\sum_{t=1}^{T} L_{CE}$$

Compare to RNN Training



$$= \frac{1}{T}\sum_{t=1}^{T} L_{CE}$$



Next word
long
and
thanks
for
all
Loss
$-\log \hat{y}_{\text{long}}$
$-\log \hat{y}_{\text{and}}$
$-\log \hat{y}_{\text{thanks}}$
$-\log \hat{y}_{\text{for}}$
$-\log \hat{y}_{\text{all}}$
...
$\frac{1}{T}\sum_{t=1}^{T} L_{CE}$
y
Softmax over Vocabulary
Vh
h
RNN
Input Embeddings
e
So
long
and
thanks
for

In RNNs, the calculation of the outputs and the losses at each step is inherently serial due to the recurrence in the calculation of the hidden states. In transformers, each training item can be processed in parallel since the output for each element in the sequence is computed separately!

KV Cache

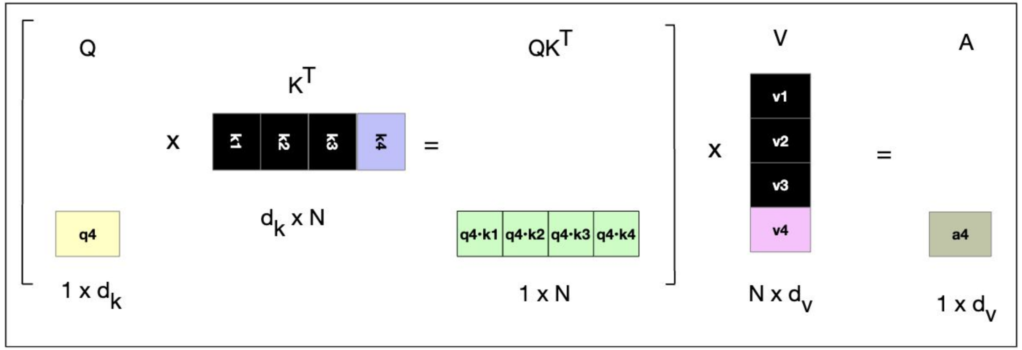
Recall Self-Attention Equation: $$\boxed{Output = softmax\left(QK^T / \sqrt{d_k}\right)V}$$

We can’t do quite the same efficient computation in inference as in training. Why?

Because at inference time, we iteratively generate the next tokens one at a time!

It would be a waste of computation time to recompute the key and value vectors for all the **prior** tokens $\mathbf{x}_{<i}$ since at prior steps we already computed these key and value vectors.

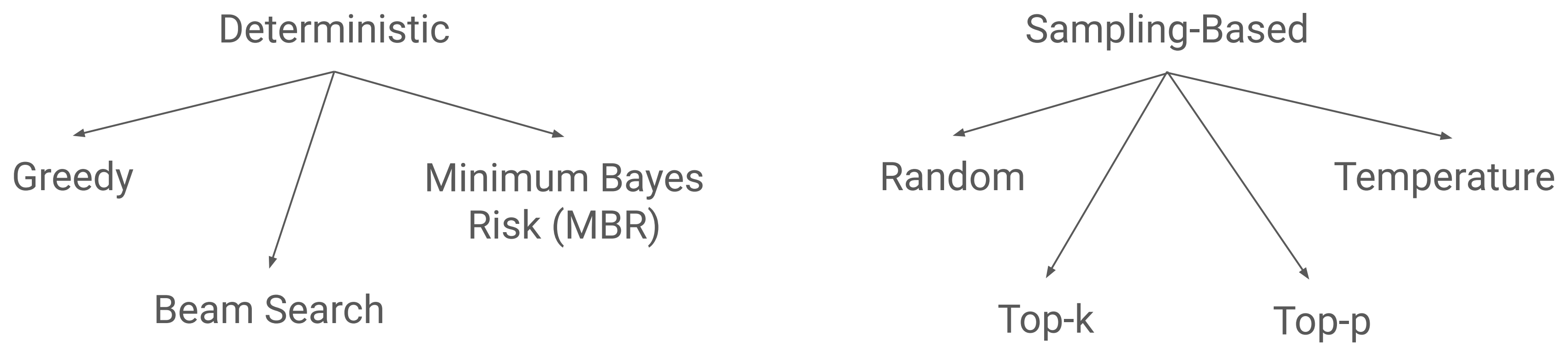
Idea: whenever we compute the key and value vectors, store them in memory in the **KV cache.**



Parts of the attention computation showing, in black, the vectors that can be cached rather than recomputed when computing the attention score for the 4th token.

Which Word to Generate at Each Step?

Broadly two kinds of approaches:



Which Word to Generate at Each Step?

Greedy decoding: Generate the most likely word given the context. That is, compute the probability for every word in the vocabulary and then choose the highest probability word:

$$\hat{w}_t = \text{argmax}_{w \in V} P(w|\mathbf{w}_{<t})$$

A major problem: since the words chosen are (by definition) extremely predictable, the resulting text is generic and repetitive. Greedy decoding is so predictable that it is deterministic!

Sampling methods: Introduce more diversity into the generation by sampling from the model's distribution over words. That is, sample to choose random words according to their probability assigned by the model.

=> We are more likely to generate words that the model thinks have a high probability in the context and less likely to generate words that the model thinks have a low probability.

Sampling Methods

Random sampling:

```
i ← 1
$w_i \sim p(w)$
**while** $w_i$ != EOS
    i ← i + 1
    $w_i \sim p(w_i \mid w_{<i})$
```

Doesn't work well enough! Although it mostly generates sensible, high-probable words, there are many odd, low probability words in the tail of the distribution, and although each one is low probability, all the rare words comprise a large enough portion of the distribution that they get chosen often enough to result in generating weird sentences.

We next introduce sampling methods that avoid generating the very unlikely words.

Each has parameters that enable trading off two important factors in generation: **quality** and **diversity**.

Methods that emphasize the most probable words tend to produce generations that are rated as more accurate, coherent, and factual, but also more boring and repetitive. Methods that give a bit more weight to the middle-probability words tend to be more creative and diverse, but less factual and more likely to be incoherent or low-quality.

Top-k Sampling

Simple generalization of greedy decoding: first truncate the distribution to the top k most likely words, renormalize to produce a legitimate probability distribution, and then randomly sample from within these k words according to their renormalized probabilities.

1. Choose in advance a number of words k.
2. For each word in the vocabulary V, use the model to compute the likelihood of this word given the context $p(w_t \mid \mathbf{w}_{<t})$.
3. Sort the words by their likelihood, and discard those not in the top k most probable words.
4. Renormalize the scores of the k words to be a legitimate probability distribution.
5. Randomly sample a word from within these remaining k most-probable words according to its probability.

k=1 => greedy decoding; k > 1 leads to sometimes select a word which is not necessarily the most probable, but is still probable enough, and whose choice results in generating more diverse but still high-enough-quality text.

Nucleus or Top-p Sampling

Problem with **top-k sampling**: k is fixed but the shape of the probability distribution over words differs in different contexts. If we set k = 10, sometimes the top 10 words will be very likely and include most of the probability mass, but other times the probability distribution will be flatter and the top 10 words will only include a small part of the probability mass.

Nucleus or **top-p sampling**: keep not the top **k** words, but the top **p** percent of the probability mass.

Goal is the same: to truncate the distribution to remove the very unlikely words. But by measuring probability rather than the number of words, the measure is more robust in very different contexts, dynamically increasing and decreasing the pool of word candidates.

Given a distribution $P(w_t \mid w_{<t})$, the top-p vocabulary $V^{(p)}$ is the smallest set of words such that

$$\sum_{w \in V^{(p)}} P(w|\mathbf{w}_{<t}) \geq p$$

Temperature Sampling

Don’t truncate the distribution; instead reshape it. Intuition comes from thermodynamics, where a system at a high temperature is very flexible and can explore many possible states while a system at a lower temperature is likely to explore a subset of lower energy (better) states.

Implement this intuition by simply dividing the logit by a temperature parameter τ before normalizing it by passing it through the softmax: $\mathbf{y} = \text{softmax}(u)$ -> $\mathbf{y} = \text{softmax}(u/\tau)$.

Low-temperature sampling: $\tau \in (0, 1]$. Lower the τ, larger the scores passed to the softmax. Softmax tends to push high values toward 1 and low values toward 0. Thus, when larger numbers are passed to a softmax the result is a distribution with increased probabilities of the most high-probability words and decreased probabilities of the low probability words, making the distribution more greedy. As τ -> 0 the probability of the most likely word -> 1.

High-temperature sampling: $\tau > 1$. Useful in situations where we want to do something quite different and flatten the word probability distribution instead of making it greedy.

Beam Search

Deterministic decoding method that extends greedy decoding and works well in tasks like machine translation, which are very constrained in that we generate a text in one language conditioned on a very specific text in another language.

Recall problem with greedy decoding: $\hat{w}_t = \operatorname{argmax}_{w \in V} P(w|\mathbf{w}_{<t})$

What looks high probability at word t might turn out to have been the wrong choice once we get to word t + 1. Beam search maintains multiple choices until later when we can see which one is best.

Intuition Underlying Beam Search

Model decoding as searching the space of possible generations, represented as a **search tree** whose **branches** represent actions (generating a token), and **nodes** represent states (having generated a particular prefix). We search for the best action sequence, that is, the string with the highest probability.

Example: Greedy search chooses yes followed by yes, instead of the globally most probable sequence ok ok. Vocabulary is V = {yes, ok, EOS}.

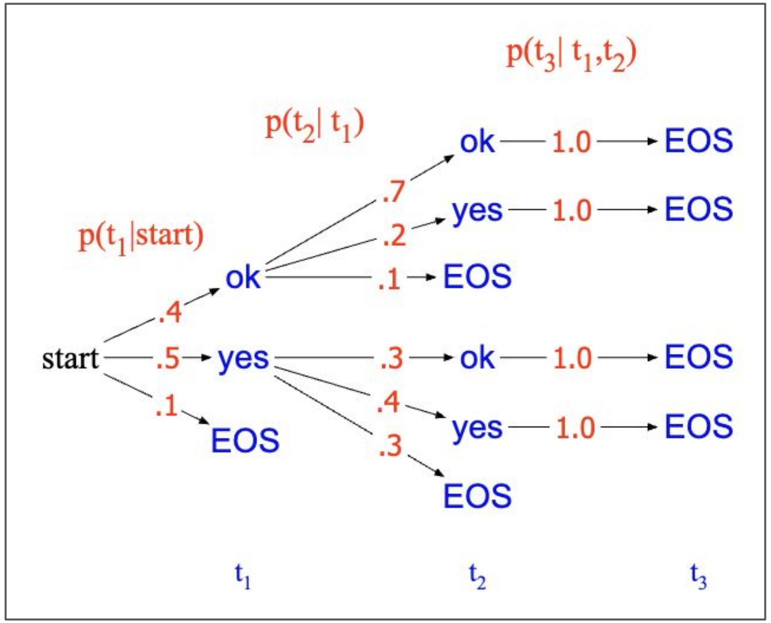


Illustration of Beam Search Algorithm with Width k=2

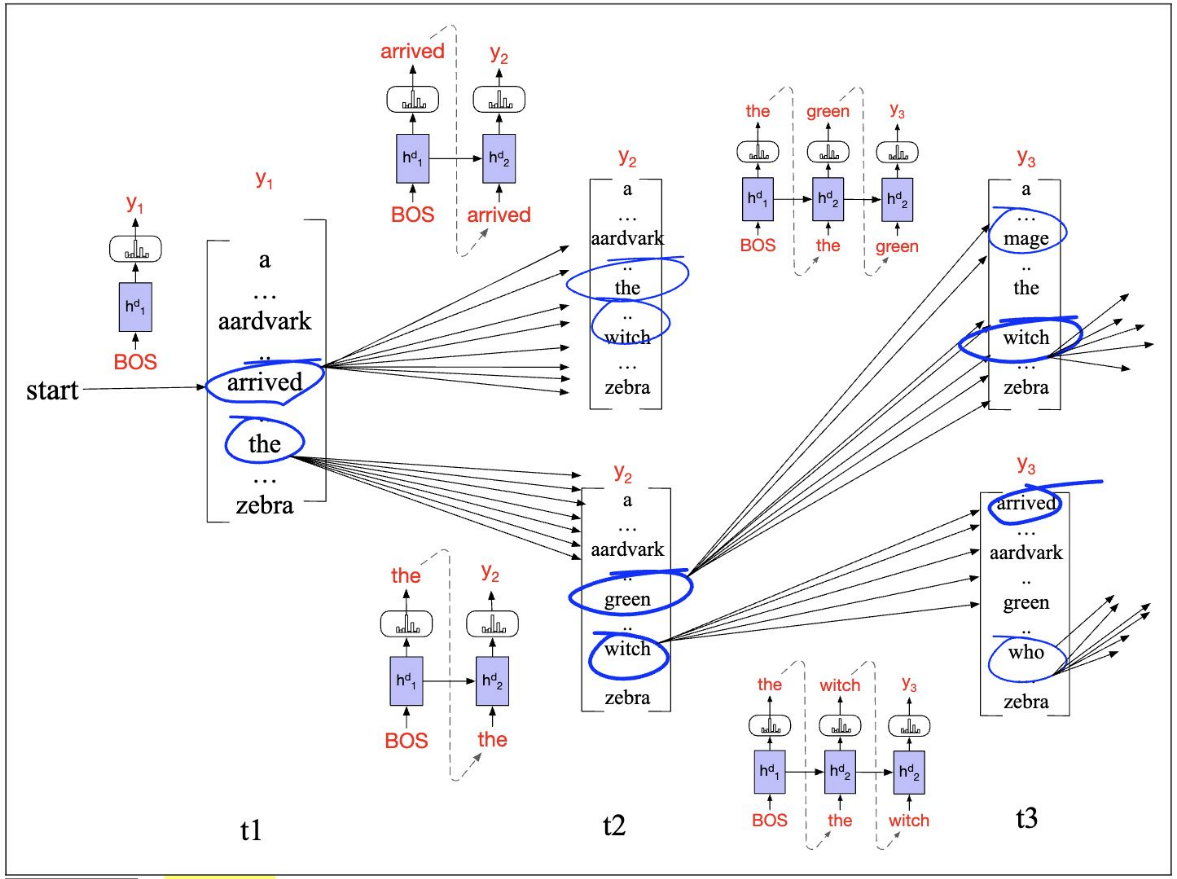
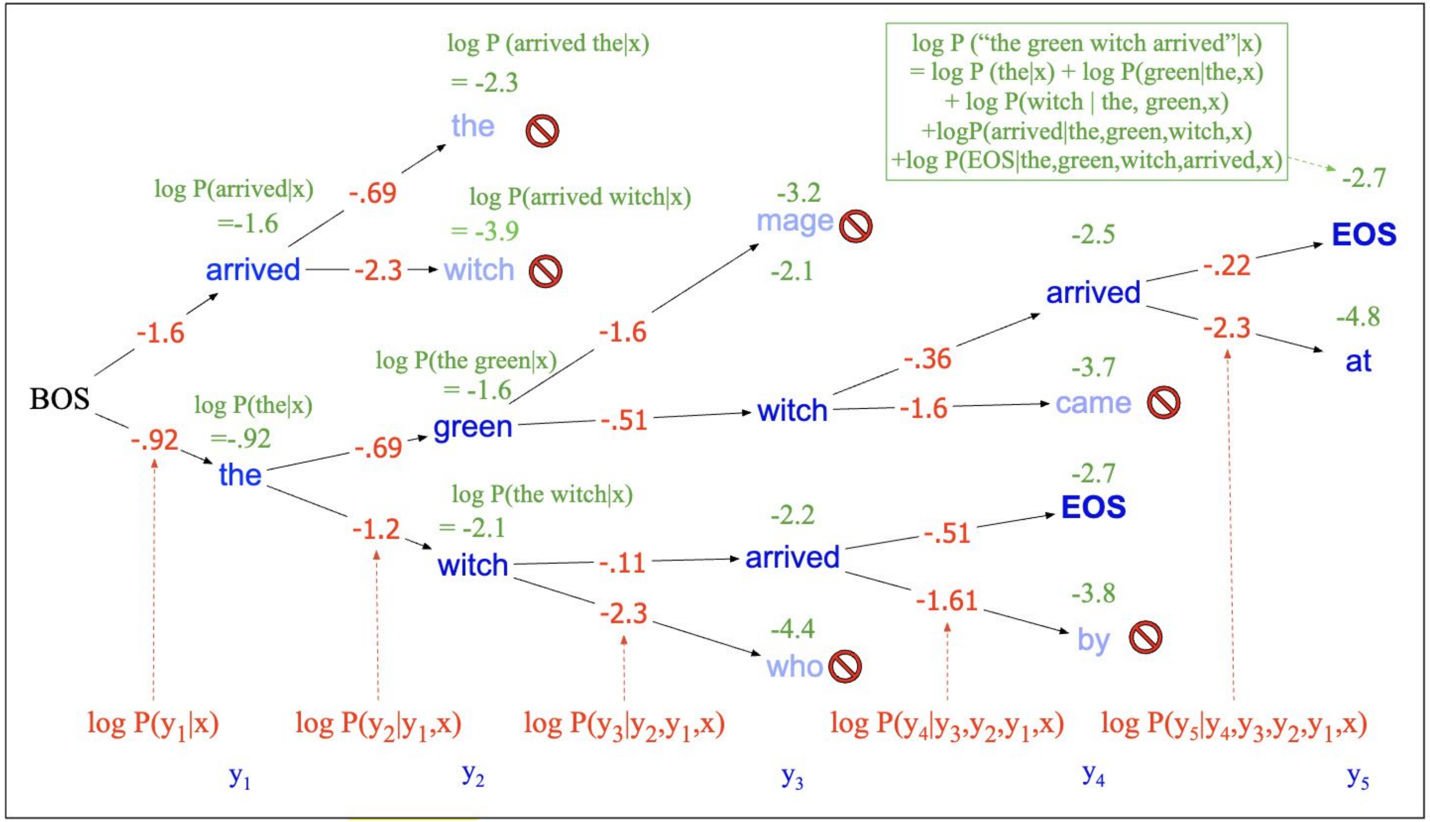


Illustration of Beam Search Algorithm with Width k=2



At each step, extend each of the k best hypotheses incrementally by passing to distinct decoders.

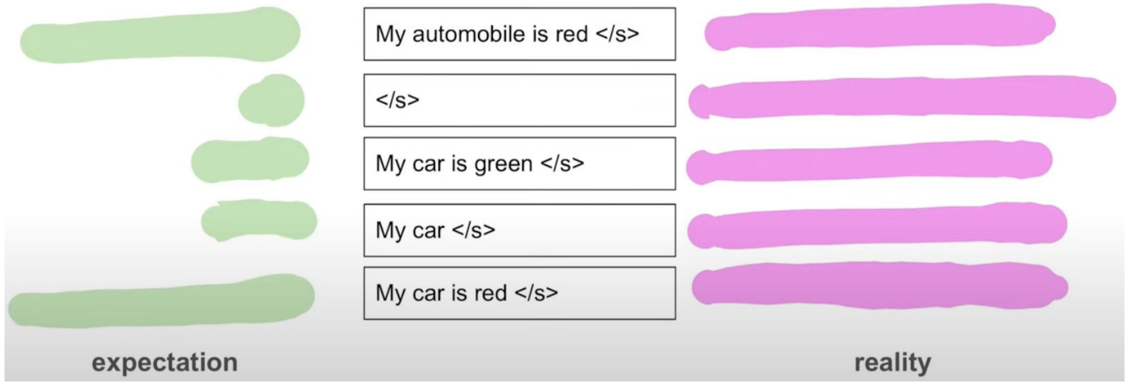
Continue process until an EOS is generated, i.e. a complete candidate output has been found.

Remove the completed hypothesis from the frontier and reduce the size of the beam by 1.

Continue the search until the beam has been reduced to 0. The result will be k hypotheses.

Problem With Beam Search

Spread of Probability Mass:



Minimum Bayes Risk (MBR) decoding to the rescue: instead of trying to choose the translation which is most probable, choose the one that is likely have the least error.

Minimum Bayes Risk (MBR)

Given a set of possible candidate translations *Y*, and some similarity or alignment function *util*, choose the best translation $\hat{y}$ as the one most similar to all the other candidate translations:

$$\hat{y} = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \sum_{c \in \mathcal{Y}} \operatorname{util}(y, c)$$

Various *util* functions can be used, like chrF or BERTscore or BLEU. We can get the set of candidate translations by sampling using one of the basic sampling algorithms. Good results can be obtained with as few as 32 or 64 candidates.

Tends to work better than beam search and other decoding algorithms like temperature sampling. Widely used in machine translation and other generation tasks (e.g. summarization, dialogue, etc.).

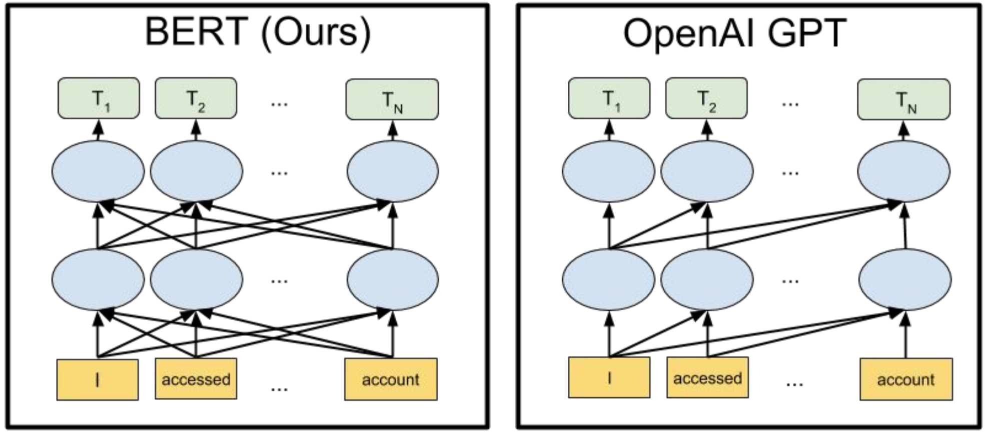
B. Eikema and W. Aziz. Is MAP Decoding All You Need? The Inadequacy of the Mode in Neural Machine Translation. COLING 2020.

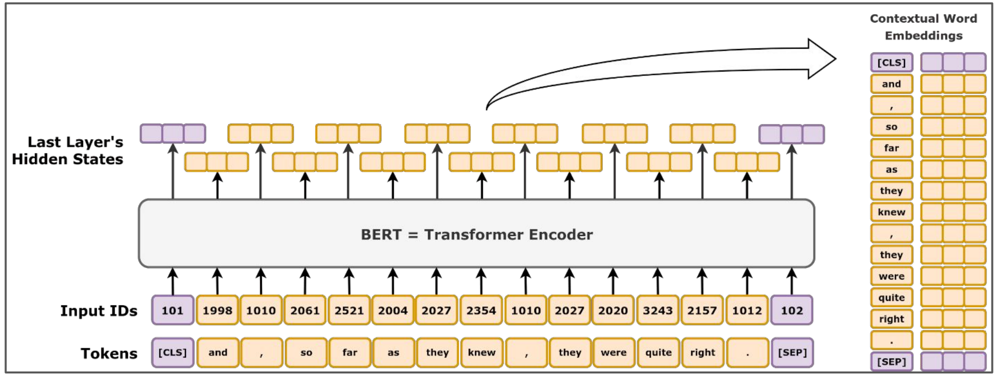
State-of-the-Art Transformer Case Studies

BERT: Bidirectional Encoder Representations from Transformers

Introduced by Google in 2018, it learned embeddings of text for use in downstream tasks. It's major changes are:

- **Segment embeddings** in addition to token embeddings and position embeddings. All are learned!
- **Encoder-only** instead of encoder-decoder
- **Bidirectional** instead of unidirectional
- **Two simultaneous loss functions** with **masked language modeling** and **next sentence prediction**




Contextual Word Embeddings
Last Layer's Hidden States
BERT = Transformer Encoder
Input IDs
101 1998 1010 2061 2521 2004 2027 2354 1010 2027 2020 3243 2157 1012 102
Tokens
[CLS] and , so far as they knew , they were quite right . [SEP]

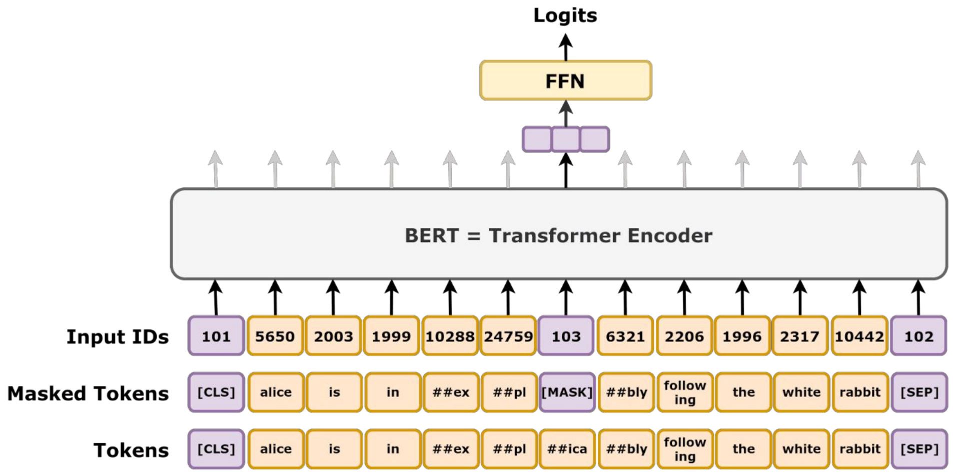
Masked Language Modeling

First, sample 15% of tokens in a sample.

Replace token with:

- [MASK] ~ 80%
- Random word token ~ 10%
- Not replaced ~ 10%

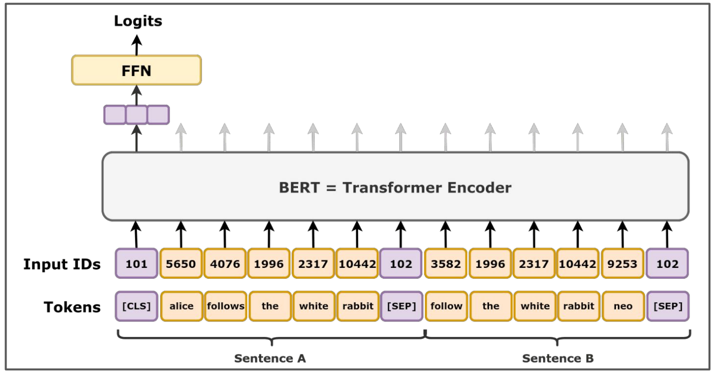
Pass sentence through the encoder and try to predict [MASK] with a simple linear layer + softmax!

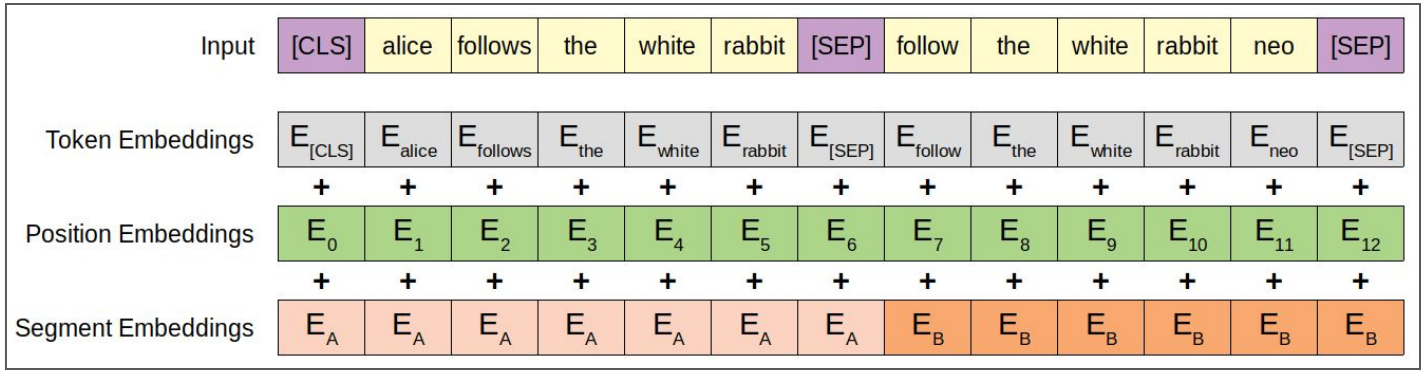


Next Sentence Prediction

Try to determine if one sentence follows another with simple binary classification. All embedding are learned! (unlike original Transformer).

Later works found this to not be useful …



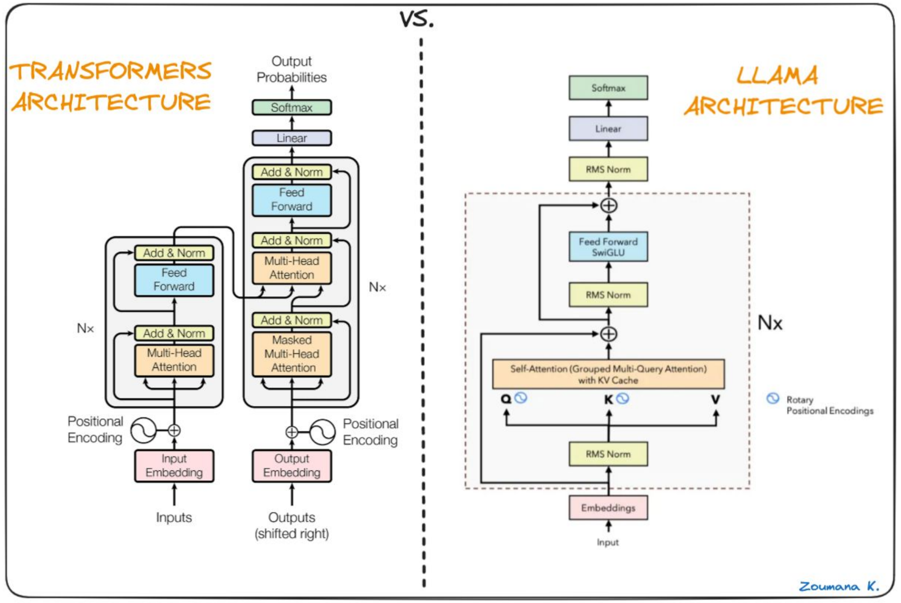


Input
[CLS] alice follows the white rabbit [SEP] follow the white rabbit neo [SEP]
Token Embeddings
$E_{[CLS]}$ E_{alice} $E_{follows}$ E_{the} E_{white} E_{rabbit} $E_{[SEP]}$ E_{follow} E_{the} E_{white} E_{rabbit} E_{neo} $E_{[SEP]}$
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10} E_{11} E_{12}
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B E_B

Llama Family

Autoregressive LLM first released by Meta in Feb 2023. (Several generations since then.) Changes include:

- **Decoder-only** instead of encoder-decoder
- **SwiGLU** activation instead of GeLU
- **Rotary positional embeddings** instead of absolute positional embeddings
- **RMSNorm** instead of LayerNorm



RMS Normalization

In LayerNorm, we re-center (subtracting from mean) and re-scale (divide by std. deviation) across (sequence length, embedding_dim) dimensions.

Zhang et al. propose that **only re-scaling matters**. This saves a small amount of compute by not needed to re-center.

Linear Layer

$$y_i = f\left(\bar{a}_i + b_i\right)$$

LayerNorm

$$\bar{a}_i = \frac{a_i - \mu}{\sigma} g_i,$$

$$\mu = \frac{1}{n}\sum_{i=1}^{n} a_i, \quad \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(a_i - \mu)^2}.$$

RMSNorm

$$\bar{a}_i = \frac{a_i}{\mathrm{RMS}(\mathbf{a})} g_i, \quad \text{where } \mathrm{RMS}(\mathbf{a}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}.$$

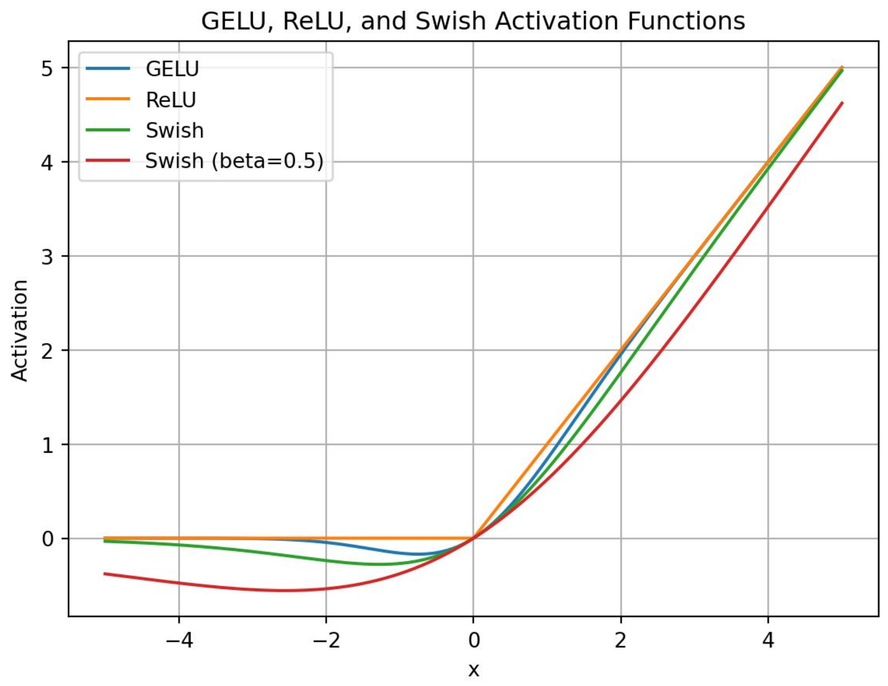
Swish-Gated Linear Unit (SwiGLU)

Swish(x) = sigmoid(β*x), β is hyperparam

GLU(x) = x*sigmoid(Wx+b); W,b is learned

SwiGLU(x) = x * sigmoid(β * x) +
(1 - sigmoid(β * x)) * (Wx + b)

Smoother than ReLU, non-monotonic, allows gating = higher performance!



Rotary Position Embedding (RoPE)

So far, we have seen two kinds of position embeddings: Sinusoidal [Vaswani et al. 2017] and learned (BERT). Where do they fall short?

Instead of adding extra numbers, RoPE rotates embeddings based on their position so that the relative position of tokens can be considered in the attention calculations rather than their absolute positions. **The angles between embedding vectors maintain the same proportional relationship as the distance between tokens in the sequence.**

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W^{(11)}_{\{q,k\}} & W^{(12)}_{\{q,k\}} \\ W^{(21)}_{\{q,k\}} & W^{(22)}_{\{q,k\}} \end{pmatrix} \begin{pmatrix} x^{(1)}_m \\ x^{(2)}_m \end{pmatrix}$$

Drawbacks of Transformers

What would we like to fix about the Transformer?

Quadratic compute in self-attention (today):

Computing all pairs of interactions means our computation grows quadratically with the sequence length!

For recurrent models, it only grew linearly!

Position representations:

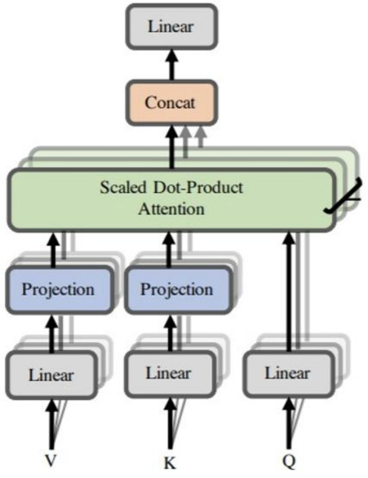
Are simple absolute indices the best we can do to represent position?

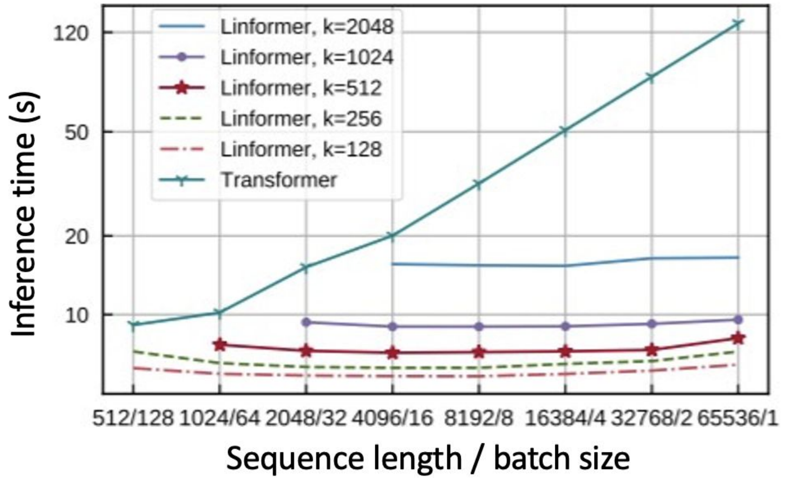
Alternatives: Relative linear position attention [Shaw et al., 2018], Dependency syntax-based position [Wang et al., 2019], Rotary Embeddings [Su et al., 2021]

Work on Improving on Quadratic Self-Attention Cost

- Much recent work has gone into the question, Can we build models like Transformers without paying the $\mathcal{O}(N^2)$ all-pairs self-attention cost?
- For example, **Linformer** [Wang et al., 2020]

Key idea: map the sequence length dimension to a lower-dimensional space for values, keys.

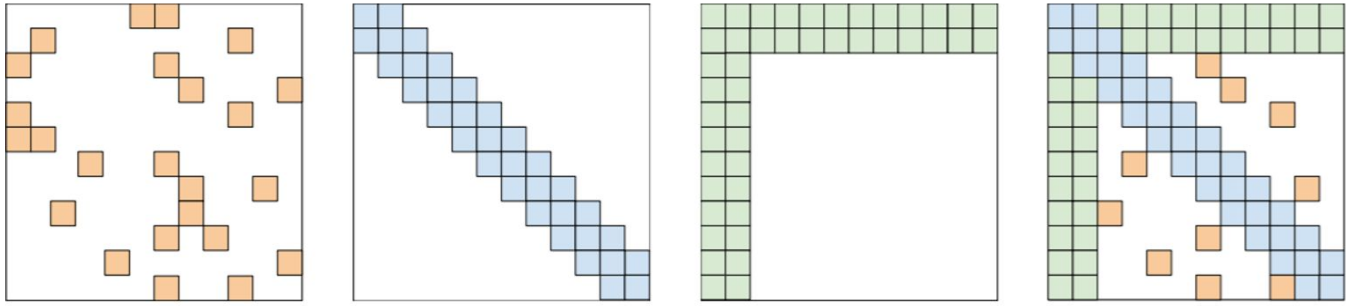




Work on Improving on Quadratic Self-Attention Cost

- Much recent work has gone into the question, Can we build models like Transformers without paying the $\mathcal{O}(N^2)$ all-pairs self-attention cost?
- For example, **BigBird** [Zaheer et al., 2021]

Key idea: replace all-pairs interactions with a family of other interactions, like local windows, looking at everything, and random interactions.



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Up Next …

Sept 23 Lecture: Guest speaker **Yann Dubois**, Ph.D. Candidate, Stanford University.

Topic: LLM Benchmarking and Evaluation

Sept 25 Lecture: Guest speaker **Hanjun Dai**, Staff Research Scientist & Research Manager, Google Brain

Topic: RLHF and LLM Reward Modeling