Transformers and Self-Attention

Joseph Le Roux

20/08/26

Notes

Outline

- Motivation: Beyond Fixed-Context Models
- Attention
- Self-Attention and the Transformer
- The Full Transformer Architecture
- Application: Machine Translation

Notes

1 Motivation: Beyond Fixed-Context Models

Notes

1 The Limits of n-gram Models

Recall: neural *n*-gram LM

$$p(w_i \mid w_{i-n+1}, \ldots, w_{i-1}) = \mathrm{softmax}\big(W_2 \tanh(W_1 [e(w_{i-n+1}); \cdots; e(w_{i-1})] + b_1) + b_2\big)$$

- Context is fixed to $n-1$ words: arbitrary long-range dependencies are invisible.
- Example: *"The trophy that the judges who evaluated the competition liked didn't fit into the box because it was too big."*
- Which antecedent does *it* refer to? A trigram cannot know.

What we need

- Access to *any* position in the input, at any distance.
- Weighted: some positions are more relevant than others.
- ⇒ Attention mechanism

Notes

2 Attention

Notes

2 Intuition: Attention as Soft Look-up

Hard look-up (dictionary)

Given a query q and a set of key-value pairs $\{(k_i, v_i)\}$:

$$\text{lookup}(q) = v_j \quad \text{where } j = \operatorname*{argmax}_i \text{match}(q, k_i)$$

Soft look-up (attention)

Instead of picking *one* value, take a weighted sum of all values:

$$\text{Attn}(q, K, V) = \sum_i \alpha_i v_i, \qquad \alpha_i = \text{softmax}\big(\text{score}(q, k_i)\big)_i$$

- $\alpha_i \geq 0$, $\sum_i \alpha_i = 1$ as a probability distribution over positions.
- The model *learns* what to attend to, based on content.

Notes

2 Scaled Dot-Product Attention

Score function

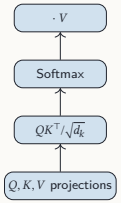
Queries Q, Keys K, Values V are all linear projections of the input:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

- $Q \in \mathbb{R}^{n \times d_k}$, $K \in \mathbb{R}^{m \times d_k}$, $V \in \mathbb{R}^{m \times d_v}$
- $QK^\top \in \mathbb{R}^{n \times m}$: each query attends to all m keys simultaneously.
- Divide by $\sqrt{d_k}$: prevents dot products from growing too large (would push softmax into flat or saturated regions).

Complexity

- $O(n^2 \cdot d)$ per layer — quadratic in sequence length.
- But fully parallelizable: no sequential dependency.
- Compare: RNNs are $O(n \cdot d^2)$ and *sequential*.



Notes

2 Attention in PyTorch

```python
import torch
import torch.nn.functional as F

def scaled_dot_product_attention(Q, K, V, mask=None):
    """
    Q: (batch, heads, seq_q, d_k)
    K: (batch, heads, seq_k, d_k)
    V: (batch, heads, seq_k, d_v)
    """
    d_k = Q.size(-1)
    scores = torch.matmul(Q, K.transpose(-2, -1)) / d_k**0.5  # (B,H,n,m)
    if mask is not None:
        scores = scores.masked_fill(mask == 0, float('-inf'))
    weights = F.softmax(scores, dim=-1)                       # (B,H,n,m)
    return torch.matmul(weights, V), weights                  # (B,H,n,d_v)
```


Notes

3 Self-Attention and the Transformer

Notes

3 Self-Attention

From cross-attention to self-attention

- *Cross-attention*: queries come from one sequence, keys/values from another (e.g., encoder–decoder attention).
- *Self-attention*: queries, keys *and* values all come from the same sequence.

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V, \qquad X \in \mathbb{R}^{n \times d}$$

- Every position can attend to every other position: full-context representation.

Effect

For position i, the output is a weighted sum of all value vectors:

$$z_i = \sum_j \alpha_{ij} v_j, \qquad \alpha_{ij} = \mathrm{softmax}\left(\frac{q_i \cdot k_j}{\sqrt{d_k}}\right)$$

The model can *dynamically* route information from any relevant position.

Notes

3 Multi-Head Attention

Idea: run attention multiple times in parallel

- Use h different learned projections $(W_j^Q, W_j^K, W_j^V)_{j=1}^h$.
- Each *head* attends to the input differently (syntactic, semantic, positional roles…).
- Concatenate and project:

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,\dots,\text{head}_h)W^O$$
$$\text{head}_j = \text{Attention}(QW_j^Q,\ KW_j^K,\ VW_j^V)$$

Hyperparameters

	Original (Vaswani et al.)	Common small model
d_{model}	512	256
Heads h	8	4
$d_k = d_v = d_{\text{model}}/h$	64	64

Notes

3 Positional Encoding

Problem: attention is permutation-invariant

- Self-attention treats the sequence as a *set*: word order is lost!
- We must inject position information explicitly.

Sinusoidal encoding [Vas+17]

Add a fixed vector to each word embedding:

$$\text{PE}(\text{pos},2i) = \sin\left(\frac{\text{pos}}{10000^{2i/d}}\right), \qquad \text{PE}(\text{pos},2i+1) = \cos\left(\frac{\text{pos}}{10000^{2i/d}}\right)$$

- pos: position in the sequence; i: dimension index.
- Different frequencies encode positions at multiple scales.
- Can generalize to sequences longer than those seen in training.
- Alternative: *learned* positional embeddings (BERT, GPT).

Notes

3 Layer Normalization [BKH16]

Transformer and Deep Learning: a Dangerous Cocktail

- We will build architecture with many layers, in particular many FFN/MLP
- *Gradient vanishing/exploding* the gradient is fragile
- Avoid extreme values, find a mechanism to *rescale* data in safe ranges

Layer Normalization

stabilise neural network training.

1. compute mean and variance of input of input x. $\mu = \frac{\sum_i x_i}{|x|}$, $\sigma^2 = \sum_i (x_i - \mu)^2$
2. normalize to $\tilde{x}$, so that mean $= 0$ and variance $= 1$;
3. rescale output dimension by dimension

$$\tilde{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$y = \gamma \cdot \tilde{x} + \beta$$

Notes

3 Transformer Encoder Block

Each encoder layer applies two sub-layers:

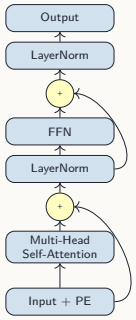
1. Multi-Head Self-Attention

$$H = \text{LayerNorm}(X + \text{MultiHead}(X, X, X))$$

2. Position-wise Feed-Forward Network (FFN)

$$Z = \text{LayerNorm}(H + \text{FFN}(H))$$

$$\text{FFN}(x) = W_2\,\text{ReLU}(W_1 x + b_1) + b_2$$



Key design choices

- *Residual connections* ($+X$): ease gradient flow through many layers.
- *Layer normalization*: stabilizes training.
- Stack $N = 6$ (or 12, 24...) layers.

Notes

3 The Transformer Decoder Block

The decoder has three sub-layers:

1. Masked Multi-Head Self-Attention prevents position i from attending to positions $> i$ (causal masking for autoregressive generation).
2. Cross-Attention queries from the decoder, keys/values from the *encoder output* (allows the decoder to "look at" the source).
3. FFN same as in the encoder.

Each sub-layer is wrapped in residual + LayerNorm, same as the encoder.

Causal masking

Set score$(i, j) = -\infty$ for all $j > i$ before the softmax:

$$\text{mask}_{ij} = \begin{cases} 0 & j \leq i \\ -\infty & j > i \end{cases}$$

This ensures that each output position depends only on *previous* outputs.

Notes

4 The Full Transformer Architecture

Notes

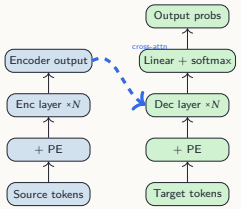
4 Encoder–Decoder Transformer [Vas+17]

Encoder (left stack)

- Input: source tokens + positional encoding.
- N identical encoder layers (self-attention + FFN).
- Output: contextualized representations $\{z_i\}$ of all source tokens.

Decoder (right stack)

- Input: target tokens shifted right (teacher forcing during training) + PE.
- N identical decoder layers (masked self-attention + cross-attention + FFN).
- Output: distribution over target vocabulary at each step.



Notes

5 Application: Machine Translation

Notes

5 Machine Translation with Transformers

Task

Given source sentence $x = x_1 \dots x_m$, generate target sentence $y = y_1 \dots y_n$:

$$p(y \mid x) = \prod_{t=1}^{n} p(y_t \mid y_{<t}, x) = \prod_{t=1}^{n} \mathrm{softmax}(W \cdot \mathrm{dec}(y_{<t}, \mathrm{enc}(x)))[y_t]$$

Training: teacher forcing

- At each decoder step t, feed the *gold* prefix $y_{<t}$ (not the model's own previous output).
- Loss: cross-entropy summed over all positions:

$$\mathscr{L}(\theta) = -\sum_{t=1}^{n} \log p_\theta(y_t \mid y_{<t}, x)$$

- *Label smoothing* (Szegedy et al., 2016): soft targets prevent overconfidence.

Notes

5 Inference: Beam Search

Greedy decoding

At each step, pick the most probable token: $\hat{y}_t = \mathrm{argmax}_w\, p(w \mid \hat{y}_{<t}, x)$.
Problem: a locally optimal choice may lead to a globally suboptimal sequence.

Beam search

Maintain B (beam width) partial hypotheses in parallel:

1. Expand each hypothesis by all vocabulary tokens.
2. Score by log-probability sum: $\sum_t \log p(y_t \mid y_{<t}, x)$.
3. Keep the top-B hypotheses.
4. Repeat until all beams produce `EOS`.
5. $B = 1$: greedy. $B \longrightarrow \infty$: exact search (intractable).
6. Typical: $B = 4 - -10$.
7. Length penalty: divide score by $|y|^\alpha$ to avoid short-sentence bias.

Notes

5 Evaluation: BLEU Score

BLEU (Bilingual Evaluation Understudy)

- Compare machine output to one or more *reference* translations.
- Measure n-gram *precision* for $n = 1, 2, 3, 4$ (modified: clip by reference counts).

$$\text{BLEU} = \underbrace{\exp\left(\min\left(1 - \frac{r}{c}, 0\right)\right)}_{\text{brevity penalty}} \times \exp\left(\sum_{n=1}^{4} \frac{1}{4} \log p_n\right)$$

- c: candidate length; r: reference length.

Limitations

- Does not capture meaning, fluency, or paraphrase.
- Scores are corpus-level and not easily interpretable in isolation.
- Modern alternative: COMET (neural MT metric, trained on human judgements).

Notes

5 Transformer in PyTorch

```python
import torch
import torch.nn as nn

# Full encoder-decoder Transformer
transformer = nn.Transformer(
    d_model=512,
    nhead=8,
    num_encoder_layers=6,
    num_decoder_layers=6,
    dim_feedforward=2048,
    dropout=0.1,
)

# src: (src_len, batch), tgt: (tgt_len, batch) - word indices -> embeddings needed
# src_key_padding_mask: (batch, src_len) - True for padding positions
output = transformer(src_emb, tgt_emb,
                     tgt_mask=nn.Transformer.generate_square_subsequent_mask(tgt_len),
                     src_key_padding_mask=src_padding_mask)
# output: (tgt_len, batch, d_model) -> project to vocab with nn.Linear
```


Notes

5 Summary

What we covered

1. Attention: soft, content-based look-up over all positions.
2. Scaled Dot-Product Attention: $\mathrm{softmax}(QK^\top/\sqrt{d_k})V$.
3. Self-Attention: queries, keys, values from the same sequence.
4. Multi-Head Attention: h attention heads in parallel.
5. Positional Encoding: sinusoidal injection of order information.
6. Transformer Encoder/Decoder: MHA + FFN + residuals + LayerNorm, stacked N times.
7. MT Application*: teacher forcing (training), beam search (inference), BLEU (evaluation).

***Next* lecture**

- What if we *pre-train* a Transformer on a huge corpus, then fine-tune on a specific task?
- $\Rightarrow$ BERT, GPT, and the pre-trained language model paradigm

Notes

6 References

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016.

[Vas+17] Ashish Vaswani et al. "Attention is all you need." In: *Advances in neural information processing systems* 30 (2017).

Notes