Transformers – Part 2

Michael Wollowski

With assistance from Claude Sonnet 4.5

1

Transformer Blocks

- The self-attention calculation lies at the core of what is called a **transformer block**.
- In addition to the self-attention layer, it includes feedforward layers, residual connections, and normalizing layers.

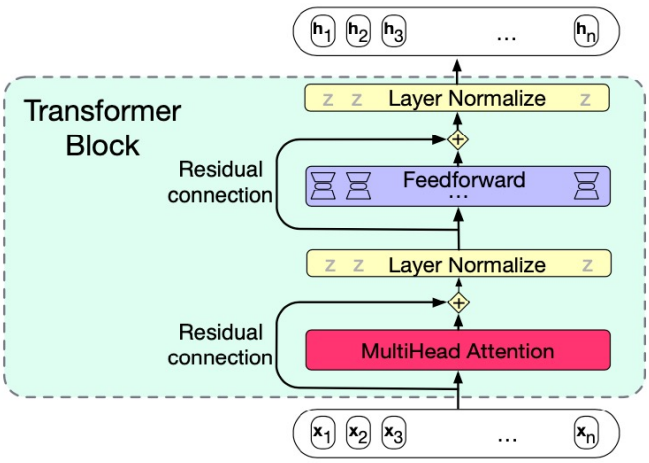
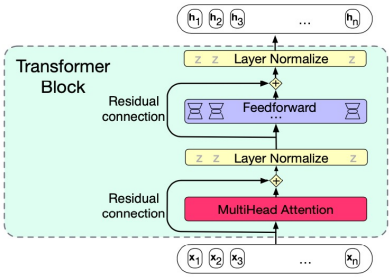


Image source: TBA

2

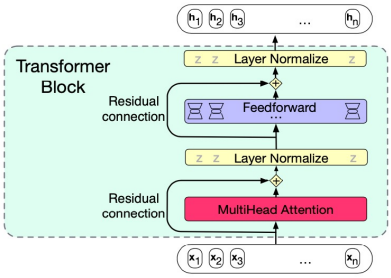
Transformer Blocks



- **Feedforward layer**: It contains *N* position-wise networks, one at each position.

3

Transformer Blocks



- **Residual connections**: They pass information from a lower layer to a higher layer without going through the intermediate layer.
- **Layer normalization (Layer norm)**.
  - Summed vectors are normalized.
  - It is used to improve training performance in deep neural networks.
  - It keeps the values of a hidden layer in a range that facilitates gradient-based training.

4

Residual Stream View of the Transformer

- A better view of a transformer is to look at the residual stream.
- The feedforward layers and the multi-head attention layers are added to a stream.
- The Multi Head Attention layer reads information from the other residual streams.
- The feedforward layer processes just one token.

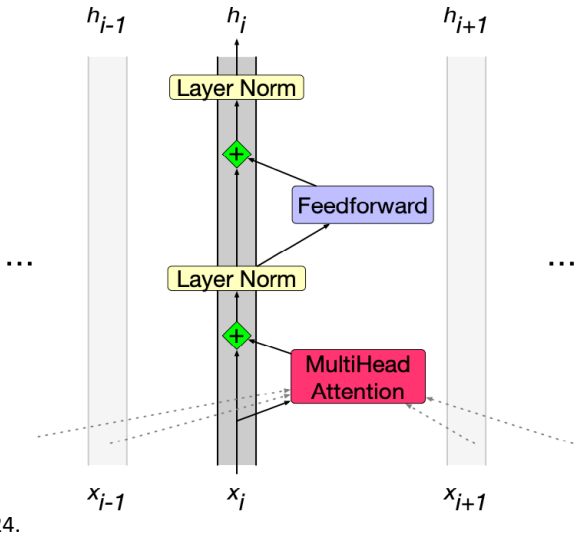


Image source: Chapter 9, Jurafsky and Martin, Aug. 20, 2024.

5

Moving Information

- The attention head can move information from token A's residual stream into token B's residual stream.

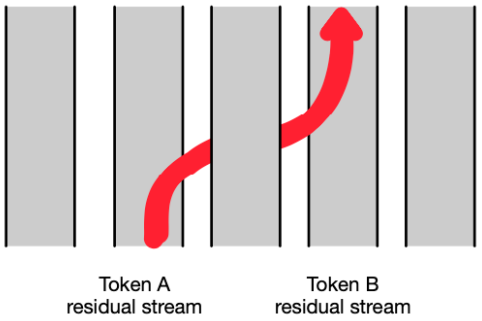


Image source: Chapter 9, Jurafsky and Martin, Aug. 20, 2024.

6

Transformer Block: Layer Normalization

- Typical: z-score $$z = \frac{x - \mu}{\sigma}$$
- X = observed value
- μ = mean of sample
- σ = standard deviation of the sample

7

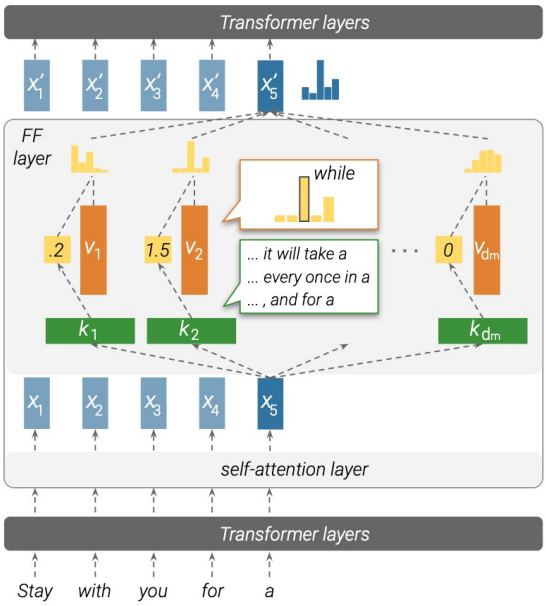
Transformer Block: Layer Normalization

- Let x = [2.0, 4.0, 6.0, 8.0]
- Mean μ = (2 + 4 + 6 + 8) / 4 = 5.0
- Variance σ^2 = [(2-5)² + (4-5)² + (6-5)² + (8-5)²] / 4 =
  [9 + 1 + 1 + 9] / 4 = 5.0
- Let z = (x - μ) / √(σ^2 + ε), where ε = 1e-5 (small constant for numerical stability)
  = [(2-5)/√5, (4-5)/√5, (6-5)/√5, (8-5)/√5] ≈
  [-1.34, -0.45, 0.45, 1.34]
- Mean: 0. Variance: 1

8

Feedforward layers

- The FF layers are not a classifier as in FFnets or CNNs.
- Instead, they perform key-value look-up.



Source: Transformer Feed-Forward Layers are Key-Value Memories. Geva et al.

9

Memory in LSTMs

- They are similar in spirit to the context vector in LSTMs.
- **Long-term memory storage:**
  - **LSTM cell state (c_t)**: Explicitly maintained memory that accumulates information via the input gate
- **Memory retrieval/access:**
  - **LSTM output gate**: Controls what gets read from cell state and exposed to the output

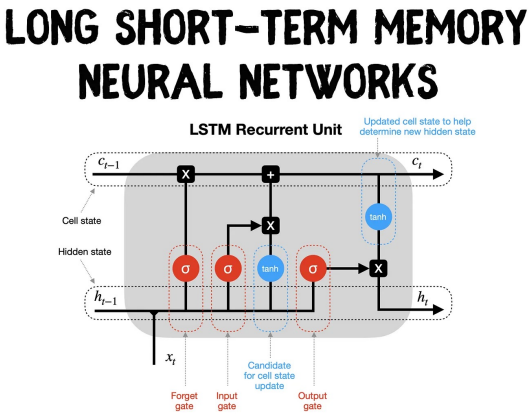
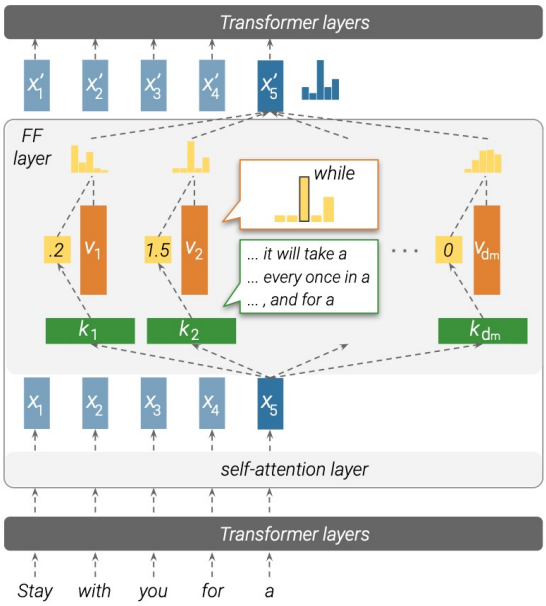


Image source: https://towardsdatascience.com/lstm-recurrent-neural-networks-how-to-teach-a-network-to-remember-the-past-55e54c2ff22e

10

Memory in Transformers

- **Long-term memory storage:**
  - **Transformer FFN layers**: Implicitly store learned knowledge in their parameters—this is where the model's "long-term memory" of patterns, facts, and transformations lives
- **Memory retrieval/access:**
  - **Transformer attention**: Controls what information gets retrieved and aggregated from across the sequence (much more flexible and powerful)



Source: Transformer Feed-Forward Layers are Key-Value Memories. Geva et al.

11

Memory in LSTMs and Transformers

- In **both** architectures, individual neurons can specialize to track specific features like sentiment:
- **LSTM**: A single dimension in the cell state vector can learn to track sentiment
- **Transformer**: A single neuron in an FFN layer can also learn to track sentiment (this has been demonstrated empirically with neuron activation studies)

12

Memory in LSTMs and Transformers

- The real difference is architectural:
- **LSTM**: Has a designated memory vector (cell state) that's separately maintained and updated
- **Transformer**: Has the residual stream where all information flows through, getting iteratively refined by attention (gathering info) and FFN layers (transforming it)

13

Memory in LSTMs and Transformers

- LSTM cell state ↔ Transformer residual stream
  - Both are the primary information highways
  - Both get iteratively refined (LSTM through time steps, Transformer through layers)
  - Both accumulate and transform information progressively
- The key operations on each:
  - **LSTM**: Gates modulate what's added/kept/read from cell state
  - **Transformer**: Attention + FFN add transformations to the residual stream
- The scale difference is massive:
  - LSTM hidden states: typically hundreds to ~2000 dimensions
  - Transformer residual streams: 4096, 12288, or even larger dimensions
  - Orders of magnitude more parameters in the FFN layers
- That scale is what gives Transformers their power—vastly more capacity to represent and refine information, plus the flexibility of attention to route it.

14

Information Flow through Transformer Architecture

Input embeddings
↓
[Layer 1: Head 1 + Head 2 + ... + Head 8 → add to stream]
↓
[Layer 1: MLP → add to stream]
↓
[Layer 2: Head 1 + Head 2 + ... + Head 8 → add to stream]
↓
[Layer 2: MLP → add to stream]
↓
... (many layers)
↓
Final contextualized embedding

15

Memory in Transformers

- Recent research suggests MLPs act as **key-value memories**.
- They store factual associations like:
  - "Eiffel Tower" → [Paris, France, landmark, tall, iron, 1889, ...]
  - "photosynthesis" → [plants, chlorophyll, sunlight, CO_2, oxygen, ...]
- When the attention mechanism has assembled ”Eiffel tower" in the residual stream, the MLP can "look it up" and add rich semantic information.

16

Memory in Transformers

- Unlike attention (which mixes tokens), MLPs process each token position **independently**.
- They can:
  - Amplify or suppress certain features
  - Perform pattern matching ("if I see these features together, activate this concept")
  - Add semantic enrichment based on what attention has already assembled
- **The division of labor:**
  - **Attention heads:** "Let me gather relevant context from other positions"
    - "Oh, 'bank' appears near 'river' → probably the geographic meaning"
    - "This 'bank' is near 'deposit' → probably the financial meaning"
  - **MLPs:** "Now that I know the context, let me add my knowledge"
    - Takes the contextualized "bank (financial)" representation
    - Adds: [money, institution, savings, loans, interest, checking account, ...]

17

Memory in Transformers

- **Factual knowledge** is primarily stored in MLP weights
- **Linguistic patterns** and attention patterns are in attention weights
- **Attention** is like your working memory - gathering and organizing relevant information
- **MLPs** are like your long-term memory - vast stores of facts and patterns that get activated based on what's in working memory.
- In GPT-style models with billions of parameters, the majority of those parameters are in the MLPs - that's where "most of the model" actually lives!

18

A Standing Example: Attention

- Let's begin with word embeddings with positional info.
- Not much semantic richness yet.
  - "The": [0.2, 0.1, 0.0, 0.3, …]
  - "Eiffel": [0.5, 0.3, 0.1, 0.0, …]
  - "Tower": [0.4, 0.2, 0.2, 0.1, …]
  - "is": [0.1, 0.0, 0.3, 0.2, …]
  - "in": [0.0, 0.2, 0.1, 0.1, …]
  - "Paris": [0.6, 0.1, 0.0, 0.4, ...]
- Layer 1: Multi-Head Attention
- Head 1: Adjacency detector might compute:
  - "Eiffel" attends strongly to "Tower" (they're adjacent)
  - Adds a vector to "Eiffel" that says "I'm followed by Tower", say: [+0.1, +0.2, 0.0, +0.1, ...]
- Head 2: Compound noun detector:
  - Recognizes "Eiffel Tower" is a compound term
  - Adds feature to both tokens indicating they form a unit
  - For "Tower": [+0.2, 0.0, +0.3, 0.0, ...]
- The residual stream for "Tower" now contains:
  - Original: [0.4, 0.2, 0.2, 0.1, ...]
  - + Head 1: [+0.1, +0.2, 0.0, +0.1, ...]
  - + Head 2: [+0.2, 0.0, +0.3, 0.0, ...]
  - = [0.7, 0.4, 0.5, 0.2, ...]

19

A Standing Example: Key Layer of MLP

- The MLP has learned patterns during training.
- Think of it as having memorized rules like:
  - IF I see pattern [high value in dim 0, high in dim 2, nearby "Eiffel"]
  - THEN add [landmark features + Paris features + France features + tourism features + architecture features]
- Each neuron in the first layer of the MLP is like a "key" - it activates when it recognizes a specific pattern.
- Continuing with "Tower" in the context we've built:
- Suppose neuron 2847 has weights that make it fire strongly for "Eiffel Tower" specifically:
  - It's checking: "Do I see the pattern that indicates Eiffel Tower?"
  - Input: [0.7, 0.4, 0.5, 0.2, ...]
  - This neuron's weights: [0.8, 0.1, 0.9, -0.1, ...] (learned during training)
  - Dot product: 0.7×0.8 + 0.4×0.1 + 0.5×0.9 + ... = **1.2** (high activation!)
- Suppose neuron 5432, with different pattern:
  - Checking for "generic tower"
  - Its weights: [0.1, 0.8, -0.2, 0.3, ...]
  - Dot product: 0.7×0.1 + 0.4×0.8 + ... = **0.4** (moderate activation)

20

A Standing Example: Value layer of MLP

- The second layer of MLP (Key-Value Memory)
- The “value” layer
- Output dimension 42 (let's say "Paris association"):
  = 1.2 × (weight from neuron 2847) +
  0.4 × (weight from neuron 5432) + ...
  = 1.2 × 0.9 + 0.4 × 0.1 + ...
  = 1.12 (strong "Paris" feature added!)
- Output dimension 73 ("landmark"):
  = 1.2 × 0.8 + 0.4 × 0.7 + ...
  = 1.24 (strong "landmark" feature!)
- Output dimension 158 ("French culture"):
  = 1.2 × 0.6 + 0.4 × 0.05 + ...
  = 0.74 (moderate association)
- After MLP, the residual stream for "Tower" becomes:
  Before MLP: [0.7, 0.4, 0.5, 0.2, ...]
  + MLP output: [0.1, 0.05, 1.12, 0.3, ...]
  = [0.8, 0.45, 1.62, 0.5, ...]
- The stream now contains:
  - Original "Tower" embedding
  - From Attention:
    - Information that it's adjacent to "Eiffel"
    - Recognition it's a compound noun
  - From MLP:
    - Paris association (dim 42: 1.12)
    - Landmark feature (dim 73: 1.24)
    - French culture (dim 158: 0.74)

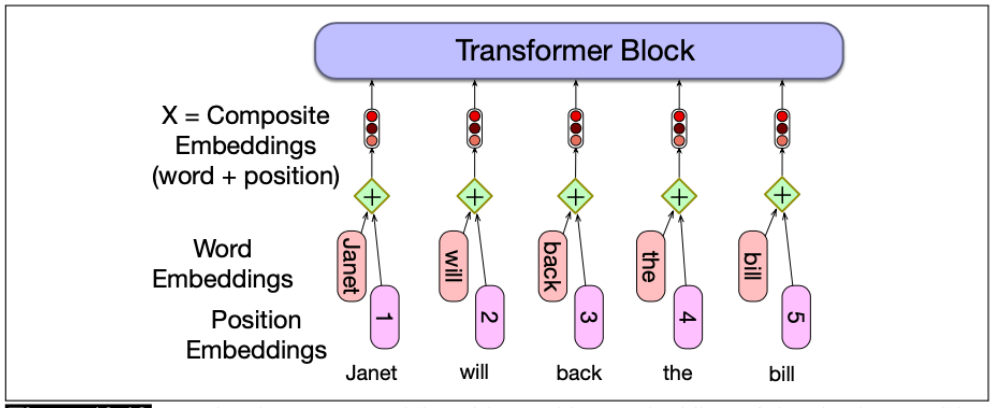
21

Positions

- While the order in which the N tokens are inserted represents word order, this is not sufficient.
- Recall that attention heads can move tokens around.
- We wish to associate with each word the order in which it appeared in the text.
- We will create position embeddings.
- For example, just as we have an embedding for the word *fish*, we will have an embedding for position 3.

22

Positions



Combining word embeddings with positions.

Image source: Chapter 9, Jurafsky and Martin, Aug. 20, 2024.

23

Positions

- **The classic approach** from the original "Attention is All You Need" paper uses sinusoidal functions with different frequencies for each dimension:
  - PE(pos, 2i) = sin(pos / 10000^(2i/d))
  - PE(pos, 2i+1) = cos(pos / 10000^(2i/d))
- This gives each position a unique "signature" that the model can learn to interpret.

24

Example of Positional Encoding

- **Setup:**
- d_model = 4 (embedding dimension - keeping it small for clarity)
- We have 3 tokens in a sequence: ["The", "cat", "sat"]
- Positions: 0, 1, 2

- **The positional encoding formulas:**
- PE(pos, 2i) = sin(pos / 10000^(2i/d_model))
- PE(pos, 2i+1) = cos(pos / 10000^(2i/d_model))
- Where i is the dimension index (0, 1, 2, 3 in our case).

- **Position 1:**
- For i=0:
  - PE(1, 0) = sin(1 / 10000^0) = sin(1) ≈ **0.841**
  - PE(1, 1) = cos(1 / 10000^0) = cos(1) ≈ **0.540**
- For i=1:
  - PE(1, 2) = sin(1 / 10000^0.5) = sin(1/100) ≈ **0.010**
  - PE(1, 3) = cos(1 / 10000^0.5) = cos(1/100) ≈ **1.000**
- Position 1 encoding: **[0.841, 0.540, 0.010, 1.000]**
- Say "cat" has word embedding: [0.5, 0.3, -0.2, 0.8]
- The actual input to the transformer becomes: [0.5, 0.3, -0.2, 0.8] + [0.841, 0.540, 0.010, 1.000] = **[1.341, 0.840, -0.190, 1.800]**

25

Alternatives: Learned Positional Embeddings

- Trainable embedding vectors
- One for each position up to a maximum sequence length.
- The model learns optimal position representations during training.
- They're typically initialized randomly.

26

Alternatives: Relative Positional Embeddings

- Encode the distance between tokens rather than absolute positions
- Distances beyond a maximum threshold are often clipped (e.g., all distances > 128 use the same embedding).
- They're initialized randomly

27

The Language Modelling Head

- Language models are word predictors.
- For example, if the preceding context is “Thanks for all the” and we want to know how likely the next word is “fish” we would compute:

  $P(fish|Thanks\ for\ all\ the)$

- The language modeling head takes the output of the final transformer layer.
- It looks at the last token N.
- Predicts the upcoming word at position N + 1.

28

The Language Modelling Head

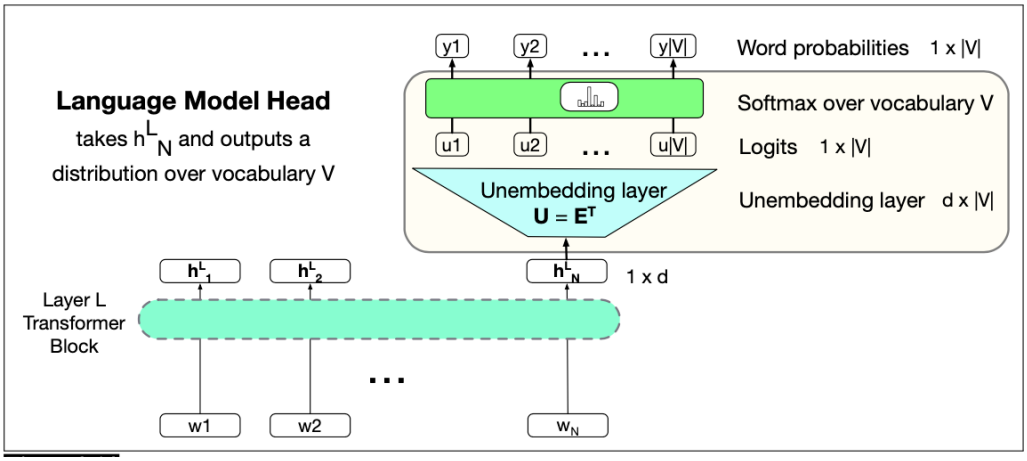


Image source: Chapter 9, Jurafsky and Martin, Aug. 20, 2024.

29

The Language Modelling Head

The unembedding layer takes as input a vector of size d.

It needs to produce an output of size |V|.

Conveniently, the input embeddings are of size |V|

Let's take the transpose of the input embeddings matrix.

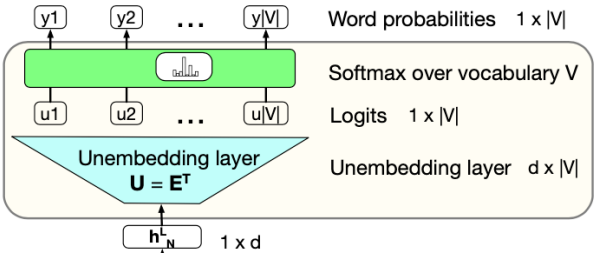


Image source: Chapter 9, Jurafsky and Martin, Aug. 20, 2024.

30

Encoder Architecture

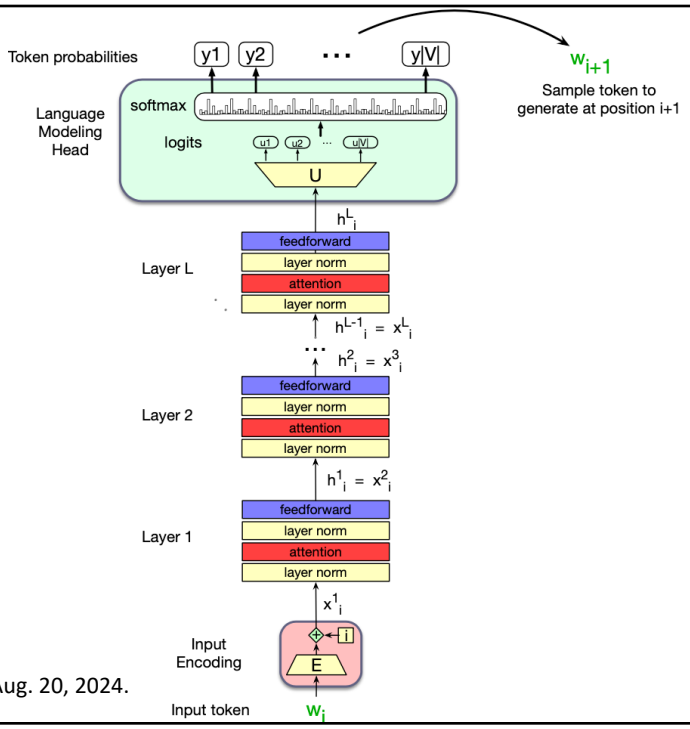


Image source: Chapter 9, Jurafsky and Martin, Aug. 20, 2024.

31