Generative and Agentic AI in Practice

DS 246 (1:2)

Generative and Agentic AI in Practice: DS 246 (1:2)

Prof. Sashikumaar Ganesan

Lecture Topics

Phase 1: Foundations (Weeks 1-8, leading to Midterm)

- Week 3(Aug 20): LLM Overview
  - Topics: (Lifecycle, Tokenization, Embedding, Positional Encoding, stop words, special tokens); Architectures (Encoder-Decoder, encoder-only, decoder-only); Attention Mechanisms (Self-Attention, Multi-Head Attention, Masked Multi-Head Attention, Cross-Attention); Normalization and Connections (Layer Normalization, Batch Normalization, Residual Connections); Advanced techniques including Efficient Attention and Grouped Query Attention.

Foundations

Foundations

LLM Lifecycle Overview

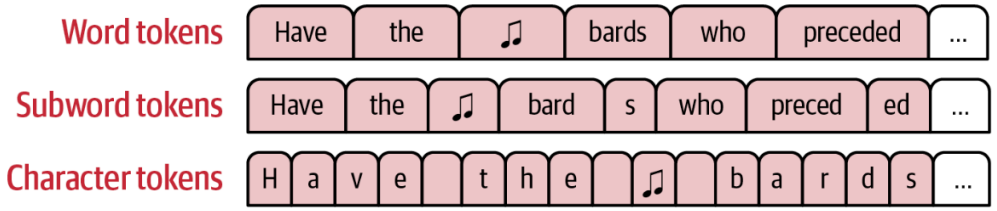
- Pre-training → Fine-tuning → Inference pipeline
- Data collection and preprocessing
- Model deployment and serving
- Practical example: GPT training stages

Foundations: Step I : Tokenization

The process of splitting up the sentences into individua words or subwords (tokens)

Subword Tokenization

- A middle ground between word and character-based tokenization strategies
- Frequently used words should not be split into smaller subwords
- Rare words should be decomposed into meaningful subwords



Vocabulary

- A collection of unique tokens
- For E.g. GPT-2 size is 50,257

Foundations: Step I : Tokenization

Popular algorithms

- Byte-Pair Encoding (BPE)
  - widely used by GPT, StartCoder2 (code gen), Galactica (S
- WordPiece (used by BERT), Unigram language modeling toke

For example, the word "unigram" could potentially be tokenized as:

- `["uni", "gram"]`
- `["u", "ni", "gram"]`
- `["un", "i", "gram"]`
- `["u", "n", "i", "g", "r", "a", "m"]`

The Tokenization Process

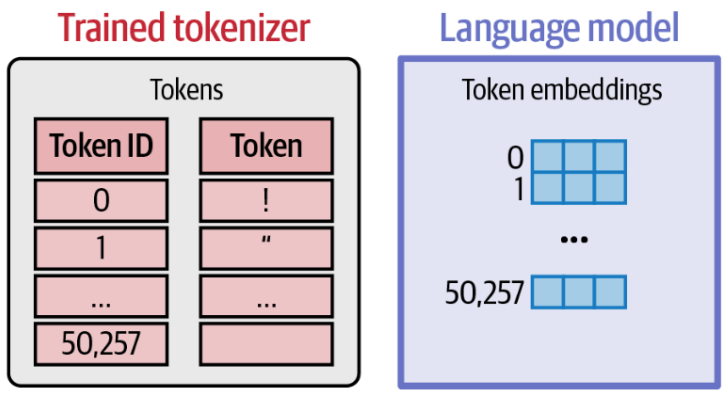
- **Training:**
  - Aims to create an optimal vocabulary of subwords from a large text corpus.
  - **Initial Vocabulary Creation:** The process begins by creating a large initial vocabulary. This can include all characters, common substrings, or even the result of a preliminary tokenization with another algorithm like BPE.
  - **Loss Calculation:** The algorithm then calculates a "loss" for the current vocabulary. This loss is typically the negative log-likelihood of the training corpus given the vocabulary. In simpler terms, it measures how well the current set of tokens can represent the text.
  - **Token Removal:** The model then iteratively removes a certain percentage (e.g., 10-20%) of tokens from the vocabulary. The tokens that are removed are those that cause the smallest increase in the overall loss. This means the algorithm prunes the tokens that are least important for representing the corpus.
  - **Repetition:** These steps of loss calculation and token removal are repeated until the vocabulary reaches a predefined target size.

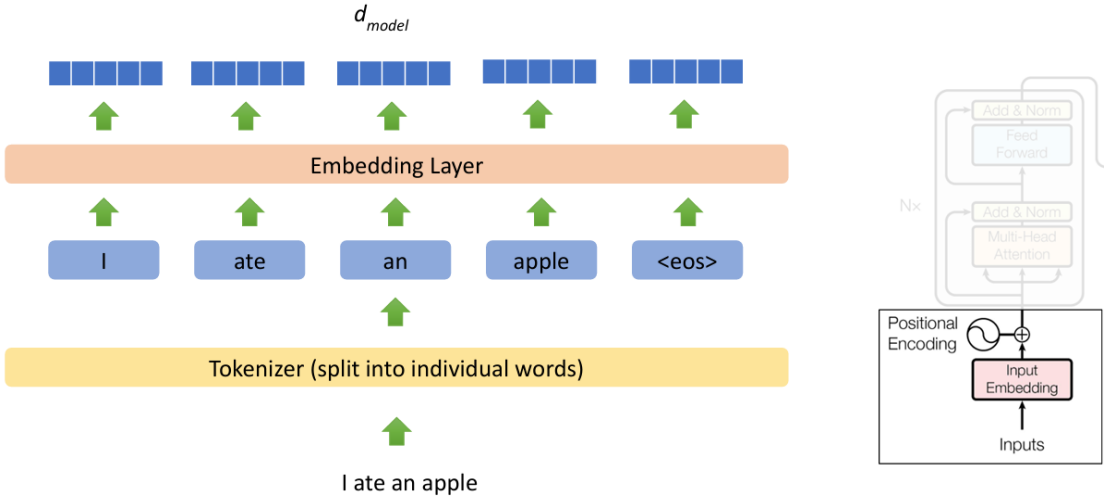
Foundations: Step I : Tokenization

Feature	Unigram Tokenization	Byte-Pair Encoding (BPE)	WordPiece
Approach	Starts with a large vocabulary and removes less likely tokens.	Starts with individual characters and merges the most frequent pairs.	Similar to BPE, but merges pairs that maximize the likelihood of the training data.
Nature	Probabilistic; can produce multiple segmentations with different probabilities.	Deterministic; always produces the same segmentation for a given word.	Deterministic; always produces the same segmentation for a given word.
Flexibility	More flexible and can better model the morphology of a language.	Less flexible as it is based on frequency counts.	Less flexible than Unigram but considers likelihood.

Foundations: Step II - **Embedding**

- In any language processing task, we start with raw text data. However, transformers work only with numerical data.
- It is therefore necessary to convert text tokens into continuous numerical representations, called **embeddings**.
- The embedding layer maps discrete token indices (integers) to dense vectors of fixed dimensions, which can take semantic meanings and provide a compact representation of the input for further processing.





Generate Input Embeddings

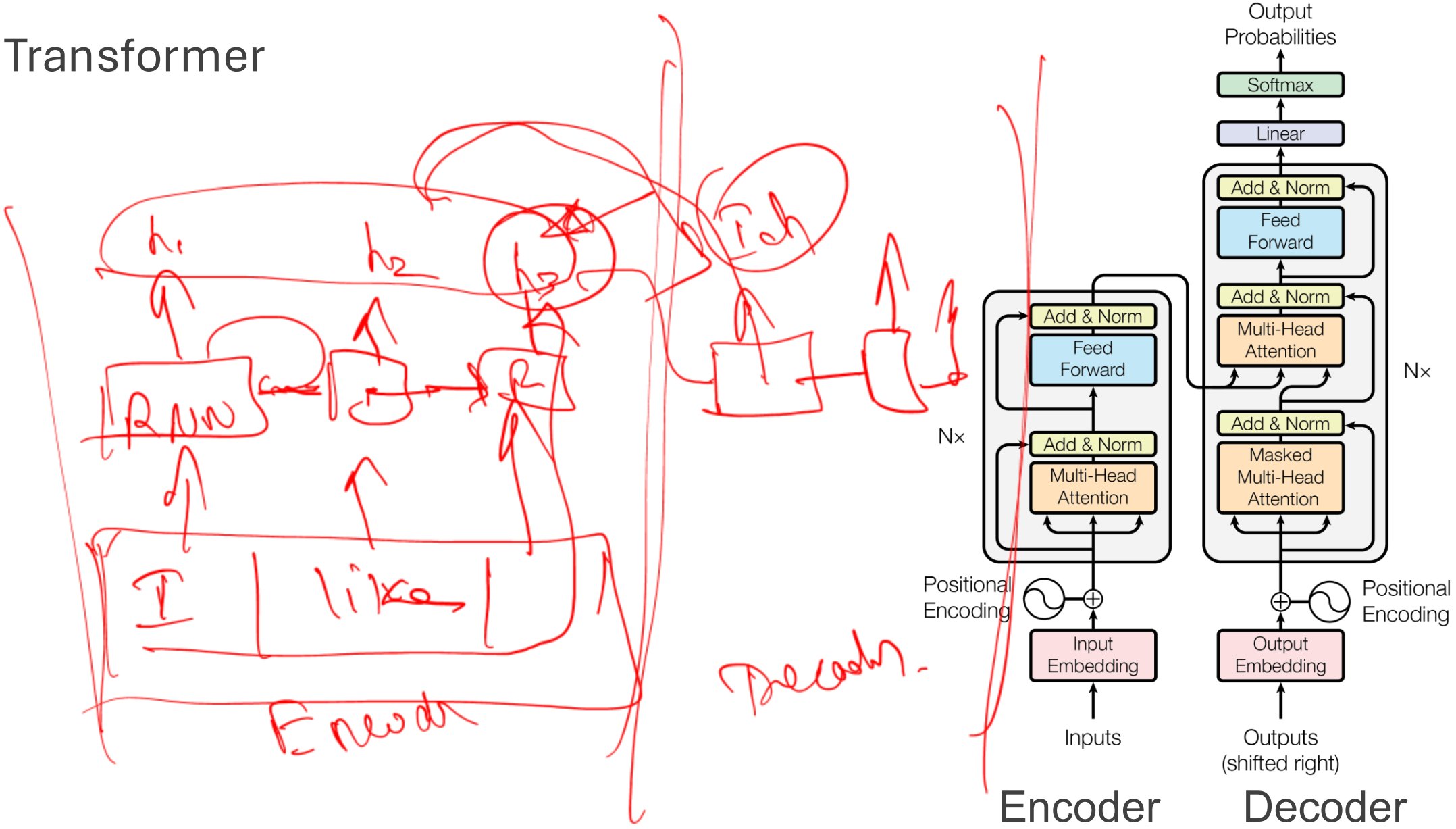
Foundations: Step III : **Positional Encoding**

- Problem - Transformer processes all the elements of a sequence in parallel without any regard for their order (Self-attention is permutation invariant)
  - Example : the sun rises in the east and permuted versions - rises in the sun the east / the east rises in the sun
- In natural language, it is important to consider the order of words in a sentence
- **Solution** : Explicitly add positional information to indicate where a word appears in a sequence
- **Common Approaches** - Binary, Integer, Sinusoidal, Rotary

Core Architectures

Core Architectures

Transformer



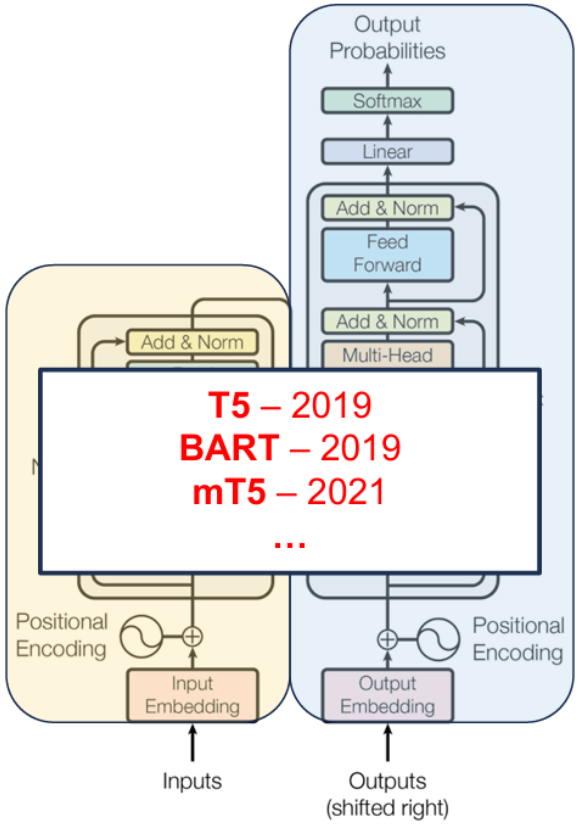
Core Architectures

Encoder-Only

BERT – 2018
DistilBERT – 2019
RoBERTa – 2019
ALBERT – 2019
ELECTRA – 2020
DeBERTa – 2020
…

Representation

Encoder-Decoder



T5 – 2019
BART – 2019
mT5 – 2021
…

Decoder-Only

GPT – 2018
GPT-2 – 2019
GPT-3 – 2020
GPT-Neo – 2021
GPT-3.5 (ChatGPT) – 2022
LLaMA – 2023
GPT-4 – 2023
…

Generation

Reference

Core Architectures

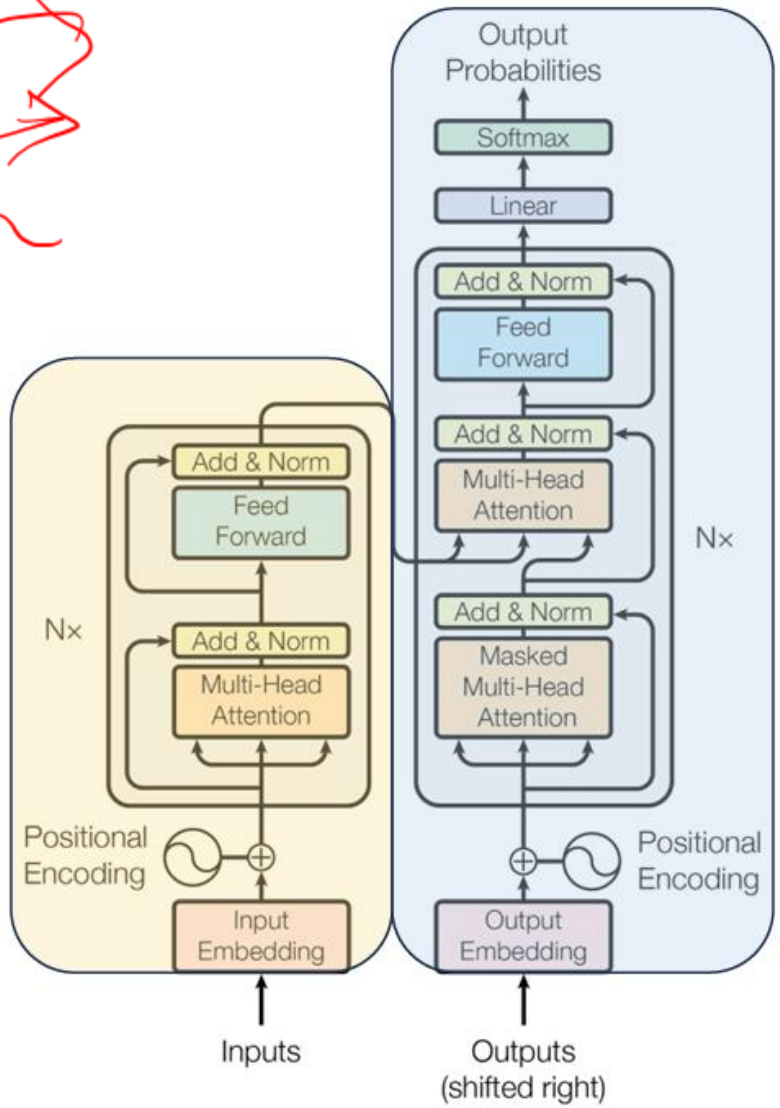
Input – input tokens
Output – hidden states

Model can see all timesteps

Does not usually output tokens, so no inherent auto-regressivity

Can also be adapted to generate tokens *by appending a module that maps hidden state dimensionality to vocab size*

Representation



Input – output tokens and hidden states*
Output – output tokens

Model can only see previous timesteps

Model is auto-regressive with previous timesteps' outputs

Can also be adapted to generate hidden states *by looking before token outputs*

Generation

Reference

Core Architectures: Encoder-Only Models

- Convert an input sequence of text into a rich numerical representation that is well suited for tasks like text classification or named entity recognition.
- The representation computed for a given token in this architecture depends both on the left (before the token) and the right (after the token) con- texts. This is often called **bidirectional attention.**

Reference

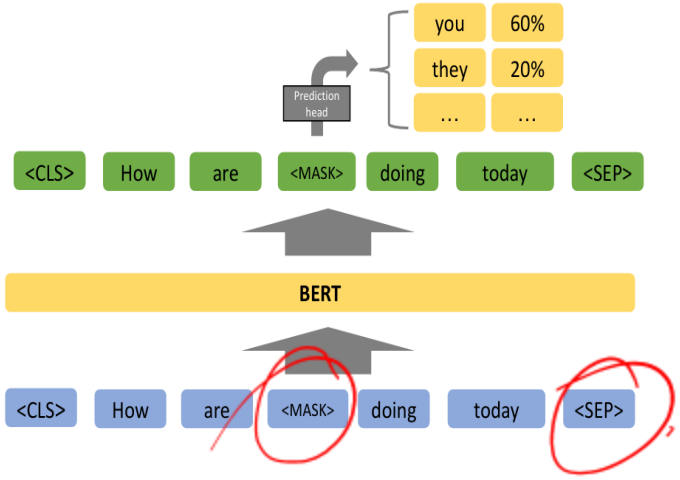
Core Architectures

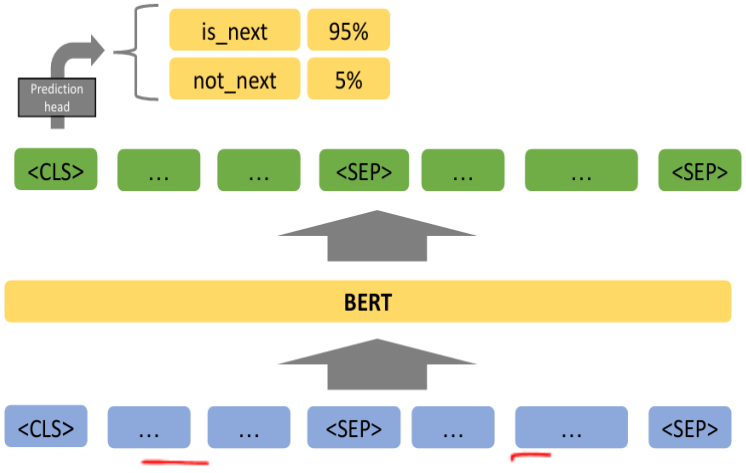
Encoder-Only Models

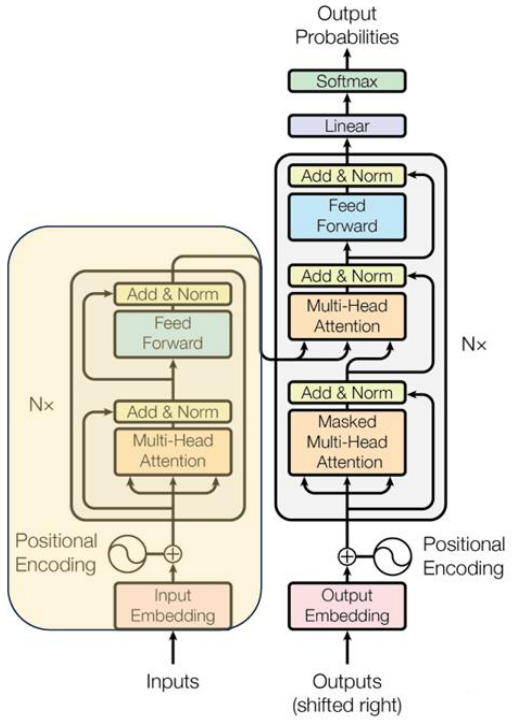
- One of the challenges in FM-building is the lack of task-specific training data. What if we learn an effective representation that can be applied to a variety of downstream tasks?

  Answer : BERT (Bidirectional Encoder Representations) trained using MLM (Masked Language Modeling – *Left*) and NSP (Next Sentence Prediction - *Right*) objectives

  Takeaway :
  - *Pre-training tasks can be invented flexibly*
  - *Different NLP tasks seem to be highly transferable with each other*







Reference

Core Architectures: Decoder-Only Models

- Given a prompt of text like “I like LLMs, I had a …” these models will auto- complete the sequence by iteratively predicting the most probable next word.
- The family of GPT models belong to this class.
- The representation computed for a given token in this architecture depends only on the left context.
- This is often called causal or **autoregressive attention.**

Reference

Core Architectures: Decoder-Only Models

- Can we leverage large corpus of unlabeled data to pretrain an FM that understands general patterns?

  Answer : GPT (Generative Pretrained Transformer)

  - GPT was trained to predict the next token, given the previous tokens (more learning signals than MLM)

Takeaway:

- *Effectiveness of Self-Supervised Learning* - model seems to be able to learn from generating the language itself, rather than from any specific task.
- *Language Model as a Knowledge Base* - a generatively pretrained model seems to have a decent zero-shot performance on a range of NLP tasks

Reference

Core Architectures: Encoder-decoder Models

- These are used for modeling complex mappings from one sequence of text to another
- Suitable for machine translation and summarization tasks.
- In addition to the Transformer architecture, which as we've seen combines an encoder and a decoder, the BART and T5 models belong to this class.

Reference

Attention Mechanisms

3Blue1Brown

https://youtu.be/wjZofJX0v4M?feature=shared

https://youtu.be/eMlx5fFNoYc?feature=shared

Attention Mechanisms

Self-Attention

- Core Dimensions:
  - N (Context Length): Number of tokens/words in the sequence
  - d_model: Dimension of input embeddings
  - d_k, d_q: Dimension of Key and Query vectors (usually equal)
  - d_v: Dimension of Value vectors

Example

- Input: "The cat sat here" (N=4 tokens, d_model=6)

```
Input Matrix X (N × d_model) = (4 × 6):

          [word]  [position] [syntax] [semantic] [tense] [entity]
X = The [  1.2     0.1       0.8      0.3       0.0      0.5  ]
    cat [  0.3     0.2       0.5      1.8       0.0      1.2  ]
    sat [  0.7     0.3       1.1      0.9       1.5      0.2  ]
    here[  0.4     0.4       0.6      1.2       0.0      0.8  ]
```

Attention Mechanisms

Step 1: Understanding Q, K, V Conceptually

- Query (Q): "What am I looking for?"
  - Represents what information each token wants to retrieve
  - Like a search query in a database
- Key (K): "What information do I offer?"
  - Represents what each token can provide to others
  - Like metadata/index terms in a database
- Value (V): "What is my actual content?"
  - The actual information that will be aggregated
    - Like the actual data records in a database

Attention Mechanisms

Step 2: Creating Q, K, V Matrices

- Let's use d_k = d_q = d_v = 4 (reduced from d_model=6)
- Weight Matrices:

```
W_Q (6×4) - "Learn what to look for"
W_K (6×4) - "Learn what to advertise"
W_V (6×4) - "Learn what to provide"
```

Attention Mechanisms

Step 2: Creating Q, K, V Matrices

- Actual Weight Matrices (learned during training):

```
W_Q = [0.5  0.2  0.1  0.3]    W_K = [0.4  0.1  0.3  0.2]    W_V = [0.3  0.2  0.1
0.4]
      [0.1  0.4  0.3  0.2]          [0.2  0.5  0.1  0.3]          [0.1  0.3  0.4
0.2]
      [0.3  0.1  0.5  0.1]          [0.1  0.3  0.4  0.2]          [0.2  0.1  0.3
0.4]
      [0.2  0.3  0.2  0.4]          [0.3  0.2  0.2  0.3]          [0.4  0.2  0.2
0.2]
      [0.1  0.2  0.3  0.3]          [0.2  0.1  0.3  0.4]          [0.3  0.4  0.1
0.2]
      [0.4  0.1  0.2  0.2]          [0.1  0.4  0.2  0.1]          [0.2  0.3  0.3
0.2]
```

Attention Mechanisms

Step 2: Creating Q, K, V Matrices

- Computing Q, K, V:

```
Q = X · W_Q (4×6 · 6×4 = 4×4)     K = X · W_K (4×4)     V = X · W_V (4×4)

Q (4×4) = [1.23  0.81  1.05  0.92]  ← "The" is looking for subject-related info
          [0.76  1.42  0.88  1.31]  ← "cat" is looking for action connections
          [1.15  0.97  1.33  1.08]  ← "sat" is looking for subject/location
          [0.89  1.28  0.94  1.17]  ← "here" is looking for spatial context

K (4×4) = [0.98  1.21  0.89  1.04]  ← "The" offers determiner/start info
          [0.82  1.53  1.12  0.95]  ← "cat" offers subject/entity info
          [1.07  1.09  1.25  1.18]  ← "sat" offers action/verb info
          [0.91  1.35  1.03  0.87]  ← "here" offers location info

V (4×4) = [1.31  1.02  0.95  1.15]  ← "The"'s actual information
          [1.08  1.45  1.23  0.89]  ← "cat"'s actual information
          [1.22  1.18  1.09  1.31]  ← "sat"'s actual information
          [1.14  1.33  1.07  0.98]  ← "here"'s actual information
```

Attention Mechanisms

Step 3: Attention Score Calculation

- Computing Attention Scores (Q · K^T):

```
Scores = Q · K^T / √d_k  (4×4 · 4×4 = 4×4)

Raw scores (before scaling):
         [The] [cat] [sat] [here]
[The]  [  4.21  4.85  5.12  4.33 ]
[cat]  [  4.56  5.91  5.43  4.87 ]
[sat]  [  5.02  5.73  5.98  5.21 ]
[here] [  4.43  5.42  5.31  4.76 ]

After scaling by √4 = 2:
         [The] [cat] [sat] [here]
[The]  [  2.11  2.43  2.56  2.17 ]
[cat]  [  2.28  2.96  2.72  2.44 ]
[sat]  [  2.51  2.87  2.99  2.61 ]
[here] [  2.22  2.71  2.66  2.38 ]
```

Attention Mechanisms

Step 4: Softmax Normalization

- Apply row-wise softmax to get attention weights:

```
Attention Weights A (4×4):
         [The] [cat] [sat] [here]
[The]  [ 0.20  0.28  0.32  0.20 ]  ← "The" pays most attention to "sat" (32%)
[cat]  [ 0.18  0.35  0.29  0.18 ]  ← "cat" focuses on itself (35%) and "sat" (29%)
[sat]  [ 0.19  0.28  0.33  0.20 ]  ← "sat" focuses on itself (33%) and "cat" (28%)
[here] [ 0.17  0.31  0.30  0.22 ]  ← "here" attends to "cat" (31%) and "sat" (30%)
```

Attention Mechanisms

Step 5: Value Aggregation

- Final Output = A · V:

```
Output (4×4) = Attention (4×4) · Values (4×4)

Output = [1.16  1.28  1.10  1.08]  ← New representation for "The"
         [1.15  1.30  1.14  1.11]  ← New representation for "cat"
         [1.17  1.29  1.12  1.10]  ← New representation for "sat"
         [1.16  1.31  1.13  1.09]  ← New representation for "here"
```

```
Input Matrix X (N × d_model) = (4 × 6):

       [word]  [position] [syntax] [semantic] [tense] [entity]
X = The [  1.2     0.1      0.8      0.3      0.0      0.5  ]
    cat [  0.3     0.2      0.5      1.8      0.0      1.2  ]
    sat [  0.7     0.3      1.1      0.9      1.5      0.2  ]
    here[  0.4     0.4      0.6      1.2      0.0      0.8  ]
```

Attention Mechanisms

Dimension Analysis Throughout the Process (Memory Requirement)

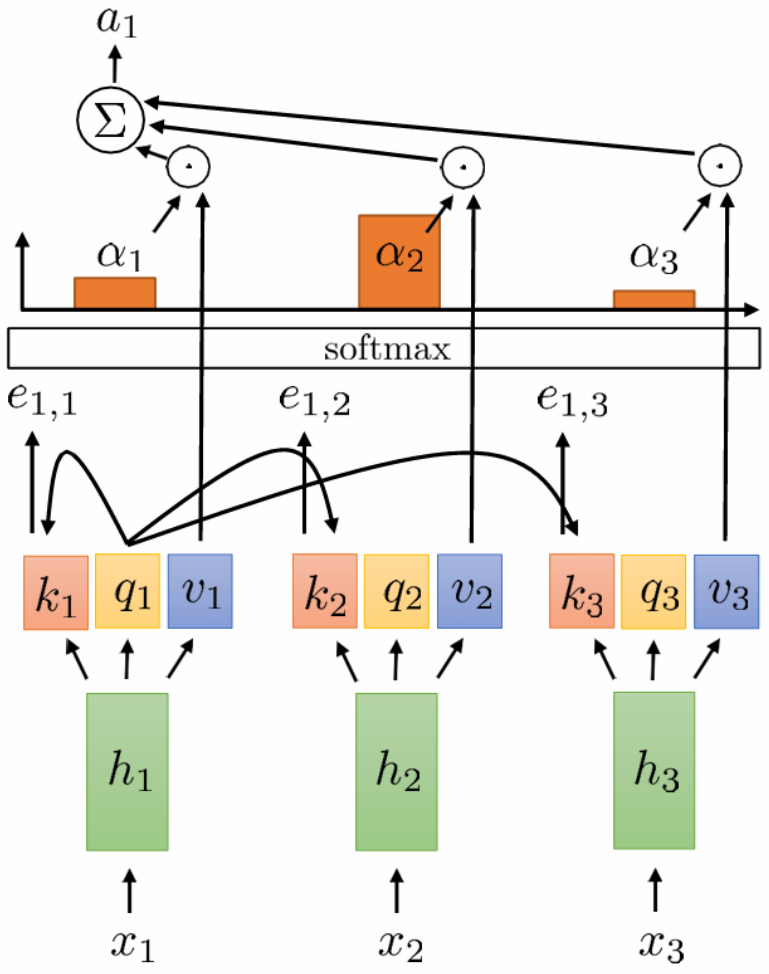
Stage	Operation	Dimensions	Memory Usage
Input	X	(N × d_model) = (4 × 6)	24 values
Projection	W_Q, W_K, W_V	(d_model × d_k) = (6 × 4) each	72 values total
Q, K, V	X · W	(N × d_k) = (4 × 4) each	48 values total
Attention Scores	Q · K^T	(N × N) = (4 × 4)	16 values
Attention Weights	softmax(scores)	(N × N) = (4 × 4)	16 values
Output	A · V	(N × d_v) = (4 × 4)	16 values

Attention Mechanisms

Scaling to Real Transformers

- GPT-3 Scale Example:
  - Context Length (N): 2048 tokens
  - d_model: 12288
  - d_k, d_v: 128 (per head)
  - Number of heads: 96
  - Attention matrix per head: 2048 × 2048 = 4.2M values
  - Total attention across all heads: 4.2M × 96 = 403M values per layer!
- Memory and Computation Complexity:
  - Memory: $O(N^2)$ - quadratic in sequence length
  - Computation: $O(N^2 \times d)$ - quadratic in sequence, linear in dimension
  - This is why context length is limited in transformers

Attention Mechanisms: Self-Attention Fundamentals



$$a_l = \sum_t \alpha_{l,t} v_t$$

$$\alpha_{l,t} = \exp(e_{l,t}) / \sum_{t'} \exp(e_{l,t'})$$

$$e_{l,t} = q_l \cdot k_t$$

$$v(h_t) = W_v h_t$$

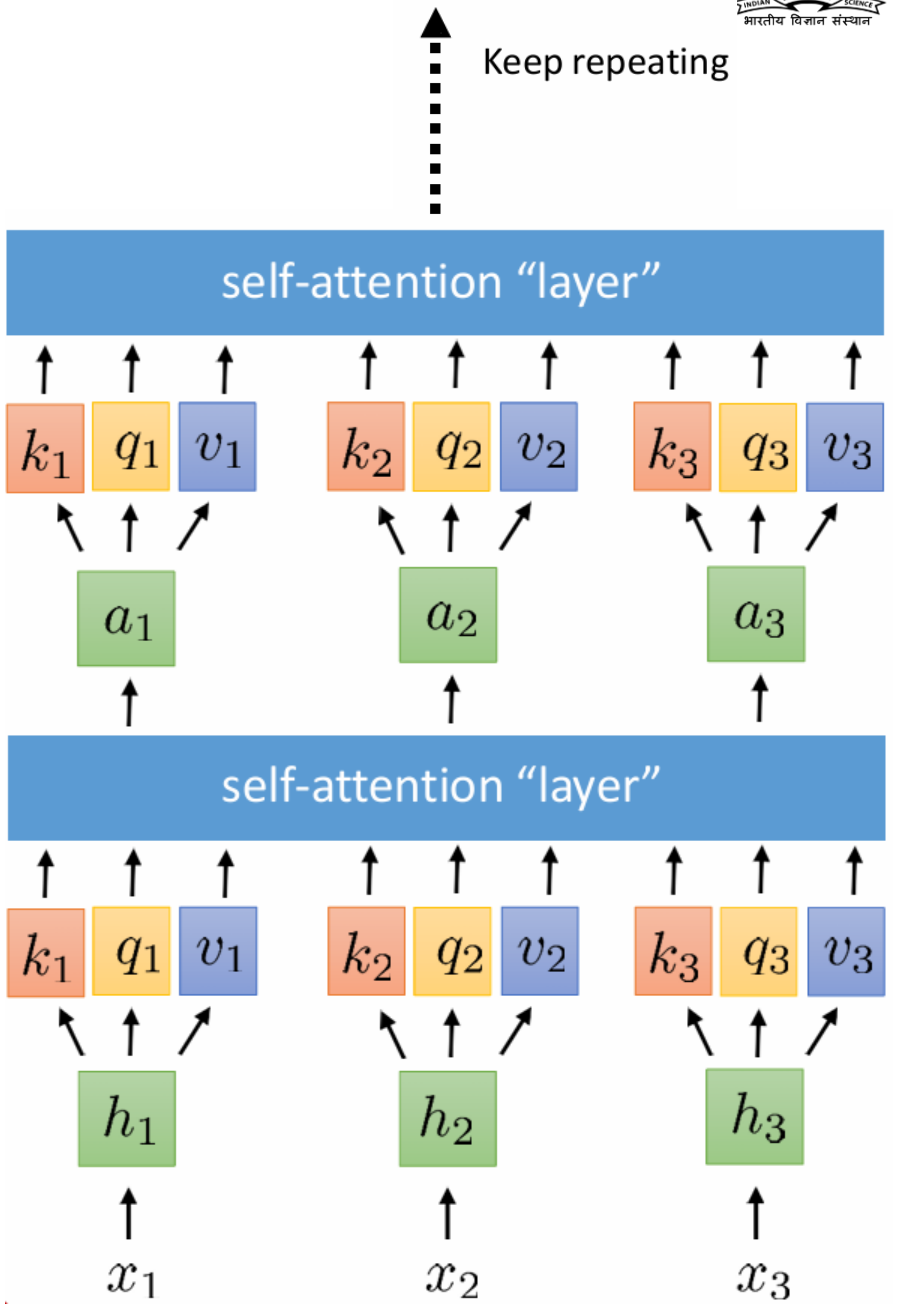
$$k_t = W_k h_t$$

$$q_t = W_q h_t$$

$$h_t = \sigma(W x_t + b)$$

shared weights at all time steps

(or any other nonlinear function)



Reference

Attention Mechanisms

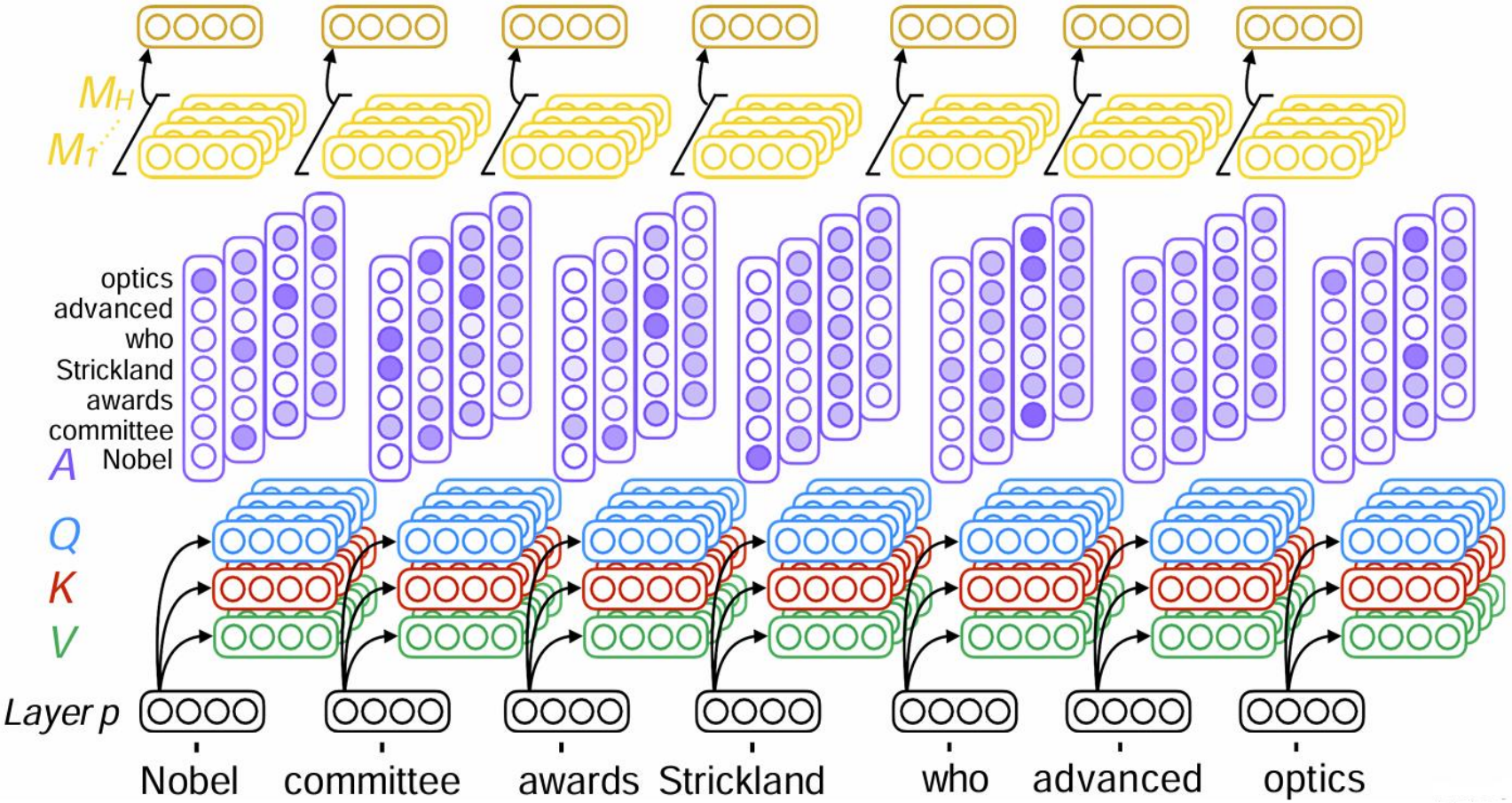
Multi-Head Attention

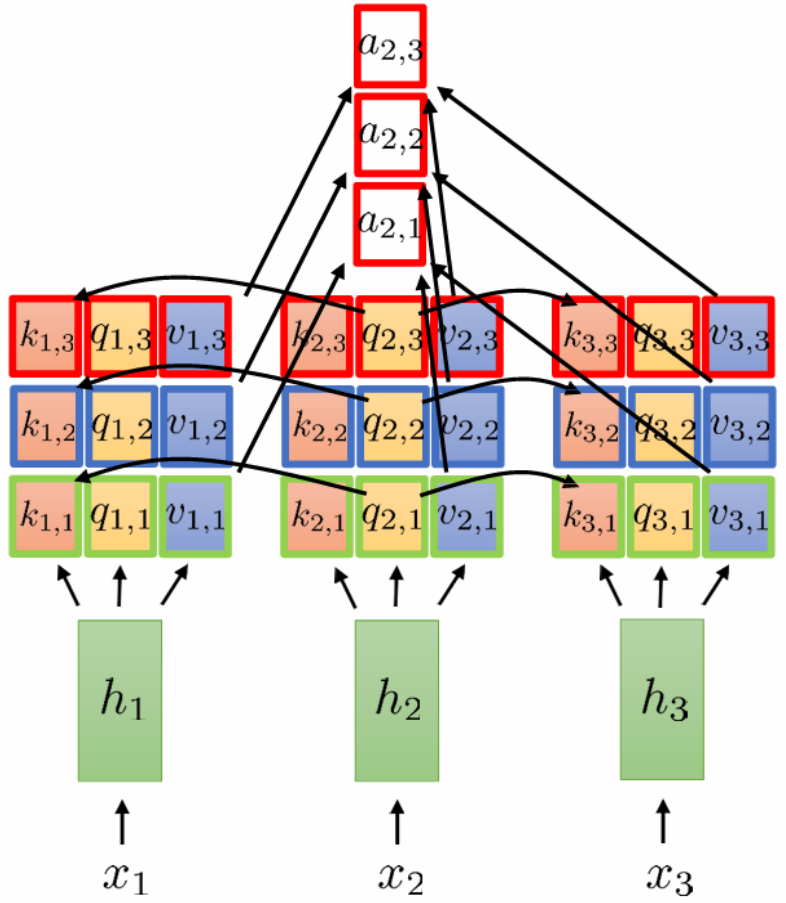
$$a_l = \sum_t \alpha_{l,t} v_t$$

$$e_{l,t} = q_l \cdot k_t$$

Due to the softmax function, this will be heavily influenced by a single value

It's challenging to clearly specify that you want two distinct elements, like the subject and object in a sentence



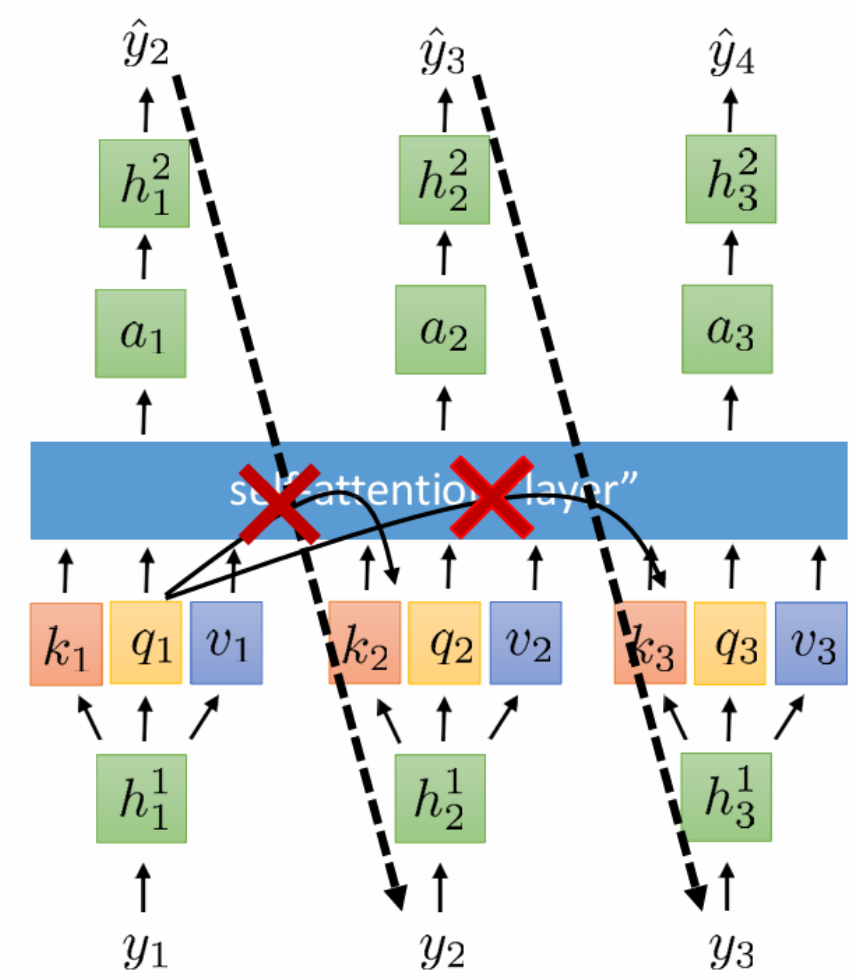


Reference

Attention Mechanisms

Masked Attention

- In a crude self-attention language model, there would be several alternating self-attention layers and position-wise feedforward networks
- **Big problem**: self-attention at step 1 can look at the value at steps 2 & 3, which is based on the inputs at steps 2 & 3
- At test time (when decoding), the inputs at steps 2 & 3 will be based on the output at step 1, which requires knowing the input at steps 2 & 3
- Must allow self-attention into the **past**, but not into the **future**



$$e_{l,t} = q_l \cdot k_t$$

SOLUTION

$$e_{l,t} = \begin{cases} q_l \cdot k_t & \text{if } l \geq t \\ -\infty & \text{otherwise} \end{cases}$$

in practice:

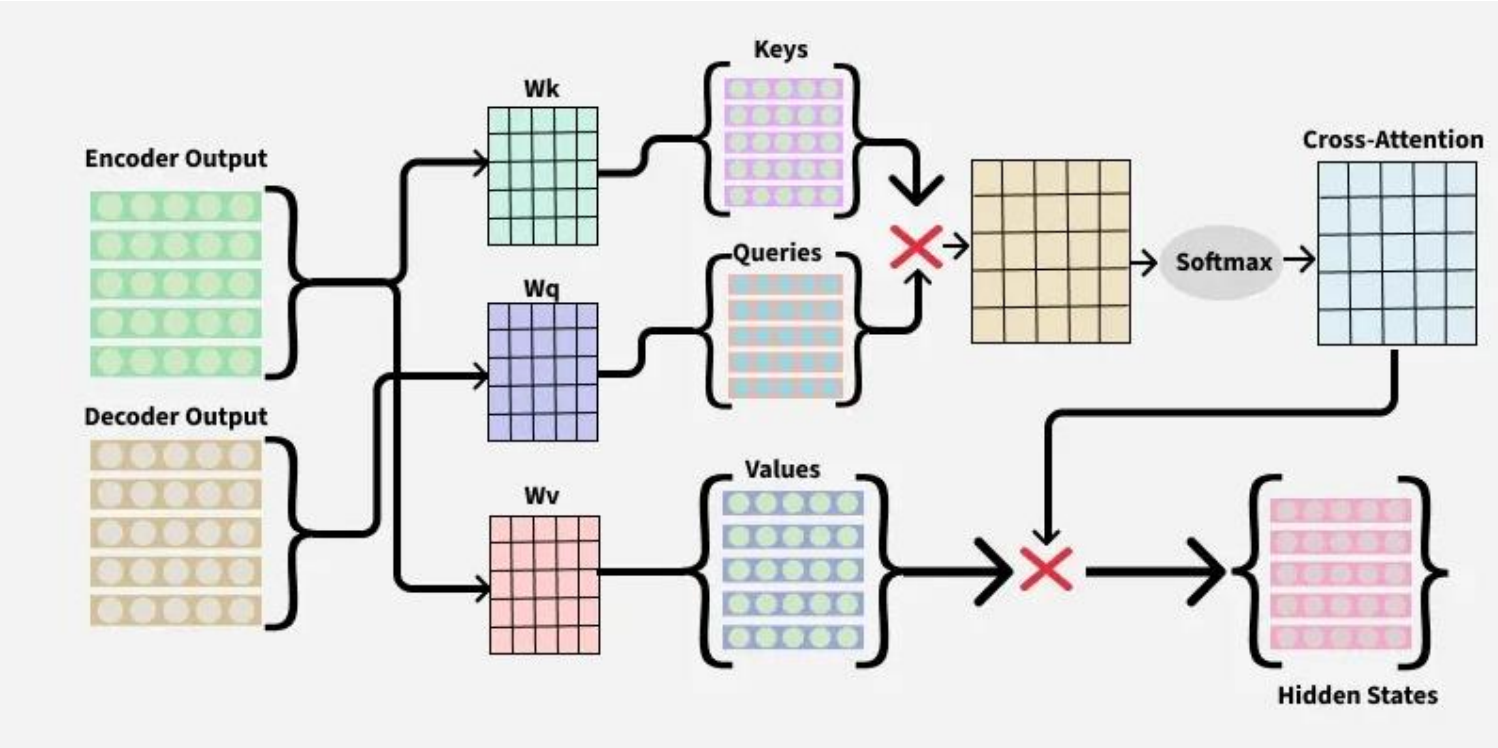
just replace $\exp(e_{l,t})$ with 0 if $l < t$

Reference

Attention Mechanisms

Cross-Attention

- Cross-Attention acts as a bridge b/w the encoder and decoder.
- Cross-attention mechanism compares the query from the decoder with the keys from the encoder.
- It calculates how well each query matches each key.



The formula for Cross-Attention is very similar to Self-Attention, except the query (Q) comes from the decoder's output and the key (K), value (V) come from the encoder's output.

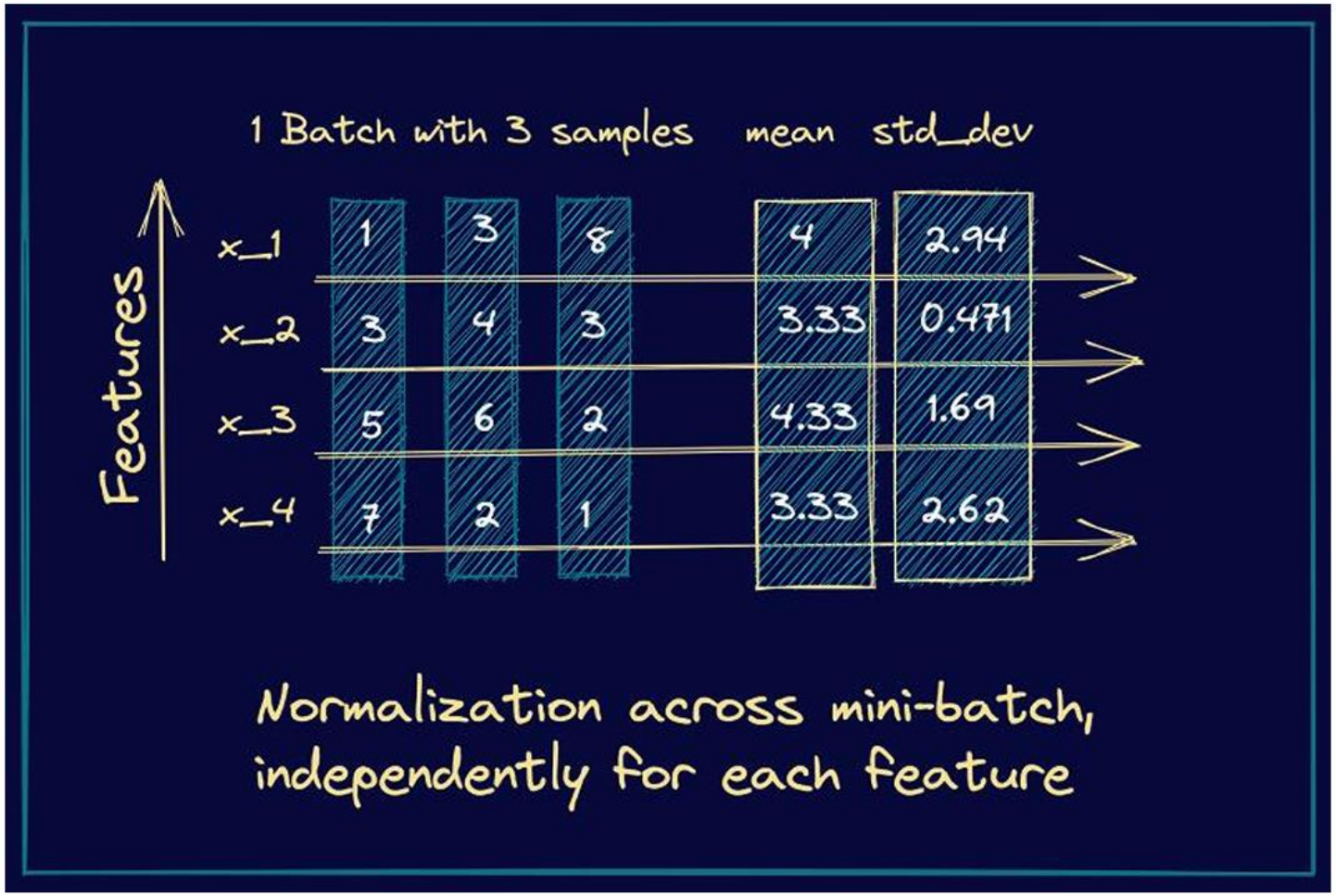
X – represents dot product

Additional Topics

Normalization and Connections

Normalization and Connections

Batch Normalization



$$\mu_b = \frac{1}{B}\sum_{i=1}^{B} x_i \quad (1)$$

$$\sigma_b^2 = \frac{1}{B}\sum_{i=1}^{B}(x_i - \mu_b)^2 \quad (2)$$

$$\hat{x}_i = \frac{x_i - \mu_b}{\sqrt{\sigma_b^2}} \quad (3)$$

$$or\ \hat{x}_i = \frac{x_i - \mu_b}{\sqrt{\sigma_b^2 + \epsilon}} \quad (3)$$

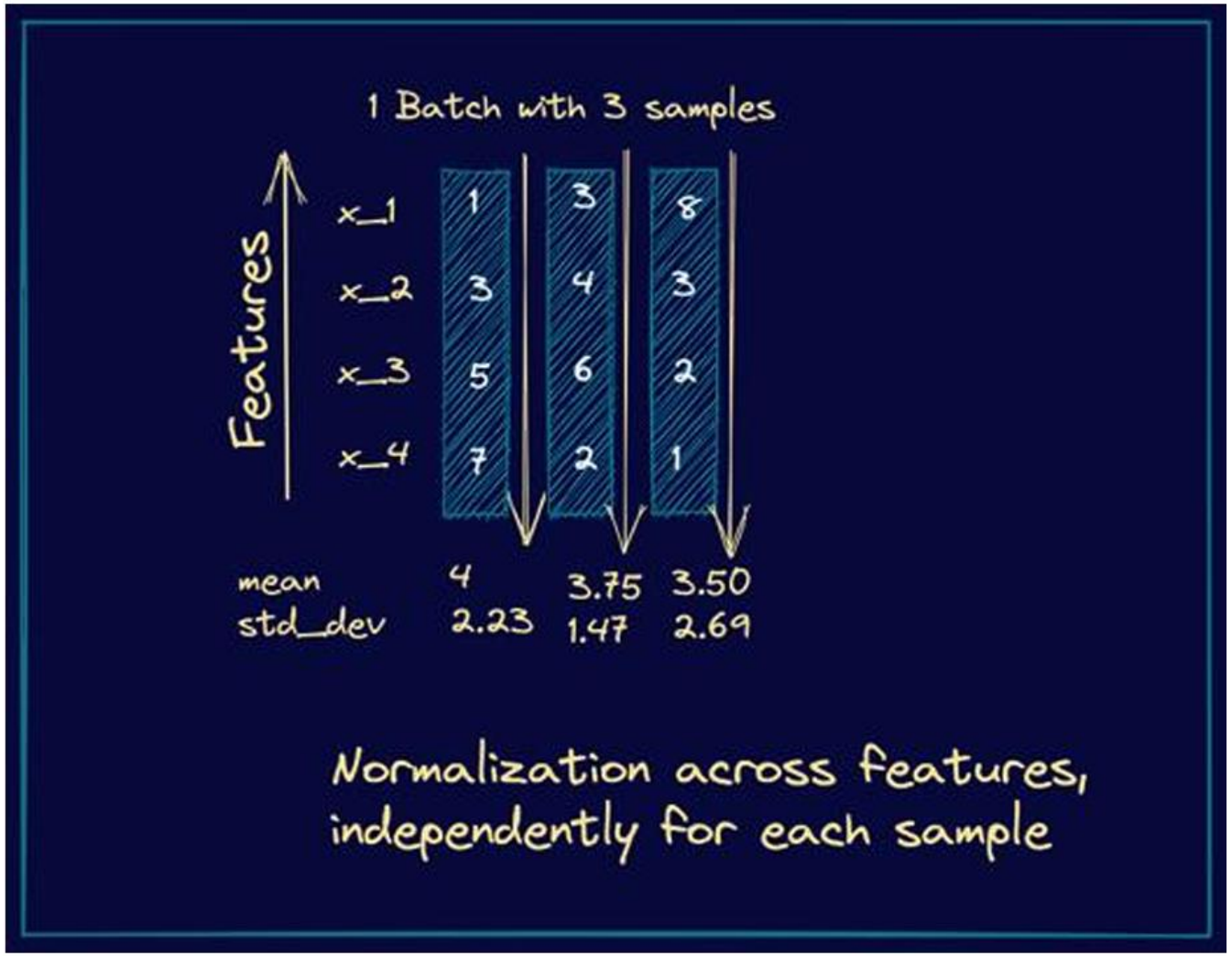
$$Adding\ \epsilon\ helps\ when\ \sigma_b^2\ is\ small$$

$$y_i = \mathcal{BN}(x_i) = \gamma . x_i + \beta \quad (4)$$

Image Source: LLMs: https://www.pinecone.io/learn/batch-layer-normalization/

Normalization and Connections

Layer Normalization



$$\mu_l = \frac{1}{d}\sum_{i=1}^{d} x_i \ (1)$$

$$\sigma_l^2 = \frac{1}{d}\sum_{i=1}^{d}(x_i - \mu_l)^2 \ (2)$$

$$\hat{x}_i = \frac{x_i - \mu_l}{\sqrt{\sigma_l^2}} \ (3)$$

$$or\ \hat{x}_i = \frac{x_i - \mu_l}{\sqrt{\sigma_l^2 + \epsilon}} \ (3)$$

Adding ϵ *helps when* σ_l^2 *is small*

$$y_i = \mathcal{LN}(x_i) = \gamma . x_i + \beta \ (4)$$

Image Source: LLMs: https://www.pinecone.io/learn/batch-layer-normalization/

Normalization and Connections

Comparison

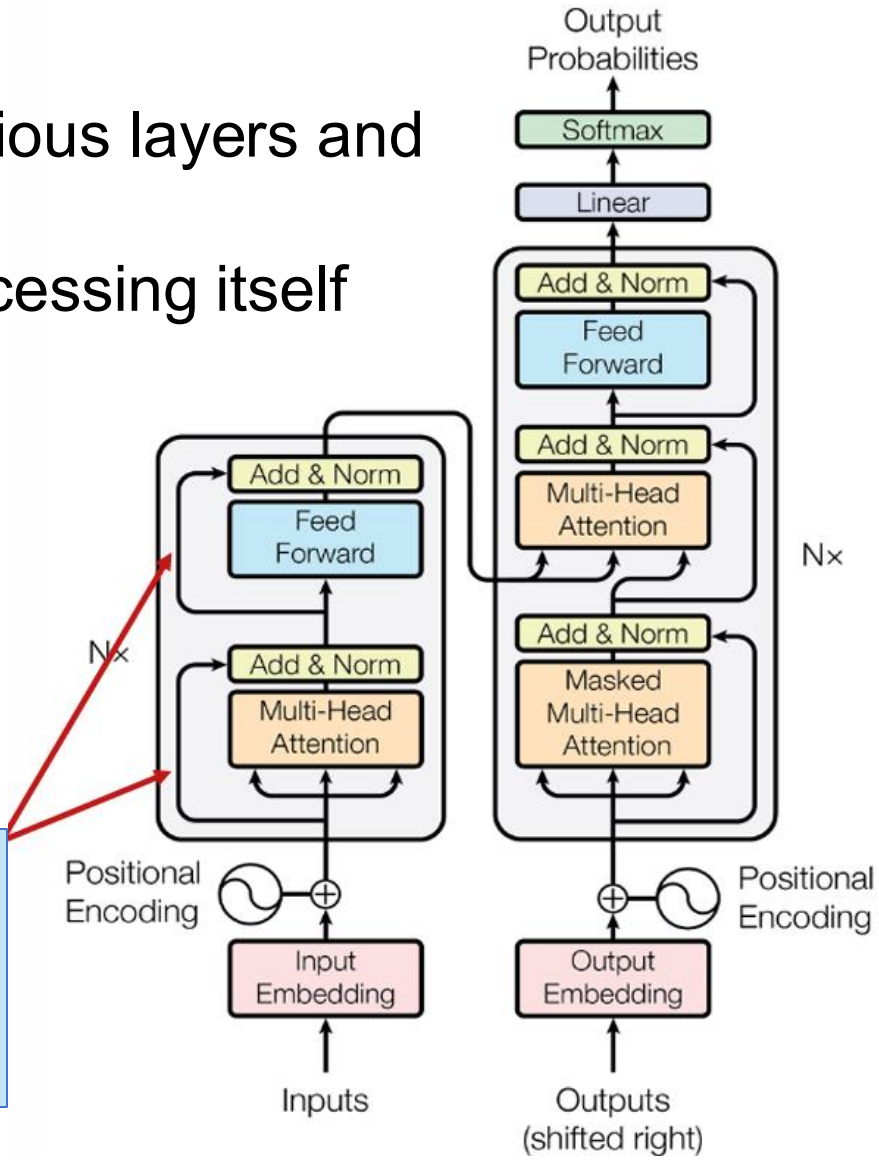
Batch Normalization	Layer Normalization
It normalizes each feature independently across the mini-batch	It normalizes each of the inputs in the batch independently across all features.
As it is dependent on batch size, it's not effective for small batch sizes	It is independent of the batch size, so it can be applied to batches with smaller sizes as well.
It requires different processing at training and inference times	As it is done along the length of input to a specific layer, the same set of operations can be used at both training and inference times

Normalization and Connections

Residual Connections

- Residual stream is simply the sum of output of all previous layers and the original embedding.
- It is a communication channel as it doesn't do any processing itself and all layers communicate through it.
- Every layer performs an arbitrary linear transformation to read in information from the residual stream at the start, and performs another arbitrary linear transformation before adding to write its output back into the residual stream.

Residual connections, which mean that we add the input to a particular block to its output, help improve gradient flow

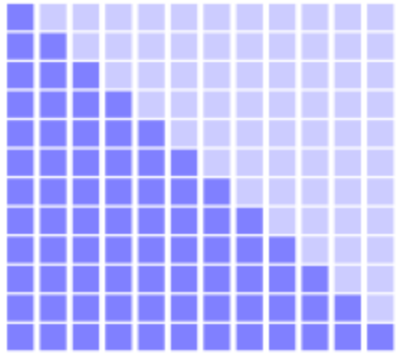


Refer : A Mathematical Framework for Transformer Circuits

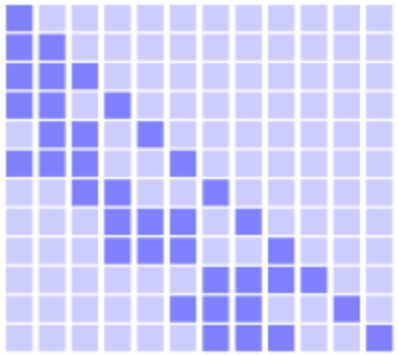
Advanced Techniques

Advanced Techniques

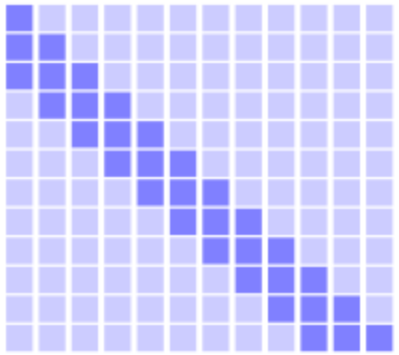
Efficient Attention Mechanisms



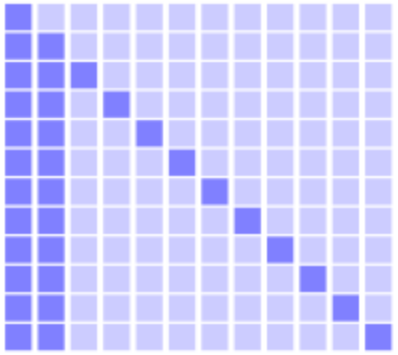
Standard Attn



Span-based/Local Attn



Strided Attn



Global Attn

Note:

- Dark cells mean, *A*'(i, j) != 0 (i.e., i attends to j), and light cells mean *A*'(i, j) = 0 (i.e., i does not attend to j).

Advanced Techniques

Grouped Query Attention (GQA)

- **Aim**: To reduce the need for storing a large amount of KV cache
- LLM server can handle more requests, larger batch sizes and increased throughput
- Cannot significantly reduce the computational load
- Quality degradation remains