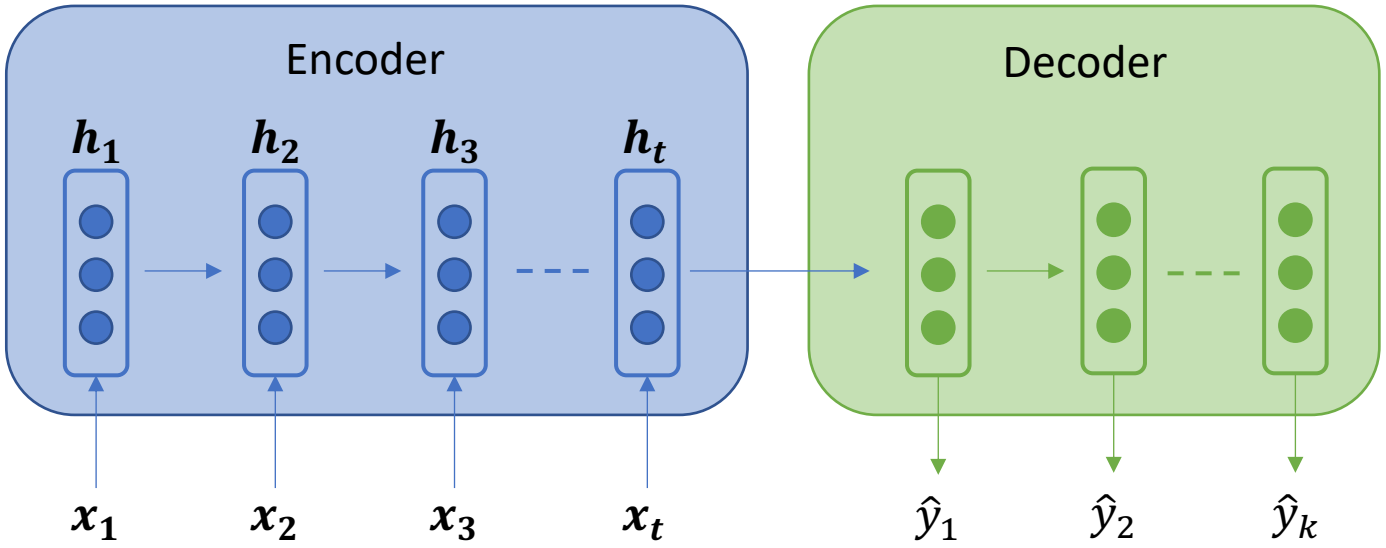
EE7207 Week 9

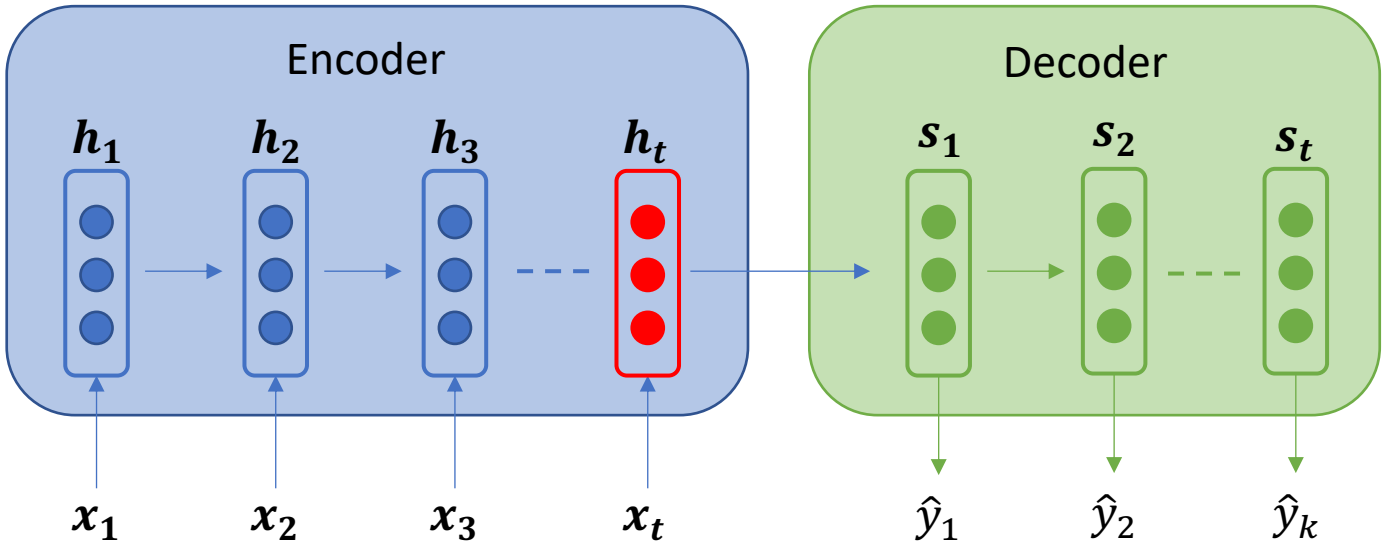
Encoder-Decoder Architecture

- Encoder: Reads and encodes the input sequence into a fixed-size representation (or set of hidden states).
- Decoder: Takes this encoded representation and decodes it step-by-step into the target sequence.



Bottleneck of encoder-decoder network

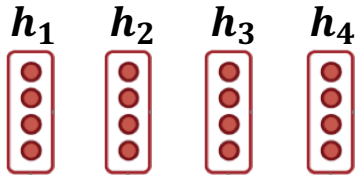
- Encoding of input sequence: a fixed length vector $\boldsymbol{h_t}$
- Need to capture all necessary information of input sequence
- Information bottleneck, especially when input sequence is long



Attention: thinking process

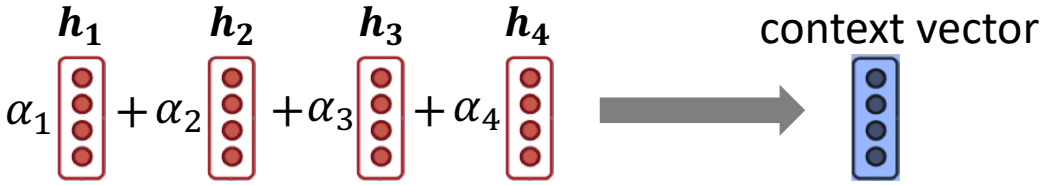
How to solve the bottleneck problem?

Instead of only using only $\boldsymbol{h_4}$, let's use all encoder hidden states!



How do we deal with variable length input sequence?

Let's do a weighted sum of all encoder hidden states!

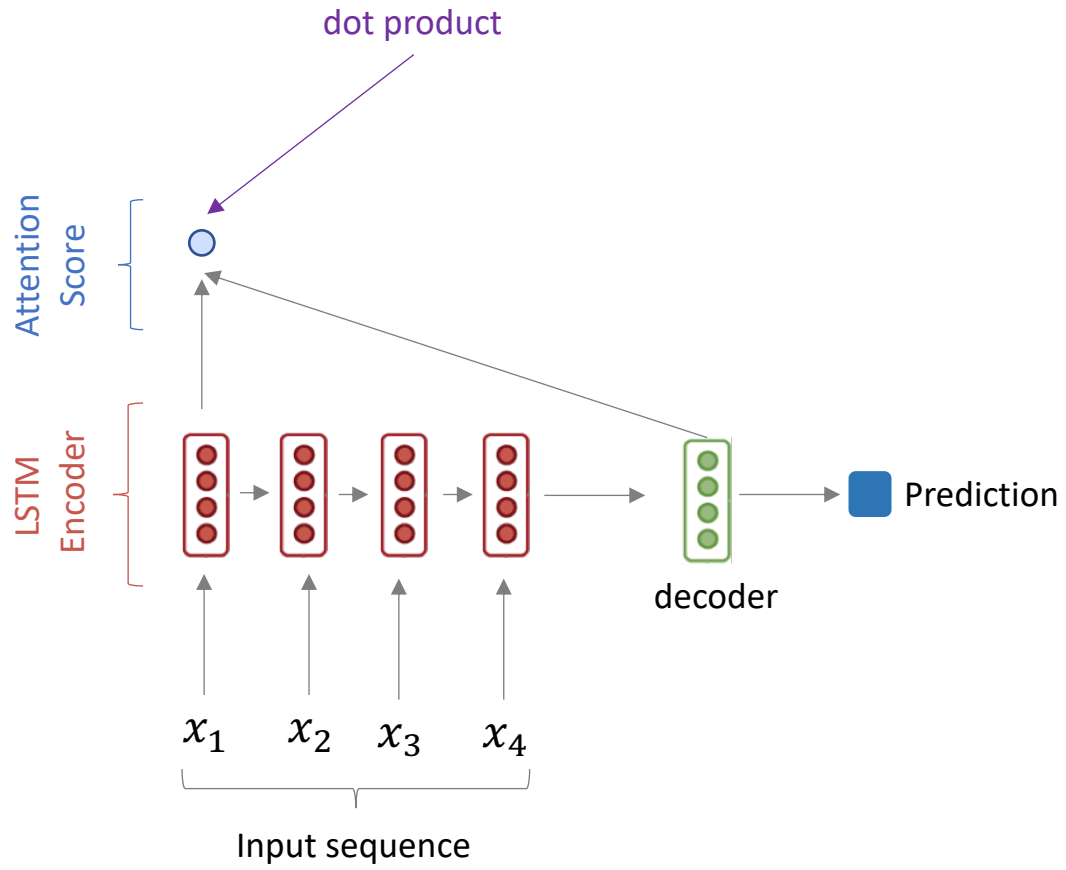


How do we get the weights α_i?

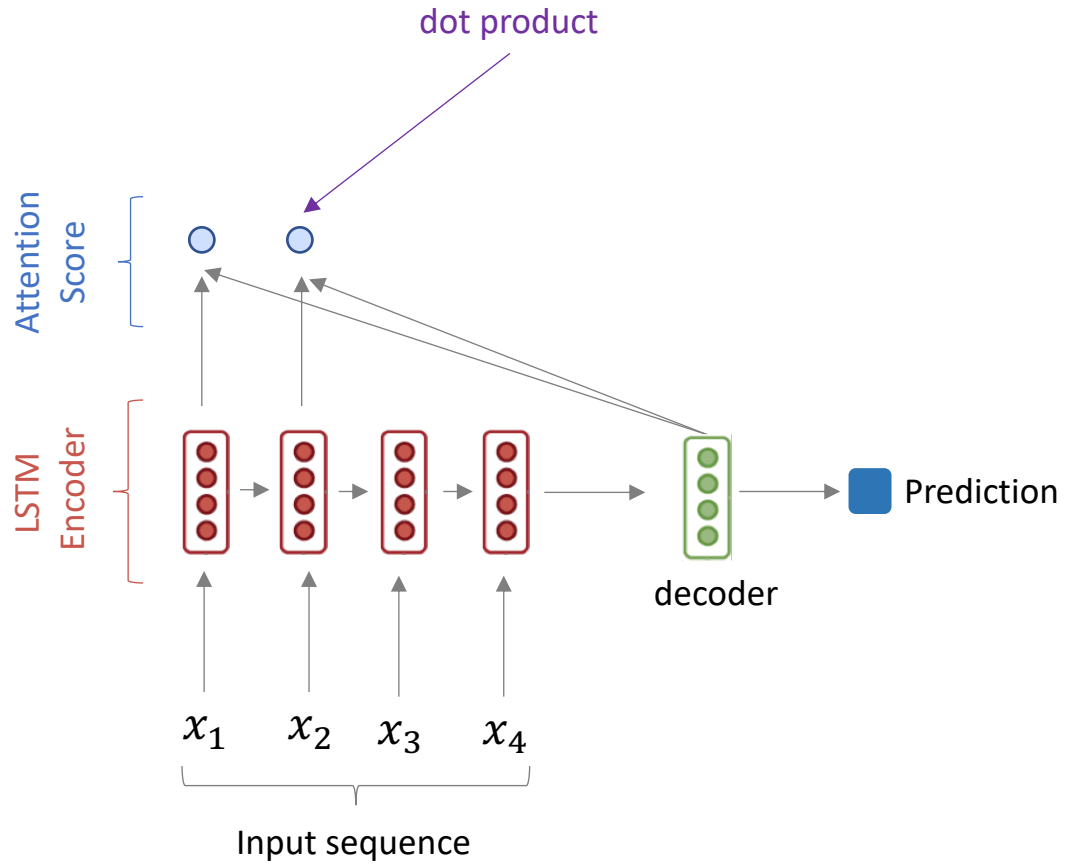
Step 1: dot product

Step 2: softmax $\alpha_i = \frac{\exp(score_i)}{\sum_{j=1}^{4} \exp(score_j)}$

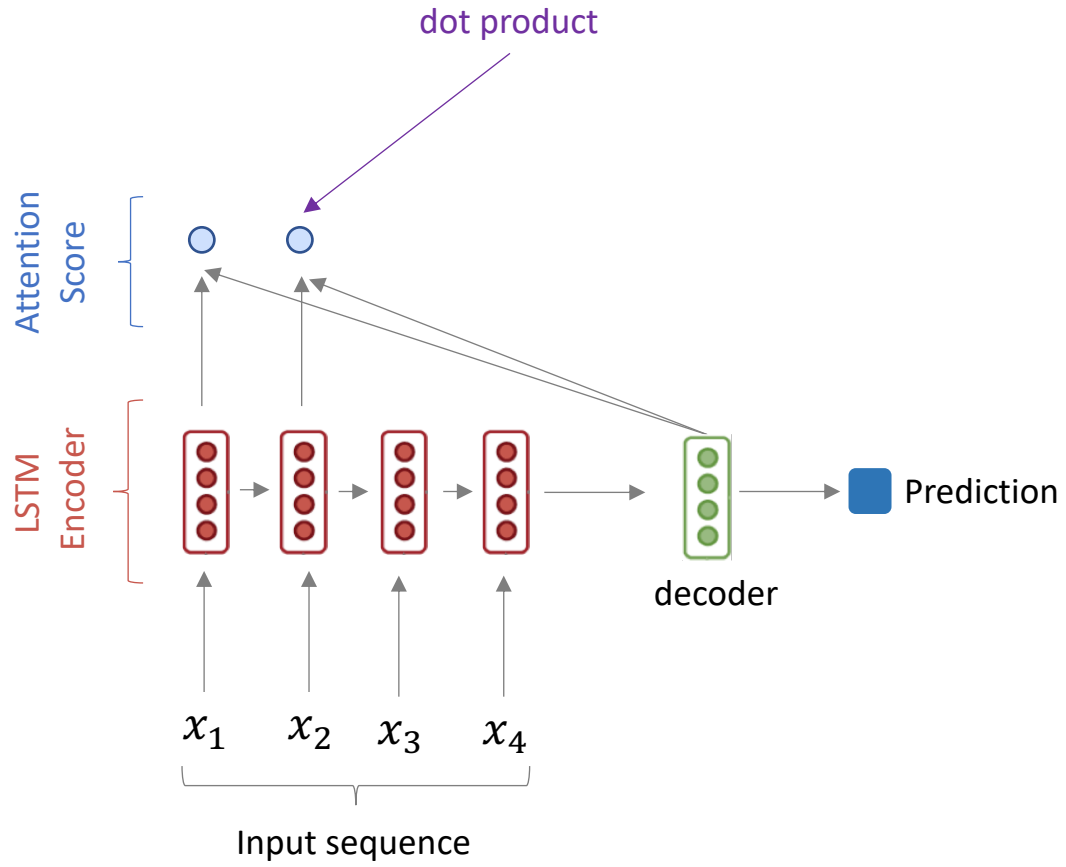
LSTM with attention



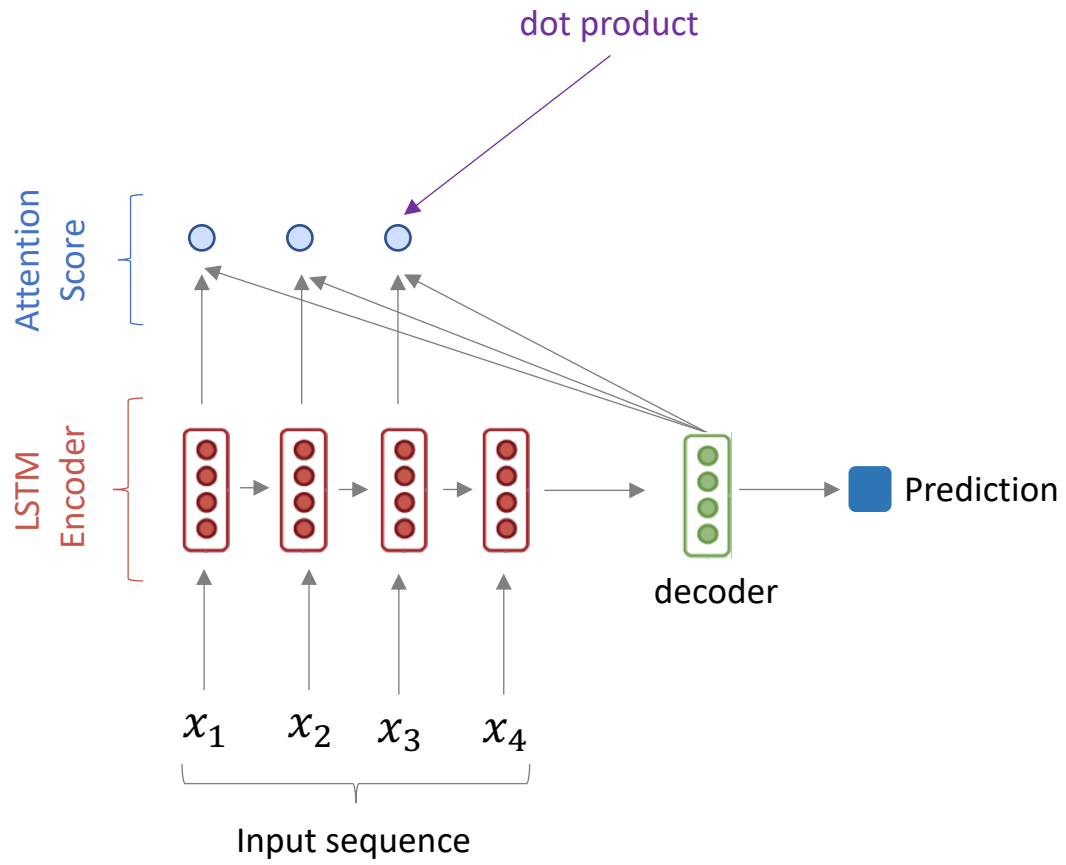
LSTM with attention



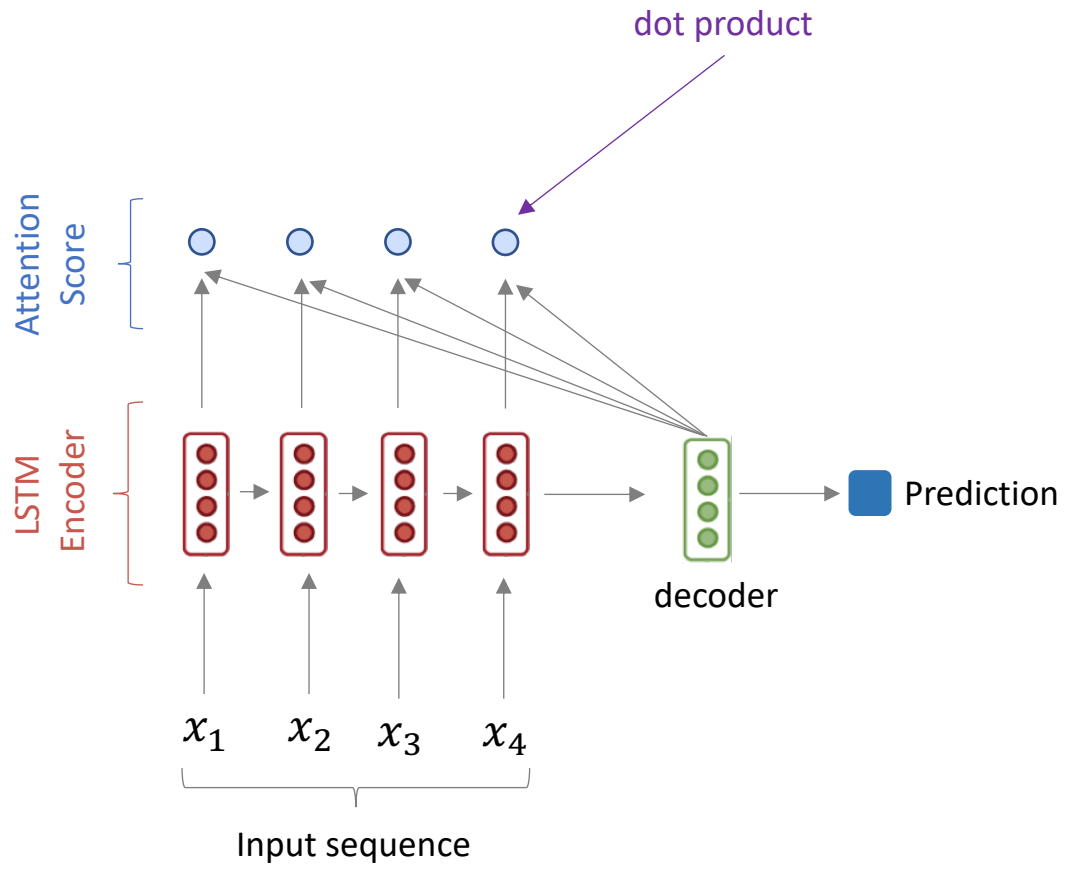
LSTM with attention



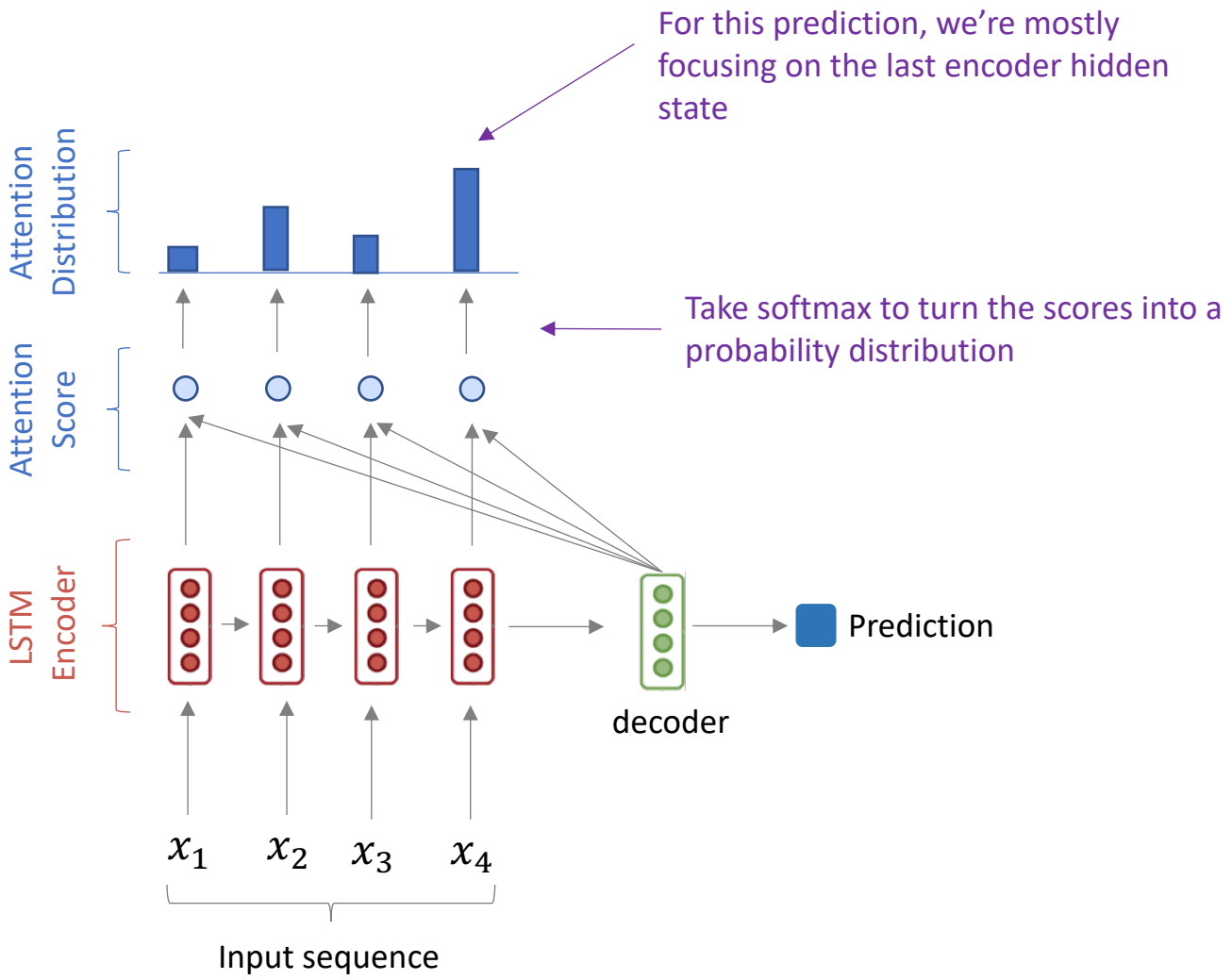
LSTM with attention



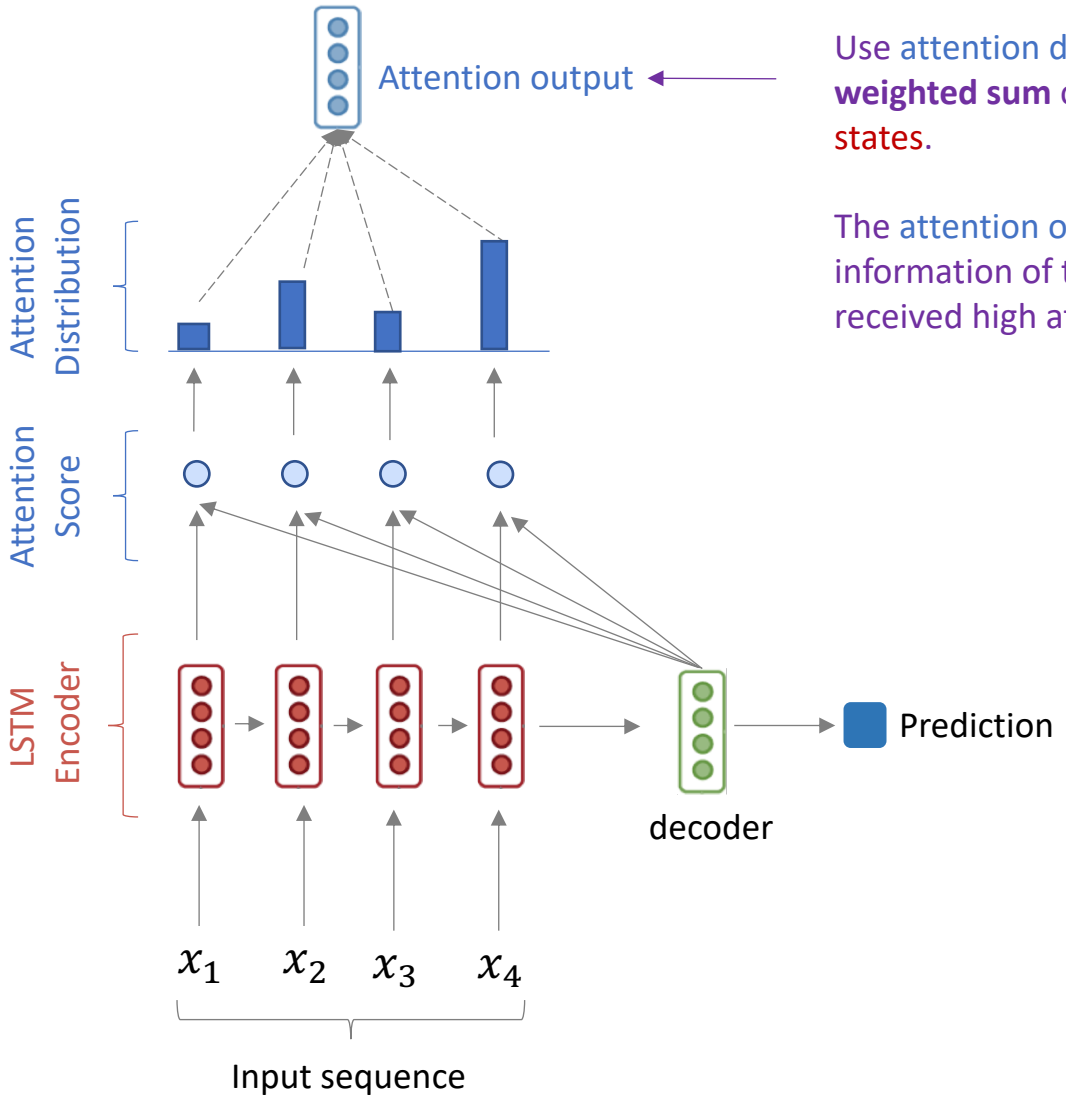
LSTM with attention



LSTM with attention



LSTM with attention



Use attention distribution to taka a **weighted sum** of the encoder hidden states.

The attention output mostly contains information of the hidden states that received high attention.

Another way to compute attention score: key-query-value attention

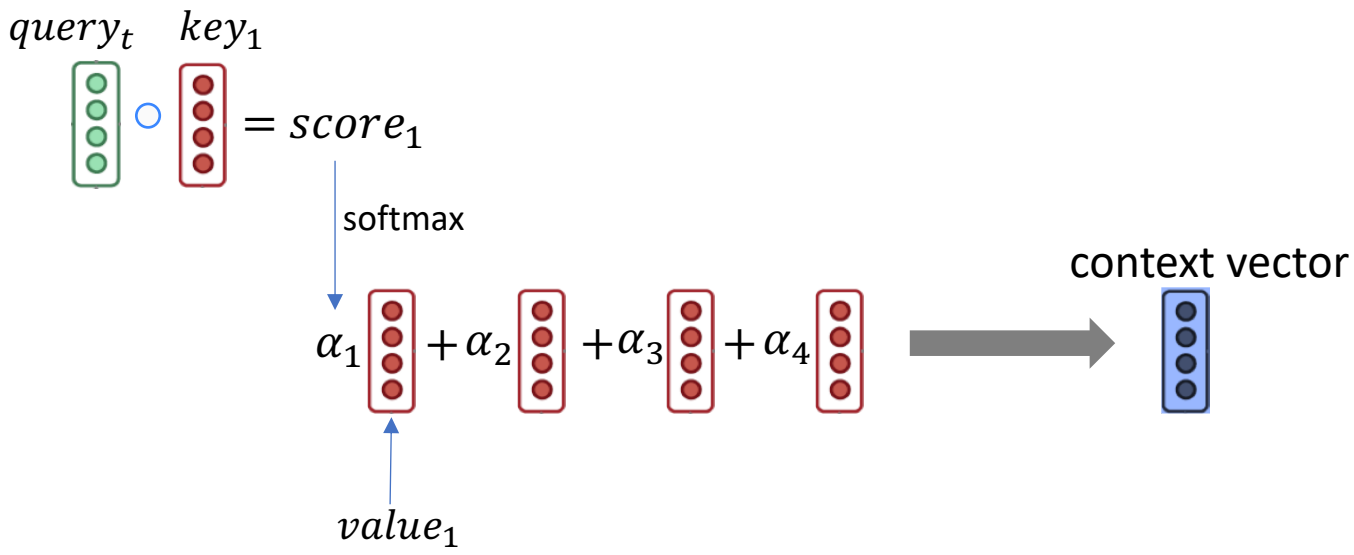
Limitation of dot-product attention:

- What if the dimensions of decoder hidden states and encoder hidden states differ

Inspiration from Information Retrieval:

- **Query**: What you're looking for.
- **Key**: Indexed attributes of data.
- **Value**: The actual content to retrieve.

Key-Query-Value attention:



Motivation: Why Move Beyond RNNs?

Sequential Processing

- RNN training is inherently sequential—token one, then token two, and so on—making it harder to speed up using modern parallel hardware.

Long-Term Dependency Issues

- As sequences grow in length, RNNs become less efficient at capturing dependencies between distant tokens.

Transformers

Attention Is All You Need

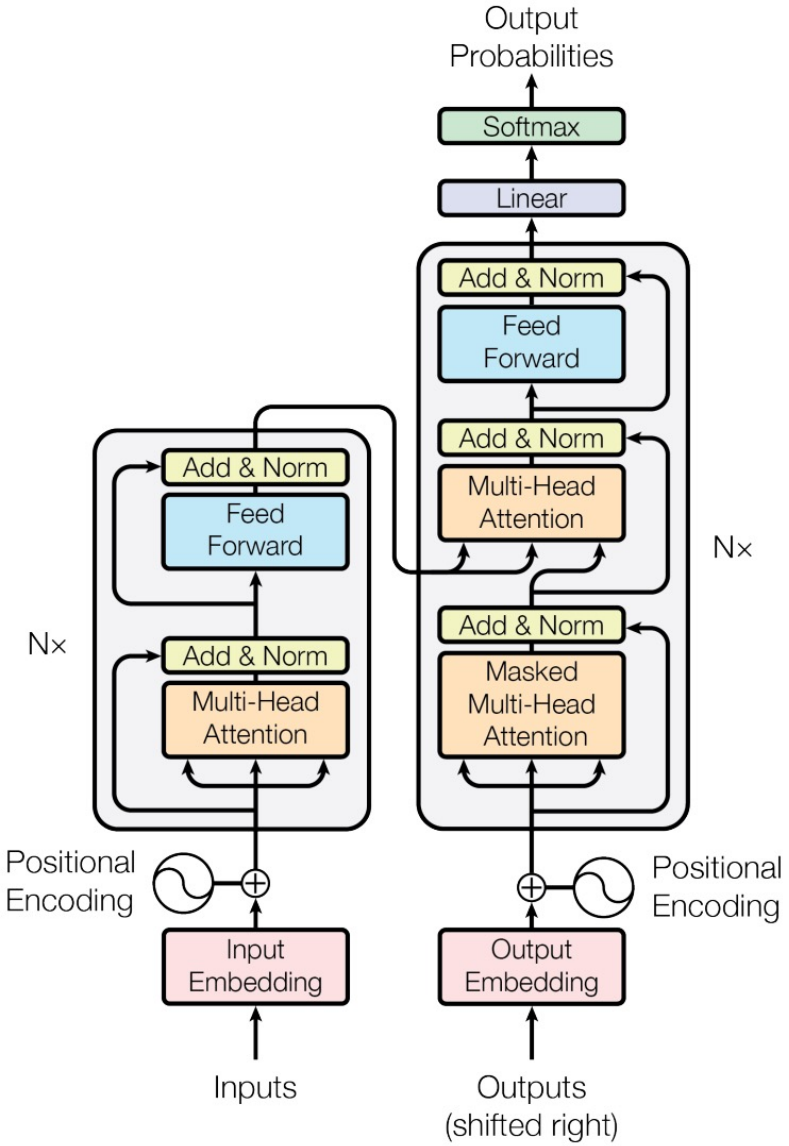
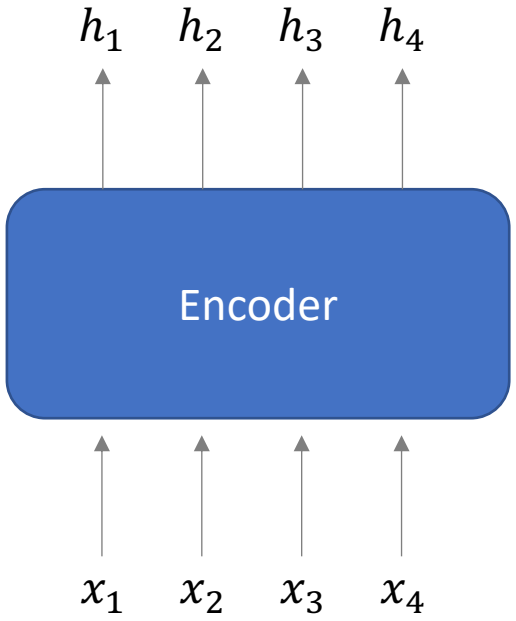
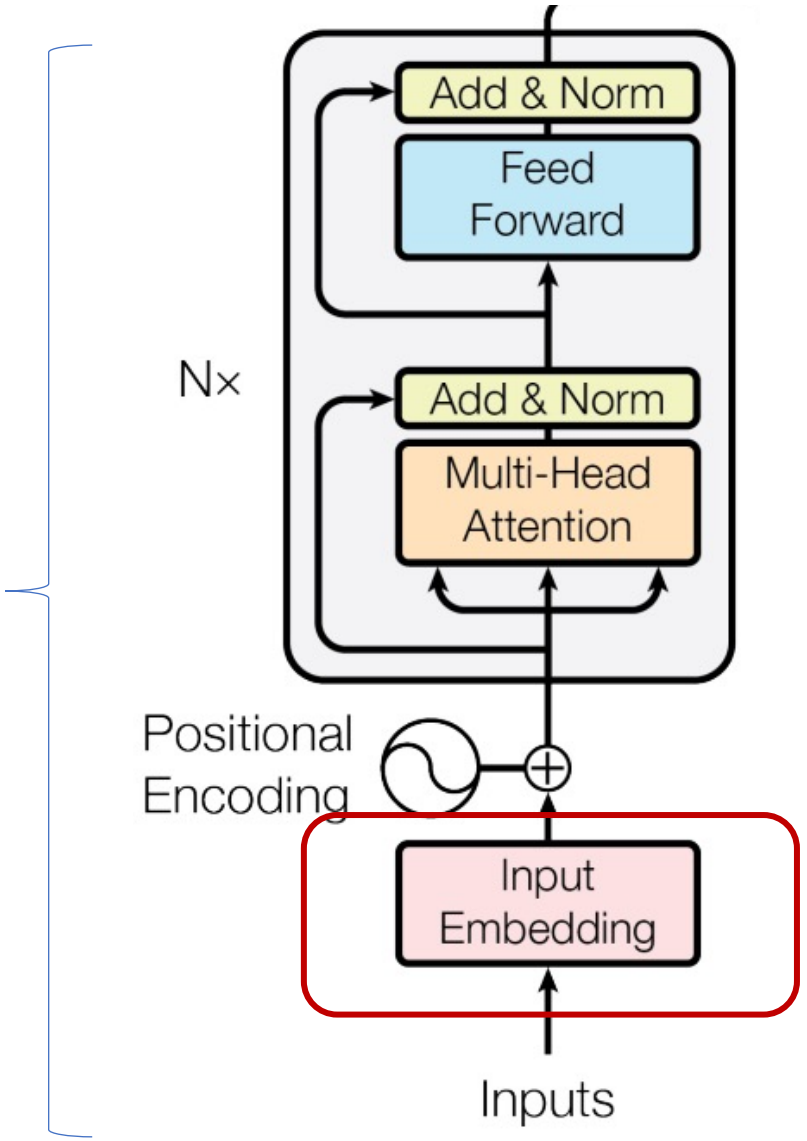


Figure 1: The Transformer - model architecture.

https://doi.org/10.48550/arXiv.1706.03762

Encoder


h_1 h_2 h_3 h_4
Encoder
x_1 x_2 x_3 x_4


Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
N×
Positional Encoding
Input Embedding
Inputs

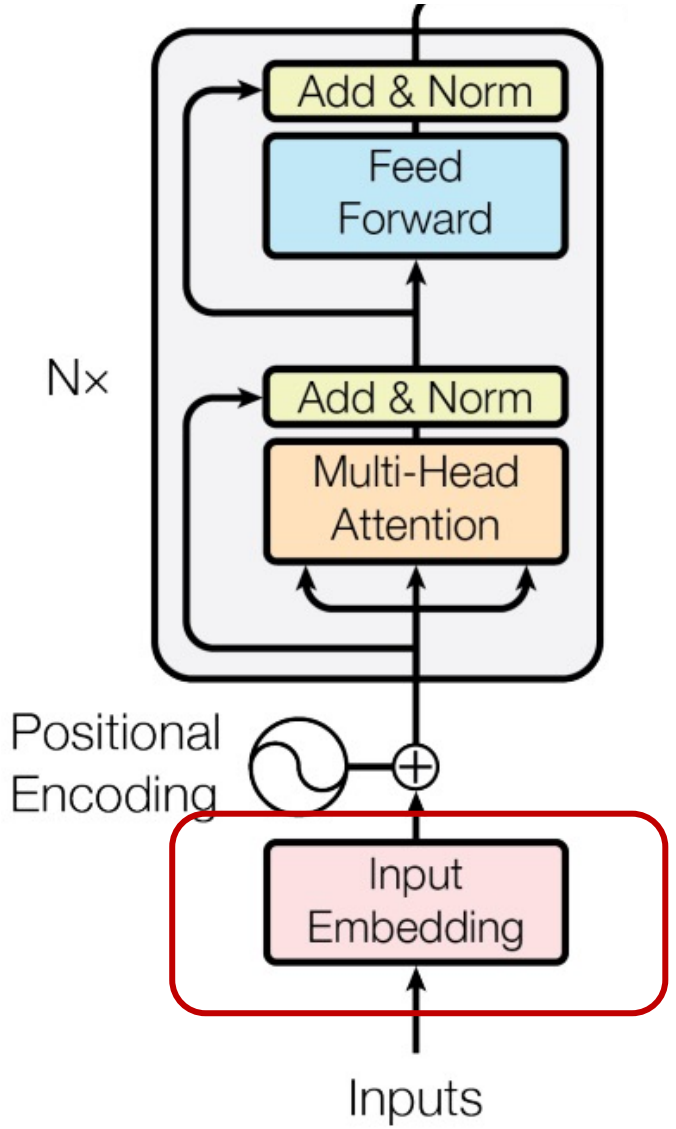
Transformers provide contextual input embeddings

Static word embeddings

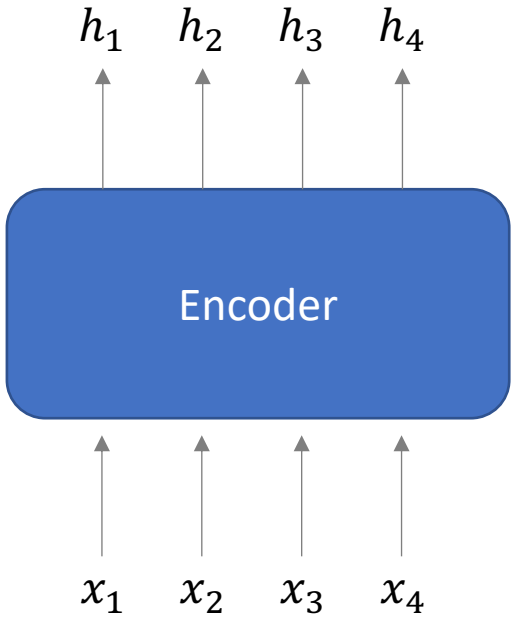
- Fixed vector for each word (independent of context)
- Examples: Word2Vec, Glove, FastText
- Transferable knowledge: reuse pretrained embeddings across tasks and domains

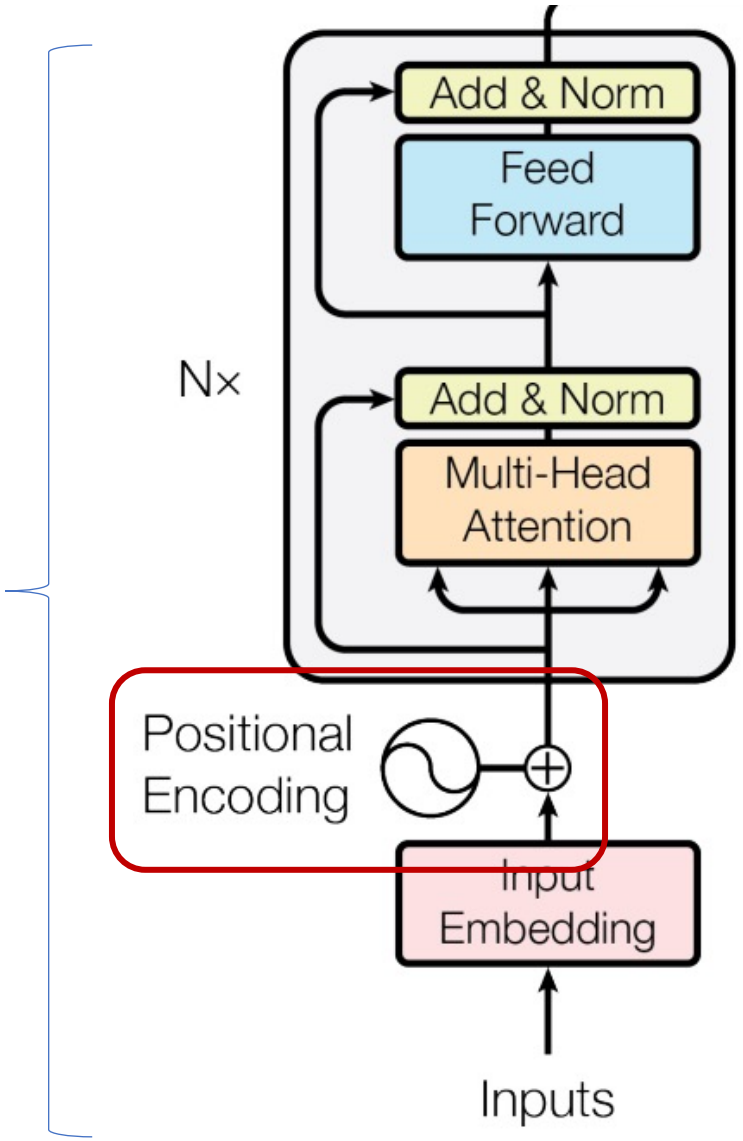
Contextual word embeddings

- Vectors change based on the context in which a word appears

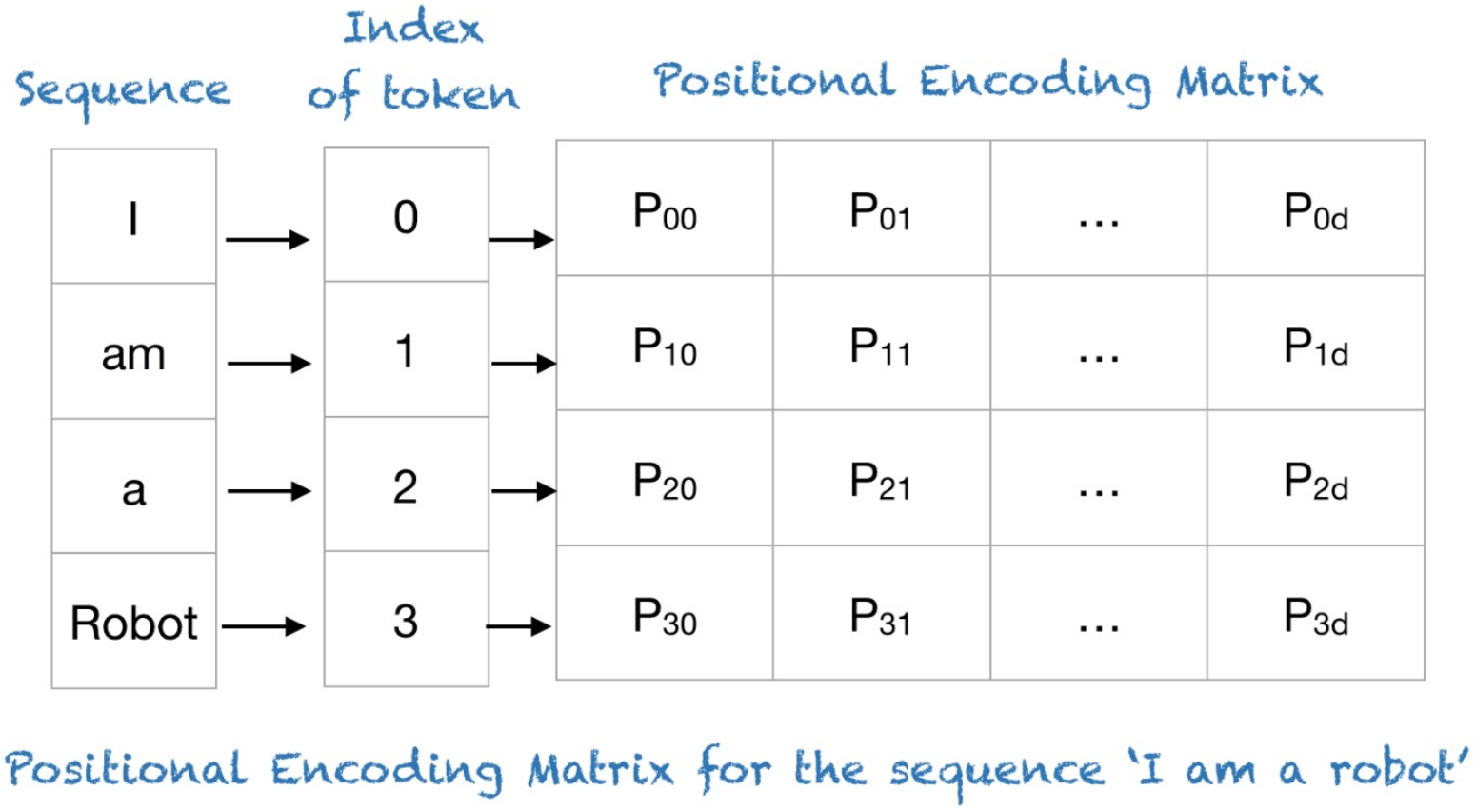


Positional Encoding





Positional Encoding



https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/

Positional Encoding

$$P(k, 2i) = \sin\left(\frac{k}{n^{2i/d}}\right)$$
$$P(k, 2i+1) = \cos\left(\frac{k}{n^{2i/d}}\right)$$

input sequence of length

Here:

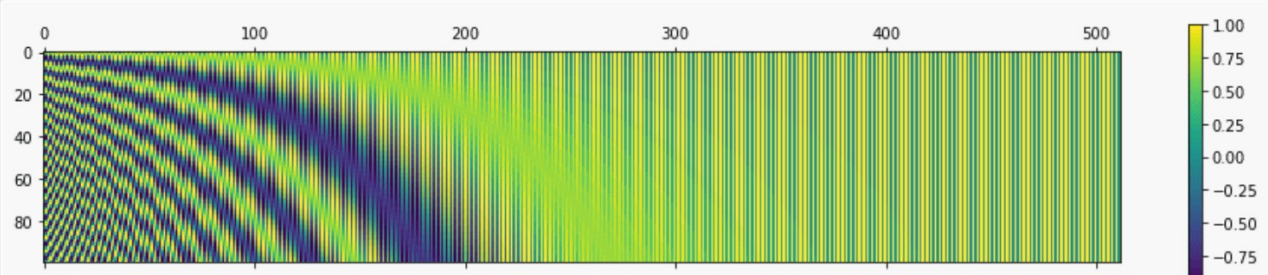
k: Position of an object in the input sequence, $0 \leq k < L/2$

d: Dimension of the output embedding space

$P(k, j)$: Position function for mapping a position k in the input sequence to index (k, j) of the positional matrix

n: User-defined scalar, set to 10,000 by the authors of Attention Is All You Need.

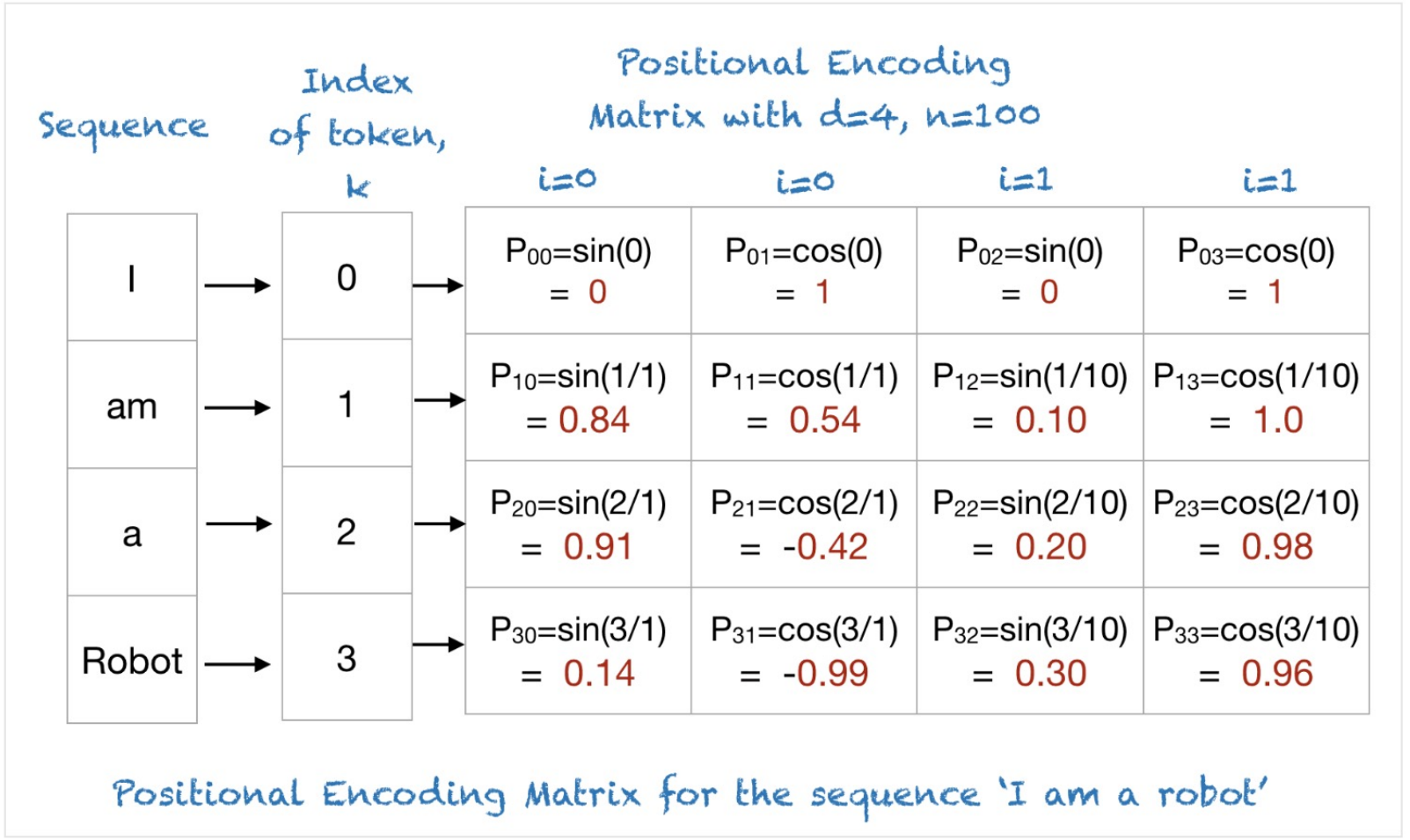
i: Used for mapping to column indices $0 \leq i < d/2$, with a single value of i maps to both sine and cosine functions



The positional encoding matrix for n=10,000, d=512, sequence length=100

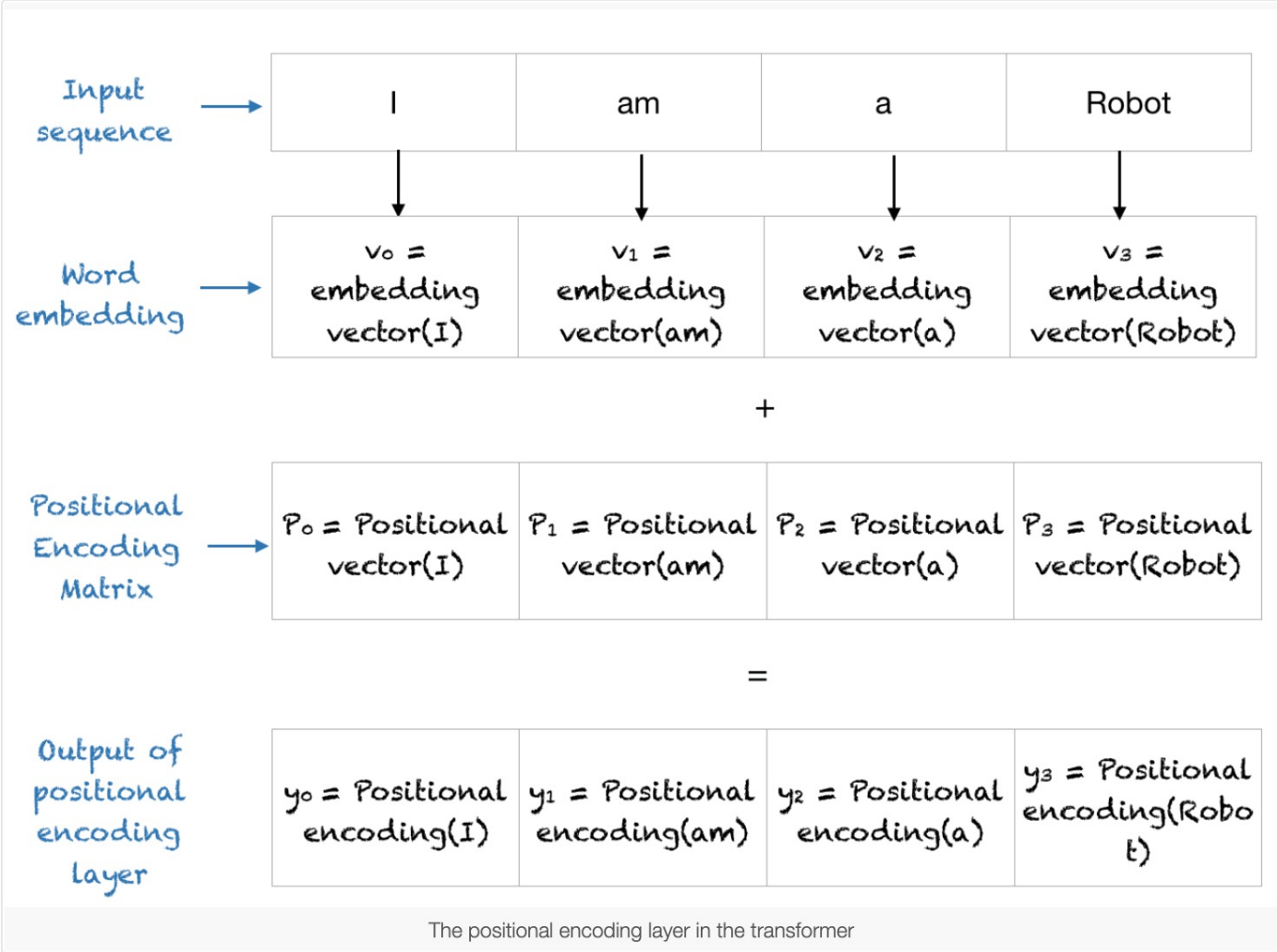
https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/

Positional Encoding



https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/

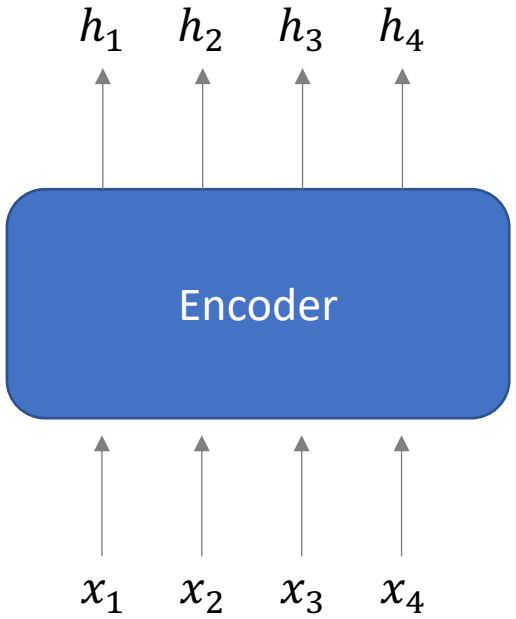
Positional Encoding

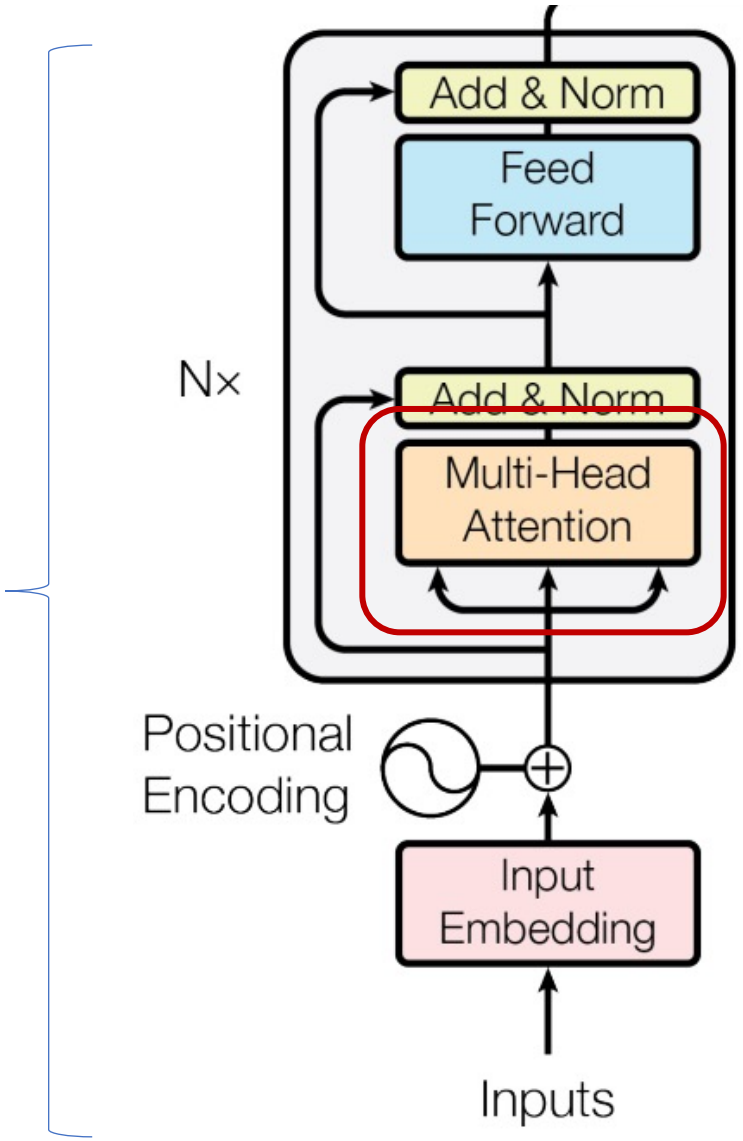


The positional encoding layer in the transformer

https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/

Encoder


h_1 h_2 h_3 h_4
Encoder
x_1 x_2 x_3 x_4


Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
N×
Positional Encoding
Input Embedding
Inputs

Self-attention Mechanism

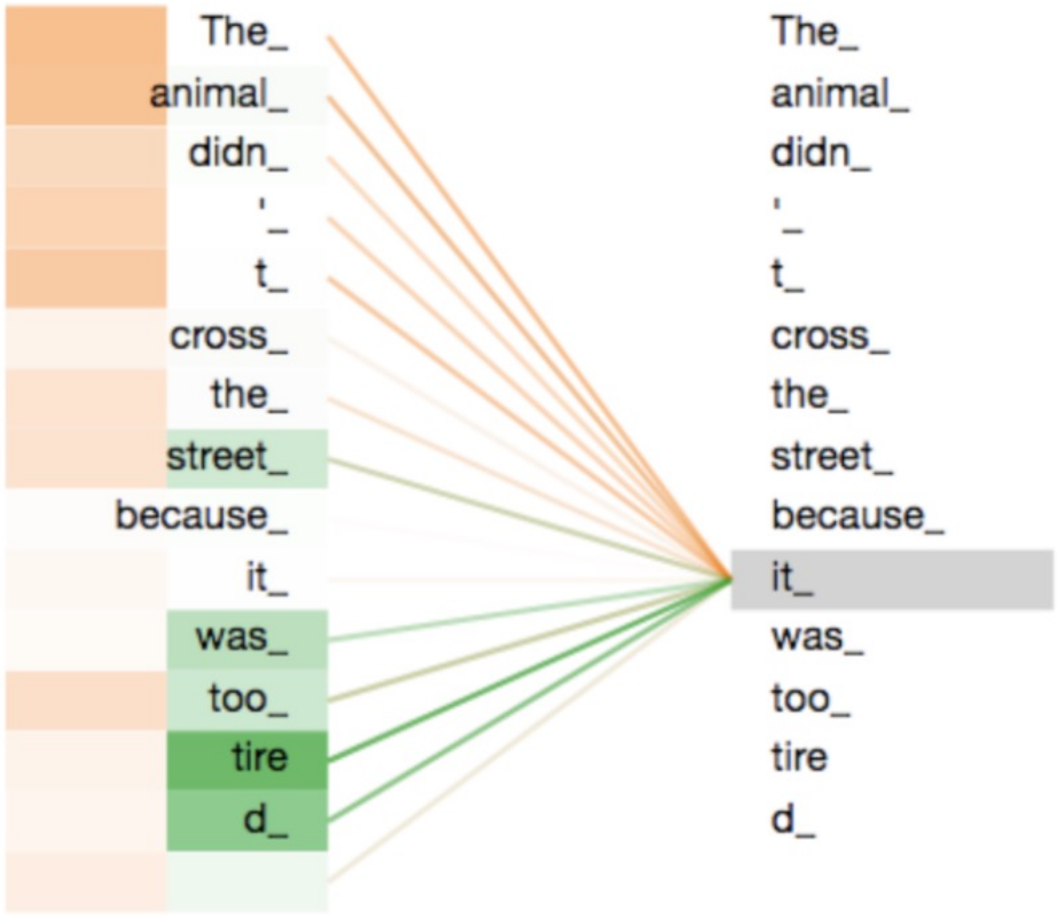
- Each token in the sequence is projected into three vectors called Query (Q), Key (K), and Value (V).
- Self-attention computes how much each token should focus on every other token by matching queries against keys to generate attention weights, and then applying these weights to values.
- This way, each token can gather information from the entire sequence, capturing both local and global context.

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

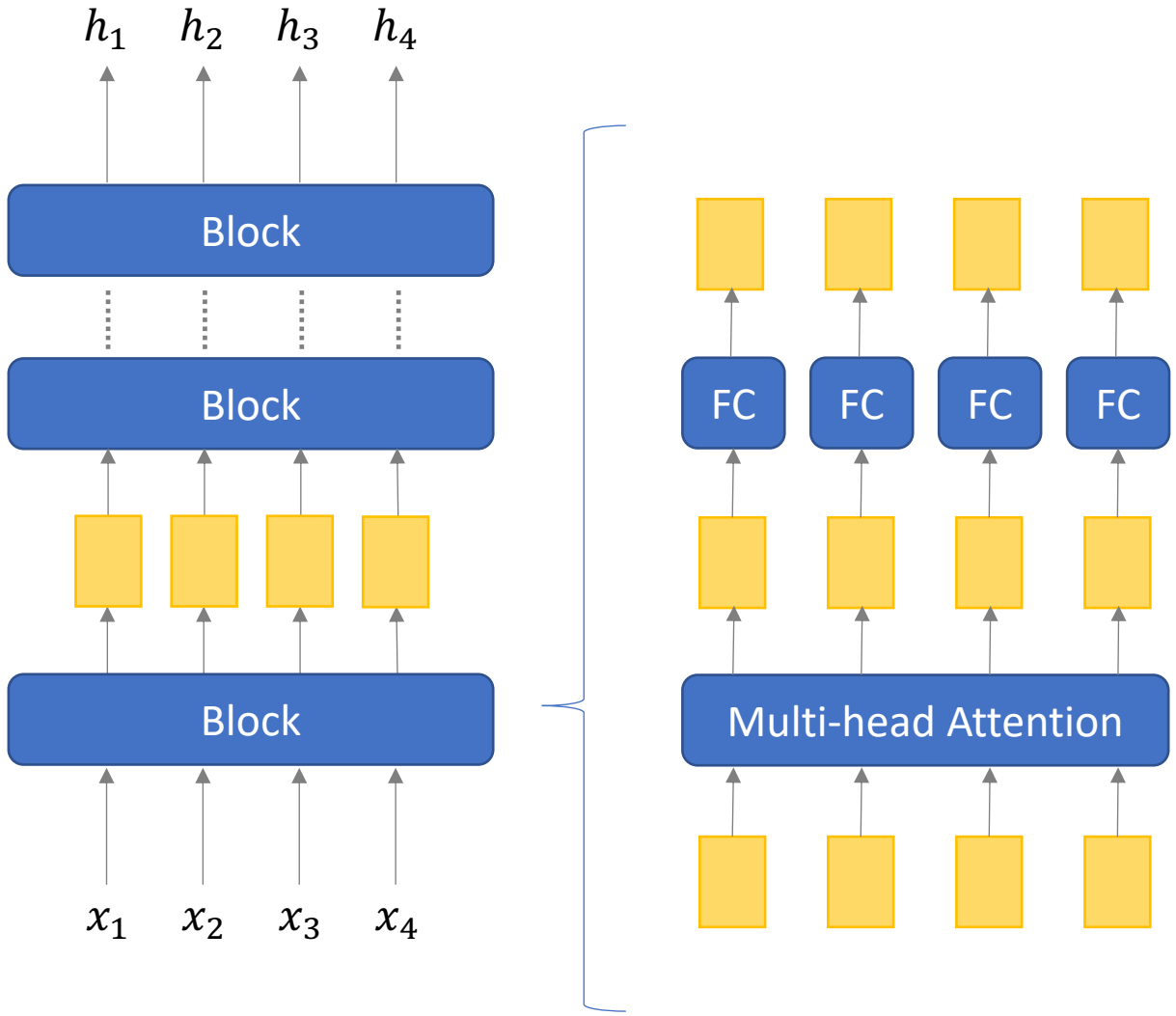
- If a token's Query is very similar to another token's Key, it implies those two tokens should pay attention to each other.
- We divide by the square root of the dimensionality d_k to keep the values stable, preventing extremely large dot products.
- Then we apply a softmax to convert these scores into probability-like attention weights.
- Finally, these weights are applied to the Value vectors, producing a weighted sum that becomes the updated representation for each token.

Multi-head attention

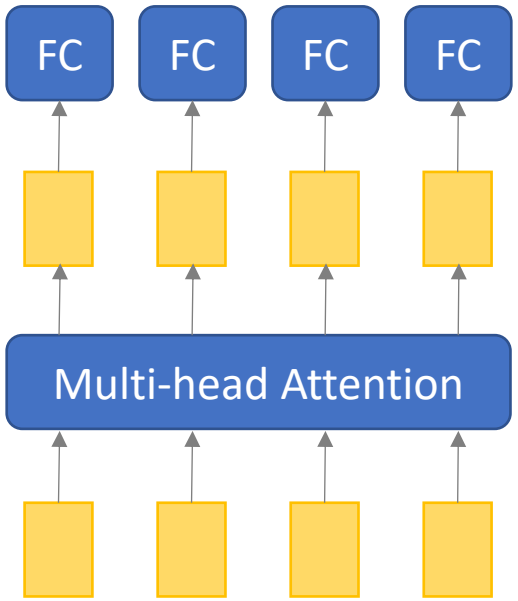
- Multiple attention “heads”
- Each head learns different relationships
- Concatenation + linear projection

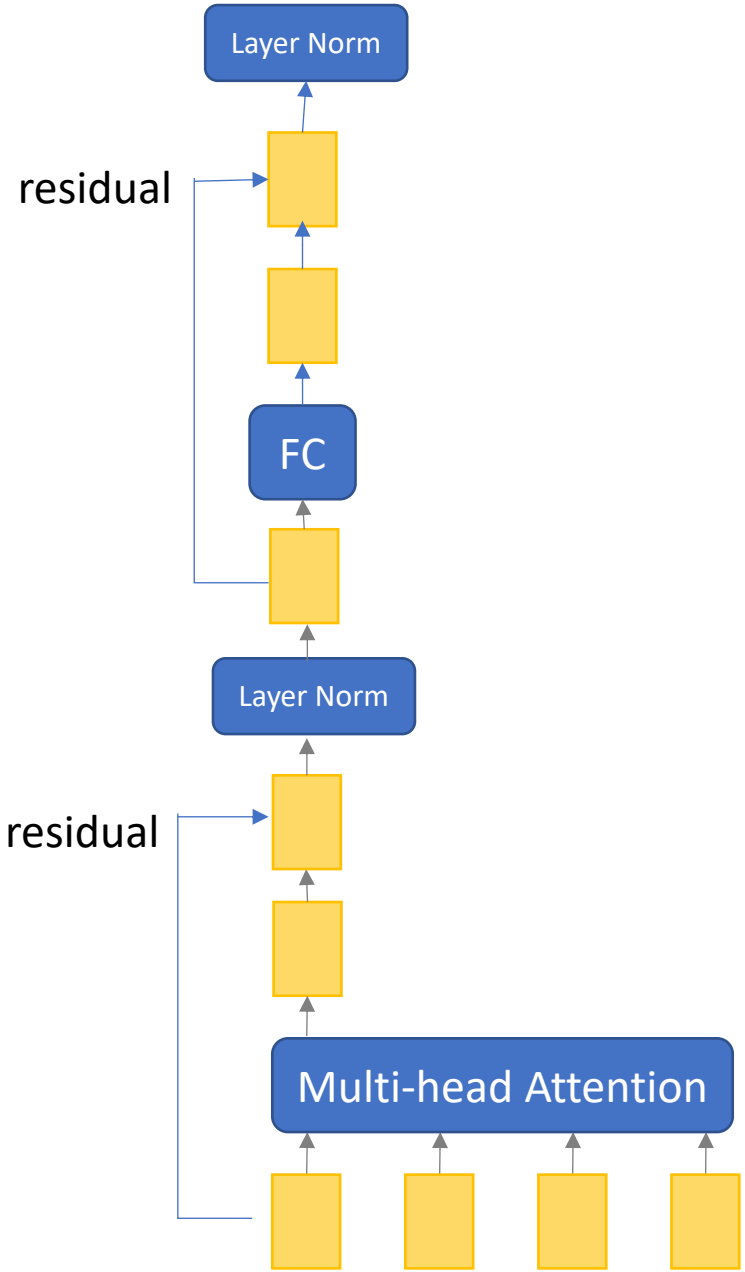


Encoder


h_1 h_2 h_3 h_4
Block
Block
Block
x_1 x_2 x_3 x_4
FC FC FC FC
Multi-head Attention

Encoder


FC
FC
FC
FC
Multi-head Attention

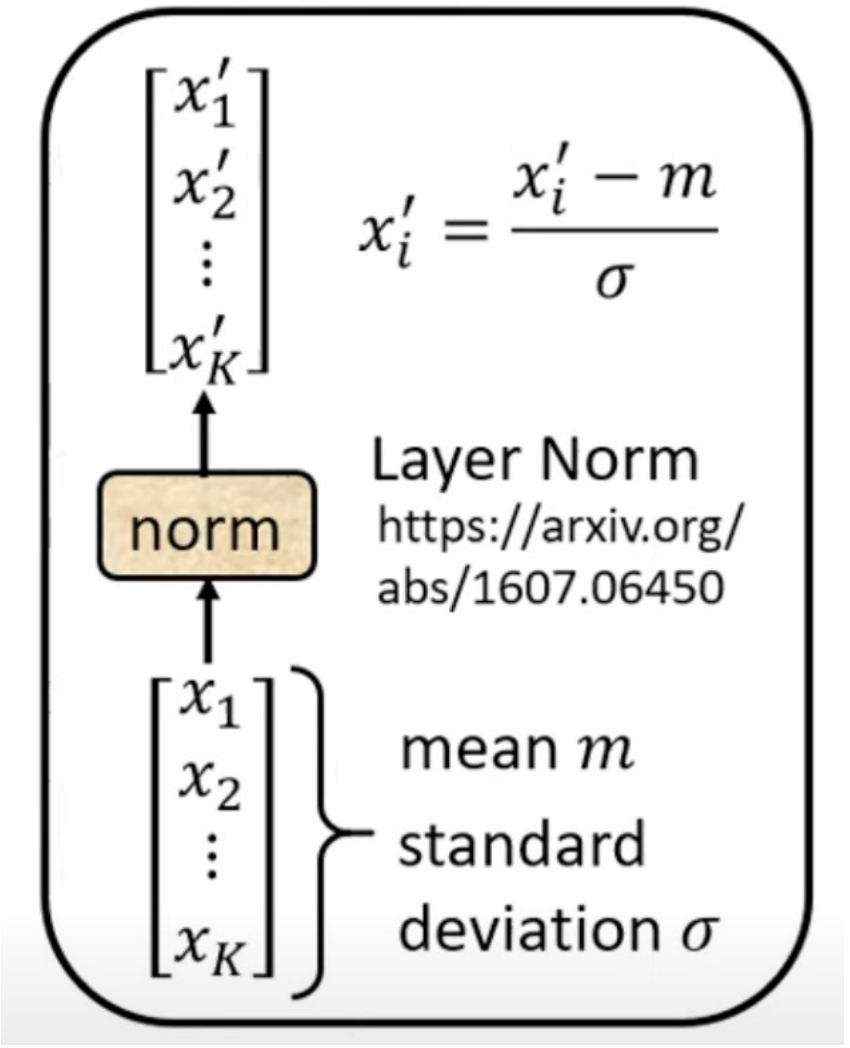

Layer Norm
residual
FC
Layer Norm
residual
Multi-head Attention

Layer Norm

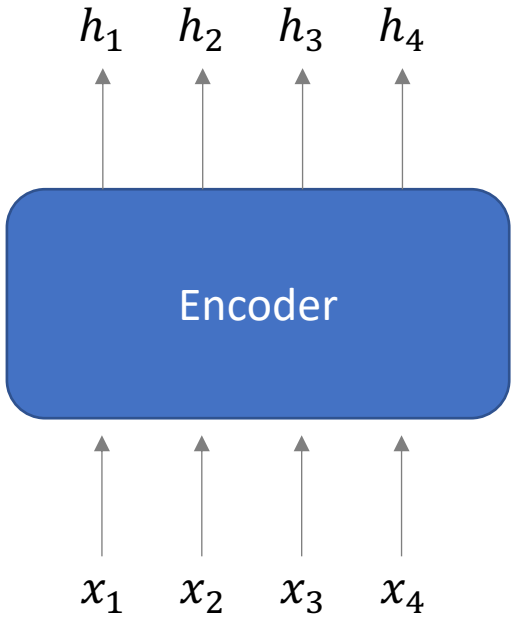
Name	Age	Height
Alice	19	158
Bob	21	172
Claire	22	163
David	20	166

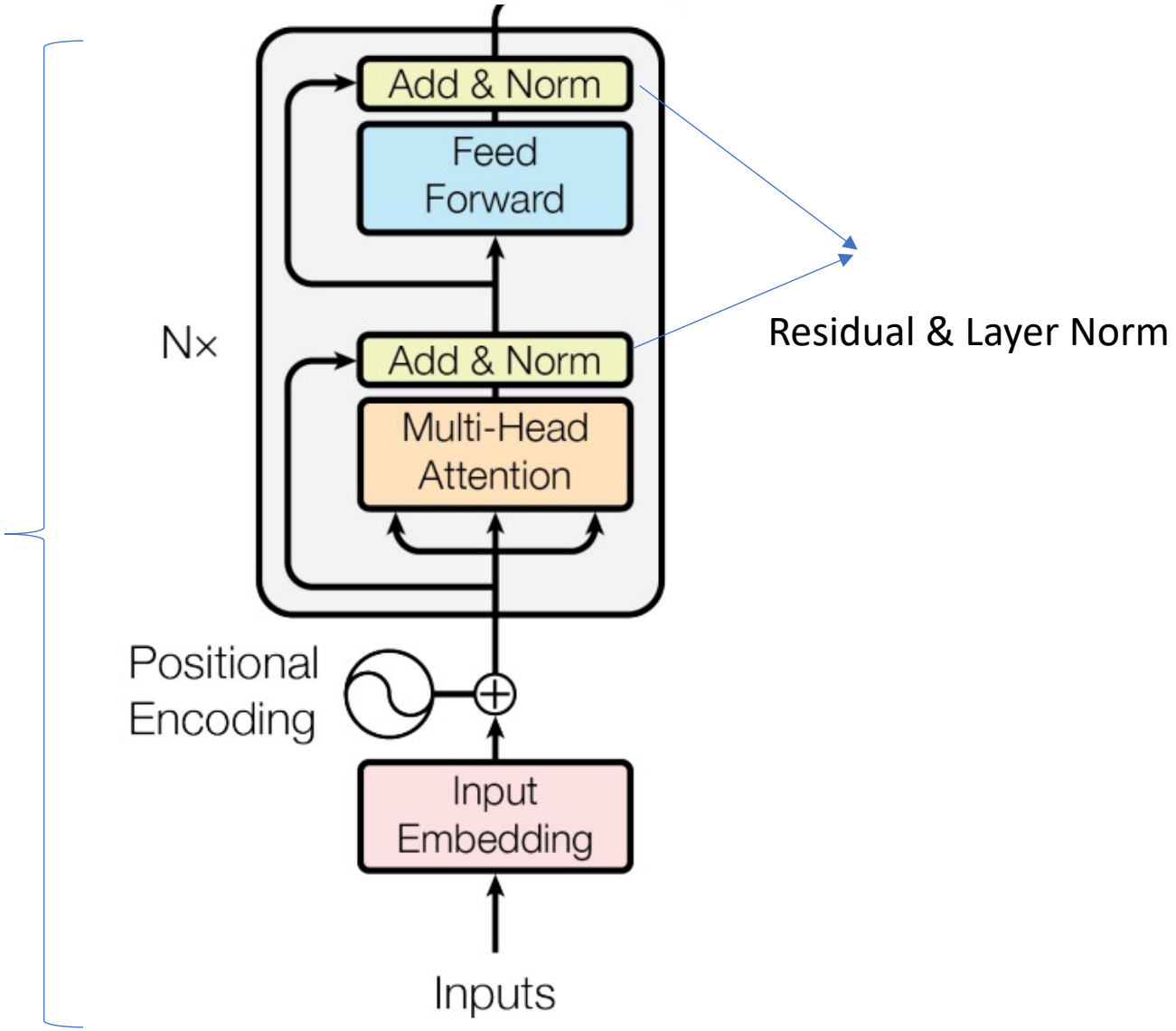
Batch Norm
- Each feature, normalize across all data points in the batch
- For example, for Age, normalize across 19, 21, 22, 20

Layer Norm
- Each data point, normalize across all features
- For example, for Alice, normalize across 19, 158

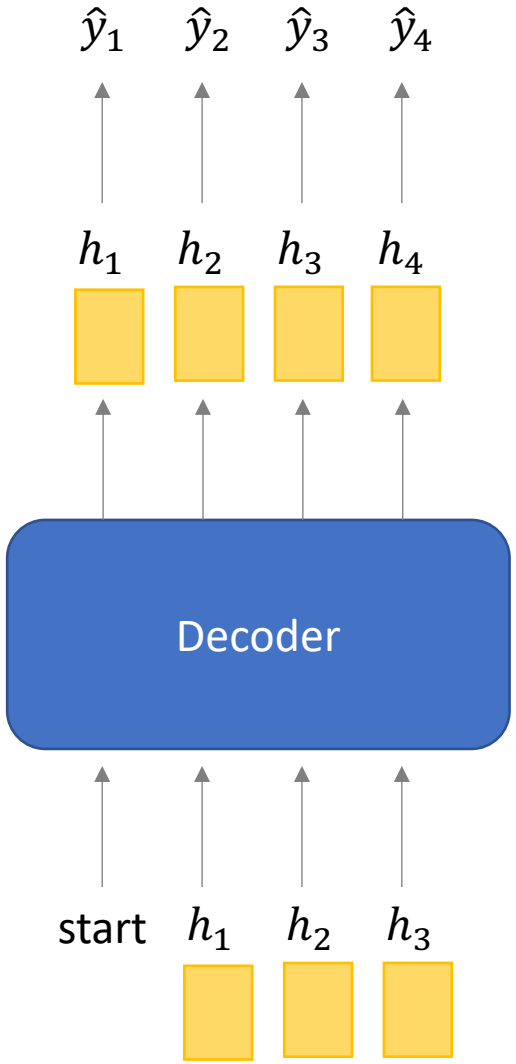


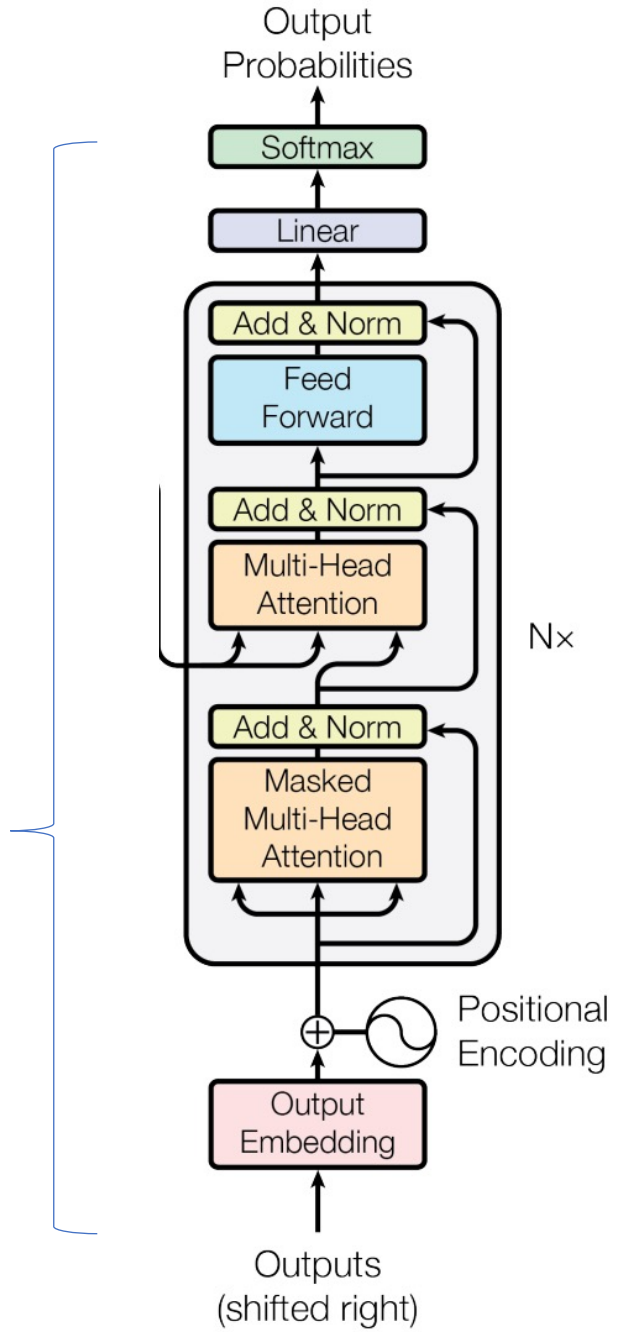
Encoder


h_1 h_2 h_3 h_4
Encoder
x_1 x_2 x_3 x_4

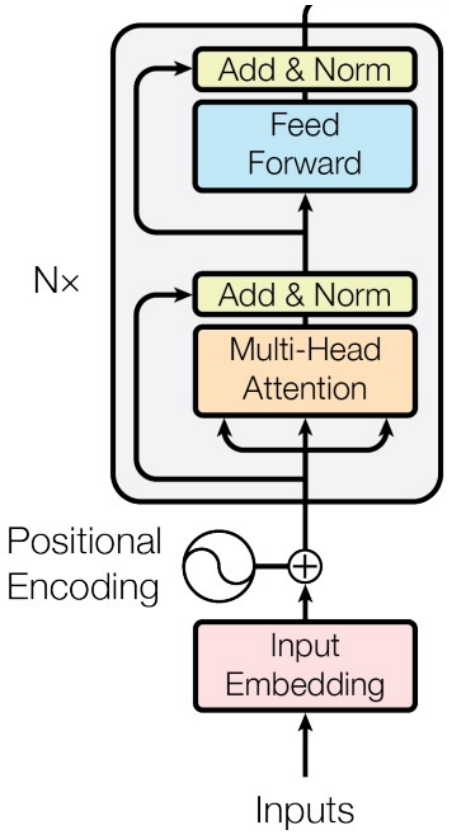

Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
N×
Residual & Layer Norm
Positional Encoding
Input Embedding
Inputs

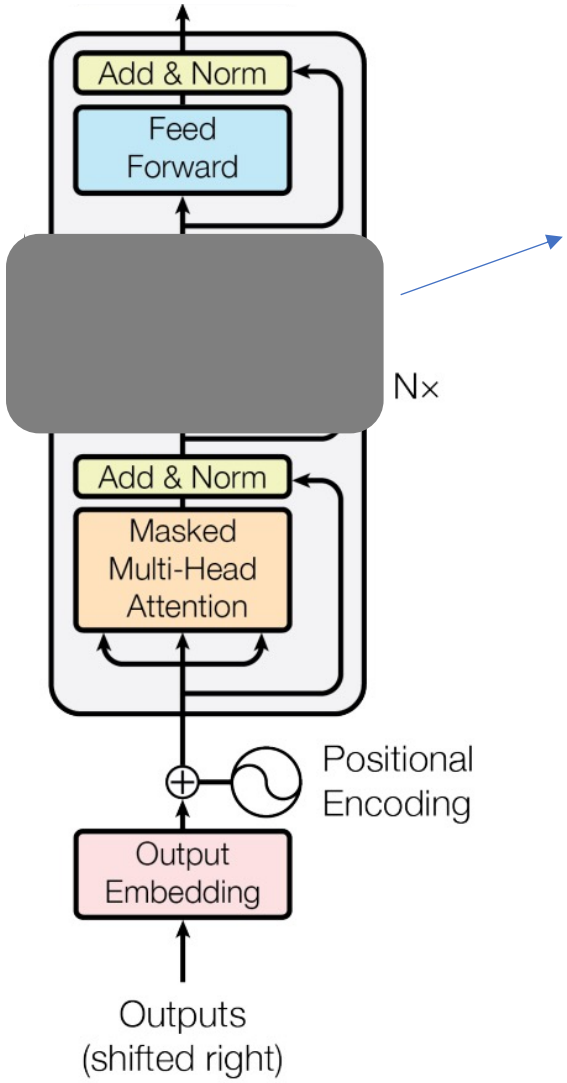
Decoder


$\hat{y}_1$ $\hat{y}_2$ $\hat{y}_3$ $\hat{y}_4$
h_1 h_2 h_3 h_4
Decoder
start h_1 h_2 h_3


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Positional Encoding
Output Embedding
Outputs (shifted right)

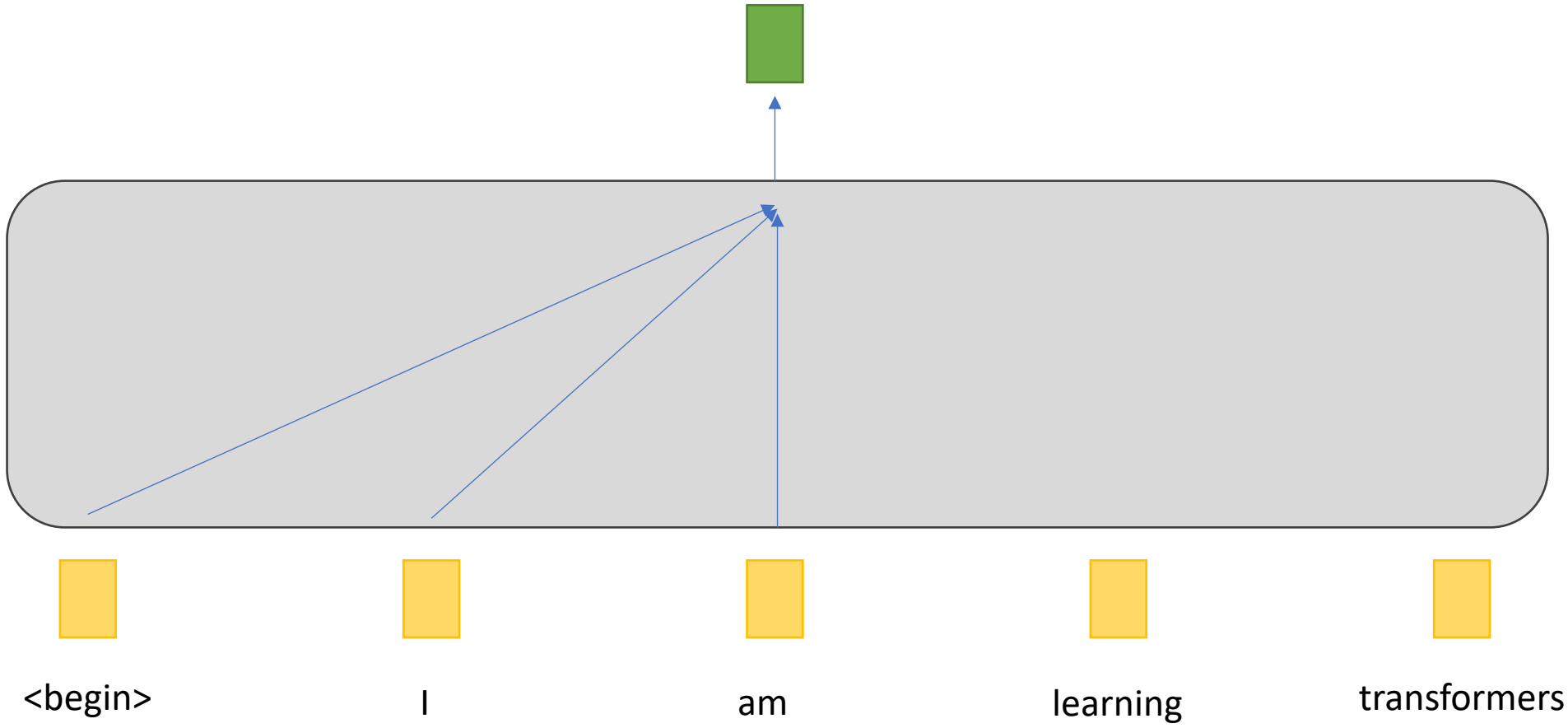
Encoder and Decoder



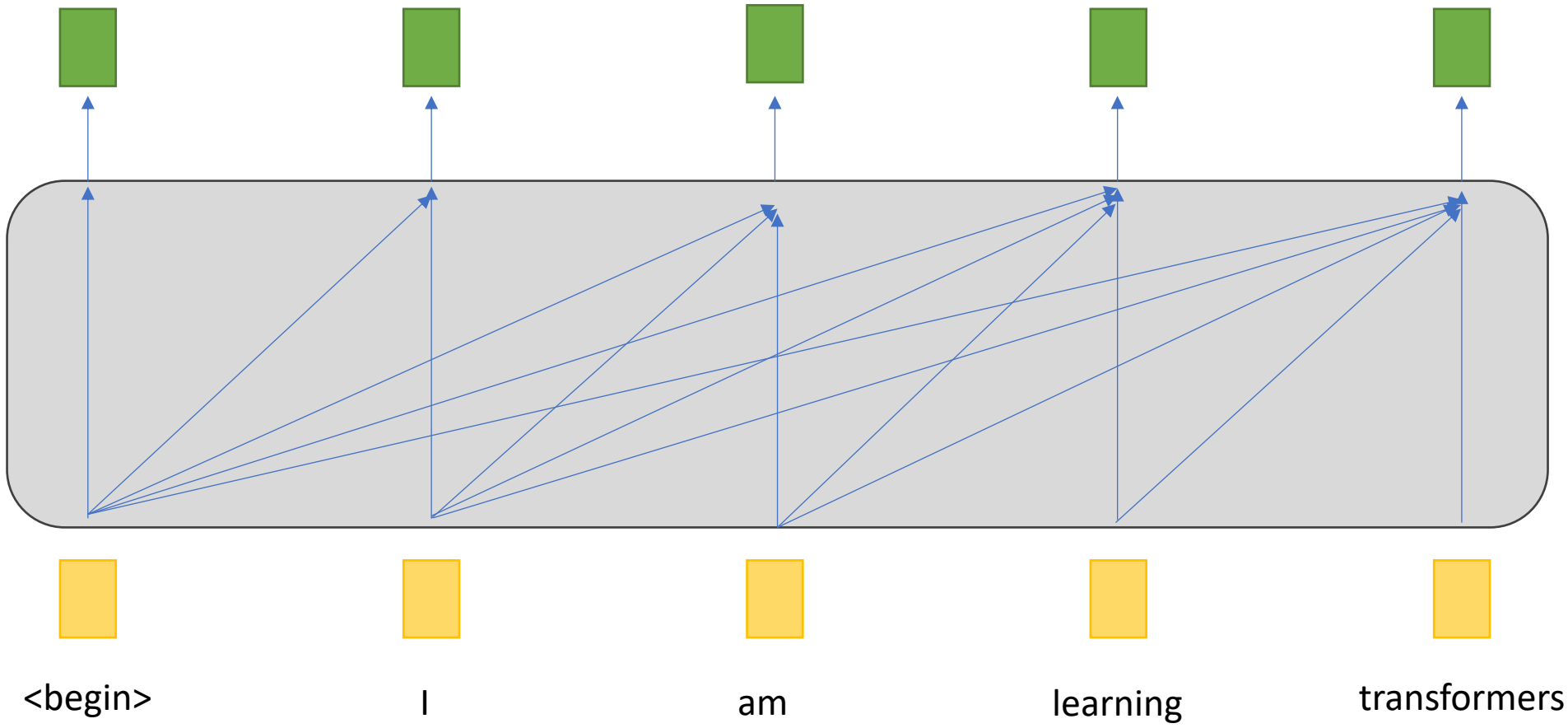


- Let's ignore this for now, will discuss later.
- The rest are very similar to encoder

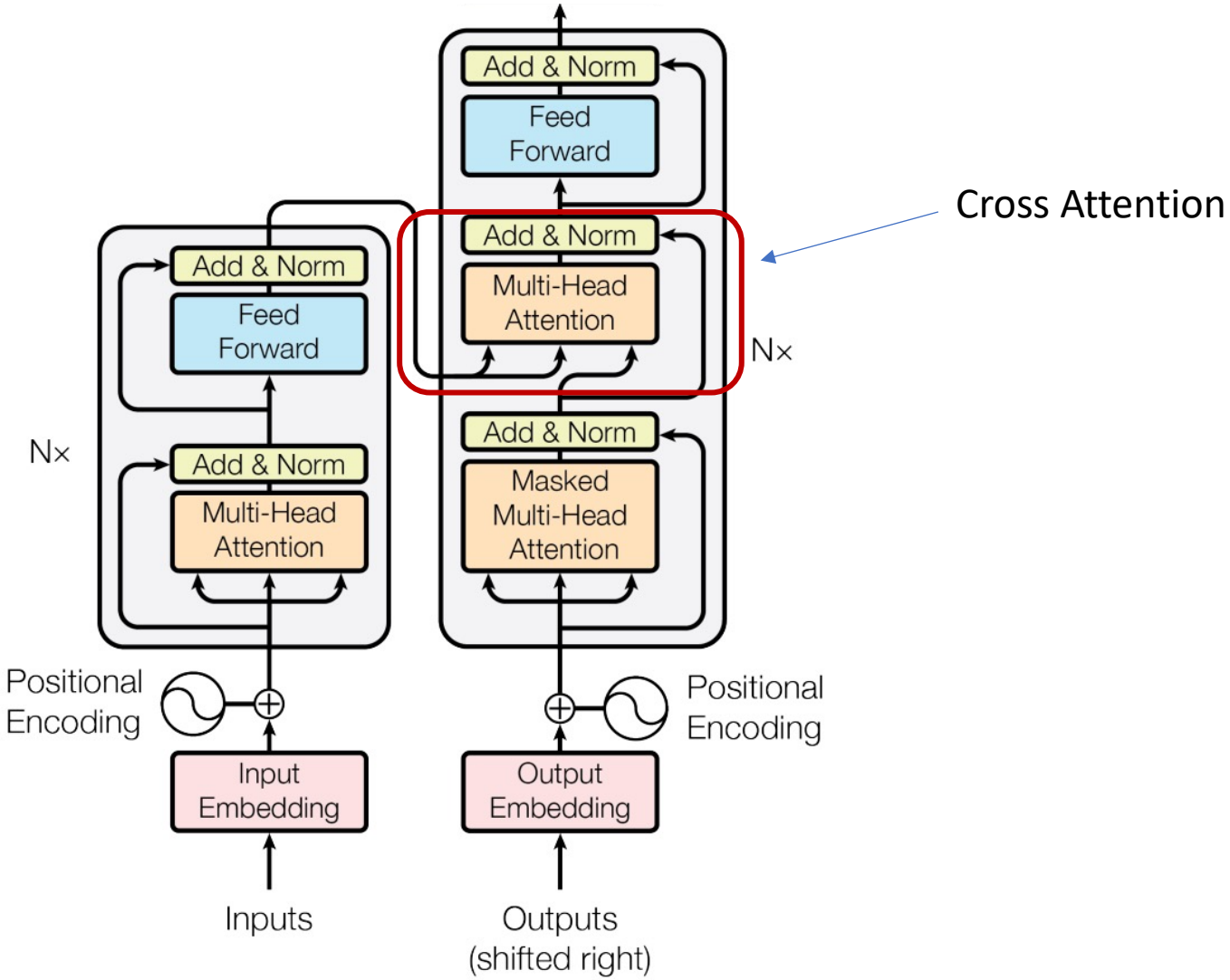
Masked Multi-head Attention



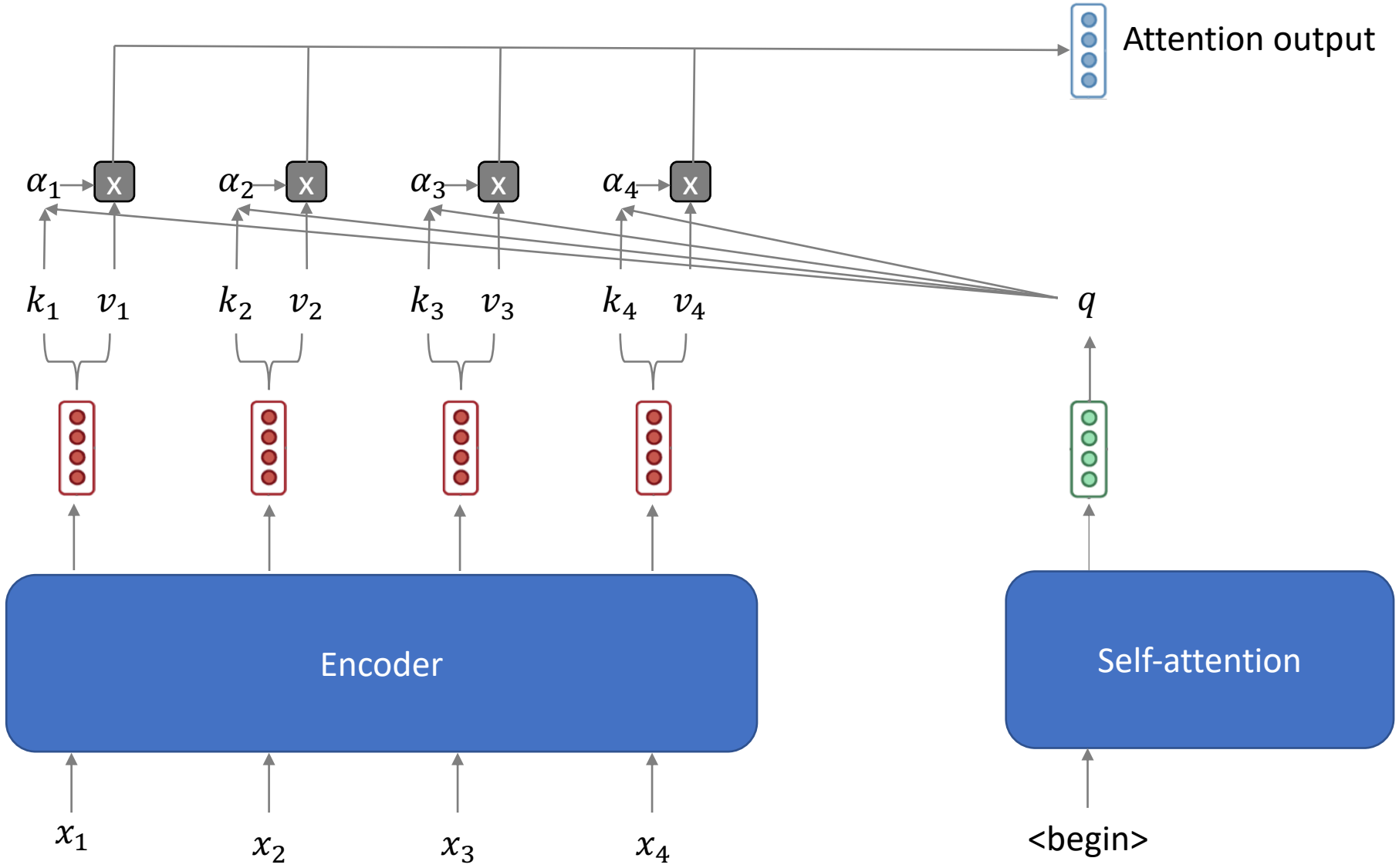
Masked Multi-head Attention



Cross Attention



Cross Attention



Limitations of transformers

- $O(n^2)$ attention complexity
  - For each of the n tokens, we compute attention to every other token (another n), producing an $n \times n$ matrix.
  - As sequence length grows, the number of operations and memory usage increases dramatically, making very long sequences computationally expensive.

- High memory usage for large sequences
  - Storing $n \times n$ attention weights for each head (and each layer) can be huge for long inputs.
  - Transformers often have large numbers of parameters (layers, heads, embedding dimensions), necessitating powerful GPUs/TPUs and large RAM to handle training.

- Often data-hungry
  - Transformers typically excel when trained on extensive corpora; small datasets often lead to overfitting or suboptimal performance.
  - The architecture's capacity is high; without enough data or regularization, the model may struggle to generalize.

- Long-Sequence Handling
  - *Fragmentation of Context*: Because of the quadratic cost, long sequences are sometimes truncated or split, potentially losing global context.

What is self-supervised learning

- **Self-supervised learning** is a technique where a model **creates its own training labels** from unlabeled data, effectively converting an unsupervised problem into a supervised one.
- This allows the model to learn from **large-scale unlabeled data** in a supervised fashion—no manual labeling needed.

Why do we need self-supervised learning

High Cost of Labelled Data

Traditional learning methods heavily rely on labeled data, which is expensive and time-consuming to obtain.

Lengthy Data Preparation Lifecycle

The process of preparing labeled data for machine learning models involves cleaning, filtering, annotating, and restructuring data, making it a lengthy process.

Self-supervised learning addresses these challenges by allowing models to learn complex patterns from unlabeled data efficiently, making it a valuable technique in machine learning for various applications like computer vision and natural language processing

Two types of self-supervised learning

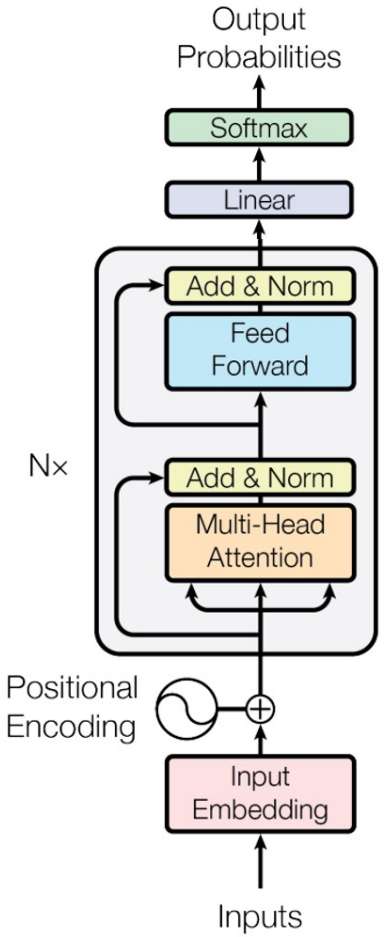
Discriminative Modeling

- Discriminative models focus on learning the decision boundary between classes, aiming to distinguish different categories or classes of data.
- **Strength:**
  - Effective at distinguishing between classes or categories.
  - Can achieve better performance with less data for classification tasks.
- **Weakness:**
  - Do not provide insights into the underlying data distribution.
  - Cannot generate new samples from the learned distribution.
- **BERT**

Generative Modeling

- Generative models aim to learn the underlying probability distribution of the data to generate new samples that resemble the training data. Suitable for tasks like text generation and image synthesis.
- **Strength:**
  - Can generate new samples resembling the training data.
  - Provide insights into the underlying structure and distribution of the data.
  - Useful for tasks beyond classification like data generation and imputing missing data.
- **Weakness:**
  - Typically have more parameters and are computationally expensive.
  - Might require more data to converge to a meaningful model.
- **GPT**

BERT – Bidirectional Encoder Representation from Transformers



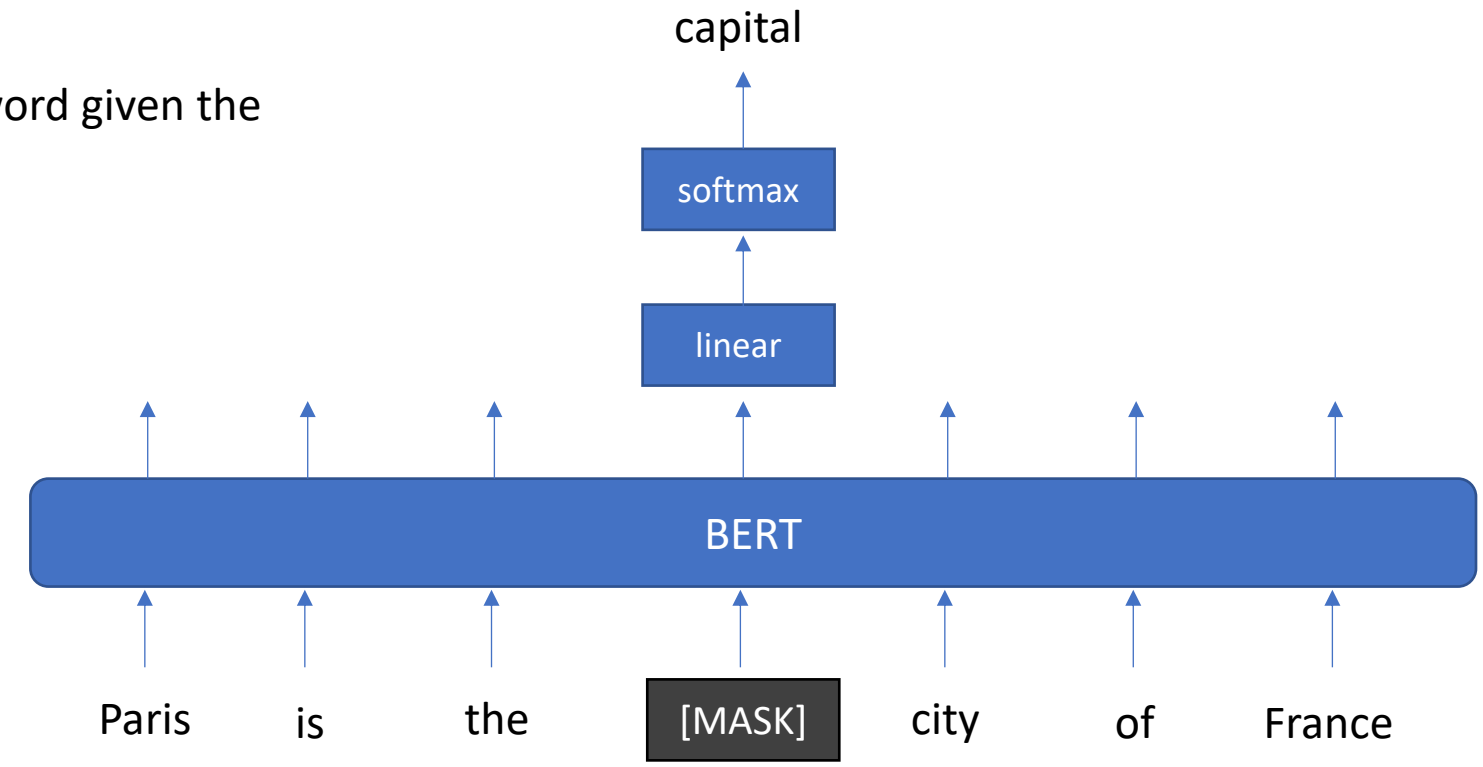
Encoders-only: BERT is made up of layers of encoders of the Transformers model

- $\text{BERT}_{\text{BASE}}$
  - 12 encoder layers
  - 12 attention heads
  - 110M parameters

- $\text{BERT}_{\text{LARGE}}$
  - 24 encoder layers
  - 16 attention heads
  - 340M parameters

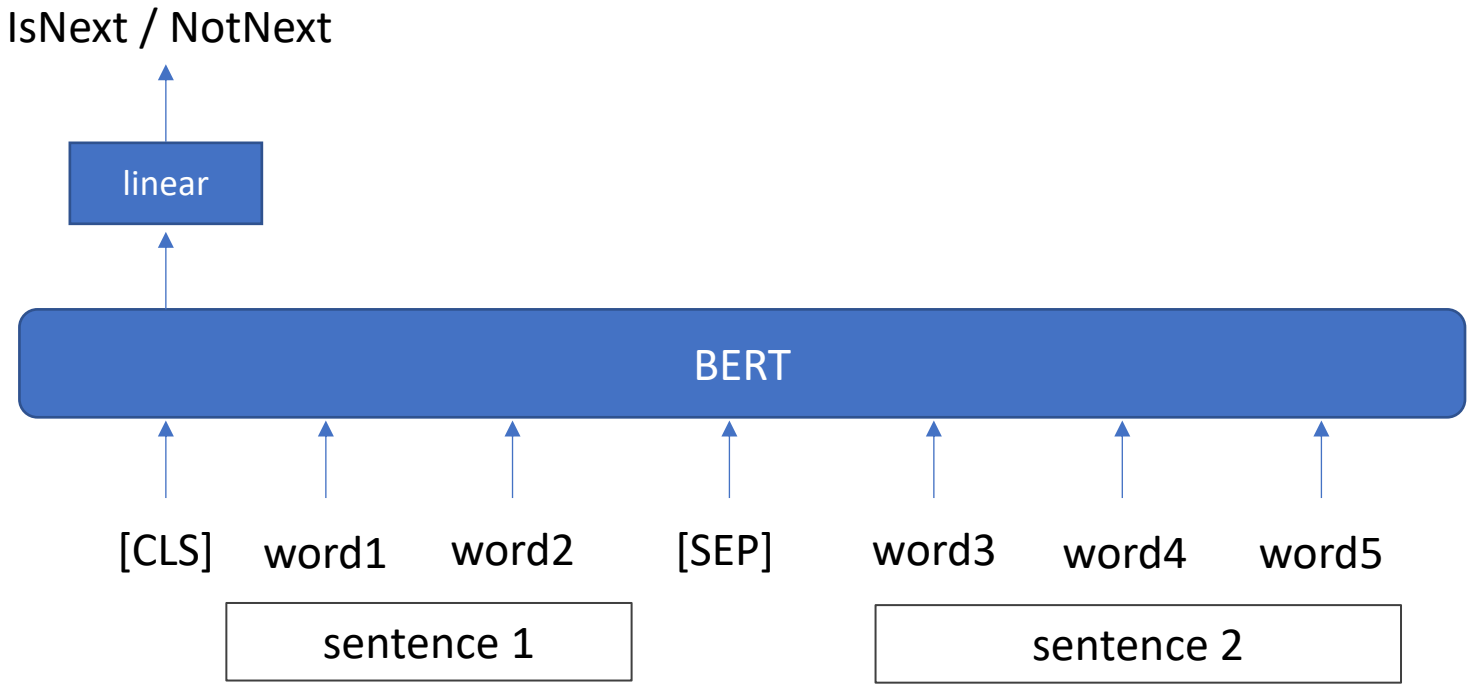
Training task 1: Masked Language Model (MLM)

- Randomly select 15% of words in a sentence and replaced by [MASK].
- BERT is asked to predict the masked word given the left and right context.



Training task 2: Next Sentence Prediction

- Learning relationships between sentences
- 50% of the time, select the actual next sentence
- 50% of the time, select a random sentence from the text



Finetune BERT to solve problems

Pretrain

BERT

- Masked Word Prediction
- Next Sentence Prediction

Self-supervised

3B Words

Finetune

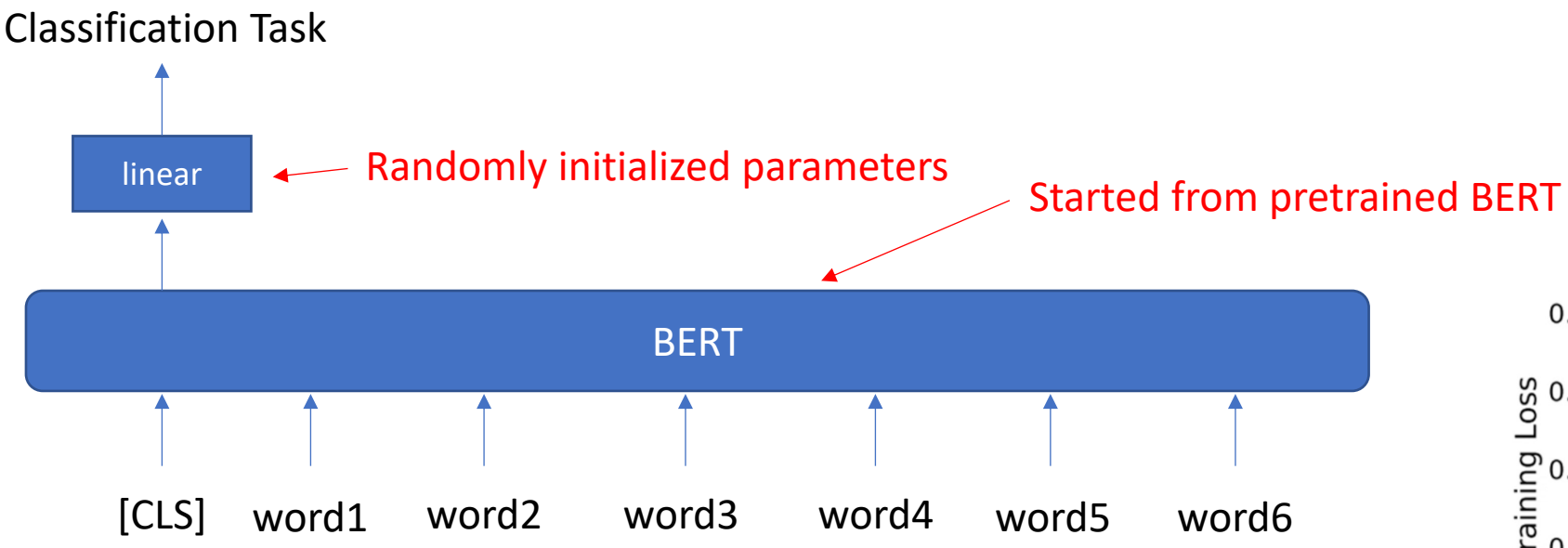
Task of interest

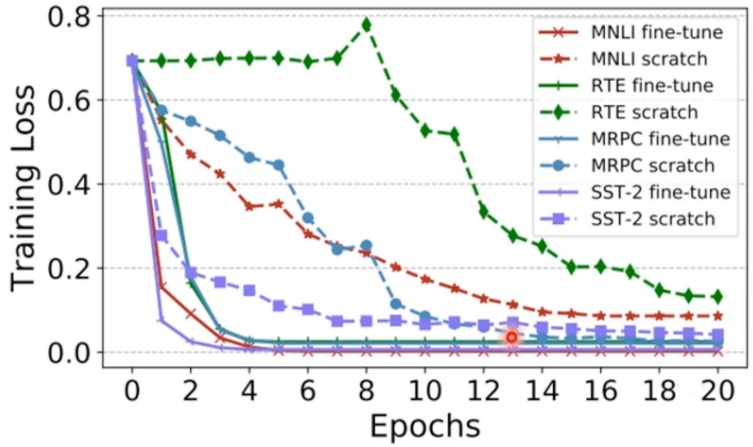
Supervised

3k Examples

- Can achieve better performance with less labelled data

How to finetune





Source of image: https://arxiv.org/abs/1908.05620

Which tasks is BERT good at

Tasks requiring a deep understanding of bidirectional context

- Sentiment analysis
- Text classification
- Neural machine translation
- Named entity recognition (NER)
- Summarization

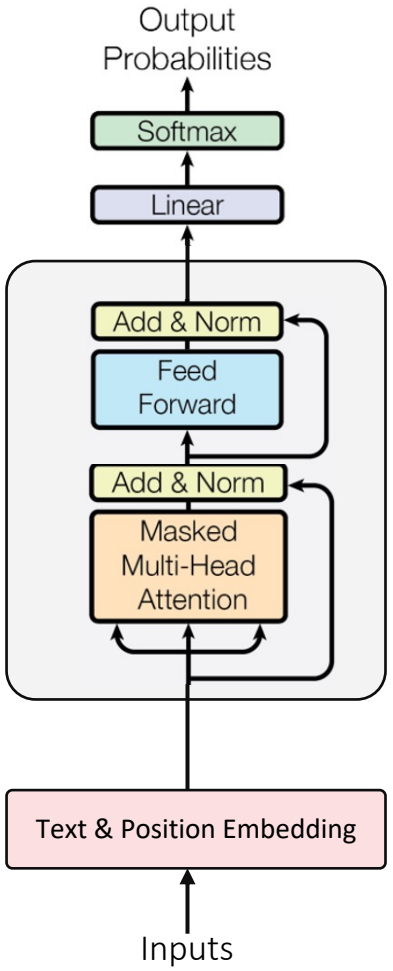
Potential issue with finetuning

- Still highly depending on labelled data
- Usually perform badly on related tasks not directly finetuned on

GPT is very ambitious to try to avoid the necessity for fine-tuning on each specific task and leverage on prompt engineering and zero-shot learning

- prompt engineering: providing detailed and specific inputs to the model
- zero-shot learning: expecting models to perform tasks without explicit training on them

GPT – Generative Pre-trained Transformer



Decoders-only: GPT is made up of layers of decoders of the Transformers model, where the input data is directly fed into the decoder without prior transformation by an encoder

- GPT2
  - Released by OpenAI in 2019
  - 1.5 billion parameters
  - Training on WebText (8 million documents from outbound Reddit links with a certain threshold of upvotes)
- GPT3
  - Released by OpenAI in 2020
  - 175 billion parameters
  - Training Corpus: Massive (Common Crawl, WebText, etc.)
- GPT4
  - Released by OpenAI in 2023
  - Rumored to contain 1.76 trillion parameters

Training task: next token prediction

- Uni-directional
- Autoregressive Language Modeling
- Causal language model
- Look back at previous words to predict the next token
- Trained specifically for text generation

How to use GPT

- Handle specific tasks with **prompts**
- Zero-shot learning: perform a specific task given an instruction and input
  - Considered a very hard problem, even for humans

Prompt	Classify the text into positive, neutral or negative: Text: This movie is awesome! Classification:
Response	Positive

Few-shot learning

- One-shot learning: one example is included in the context
- Few-shot learning: multiple examples are included in the context

No parameters in GPT are updated

Prompt	Classify the text into positive, neutral or negative: Text: This movie is awesome! Classification: Positive Text: Lot of silly plot holes in the film. Classification: Negative Text: This movie was obscenely obvious and predictable. Classification: Negative Text: This movie has great style and fantastic visuals! Classification:
Response	Positive

Which tasks is GPT good at

Tasks requiring text generation and context-based responses, such as

- Text generation, where the model generates coherent and contextually relevant text given a prompt or initial input.
- Dialogue systems and chatbots, where GPT generates responses based on the conversation history. GPT's ability to produce contextually relevant responses makes it suitable for creating engaging conversational agents.

The **trade-off** is that GPT does not see future context to the right, unlike BERT. This makes GPT exceptionally good at generating coherent text sequences but less direct for tasks requiring deep bidirectional understanding (though it can still perform well with large scale).

BERT vs. GPT

Architecture: Encoder (BERT) vs. Decoder (GPT)

Training: MLM + NSP vs. Autoregressive LM

Strengths: BERT (understanding) vs. GPT (generation)

Use Cases: BERT (classification) vs. GPT (generation)

BERT Variants

RoBERTa

Trained by Facebook, removed the Next Sentence Prediction task and trained on more data, achieving improved performance.

DistilBERT

Uses knowledge distillation to create a smaller, faster model while retaining much of BERT's accuracy.

ALBERT

Shares parameters across layers and factorizes embeddings to reduce memory usage.

Others

Domain-specific models like **SciBERT**, **BioBERT**, and **FinBERT** adapt BERT for specialized scientific, biomedical, or financial vocabularies, respectively.

RoBERTa (*Robustly Optimized BERT Pretraining Approach)*

Architecture: *Same Transformer encoder* structure as BERT (same number of heads, layers, etc. for a given “base” or “large” variant).

Removal of Next Sentence Prediction (NSP)

- BERT used NSP to help the model understand relationships between sentence pairs.
- **RoBERTa** found NSP **unnecessary** (and potentially suboptimal), so it **omits this task** and relies entirely on Masked Language Modeling (MLM).

Dynamic Masking

- BERT often used a **static masking** pattern for each training example.
- **RoBERTa** uses **dynamic masking**, which re-samples the masked positions during training.
- **Benefit**: The model sees more diverse training examples and learns more robust contextual representations.

Larger Datasets & Longer Training

- RoBERTa is trained on a **significantly larger corpus** (160GB of text vs. the original BERT’s ~16GB).
- Emphasizes **more training iterations** and heavier data augmentation techniques.

Key Takeaway: If you’re already using BERT, switching to RoBERTa can be a straightforward upgrade if you have **sufficient resources** (GPU memory for larger batches, etc.).

Performance: In practice, you may notice a ~1-3% improvement over BERT in many tasks, though results can vary by domain and data size.

Knowledge Distillation

Transfer knowledge from (larger) teacher model to (smaller) student model

- Process of transferring knowledge from a large, high-capacity **teacher model** to a smaller, lighter **student model**.
- Reduce model size, inference time, and computational demands while retaining much of the teacher's performance.

Typical process:

- **Teacher model** produces soft outputs for each input.
- **Student model** is trained to mimic these outputs.

DistilBERT: Lightweight BERT Through Distillation

- **Definition**: A **smaller, faster** variant of BERT obtained through a **distillation** process, aiming to retain most of BERT's performance while reducing size and inference time.
- **Creators**: Developed by the team at Hugging Face.
- **Key Idea**: Use a large, pretrained BERT as a "teacher" and transfer its "knowledge" to a "student" model with fewer parameters.

Fewer Layers

- DistilBERT typically has **6 Transformer layers** instead of 12 (BERT Base).
- This drastically reduces the parameter count while preserving core Transformer functionality.

Maintaining Performance

- DistilBERT retains about **97%** of BERT's language understanding capabilities on GLUE benchmark tasks, despite a significant reduction in size.

Speed & Efficiency

- **~60% faster** inference compared to BERT Base.
- Uses **~40% fewer** parameters, cutting down memory usage and storage requirements.

ALBERT (A Lite BERT)

- **Acronym**: *A Lite BERT for Self-supervised Learning of Language Representations*
- **Creators**: Developed by Google Research.
- **Key Goal**: Reduce the **memory footprint** and **training time** of BERT while maintaining (or even improving) performance.

Factorized Embedding Parameterization

- **BERT** has large word embeddings (e.g., 768 dimensions in BERT Base) directly tied to the size of the hidden layer.
- **ALBERT** decouples the embedding size (e.g., 128 or 256) from the hidden layer size (e.g., 768 or 1024), dramatically reducing the number of embedding parameters.
- **Result**: Fewer parameters in the embedding matrix, leading to lower memory usage and faster training.

Cross-Layer Parameter Sharing

- In **BERT**, each Transformer layer has its own set of parameters.
- **ALBERT** shares parameters across **all** (or groups of) layers—specifically attention and feed-forward sub-layers.
- **Benefit**: This heavy reuse of parameters drastically reduces the overall model size.

Sentence-Order Prediction (SOP)

- **BERT** used Next Sentence Prediction (NSP) to help the model learn inter-sentence relationships.
- **ALBERT** replaces NSP with **Sentence-Order Prediction** (SOP), which checks if two consecutive segments are in the correct order or swapped.
- **Claim**: SOP captures inter-sentence coherence more effectively than NSP.

Fewer Parameters:

- ALBERT Base can have **12× fewer** parameters than BERT Base, depending on configuration.

Comparable/Better Accuracy:

- On benchmarks like **GLUE** and **SQuAD**, ALBERT often performs on par with or slightly **better** than BERT with far fewer parameters.

FinBERT

- **Definition**: A variant of BERT that is **pretrained specifically on financial text**, aiming to capture domain-specific language patterns and terminology.
- **Domain**: Finance and economics, including news articles, financial reports (e.g., SEC filings), analyst reports, and more.
- **Key Goal**: Provide **improved performance** on financial NLP tasks (sentiment analysis on financial statements, event extraction, company-related news classification, etc.).

Domain-Specific Corpus

- **FinBERT** is trained on **financial texts**, unlike vanilla BERT trained on general texts (Wikipedia, BookCorpus).
- It better understands jargon, abbreviations, and context unique to finance (e.g., “EPS,” “IPO,” “bullish,” “bearish”).

Performance

- **Accuracy**: Achieves **2–5.7% higher F1-score** than generic BERT in financial tasks.

Gradual unfreezing

Definition

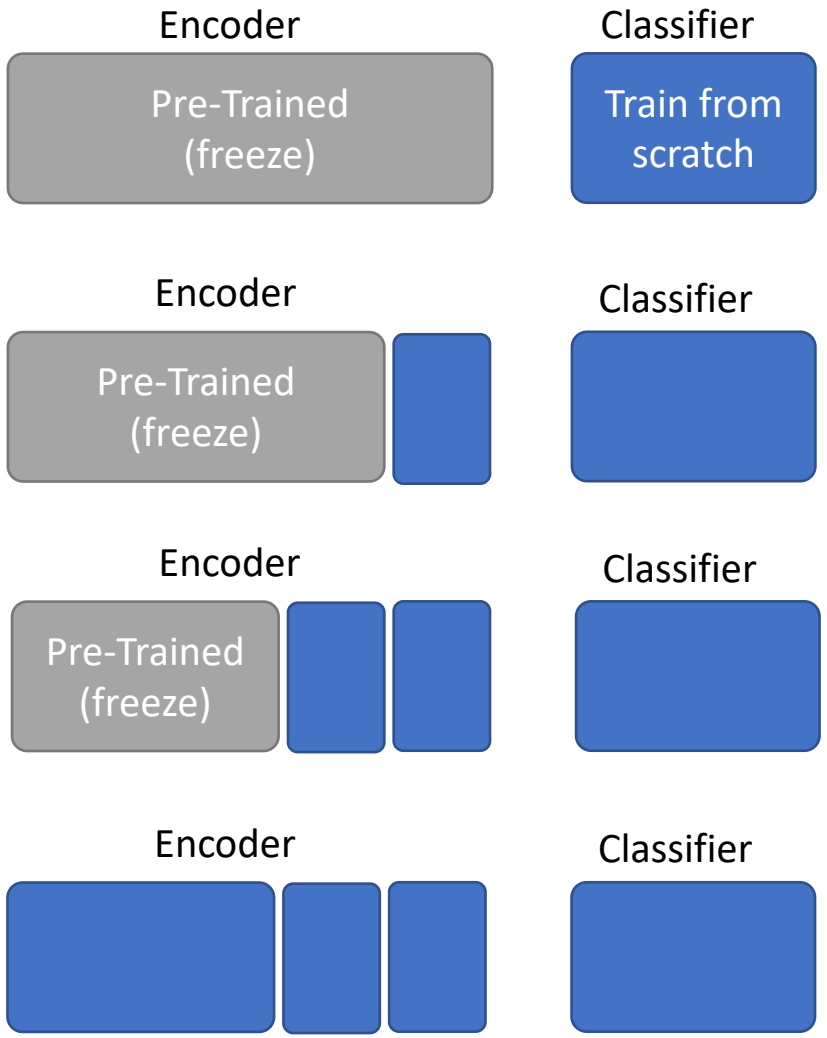
A technique where we start by keeping most of the pretrained layers “frozen” (i.e., their weights are not updated) and only train a new head layer. Over time, we selectively “unfreeze” deeper layers, allowing them to be fine-tuned.

Rationale

Avoids catastrophic forgetting and overfitting by carefully updating only the parts of the network that are most beneficial for the new task.

Gradual unfreezing

- **Phase 1**: Encoder (frozen) + Classifier head → Train from scratch (head only)
- **Phase 2**: Unfreeze top encoder layer → Fine-tune both top layer & head
- **Phase 3**: Unfreeze more encoder layers → Fine-tune incrementally
- **Phase 4**: (Optional) Unfreeze all layers → Full fine-tuning



Why Use Gradual Unfreezing?

Stabilized Training

- Minimizes abrupt weight changes in early layers, which often capture general language features.
- Reduces the risk of destabilizing the pretrained representations.

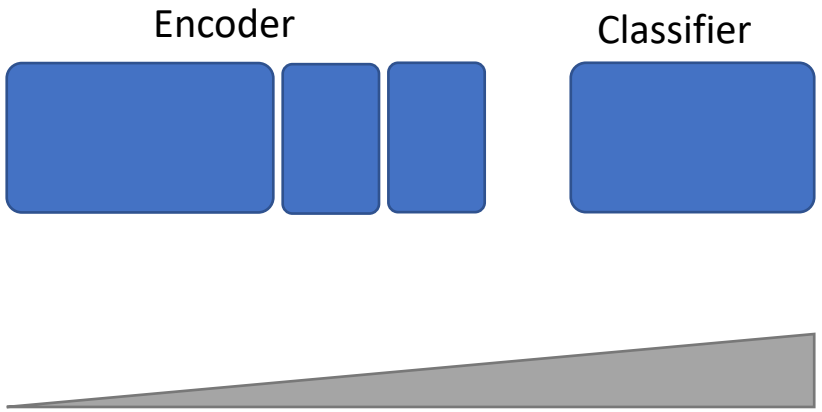
Better Generalization

- Allows the model to retain the broad knowledge it gained from large-scale pretraining.
- Incremental layer-wise updates adapt relevant features for the new task without discarding valuable “universal” features.

Efficiency in Early Stages

- You can train fewer parameters at first (only the classification head, for example), which often converges faster.
- Once confident with the results, you unfreeze additional layers for more fine-grained adaptation.

Discriminative Fine-Tuning



- Different learning rate for different layer groups
- Larger learning rate for top layers and classifier head
- Smaller learning rate for earlier/bottom layers

Introduced by Howard et al. in *Universal Language Model Fine-tuning for Text Classification*

Learning Rate Scheduling

Definition:
A strategy to **systematically adjust** the learning rate (LR) over the course of training.

Why It Matters:
- Helps **avoid overshooting** during later training stages.
- Can **improve convergence** and final model performance.

Common Scheduling Methods

Constant LR

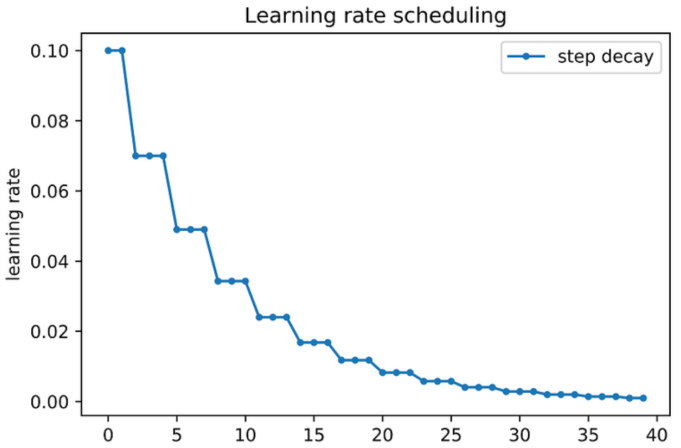
- Keeps the learning rate fixed throughout training.
- **Pros**: Simple to implement and reason about.
- **Cons**: May not adapt well as training progresses; might converge slowly or get stuck.

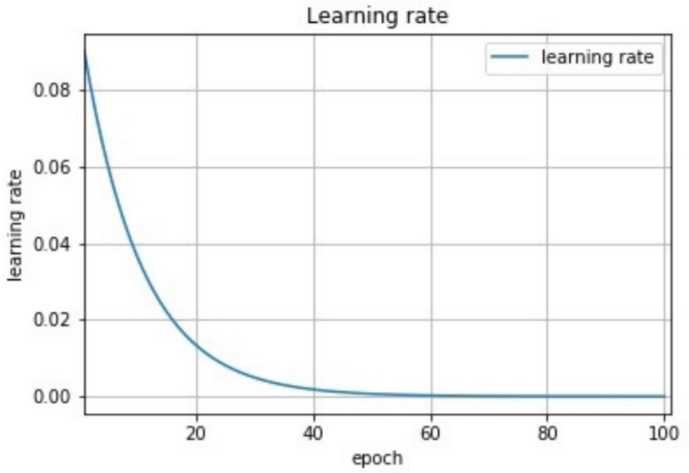
Step Decay

- LR is **reduced by a factor** (e.g., 0.1) after certain training epochs.
- **Pros**: Easy and often effective in practice.
- **Cons**: Abrupt changes can disrupt convergence; not as smooth as some alternatives.

Exponential Decay

- LR decays **exponentially** over time
- **Pros**: Smooth, gradual decrease.
- **Cons**: Can reduce LR too quickly if not tuned properly.

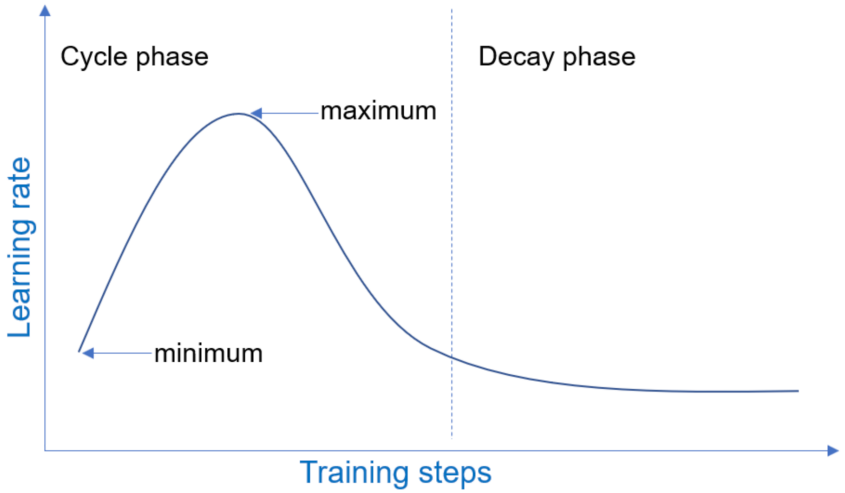




Common Scheduling Methods

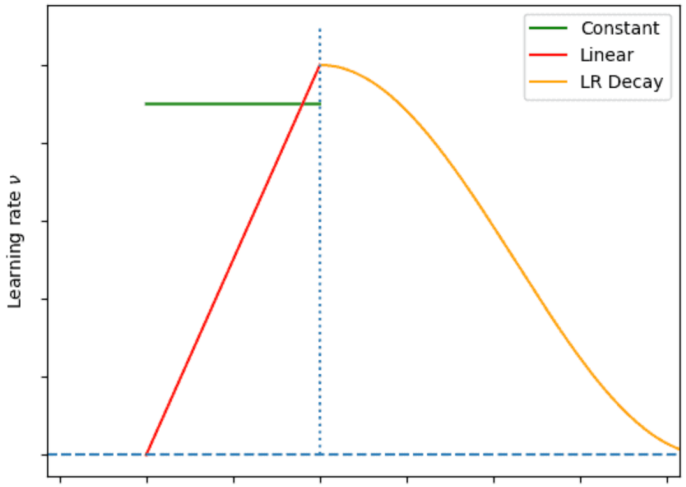
Cyclical Schedules (e.g., **Cyclical LR**, **OneCycle**):

- LR **cycles** between a lower bound and an upper bound throughout training.
- **Pros**: Sometimes prevents getting stuck in local minima; can speed up convergence.
- **Cons**: More hyperparameters to tune (cycle length, min/max LR).



Warmup + Decay (common in Transformer training):

- Start with a small LR and **linearly ramp up ("warmup")** over a few thousand steps, then **gradually decay** (linear or other patterns).
- **Pros**: Stabilizes early training, especially helpful for large models (e.g., BERT, GPT).
- **Cons**: Requires choosing optimal warmup steps and final decay shape.



Learning Rate Finder

Initial Setup

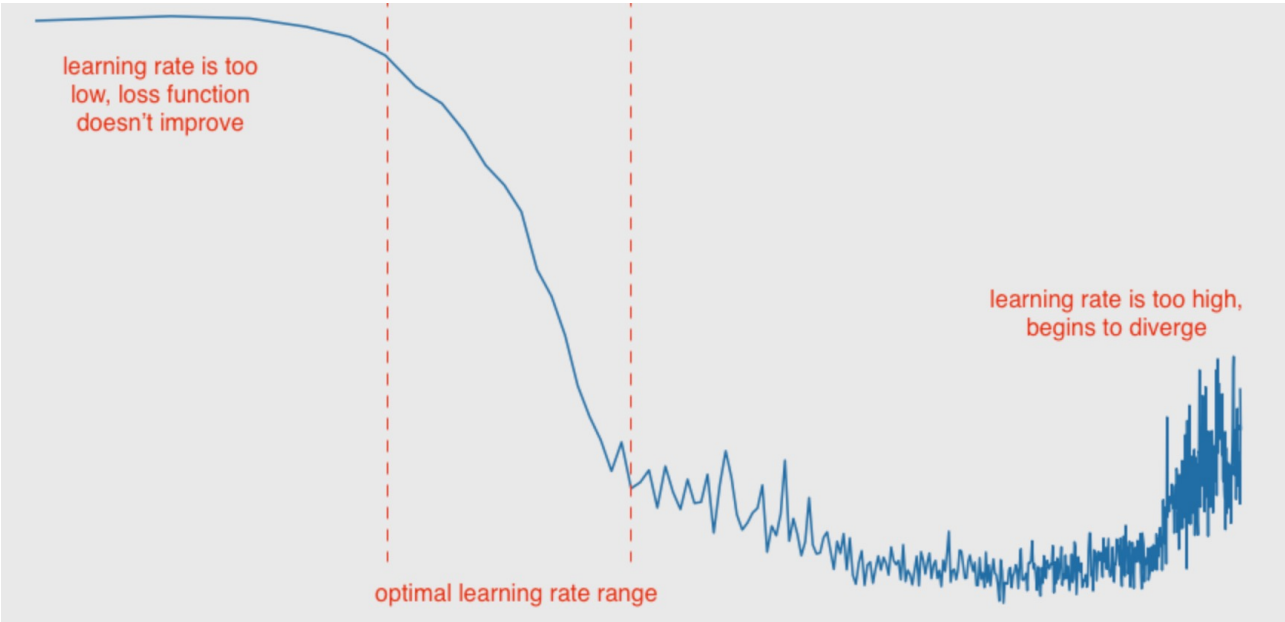
- Start training from a small LR (e.g., 1e-7) or a user-chosen lower bound.
- Gradually **increase** the LR (often exponentially) over a fixed number of batches or epochs.

Record Loss

- Track the training loss at each step.
- Typically, the LR is plotted (x-axis) against the loss (y-axis).

Identify the Optimal Range

- Look for where the loss **starts to decrease steeply** (too high an LR can cause divergence).
- A common rule of thumb: pick an LR slightly **before** the loss begins to shoot up again.



https://www.jeremyjordan.me/nn-learning-rate/

Leveraging LLMs to Generate Labels for BERT Finetuning

- **Use Case**: Unlabeled data + Large Language Model → Synthetic labels → Fine-tuned

Method:

- **LLM as Teacher**: Prompt the LLM with instructions or examples to classify or annotate unlabeled text.
- **Generate Pseudo-labels**: The LLM outputs labels/annotations for each unlabeled instance.
- **Train Student Model (BERT)**: Combine original input with the LLM-generated labels to train or fine-tune BERT.

Advantages:

- Quickly create labeled datasets at scale
- Potentially capture the LLM's richer understanding
- Cost-effective vs. large-scale manual annotation

Caveats:

- **Label Quality**: LLMs can generate erroneous or biased labels (hallucinations, domain mismatch)
- **Prompt Design**: Carefully engineered prompts often yield better labeling results
- **Ethical & Quality Checks**: Must validate synthetic data if used for real-world applications