# BERT

## Pre-training of Deep Bidirectional Transformers for Language Understanding

2019 @ **N**ations of the **A**mericas Chapter of the **A**ssociation for **C**omputational **L**inguistics

URL to the paper

# AGENDA

04/11/2024

1. **The Big Picture:** BERT in the context of language modelling
2. A bird's eye view of the **architecture**
3. BERT‘s **versatility** – pre-training / fine-tuning process
4. **Conclusion** & **Discussion**

# The (Single-Slide) Big Picture

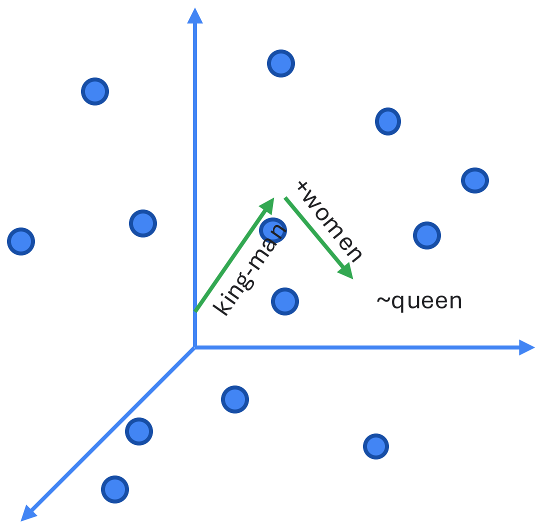
BERT in the context of language modelling

# THE BIG PICTURE: BERT IN THE CONTEXT OF LANGUAGE MODELING

Word2Vec is a static lookup table; BERT dynamically computes language encodings considering the whole sequence context.

## Word2Vec: Static Lookup Table

Word2Vec aims to learn vector representations of words capturing their semantic / syntactic.

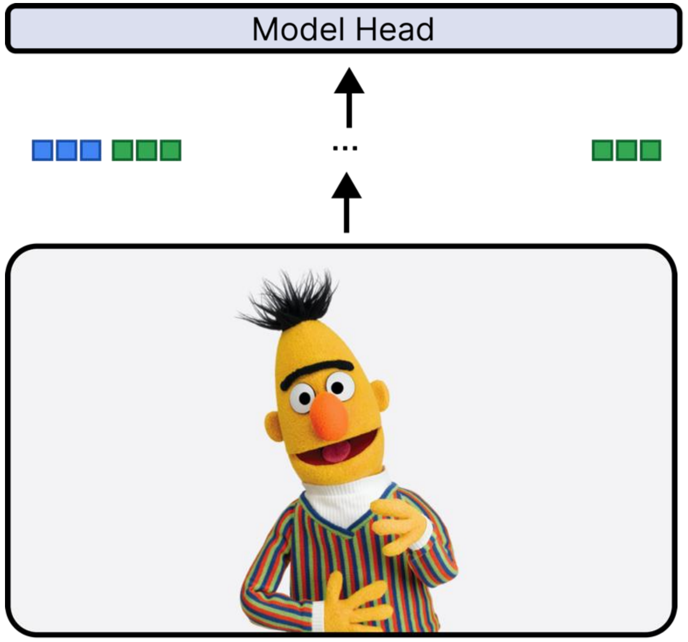



Has a fixed representation after the training ignoring current context of words.

*Please turn on the* ***light*** vs. *The feather is very* ***light***

## BERT: Dynamic Language Understanding

BERT aims to understand language by considering the full context of a word during training and inference.



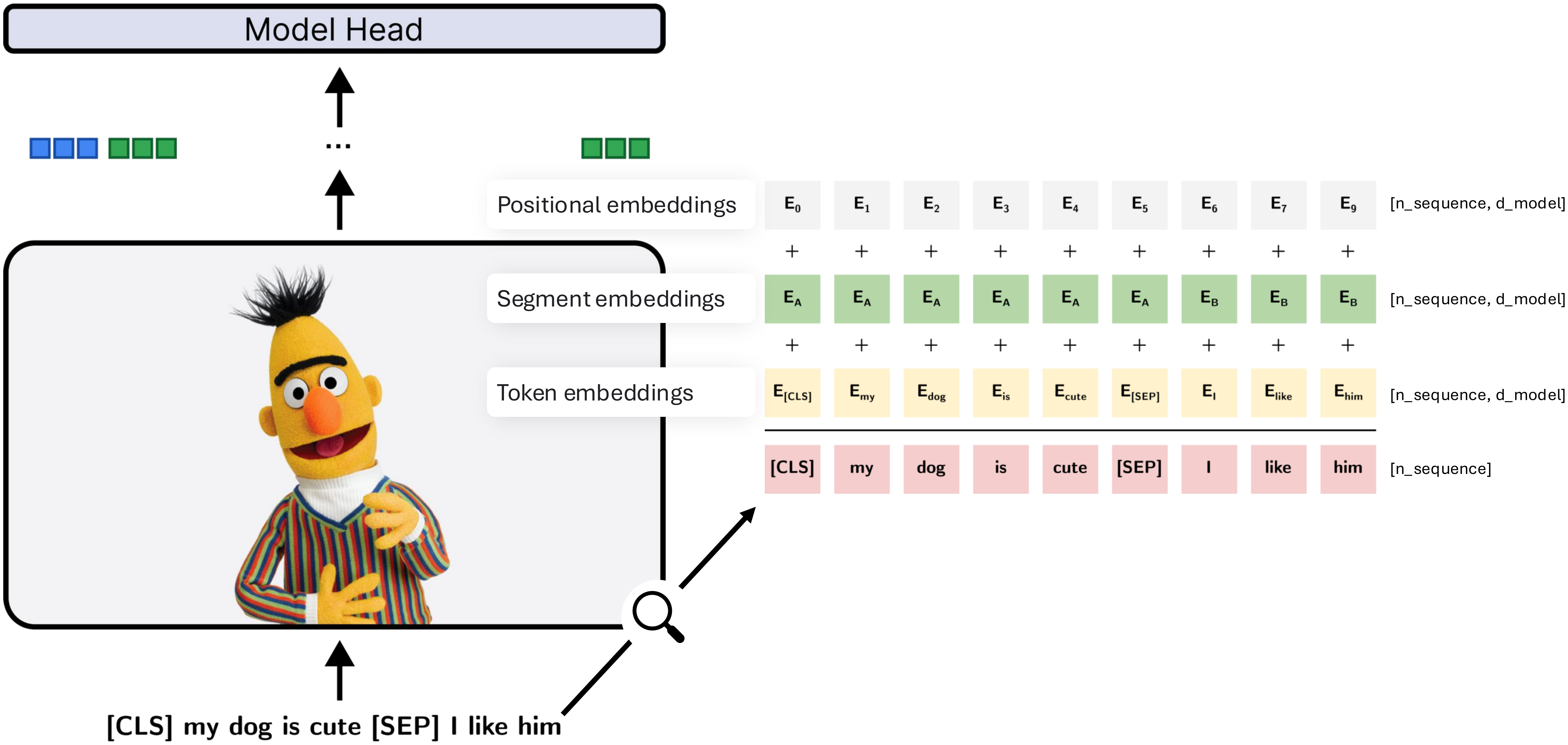

BERT can solve different NLP tasks using different fine-tuned model heads.

# A BERT’s* eye view of the architecture.

*bird’s

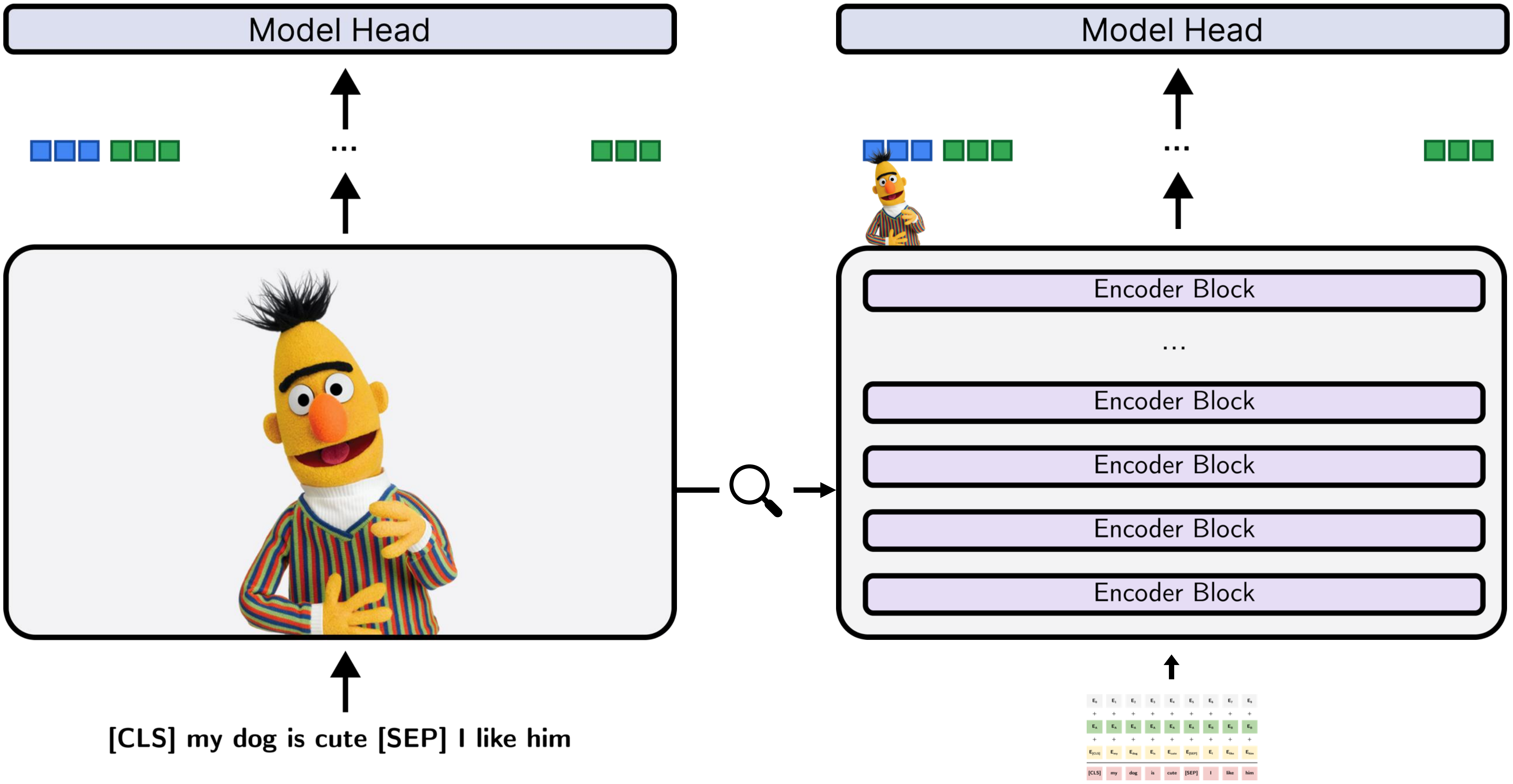
## A bird's eye view of the core architecture

The input sequence is projected input an embedding space before adding token-, segment- and positional embeddings.



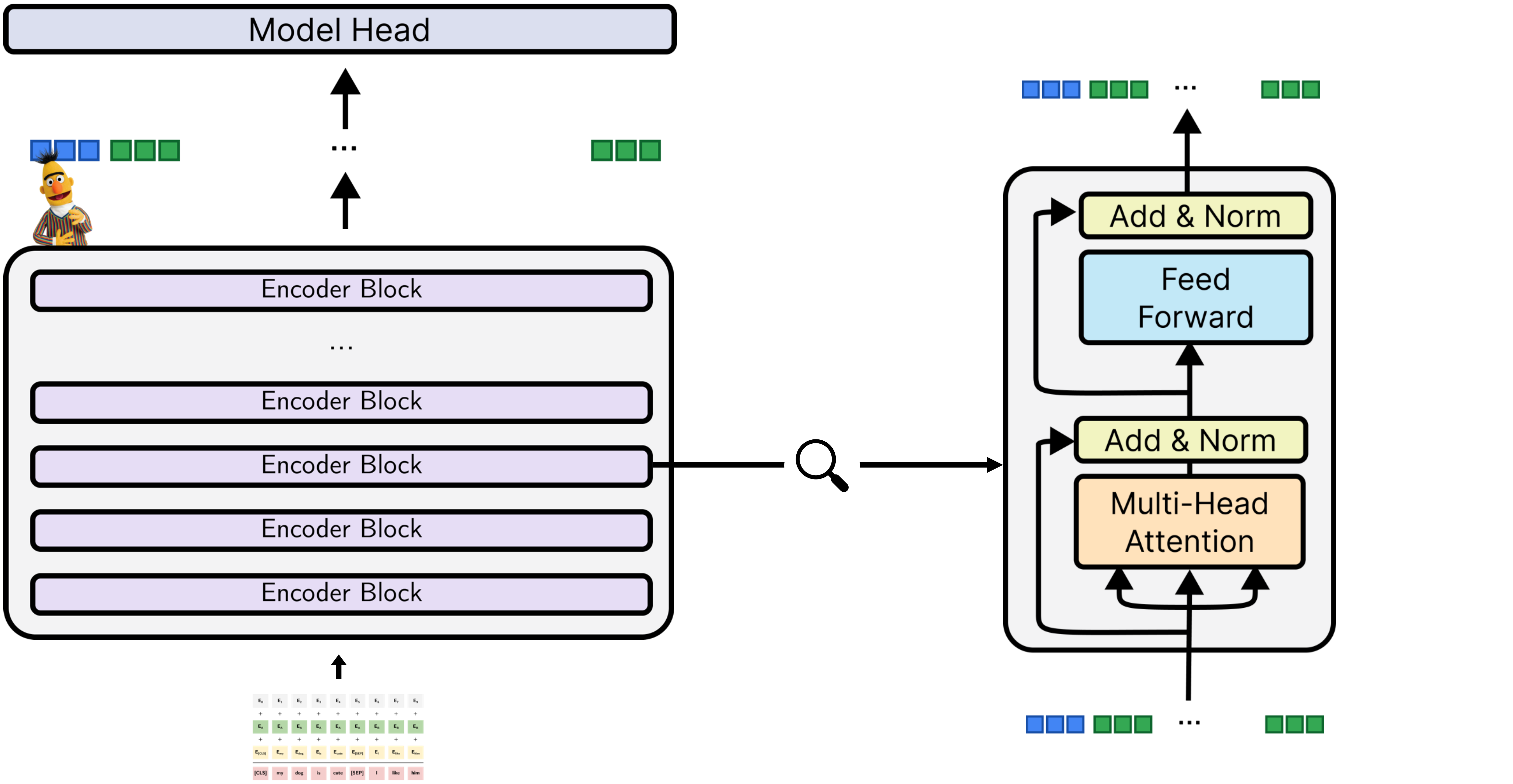
## A bird's eye view of the core architecture

BERT's model body is a repetition of multiple sequentially concatenated equivalent encoder blocks.



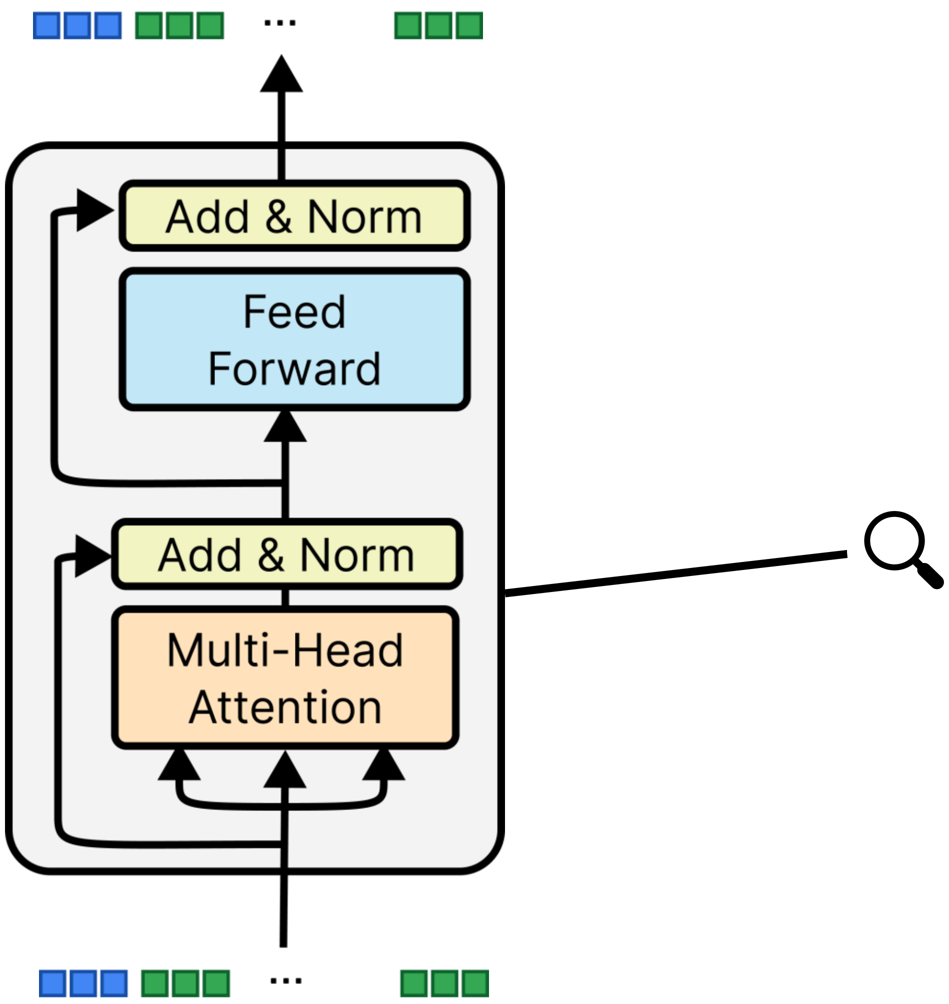
**A bird's eye view of the core architecture**

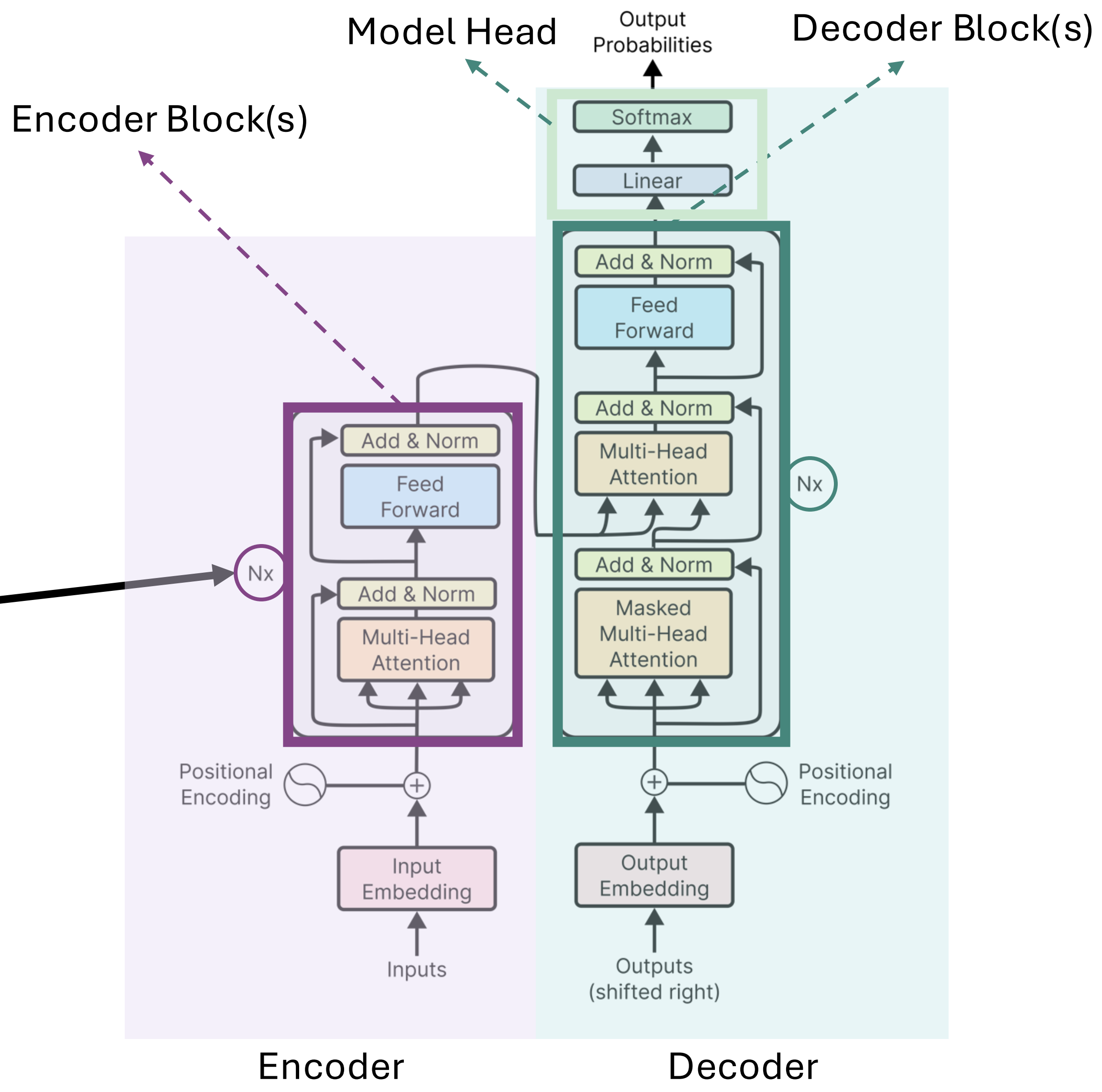
The encoder block consists of multi-head attention and a feed forward network with layer normalization and residual connectoins.



## A bird's eye view of the core architecture

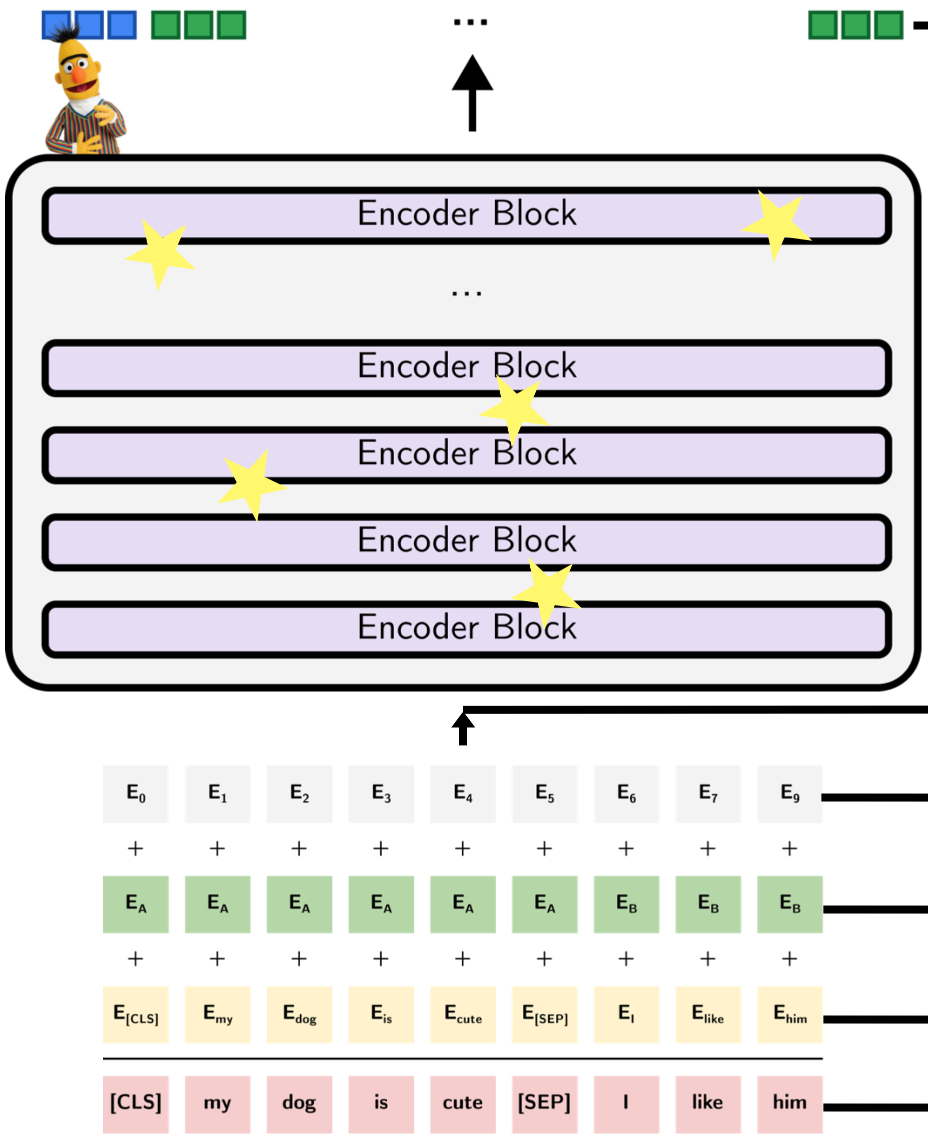
BERT "steals" its encoder block from the transformer architecture.



## A bird's eye view of the core architecture

BERT transforms a token-sequence into a sequence of dense vectors capturing their contextual meanings.




Tensor of shape [n_sequence_length, d_ff]

Each input token is transformer to an d_ff-dimensional vector, capturing its meaning within the sequence.

The [CLS] token captures the semantic meaning of the whole sequence.

Tensor of shape [n_sequence_length, d_model]

Tensor of shape [n_sequence_length, d_model]

Tensor of shape [n_sequence_length, d_model]

Tensor of shape [n_sequence_length, d_model]

Tensor or shape [n_sequence_length]

d_ff stands for the output dimension of the feed forward network of the encoder block.

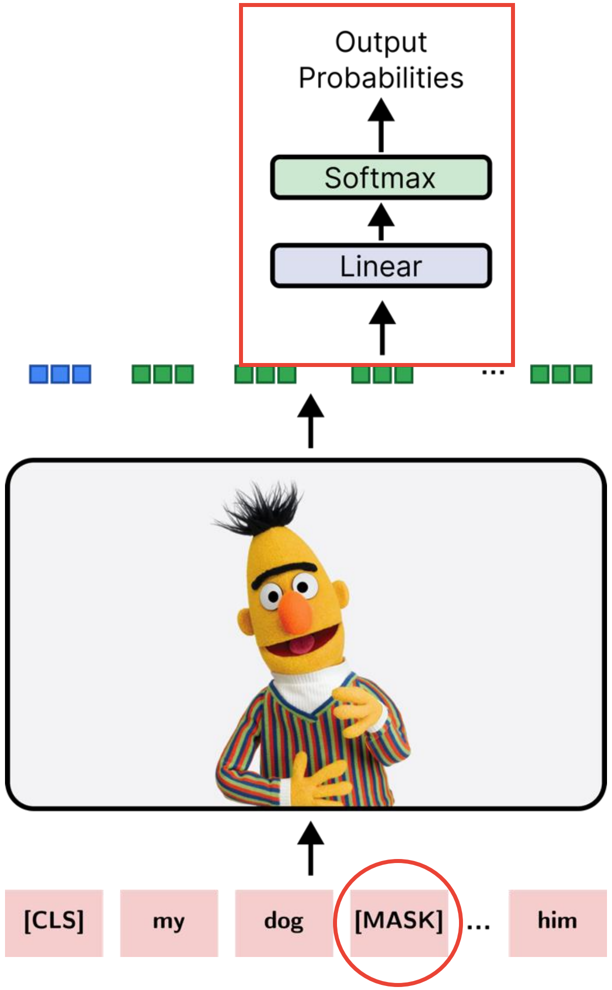
# BERT`s versatility

The pre-training / fine-tuning process

## BERT's versatility - The pre-training / fine-tuning process

BERT learns contextual language modelling by partially masking out input tokens.

Mask a 15% of the input at random and predict those masked tokens.

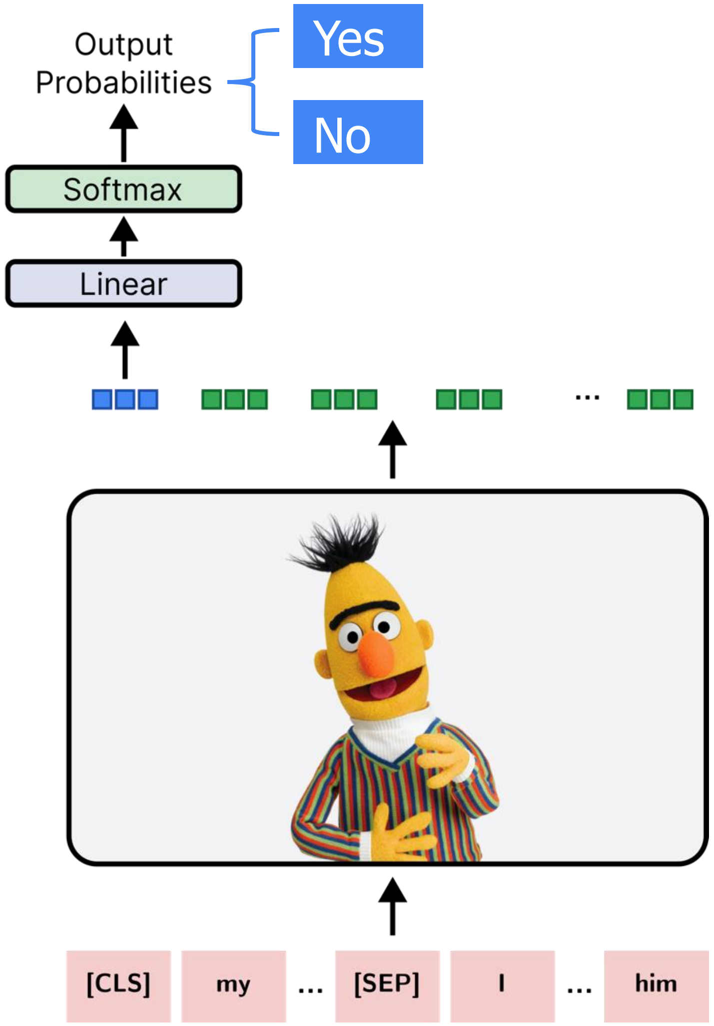



- 80% of the time: Replace the word with the *[MASK]* token
  To learn how words relate to their surrounding context to make accurate predictions.
- 10% of the time: Replace the token with a random token
  To prevent the model from simply learning that it should pay special attention to [MASK] tokens.
- 10% of the time: Keep the word unchanged
  To bias the representation towards the actual observed word.

# BERT's versatility - The pre-training / fine-tuning process

BERT learns sentence level contextual language modeling through next sentence predictions.

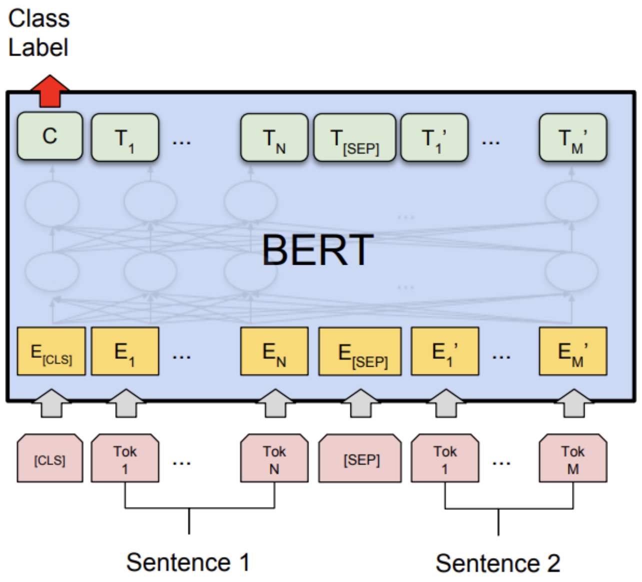
To learn In order sentence relationships, we pre-train for a binarized next sentence prediction.



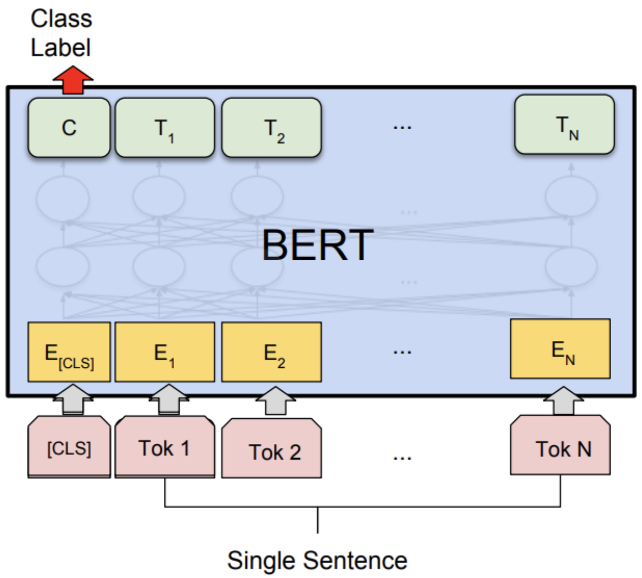

- 50% of the time actual next sentence is used 50% of the time it is a random sentence from the corpus
- [CLS] related output is used for binary classifcation

## BERT's versatility - The pre-training / fine-tuning process

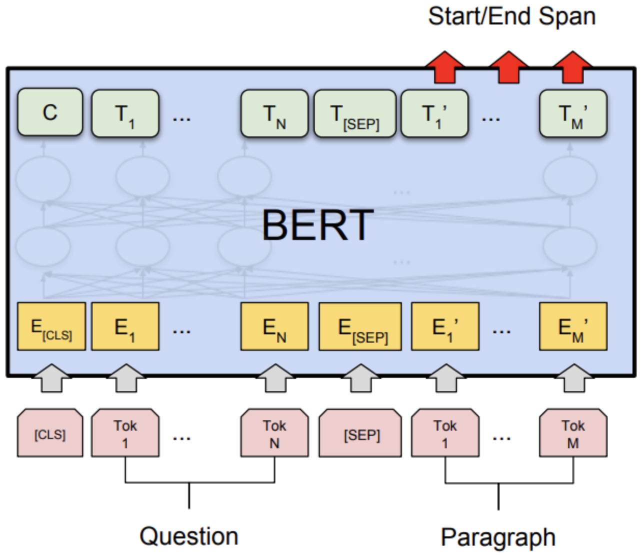
We can adapt BERT to different tasks by using different model heads and fine-tuning all the parameters end-to-end.



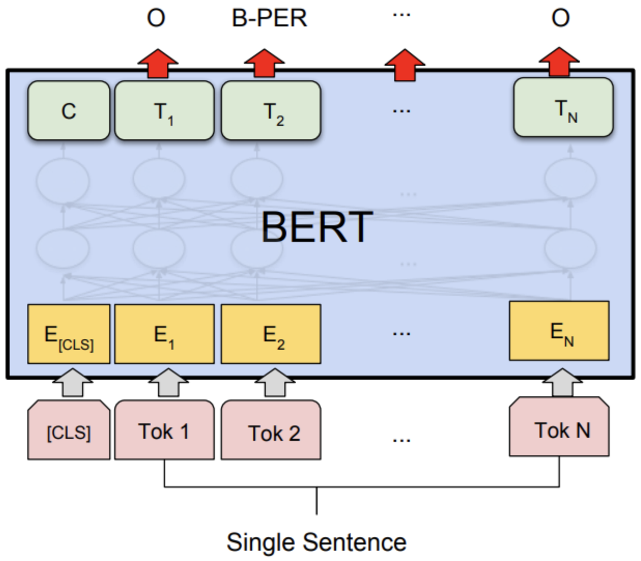

Sentence Pair Classification




Single Sentence Classification Tasks




Question Answering Tasks




Single Token Tagging Tasks

# Conclusion & Discussion

And a few numbers

## Conclusion & Discussion
Some numbers

| | Encoder Blocks | $d_{\text{Feed Forward \& Embeddings}}$ | Attention Heads | Total Params | Pretraining | TPUs |
|---|---|---|---|---|---|---|
| $\text{BERT}_{\text{BASE}}$ | 12 | 768 | 12 | 110 millions | 4 days | 4 |
| $\text{BERT}_{\text{LARGE}}$ | 24 | 1024 | 16 | 340 millions | 4 days | 16* |

* At the current on-demand prices of v3 TPUs - which were most likely used at that time – of 2.2$ the large model would cost around 10k of pretraining. Keep in mind that Google trained it on their own infrastructure, which will have cost them much less.

### Dataset
- BooksCorpus (800 million words) & text passages of english wikipedia (2,5 billion words) with WordPiece tokenization with 30,000 token vocabulary

### Batching/Training Config
- Total sequence length ≤ 512 tokens[1]
- Batch size of 256 sequences (256 sequences x 512 tokens = 128,000 tokens/batch)
- 1,000,000 total steps, which is approximately 40 epochs over the 3.3 billion word corpus.

### For the geeks
- Adam optimizer with lr of 1e-4, β1 = 0.9, β2 = 0.999, L2 weight decay of 0.01, warmup over the first 10,000 steps, and linear lr decay
- Dropout of 0.1

1. To speed up pretraing a sequence length of 128 for 90% of the steps was used and only the final 10% of the steps of sequence of 512 to learn the positional embeddings.

## Conclusion & Discussion

GLUE was used as the primary benchmark for BERT which it domiated in 2019.

### Back in 2019

| System | MNLI-(m/mm) 392k | QQP 363k | QNLI 108k | SST-2 67k |
|---|---|---|---|---|
| Pre-OpenAI SOTA | 80.6/80.1 | 66.1 | 82.3 | 93.2 |
| BiLSTM+ELMo+Attn | 76.4/76.1 | 64.8 | 79.8 | 90.4 |
| OpenAI GPT | 82.1/81.4 | 70.3 | 87.4 | 91.3 |
| BERT$_{\text{BASE}}$ | 84.6/83.4 | 71.2 | 90.5 | 93.5 |
| BERT$_{\text{LARGE}}$ | **86.7/85.9** | **72.1** | **92.7** | **94.9** |

| System | CoLA 8.5k | STS-B 5.7k | MRPC 3.5k | RTE 2.5k | **Average** - |
|---|---|---|---|---|---|
| Pre-OpenAI SOTA | 35.0 | 81.0 | 86.0 | 61.7 | 74.0 |
| BiLSTM+ELMo+Att | 36.0 | 73.3 | 84.9 | 56.8 | 71.0 |
| OpenAI GPT | 45.4 | 80.0 | 82.3 | 56.0 | 75.1 |
| BERT$_{\text{BASE}}$ | 52.1 | 85.8 | 88.9 | 66.4 | 79.6 |
| BERT$_{\text{LARGE}}$ | **60.5** | **86.5** | **89.3** | **70.1** | **82.1** |

### And Now?

| Rank | Name | Model | Score | CoLA | SST-2 | MRPC | STS-B | QQP | MNLI-m | MNLI-mm | QNLI | RTE | WNLI | AX |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Microsoft Alexander v-team | Turing ULR v6 | 91.3 | 73.3 | 97.5 | 94.2/92.3 | 93.5/93.1 | 76.4/90.9 | 92.5 | 92.1 | 96.7 | 93.6 | 97.9 | 55.4 |
| 2 | JDExplore d-team | Vega v1 | 91.3 | 73.8 | 97.9 | 94.5/92.6 | 93.5/93.1 | 76.7/91.1 | 92.1 | 91.9 | 96.7 | 92.4 | 97.9 | 51.4 |
| 3 | Microsoft Alexander v-team | Turing NLR v5 | 91.2 | 72.6 | 97.6 | 93.8/91.7 | 93.7/93.3 | 76.4/91.1 | 92.6 | 92.4 | 97.9 | 94.1 | 95.9 | 57.0 |
| 4 | DIRL Team | DeBERTa + CLEVER | 91.1 | 74.7 | 97.6 | 93.3/91.1 | 93.4/93.1 | 76.5/91.0 | 92.1 | 91.8 | 96.7 | 93.2 | 96.6 | 53.3 |

...

| Rank | Name | Model | Score | CoLA | SST-2 | MRPC | STS-B | QQP | MNLI-m | MNLI-mm | QNLI | RTE | WNLI | AX |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 47 | Mikita Sazanovich | Routed BERTs | 80.7 | 56.1 | 93.6 | 88.6/84.7 | 88.0/87.6 | 71.0/88.8 | 85.2 | 84.5 | 92.6 | 80.0 | 65.1 | 9.2 |
| 48 | USCD-AI4Health Team | CERT | 80.7 | 58.9 | 94.6 | 89.8/85.9 | 87.9/86.8 | 72.5/90.3 | 87.2 | 86.4 | 93.0 | 71.2 | 65.1 | 39.6 |
| 49 | Jacob Devlin | BERT: 24-layers, 16-heads, 1024-hidden | 80.5 | 60.5 | 94.9 | 89.3/85.4 | 87.6/86.5 | 72.1/89.3 | 86.7 | 85.9 | 92.7 | 70.1 | 65.1 | 39.6 |
| 50 | Chen Qian | KerasNLP XLM-R | 80.4 | 56.3 | 96.1 | 89.8/86.3 | 88.4/87.7 | 72.3/89.0 | 87.7 | 87.1 | 92.8 | 69.2 | 65.1 | 40.6 |

**G**eneral **L**anguage **U**nderstanding **E**valuation (GLUE) is a collection of diverse natural language understanding tasks.

## Conclusion & Discussion

An overview of the datasets used to evaluate BERT.

| Acronym | Full Name | Description |
|---|---|---|
| **MNLI** | Multi-Genre Natural Language Inference | Predicting whether sentence pairs are *entailment, contradiction,* or *neutral* |
| **QQP** | Quora Question Pairs | Determining if two questions are semantically equivalent |
| **QNLI** | Question Natural Language Inference | A version of Stanford Question Answering Dataset converted to binary classification |
| **SST-2** | Stanford Sentiment Treebank | Binary sentiment classification of movie reviews |
| **CoLA** | Corpus of Linguistic Acceptability | Predicting whether English sentences are linguistically acceptable |
| **STS-B** | Semantic Textual Similarity Benchmark | Rating similarity of sentence pairs on a scale of 1-5 |
| **MRPC** | Microsoft Research Paraphrase Corpus | Identifying whether sentence pairs are semantically equivalent |
| **RTE** | Recognizing Textual Entailment | Similar to MNLI but with much less training data |
| **WNLI** | Winograd NL | A small natural language inference dataset[1] |

1. Authors excluded this due to issues with the dataset)

## Conclusion & Discussion

One can extract fixed features from the pretrained model.

| System | Dev F1 | Test F1 |
|---|---|---|
| ELMo (Peters et al., 2018a) | 95.7 | 92.2 |
| CVT (Clark et al., 2018) | - | 92.6 |
| CSE (Akbik et al., 2018) | - | **93.1** |
| Fine-tuning approach | | |
| $\text{BERT}_{\text{LARGE}}$ | 96.6 | 92.8 |
| $\text{BERT}_{\text{BASE}}$ | 96.4 | 92.4 |
| Feature-based approach ($\text{BERT}_{\text{BASE}}$) | | |
| Embeddings | 91.0 | - |
| Second-to-Last Hidden | 95.6 | - |
| Last Hidden | 94.9 | - |
| Weighted Sum Last Four Hidden | 95.9 | - |
| Concat Last Four Hidden | 96.1 | - |
| Weighted Sum All 12 Layers | 95.5 | - |

CoNLL-2003 Named Entity Recognition results. Hyperparameters were selected using the Dev set. The reported Dev and Test scores are averaged over 5 random restarts using those hyperparameters.

☠ Word2Vec[2]

**Reminder[1]**

- $F1 = 2 * (P * R) / (P + R)$
- $P = TP/(TP + FP)$
  *"Of all entities we predicted, what fraction did we get right?"*
- $R = TP/(TP + FN)$
  *"Of all actual entities, what fraction did we find?"*

› Why would we like to do that?

*Authors:* Computational Efficiency & adaptability to task specific model architectures.

› Why might it make sense to use different layers instead of just the final layer?

*A few guesses from the presenter:*

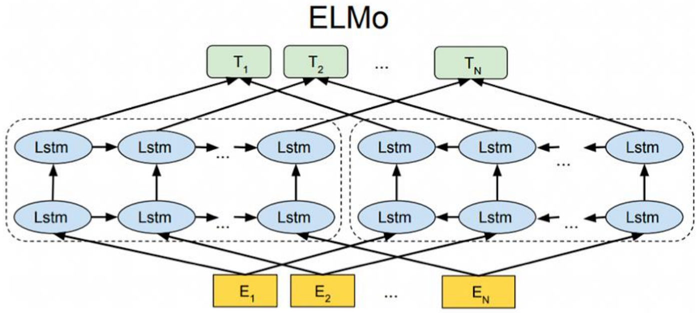
- *Different layers capture different types of information*
- *Combining layers might provide complementary information*
- *Final layer might be specialized on pretraining task*

1. P … Precision, R … Recall, TP … True Positives, FP … False Positives, FN … False Negatives
2. This represents the personal opinion of the presenter. He does not have an affiliation with BERT or Google to praise or advise against any models/products.

# Conclusion & Discussion

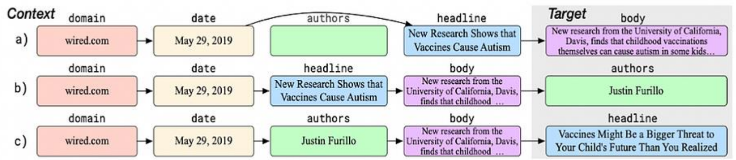
Can you name other models inspired by the Sesame Street?

ELMO (1980)




ELMo (2018)

GROVER (1970)

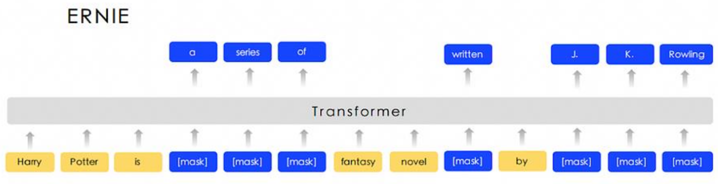



GROVER (2019)

Embeddings from Language Models

Generates realistic and controlled fake news

ERNIE (1969)




ERNIE (2019)

BERT with entity-level and phrase-level masking

**Conclusion & Discussion**

The BERT architecture has a huge ecosystem[1] of different specialised BERT-like models

- ALBERT
- RoBERTa
- HerBERT
- *[…]*
- I-BERT

› You-BERT?

*Have you ever used a BERT-like mode? What for / What was your experience?*

› Isn't a decoder all you need?

*Will BERT like models be swallowed by decoder only models (e.g. GPT)*


1. https://huggingface.co/docs/transformers/model_doc/bert

# APENDIX

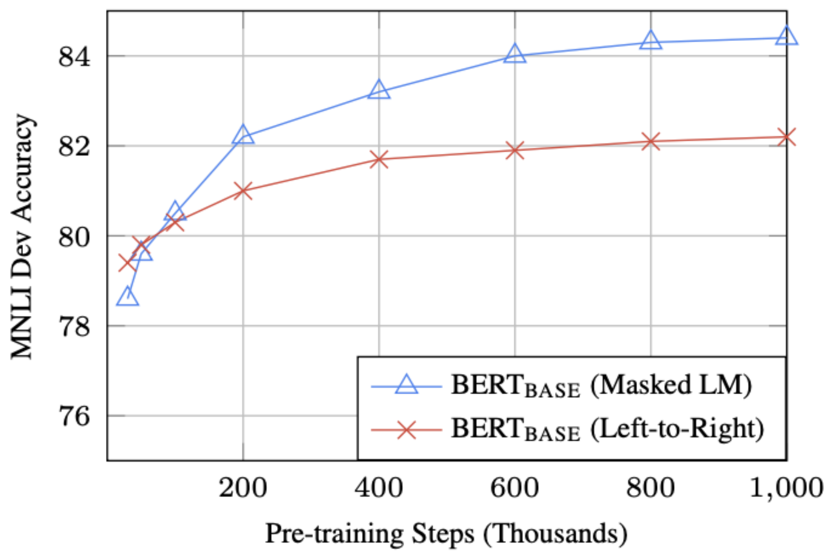
## Ablation Studies

The effect of the different train strategies on the overall performance of BERT.

| Tasks | Dev Set MNLI-m (Acc) | QNLI (Acc) | MRPC (Acc) | SST-2 (Acc) | SQuAD (F1) |
|---|---|---|---|---|---|
| $\text{BERT}_{\text{BASE}}$ | 84.4 | 88.4 | 86.7 | 92.7 | 88.5 |
| No NSP | 83.9 | 84.9 | 86.5 | 92.6 | 87.9 |
| LTR & No NSP | 82.1 | 84.3 | 77.5 | 92.1 | 77.8 |
| + BiLSTM | 82.1 | 84.1 | 75.7 | 91.6 | 84.9 |

Table 5: Ablation over the pre-training tasks using the $\text{BERT}_{\text{BASE}}$ architecture. "No NSP" is trained without the next sentence prediction task. "LTR & No NSP" is trained as a left-to-right LM without the next sentence prediction, like OpenAI GPT. "+ BiLSTM" adds a randomly initialized BiLSTM on top of the "LTR + No NSP" model during fine-tuning.

*Note:* I have not found any indication on how the model is modified for LTR (i.e. whether it's only a change in the loss function or whether they've added future token masking like in decoder only models)



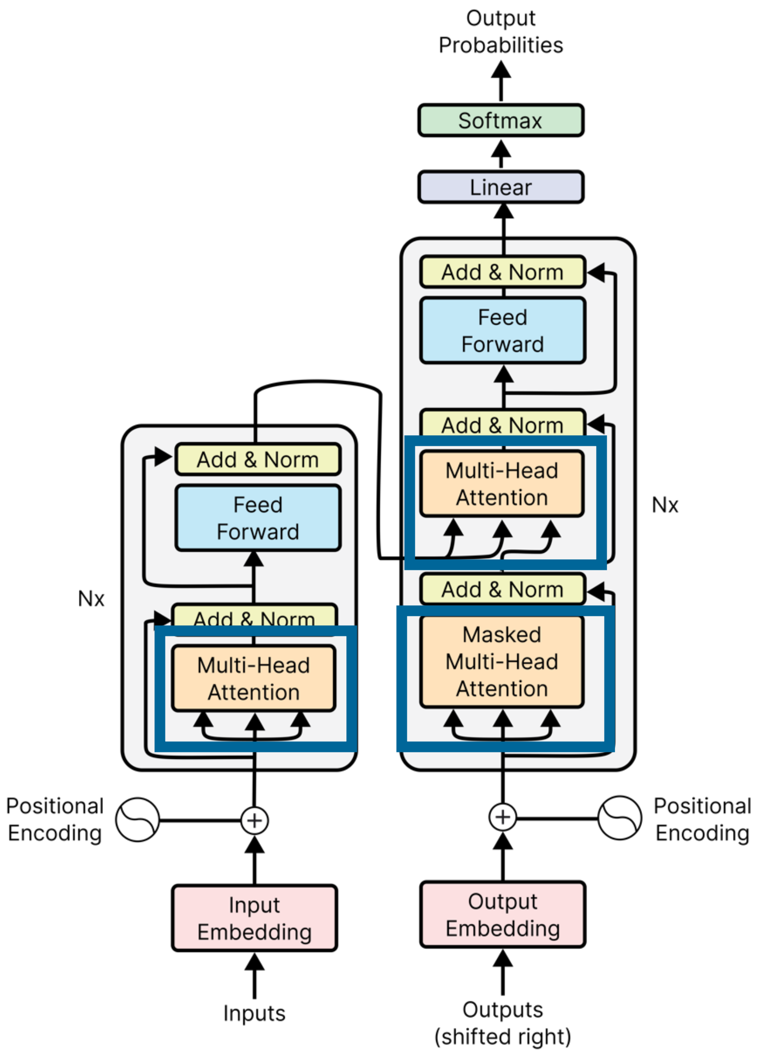

*MLM vs. LTR*

| Masking Rates MASK | SAME | RND | Dev Set Results MNLI Fine-tune | NER Fine-tune | NER Feature-based |
|---|---|---|---|---|---|
| 80% | 10% | 10% | 84.2 | 95.4 | 94.9 |
| 100% | 0% | 0% | 84.3 | 94.9 | 94.0 |
| 80% | 0% | 20% | 84.1 | 95.2 | 94.6 |
| 80% | 20% | 0% | 84.4 | 95.2 | 94.7 |
| 0% | 20% | 80% | 83.7 | 94.8 | 94.6 |
| 0% | 0% | 100% | 83.6 | 94.9 | 94.6 |

*Performance of different masking strategies*

# The attention mechanism

The cornerstone of the transformer's ability to capture context.



## The attention mechanism

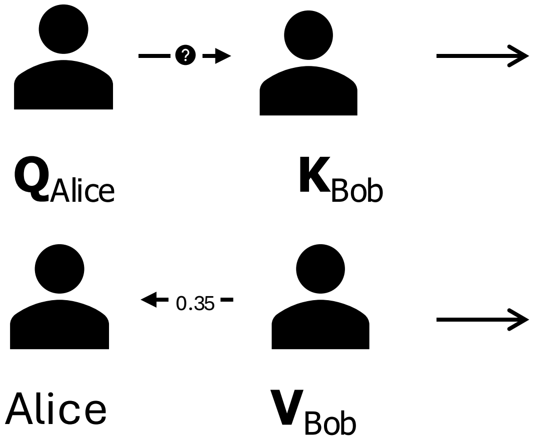
An example of the exchange of attention in a group of people eager to share and gather knowledge.

- Alice, Bob, Charlie, David, and Eve want to advance their knowledge in certain areas by spending time and attention.
- Each of them has some knowledge they can teach.
- Each of them has a **limited capacity** to **pay attention** (one can only learn so much in a week).
- Each of them can **receive attention indefinitely** (you can be listened to by everyone).

**Q**uery ... a description of the knowledge they want to gather

**K**ey ... a description of what they can teach

**V**alue ... the actual knowledge




Alice decides to pay 35% of her attention to what Bob has to teach.

Alice is influenced by 35% of what Bob has taught.

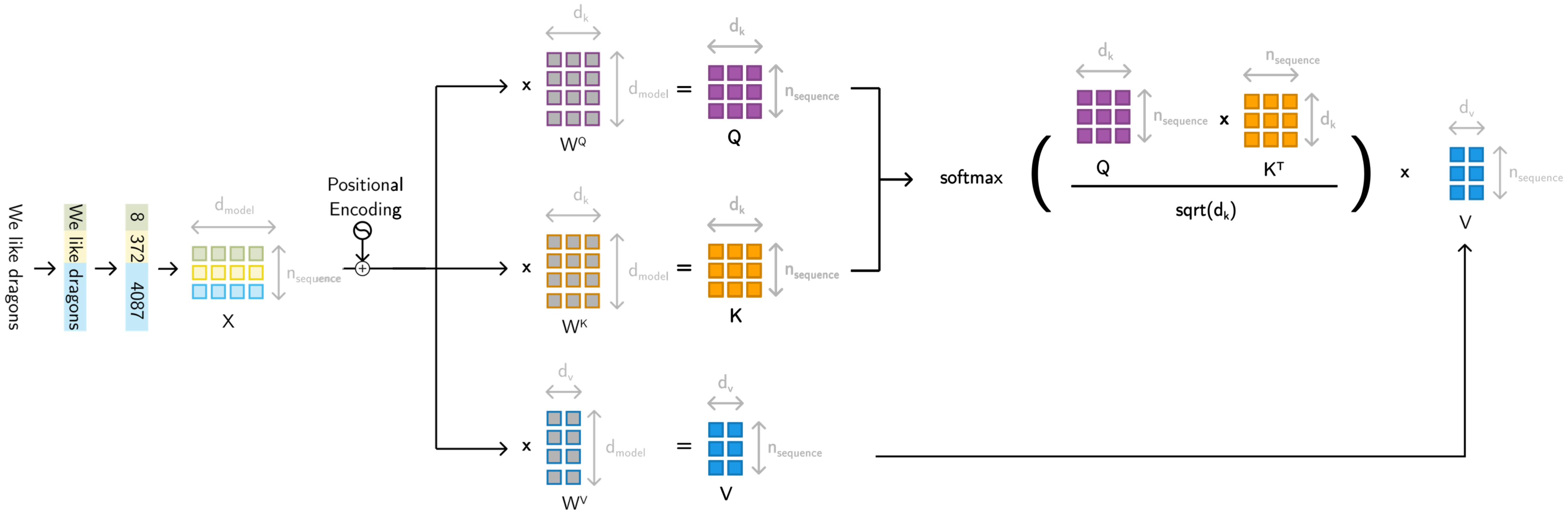
*Visualisation of an interaction between two people[1]*

The goal of the attention mechanism is to compute and ingest *how much* and *how* each token should influence the representation (“meaning”) of every other token within the input sequence.

1. Corresponds to two tokens

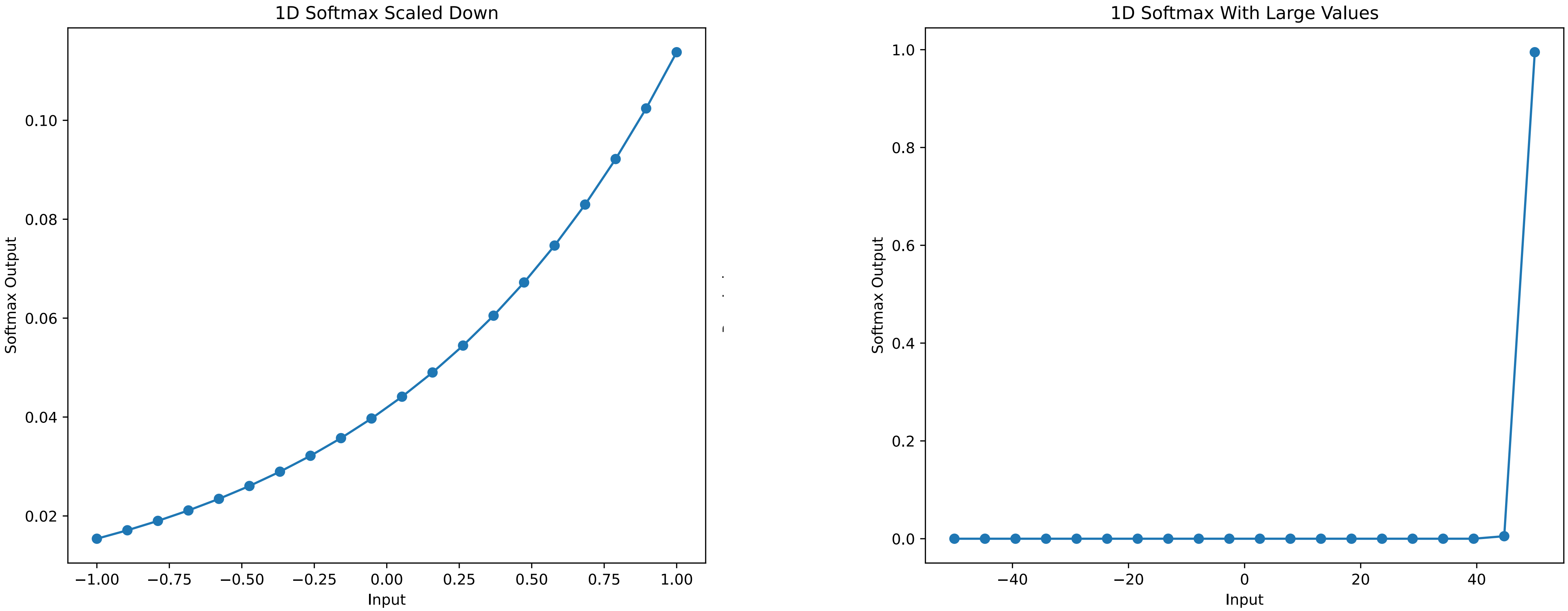
# The attention mechanism

Compute and ingest *how much* and *how* each token should influence the representation of every other token.



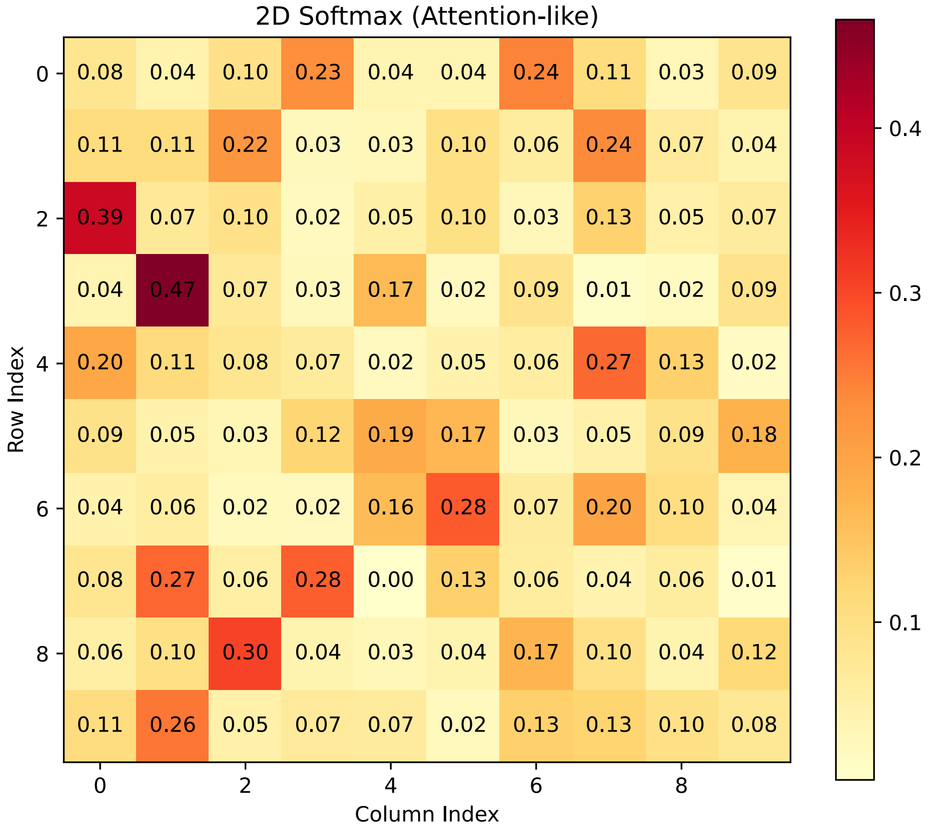
## The attention mechanism

Scaling the softmax input smoothes the curve and ensures gradient stability.



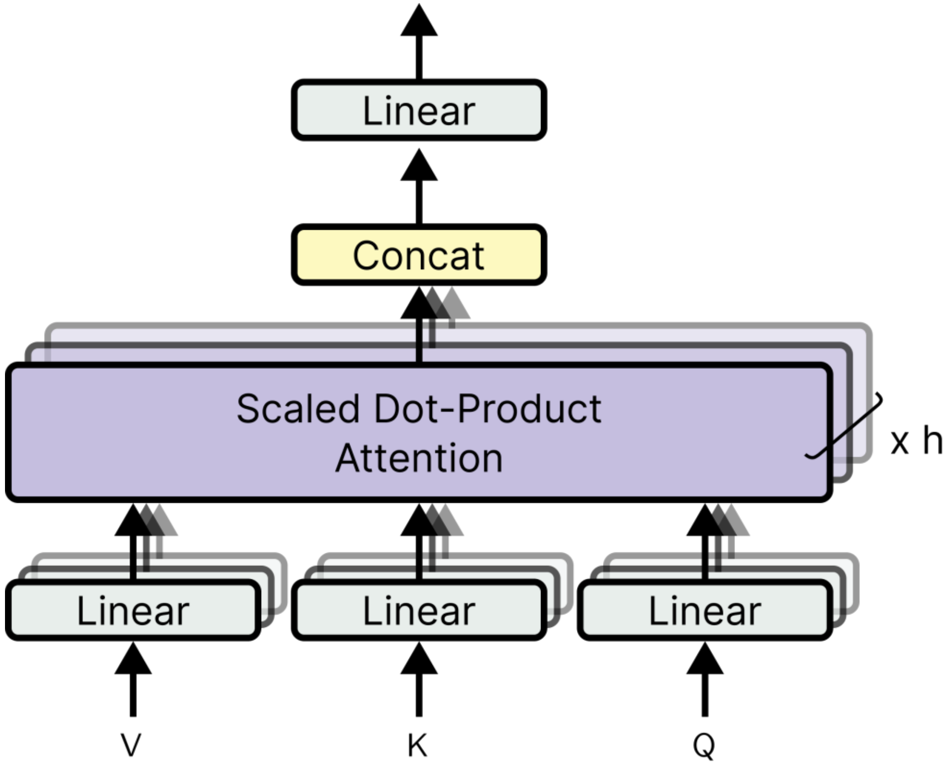
# The attention mechanism

Example of randomly assigned attention scores to demonstrate the ability of the softmax function to normalise its input.



# Multi-Head Attention

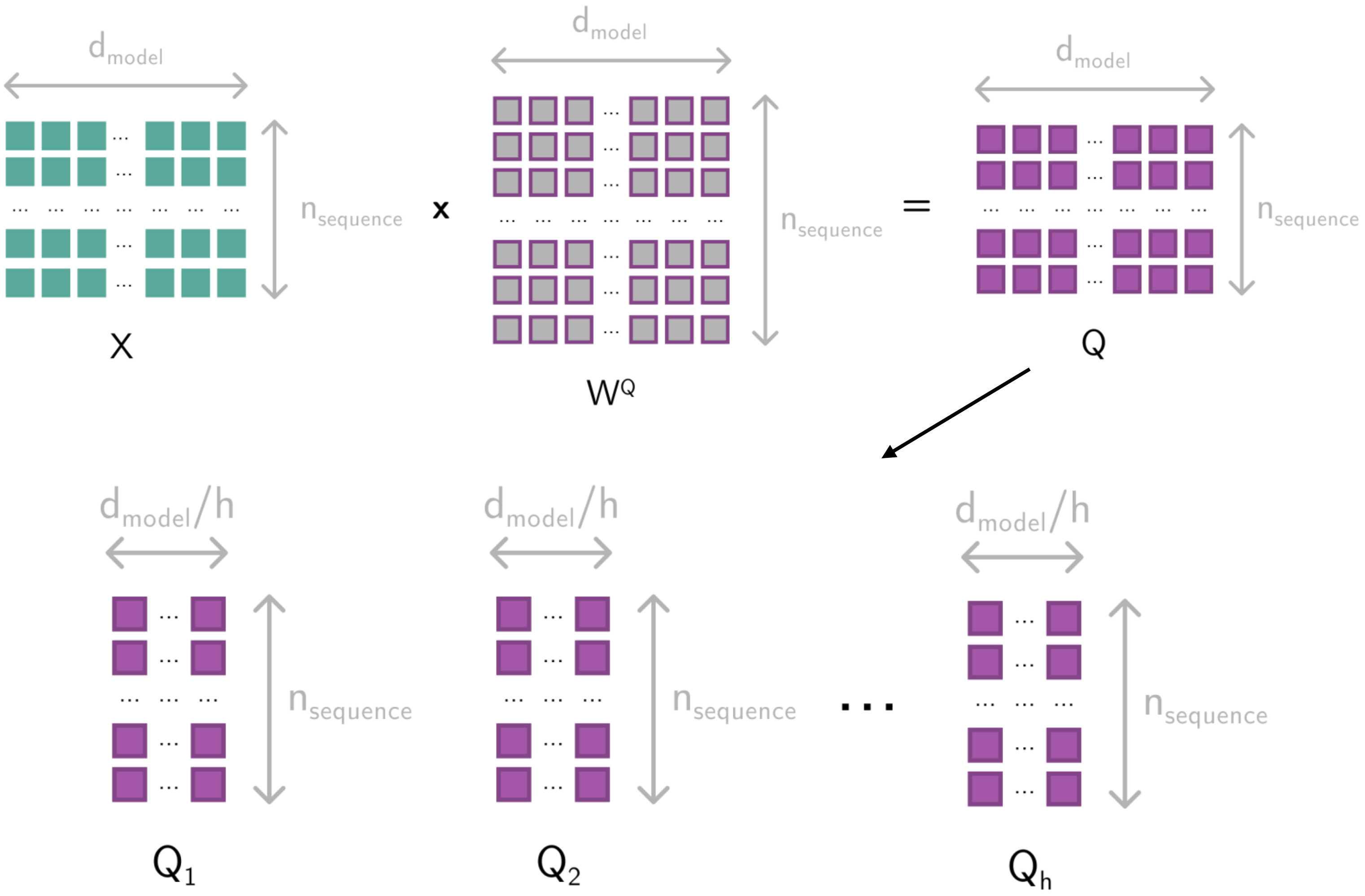
How transformers pay attention to different "semantic properties" of tokens



## The attention mechanism

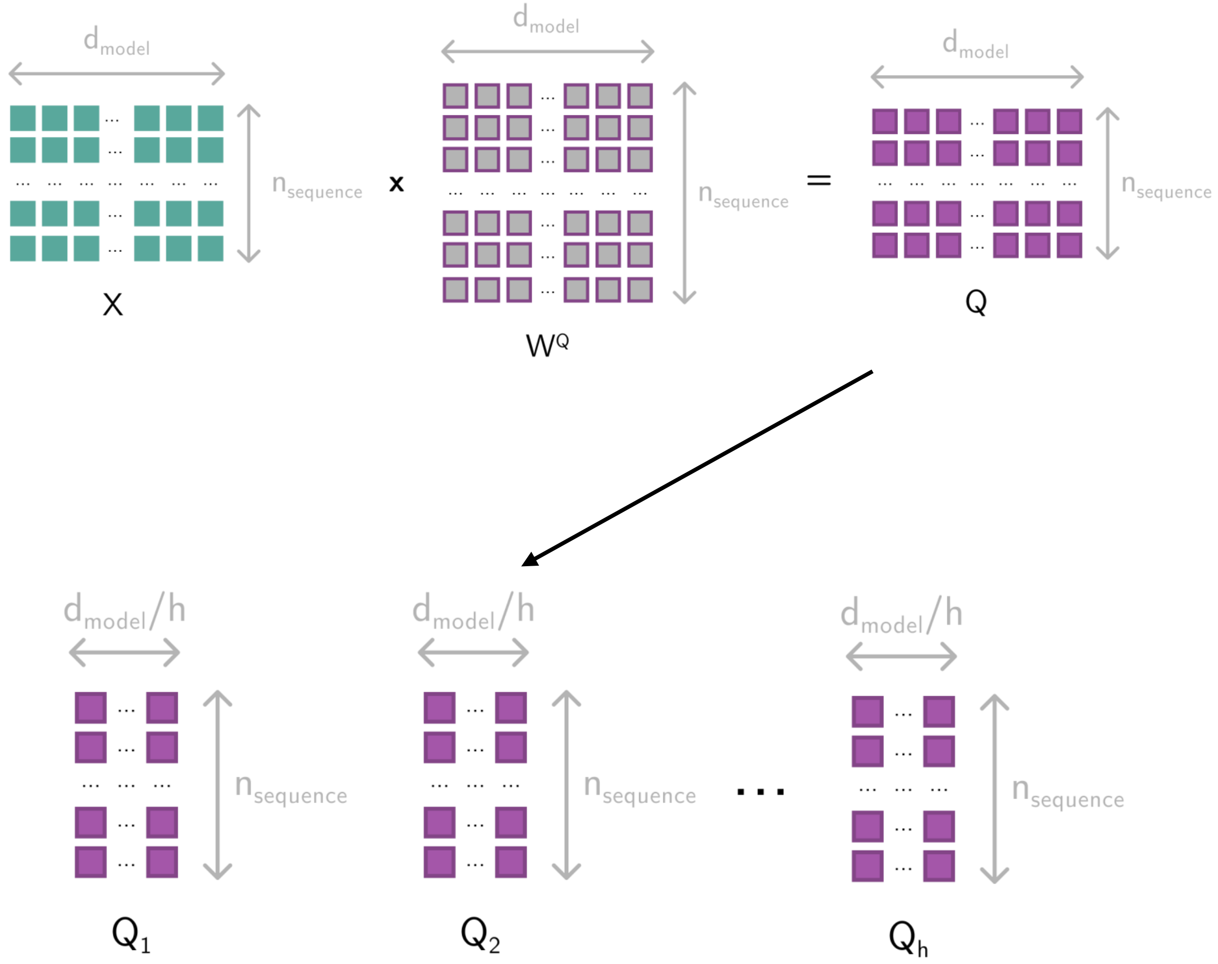
Mathematical intuition behind multi-head attention.

The goal of the multi-head attention is to attend to tokens from different perspectives in parallel.



## The attention mechanism

Mathematical intuition behind multi-head attention.




```python
import torch
from torch import nn

d_model = 256
n_context = 1048
h = 8
d_k = d_model // h

x = torch.randn(n_seq, d_model)

W_Q = nn.Linear(d_model, d_model, bias=False)
W_K = nn.Linear(d_model, d_model, bias=False)
W_V = nn.Linear(d_model, d_model, bias=False)
W_O = nn.Linear(d_model, d_model, bias=False)

Q = W_Q(x)
K = W_K(x)
V = W_V(x)

Q = Q.view(n_seq, h, d_k).transpose(0, 1)
K = K.view(n_seq, h, d_k).transpose(0, 1)
V = V.view(n_seq, h, d_k).transpose(0, 1)
```

In multi-head attention, we produce *h* slices each of shape *n_seq, d_k* and stack these matrices along the first dimension. *Q, K* and *V* have the shape *h, n_seq, d_k*

# The attention mechanism

Mathematical intuition behind multi-head attention.

```python
import torch
from torch import nn

d_model = 256
n_seq = 1048
h = 8
d_k = d_model // h

x = torch.randn(n_seq, d_model)

W_Q = nn.Linear(d_model, d_model, bias=False)
W_K = nn.Linear(d_model, d_model, bias=False)
W_V = nn.Linear(d_model, d_model, bias=False)
W_O = nn.Linear(d_model, d_model, bias=False)

Q = W_Q(x) # [n_seq, d_model]
K = W_K(x) # [n_seq, d_model]
V = W_V(x) # [n_seq, d_model]

# [h, n_seq, d_k]
Q = Q.view(n_seq, h, d_k).transpose(0, 1)
K = K.view(n_seq, h, d_k).transpose(0, 1)
V = V.view(n_seq, h, d_k).transpose(0, 1)

K_T = K.transpose(-2, -1)

attn_scores = torch.matmul(Q, K_T) / torch.sqrt(torch.tensor(d_k))
attn_weights = nn.functional.softmax(attn_scores, dim=-1)
attn = torch.matmul(attn_weights, V) # [h, n_seq, d_k]

attn = attn.transpose(0, 1).contiguous().view(n_seq, d_model)

output = W_O(self_attention)
```

$$MultiHead(Q, K, V) = Concat(head_1, head_2, \ldots headh)W^O$$

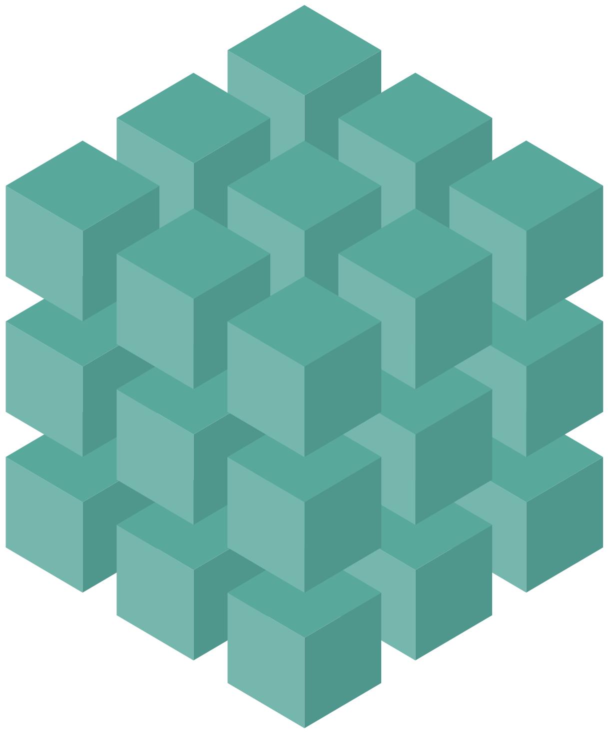
$$head_i = Attention(Qi, Ki, Vi)$$

Multi-head attention learns contextual dependencies from different perspectives in parallel by splitting the **Q**uery, **K**ey and Value matrices into multiple heads.

contiguous(): Ensures the tensor is stored in contiguous memory; doesn't change the shape, but prepares for the view operation

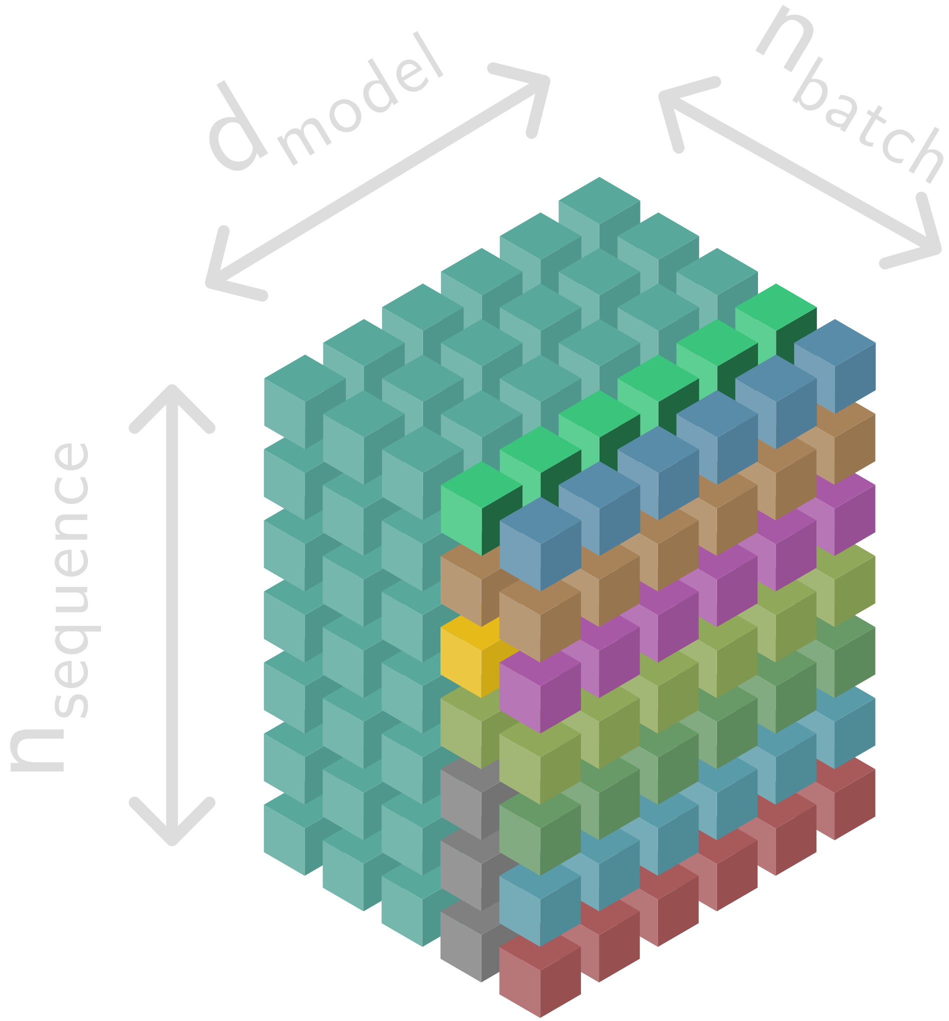
# Batching

Simultaneously process multiple training samples



## Batching

Batching allows us to train on multiple samples in parallel. All sequences need to be padded to the same length.


Note: d_model, n_sequence and n_batch are usually significantly larger; however it generally holds n_batch < d_model < n_sequence