BERT

Bidirectional Encoder Representations from Transformers

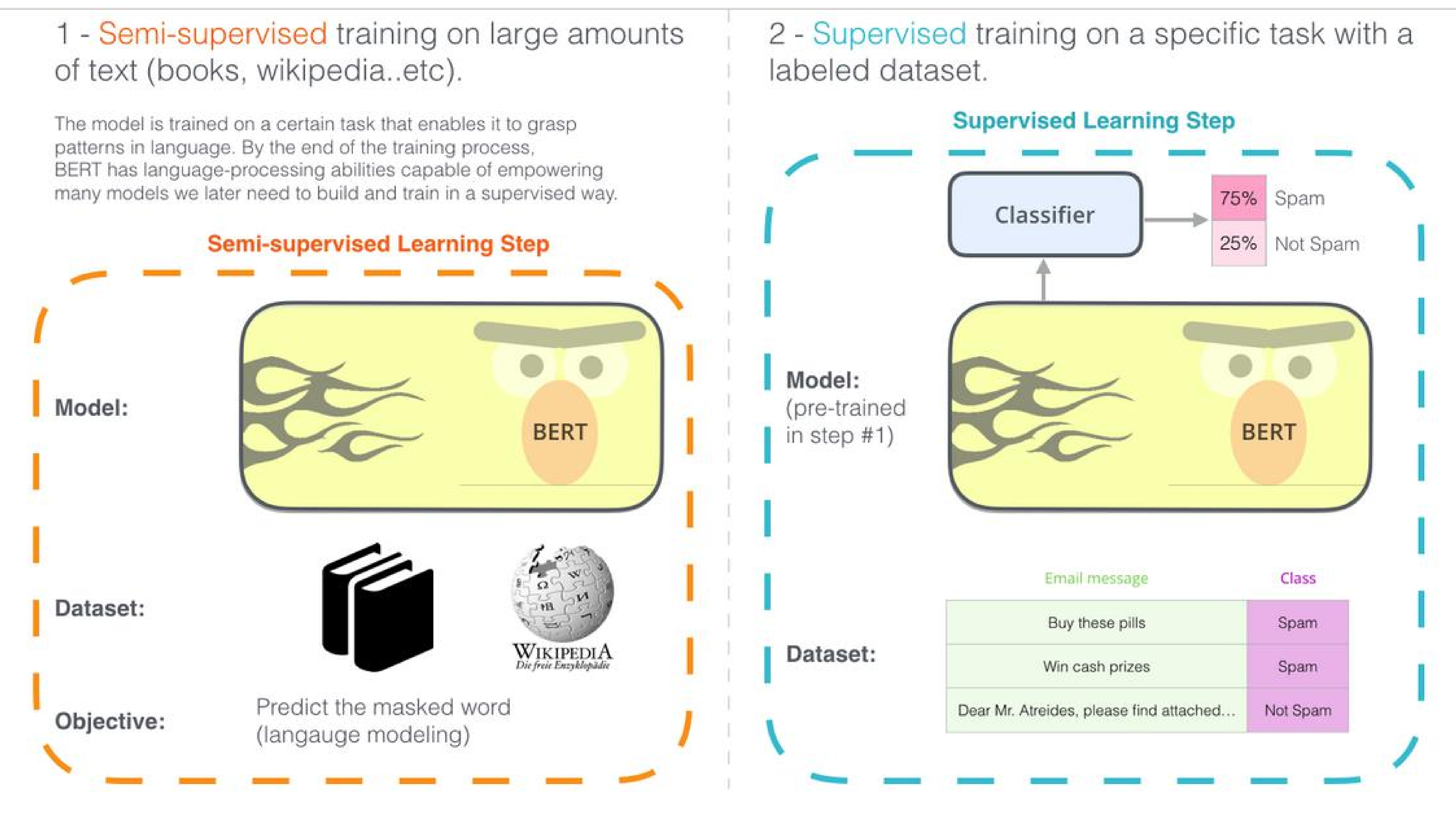
Dr M Janaki Meena

BERT

- In 2018, published by researchers at Google AI Language
- caused a stir in the Machine Learning community by presenting state-of-the-art results in a wide variety of NLP tasks
- broke several records for how well models can handle language-based tasks
- Soon after the release of the paper describing the model, the team also open-sourced the code of the model, and made available for download versions of the model that were already pre-trained on massive datasets

BERT

- Momentous development since it enables anyone building a machine learning model involving language processing to use this powerhouse as a readily-available component – saving the time, energy, knowledge, and resources that would have gone to training a language-processing model from scratch
- Including Question Answering (SQuAD v1.1), Natural Language Inference (MNLI), and others
- Key technical innovation is applying the bidirectional training of Transformer, a popular attention model, to language modelling

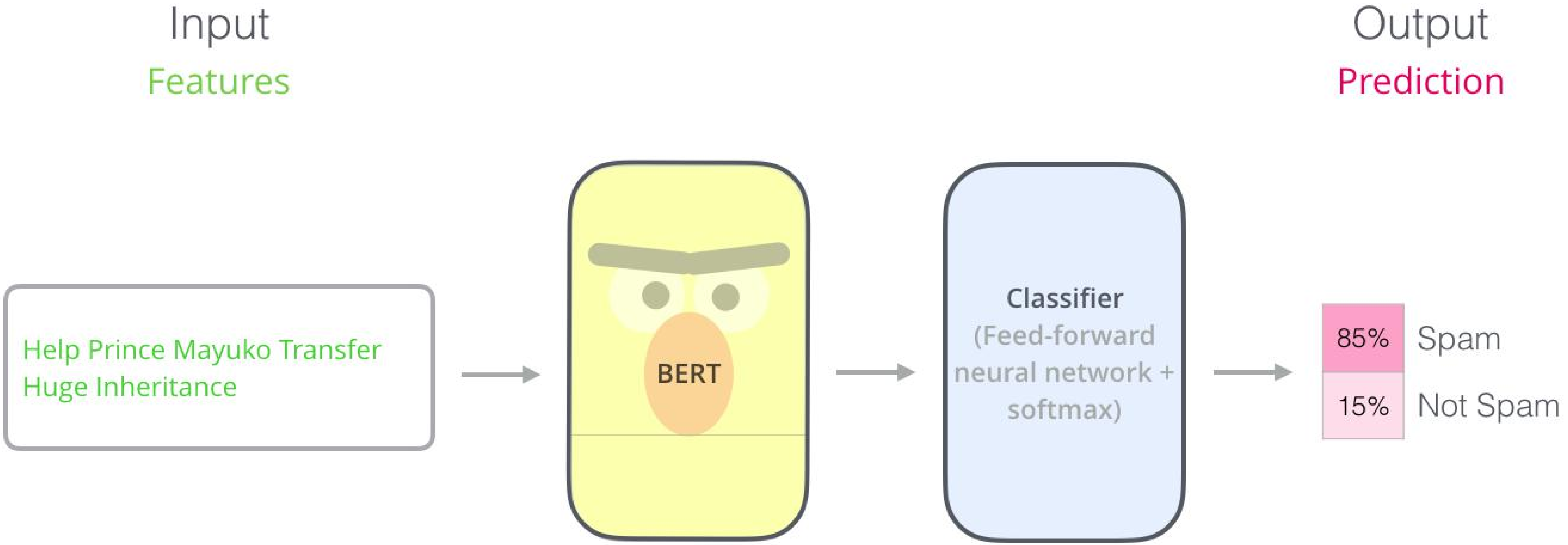


1 - Semi-supervised training on large amounts of text (books, wikipedia..etc).
The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.
Semi-supervised Learning Step
Model:
BERT
Dataset:
WIKIPEDIA
Die freie Enzyklopädie
Objective:
Predict the masked word (langauge modeling)
2 - Supervised training on a specific task with a labeled dataset.
Supervised Learning Step
Classifier
75% Spam
25% Not Spam
Model: (pre-trained in step #1)
BERT
Dataset:
Email message
Class
Buy these pills
Spam
Win cash prizes
Spam
Dear Mr. Atreides, please find attached…
Not Spam

BERT

- Builds on top of a number of clever ideas that have been bubbling up in the NLP community
- Semi-supervised Sequence Learning
- ELMo - AllenNLP
- ULMFiT (by fast.ai founder)
- OpenAI transformer (by OpenAI researchers), and
- the Transformer (Vaswani et al)

Applications of BERT – Sentence Classification



Applications of BERT – Sentence Classification

- To train such a model, you mainly have to train the classifier, with minimal changes happening to the BERT model during the training phase.
- This training process is called Fine-Tuning, and has roots in Semi-supervised Sequence Learning and ULMFiT.

Applications of BERT – Sentiment analysis

- Input: Movie/Product review. Output: is the review positive or negative?
- Example dataset: SST

Applications of BERT – Sentiment analysis

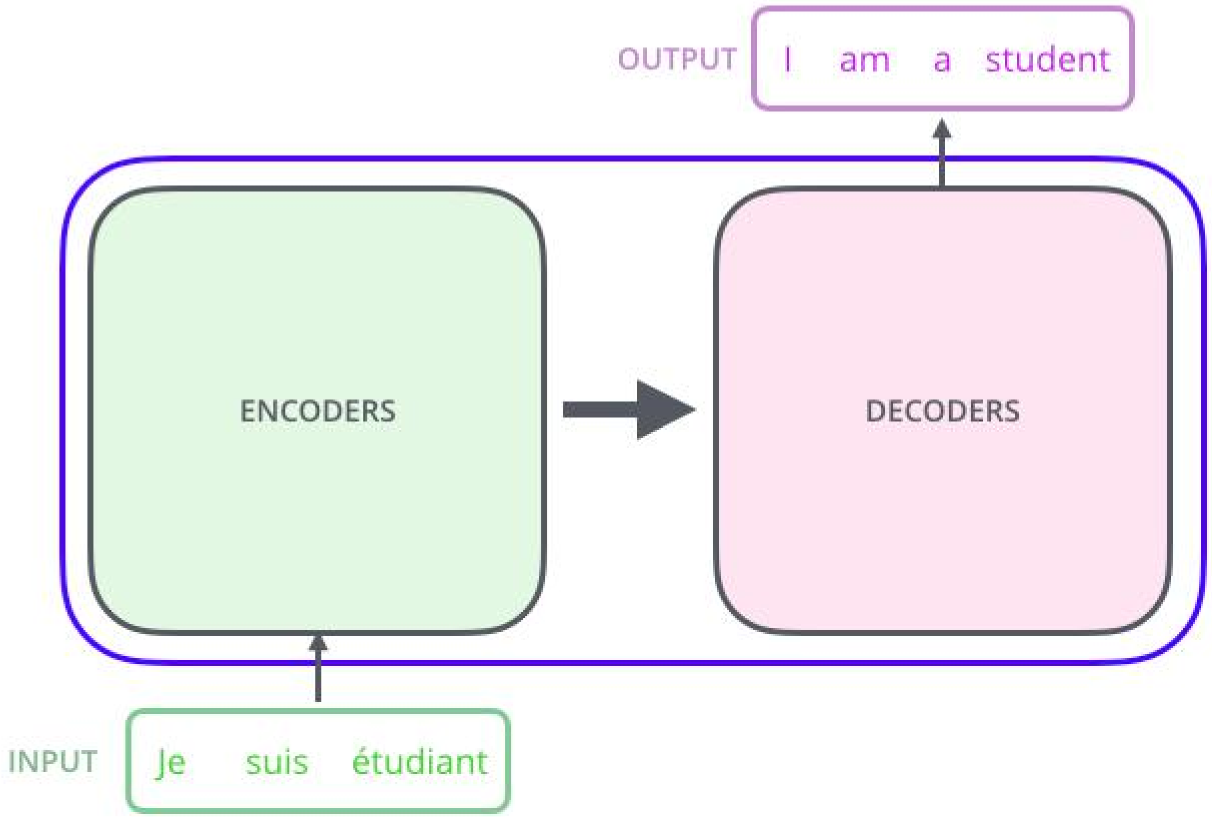
- Fact-checking
- Input: sentence
- Output: “Claim” or “Not Claim”
- More ambitious/futuristic example:
- Input: Claim sentence. Output: “True” or “False”
- Full Fact is an organization building automatic fact-checking tools for the benefit of the public. Part of their pipeline is a classifier that reads news articles and detects claims (classifies text as either “claim” or “not claim”)

Transformer Architecture

- The first transformer model was developed for translating English to German
- Basically it was a sequence to sequence model
- Here we see a illustration of a transformer which will translate a french sentence to English

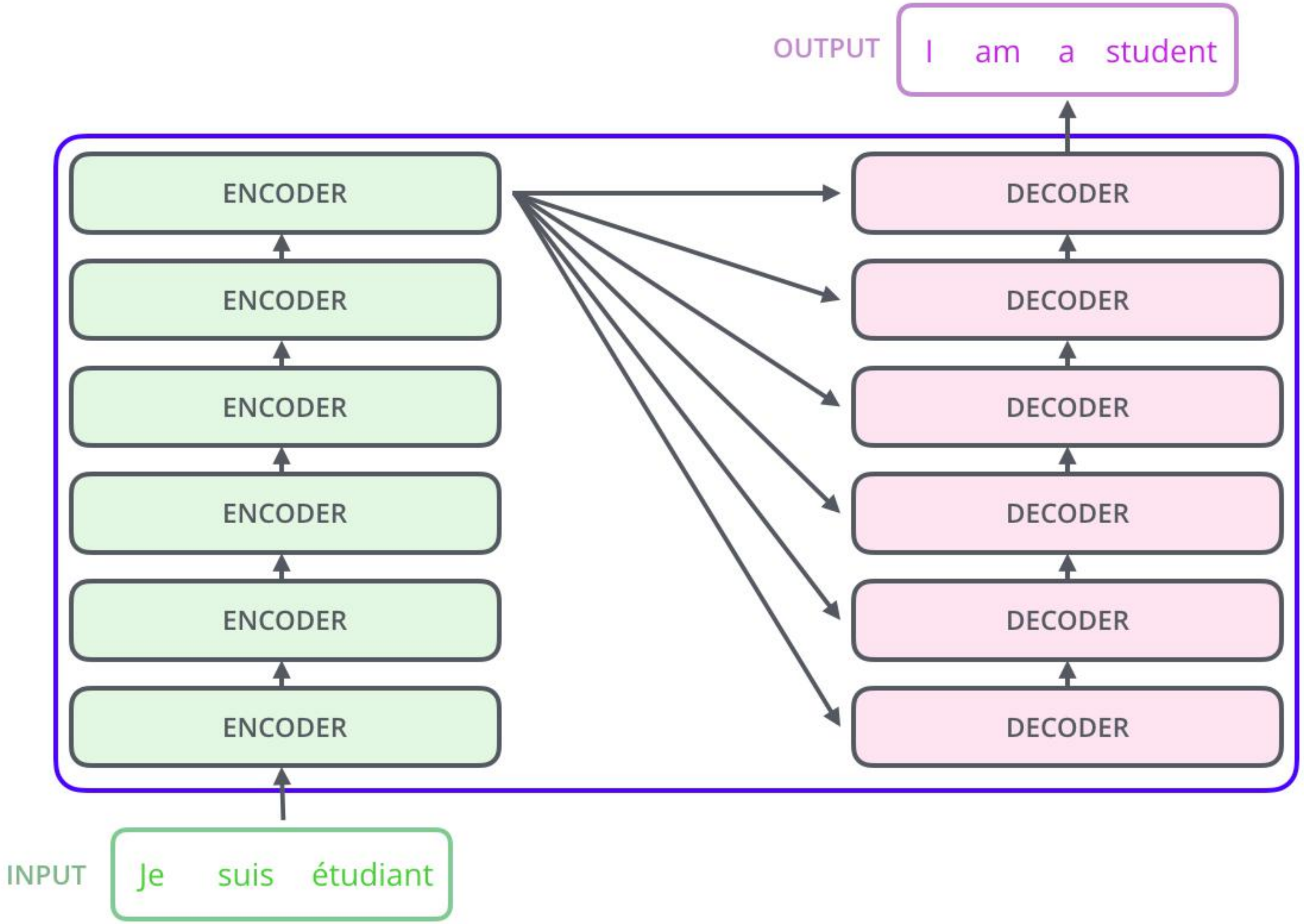
Transformer Architecture

- Transformer has got encoder to encode the input sentence and a decoder decodes the encoded sentence to target sentence



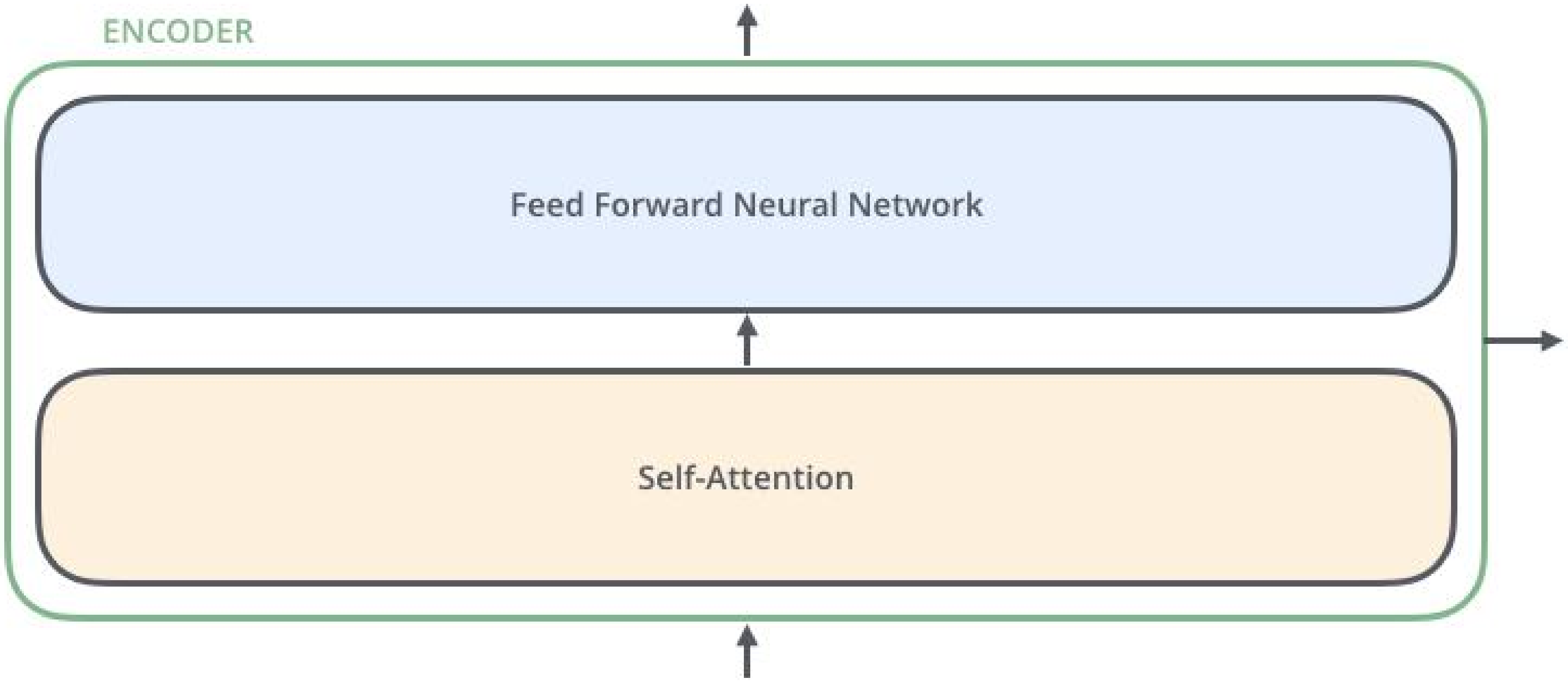
Transformer Architecture

- A set of six encoder and six decoders were stacked in the transformer model



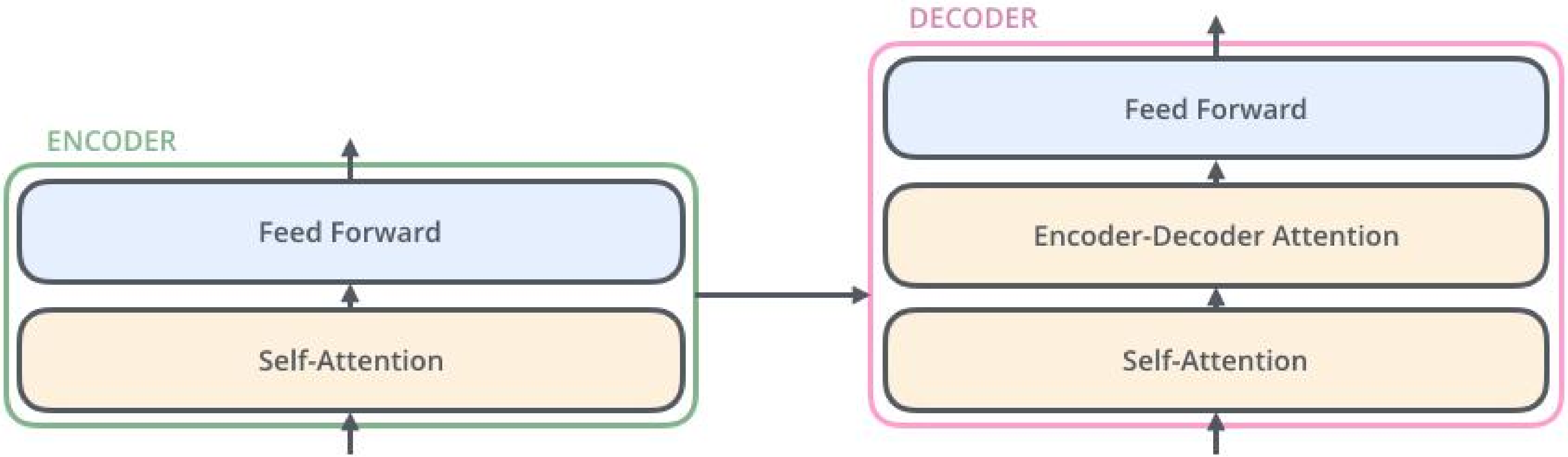
Transformer Architecture

- Each encoder layer consists of a self-attention and feed forward neural network layer



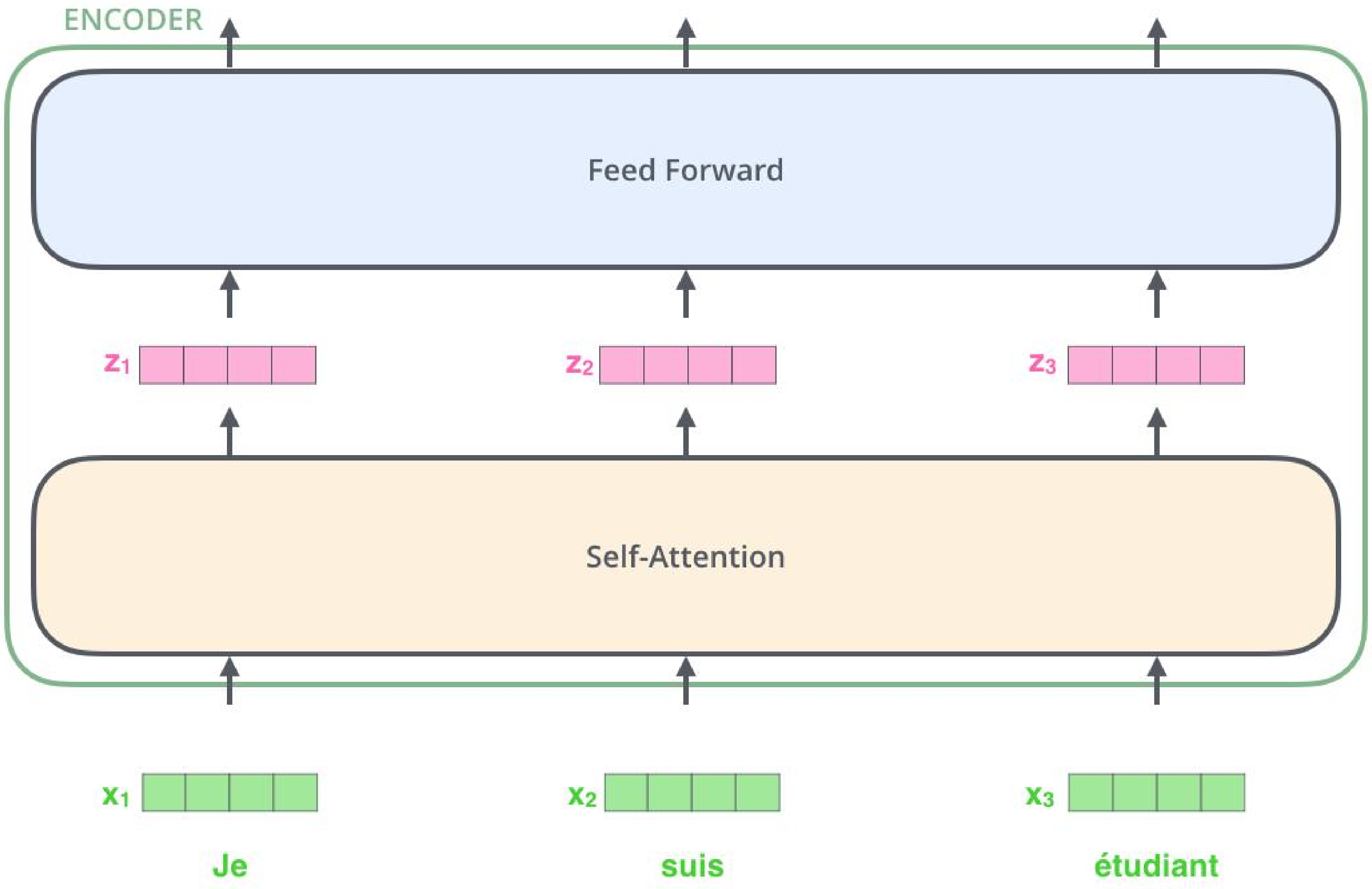
Transformer Architecture

- Each decoder layer consists of a self-attention and feed forward neural network layer and a cross-attention layer also
- Cross attention layer is added so that the input sentence may be referred during translation

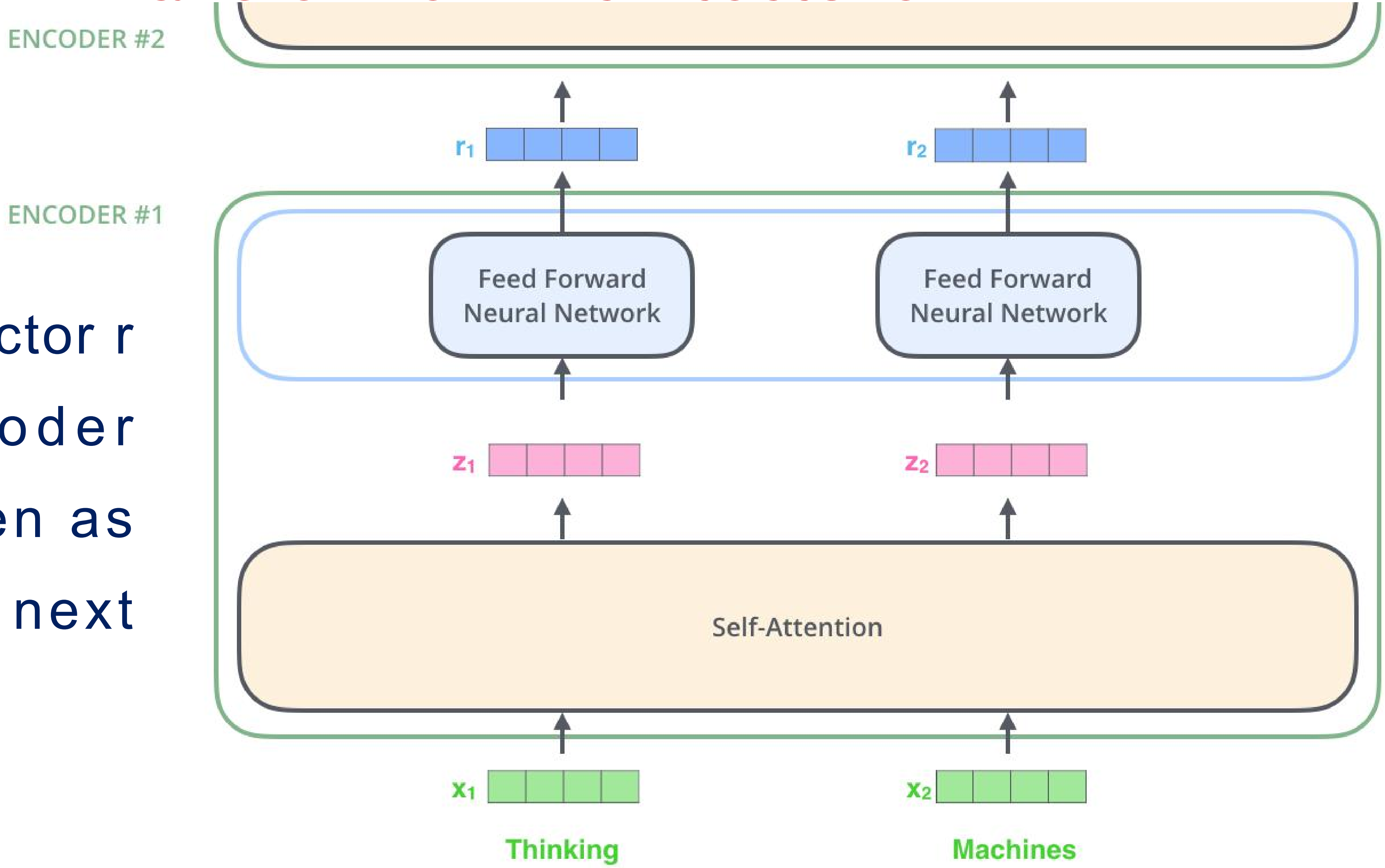


Transformer Architecture

- Resultant vector z is got by adding self-attention to the input embedding vector x
- Feedforward layer of the encoder adds non-linearity to the z vector to give r vector



Transformer Architecture



- Resultant vector r of one encoder layer is given as input to the next layer

Attention

- Attention treats each word's representation as a query to access and incorporate information from a set of values
- Self Attention - Attention within single sentence
- Number of unparallelizable operations doesn't increase with sequence length as it increases in LSTM
- All words attend to all words in the previous layer - So layerwise we cannot parallelize but we can parallelize in a layer
- Maximum interaction distance: $O(1)$, Since all words interact at every layer

Self – Attention

- Attention operates on queries, keys and values, for a sequence of length T
- d - 64 (Some number chosen by authors)
- We have some queries $q_1, q_2, q_3, ..., q_T$. Each query is q_i Є R_d
- We have some keys $k_1,k_2, ..., k_T$. Each value is k_i Є R_d
- We have some values $v_1, v_2, ..., v_T$. Each value is v_i Є R_d
- Number of queries can vary from number of keys and values in practice
- In Self-Attention, the queries, keys and values are drawn from the same source

Self – Attention

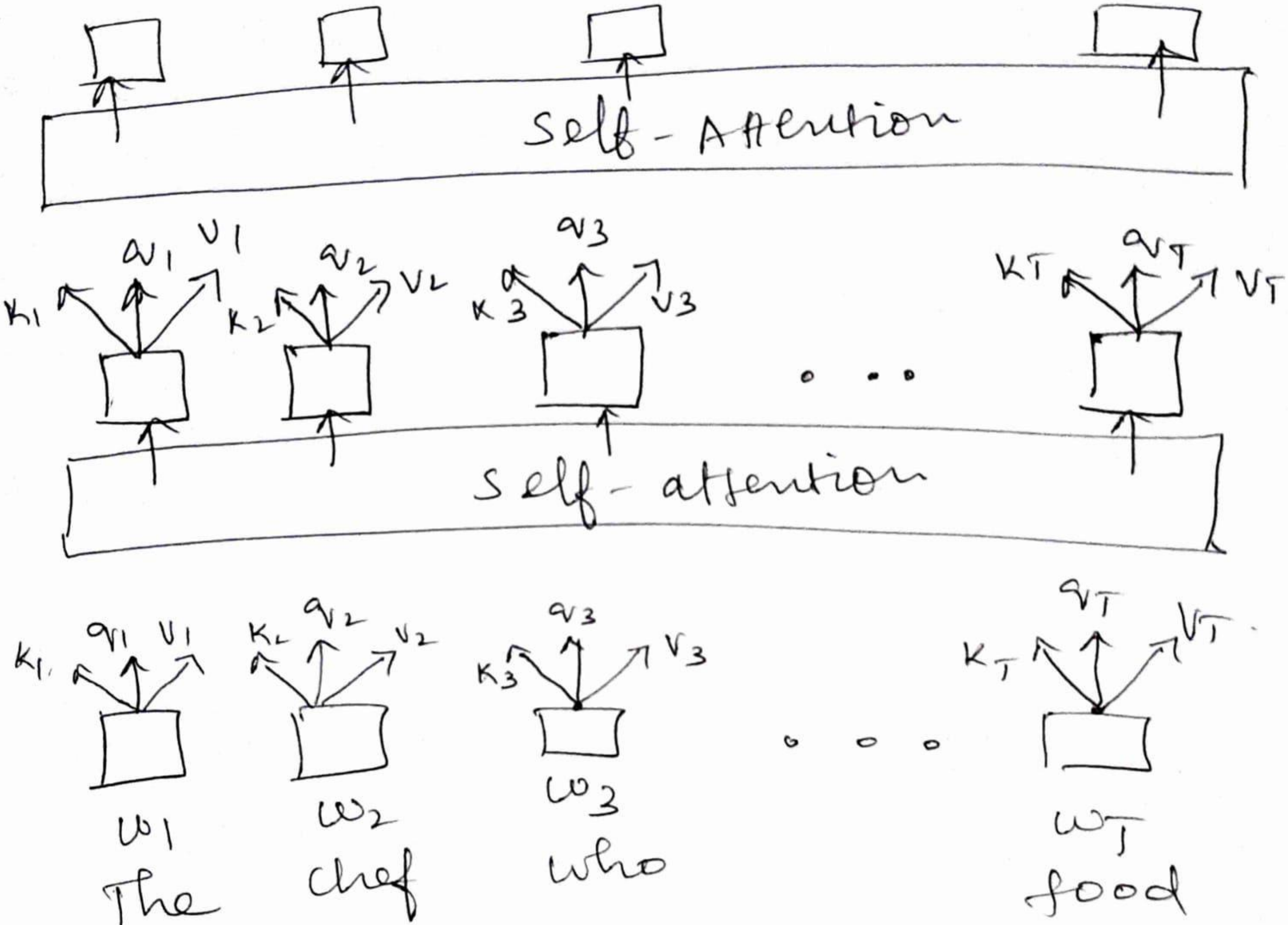
- For example, if the output of the previous layer is x_1, ..., x_T (one vec per word) we could let $v_i = k_i = q_i = x_i$ (use same vectors for all of them (i.e.) embedding of the word)
- The dot-product self-attention operation is as follows:
- Compute key-query affinities - $e_{ij} = q^T{}_i k_j$ - Scalar value not bounded by size
- Compute attention weights from affinities (Apply Softmax)
- $\alpha_{ij} = \exp(e_{ij}) / \Sigma_j' \exp(e_{ij'})$ - Given query sum over all keys for normalization
- Compute output for query as weighted sum of values $\text{output}_i = \Sigma_j \alpha_{ij} v_j$

Self – Attention

- Query is going to interact with keys to produce values
- Can view as query is looking for information in keys
- We connect everything to everything how is different from fully connected network:
- In attention we have to learn interaction weights between query and key vectors and it depends on the input
- Input changes - weights are allowed to change as a function of input
- Interaction weights are dynamic

Self – Attention

- Parameterization is different
- Parameters are computed as dot product of vectors
- We have stacked self-attention blocks, like we can stack LSTM layers

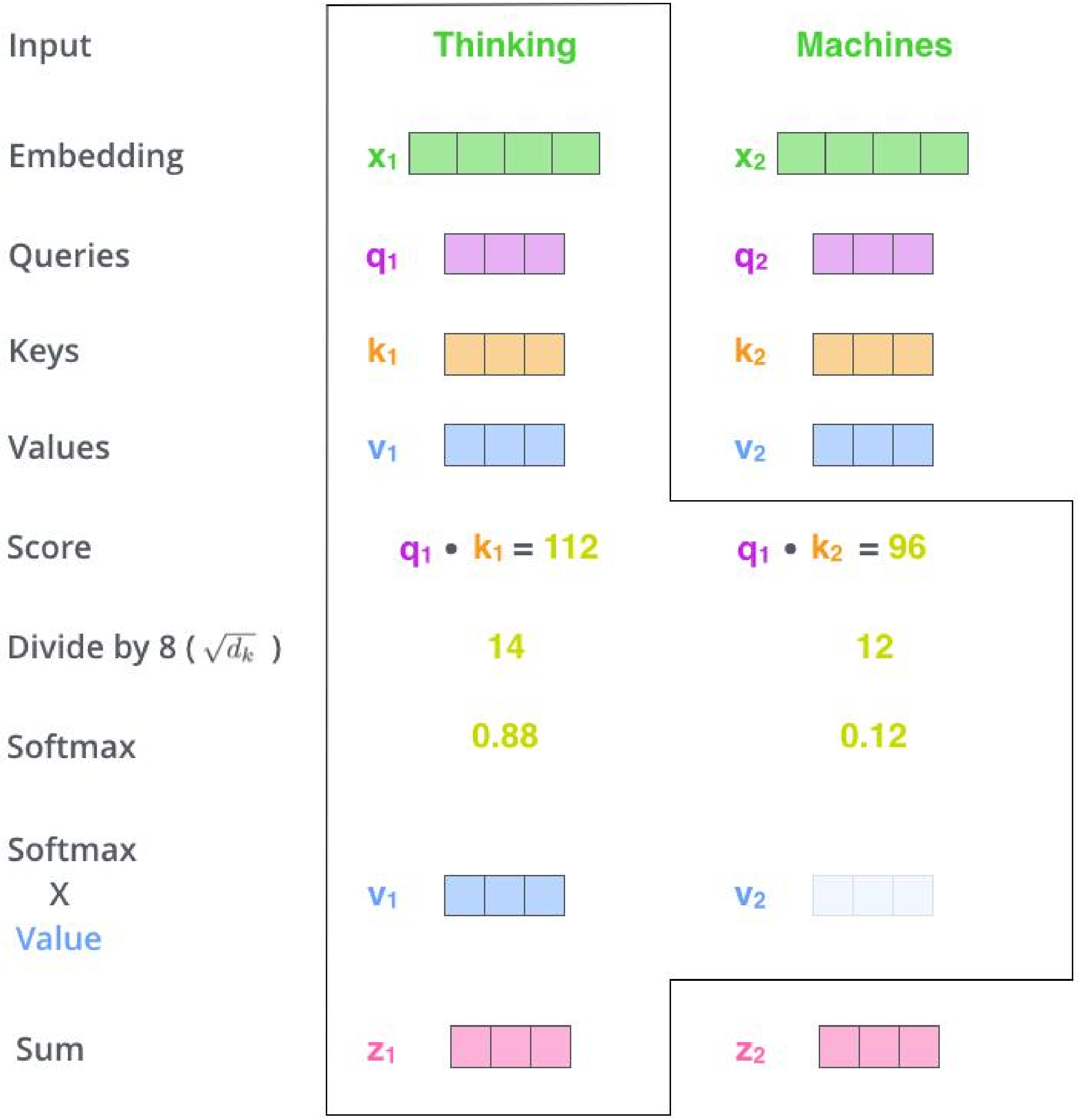


Self-Attention
k1
q1
v1
k2
q2
v2
k3
q3
v3
kT
qT
vT
Self-attention
k1
q1
v1
k2
q2
v2
k3
q3
v3
kT
qT
vT
w1
w2
w3
wT
The
chef
who
food

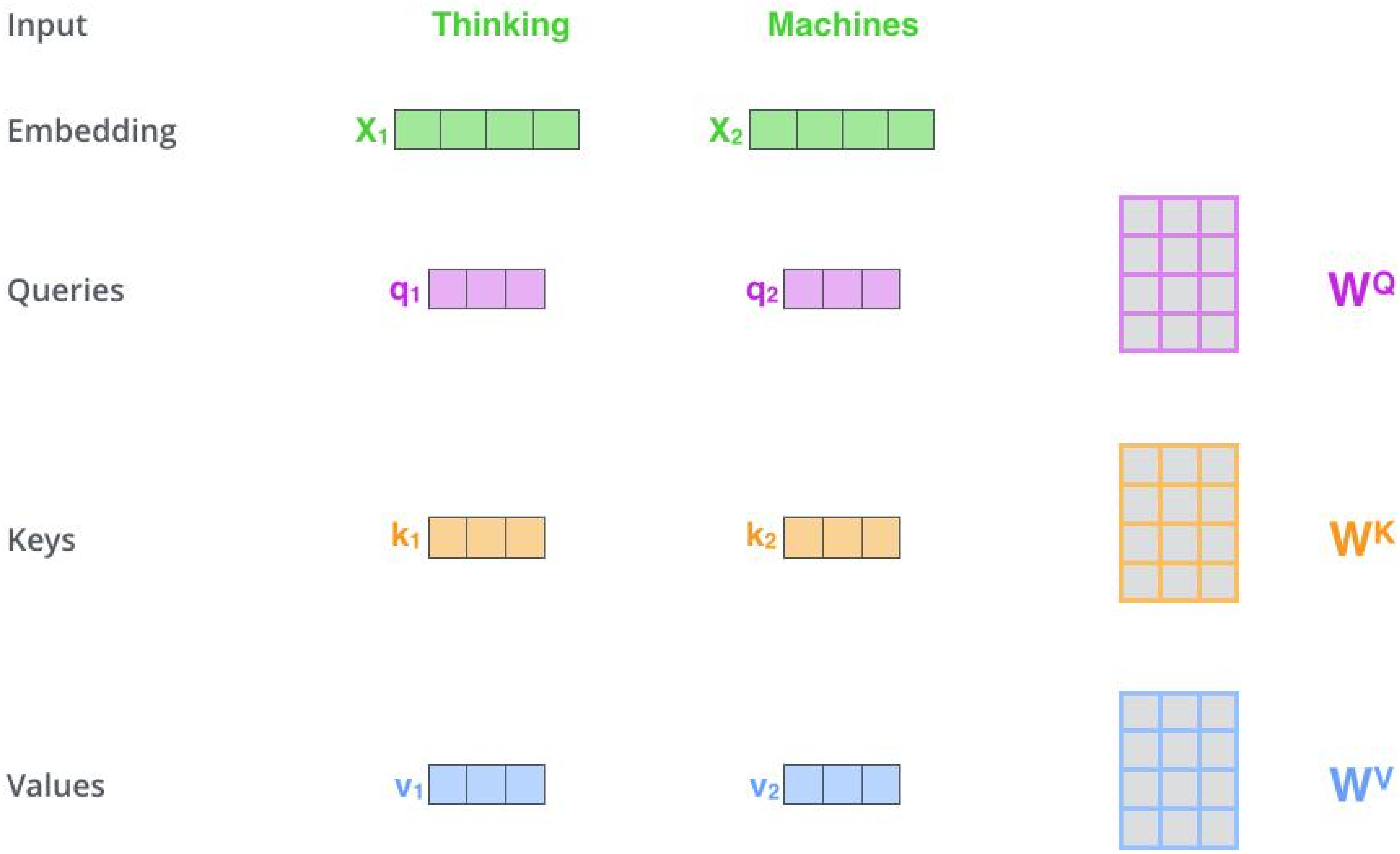
Self – Attention as a NLP building block

- LSTM layers are removed
- Self-attention is a function of keys, queries and values and can be stacked
- After self-attention layer we get new set of queries, keys and values
- Can self-attention can be a drop in replacement for recurrence?
- No ....
- First Self-attention is an operation on sets. It has no inherent notion of order

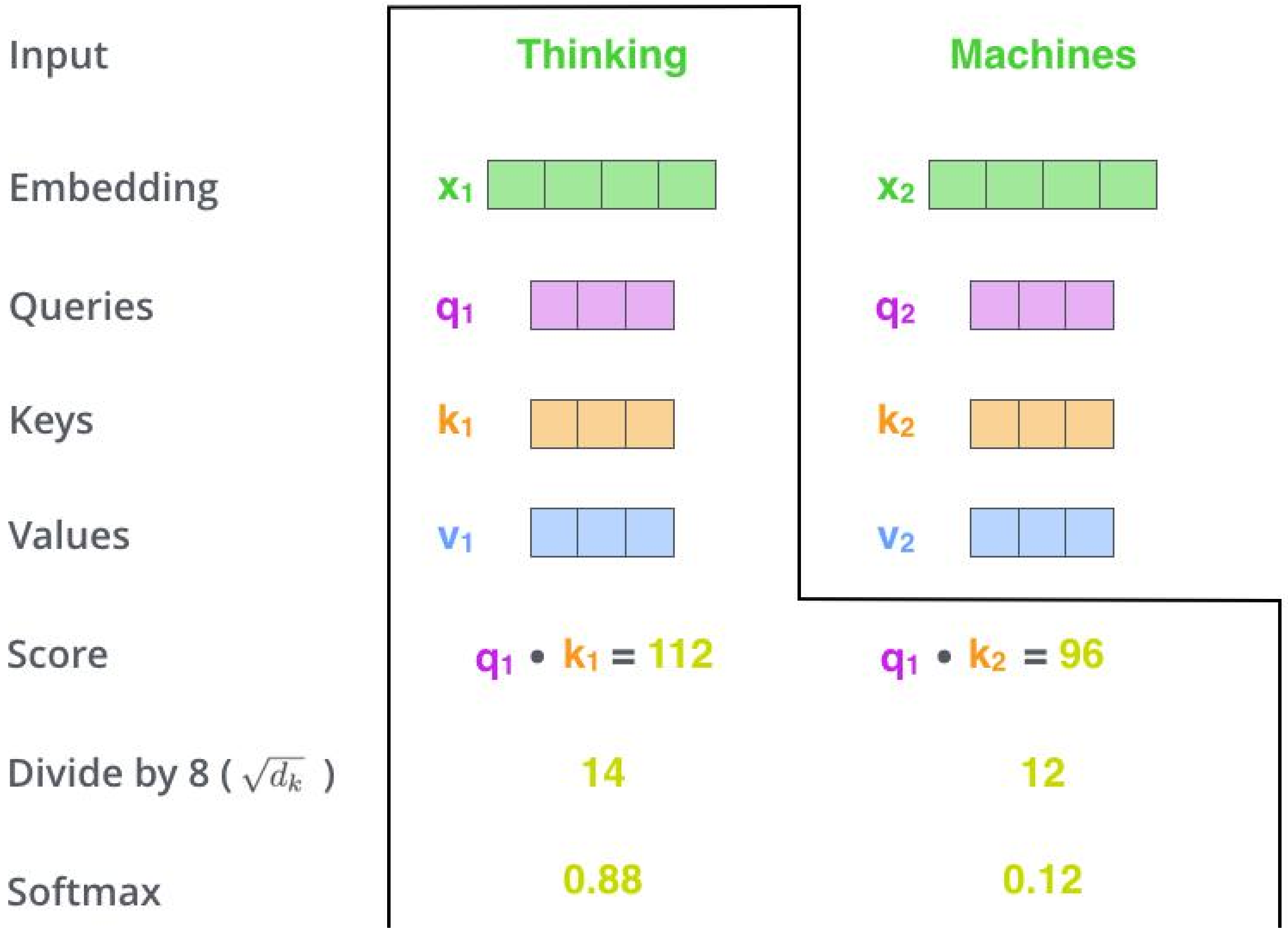
Self Attention – Transformer Architecture



Self Attention – Transformer Architecture



Self Attention – Transformer Architecture



Fixing the first self–attention problem: sequence order

- Since self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries and values
- Consider p_i Є R_d, for i Є {1, 2, ..., T} are position vectors
- To incorporate the position info just add it to our inputs
- In the first layer, let v'_i, k'_i, q'_i be our old keys, values and queries
- $v_i = v'_i + p_i$
- $q_i = q'_i + p_i$
- $k_i = k'_i + p_i$

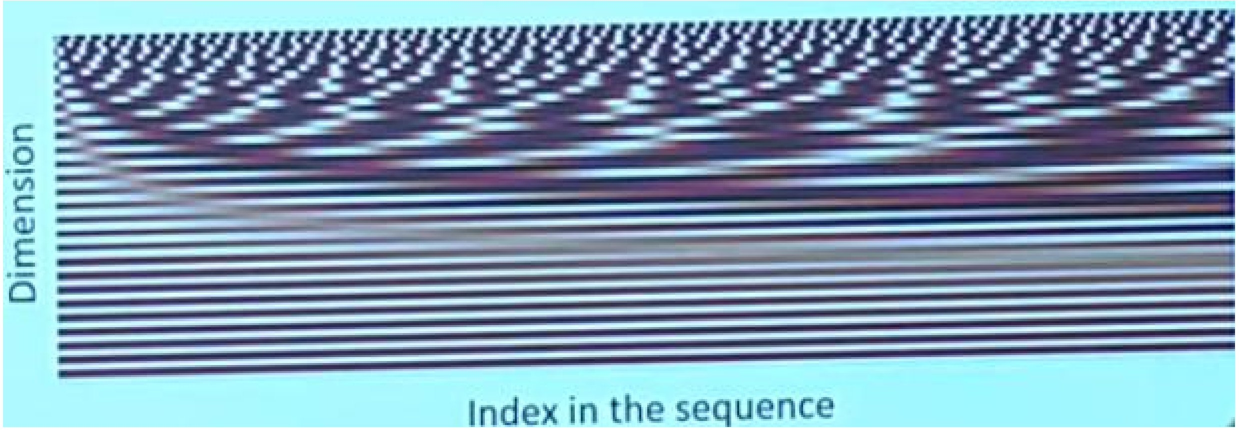
Fixing the first self–attention problem: sequence order

- In deep self-attention networks, we do this at the first layer
- You could concatenate them as well but people mostly just add

Position representation vectors through sinusoids

- Can happen through concatenation of sinusoids of varying periods - varying wave length

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



- Pros
- Perodicity indicates that maybe "absolute position" isn't important
- Maybe can extrapolate to longer sequences as periods restart
- Cons
- Not learnable; also the extrapolation doesn't really work

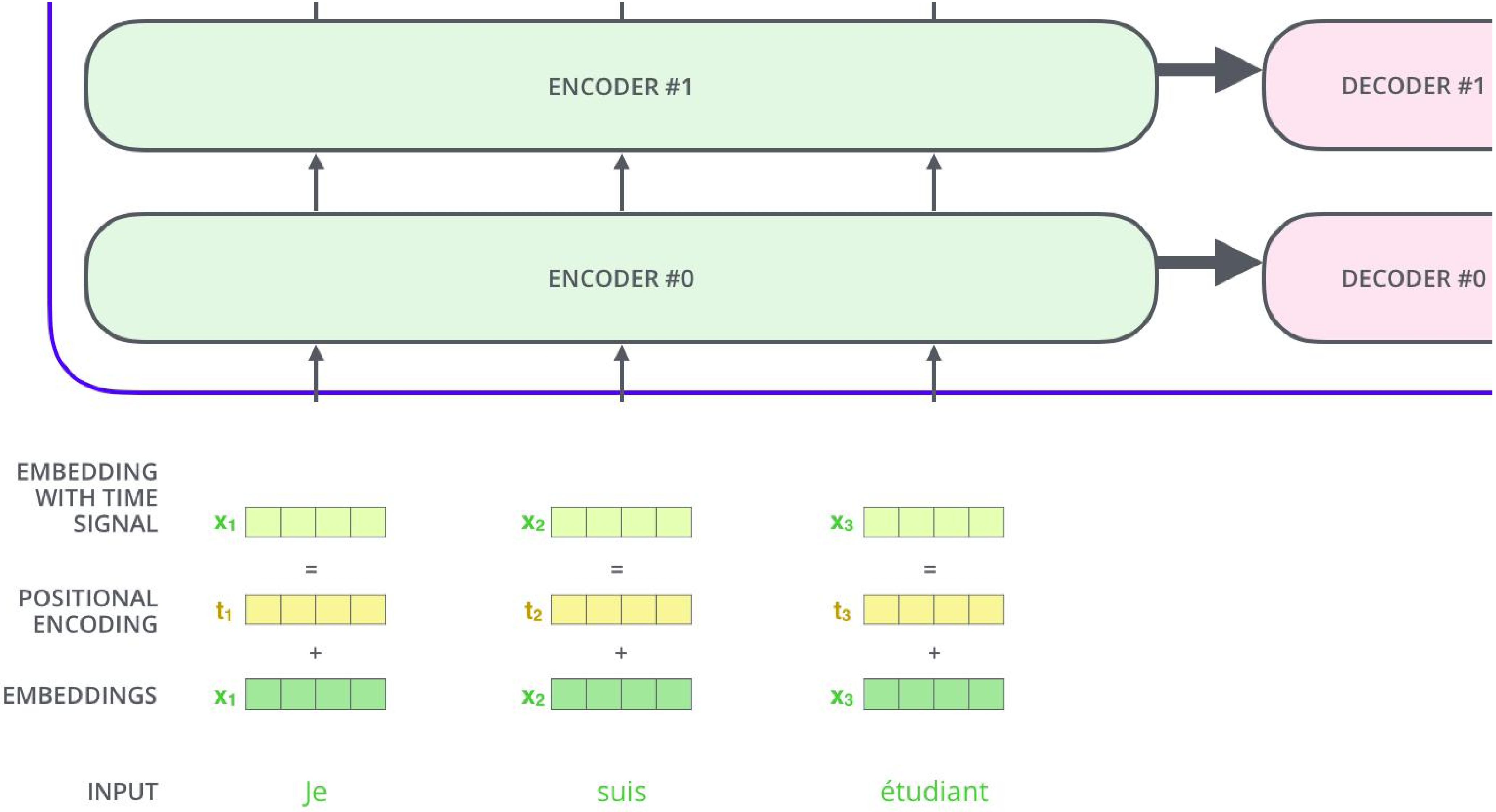
Position representation vectors learned from scratch

- Learned absolute position representation: Let all p_i be learnable parameters!
- Learn a matrix p Є RdxT, and let each pi be a column of that matrix
- Pros
- Flexibility - each position gets to be learned to fit the data
- Cons
- Definitely can't extraplorate to indices outside 1, ..., T
- Most systems use this!

Position representation vectors learned from scratch

- Sometimes people try more flexible representations of position
- Relative linear position attention
- Dependency syntax-based position
- Problem 1 of self-attention to replace LSTM is it doesn't have an inherent order of notion - Solved by - adding position representation to the inputs
- Problem 2 - No non-linearities for deep learning - It's all just weighted averages

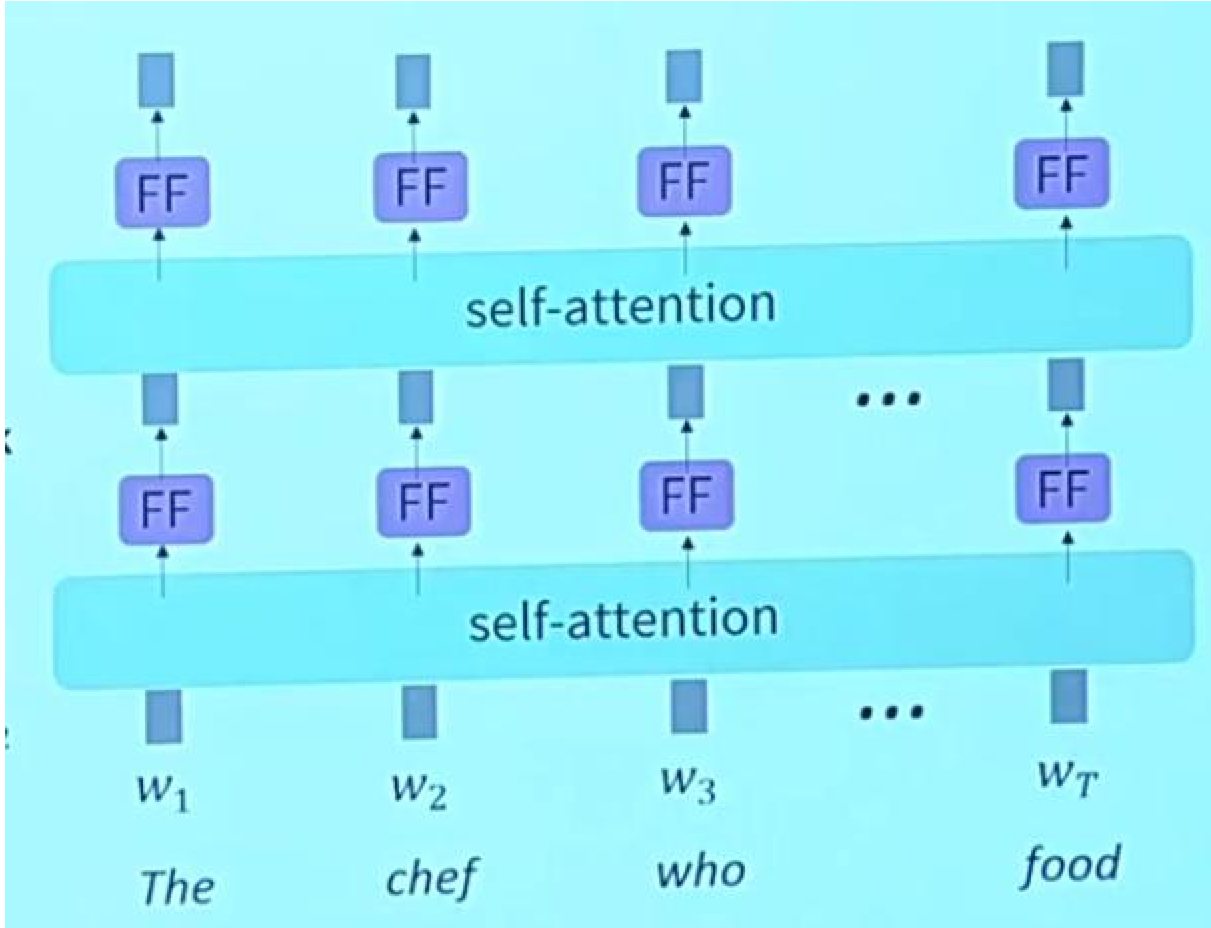
Positional Encoding



Adding non–linearities in Self Attention

- No elementwise non-linearities in self-attention stacking more self-attention layers just re-averages value vectors
- Easy fix is to add a feed-forward network to post-process each output vector
- Intution - Feed forward network processes the result of self-attention

Adding non–linearities in Self Attention



Don't look Future during Prediction

- Need to ensure that we don't look at the future when predicting a sequence
- Like in machine translation or language modeling
- In recurrent networks it is so natural, we don't unroll it further

Masking the Future in Self–Attention

- Important on the decoder side
- One idea - At every timestep change the set of keys and values to include only past words (inefficient!)
- To enable parallelization - We mask out attention to future words by setting attention score to $-\infty$
- $e_{ij} = q^T{}_i k_j$, k<j
- $= -\infty$, k>=j

Matrix of e_{ij} values

We can look at these (not greyed out) words

For encoding these words

	[START]	The	chef	who
[START]	$-\infty$	$-\infty$	$-\infty$	$-\infty$
The		$-\infty$	$-\infty$	$-\infty$
chef			$-\infty$	$-\infty$
who				$-\infty$

Barriers and Solutions for Self–Attention as a Building Block

Barriers	Solutions
Doesn't have an inherent notion of order	Add position representations to the inputs
No non-linearities for deep learning magic! It's all just weighted averages	Apply same feedforward network to each self-attention output
Ensure we don't look at future when predicting sequence - on decoder side during translation	Mask out the future by artifically setting attention weights to 0

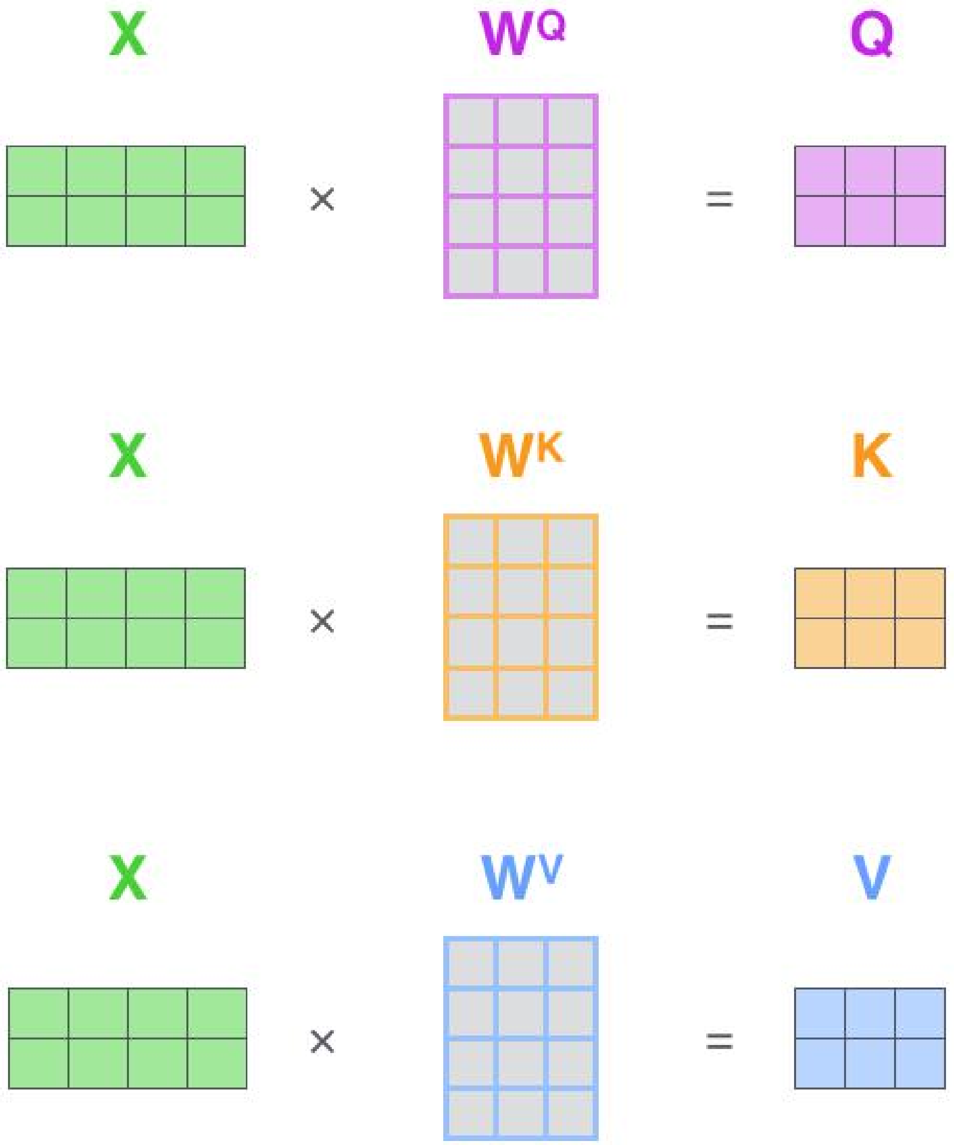
Necessities for a Self–Attention Building Block

- Self-Attention - Basis of methods
- Position representations - Specify sequence order
- Non-linearities - at output of self-attention
- Masking - to parallelize operations while looking at the future

Key–query–value attention

- How do we get the k, q and v vectors from a single word embedding
- $k_i = W_k * x_i$, where $W_k \in R^{dxd}$ is the key matrix
- $q_i = W_q * x_i$, where $W_q \in R^{dxd}$ is the query matrix
- $v_i = W_v * x_i$, where $W_v \in R^{dxd}$ is the value matrix
- Where d - dimension of the hidden layer - Which was 512 for transformer model
- Matrices W_k, W_q and W_v can be very different from each other
- These matrices allow different aspects of the x vectors to be used/emphasized in each of the three roles

Matrix Calculation of Self-Attention



Key–query–value attention

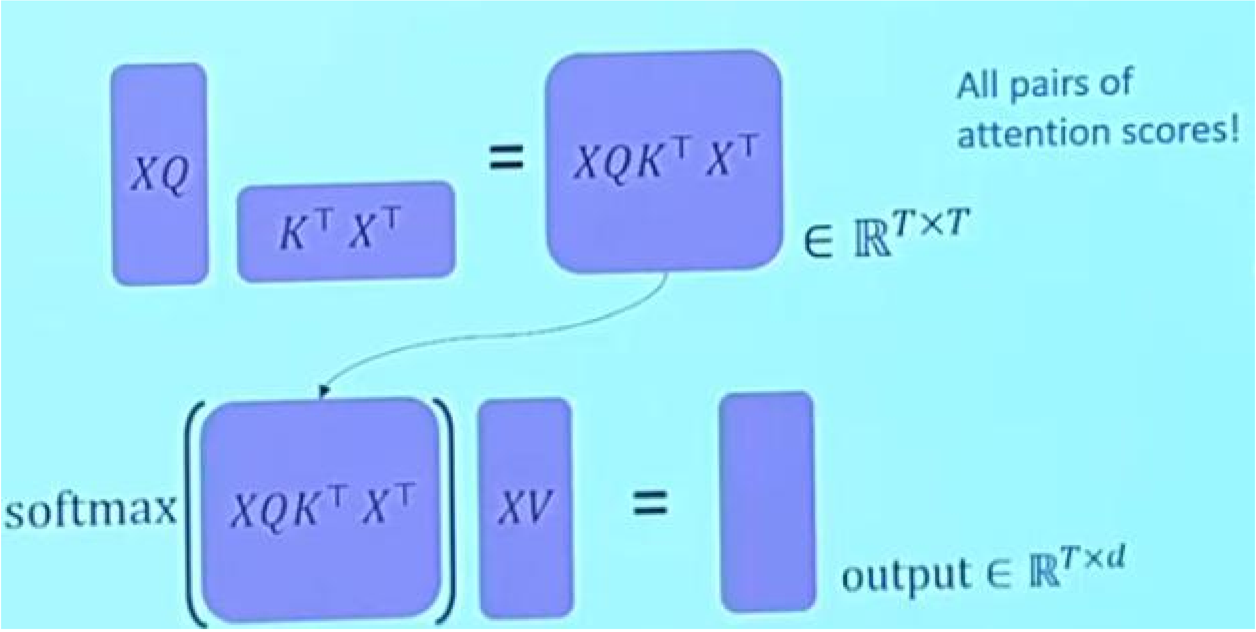
- k and q vectors helps to figure out where to look for different part of x
- v vector - Some information is passed along and it helps to access the information

Key–query–value Attention Computation

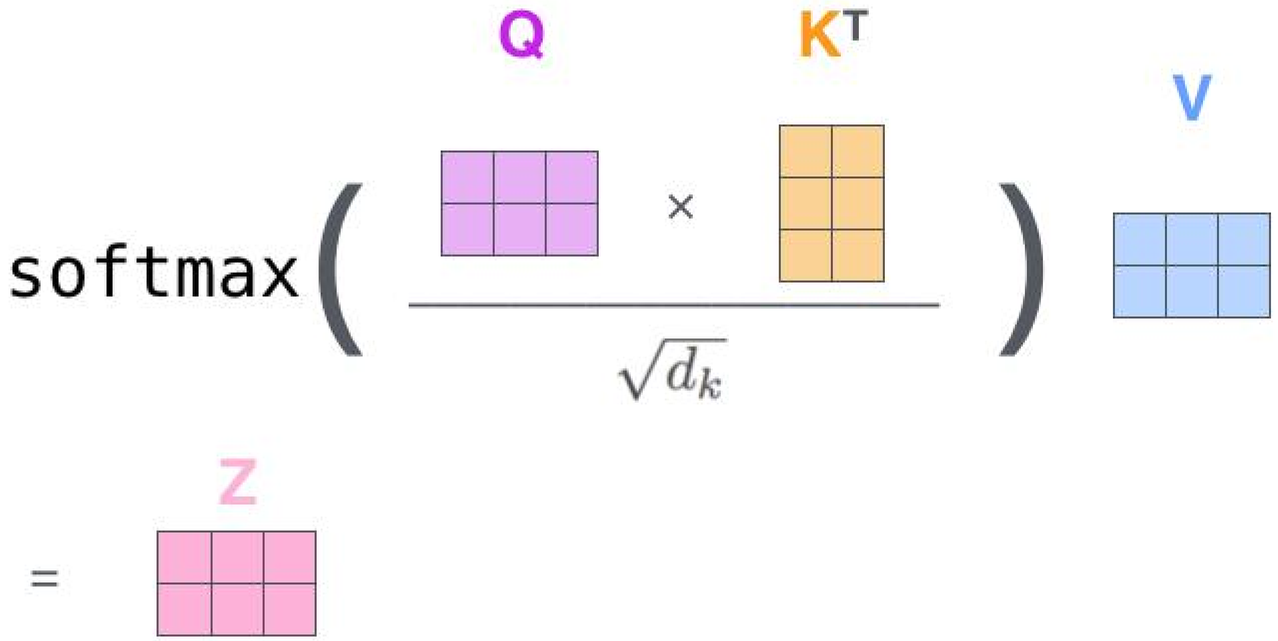
- Computed with big tensors
- $X = [x_1; x_2; \ldots x_T] \in R^{Txd}$ be the concatenation of input vectors
- $X*K \in R^{Txd}$, $X*Q \in R^{Txd}$, $X*V \in R^{Txd}$
- Output tesor is same dimension as X, R^{Txd}
- Output = $(X*Q(X*K)^T) * (X*V)$
- Affinity between key and input is calculated and then averaged

Key–query–value Attention Computation

- Take query-key dot products in one matrix multiplication
- $XW_Q(XW_K)T$
- Softmax and compute the weighted average with another matrix multiplication



Matrix Calculation of Self-Attention



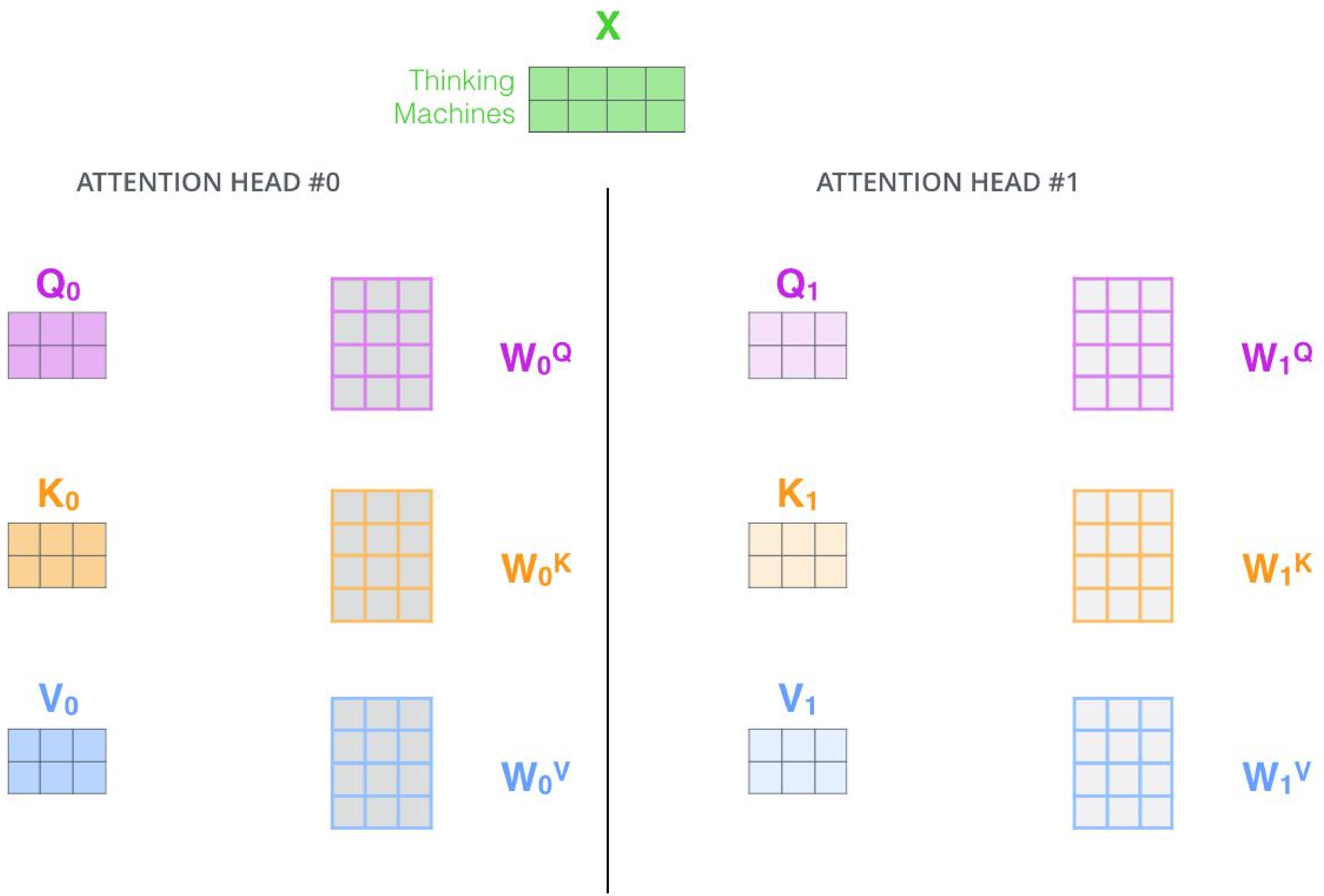
Multi–head Attention

- Attend to multiple places in a single layer
- We want to look at multiple places in the sentence at once
- In single attention, we consider only the point where x^T_i, $Q^T K x_j$ is high
- We encode different things via different query, key and value matrices
- If we are to have h - attentions then we will h number of query, key and value vectors for each word in the sequence with dimensionality as follows:
- Q_l, k_l, v_l Є $R^{d \times d/h}$, l ranges from 1 to h

Multi–head Attention – Focus on many Positions

- Smaller key, value and query matrices with lesser number of columns are made for each head
- Same amount of computation as single–head self–attention

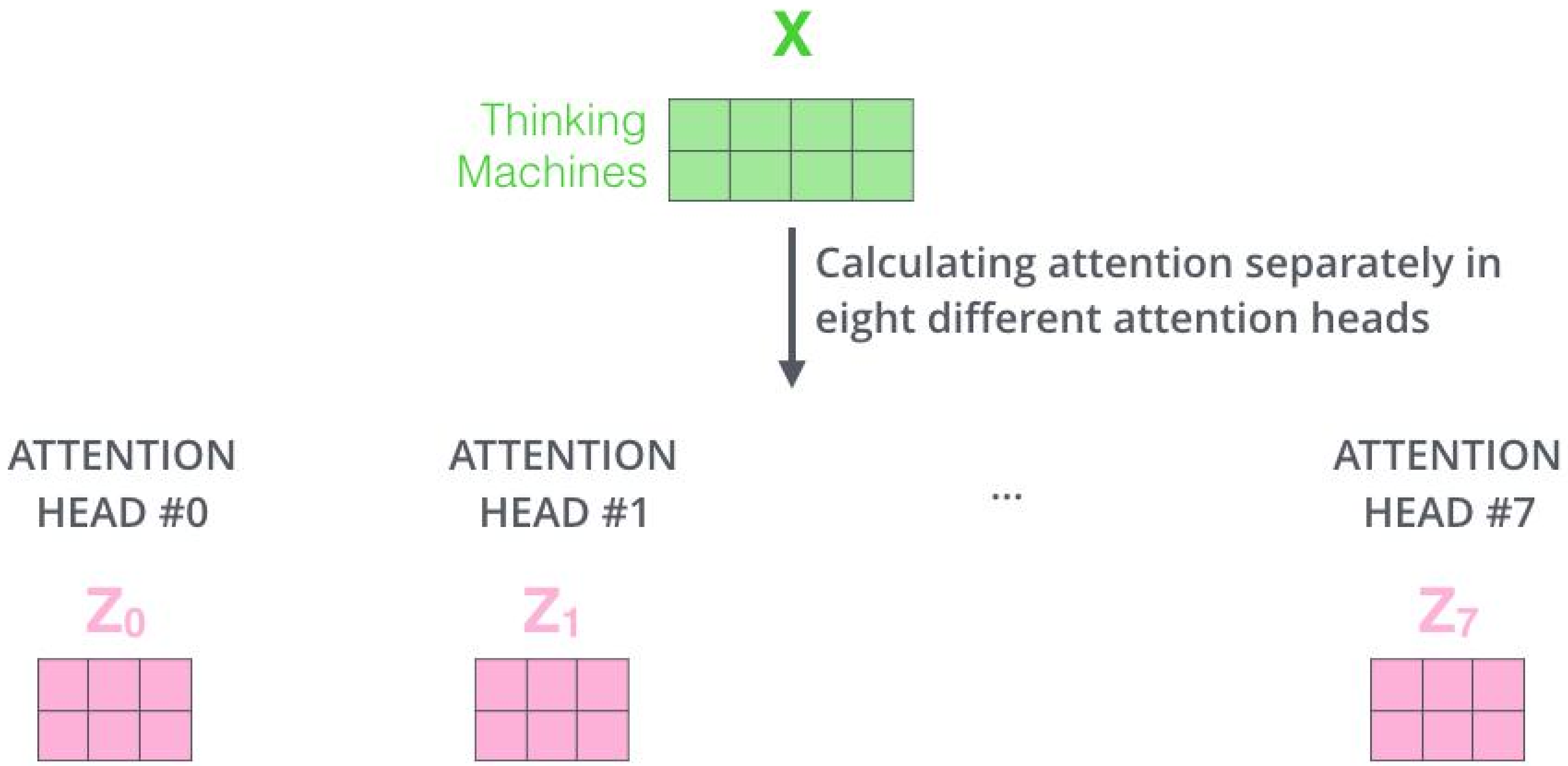
Multi–head Attention – Focus on many Positions



Multi–head Attention

- $Output_l = Softmax(XQ_l\ K^T_l\ X^T) * XV1$ where $output_l \in R^{d/h}$
- $Q_l, k_l, v_l \in R^{d\ X\ d/h}$, l ranges from 1 to h
- Each attention head performs attention independently
- $z_l = softmax(XQ_l K^T_l X^T) * XV_l$, where $Output_l \in R^{d/h}$

Multi-head Attention – Focus on many Positions

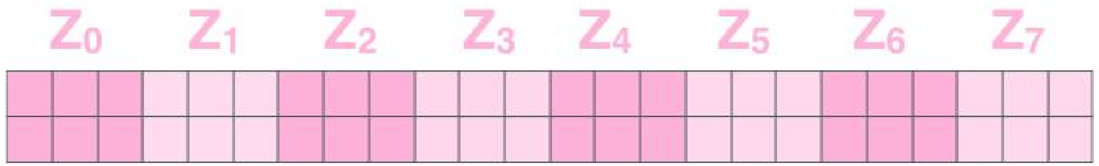


Multi–head Attention

- Then output of all heads are combined by concatenation
- $Z = Wo * [z_1; \ldots ;z_h]$ where $Wo \in R^{d \times d}$
- Wo is a learned weight matrix
- Each head look at different things and construct value vectors differently

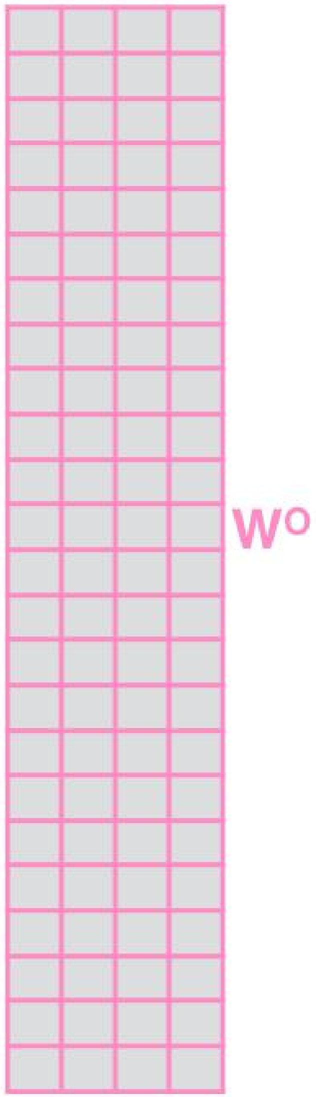
Multi–head Attention – Pictorial Representation

1) Concatenate all the attention heads



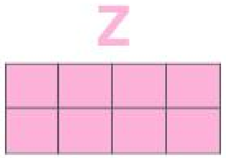
2) Multiply with a weight matrix W^O that was trained jointly with the model

X

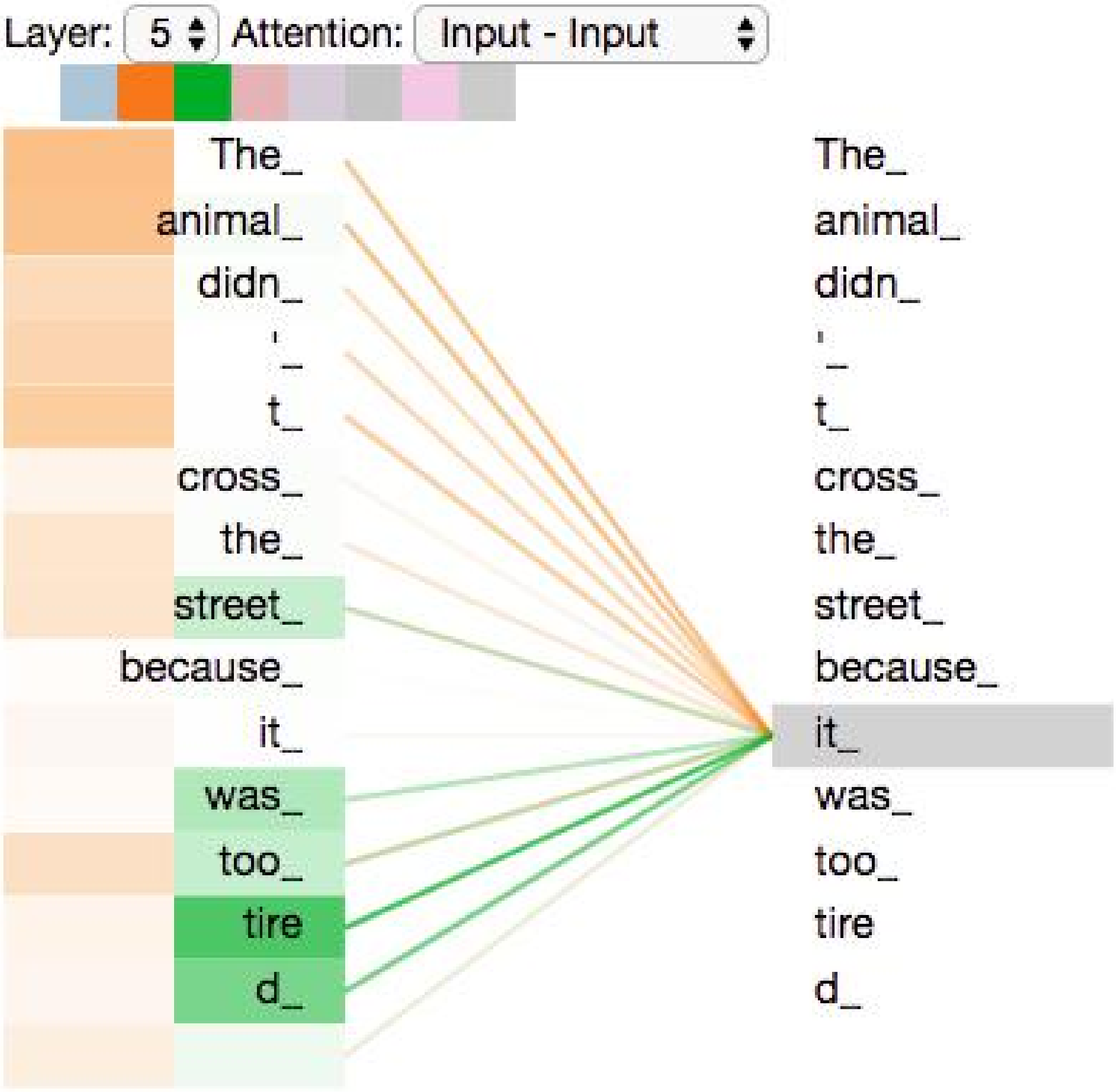


3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=



Multi-head Attention – Focus on many Positions

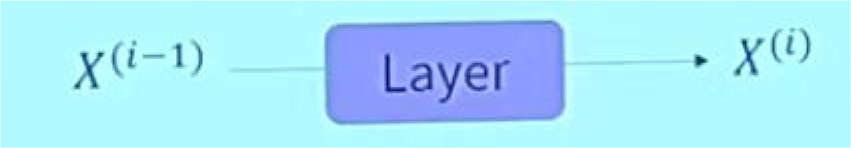


Tricks to help with training

- Residual connections
- Layer normalization
- Scaling the dot product
- These tricks do not improve what model is able to do; they improve the training process. Both of these types of modeling improvements are very important

Residual Connections

- Instead of $X^{(i)} = \text{Layer}(X^{(i-1)})$ (where i represents the layer in depth)

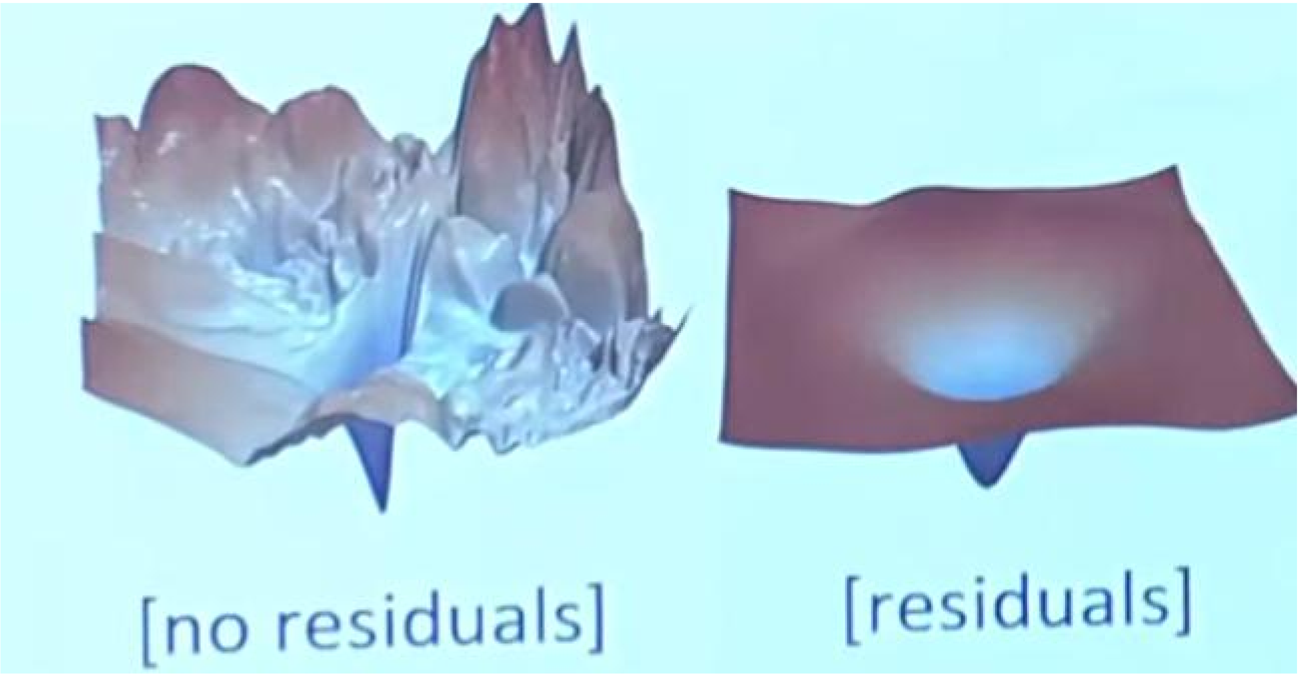


- $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$ (where i represents the layer in depth)

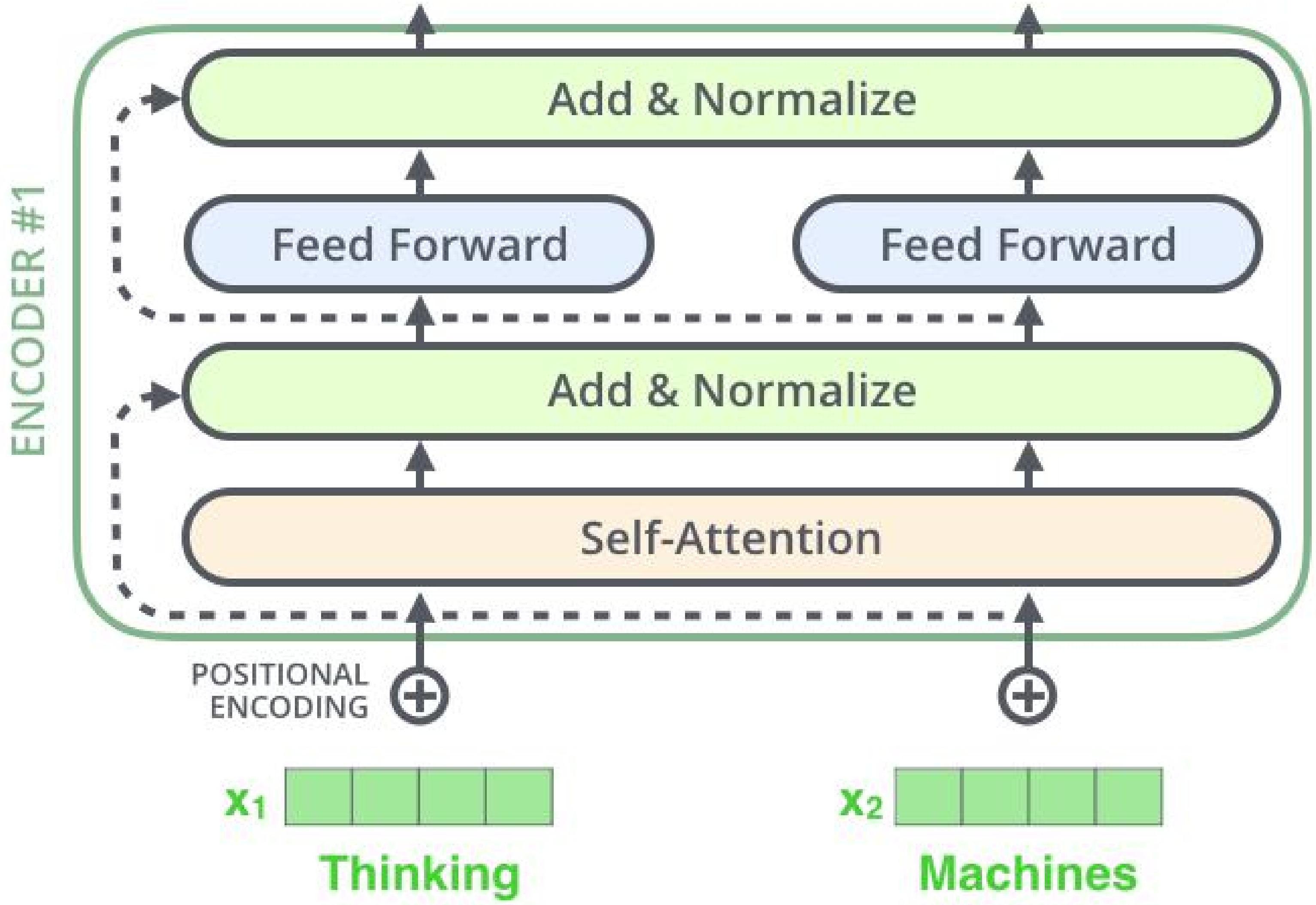
- So we only have to learn "the residual" from the previous layer
- (i.e.) learn only how layer 'i' is different from layer 'i–1'

Residual Connections

- Residual connections make the loss landscape considerably smoother
- Gradients do not affect the backpropagation so much when residual connections are added – Introdcued in ResNet



The Residuals



Normalization

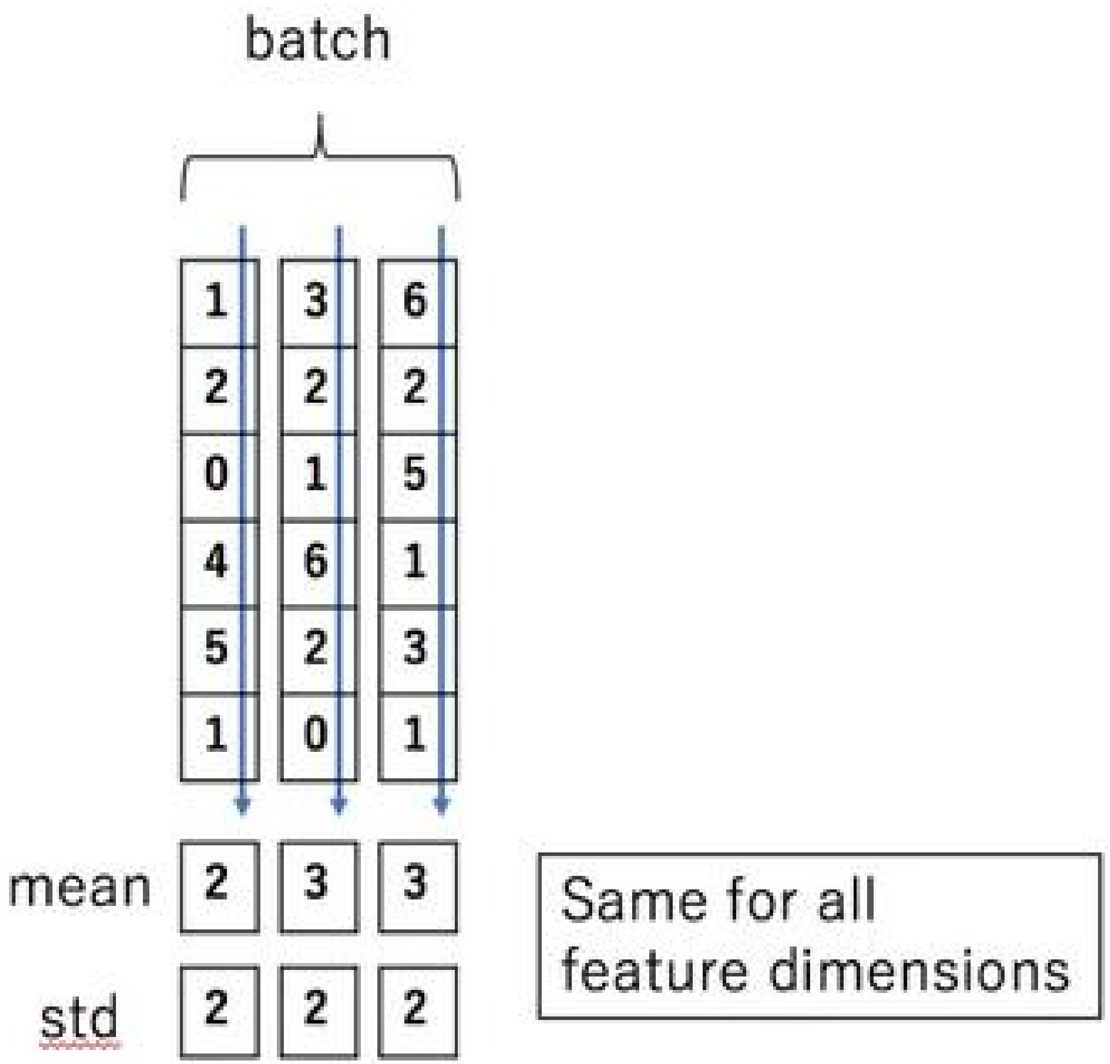
- Provide a uniform scale for numerical values
- If the dataset contains numerical data varying in a huge range, it will skew the learning process, resulting in a bad model
- The normalization method ensures there is no loss of information and even the range of values isn't affected

Normalization

- Elements in a vector x may be normalized by subtracting the mean and dividing by the standard deviation.

$$\hat{x} = \frac{x - mean(\bar{x})}{std(\bar{x}))}$$

Layer Normalization



Layer Normalization

- During forward pass cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation within each layer
- Layer norm's succes may be due to its normalizing gradients
- Let x ∈ R^d be an individual (word) vector model
- Let mean be $\mu = \sum_{j=1}^{d} x_j$ where $\mu \in \mathbb{R}$
- Let standard deviation be $\sigma = \sqrt{\frac{1}{d} \sum_{j=1}^{d} (x_j - \mu)^2}$ where $\sigma \in \mathbb{R}$
- Optional learned gain and bias parameters $\gamma \in \mathbb{R}^d$ and $\beta \in \mathbb{R}^d$ may be used in compuation

Layer Normalization

$$\text{output} = \frac{x - \mu}{\sigma + \epsilon} * \gamma + \beta$$

Normalize by scalar mean and variance

Modulate by learned elementwise gain and bias

Scaled Dot Product Attention

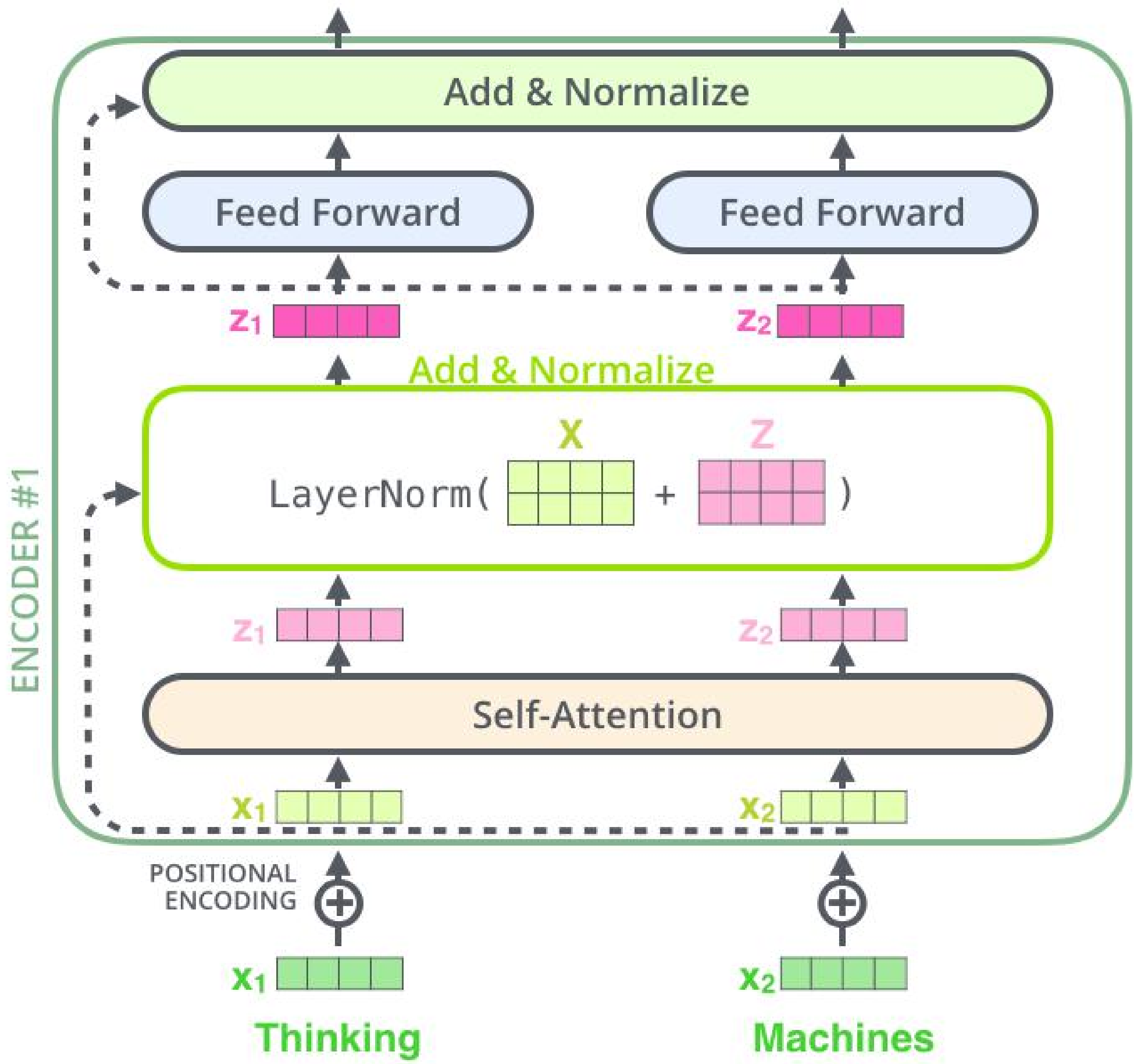
- Final variation to aid transformer training
- When dimensionality d becomes very large, dot products between vectors tend to become large
- May make the softmax function to be large, making gradients to be small
- Instead of the self–attention like:

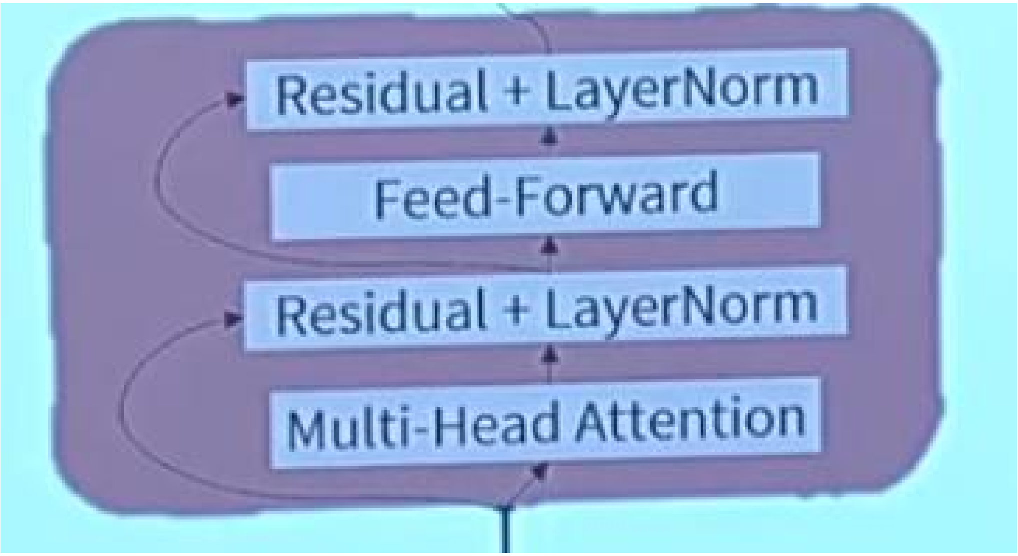
$$\text{output}_\ell = \text{softmax}(XQ_\ell K_\ell^\top X^\top) * XV_\ell$$

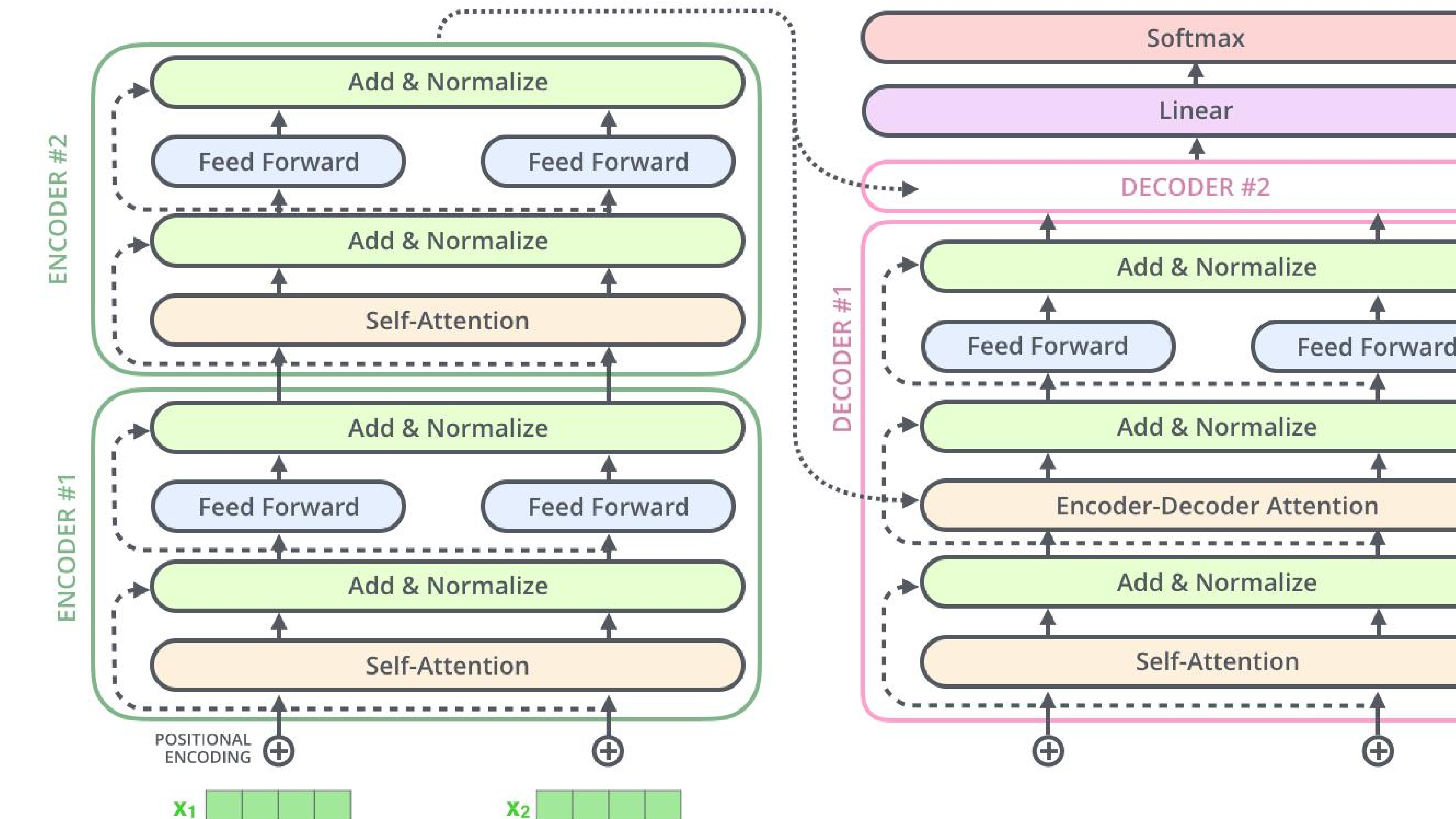
- We calculate output value vector z_l as:

$$\text{output}_\ell = \text{softmax}\left(\frac{XQ_\ell K_\ell^\top X^\top}{\sqrt{d/h}}\right) * XV_\ell$$

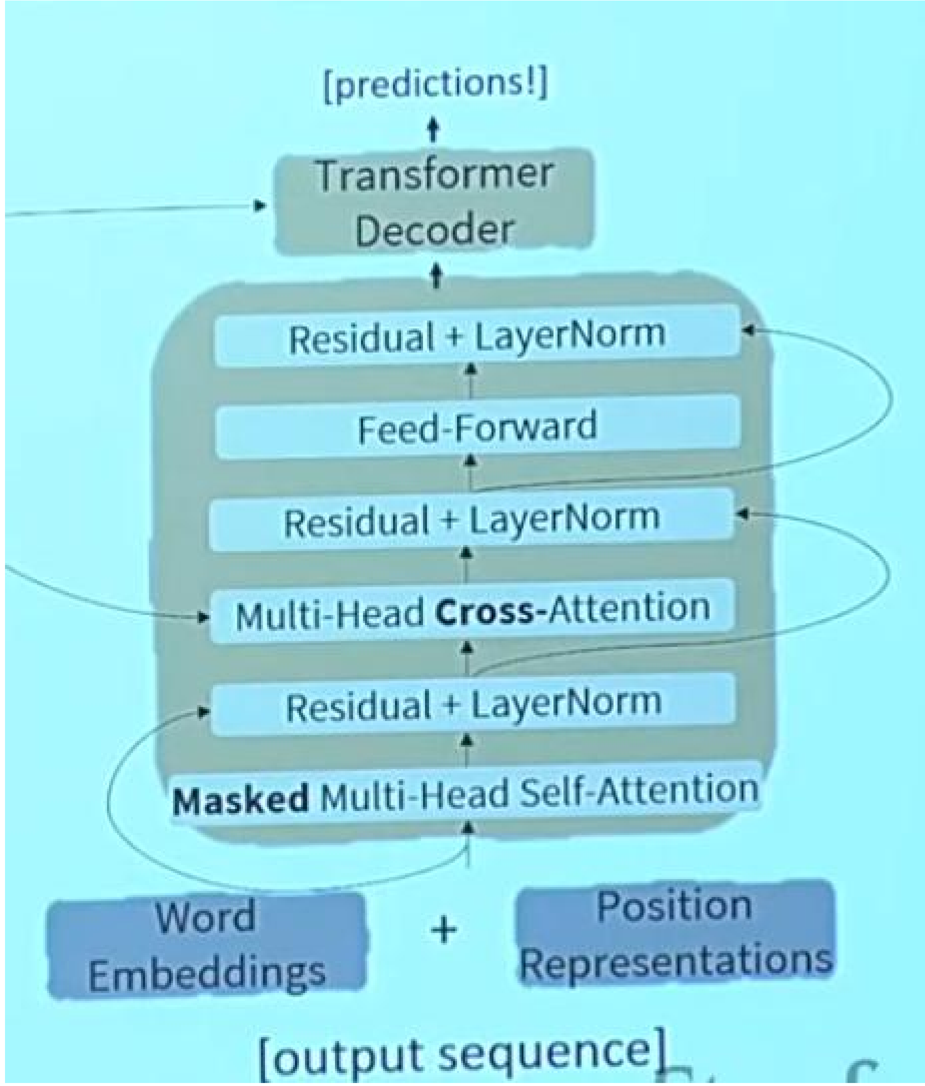
Expanded View of Each Encoder






Softmax
Linear
DECODER #2
ENCODER #2
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
DECODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Encoder-Decoder Attention
Add & Normalize
Self-Attention
ENCODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
POSITIONAL ENCODING
x_1
x_2

Decoder Block



- Output from the last encoder layer is fed to each decoder layer
- Masking is done at each decoder layer
- After each operation like multi–head attention, cross–attention and feed–forward we have got residual and layer normalization layer

Cross Attention

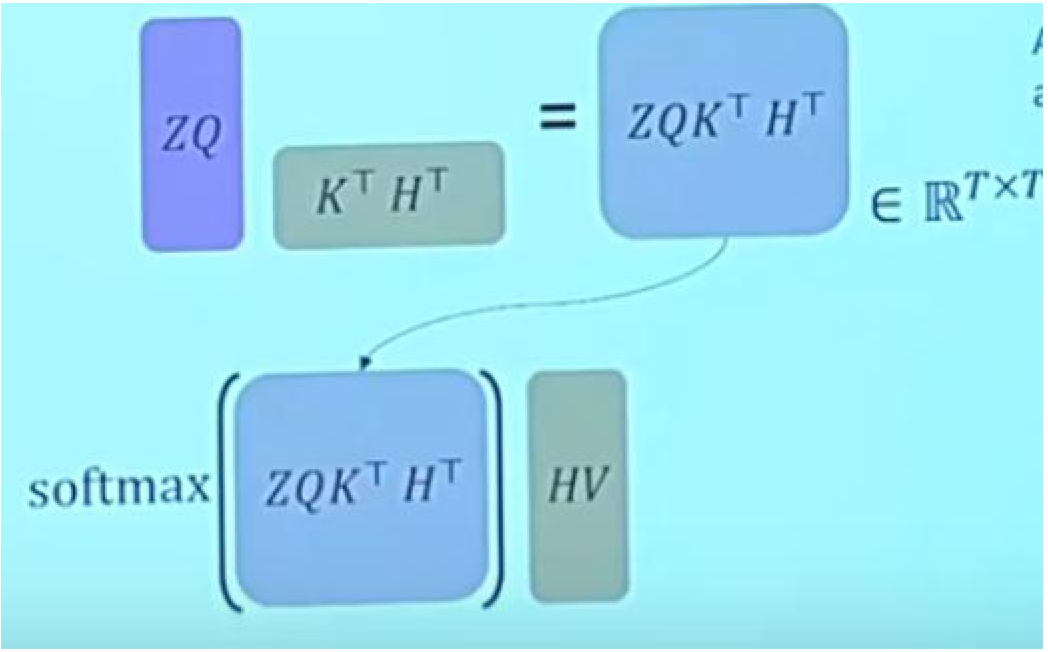
- Compuation done is similar to self–attention
- Self–attention – query, key and value come from same source
- Let h_1, h_2, ..., h_T be output vectors from the transformer encoder; xi ∈ R^d
- Let z_1, z_2, ..., z_T be vectors from transformer decoder, z_i ∈ R^d
- The keys and values are drawn from the encoder (like a memory):
- $k_i = K*h_i$, $v_i = V*h_i$
- Queries are drawn from the decoder, qi = Qzi

Cross Attention – Computation

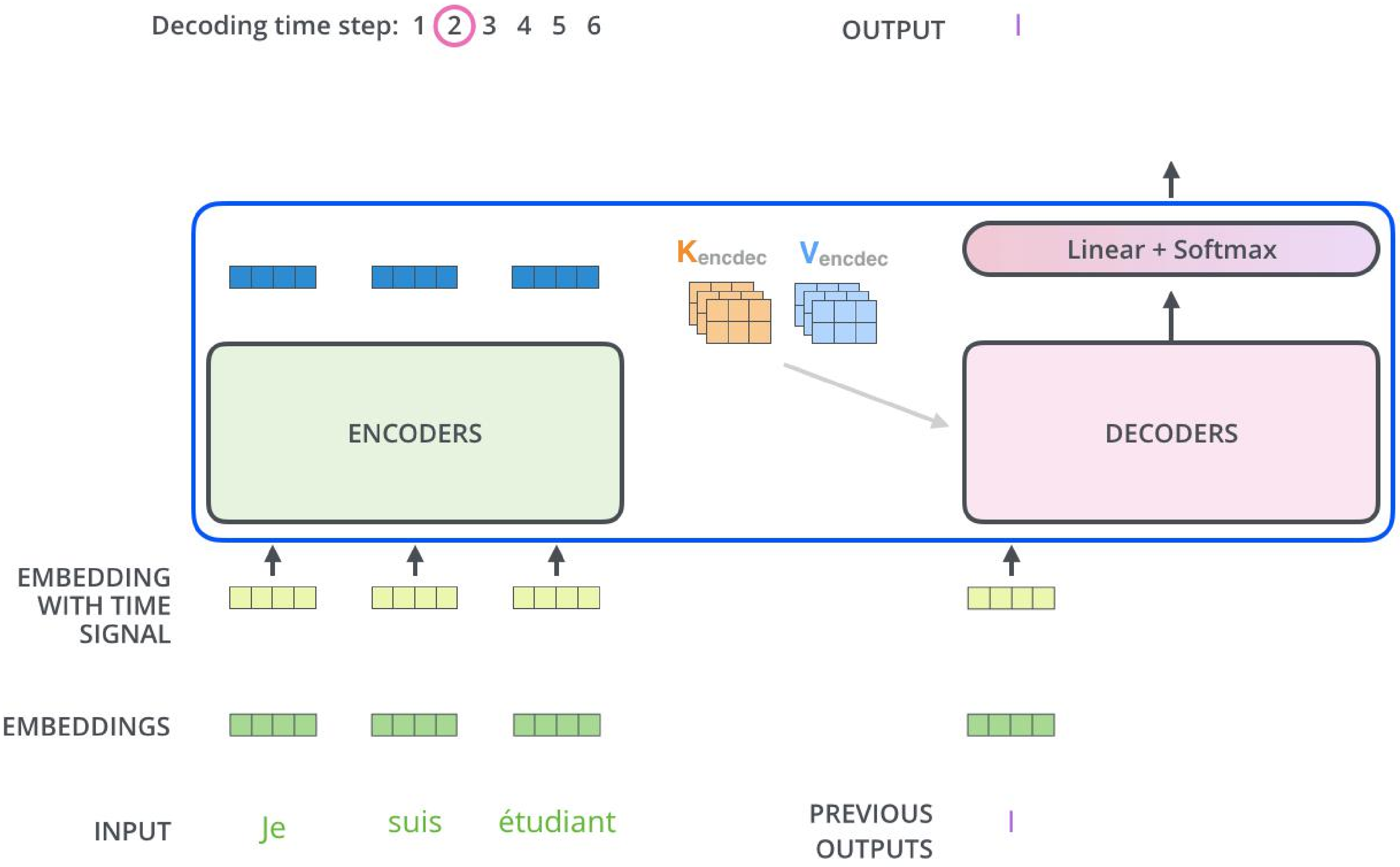
- Computed with big tensor**s** – Matrices
- $H = [h_1; h_2; \ldots h_T] \in R^{Txd}$ be the concatenation of encoder vectors
- $Z = [z_1; z_2; \ldots z_T] \in R^{Txd}$ be the concatenation of decoder vectors
- Output = $(Z*Q(H*K)^T) * (H*V)$

Cross Attention – Computation

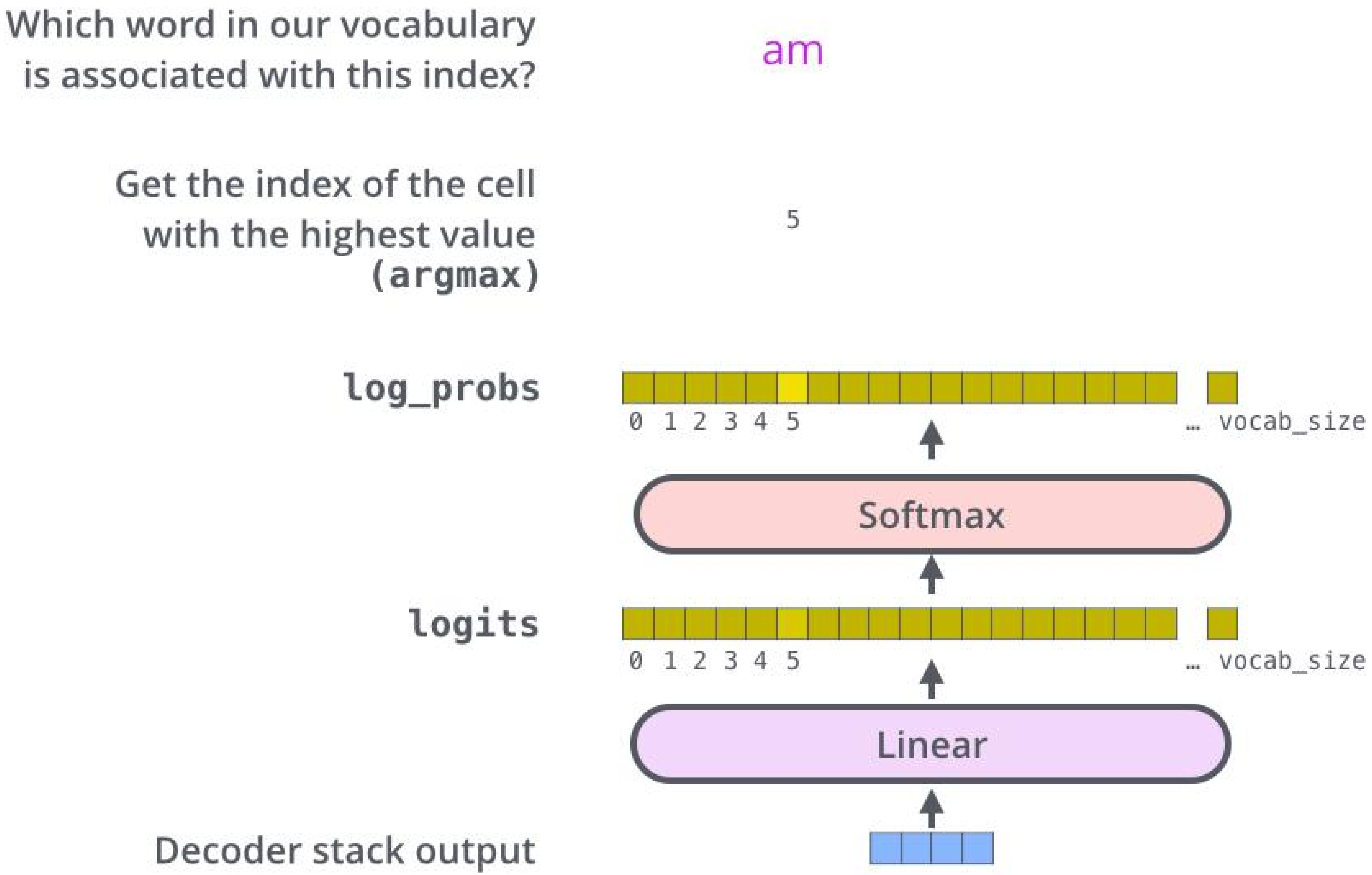
- First take query–key dot products in one matrix multiplication: $ZQ(HK)^T$
- Next softmax and then compute the weighted average with another matrix multiplication to get all pairs of attention scores



Self–Attention layer – Different



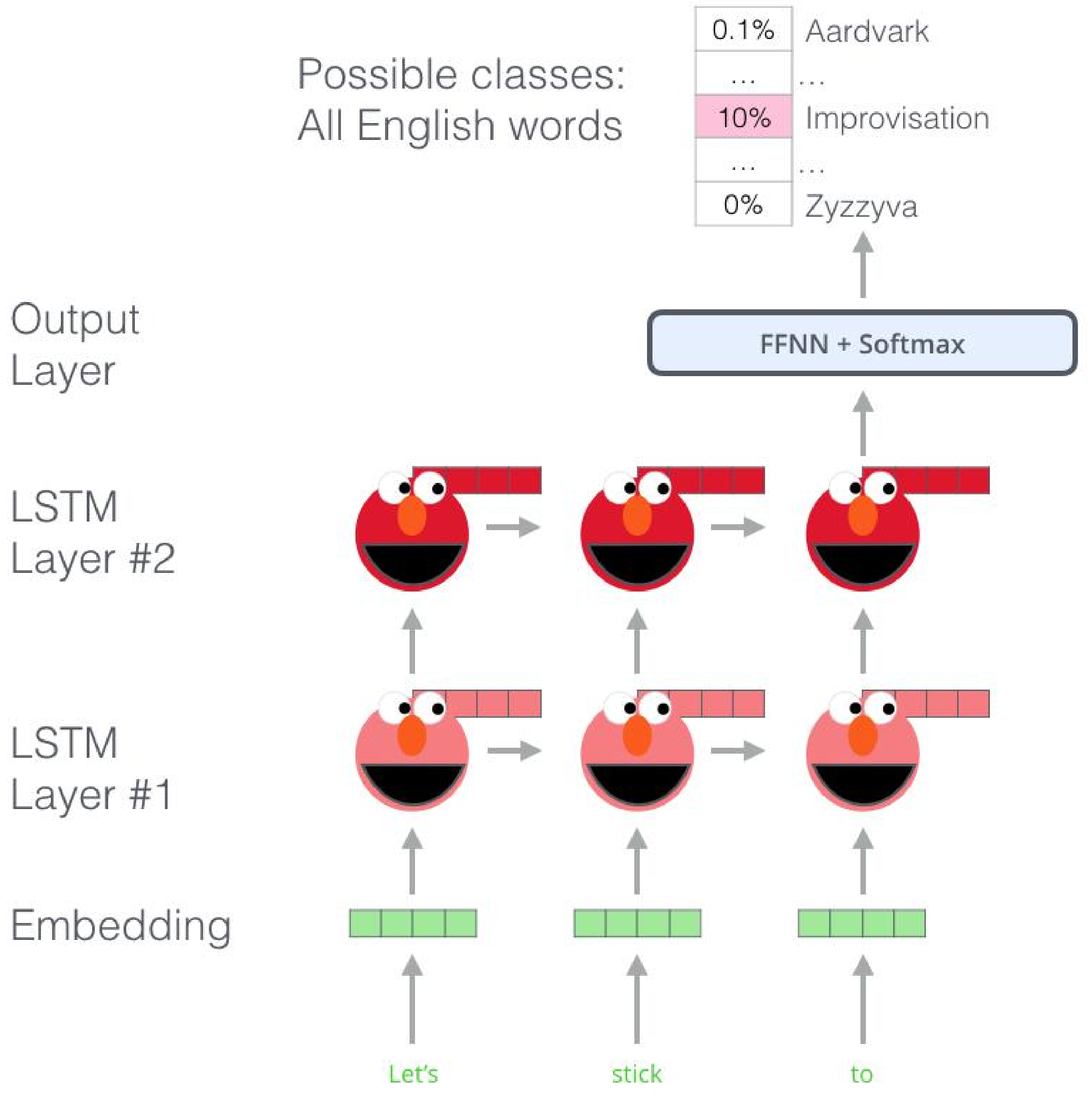
The Final Linear and Softmax Layer



- A linear layer is added to the decoder output to rescale the output of decoder to the dimension of the vocabulary

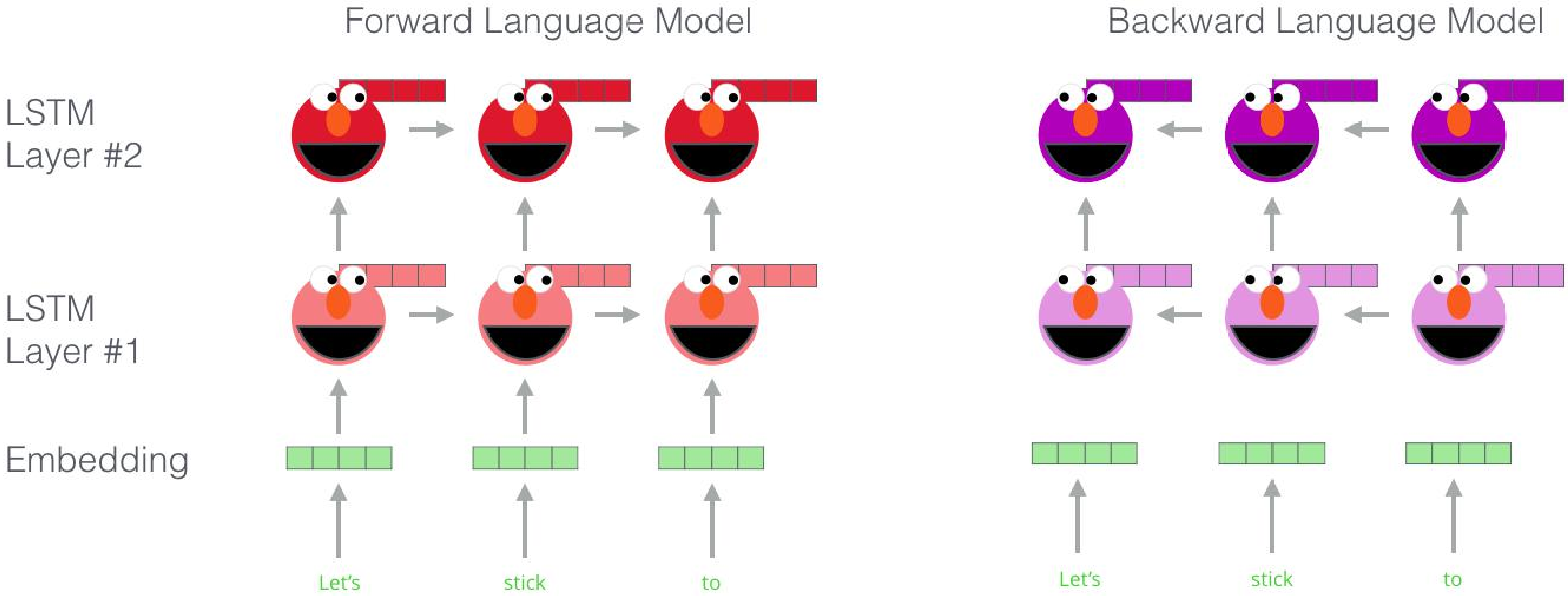
ELMo: Context Matters

ELMo: Context Matters



ELMo was Bidirectional

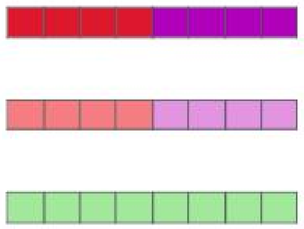
Embedding of "stick" in "Let's stick to" - Step #1



ELMo was Bidirectional

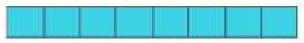
Embedding of "stick" in "Let's stick to" - Step #2

1- Concatenate hidden layers

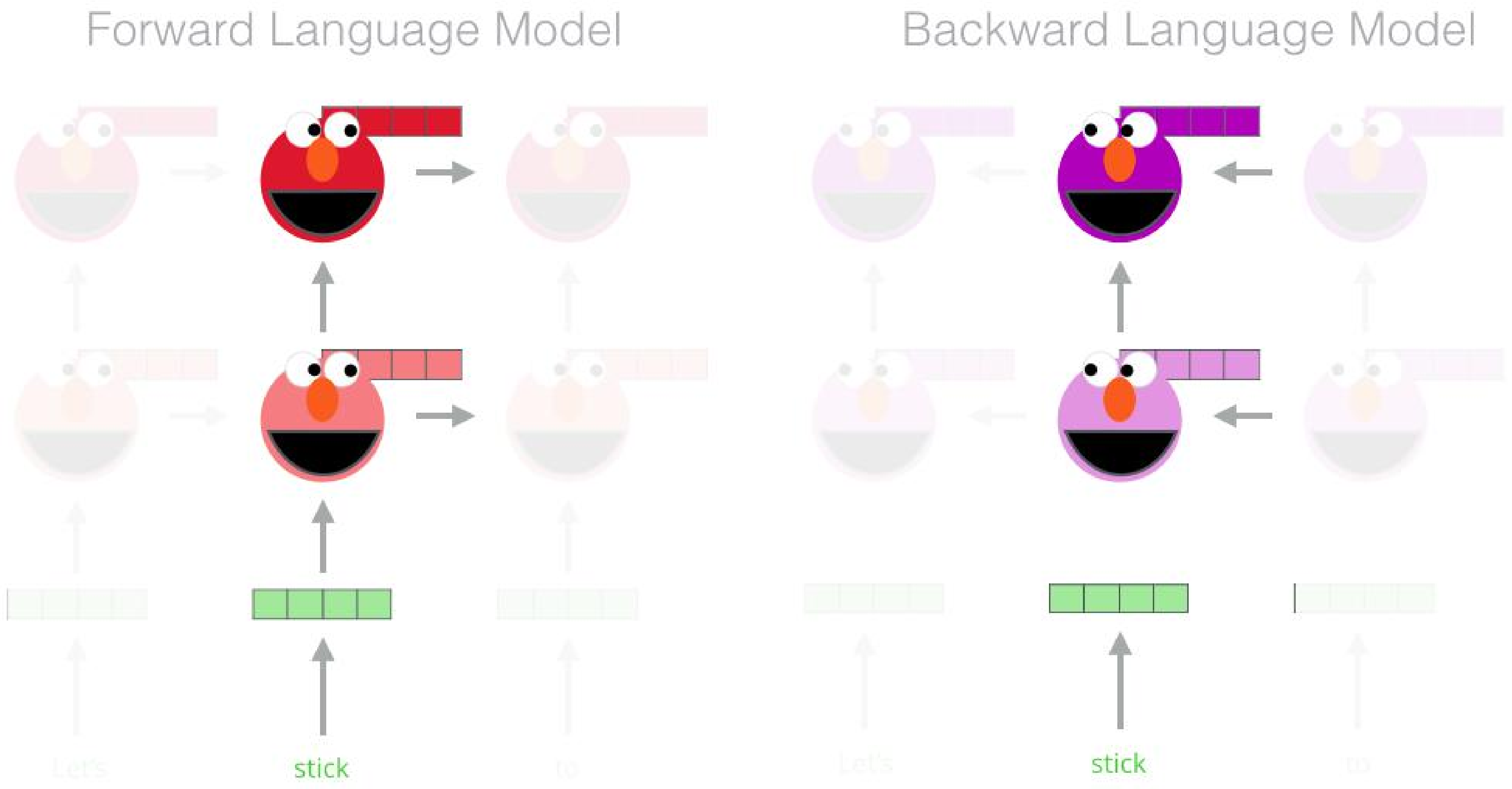


2- Multiply each vector by a weight based on the task

3- Sum the (now weighted) vectors

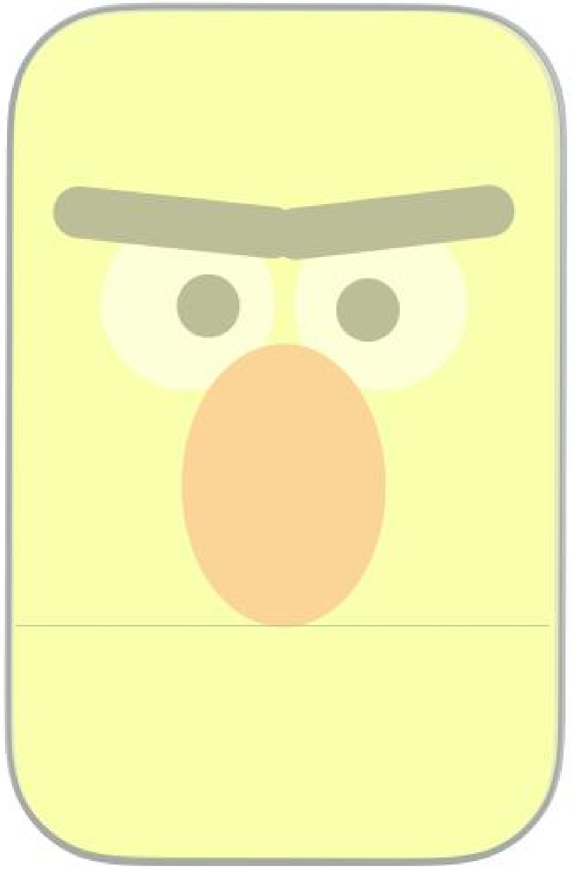


ELMo embedding of "stick" for this task in this context



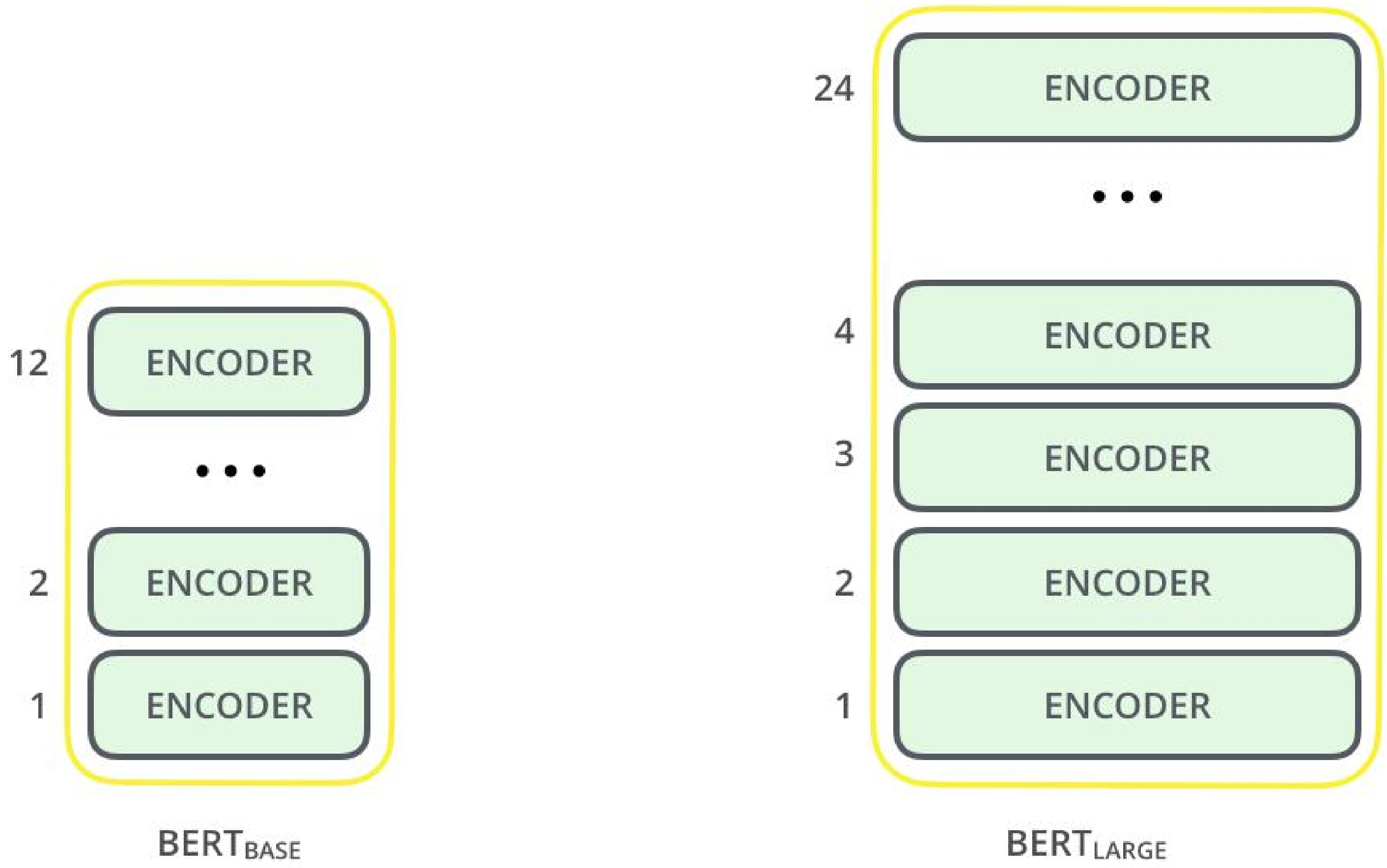
Model Architecture

$\text{BERT}_{\text{BASE}}$

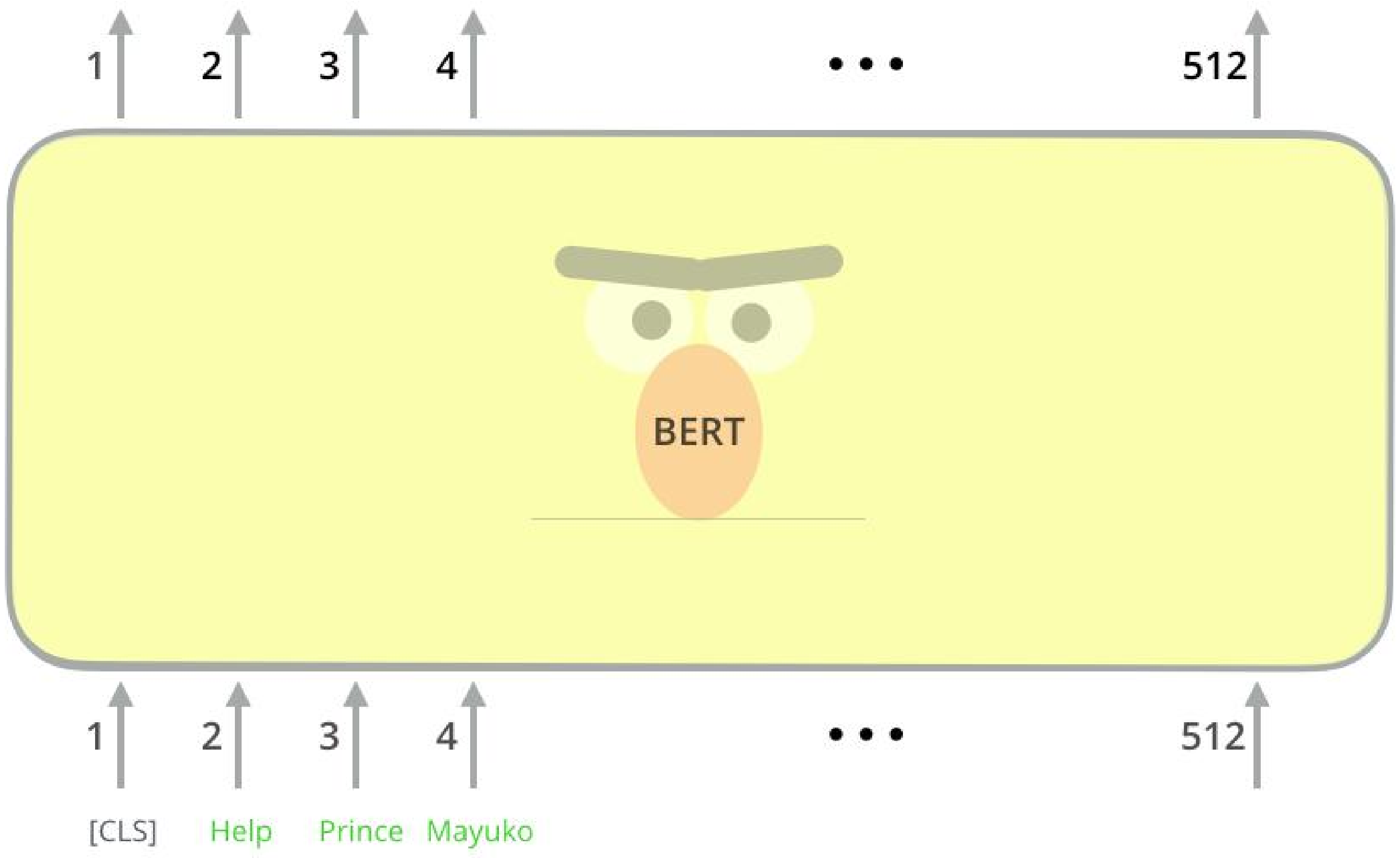


$\text{BERT}_{\text{LARGE}}$

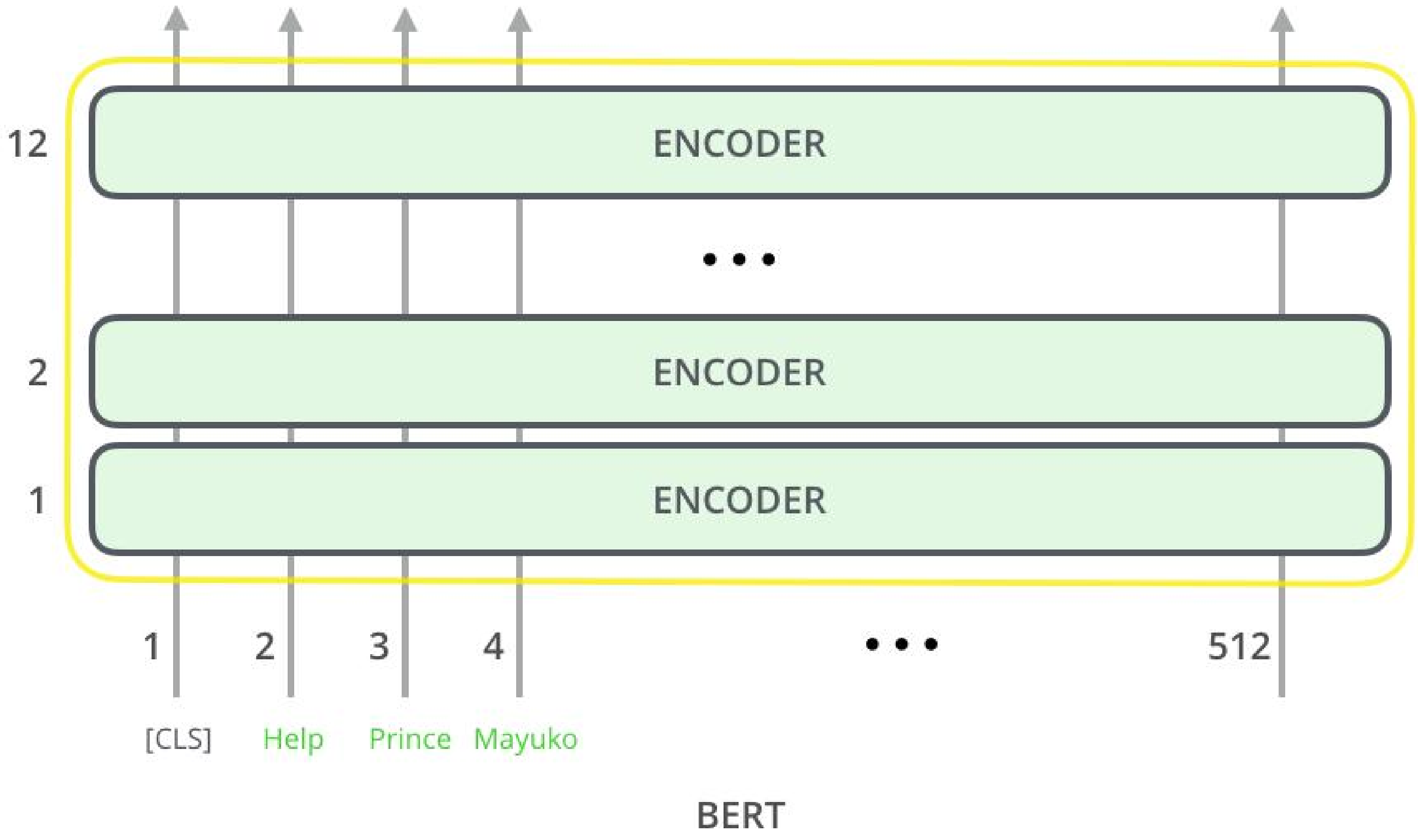
Model Architecture

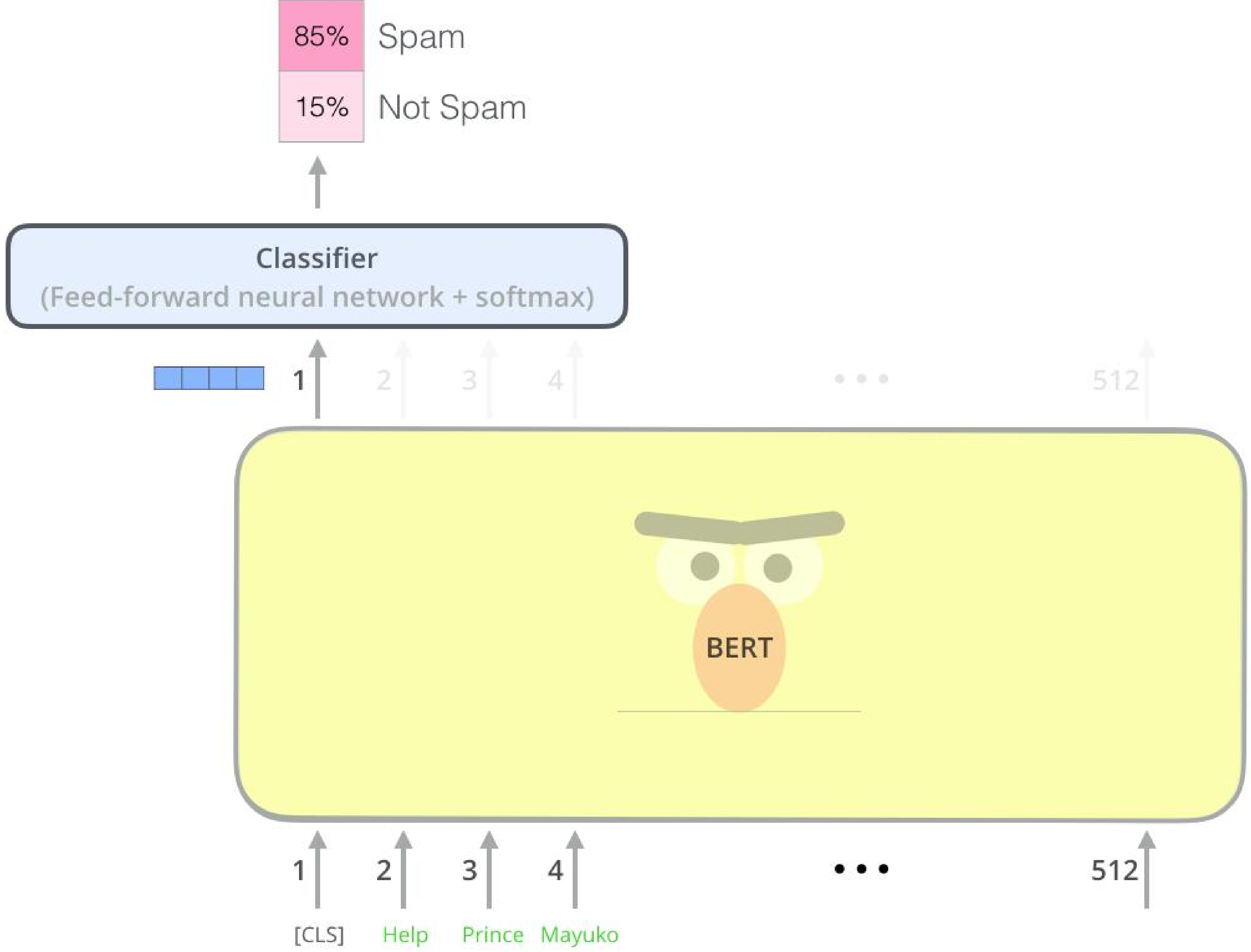


Model Architecture



Model Architecture




85%
Spam
15%
Not Spam
Classifier
(Feed-forward neural network + softmax)
1
2
3
4
• • •
512
BERT
1
2
3
4
• • •
512
[CLS]
Help
Prince
Mayuko

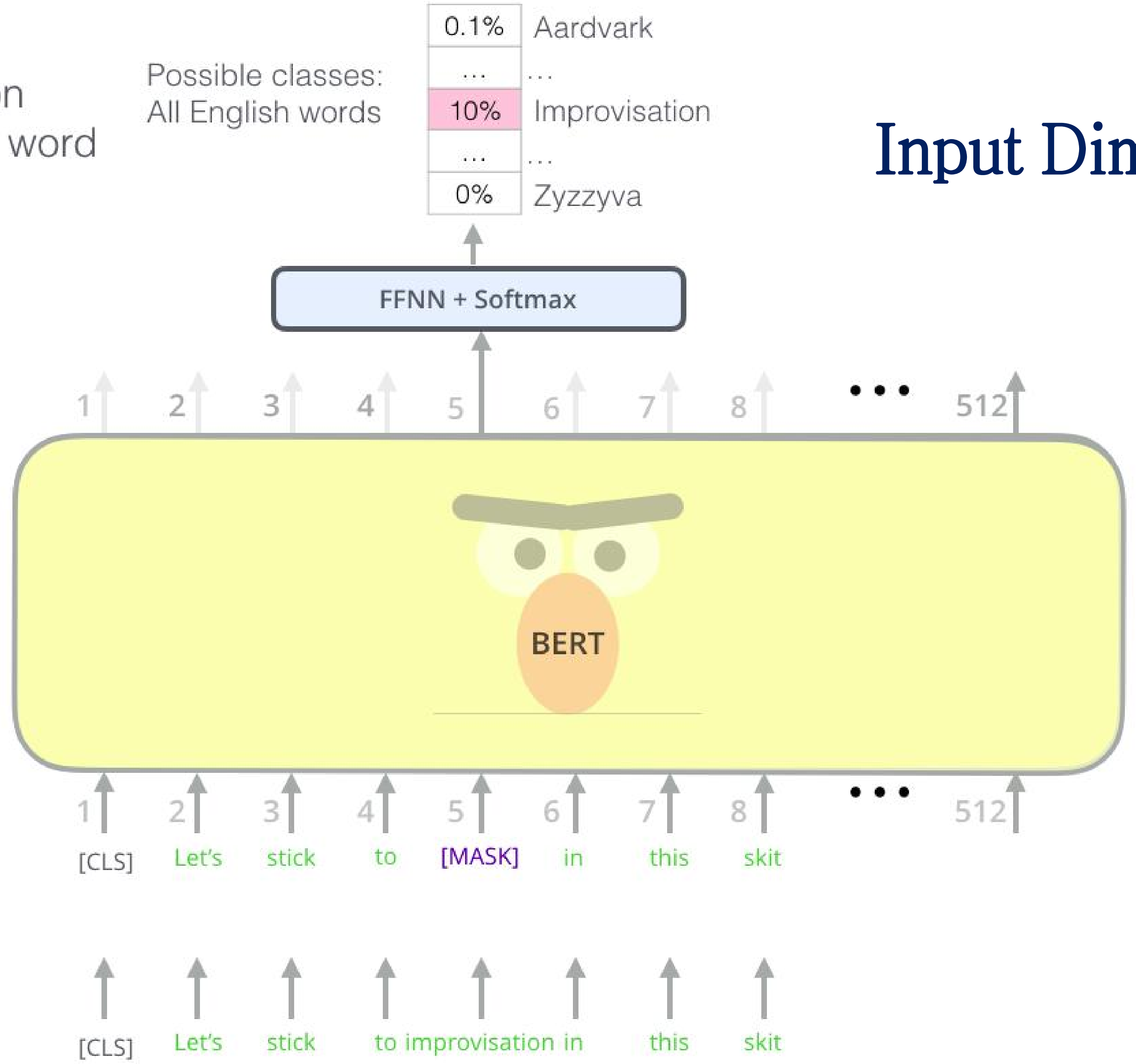
Masked Language Model

Use the output of the masked word's position to predict the masked word

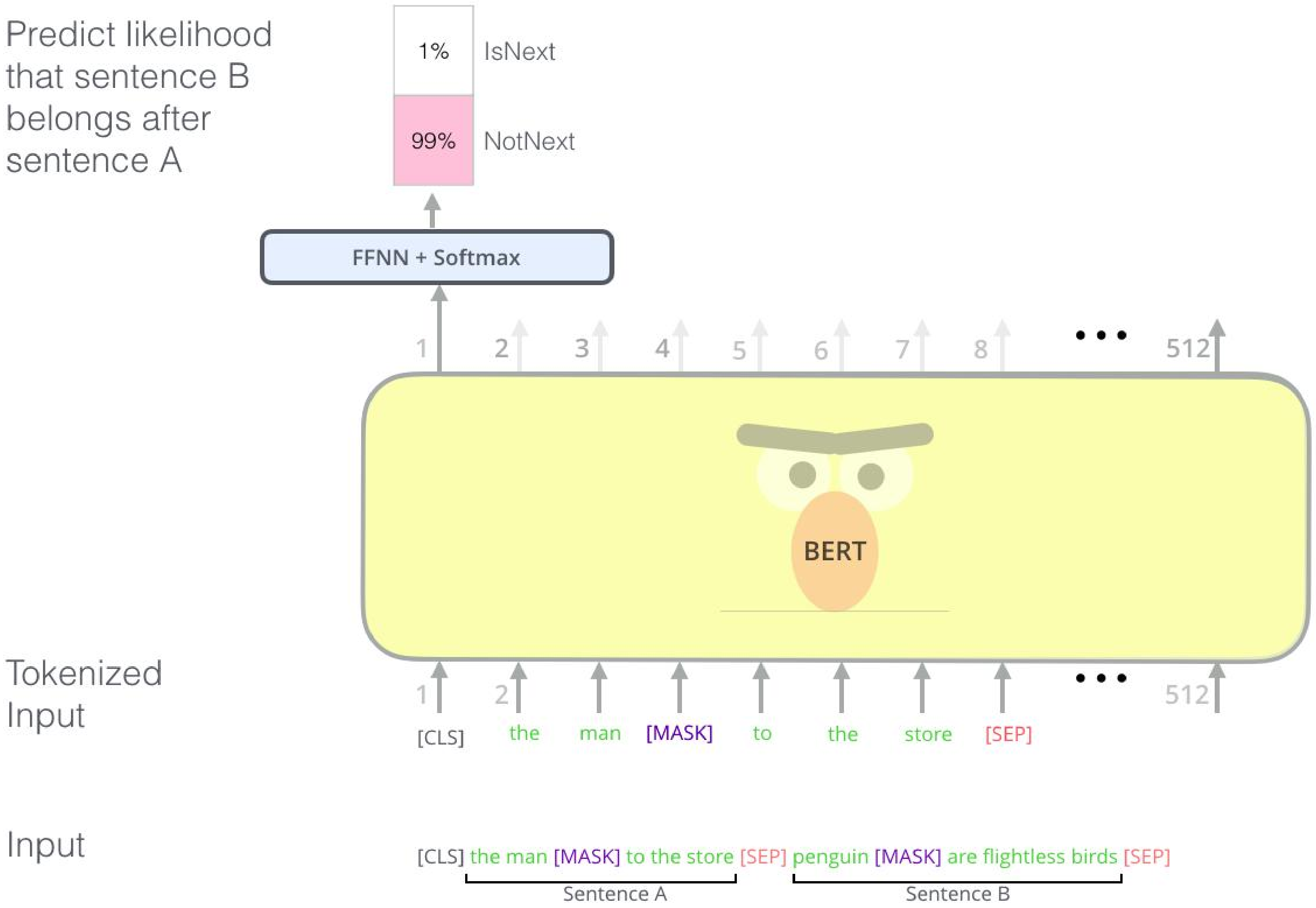
Input Dimension – 512

Randomly mask 15% of tokens

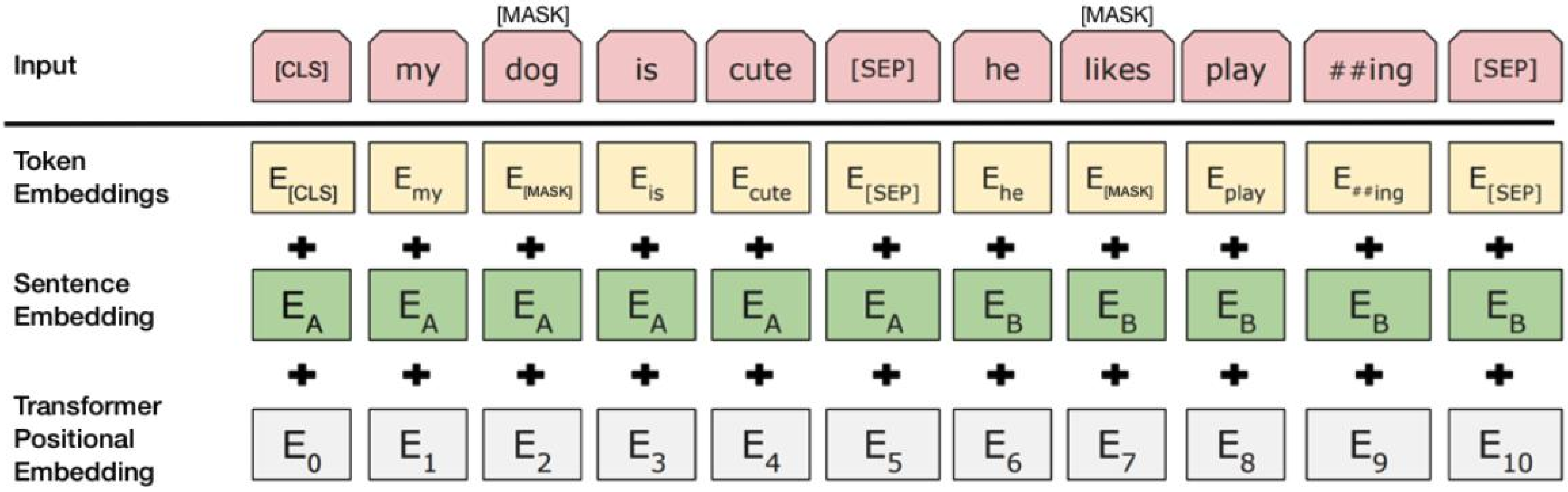
Input



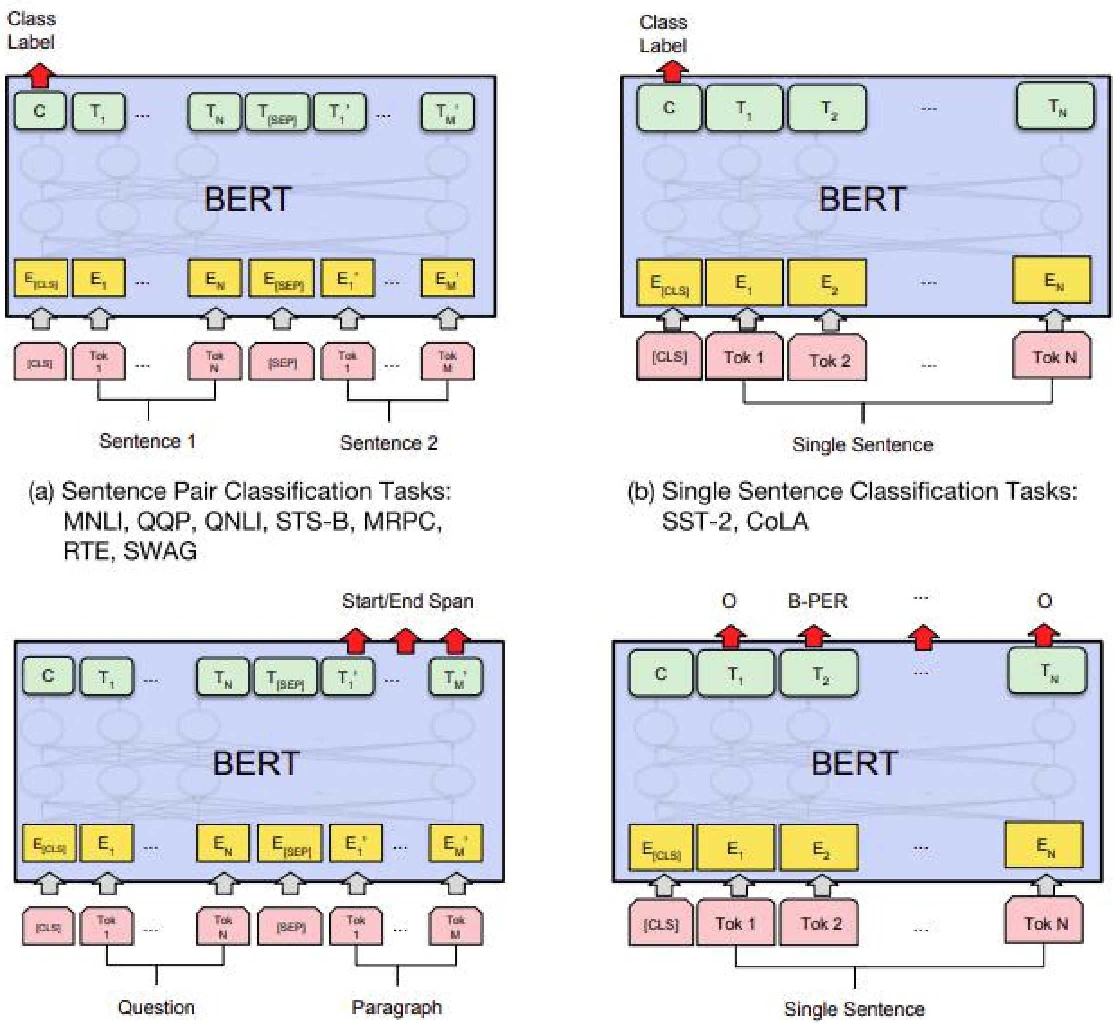
Next Sentence Prediction



Embeddings in BERT



Next Sentence Prediction



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Model Architecture

	BERT	RoBERTa	DistilBERT	XLNet
Size (millions)	**Base**: 110 **Large**: 340	**Base**: 110 **Large**: 340	**Base:** 66	**Base**: ~110 **Large**: ~340
Training Time	**Base**: 8 x V100 x 12 days* **Large:** 64 TPU Chips x 4 days (or 280 x V100 x 1 days*)	**Large:** 1024 x V100 x 1 day; 4-5 times more than BERT.	**Base:** 8 x V100 x 3.5 days; 4 times less than BERT.	**Large:** 512 TPU Chips x 2.5 days; 5 times more than BERT.
Performance	Outperforms state-of-the-art in Oct 2018	2-20% improvement over BERT	3% degradation from BERT	2-15% improvement over BERT
Data	16 GB BERT data (Books Corpus + Wikipedia). 3.3 Billion words.	160 GB (16 GB BERT data + 144 GB additional)	16 GB BERT data. 3.3 Billion words.	**Base**: 16 GB BERT data **Large**: 113 GB (16 GB BERT data + 97 GB additional). 33 Billion words.
Method	BERT (Bidirectional Transformer with MLM and NSP)	BERT without NSP**	BERT Distillation	Bidirectional Transformer with Permutation based modeling

Next Sentence Prediction (NSP)

- During training process, model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original document
- During training, 50% of the inputs are a pair in which the second sentence is the subsequent sentence in the original document
- While in the other 50% a random sentence from the corpus is chosen as the second sentence
- assumption is that the random sentence will be disconnected from the first sentence

BERT (Fine–tuning)

- Classification tasks such as sentiment analysis are done similarly to Next Sentence classification, by adding a classification layer on top of the Transformer output for the [CLS] token.
- In Question Answering tasks (e.g. SQuAD v1.1), the software receives a question regarding a text sequence and is required to mark the answer in the sequence.
- Using BERT, a Q&A model can be trained by learning two extra vectors that mark the beginning and the end of the answer.

BERT (Fine–tuning)

- In Named Entity Recognition (NER), the software receives a text sequence and is required to mark the various types of entities (Person, Organization, Date, etc) that appear in the text.
- Using BERT, a NER model can be trained by feeding the output vector of each token into a classification layer that predicts the NER label.

BERT (Fine–tuning)

- In the fine-tuning training, most hyper-parameters stay the same as in BERT training
- The BERT team has used this technique to achieve state-of-the-art results on a wide variety of challenging natural language tasks,

Model Sizes

- Model size matters, even at huge scale
- BERT_large, with 345 million parameters, is the largest model of its kind.
- It is demonstrably superior on small-scale tasks to BERT_base, which uses the same architecture with “only” 110 million parameters.

Training Steps

- With enough training data, more training steps == higher accuracy

- For instance, on the MNLI task, the BERT_base accuracy improves by 1.0% when trained on 1M steps (128,000 words batch size) compared to 500K steps with the same batch size.

Compute considerations (training and applying)

	Training Compute + Time	Usage Compute
$\text{BERT}_{\text{BASE}}$	4 Cloud TPUs, 4 days	1 GPU
$\text{BERT}_{\text{LARGE}}$	16 Cloud TPUs, 4 days	1 TPU

Knowledge Distillation

- Model compression technique in which a small model is trained to reproduce the behavior of a large pre-trained model
- Also referred to as teacher-student learning, where the large pre-trained model is the teacher and the small model is the student

Knowledge Distillation

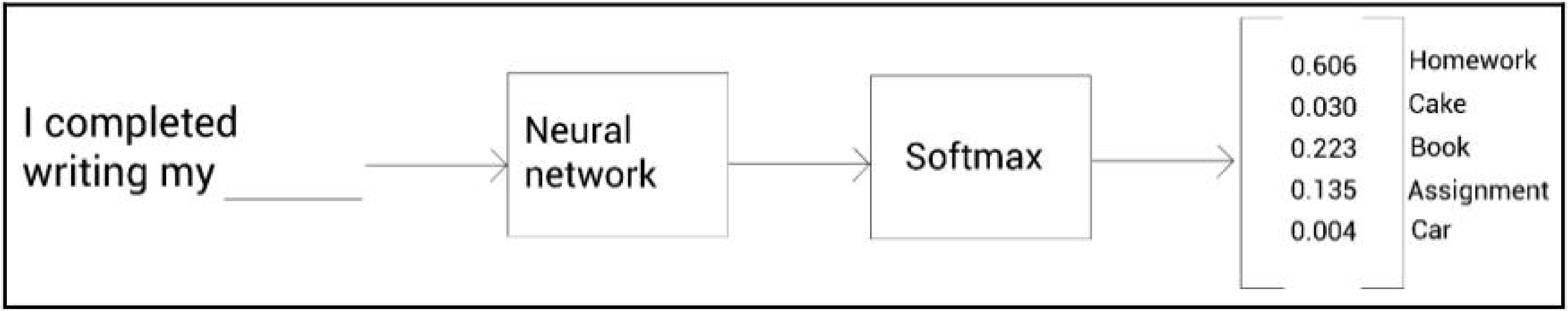


Figure 5.1 – Teacher network

Knowledge Distillation

- Apart from the word Homework, the words Book and Assignment are more relevant to the given sentence compared to words like Cake and Car
- This is known as dark knowledge
- During knowledge distillation, we want our student network to learn this dark knowledge from the teacher.

References

- https://www.youtube.com/watch?v=ptuGllU5SQQ&t=1737s
- https://jalammar.github.io/illustrated-transformer/
- https://jalammar.github.io/illustrated-bert/