Lecture 1

CME 295: Transformers & Large Language Models

Afshine Amidi & Shervine Amidi

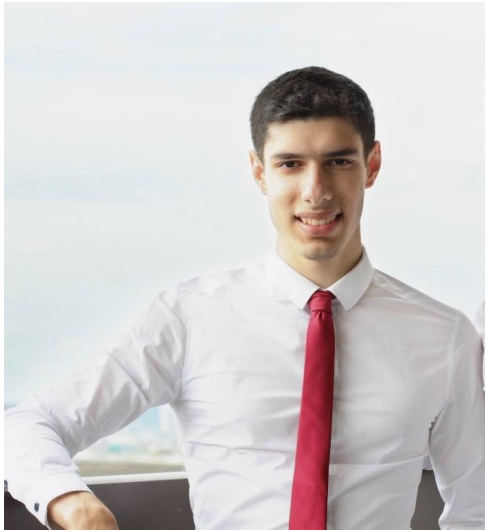
Stanford University

ICME

Teaching staff

Afshine and **Shervine**

Teaching staff



Afshine
Centrale Paris ('16), MIT ('17)
Uber, Google, Netflix

Shervine
Centrale Paris ('16), Stanford ('19)
Uber, Google, Netflix

Welcome to CME 295!

Goals.

1. Understand how **Transformers** work and how they **relate** to **LLMs**
2. Learn how **LLMs** are **trained** and used in various **applications**

Welcome to CME 295!

Goals.

1. Understand how **Transformers** work and how they **relate** to **LLMs**
2. Learn how **LLMs** are **trained** and used in various **applications**

Audience.

- Interested in LLMs
  - Career goal
  - Personal project
  - "AI literacy" or curiosity
- Prerequisite: machine learning basics, linear algebra

Logistics

Date & time.

- Fridays from 3:30pm to 5:20pm
- Thornton 110

Logistics

Date & time.

- Fridays from 3:30pm to 5:20pm
- Thornton 110

Details about the class.

- 2 units
- Letter or Credit/No credit
- Lectures are recorded
- Midterm (50% grade), scheduled on October 24th
- Final exam (50% grade), week of December 8th (exact date TBD)

Material

Class website. cme295.stanford.edu

- Contains syllabus & logistics
- Slides and recordings will be posted there

Material

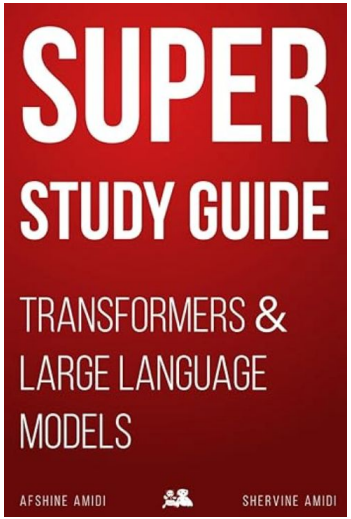
Class website. cme295.stanford.edu

- Contains syllabus & logistics
- Slides and recordings will be posted there

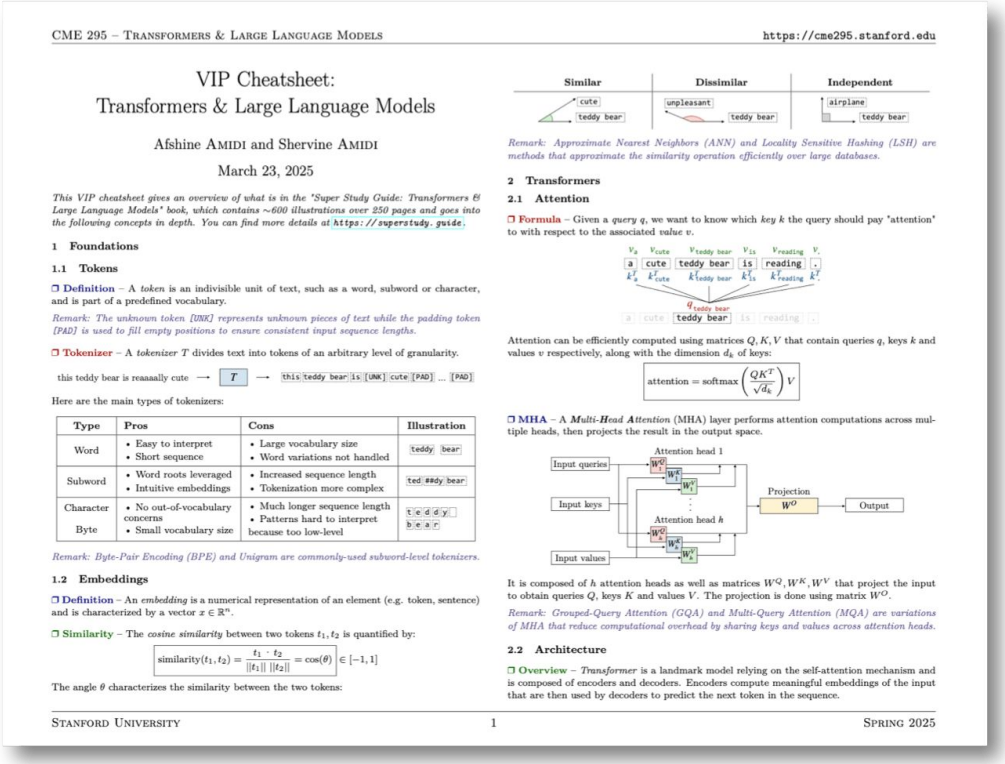
Class textbook.

Super Study Guide:
Transformers &
Large Language Models

Book: https://superstudy.guide



Material



CME 295 – Transformers & Large Language Models — https://cme295.stanford.edu

VIP Cheatsheet: Transformers & Large Language Models

Afshine Amidi and Shervine Amidi

March 23, 2025

This VIP cheatsheet gives an overview of what is in the "Super Study Guide: Transformers & Large Language Models" book, which contains ~600 illustrations over 250 pages and goes into the following concepts in depth. You can find more details at `https://superstudy.guide`.

1 Foundations

1.1 Tokens

❒ **Definition** – A *token* is an indivisible unit of text, such as a word, subword or character, and is part of a predefined vocabulary.

Remark: The unknown token [UNK] represents unknown pieces of text while the padding token [PAD] is used to fill empty positions to ensure consistent input sequence lengths.

❒ **Tokenizer** – A *tokenizer* T divides text into tokens of an arbitrary level of granularity.

this teddy bear is reaaaally cute → T → this teddy bear is [UNK] cute [PAD] ... [PAD]

Here are the main types of tokenizers:

Type	Pros	Cons	Illustration
Word	• Easy to interpret • Short sequence	• Large vocabulary size • Word variations not handled	teddy bear
Subword	• Word roots leveraged • Intuitive embeddings	• Increased sequence length • Tokenization more complex	ted ##dy bear
Character Byte	• No out-of-vocabulary concerns • Small vocabulary size	• Much longer sequence length • Patterns hard to interpret because too low-level	t e d d y b e a r

Remark: Byte-Pair Encoding (BPE) and Unigram are commonly-used subword-level tokenizers.

1.2 Embeddings

❒ **Definition** – An *embedding* is a numerical representation of an element (e.g. token, sentence) and is characterized by a vector $x \in \mathbb{R}^n$.

❒ **Similarity** – The *cosine similarity* between two tokens t_1, t_2 is quantified by:

$$\text{similarity}(t_1, t_2) = \frac{t_1 \cdot t_2}{||t_1||\,||t_2||} = \cos(\theta) \in [-1, 1]$$

The angle θ characterizes the similarity between the two tokens:

Remark: Approximate Nearest Neighbors (ANN) and Locality Sensitive Hashing (LSH) are methods that approximate the similarity operation efficiently over large databases.

2 Transformers

2.1 Attention

❒ **Formula** – Given a *query* q, we want to know which *key* k the query should pay "attention" to with respect to the associated *value* v.

Attention can be efficiently computed using matrices Q, K, V that contain queries q, keys k and values v respectively, along with the dimension d_k of keys:

$$\text{attention} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

❒ **MHA** – A *Multi-Head Attention* (MHA) layer performs attention computations across multiple heads, then projects the result in the output space.

It is composed of h attention heads as well as matrices W^Q, W^K, W^V that project the input to obtain queries Q, keys K and values V. The projection is done using matrix W^O.

Remark: Grouped-Query Attention (GQA) and Multi-Query Attention (MQA) are variations of MHA that reduce computational overhead by sharing keys and values across attention heads.

2.2 Architecture

❒ **Overview** – *Transformer* is a landmark model relying on the self-attention mechanism and is composed of encoders and decoders. Encoders compute meaningful embeddings of the input that are then used by decoders to predict the next token in the sequence.

Stanford University — 1 — Spring 2025

VIP Cheatsheet

(also translated in 11 languages so far)

Link to PDF: https://github.com/afshinea/stanford-cme-295-transformers-large-language-models

Class communications

Canvas.

- Announcements
- Class discussions via Ed

Class communications

Canvas.

- Announcements
- Class discussions via Ed

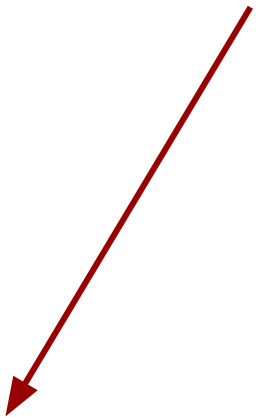
Any other general **inquiries** / **questions**, email us:

- cme295-aut2526-staff@lists.stanford.edu
- afshine@stanford.edu, shervine@stanford.edu

Example of a slide in this class

Explanation of a CME 295 concept

Source & suggested reading, if interested



Some paper

Disclaimer before starting: many abbreviations....

BERT SFT BLEU FLAN BPE WNLI MLM

MRPC

GPT LaaJ LSTM PoS T5

QA

ROUGE GLUE PEFT PPO NER

F1 RAG PPL LLaMA C4 MT RLHF

GRU SQuAD WP DPO SP NLG METEOR

...but don't worry!

BERT, T5, GPT, LLaMA

Transformer-based models

LSTM, GRU, GloVe, BPE, CoT, ToT, SC, RAG

Misc architectures & techniques

SFT, PEFT, FLAN, RL, RM, RLHF, PPO, DPO

Training strategies

NER, PoS, MLM, NSP, MT, QA, NLG

Tasks

MNLI, WNLI, C4, SQuAD, GLUE, MRPC

Datasets

F1, PPL, ROUGE, BLEU, METEOR, LaaJ, WER

Metrics

CME 295 Transformers & Large Language Models

NLP overview

Tokenization

Word representation

RNNs

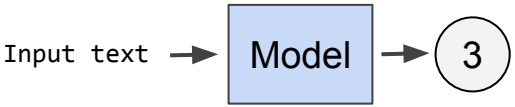
Self-attention mechanism

Transformer architecture

End-to-end example

NLP tasks overview

Classification



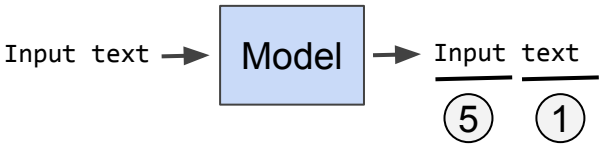
- Sentiment extraction
- Intent detection
- Language detection
- Topic modeling

NLP tasks overview

Classification

- Sentiment extraction
- Intent detection
- Language detection
- Topic modeling

“Multi”-classification



- Part of speech tagging → understanding noun, verb etc..
- Named entity recognition → labelling some specific word (Intent)
- Dependency parsing
- Constituency parsing

NLP tasks overview

Classification

- Sentiment extraction
- Intent detection
- Language detection
- Topic modeling

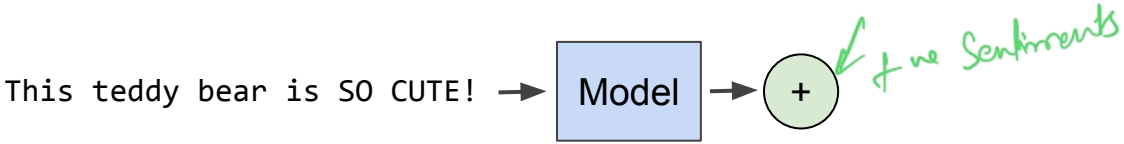
“Multi”-classification

- Part of speech tagging
- Named entity recognition
- Dependency parsing
- Constituency parsing

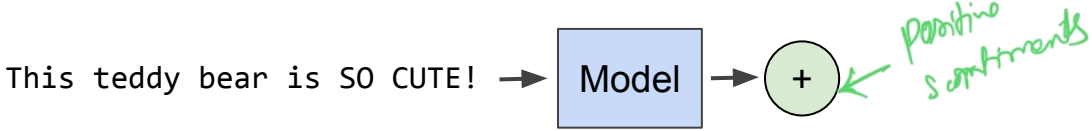
Generation

- Machine translation
- Question answering
- Summarization
- Text generation

NLP task: Sentiment Extraction



NLP task: Sentiment Extraction



Datasets

Amazon reviews

IMDB critiques

Twitter

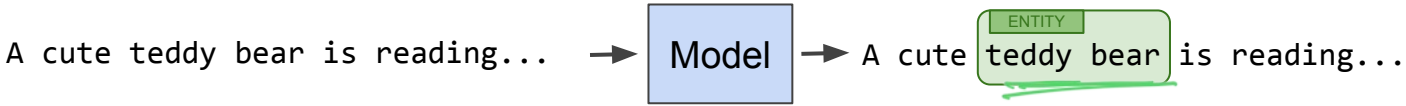
Evaluation metrics

- Accuracy → % of observations that were correctly predicted?
- Precision → % of predicted positive that were correct?
- Recall → % of actually positive that were correct?
- F1 score → score that is a function of precision and recall [used in case of imbalanced dataset]

NLP task: Named Entity Recognition

multi classification form

NLP task: Named Entity Recognition



Datasets

- Annotated Reuters newspaper (CoNLL-2003, CoNLL++)

Evaluation metrics

- Accuracy
- Precision
- Recall
- F1 score

at a token level, per entity type

NLP task: Machine Translation

NLP task: Machine Translation

Datasets

WMT'14 English-French

WMT'14 English-German

Evaluation metrics

- BLEU → quality of text translated, similar to "precision"
- ROUGE → quality of text generated, similar to "recall"
- Perplexity → quantifies how 'surprised' the model is to see some words together

Formulas can be found on this website: http://www.statmt.org/book/slides/08-evaluation.pdf

High-level timeline

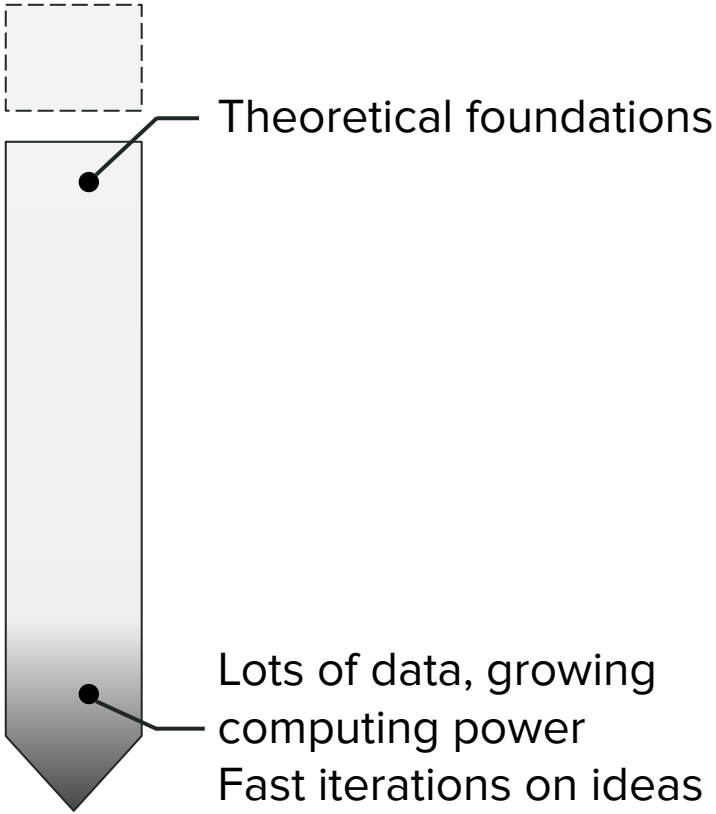
1980s Recurrent neural networks (RNNs)

1997 Long short-term memory (LSTM)

2013 Word2vec

2017 Transformers

2020s Large Language Models



CME 295 Transformers & Large Language Models

NLP overview

Tokenization

Word representation

RNNs

Self-attention mechanism

Transformer architecture

End-to-end example

Tokenization

Cutting the sentence in the form of model can understand.

i.e. pass the sentence in such a way a model can understand the text being provided.

A cute teddy bear is reading.

Tokenization

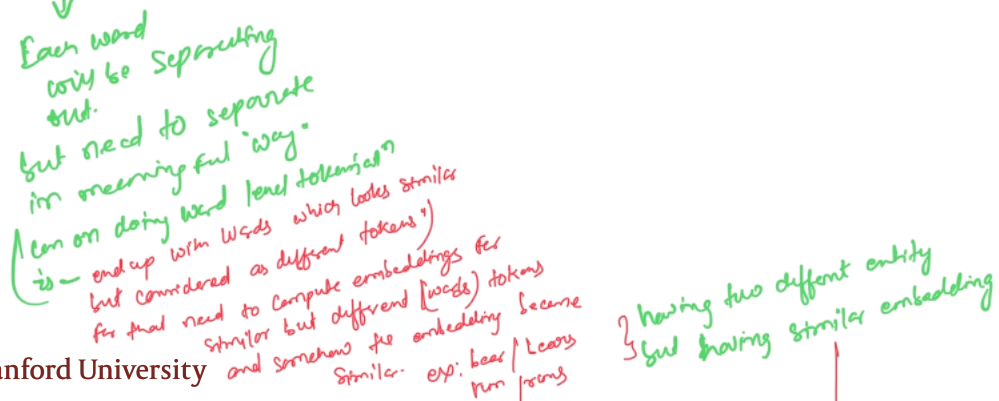
A cute teddy bear is reading.

Tokenization

A cute teddy bear is reading.

A	cute	teddy bear	is	reading	.

word	A	cute	teddy	bear	is	reading	.



Tokenization

to get rid of the limitation on word level

tokenization is ee sub-word

leveraging roots of words

A cute teddy bear is reading.

bear / bears

play / playing

read / reading

A	cute	teddy bear	is	reading

pros: leveraging the root of words
cons: longer sequence length

A	cute	teddy	bear	is	reading

but cons of doing sub word is sequence will be longer

sub-word	A	cute	ted	##dy	bear	is	read	##ing	.

* Complexity of large language model is — sequence length - the more token will be passed to model more time it will take to run/process.

Tokenization

A cute teddy bear is reading.

Tokenization summary

Method	Pros	Cons
Word-level	• Simple • Interpretable	• Risk of OOV • Does not leverage knowledge of root
Subword-level e.g. WordPiece, BPE	• Leverages common prefixes and suffixes • Learned from the data	• Risk of OOV, though less than word-level
Character-level	• Small chance of OOV • RoBUsT tO CASinG anD MIspeliNGs	• Makes computations slower • Embeddings not interpretable

CME 295 Transformers & Large Language Models

NLP overview

Tokenization

Word representation

RNNs

Self-attention mechanism

Transformer architecture

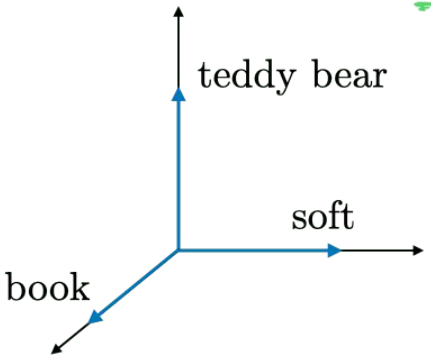
End-to-end example

Token representations

Motivation

in the form of model to understand the sentences, we need to have representations of token — called as "Token representations"

Naive (one-hot) encoding



vocabulary of 3 tokens

soft : $\begin{pmatrix}1\\0\\0\end{pmatrix}$ // Vector representation of soft in vocabulary

teddy bear : $\begin{pmatrix}0\\1\\0\end{pmatrix}$ // Vector representation of teddy bear

book : $\begin{pmatrix}0\\0\\1\end{pmatrix}$ // Vector representation of book

way of representing tokens.

ie basically we compare the token i.e which are similar to other ones and similarity measurement is being used ie Cosine Similarity"

$$\text{soft} = \begin{pmatrix}1\\0\\0\end{pmatrix}$$

$$\langle \text{teddy bear}, \text{book} \rangle = 0$$

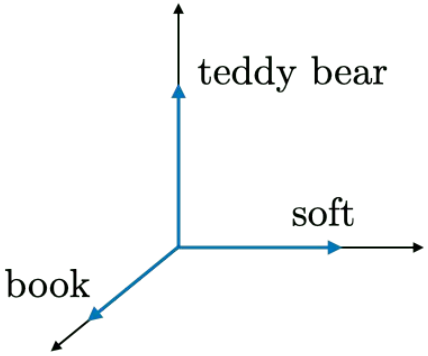
$$\langle \text{teddy bear}, \text{soft} \rangle = 0$$

Figure adapted from "VIP cheatsheets for Stanford's CS 230", Amidi. stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Token representations

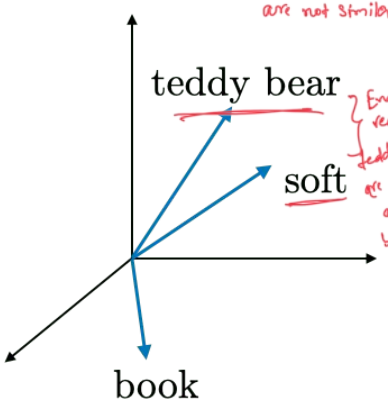
Motivation

Naive (one-hot) encoding



$$\text{soft} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad \begin{aligned} \langle \text{teddy bear}, \text{ book} \rangle &= 0 \\ \langle \text{teddy bear}, \text{ soft} \rangle &= 0 \end{aligned}$$

Learned embedding



$$\text{soft} = \begin{pmatrix} 0.95 \\ 0.32 \\ 0.01 \end{pmatrix} \quad \begin{aligned} \langle \text{teddy bear}, \text{ book} \rangle &\sim 0 \\ \langle \text{teddy bear}, \text{ soft} \rangle &\sim 1 \end{aligned}$$

Figure adapted from "VIP cheatsheets for Stanford's CS 230", Amidi. stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Word2vec

till here, we get to know that one-hot encoding is not great bcz words can be orthogonal in n-dimension using one-hot encoding so,
so, basically sub-word tokenization learnt from data / embeddings so,
Word2vec being used bcz it make sense in term of embedding.

Overview

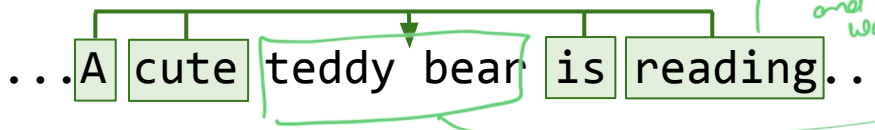
- Neural network with a **proxy task** over billions of words worth of text
- Learns an embedding layer

because it already haven kept in such a way → what words will be predicted after
what word is already being kept in vector space which are skip-gram, and continuous word of bags
i.e. taking the leverage of texts we have and try to predict something part of text based on "context"

Proxy tasks

- CBOW (continuous bag of words)

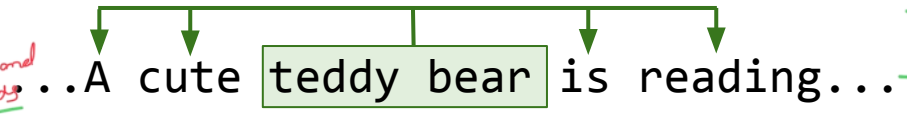
as "takes into consideration" the words which are in around given a target words, and goal is to predict target word



given the around words, and predict the target word. like here - teddy bear has to be predicted based on input given as "A cute ____ is reading"

- Skip-gram

go to the target words and try to predict the words around it.



opposite of CBOW
target to predicted around word given "teddy is"

Word2vec paper "Efficient Estimation of Word Representations in Vector Space", Mikolov et al., 2013.

Word2vec

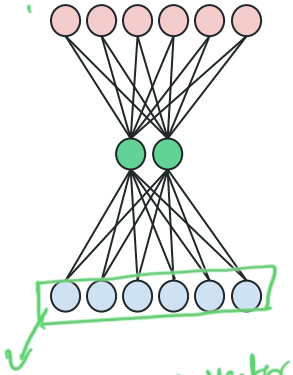
So, it would be better to have a model which can understand the content being input to predict target words/around so, word2vec has introduced.

Architecture → Predicting the next word

output — size V

hidden — size d

input — size V

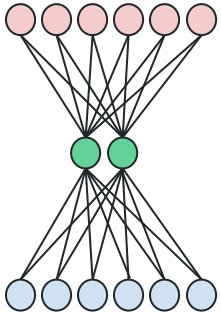


input is given as vector of word and with the help of hidden state; output will be predicted. (vocabulary is being given) → basically learning the "word representation" through this task — try to consider as input and predict the next word.

Word2vec

Example with predicting next word

A cute teddy bear is reading



this is text → it has go through some processing for model to understand

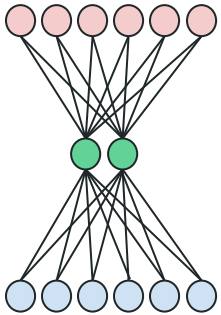
A cute teddy bear is reading

i.e. Embedding like one-hot encoding to predict the relationship b/w words for prediction with cosine similarity in m dimension of vector space

Word2vec

Example with predicting next word

A [cute] teddy bear is reading



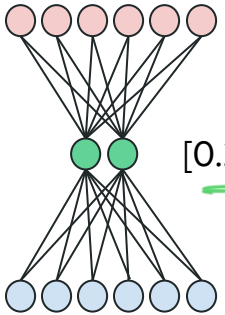
[1,0,0,0,0,0] ← Embedding using one-hot encoding for vocabulary.

[A] cute teddy bear is reading

Word2vec

Example with predicting next word

A [cute] teddy bear is reading



[0.2, 0.9]

[1,0,0,0,0,0]

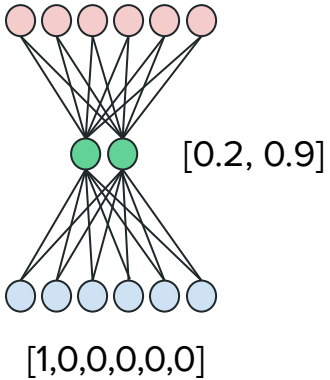
One-hot encoding is passed through network; and weights are stored (after so much of mathematical calculation known as training). this is the optimal weight stored

[A] cute teddy bear is reading

Word2vec

Example with predicting next word

A [cute] teddy bear is reading



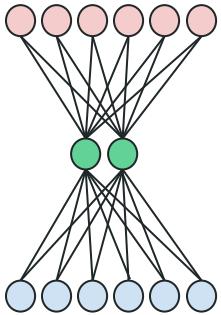
finding the probability of words which will be highest probability will be optimal word for word "A" which is [1,0,0...] had the embedding but cute might have [0,1,0,0] so just in n-dimensions of vector space, the cosine similarity has given the highest similarity value for prediction of next word.

[A] cute teddy bear is reading

Word2vec

Example with predicting next word

A cute **teddy bear** is reading



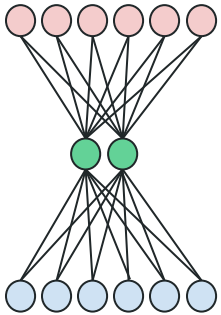
A **cute** teddy bear is reading

Some repeats with next word and it continously goes on - remember it so much of computation

Word2vec

Example with predicting next word

A cute **teddy bear** is reading



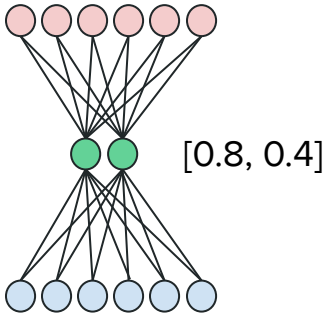
[0,1,0,0,0,0]

A **cute** teddy bear is reading

Word2vec

Example with predicting next word

A cute **teddy bear** is reading

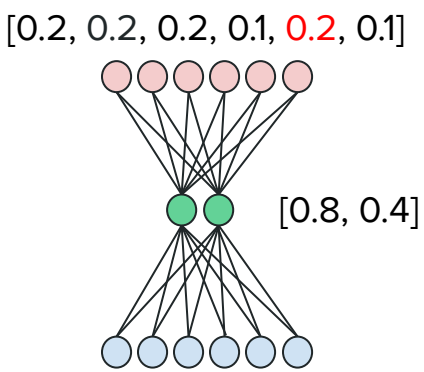


A **cute** teddy bear is reading

Word2vec

Example with predicting next word

A cute **teddy bear** is reading

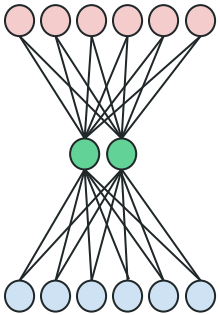


A **cute** teddy bear is reading

Word2vec

Example with predicting next word

A cute teddy bear **is** reading

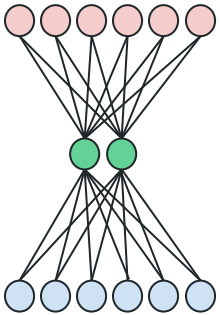


A cute **teddy bear** is reading

Word2vec

Example with predicting next word

A cute teddy bear is **reading**

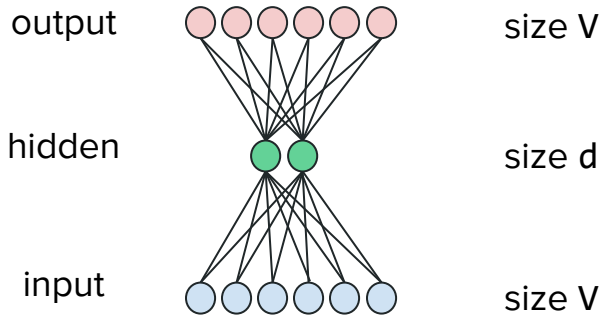


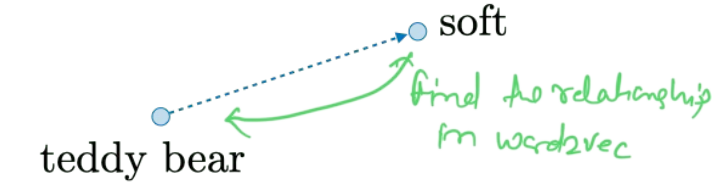
A cute teddy bear **is** reading

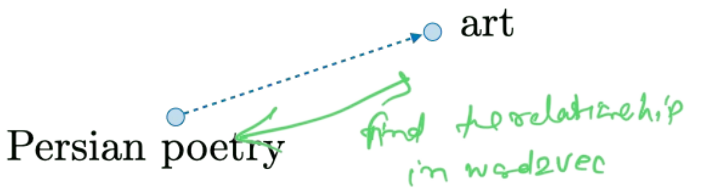
Word2vec

after a lot of computation for predicting next word is bit hectic but this is how a naive/state of art architecture is being prepared for complex task.

A **cute** teddy bear is reading







A **cute** teddy bear is reading

Figure adapted from "VIP cheatsheets for Stanford's CS 230", Amidi. stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

CME 295 Transformers & Large Language Models

NLP overview

Tokenization

Word representation

RNNs

Self-attention mechanism

Transformer architecture

End-to-end example

Recurrent Neural Networks (RNNs)

So, we have a naive and state of art method to predict the next words/arounds but we wanted to make it complicated by not only predicts the words but sentences/pieces of text/absolute relation which is given in puts i.e. we need to find the best relation word inputs → which does by Sequential model known as "RNN"

it basically captures the sequential nature of how text appears.

Instead of processing words at a time, it also keeps hidden representation of the sentence and consider tokens one at a time

Overview

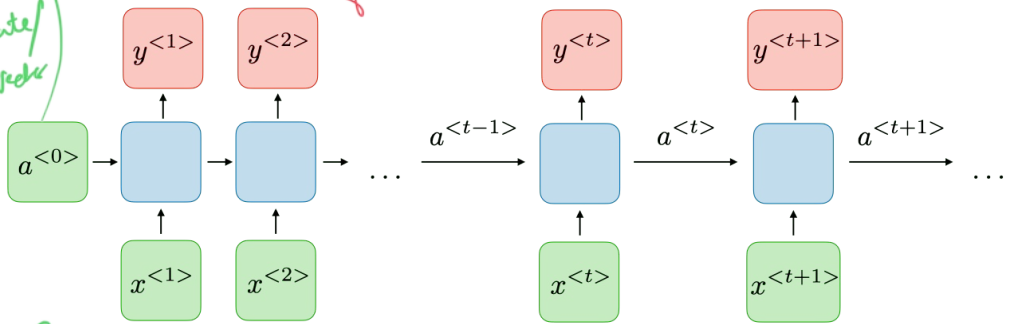
- First introduced in the 80s
- Class of neural networks where connections form a temporal sequence

General form →

So basically based on input $x^{<1>}$ and hidden state $a^{<0>}$ $y^{<1>}$ is being output after learning many iterations.

bcz order of words matter

known as hidden state/ activation/ context vector

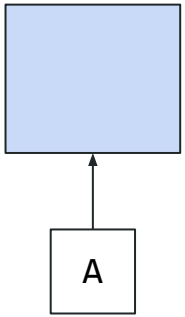


take consideration as order of word/token appears.

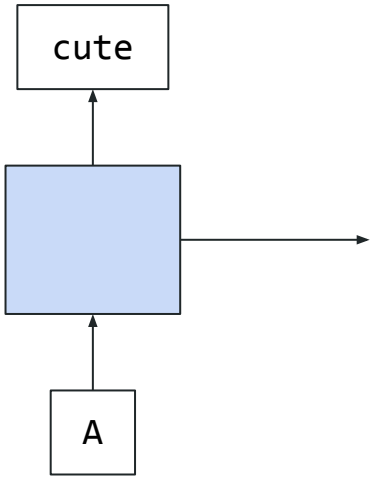
Figure adapted from "VIP cheatsheets for Stanford's CS 230", Amidi. stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Recurrent Neural Networks (RNNs)

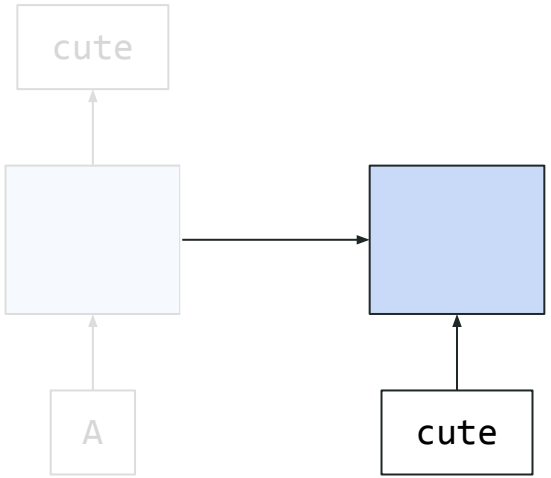
Recurrent Neural Networks (RNNs)



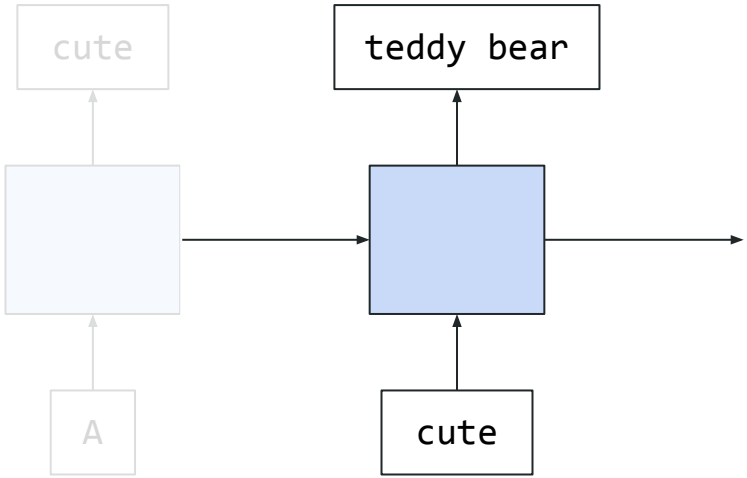
Recurrent Neural Networks (RNNs)



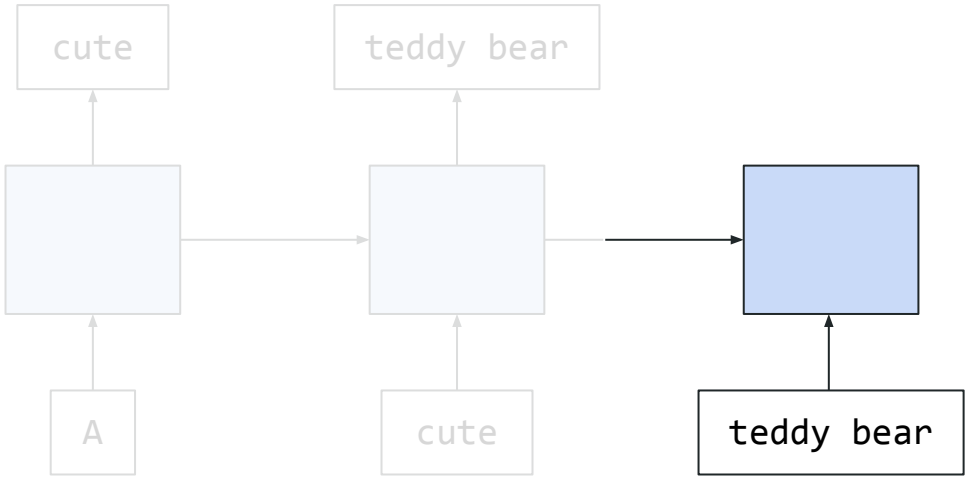
Recurrent Neural Networks (RNNs)



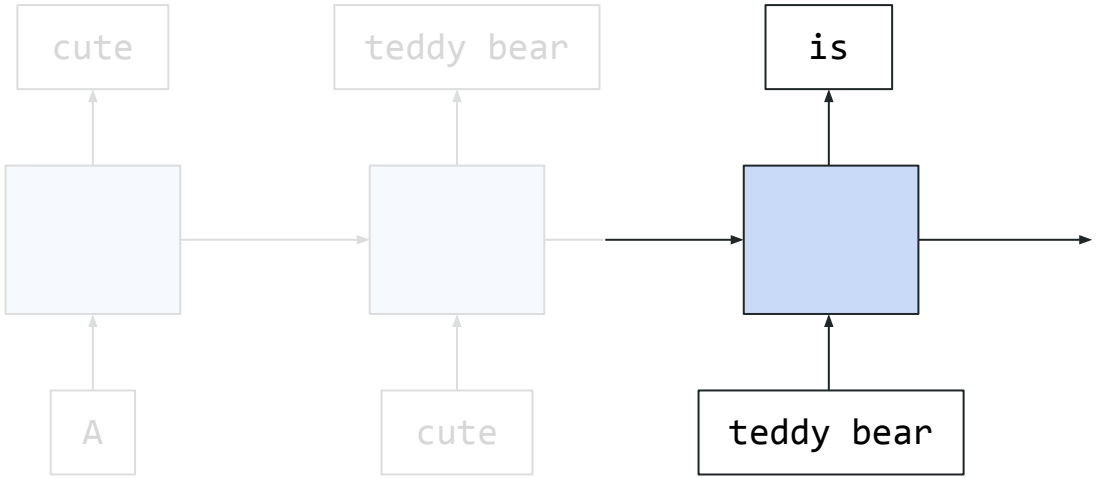
Recurrent Neural Networks (RNNs)



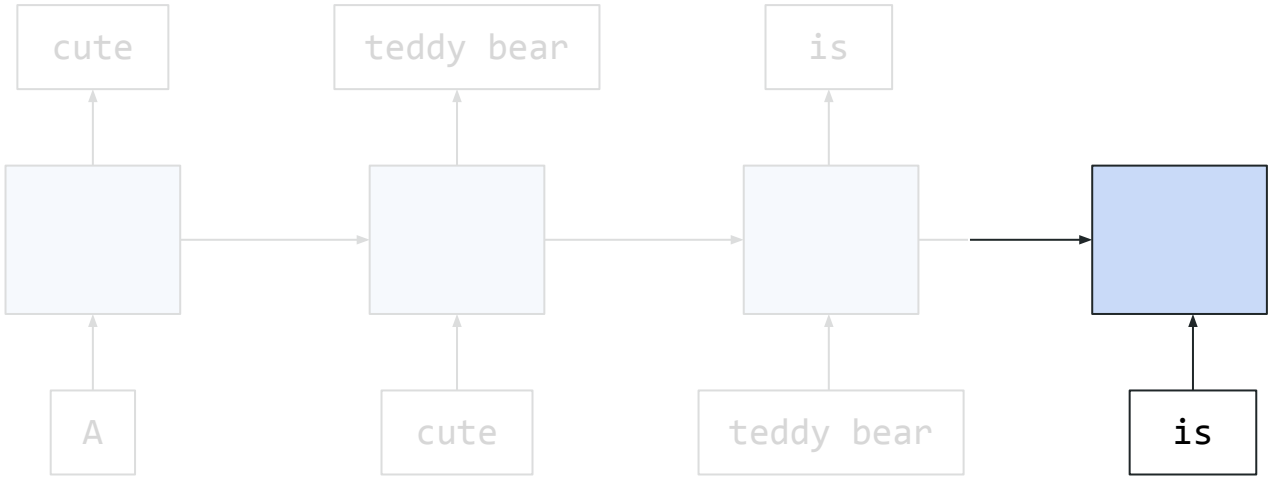
Recurrent Neural Networks (RNNs)



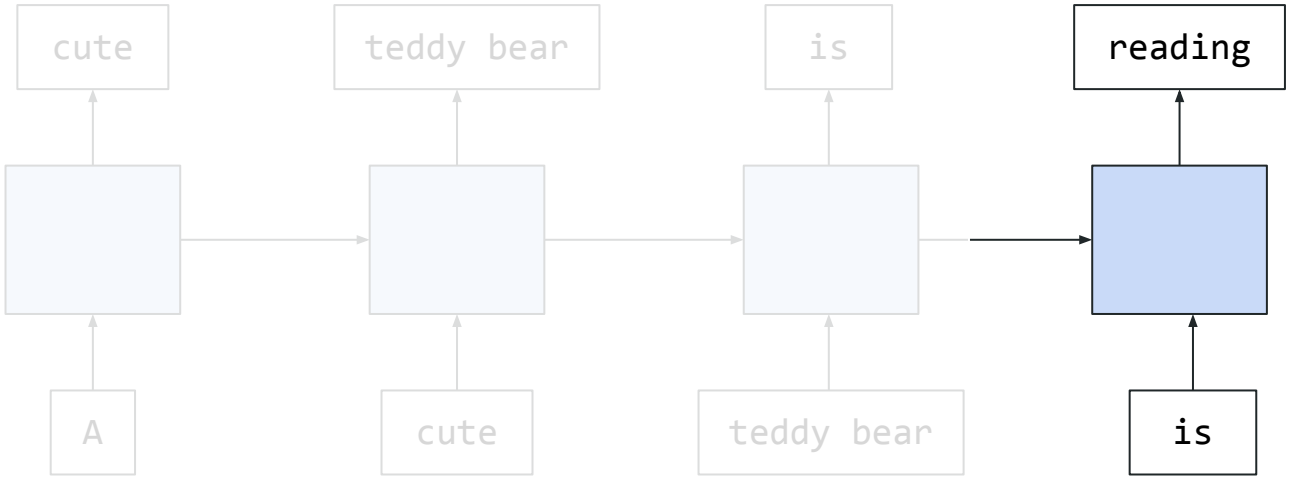
Recurrent Neural Networks (RNNs)



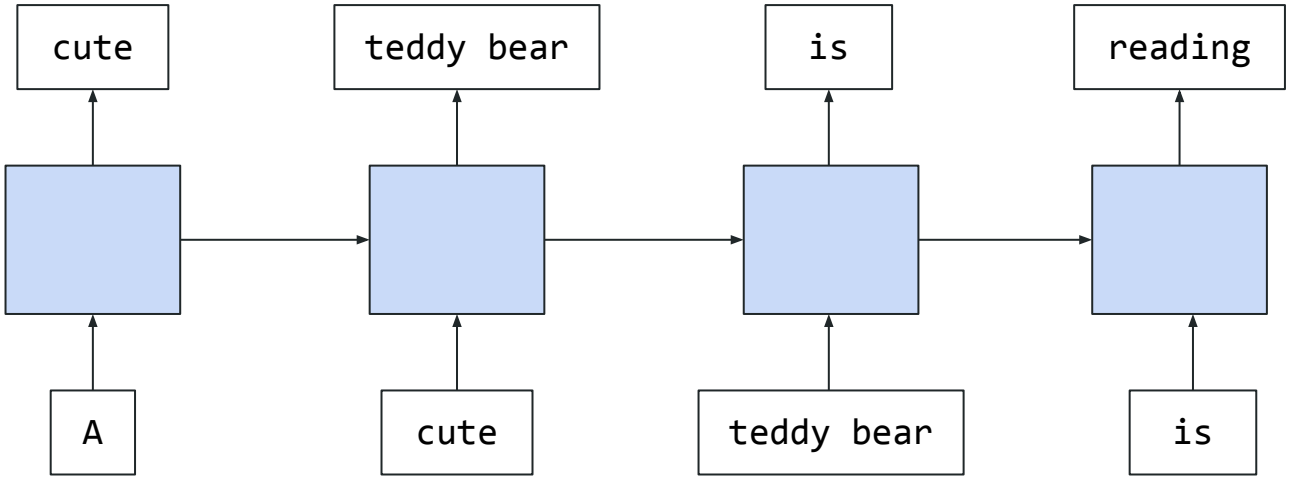
Recurrent Neural Networks (RNNs)



Recurrent Neural Networks (RNNs)



Recurrent Neural Networks (RNNs)

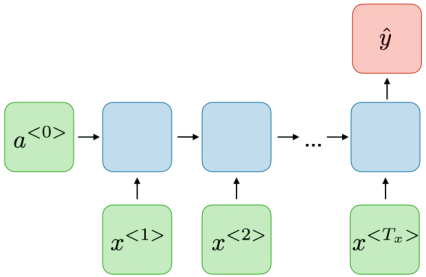


Recurrent Neural Networks (RNNs)

Classification

Input text → Model → 3

Sentiment



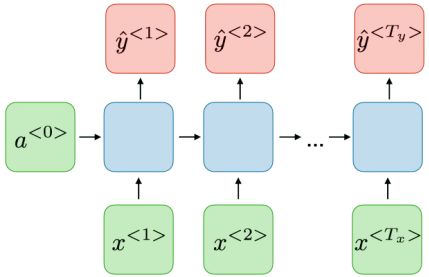
Opinion

"Multi"-classification

Input text → Model → Input text

5 1

Tags

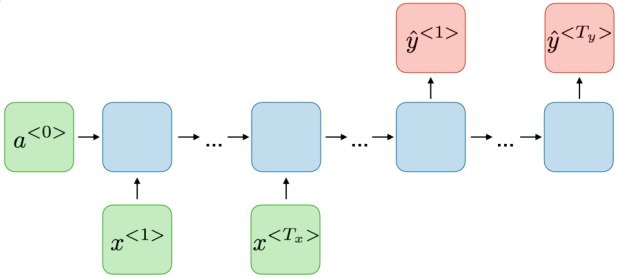


Text

Generation

Input text → Model → Out text

Translation



Source

Figure adapted from "VIP cheatsheets for Stanford's CS 230", Amidi. stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Long Short-Term Memory (LSTM)

Overview

- Introduced in "Long short-term memory" (1997)
- Uses a more structured approach in the cell's hidden state

General form

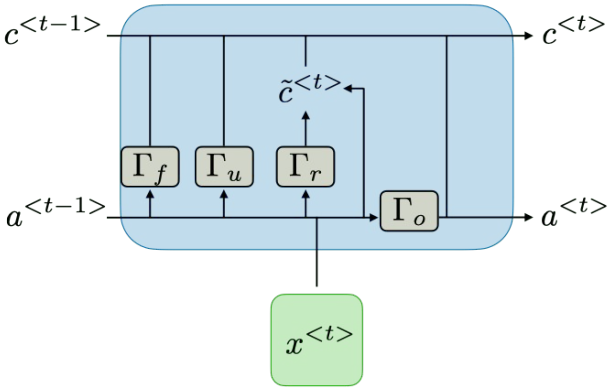


Figure adapted from "VIP cheatsheets for Stanford's CS 230", Amidi. stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Summary of main methods (non-exhaustive list)

Method	Pros	Cons
Word2vec e.g. CBOW, Skip-gram	• Very simple, yet powerful • Intuitive embeddings	• Word order does not count • Embeddings not context aware
Recurrent Neural Networks e.g. traditional RNN, LSTM	• Word order matters • State-of-the-art results	• Vanishing gradient problem • Slow computations

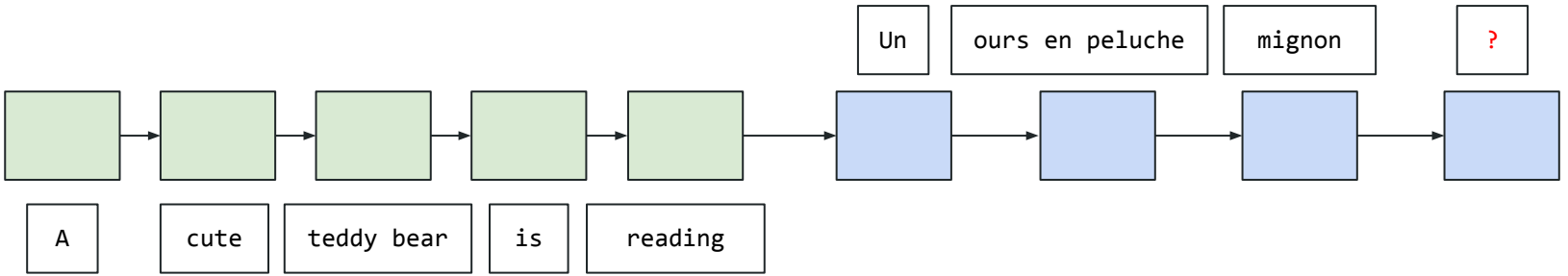
History of attention

- Introduced in 2014
- Translation tasks had a real issue with long-term dependencies
- Seq2seq unable to "remember" what input sentence was saying

"Neural Machine Translation by Jointly Learning to Align and Translate", Bahdanau et al., 2014.

History of attention

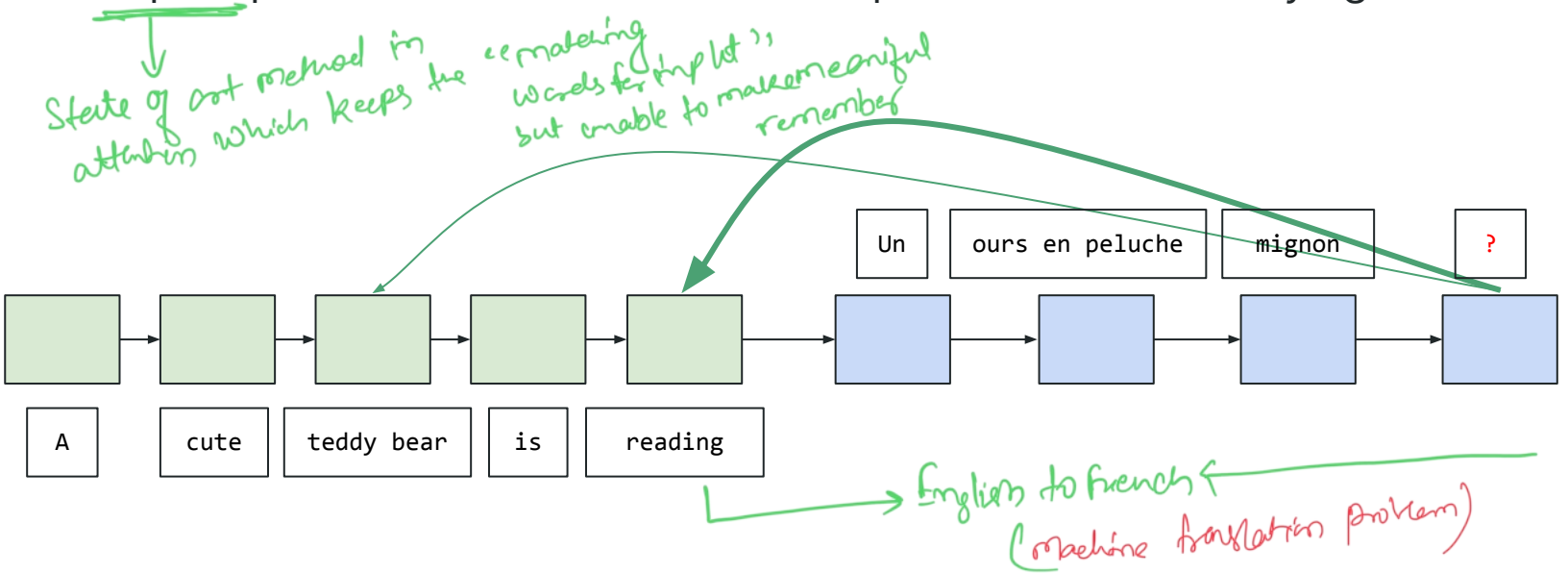
- Introduced in 2014
- Translation tasks had a real issue with long-term dependencies
- Seq2seq unable to "remember" what input sentence was saying



"Neural Machine Translation by Jointly Learning to Align and Translate", Bahdanau et al., 2014.

History of attention

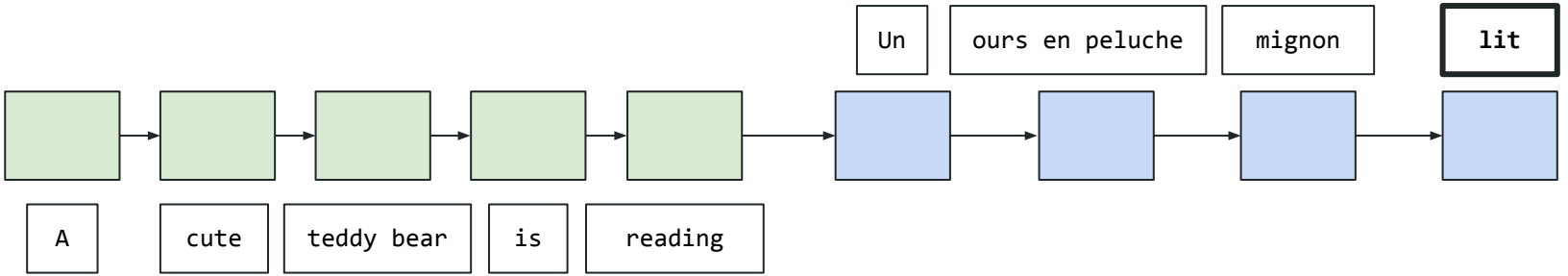
- Introduced in 2014
- Translation tasks had a real issue with long-term dependencies
- Seq2seq unable to "remember" what input sentence was saying



"Neural Machine Translation by Jointly Learning to Align and Translate", Bahdanau et al., 2014.

History of attention

- Introduced in 2014
- Translation tasks had a real issue with long-term dependencies
- Seq2seq unable to "remember" what input sentence was saying



"Neural Machine Translation by Jointly Learning to Align and Translate", Bahdanau et al., 2014.

Transformers & Large Language Models

NLP overview

Tokenization

Word representation

RNNs

Self-attention mechanism

Transformer architecture

End-to-end example

Overview of the Transformer

transformer also solve the problem of long-term dependencies, memory, matching-Context pair".

- Introduced in the 2017 paper "Attention is All You Need"
- Relies on the **self-attention** mechanism
- State-of-the-art results on machine translation tasks

↳ it says "move away from sequential processing of text and let the model have direct connection between all parts of texts at once" i.e. each token will be connection with all

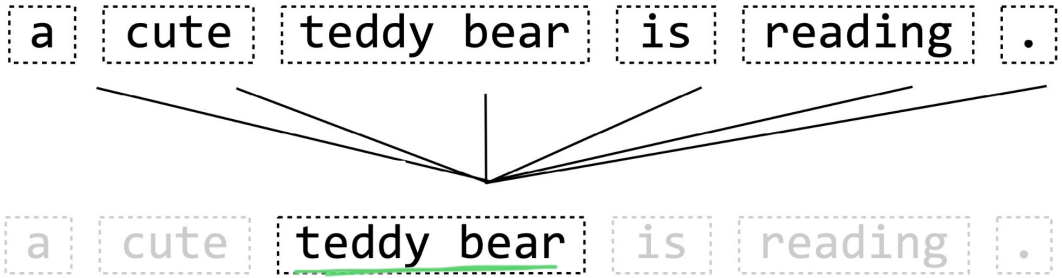
Overview of the Transformer

- Introduced in the 2017 paper "Attention is All You Need"
- Relies on the **self-attention** mechanism
- State-of-the-art results on machine translation tasks

a cute teddy bear is reading .

Overview of the Transformer

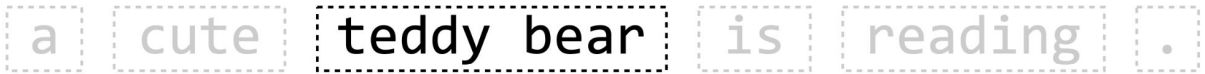
- Introduced in the 2017 paper "Attention is All You Need"
- Relies on the **self-attention** mechanism
- State-of-the-art results on machine translation tasks



Attention mechanism

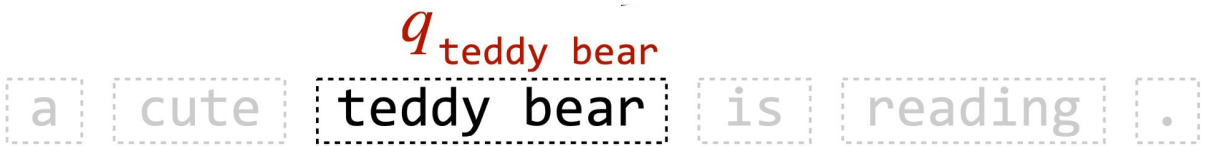
Concept of **Q**uery, **K**ey, **V**alue

In the example, "what are token are similar to teddy bear"
It is a query → using query tries to identify the similar key which should have been closer to already provided key-value pair dataset.



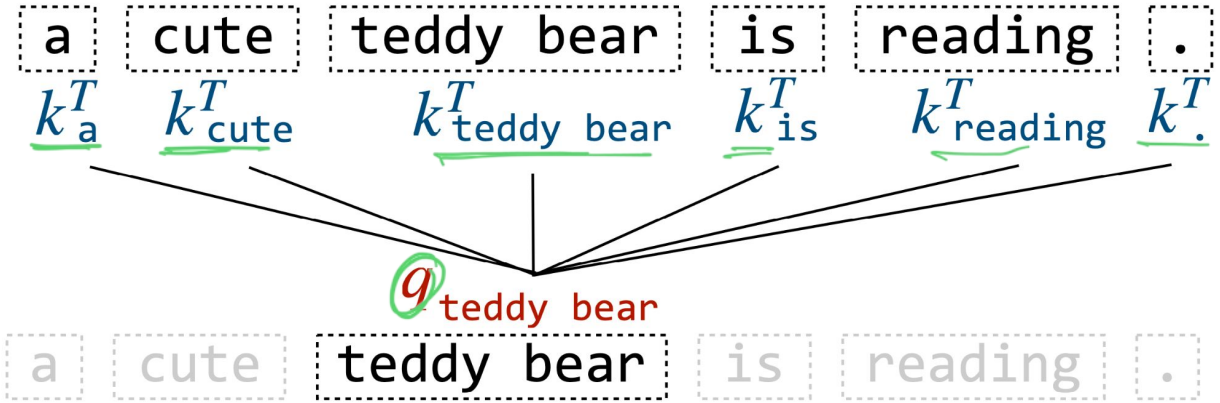
Attention mechanism

Concept of **Q**uery, **K**ey, **V**alue



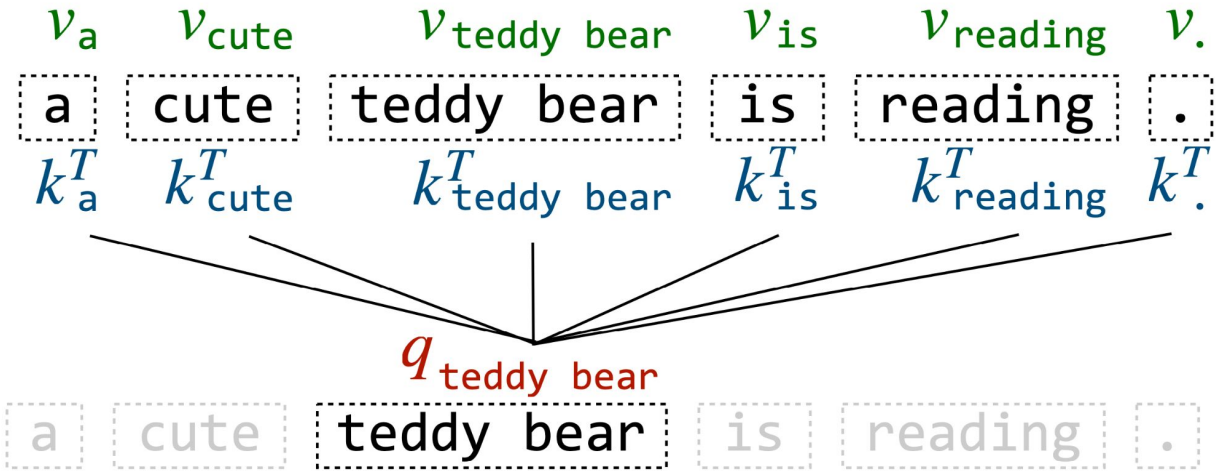
Attention mechanism

Concept of **Q**uery, **K**ey, **V**alue



Attention mechanism

Concept of **Q**uery, **K**ey, **V**alue



Attention mechanism

Benefit of using self-attention mechanism here input can be representing using matrix format and GPU loves matrix so, computation is not the problem.

Efficient computations with matrices:

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

This gives the value in terms of probability which says that we are based on given query, which value will be more important smaller important will have smaller weight and higher important will have higher weight

Attention mechanism

Efficient computations with matrices:

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

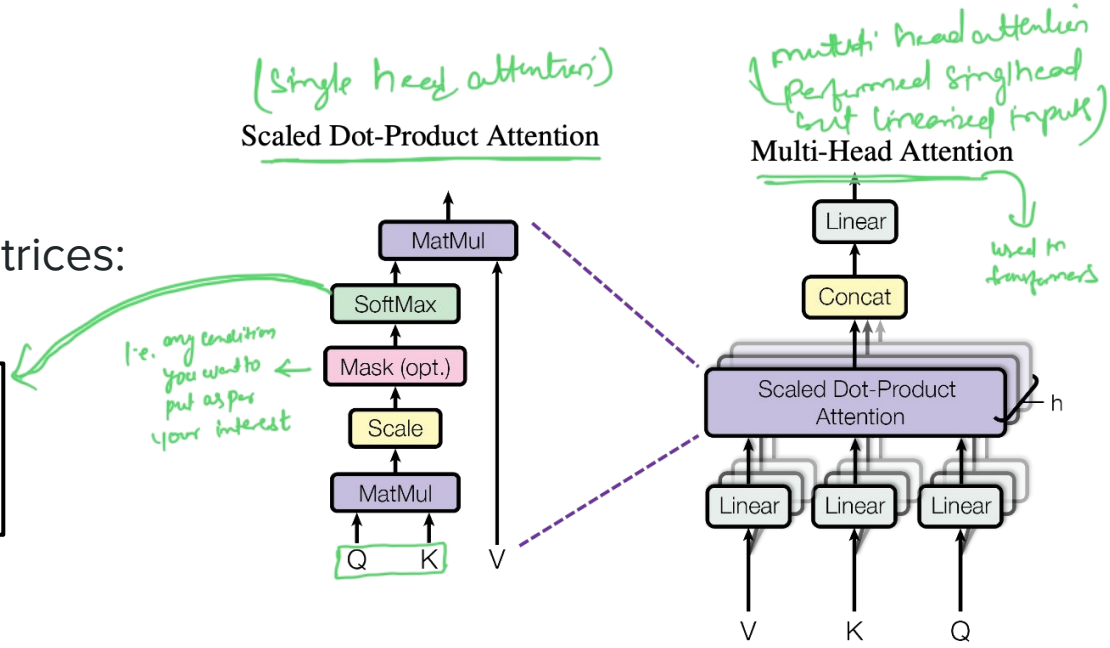


Figure adapted from "Attention Is All You Need", Vaswani et al., 2017.

Transformers & Large Language Models

NLP overview

Tokenization

Word representation

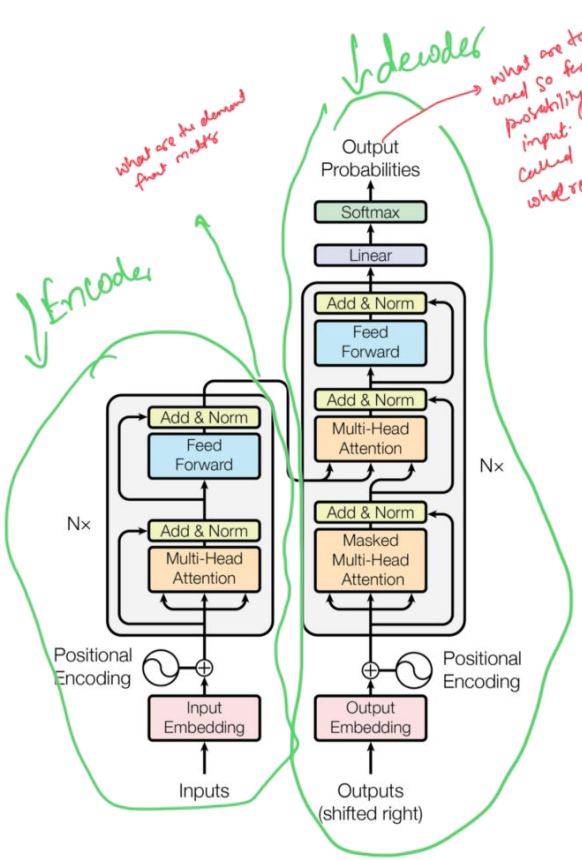
RNNs

Self-attention mechanism

Transformer architecture

End-to-end example

Transformer architecture



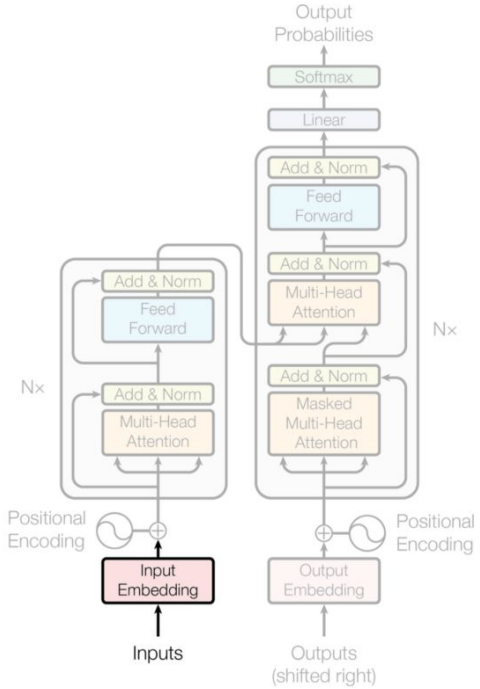
- **Attention layer** (MHA)
  - Self-attention (Encoder-Encoder, Decoder-Decoder)
  - Encoder-Decoder attention layer

Feed Forward Neural Network (FFNN)

Positional Encoding (PE)

Figure adapted from "Attention Is All You Need", Vaswani et al., 2017.

Input



Overview

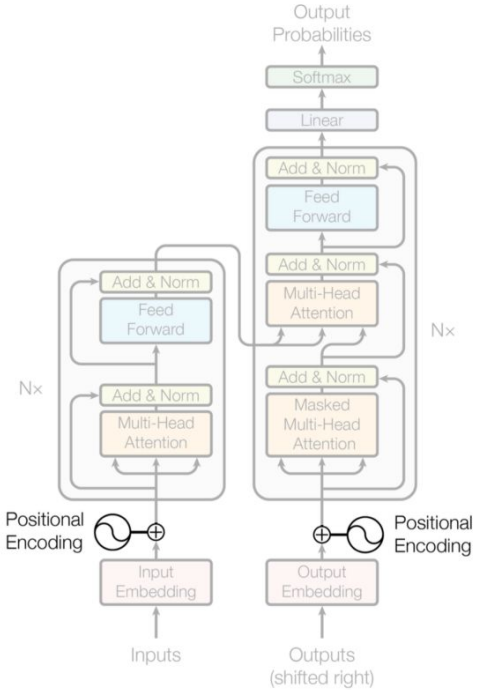
- Text is "tokenized"
- Learned embeddings for tokens

Parameters

- V: vocabulary size
- `d_model`: embedding dimensions

Figure adapted from "Attention Is All You Need", Vaswani et al., 2017.

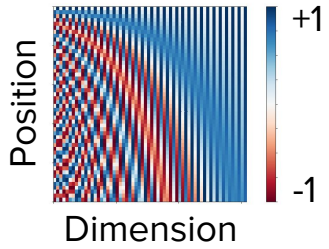
… with a trick!

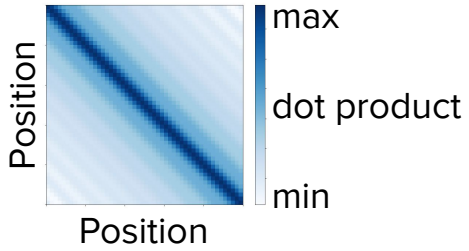


Positional encoding → gives a position information for the embedding of text is being provided as input for relationship b/w all tokens to all.

Idea:

- Add **position information** to inputs
- Can be either learned or hardcoded

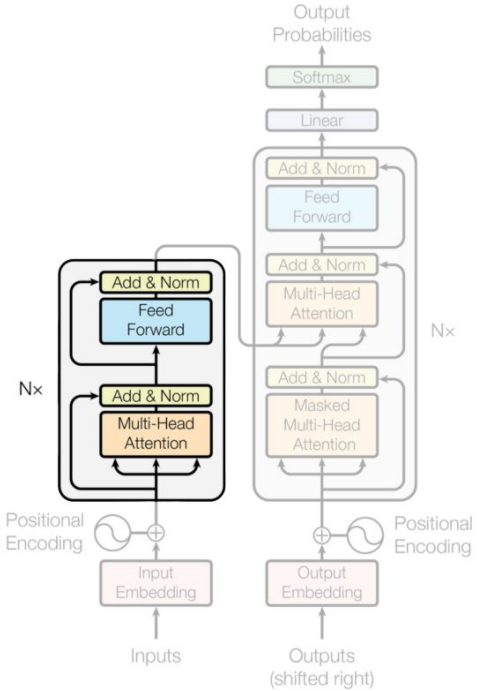




Goal: let model understand relative input position

Left figure adapted from "Attention Is All You Need", Vaswani et al., 2017.
Right figures adapted from "Transformer Architecture: The Positional Encoding", Kazemnejad, 2019.

Encoder



Overview

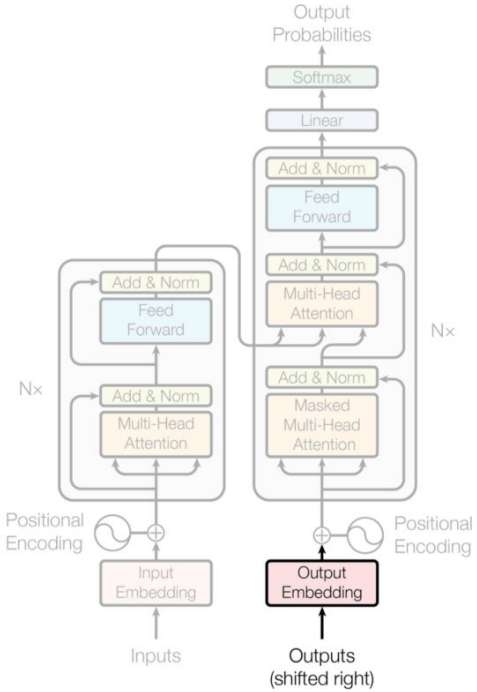
- Encoder-Encoder attention / self-attention
- Feed Forward Neural Network
- Normalization layer

Parameters

- `N`: layers stacked
- `h`: number of attention heads
- `d_FF`, `d_key`, `d_value`: sub-layer dimension
- `d_model`: embedding dimensions

Figure adapted from "Attention Is All You Need", Vaswani et al., 2017.

Output "shifted right"



Overview

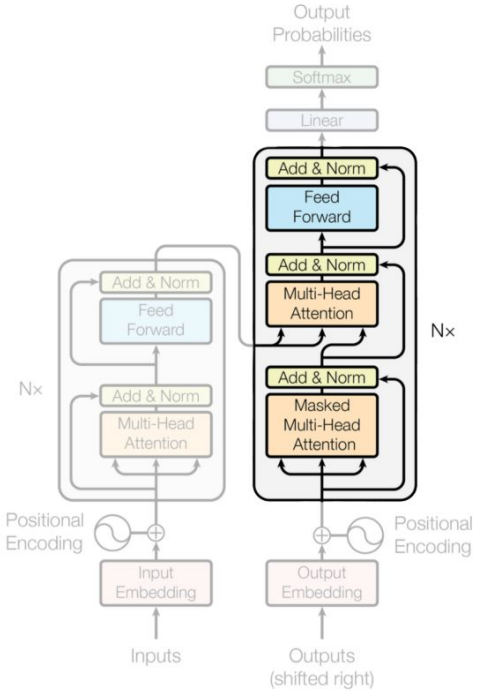
- Learned embeddings for output tokens
- In practice, will start with `[BOS]` during translation

Parameters

- V: vocabulary size
- `d_model`: embedding dimensions

Figure adapted from "Attention Is All You Need", Vaswani et al., 2017.

Decoder



Overview

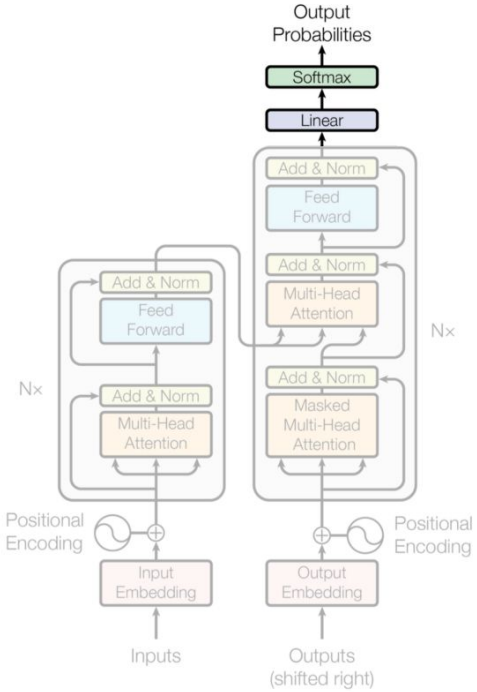
- Decoder-Decoder attention / self-attention
- Encoder-Decoder attention
- Feed Forward Neural Network
- Normalization layer

Parameters

- `N`: layers stacked
- `h`: number of attention heads
- `d_FF`, `d_key`, `d_value`: sub-layer dimension
- `d_model`: embedding dimensions

Figure adapted from "Attention Is All You Need", Vaswani et al., 2017.

Output



Overview

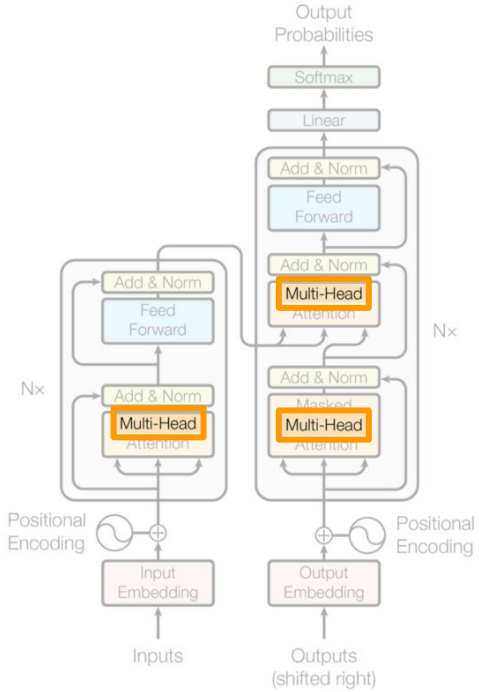
- Linear projection
- Classification problem that outputs probability of belonging to a class, where class = word

Parameters

- V: vocabulary size
- `d_model`: embedding dimensions

Figure adapted from "Attention Is All You Need", Vaswani et al., 2017.

Computational tricks



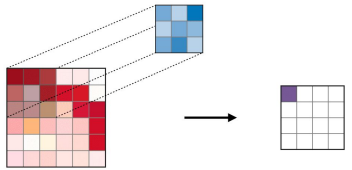
Multi-head attention

Idea:

- Run **multiple** self-attention layers in parallel

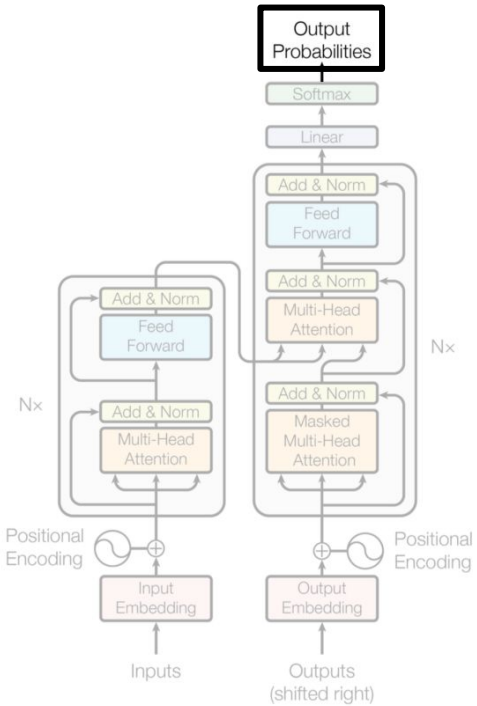
Benefits:

- Enables the model to capture different attention features in parallel
- Comparison: **multiple** filters of a convolutional layer in computer vision



Left figure adapted from "Attention Is All You Need", Vaswani et al., 2017. Right figure from "VIP cheatsheets for Stanford's CS 230", Amidi, 2019.

Computational tricks



Label smoothing

in NLP, when used to predict some tasks
then it has kind of multiple choices so
label smoothing tries to fix the problem of
multiple choices and keeping one choices
more particular/or content"

Idea:

- 2015 vision paper: overconfidence is bad
- Introduce **noise** in true labels

$$q(k|x) = \delta_{k,y} \longrightarrow q'(k|x) = (1-\epsilon)\delta_{k,y} + \epsilon u(k)$$

Benefits:

- General technique that prevents overfitting
- Improves accuracy and BLEU score

Figure adapted from "Attention Is All You Need", Vaswani et al., 2017.

Transformers & Large Language Models

NLP overview

Tokenization

Word representation

RNNs

Self-attention mechanism

Transformer architecture

End-to-end example

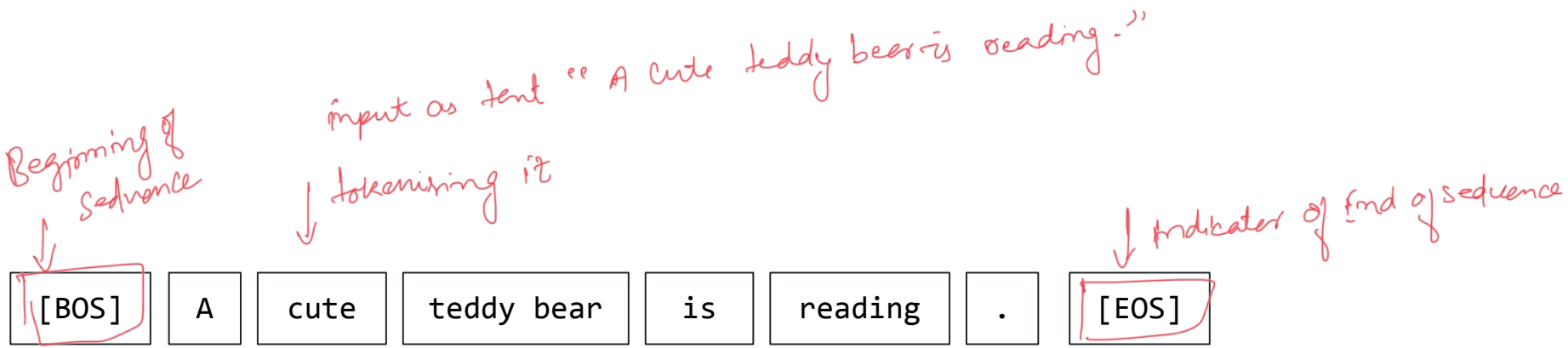
Stitching all the pieces together with an example

```
A cute teddy bear is reading.
```

Stitching all the pieces together with an example

A	cute	teddy bear	is	reading	.

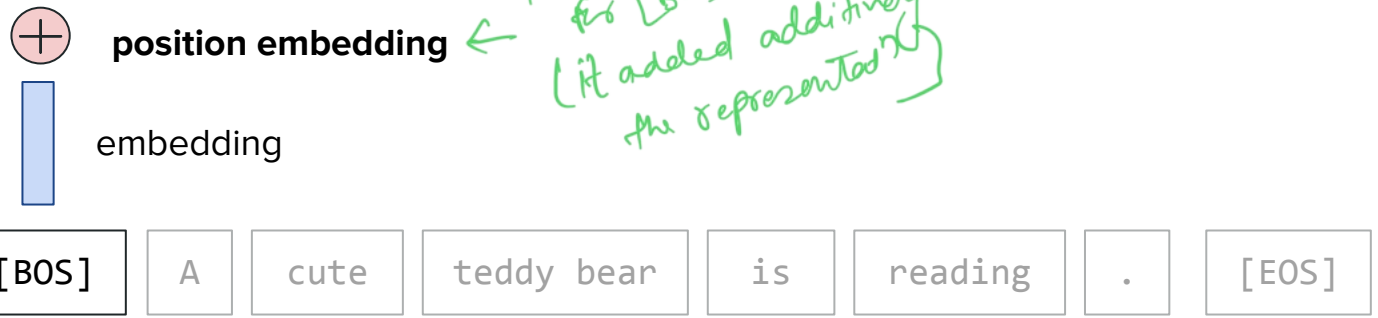
Stitching all the pieces together with an example



Stitching all the pieces together with an example

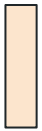
Stitching all the pieces together with an example

Stitching all the pieces together with an example



Provides the position embedding for [BOS] and [EOS] (it added additively with the representation)

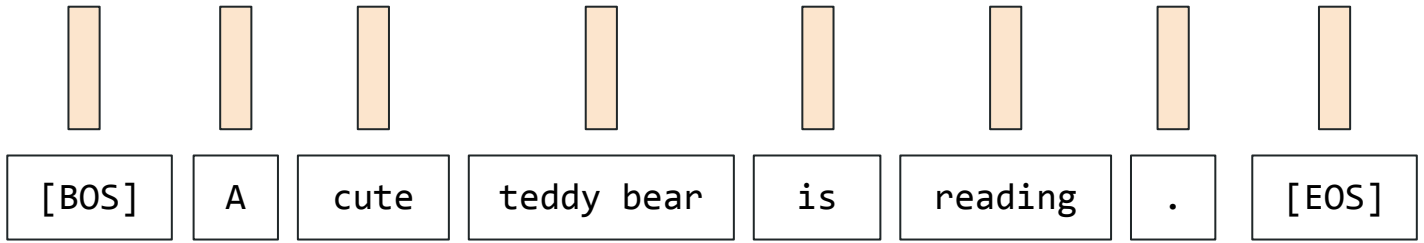
Stitching all the pieces together with an example



position-aware embedding ← bez using position Embedding (based on position, time) we know where [BOS] will add with representation

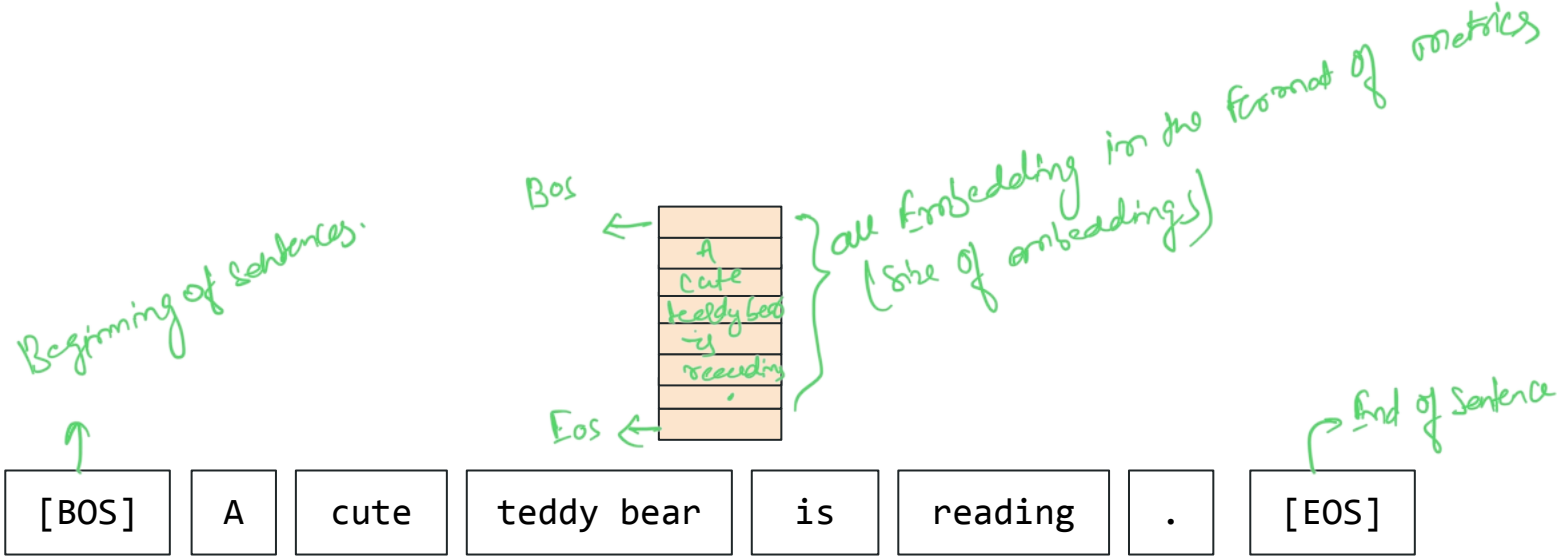
[BOS]	A	cute	teddy bear	is	reading	.	[EOS]

Stitching all the pieces together with an example



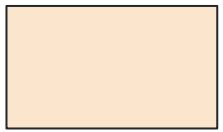
position-aware embeddings

Stitching all the pieces together with an example



position-aware embeddings matrix

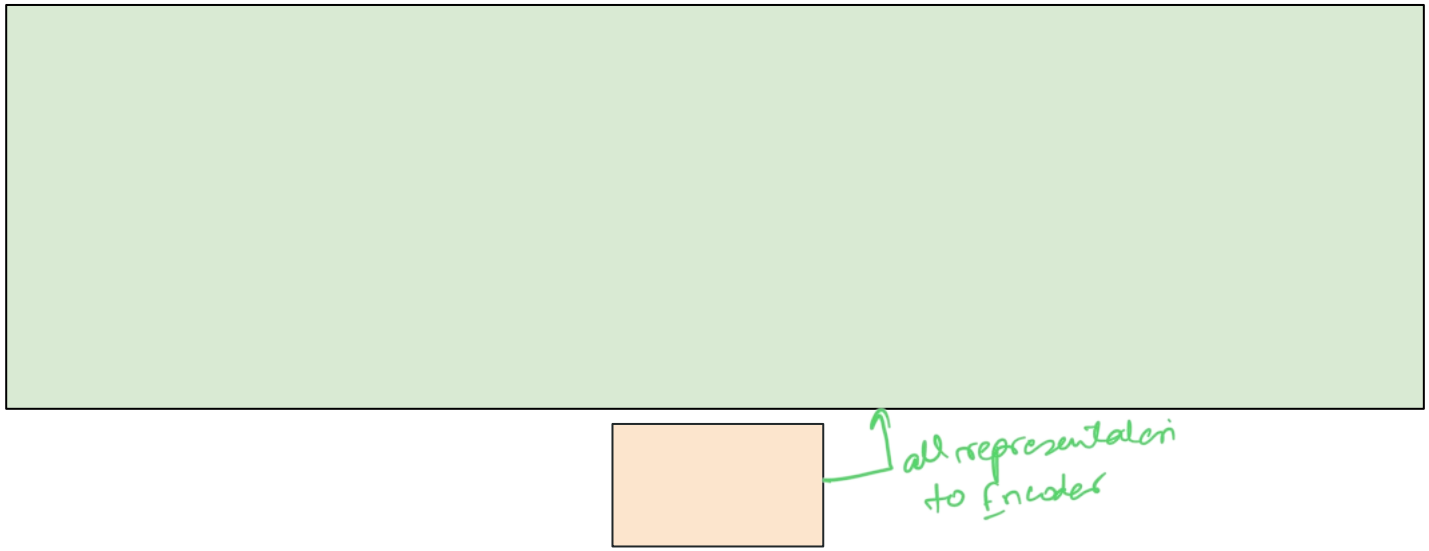
Stitching all the pieces together with an example



position-aware embeddings matrix

[BOS]	A	cute	teddy bear	is	reading	.	[EOS]

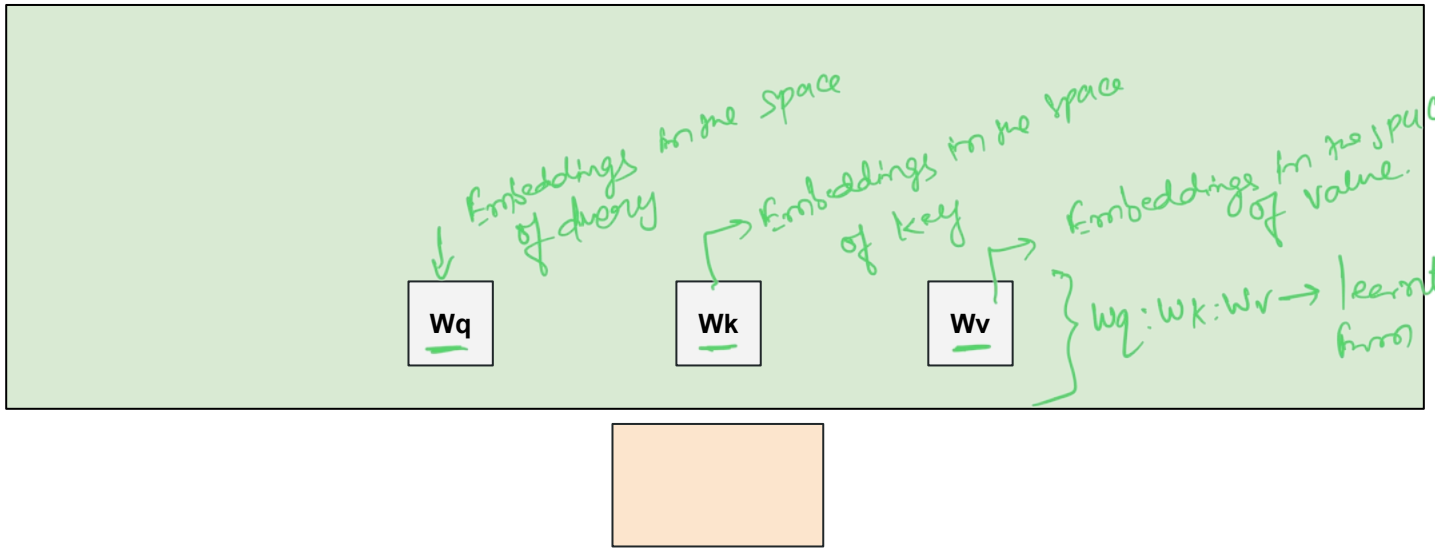
Stitching all the pieces together with an example



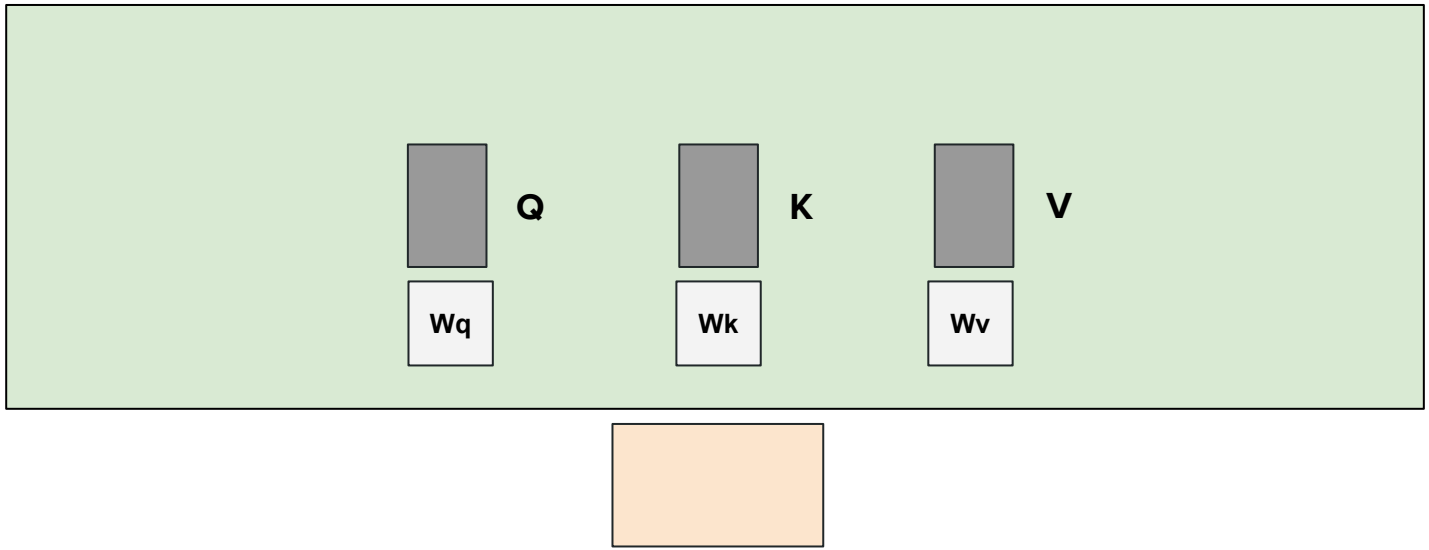
encoder

position-aware embeddings matrix

Stitching all the pieces together with an example



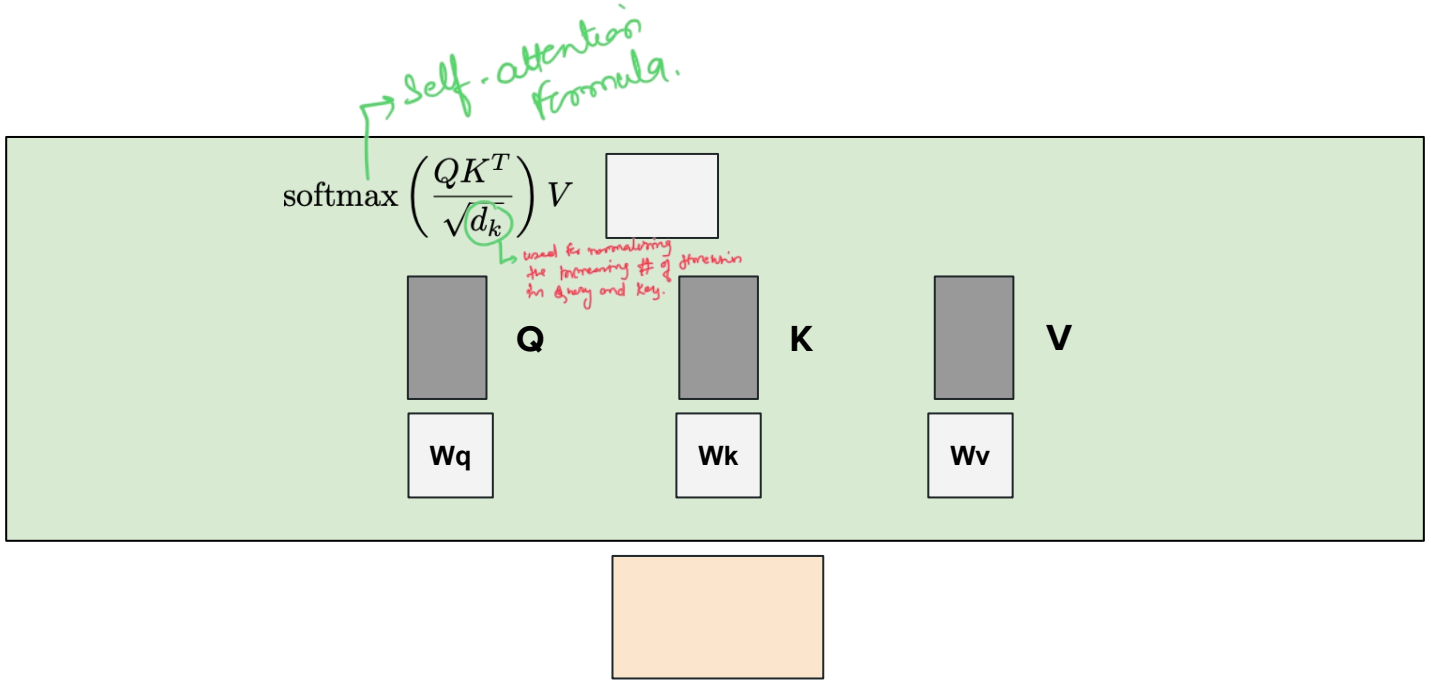
Stitching all the pieces together with an example



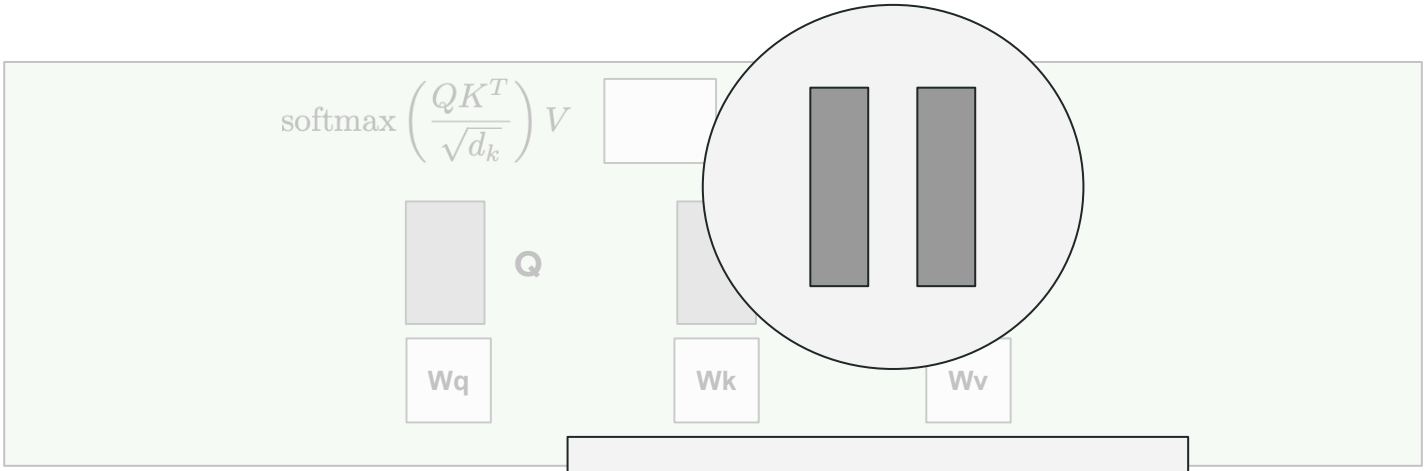
encoder

position-aware embeddings matrix

Stitching all the pieces together with an example



Stitching all the pieces together with an example



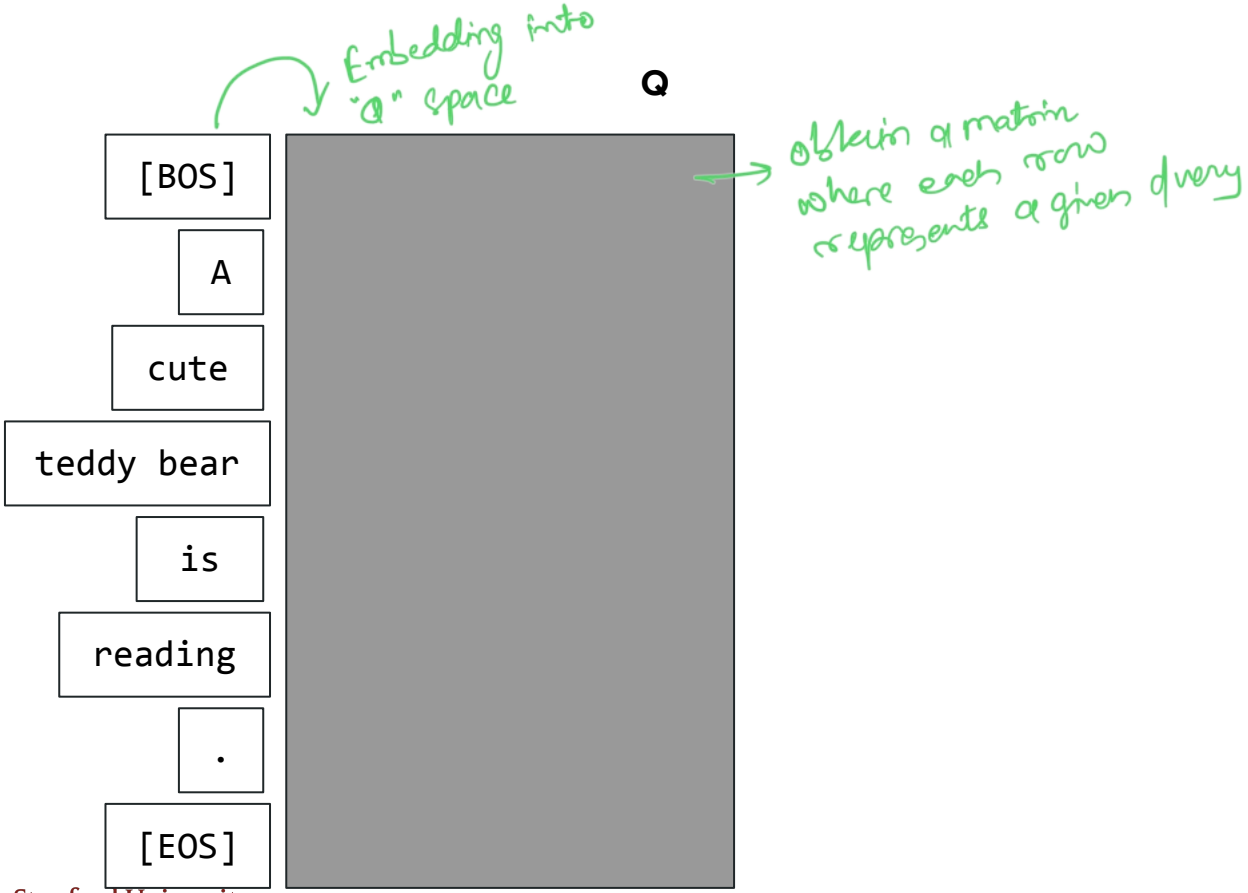
encoder

PAUSE

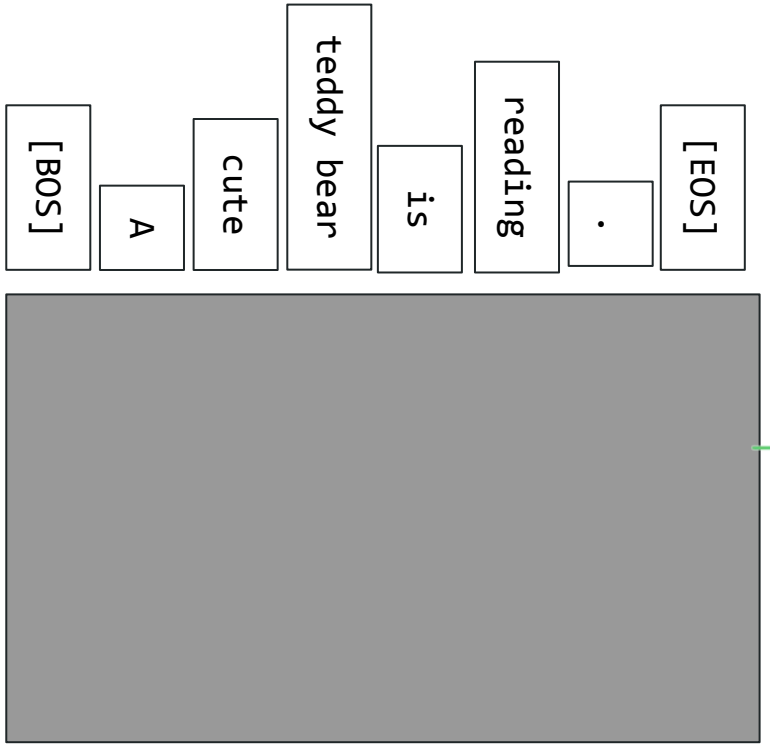
position-aware embeddings matrix

[BOS] A cute teddy bear is reading . [EOS]

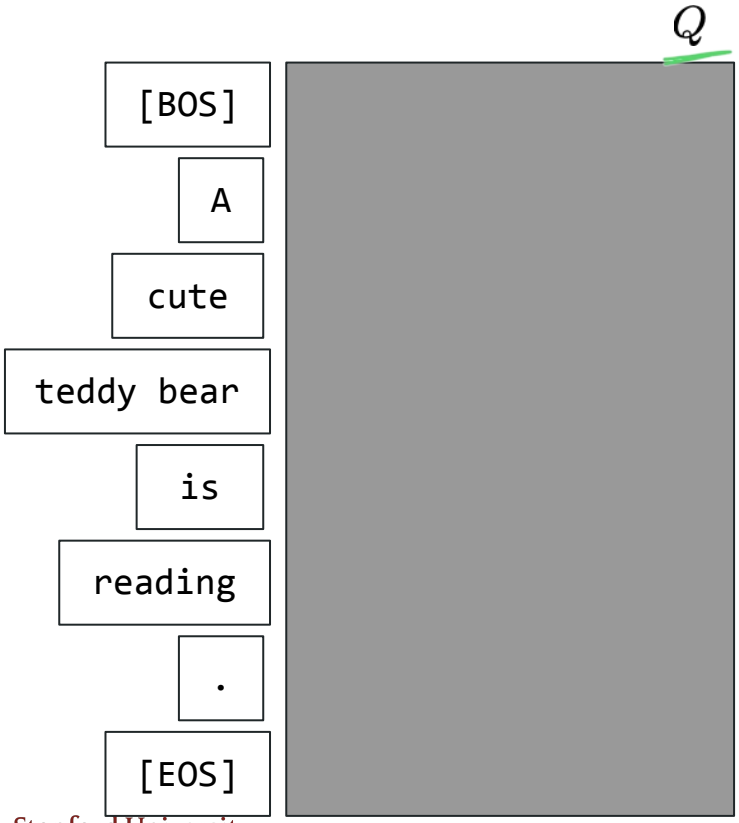
Stitching all the pieces together with an example

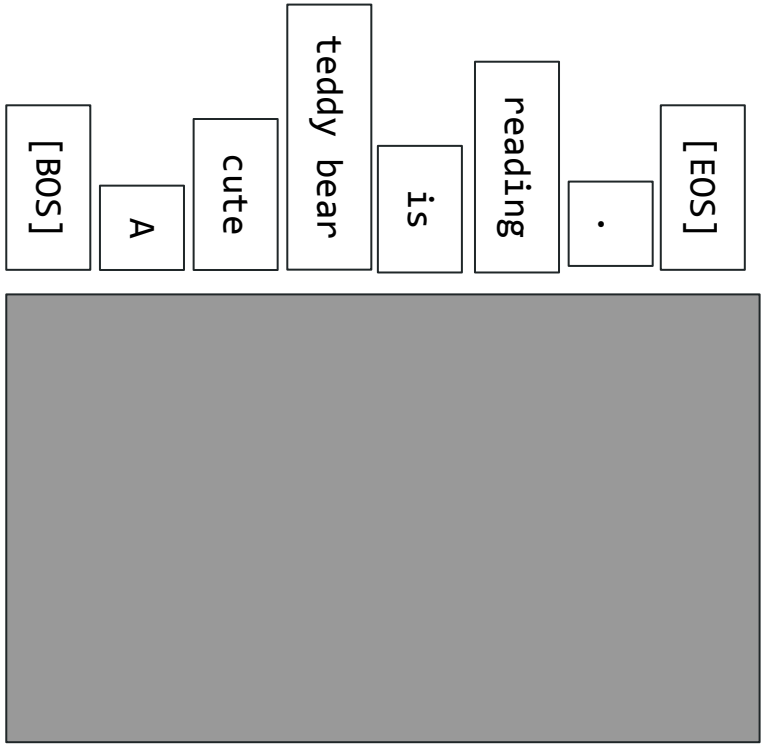


Stitching all the pieces together with an example

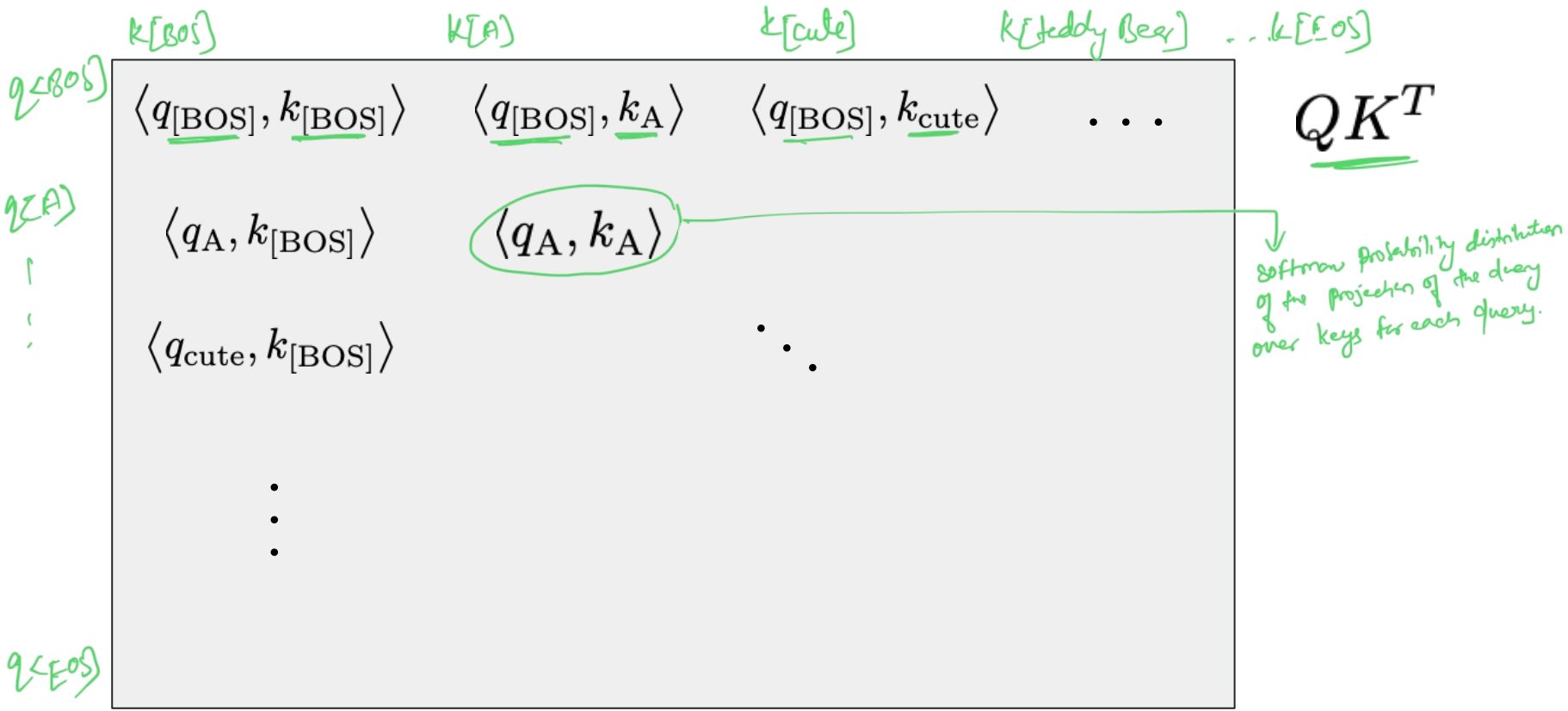


Stitching all the pieces together with an example



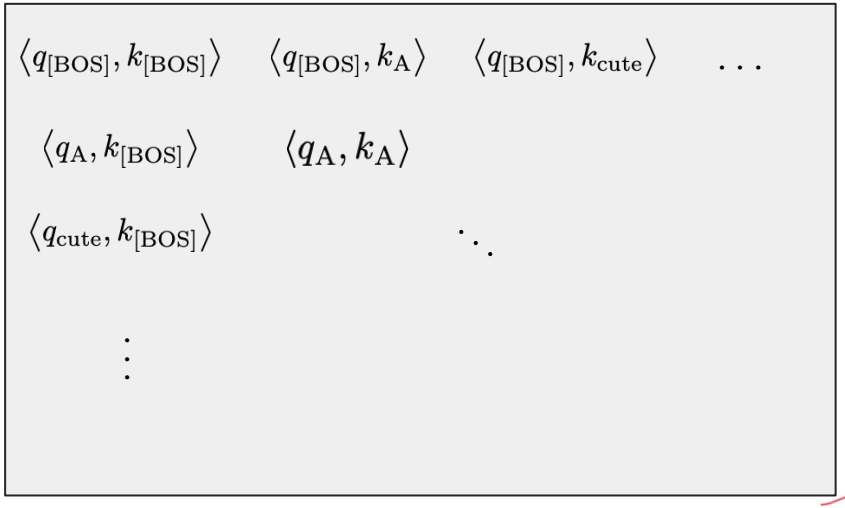


Stitching all the pieces together with an example

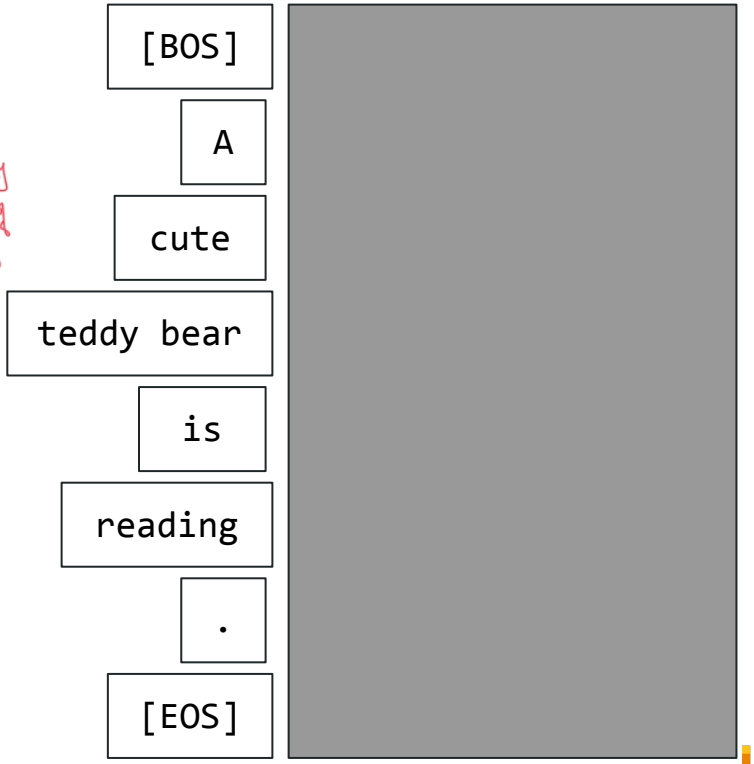


Stitching all the pieces together with an example

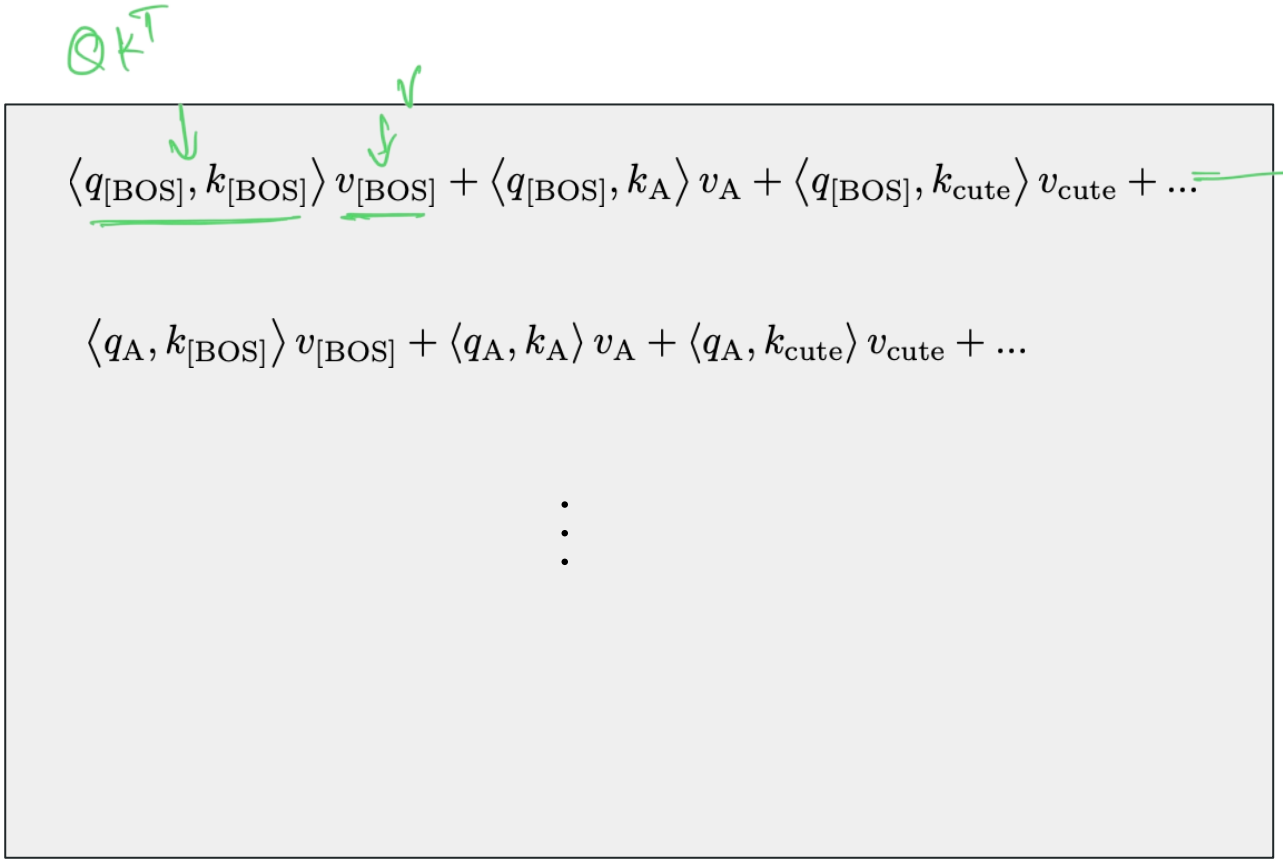
QK^T



V



Stitching all the pieces together with an example



QK^TV

Each row represents the weighted sum of each value with each query.

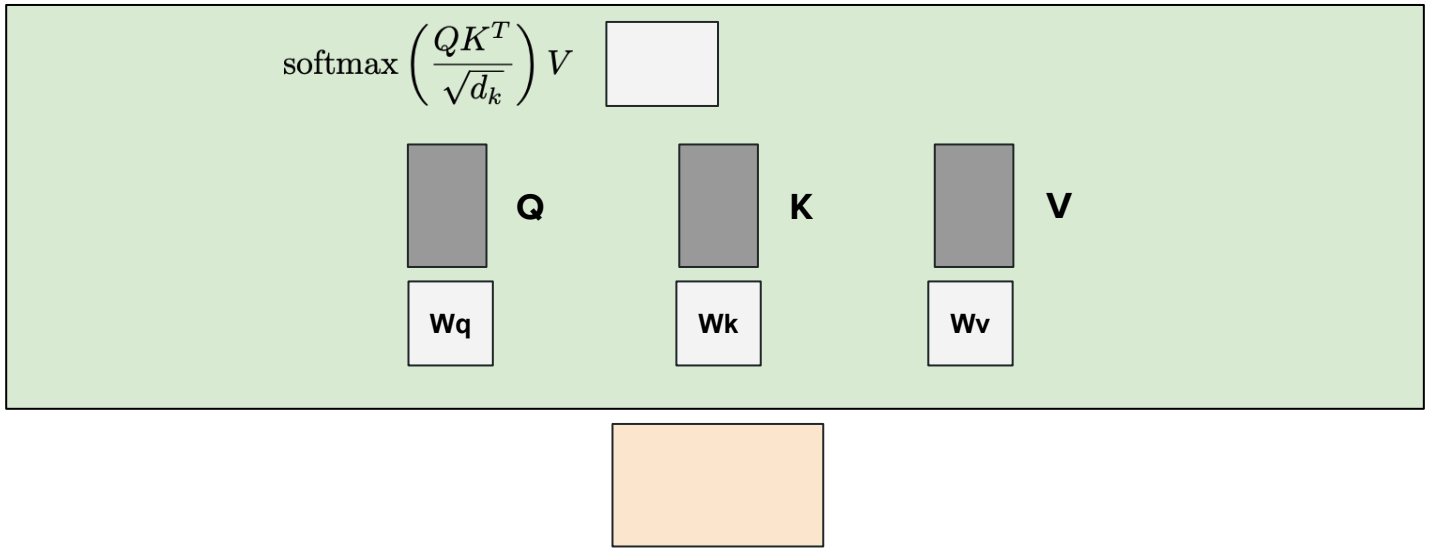
Stitching all the pieces together with an example

weighted average of values

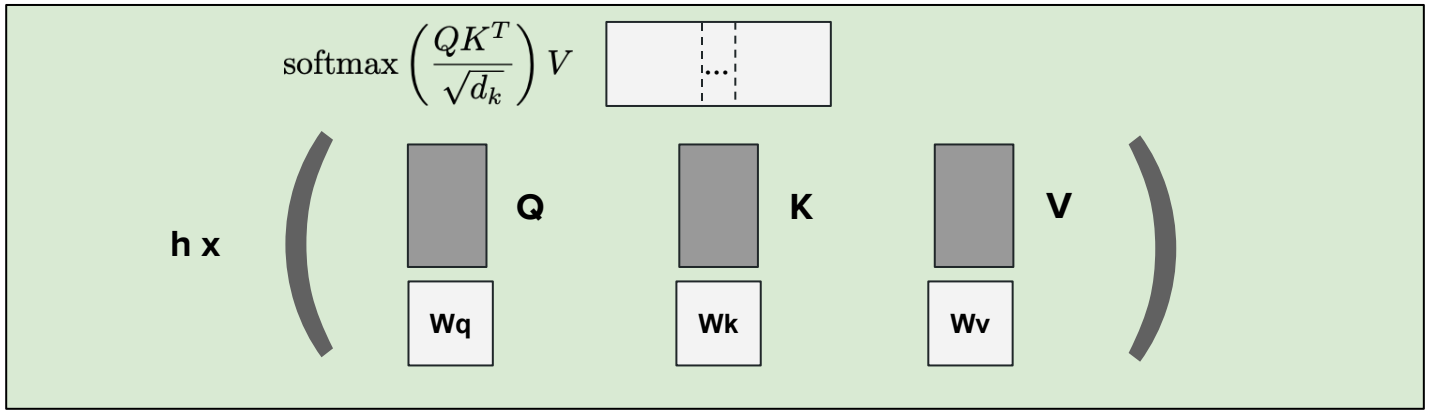
with weights being a function of $\langle q, k \rangle$

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Stitching all the pieces together with an example



Stitching all the pieces together with an example

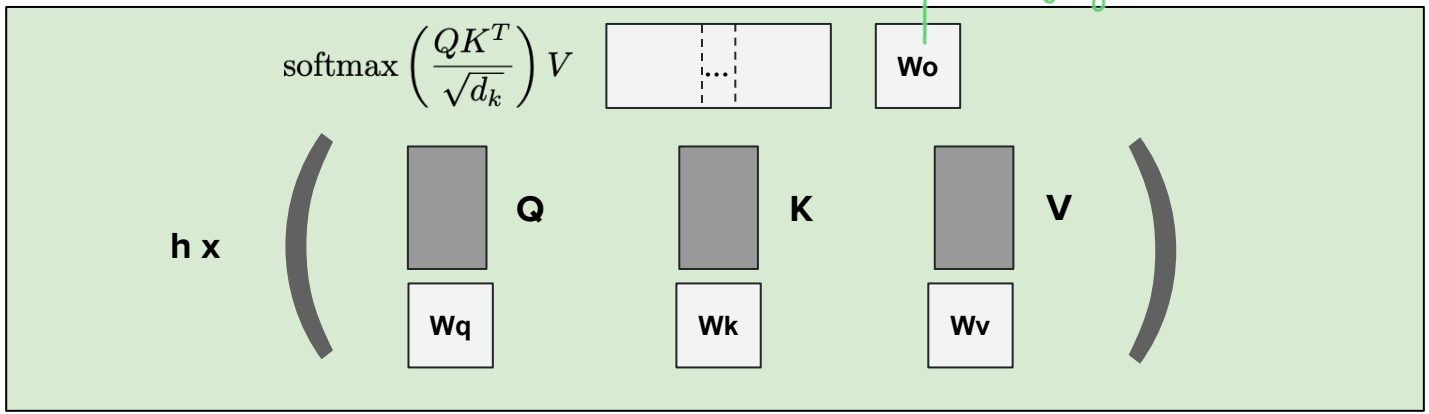


encoder

position-aware embeddings matrix

[BOS]	A	cute	teddy bear	is	reading	.	[EOS]

Stitching all the pieces together with an example

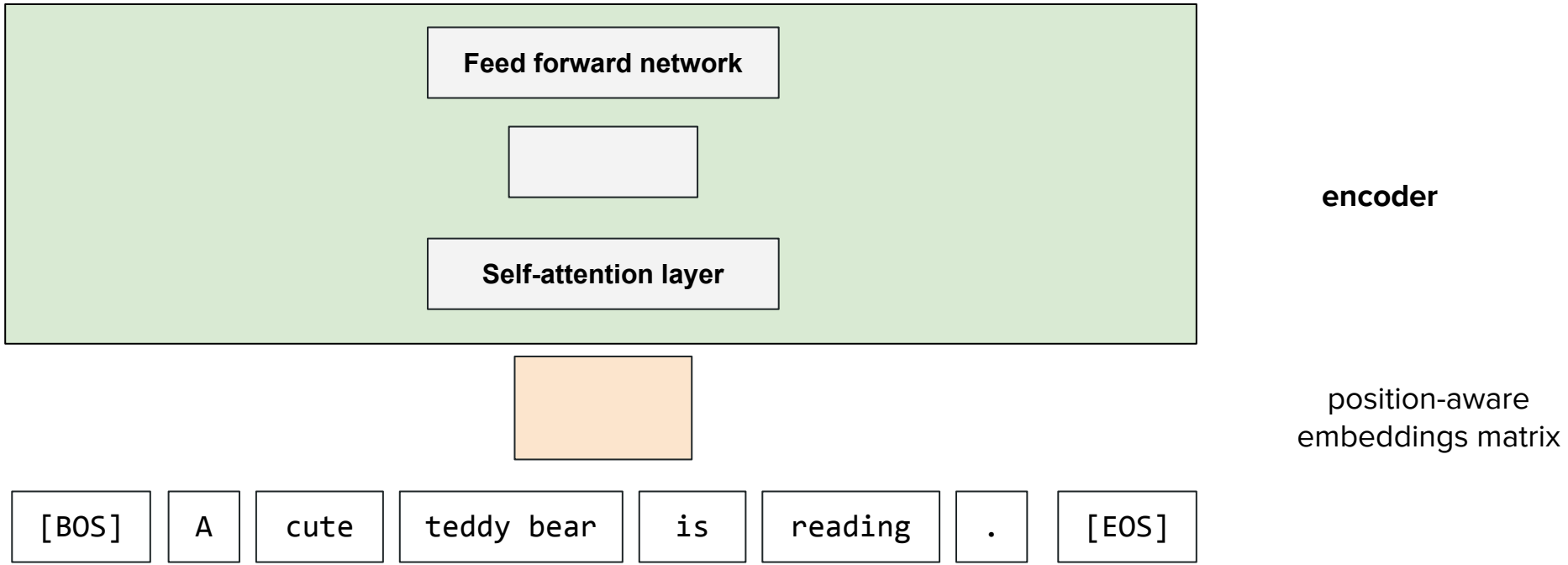


encoder

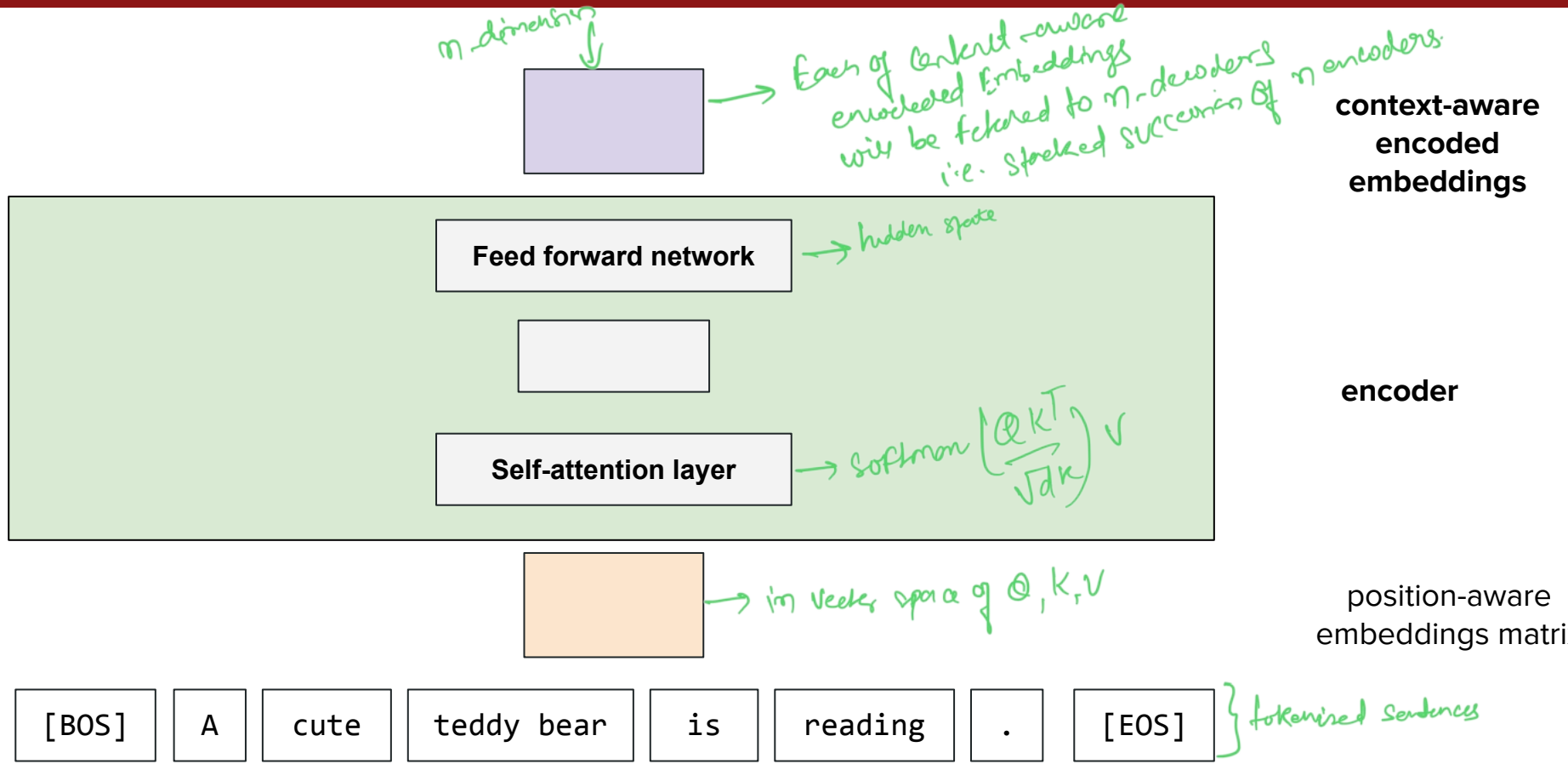
position-aware embeddings matrix

[BOS] A cute teddy bear is reading . [EOS]

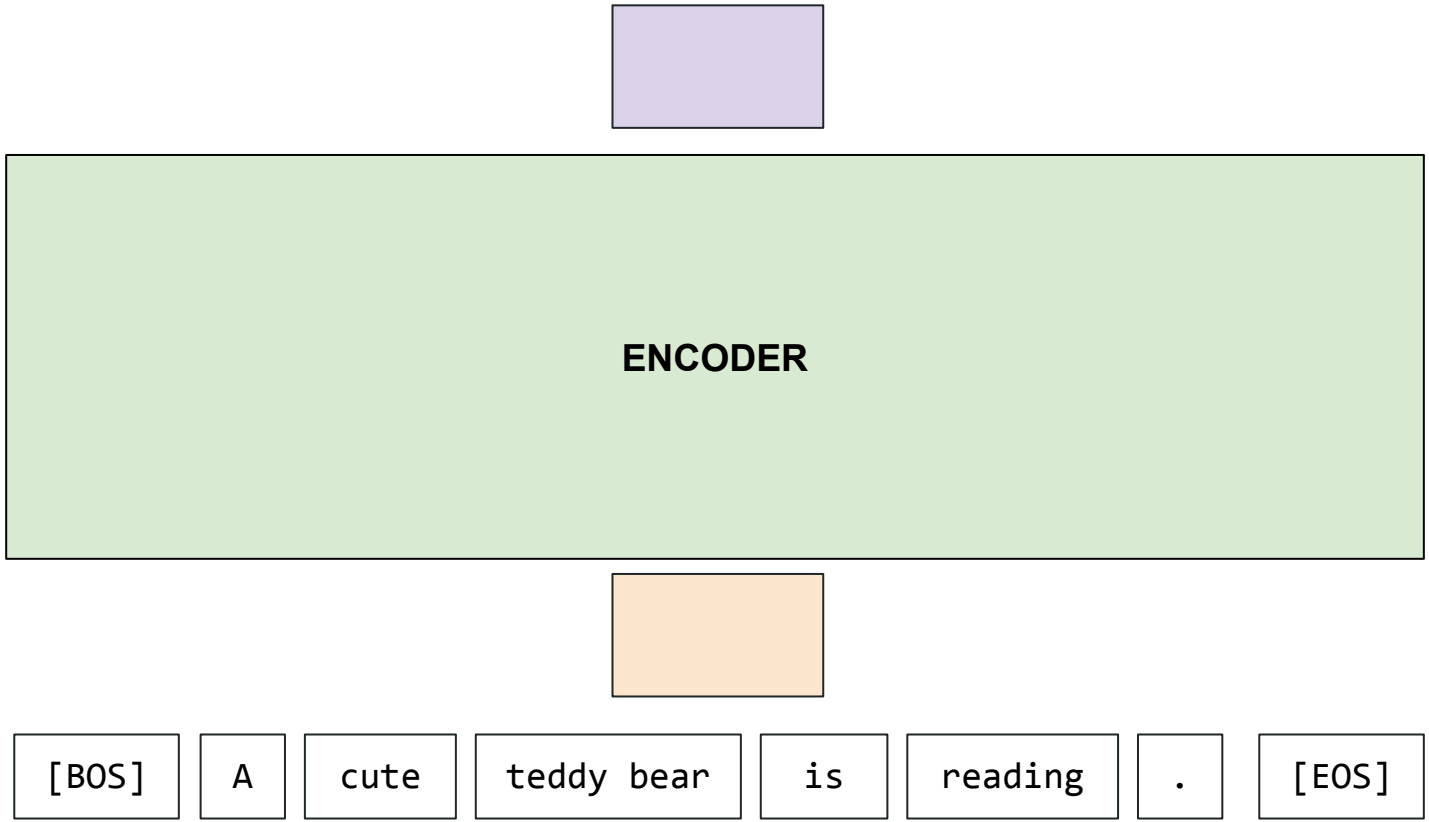
Stitching all the pieces together with an example



Stitching all the pieces together with an example

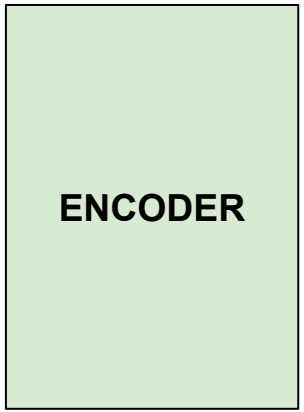


Stitching all the pieces together with an example



Stitching all the pieces together with an example

encoded
embedding




```
A cute teddy bear
   is reading.
```

Stitching all the pieces together with an example

encoded embedding

ENCODER

A cute teddy bear is reading.

Stitching all the pieces together with an example

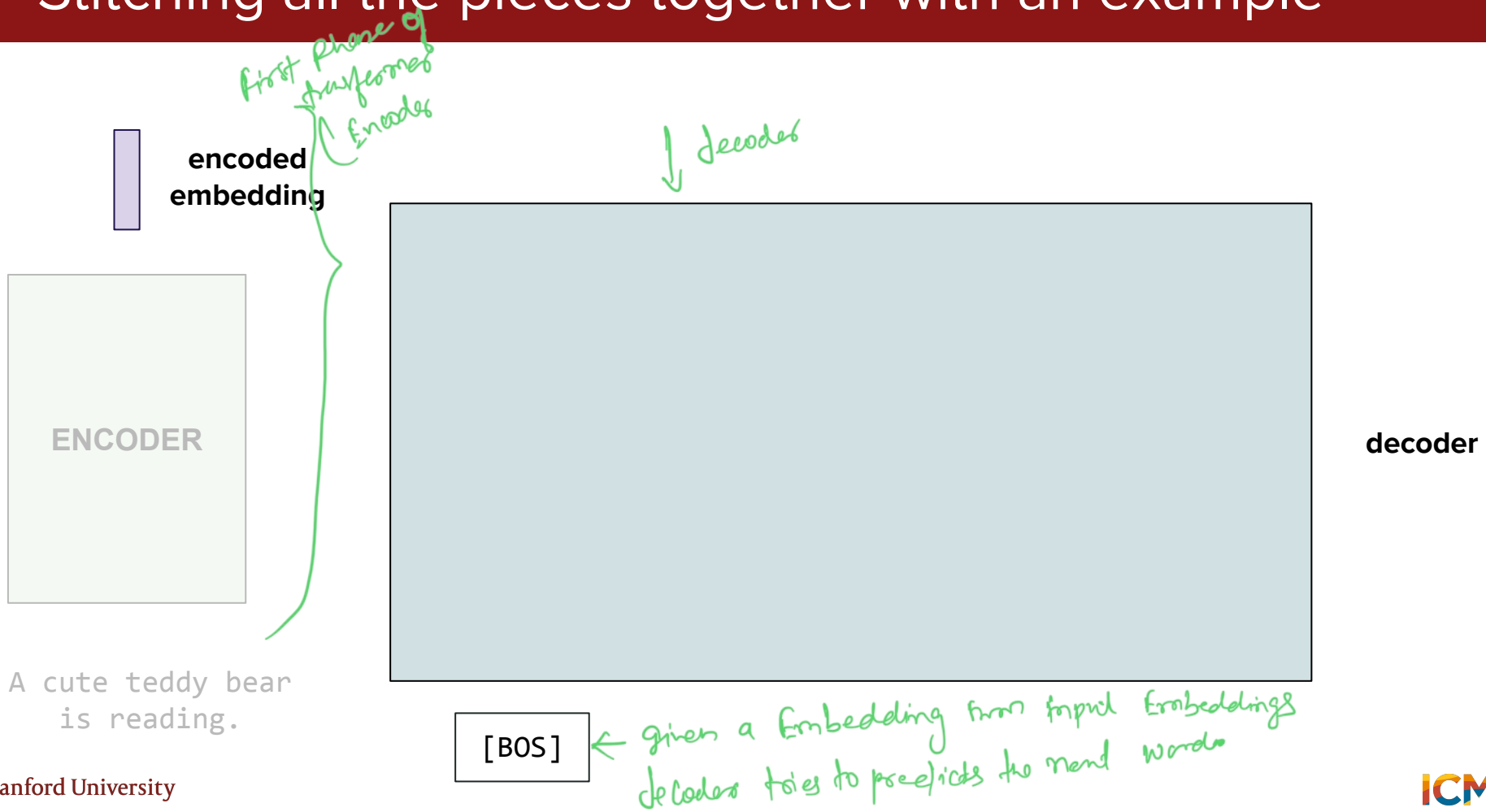
encoded
embedding

ENCODER

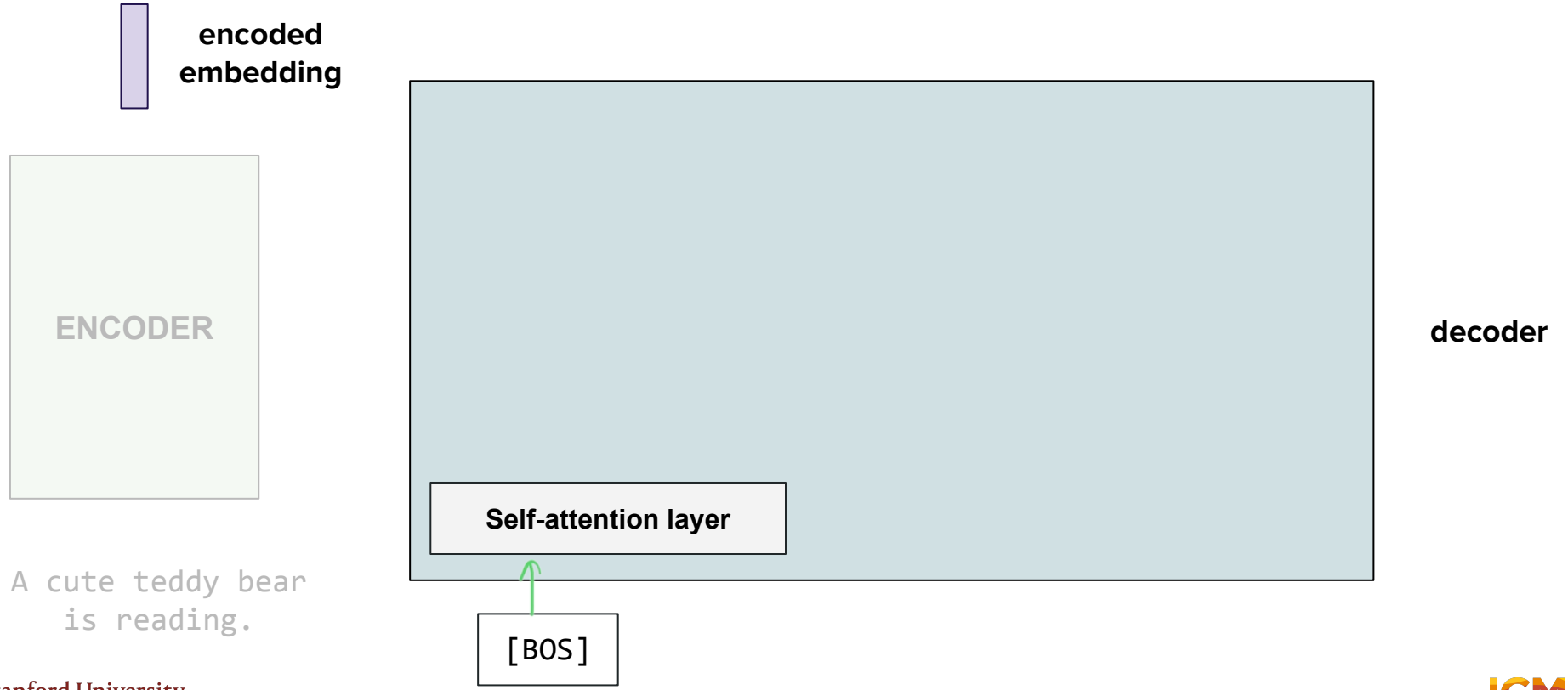
A cute teddy bear
is reading.

[BOS]

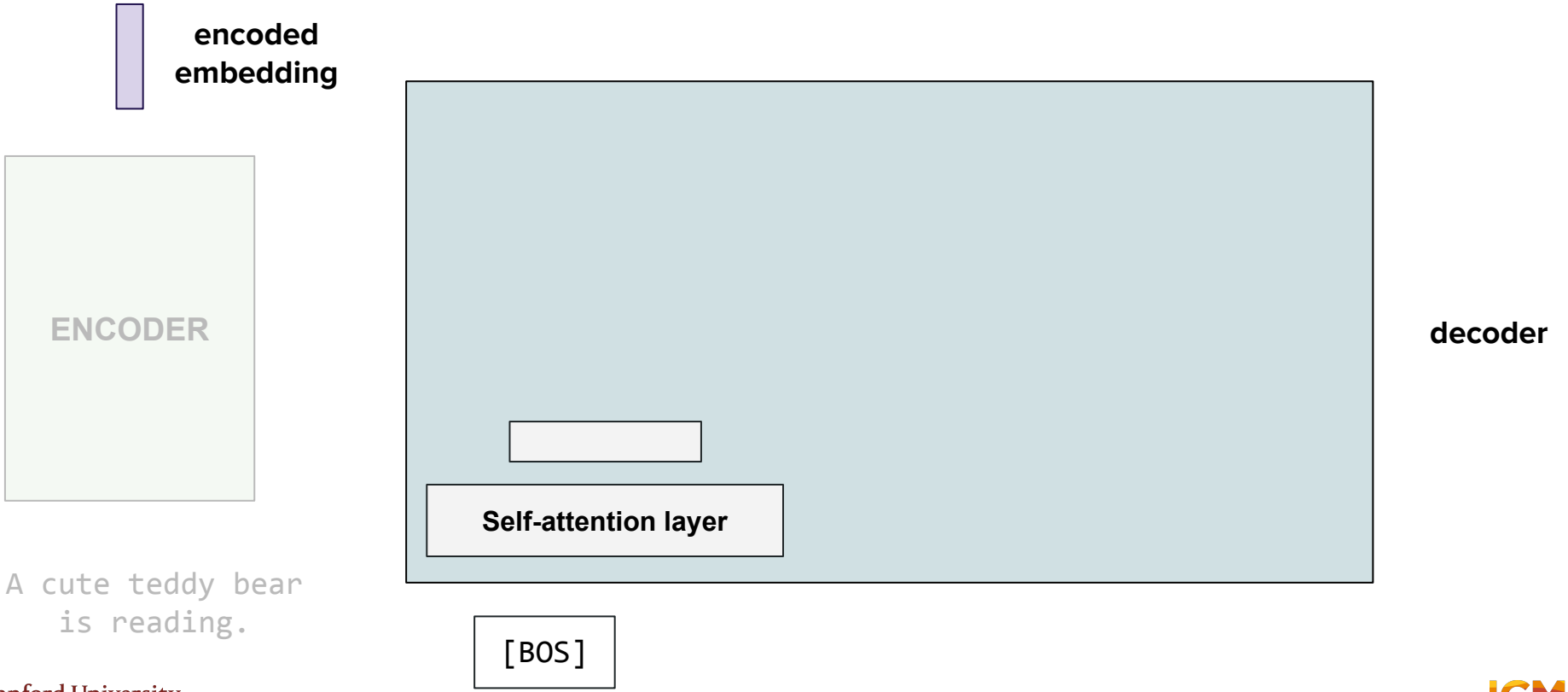
Stitching all the pieces together with an example



Stitching all the pieces together with an example



Stitching all the pieces together with an example



Stitching all the pieces together with an example

encoded embedding

ENCODER

A cute teddy bear is reading.

Encoder - Decoder attention layer

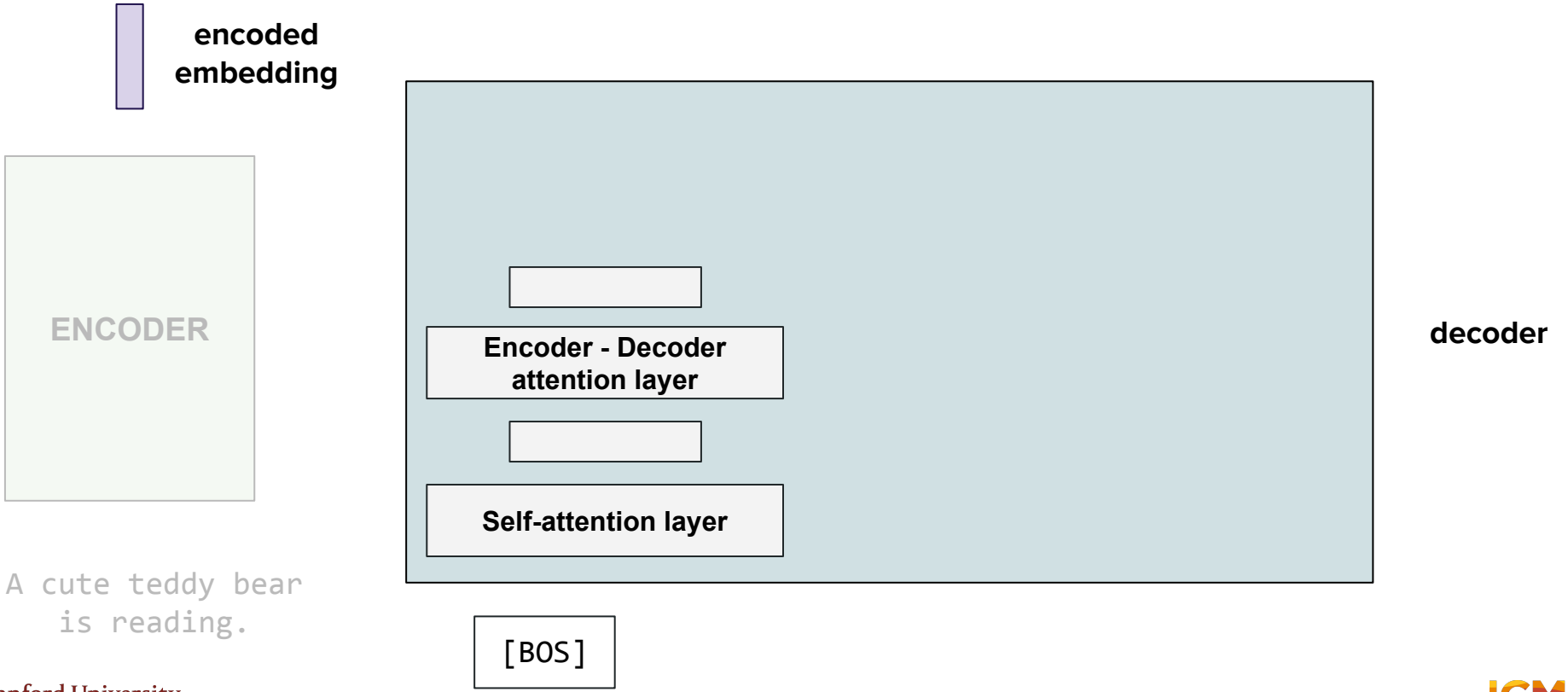
Cross attention layer (takes input as Q, K, V)

Self-attention layer

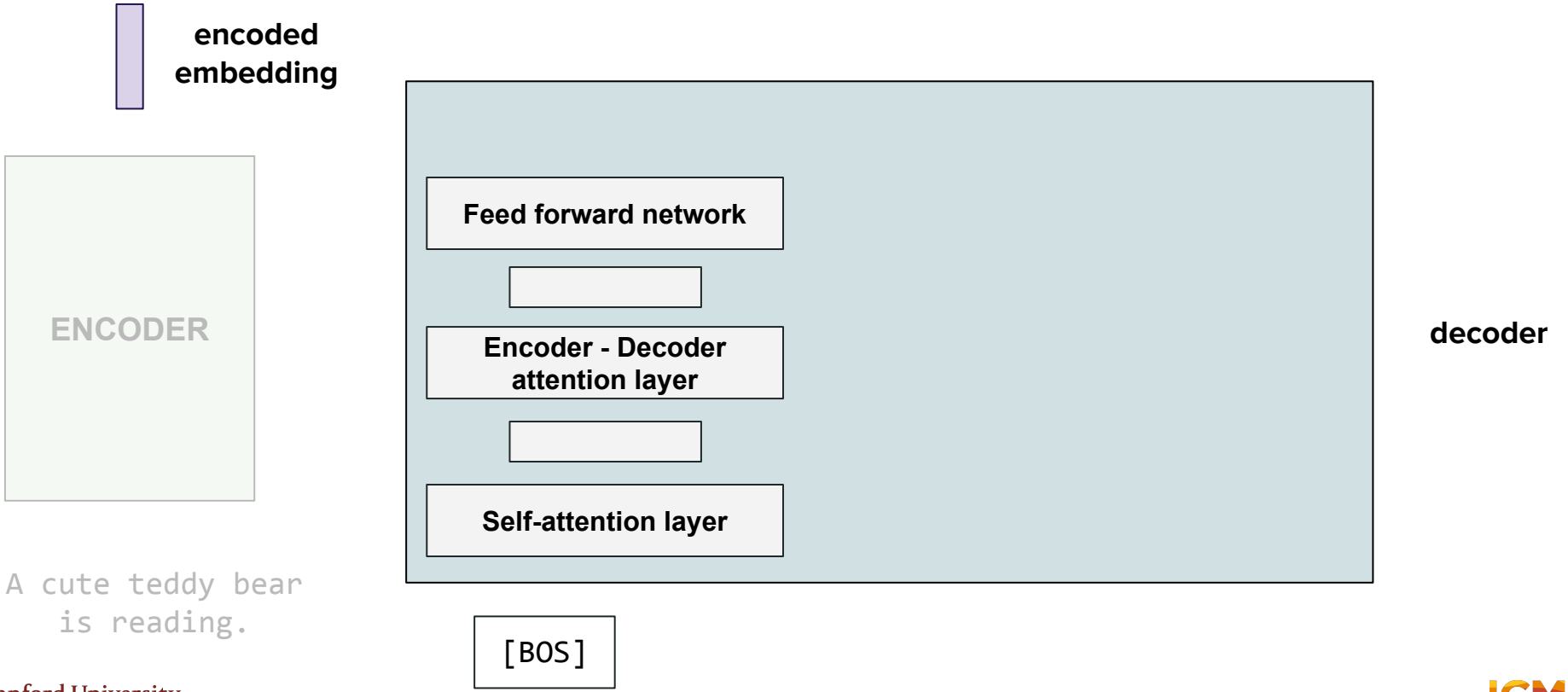
[BOS]

decoder

Stitching all the pieces together with an example



Stitching all the pieces together with an example



Stitching all the pieces together with an example

encoded embedding

ENCODER

A cute teddy bear is reading.

Softmax layer

Feed forward network

Encoder - Decoder attention layer

Self-attention layer

decoder

[BOS]

Stitching all the pieces together with an example

encoded embedding

ENCODER

A cute teddy bear is reading.

[0.001, 0.0003, ..., 0.4, ..., 0.002]

(Prediction of probability distribution over the vocabulary based on input Embeddings given.

Softmax layer

Feed forward network

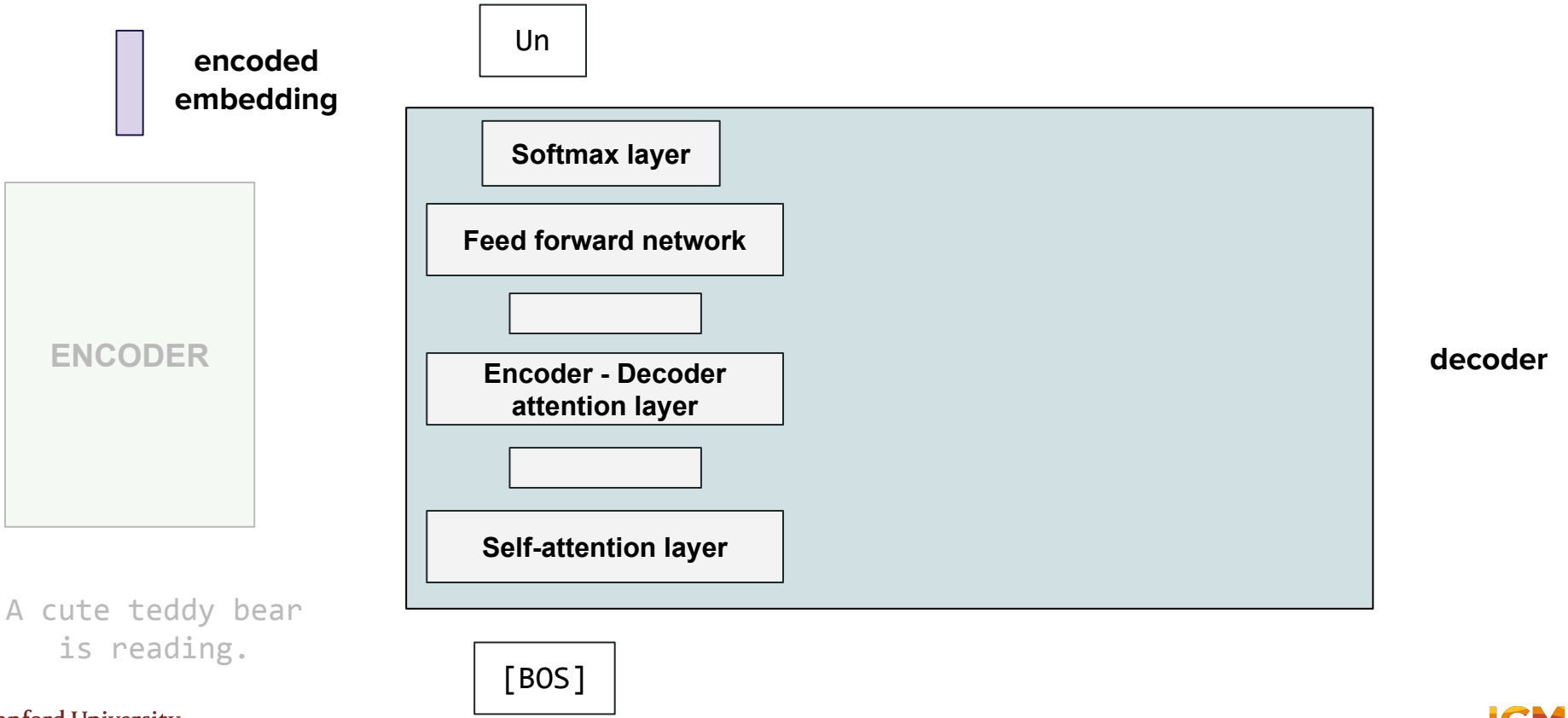
Encoder - Decoder attention layer

Self-attention layer

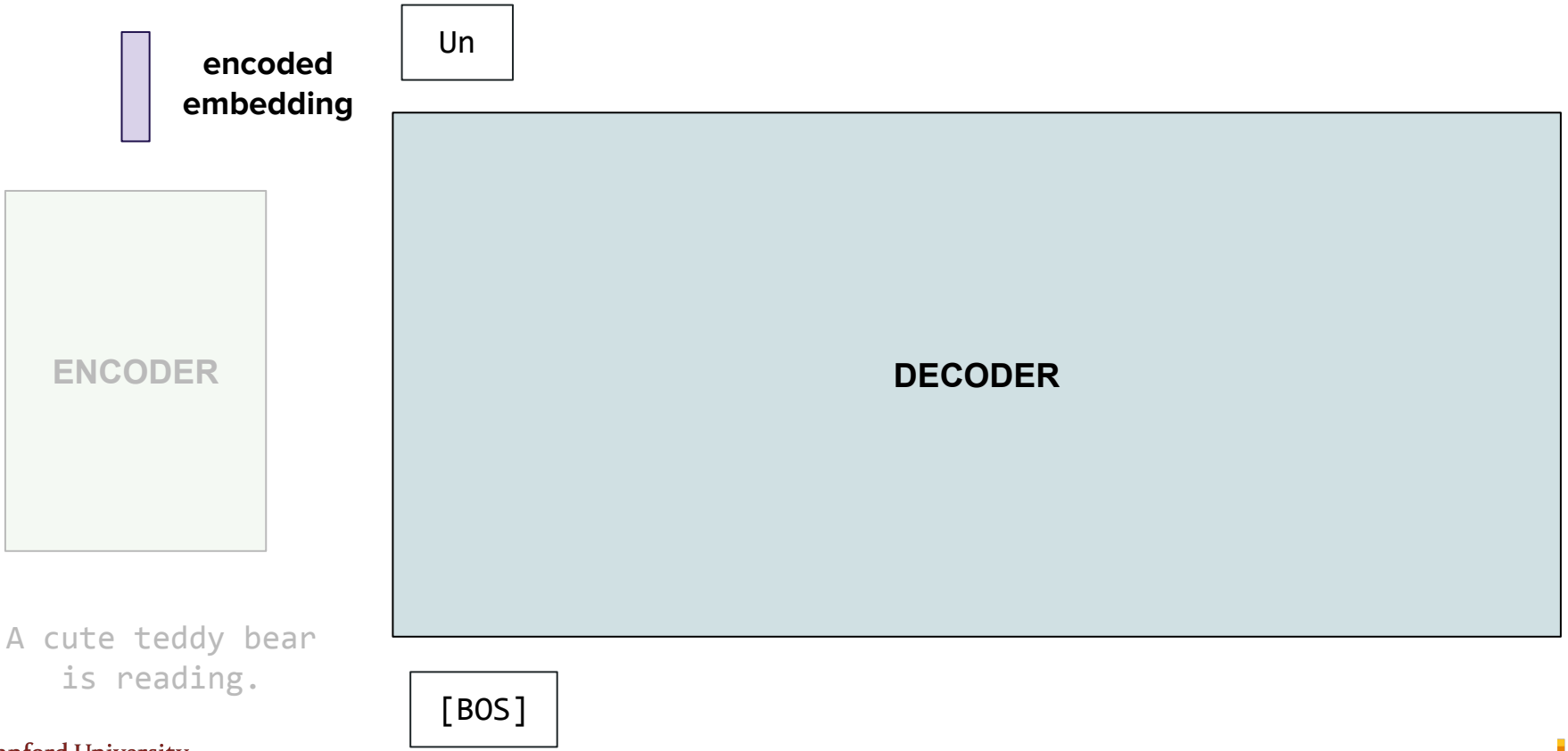
decoder

[BOS]

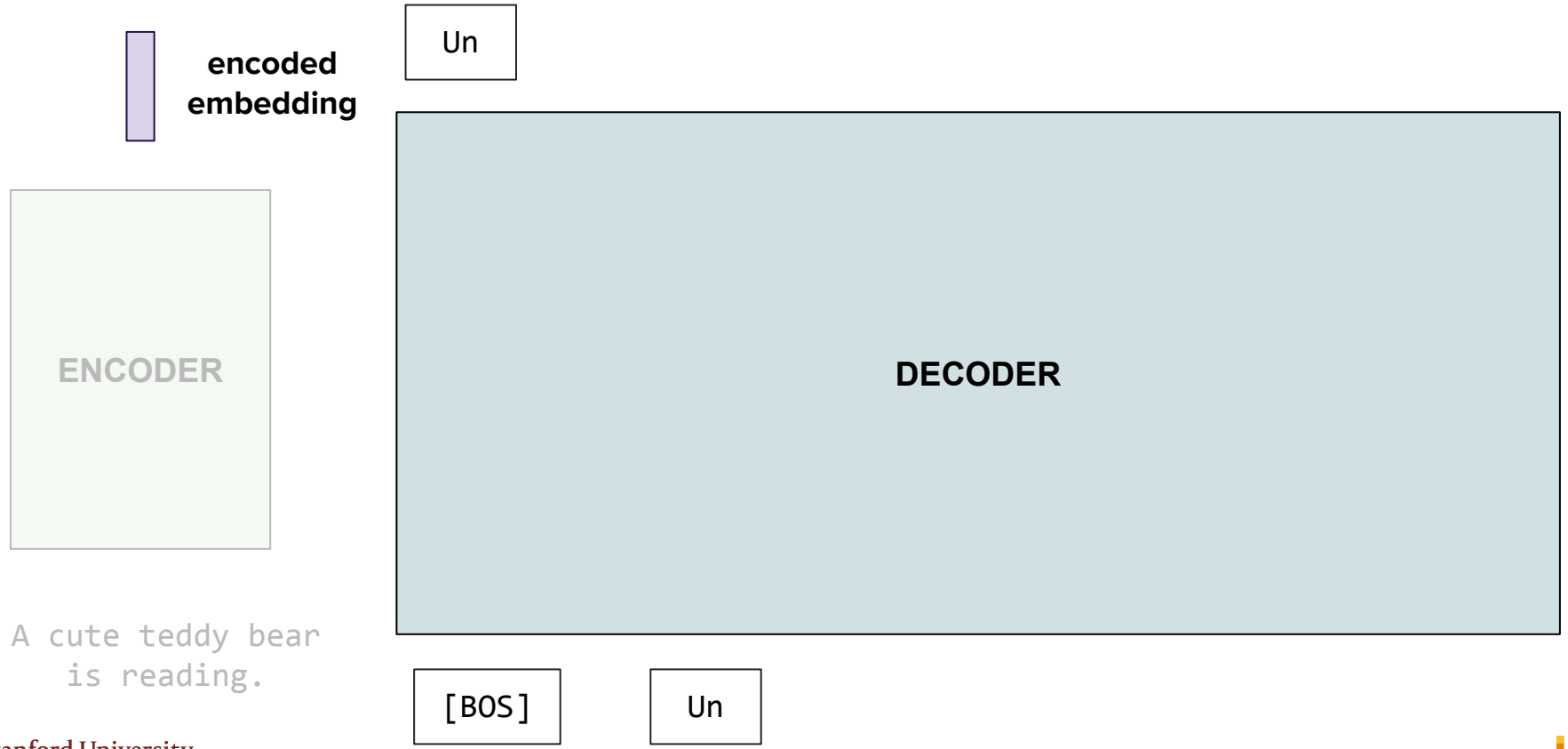
Stitching all the pieces together with an example



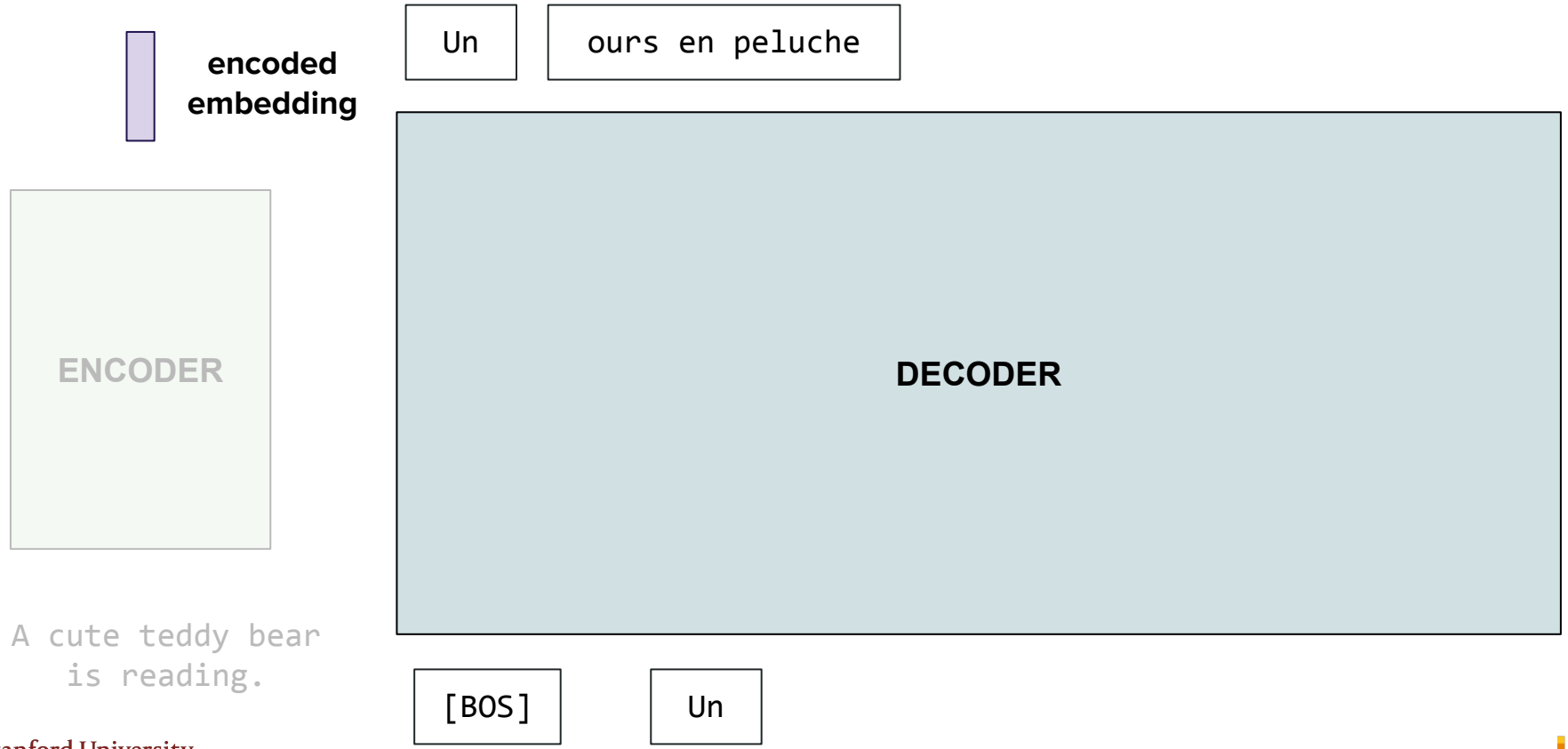
Stitching all the pieces together with an example



Stitching all the pieces together with an example



Stitching all the pieces together with an example



Stitching all the pieces together with an example

machine translation summary.

encoded embedding

ENCODER

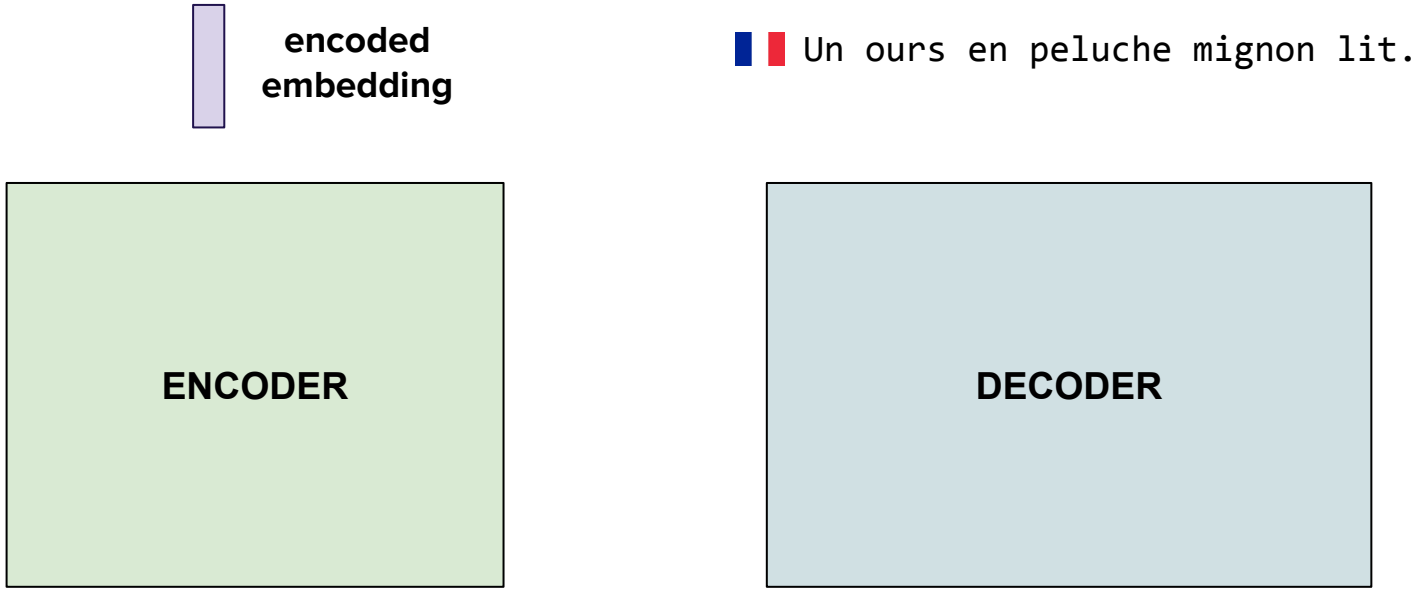
A cute teddy bear is reading.

DECODER

Un | ours en peluche | mignon | lit | [EOS]

[BOS] | Un | ours en peluche | mignon | lit

Stitching all the pieces together with an example



Thank you for your attention!