EEP 596: LLMs: From Transformers to GPT || Lecture 8

Dr. Karthik Mohan

Univ. of Washington, Seattle

February ~~4~~ 3, 2025

Deep Learning References

Deep Learning

Great reference for the theory and fundamentals of deep learning: Book by Goodfellow and Bengio et al Bengio et al

Deep Learning History

Embeddings

Two-Tower

SBERT and its usefulness SBert Details Instacart Search Relevance Instacart Auto-Complete

Attention

Illustration of attention mechanism

HOUSE KEEPING

1. MP1 – TASKS – Has updates
(Part 2) ↳ Rahul
↳ Announcement on Discord

2. Schedule of Assignments

Bonus Coding [TICE] assignments

Dates	
Feb 9	MP1 is due
Feb 7	MP2 (Part 1)
Feb 21	MP3
Feb 15	Conceptual Assignment (multiple choice)
Mar 17, 18	Final Presentation ↳ Your demos should work

Last Lecture

- BERT and Transformers Architecture
- Coding Exercise

Today's Lecture

- Multi-Head Attention
- SBERT
- Application of Embeddings to Auto-complete and Search Relevance

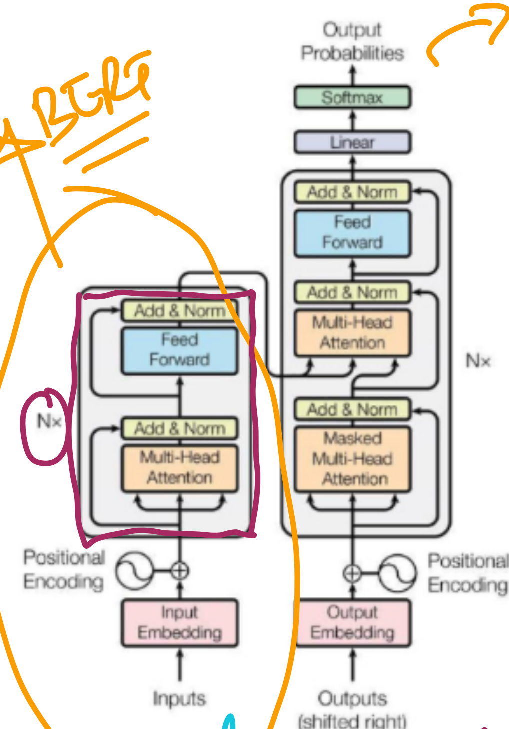
Math / Detail

Understanding Encoder/BERT at high-level

BERT

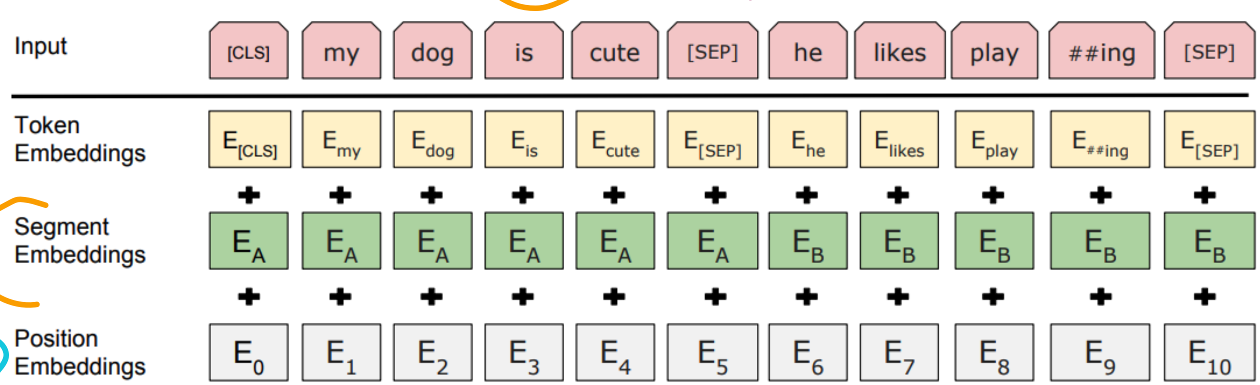
GPT

Transformer



Encoder – Decoder

Practically



Transformer Types in Practice

Encoder Only

- BERT, SBERT, ViTransformer, etc → Image Embedding
- Uses only self-attention and FFN blocks
- Good for classification, summarization, intent detection, image embeddings, etc

Lot of use-cases in the industry

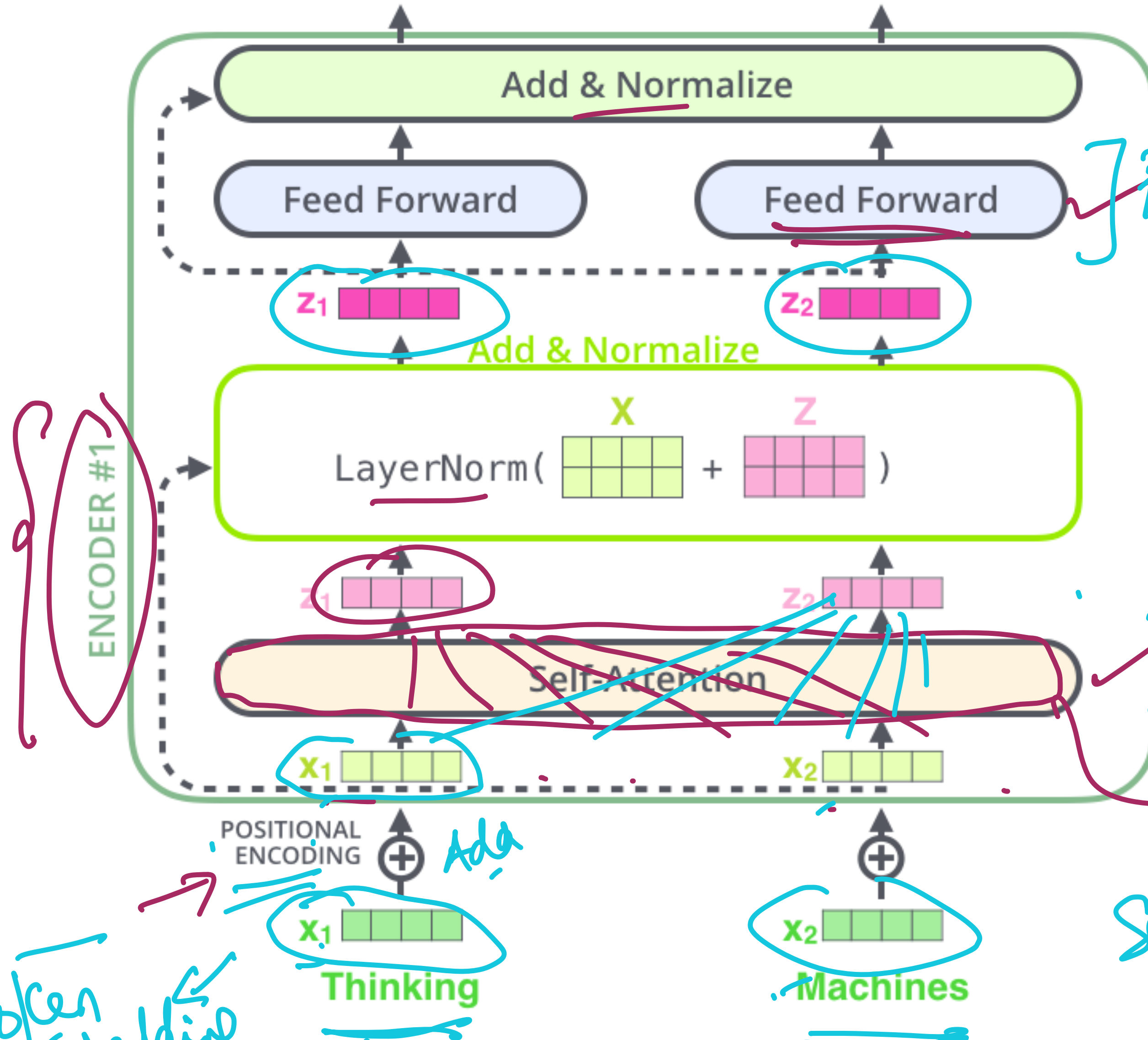
Encoder-Decoder

T5, BART. Also uses encoder-decoder attention

Decoder Only

- GPT, Llama, DeepSeek, etc
- Good for many tasks including encoder-only tasks and also generation tasks
- Uses self-attention and FFN blocks

Parsing Encoder: Multi-Head Attention and FFN



Parallel

Encoder Single Transformer Block

Sequential

Multi-Head Attention

Add

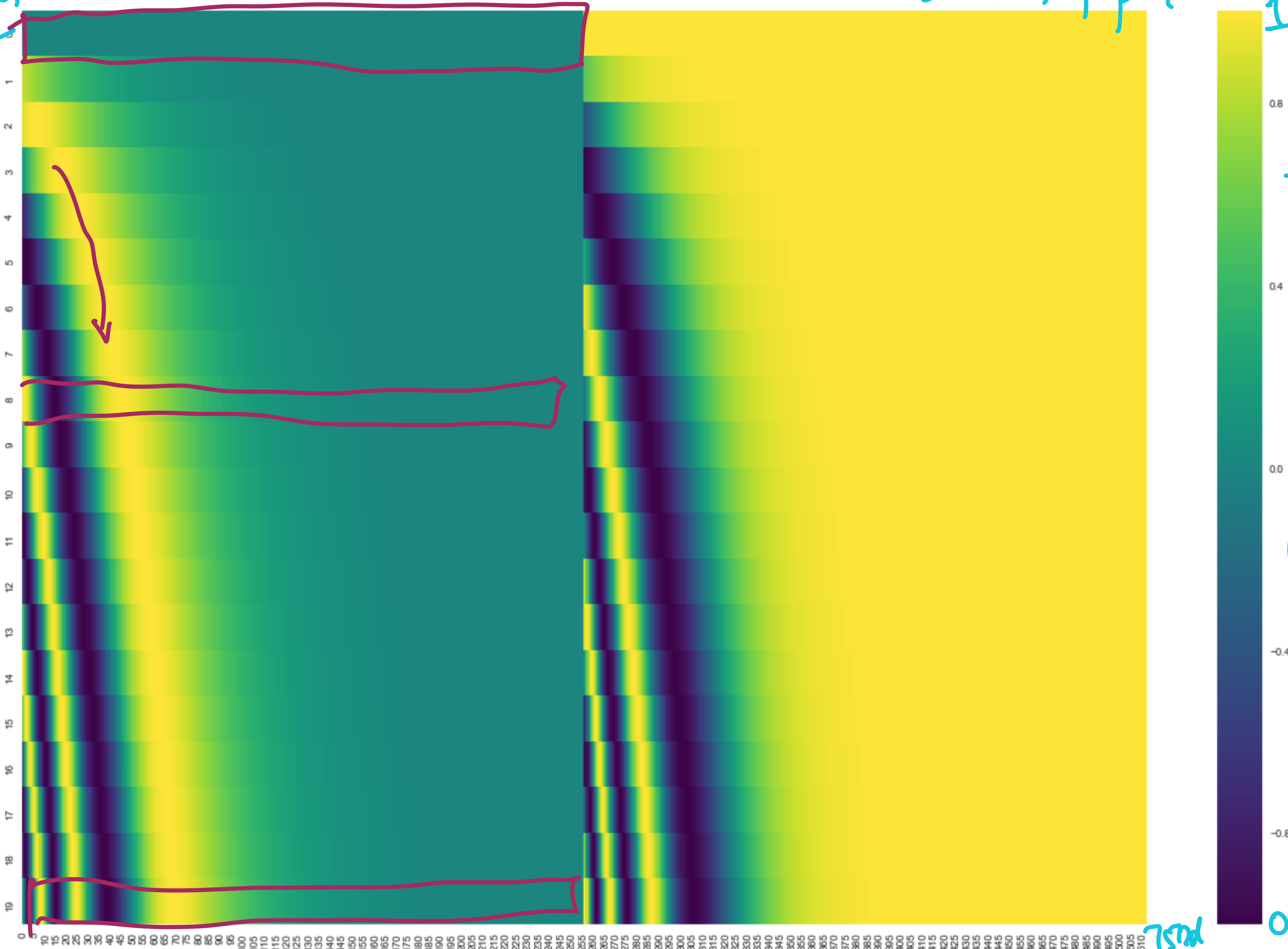
Token Embedding

Sentence "Thinking Machines"

Positional Encoding

2017 paper:- "Attention is all you need"

20 tokens



1

0

256d

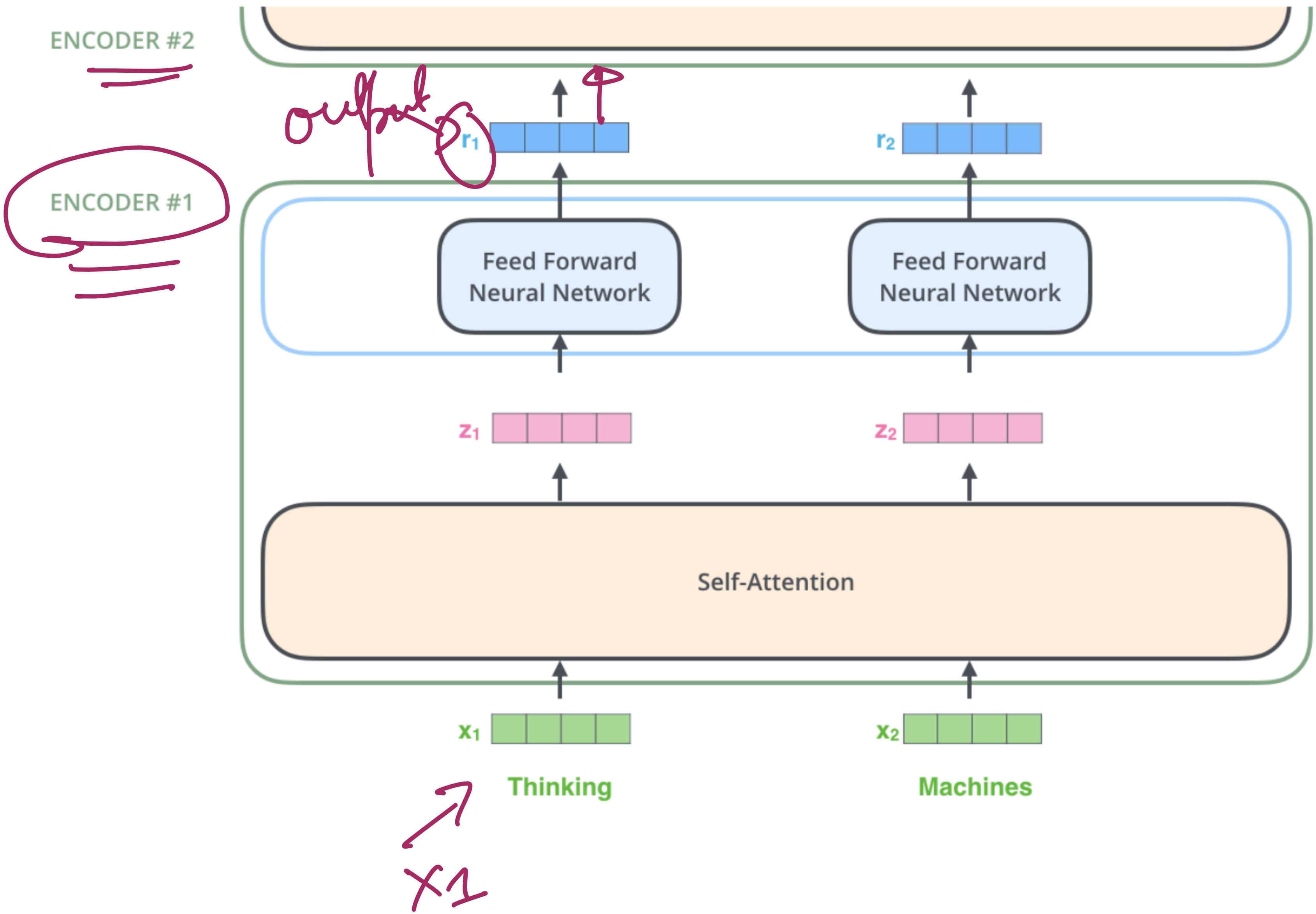
512d

Sin(i·C)

Cos(i·C)

0, 1, 2, ...

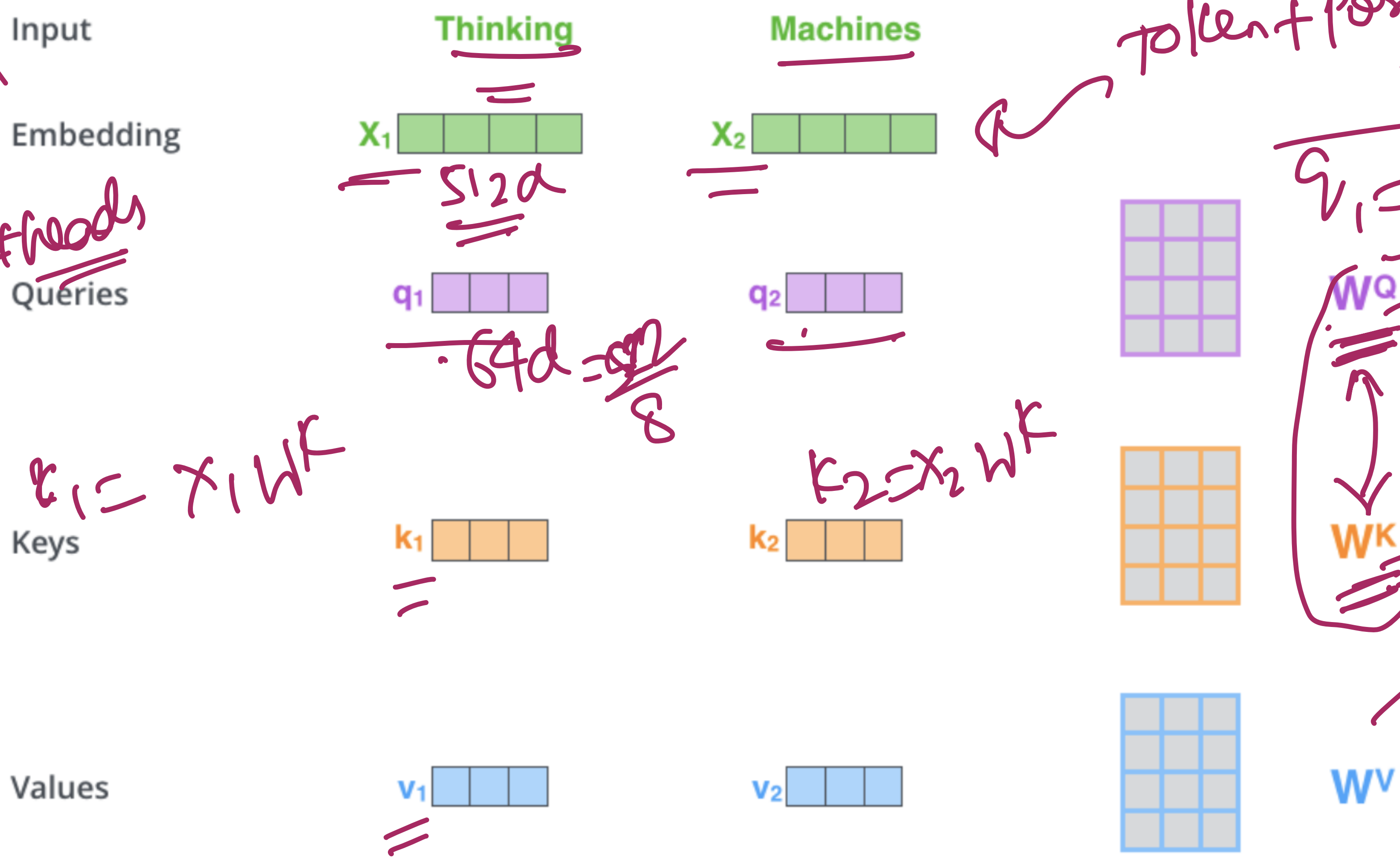
Parsing Encoder: Multi-Head Attention and FFN



Parsing Encoder: Single Head Attention

Scaled Dot-Product Attention :- Query, Key, values

Token + Positional Embedding



Single head dim

$64 \times 8 \rightarrow$ #heads $= 512$

512d

$64d = 512/8$

$q_1 = x_1 W^Q$

$q_2 = x_2 W^Q$

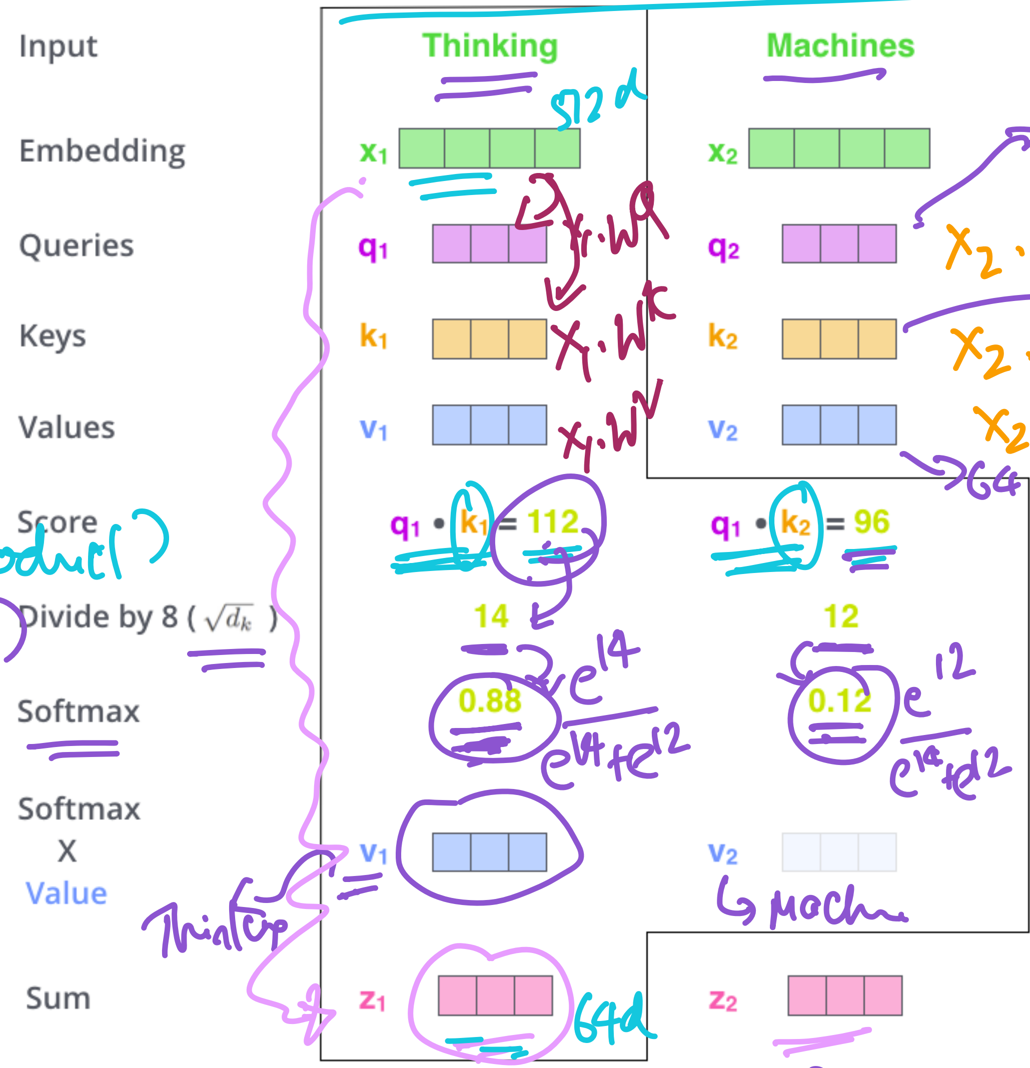
$k_1 = x_1 W^K$

$k_2 = x_2 W^K$

learned from data

Parsing Encoder: Single Head Attention

Scaled Dot-Product Attention



(Dot-Product)
(Scaled)

Machine

$\text{Sfm}\left(\frac{q_2 \cdot k_1}{8}, \frac{q_2 \cdot k_2}{8}\right) = [0.2 \quad 0.8]$

$z_2 = 0.2 v_1 + 0.8 v_2$

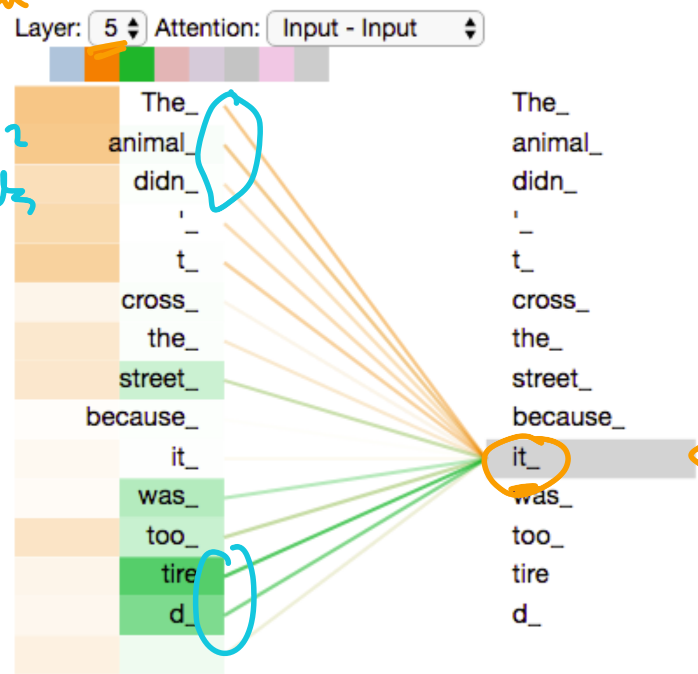
[112 96] 1x2

Thinking
Machine
k_1
k_2
Thinking
Machine

Result(Thinking) = 0.88 value(Thinking) + 0.12 value(Machine)
$= 0.88 v_1 + 0.12 v_2 = z_1$

Parsing Encoder: Single Head Attention

Sentence = "The animal didn't cross the street because it was too tired" → 11 words



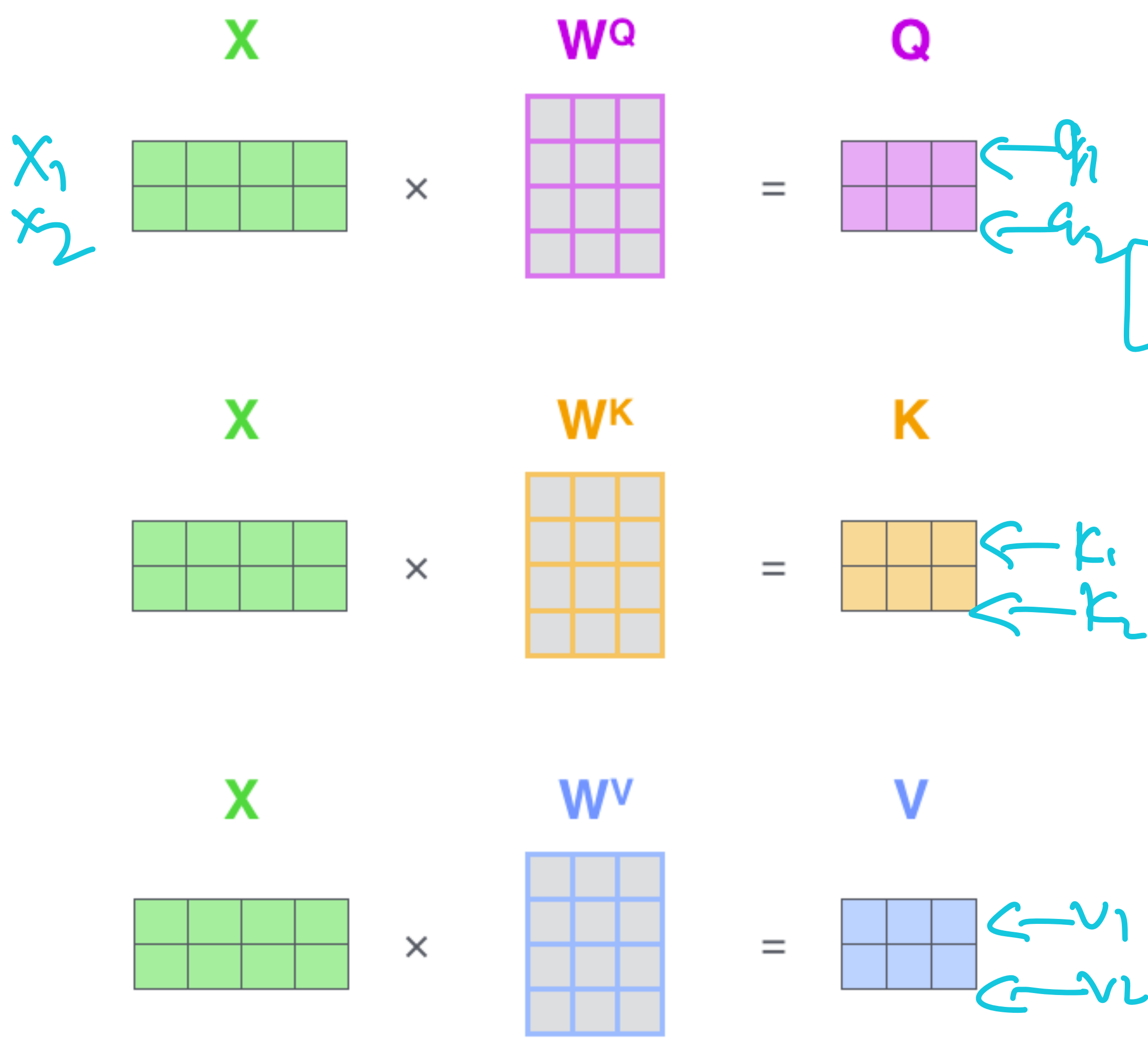
v1
v2
v3

Token for a sentence

14 tokens for 11 words

As we encode the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired" -- in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired".

Parsing Encoder: Single Head Attention



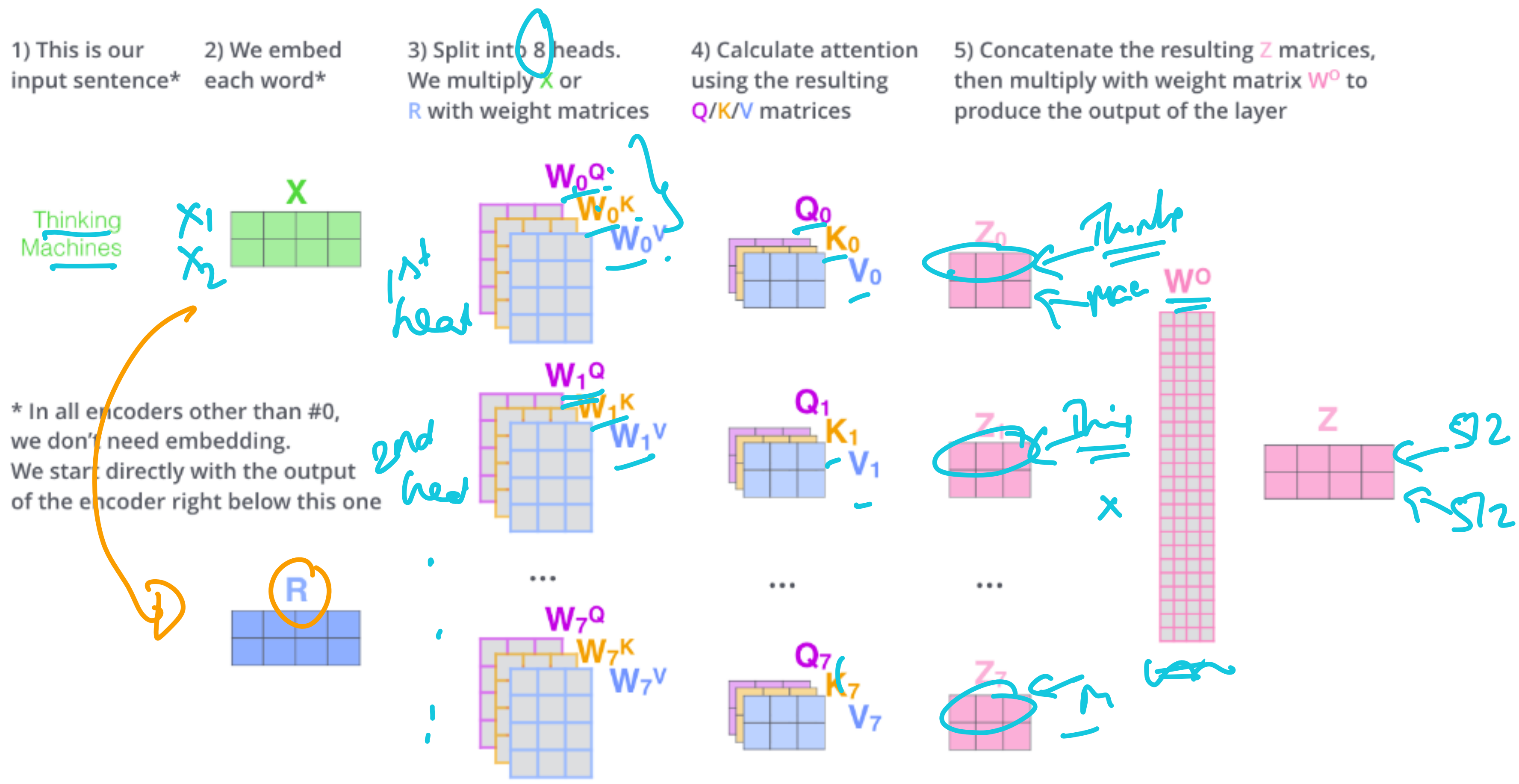
$x_1 W^Q = q_1$

$x_2 W^Q = q_2$

$\underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{X} W^Q = \underbrace{\begin{bmatrix} q_1 \\ q_2 \end{bmatrix}}_{Q}$

Every row in the X matrix corresponds to a word in the input sentence. We again see the difference in size of the embedding vector (512, or 4 boxes in the figure), and the q/k/v vectors (64, or 3 boxes in the figure)

Parsing Encoder: Multi-Head Attention and FFN

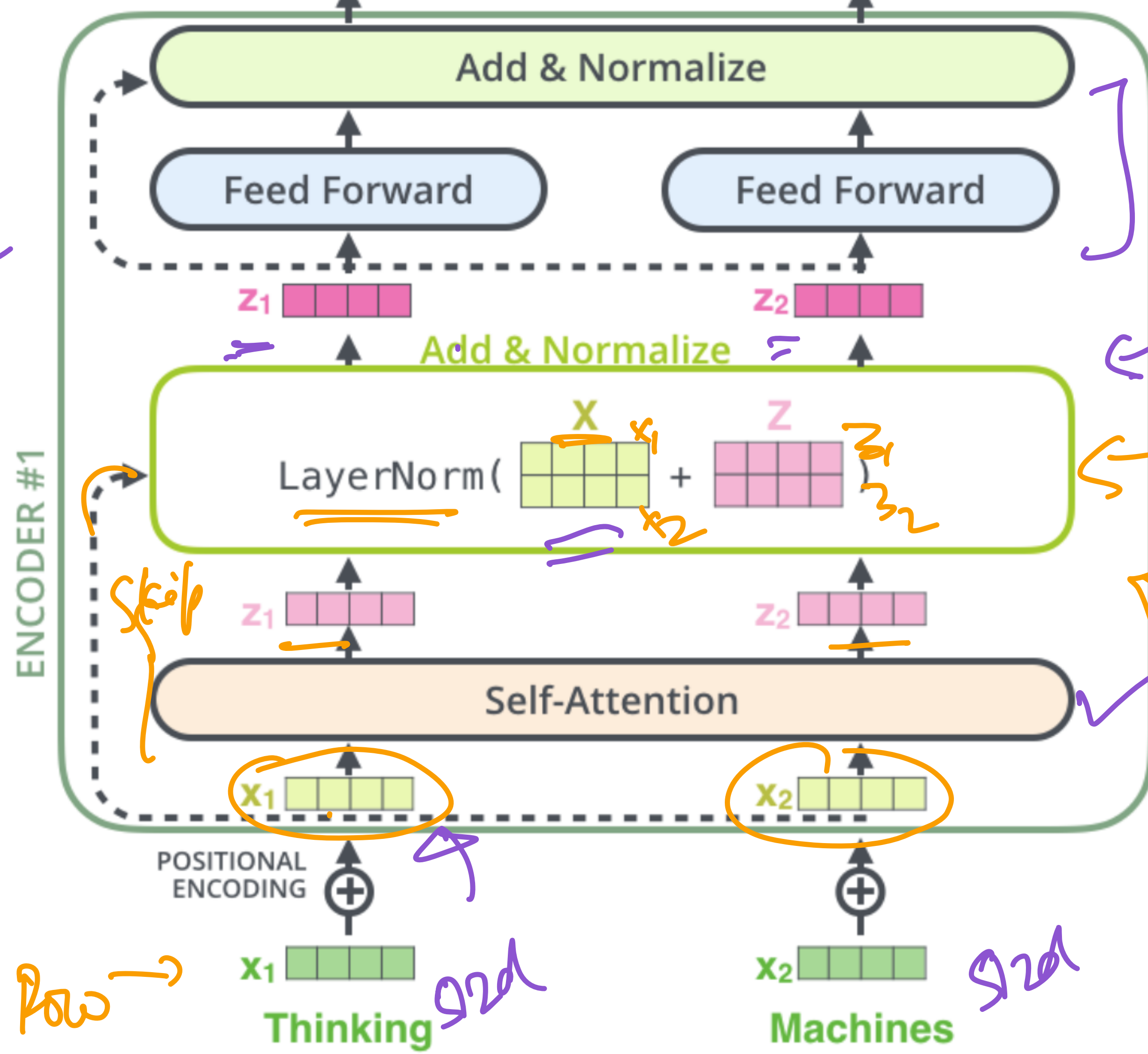


Layer Normalization

New x_1 512d

New x_2 512d

New x_3 512d . . .



Skip

Row →

512d

512d

RNN/LSTM

Stabilizing the training of transformers

Sensitivity issues to starting point

Self-Attention Math Walk-through

x_1 $\qquad$ x_2 $\qquad$ $x_3 \to 512d$

$x_3 W^q \to 64d \to$ vector×Matrix multiple

$x_3 W^k$

$x_3 W^v$

Query $\quad q_1 = x_1 W^q \qquad x_2 W^q$

Key $\quad k_1 = x_1 W^k \qquad x_2 W^k$

value $\quad v_1 = x_1 W^v \qquad x_2 W^v$

$$\frac{q_1 \cdot k_1}{\sqrt{64} = 8} \qquad \frac{q_1 \cdot k_2}{8} \qquad \frac{q_1 \cdot k_3}{8} \quad \ldots \to T \times 64$$

$Q = X W^q$

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_T \end{bmatrix}_{T \times 512} \qquad W^q_{512 \times 64}$$

$Q_{T\times 64} \quad K_{T\times 64} \quad V_{T\times 64}$

$q\, K^T{}_{1\times T} \qquad q_{1\times 64}\, K^T{}_{64\times T}$

D

$$\left\{ \begin{array}{l} \underline{\underline{q_1}} K^T \quad 1\times T \\ \underline{q_2} K^T \\ \underline{q_3} K^T \end{array} \right.$$

$[4\ 4\ \ 4\ 4\ 4\ 4\ 4\ 4]\ 1\times T$

Attention for token 1 from all other tokens

$$\begin{bmatrix} \underline{q_1} \\ \hline \underline{q_2} \\ \vdots \\ \underline{q_T} \end{bmatrix} \underline{\underline{K}}^T_{64\times T}$$

$Q_{T\times 64}$

$QK^T_{\underline{\underline{T\times T}}}$ ← self-Attention Matrix

↓ scale

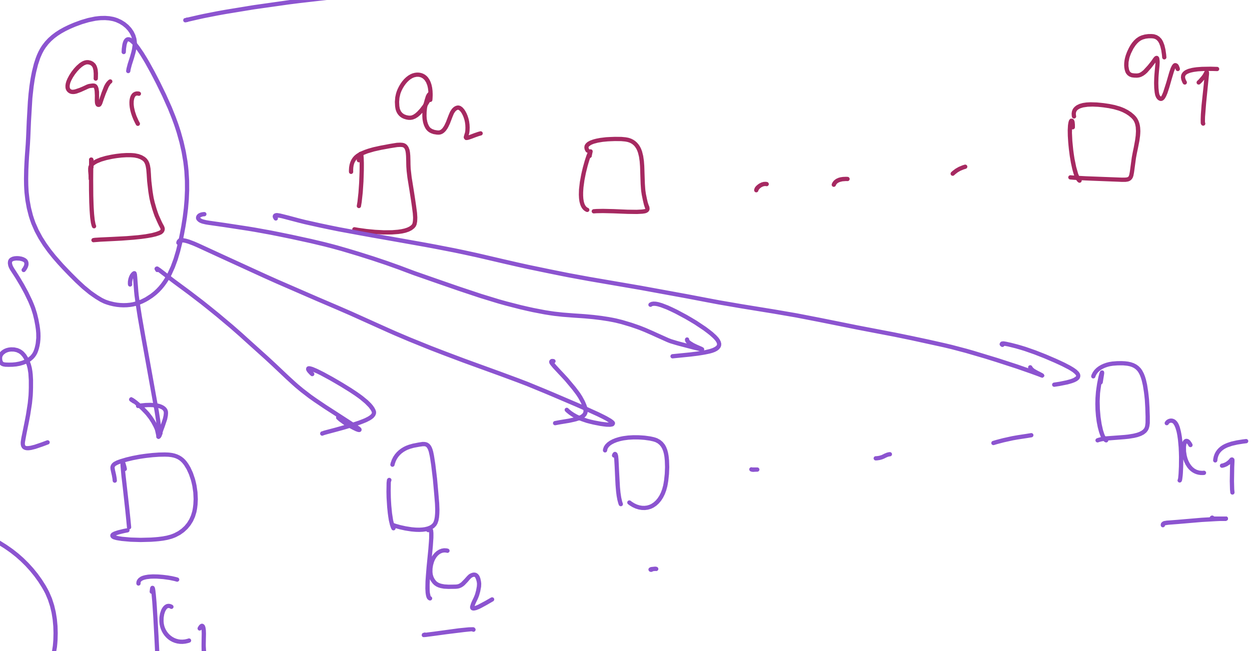
$$\text{softMax}\left(\frac{QK^T}{8}\right) \longrightarrow \text{scaled Dot-product Attention weight}$$

$$q_1 K^T \begin{bmatrix} V_1 \\ V_2 \\ \vdots \\ V_T \end{bmatrix} = (\underline{\underline{q_1}} K^T)_1 \underline{\underline{V_1}} + (\underline{q_1} K^T)_2 \underline{V_2} + \cdots + (\underline{q_1} K^T)_T V_T$$

$$= q_1 K^T V_{T\times 64}$$

Token 1:- $\text{softmax}\left(\frac{q_1 K^T}{8}\right) V \rightarrow z_1$

$$\underbrace{\text{softmax}\left(\frac{Q K^T}{8}\right) V = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_T \end{bmatrix}}_{\text{Scaled Dot-product Attention}} = Z$$

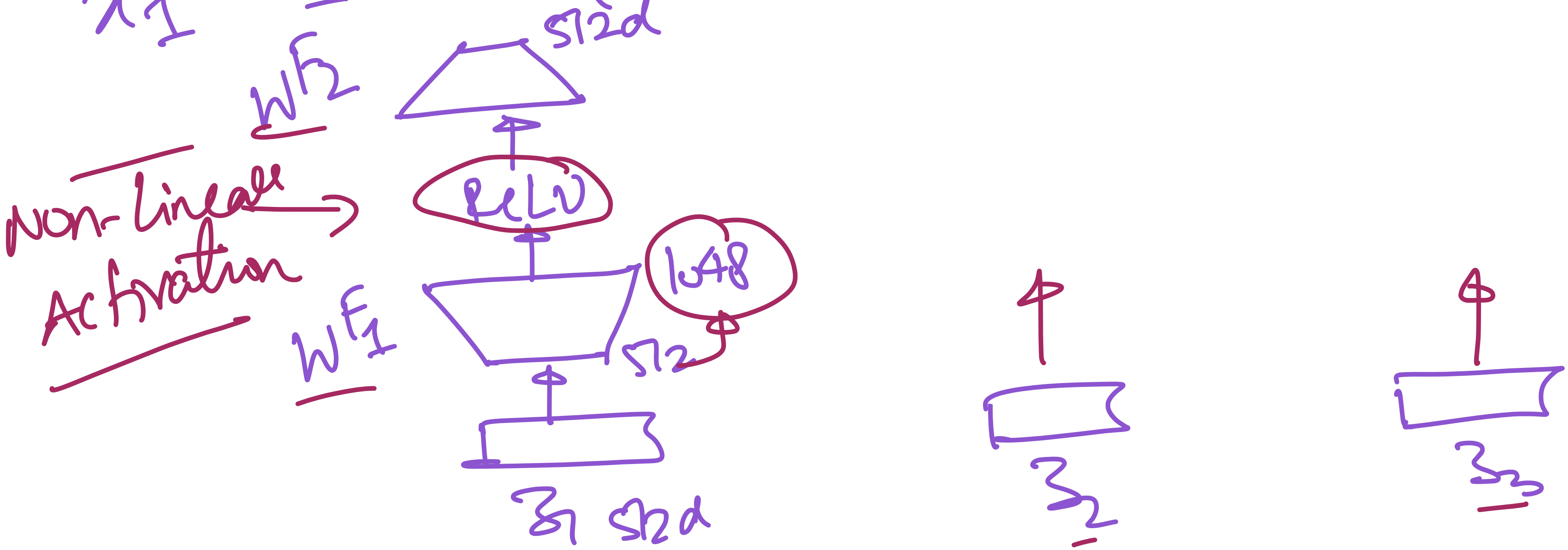


$\text{softmax}\left(\frac{q_1 K^T}{8}\right)$

FFN Math Walk-through

FFN in Encoder Transformer block

New $x_1 = \max(0, z_1 W^{F_1}) W^{F_2}$



ICE #0: Self-attention Exercise

x_1 $\quad$ x_2 $\quad$ $X_{2\times4}$ $\quad$ $W^Q_{4\times3}$ $\quad$ $W^K_{4\times3}$ $\quad$ $W^V_{4\times3}$

Let's go through a self-attention python calculation exercise to understand it better. Let $x = [[1, 2, 3, -1], [3, -4, -7, 5]]$ be the input token embeddings. In the first layer of the encoder of the transformer, the weight matrices are given by $W^Q = [[-1, 2, 0], [2, 3, -5], [1, 0, 0], [-3, 1, 2]]$, $W^K = [[1, 2, 3], [2, 4, 3], [3, 0, 3], [-1, 5, 2]]$, $W^V = [[-1, -2, 3], [2, -4, 0], [0, 0, 1], [1, 0, -7]]$. Compute the soft-max similar to what we did in the previous walk-through. You can use python matrix multiplication (e.g. numpy) to arrive at the solution. Question is which token (token 1 or token 2) does token 2 place more attention on and what is the attention probability?

Self-Attention Matrix $= \text{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right)_{2\times2}$

BERT pre-training

Two Tasks

1. **Masked LM Model:** Mask a word in the middle of a sentence and have BERT predict the masked word
2. **Next-sentence prediction:** Predict the next sentence - Use both positive and negative labels. How are these generated?

BERT pre-training

Two Tasks

1. **Masked LM Model:** Mask a word in the middle of a sentence and have BERT predict the masked word
2. **Next-sentence prediction:** Predict the next sentence - Use both positive and negative labels. How are these generated?

ICE: Supervised or Un-supervised?

1. Are the above two tasks supervised or un-supervised?

BERT pre-training

Two Tasks

1. **Masked LM Model:** Mask a word in the middle of a sentence and have BERT predict the masked word
2. **Next-sentence prediction:** Predict the next sentence - Use both positive and negative labels. How are these generated?

ICE: Supervised or Un-supervised?

1. Are the above two tasks supervised or un-supervised?

Data set!

English Wikipedia and book corpus documents!

Loss Function for Masked Language Model (MLM)

Loss Function for MLM mimicks which type of classic ML model?

Loss Function for Masked Language Model (MLM)

Loss Function for MLM mimicks which type of classic ML model?

Cross-Entropy

$L(p, \hat{p}) = -\sum_i \left[p_i \log(\hat{p}_i) + (1 - p_i) \log(1 - \hat{p}_i)\right]$

Loss Function for Masked Language Model (MLM)

Loss Function for MLM mimicks which type of classic ML model?

Cross-Entropy

$L(p, \hat{p}) = -\sum_i \left[p_i \log(\hat{p}_i) + (1 - p_i) \log(1 - \hat{p}_i)\right]$

ICE: What is the loss function for Binary Classification?

Sentence BERT a.k.a sBERT

Uses Siamese Twins architecture

Sentence BERT a.k.a sBERT

Uses Siamese Twins architecture

Advantages of sBERT

More optimized for Sentence Similarity Search.

Sentence BERT - Siamese BERT architecture

Two networks = Siamese Twins

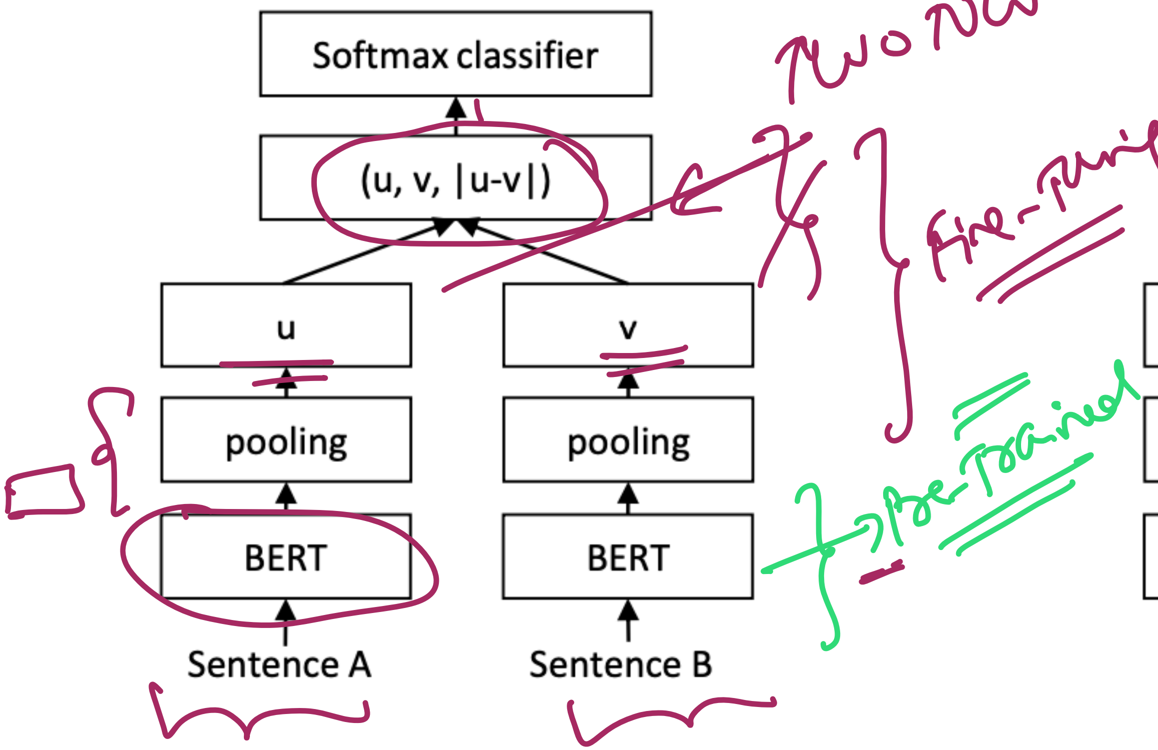


Figure 1: SBERT architecture with classification objective function, e.g., for fine-tuning on SNLI dataset. The two BERT networks have tied weights (siamese network structure).

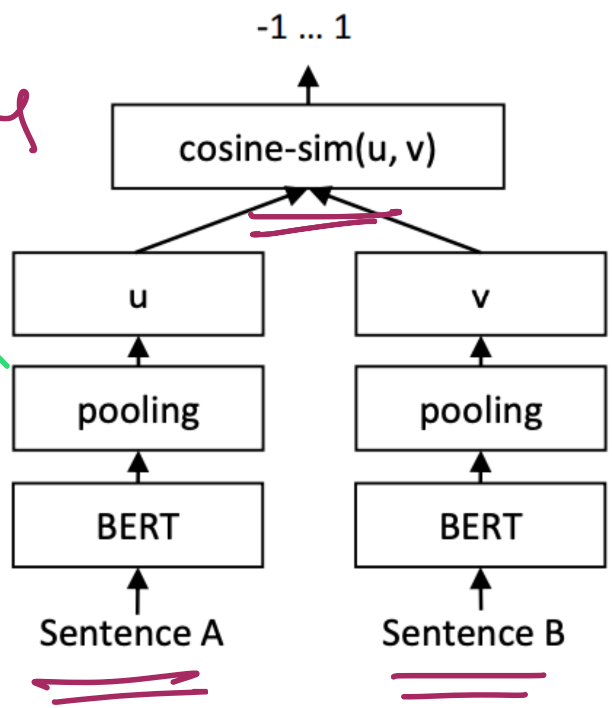


Figure 2: SBERT architecture at inference, for example, to compute similarity scores. This architecture is also used with the regression objective function.

Loss Function for SBERT

Pooling Strategy for SBERT

	NLI	STSb
Pooling Strategy		
MEAN	**80.78**	**87.44**
~~MAX~~	79.07	69.92
CLS	79.80	86.62
Concatenation		
(u, v)	66.04	-
$(\|u - v\|)$	69.78	-
$(u * v)$	70.54	-
$(\|u - v\|, u * v)$	78.37	-
$(u, v, u * v)$	77.44	-
$(u, v, \|u - v\|)$	**80.78**	-
$(u, v, \|u - v\|, u * v)$	80.44	-

Table 6: SBERT trained on NLI data with the classification objective function, on the STS benchmark (STSb) with the regression objective function. Configurations are evaluated on the development set of the STSb using cosine-similarity and Spearman's rank correlation. For the concatenation methods, we only report scores with MEAN pooling strategy.

Sentence BERT Cosine Similarity Results

Model	STS12	STS13	STS14	STS15	STS16	STSb	SICK-R	Avg.
Avg. GloVe embeddings	55.14	70.66	59.73	68.25	63.66	58.02	53.76	61.32
Avg. BERT embeddings	38.78	57.98	57.98	63.15	61.06	46.35	58.40	54.81
BERT CLS-vector	20.16	30.01	20.09	36.88	38.08	16.50	42.63	29.19
InferSent - Glove	52.86	66.75	62.15	72.77	66.87	68.03	65.65	65.01
Universal Sentence Encoder	64.49	67.80	64.61	76.83	73.18	74.92	**76.69**	71.22
SBERT-NLI-base	70.97	76.53	73.19	79.09	74.30	77.03	72.91	74.89
SBERT-NLI-large	72.27	**78.46**	**74.90**	80.99	76.25	**79.23**	73.75	76.55
SRoBERTa-NLI-base	71.54	72.49	70.80	78.74	73.69	77.77	74.46	74.21
SRoBERTa-NLI-large	**74.53**	77.00	73.18	**81.85**	**76.82**	79.10	74.29	**76.68**

Table 1: Spearman rank correlation ρ between the cosine similarity of sentence representations and the gold labels for various Textual Similarity (STS) tasks. Performance is reported by convention as $\rho \times 100$. STS12-STS16: SemEval 2012-2016, STSb: STSbenchmark, SICK-R: SICK relatedness dataset.

SentEval DataSets

- **MR**: Sentiment prediction for movie reviews snippets on a five start scale (Pang and Lee, 2005).
- **CR**: Sentiment prediction of customer product reviews (Hu and Liu, 2004).
- **SUBJ**: Subjectivity prediction of sentences from movie reviews and plot summaries (Pang and Lee, 2004).
- **MPQA**: Phrase level opinion polarity classification from newswire (Wiebe et al., 2005).
- **SST**: Stanford Sentiment Treebank with binary labels (Socher et al., 2013).
- **TREC**: Fine grained question-type classification from TREC (Li and Roth, 2002).
- **MRPC**: Microsoft Research Paraphrase Corpus from parallel news sources (Dolan et al., 2004).

Sentence BERT on SentEval Results

Model	MR	CR	SUBJ	MPQA	SST	TREC	MRPC	Avg.
Avg. GloVe embeddings	77.25	78.30	91.17	87.85	80.18	83.0	72.87	81.52
Avg. fast-text embeddings	77.96	79.23	91.68	87.81	82.15	83.6	74.49	82.42
Avg. BERT embeddings	78.66	86.25	94.37	88.66	84.40	92.8	69.45	84.94
BERT CLS-vector	78.68	84.85	94.21	88.23	84.13	91.4	71.13	84.66
InferSent - GloVe	81.57	86.54	92.50	**90.38**	84.18	88.2	75.77	85.59
Universal Sentence Encoder	80.09	85.19	93.98	86.70	86.38	**93.2**	70.14	85.10
SBERT-NLI-base	83.64	89.43	94.39	89.86	88.96	89.6	**76.00**	87.41
SBERT-NLI-large	**84.88**	**90.07**	**94.52**	90.33	**90.66**	87.4	75.94	**87.69**

Table 5: Evaluation of SBERT sentence embeddings using the SentEval toolkit. SentEval evaluates sentence embeddings on different sentence classification tasks by training a logistic regression classifier using the sentence embeddings as features. Scores are based on a 10-fold cross-validation.

ICE #1

Let's say we want to automatically convert a **Natural Language Query** to a **SQL** query. E.g. "Which quarter in the past 5 years had the most amount of sales for fashion products" to "SELECT ... FROM ... WHERE ..." What kind of deep learning architecture would support this problem?

1. SBERT
2. LSTM to LSTM sequence model
3. GPT-2 ✓ / GPT-3, .
4. Feed Forward Neural Network

Fine-Tuning Transformers for down-stream tasks

A methodology for fine-tuning transformers for classification tasks

1. **Pick Base pre-trained Architecture:** Pick a base pre-trained architecture as a starting point for your fine-tuning. Example: `bert-base-uncased` is one such pre-trained model that can be loaded through `Hugging Face Transformers Library`

Fine-Tuning Transformers for down-stream tasks

A methodology for fine-tuning transformers for classification tasks

1. **Pick Base pre-trained Architecture:** Pick a base pre-trained architecture as a starting point for your fine-tuning. Example: `bert-base-uncased` is one such pre-trained model that can be loaded through `Hugging Face Transformers Library`
2. **Extract output from pre-training:** How do you want to use the output from pre-training going into *fine-tuning*? a) Extract embedding from the first token, `CLS` b) Average embeddings of all tokens as a starting point (mean pooling).

Fine-Tuning Transformers for down-stream tasks

A methodology for fine-tuning transformers for classification tasks

1. **Pick Base pre-trained Architecture:** Pick a base pre-trained architecture as a starting point for your fine-tuning. Example: `bert-base-uncased` is one such pre-trained model that can be loaded through `Hugging Face Transformers Library`
2. **Extract output from pre-training:** How do you want to use the output from pre-training going into *fine-tuning*? a) Extract embedding from the first token, `CLS` b) Average embeddings of all tokens as a starting point (mean pooling).
3. **Add fine-tuning layers:** Add fine-tuning layers on top of the pre-trained layers. Example, starting with the pooled embeddings, construct one or more dense layers (`Feed-Forward NN style`) to extract finer representations of the input. Add the output layer and its activation (typically softmax for classification tasks).

Fine-Tuning Transformers for down-stream tasks

A methodology for fine-tuning transformers for classification tasks

1. **Pick Base pre-trained Architecture:** Pick a base pre-trained architecture as a starting point for your fine-tuning. Example: `bert-base-uncased` is one such pre-trained model that can be loaded through `Hugging Face Transformers Library`
2. **Extract output from pre-training:** How do you want to use the output from pre-training going into *fine-tuning*? a) Extract embedding from the first token, `CLS` b) Average embeddings of all tokens as a starting point (mean pooling).
3. **Add fine-tuning layers:** Add fine-tuning layers on top of the pre-trained layers. Example, starting with the pooled embeddings, construct one or more dense layers (`Feed-Forward NN style`) to extract finer representations of the input. Add the output layer and its activation (typically softmax for classification tasks).
4. **Set training schedule, hyper-parameters, etc:** Set up optimizer (e.g. ADAM), hyper-parameters, training schedule, etc for training.

ICE #2

BERT Embeddings and Emotion Detection

Let's say you want to do emotion detection by fine-tuning BERT (Encoding Transformer) on a data set. One of the outputs of the *BERT pre-trained model* for a given input is the *last-hidden-state*. This includes an embedding for every token that was passed into BERT.

Let's say you are going to start with the last hidden layer and use that as input for your *fine-tuned* model. This ICE is about the dimensionality of the inputs and outputs. Let's say you have sentences of the kind: "I am looking forward to today! It's going to be a big day" This sentence conveys excitement. There are 13 words in this input and using *word-piece tokenization*, you arrive at 20 sub-tokens as input into the BERT model. The last hidden layer includes an embedding for every single token. Let's say the embedding dimension for a token is 768.

ICE #2 continued

BERT Embeddings and Emotion Detection

There are 13 words in this input and using *word-piece tokenization*, you arrive at 20 sub-tokens as input into the BERT model. The last hidden layer includes an embedding for every single token. Let's say the embedding dimension for a token is 768. For the purpose of emotion detection - You can either use the *CLS* token (Start token) embedding (also called the pooled embedding) or you can take the average of the embeddings of the tokens in the last hidden layer of BERT. a) What's the dimension of the pooled BERT embedding of this particular input example b) What's the dimension of the CLS/Start token embedding in this example? c) what's the total dimension of the last hidden layer?

1. 768, 15630 and 768
2. 768, 768 and 768
3. 15360, 768 and 15630
4. 768, 768 and 15360

ICE #3

Why does pooling of the output need to be done for sequence classification (e.g. emotion detection)?

1. Reduces the dimensionality
2. Averages context from all the tokens
3. Computational concerns for training the fine-tuned model
4. All of the above

Application of SBERT Embeddings to Instacart Recommendations

Instacart Recommendations

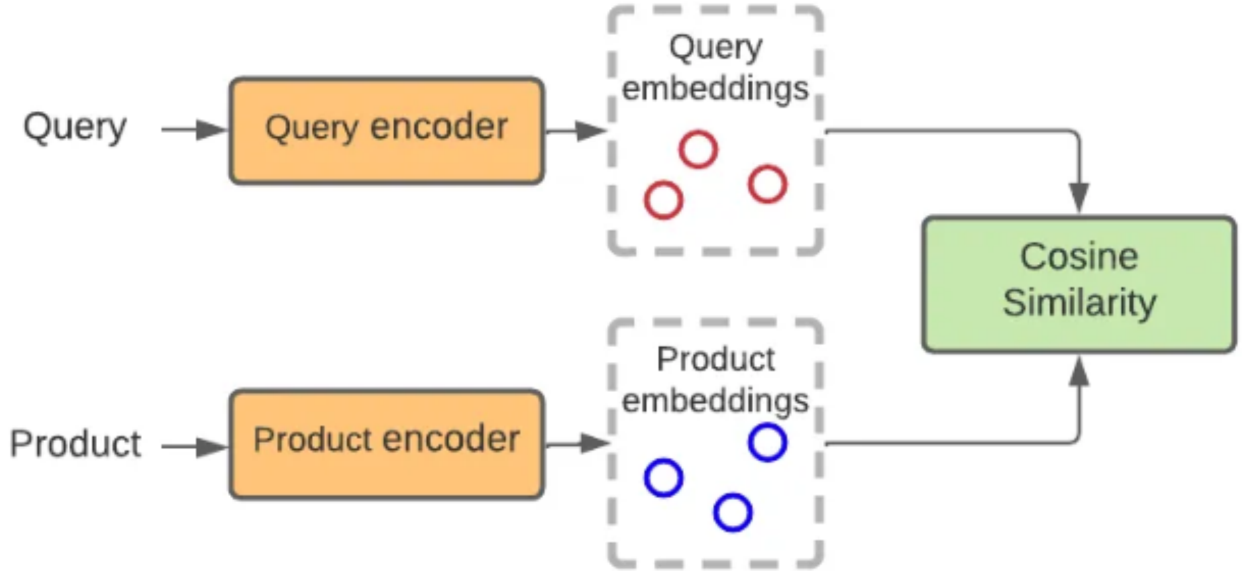
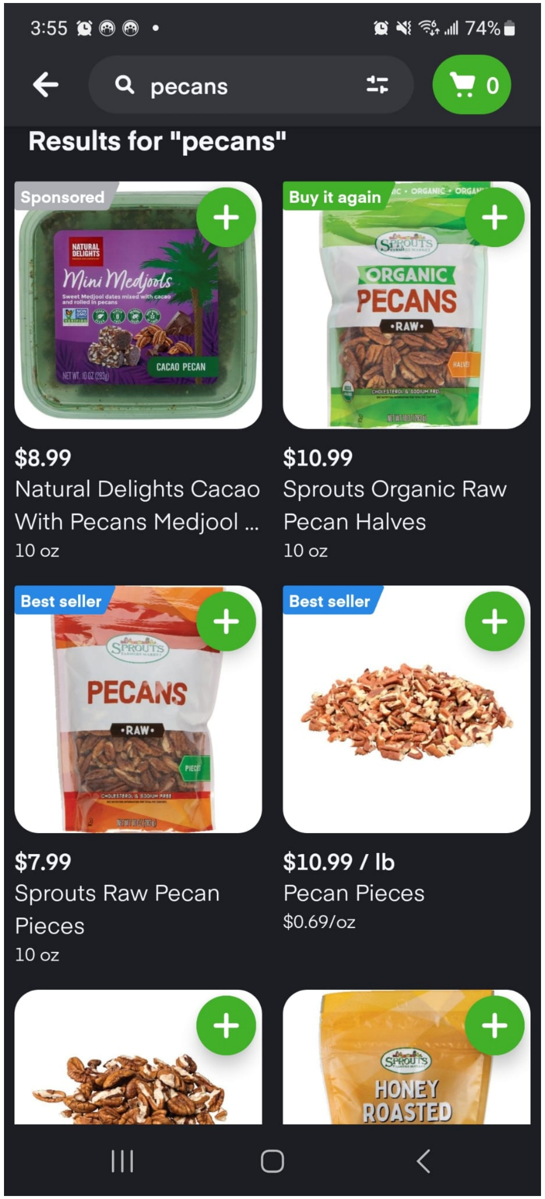


Figure 1. Conceptual diagram of a two-tower model

Positive Examples



High-quality Positive Examples

Converted Products for Search Query “Orange”
Navel Oranges
Clementines
Mandarins
...
Bananas
...
Strawberries

Negative Examples

Figure 3. (Left) In the vanilla implementation of in-batch negative, all off-diagonal negative samples are given the same weight. (Right) In our implementation with self-adversarial re-weighting, harder examples are given more weight (darker color), making the task more challenging for the model.

Model Training Architecture

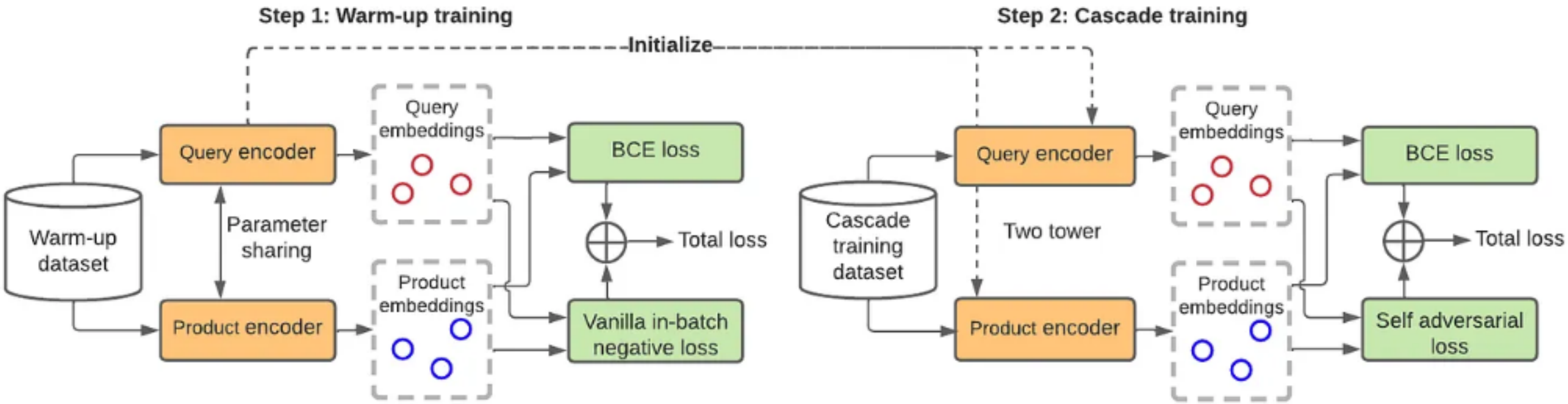


Figure 4. Two-step cascade training for ITEMS.

System Design

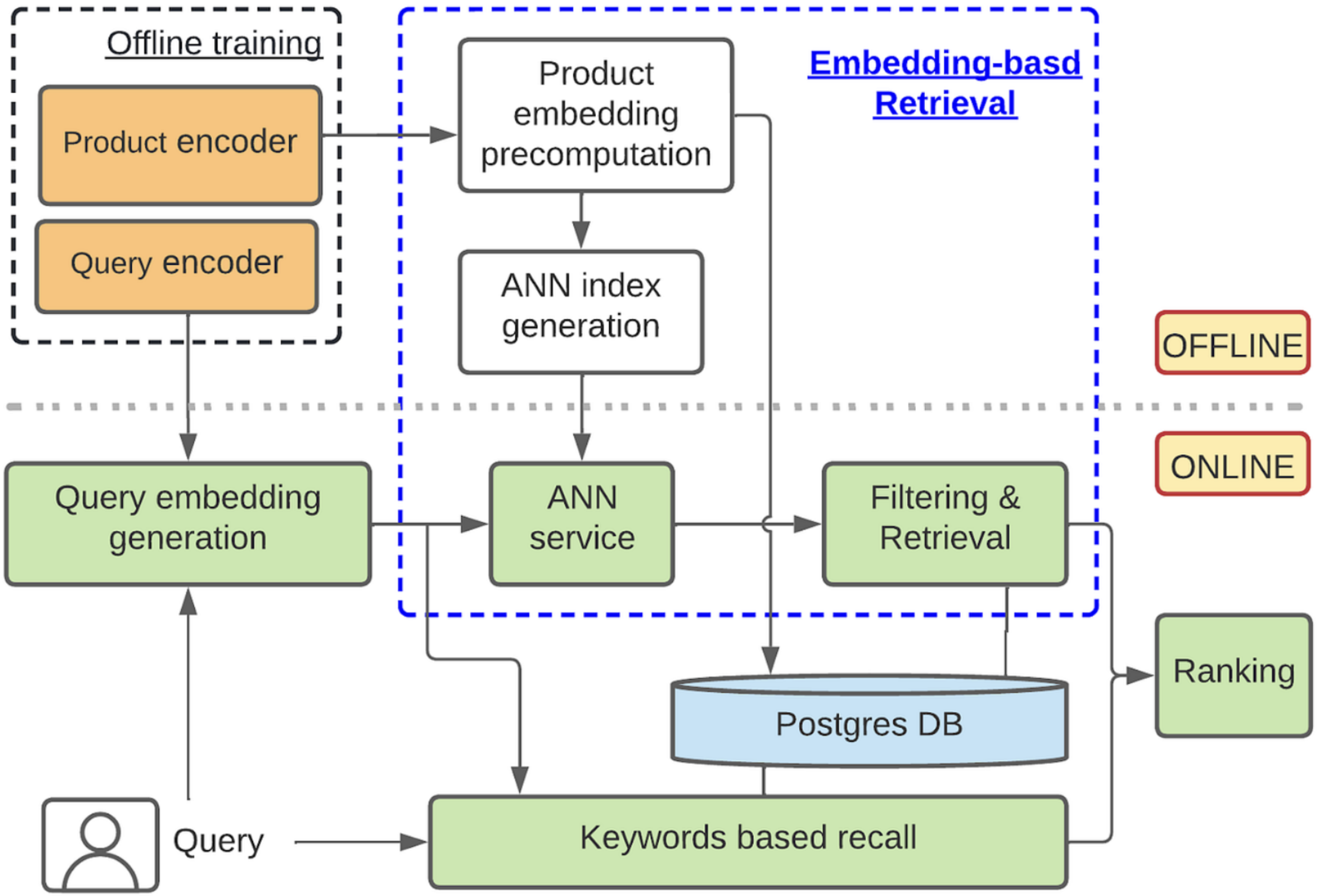


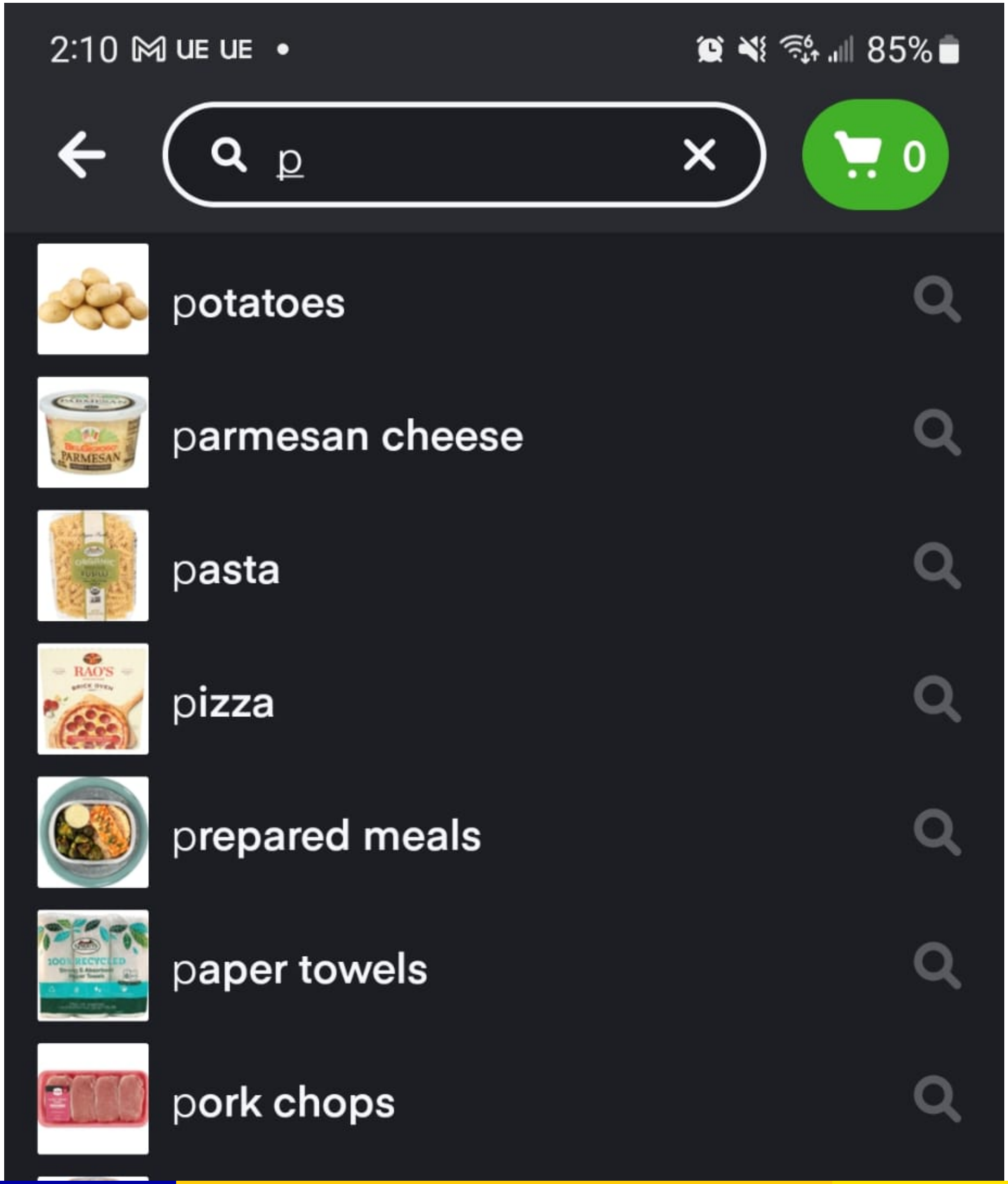
Figure 7. ITEMS system architecture.

Breakouts Time #1

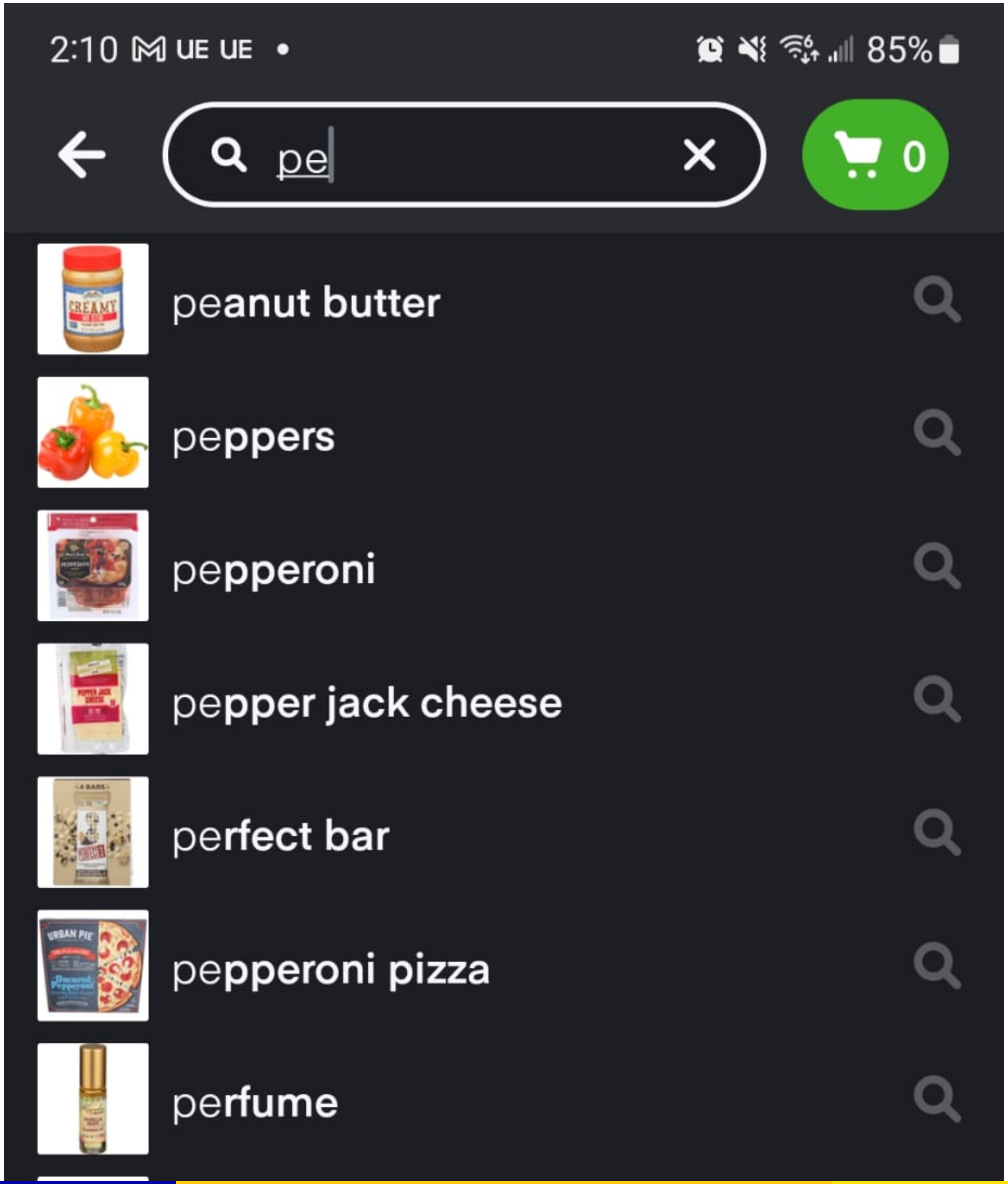
Auto-complete — 5 mins

Let's say you are tasked with building an in-email auto-completion application, which can help complete partial sentences into full sentences through suggestions (auto-complete). How would you use what we have learned so far to model this? What architecture would you use? What would be your data? And what are some pitfalls or painpoints your model should address?

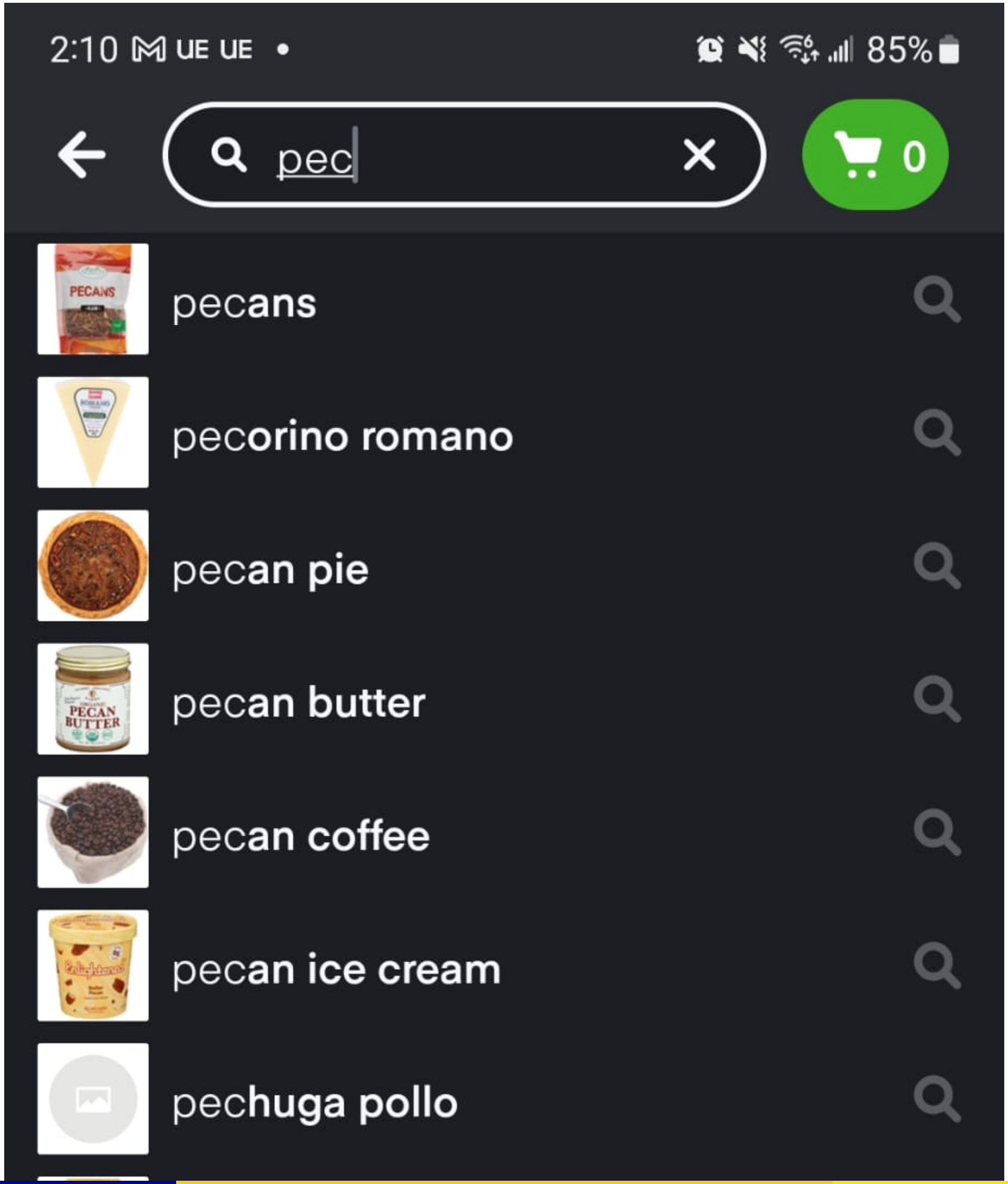
Instacart Auto-Complete and Search Relevance



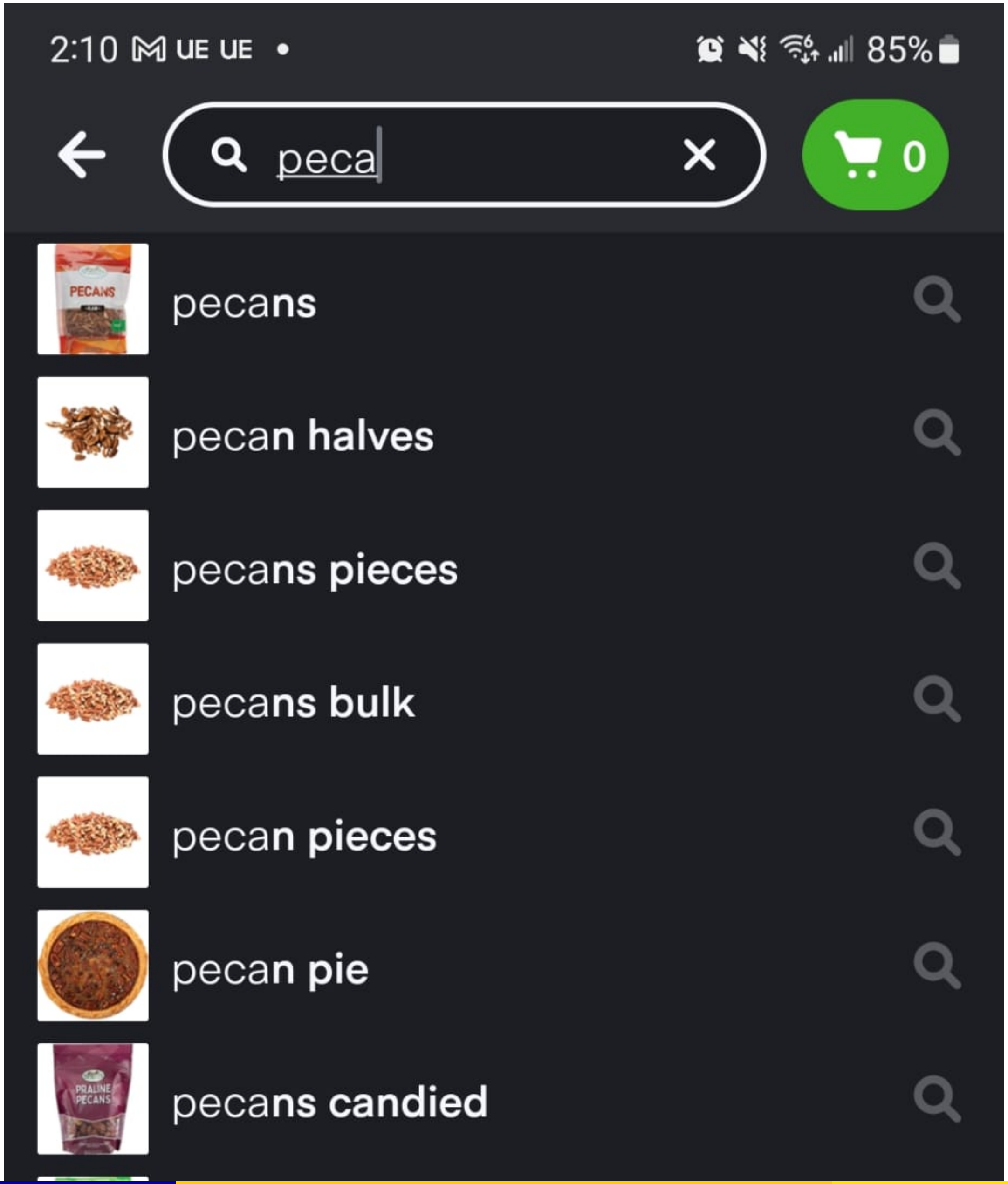
Instacart Auto-Complete



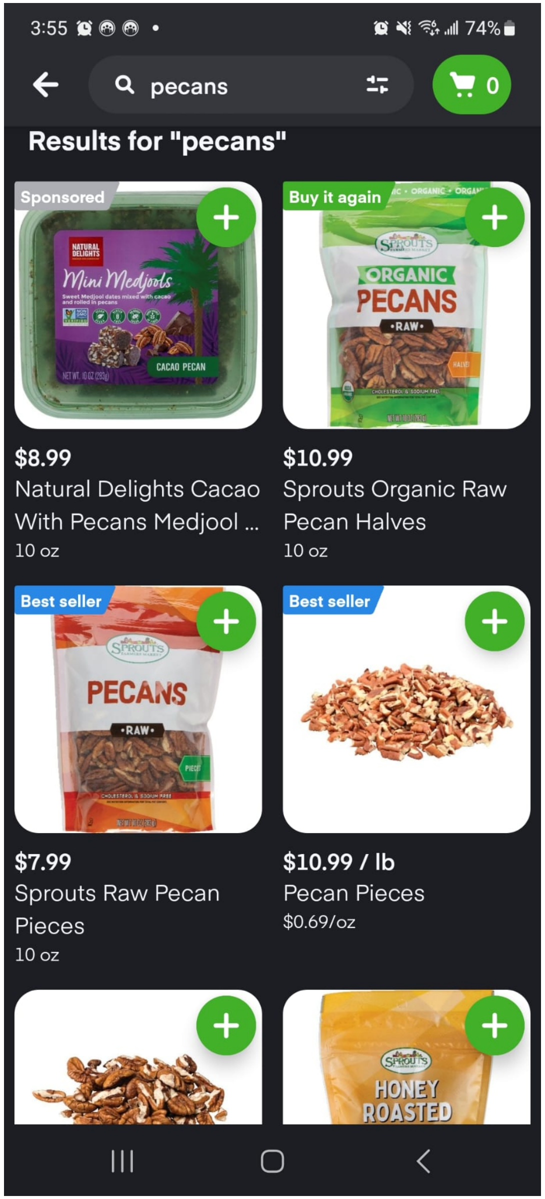
Instacart Auto-Complete



Instacart Auto-Complete



Instacart Auto-Complete and Search Results



Instacart Diversifying Auto-Complete

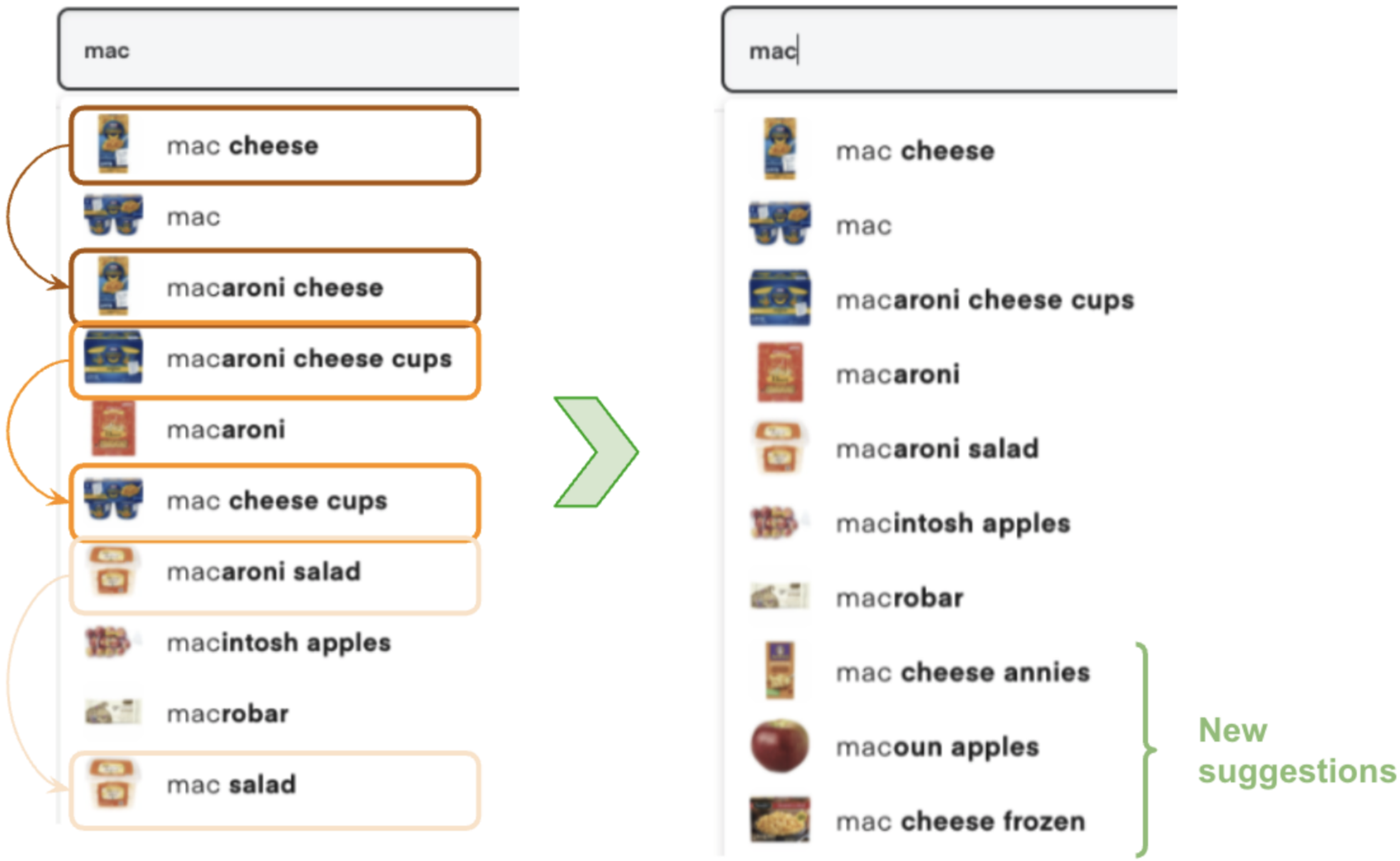


Figure 9. Autocomplete when a customer searches for "mac", before (left) and after (right) semantic deduplication.