EEP 596: LLMs: From Transformers to GPT || Lecture 7

Dr. Karthik Mohan

Univ. of Washington, Seattle

January 25, 2024

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

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

Attention

Illustration of attention mechanism

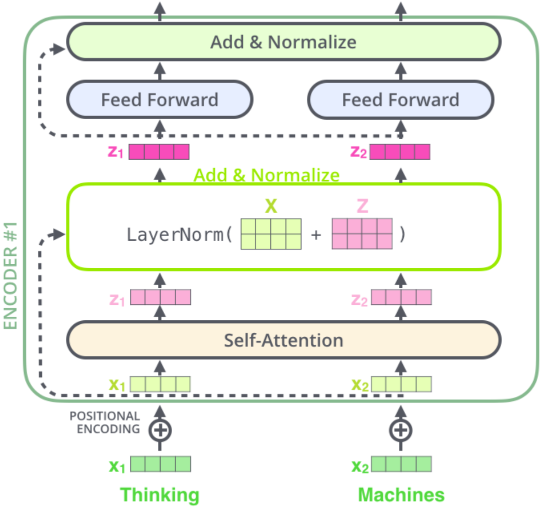
Previous Lecture

- Sentence Transformers and BERT
- Loss functions for BERT
- Multi-Head Attention
- SBERT

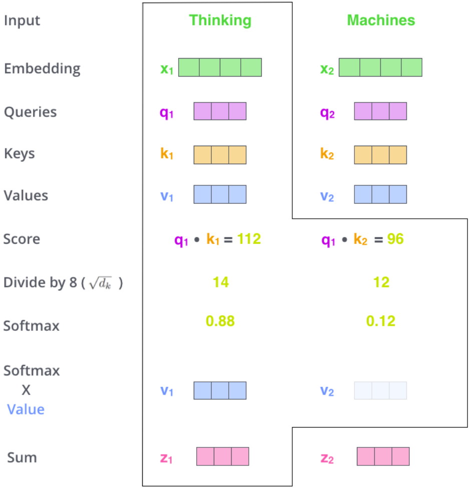
Today's Lecture

- Multi-Head Attention
- Triplet Loss vs Classification Loss
- Fine-tuning BERT and SBERT
- Application of Embeddings to Autocomplete and Search Relevance

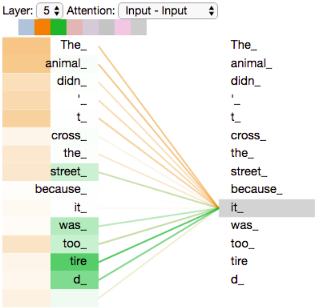
Parsing Encoder: Multi-Head Attention and FFN



Parsing Encoder: Multi-Head Attention and FFN

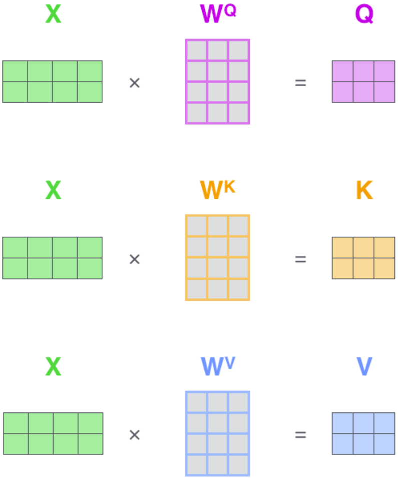


Parsing Encoder: Multi-Head Attention and FFN



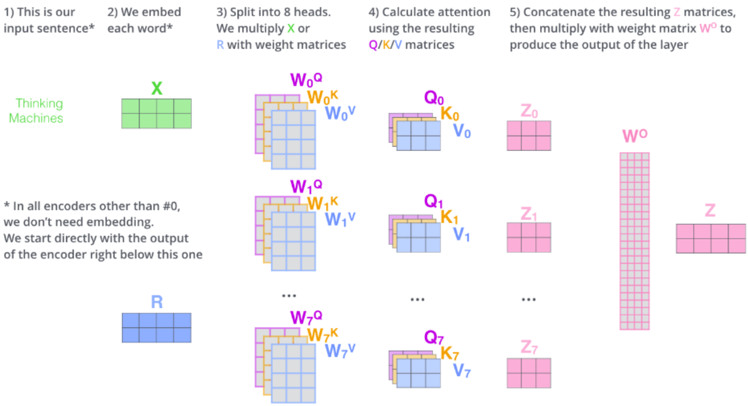
As we encode the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired" -- in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired".

Parsing Encoder: Multi-Head Attention and FFN



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



ICE #0: Self-attention Exercise

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?

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

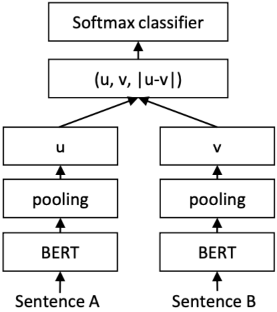


Figure 1: SBERT architecture with classification objective function, e.g., for fine-tuning on SNLI dataset. The two BERT networks have tied weights (siamese network structure).

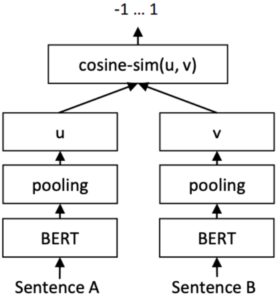


Figure 2: SBERT architecture at inference, for example, to compute similarity scores. This architecture is also used with the regression objective function.

Loss Functions for SBERT

1. Triplet Loss

Let e be the embedding of a query, e^p be the embedding of a positive example and e^n be the embedding of a negative example. The triplet loss function is given by:

$$f(e, e^p, e^n) = \max\left(\|e - e^p\|_2 - \|e - e^n\|_2 + \epsilon, 0\right)$$

Loss Functions for SBERT

2. Classification Loss

Let e_1 and e_2 be the embeddings coming out of the SBERT's last hidden layer for two sentences. Let the prediction of the SBERT for classification be as follows:

$$\hat{p} = \text{softmax}(W([e_1, e_2, |e_1 - e_2|])$$

Then the classification loss is a binary cross entropy loss betweeen $\hat{p}$ and p (the ground truth for example).

$$L = \frac{1}{N} \sum_{i=1}^{N} l(p_i, \hat{p}_i)$$

where N is the number of examples or data points in the training set. Here l is the *binary cross-entropy* loss between the prediction probability vector $\hat{p}$ and the ground truth probability vector p.

Loss Functions for SBERT

2. Classification Loss

$$\hat{p} = \text{softmax}(W([e_1, e_2, |e_1 - e_2|])$$

Then the classification loss is a binary cross entropy loss betweeen $\hat{p}$ and p (the ground truth for example).

$$\begin{aligned} L &= \tfrac{1}{N}\textstyle\sum_{i=1}^{N} l(p_i, \hat{p}_i) \\ &= \tfrac{1}{N}\textstyle\sum_{i=1}^{N} -p_i \log(\hat{p}_i) - (1 - p_i)\log(1 - \hat{p}_i) \\ &= \tfrac{1}{N}\textstyle\sum_{i=1}^{N} -p_i \log(\text{softmax}(W([e_1^i, e_2^i, |e_1^i - e_2^i|])) \\ &\quad -(1 - p_i)\log(1 - \text{softmax}(W([e_1^i, e_2^i, |e_1^i - e_2^i|])) \end{aligned}$$

where N is the number of examples or data points in the training set. Here l is the *binary cross-entropy* loss between the prediction probability vector $\hat{p}$ and the ground truth probability vector p.

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.

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 #2

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
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.

Adding Fine-tuning layers

Fine-tuning in Assignment 3

Assignment 3 to be released Friday night/Saturday morning looks at an entailment problem of whether two sentences are in agreement or in contradiction. Here, instead of cosine similarity approach, we will fine-tune a BERT model using two sentences as input. For **fine-tuning**, you will get to add hidden layers on top of a **pooled BERT embedding** and understand the performance. Notice the difference between SBERT fine-tuning and the fine-tuning we just discussed. In the former, the layers and architectures are already in place but the weights in all layers need fine-tuning. Whereas in the latter, we also add new layers on top of the **pre-trained BERT model** and fine-tune all the layers. In computer vision, there is a concept of freezing the representation layers and fine-tuning the downstream feed forward layers. So fine-tuning can be done in multiple ways and depends on the architecture and data set.

ICE #3

Assume that we are doing emotion detection using a BERT model. Why do we need to pool the output of the BERT model for the downstream task of sentence 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

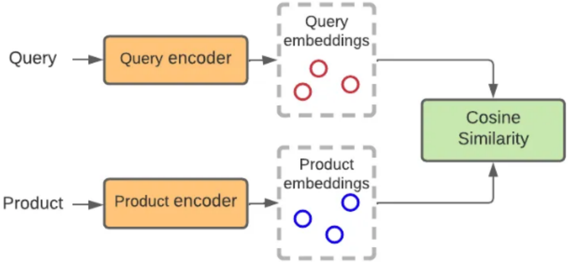


Figure 1. Conceptual diagram of a two-tower model

Two Tower Architecture

Two Towers

Self-explanatory, but there are two towers that represent two distinct objects (e.g. sentence A and sentence B or query and product or customer and product, etc).

Two Tower Architecture

Two Towers

Self-explanatory, but there are two towers that represent two distinct objects (e.g. sentence A and sentence B or query and product or customer and product, etc).

SBERT Two Tower

Is a **Siamese Two Tower**, where the weights and layers of the two towers are *identical*. In the training of a Siamese two-tower, the weights are said to be tied together between the two towers and gradients are computed keeping the tying in place.

Two Tower Architecture

Two Towers

Self-explanatory, but there are two towers that represent two distinct objects (e.g. sentence A and sentence B or query and product or customer and product, etc).

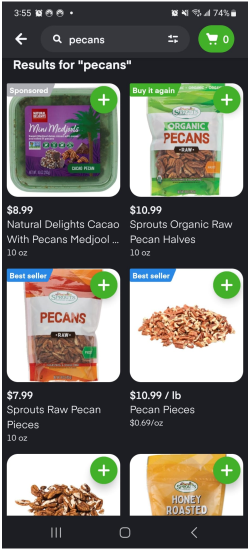
SBERT Two Tower

Is a **Siamese Two Tower**, where the weights and layers of the two towers are *identical*. In the training of a Siamese two-tower, the weights are said to be tied together between the two towers and gradients are computed keeping the tying in place.

Instacart/Recommendations Two Tower

In this example, the two towers don't refer to the same kind of object (e.g. sentence) but refer to a product and query. Hence the two towers have distinct weights learned from the data.

Positive Examples



ICE #4

Negative Examples

If **converted products** act as high-quality positive examples, by the same logic, can un-coverted products be used as negative examples from the search query? Discuss why or why not?

High-quality Positive Examples

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

Negative Examples

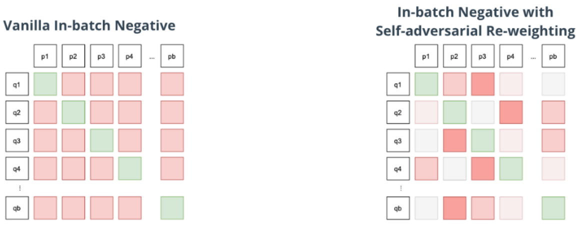


Figure 3. (Left) In the vanilla implementation of in-batch negative, all off-diagonal negative samples are given the same weight. (Right) In our implementation with self-adversarial re-weighting, harder examples are given more weight (darker color), making the task more challenging for the model.

Negative Examples

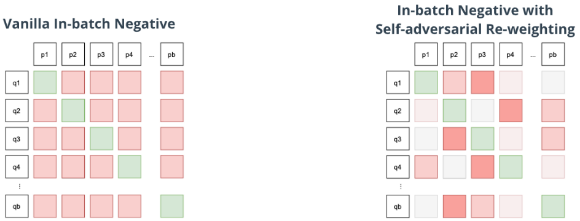


Figure 3. (Left) In the vanilla implementation of in-batch negative, all off-diagonal negative samples are given the same weight. (Right) In our implementation with self-adversarial re-weighting, harder examples are given more weight (darker color), making the task more challenging for the model.

Self-adverserial data annotation

Easy Negative examples: Tortilla $\rightarrow$ Coffee mug
Hard Negative examples: Tortilla $\rightarrow$ Tostitos Tortilla Chips

Data Augmentation for Data Set expansion

Model Training Architecture

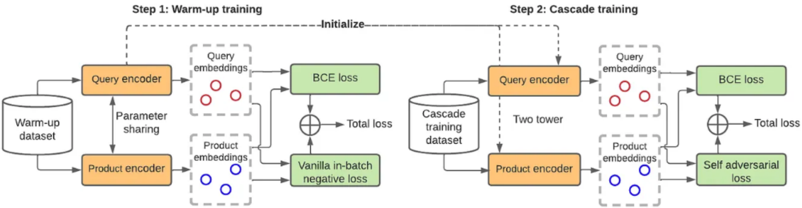


Figure 4. Two-step cascade training for ITEMS.

System Design

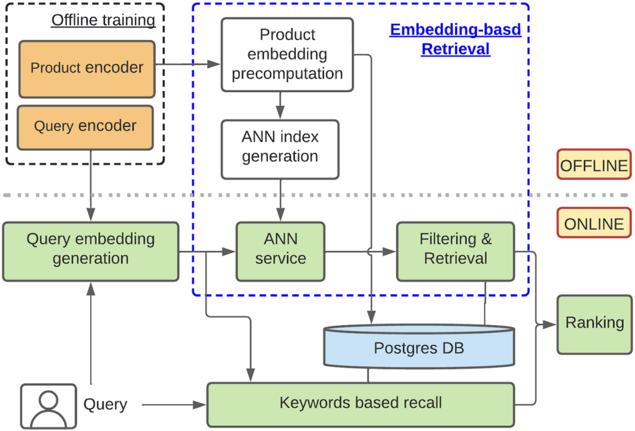


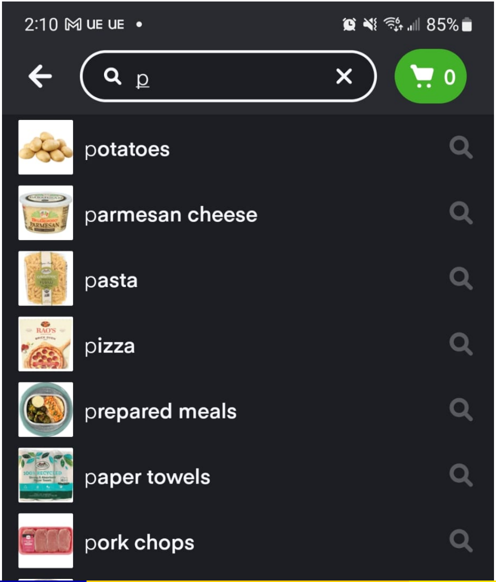
Figure 7. ITEMS system architecture.

Breakouts Time #1

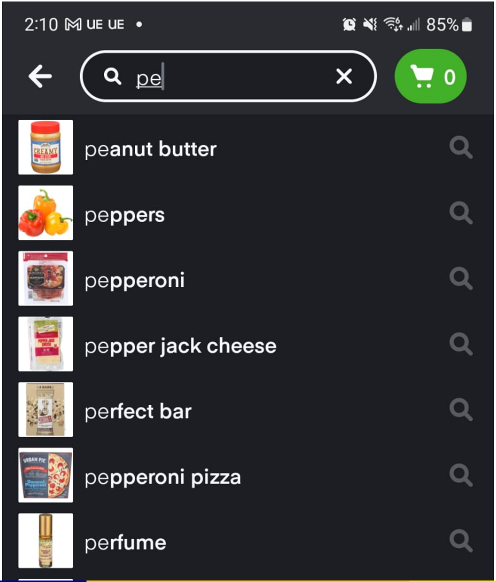
Auto-complete — 5 mins

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?

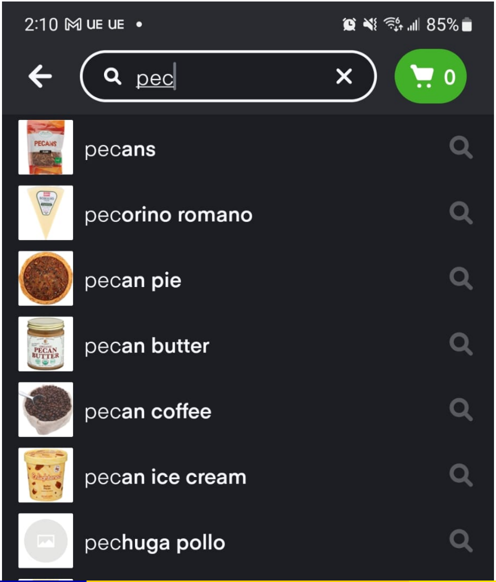
Instacart Auto-Complete and Search Relevance



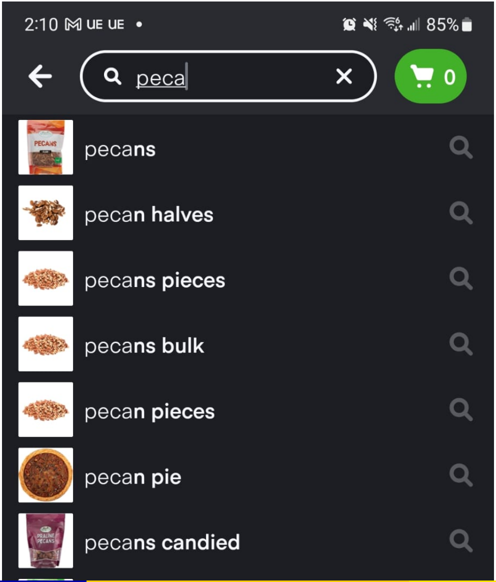
Instacart Auto-Complete



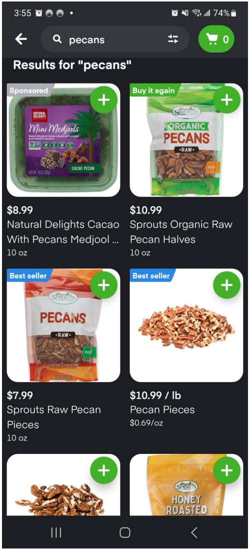
Instacart Auto-Complete



Instacart Auto-Complete



Instacart Auto-Complete and Search Results



Instacart Diversifying Auto-Complete

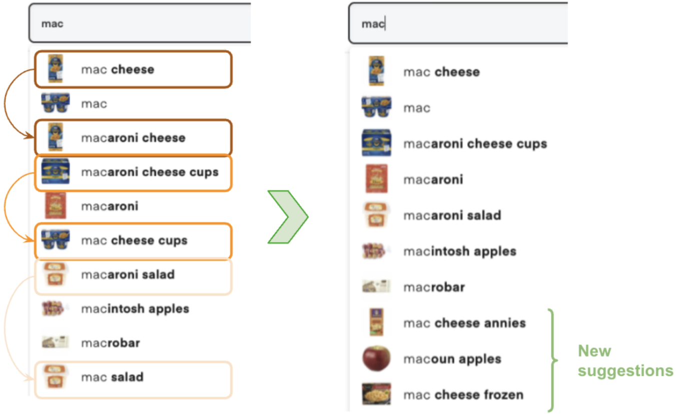


Figure 9. Autocomplete when a customer searches for "mac", before (left) and after (right) semantic deduplication.