Combining semantic search and twin product classification for recognition of purchasable items in voice shopping

Dieu-Thu Le
Amazon.com, Inc.
deule@amazon.com

Verena Weber
Amazon.com, Inc.
wverena@amazon.com

Melanie Bradford
Amazon.com, Inc.
neunerm@amazon.com

Abstract

The accuracy of an online shopping system via voice commands is particularly important and may have a great impact on customer trust. This paper focuses on the problem of detecting if an utterance contains actual and purchasable products, thus referring to a shopping-related intent in a typical Spoken Language Understanding architecture consisting of an intent classifier and a slot detector. Searching through billions of products to check if a detected slot is a purchasable item is prohibitively expensive. To overcome this problem, we present a framework that (1) uses a retrieval module that returns the most relevant products with respect to the detected slot, and (2) combines it with a twin network that decides if the detected slot is indeed a purchasable item or not. Through various experiments, we show that this architecture outperforms a typical slot detector approach, with a gain of +81% in accuracy and +41% in F1 score.

1 Introduction

Spoken Language Understanding (SLU) constitutes the backbone of voice-controlled devices such as Amazon Alexa or Google Assistant. SLU is typically divided into two sub-tasks, intent classification (IC) and slot filling (SF). IC determines the user's intent and slot filling (SF) extracts the semantic constituents. For instance, if a user says $buy\ batteries$, IC should classify the intent as $BuyItem$ and SF should label the utterance as $buy|Other\ batteries|ItemName$. The slot in this example is *ItemName* and the slot value is *batteries*. This constitutes a way to map the utterance on a semantic space. Throughout this paper the term carrier phrase denotes all tokens that are not part of the slot, e.g. *buy* in the previous example.

In typical SLU architectures, the final confidence score is obtained through multiplication of the two scores from the IC and the SF model. For each utterance, the model produces n-best hypotheses ranked by the final confidence score. One hypothesis consists of an intent and slot labels.

Problem statement - The complexity for both IC and SF tasks is continuously increasing due to the growth of voice-controlled device functionalities. Investigations have revealed that the model tends to put more weight on the carrier phrase than the slot value. This bias implies that particular slot labels are assigned based on the carrier phrase rather than the slot value. Since with a rising amount of functionalities carrier phrases can be identical across functionalities and the correct class then solely depends on the slot value. Label confusions are particularly observed for Shopping intents, e. g. *BuyItem*. For example, $how\ much\ is$ {*toilet paper*} / $was\ kostet$ {*Klopapier*} is a request that falls into the realm of Shopping functionalities because the user could actually purchase this item through the device. In contrast, if the user asks $how\ much\ is$ {*Ronaldo*} $worth$ / $was\ kostet$ {*Ronaldo*} the request needs to be processed by QA functionalities. Note that in German the two sentences have an identical pattern $How\ much\ is$ {*slot_value*} / $wie\ viel\ kostet$ {*slot_value*} while the carrier phrase slightly differs in English.

Shopping functionalities in a voice-controlled device have a particular importance due to their high potential impact on customer trust and experience. If a user has the feeling they could accidentally buy something, this may reduce trust in the system and drive customers away. On the contrary, not recognizing Shopping intents is equally harmful as users will not shop with the device anymore.

Contribution and approach - In this paper, we focus on solving the problem of label confusions, where detected slots are mistaken as *ItemName*,

Proceedings of the 4th Workshop on e-Commerce and NLP (ECNLP 4), pages 150–157

hence are routed to Shopping (false accept) and the other way around, when an item name is not identified as *ItemName*, hence are not correctly handled as Shopping (false reject). In order to correctly determine if an item is purchasable or non-purchasable, one needs to match the *ItemName* candidates to products sold on the respective platform. If the item is similar (enough) to products in the product catalog, it is purchasable.

A straightforward way of solving this problem is to search through the whole Shopping catalog to identify if the detected slot is found within the catalog or not through simple string or substring matching. However, this approach has two main drawbacks: First, it is extremely expensive to search through the full catalog for all slot candidates and is therefore prohibitive in real applications. Second, the approach is not precise and does not utilize meaning at all. Take the non-purchasable *ItemName* candidate *lamborghini* as an example. A Lamborghini cannot be purchased on e-commerce platforms and is therefore non-purchasable in this context. String or substring matching however returns matches like *lego lamborghini, toy lamborghini* from the product catalog and as such would indicate that a lamborghini could be bought through the device. Instead, we need an approach that is able to detect the difference between a *lego lamborghini* and a *lamborghini*.

To improve the accuracy of the detected slots with respect to the *ItemName* entities, we propose to combine a retrieval module with a Twin Product Classifier. Instead of searching through all billions of possible products, this approach uses a twin network (Bromley et al., 1994; Reimers and Gurevych, 2019a) to compare the candidate slot only with the semantically most similar products returned by the retrieval module from the product catalog. By looking at negative (non-purchasable) and positive (purchasable) examples of *ItemName* candidates and their matched (retrieved) items from the product catalog, the twin network learns to pull together pairs that result in a correct match (positive) and push apart pairs that are not a match (negative). The Twin Product Classifier can be used as a signal to adjust the final confidence scores for the n-best hypotheses from IC-SF.

This framework overcomes the problem of high numbers of false accept and false reject for Shopping functionalities in voice-controlled devices, while taking into account the efficiency and speed requirement for a real world application.

2 Related Work

Pre-trained transformer (Devlin et al., 2018; Liu et al., 2019) models have proven successful in many language applications including Spoken Language Understanding (SLU) (Radfar et al., 2020; Chen et al., 2019; Do and Gaspers, 2019). Transformers are typically used to create an embedding of input tokens that is then fed to a downstream task. However, as explained in more detail by Reimers and Gurevych (2019a), transformer models such as BERT are unsuitable for semantic similarity search as well as for unsupervised tasks like clustering. Reimers and Gurevych (2019a) therefore proposed Sentence-BERT (SBERT). SBERT is a modification of the pretrained BERT network based on a twin network to derive semantically meaningful sentence embeddings that can be compared via cosine similarly.

In e-commerce, product retrieval plays a key role and is a well-researched topic (Lu et al.; Ahuja et al., 2020). E-commerce systems typically perform two steps: a retrieval and a ranking step. In the retrieval step relevant products are retrieved from the product catalog. In the second step, the retrieved products are ranked by relevance with respect to the query. Semantic search (Huang et al., 2020; Nigam et al., 2019; Johnson et al., 2017; Huang et al., 2013) is state-of-the-art for product retrieval and replaced previously prevalent keyword matching (Schütze et al., 2008).

In this paper we propose a product retrieval stage in the context of SLU to retrieve potential matches for tokens labelled as *ItemName* by the SF model (*ItemName* candidate) from a product catalog. We then replace the ranking stage with a similarity-based classification to identify if the ItemName *candidate* is indeed a purchasable item or not. The goal of this approach is to reduce false accepts and false rejects for voice commerce functionalities as explained in Section 1.

3 Twin Product Classifier

In order to solve the problem of deciding if an *ItemName* candidate is purchasable or non-purchasable, we leverage information from the product catalog and employ a twin network to differentiate between these two categories. Figure 1 illustrates the general architecture of the proposed approach, which consists of two main components: (1) the catalog

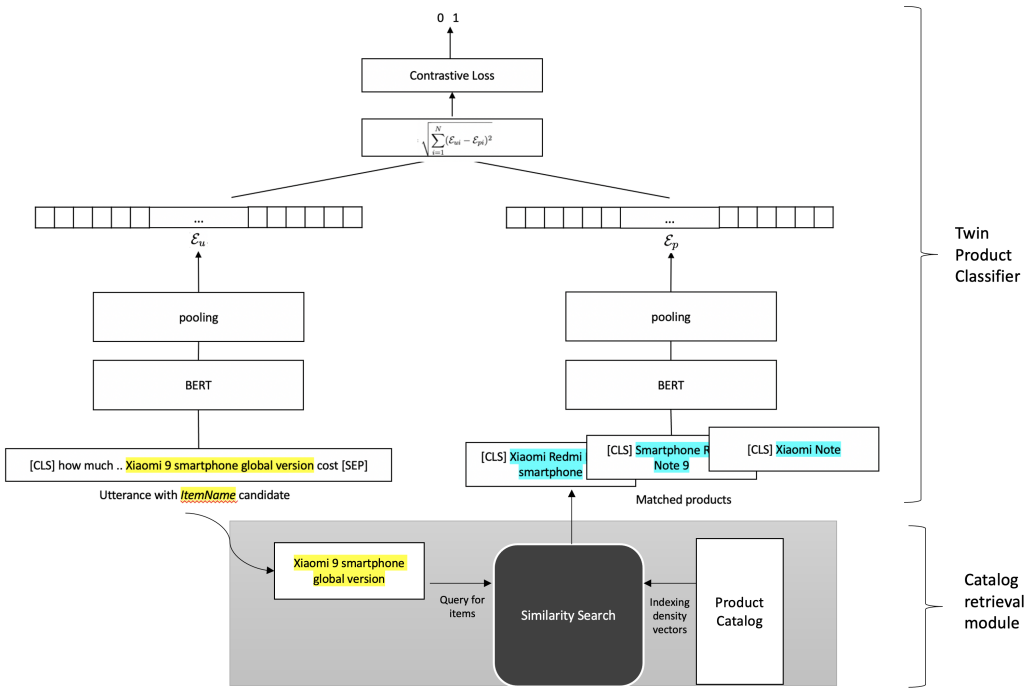

Figure 1: The general architecture of the Twin Product Classifier

retrieval module and (2) the Twin Product Classifier. In the retrieval module, the detected *ItemName* candidate will be matched with similar items indexed from the product catalog in order to find the best matching items with respect to the candidate. After that, we employ a twin network architecture (Bromley et al., 1994) and pass the original utterance on the left side and the set of matching items on the right side. Instead of learning to classify the inputs, twin networks aim at learning to differentiate between two inputs, i.e., to learn the similarity between them and to identify pairs that don't match. The similarity between these two parts (the original utterance vs. the matching items) will be computed. After that, the twin network is trained using a contrastive loss to learn to differentiate between pairs of utterances and matching catalog items to finally decide if the given item name is purchasable, i.e., could be matched with items from the product catalog.

Let u be an input utterance and c_u be the *ItemName* candidate that this utterance contains. Our task is to decide if u should be classified as a Shopping intent or not based on checking if c_u is a purchasable product. Note that the candidate c_u is provided by the n-best hypotheses from the ICSF architecture described in the previous section. From the product catalog, we retrieve a set of products $p = \{p_1, p_2, .., p_k\}$ that are semantically most similar to c_u, with $p \in \mathbf{P}$ where $\mathbf{P}$ denotes the product catalog. u and p are passed through the twin network, which yields two embeddings $\mathcal{E}_u$ and $\mathcal{E}_p$ respectively. We define a pair of utterance u and matched products p to be positive if the *ItemName* candidate c_u contained in u can be potentially found in the product catalog $\mathbf{P}$ and negative if c_u is a non-purchasable item.
Once p is retrieved from the catalog, we compute the distance between $\mathcal{E}_u$ and $\mathcal{E}_p$ using the Euclidean distance and the contrastive loss $\mathcal{L}$:

$$\mathcal{L}(u,p) = y * d(\mathcal{E}_u, \mathcal{E}_p)^2 + (1-y) * \max(\tau - d(\mathcal{E}_u, \mathcal{E}_p), 0)^2 \quad (1)$$

where y denotes the output, which is 0 for negative samples and 1 for positive samples. The distance d is computed as:

$$d(\mathcal{E}_u, \mathcal{E}_p) = \sqrt{\sum_{i=1}^{N} (\mathcal{E}_{ui} - \mathcal{E}_{pi})^2} \quad (2)$$

N is the embedding size and τ the margin parameter which determines the minimum distance a pair should have in order to be classified as negative. The final loss is the sum of both positive and negative loss. Contrastive loss has been shown to be very effective for training twin networks (Radenovic et al., 2017; Reimers and Gurevych, 2019b),

which pulls together similar pairs and push dissimilar pairs apart. The gradient of the constrastive loss is computed as:

$$\nabla\mathcal{L}(u,p) = \begin{cases} d(\mathcal{E}_u, \mathcal{E}_p) & y = 1 \\ min(d(\mathcal{E}_u, \mathcal{E}_p) - \tau, 0) & y = 0 \end{cases} \quad (3)$$

In practice, to accelerate the training process and improve the performance, we employ online contrastive loss, which selects only hard positive and hard negative samples in each mini-batch. In particular, the distance between each pair is computed and the loss is only added for pairs with the smallest or biggest distances.

With the aim of finding the best set of matching products p for an *ItemName* candidate c_u, we argue that the top products returned by fast semantic search provide essential information to the twin network to learn the similarity between c_u and p for purchasable items and generalize on unseen items. Keyword exact match search is usually faster using a reverse indexing system, but is limited since it cannot find items that are differently spelled, written or semantically close.

We use FAISS (Johnson et al., 2017), a library that has shown to be fast and efficient in similarity search for billion-scale data sets. FAISS takes embedded catalog vectors as inputs and starts the indexing process, which involves clustering the data into different clusters represented by their centroids - which are used as inverted file or index. When a query vector comes in, the most suitable cluster found based on its similarity with the centroids is returned together with the top K nearest items. FAISS is implemented for running both in CPU and GPU, supporting single and multi-GPUs together with batch processing. The encoding and indexing of $\mathbf{P}$ is done once completely offline and could be stored in memory for instant RAM search of incoming *ItemName* candidates c_u.

Integration of the Twin Product Classifier into the SLU system

Finally, the integration of the Twin Product Classifier in the whole SLU architecture is displayed in Figure 2. It illustrates the previous SLU architecture (IC-SF) on the left and the added Twin Product Classifier on the right. The IC-SF component illustrated on the left side is an independent part that one can integrate together with the twin product classifier proposed here. The flow would be as follows: the utterance is first encoded using character, positional and catalog embeddings. The catalog embedding is a fixed-size real vector for each token that indicates if the token or a substring of the token is present in one of the catalogs. Note that the product catalog used for generating the catalog embedding is a superset of the product catalog used in the Twin Product Classifier (see next Section 4.2). This first-stage embedding is then fed through a tinyBERT encoder block. The resulting encoding is then passed to the respective decoders, the SF and the IC decoder. Since the model as is has difficulties to distinguish between purchasable and non-purchasable items, we pass the utterances with *ItemName* candidates in the n-best hypothesis through the Twin Product Classifier. Dependent on the classifier feedback, the Shopping IC-SF score can be adjusted accordingly.

4 Data

4.1 Product catalog

We replace the full product catalog with the top product search queries to reduce the size and improve the matching between c_u and $p \in \mathbf{P}$. The product catalog entries contain too many details that a user would usually not include in their request, e.g *Samsung Galaxy Book Pro 15.6" — i5 11th Gen, 8GB Memory, 512GB SSD — Mystic Blue — (NP950XDB-KB2US) 2021* versus just *Samsung Galaxy Book Pro*. Moreover, the actual product catalog has a size and granularity that is not needed to detect if an item is purchasable or not. The top one million search queries already cover a wide variety of products and product categories that are representative of the full product catalog. Note that the this catalog is a subset of the catalog used for the catalog embedding in IC-SF. For simplicity, we will refer to the one million product search queries as product catalog.

4.2 Training and test data

We use a dataset with positive and negative examples to train the classifier. One example contains *(utterance, ItemName candidate, product matches, class label)*. The *ItemName* candidate is produced by the IC-SF architecture. The catalog matches are retrieved via semantic search from the product catalog using the *ItemName* candidate as a query. The data was collected and annotated over the time span of one year. All data is in German and anonymized such that users are not identifiable.

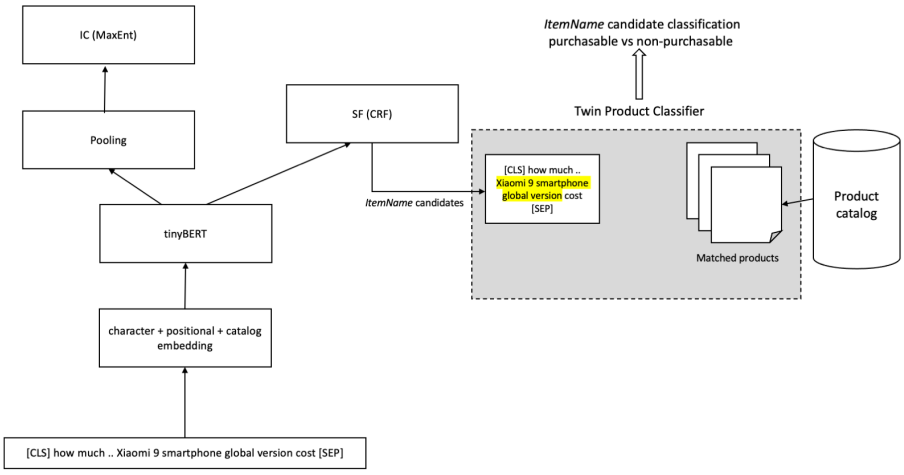

Figure 2: Integrating the Twin Product Classifier into the SLU system

To create positive and negative examples, we use all utterances for which any slot in the n-best hypotheses from IC-SF is an *ItemName* slot. The hypothesized slot is recorded as *ItemName* candidate. If the ground-truth annotation of the utterance is Shopping, the utterance serves as positive example. If the ground-truth annotation is not Shopping, we use it as a negative example. Table 1 shows that we have slightly more negative than positive examples in our dataset.

Table 2 displays the sequence length distribution for utterance, *ItemName* candidate and product matches when split by white space. While the utterances tend to be rather short sentences (on average only 5.8 tokens), the slot value can take up a substantial portion of the request itself, on average 2.2 tokens out of 5.8. For each *ItemName* candidate an average of about 19 product matches is retrieved from the product catalog using semantic search.

Dataset	Share unique *ItemName* cand.	Share positive
train	35 %	41 %
test	51 %	41 %

Table 1: Dataset statistics.

	Utterance	Item name candidate	Product matches
mean	5.8	2.2	19.0
std	2.5	1.6	8.8
min	1.0	1.0	10.0
25 %	4.0	1.0	11.0
50 %	5.0	2.0	18.0
75 %	7.0	3.0	24.0
max	53.0	30.0	85.0

Table 2: Sequence length distribution when split by white space for training set. Displays min, max, mean, std and standard quantiles of number of tokens in sequence.

5 Results

5.1 Experimental setup

BERT model for Twin network - For the BERT embedding block shown in Figure 1, we use a pretrained German BERT model, gbert-base (Chan et al., 2020; Devlin et al., 2018), fine-tuned on annotated data sampled from live traffic. For fine-tuning we use a dataset of 1.5 million live traffic utterances. Shopping utterances as well as Shopping false rejects are upsampled moderately in this dataset. The BERT model is fine-tuned in a multi-task fashion to predict two binary targets: (1) is the utterance a Shopping utterance or not, (2) is the utterance a Shopping false reject or not. Later we discard the classification layer and only use the fine-tuned BERT for the purposes of this paper.

Retrieval module - We see that semantic search works well for numbers (e.g., *hundred liter* matches *100 liter*) and different variants of a product (*i. phone x.* matches *i phone x, i phone xs*, *öl* matches *öle*). The results are quite as expected: the matches for shoppable items are usually more relevant, whereas the matches for negative examples are not immediate matches (e.g., *porsche* is matched with *porges, porsche shoes, porsche lego, porsche knife, porsche book, porsche watch*).

Twin network setup - For the implementation of

the classifier, we use the sentence transformer package[1] to setup the experiments. In each experiment, we set the number of epochs to four, the batch size for the contrastive loss is 64 and the margin τ is 0.5. We experiment with two more loss functions, multiple negatives ranking loss and multi-task learning loss. The multiple negatives ranking loss is defined as follows: Let $\{u_i, p_i\}$ be all positive pairs in our training data. Multiple negative ranking loss will generate all $\{u_i, p_j\}$ for $\forall i \neq j$ and compute the negative log-likelihood (Reimers and Gurevych, 2019b). The multi-task learning loss combines both contrastive loss and multiple negatives ranking loss by alternating between both in each batch. For experiments with the multiple negatives ranking loss and the multi-task learning loss we use a batch size of 128 together with a max sequence length of 50.

5.2 Experimental results

We design the experiments to (1) quantify the effect of the number the retrieved products, (2) compare the effect of adding the Twin Product Classifier to the original IC-SF classifier outputs. Finally, we conduct a couple of experiments using different loss functions such as multiple negatives ranking and multi-task learning and compare them to the contrastive loss, as well as cosine, Manhattan and Euclidean distance measurement in the twin network.

First of all, we compare our proposed system to the baseline, which takes the utterance as input and the hypothesis ranked first to classify if the utterance has a Shopping intent. In this baseline, the catalog is used to produce a catalog embedding to encode the utterance, but no product classifier is used. The results of this experiment are reported in Table 3.

We see a significant improvement in both accuracy (+81.6%) and F1 score (+42.2%) when adding the product classifier compared to the baseline using only intent and slot classifier with catalog embeddings as input. We observe a slight decrease in recall of -11.6% compared to an increase of +97.7% in precision. This is due to the fact that the dataset created for training and testing as described in Section 4 contains many more false accept samples than false reject samples. This is the case since the dataset was created by pulling and annotating utterances where any of the hypothesis in the n-best from IC-SF was Shopping and contained an *ItemName* slot, hence entails that IC-SF Shopping false accepts are much better represented in the dataset than IC-SF Shopping false rejects. This is hard to circumvent since annotating random samples from the overall live traffic only yields few IC-SF Shopping false rejects. Therefore, creating a dataset with many false reject samples is challenging due to the annotation limits.

Metrics	Twin Product Classifier
Accuracy	+81.58%
F1	+42.27%
Precision	+97.67%
Recall	-11.55%

Table 3: Relative difference between the baseline and Twin Product Classifier. Note that the IC-SF architecture is evaluated here on a binary task similar to the Twin Product Classifier. The IC-SF classification is evaluated as correct if the first hypothesis is a Shopping intent.

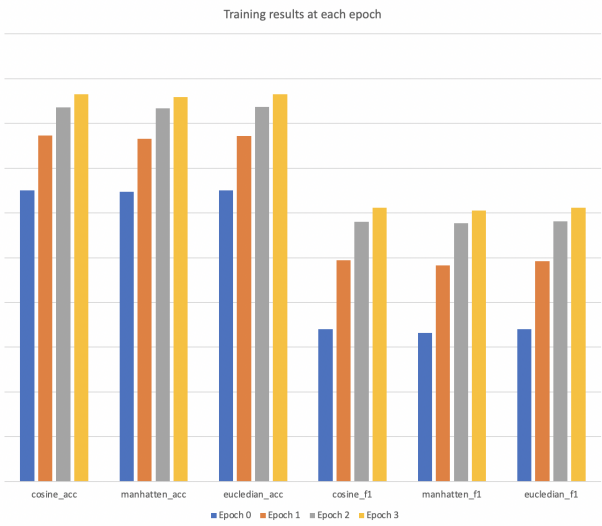

Figure 3: Performance on validation set in terms of Accuracy & F1 at each epoch using cosine, Manhattan and Euclidean distance.

We quantify the effect of the number of retrieved products in the retrieval module by comparing the results of the classifier when using only the first matched item vs. using the top five matched items. As expected, retrieving more relevant products instead of using only the first matched product gives a much better result with a gain of +14.95% in accuracy and +10.88% in precision. Interestingly, we also see a small increase in recall when using only one matched product. This can be explained by the

[1] https://github.com/UKPLab/sentence-transformers/

fact that when using only one matched item, it is likely to classify more items as purchasable (positive) and get a few more false accepts since the first matching item might not reflect the full characteristics of the *ItemName* candidate. An example we often observe is when a user searches for a Lamborghini (which is not purchasable on e-commerce platforms), the returned matches are Lamborghini toys, legos, etc. Retrieving more matching items helps to better characterize and hence to give better accuracy for purchasable items.

Metrics	Cosine similarity	Manhattan distance	Euclidean distance
Accuracy	+14.95%	+14.92%	+14.95%
F1	+10.69%	+10.84%	+10.88%
Precision	+23.30%	+24.95%	+23.88%
Recall	-3.65%	-4.88%	-3.85%

Table 4: Relative difference between the system using the first matching product and the top five matching products

In the next experiment, we further extend the left side of the twin network with the whole utterance, instead of using only the *ItemName* candidate. We see a big gain in both accuracy (+8.5%) and F1 (+9.72%) (Table 5). In this case, the whole utterance is proven to contain important information for deciding if the intent is related to Shopping or not. Hence extending the comparison between *itemName* candidates and matched products with the whole utterance information is beneficial.

Metrics	Cosine similarity	Manhattan distance	Euclidean distance
Accuracy	+8.53%	+8.49%	+8.54%
F1	+9.79%	+9.64%	+9.72%
Precision	+18.72%	+18.11%	+18.26%
Recall	+1.17%	+1.56%	+1.44%

Table 5: Relative difference between the system using only the *ItemName* candidate vs. using the whole utterance in training the classifier

Next, we quantify the effect of different similarity and distance measures for computing the contrastive loss (Table 6). Taking the cosine similarity as the baseline, we compute the relative changes when using the Manhattan distance and Euclidean distance. From the experimental results, we do not see much difference overall with respect to using different distance measures. Euclidean distance gives a slightly better result though.

Finally, we evaluate the impact of using different loss functions for our approach. We compare online contrastive loss with multiple negatives ranking loss and the multi-task loss. The results are reported in Table 7. We see that using the multiple negatives ranking loss function as well as the multi-task loss function gives much lower performance than the contrastive loss. The multiple negatives ranking loss function is usually more useful in the information retrieval use case, when one has more positive pairs. Overall, the contrastive loss function has shown to yield the best performance for this use case.

Metrics	Manhattan distance	Euclidean distance
Accuracy	-0.03%	+0.01%
F1	-0.11%	+0.00%
Precision	+0.50%	-0.14%
Recall	-0.70%	+0.10%

Table 6: Relative difference when using different distance metrics (with cosine similarity as the baseline)

Metrics	Negative ranking	Multi-task loss
Accuracy	-30.02%	-12.64%
F1	-32.33%	-13.21%
Precision	-50.74%	-22.56%
Recall	+6.23%	-1.69%

Table 7: Relative difference when using alternative loss functions compared to online contrastive loss

We also report the performance (Accuracy and F1) at each epoch measured on a validation set that has same size and class distribution as the test set for the three different distance metrics, where we see steady improvement at each epoch (Figure 3).

6 Conclusion

We have presented an architecture with a retrieval module and a twin product classification module to verify if a detected *ItemName* slot from the SF model contains a purchasable item or not. The experimental results have shown the effectiveness of the framework to increase accuracy by +81% for purchasable item detection. For the retrieval module, we have experimented with different numbers of matching products returned by semantic search. We showed that using the top five most relevant product names yields the best results. Moreover, adding the whole utterance in the twin network and using an online contrastive loss function resulted in the best performance. This approach can be lever-

aged in a voice-based shopping system to decrease the number of false rejects and false accepts and thereby improve customer experience.

References

Aman Ahuja, Nikhil Rao, Sumeet Katariya, Karthik Subbian, and Chandan K Reddy. 2020. Language-agnostic representation learning for product search on e-commerce platforms. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 7–15.

Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1994. Signature verification using a "siamese" time delay neural network. In *Advances in Neural Information Processing Systems*, volume 6. Morgan-Kaufmann.

Branden Chan, Stefan Schweter, and Timo Möller. 2020. German's next language model. *arXiv preprint arXiv:2010.10906*.

Q. Chen, Z. Zhuo, and W. Wang. 2019. Bert for joint intent classification and slot filling. *arXiv:1902.10909*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding.

Quynh Ngoc Thi Do and Judith Gaspers. 2019. Cross-lingual transfer learning for spoken language understanding. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5956–5960. IEEE.

Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-based retrieval in facebook search. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2553–2561.

Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 2333–2338.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with gpus. *CoRR*, abs/1702.08734.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Hanqing Lu, Youna Hu, Tong Zhao, Tony Wu, Yiwei Song, and Bing Yin. Graph-based multilingual product retrieval in e-commerce search.

Priyanka Nigam, Yiwei Song, Vijai Mohan, Vihan Lakshman, Weitian Ding, Ankit Shingavi, Choon Hui Teo, Hao Gu, and Bing Yin. 2019. Semantic product search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2876–2885.

Filip Radenovic, Giorgos Tolias, and Ondrej Chum. 2017. Fine-tuning CNN image retrieval with no human annotation. *CoRR*, abs/1711.02512.

Martin Radfar, Athanasios Mouchtaris, and Siegfried Kunzmann. 2020. End-to-end neural transformer based spoken language understanding. *arXiv preprint arXiv:2008.10984*.

Nils Reimers and Iryna Gurevych. 2019a. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Nils Reimers and Iryna Gurevych. 2019b. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. 2008. *Introduction to information retrieval*, volume 39. Cambridge University Press Cambridge.