Transformer-Based Deep Siamese Network for At-Scale Product Matching and One-Shot Hierarchy Classification

ALEXANDRE VILCEK, SETH MOTTAGHINEJAD, and STEVEN SHI, Microsoft, USA
KETKI GUPTE, Walmart Labs, INDIA
SUJITHA PASUMARTY, LINSEY PANG, and PRAKHAR MEHROTRA[*], Walmart Labs, USA

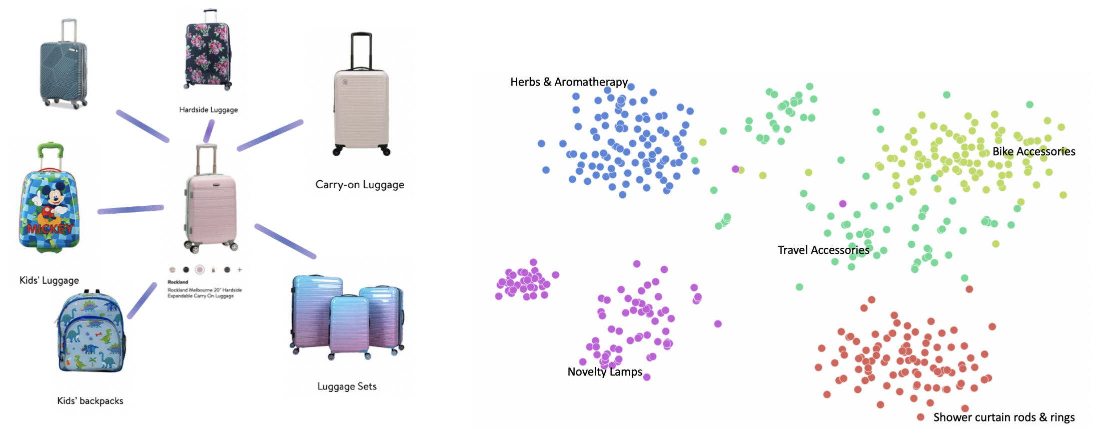


Fig. 1. Product Matching, Category Mapping and Embedding

In the current digital world, improving user-product interaction is more important than ever for both consumers and merchants. In this paper, we present a Transformer-based deep Siamese network for product matching and one-shot product taxonomy classification: We begin by using textual descriptions of products as input data to train the Siamese network initialized using a Microsoft DeBERTa[1] pre-trained model. During training, the network minimizes the distance between similar pairs of products while pushing dissimilar pairs away from each other, and by doing so it learns to map products with similar descriptions to nearby locations in an embedding space, thereby creating "clusters" of similar products. Once trained, the model can output the contextual embeddings for any given product description. We then use these product embeddings as inputs to a downstream one-shot hierarchical classification task in order to predict the product's taxonomy. For all existing products, the label (the product taxonomy) is based on the product internal catalog. Finally, we design and conduct multiple experiments to validate and verify this approach.

[*]Authors contributed equally to this work.

Authors' addresses: Alexandre Vilcek, alvilcek@microsoft.com; Seth Mottaghinejad, sethmott@microsoft.com; Steven Shi, Steven.Yukai.Shi@microsoft.com, Microsoft, USA; Ketki Gupte, Ketki.Gupte@walmart.com, Walmart Labs, INDIA; Sujitha Pasumarty, Sujitha.Pasumarty@walmart.com; Linsey Pang, Linsey.Pang@walmart.com; Prakhar Mehrotra, Prakhar.Mehrotra@walmart.com, Walmart Labs, USA.

CCS Concepts: • **Data Mining** → **Embedding Vector Space**; • **Machine Learning** → *Deep Saimese Network, one-shot learning*; • **Recommendation** → Product matching, categorization.

Additional Key Words and Phrases: Deep neural networks, Encoder, Product similarity, Product Hierarchy Classification

ACM Reference Format:
Alexandre Vilcek, Seth Mottaghinejad, Steven Shi, Ketki Gupte, Sujitha Pasumarty, Linsey Pang, and Prakhar Mehrotra. 2018. Transformer-Based Deep Siamese Network for At-Scale Product Matching and One-Shot Hierarchy Classification. 1, 1 (July 2018), 11 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

In E-commerce, each day billions of products are listed and sold with millions of active sellers trading online. Therefore providing satisfactory search and purchase experience is essential for growth in the digital world, but presents many challenges as well. One of the big challenges is finding ways to keep improving product offerings based on feedback from both consumers and merchants. On the consumer side, a user might see the same product but the amount and extent of information on each merchant's website about the product may vary considerably. A user might end up purchasing the wrong product if the right product is wrongly classified or if its attributes do not align well with those of similar products in the catalog, resulting in poor retrievals when querying products based on their shared attributes. To improve user shopping experience, for any product, we need to identify whether it exists in the current catalog and find similar products that can act as a substitute, and we need to be able to do so on the fly and at scale. This gives consumers a better navigation and search experience as they look for products, and it helps merchants identify substitute products, classify new products as they come to keep the catalog up-to-date, or even to consolidate two different catalogs by aligning their taxonomies.

To tackle this problem, we rely on deep learning techniques to (1) read a new product's textual description, (2) find similar products to it based on semantic similarity in their descriptions, and (3) use them to predict the new product's taxonomy. Since the product taxonomy follows a hierarchical structure, the prediction task is a hierarchical multi-class classification task, so for each product, the model predicts one class for each level of the hierarchy. By learning a similarity function instead of training a classifier from scratch, at inference time we view the product taxonomy prediction as a one-shot learning problem [2], as it does not require that every class in the taxonomy be present in the training data. This is especially important as product catalogs evolve over time and we do not wish to retrain the model every time there is a change to the catalog, such as when new products are introduced. Moreover, after we train the model, we use it to precompute embedding vectors for all product descriptions in our catalog. In this way, we can use these precomputed vectors when finding similar products and performing taxonomy classification in a very efficient manner, at scale.

In summary, here's what we set out to accomplish:

- Train a product similarity model using a Transformer-based deep Siamese network based on a pre-trained Microsoft DeBERTa[1] model, which is fine-tuned using textual product descriptions
- Use the model to generate contextual embeddings that capture the semantic information in product descriptions
- Use the generated embeddings for a downstream task consisting of one-shot product taxonomy classification
- Design and conduct multiple experiments to verify and validate the approach

2 RELATED WORK

2.1 Product Matching

Product matching identifies how similar two products or groups of products are. This problem has been explored in the past using various approaches, such as (1) extracting attribute-value pairs from product descriptions to convert unstructured text to a structured representation, (2) using fuzzy string-matching methods (e.g. Levenshtein distance), or (3) count-based methods (e.g. bag of words, TF-IDF), or NLP-based methods usch as entity extraction or key phrase extraction.

Instead of explicitly learning structured data, newer approaches propose to let models learn representations from large unlabeled data. Vector based models such as word2vec [3], GloVe [4] and Skip-Thought [5] have shown promising results on textual data for learning semantic representations.

Recent advancement in representation learning using zero/one-shot learning has found successful applications in verification tasks such as facial recognition for passport checks.[6] Early variants of Siamese neural networks trained on sequence data relied on recurrent neural networks for their architecture, including Manhattan LSTMs [7] and bi-directional LSTMs [8]. Our work takes the motivation from these Siamese models and improves the architecture by using the Transformer-based Microsoft DeBERTa as the encoder and fine-tuning the model using data in the E-commerce domain.

2.2 Product Taxonomy Classification

Product taxonomy classification is especially important on an E-commerce platform as it gives consumers a systematic way of navigating the catalog and finding relevant products. A product's taxonomy is usually described using a hierarchical (tree-like) structure. Hierarchical classification can be performed in several ways. For example we could train a single "flat" classifier to predict the exact class down to leaves of the tree (the deepest level), ignoring its hierarchical structure. Alternatively, we can perform some sort of step-wise classification either by training a classifier for each level of the hierarchy, or by training a classifier for each parent class. At prediction time, we first predict the class for the highest level of the hierarchy. Then for each level after that, we limit the prediction to the classes that belong to parent class predicted for the previous level. We do this so that the predicted classes at each level align with the taxonomy.

For an input data consisting of product descriptions, we can extract features using N-grams and use SVMs for classification, as shown by [9]. We can use context-free embeddings like word2vec to create richer features as shown by [10]. More recently, recurrent neural networks (see [11]) and convolutional neural networks (see [12]) were used to learn contextual representations of product attributes. Siamese neural networks have also been studied in this context by [7] who used Manhattan LSTMs and [13] who used bi-directional LSTMs to improve the contextual representation for the purpose of product classification. The approach we propose is also based on Siamese neural networks and as such it can be used to do one-shot classification: Once deployed the model can categorize any new product or even identify niche product lines over time without the need to update or re-train the model.

3 PROPOSED APPROACH

3.0.1 Siamese Network. Siamese networks[14] (see Figure 2) are a special type of neural network architecture having two or more identical sub-networks (trained simultaneously with shared weights). Instead of a model learning to classify its inputs directly, the neural network learns to differentiate between two inputs (using contrastive loss) or three inputs (using triplet loss, our choice). These networks are able to learn good contextual representations (embeddings) from text data such as product descriptions. To learn these embeddings during training, our Siamese network runs three parallel identical sub-networks (illustrated in Figure 2b), one takes the anchor as input, and the other two take a positive match and negative match as input. These terms are explained in the next section. At inference time, we can use the trained sub-network to get embeddings from the raw input data (illustrated in Figure 2c).

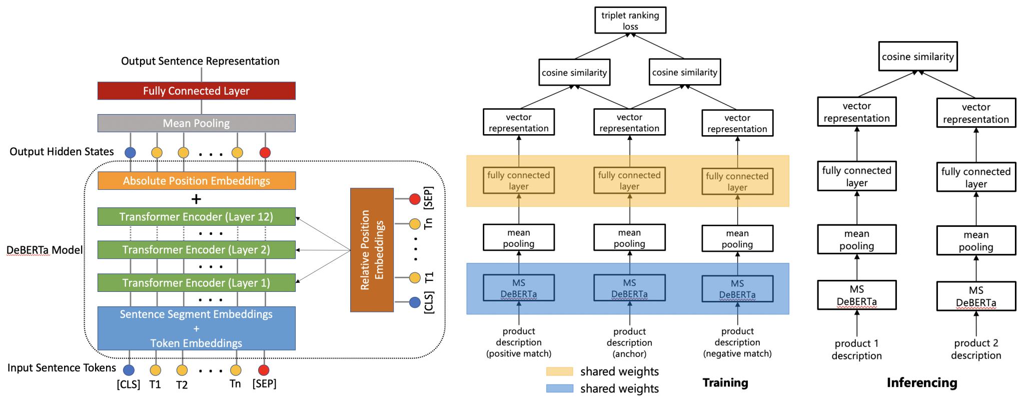


Fig. 2. Framework: (a) Encoder Architecture (b) Architecture for Model Training (c) Architecture for Model Inference

For the sub-networks in the Siamese network to learn a good embedding, we need a way to compute similarity between two embedding vectors: in other words, given two inputs (product descriptions, in our case) x and y, the function should return a similarity score $s(x, y)$. Cosine similarity is the most common choice and is given by

$$s(x,y) = \frac{e(x) \cdot e(y)}{\|e(x)\| \|e(y)\|}$$

where $e()$ is the function that projects the raw input to an embedding vector (the sub-network in the Siamese network) and $\|e(x)\|$ denotes the norm of the vector $e(x)$.

3.1 Sub-network Architecture

For each sub-network of the Siamese network, we use DeBERTa (decoding-enhanced BERT with disentangled attention), which is a Transformer-based neural language model pre-trained on large amounts of raw text corpora using self-supervised learning. Like other similar models, DeBERTa is intended to learn universal language representations that can be adapted to various downstream natural language understanding (NLU) tasks. DeBERTa improves previous state-of-the-art models (for example, BERT[15], RoBERTa[16]) using two novel architecture improvements (illustrated in Figure 2a: a disentangled attention mechanism and an enhanced masked decoder. DeBERTa represents each word using two vectors that encode its content and position, respectively, and the attention weights among words are computed using disentangled matrices based on their contents and relative positions, respectively. For the Masked Language Modeling task in pre-training, the Enhanced Mask Decoder in DeBERTa incorporates absolute word position embeddings just before the softmax layer where the model decodes the masked words based on the aggregated contextual embeddings of word contents and positions. In their work, DeBERTa authors also propose a new virtual adversarial training method for fine-tuning, but in our work we are performing the fine-tuning as described in section 3.3. On top of the DeBERTa

Encoder, we perform mean-pooling on the output hidden states from the last Transformer layer and pass the result to be processed by a fully-connected layer with a *tanh* activation function. From this layer, we get embeddings of size 128 for any given product description.

3.2 Triplet Loss Objective

We start with the pre-trained model from the previous section, but we further fine-tune it by training it on product descriptions for products in our catalog. Each training data point is defined as a triplet $x = (a, p, n)$, where a is the anchor product description (baseline), p is a matching product description (truthy) and n is a non-matching negative product description (falsy). So the anchor input is compared to a positive input and a negative input. We rely on the triplet loss during training: The distance from the anchor to the positive example is minimized, while the distance from the anchor to the negative example is maximized. The triplet loss function is given by

$$L_{triplet} = \frac{1}{|X|} \sum_{(a,p,n) \in X} max(|s(a,p) - s(a,n) + \alpha|, 0)$$

where X is the set of (a, n, p) triplets and α is a margin between positive and negative pairs. The triplet loss pushes $s(a, p)$ towards 1 and pushes $s(a, n)$ below than $s(a, p)$ by at least α. We set $\alpha = 0.3$ for the output shown in this work.

3.3 Model Training

The training data is created from our internal catalog using the following approach: For each triplet, we randomly pick a product as anchor, another product with a matching taxonomy as a positive example and a third product with a non-matching taxonomy as a negative example. We provide two kinds of negative examples: "easy negatives" are products whose taxonomy is very different, and "hard negatives" are products whose taxonomy is similar but still different from the anchor. For example, if tooth paste is the anchor, then men's shoes could be an easy negative example, while mouth wash would be a hard negative example. Easy negatives can be sampled randomly, but to find hard negative examples we rely on the hierarchical structure of the product taxonomy. A hard negative example is one that agrees with the product up to some level of the taxonomy but diverges after that. Since our internal taxonomy uses a 5-level hierarchy, we can choose hard negative examples so that they match with the anchor in levels 1 through 4 of the hierarchy, but not match at level 5. By including enough hard negative examples, we force the model to focus on more subtle differences in order to distinguish products based on their descriptions. In this work, we have roughly the same proportion of easy and hard negative examples in the data. As we are working with short product descriptions, when tokenizing the input data we truncate (and pad as needed) the text into a list of 128 tokens.

We split the data into training, evaluation and test sets. We use the validation data for hyper-parameter tuning. We use the test data to compare the model's performance on examples it never saw. The training set has approximately 281,000 data points, the test and validation data have 5,400 and 1,800 respectively. The relative agreement between the product taxonomies at different levels is shown in Figure 3, with the extent of agreement as a percentage (y-axis) between the anchor and positive examples (blue line), anchor and easy negative examples (red line) and anchor and hard negative examples (green line) at each of the five levels (x-axis) of the product taxonomy. We can see that while the anchor and positive examples agree at all five levels, the anchor and hard negative examples agree for levels 1-4 but disagree at level 5, while the anchors and easy negative examples disagree at almost all levels (if they agree it is due to chance only).

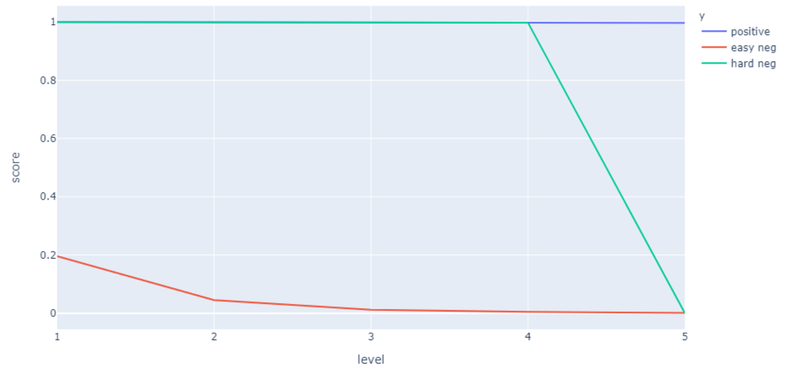


Fig. 3. Triplet Agreement

No. of GPUs	Execution Time Per Epoch (minutes)
8	39.93
16	19.98
24	13.29
32	9.46
40	8.00

Table 1. Performance scalability at model training

We run the model fine-tuning in a data-parallel, distributed approach using PyTorch on Azure Machine Learning, and evaluate its scalability in terms of training time per epoch across many back-end configurations comprised of GPU-nodes. In our cluster, each node has 8 GPUs (NVidia Tesla V100). Table 1 shows the training times, in terms of training duration per epoch, scaling from 8 to 40 GPUs, in increments of 8 GPUs. We trained the model for a total of 30 epochs, by fine-tuning all model weights.

Once the model is trained on this data using the triplet loss described in section 3.2, we can get embeddings for each product in the catalog, as well as any future product with a description. These embeddings encode the contextual information contained in their product descriptions: Products with similar descriptions will have similar embeddings. In Figure 4, we see a heatmap of the similarity matrix between product embeddings after training is done. The axes represent the product index, while the color gradient encodes the degree of similarity between two products (i.e. the similarity score): from purple for very dissimilar to yellow for very similar. Our catalog contained far more products, but we downsampled the similarity matrix in order to render the visualization. Moreover, in the matrix products are ordered so that products with the same taxonomy are next to each other. The square patches along the diagonal line indicate that products with the same taxonomy have similar embeddings, while the streaks yellow vertical (or horizontal) lines extending further out indicates that certain groups of products share similar embeddings with products with different taxonomies. Both conclusions match our expectation of the data.

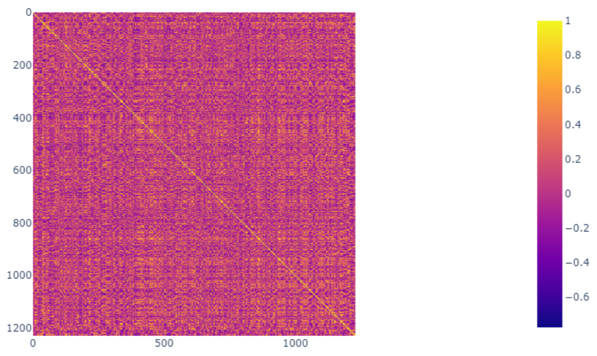


Fig. 4. Heatmap of product embeddings

3.4 Model Inference

At inference time, the model is given a product description for some new product, and returns its embedding vector. With some post-processing which we now describe, we can predict the product's taxonomy using its embedding and the embeddings of products currently in the catalog. Our approach for classification is very similar to the k-nearest neighbor algorithm: The prediction relies on finding k similar products and aggregating their taxonomy in order to predict the new product's taxonomy. This approach is relatively simple compared to training a hierarchical multi-stage classifier. The steps involved are explained in more detail below:

Product Matching: Given a new product description as "query", our model finds its embedding representation followed by the top-k products whose embeddings are most similar to the query product using cosine similarity. This is similar to how recommender systems work. To speed up the computation at inference time, we can not only pre-compute embeddings for all the products in the catalog, but also also pre-compute the top-k matches for every product in the catalog. For an out-of-catalog product whose top-k most similar products from the catalog need to be found on-the-fly, we can speed up the search by pre-computing aggregated taxonomy-level embeddings (take the average embeddings for all products with the same taxonomy), and limiting the search to products whose taxonomy has aggregated embeddings that are among the top-k most similar to the query product's embedding. After hyper-parameter tuning, we chose $k = 50$.

Classification: Once we have the top-k matches, we predict the query product taxonomy by simply taking a vote: We predict the most common taxonomy of the top-k matches. If needed, we break ties by choosing the taxonomy whose aggregated embedding is closer to the query product embedding. The prediction step here is very similar to using a k-NN classifier with product embeddings as input and the multi-level taxonomy as target. Let's look at a simple example: Using a 5-level taxonomy, with level 1 being the least granular and level 5 the most granular, a product description such as "women's light-weight breathable pair of yoga pants, black" the model would predict the following taxonomy:

```
{1: "apparel", 2: "women's apparel", 3: "sportswear", 4: "pants", 5: "yoga pants"}
```

This means that of the $k = 50$ products with the most similar descriptions, most fell into the above taxonomy. In this example, we predicted the taxonomy up to the deepest level (level 5), but we can also predict up to some lower level, in which case the model will have higher accuracy at the cost of having less precision. Note that if for example we ask the model to predict up to level 4, this doesn't necessarily mean that the model would still predict "pants" for the above example, because the most common category (mode) at up to level 4 (but ignoring level 5) for the $k = 50$ most similar products is not necessarily the same category. So while the 5-level predictions are always consistent with respect to the hierarchy of the taxonomy, they can change based on how precise we want them to be.

Let's now suppose the model's prediction mis-classifies the object as such:

```
{1: "apparel", 2: "women's apparel", 3: "sportswear", 4: "leggings", 5: "sweat pants"}
```

It would be more correct to say that the model correctly classifies the product up to level 3, but mis-classifies it at levels 4 and 5. But let's say we ask the model to provide the top-2 predictions (instead of just the top prediction), given by

```
{1: ["apparel", "durables"],
 2: ["women's apparel", "men's apparel"],
 3: ["sportswear", "comfort line"],
 4: ["leggings", "pants"],
 5: ["sweat pants", "yoga pants"]}
```

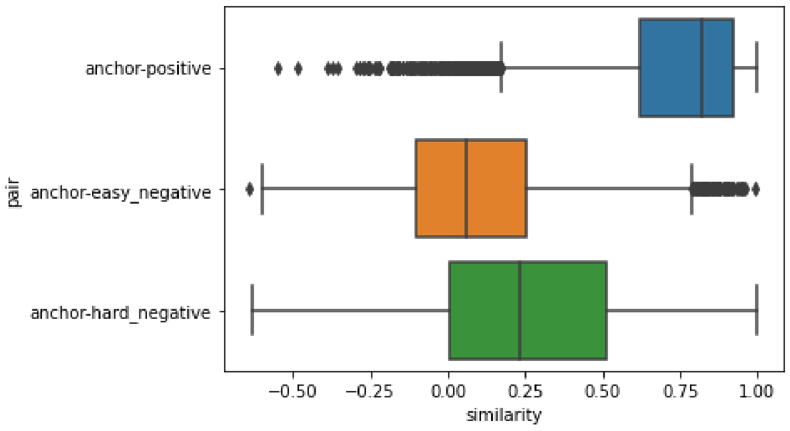
At each level, we get two predictions in order of importance, i.e. the most common, followed by the next most common category. We can say that the model has 100 percent top-1 accuracy up to level 3, but 100 percent top-2 accuracy up to level 5. This approach is simple but effective: Since finding similar products is always possible (even if the similarity values are low), all we have to do is count and sort their taxonomy up to some level of the hierarchy to get a prediction up to that level. And just as with recommender systems, we can ask the model for top-m predictions, not just the top prediction.

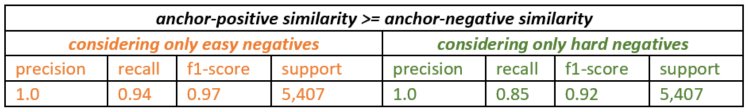
3.5 Model Evaluation

As discussed in section 2.2 hierarchical multi-class classification models pose unique challenges, especially when the number of categories in the taxonomy is very large. So to evaluate the model, we instead use a rank-aware evaluation metric, namely top-m accuracy (we vary m from 1 to 10) and compare model performance on different kinds of examples to see if the model shows signs of having learned the taxonomy. We also examine how top-m accuracy differs when comparing the ground truth (anchor) to top-m predictions made for anchor, positive, hard negative and easy negative examples. A good model should be able to predict the taxonomy of the anchor itself, but also predict the same taxonomy for the anchor and positive examples, and predict different taxonomies for the anchor and negative examples. Finally, the model should find it harder to do so for hard negative compared with easy negative examples, and we want to know how much harder. In the following tests, we set out to verify and answer these claims.

Test I: In this test, let $p\_positive$ be the similarity score between anchor-positive pairs, and $p\_negative$ the similarity score between anchor-negative pairs. So if presented with a triplet we should see $p\_positive >= p\_negative$. We perform this test once using easy negative examples and again using hard negative examples. In Figure 5, we can see how well the model separates the anchor-positive pairs from the anchor-negative pairs.

Test II: In this test, we design a 5-level hierarchical classification task: For each product in the test set, the classifier predicts the product taxonomy at each of the five levels of its hierarchy. We evaluate predictions by taking hierarchy into account in the following way: We ask how often the actual and predicted class match *up to* some level $1 <= l <= 5$ of the hierarchy, where the larger l is the harder the prediction task gets. So the best prediction is correct up to level 5,





anchor-positive similarity >= anchor-negative similarity							
considering only easy negatives				*considering only hard negatives*			
precision	recall	f1-score	support	precision	recall	f1-score	support
1.0	0.94	0.97	5,407	1.0	0.85	0.92	5,407

Fig. 5. The boxplot in the figure above shows the distribution of cosine similarities (x-axis) between anchor and positive examples, anchor and hard negative examples, and anchor and easy negative examples, from the test set. The subsequent table shows the corresponding classification metrics.

and the worst prediction fails to classify correctly even at level 1. In Figure 6 we can see top-m accuracy values (y-axis) as m (x-axis) changes. The colors represent the levels in the taxonomy (level 1 was excluded to avoid over-plotting). At every level of the taxonomy, we use the class of the anchor as the label, and let the model predict the top-m classes for the anchor (solid line), positive (dotted line), hard negative (dashed line) and easy negative examples (long dashed line).

We can see some rather obvious, but also some noteworthy results: (1) accuracy drops as we try to predict the taxonomy at a deeper level; (2) accuracy increases as we increase m; (3) accuracy is near zero for all the easy negative examples as well as for the hard negative example at level 5 (recall that the hard negative and anchor examples have the same taxonomy for levels 1-4 but differ at level 5); (4) the model predicts the taxonomy for anchor and positive examples with near equal accuracy, confirming how it's able to learn similarities; (5) for hard negative examples at levels 1-4, the model predicts the correct class, but with a drop in accuracy of around 10 percent compared to positive examples. This dip in accuracy is due to the added difficulty of classifying hard negative examples. However, the gap seems to narrow a little as we increase m.

Note that throughout the analysis, the product taxonomy is only used on two separate occasions: (1) to find anchor, positive, and easy/hard negative triplets for the training data, and (2) to evaluate the model's performance at different levels of the taxonomy. Our approach does *not* use product taxonomy during training, yet the learned embeddings are rich enough that a simple downstream task can reveal the product taxonomy with impressive accuracy. To further explore this, we see in Figure 7, how deep (y-axis) on average the model can predict the taxonomy correctly, using top-m accuracy and for different values of m (x-axis). For example, using top-2 accuracy, if a model correctly classifies levels 1 to 3 of the product taxonomy but mis-classifies the remaining two levels, this means that the product's actual

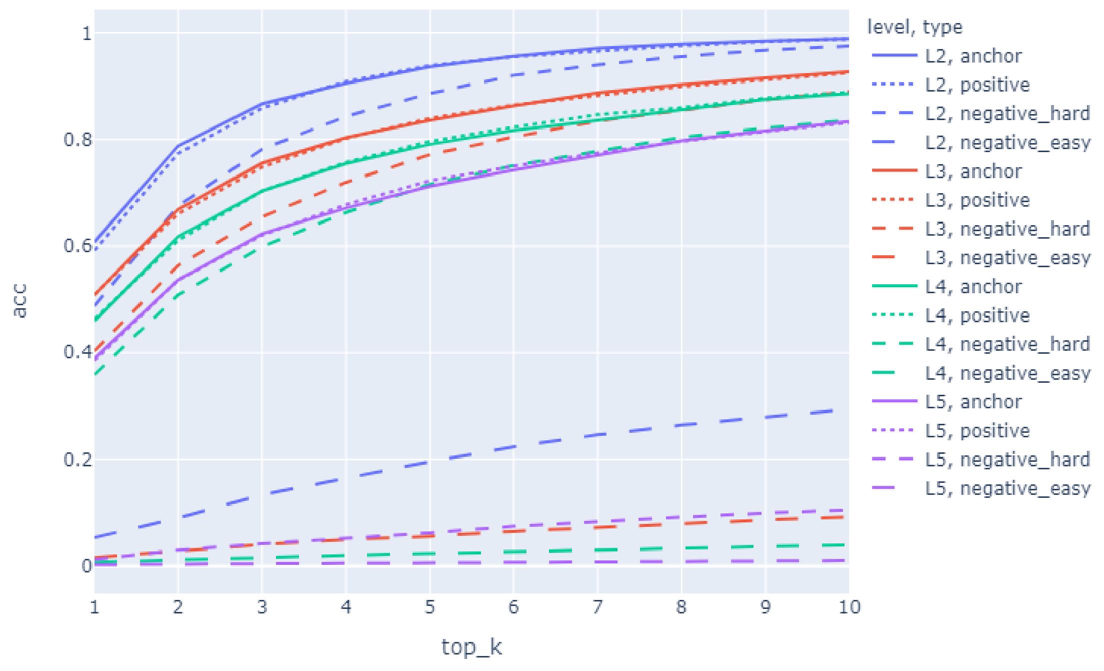


Fig. 6. top-m accuracy values (y-axis) as m (x-axis) increases

taxonomy (ground truth) is in the model's top-2 predictions for levels 1-3, but not for levels 4 and 5. We can refer to this as the *depth* of the prediction, if we imagine the taxonomy as a tree. In the plot, we see how using $m = 1$ the model has an average depth ≈ 2.75, so on average we expect it to predict only the first 2 to 3 levels correctly, but if we rely on the top-2 predictions instead ($m = 2$), then the model's predictions are correct on average up to 3 to 4 levels deep. To get correct predictions up to level 5, which is the deepest (most granular) level in the taxonomy, $m = 8$ seems like a reasonable choice. This number is practically useful because it indicates that if we wanted an expert to label an out-of-catalog product but choose the label among a set of choices (instead of navigating the taxonomy which would require a lot more domain knowledge), then a reasonable approach is to let the model give its top-8 choices and we should expect good results.

4 CONCLUSION AND FUTURE WORK

Many of the results we found showcase the power of Siamese networks and self-supervised learning on one hard, and the flexibility with which they can be applied to downstream tasks on the other hand. The future scope of work could include extending this approach to multi-modal data that includes not only descriptions but also images. It would also be interesting to compare the use of the triplet loss with the quadruplet loss to see if it can improve classification accuracy for hard negatives. Finally, in production systems, it could be interesting to explore ways to monitor new

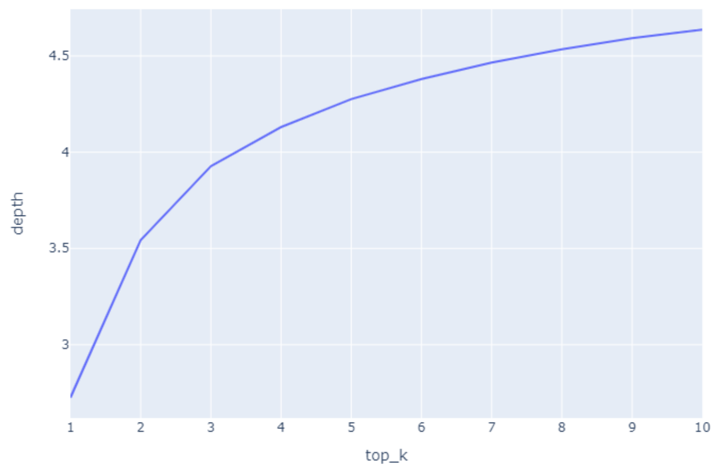


Fig. 7. top-m depth (y-axis) as m (x-axis) increases

product description embeddings over time to detect new clusters of products and refine the product taxonomy (product catalog) in the face of constantly changing user preferences as new products are introduced.

REFERENCES

[1] P. He, X. Liu, J. Gao, and W. Chen, "Deberta: Decoding-enhanced bert with disentangled attention," in *2021 International Conference on Learning Representations*, May 2021. Under review.

[2] S. R. G. J. Lake, Brenden M and J. B. Tenenbaum, "One shot learning of simple visual concepts.," *n Proceedings of the 33rd Annual Conference of the Cognitive Science Society*, vol. 172, 2011.

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[4] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," pp. 1532–1543, Oct. 2014.

[5] R. Kiros, Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler, "Skip-thought vectors," *CoRR*, vol. abs/1506.06726, 2015.

[6] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," vol. 1, pp. 539–546 vol. 1, 2005.

[7] A. Thyagarajan, "Siamese recurrent architectures for learning sentence similarity," 11 2015.

[8] P. Neculoiu, M. Versteegh, and M. Rotaru, "Learning text similarity with siamese recurrent networks," 01 2016.

[9] H.-F. Yu, C. Ho, P. Arunachalam, M. Somaiya, and C. Lin, "Product title classification versus text classification," 2012.

[10] Z. Kozareva, Q. Li, K. Zhai, and W. Guo, "Recognizing salient entities in shopping queries," pp. 107–111, Aug. 2016.

[11] J.-W. Ha, H. Pyo, and J. Kim, "Large-scale item categorization in e-commerce using multiple recurrent neural networks," 08 2016.

[12] Y. Xia, A. Levine, P. Das, G. Di Fabbrizio, K. Shinzato, and A. Datta, "Large-scale categorization of Japanese product titles using neural attention models," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, (Valencia, Spain), pp. 663–668, Association for Computational Linguistics, Apr. 2017.

[13] K. Shah, S. Kopru, and J.-D. Ruvini, "Neural network based extreme classification and similarity models for product matching," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*, (New Orleans - Louisiana), pp. 8–15, Association for Computational Linguistics, June 2018.

[14] D. Chicco, *Siamese Neural Networks: An Overview*, pp. 73–94. New York, NY: Springer US, 2021.

[15] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018.

[16] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," 2019.