E-commerce Product Ranking

Multimodal Machine Learning

VICTOR KLÖTZER, AI Engineering student at the University of Passau
MOHAMMED AL-MAAMARI, AI Engineering student at the University of Passau
EVREN CAN, AI Engineering student at the University of Passau

In 2021, global retail e-commerce sales totaled 4.89 trillion USD, with e-commerce revenues expected to reach 6.38 trillion USD by 2024 [4]. Such rapid expansion portends a bright future for the global e-commerce business, indicating a healthy market and rising client demand. The majority of e-commerce and retail businesses meet their goals by utilizing the power of data and increasing sales through the use of search and recommending systems on their websites. To contribute to this expanding trend, we worked on various multimodal models to assist effective semantic understanding. The objective is to improve the retrieval and recall of products given a query, and this is done by scoring the product images using relevance between queries and images. The used data is provided by Taobao online shopping platform in the KDD Cup 2020 Challenge, [3]. Toward this goal, we achieved an overall score of 54.1% on the normalized Discounted Cumulative Gain among the top 5 products (nDCG@5), which is used as the evaluation metric.

1 INTRODUCTION

With the continued expansion of the e-commerce industry, the creation of an e-commerce platform faces new obstacles due to new businesses and exponential increase in data. To improve the effectiveness of their search and recommendation systems, the great majority of e-commerce and retail organizations have implemented various data analysis and mining algorithms. Multimodal semantic comprehension is critical in this process. A high-quality semantic understanding model can help the platform better comprehend the demands of consumers and return more relevant items in response to their requests, hence improving the platform's service quality and user experience.

In this connection, the KDD Cup held a multimodal recall competition in 2020, which was named *Modern E-Commerce Platform: Multimodalities Recall*. The objective of this challenge is to find the five most relevant products for a given user query, and these five best products must be selected from a pool of candidate products.

Figure 1 shows a simplified example of the scenario. The query *"red Chinese coat"* is entered by a user and the pool of three products shown in the Figure is available. A model has then to deduce that the image on the left is related to the query based on the query's semantic information, the product in the middle is slightly relevant, and the one on the right is unrelated to the query.

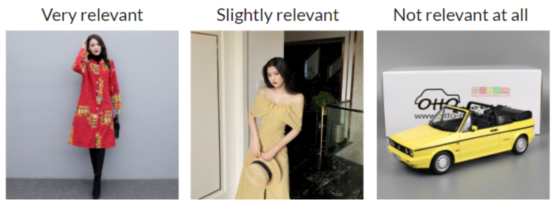


Fig. 1. Query: "red Chinese coat"

The challenge is thus a multimodal recall task, which in this case is a text image matching problem, as shown in the example. In that regard, we want to rate the query product pairs by relevance after training a multimodal model, and then sort them to determine the final recall list of the most related query product pairs.

To address this problem, we first focused on the preprocessing of the given text and image data in order to retrieve the most important information. And then we came up with two different modeling approaches, which in fine gave us performances of 54.1% and 49.3% on the nDCG@5.

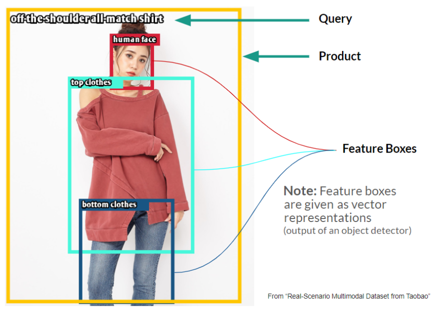
We preferred to come up with our own ideas instead of applying existing advanced winner solutions in order to understand the underlying concepts, and challenge ourselves since we are students, not competitors.

2 DATASET

This competition's data is derived from user queries and products in the real-world Taobao platform, and is divided into three parts: training set (`train`), validation set (`valid`), and test sets. There are two test sets: `testA` and `testB`, depending on the competition timeline, `testB` being used to finally rank the competitors. The actual sample data does not include visual graphics, and only a few photos are provided for reading and comprehension purposes.

The `train` set consists of 3 million related pairs of queries and product images that can be taken as positive examples to train multimodal models. In terms of data, the following points should be considered:

- Each of the four provided datasets consist of 9 columns, which are the *product id*, the *weight* and *height* of the product image, the *number of boxes* contained in an image, the *position* of those *boxes* in the product image, the *feature* representation of each box, the *labels* associated to every box, the *query* and the *query id* [2]. For instance, in Figure 2, the query is "off-the-shoulder all-match shirt", there are three boxes containing the main items of the product image, their labels are "human face", "top clothes" and "bottom clothes", and each of these boxes has a vector representation called *feature*.
- In addition to the 9 columns, for each query in the `valid`, `testA` and `testB` set, a candidate pool of around 30 products is prepared for the model evaluation. These three datasets contain respectively 500, 1000 and 1000 queries.
- Using fast RCNN, the event organizers retrieved boxes of objects from all images and transferred them into 2048 dimensional vector representation, the so-called boxes *features*. As a result, there is no need to incorporate image feature extraction in the model, but on the other hand, there is no visual image information we can use.
- Despite having 33 classes mentioned in the competition, only 14 are represented in the data. Moreover, one of these classes associated with the label "others" is over-represented, with 48% of the products assigned to this class, see Figure 3. Since the provided classes are unbalanced, that led us to use Latent Dirichlet Allocation (LDA) [5] to recreate balanced groups of similar observations.


off-the-shoulder all-match shirt
human face
top clothes
bottom clothes
Query
Product
Feature Boxes
Note: Feature boxes are given as vector representations (output of an object detector)
From "Real-Scenario Multimodal Dataset from Taobao"
Fig. 2. Representation of a product

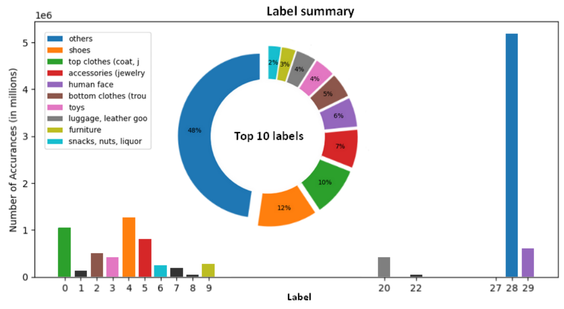


Fig. 3. Labels distribution

3 PERFORMANCE EVALUATION

We want to create a model that, given a query, is able to predict and rank the five best matching products out of a candidate pool of about 30 products. More precisely, for a given query, the prediction is an ordered list of five products. Since this is a specific prediction format, an appropriate evaluation metric is used, namely the *normalized Discounted Cumulative Gain* at rank 5, or nDCG@5. This metric gives a grade to each of the five predicted products for a query: if the predicted product is a ground-truth product, it gets a positive grade; otherwise, it gets a grade of 0. Additionally, the higher the rank of a correctly predicted product, the higher its grade. For a query q of a test set $\mathcal{S}$, the nDCG@5 of an ordered list of five products is computed as follows:

$$nDCG@5 := \frac{DCG@5}{IDCG@5} \text{ with } DCG@5 := \sum_{i=1}^{5} \text{grade}_i = \sum_{i=1}^{5} \text{rel}_i \cdot \frac{1}{\log_2(i+1)}$$

where $\text{rel}_i = 1$ if the i^{th} ranked product is ground-truth, $\text{rel}_i = 0$ otherwise, and $IDCG@5$ is the Ideal DCG@5. $IDCG@5 = \sum_{i=1}^{\min(5,x)} 1/\log_2(i+1)$ if for query q there are x ground-truth products in the test set $\mathcal{S}$.

Here are some nDCG@5 values that can be used as reference points. As for each query, there are about 30 candidate products, and most of the time 6 of them are ground-truth, a random prediction of five products gives an nDCG@5 of about 20%. Moreover, the baseline of the challenge was given at 55.4% on `testB`, and the winner team got 84.8%.

4 DATA PREPROCESSING

4.1 Product Images

Every product image was preprocessed into a certain number of vectors of size 2048, each representing a box in the image. For a neural network, dealing with inputs of varying sizes and which have no sequential order (the boxes of an image have no meaningful order) seems inconvenient, thus we thought about different ways to fix the size of the feature representing a product image.

A first idea is to concatenate all the 2048-vectors together by fixing, e.g., 10 vectors to concatenate: for a product, if there are more than 10 vectors, then we only take the first 10 or we randomly pick 10, and if there are fewer than 10 vectors, then we complete the blanks with zeros. A choice of 10 can be considered as fairly reasonable because among the 3 million products in `train`, more than 92% of product images have less than 10 vectors of size 2048, i.e., less than 10 boxes. Nevertheless, this idea is not used by a model presented in this report, because the computation time with $10 \times 2048 = 20480$ size inputs was too long, and no decent results were obtained.

Another idea was to merge the 2048-vectors together into only one vector of size 2048. To do so, we used two different methods: either summing up all 2048-vectors of a product image, or taking the element-wise maximum values. We haven't noticed any difference in using one or the other method. Even though both methods are pretty crude, they seem to nicely summarize the information into a single 2048-vector, since all the 2048-vectors of the boxes are sparse.

From now on, we refer to this one 2048-vector representing a product image as *product image* for the reader's ease.

4.2 Query Embedding

We preferred leveraging a pre-trained BERT model [6] to map queries to a 384-dimensional vector space since it provides a good vector representation of a sentence to be later incorporated into our multimodal models. The BERT embedding we apply is used for tasks like clustering or semantic search and allows to summarize textual information into one simple vector representation. In our case, this will be useful since similar queries will have similar embeddings, thus making it easy to compare them.

5 MODELING APPROACHES

To solve this multimodal problem, which deals in particular with images and texts, we have proposed two different modeling approaches, having in common the computation of a matching score for each query-product pair. The first idea consists in using both the query text and the product image as input to a neural network that directly outputs the aforementioned score. The second idea is to only input the product image into a model that transforms it into a mimicked query, which can then be compared to a real query to produce a score.

In both approaches, the scores of the query-product pairs are used to predict and rank the top-5 matching products among the 30 products given for each query in the test sets.

5.1 Model using both image products and queries as input

There are many possible ways to deal with multimodal data, but finding the proper method depends on the problem at hand. In our case, we have two modalities, one is the text query which we convert to an embedding vector using BERT, the other is the product images which are given as vector representation. And the goal is to build a system that sorts a pool of products from the most to the least relevant products to the given query. Our way to handle this problem is to construct a model that uses a query, then iterates through the pool of products and gives each of them a score indicating its relevance to the query. Next, we sort the products by their scores and return the top five related products to the given query. This approach mainly consists of two parts, the scorer and the sorter, the latter is straightforward and needs no more explanation, but the former will be thoroughly explained.

Having two types of inputs usually requires a model with two input layers; this is what we fundamentally used for our first method, where we worked on a simple fully connected neural network. Secondly, we tried to train a model that merges the two inputs by concatenating them. Lastly, for our final model, we built a neural network that is complex enough to capture the information needed to score the products and get a good nDCG@5. Additionally, we developed three techniques that boost the model's performance. After we settled with an architecture for the used model, we started optimizing some hyper-parameters such as the activation functions, optimizers, and other hyper-parameters. Finally, some post-processing is performed to improve the performance.

5.1.1 Negative Sampling: For this first approach, the model has to learn whether the product and the query it receives as inputs are related or not. In the data provided by the competition there are only positive samples, i.e., query product pairs that are considered to be perfectly matching. Therefore, if our model would only learn from these data points, it will be a dummy model that considers any query product pair to be related. Hence, for our model to learn what unrelated queries and products are, we need to feed it with negative samples, i.e., query product pairs that are not matching at all. Those are not provided in the dataset, thus we have to create them.

As mentioned before, the used dataset was too big to the point that we could not handle it all simultaneously. Thus, we started by using a 10K chunk and gradually increased the chunk size until we eventually used one million instances, one million positive samples used then to create the negative samples and to finally make a dataset of positive and negative query product pairs.

Another faced problem was the class "others". The unbalance that this class creates caused us to find another way of labeling the products, which is using LDA in order to analyze the queries. LDA is a topic generation model that uses the input documents to build a distribution of topics with parameter T being the number of topics to specify, and for each topic, it builds a distribution of words, with w_i being the probability of each word to occur in each topic t_j. In our case, we have 3 million queries, and for each of these queries, we want to know the topics it belongs to. Using a grid search, we found that the optimal number of topics is 62. Utilizing LDA gave us uniformly distributed topics for the queries. These topics are then used to help create the negative samples.

The used dataset only contains positive samples, but one of our modeling methods needs negative samples to give good results; that is why we generated negative samples ourselves. To do so, we followed the next steps:

(1) Train an LDA model to generate 62 topics.
(2) For each query, produce its *topic probability vector*, which is a vector of size 62 that contains the probability of a query to be in each topic.
(3) Give each topic a weight, then take the dot product between the *topic weights vector* and *topic probability vector* to obtain a scalar *query topic weight*.
(4) Sort the dataset in the order of the *query topic weights*.
(5) For each query q, find its negative candidates by selecting all the queries p with $|w_q - w_p| > \alpha w_q$, where w_i is the *query topic weight* of a query i and for the *difference ratio* α we used the value 0.8 (see Figure 4).
Then, select all the similar queries that have similar weights to the current query: the similar queries of q are all the queries p such that $|w_q - w_p| < \beta w_q$, with the *similarity ratio* β equals 0.3.
(6) Finally, for each positive sample, choose some negative samples from the pool of negative candidates.

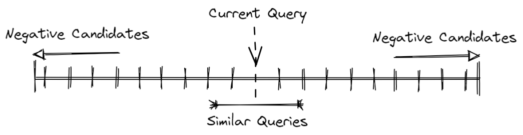


Fig. 4. Candidates weights

5.1.2 Baby Steps, starting with simple architectures: First, we started by building a simple model as a baseline, consisting of two branches of layers, one for the query vector representation and the other for the product image. Each one of these branches is made up of a sequence of simple dense layers. Then, we concatenate the outputs of these two branches and pass the concatenated vector through some other dense layers, finally having an output layer of one node that indicates the score of relevance between the query and the image. This simple model got a result of 35% nDCG@5.

The second architecture is even simpler, yet, it performs better than the first one. As its input, it only uses the concatenation of the query and the product image vectors. This input is fed to a bunch of fully connected layers with a one-node output as the last layer, and this eventually gave an nDCG@5 of 41%.

5.1.3 Common Learning Layers and Circulated Weights: The third and most promising model is slightly more complex than the previous ones, see Figure 5 for the used architecture. One should note that it is mainly composed of three fully connected parts: one for the query, another for the product image, and the third one merges what is learned from the two previous parts. Finally, we concatenate the output of these three parts into a vector and feed it to fully connected layers that eventually give the output score.

Furthermore, to prevent overfitting, and besides using dropout layers, we developed a novel technique that we call *Circulating Weights* (CW). This technique connects similar-shaped consecutive layers together by letting the weights of those layers circulate during the training. More precisely, during the training, chosen groups of connected layers repeatedly exchange their weights after a certain number of epochs. Using this technique, layers are less likely to overfit to certain features, and within a group of connected layers, the learned information is shared. Hypothetically, CW could also help solve the problem of slow learning that appears when layers are too far away from the output (vanishing gradient). We think it could prevent this issue because when the weights are circulating, they move from downstream layers closer to the output towards upstream layers nearer to the input.

Another technique that we also utilize is *Periodic Changing Hyper-Parameters* (PCHP) which allows to periodically change the value of a hyper-parameter during the training. For example, using PCHP on the batch size, we can define a list of batch sizes (e.g., [256, 512, 1024]) to use periodically, thus allowing to have a small batch size that induces more exploration in the early stages of the training, and to have a larger batch size by the end of the training, leading to more exploitation. In such a manner, we try to deal with the balance between exploration and exploitation. Regarding the nDCG@5, after using the previous techniques on the best model, we get a score of 53.2%, and this score is even boosted up to **54.1%** using ensembling. Figure 6 shows the performance of the three used architectures.

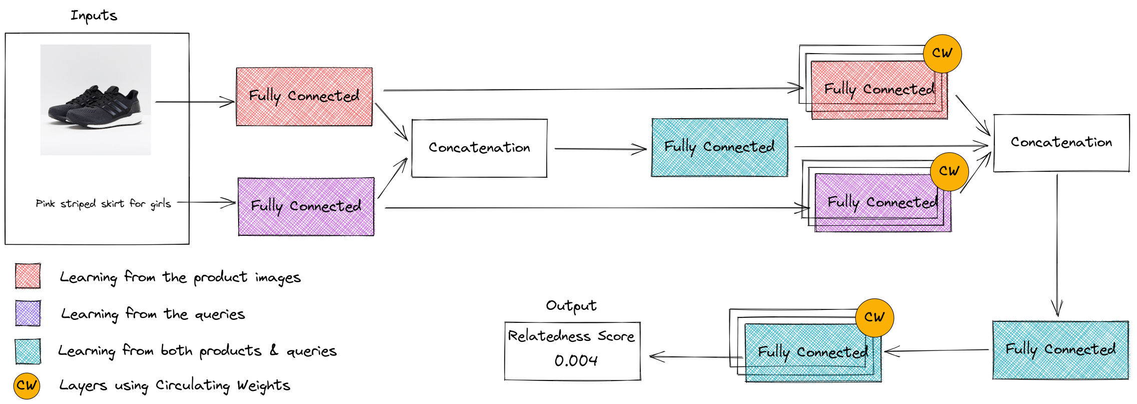


Fig. 5. Model 3: Multi-input, CW

5.1.4 *Hyper-Parameter Optimization and Ensemble Learning:*

Focusing our efforts only on the architecture is not enough; thus, we conduct some hyper-parameter optimization. First, we compared eight different optimizers, and Adamax is the most promising. Subsequently, we compared the model's performance using different loss functions, where Poisson performed slightly better than MSE, and MAE was inferior. Another hyper-parameter we optimized is the learning rate, where we tried different values and found out that a value around 0.001 is the most suitable. Finally, comparing activation functions showed that ReLU outperforms all the other tried functions for this modeling approach. Figure 6 displays the above-mentioned comparisons.

Related to ensemble learning, we came up with the idea of looking at all possible model combinations in order to make use of the predicted scores of different models. First, we compute the nDCG@5 for every single prediction. Then, we look at all the possible combinations of the predictions, and compute a weighted average using the nDCG@5 as the weights. Finally, we use the combination of model scores that gave the best nDCG@5.

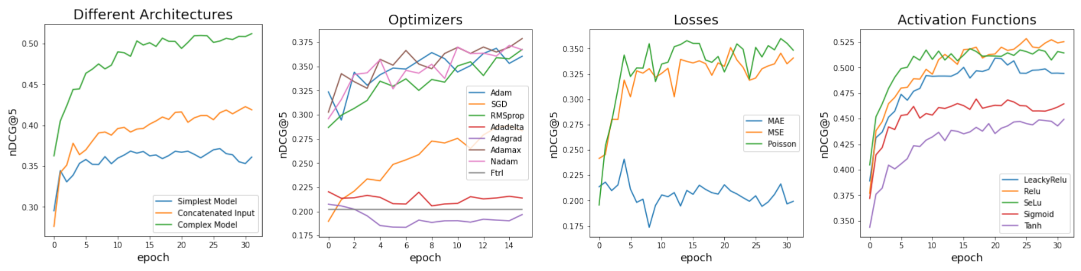


Fig. 6. From left: Scores of different models, optimizers, loss functions, activation functions

5.2 Model using encoding

The second approach uses a slightly less intuitive modeling than the first one, illustrating that a problem can always be tackled from different perspectives. The main challenge and goal of this approach is to find a way to transform a product image into a query-like representation. As presented above, first we use a pre-trained BERT model, which has been trained such that two similar sentences yield two similar vector representations, to preprocess queries and create embeddings of size 384. Since we want to produce a query-like embedding to compare to a real query embedding, the use of BERT seems appropriate because for both mimicked and real query, the goal is to extract query-specific information. Figure 7 shows the general idea of this modeling.

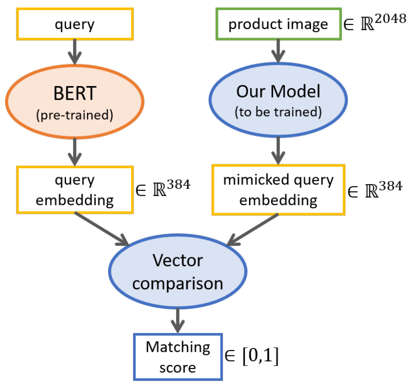


Fig. 7. General idea

5.2.1 *Direct encoding of the image.*

Firstly, we attempted to create a neural network that encodes a product image directly into a query representation, i.e., one network is trained to transform a vector of size 2048 representing an image into a vector of size 384 that mimics a query embedding. The inputs are the product images, and the ground-truths to be predicted are preprocessed embeddings of the queries of the `train` set, which are considered to be perfectly related to the associated images. When training the network, the loss is therefore computed using these query embeddings as the optimal query embedding to create from the associated product image.

Then, in order to predict the five best matching products for each query, a matching score is computed for each query-product pair. This score is obtained by calculating the cosine similarity between the BERT embedding of the real query and the mimicked query embedding computed by the model for each product image:

$$\forall \text{ real query embedding } e \in \mathbb{R}^{384}, \text{ mimicked query embedding } \hat{e} \in \mathbb{R}^{384} \ : \ \cos_{\text{sim}}(e, \hat{e}) := \frac{e \cdot \hat{e}}{\|e\|_2 \|\hat{e}\|_2}$$

We tried various network architectures, and after tuning different hyper-parameters, none of the tried networks gave descent results on the `valid` set, with the best nDCG@5 obtained being about 28%, only 8% better than a random prediction. The main problem is certainly that it is too complex for a single neural network to determine the key features to recreate a 384-length query embedding. Using much larger networks with more parameters than the one we tried, and also using more data could probably help to get better results. Nevertheless, this approach still seems quite complex for a single network. This is why we came up with the idea of separating the work of the model and facilitating the learning by integrating the query embedding understanding into another neural network, which is trained beforehand.

5.2.2 *Query representation dimensionality reduction.*

Secondly, we want to ease the learning of the model by combining two different networks. One of them compresses a query embedding while making sure to memorize the query semantic information. And the other one converts a product image into a vector of low dimension, which is intended to be an easier task than predicting a 384-long query embedding.

This separation of labor has a main advantage compared to directly converting a product image into a query embedding. It eases the task of the product encoder because it has to predict a small vector that contains little or no semantic information about the query. Indeed, the product encoder does not need to recreate a tangible BERT embedding because the embedding information is now handled by the query autoencoder, see Figure 8.

The first network is a *query autoencoder* and will be used to create *query codes*, see Figure 8. More precisely, this network is trained to recreate the same BERT embedding query that it receives as input. It is composed of a *query encoder* that takes as input a 384-long query embedding and compresses it into a query code of low dimension, and of a *query decoder* that must learn to reconstruct the query embedding from the query code produced by the query encoder. The query autoencoder is not intended to be used as such, but the query codes in the middle, which are smaller query representation, and the query decoder, which contains query understanding, are the two components we are actually interested in. For the query code we choose a size of 32.

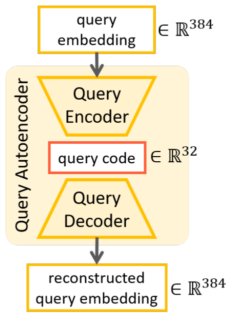


Fig. 8. Query autoencoder

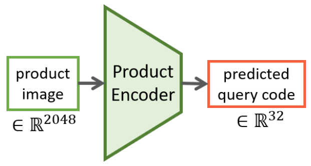


Fig. 9. Product encoder

The second neural network, called *product encoder*, makes use of the query codes produced by the query encoder. Actually, this network is trained to predict a query code given a product image as input. Therefore, it transforms a vector of size 2048 representing a product image into a vector of size 32 representing a query, see Figure 9.

The last step is to combine those two neural networks in order to recreate a model that takes a product image as input and outputs a mimicked query embedding, see Figure 7. For this purpose, the product encoder first predicts a query code from the inputted product image, and then, this predicted query code is reconstructed to a mimicked query embedding by the query decoder, see Figure 10.

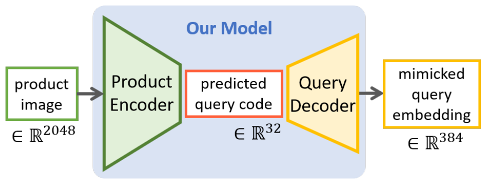


Fig. 10. Our model

5.2.3 Training, parameter tuning and performances. The first step in this modeling approach is to build the query autoencoder. To make it as accurate as possible, we used all the queries from the training data (out of the 3 million data points, there are about 1.3 million different queries). After fine-tuning various hyperparameters, we obtained the following architecture:

- query encoder: 384 $\longrightarrow$ 256 $\longrightarrow$ batch norm, tanh $\longrightarrow$ 128 $\longrightarrow$ batch norm, tanh $\longrightarrow$ 64 $\longrightarrow$ batch norm, tanh $\longrightarrow$ 32
- query decoder: 32 $\longrightarrow$ 64 $\longrightarrow$ batch norm, tanh $\longrightarrow$ 128 $\longrightarrow$ batch norm, tanh $\longrightarrow$ 256 $\longrightarrow$ batch norm, tanh $\longrightarrow$ 384

No additional ground-truth data was needed for this self-supervised machine learning, as the input to the network is also the object to be recreated. For the loss function MSE was applied, as it is often the case with autoencoders. One can also note the use of tanh activation functions, which gave the best results we observed. We tried to standardize the input data but this did not improve the training. On the other hand, adding batch normalization gave better results.

The second step is to train the product encoder, for which the ground-truths to predict are the query codes outputted by the trained query encoder, see Figure 9. The query encoder was fairly well trained, but after all it did not obtain a loss of zero. Thus, its inaccuracy is retained when using its outputs as ground-truths for the product encoder; it biases the model. Various hyper-parameters have been tried for the product encoder using 200,000 data points, see Table 1.

By combining a product encoder with the previously trained query decoder, see Figure 10, we can create a mimicked query embedding for each product image, and compare these vectors to real query embeddings with cosine similarity

Hidden dimensions	Batch norm	Activation function	Validation loss	nDCG@5 on `valid`
1024 - 256 - 128 - 64	Yes	tanh	0.219796	47.1037 %
1024 - 1024 - 1024 - 64	Yes	tanh	0.217748	46.6326 %
512 - 512 - 512 - 64	No	tanh	0.220597	46.5442 %
512 - 512 - 64	Yes	tanh	0.217830	46.4708 %
1024 - 1024 - 1024 - 64	Yes	ReLU	0.217870	45.9250 %
1024 - 512 - 256	Yes	tanh	0.219825	45.8849 %
2048 - 1024 - 512 - 256 - 128 - 64	Yes	tanh	0.217727	45.8821 %

Table 1. Hyper-parameter results for the product encoder, the nDCG@5 are obtained by adding the query decoder

as we did in 5.2.1. Finally, we predict and rank the five best matching products for each query in the `valid` dataset and then compute the nDCG@5. As shown in Table 1, the best nDCG@5 obtained here is about 47.10% on the `valid` set and a similar value is then obtained on the `testB` set, 47.78%.

Last, to further improve the result, we ensemble different models of product encoders, i.e., slightly different network structures that also have different initial weights. We simply compute the average scores of the assembled models and select the combination giving the best nDCG@5 on the `valid` set. We eventually used the six top models of Table 1, obtaining an nDCG@5 of **49.26%** on the `testB` set.

6 DISCUSSION

In our journey with multimodal learning we broadened our horizons and discovered various ways to handle such problems, for example, we utilized new deep learning architectures such multi-input networks, shared weights, auto-encoders, and other different networks. Additionally, we developed and implemented some new techniques like Circulating Weights, Periodic Batch-sizes, and weights shuffling, some of which worked surprisingly well, while some others did not work as planned. Next we discuss the encountered limitations during our work on the project, then we list the possible improvements that could be done to have better results.

6.1 Limitations

Since we work with a huge dataset of size 120 GB and we train deep neural networks, it requires a big amount of memory and computational power that an average computer can not handle. Thankfully we were given access to the university server, which provided us two Tesla K80 GPUs. Although, we couldn't use the complete dataset due to limited computational resources, particularly because of too long training times. Instead, we used some chunks of the data.

After a point, when using the first modeling approach, the loss kept reducing effectively, but the nDCG@5 was no longer increasing accordingly because it is not perfectly correlated to the loss function that we used. Indeed, if it exists, we didn't find a way to use a loss that exactly solves the problem that we deal with, i.e., maximizing the nDCG@5.

The BERT embedding allows to grab textual context information, but since the English queries provided in the data were translated from Chinese by the challenge organizers, the vector representation might lack precision.

6.2 Improvements

A first modification that would be interesting to test is to increase the amount of data used for training. In our models, we only use a maximum of 200,000 data points, which represents a bit less than 7% of the data given to us.

For the second approach, even though the last obtained nDCG@5 does not beat the one gotten with the first approach, the result is sufficiently far from a random prediction and close enough to the 54% obtained above to be interesting. Notice in particular that the network architectures used in the second approach are much simpler than in the first one and that no additional helping data such as the negative samples were used. This leaves room for improvement.

More generally, a concept that would be interesting to work more on is to help the models to extract the characteristics of their input more easily. The negative sampling used for the first modeling approach was already a nice way to try to help in this respect and could certainly be further improved. Also, for the decoder networks used in the second approach, using noisy images as input could help to learn more about important features. Furthermore, adding additional information about the product images, such as the number of boxes, their sizes and positions, and also using the created LDA classes as inputs, could certainly help the model to grab more knowledge and thus to better understand the inputs and then produce better outputs. Another way to deal with this information extraction problem could be to find a nicer way to preprocess the product image features in order to get a more concise representation of them, trying to reduce the vectors of size 2048 which are pretty sparse. An autoencoder could be used there to compress the information of a product image.

Besides, weights sharing, i.e. having two different layers of a network using the same weights simultaneously, is a technique we did not go into deeply but that was used by some contestants in the competition.

Last but one, generating a low dimensional representation of concatenated 10 features vectors with autoencoders, could avoid information leakage rather than summing 10 of them in a vector.

Finally, the ensemble methods we used were quite simple, so there is certainly room for improvement. For example, one could train a meta-model to ensemble the results of the two very different modeling approaches we have presented in this report.

6.3 Related Work

To broaden our horizons, we had a look at the solutions of other contestants. The most notable technique was the approach of the winner team (WinnieTheBest [8]) that scored 84.8% by implementing two different model architectures which are Modular Co-Attention Network (MCA) [9] and VisualBERT [7]. The second-highest score 84.3% was attained by the MDTP_CVA team [1], who chose the latest Vision-and-Language Pre-training algorithm based on a transformer to construct the main body of the model, and added text image matching as the downstream task.

7 CONCLUSION

This paper describes our own approaches to the Multimodalities Recall competition proposed by Tianchi, which served as a project topic for the Applied AI Lab course at the University of Passau. For this purpose, we came up with two modeling approaches that deal with multimodality in different ways, yet, both obtained similar and interesting results. Furthermore, we investigated processing text queries with Latent Dirichlet Allocation to produce proper negative samples. We also experimented some techniques such as circulating weights and periodic changing hyper-parameters that were used to increase the performance of our models. Moreover, ensembling methods helped to make even better predictions. After all, we got very close to the baseline of the competition by only using our own ideas and techniques, and we firmly believe that further improvements could lead to even higher results. In conclusion, we learned a lot by exploring new techniques, and we will keep working on them even after this project. We particularly found out that using autoencoders might give interesting results; hence, there is undoubtedly space for research. In a nutshell, it was an enlightening experience and a nice discovery of the multimodal world where lots of secrets still have to be discovered.

REFERENCES

[1] 2020. KDD Cup 2020 Challenges for Modern E-Commerce Platform: Multimodalities Recall. https://github.com/zuokai/KDDCUP_2020_MultimodalitiesRecall_2nd_Place

[2] 2020. KDD Cup 2020 Challenges for Modern E-Commerce Platform: Multimodalities Recall-Competition introduction-Tianchi competition-Alibabacloud Tianchi. https://tianchi.aliyun.com/competition/entrance/231786/information

[3] 2020. KDD Cup 2020 Challenges for Modern E-Commerce Platform: Multimodalities Recall-Tianchi competition-Alibabacloud Tianchi. https://tianchi.aliyun.com/competition/entrance/231786/introduction

[4] 2021. Global retail e-commerce market size 2014-2023. https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/

[5] 2022. Latent Dirichlet allocation. https://en.wikipedia.org/w/index.php?title=Latent_Dirichlet_allocation&oldid=1067732125 Page Version ID: 1067732125.

[6] 2022. sentence-transformers/all-MiniLM-L6-v2 · Hugging Face. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

[7] Liunian Harold Li, Mark Yatskar, Da Yin, Cho-Jui Hsieh, and Kai-Wei Chang. 2019. VisualBERT: A Simple and Performant Baseline for Vision and Language. *arXiv:1908.03557 [cs]* (Aug. 2019). http://arxiv.org/abs/1908.03557 arXiv: 1908.03557.

[8] WinnieTheBest. 2020. KDD CUP 2020: Multimodalities Recall. https://github.com/steven95421/KDD_WinnieTheBest

[9] Zhou Yu, Jun Yu, Yuhao Cui, Dacheng Tao, and Qi Tian. 2019. Deep Modular Co-Attention Networks for Visual Question Answering. *arXiv:1906.10770 [cs]* (June 2019). http://arxiv.org/abs/1906.10770 arXiv: 1906.10770.

PERSONAL CONTRIBUTIONS

Victor Klötzer

During this project, I actively participated in the brainstorming sessions during which we discussed a lot of ideas from each other, trying to explain them, find their limits and eventually improve them. I took some interest in creating negative samples, using a cruder method than the one presented in this report, but which did not yield better results. My main focus was on the second modeling approach presented in this report: starting with simple networks until I came up with the idea of the query autoencoder and the product encoder. Finally, of course, I also worked hard on the presentations and the report.

Mohammed Al-Maamari

In the course of this lab, my team and I had many meetings in which we discussed various ideas, distributed the tasks between us, and on many occasions sat to develop solutions to faced problems. I started with exploring the dataset and making different analyses and visualizations. Then, utilized LDA to create the negative samples. After that, I started implementing various neural networks until I had the one presented in this report. Finally, to boost the performance of the used model, I implemented and used the circulating weights CW, exploited periodic changing hyper-parameters PCH technique, performed some hyper-parameter optimizations, and lastly used different ensembling approaches such as finding the best combination of models to ensemble.

Evren Can

With my team, we always kept our excitement high and worked hard throughout the semester. I personally contributed with brainstorming and giving/getting feedback during the meetings, helping with preparing some parts of the presentations and the final report, and implementing text embedding for the text queries. In addition, I tried to preprocess data in order to get the best representation of it and tried to implement autoencoder and CNN models mainly on the top of the first approach, but those techniques didn't give promising results as much as the other approaches.