Siamese Neural Networks for Detecting Complementary Products

Marina Angelovska
KTH Royal Institute of Technology
angelovs@kth.se

Sina Sheikholeslami
KTH Royal Institute of Technology
sinash@kth.se

Bas Dunn
bol.com
bas@dunn.nl

Amir H. Payberah
KTH Royal Institute of Technology
payberah@kth.se

Abstract

Recommender systems play an important role in e-commerce websites as they improve the customer journey by helping the users find what they want at the right moment. In this paper, we focus on identifying a complementary relationship between the products of an e-commerce company. We propose a content-based recommender system for detecting complementary products, using *Siamese Neural Networks (SNN)*. To this end, we implement and compare two different models: *Siamese Convolutional Neural Network (CNN)* and *Siamese Long Short-Term Memory (LSTM)*. Moreover, we propose an extension of the SNN approach to handling millions of products in a matter of seconds, and we reduce the training time complexity by half. In the experiments, we show that Siamese LSTM can predict complementary products with an accuracy of $\sim 85\%$ using only the product titles.

1 Introduction

As much as the diverse and rich offers on e-commerce websites help the users find what they need at one market place, the online catalogs are sometimes too overwhelming. Recommender systems play a significant role in making this process convenient for users. A specific case for recommender systems is *complementary products* (also known as *add-ons*), which are the products that are sold separately but are used together, each creating a demand for the other. Figure 1 shows some examples of complementary products.

Detecting complementary products in many of the current platforms is mainly based on co-purchase history and business rules. In this approach, if two items have been bought together more than a certain number of times, they are assumed to complement one another with a high probability. However, complementarity among products

Figure 1: Complementary product examples.

cannot be accurately detected using only the purchase history because (i) identical items having different sizes or colors are likely to be bought together and are pure substitutes instead of complementary products (e.g., a user buys three flower vases in different sizes), and (ii) if there are no purchases made yet, the ground truth is missing (it is known as the cold-start problem). One solution to overcome these problems is to introduce human labeling for accurate validation. The problem of this approach lies in the time and scalability limitations. Moreover, these approaches focus on popular items; thus, unpopular (less frequently bought) products will stay undiscovered.

To address the aforementioned problems, we propose a supervised deep learning approach based on *Siamese Neural Networks (SNN)* (Chicco, 2021), and in particular *Siamese Convolutional Neural Network (CNN)* and *Siamese Long Short-Term Memory (LSTM)*. To train the model, we give the input dataset in the format of `MainProduct`, `AddOnProduct`, and `Label(Y/N)` that identifies if two products are complementary. Using this data, the SNN creates embeddings and generates vector outputs that show the actual distance between the two given products in terms of their complementarity. For each product, we consider the title, the

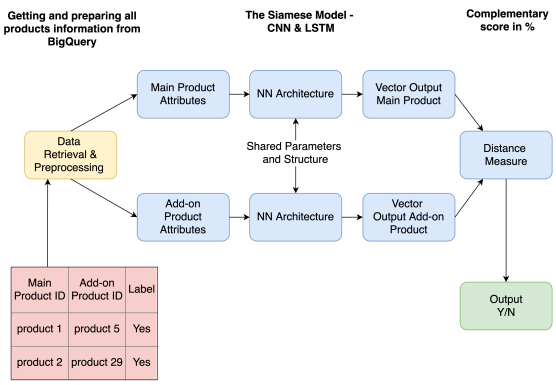

Figure 2: The proposed model pipeline using SNN.

description, and the brand as its attributes. Figure 2 shows the pipeline of the proposed model.

The objectives of this work are three-fold:

1. Studying the performance of Siamese CNN and Siamese LSTM in predicting complementary products using textual attributes.

2. Studying the impact of different product attributes (such as the product title, the description, and the brand) on predicting complementary products.

3. Transforming the problem into a K-Nearest-Neighbour (KNN) solution to predict complementarity among millions of products in a matter of seconds.

Our work builds upon the Siamese CNN introduced by Zhao et al. (2017) by comparing Siamese CNN and Siamese LSTM models and showing how Siamese architecture can be transformed to handle massive data. Through the experiments we show that Siamese LSTM outperforms Siamese CNN in predicting complementary products using textual product attributes with an accuracy of $\sim 85\%$. We also observe that among different attributes of products, the product title produces results with a higher accuracy. Moreover, we show that we can extend the proposed Siamese LSTM approach to a KNN problem that reduces the time complexity by half. The source code of our model is available on GitHub[1].

2 Preliminary

In this section, we briefly present the SNN architecture and explain how it works. SNN (Chicco, 2021) is a twin neural network (NN) composed of two separate NNs sharing the same architecture and the same weights, with no limitation on the NN architecture (Figure 2). In other words, SNN is an NN architecture capable of learning similarity between data samples by receiving pairs of samples and analyzing the differences between their features to map them to a multidimensional feature space (Martin et al., 2017). By receiving two different inputs, the main goal of such networks is to develop similarity knowledge between the two produced outputs.

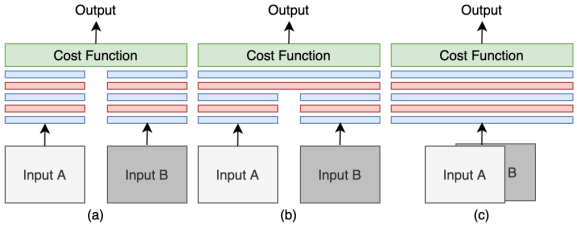

Figure 3: Three types of SNNs (a) late merge, (b) intermediate merge, and (c) early merge (Fiaz et al., 2019).

Fiaz et al. (2019) categorize SNNs in three groups based on the time of merging the layers: *late merge* (LM), *intermediate merge* (IM), and *early merge* (EM), which are shown in Figure 3. In LM, the output vectors of each network are merged at the last dense layer. IM suggests to merge the outputs of the two networks in the middle of the network and process them as one output in the last layers. In EM, the two inputs are merged right before the actual network, resulting in a single-like NN architecture.

One of the benefits of using SNN is its scalability. It processes each data sample once and then computes each pair's compatibility score, which results in a significantly lower complexity than iterating through the whole model for each pair of products. In a real-life scenario, we are usually given target products set $Q = \{q_1, q_2, \cdots, q_n\}$ and candidate set for the add-ons $C = \{c_1, c_2, \cdots, c_m\}$ where n and m have values larger than 10^6, indicating a few millions of products. Thus, to train a NN, we need to create $n \times m$ pairs of products to make input samples to the NN. However, usually, we are interested in the top k candidate add-ons for a given target product, and SNNs enable us to do so.

3 Method

We now discuss the network architectures used in our Siamese CNN and Siamese LSTM models.

[1] https://github.com/marinaangelovska/complementary_products_suggestions

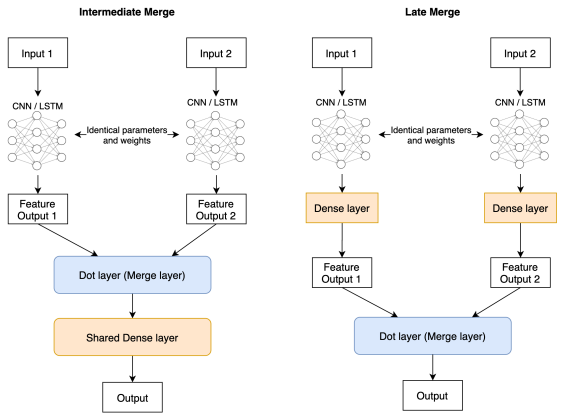

Figure 4: The difference between IM and LM in the implementation of the proposed model.

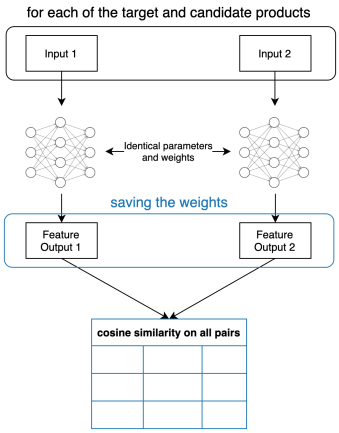

Figure 5: The place where we save the weights in the SNN architecture.

The input to these networks is the pair of any two product attributes (e.g., title and description).

CNN Architecture. The CNN network has 11 layers. The first layer is an *embedding* layer to create embeddings for each of the words in the product's attributes. The output dimension is set to 300, so that each word will be represented in 300 different features in the multi-dimensional space. Then, the *ZeroPadding1D*, the *Conv1D*, and the *MaxPooling1D* layers follow one after each other. We repeat these three layers by only reducing their filter length and pooling size. Finally, after flattening we use a *dense* layer with 100 neurons and ReLU activation. We use the *dot layer* to combine the two products from the Siamese input and compute their similarity. By normalizing the input given to the *dot layer* we compute the cosine proximity between the products.

LSTM Architecture. The LSTM network has seven layers in total. Like the CNN architecture, the *embedding layer* is the first layer with the output dimension of 300. The *LSTM* layer with 150 neurons and ReLU activation is the core part of this pipeline that learns the sequential characteristics of the words in the product titles or descriptions. After the *flatten* layer, for the same reasons as in the CNN architecture we use *dot* layer to merge the two inputs and obtain their similarity score. Then, we have a *dense* layer, which is a fully-connected layer with 100 neurons.

In both CNN and LSTM models, we use the Sigmoid activation in the output layer for the binary classification problem. Figure 4 illustrates the difference between IM and LM in our implementation. In IM, we apply the *dot* layer right after the *flatten* layer, meaning that one *dense* layer is available after the merging and before the final output layer. LM represents the architecture when the *dot* layer is implemented right before the final output layer. For comparing both Siamese architectures (IM and LM), we apply exactly the same layers just in a different order.

In both CNN or LSTM implementations, the Siamese approach can efficiently find the top K most complementary products for a given product over massive data (Martin et al., 2017). For a target product q and a candidate add-on product c from the sets of Q and C, we first generate their vector representations. However, we are only interested in the weights that the network produces before applying the dot product. The Siamese setup treats each product separately until the merge point in the model, thus for each of the products in sets Q and C, we can get the weights as shown on Figure 5. This means that the embeddings part is done only once for each product separately.

Once we have the vector representations X_Q and X_C for each product from Q and C, respectively, we can compute the similarity. From this point on, we have a KNN problem. Then, we save those weights in the forms of matrices and apply the normalized dot product between the two matrices having the weights for each target and candidate product (we calculate the *cosine similarity*), representing their complementarity.

4 Experiments

In this section, we first explain the dataset and the preprocessing step to make it ready to be given to

the models, and then present the experiments.

4.1 Dataset

To train the models, we use manually labeled data points from an e-commerce company[2]. We consider the product title, description, and brand as the attributes for each product. We get pairs of positive matches for each product, which has at least one add-on, meaning that if a product has multiple add-ons, it will appear multiple times as the `MainProduct` in the dataset. Also, a product might be an add-on for multiple different main products.

Initially, there are 18346 pairs of complementary products. We assume that two products are non-complementary if they were never bought together and are not included in our initial dataset. Moreover, we want each product to have the same number of positive and negative samples, so that the model is able to generalize well. We achieve this by iterating through the add-ons list, and for each add-on, we make sure that we generate as many negative samples as there are positive.

For example, given `Product5`, which is an add-on to 10 different products, we find another 10 products for which product `Product5` will not be an add-on. After making sure that we have $50-50$ ratio in terms of the labels for each add-on in our dataset, we repeat the same process for the main products. In the end, we end up having 60442 total pairs of products, out of which 35978 are unique products.

Before giving the data to the models we: lowercase all letters, remove punctuation signs, digits, measurements (e.g., cm, m), and stop words. We observe that excluding the digits from the products in the dataset improves the performance by 10%. We consider 80% of the dataset for training the models and the rest for testing them. We also use 10% of the training data for validation during training. To make sure that the model's performance is calculated on new unseen data, we use *Group Shuffle Split* so that each product that will appear as an add-on in the train set will not appear as an add-on in the test set. We train the embeddings using Word2Vec (Mikolov et al., 2013) before the embedding layer in the models. Word2vec was trained using the whole corpus of titles in the category of interest. Once we have the embeddings for each of the words in the corpus, we add those weights to the weights parameter in the embedding layer.

[2]Due to the company's policy we do not reveal the company's name. It is an e-commerce company that offers various products in multiple categories.

Table 1: Comparative results showing the performance of Siamese CNN and Siamese LSTM based on the place of merging the two product outputs.

Siamese model	AUC	Accuracy
CNN - IM	72%	65%
CNN - LM	82%	78%
LSTM - IM	93%	85%
LSTM - LM	80%	75%

4.2 Results

Before conducting the experiments, we measure the impact of the merging location in the two Siamese models. Table 1 shows that Siamese CNN performs better with LM. However, Siamese LSTM performs better with IM and outperforms all other models' architectures, thus that is the architecture we will use in the rest of the experiments.

We first compare the performance of Siamese LSTM with three frequently used methods: *Random Forest (RF)*, *Single LSTM* network, and *Vanilla NN*. The RF baseline, which is used in Martin et al. (2017), combines the inputs from each sample in the dataset and tokenizes the product titles using *Bag of Words representation*. The single LSTM is the second baseline we consider.

The main difference between the single and the Siamese LSTM is in the way the input is processed. In the single LSTM model's input, we concatenate the two products' attributes in the form of `MainProductTitle_AddonProductTitle`. On the other hand, in the Siamese approach the two input products are treated separately until the moment of merging the two vector outputs. This enables us to later transform the Siamese approach to a KNN model. Lastly, we also implement and test a vanilla NN with six layers: *input*, *emedding*, *flatten*, *dense*, *dropout* and *output* layer. The *dense* layer has 100 neurons and ReLU activation.

Figure 6 shows the accuracy and AUC score for each of these models. Although Siamese LSTM and the Single LSTM perform with the same accuracy, Siamese LSTM can be transformed to a KNN model and used with massive data. To pair 13000 unique products from the test dataset with each other, we would get roughly $170M$ pairs of products for which we want to know their complementary relationship. However, in this work, due to the hardware limitations, we only take $1M$ pairs

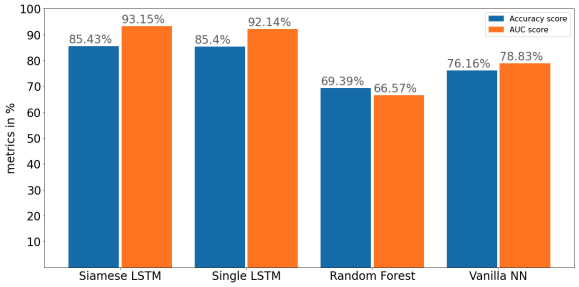

Figure 6: Comparative results showing the accuracy and AUC for Siamese LSTM, Single LSTM, Vanilla NN and Random Forest.

Table 2: Comparing the time needed for predicting complementarity among $1M$ pairs of products and the time complexity for creating the embeddings in the NN.

Model	Prediction time	Time complexity
Single LSTM	11min	$O(N^2)$
Siamese LSTM	8min	$O(N^2)$
Transformed Siamese LSTM	10sec	$O(N)$

of products in the following experiments. Both, the single and Siamese LSTM network are traversed $1M$ times, once for creating the embeddings for each pair of products.

The Siamese LSTM has a slightly better time performance due to the ability to learn faster. In the transformed Siamese LSTM, we calculate the embeddings for each product only once by using the Siamese LSTM model and we save those embeddings before the *dot* layer. This means that the Siamese LSTM will be traversed only 13000 times, once for each product. Once we have these embeddings for each of the 13000 products, the complementarity score for the $1M$ pairs is calculated very fast, as we do simple matrix multiplication operation using *cosine similarity*.

Here we use cosine similarity as it is basically a normalized dot product. If we were using single LSTM approach, we would not have been able to achieve this because in that setup we cannot get embeddings for a single product, but only for a pair of products. Table 2 shows the time analysis for the three approaches, where N represents the number of unique products.

Using the transformed Siamese LSTM (the KNN approach), we compute the complementarity score between all possible products from the test set within seconds. Table 3 shows the complementarity score (cosine similarity) of the top five add-ons

Table 3: Complementarity score for five add-on suggestions using the the initial Siamese LSTM model and the transformed Siamese LSTM.

Target product:	Suggested add-on products				
	Transformed Siamese LSTM	83%	72%	69%	68%	66%

Table 4: Comparing accuracy, AUC score and training time for Siamese LSTM using different product attributes when the training was done on 10 epochs.

Product attribute(s)	Accuracy	AUC	Training time
Title	85%	93%	13min
Title + Description	89%	95%	58min
Description	72%	81%	58min
Title + Brand	80%	83%	14min

suggested by the KNN approach for a randomly selected product. The third and fifth products from Table 3 are newly detected add-ons, while the second product is a correctly detected add-on already present in the ground truth. The first and fourth products are substitutes to the target product, hence false positives. Our KNN approach suggests substitute products because, in some cases, in the ground truth, the add-on products can be similar/substitute products to the target product. Ideally, we would not want to have this in our training set.

Table 4 shows the results from including different textual attributes (e.g., the title, the description, and the brand) in the Siamese LSTM. Although the description, as an addition to the title, increases the accuracy and AUC score, we conclude that speed-accuracy trade-off needs to be made since including the description slows down the training process for about four times.

5 Related Work

We split the available methods for measuring similarity and complementarity into two groups: *unsupervised* and *supervised* learning approaches.

One of the most common unsupervised learning methods using co-purchase history is the *Frequent Pattern (FP) Growth* (Han et al., 2004) algorithm. Other groups of research focus on using the paradigm of *Word2Vec* (Mikolov et al., 2013). Grbovic et al. (2015) propose a *Prod2Vec* model that learns product representations from sequences of past orders by considering the purchase sequence

as a sentence and products within the sequence as words. The *Meta-Prod2Vec* model by Vasile et al. (2016) extends the *Prod2Vec* model by taking into account products' metadata. The *BB2Vec* model (Trofimov, 2018) eliminates the cold-start problem by using browsing and purchase session data, and is a combination of several *Prod2Vec* models.

Zhao et al. (2017) introduce the Siamese CNN approach, which this work is based on. Other supervised learning approaches focus on image data, text attributes, or both. *SCEPTRE* is a model introduced by McAuley et al. (2015), and its main goal is topic modeling using Latent Dirichlet Allocation (LDA) (Blei et al., 2003) and edge detection of related topics. Zhang et al. (2018) suggest *ENCORE*, a three-step algorithm: (i) detecting the complementarity among products based on their embedding distances of image and text attributes, (ii) taking into account user preferences for detecting validity of each complementarity distance, and (iii) training a NN with the outcomes of the previous two steps (Yu et al., 2019). Kalchbrenner et al. (2014) explore Dynamic CNN (DCNN) for semantic modeling of sentences using CNNs.

6 Conclusion

In this paper, we present a supervised learning approach for complementary product recommendations. We take manually labelled pairs of complementary products from an e-commerce company and propose a scalable solution. To this end, we design and compare Siamese CNN and Siamese LSTM architectures to create embeddings for products' features and compute a complementarity score for a given pair of products. We conclude that Siamese LSTM outperforms Siamese CNN and its baselines. We show that the product title is the most valuable attribute. Lastly, we show that our model can be transformed into a KNN solution to handle big data scenarios.

This work can be extended by introducing user click history to include items that have been viewed in the same session (items which are very similar) in the negative training sample, thus teaching the model the difference between substitute and complementary relationships. Furthermore, including more product attributes (such as the price or subcategory) could improve the model's performance.

References

D. Blei et al. 2003. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022.

F. Vasile et al. 2016. Meta-prod2vec: Product embeddings using side-information for recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 225–232.

H. Yu et al. 2019. Complementary recommendations: A brief survey. In *2019 International Conference on High Performance Big Data and Intelligent Systems (HPBD&IS)*, pages 73–78. IEEE.

J. Han et al. 2004. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data mining and knowledge discovery*, 8(1):53–87.

J. McAuley et al. 2015. Inferring networks of substitutable and complementary products. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 785–794.

K. Martin et al. 2017. A convolutional siamese network for developing similarity knowledge in the selfback dataset. CEUR Workshop Proceedings.

K. Zhao et al. 2017. Deep style match for complementary recommendation. In *AAAI Workshops*.

M. Grbovic et al. 2015. E-commerce in your inbox: Product recommendations at scale. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1809–1818.

M. Fiaz et al. 2019. Deep siamese networks toward robust visual tracking. In *Visual Object Tracking with Deep Neural Networks*. IntechOpen.

N. Kalchbrenner et al. 2014. A convolutional neural network for modelling sentences. In *52nd Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

T. Mikolov et al. 2013. Efficient estimation of word representations in vector space. pages 1–12.

Y. Zhang et al. 2018. Quality-aware neural complementary item recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 77–85.

Davide Chicco. 2021. Siamese neural networks: An overview. *Artificial Neural Networks*, pages 73–94.

I. Trofimov. 2018. Inferring complementary products from baskets and browsing sessions. *arXiv preprint arXiv:1809.09621*.