Czech Technical University in Prag

Faculty of Mechanical Engineering

Department of Instrumentational and Control Engineering

Automation and Industry informatics

Master's thesis

Product categorization using machine learning

Supervisor: Ing. Matouš Cejnek Ph.D.

January 16, 2022 **Bc. Roman Dušek**

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Dušek Roman**
Personal ID number:	**466522**
Faculty / Institute:	**Faculty of Mechanical Engineering**
Department / Institute:	**Department of Instrumentation and Control Engineering**
Study program:	**Automation and Instrumentation Engineering**
Specialisation:	**Automation and Industrial Informatics**

II. Master's thesis details

Master's thesis title in English:

Product categorization using machine learning

Master's thesis title in Czech:

Kategorizace produktů pomocí strojového učení

Guidelines:

- make analysis of product data for purpose of feature extraction
- research suitable algorithms and approaches for product classification
- implement and test a selected approach on a real product data

Bibliography / sources:

[1] Jurafsky, Dan. Speech & language processing. Pearson Education India, 2000.
[2] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

Name and workplace of master's thesis supervisor:

Ing. Matouš Cejnek, Ph.D., U12110.3

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **29.10.2021** Deadline for master's thesis submission: **20.01.2022**

Assignment valid until: ______________

[illegible]
Ing. Matouš Cejnek, Ph.D.
Supervisor's signature

[illegible]
Head of department's signature

[illegible]
prof. Ing. Michael Valášek, DrSc.
Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

____
Date of assignment receipt

[illegible]
Student's signature

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Date: ....

Signature

Acknowledgement

Acknowledgment I wanted to thank everyone who supported me during the preparation and the work on my thesis. I also want to thank my supervisor doc. Ing. Matouš Cejnek PhD. for his insides and guidance. I also want to thank the Machine learning team at mall.cz for providing me with data for such an interesting problem and giving me the allowance of computing the models using Keboola jobs. Last but not the least, I want to thank my family for providing me with moral and material support during my studies.

Abstrakt

Špatně zařazené produktu nejen ovlivňují zákaznickou zkušenost na stránkách webových obchodů, ale také vytváří problémy při vytváření systémů pro doporučování a hledání. V této práci otestuji několik různých modelů založených na metodách strojového učení na získané informaci z textu a obrázků produktů. Modely jsem zvolil jak z řad standardních algoritmů strojového učení, tak i metody založené na hlubokém strojovém učení. Reprezentace textu a obrázků je získána za pomocí velkých modelů předtrénovaných, které jsou volně dostupné. Přístupy založené na kombinaci modalit byli také otestovány pro účely kategorizace. Experimenty provádím na reálných datech jednoho z největších webových obchodů v Česku.

Klíčová slova: Strojové učení, Deep learning, Konvoluční neuronové sítě, Seq2Seq modely, Multimodální modely

Abstract

Wrong categorization of products not only affects the customer experience on e-commerce websites but also creates problems for effective use of search and recommendation systems. In this thesis, I am going to test machine learning models based on extracted text or image representations. Models are based on ordinary machine learning algorithms, but also on deep learning architectures. Text and image representations were extracted using large pretrained models. Multimodal models were also tested for the categorization task. Experiments were done on real data of one of the Czech largest e-commerce website.

Keywords: Machine learning, Deep learning, Convolutional Neural Networks, Seq2Seq models, Multimodal models

Contents

1 Introduction

E-commerce grows fast and so is the number of products listed on an e-commerce website. To create the best experience for customers, e-commerce companies provide a structured category tree for easy navigation through the web. But with thousands of categories, it is hard for listers (people that add new products to the category tree) to choose the best fitting category. Each of the listers also evaluates each product differently, and therefore the assigned categories can differ for similar materials assigned by different listers. In recent decades, large e-commerce companies started providing so-called marketplace. This service provides other shops to sell their products on big e-commerce websites. From this comes another possible errors, because marketplace partners have almost no idea what should the correct category be. The wrong categorized products do not only affect customers that search products in categories, but it also affects other processes. For example, most search and recommendation system assumes that the products are in the correct category. Therefore, the correct working process of product categorization should be the foundation. In recent decades there have been several studies focusing on solving this task using machine learning. In my work I am going to try to solve this issue for one of the largest e-commerce companies in the Czech Republic, mall.cz. I am going to test selected models on example data and see how well can it perform. Data for each product consist of text and images. The reference category is specified by category id and its path.

2 Theory

2.1 Natural Language Processing

In this part, I cover how text can be represented for use in machine learning algorithms and what are the methods of extracting features from the text (also called embeddings).

2.1.1 Vector semantics and embeddings

Vector semantics is about representing words as points in a multidimensional semantic space. Words that occur in the same context have most likely also similar meanings, this hypothesis was formulated in the 1950s, and it is called the distributional hypothesis [45]. Vector semantics represent linguistic hypotheses by learning representations of the meaning of a word, called embeddings. There are two main kinds of embeddings, static and dynamic (contextualized). Static embedding does not change in a different contexts, these embeddings are created with models like Word2vec. On the opposite dynamic embeddings change in a different context, an example of this would be embeddings using a Transformer, using Bidirectional Encoder Representations from Transformers (BERT) model for example. Both static and dynamic embeddings are dense vectors, in comparison to traditional sparse vectors. These approaches for dense embeddings are based on representation learning, which is a self-supervised type of learning, that does not need labeled data, more on it in a section about transformers. Before talking about dense embeddings, I will review the basic approaches for representing text with matrices.

	As You Like It	**Twelfth Night**	**Julius Caesar**	**Henry V**
battle	2	15	3	13
good	50	0	5	89
fool	5	25	2	4
wit	10	5	12	3

Fig. 1: Example of term-document matrix [45]

Two of the most basic text representations are the term-document matrix (also known as a bag-of-words) and the term-term matrix [45]. These representations are types of sparse static embeddings. The term-document matrix counts the number of term occurrences in a document, this is used in basic information retrieval when searching for the best matching document from a number of documents given a query. This can be seen in figure 1. We can use these to compare documents in space, I will get to it in section 2.1.5, where I show visualized version of the term-document matrix in the figure 6. We can also look at this task from the other way, and find for a given word similar words by the context they appeared in. The term-term matrix creates a square matrix containing all the words in vocabulary on both axes. We count the number of times other words occur in the context of the target word (figure 2). The context is usually a couple of words to the right and the left from the target word. The problem with these vector representations is that they are sparse and quite big because the dimensions are defined by the size of vocabulary size or document size.

	aardvark	...	computer	data	pie	sugar
cherry	0	...	2	8	442	25
strawberry	0	...	0	0	60	19
digital	0	...	1670	1683	5	4
information	0	...	3325	3982	5	13

Fig. 2: Example of term-term matrix [45]

Different kinds of re-weighting can be applied to a term-document matrix representation for a more sophisticated representation. All of the re-weighting approaches correct the the problem of raw frequencies of the basic term-document matrix. Commonly in language, there is a lot of words that do not have any meaning, we call them stop words. It can also happen that there will be some very frequent words that do not give any information because they appear in each of the documents. These words will not help separate documents between each other. To fix it, we use Term frequency and Inverse document frequency (TF-IDF) re-weighting approach [45]. This approach lowers the weights for words that occur very frequently. TF is calculated as count of the given term in document divided by number of terms in document. IDF represents the number of documents in the collection N, divided by the number of documents in which the word occurs df_t. To squash both TF and IDF down, we transform them with a logarithmic function. Similar to the former methods, this method has also its use in information retrieval. In the figure 3 how TF-IDF, gets completly rid of word "good", since it appears in each of the documents.

$$\mathrm{tf}_{t,d} = \log_{10}(\mathrm{count}(t,d) + 1) \tag{1}$$

$$\mathrm{idf}_t = \log_{10}\left(\frac{N}{\mathrm{df}_t}\right) \tag{2}$$

$$w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t \tag{3}$$

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

Fig. 3: Example of TF-IDF matrix [45]

Rather than counting the number of times a word has occurred in a context, we can use a machine-learning algorithm to predict the probability, that a given word will occur in surroundings of other words (its context), this can be formalized as binary classification, it is formally called word2vec.

2.1.2 Word2vec

Word2vec is the most popular algorithm for learning short static representations of words. The length of the representation is usually 300 dimension (can be 50 or 1000). These dense representations work better as features for text classifier. Mostly, because the classifier does not require so many weights as in the case of previous algorithms, and with dense representations, it is easier to converge. It also does a better job at capturing the meaning of a word, for searching synonyms, for example. Since the embeddings are static, the words are going to have same meaning in a different context (for example, bank robbery opposite to river bank). This algorithm was discovered by Czech researcher Mikolov in 2013 [67, 66]. He and his team created two types of methods for extracting word embeddings (figure 4). The first of them is called Skip-Gram (SG), where we try to predict the target word given its context, where the context are words around it. The other method is Continous Bag of Words (CBoW), which works exactly the opposite. Given the target word, we try to predict the context. In the case of SG, it is a step-by-step process where for each target world we train the model for each word in the context, which can be words in some window around the target word. While with CBoW we concatenated, sum, or average the embeddings of context words while predicting the target word.

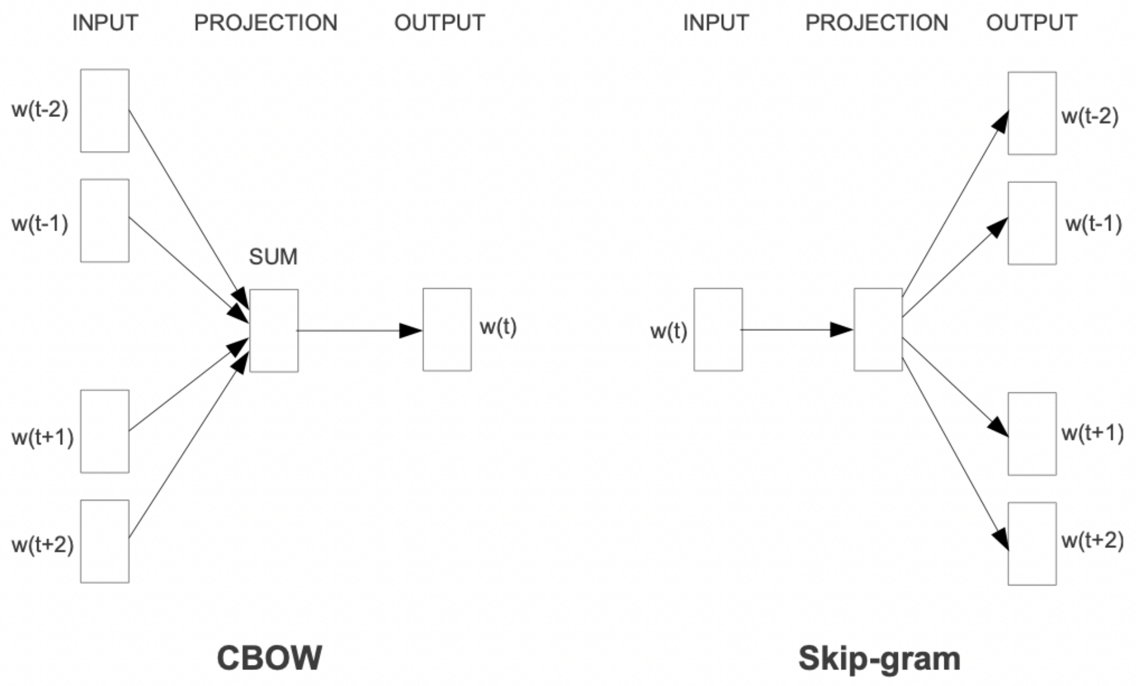


Fig. 4: Possible implementations of Word2vec [66]

This can be implemented with a shallow neural network or logistic regression, trained using backpropagation [45]. The great advantage of this type of learning is that we don't need labeled data. The only thing we require is a structured sequence of text or other data. This approach can also be used for non-NLP (Natural Language Processing) tasks like recommendation systems, where the sequence can be represented by songs that users listened to, in the case of Spotify for example. The basic algorithms got later improved by negative sampling that maximizes the training efficiency. It randomly samples tokens from the whole lexicon based on the unigram frequency. This tries to minimize the similarity of pairs that do not occur together and still maximizes the similarity of the context words. Since then, there have also been different variants, which I will talk about in next section

Further in [57] Doc2vec was introduced as an extension of Word2vec for documents (sentences, paragraphs). In that case unique input tokens, representing document, paragraph, or sentence, are added to context vocabulary and the algorithms update the embeddings of documents based on the words that are present in the document.

2.1.3 Word2vec alternatives

The most used word2vec alternative is Global Vectors for Word Representation (GloVe)[1]. This approach tries to build embeddings by capturing global corpus statistics. It combines the intuition of global matrix factorization a model like Positive Point-wise Mutual Information (PPMI) from [72] and also uses the structure of the local context window of a model like Word2vec.

Researchers came with another model that is robust to spelling errors and can also deal with unknown words that can occur in the inference time by taking advantage of subword information [34]. A library that implements this model is called fasttext[2]. It works by training embeddings on words represented as a bag of character n-gram, including the word itself. The n-grams are usually between 3 and 6 characters long. After the model had been trained, all the embeddings of a given word are summed to get the unique word embedding. [12, 65] The library provides pretrained models on 157 languages and also provides a text classifier, that can be trained. [44] Another way of obtaining embeddings is Bidirectional Recurrent Neural Networks or Transformer models. These models give us contextualized representation, therefore we call them dynamic embeddings since the embeddings change depending on the context. I will talk more about these models in a separate section.

2.1.4 Combining word vectors

The two most basic ways to combine words to get document embedding are taking a maximum in every dimension of the vectors or averaging all the embedding from given words. Fasttext implements a special method (`get_sentence_vector`) for creating

[1]https://nlp.stanford.edu/projects/glove/
[2]https://fasttext.cc

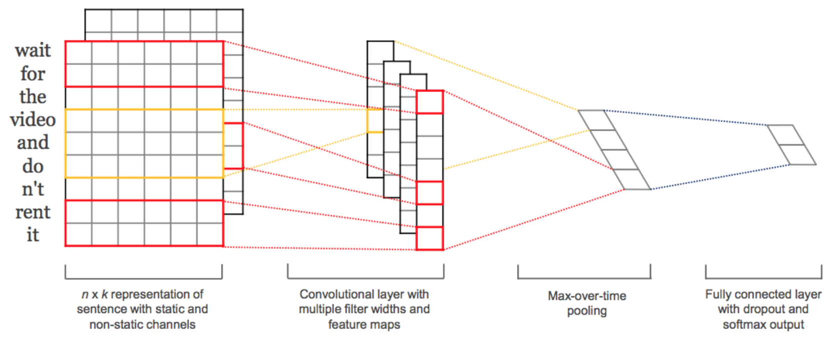

wait
for
the
video
and
do
n't
rent
it
n x *k* representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 1: Model architecture with two channels for an example sentence.

Fig. 5: Convolutional Neural Network applied for text classification from [48]

sentence embeddings. It first divides the vectors by its L2 norm and then takes average, it is called the average of unit vectors. There are other methods proposed in papers that can improve the performance of combined embeddings representation. One of them is based on taking the weighted average of words , which was introduced in [5]. The weights are estimated by the frequency of words in the corpus, the paper also goes into depth about how does it work in the case of CBOW, when the context words are being averaged. Some other approaches recommend weighing the word vectors based on the TF-IDF vector.

To build stronger sentence representations, we can also build another model on top of the embeddings. Many of such models were created, namely InferSent [24], Universal Sentence Vectors [15], Skip-through vectors [50] and other [63, 88].

Convolutional Neural Network (CNN) which will be presented later for image processing can also be used for text classification (figure 5), as shown in [48]. These types of models build different sizes of kernels along static embeddings, as seen in the figure 5. It can be found implemented from the original paper on github[3].

2.1.5 Similarity metric

The metric that is the golden standard for finding similar vector representations is cosine similarity [45]. This metric gives us the cosine of the angle between two vectors. Cosine similarity is mathematically represented as a normalized dot product of two vectors. It must be normalized because otherwise the vectors with large counts would be favored.

$$\text{similarity}(A,B) = cos(\theta) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{4}$$

In the figure 6, I show on term-document matrix from earlier, how the documents are related. In this case documents are represented with words occurring in them. If we would apply the

[3]https://github.com/cezannec/CNN_Text_Classification

cosine similarity, we would find out that the cosine between Henry V and Julius Caesar is very high and similar for the two other documents. But if we would compare Henry V and Twelfth Night, the similarity would be very low, as the angle is big.

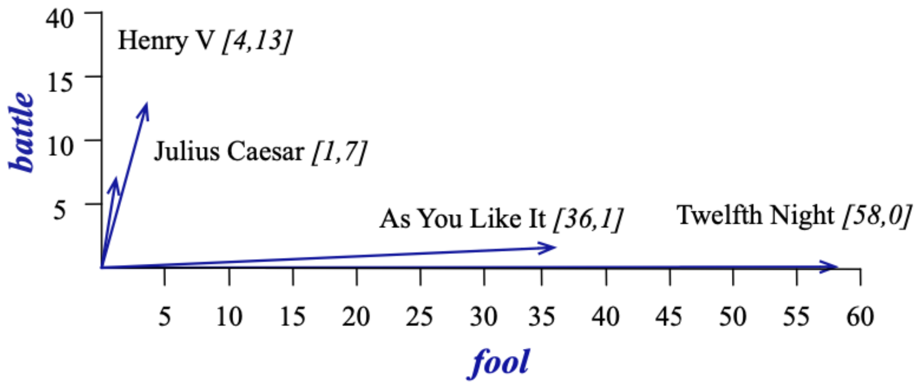
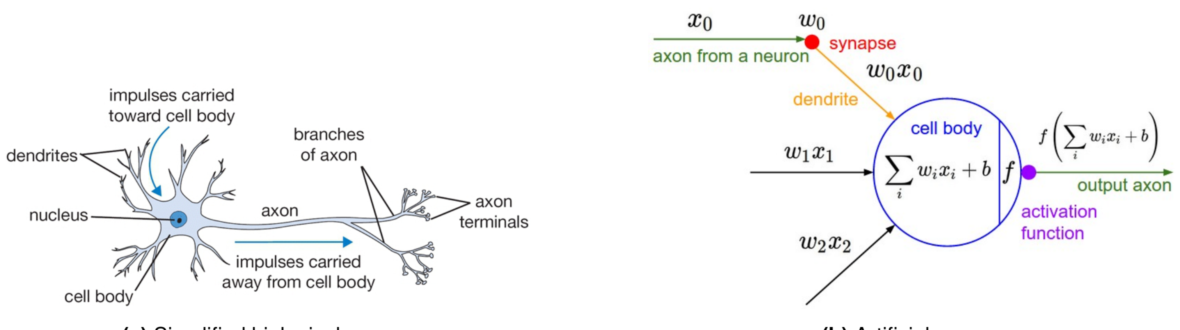


Fig. 6: Visualizing similarity of two documents based on word counts [45]

2.2 Deep Learning

This part provides the theory behind basic Feed Forward Neural Network (FFNN) and the way how they are trained. Much of this work is based on deep learning techniques that emerged in the 21st century, mainly due to computation power shortage, but actually, this branch of machine learning already started in the 1940s [2]. Origins come from McCulloch-Pitts neuron, which was a simplified model of the human neuron, which was described in propositional logic. Nowadays, we don't use neural networks that have anything to do with the biological structure of the human brain. The great advantage of neural networks is that they are, good at learning features from data themselves, and we don't need to spend much of the time on feature engineering. Another advantage of neural networks is that they are easily scalable to a given problem. The scalability lies in the breadth and width of the neural network architecture. Their simple construction of basic matrix operations makes them also easily parallelizable with modern Graphics Processing Unit (GPU).



(a) Simplified biological neuron

(b) Artificial neuron

Fig. 7: Comparison between biological and artificial neuron

2.2.1 Neural unit

The basic element of a Neural Network (NN) is a neural unit, also historically called a perceptron. It is a combination of a very simple linear model and non-linear activation function, similarly to General Linear Models (GLM) with a link function (logistic regression applied with sigmoid function). As in the case of Linear Regression, the model has weights for each of the inputs and also a bias term. You can see the resemblance of biological neurons with our artificial figure 7.

The non-linear activation function is very important because it allows approximating any non-linear function when combining multiple neurons. This would not be possible with a linear output of each neuron. Nowadays the most used activation functions are the sigmoid, the tanh, and the Rectified Linear unit (ReLu), which are represented in the figure 8. The first activation function used by McCulloch-Pitts was the sign function.

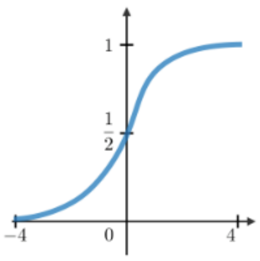
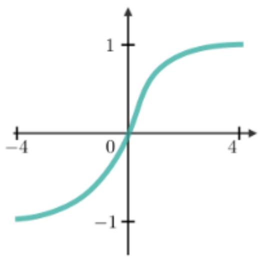
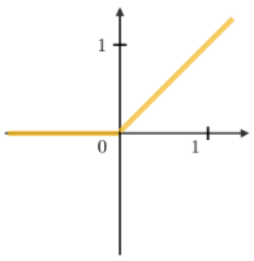
Sigmoid	Tanh	RELU
$g(z) = \dfrac{1}{1+e^{-z}}$	$g(z) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		

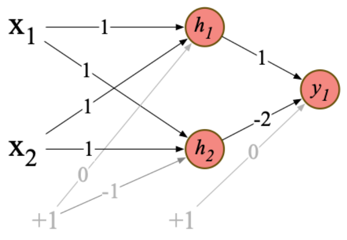
Fig. 8: Summary of most used activation functions [4]

As seen in the picture 8 on the left, the sigmoid function maps the output to the range of [0,1]. We usually only use the sigmoid function as the last layer for binary classification or if we need a layer that needs output between 0 and 1, an example of that would be the forget and update gate in the Long-Short Term Memory (LSTM) cell in Recurrent Neural Networks (RNN) [45]. Tanh is preferable used over sigmoid if the output of a neuron needs to be squashed into a certain range. It helps, because of the range, as the output can be either positive or negative. And also tanh has 4 times bigger maximal derivation, which helps faster backpropagation. Nowadays, almost every network is based on ReLu activations, if it is not a special case that I talked about. The advantages of ReLu are that they are easily computed and also differentiable. But it can also fail in some cases producing a dying ReLu problem. That is when most of the neurons output 0 and the network cannot be optimized further. There has been many modifications of the standard ReLu, but none of them ever performed way better than ReLu.

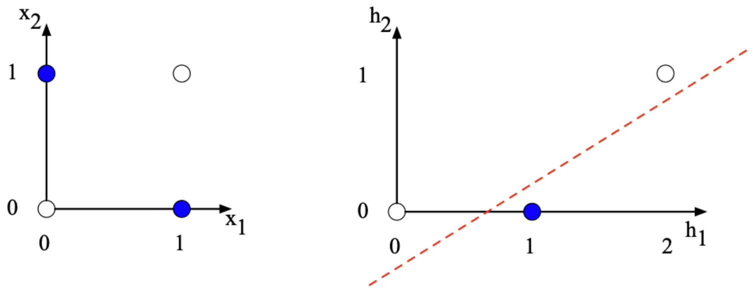
[4]Picture taken from https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

2.2.2 One hidden layer neural network

In 1969, it was demonstrated that a neural unit is unable to solve the XOR problem [45]. Back then it was not accepted that the model of the neuron can solve only linearly separable problems. To solve the XOR problem, we have to combine multiple single units as a hidden layer was used (figure 9a). Each of the neural unit takes in inputs, aggregates them, and applies activation function after. Then both of the values are combined into output. For the combination of neuron outputs, we also use weights that can be trained and the final layer uses the sigmoid function. What each of the neurons does is that it produces a representation of the inputs. Using two neurons we are able to create a two-dimensional space where the function is linearly separable, as seen in the picture 9b, which shows difference between input and hidden representations in neurons. This way we can combine more neurons into layers to be able to learn very complex non-linear functions, which leads us to FFNN.



(a) Architecture of simplest neural network that is able to solve XOR problem



(b) Comparison between inputs and hidden representation

Fig. 9: Architecture and visualization of representation of neuron [45]

2.2.3 Feed forward neural networks

FFNN is the most basic architecture of NNs used nowadays, historically it was known as Multilayer perceptrons (MLP) [45]. It is called feedforward because the input is passed through the NN layer by layer until the output is produced. Each of the hidden layers is formed of hidden units, same as I presented earlier in one hidden layer NN. All the layers are fully-connected as shown in the figure 10. We can represent weights of one layer with one single matrix $\mathbf{W}$ (and biases $\mathbf{b}$) and thus compute the outputs $\mathbf{h}$ of the hidden layers with simple matrix operations and transformation by activation σ.

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{5}$$

The type of machine learning problem, dictates a final layer that we usually have to use [45]. For a regression task, the final node will be typically an identity function, but can also be a probability distribution function. For binary classification the output can be transformed using sigmoid function, similarly to logistic regression, which outputs a number

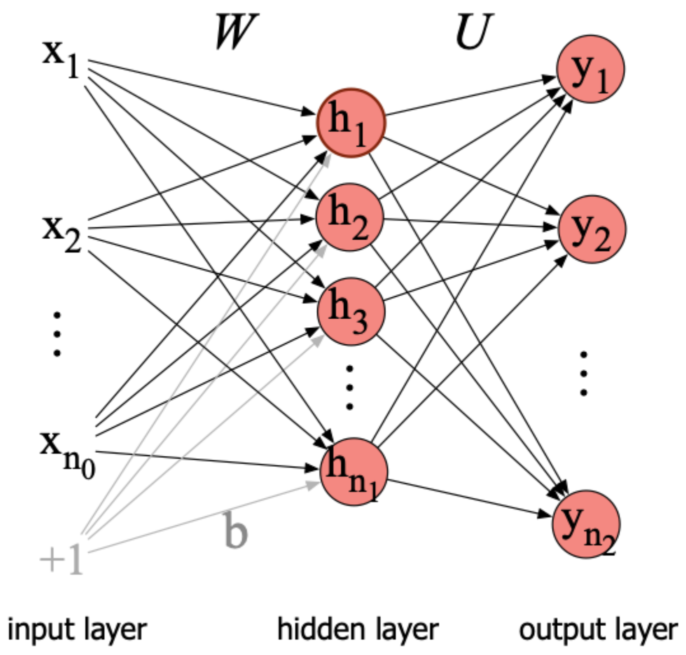


Fig. 10: Feed forward neural network with one hidden layer[45]

between [0,1]. For multi-label classification task, we use softmax function, which maps out probabilities for every label. It does so by normalizing the real numbers using:

$$\operatorname{softmax}(\mathbf{z}_i) = \frac{\exp(\mathbf{z}_i)}{\sum_{j=1}^{d} \exp(\mathbf{z}_j)} 1 \leq i \leq d \tag{6}$$

where z is vector of outputs of possible classes.

2.2.4 Training Neural Networks

The way a neural network learns is by minimizing the error, specified by a chosen loss function. The loss function computes the distance between the predicted value and the gold output. For regression problems, we usually use Mean Squared Error (MSE) or Mean Absolute Error (MAE).

$$\mathrm{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{7}$$

$$\mathrm{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{8}$$

In the case of binary classification, we use cross-entropy loss (also Log loss), as we do in logistic regression. For a multi-label classification, we use categorical-cross entropy.

$$L_{CE}(\hat{y}, y) = -\log p(y \mid x) = -[y \log \hat{y} + (1-y) \log(1-\hat{y})] \tag{9}$$

$$L_{CE}(\hat{y}, y) = -\sum_{i=1}^{C} y_i \log \hat{y}_i \tag{10}$$

After computing the loss, we can compute the partial derivative of the loss function with respect to each parameter, this is called backpropagation, which was first formalized by Rumelhart in [77]. To be able to backpropagate errors through a whole network of possible many hidden layers, we use chain rule from calculus. This allows us to split each of the partial derivatives into subderivatives. When the gradient for each weight is backpropagated, we update the weights using stochastic gradient descent, where η is learning, rate specifying the size of change. The updated weight is calculated as:

$$w_1^+ = w_1 - \eta \frac{\partial E_{\text{total}}}{\partial w_1} \tag{11}$$

. This is a basic way to optimize weights, there is plenty more like RMSprop, Adam, or Adagrad. These special optimizers use memory from the previous steps to compute the final gradient. The gradient descent can be further optimized by taking mini-/batches and updating the weights using the mean gradient of the batch, instead of iteratively be each example. We can also tune the learning rate using various schedulers, to further improve convergence, with methods like decay or warm-up. [45, 32]

2.2.5 Recurrent neural networks

This architecture allows for sequential data processing, which is great for time series data (financial stock market or weather forecast for example) or natural language processing data like voice and text data (translation, name-entity recognition or question-answering). These models are also being used in biology for DNA sequencing. The principle of RNN is that every input produces output, but also creates a hidden state, which acts as dynamic memory. [45] The hidden state is then forwarded to the next input in the sequence. Thus, the output of the next input is affected by previous hidden states. The initial hidden state for the first input is usually initialized as a vector of zeros.

The idea of RNN was first introduced by Rummelhart in [77] and first used for discovering semantic feature for words by Elman in [29]. RNN's weights are updated using backpropagation through time algorithm [101], which differs from regular backpropagations, because we have to backpropagate over all of the steps, but the the principle stays the same.

The basic RNN architecture adds the vector of hidden state from the previous step h_{t-1} to inputs of the current step x_t, then tanh is applied to transform the vector, which results in output, as seen in figure 11. Before the combination, they are both multiplied trainable weights W_h and U_h.

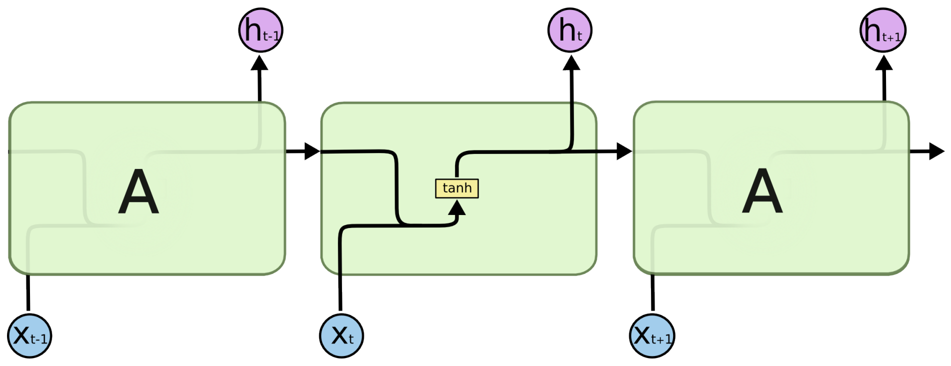


Fig. 11: Basic architecture of RNN

$$h_t = \sigma_h(i_t) = \sigma_h(U_h x_t + V_h h_{t-1} + b_h) \tag{12}$$

$$y_t = \sigma_y(a_t) = \sigma_y(W_y h_t + b_h) \tag{13}$$

This so-called "Vanilla" RNN architecture has a problem with exploiting and vanishing gradient [68]. Exploiting gradient happens, when too big a gradient accumulates over all of the steps we are backpropagating and then a weight is an update by a large margin. This can be prevented by gradient clipping, which doesn't allow the gradients to grow so big, the threshold is usually set at 1.0. The vanishing gradient happens when the gradient backpropagated through the network is too small, which results in small relations between each of the states, because the network can't hold the memory for so long. For the purpose of holding the dynamic memory over longer distances in the hidden state, new architectures of RNN were created. Namely LSTM and Gated Recurrent Unit (GRU). Both of these alternatives make use of gates, that decide which of the information is going to be kept in a hidden state.

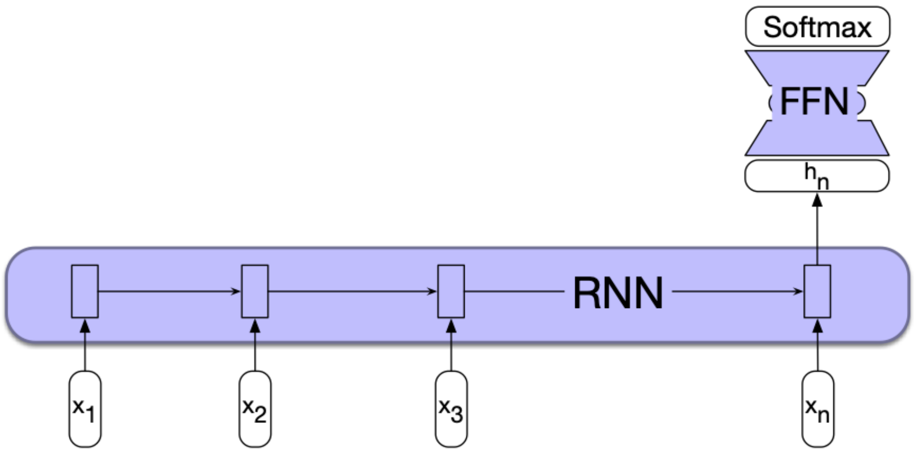


Fig. 12: Architecture of RNN for multi-class classification [45]

RNN can be used for different types of problems, depending on the number of inputs/outputs. One-to-many architecture is applied in music generation, where the model is able to generate a music piece from one musical note at the input. Many-to-one architecture is common for sentimental analysis, where the model takes text transformed to embeddings and outputs a label of a positive and negative class. This can also be applied in the case of multi-class classification when for example predicting the number of stars given from a reviewer by his written review. For more efficient classification, FFNN is placed at the end of this network either based on the last hidden state (figure 12), or as an average of all hidden states of the RNN. This architecture can be also used for regression, like a stocks prediction. Many-to-many can have two different modifications, the first one creates output in each of the steps, so the number of inputs is equal to the number of outputs. This is used in problems like Named-entity Recognition and can be also applied for autoregressive text generation. The second many-to-many architecture first encodes the input sequence, which creates a context (hidden state) for the other part that decodes the state and outputs the prediction. This type of architecture is called Encoder-Decoder, it is heavily used in NLP, for tasks like machine translation or question and answering.

Similar to FFNN, we can also stack multiple layers on each other to create bigger networks that can learn more relationships from the data [45]. The size of each neural network should be appropriate to the data, so we don't run to the problems of under-/overfitting. To take full advantage of the input sequence, a biRNN was proposed. These networks calculate the hidden states from both directions of the input sequence and then sum up the hidden state to create outputs. In real production, when used for NLP tasks, RNNs usually have multiple layers and use bidirectionally to use the full potential, we then call them stacked biLSTMs. [81] They also use attention mechanism, which I will talk about later.

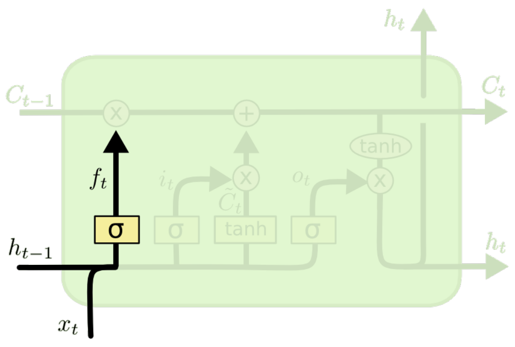
The big downside of RNN is that it is not possible to compute them in parallel due to their step-by-step processing, therefore they cannot take advantage of the most recent powerful GPU or Tensors Processing Unit (TPU) computational units. That is why they are being replaced by Transformers, which were created to be highly parallelizable, but for some smaller use cases the RNN will still be worth the try. I am going to use them in the practical part for translation like multi-label classification.

2.2.6 Long-short term memory

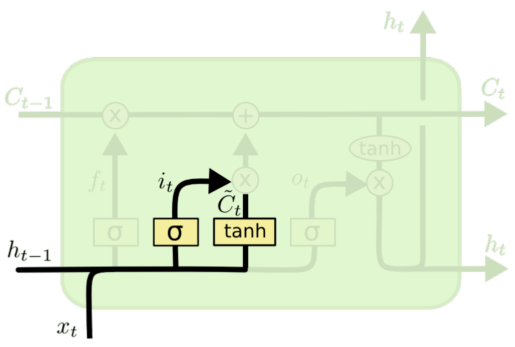
This model architecture was first introduced in 1997 by Hochreiter in [39]. It was explicitly designed to be able to learn long-term dependencies in sequences and therefore solved the vanishing gradient problem. The LSTM architecture has the ability to decide which one of the elements in the hidden state is going to be forgotten or updated. For this purpose, the LSTM has four separate layers combined with three gates.[5] [89, 45]

In the first step (figure 13a), the model decides which of the previous cell states are going

[5]Images for LSTM, GRU and figure 11 were taking from blogpost https://colah.github.io/posts/2015-08-Understanding-LSTMs/ as some of the explanations about LSTM and GRU



(a) Calculation forget vector



(b) Calculation of update vector

Fig. 13: First two steps of LSTM

to be kept, therefore this gate is called forget gate. It uses a sigmoid function applied to a sum of the last output h_{t-1} (also hidden state) and current input x_t to create a vector f_t with numbers between [0,1].

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{14}$$

$$\tilde{C}_t = \tanh\left(W_c \cdot [h_{t-1}, x_t] + b_c\right) \tag{15}$$

This vector then multiplies the previous cell state C_{t-1}. The state multiplied with numbers close to zero will be forgotten. In the next step (figure 13b), the model decides what is going to be stored. The first update gate is applied, to decide which of the states is going to be updated. Similarly to forget, the gate uses a sigmoid function to decide so, creating vector i_t.

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \tag{16}$$

Then a tanh function is applied to create and vector of the current cell cell state $\tilde{C}_t$. These two are combined and finally added to the last cell state C_t, which was already changed by the forget gate f_t. In the last step, the model decides what are we going to output.

$$o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right) \tag{17}$$

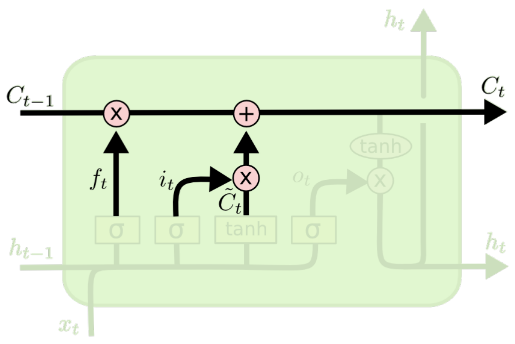
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{18}$$

$$h_t = o_t \odot \tanh\left(C_t\right) \tag{19}$$

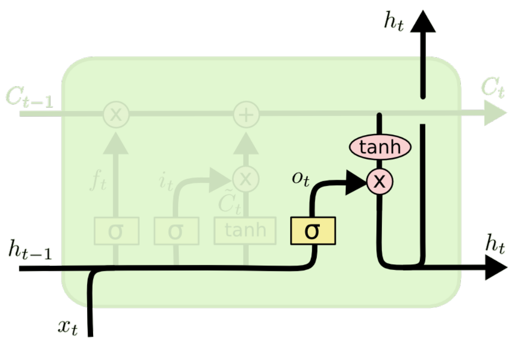
The output (figure 14b) is a combination of updated cell state transformed by tanh function, multiplied by the sum of last output and current input transformed by a sigmoid function o_t. To get the output vector, we multiply these two to get vector h_t.

2.2.7 Gated Recurrent Units

Gated Recurrent Unit (figure 15) is a simplification of LSTM. It was introduced by Cho in 2014 [19], applied to machine learning translation using Encoder-decoder model. It merges



(a) Forgetting and updating of last cell state



(b) Creation of output/hidden state

Fig. 14: Third and fourth step in calculating LSTM

input and forget gate into a single update gate and also removes the output gate. Since the output gate is removed, it only holds on the hidden state (last output vector), rather than both cell and hidden states as LSTM. This simplification makes this model smaller and therefore easier to train. But it also has downsides, models based on this architecture are usually worse in performance than LSTM. [22, 18]

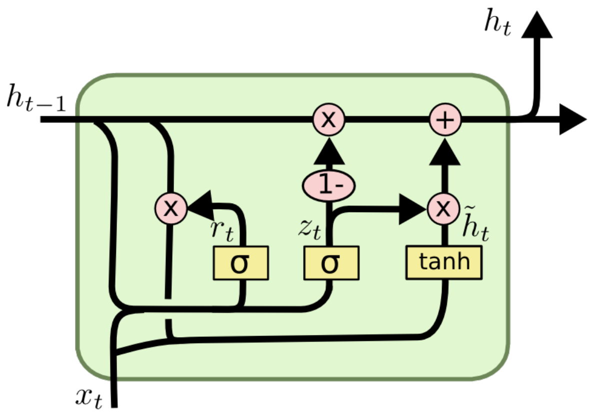


Fig. 15: Gated Recurrent Unit cell

2.2.8 Translation using RNN Encoder-Decoder

The disadvantage of FFNNs is that they can only work with fixed inputs and outputs, this made it hard to work with text, where every sentence has most of the time different length [45]. The models worked with windows, but it gave them poor context about the whole sequence. Sequence to Sequence (Seq2Seq) approach uses an Encoder-Decoder model to model sequences of different lengths[90, 18, 19, 46]. For know, I will now introduce the Encoder-Decoder models as RNN based, later I will also review the inner working of Encoder-Decoder in Transformer models. The task of the first RNN model is to obtain a large fixed-dimensional vector representation of the input sequence (h_n in the figure 16), usually called context. The second RNN (which is called a decoder) works as a language model that generates an output sequence given the context of an input sequence. The decoder always starts with the initial token (typically the start of the sequence, “sos” for short). When generating output, the model chooses the tokens with the highest probability based on the last softmax layer. The process is autoregressive, so the next input for the Decoder is the previously outputted token. To make it faster to train the model, we use a technique called teacher forcing. What it does is that during training we don’t take the next input from autoregressive generation, meaning taking the last predicted token, rather we take the correct token, that should be predicted, so the next step is trained on the correct word. For this, the input of the sequence during training is represented by the input sequence and correct output sequence separated by separator token. When the network read the separator token, it switches to decoding and creating outputs itself.

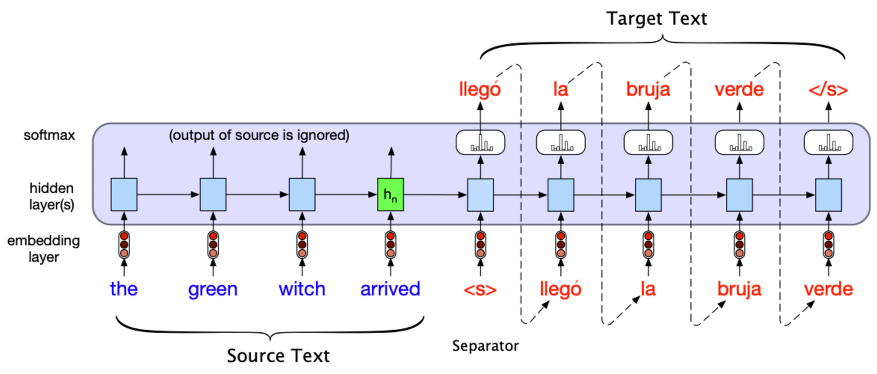


Fig. 16: Example of translating for English to Spanish [45]

The performance with this approach degrades rapidly as the length of the sentence increases since the decoder takes in the context from the last hidden state of the encoder (also called bottleneck) and updates it every decoding step, which overwrites the information from the original encoder representation [45]. This way, the information from the bottleneck may not be equally represented at the end of the sequence. To fix it we can make use of the context vector for each of the decoder steps since the hidden state relations from the encoder slowly degrades when the decoder makes each step. This can help model later in the sequence to produce outputs based on the encoder context. This lead researcher to come up with a so-called attention mechanism for RNN models, which does not only work with the last hidden state of the encoder but weights the importance of all encoder states. This approach was proposed in [6, 64].

2.2.9 Attention

As I wrote earlier, it solves the bottleneck problem. It is done by allowing the decoder to dynamically get information from all the hidden states in the encoder. We cannot simply use all the concatenated hidden states of the encoder together, since every input sequence is of a different length. So the attention produced a single fixed-length vector at each of the decoder steps, by taking a weighted sum of all the encoder hidden states $\mathbf{h}^e$ [45]. The weight represents how relevant a hidden state from the encoder is to the current step, meaning which of the token should the decoder focus on ("attend to"). To decide which of the encoder states is most relevant, we compute dot-product attention, for the previous decoder state h_{i-1} with all the encoder states.

$$\text{score}\left(\mathbf{h}_{i-1}^{d}, \mathbf{h}_{j}^{e}\right) = \mathbf{h}_{i-1}^{d} \cdot \mathbf{h}_{j}^{e} \tag{20}$$

$$\alpha_{ij} = \text{softmax}\left(\text{score}\left(\mathbf{h}_{i-1}^{d}, \mathbf{h}_{j}^{e}\right) \forall j \in e\right) \tag{21}$$

$$\mathbf{c}_i = \sum_{j} \alpha_{ij} \mathbf{h}_{j}^{e} \tag{22}$$

Resulting in a vector of scalars, it represents the similarity between the states. To create weights α for all the states, we use the softmax activation layer, which normalizes the scores into ratio between 0 and 1. This tells us the proportional relevance across each of the encoder states. Given the vector, we can compute the context vector c_i as a weighted average over all the encoder states. Then the current hidden state of decoder h_i is equal to combination of input $\hat{y}_{t-1}$ (output of last step), last hidden state h_{i-1} and context c_i:

$$\mathbf{h}_{i}^{d} = g\left(\hat{y}_{i-1}, \mathbf{h}_{i-1}^{d}, \mathbf{c}_i\right) \tag{23}$$

The process is also visualized in figure 17 To create a more sophisticated way of calculating the similarity between decoder and encoder state, we can add a set of weights to the dot-product attention, which is updated during the training. Non-squared size of the weights matrix allows using different sizes of hidden states in the encoder and the decoder, this type of encoder-decoder is called bilinear. It allows for capturing stronger representation with the encoder and saving computation in the decoder. [45] Attention can be also visualized with a heat map that shows attention from languages of Machine Translation (MT) model.

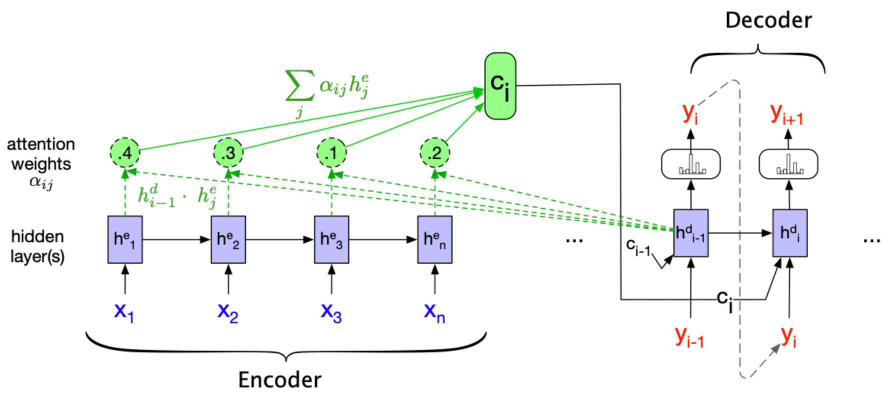


Fig. 17: Example of translating with attention [45]

2.2.10 Beam search

Greedy search fails to produce the most accurate choices. To produce the most likely translation of the input sequence, we use beam search [45]. Beam holds on to several most likely possible hypotheses (possible output sequence) in each iteration. When new hypotheses are created in each step, again pruned to the limit, taking the highest probable hypotheses. At the end of a generation, when all of the sequences reached "EOS", all of the probabilities are compared and the sequence with the highest one is taken as output. Using the chain rule, we can break down the product of each word given its prior context, which we can sum as logs. The closer the log probability is to zero, the higher the probability. This version of beam search still has problems with different lengths of sequences. As Naive Bayes algorithm, it assigns lower probabilities to a longer sequence. To accommodate for this problem we normalize all the probabilities by the number of tokens in sequence.

$$\operatorname{score}(y) = -\log P(y \mid x) = \frac{1}{T}\sum_{i=1}^{t} -\log P\left(y_i \mid y_1, \ldots, y_{i-1}, x\right) \tag{24}$$

2.2.11 Transformer

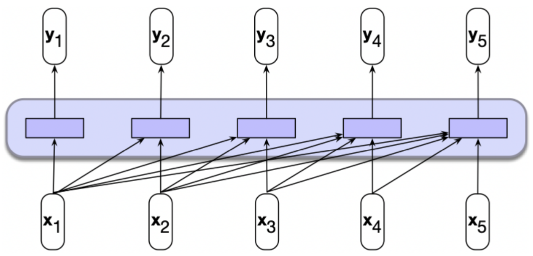
Transformers took over RNNs for most of the tasks in NLP, due to their ability to be computed in parallel. Even though they are being used for seq2seq problems, they no longer, demand step-by-step computation [45]. Usually, a transformer is made of transformer blocks stacked on top of each other, similarly to stacked RNN. The transformer architecture can take the form of Encoder, Decoder, or Encoder-decoder, all of which are used for specific tasks, trained in a specific way. In this part, I am going to explain the original transformer from Vaswani [98]. Also, I will later use this model in translation multi-label classification to solve the product categorization.

2.2.12 Self-attention

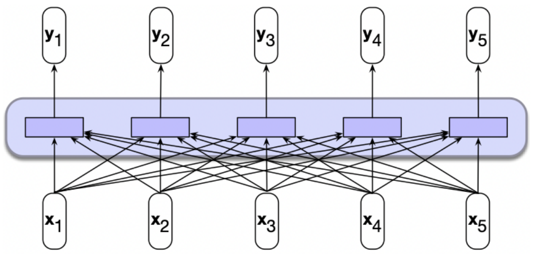
Self-attention is a special layer, that allows for a direct extraction and use of information from arbitrarily large contexts without the need to pass it through intermediate recurrent connections as in RNNs [98]. Similarly, as attention in RNN decoder, where we calculated the dot-products from hidden states of the encoder, we calculate the dot product of input embeddings, to find their relevance in the current context.

To compare the scalars for each previous input token, we use the softmax function to normalize them. This creates a weights vector, which is used to generate the output value as the weighted average of inputs seen so far, weighted by their respective weight. In the case of the encoder, we use a bidirectional self-attention layer.

The self-attention mechanism goes further by looking at the embeddings in three different

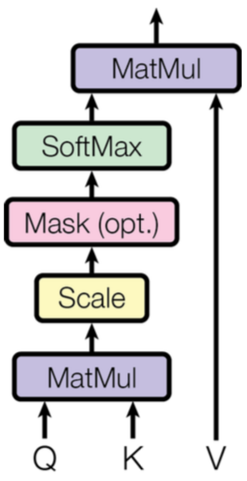


(a) Decoder self-attention layer

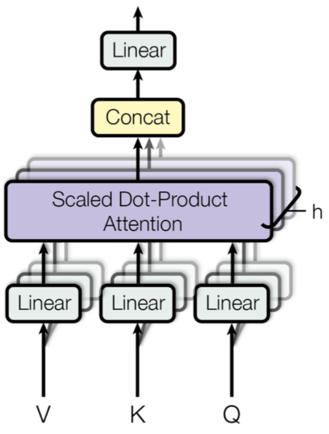


(b) Encoder self-attention layer

Fig. 18: Visual comparison between Encoder and Decoder attention layers



(a) Inside a self-attention layer



(b) Visualization of Multi-head attention

Fig. 19: Visualization of self-attention and multi-head attention

ways (figure 19a), namely query, key, and value [45]. The query says where the focus of attention should be. Key says something about the preceding input compared to the current query. And finally, values that represent the plain input, which should be multiplied by attention. All of these variables are created as a multiplication of input and their weights, which can be updated during training.

$$\mathbf{q}_i = \mathbf{W}^Q \mathbf{x}_i ; \mathbf{k}_i = \mathbf{W}^{\mathrm{k}} \mathbf{x}_i ; \mathbf{v}_i = \mathbf{W}^{\mathrm{v}} \mathbf{x}_i \tag{25}$$

To calculate the score for relevance between query (the current focus of attention) and key (preceding context) we use dot product as previously. And also we combined them as a sum of weights vector created by softmax and our values. To make sure the results of the dot product will not run into numerical issues, because of too big of a gradient, we scale the dot product by the square root of dimensionality of query and key d_k.

$$\operatorname{score}\left(\mathbf{x}_i, \mathbf{x}_j\right) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}} \tag{26}$$

Since the computations are for each of the word are independent, we can parallelize the computation by putting embeddings into one matrix, where each row represents one word. We can then create matrix representation of query, key, and values as:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}}; \mathbf{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}}; \mathbf{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}} \tag{27}$$

Then the self attention for whole sequence is computed as:

$$\mathrm{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathrm{T}}}{\sqrt{d_k}}\right)\mathbf{V} \tag{28}$$

For language modelling, we have to set the upper-triangular portions of the matrix to $-\infty$, which results in zero after applying softmax. It is because otherwise, the model would work in training settings with something it should not know yet. We called it Masked Attention Layer. A problem with Transformers is that Attention is quadratic in the length, so the computation of extremely long documents is limited. We usually set a specified length of a maximum number of input tokens.

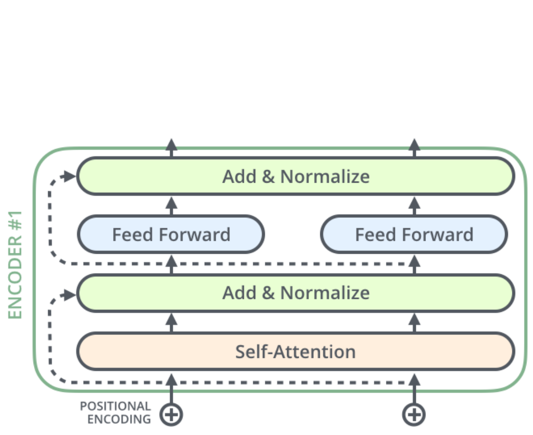
2.2.13 Transformer block

The transformer block consists of the previously explained self-attention layer, normalization layer, and feedforward layer, it also takes advantage of the residual connection to improve learning and allow higher levels layers to have direct access to lower layers [45]. The residual connection is implemented over the attention and feedforward layer. These residual connection helps to train bigger NNs, because the gradient can flow through the identity mappings all the way to the beginning of the network. This also prevents overfitting. Layer normalization is another technique that is used to improve learning. It works by keeping the values of hidden layers in a certain range, which helps backpropagation. The normalization is based on z-score method. If Encoder-Decoder Transformer is used, each Decoder block consists of one more layer that is called the cross-attention layer. This layer takes in keys and values from the Encoder's last layer and combines it with Decoder's queries from the last self-attention layer. After each of the cross-attention layers follows the normalization layer as it is after the self-attention layer. If Decoder is used as a language model there are no cross-attention layers.

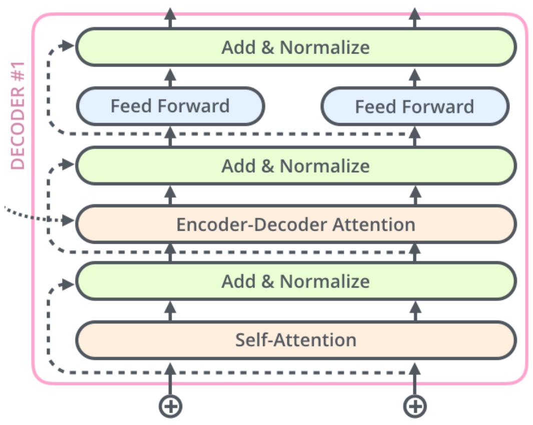
2.2.14 Multi-Head Attention

Multi-Head Attention (figure 19b) allows for finding more than one relation in the input sequence [45]. It is a set of self-attention layers, applied at once. Each of them is called a head. After each of the self-attention layer is computed it concatenates outputs of all the layers and applies linear projection layer.

[5] https://jalammar.github.io/illustrated-transformer/



(a) Encoder block



(b) Decoder block

Fig. 20: Encoder and Decoder block side by side

$$\begin{aligned} \text{MultiHeadAttention}(\mathbf{Q},\mathbf{K},\mathbf{V}) &= \text{Concat}\ (\text{head}_{1},\ldots,\text{head}_{\mathrm{h}})\,\mathbf{W}^{\mathbf{O}} \\ \text{where head}_{\mathrm{i}} &= \text{SelfAttention}\left(\mathbf{Q}\mathbf{W}_i^{\mathbf{Q}},\mathbf{K}\mathbf{W}_i^{\mathbf{K}},\mathbf{V}\mathbf{W}_i^{\mathbf{V}}\right) \end{aligned} \tag{29}$$

2.2.15 Positional Encoding

Positional Encoding allows the model to distinguish between the meaning of a token in a different position. It adds the same embeddings for each of the positions in every iteration for both input and output. In the original paper [98], it was done by adding sin/cos to the word embeddings.

$$PE_{(\text{pos},2i)} = \sin\left(\text{pos}/10000^{2i/d_{\text{model}}}\right) \tag{30}$$

$$PE_{(\text{pos},2i+1)} = \cos\left(\text{pos}/10000^{2i/d_{\text{model}}}\right) \tag{31}$$

2.2.16 Translation using Transformer Encoder-Decoder

[98] was the first implementation of this model for MT. The encoder was constructed from 6 transformer blocks, introduced earlier. It can create a dynamic context embedding for each of the tokens since it has access to the whole sequence at the start. The final layer of the encoder then passes context (key and values) to the decoder, similarly in the case of the RNN model. The exchange of a context between the encoder and decoder happens in so-called cross-attention layers. This is a special layer between the feedforward and multi-head attention layer in the decoder block. In this layer, the encoder provides keys and values from the final layer to join decoder queries from the self-attention layer. This happens in every decoder block. [45]

Nowadays, Transformers in NLP are usually pretrained on a large corpus to gain a language understanding and then transfer-learned for a specific corpus or fine-tuned for a specific downstream task, which is what the field calls those supervised-learning tasks that utilize

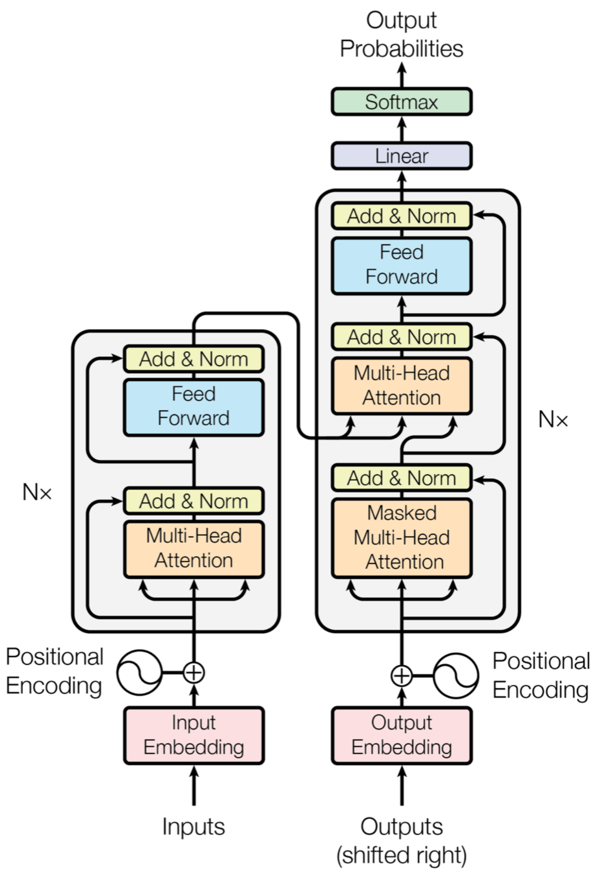


Fig. 21: Transformer from [98]

pretrained models. The models don't have to be only of encoder-decoder architecture but can also take the form of separate encoder and decoder for specific tasks. In recent years, researchers and other big companies started open-sourcing their models online. One of the most popular websites for sharing models is called hugging face. There are also two libraries that implement the most recent state-of-the-art models like Transformer tenseror2tensor[6] (implemented in TensorFlow[7]) and fairseq[8] (PyTorch[9]).

2.2.17 Encoder Transformers

One of the most used encoder transformer is called BERT, which stands for Bidirectional Encoder Representations from Transformers [96]. Similar to Embeddings from Language Model (ELMo) from [73], which builds biLSTM to extract dynamic contextualized embeddings from sequences, the BERT approach uses a transformer for this task but trains the model in a different way. The advantage of BERT is also that it doesn't read the text left-to-right or right-to-left, but it all at once. It is trained on a task called Masked Language Modelling (MLM) (earlier Cloze task) and also Next Sentence Prediction (NSP).

[6]https://github.com/tensorflow/tensor2tensor
[7]https://github.com/tensorflow/tensorflow
[8]https://fairseq.readthedocs.io/en/latest/
[9]https://pytorch.org

The embeddings are built using WordPiece tokenizer, with two additional tokens [CLS], [SEP] and [MASK]. The final hidden state of [CLS] is used as a representation for a classification task. [SEP] as a separator between the first sequence and the other one for the NSP task. [MASK] is replaced instead of masked words in MLM. In training, MLM is applied to 15% of the input is given to the generator to be masked. It then chooses 80% that are going to be masked, 10% that are going to be replaced by a random word, and 10% that are going to stay the same. The final layer of the Encoder then outputs the correct masked word or is randomly replaced. For the NSP, the second sentence is randomly sampled in 50% of the times and evaluated by a special token as right or wrong. This way, the model is better prepared for tasks like question answering and natural language inference. After the model is pretrained to understand language, it can be fine-tuned to a different task, usually classification.[10]

2.2.18 Feature-based Approach with BERT

The original paper also describes the use of contextual embeddings as features for a standalone model [96]. It used those to evaluate on Named Entity Recognition task. Two approaches that performed best were the weighted sum of the last four hidden layers and concatenation of the last four hidden layers. Figure **??** shows possible extractions of tokens and their score on name entity task. The biggest advantage is the reduction of computation cost, while the classifier is not trained on top of the Transformer. But we also lose some performance that could be gain from fine-tuning the Transformer representation for a specific task. The sequence representation can be greatly improved by approaches like () or SBERT-WK [100].

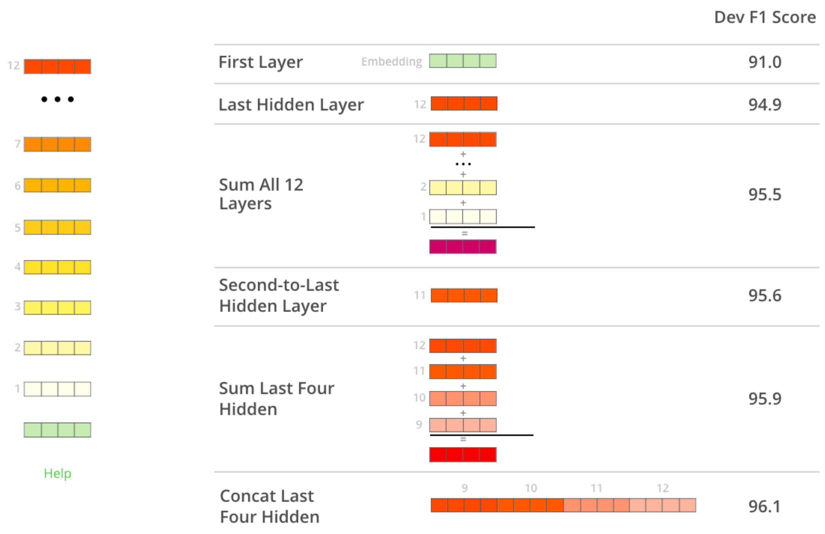


Fig. 22: Results of different features extraction approaches for a name entity recognition task [96][11]

[10]BERT is well describe here https://jalammar.github.io/illustrated-bert/ and https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/

[11]Picture taken from https://jalammar.github.io/illustrated-transformer/

2.2.19 Modifications of BERT

A Robustly optimized BERT (RoBERTa) from [62] is trained longer with bigger batches on longer sequences. Also removes NSP and trains with dynamically applied masking patterns. This resulted in better accuracy, it also showed that BERT was under-fitted . Electra iteclark2020 is a special kind of encoder transformer that differs by its training approach. It uses a small generator for replacing masked sequence, but after the output uses another model, discriminator, that tries to predict if the words are right (figure 23). This way, researchers were able to train the model faster with better accuracy than BERT. There have been also papers that not only try to optimize the performance but also try to lower computation power needed at training or inference times, that would be for example ALBERT (A Lite BERT) [56] and DistillBERT (distilled vesrion of BERT) [80].

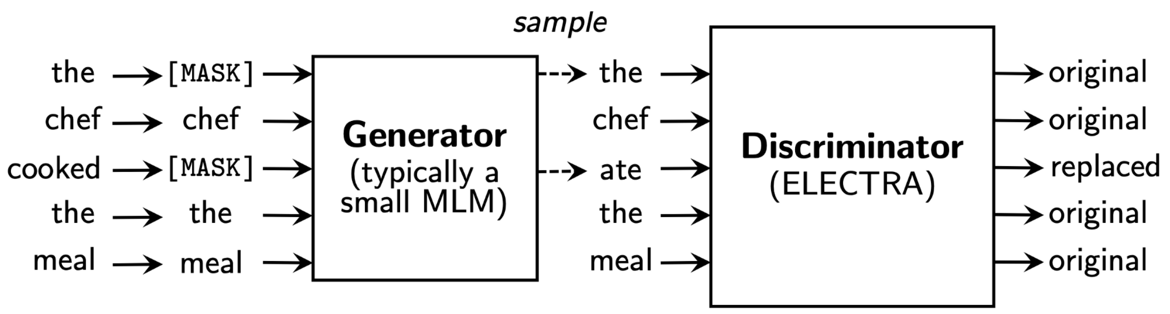


Fig. 23: Example of training ELECTRA Transformer [23]

Both of these two modifications has also open-sourced version trained on Czech corpus. I will use them to extract contextualized embeddings in the practical part (RobeCZECH [87], Small-e-Czech [51]. I have also found other Czech BERT like transformers Czert [84], Czech ALBERT [106], or multilingual variations Slavic BERT (pl, ru, cs, bg) [4]. There are also Multilingual SBERT for 50+ languages[12].

2.2.20 Decoder Transformers

Standalone transformer decoder (autoregressive model) works as a language model for text generation, in tasks like Question Answering, Commonsense Reasoning and Summarization, or also for some kind of classification tasks, like entailment, similarity, multiple choice. It learns as a standard language model by masking the sequence further right from the current step. During the inference, it consecutively generates the word and uses it as the next input. The model is usually pretrained on the similar text we will use the downstream task for, so the model learns the probability distribution of words in the given task. Most used decoder transformers are from Generative Pre-trained Transformer (GPT) family from OpenAI [74]. The last of them, GPT3 [14], has 175 billion parameters and shows a very high understanding of language, it is trained using in-context learning. Recently two new models, which beats

[12]https://huggingface.co/sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2

the GPT3 , were announced Gopher from DeepMind [75] with 280 billion parameters and Generalist Language Model (GLaM) from GoogleAI [27] with 1.2 trillion parameters, even though it has so many parameters it can be more efficient during inference.

2.3 Image processing

2.3.1 Convolutional neural networks

These NNs use kernels of different sizes for processing data. Convolution is a mathematical operation that allows for a combination of data points over a given kernel (It can be also thought of as a weighted averaging window) [32]. This operation can be done in many dimensions, usually, we use a 1-D kernel grid for text or time-series data and a 2-D kernel grid for image processing. Convolutional neural networks were first introduced in LeCun 1989 [58] for the handwritten digit-recognition task, until that time images were processed as flatten vectors, which required way more parameters while processing with FFNN. This didn't progress so much until the early 21st century, because of computational power and lack of image data. Later in 2012, it was the first time that CNN won the ImageNet[13] Large Scale Visual Recognition Challenge, then almost all of the image applications involved some kind of CNN. The winner architecture is known as AlexNet [54] (figure 25).

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{32}$$

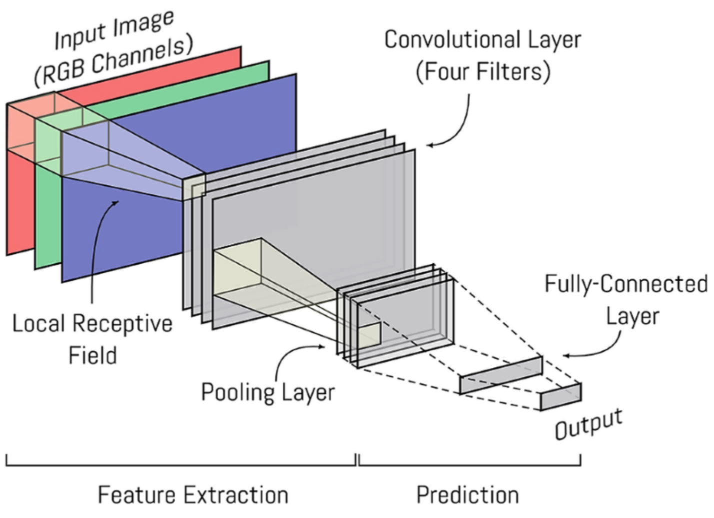


Fig. 24: Convolutional Neural Network with one convolutional layer [69]

CNN finds sparse interactions, since kernels are smaller than the input it can detect meaningful features that occur just in some cases, not in all the pixels [107]. Therefore, we use multiple kernels to be able to learn more features. One kernel has the same parameters

[13]https://www.image-net.org

for the whole image (meaning the kernel weights are shared). So the kernel is looking for the same patterns in different fields (local receptive fields). We use pooling to make the representation smaller, therefore, more saturated. When using pooling we use kernel (usual 2x2, 3x3) with a different length of stride, meaning the step when translating the pooling kernel. Instead of combining the inputs with different trainable weights, we take the maximum of the inputs (it is also possible to use the average of the inputs). Similar to FFNN, there is usually more than one hidden convolutional layer. We also use activation function, nowadays mostly ReLU, in between the layers. Before the final layer of the softmax activation function, we flatten all of the last kernel representation into a dense layer and create output with the softmax layer. Historically, multiple dense layers were used before the last output layer, but this led to a large number of parameters. CNNs are translation invariant and equivariant. The former means they can learn patterns in different positions of the image and the latter that they can also learn the same object rotated and at a different scale. To use it to its full potential we usually augment the training dataset by rotating, scaling, and offsetting the images, thus creating more data examples to train on. CNNs can have problems in the real world, because they are looking for patterns, but not checking if they make positional sense. There has been research around models that can distinguish between patterns in different positions, these models are called Capsule Networks [78].

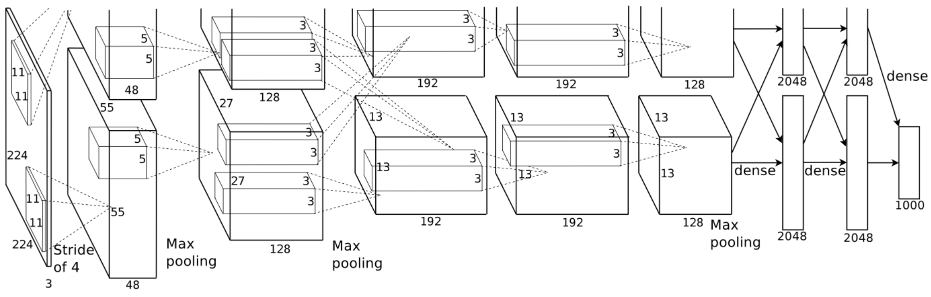
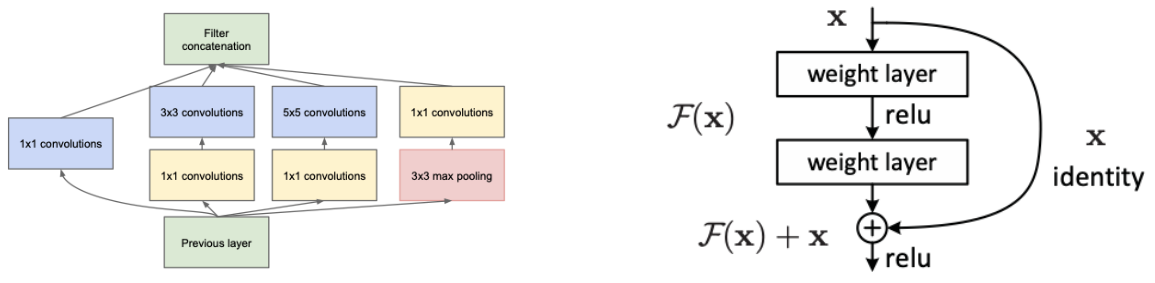


Fig. 25: Architecture of AlexNet taken from the original paper [54]

Nowadays, we also use different types of CNN blocks and special connections. First of the block were introduced by Visual Geometry Group (VGG) in their famous networks VGG16 and VGG19 [85]. The VGG block is composed of the convolutional layer with padding (adding zeros along with the original input) to maintain the resolution, activation function applying nonlinearity such as ReLU and maximum pooling layer. The VGG network had 5 blocks, where the first two of them applied one convolutional layer and the other three applied two convolutional layers.

Then Google came with GoogLeNet [92], which implemented an Inception block. The inception block is composed of four different types of processing, which are concatenated at the end of the block. The combination can be seen in the figure 26a. Researchers at Google later came with batch normalization. Batch normalization rescales and offsets all of the output from the previous layer. It is used usually after each convolutional layer with an activation function. This helps to make the optimization surfaces smoother, thus easier to optimize.



(a) Inception block from [92]

(b) Residual block from [37]

Fig. 26: Inception and residual block

Another architecture introduced was ResNet [37]. This model introduced residual connections (figure 26b), which can learn identity function easily and also improve backpropagation. The residual connections are added at the end of each residual block. If there is a need to reduce the number of channels we apply a convolutional layer of 1x1 to the residual connection. These architectures have been continuously improving the recent improvement was name as Big Transfer (BiT) [52] which was also followed up with its distilled version [10]. Another architecture that applied a similar method was DenseNet [42]. Instead of adding the connection at the end of the block, it concatenates them, thus propagating the representation deeper. Other new features were introduced in MobileNet [40], which goal is the scalability of the models so they can be easily embedded. In this implementation, they split standard convolution into depthwise and pointwise convolution, which reduces the computation costs. They ale proposed linear bottlenecks and inverted residual connections to make better use of low-level features. In my practical part, I will make use of pretrained EfficientNetB0 [93, 94], which takes ideas from the mobile designs. [32, 107] In recent years researchers came with Visual Transformer (ViT) [26]. It performs similarly to CNN and may replace them in the future.

2.4 Multimodal machine learning

In recent years, there has been a popular approach using multi-modal classification models, these models use both text and image (or other modalities) combined together [7]. For text language transformer model is usually used to extract context. This was already elaborated on in the theory part, where I showed the theory behind transformer language models capturing semantics relationships on text. In the case of image embedding CNN is used.

The theory of multimodal machine learning is based on 5 main challenges: Representation, Translation, Alignment, Fusion, and Co-learning.

The representation talks about how the representations from different modalities can be represented together in vector space. There are two main categories: joint and coordinated representation. The joint representation tries two combine multiple modalities in one dimensional space. This can be done by different deep learning models, like RNN, Deep Boltzman networks, or Deep Belief networks. On the other hand, coordinated representation leaves modalities in their own space, but tries to create similarity constraints, which creates coordinated space. This mapping of two representations can also be done with deep learning models, but most of the models are based on canonical correlation analysis.

In the other challenge translation, we try to map data from one modality to another. This can be used when we want to generate one modality from the other. There are two different approaches of example and generative-based translation. The example-based approach uses a prebuilt dictionary. The example-based approach can be further separated into retrieval and combination-based. The retrieval-based approach works kind of like k-NN and tries to find the closest samples from the training set. This can be done either in unimodal or created semantic space. The combination-based approach retrieves multiple samples and combines them using complex handcrafted or heuristic-based rules. The generative approach creates a model to translate from one modality to another. Currently, there are three main types: Grammar-based, Encoder-Decoder, and Continuous generation models. Grammar-based uses pre-defined grammar combined with the high-level concepts presented in the input modality. Encoder-Decoder is a trained neural network for a generation. Continuous generation models are focused on translating sequences like the text to speech for example.

Aligning is about finding direct relationship between the modalities. For example that a word in input corresponds to a part of an image. Explicit alignment is a similarity-based approach where we try to find similar sub-components in the modalities, this can be either supervised or unsupervised. Implicit models try to create latent space for another task.

Fusion talks about the theory behind building models from multiple modalities. Early, late and hybrid fusion talks about combining the features to produce predictions. These types of fusion are also called model-agnostic approaches. Early fusion builds a model from a concatenation of feature vectors. Late fusion combines the decisions of models trained in a unimodal task. Hybrid combines both early and late approaches. There is also another model-based approach that combines features using different types of architecture, like Multi kernel learning methods, Graphical models, and types of neural networks, for example, Multi-View RNN is used for a recommendation based on text and images [25].

Co-learning is about how transferring knowledge between multiple modalities can improve the training of models. It is needed for example in situations where one of the modalities is not always rich in information.

In the practical part, I will only build an early fusion classifier based on multimodal representation using simple concatenation and a late fusion classifier based on a summation of unimodal classifier decisions.

2.5 Keboola

Keboola is a service that allows companies to store and work with their data. It has many features, but in my work, I have only used their snowflake and python transformations and workspaces, where I implemented all of my functionality. These transformations and workspaces run on powerful backends, so the computation is faster than what would be possible on my laptop. For most of the following tasks I used SMALL computational jobs, which run on a system of 16 GB RAM, 2 CPU cores, 1 TB SSD shared. For some of the more expensive computational procedures, I used MEDIUM job, which runs on 32 GB RAM, 4 CPU cores, 1 TB SSD shared.[14]

2.6 Ensemble models

There are multiple types of ensemble models[79][15]. One of them is bagging, it aggregates models based on bootstrapped data, an example of this would be Random Forrest. Which is an algorithm that builds Decision Trees on sampled data and combines their prediction. Boosting builds weak learners (very simple models) in sequence. The most used boosting model nowadays is Gradient Boosted Trees (GBT). Stacking approach builds a model on predictions of multiple models. Blending or voting combines predictions from multiple models to make a final prediction. The combination can be also weighted.

[14]https://help.keboola.com/management/project/limits/
[15]https://scikit-learn/stable/modules/ensemble.html

3 Related work

Most of the work on product categorization was done in the last decade. [82] used Naive Bayes classifier combined with selected words from a bag of words (which is another name for the term-document matrix) by their mutual information. [83] proposed a two-stage classifier. Using a coarse level classification algorithm, they estimated confusion probability between different classes, which allowed them to group similar categories in the latent groups. In the second stage, they built one classifier for each group. [53] created FFNN with an embedding layer to extract meaning from words of text. The embeddings of a sentence were averaged to create a vector representing the title. [35] create three types of classifications based on transformed distributional semantics from descriptions of products. The classifiers were divided based on their possible labels. A path-wise classifier was made to classify from each of the possible paths available. Node-wise type uses all the nodes as possible labels and finally depth-wise approach, which builds a classifier at each of the hierarchical category layers. For the distributional semantics, they used Skip-Gram with Negative Sampling combined using a graded weighted bag of words to produce document representation. [76] concatenated doc2vec with image embeddings brought the best results for three-level classification problems. [36] creates Deep Categorizing Network frame, that builds an RNNs on a sequence of product attributes (title, brand, ...) to extract embeddings about a product. All the RNNs outputs are concatenated together and passed to FFNN with a softmax layer at the top. [105] compared the prediction performance of text and image as an input source for the classifier. They discovered that in unimodal classification, text outperforms image-based classifier. They suggested creating image embeddings ahead since the CNN takes a lot of propagation time and is not therefore best to use as a model in multimodal architecture. Needless to say, the combination of both brings the best results. The decision-level (late) fusion outperformed the feature-level (early) fusion. [103] proposed Attention-based CNN for text processing, which performed similarly to GBT, while training seven times faster. [71] used text features extracted using TF-IDF and chi-squared test to build a flat classifier. [1] creates lexicon based on n-grams from titles for similarity-based categorization. Each of the tokens from n.gram is weighted based on its importance and relevance with respect to the category assigned in training. [59] builds MT like classification translator. In this case, the model is trying to generate a sequence of category tree given the title of the product. Other studies build models based on stacking of multimodal models, transfer learning of pretrained CNN networks, deep metric or similarity learning using siamese networks, building domain-specific MLM, and finetuning it to the task of categorization or building cross-modal attention fusion to combine both visual and text transformers. [97, 47, 102, 16, 108, 13, 17]

There were also a couple of e-commerce machine learning challenges, proposed by the organization SIGIR eCOM. In 2018, they held a contest for product categorization using just a title of the product. Teams created models based on biLSTM, k Nearest Neighbours (k-NN), text CNN. [61, 33, 38, 41, 43, 55, 60, 86, 91, 104, 30]

In 2020, they SIGIR eCOM also held a contest regarding product categorization, but the objective of this challenge was to solve it as multimodal challenge. Teams submitted models based on Efficient Manifold Density Estimator combined with multimodal fusion, classifier using features from BERT and BiT combined with Highway network-based fusion, late decision level classifier based on BERT and ResNet152 model, ResNet and BERT/biLSTM features combined using co-attention and finally SE-ResNeXT and BERT features combined using addition, concatenation and attention maps to build boosted late fusion classifier. [3, 8, 21, 11, 20, 99] In these challenges, a different kind of F1 loss function was used to compare results of the teams, I will F1 weighted metric along with all my experiments. Earlier approaches were only focusing on categorizing materials given reference data.

4 Practical part

In this part, I first try to understand the nature of given data through analysis. Then use some preprocessing to make the data cleaner, meaning easier to understand for machine learning models. Then I train many different models. I split this task into two different scenarios. I first try all the selected algorithms on L1→LX problem, which is classifying products that have already been categorized at least to level 1. After choosing the best algorithms for this task, I test the best models on categorizing from the root, which is also needed for product categorization.

4.1 Data preparation

I first had to merge some tables in Keboola database using snowflake transformation, namely *item* table, which holds the primary keys about all the products displayed on mall.cz. *item_photo* a table containing URLs to photos of products, *item_content* table containing text for the *title*, brief and description for every product, *item_category* table that holds information about where a given the material should be displayed and *list_category* table containing a valid category for the current date. While merging the data, I decided that I will first work with level one category *Hobby and zahrada* (*Hobby a garden* in English), for the categorizing from the root I will sample the categories later. So now I am left with tables containing the information just about this category and its nodes. In total, it is more than 240 thousand materials in 647 categories, but only 509 unique *category_path* ids. I also tried merging information about parameters belonging to a material, but this data was very sparse and the input of parameters while listing a new product comes after choosing a category, so it would be useless in real use.

4.2 Data analysis

4.2.1 Categories

So I start looking at the level 1 (L1) category *Hobby and garden*, which is one of the main categories on mall.cz (root would be L0). The nodes in this category reach a maximum of level 6. Most of the materials (96%) are placed in the categories L3 and L4, as you can see in the figure 27. No materials are placed directly in *Hobby and garden* and only some of the materials are placed in the L2, which is usually in some category that doesn't have many subcategories. In the figure 28 you can see the ten largest category nodes in this tree, which all come from L3 or L4.

The largest category *Ruční nářadí > Ostaní ruční nářadí* (translated as *Hand tools > Other hand tools*) have almost 10% materials of the whole tree. Just from this graph, we can see that the categories are very unbalanced. To show how unbalanced the labels are, I

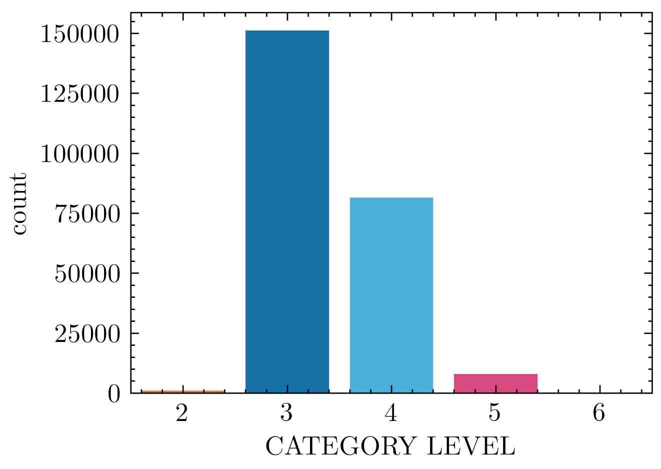


Fig. 27: Frequency of materials in each of the category level

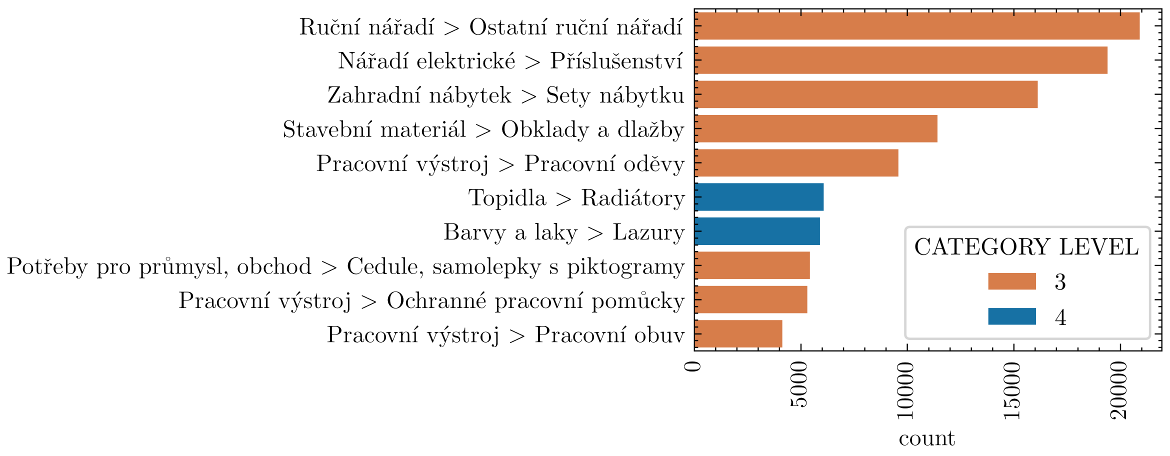


Fig. 28: Ten largest categories from Hobby and garden

created a histogram (figure 29) with a number of materials in a category. It shows that most of the categories have less than 150 materials. The most problematic categories for the classification will be the ones with very few samples, I will also address this problem in preprocessing part by oversampling the small categories.

I noticed that one material can be placed in more than one category. You can see in the figure 30 there are not that many materials that would be assigned to 3 or 4 categories at the same time, but the number of materials that are assigned to 2 categories is not negligible.

Further, inspecting this behavior, I also found out that a category is uniquely specified by its *category_id*, but can have different path and names in the tree structure, meaning you can get to a specific category more than one way.

Therefore, the category tree can be visualized using Directed Acyclic Graph (DAG). For example, in the table 1 there are 4 different names for *category_name Houpací sítě*. Even non-Czech speakers can notice that some of the category names are affected by misspells. This problem will not affect flat classifiers (predicting all nodes as one level) since we are

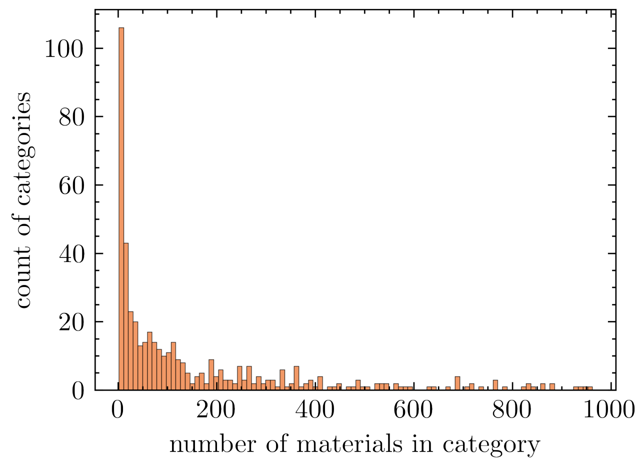


Fig. 29: Count of categories by number of materials

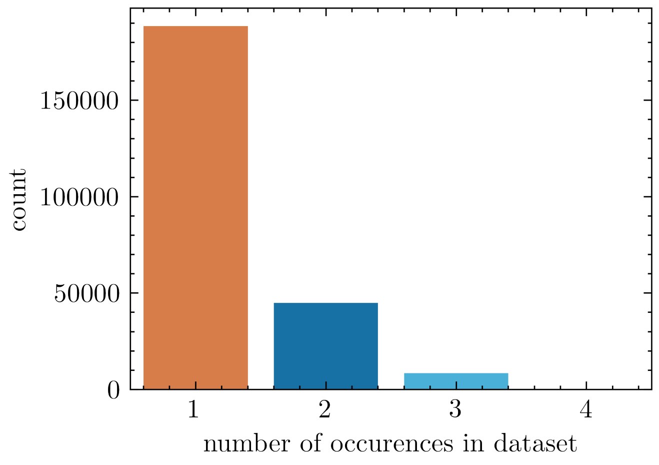


Fig. 30: Graph showing how many materials are assigned to more than one category

going to use the unique *category_id* as labels but could be making the task harder for seq2seq models or would be hard if we would apply hierarchical classification. I will try to solve this issue in preprocessing part, by reducing the number of paths per *category_id*.

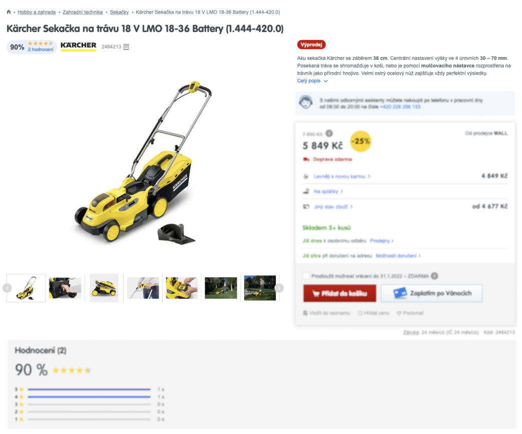
4.2.2 Inspecting text data

In the figure 31 you can see our data sources from a product detail view of a random lawnmower from the website. The first subfigure contains category path, *title*, *brief*, and *images*, which I will get to in another section. The other picture shows a *description*. This example shows that the *title* usually contains a lot of specifications and serial numbers. Although it could give quite a lot of information to humans, it will not be that beneficial in most cases for algorithms since a lot of the tokens are quite rare and will result in default embedding.

I have also noticed that some titles don't pose any meaning, because they only contain only companies and/or vendor names and some only serial numbers. This would be a problem if

Tab. 1: Example of categories with multiple possible paths

Zahradní nábytek >Houpací sítě , křesla >Příslušenství
Zahradní nábytek >Houpací sítě >Příslušenství
Zahradní nábytek >Houpací sítě, křesla >Příslušenství
Zahradní nábytek >Zahradní houpačky, houpací sítě >Houpací sítě >Příslušenství
Zahradní nábytek >Stany, altany
Stavby na zahradě >Domky, altány >Altány
Zahradní nábytek >Slunečníky, zastínění >Stany, altany
Dům, byt >Koupelna, sanitarni technika >Umyvadla
Dům, byt >Koupelna a sanitární technika >Umyvadla
Dům, byt >Koupelna, sanitarni technika >Vany, sprchy >Umyvadla
Hobby a zahrada >Dům, byt >Koupelna, sanitarni technika >Vodovodní baterie
Hobby a zahrada >Dům, byt >Koupelna a sanitární technika >Vodovodní baterie
Hobby a zahrada >Dům, byt >Dveřní, okenní kování
Hobby a zahrada >Dům, byt >Dveře >Dveřní, okenní kování



Hobby a zahrada • Zahradní technika • Sekačky • Kärcher Sekačka na trávu 18 V LMO 18-36 Battery (1.444-420.0)

Kärcher Sekačka na trávu 18 V LMO 18-36 Battery (1.444-420.0)

90% 2 hodnocení KÄRCHER 2464213

Výprodej

Aku sekačka Kärcher se záběrem **36 cm**. Centrální nastavení výšky ve 4 úrovních **30 – 70 mm**. Posekaná tráva se shromažďuje v koši, nebo je pomocí **mulčovacího nástavce** rozprostřena na trávník jako přírodní hnojivo. Velmi ostrý ocelový nůž zajišťuje vždy perfektní výsledky.

Celý popis

5 849 Kč -25%

4 849 Kč

od 4 677 Kč

Skladem 5+ kusů

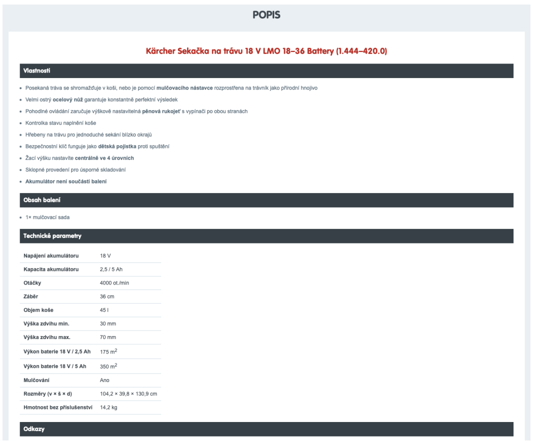
Přidat do košíku

Zaplatím po Vánocích

Hodnocení (2)

90 %

(a) Product detail view of random lawnmower



POPIS

Kärcher Sekačka na trávu 18 V LMO 18–36 Battery (1.444–420.0)

Vlastnosti

- Posekaná tráva se shromažďuje v koši, nebo je pomocí **mulčovacího nástavce** rozprostřena na trávník jako přírodní hnojivo
- Velmi ostrý **ocelový nůž** garantuje konstantně perfektní výsledek
- Pohodlné ovládání zaručuje výškově nastavitelná **pěnová rukojeť** s vypínači po obou stranách
- Kontrolka stavu naplnění koše
- Hřebeny na trávu pro jednoduché sekání blízko okrajů
- Bezpečnostní klíč funguje jako **dětská pojistka** proti spuštění
- Žací výšku nastavíte **centrálně ve 4 úrovních**
- Sklopné provedení pro úsporné skladování
- **Akumulátor není součástí balení**

Obsah balení

- 1× mulčovací sada

Technické parametry

Napájení akumulátoru	18 V
Kapacita akumulátoru	2,5 / 5 Ah
Otáčky	4000 ot./min
Záběr	36 cm
Objem koše	45 l
Výška zdvihu min.	30 mm
Výška zdvihu max.	70 mm
Výkon baterie 18 V / 2,5 Ah	175 m^2
Výkon baterie 18 V / 5 Ah	350 m^2
Mulčování	Ano
Rozměry (v × š × d)	104,2 × 39,8 × 130,9 cm
Hmotnost bez příslušenství	14,2 kg

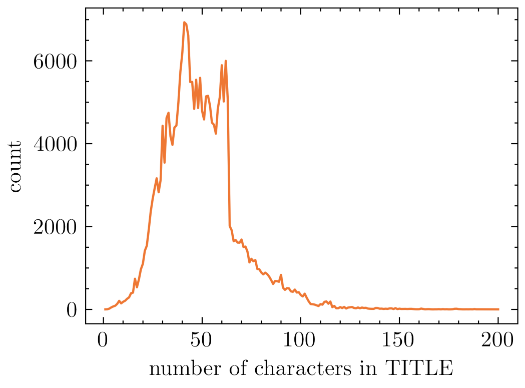
Odkazy

(b) Description from product detail view of random lawnmower

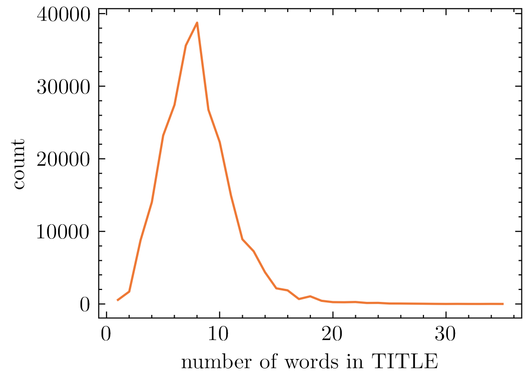
Fig. 31: Data sources from product detail view

we would be using only *title* text, but since we are going to use multiple texts concatenated this should not be a problem. Before training, I will remove these materials. Further analyzing the sources, I found that *brief* can be the same as *description* or can be a part of the description. Thus, providing both *brief* and *description*, might not be beneficial and can lead to bias classification. I will address this in preprocessing.

In the following graphs, I have looked at the distributions of a number of words and characters per the source of text. In the *title* distribution isn't much to inspect, but we can see that the distribution is quite close to normal, with some long tail for longer titles. In the brief distribution, we can see that most of the briefs have a length of 300. It is because the brief usually copies the *description* and for marketplace partners, there is only an allowance of 300 characters. There are also some products that are listed by mall.cz, and they reach a character length of up to 700. The is also plenty of descriptions that have exactly 150



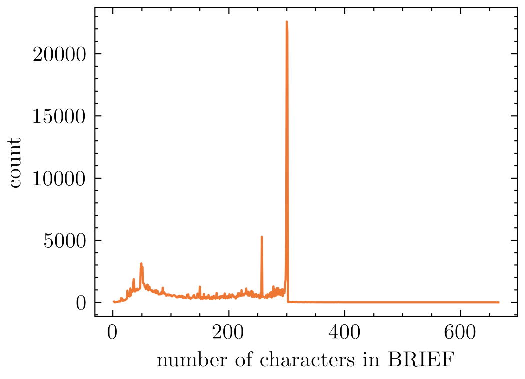
(a) Distribution of characters in *title*



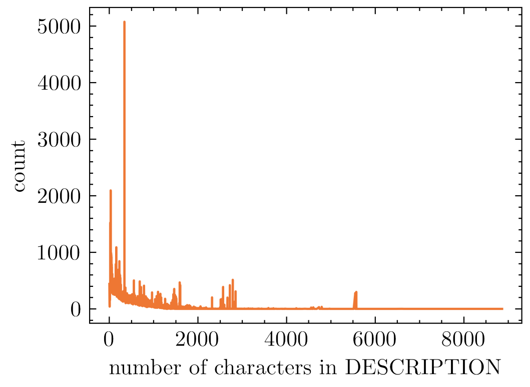
(b) Distribution of words in *title*

Fig. 32: Distributions of characters and words in *title*

characters.



(a) Distribution of characters in brief text



(b) Distribution of characters in the description text

Fig. 33: Distribution of character in brief and description

4.2.3 Extracting features from text

To extract information from text data, I am going to use pretrained models on the Czech corpus, that give a representation of words in dimensional vector space, namely fasttext[16] models and pretrained transformer from hugging face[17]. Before transforming the text into any embedding, I am going to preprocess it. The preprocessing of the text includes: lower casing, special characters and number removal, lemmatizing[18] and stemming[19] and stop words removal.

For description source text I also used library BeautifulSoup[20], for removing hypertext tags, which can be seen in table 2. After preprocessing, I have created multiple columns representing the text sources, namely *title*, *brief*, *description*, *title+brief* (further *TB_text*),

[16]https://fasttext.cc
[17]https://huggingface.co
[18]https://nlp.fi.muni.cz/ma/
[19]https://research.variancia.com/czech_stemmer/
[20]https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Tab. 2: Example of preprocessing description

before	<h3>Elektrický vyžínač 400W0– 270mm </h3><h3 class="pnl pnl–default con-left h3"> TECHNICKÁ SPECIFIKACE:</h3><ul><li>Pracovní zábě...
after lowercasing hypertags removal and words filtering	elektrický vyžínač technická specifikace pracovní záběr otáčky naprázdno min jmenovitý příkon napětí struna délka kabelu hmotnost vlastnosti ochranný háček proti pnutí...
after removal of stop words, lemmatizing and stemming	elektrick vyžínač technick specifikac pracovn záběr otáčk naprázdn min jmenovit příkon napět strun délk kabel hmotnost vlastnost ochrann háček pnut kabel měkčen rukojeť...

title+description (*TD_text*), *title+brief+description* (*TBD_text*). As I already touched upon large portions of the text, in *brief*, are the same as the beginning of *description*, therefore in *TBD_text* I removed the *brief* if it was the same as the beginning of the *description.* I will evaluate the performance of each option with one of the classifiers.

I have already presented fasttext library earlier in the theory, in this part I want to take advantage of their pretrained subword word2vec for classification task. I am going to create sentence embeddings using their *get_sentence_vector* method, which produces an average of unit vectors over each word embedding in the whole sentence.

In figure 34, you can see a visualization of text embeddings based on *title+brief* using reduction t-SNE[21] of category *Hobby a Zahrada > Zahradní technika > Sekačky* (*Lawnmowers category*). It should be noted that this 2-dimensional representation might be far from the 300-dimensional representation that the vectors have. But it is a simple way to check if the same products get somehow clustered together. This text representation isn't very friendly to read, but later in the section about extracting features from images I will visualize the embeddings by products images and talk a little bit about what can be seen in the images.

4.2.4 Inspecting images

Each of the products can have multiple images. In my task, I am going to only use images that are flagged in column *IS_MAIN_MEDIA*. This works in most cases, but there are some products, which don't use a relevant image as the main media. All the main images should also be on white background, which isn't also always true. Some products can have images that are not very relevant. Like images with a lot of noise in the background or even incorrect images showing other products. The images were available to me as a link in the table, therefore before using them I have to download them first.

[21] https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

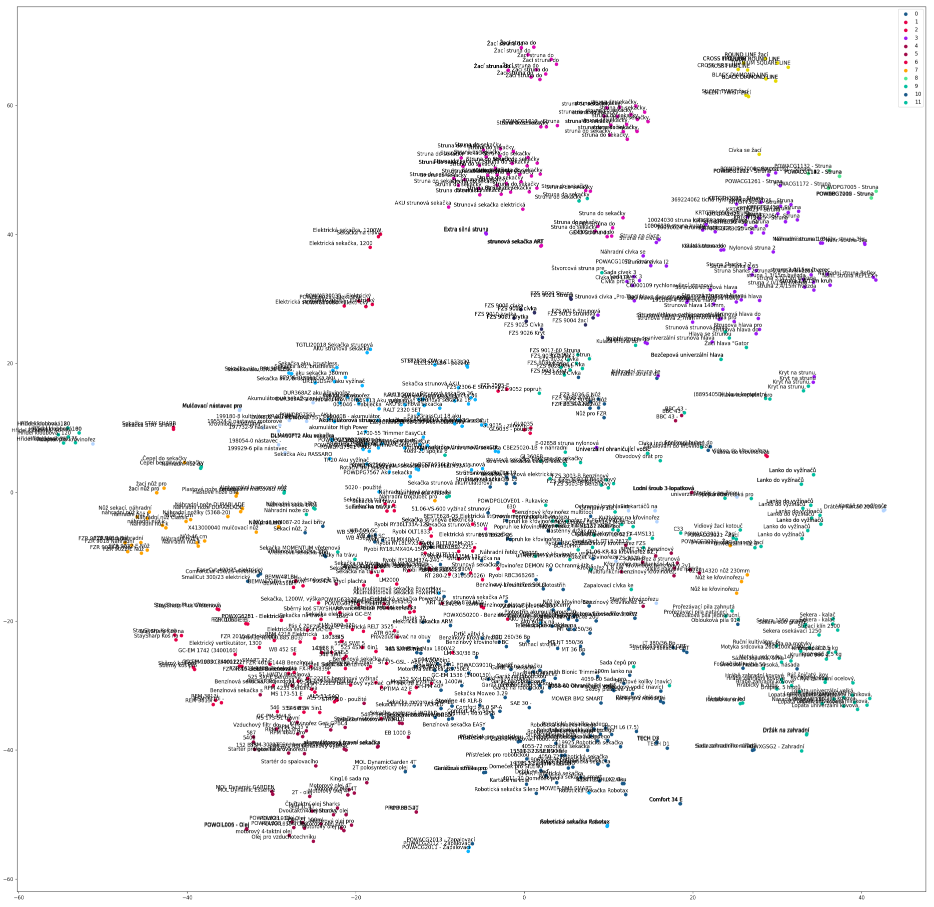


Fig. 34: Visualization of t-SNE reduction on fasttext embeddings applied to name and brief of lawnmowers category

4.2.5 Extracting features from images

While working on this thesis, I created a process that downloads the main image for any current or new product listed on mall.cz. While downloading the picture, I extract the features from the image right away and save it to the database. Therefore, using them for other downstream tasks is more compute effective, as some studies noted. For feature extraction from images, I used EfficientNetB0[22] that I have introduced earlier in the theory part. Removing the top of this CNN, we can obtain a 1280-dimensional representation of each image. I could have also used another CNN, but I chose this one because of its size and effectiveness. If I had access to GPU (TPU) hardware, this would make the computations

[22] https://www.tensorflow.org/api_docs/python/tf/keras/applications/efficientnet/EfficientNetB0

much easier and CNN with a larger number of parameters could be used. This would also improve the performance of downstream classification tasks.

For visualizing the dimensional space I used similarly to text embedding dimension reduction technique t-SNE, but also UMAP[23] and PCA[24]. Here I only present UMAP embeddings on image data, the rest of the visualization I put to the appendix.[25] This UMAP visualisation nicely shows how similar images are clustered together in representation taken from CNN.

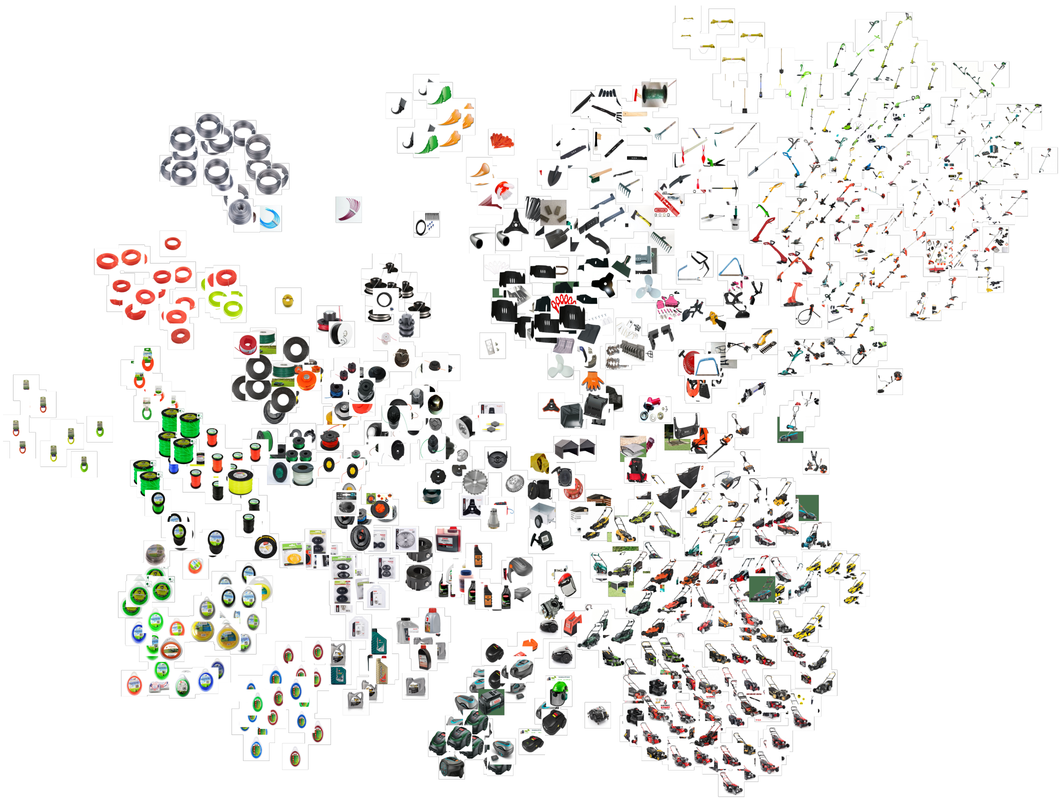


Fig. 35: Visualization of image embeddings using UMAP

4.3 Train/Test split

For each of the models created, I will use the same train/test set to be able to objectively compare predictions of all the various algorithms. To split them, I used *sklearn.model_selection.train_test_split*[26] with the same *random_state* seed every time. This way the split will be the same every time and I don't have to keep track of two datasets in the database. Before splitting, I always remove the smallest categories with which I will not be training. These will be the categories that have less than 20 products listed in them. After

[23]https://umap-learn.readthedocs.io/en/latest/#

[24]https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[25]These pages helped me with settings rigth hyperparameters for t-SNE https://distill.pub/2016/misread-tsne/ and UMAP https://pair-code.github.io/understanding-umap/

[26]http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

removal of these categories, the number of target categories was brought to half number (around 300). During splitting, I also use the stratified option to create evenly distributed splits by *category_id* column. The size of the test dataset will be 30%. To further support the unbalanced classification problem, I over-sampled categories that contained between 21 and 99 products. After doing so, the minimal number of materials in the training dataset per category is 100.

In the following sections, I will choose between a couple of different features, which in this case will be embeddings from different sources and modalities. Most of the time the target variable used for classification will be *category_id* (or *category_path* for MT like multi-label classification).

4.4 Evaluation of results

For evaluation, I will use weighted F1 metric[27] since accurately classified materials are what we are looking for, this the choice was also used in most of the papers regarding product categorization.

4.5 Unimodal models

In this section, I tried to build a couple of different classifiers based on each modality separately (only text or images). While creating models, I started with well-known machine learning algorithms, that didn't turn out well for the high dimensional representation that I extracted. Most of the successful algorithms are based on Deep learning architecture of some sort. Algorithm that worked surprisingly well was k-NN classifier both on text and images.

4.6 Using text

4.6.1 Flat classification model

By flat classification I mean, that I am going to put all the categories in the same level, and try to predict which of the category has the highest probability of being the right one. For this, I will experiment with a couple of models. Ordinary machine learning models like Logistic Regression, Naive Bayes, Random Forrest (RF), and XGBoost and also deep learning model FFNN. I also tried to fit data on models like Naive Bayes (sklearn.linear_model.MultinomialNB) and Logistic Regression. NB had very low accuracy, around 30% on this data and LR, wasn't even able to converge. When fitting Random

[27] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

Forrest[28] and Gradient Boosted Trees (XGBoost[29]), I had a problem with computation power, since the train was quite large (170k of materials) and these models are not iterative. So I decided to create multiple models of the same kind (three for each) on sampled train datasets and then ensemble them into one (blending approach). Every iteration, I sampled 10% of the training dataset. The RF models were quite accurate itself, they all scored above 80%, one of them even 81%. When combining them into ensemble models, they achieve an accuracy of 85.15% if ensembles with mean of probability and 85.25% if ensemble with a max of probabilities. Even though this is a great way to make accurate predictions, it is not a production-efficient solution. In the XGBoost case, I wasn't able to build due to a computational difficulty. To solve it, I also try reducing the embedding dimension to 100 dimensions via PCA. This helped in faster computation, but resulted only in an accuracy of 43%.

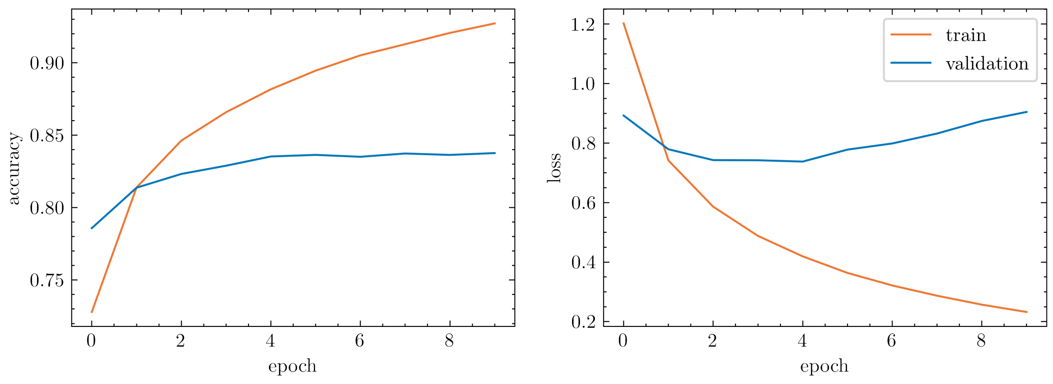


Fig. 36: Example of model being overfitted on training data

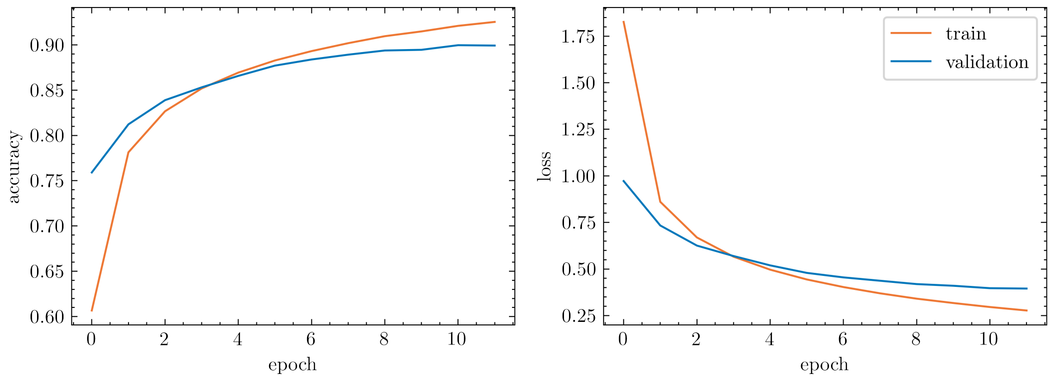


Fig. 37: Correctly fitted model

The next models I built were FFNN using Keras library[30]. I build a sequential model using flatten and dense layers provided in the library. As an optimizer, I chose Adam [49] with

[28]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[29]https://xgboost.readthedocs.io/en/stable/
[30]https://keras.io

Tab. 3: Results from training FFNN with different settings and text sources

hidden layers	hidden units	*title*	*brief*	*description*	
1	32	0.816	0.795	0.807	
1	64	0.8436	0.846	0.848	
1	128	0.868	0.867	0.865	
1	256	0.882	0.879	0.881	
1	512	0.89	0.899	0.902	
2	128	0.872	0.862	0.848	
2	258	0.878	0.882	0.878	
2	512	0.886	0.895	0.888	
hidden layers	hidden unist	*TB_text*	*TD_text*	*TBD_text*	*concat_NBD*
1	32	0.818	0.831	0.827	-
1	64	0.853	0.864	0.87	-
1	128	0.877	0.887	0.889	-
1	256	0.890	0.902	0.91	-
1	512	0.909	0.912	0.912	0.931
2	128	0.889	0.88	0.889	-
2	258	0.894	0.895	0.907	-
2	512	0.908	0.894	0.915	-

the default setting. The models were trained using SparseCategoricalCrossenthropy[31] with a batch size of 300. While training MLP, I experimented with a couple of settings, choosing different breadth and widths of the network. Starting with MLP of one layer with 64 hidden units, the model already performed as well as RF ensemble models. I also benchmarked all the text sources on these models since it was easy to train. Results from all the runs are written in the table 3. During training, I kept an eye on the training and validation loss, checking if the model does not overfit the training data. To control it, I set up an early stopping mechanism that would stop if the validation loss would stop improving. In the figure 36 I show how fast can the FFNN start overfitting the training data (picture is a case of overfitting FFNN trained on the image embeddings). In the figure 37, I show training and testing validation of FNNN with one hidden layer with 256 units.

From the results, we can see that the best performing architecture for most of the text sources is the architecture with one hidden layer of 512 units. This architecture was outperformed only in the case of *TBD_text* source, which was the best performing source of text. The best performing model, in this case, was a network with two hidden layers of 512 units, which was better just by a small margin. Later I also got an idea to train FFNN on concatenated vectors for all three text sources (900 dimensions). This approach was superior to the previous ones and achieved highest accuracy of 93.1%.

[31] https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy

4.6.2 Fasttext similarity model

Using the knowledge from previous classifiers I only used *TBD_text* embeddings for each sentence to create a k-NN model based on cosine similarity. I implemented this using Facebook AI Similarity Search (FAISS) library[32]. This library is CPU and can be also GPU optimized for searching k-NN in dimensional space based on metric of choice. To create the model I first normalized the embeddings using L2 norm*faiss.normalized_L2* then create an index (which is the model itself) using *faiss.IndexFlatIP*[33].

```
# add faiss initialization
def get_n_similar_materials(features, k=50):

    index = faiss.IndexFlatIP(features.shape[1])
    faiss.normalize_L2(features)
    index.add(features)

    distances, neighbors = index.search(features, k)

return distances, neighbors

# building model
index = faiss.IndexFlatIP(train_vector.shape[1])
faiss.normalize_L2(train_vector)
index.add(train_vector)

# predicting
faiss.normalize_L2(test_vector)
nearest_neighbours = index.search(test_vector, k-NN)
predictions = [train.iloc[neighbour].CATEGORY_ID.value_counts().index[0]
               for neighbour in nearest_neighbours[1]]
```

This index can be then used to find k-NN for the test set. When predicting I use the majority of categories from the nearest neighbor from the train set as the prediction. During testing I tried a couple of different settings for the number of nearest neighbors, the best of them was 1-NN shown in the table 4. The best performing option was using the model with 1-NN. This approach outperformed the flat classifiers by little more than 2%. Using concatenated vectors resulted in very similar results as in case of the *TBD_text*.

4.6.3 Classification as machine translation problem

In this case, I am going to train to seq2seq models similarly to [59] (figure 38). For this purpose, I used fairseq library[34], which holds the implementation of state-of-the-art

[32]https://github.com/facebookresearch/faiss
[33]https://faiss.ai/cpp_api/struct/structfaiss_1_1IndexFlatIP.html
[34]https://github.com/pytorch/fairseq

Tab. 4: Performance of cosine similarity using faiss library on the test set

k-NN	*TBD_text*	*concat_NBD*
1	0.938	0.939
3	0.923	-
5	0.914	-

neural network models from recent papers in PyTorch[35]. One of the models is going to be bidirectional RNN using LSTM units with attention block[36] and the other one is going to be Transformer[37]. For this purpose, I had to use a medium-sized Keboola machine, since the the small one couldn't handle the memory requirements for these architectures. Even the medium-sized machine wasn't much powerful for this algorithm to train somewhat fast, and GPU implemented version should be considered in the future.

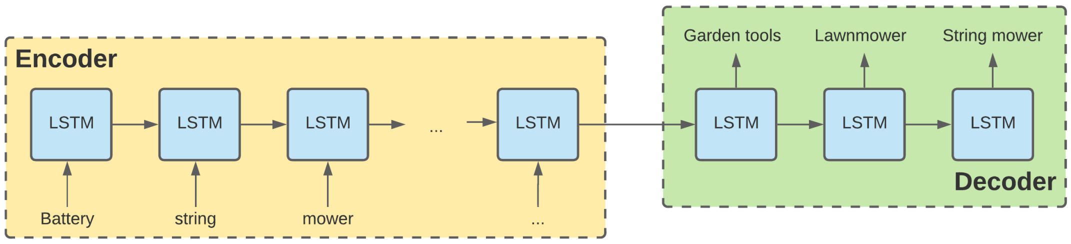


Fig. 38: Visualization of MT like multi-label categorization algorithm

I had to downgrade the size of my models since my task isn't as large as translating language and also because of the computational power. When training the first RNN model, I chose a bidirectional architecture of 512 Input/Output embeddings and 1 hidden layer with a size of 1024. One epoch of these models on 170k data took 30 minutes, and the results weren't great. Thus, I downgraded the model, even more, using only 256 neural units in the hidden layer. Training of this RNN using a medium-size machine took approximately 7 minutes. After 6 epochs, the validation loss stagnated, and I decided to stop the training. Following similar approach I also created Encoder-Decoder Transformer. Parameters of both models and results can be seen in table 5.

4.6.4 Classifier with transformer embeddings

I will be using a pretrained Encoder transformer on the Czech corpus for representation of text data. The models I tried were Czech ELECTRA[38], RoBERTa[39] and multilingual

[35]https://pytorch.org
[36]https://fairseq.readthedocs.io/en/latest/models.html#module-fairseq.models.lstm
[37]https://fairseq.readthedocs.io/en/latest/models.html#module-fairseq.models.transformer
[38]https://huggingface.co/Seznam/small-e-czech
[39]https://huggingface.co/ufal/robeczech-base

Tab. 5: Performance of multi-label classification using translation models

	biLSTM RNN with attention	Encoder-Decoder Transformer	
Input/Output Embedding Dimension	512	512	512
RNN Hidden Layer Size	256	-	-
FFN Hidden Layer Size	-	512	512
Stacked Layers	1	2	1
Attention Heads No. of Parameters	-	8	8
Dropout	0.2	0.2	0.2
Results	0.790	0.780	0.813

SBERT[40]. All of these were pretrained on a large corpus of text. Models were downloaded using hugging face library transformers.

The first idea was to use a pretrained Transformer and fine-tune it for my task. I tried fine-tuning the smallest transformer, ELECTRA, and one epoch on all the training examples (170k) would take 8 hours running on a medium-sized job. Therefore, I decided to use extracted the embeddings just from the pretrained network without fine-tuning. ELECTRA produces 256 long embeddings for each input token, and the others produce 768 long embeddings. Because of the limitation, in computational power, I decided to only use the *TB_text* source and clip the max length of the input sequence to Transformer. For ELECTRA, I chose a length of 30 tokens in the input sequence and for the other, I chose 10, since their computation is three times more expensive because of the embedding size. The embeddings can be extracted in many ways, depending on which layer we pick the embeddings from, and how do we combine each token embedding into the sequence embedding. For the Sentence BERT, I will be using the last hidden state as it is meant to be used. In case of ELECTRA, and RoBERTa I will also last hidden state, but also sum of last 4 state and their concatenation. For RoBERTa, I won't do the concatenation because the vector would be too long. On these representations I trained FFNN as I did in previous section. I wasn't able to make the model converge while using the concatenation of 4 embeddings from ELECTRA. All results can be seen in the table 6.

4.7 Using Images

In this part, I am going to use extracted features from EfficientNetB0 for classification.

4.7.1 Flat classification model

Since extracted features are the size of the 1280-dimensional vector, it is hard to fit it with standard machine learning models, like I tried in example with text feature with embeddings of size 300. Therefore, for images, I am going to use neural network models right away.

[40] https://huggingface.co/sentence-transformers/paraphrase-multilingual-mpnet-base-

Tab. 6: Performance of multi-class classification using representation extracted from pretrained Transformers

			FFNN architecture	
Transformer	hidden states	Results	hidden layers	hidden units
Multilingual SBERT	last layer	0.83	1	768
ELECTRA	last layer	0.658		512
	sum of last 4 layers	0.849		
	concatenation of last 4 layers	0.14		1024
RoBERTa	last layer	0.882		768
	sum of last 4 layers	0.869		

Similarly, I trained models of different sizes like in the text. The results can be seen in the table 7. The best architecture was a neural network with 1 hidden layers with 512 hidden units. The results are on average 10% below the text classification.

Tab. 7: Results from training FFNN with image data using different settings

Hidden layers	Hidden units	Results
1	64	0.807
1	128	0.825
1	256	0.831
1	512	0.847
2	128	0.809
2	256	0.829
2	512	0.845

4.7.2 Similarity model based on images features

Again, like in the text part, I used the FAISS library to build an index (model) for finding k-NN in this case of image features. This approach outperformed the flat classification model based on image by 3%. Results can be seen in table 8.

Tab. 8: Performance of cosine similarity using image data on the test set

k-NN	Results
1	0.879
3	0.864
5	0.850

4.7.3 Unimodal error analysis

Since in training there are almost 300 possible categories, it is impossible to print a confusion matrix showing the wrong classified number of materials with decent quality to provide some visual feedback. Therefore, I am going to choose manual categories that had low accuracy. I am going to filter out categories that had more than 100 materials, to get a more representative sample.

Inspecting text-based classified materials from the model which had 94% accuracy (similarity 1-NN fasttext). The lowest accuracy occurred in categories *Substrates* and *Fertilizers*. Both of these categories have a lot of words As expected, there were also some wrongly assigned materials in each sentence. There are also case where the model could have been chosen wrong because of a mislabel. This problem appeared in more categories.

Image-based classifier had an overall lower accuracy for more classes. I found a lot of misclassifications, either because of the nature of data or noise labels. By the nature of data, I mean how data gets processed with CNN. There are plenty of materials that might look the same but may belong to a different category. For example *Sink taps category*, have many types (*bathroom sink, kitchen sink, show/bathtub sink*) listed in different categories, but visual they look almost the same. A similar problem was with materials from *Hand tool accessories category*, where a lot of the materials are displayed on the picture in the box, with triggers different features than for the specific materials. This issue is very hard to deal with using images and the best thing we can do is use the images with other data like text, as we have done previously. Another issue are noisy labels. For example, in the category *Hobby and Garden > Alarms* there are a lot of wrongly listed materials. In this category we can find *electric scooter, fridge, freezer or audio player*. These types of mislabeled data can be seen almost over the whole dataset. Cleaning this noise is not an easy task, I will look at one option how to deal with it in the next section.

In the translation task, I evaluated models based on generated *category_path*. As I touched upon earlier, the *category_path* is not unique, which might be misleading in some cases. When inspecting predictions, it can be seen that the models created simplification of some paths, or they put it into a category with the same ID but a different path. For example, the material got categorized into the category *Dryers*, but the right category was *Clotheslines and Dryers*. I did some postprocessing of the translation Transformer predictions, where I compared the last categories in the path. I turn out the prediction of this model is higher than in testing. It achieved 90% accuracy. Inspecting the rest of the 10% predicted, I found other materials, that were semantically in the right category. Therefore the translation model could achieve even better results with clean data.

4.8 Label cleaning

As we have seen in the last section, data are quite noisy, meaning the materials are not always placed in the correct category. This makes it hard for classifiers to correctly separate data in multidimensional space. There is typically quite a lot of materials listed in higher-level categories than they should be. These are mostly accessories listed in the main category of products. I was looking for a way to deal with this, and I found a library called cleanlab[41]. This library provides a tool to find incorrectly classified samples. It uses the assigned probability of any learned classifier on the data. With the probabilities, there is an algorithm that chooses the most concerning samples for each label. While using this on the whole dataset of 240 thousand materials, it filtered out 2200 materials with wrong labels. It was based on text data, so a noisy label does not only have to mean it was the wrong category, but also for some samples the text semantics quality was very poor, and therefore the model had no information to correctly classify the product. Later, I train FFNN based on *TBD_text* with the mislabeled data. This model achieved 3.3% better performance in training and 0.8% improvement in testing. Even though this isn't a big leap, iteratively removing and repositioning wrongly listed materials can lead to even better performance.

4.9 Multimodal

In this part, I am going to train both early and late fusion models based on both image and text.

4.9.1 Early fusion classifier

When constructing an early fusion classifier, the modalities are joint and enter the classifier together. In this case, I am going to concatenate the text source and image. As a text source, I choose the best performing one from text modality which was *TBD_text* extracted via fasttext. For the image, I am going to use the only embeddings I have and which are the embeddings from EfficientNetB0. Before training, I create one matrix from the two sources. After concatenation embedding represents one product that has 1580 dimensions. Similar to studies I read the early fusion classifier is not particularly better than the best score of the unimodal model. In this case, it got better performance than the image model alone, but way worse performance than the text model. The results can be seen in the table 9.

Tab. 9: Results from training FFNN while combining text and image sources into early fusion classifier

Hidden layers	Hidden units	Results
1	64	0.856
1	128	0.858
1	256	0.840
1	512	0.865
1	1024	0.858
2	128	0.859
2	256	0.852
2	512	0.849

[41] https://github.com/cleanlab/cleanlab

4.9.2 Late fusion classifier

A late fusion classifier is a kind of similar to an ensemble model, but in this case the each of the models is trained using a different modality. To build this classifier I am going to use the classifier that I have already built for the image and text itself. Again I am going to use the best performing model that I had since now. That is the *TDB_text* model and the image model. To use both of these models I use the sum of their probabilities to predict the category. The late fusion classifier helps the performance opposite to the early one. Achieving 92% accuracy, it performed one percent better than the best text classifier but didn't improve over the text-similarity 1-NN model.

4.10 Categorization classification from root category

In all the previous examples, I was showing how accurate can the predictions get when working with products from a given tree of L1 category. In this section, I would like to find out how hard is the task of assigning these L1 categories. I have used a mix of L1 categories, containing the largest L2 categories in each, over which I built the best text classifier. In total, it was 371k products from L1 categories *Sport > Bags, Car > Car accessories, Home > Decorations, Hobby and Garden > Hand tools and PC and Office > Apple Store*. All the text data went through the same preprocessing as described previously. I have again used the best performing FFNN like in previous sections, which is the architecture of 1 hidden layer and 512 units in the layer and also 1-NN based on cosine similarity. As text data, I used both combinations of *TBD_text* and concatenation of TBD. The results are displayed in the table 10. As you can see, it is almost perfect for each of the models or text sources. I did not use the concatenation of TBD for the similarity model, because it turned to outperform as well as *TBD_text*. It might be expected that the performance drops a little if we will build a model based on all the L1 categories, but it will be minimal. The use of this model will be described in the next section.

Tab. 10: Results of categorization from root

Models using fasttext	*TBD_text*	*concat_NBD*
Cosine similarity 1-NN	0.985	-
FFNN 1 hidden laye, 512-units	0.996	0.997

4.11 Use in production

In all the tests, I was testing for top 1 accuracy, to see if the model could be used automated without supervision. We saw that the highest accuracy the models got was 93%. This would not be certainly perfect in production and in the long run as it would hurt us more when creating noisy data while wrongly listing new products. There is also another possible way of using the models in production, and it is with a human in the loop. While listers add new products, we can provide them with additional information of possible categories to list into. Therefore, I also checked what will be the top 5 and 10 accuracies. The classification from the root will be almost perfect so that part will be automatic. But in case that the users thinks the algorithm predicted wrong L1 category, they might be able to choose other category. These decisions should be logged and later checked if a correct decision was made. This could also help with finding the wrong labels. Then L1$\rightarrow$LX model for a given part tree can provide categorization to possible categories. Using the top 5 or 10 options in prediction makes the problem easier and therefore greatly improves performance. You can see the results of showing top@5 or top@10 in the table 11.

Tab. 11: Results of possible production models for use in a human-in-the-loop process

Approach	Accuracy@5	Accuracy@10
best image model	0.948	0.967
best text model	0.984	0.991
late fusion modol	0.985	0.992

5 Conclusion

I have successfully finished all the tasks. First, I have analyzed the data for L1$\rightarrow$LX categorization. I chose the L1 category *Hobby and Garden*. Containing almost 250 thousand products in 647 categories spread over 6 levels. With most categories in levels 3 and 4. It was shown that the dataset is very unbalanced. Although some of the classes contain thousands or ten thousand of materials, most of the categories contain less than 500 materials.

I have shown that this type of classification is a hard problem and its the foundation is clean data.

I have gained knowledge of state-of-the-art algorithms, which I successfully used on real data, solving the product categorization. Ordinary machine learning algorithms were hard to use with the high dimensional data of text and image representation represented. Therefore, most of the models built were deep learning-based. These models were easy to train and manage. I have also done an ablation study to test the accuracy of different combinations of text sources. As expected, the best performing text source was achieved when combining all of the text sources. Even better results were achieved when combining representation for each of the text sources separately, which is also quite self-explainable.

For the representation of text, I used pretrained models on the Czech corpus. The most successful model was fasttext. This library implements word2vec with subword information. Even though the corpus it was trained on was general, it worked very well for separating the categories. I was kind of unhappy that the Transformer models, came nowhere close in terms of accuracy. I have read in studies that domain-specific models should work way better, that should be also true for subword word2vec. Training my own Transformer wasn't possible, because it would not be possible with the CPU jobs. I was very surprised with the performance of k-NN based on cosine similarity models. The best thing about this model is that it is dynamic and training doesn't take any time.

The image representation was in terms of performance behind the text models. It might be also partially because the model is general and doesn't have specific features. It is also one of the weakest models available nowadays. Again, I have not been training my own CNN, because of the computational. A bigger CNN would achieve better performance.

Translation models used as multi-label classification were behind the classical FFNN in terms of performance. But after postprocessing of the path predictions, I found out that most of them were right, but only chose a different path to the final category. In the end, they came almost to the performance of the best text models based on fasttext. This approach is only hard because of the category tree, as I showed in the data analysis part. When preprocessing these paths, it is hard to decide which path should be the right one. The pruning nature of translation model predictions should be more closely analyzed. They might as well make the category tree cleaner to navigate through.

As was talked about in studies, the early fusion multimodal model does not improve performance. But it was shown that the late fusion model does by a small margin.

Throughout my work on this thesis, I was mostly focused on a model-centric view of the problem. But throughout the work, I saw that way more should have been on the data.

When analyzing the prediction, I found that way too many materials are miscategorized and a certain process should be built to start iteratively repositioning them to the right categories. I have also tested a library called cleanlab. Used for finding predictions with a questionable probability assigned by model. This approach actually removed problematic materials and improve the accuracy of both the training set and test set. The possibilities of such approaches should be further inspected, as cleaner data would help with the categorization.

I have also tested classifying from the root by choosing a couple of different categories. This wasn't a real challenge as the task was straightforward forward and the model achieved almost perfect results. It might be possible to fit both classification tasks, root→L1 and L1→LX together. If we build a model with a lot of parameters, it will be probably able to generalize. The only problem is it would have to choose from 18 thousand categories. This would be considered extreme classification.

In the end, I have also talked about how it could be used as a human-in-the-loop when predicting the top 5 or 10 predictions. This would help with the generalization of product categorization so that there is minimal space for human error. The models are ready to be implemented in production.

6 Further work

When solving the product categorization, I came across things that need to be implemented in the future to improve the process, here are some examples.

All the models were trained using a CPU machine. Neural networks greatly benefit from GPU (TPU) computation units, as it is the best we have currently for matrix computation. Therefore, using cloud computing or standalone GPU would be beneficial in the future of tackling any problem that will be using large deep learning models. We have also seen that some new models like Transformers are very time ineffective when it comes to training on CPU, in some case unusable when using with a lot of data.

In my test, I tried to spare computational power and only used EfficienNetB0, which is the smallest of EfficientNets. Bigger CNN would certainly improve the categorization task. It would be also beneficial to fine-tune the pretrained CNN as it would update the weights to our case or train our domain-specific model [96]. The fine-tuning or building own model also applies to Transformers.

I have tested only some representations of text and images. There are other methods to get these representations. For example, the StarSpace[42] the library can be used to get embeddings based on some similarities between subjects. Other deep similarity or metric learning like siamese networks or triplet/quadruplet loss networks can transform representations into more sophisticated space. For images, unsupervised autoencoders can be also used to gain representation [28, 31]. There are many other models for text classification that I haven't explored, for example, the CNN text classifier mentioned earlier or Hierarchical Neural Networks [70].

There are plenty of other options to try when working with multiple modalities. I just scratched the surface when using simple early and late fusion models, As I talked about earlier in the section 3 different types of representation and fusion can be used. There have been papers implementing co-attention for two modalities to improve the performance. These models have a much higher understanding of both modalities. Another approach can also be transforming multiple representations into one representation using multimodal autoencoders.

It has been shown that combining multiple ensemble models can improve the performance of models. I used ensembles of Random Forrest, which improved the performance by 4%. This can be also done for FFNN.

Adding sparse or other features might be beneficial for categorization, The best models for this job would be probably tree, but FFNN can also be tested. The utilization of parameters should be also tested in the future, but there is a need to introduce more materials with

[42]https://github.com/facebookresearch/StarSpace

parameters since at this moment, the parameters are very sparse.

I have touched on the topic of label cleaning, for materials that are currently categorized in the wrong category. A tool that finds wrongly categorized products needs to be created and used iteratively. This would help with the categorization of the new products themselves, as it removes the noise from training data. I used a library called cleanlab, but there are also different approaches in research, for example in [95, 9, 70][43]. Maybe simpler rules-based methods could be also used.

[43]Code for [95] `https://github.com/Cysu/noisy_label`

Reference

[1] Leonidas Akritidis, Athanasios Fevgas, and Panayiotis Bozanis. “Effective Products Categorization with Importance Scores and Morphological Analysis of the Titles”. In: Nov. 1, 2018, pp. 213–220. DOI: 10.1109/ICTAI.2018.00041.

[2] Md Zahangir Alom et al. “The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches”. In: (Mar. 3, 2018). URL: https://arxiv.org/abs/1803.01164v2 (visited on 12/10/2021).

[3] Hesam Amoualian et al. “SIGIR 2020 E-Commerce Workshop Data Challenge Overview”. In: (2020), p. 6.

[4] Mikhail Arkhipov et al. “Tuning Multilingual Transformers for Language-Specific Named Entity Recognition”. In: *Proceedings of the 7th Workshop on Balto-Slavic Natural Language Processing*. Florence, Italy: Association for Computational Linguistics, Aug. 2019, pp. 89–93. DOI: 10.18653/v1/W19-3712. URL: https://aclanthology.org/W19-3712 (visited on 12/12/2021).

[5] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. “A Simple but Tough-to-Beat Baseline for Sentence Embeddings”. In: *ICLR*. 2017.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. May 19, 2016. arXiv: 1409.0473 [cs, stat]. URL: http://arxiv.org/abs/1409.0473 (visited on 12/06/2021).

[7] Tadas Baltrušaitis, Chaitanya Ahuja, and Louis-Philippe Morency. *Multimodal Machine Learning: A Survey and Taxonomy*. Aug. 1, 2017. arXiv: 1705.09406 [cs]. URL: http://arxiv.org/abs/1705.09406 (visited on 11/21/2021).

[8] Dominika Basaj et al. “Synerise at SIGIR Rakuten Data Challenge 2020: Efficient Manifold Density Estimator for Multimodal Classification”. In: (2020), p. 5.

[9] Melanie Bernhardt et al. “Active Label Cleaning: Improving Dataset Quality under Resource Constraints”. In: (Sept. 1, 2021). URL: https://arxiv.org/abs/2109.00574v1 (visited on 01/16/2022).

[10] Lucas Beyer et al. “Knowledge Distillation: A Good Teacher Is Patient and Consistent”. In: (June 9, 2021). URL: https://arxiv.org/abs/2106.05237v1 (visited on 01/15/2022).

[11] Ye Bi, Shuo Wang, and Zhongrui Fan. “A Multimodal Late Fusion Model for E-Commerce Product Classification”. In: (2017), p. 4.

[12] Piotr Bojanowski et al. “Enriching Word Vectors with Subword Information”. In: (July 15, 2016). URL: https://arxiv.org/abs/1607.04606v2 (visited on 12/12/2021).

[13] Alexander Brinkmann and Christian Bizer. “Improving Hierarchical Product Classification Using Domain-specific Language Modelling”. In: (2021), p. 8.

[14] Tom B. Brown et al. *Language Models Are Few-Shot Learners*. July 22, 2020. arXiv: 2005.14165 [cs]. URL: http://arxiv.org/abs/2005.14165 (visited on 12/12/2021).

[15] Daniel Cer et al. "Universal Sentence Encoder". In: (Mar. 29, 2018). URL: https://arxiv.org/abs/1803.11175v2 (visited on 12/22/2021).

[16] Hongshen Chen, Jiashu Zhao, and Dawei Yin. "Fine-Grained Product Categorization in E-commerce". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. CIKM '19: The 28th ACM International Conference on Information and Knowledge Management. Beijing China: ACM, Nov. 3, 2019, pp. 2349–2352. ISBN: 978-1-4503-6976-3. DOI: 10.1145/3357384.3358170. URL: https://dl.acm.org/doi/10.1145/3357384.3358170 (visited on 12/10/2021).

[17] Lei Chen et al. "Multimodal Item Categorization Fully Based on Transformer". In: *Proceedings of The 4th Workshop on E-Commerce and NLP*. ACL-ECNLP-IJCNLP 2021. Online: Association for Computational Linguistics, Aug. 2021, pp. 111–115. DOI: 10.18653/v1/2021.ecnlp-1.13. URL: https://aclanthology.org/2021.ecnlp-1.13 (visited on 11/23/2021).

[18] Kyunghyun Cho et al. "Learning Phrase Representations Using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1724–1734. DOI: 10.3115/v1/D14-1179. URL: http://aclweb.org/anthology/D14-1179 (visited on 12/05/2021).

[19] Kyunghyun Cho et al. *On the Properties of Neural Machine Translation: Encoder-Decoder Approaches*. Oct. 7, 2014. arXiv: 1409.1259 [cs, stat]. URL: http://arxiv.org/abs/1409.1259 (visited on 11/17/2021).

[20] Varnith Chordia and Vijay Kumar. "Large Scale Multimodal Classification Using an Ensemble of Transformer Models and Co-Attention". In: (2017), p. 5.

[21] Hou Wei Chou et al. "CBB-FE, CamemBERT and BiT Feature Extraction for Multimodal Product Classification and Retrieval". In: (2020), p. 5.

[22] Junyoung Chung et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: (Dec. 11, 2014). URL: https://arxiv.org/abs/1412.3555v1 (visited on 12/10/2021).

[23] Kevin Clark et al. "ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators". In: (Mar. 23, 2020). URL: https://arxiv.org/abs/2003.10555v1 (visited on 12/12/2021).

[24] Alexis Conneau et al. “Supervised Learning of Universal Sentence Representations from Natural Language Inference Data”. In: (May 5, 2017). URL: https://arxiv.org/abs/1705.02364v5 (visited on 12/12/2021).

[25] Qiang Cui et al. *MV-RNN: A Multi-View Recurrent Neural Network for Sequential Recommendation*. Nov. 20, 2018. arXiv: 1611.06668 [cs]. URL: http://arxiv.org/abs/1611.06668 (visited on 01/15/2022).

[26] Alexey Dosovitskiy et al. “An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: (Oct. 22, 2020). URL: https://arxiv.org/abs/2010.11929v2 (visited on 01/15/2022).

[27] Nan Du et al. “GLaM: Efficient Scaling of Language Models with Mixture-of-Experts”. In: (Dec. 13, 2021). URL: https://arxiv.org/abs/2112.06905v1 (visited on 01/15/2022).

[28] Vincent Dumoulin et al. *Adversarially Learned Inference*. Feb. 21, 2017. arXiv: 1606.00704 [cs, stat]. URL: http://arxiv.org/abs/1606.00704 (visited on 01/16/2022).

[29] Jeffrey L. Elman. “Finding Structure in Time”. In: *Cognitive Science* 14.2 (1990), pp. 179–211. ISSN: 1551-6709. DOI: 10.1207/s15516709cog1402_1. URL: https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1402_1 (visited on 12/10/2021).

[30] Angshuman Ghosh, Vineet John, and Rahul Iyer. “Team Waterloo at the SIGIR E-Commerce Data Challenge”. In: (), p. 3.

[31] Yunye Gong et al. *Learning Compositional Visual Concepts with Mutual Consistency*. Mar. 28, 2018. arXiv: 1711.06148 [cs]. URL: http://arxiv.org/abs/1711.06148 (visited on 01/16/2022).

[32] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016. 800 pp. ISBN: 978-0-262-03561-3. URL: https://www.deeplearningbook.org/.

[33] Sylvain Goumy and Mohamed-Amine Mejri. “Ecommerce Product Title Classification”. In: (2018), p. 4.

[34] Edouard Grave et al. “Learning Word Vectors for 157 Languages”. In: (Feb. 19, 2018). URL: https://arxiv.org/abs/1802.06893v2 (visited on 12/12/2021).

[35] Vivek Gupta et al. “Product Classification in E-Commerce Using Distributional Semantics”. In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. COLING 2016. Osaka, Japan: The COLING 2016 Organizing Committee, Dec. 2016, pp. 536–546. URL: https://aclanthology.org/C16-1052 (visited on 12/10/2021).

[36] Jung-Woo Ha, Hyuna Pyo, and Jeonghee Kim. “Large-Scale Item Categorization in e-Commerce Using Multiple Recurrent Neural Networks”. In: *Proceedings of*

the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '16: The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Francisco California USA: ACM, Aug. 13, 2016, pp. 107–115. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939678. URL: https://dl.acm.org/doi/10.1145/2939672.2939678 (visited on 09/27/2021).

[37] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: (Dec. 10, 2015). URL: https://arxiv.org/abs/1512.03385v1 (visited on 01/15/2022).

[38] Makoto Hiramatsu and Kei Wakabayashi. "Encoder-Decoder Neural Networks for Taxonomy Classification". In: (2018), p. 4.

[39] Sepp Hochreiter and Jürgen Schmidhuber. "LSTM Can Solve Hard Long Time Lag Problems". In: *Advances in Neural Information Processing Systems*. Vol. 9. MIT Press, 1997. URL: https://proceedings.neurips.cc/paper/1996/hash/a4d2f0d23dcc84ce983ff9157f8b7f88-Abstract.html (visited on 12/07/2021).

[40] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: (Apr. 17, 2017). URL: https://arxiv.org/abs/1704.04861v1 (visited on 01/15/2022).

[41] Haohao Hu et al. "A Best Match KNN-based Approach for Large-scale Product Categorization". In: (2018), p. 6.

[42] Gao Huang et al. "Densely Connected Convolutional Networks". In: (Aug. 25, 2016). URL: https://arxiv.org/abs/1608.06993v5 (visited on 01/15/2022).

[43] Yugang Jia et al. "An Empirical Study of Using An Ensemble Model in E-commerce Taxonomy Classification Challenge". In: (2018), p. 7.

[44] Armand Joulin et al. *Bag of Tricks for Efficient Text Classification*. Aug. 9, 2016. arXiv: 1607.01759 [cs]. URL: http://arxiv.org/abs/1607.01759 (visited on 12/12/2021).

[45] Jurafsky. *Speech and Language Processing*. 2021. URL: https://web.stanford.edu/~jurafsky/slp3/ (visited on 11/28/2021).

[46] Nal Kalchbrenner and Phil Blunsom. "Recurrent Continuous Translation Models". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. EMNLP 2013. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1700–1709. URL: https://aclanthology.org/D13-1176 (visited on 11/17/2021).

[47] Rashmeet Kaur Khanuja. "Optimizing E-Commerce Product Classification Using Transfer Learning". Master of Science. San Jose, CA, USA: San Jose State University, May 20, 2019. DOI: 10.31979/etd.egyw-ktc5. URL: https://scholarworks.sjsu.edu/etd_projects/679 (visited on 09/27/2021).

[48] Yoon Kim. *Convolutional Neural Networks for Sentence Classification*. Sept. 2, 2014. arXiv: 1408.5882 [cs]. URL: http://arxiv.org/abs/1408.5882 (visited on 11/21/2021).

[49] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: (Dec. 22, 2014). URL: https://arxiv.org/abs/1412.6980v9 (visited on 01/04/2022).

[50] Ryan Kiros et al. “Skip-Thought Vectors”. In: (June 22, 2015). URL: https://arxiv.org/abs/1506.06726v1 (visited on 12/22/2021).

[51] Matěj Kocián et al. *Siamese BERT-based Model for Web Search Relevance Ranking Evaluated on a New Czech Dataset*. Dec. 3, 2021. arXiv: 2112.01810 [cs]. URL: http://arxiv.org/abs/2112.01810 (visited on 12/12/2021).

[52] Alexander Kolesnikov et al. “Big Transfer (BiT): General Visual Representation Learning”. In: (Dec. 24, 2019). URL: https://arxiv.org/abs/1912.11370v3 (visited on 01/15/2022).

[53] Zornitsa Kozareva. “Everyone Likes Shopping! Multi-class Product Categorization for e-Commerce”. In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. NAACL-HLT 2015. Denver, Colorado: Association for Computational Linguistics, May 2015, pp. 1329–1333. DOI: 10.3115/v1/N15-1147. URL: https://aclanthology.org/N15-1147 (visited on 12/10/2021).

[54] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html (visited on 01/15/2022).

[55] Quentin Labernia et al. “Large-Scale Taxonomy Problem: A Mixed Machine Learning Approach”. In: (2018), p. 6.

[56] Zhenzhong Lan et al. “ALBERT: A Lite BERT for Self-supervised Learning of Language Representations”. In: (Sept. 26, 2019). URL: https://arxiv.org/abs/1909.11942v6 (visited on 01/15/2022).

[57] Quoc V. Le and Tomas Mikolov. “Distributed Representations of Sentences and Documents”. In: (May 16, 2014). URL: https://arxiv.org/abs/1405.4053v2 (visited on 12/12/2021).

[58] Y. LeCun et al. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1.4 (Dec. 1, 1989), pp. 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541. URL: https://doi.org/10.1162/neco.1989.1.4.541 (visited on 01/15/2022).

[59] Maggie Yundi Li, Stanley Kok, and Liling Tan. *Don't Classify, Translate: Multi-Level E-Commerce Product Categorization Via Machine Translation*. Dec. 13, 2018. arXiv: 1812.05774 [cs]. URL: http://arxiv.org/abs/1812.05774 (visited on 09/27/2021).

[60] Maggie Yundi Li et al. "Unconstrained Product Categorization with Sequence-to-Sequence Models". In: (2018), p. 6.

[61] Yiu-Chang Lin, Pradipto Das, and Ankur Datta. "Overview of the SIGIR 2018 eCom Rakuten Data Challenge". In: (2018), p. 8.

[62] Yinhan Liu et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: (July 26, 2019). URL: https://arxiv.org/abs/1907.11692v1 (visited on 12/12/2021).

[63] Lajanugen Logeswaran and Honglak Lee. "An Efficient Framework for Learning Sentence Representations". In: (Mar. 7, 2018). URL: https://arxiv.org/abs/1803.02893v1 (visited on 12/22/2021).

[64] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: (Aug. 17, 2015). URL: https://arxiv.org/abs/1508.04025v5 (visited on 12/10/2021).

[65] Tomas Mikolov et al. "Advances in Pre-Training Distributed Word Representations". In: (Dec. 26, 2017). URL: https://arxiv.org/abs/1712.09405v1 (visited on 12/12/2021).

[66] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and Their Compositionality". In: (Oct. 16, 2013). URL: https://arxiv.org/abs/1310.4546v1 (visited on 12/12/2021).

[67] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: (Jan. 16, 2013). URL: https://arxiv.org/abs/1301.3781v3 (visited on 12/12/2021).

[68] Tomas Mikolov et al. "Recurrent Neural Network Based Language Model". In: (), p. 4.

[69] Kyle Millar et al. "Using Convolutional Neural Networks for Classifying Malicious Network Traffic". In: *Deep Learning Applications for Cyber Security*. Ed. by Mamoun Alazab and MingJian Tang. Advanced Sciences and Technologies for Security Applications. Cham: Springer International Publishing, 2019, pp. 103–126. ISBN: 978-3-030-13057-2. DOI: 10.1007/978-3-030-13057-2_5. URL: https://doi.org/10.1007/978-3-030-13057-2_5 (visited on 01/15/2022).

[70] Nicolas Michael Müller and Karla Markert. "Identifying Mislabeled Instances in Classification Datasets". In: (Dec. 11, 2019). DOI: 10.1109/IJCNN.2019.8851920. URL: https://arxiv.org/abs/1912.05283v1 (visited on 01/16/2022).

[71] Vijay Nair et al. “A Machine Learning Algorithm for Product Classification Based on Unstructured Text Description”. In: *International Journal of Engineering Research* 7.06 (2018), p. 4.

[72] Yoshiki Niwa and Yoshihiko Nitta. “Co-Occurrence Vectors From Corpora vs. Distance Vectors From Dictionaries”. In: *COLING 1994 Volume 1: The 15th International Conference on Computational Linguistics*. COLING 1994. 1994. URL: https://aclanthology.org/C94-1049 (visited on 12/22/2021).

[73] Matthew E. Peters et al. *Deep Contextualized Word Representations*. Mar. 22, 2018. arXiv: 1802.05365 [cs]. URL: http://arxiv.org/abs/1802.05365 (visited on 12/11/2021).

[74] Alec Radford and Karthik Narasimhan. “Improving Language Understanding by Generative Pre-Training”. In: *undefined* (2018). URL: https://www.semanticscholar.org/paper/Improving-Language-Understanding-by-Generative-Radford-Narasimhan/cd18800a0fe0b668a1cc19f2ec95b5003d0a5035 (visited on 01/15/2022).

[75] Jack W. Rae et al. “Scaling Language Models: Methods, Analysis & Insights from Training Gopher”. In: (Dec. 8, 2021). URL: https://arxiv.org/abs/2112.11446v1 (visited on 01/15/2022).

[76] Petar Ristoski. “A Machine Learning Approach for Product Matching and Categorization”. In: (2016).

[77] D. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning Representations by Back-Propagating Errors”. In: *Nature* (1986). DOI: 10.1038/323533a0.

[78] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. *Dynamic Routing Between Capsules*. Nov. 7, 2017. arXiv: 1710.09829 [cs]. URL: http://arxiv.org/abs/1710.09829 (visited on 01/15/2022).

[79] Omer Sagi and L. Rokach. “Ensemble Learning: A Survey”. In: *undefined* (2018). URL: https://www.semanticscholar.org/paper/Ensemble-Learning-for-Classification-A-Survey-Krishna/e1353b3a64c70d08b4acdff264b662856e71417b (visited on 01/15/2022).

[80] Victor Sanh et al. “DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter”. In: (Oct. 2, 2019). URL: https://arxiv.org/abs/1910.01108v4 (visited on 01/15/2022).

[81] Mike Schuster and Kuldip Paliwal. “Bidirectional Recurrent Neural Networks”. In: *Signal Processing, IEEE Transactions on* 45 (Dec. 1, 1997), pp. 2673–2681. DOI: 10.1109/78.650093.

[82] Sushant Shankar and Irving Lin. “Applying Machine Learning to Product Categorization”. In: (2011), p. 5.

[83] Dan Shen, Jean-David Ruvini, and Badrul Sarwar. “Large-Scale Item Categorization for e-Commerce”. In: ACM International Conference Proceeding Series. Oct. 29, 2012, pp. 595–604. DOI: 10.1145/2396761.2396838.

[84] Jakub Sido et al. *Czert – Czech BERT-like Model for Language Representation*. Mar. 24, 2021.

[85] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: (Sept. 4, 2014). URL: https://arxiv.org/abs/1409.1556v6 (visited on 01/15/2022).

[86] Michael Skinner. “Product Categorization with LSTMs and Balanced Pooling Views”. In: (2018), p. 8.

[87] Milan Straka et al. “RobeCzech: Czech RoBERTa, a Monolingual Contextualized Language Representation Model”. In: (May 24, 2021). DOI: 10.1007/978-3-030-83527-9_17. URL: https://arxiv.org/abs/2105.11314v2 (visited on 12/12/2021).

[88] Sandeep Subramanian et al. “Learning General Purpose Distributed Sentence Representations via Large Scale Multi-task Learning”. In: (Mar. 30, 2018). URL: https://arxiv.org/abs/1804.00079v1 (visited on 12/22/2021).

[89] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. *LSTM Neural Networks for Language Modeling*.

[90] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. Dec. 14, 2014. arXiv: 1409.3215 [cs]. URL: http://arxiv.org/abs/1409.3215 (visited on 11/17/2021).

[91] Shogo D Suzuki et al. “Convolutional Neural Network and Bidirectional LSTM Based Taxonomy Classification Using External Dataset at SIGIR eCom Data Challenge”. In: (2018), p. 5.

[92] Christian Szegedy et al. “Going Deeper with Convolutions”. In: (Sept. 17, 2014). URL: https://arxiv.org/abs/1409.4842v1 (visited on 01/15/2022).

[93] Mingxing Tan and Quoc V. Le. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. In: (May 28, 2019). URL: https://arxiv.org/abs/1905.11946v5 (visited on 01/15/2022).

[94] Mingxing Tan and Quoc V. Le. *EfficientNetV2: Smaller Models and Faster Training*. June 23, 2021. arXiv: 2104.00298 [cs]. URL: http://arxiv.org/abs/2104.00298 (visited on 01/15/2022).

[95] Tong Xiao et al. “Learning from Massive Noisy Labeled Data for Image Classification”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Boston, MA, USA: IEEE, June 2015, pp. 2691–2699. ISBN:

978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7298885. URL: http://ieeexplore.ieee.org/document/7298885/ (visited on 01/04/2022).

[96] Janusz Tracz et al. “BERT-based Similarity Learning for Product Matching”. In: *Proceedings of Workshop on Natural Language Processing in E-Commerce*. COLING-EcomNLP 2020. Barcelona, Spain: Association for Computational Linguistics, Dec. 2020, pp. 66–75. URL: https://aclanthology.org/2020.ecomnlp-1.7 (visited on 10/16/2021).

[97] Venkatesh Umaashankar, Girish Shanmugam S, and Aditi Prakash. *Atlas: A Dataset and Benchmark for E-commerce Clothing Product Categorization*. Version 1. Aug. 12, 2019. arXiv: 1908.08984 [cs, stat]. URL: http://arxiv.org/abs/1908.08984 (visited on 10/17/2021).

[98] Ashish Vaswani et al. *Attention Is All You Need*. Dec. 5, 2017. arXiv: 1706.03762 [cs]. URL: http://arxiv.org/abs/1706.03762 (visited on 11/22/2021).

[99] Ekansh Verma, Souradip Chakraborty, and Vinodh Motupalli. “Deep Multi-level Boosted Fusion Learning Framework for Multi-modal Product Classification”. In: (2020), p. 5.

[100] Bin Wang and C.-C. Jay Kuo. “SBERT-WK: A Sentence Embedding Method by Dissecting BERT-based Word Models”. In: (Feb. 16, 2020). URL: https://arxiv.org/abs/2002.06652v2 (visited on 12/12/2021).

[101] Paul Werbos. “Backpropagation through Time: What It Does and How to Do It”. In: *Proceedings of the IEEE* 78 (Nov. 1, 1990), pp. 1550–1560. DOI: 10.1109/5.58337.

[102] Pasawee Wirojwatanakul and Artit Wangperawong. *Multi-Label Product Categorization Using Multi-Modal Fusion Models*. Sept. 16, 2019. arXiv: 1907.00420 [cs, stat]. URL: http://arxiv.org/abs/1907.00420 (visited on 10/17/2021).

[103] Yandi Xia et al. “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*. EACL 2017. Valencia, Spain: Association for Computational Linguistics, Apr. 2017, pp. 663–668. URL: https://aclanthology.org/E17-2105 (visited on 10/17/2021).

[104] Wenhu Yu et al. “Multi-Level Deep Learning Based E-commerce Product Categorization”. In: (2018), p. 6.

[105] Tom Zahavy et al. *Is a Picture Worth a Thousand Words? A Deep Multi-Modal Fusion Architecture for Product Classification in e-Commerce*. Nov. 29, 2016. arXiv: 1611.09534 [cs]. URL: http://arxiv.org/abs/1611.09534 (visited on 11/21/2021).

[106] Petr Zelina. “Pretraining and Evaluation of Czech ALBERT Language Model”. In: (), p. 62.

[107] Aston Zhang et al. “Dive into Deep Learning”. In: (June 21, 2021). URL: https://arxiv.org/abs/2106.11342v2 (visited on 12/19/2021).

[108] Zhen Zuo et al. “A Flexible Large-Scale Similar Product Identification System in E-commerce”. In: (2020), p. 9.

List of Abbreviations

Acronyms

BERT Bidirectional Encoder Representations from Transformers.

BiT Big Transfer.

CBoW Continous Bag of Words.

CNN Convolutional Neural Network.

DAG Directed Acyclic Graph.

ELMo Embeddings from Language Model.

FAISS Facebook AI Similarity Search.

FFNN Feed Forward Neural Network.

GBT Gradient Boosted Trees.

GLaM Generalist Language Model.

GLM General Linear Models.

GloVe Global Vectors for Word Representation.

GPT Generative Pre-trained Transformer.

GPU Graphics Processing Unit.

GRU Gated Recurrent Unit.

k-NN k Nearest Neighbours.

LSTM Long-Short Term Memory.

MAE Mean Absolute Error.

MLM Masked Language Modelling.

MLP Multilayer perceptrons.

MSE Mean Squared Error.

MT Machine Translation.

NLP Natural Language Processing.

NN	Neural Network.
NSP	Next Sentence Prediction.
PPMI	Positive Point-wise Mutual Information.
ReLu	Rectified Linear unit.
RF	Random Forrest.
RNN	Recurrent Neural Networks.
RoBERTa	Robustly optimized BERT.
Seq2Seq	Sequence to Sequence.
SG	Skip-Gram.
TF-IDF	Term frequency and Inverse document frequency.
TPU	Tensors Processing Unit.
VGG	Visual Geometry Group.
ViT	Visual Transformer.

List of Figures

List of Tables

List of Attachments

Attatchment 1:	Github repository: https://github.com/Duuuscha/product_categorization
Attatchment 2:	Large images of image and text representation reduced into two dimensions using PCA, t-SNE, UMAP

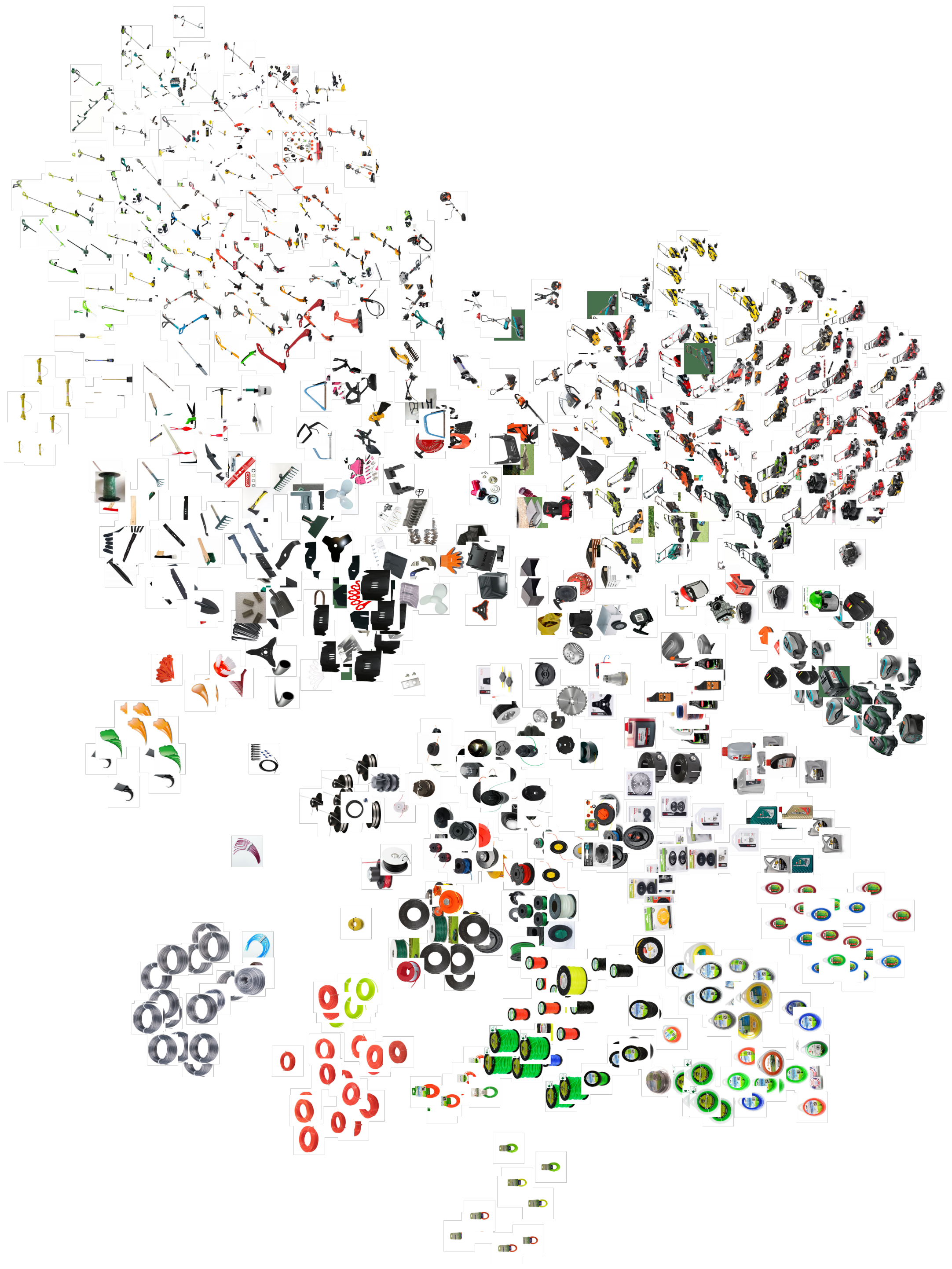


Fig. 39: UMAP of image feature embeddings

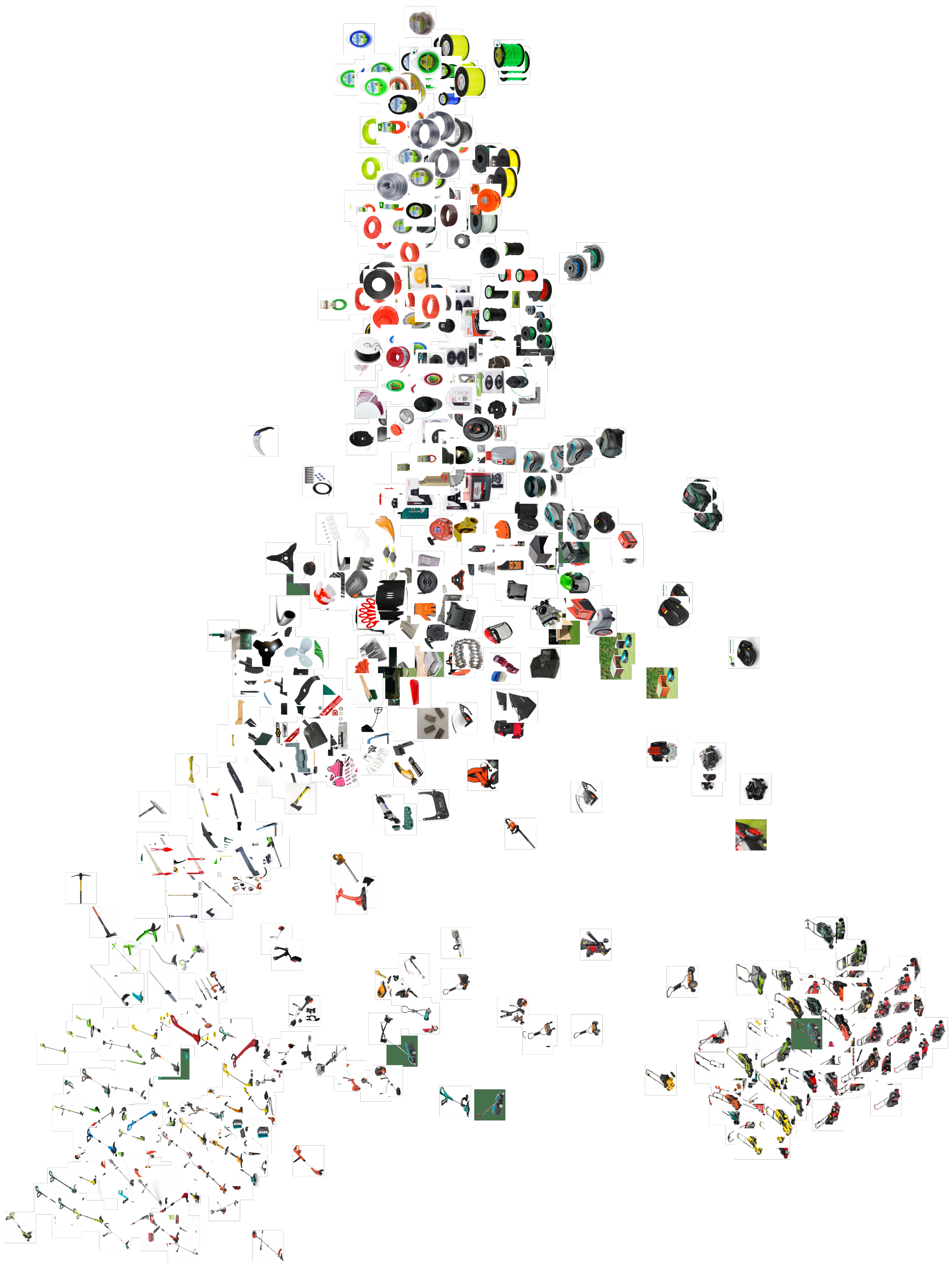


Fig. 40: PCA of image feature embeddings

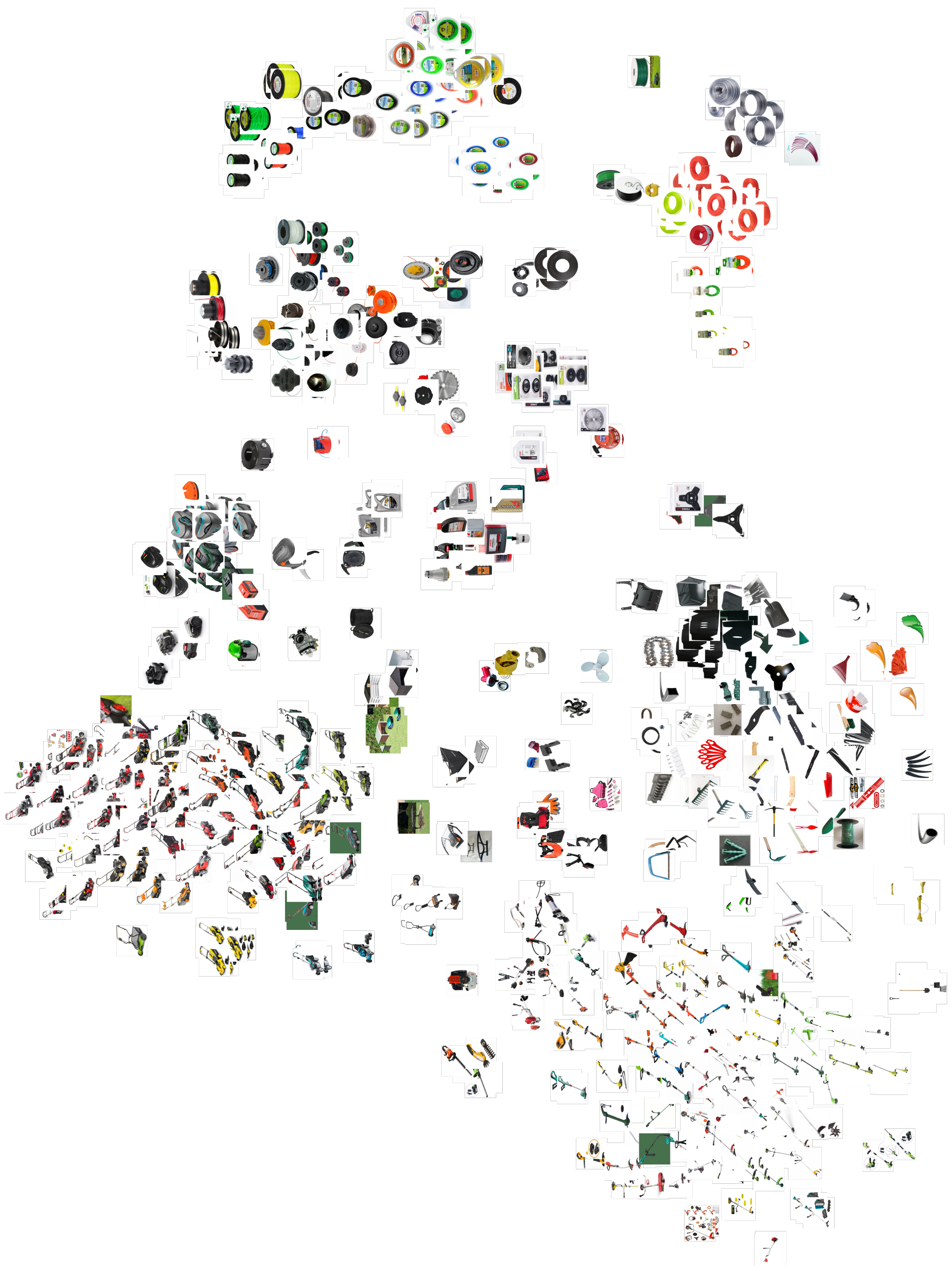


Fig. 41: t-SNE of image feature embeddings

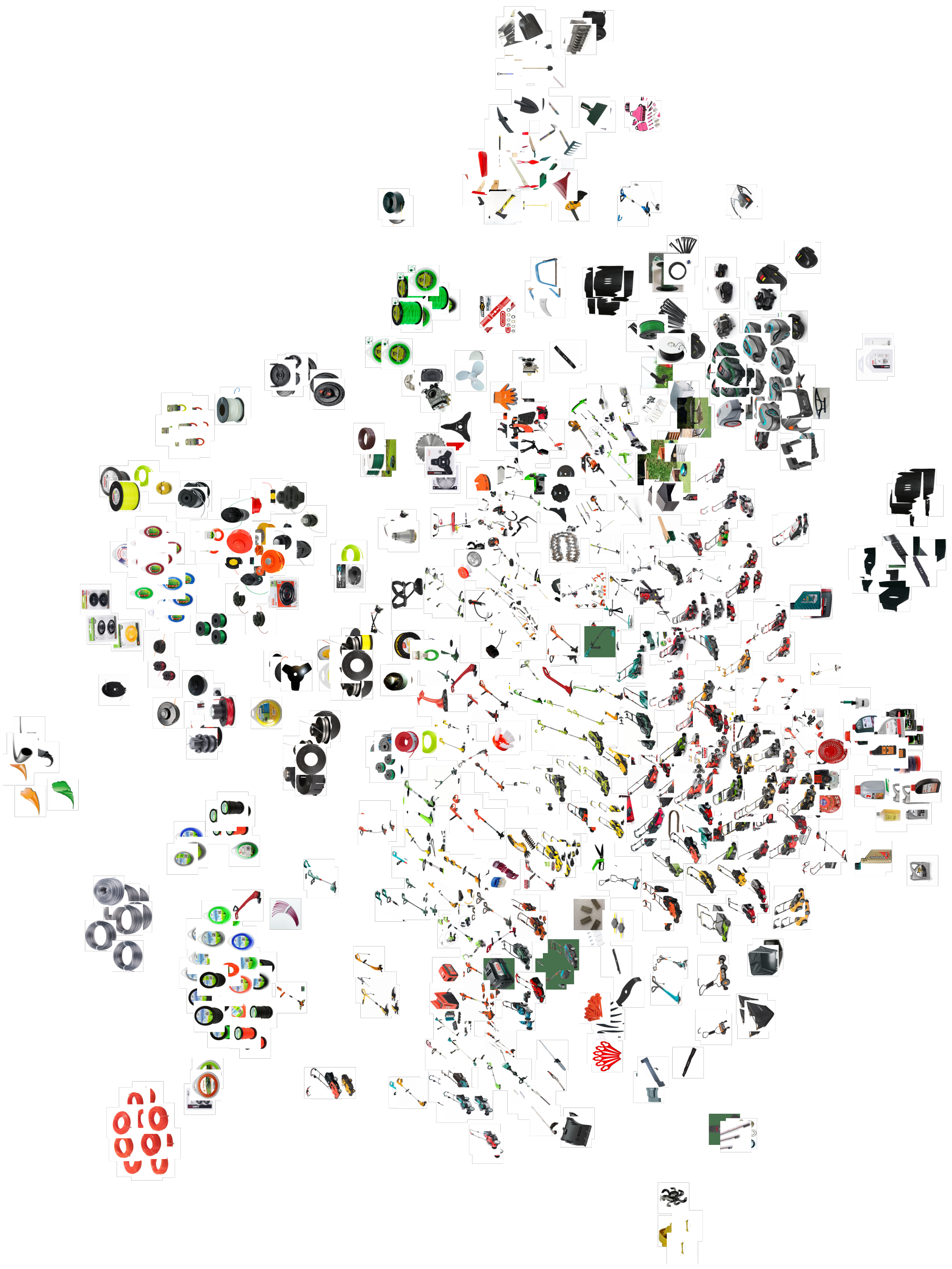


Fig. 42: t-SNE of text embeddings