MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE NATIONAL TECHNICAL UNIVERSITY OF UKRAINE «IGOR SIKORSKY KYIV POLYTECHNIC INSTITUTE»

On the rights of the manuscript

Lazarenko Denys

Text Classification with Deep Learning

Degree 124 —
«System analysis and control»

Thesis
bachelor of System analysis

Supervisor:
Ph.D. in Physico-mathematical Science, docent
Anton Maltsev

Kyiv — 2018

CONTENTS

p.

ABBREVIATION 4

INTRODUCTION 5

Chapter 1. Text classification 7

1.1 Statement of the classification problem 7

1.2 A short review of existing mathematical models, which can be used to solve the classification problem 8

1.3 Model evaluation and validation 9

1.3.1 Model Evaluation Applications 10

1.3.2 Model Evaluation Techniques 10

1.4 Classification metrics 11

1.5 Summary of the section 13

Chapter 2. Mathematical models and algorithms for text classification 14

2.1 Words representations 14

2.1.1 Bag-of-Words Approach 14

2.1.2 TF-IDF Approach 16

2.1.3 Embeddings 16

2.2 Deep learning algorithms for text classification 24

2.2.1 Convolution Neural Networks 24

2.2.2 Recurrent neural networks and their modifications 31

2.2.3 Advantages and drawbacks of different architectures 33

2.2.4 Summary of the section 34

Chapter 3. Testing and practical application of text classification using software 35

3.1 Software selection 35

3.2 Dataset selection and exploration 36

3.3 Data preparation 39

3.4 Network design and training 42
3.5 Summary of the section 43

Chapter 4. Classification results evaluation 44
4.1 General steps 44
4.2 Base line model 44
4.3 Convolution neural network 50
4.4 Convolution neural network with different regularization 57
4.5 Final model 63
4.6 Summary of the section 65

CONCLUSION 66

BIBLIOGRAPHY 67

LIST OF FIGURES 70

LIST OF TABLES 72

APPENDIX A 73

ABBREVIATION

ANN (Artificial Neural Network) - artificial neural network
LSTM (Long Short Term Memory) - Network with Long Redundant Memory
BiLSTM (Bidirectional Long Short Term Memory) is a two-way network with a long rectangular memory
CNN (Convolutional Neural Network) - Converging Neural Network
CTC (Connectionist Temporal Classification) - neural network timing classification
ReLU (Rectified Linear Unit) - activation function corrected linear module

INTRODUCTION

Relevance of the problem:

Nowadays, retail e-commerce sales are quickly increasing. A large online e-commerce websites serve millions of users' requests per day. Therefore it is necessary to make the processes of registrations and purchases as much convenient and fast as possible. For many classified platforms such as Amazon or Avito users who would like to create a new advertisement must to fill in the required fields: title, description, price and category. Choosing a category can be a tricky moment because in most cases users have to make a choice from more than hundred categories. Therefore, the problem of advertisement automatic category prediction is very important in terms of saving moderators' time and as a result, decreasing the number of necessary moderators to process them. The effective algorithms which would work with text data, have high accuracy and an appropriate speed are in high demand.

Objective of this thesis is building an effective model which have high accuracy and an appropriate speed for classification of advertisements at the e-commerce platform Jiji.ng. In particular:

1. consider different models that are used for texts classification
2. compare performance of Deep Learning models
3. prove the efficiency of Convoloutional neural netwoks for NLP related tasks

Scientific novelty: Consideration and implementation of Deep Neural Networks for solving text classification tasks, search of the most efficient architecture and parameters. Analysis of the best model and results.

Volume and structure. Thesis consists introduction, five sections, summary and two appendix. Full volume of thesis is 79 pages with 49 figures and 17 tables. Bibliography consists 23 cites.

In the first section I consider the problem of text classification and make an overview of the basic concepts, models and criteria which exists to solve the text classification problem.

In the second section, I present the process how to transform raw texts into features vectors. Also, I describe in details theoretical basis of deep neural networks which I use for my experiments, pay attention on their possible benefits and drawbacks for solving particular problems.

The third section examines the software used for experiments, as well as a code with explanations and diagrams that reproduce the process text processing, models selection and evaluation.

The fourth section contains the results and explanations of experiments. I compare models with different parameters and architecture. Also, I propose future development options.

Chapter 1. Text classification

1.1 Statement of the classification problem

Classification problem is a problem of identifying to which category a new observation belongs to. An example can be a situation when you receive a new email and the algorithm automatically decides whether it belongs to social network, promotions or business letters.

In text classification, we are given a description $d \in \mathbb{X}$ of a document, where $\mathbb{X}$ is the document space ; and a fixed set of classes $\mathbb{C} = \{c_1, c_2, \ldots, c_J\}$. Classes are also called categories or labels . Typically, the document space $\mathbb{X}$ is some type of high-dimensional space, and the classes are defined by people for the needs of an application, as in the examples China and documents that talk about multicore computer chips above. We are given a training set $\mathbb{D}$ of labeled documents d, where $d \in \mathbb{X} \times \mathbb{C}$. For example:

$$\langle \text{d, c} \rangle = \langle \text{Beijing joins the World Trade Organization, } \textit{China} \rangle \tag{1.1}$$

for the one-sentence document, Beijing joins the World Trade Organization and the class (or label) China. Using a learning method or learning algorithm , we then wish to learn a classifier or classification function γ that maps documents to classes:

$$\gamma : \mathbb{X} \to \mathbb{C} \tag{1.2}$$

This type of learning is called supervised learning because a supervisor (the human who defines the classes and labels training documents) serves as a teacher directing the learning process. We denote the supervised learning method by Γ and write $\Gamma(\mathbb{D}) = \gamma$. The learning method Γ takes the training set $\mathbb{D}$ as input and returns the learned classification function γ.

The classes in text classification often have some interesting structure such as the hierarchy in Figure 1.1. There are two instances in each of region categories, industry categories, and subject area categories. A hierarchy can be an important aid in solving a classification problem. Our goal in text classification is high accuracy of test data or new data - for example, the newswire articles that we will encounter tomorrow morning in the multicore chip example. It is easy to achieve high accuracy on the training set (e.g., we can simply memorize the labels). But high accuracy on the training set in general does not mean that the classifier will work well on new data in the application. When we use the training set to learn a classifier for test data, we make an assumption that training data and test data are similar or from the same distribution. [2, p.256-257]

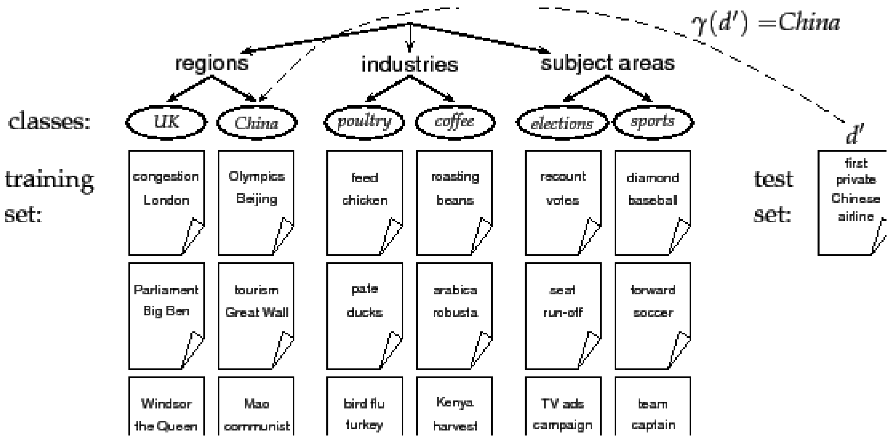


Figure 1.1 — Classes, training set, and test set in text classification.

1.2 A short review of existing mathematical models, which can be used to solve the classification problem

Supervised learning is the machine learning task of inferring a function from labelled training data. The training data consist of a set of training examples. Between inputs and reference outputs there may be some dependence, but it is unknown. On the basis of these data, it is necessary to restore the dependence. In order to measure the accuracy, a quality function can be introduced. [3, p.7] The diagram of the supervised learning process is presented in Figure 1.2

Here are some of the most important supervised learning algorithms:

1. Naive Bayes . [6]. [7]
2. Logistic Regression . [5]
3. Support Vector Machines (SVMs) . [8]

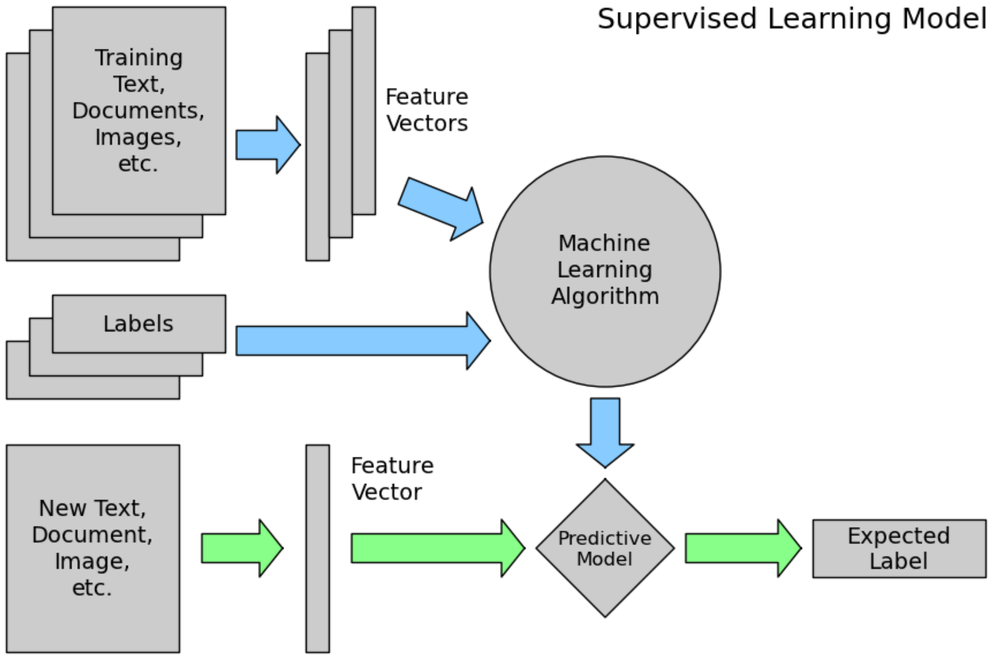


Figure 1.2 — Supervised learning workflow.

4. Decision Trees and Random Forests . [2]
5. Neural networks . [2]

1.3 Model evaluation and validation

Machine learning pipeline is not finished with a model evaluation. We want to estimate correctly future data by using special techniques and metrics that are suitable for a particular task.

Now let us find out what validation is for?

1. Validation helps to evaluate model performance, its quality, its ability to generalise.
2. Validation can be used to select the best model to perform on unseen data.
3. Overfitting of the model leads to the inconsistent and poor performance of the model on future data.

To better understand each point we need to examine it more deeply.

1.3.1 Model Evaluation Applications

Very important property of learning models is Generalization performance. In general we want to estimate the predictive performance of our model on future data. Therefore, it is necessary to use special techniques and metrics that are suitable for a particular task to track the performance of our models.

When we have a set of candidate models model selection helps us to increase the predictive performance by tweaking the learning algorithm and selecting the best performing model from a given hypothesis space.

- Before machine learning engineers find the best model, they make a bunch of experiments. Running a learning algorithm over a training dataset with different hyperparameter settings and various features will result in different models. The final goal is to select the best one from the set, ranking their performances against each other.

Algorithm selection - in most cases we deal with many algorithms to find the best one under the given circumstances. Therefore, we naturally need to compare different algorithms to each other, often regarding predictive and computational performance. Nevertheless, these three sub-tasks have some similarities. When we want to estimate the performance of a model, they all require different approaches.

1.3.2 Model Evaluation Techniques

Holdout method (simple train/test split) The holdout method is the most straightforward model evaluation technique. We take our labelled dataset and split it randomly into two parts: a training set and a test set.Then, we fit a model to the training data and predict the labels of the test set. And the fraction of correct predictions reflects our estimate of the prediction.

We don't want to train and evaluate our model on the same training dataset, because it will lead to overfitting - the model will simply memorise the training data, and it will generalise wrong to unseen data. [16]

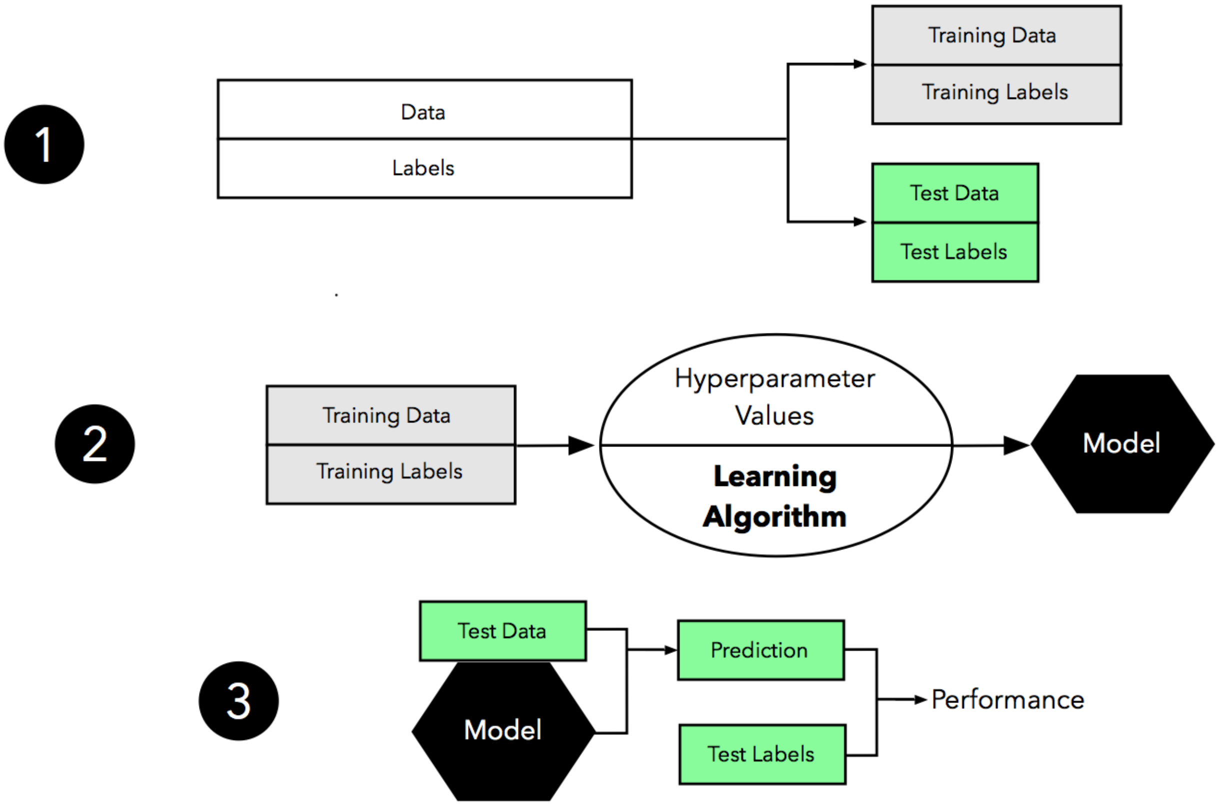


Figure 1.3 — Holdout method

1.4 Classification metrics

Classification problems are probably the most common type of ML problem, and therefore many metrics can be used to evaluate predictions of these problems. The most frequently used metrics for classification problems are:

1. Accuracy 1.3

   Accuracy simply measures what percent of your predictions was correct. It's the ratio between the number of correct predictions and the total number of predictions.

$$accuracy = \frac{correct}{predictions} \tag{1.3}$$

   Accuracy measures merely what percent of forecasts were correct. Accuracy is also the most misused metric. It is actually only suitable when there is an *equal number of observations in each class* (which is rarely the case) and that all *predictions and prediction errors are equally important, which is often not the case.

2. Confusion Matrix ??

   The confusion matrix is a handy presentation of the accuracy of a model with 2 or more classes. The table presents predictions on the x-axis and accuracy outcomes on the y-axis. The cells of the table are the number of predictions made by a machine learning algorithm.

		Prediction outcome		
		p	**n**	**total**
actual value	**p′**	True Positive	False Negative	P′
	n′	False Positive	True Negative	N′
	total	P	N	

   Figure 1.4 — Confusion matrix

   Confusion matrix allows one to compute various classification metrics.

3. Precision 1.4 and Recall 1.5

   Precision and recall are two metrics. But they are often used together. Precision answers the question: What percent of positive predictions was correct?

   $$precision = \frac{\#\ true\ positive}{\#\ true\ positive + \#\ false\ positive} \tag{1.4}$$

   Recall answers the question: What percent of the positive cases did you catch?

   $$recall = \frac{\#\ true\ positive}{\#\ true\ positive + \#\ false\ negative} \tag{1.5}$$

4. F1-score 1.6

   The F1-score (sometimes known as the balanced F-beta score) is a single metric that combines both precision and recall via their harmonic mean:

$$F_1 = 2\frac{precision * recall}{precision + recall} \tag{1.6}$$

Unlike the arithmetic mean, the harmonic mean tends toward the smaller of the two elements. Hence the F1 score will be small if either precision or recall is small.

1.5 Summary of the section

In the first section the relevance of the problem and the main concepts associated with it are considered, namely, classification, its formation, intellectual analysis. It is a review of the main methods and algorithms of classification and criteria for its management.

Since it is important to investigate not only the ways to classify texts, but also attempts to understand main features, which had the highest importance. It is important to study the theory of how to represent textual information before applying algorithms. Then, from the examined algorithms, the deep neural networks will be used.

With the criterion for further work, the top-5 accuracy and F1-score curve were selected.

Chapter 2. Mathematical models and algorithms for text classification

2.1 Words representations

In supervised learning domain, to perform classification tasks, our goal is usually to find a parametrized model, best in its class:

$$A(X, \hat{w}) : A(X, \hat{w}) \simeq f(X) \Leftrightarrow A(X, \hat{w}) = \arg\min_{w} \|A(X, w) - f(X)\| \tag{2.1}$$

Where $X \in R^{n \times m}$ - feature matrix (n observations with m features), $w \in R^m$ - vector of model parameters, $\hat{w}$ - "best" model parameters. However, as a candidate for X - all that we have is raw text input, algorithms can not use it as it is. In order to apply machine learning on textual data, firstly content should be transformed into the specific numerical format, other words it is necessary to form feature vectors. In Natural Language Processing automated feature extraction may be achieved in many ways. I will cover the most frequently used in the modern practices.

2.1.1 Bag-of-Words Approach

Bag-of-words is an unordered set of words, with their exact position ignored. [1, p.641],

In bag-of-words approach we work under the following assumptions:

- The text can be analyzed without taking into account the word/token order.
- It is only necessary to know which words/tokens of the text consists of and how many times.

Formally, there is a collection of texts $T_1, T_2, ..., T_n$. Unique tokens $w_1, w_2, ..., w_m$ are extracted to form a dictionary. Thus, each text T_i is represented by feature vector $F_j = \{x_{ij},\, j \in [1{,}m]\}$, where x_{ij} corresponds to number of occurrences of word w_j in text T_i.

Example: Our corpus is represented by 2 texts: ["The sun is yellow", "The sky is blue"]

Our tokens are simple unigrams, therefore there are 6 unique words: the, sun, is, yellow, sky, blue. Then, given corpus is mapped to feature vectors: $T_1 = (1,1,1,1,0,0)$, $T_2 = (1,0,1,0,1,1)$

Table 2.1 — Feature vector

Text	the	sun	is	yellow	sky	blue
T_1	1	1	1	1	0	0
T_2	1	0	1	0	1	1

Benefits:

- Despite its simplicity, it demonstrates good results.
- Fast preprocessing.
- Built-in in many scientific/NLP libraries

Drawbacks:

- Huge corpus usually leads to huge vocabulary size.
- Not memory-efficient: if we have corpus with 20 thousand texts then this textual corpus might spawn a dictionary with around 100 thousand elements. Thus, storing feature vectors as an array of type int32 would require 20000 x 100000 x 4 bytes 8GB in RAM.
- A bag of words is an orderless representation: throwing out spatial relationships between features leads to the fact that simplified model cannot let us distinguish between sentences, built from the same words, while having opposite meanings:"These paintings don't feel like ugly - buy them!" (positive) and "These paintings feel like ugly - don't buy them!" (negative)

In order to capture dependencies between words N-grams technique can be used. N-gram is a sequence of N basic tokens, which can be defined in different ways.

Word n-grams - catch more semantics:

- unigrams: "The sun is yellow." → ['The', 'sun', 'is' ...]
- bigrams: "The sun is yellow." → ['The sun', 'sun is' ...]
- 3-grams: "The sun is yellow." → ['The sun is ', 'sun is yellow']

In TF-IDF approach (term frequency - inverse document frequency), in addition to usual BoW-model, the following augmentation is made:

2.1.2 TF-IDF Approach

Instead of just counting up the overlapping words, the algorithms apply a weight to each overlapping word. The TF weight measures how many times the word occurs in a particular document while the IDF weight measures how many different documents a word occurs in and is thus a way of discounting function words. Since function words like the, of, etc., occur in many documents, their IDF is very low, while the TF of the content words is high. [1, p.647] Formally it can be defined as:

$$\begin{cases} TF(w,T) = n_{Tw} \\ IDF(w,T) = log\frac{N}{n_w} \end{cases} \implies TF\text{-}IDF(w,T) = n_{Tw}\ log\frac{N}{n_w} \quad \forall w \in W \tag{2.2}$$

where T corresponds to current document (text),
w - selected word in document T,
n_{Tw} - number of occurences of w in text T,
n_w - number of documents, containing word w,
N - total number of documents in a corpus.

$$\lim_{n_w \to N} TF\text{-}IDF(w,T) = 0 \tag{2.3}$$

2.1.3 Embeddings

Core idea: the meaning of a word is given by the words that frequently appear close-by. The fundamental ideas are stated in the following publications [9], [10].

1. Skip-gram model

I would like to start the consideration of embeddings methods with the key definitions of softmax 2.4 and sigmoid 2.5 functions,

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1} e^{x_j}} \tag{2.4}$$

$$\text{sigmoid} = \sigma(z) = \frac{1}{1+e^{-z}}. \tag{2.5}$$

The gradient of sigmoid function is follows:

$$\sigma'(z) = \sigma(z)(1-\sigma(z)) \tag{2.6}$$

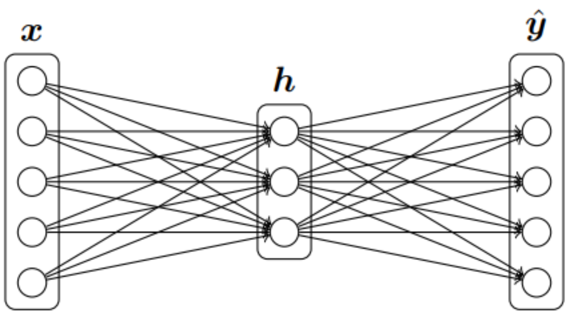


Figure 2.1 — Neural Network

where x is one-hot input vector, h - hidden layer, y is the one-hot label vector, and ŷ is the predicted probability vector for all classes.The neural network Figure 2.1 employs sigmoid activation function for the hidden layer, and softmax for the output layer and cross entropy cost 2.16 is used.

$$\text{CE}(\mathbf{y},\hat{\mathbf{y}}) = -\sum_i y_i \log \hat{y}_i \tag{2.7}$$

Now, we will compute the gradient of cross entropy:

$$\frac{\partial(\text{CE})}{\partial \hat{y}_i} = -\frac{y_j}{\hat{y}_i} \tag{2.8}$$

That leads,

$$\frac{\partial(\text{CE})}{\partial \theta_k} = \frac{\partial(\text{CE})}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial \theta_k} = -\frac{y_j}{\hat{y}_i}\frac{\partial \hat{y}_i}{\partial \theta_k} \tag{2.9}$$

Function $softmax$ for i-th output depends not only on its θ_i, but also on all other θ_k, the sum of which lies in the denominator of the formula for direct passage through the network. Therefore, the formula for back propagation "splits" into two: the partial derivative with respect to θ_i and θ_k:

$$
\begin{aligned}
\frac{\partial \hat{y}_i}{\partial \theta_i} = \frac{\partial}{\partial \theta_i}\left(\frac{e^{\theta_i}}{\sum_{j=1} e^{\theta_j}}\right) = \\
= \frac{e^{\theta_i}}{\sum_{j=1} e^{\theta_j}} - \left(\frac{e^{\theta_i}}{\sum_{j=1} e^{\theta_j}}\right)^2 = \\
= \hat{y}_i \cdot (1 - \hat{y}_i)
\end{aligned}
\tag{2.10}
$$

and (where $i \neq k$),

$$
\begin{aligned}
\frac{\partial \hat{y}_i}{\partial \theta_k} = \frac{\partial}{\partial \theta_k}\left(\frac{e^{\theta_i}}{\sum_{j=1} e^{\theta_j}}\right) = \\
= -\left(\frac{e^{\theta_i} e^{\theta_k}}{\sum_{j=1} e^{\theta_j}}\right) = -\hat{y}_i \hat{y}_k
\end{aligned}
\tag{2.11}
$$

After combination of equations 2.8, 2.10, 2.11,

$$
\frac{\partial(\mathrm{CE})}{\partial \theta_k} = \begin{cases} -y_j(1 - \hat{y}_k) & \text{for } i = k \\ y_j \hat{y}_k & \text{for } i \neq k \end{cases}
\tag{2.12}
$$

y_j should be non-zero, $k = j$ and $y_j = 1$, leads to,

$$
\frac{\partial(\mathrm{CE})}{\partial \theta_j} = \begin{cases} (\hat{y}_j - 1) & \text{for } i = j \\ \hat{y}_j & \text{for } i \neq j \end{cases}
\tag{2.13}
$$

Which is equivalent to,

$$
\frac{\partial(\mathrm{CE})}{\partial \boldsymbol{\theta}} = \hat{\mathbf{y}} - \mathbf{y}
\tag{2.14}
$$

Forward propagation is as follows:

$$
\mathbf{h} = \mathrm{sigmoid}(\mathbf{x}\mathrm{W}_{\mathbf{1}} + \mathbf{b_1})
\tag{2.15}
$$

$$
\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{h}\mathrm{W}_{\mathbf{2}} + \mathbf{b_2})
\tag{2.16}
$$

where $\mathrm{W}_{\mathbf{i}}$ and $\mathbf{b_i}$ ($i \in \{1,2\}$) are the weights and biases, respectively of the two layers. To optimize weights for each layer of neural network a back propagation algorithm is used. Therefore, it is necessary to calculate the gradients for each layer.

In order to simplify the notation used to solve the problem, define the following terms:

$$\begin{aligned} \mathbf{z}_1 \equiv & \quad \mathbf{x}\mathrm{W}_{\mathbf{1}} + \mathbf{b_1} \\ \mathbf{z}_2 \equiv & \quad \mathbf{h}\mathrm{W}_{\mathbf{2}} + \mathbf{b_2} \end{aligned} \tag{2.17}$$

Starting with the results from 2.7:

$$\frac{\partial J}{\partial \mathbf{z}_2} = \hat{\mathbf{y}} - \mathbf{y} \tag{2.18}$$

and

$$\frac{\partial \mathbf{z}_2}{\partial \mathbf{h}} = \mathrm{W}_{\mathbf{2}}^{\top} \tag{2.19}$$

Sigmoid (σ) derivative 2.6:

$$\frac{\partial \mathbf{h}}{\partial \mathbf{z}_1} \equiv \sigma'(z_1) \tag{2.20}$$

Combining these, and using $\cdot$ to denote element-wise product:

$$\frac{\partial J}{\partial z_i} = (\hat{\mathbf{y}} - \mathbf{y})\mathrm{W}_{\mathbf{2}}^{\top} \cdot \sigma'(\mathbf{z_1}) \tag{2.21}$$

Finally, using the results from Equation 2.19:

$$\frac{\partial J}{\partial \mathbf{W}^{(1)}} = (\hat{\mathbf{y}} - \mathbf{y})\mathrm{W}_{\mathbf{2}}^{\top} \cdot \sigma'(\mathbf{z_1}) \cdot \mathrm{X}^{\top} \tag{2.22}$$

$$\frac{\partial J}{\partial \mathbf{W}^{(2)}} = (\hat{\mathbf{y}} - \mathbf{y})\mathrm{h}^{\top} \tag{2.23}$$

We have everything to update our weights:

Now, turn definitely to skip-gram model shown in Figure 2.2 [13]:

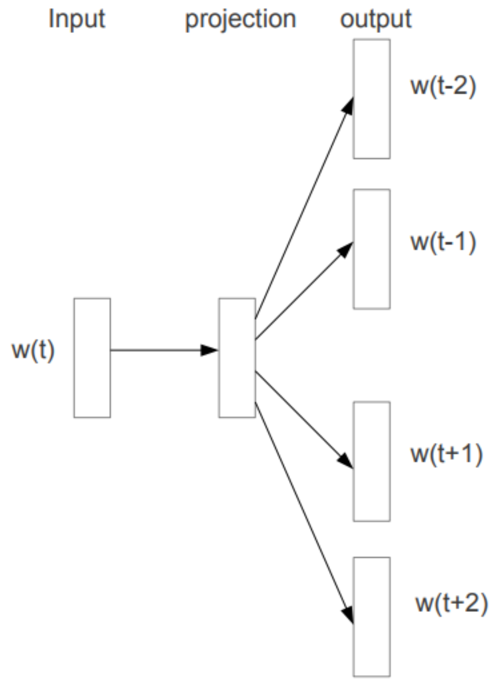


Figure 2.2 — The Skip-gram model architecture.

Now, let's transfer knowledge from above to our skip-gram model. We have a word vector $\boldsymbol{v}_c$ corresponding to the center word c for `skip-gram`, and word prediction is made with the `softmax` function:

$$\hat{\boldsymbol{y}}_{\boldsymbol{o}} = p(\ \boldsymbol{o} \mid \boldsymbol{c}\ ) = \frac{\exp\left(\boldsymbol{u}_o^\top \boldsymbol{v}_c\right)}{\sum_{j=1}^{|W|} \exp\left(\boldsymbol{u}_j^\top \boldsymbol{v}_c\right)} \tag{2.24}$$

where w denotes the w-th word and $\boldsymbol{u}_w$ ($w = 1,...,|\mathrm{W}|$) are the 'output' word vectors for all words in the vocabulary. Cross entropy cost is applied to this prediction and word o is the expected word (the o-th element of the one-hot label vector is one). $\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2,...,\boldsymbol{u}_{|\mathrm{W}|}]$ is the matrix of all the output vectors.

Applying cross-entropy cost to the softmax probability defined above:

$$J = -\log p = -\boldsymbol{u}_o^\top \boldsymbol{v}_c + \log \sum_{j=1}^{|V|} \exp\left(\boldsymbol{u}_j^\top \boldsymbol{v}_c\right) \tag{2.25}$$

Let $z_j = \boldsymbol{u}_j^\top \boldsymbol{v}_c$, and δ_j^i 2.26 be the indicator function, then

$$\delta_j^i = \begin{cases} 1, & \text{for } i = j \\ 0, & \text{for } i \neq j \end{cases} \tag{2.26}$$

$$\frac{\partial J}{\partial z_k} = -\delta_k^i + \frac{\exp\left(\boldsymbol{u}_i^\top \boldsymbol{v}_c\right)}{\sum_{j=1}^{|V|} \exp\left(\boldsymbol{u}_j^\top \boldsymbol{v}_c\right)} \tag{2.27}$$

Now, using the chain rule, we can calculate,

$$\begin{aligned} \frac{\partial J}{\partial \boldsymbol{v}_c} &= \frac{\partial J}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{v}_c} = \\ &= \sum_{j=1}^{|V|} \boldsymbol{u}_j^\top \left(\frac{e^{z_j}}{\sum_{k=1}^{|V|} e^{z_k}} - 1 \right) = \\ &= \sum_{k=1}^{|V|} \boldsymbol{P}(\boldsymbol{u}_j | \boldsymbol{v}_c) \boldsymbol{u}_j - \boldsymbol{u}_j \end{aligned} \tag{2.28}$$

For the 'output' word vectors $\boldsymbol{u}_w$'s

$$\begin{aligned} \frac{\partial J}{\partial \boldsymbol{u}_j} &= \frac{\partial J}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{u}_j} = \\ &= \boldsymbol{v}_c \left(\frac{\exp\left(\boldsymbol{u}_0^\top \boldsymbol{v}_c\right)}{\sum_{j=1}^{|V|} \exp\left(\boldsymbol{u}_j^\top \boldsymbol{v}_c\right)} - \delta_j^0 \right) \end{aligned} \tag{2.29}$$

We have calculated the gradient for one particular word, now we will generalize this to a number of words. We have a set of context words $[\texttt{word}_{c-\mathbf{m}}, \ldots, \texttt{word}_{c-\mathbf{1}}, \texttt{word}_c, \texttt{word}_{c+\mathbf{1}}, \ldots, \texttt{word}_{c+\mathbf{m}}]$, where $\mathbf{m}$ is the context size. We denote the 'input' and 'output' word vectors for $\texttt{word}_k$ as $\boldsymbol{v}_k$ and $\boldsymbol{u}_k$ respectively for convenience.

Also it is a good idea to use $F(\boldsymbol{o}, \boldsymbol{v}_c)$ (where $\boldsymbol{o}$ is the expected word) as a placeholder for $J(\boldsymbol{o}, \boldsymbol{v}_c, \ldots)$ cost functions.
Then we can rewrite cost function as follows:

$$J = \sum_{-m \leq j \leq m, j \neq 0} F(\boldsymbol{w}_{c+j}, \boldsymbol{v}_c) \tag{2.30}$$

where $\boldsymbol{w}_{c+j}$ refers to the word at the j-th index from the center.

The derivative of the loss has two terms, $\boldsymbol{w}_{c+j}$ and $\boldsymbol{v}_c$, which yields the following [12],

$$
\begin{aligned}
\frac{\partial J}{\partial \boldsymbol{w}_k} &= \\
&= \frac{\partial}{\partial \boldsymbol{w}_k} \sum_{-m \leq j \leq m, j \neq 0} F(\boldsymbol{w}_{c+j}, \boldsymbol{v}_c) = \\
&= \sum_{-m \leq j \leq m, j \neq 0} \frac{\partial F}{\partial \boldsymbol{w}_{i+j}} \delta_k^{i+j}
\end{aligned}
\tag{2.31}
$$

and

$$
\frac{\partial J}{\partial \boldsymbol{v}_c} = \sum_{-m \leq j \leq m, j \neq 0} \frac{\partial F}{\partial \boldsymbol{v}_c}
\tag{2.32}
$$

Now, we can update our weight using gradient descent algorithm:

$$
\begin{aligned}
w_k^{new} &= w_k^{old} - \eta \frac{\partial J}{\partial w_k} \\
v_c^{new} &= v_c^{old} - \eta \frac{\partial J}{\partial v_c}
\end{aligned}
\tag{2.33}
$$

where η is a learning rate.

After training the skip-gram model, we take the hidden layer weight matrix that will represent our words in the multidimensional space. If we make projection into two- dimensional space, we can have the following Figure 2.3:

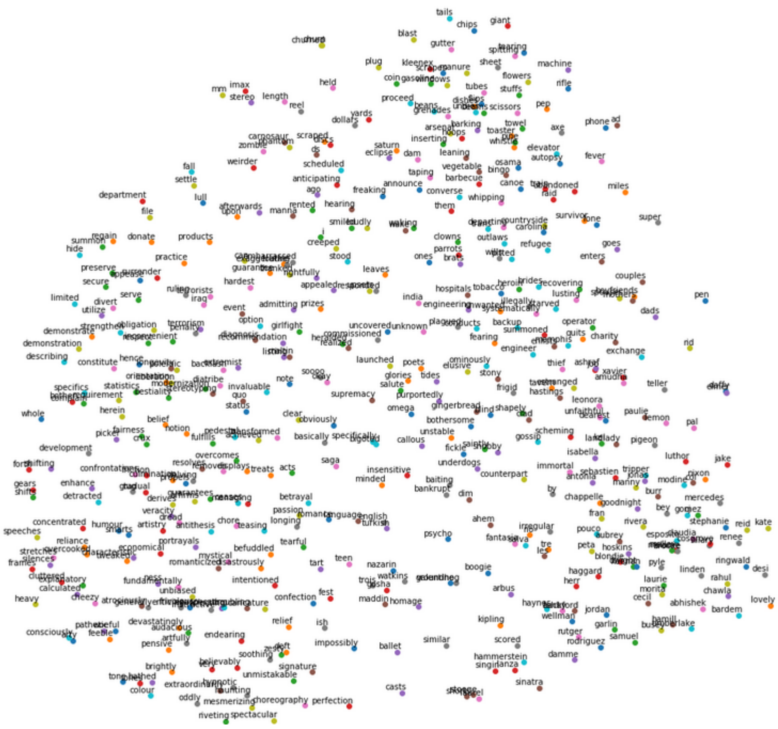


Figure 2.3 — Words representation

However, this type of architecture, where for each output we need to compute separate $softmax$ function is very expensive in terms of computational resources and as a result time. Therefore, there are different ways to approximate the expensive $softmax$ function. The most famous of them are:

– Negative Sampling technique
– Hierarchical Softmax

The only difference between Negative Sampling technique and the original model is that we introduce new loss function - negative sampling loss for the predicted vector $\boldsymbol{v}_c$, and the expected output word is $\boldsymbol{o}(\boldsymbol{u}_o)$. Assume that K negative samples (words) are drawn, and they are $\boldsymbol{u}_1,...,\boldsymbol{u}_k$, respectively for simplicity of notation ($k \in \{1,...,K\}$ and $o \notin \{1,...,K\}$). Again for a given word, $\boldsymbol{o}$, denote its output vector as $\boldsymbol{u}_o$. The negative sampling loss function in this case is,

$$J(\boldsymbol{u}_o, \boldsymbol{v}_c, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)) \tag{2.34}$$

where $\sigma(\cdot)$ is the sigmoid function.

As it can be clearly seen, now we make calculations not on the whole vocabulary V, but only on the part of it, which is randomly generated each time.

Hierarchical Softmax is an approximation which uses a binary tree to compute the necessary probability. This gives us a possibility to decompose calculating the probability of one word into a sequence of probability calculations. Balanced threes have a maximum depth of $log_2(|V|)$, which means that in the worst case we need to calculate $log_2(|V|)$ nodes to find the necessary probability of a certain word.

Both methods enable us a possibility to significantly decrease the amount of time for computation.

2.CBOW model This model is very similar to skip-gram, but CBOW predicts a target word from the bag of words context. From the practical point of view, skip-gram works well with a small amount of training data and represents well even rare words or phrases. CBOW is several times faster to train than the skip-gram and has slightly better accuracy for the frequent words. This model is shown in Figure 2.4 [13]:

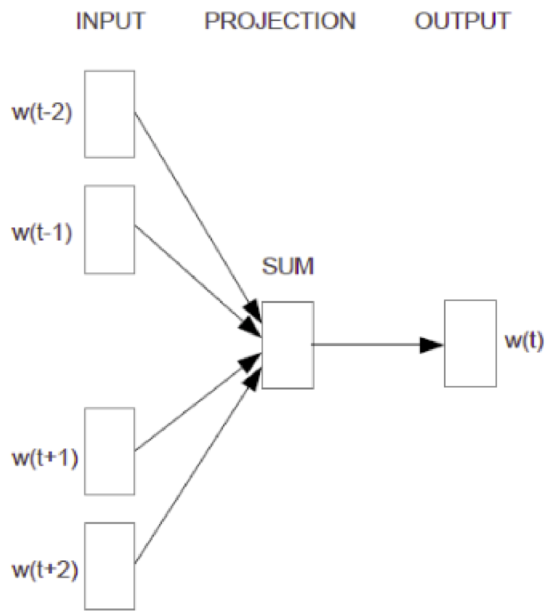


Figure 2.4 — The CBOW model architecture.

2.2 Deep learning algorithms for text classification

When people hear about NLP problems and neural networks in one context they probably think about Recurrent neural networks or their modification. However, recently some papers which apply CNN's to problems in Natural Language Processing were published and they got some interesting results [18] [19]. In this section I will consider both CNN and RNN models and their modifications.

2.2.1 Convolution Neural Networks

The model architecture, shown in Figure 2.5 [14], is a variant of the CNN architecture. Let $x_i \in \mathbb{X}$ be the k-dimensional word vector corresponding to the i-th word in the sentence, a sentence of length n. In general, let $x_{i:i+j}$ refers to the concatenation of words $\{x_i, x_{i+1}, ..., x_{i+j}\}$. [14]

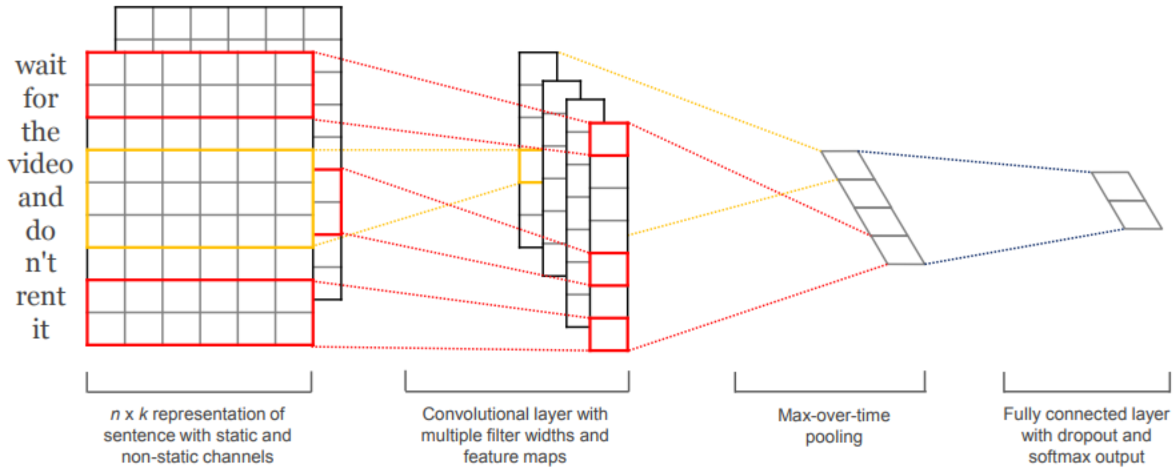


Figure 2.5 — Convolution Neural Networks architecture for text classification

Convolution

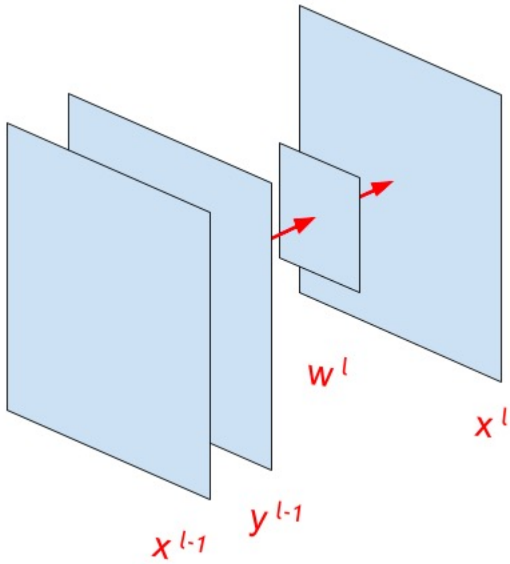


Figure 2.6 — Basic variables used in the convolution layer

In the convolution neural network, a limited matrix of small weights is used in the convolution operation, which is moved along the entire processed layer, forming after each shift the activation signal for the neuron of the next layer with the same position. The same matrix of weights, called kernel, is used for different neurons of the output layer. The schema of this process is illustrated in the Figure 2.6 [15].

The following equation 2.35 describes the words above into mathematical way:

$$x_{ij}^{l} = \sum_{a=-\infty}^{+\infty} \sum_{b=-\infty}^{+\infty} w_{ab}^{l} \cdot y_{(i \cdot s-a)(j \cdot s-b)}^{l-1} + b^{l} \qquad \forall i \in (0,...,N) \;\; \forall j \in (0,...,M) \tag{2.35}$$

where i, j, a, b - indexes of elements in matrices, s - step's size of convolution The superscripts l and $l-1$ are the indices of the network layers.
x_{l-1} - the output of some previous function, or the input of the network
y_{l-1} - x_{l-1} after passing the activation function
w_{l} - the convolution kernel
b_{l} - bias or offset
x_{l} - the result of the operation of convolution. That is, the operations which goe separately for each element i,j of the matrix x_{l}, whose dimension is (N, M).

The important moment which I should pay attention to is Central Core Element, because indexing of the elements takes place depending on the location of the central element. In fact, the central element determines the origin of the "coordinate axis" of the convolution kernel.

Activation functions

Activation function is a transformation which has a general view $y^{l} = f(x^{l})$. I did not cover all existing activations functions, I chose only those which were used in the current model.

1) ReLu 2.36, 2.7 - this activation function was used at Convolution layers. It has the following properties:

$$f_{ReLU} = max(0,x) \tag{2.36}$$

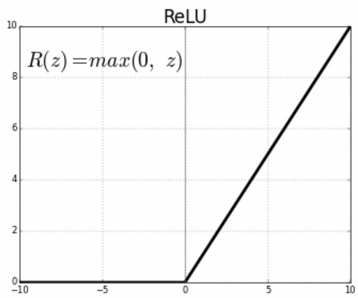


Figure 2.7 — ReLu activation function

2) Softmax 2.4 - I deal with multi-class classification, therefore this activation was picked.

Max pulling layer

This layer allows one to highlight important features on the maps of features obtained from convolution layer, gives an invariance to find the object on the cards, and also reduces the dimensionality of the maps, speeding up the network time. It works in the following way Figure 2.8: we divide our features from convolution layer into disjoint $m \times n$ regions, and take the maximum feature activation over these regions. These new features can be used for classification.

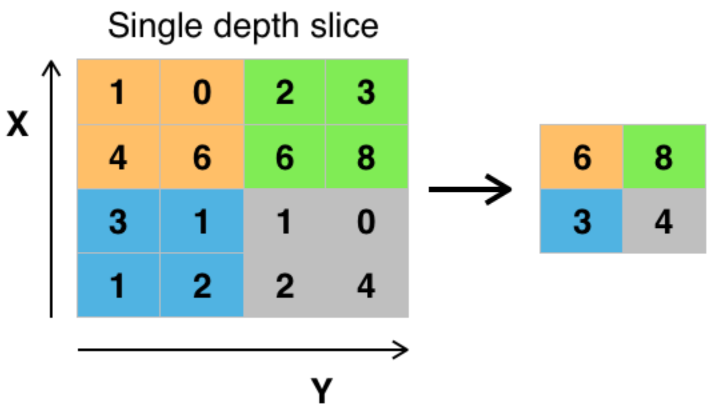


Figure 2.8 — Max pulling layer

Fully connected layer

After layers of the convolution and max pooling, we obtain a set of feature cards. We connect them into one vector and this vector will be fed into the fully connected network. The Figure 2.9 describes this stage.

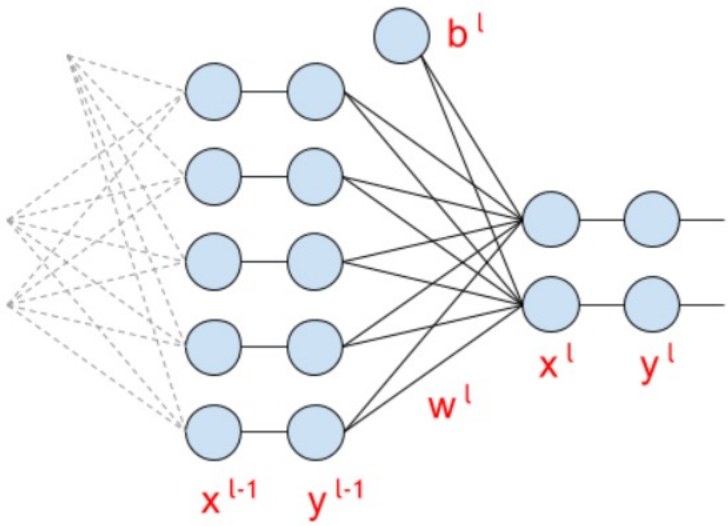


Figure 2.9 — Fully connected layer of CNN

$$x_i^l = \sum_{k=0}^{m} w_{ki}^l y_k^{l-1} + b_i^l \qquad \forall i \in (0,...,n) \tag{2.37}$$

in matrix representation:

$$X^l = Y^{l-1} W^l + B_i^l \tag{2.38}$$

Loss function for the model is Cross Entropy 2.7 described above.

Now after all components of CNN are known, we need to optimize weights for each layer. Therefore, it is necessary to derive the formula for back propagation through the loss function.

1) Hopefully, the gradient for loss function was already founded 2.10, 2.11, 2.12. Therefore, we have the following equation 2.39:

$$\begin{aligned} \frac{\partial J}{\partial x_i^l} = \sum_{k=0}^{n} \frac{\partial J}{\partial y_k^l} \frac{\partial y_k^l}{\partial x_i^l} = \frac{\partial J}{\partial y_0^l} \frac{\partial y_0^l}{\partial x_i^l} + ... \\ + \frac{\partial J}{\partial y_1^l} \frac{\partial y_1^l}{\partial x_i^l} + ... + \frac{\partial J}{\partial y_n^l} \frac{\partial y_n^l}{\partial x_i^l} \qquad \forall i \in (0,...n) \end{aligned} \tag{2.39}$$

or

$$
\begin{bmatrix} \frac{\partial J}{\partial x_0^l} & \frac{\partial J}{\partial x_1^l} & \cdots & \frac{\partial J}{\partial x_n^l} \end{bmatrix} =
$$
$$
= \begin{bmatrix} (\frac{\partial J}{\partial y_0^l}\frac{\partial y_0^l}{\partial x_0^l} + \frac{\partial J}{\partial y_1^l}\frac{\partial y_1^l}{\partial x_0^l} + \ldots + \frac{\partial J}{\partial y_n^l}\frac{\partial y_n^l}{\partial x_0^l}) & (\frac{\partial J}{\partial y_0^l}\frac{\partial y_0^l}{\partial x_1^l} + \frac{\partial J}{\partial y_1^l}\frac{\partial y_1^l}{\partial x_1^l} + \ldots + \frac{\partial J}{\partial y_n^l}\frac{\partial y_n^l}{\partial x_1^l}) & \ldots & (\frac{\partial J}{\partial y_0^l}\frac{\partial y_0^l}{\partial x_n^l} + \frac{\partial J}{\partial y_1^l}\frac{\partial y_1^l}{\partial x_n^l} + \ldots + \frac{\partial J}{\partial y_n^l}\frac{\partial y_n^l}{\partial x_n^l}) \end{bmatrix}
$$
$$
= \begin{bmatrix} \frac{\partial J}{\partial y_0^l} & \frac{\partial J}{\partial y_1^l} & \cdots & \frac{\partial J}{\partial y_n^l} \end{bmatrix} \begin{bmatrix} \frac{\partial y_0^l}{\partial x_0^l} & \frac{\partial y_0^l}{\partial x_1^l} & \cdots & \frac{\partial y_0^l}{\partial x_n^l} \\ \frac{\partial y_1^l}{\partial x_0^l} & \frac{\partial y_1^l}{\partial x_1^l} & \cdots & \frac{\partial y_1^l}{\partial x_n^l} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial y_n^l}{\partial x_0^l} & \frac{\partial y_n^l}{\partial x_1^l} & \cdots & \frac{\partial y_n^l}{\partial x_n^l} \end{bmatrix} \tag{2.40}
$$

Next, we should update the weights of fully connected layer matrix w^l.

$$
\frac{\partial J}{\partial w^l} = \frac{\partial J}{\partial y^l}\frac{\partial y^l}{\partial x^l}\frac{\partial x^l}{\partial w^l} = \delta^l \cdot \frac{\partial x^l}{\partial w^l} = \left(y^{l-1}\right)^T \cdot \delta^l \tag{2.41}
$$

and b^l

$$
\frac{\partial J}{\partial b^l} = \delta^l \tag{2.42}
$$

Equation for back propagation through y^{l-1}

$$
\frac{\partial J}{\partial y^{l-1}} = \delta^l \cdot \frac{\partial x^l}{\partial y^{l-1}} = \delta^l \cdot (w^l)^T = \delta^{l-1} \tag{2.43}
$$

After this, we need to go with backprop through the layer of max pulling Figure 2.10. The error "passes" only through those values of the original matrix, which were chosen by the maximum at the step of the max puling. The remaining error values for the matrix will be zero.

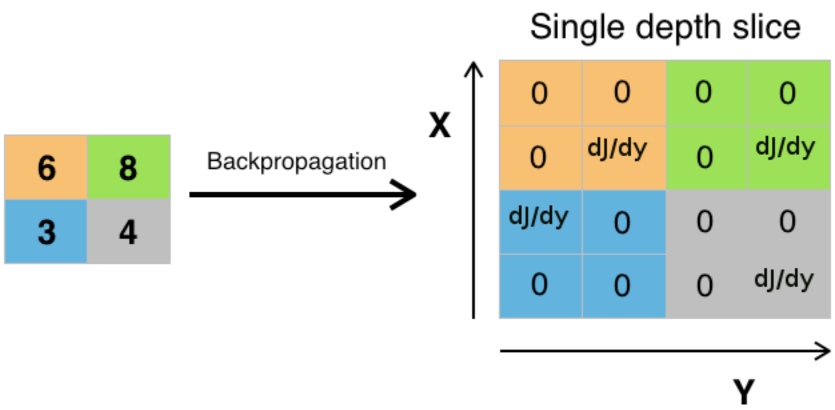


Figure 2.10 — Back propagation through max pulling layer

It is necessary to derive weights update for kernel Figure 2.11.

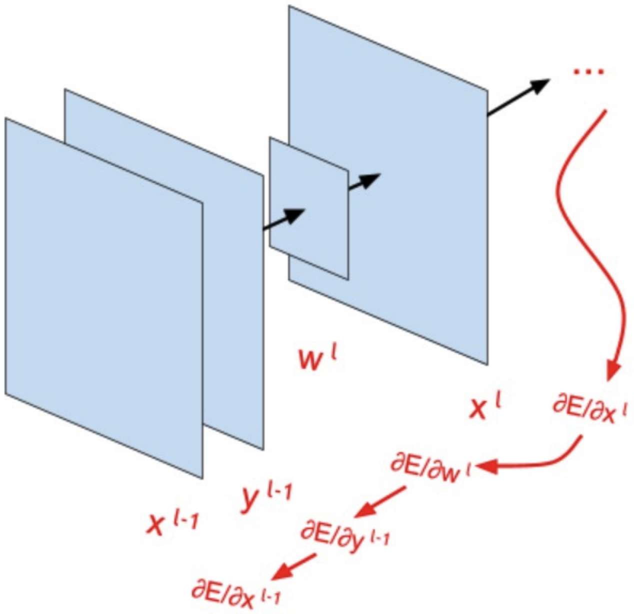


Figure 2.11 — Back propagation through convolution layer

$$\begin{aligned}
\frac{\partial J}{\partial w^l_{ab}} &= \sum_i \sum_j \frac{\partial J}{\partial y^l_{ij}} \frac{\partial y^l_{ij}}{\partial x^l_{ij}} \frac{\partial x^l_{ij}}{\partial w^l_{ab}} \\
&=^{(1)} \sum_i \sum_j \frac{\partial J}{\partial y^l_{ij}} \frac{\partial y^l_{ij}}{\partial x^l_{ij}} \cdot \frac{\partial \left(\sum_{a'=-\infty}^{+\infty} \sum_{b'=-\infty}^{+\infty} w^l_{a'b'} \cdot y^{l-1}_{(is-a')(js-b')} + b^l \right)}{\partial w^l_{ab}} \\
&=^{(2)} \sum_i \sum_j \frac{\partial J}{\partial y^l_{ij}} \frac{\partial y^l_{ij}}{\partial x^l_{ij}} \cdot y^{l-1}_{(is-a)(js-b)} \\
&\quad \forall a \in (-\infty, \dots, +\infty) \;\; \forall b \in (-\infty, \dots, +\infty)
\end{aligned} \tag{2.44}$$

all partial derivatives in the numerator, except those for which $a' = a, b' = b$, will be zero.

Derivation of the gradient for the bias element.

$$\frac{\partial J}{\partial b^l} = \sum_i \sum_j \frac{\partial J}{\partial y^l_{ij}} \frac{\partial y^l_{ij}}{\partial x^l_{ij}} \frac{\partial x^l_{ij}}{\partial b^l} = \sum_i \sum_j \frac{\partial J}{\partial y^l_{ij}} \frac{\partial y^l_{ij}}{\partial x^l_{ij}} \tag{2.45}$$

The derivation of the equation for backprop through the convolution layer.

$$\frac{\partial J}{\partial y^{l-1}_{ij}} = \sum_{i'} \sum_{j'} \frac{\partial J}{\partial y^l_{i'j'}} \frac{\partial y^l_{i'j'}}{\partial x^l_{i'j'}} \cdot w^l_{(i-i's)(j-j's)} \tag{2.46}$$

2.2.2 Recurrent neural networks and their modifications

A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed graph along a sequence. This allows it to exhibit dynamic temporal behavior for a time sequence. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to such tasks as natural language processing. [21] Recurrent neural networks are networks with loops in them, allowing information to persist. Figure 2.12 [22] illustrates RNN where, A looks at some input x_t and outputs a value h_t. A loop allows information to pass from one step of the network to the next.

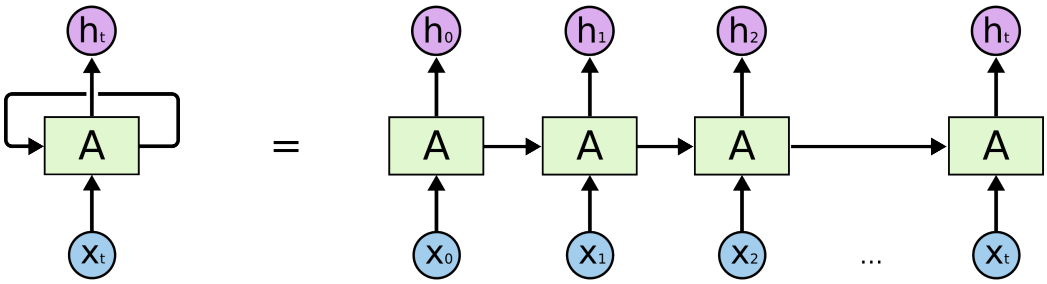


Figure 2.12 — The structure of Recurrent neural network

However, this type of NN has problems called "Vanishing Gradient and Gradient Explosion Problems". This problem was studied in detail in [23]. Therefore, the most successful for practical issues are modified RNN. I will use the most popular type - Long Short Term Memory networks.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior. This features is achieved by more complex structure. We can compare both architectures in Figure 2.13 [22] and Figure 2.14 [22].

The first step in LSTM is to decide what information we are going to throw away from the cell state. This decision is made by a sigmoid layer called the forget gate layer which looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{2.47}$$

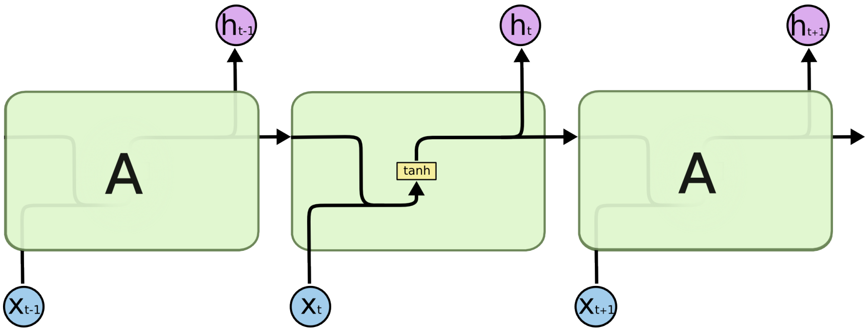


Figure 2.13 — The architecture of Recurrent neural network

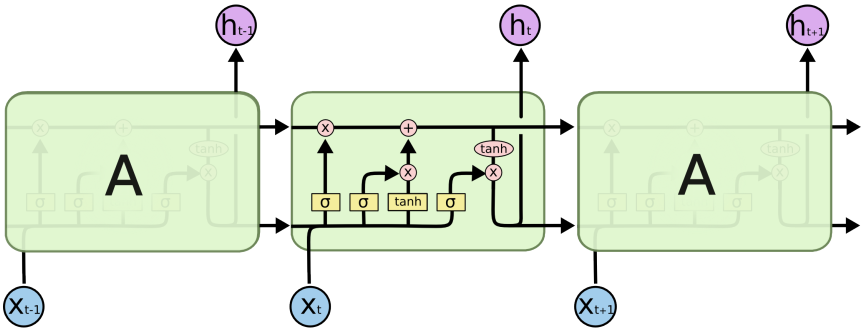


Figure 2.14 — The architecture of Long Short Term Memory neural network

The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the input gate layer decides which values we'll update. Next, a *tanh* layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state. In the next step, we combine these two to create an update to the state.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{2.48}$$

$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C) \tag{2.49}$$

To update the old cell state, C_{t-1}, into the new cell state C_t we multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$. This is new candidate values, scaled by how much we decided to update each state value.

$$\tilde{C}_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.50}$$

Finally, it is necessary to decide what will go to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to [22] .

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.51}$$

$$h_t = o_t * \tanh(C_t) \tag{2.52}$$

2.2.3 Advantages and drawbacks of different architectures

Recurrent Neural Networks have intuitive sense in NLP tasks. They reflect the way we process the language : reading sequentially from left to right. In contrary, CNNs which widely are used in Computer Vision have such features as Location Invariance and local Compositionality made intuitive sense for images, but not so much for NLP because it is important where a word appears in the sentence. Pixels close to each other are likely to be semantically related , but the same isn't always true for words. In many languages, parts of phrases could be separated by several other words. The compositional aspect isn't obvious either. Clearly, words are compose in some ways, like an adjective modifying a noun, but how exactly this works, what higher level representations actually "mean" isn't as obvious as in the Computer Vision case. Fortunately, this doesn't mean that CNNs don't work. All models are wrong, but some are useful. It turns out that CNNs applied to NLP problems perform quite well. The simple Bag of Words model is an obvious oversimplification with incorrect assumptions, but has nonetheless been the standard approach for years and lead to pretty good results.

A big argument for CNNs is that they are very fast. Very fast. Convolutions are a central part of computer graphics and are implemented on a hardware level on GPUs. Compared to something like n-grams, CNNs are also efficient in terms of representation. With a large vocabulary, computing anything more than 3-grams can quickly become expensive. Even Google does not provide anything beyond 5-grams. Convolutional Filters learn good representations automatically, without representing the whole vocabulary. It's completely reasonable to have filters of the size larger than 5. [20] In the next chapter, I will implement both architectures and evaluate them.

2.2.4 Summary of the section

The second section provides a theoretical overview of different methods for textual information encoding such as Bag-of-words and embeddings. We also deeply analyzed different architectures of neural networks which are useful for text classification problem.

Chapter 3. Testing and practical application of text classification using software

3.1 Software selection

The most popular languages in data analysis area are Python and R. Python3 language was chosen as more convenient for machine learning and beyond variety of libraries, including:

– pandas
– sklearn
– gensim
– keras
– tensorflow
– matplotlib
– psycopg2

The prototyping of models was made in separate Jupyter notebooks and then re-factored into project using the Python IDE for developers be JetBrains company - PyCharm. The server with Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz, 15 $\times$ 2 GB DDR3-1333 was used.

As a word vectors representation I used pre-trained word vectors which were trained on Wikipedia using fastText technique by Facebook research team and shared to the community [17]. These vectors in dimension 300 were obtained using the skip-gram model with default parameters.

To build deep neural network models my main requirements for the framework were:

– well described documentation
– simplicity of usage
– learning speed
– reliability

Among a wide range of frameworks available in open source: CNTK, Theano, MAXNET, Lasagne - Tensorflow framework was chosen. Tensorflow has a flexible architecture allows easy deployment of computation across a variety of platforms

(CPUs, GPUs, TPUs). To speed up the experiments with architecture of NN, I switched to a high-level neural networks API, written in Python and capable of running on top of TensorFlow.

3.2 Dataset selection and exploration

The target attribute to be predicted is the category of the advertisements. The category is represented by the two-level hierarchy. The parent category consists of 16 categories which then branches into 183 subcategories. These categories can be mapped together in the following way:

Table 3.1 — The hierarchy of categories

lvl2	category ID
lvl1	identifier of the parent category
name	category name

The dataset contains 455,000 ads classified into 183 categories. The sample of dataset can be seen from Table 3.2. First, let us get familiar with data and how it distributes between categories. As it can be clearly seen from Table 3.3 we have 2 categorical and 2 numerical variables and our data does not have any missing fields. It is good to start examining our data by each variable separately. From Table 3.4 we can see that data is not distributed equally between categories. First-level categories with number 6,5,1 consist almost of half of all advertisements. That means that our data is imbalanced and we can not use accuracy as only one metric for evaluation. Then if we take a look at second-level categories Table **??** we will see even the worst picture: one third of the date is concentrated in the categories which are marked 29,14 and 55 respectively. That means that it is necessary to use techniques to regularize distributions: under/over sampling or weight balanced.

For evaluation such metrics as categorical_accuracy , categorical_crossentropy, loss, timing and top_k_categorical_accuracy were chosen.

Table 3.2 — Structure of the data files

lvl1	lvl2	titles	descriptions
6	29	Clean Toyota Camry 2008 Silver	Fairly used Toyota 08 Camry with no problems V4 engine fabric seats and interior
5	25	Look Unique	Nice, quality, adorable,unique dress available now, whatsapp me
6	29	Mercedes Benz Ml 430 2001 Silver	mercedes benz ml430 , 2001 model in good condition , engine and gear box ok, ac , cd player
5	25	Versace Shirt Dress	Adorable versace shirt dress, whatsapp me on _large_number_
5	25	Addidas Jumpsuit	Nice quality addidas jumpsuit available, whatsapp me

Table 3.3 — Training set general information

Number of variables	4
Numeric variables	2
Categorical variables	2
Number of observations	455000
Total Missing (%)	0.0%
Total size in memory	57.7 MiB
Average record size in memory	48.0 B

Table 3.5 — Information about second-level categories

Value	**Count**	**Frequency** (%)
29	194714	19.5%
14	115471	11.5%
55	72050	7.2%
25	61308	6.1%
16	32719	3.3%
20	23298	2.3%
169	18743	1.9%
42	18490	1.8%
44	17740	1.8%
279	15544	1.6%
Other values (172)	429923	43.0%

Table 3.4 — Information about first-level categories

Value	**Count**	**Frequency** (%)
6	207695	20.8%
5	184934	18.5%
1	133135	13.3%
4	97799	9.8%
3	87574	8.8%
110	60214	6.0%
9	55459	5.5%
27	52419	5.2%
47	38985	3.9%
140	36442	3.6%
Other values (6)	45344	4.5%

Table 3.6 — Information about categorical features

Column	**Distinct count**	**Unique (%)**	**Missing (%)**
titles	619948	62.0%	0.00%
descriptions	869554	87.0%	0.00%

I decided to divide the whole dataset into train.csv / test.csv files which have the following structure:training set contains 400000 observation and control sample - 55,000 ads.

3.3 Data preparation

The following diagram describes steps which where made to process texts Figure ??.

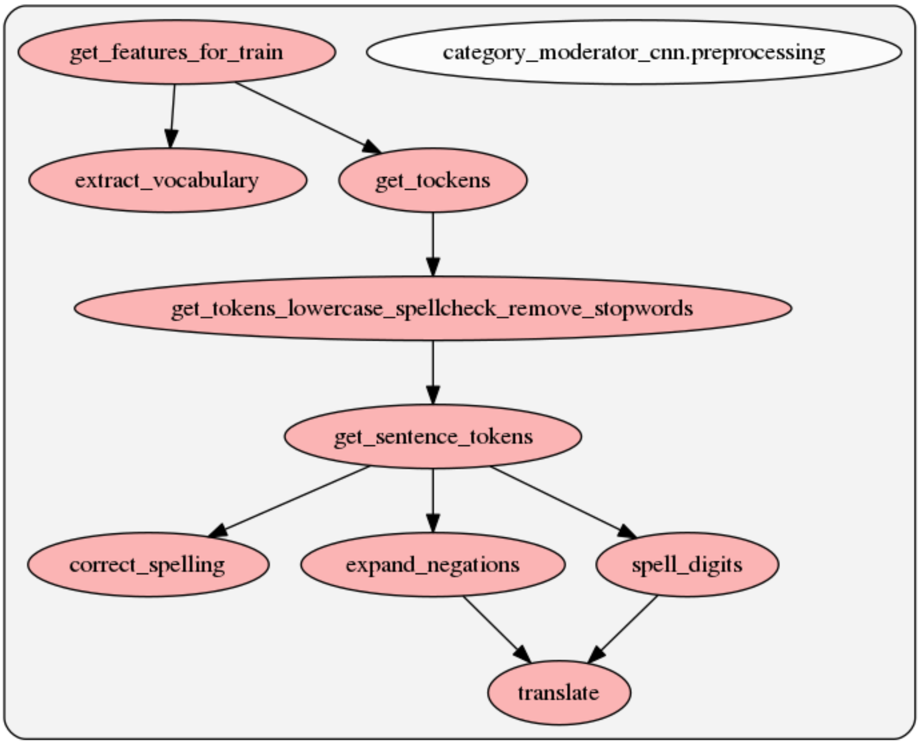


Figure 3.1 — Simplified event structure of data preprocessing

1.Tokenize Text

The given text was split by spaces and then lemmatized.

2. Remove infrequent words

Words appearing less than 3 times in the whole corpus were removed. It's a good idea to remove these infrequent words because a huge vocabulary will make our model slow to train and not all these words are presented in pretrained embeddings. Special attention should be paid to numbers which can be represented as price, year, mobile number. The telephone numbers occur really frequently in advertisements, but in most cases, they are unique because different people have their telephone numbers - the information about whether a telephone number presented in the text or not is meaningful. I used regular expressions to replace all numbers with the words _large_number_, _small_num_, _price_, _year_.

which gave me a possibility not to lose precious information.

r'[0-9a-z_]+@[a-z]+.[a-z]+': '_email_',
r'[0-9]5,20': '_large_number_',
r'[1-9][0-9]*k': '_price_',
r'[1-9][0-9]+?,[0-9]*': '_price_',
r'[1-9][0-9]*?,[0-9]* thousand': '_price_',
r'19[0-9]2': '_year_',
r'200[0-9]': '_year_',
r'201[0-8]': '_year_',
r'[0-9]+': '_small_num_',

3. Correct misspellings

I analyzed properly the most frequent cases where users do mistakes. Then I created a dictionary which consists wrong and right written words, so each time when the wrong written word appears it is replaced with the right written equivalent.

Table 3.7 — Simplified event structure of data preprocessing

Functions	**Explanation**
get_features_for_train	function which unify uploads of raw data and calls nested functions
extract_vocabulary	form vocabulary from unique words
get_tokens	parallel batch execution of texts preprocessing
get_tokens_lowercase_spellcheck	wrap over get_sentence_tokens which set necessary flags for it
get_sentence_tokens	receive single sentence, break it into tokens, make all of them to lowercase, correct spelling mistakes and replace specific words
correct_spelling	correct spelling mistakes using dictionary with around 15000 most common mistakes
expand_negations	replace mobile phones, dates, prices, and common abbreviation with specific words such as _year_, price, _large_number_ etc.
spell_digits	single digits are replaced with corresponding word 1 $\rightarrow$ one ...
translate	function which makes replacement of words

4. Build embeddings The following diagram describes steps which where made to build embeddings for each word Figure ??.

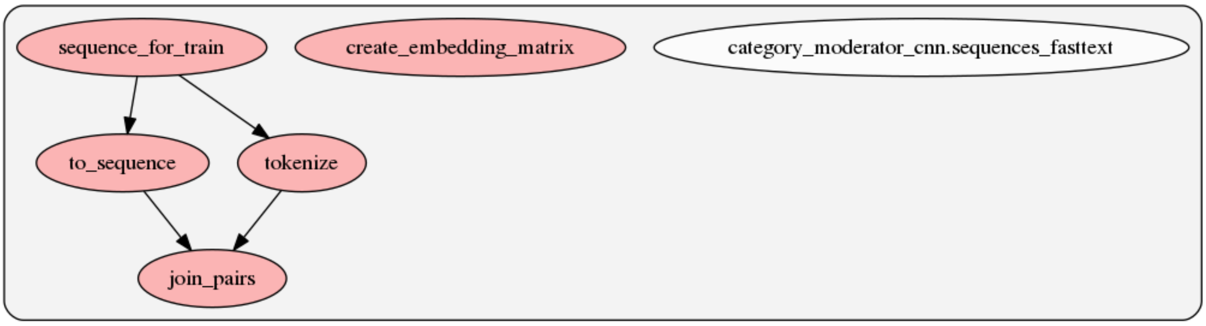


Figure 3.2 — Build embeddings structure

The input to Neural Networks are vectors, not strings. The mapping between words and indices was created, index_to_word, and word_to_index. For example, the word "buy" may be at index 201.

Table 3.8 — Simplified event structure of data preprocessing

Functions	**Explanation**
sequence_for_train	load preprocessed tokens from the previous model and use functions listed below to create sequences of indices which corresponds to particular word.
to_sequence	encode each word with the corresponding index and organize them into sequences of indices with the particular length
tokenize	build and save tokenizer which maps words and their indices
join_pairs	helpful function for transformation
create_embedding_matrix	build the embedding matrix for unique words from our dataset. It will have the structure word represented by its index in our vocabulary and the corresponding vector from the pre-trained embedding model.

3.4 Network design and training

The following diagram shows structure of the final module, which main responsibility is to train model. ??.

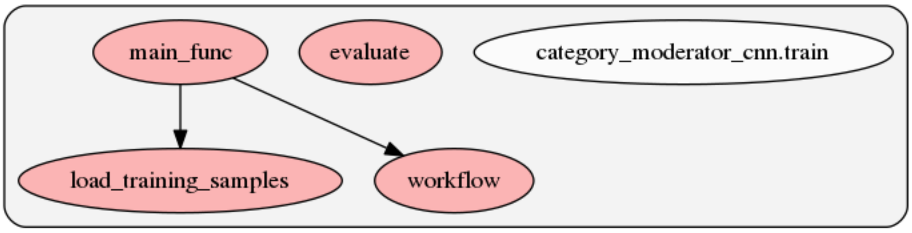


Figure 3.3 — 4

In this module I implemented different types of neural networks. Each network design module describes only one type of networks, which makes text classification. After the design of NN was implemented it should be trained. Apart from the training part this module saves training statistics ?? such as:

– categorical accuracy
– top k categorical accuracy
– batch timer
– categorical loss
– histograms of NN weights

These metrics will be used to judge the performance of our model. Such statistics give more information about changes in the neural network on each step of training. As tested network types on this step, I choose bidirectional recurrent network and convolutional network with different window sizes.

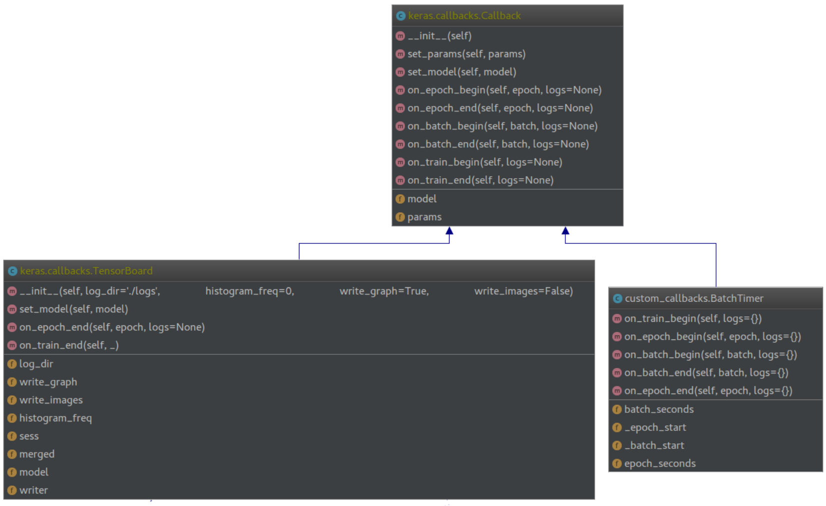


Figure 3.4 — Metrics which were logged

3.5 Summary of the section

This section argues the usage of the selected software. A detailed analysis of the data was provided. Also, we went through the implementation process: from the raw data to the trained model. An explanation of the most significant functions was made.

4.1 General steps

In this section I need to:

- visualize results of each training step, to understand hyperparameters better
- estimate networks by types
- chose hyperparameters to gain the best evaluation metrics
- train each network on different amount of epochs
- save network parameters and weights after each epoch. It will give me a possibility to restore the best model if network accuracy goes down.

Training time is a very important parameter. Therefore, I would like to train NN models which have nearly the same training time. To make it easier to understand the results of NN, a famous suite of visualization tools called TensorBoard was used. With TensorBoard it is possible to visualize graphs, plot quantitative metrics and show additional data like images passing through it.

4.2 Base line model

Firstly, I built the baseline model which CNN models should achieve in the future. I started this step with LSTM networks, as most often mentioned in the articles on NLP problems. At this step I was trying different configurations of LSTM parameters. The architecture which gave me the best score is shown in the Figure ??. According to my experiments bidirectional recurrent networks perform better than the same type and same size by parameters one directional networks, therefore it was chosen as the main layer. As input to the NN I gave two embedded sequences of words: one for the description of advert , another for title. This model has the bi-LSTM layer with 100 units. Each sequence goes through the bi-LSTM layer and then merged together. To solve overfitting in this model I used BatchNormalization and dropout. LSTM dropout rate was chosen manually and was equal to 0.332. Besides the bi-LSTM layer, there are two fully connected layers: one at the end because we want to estimate probabilities for each class, so it has 183 neurons in it and one in the middle with 130 neurons. As a training algorithm I chose Adam with default parameters: (lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0). As I have mentioned earlier, to better understand the network I visualize the histograms and percentile weights in each layer. The model was trained on 15 epochs.

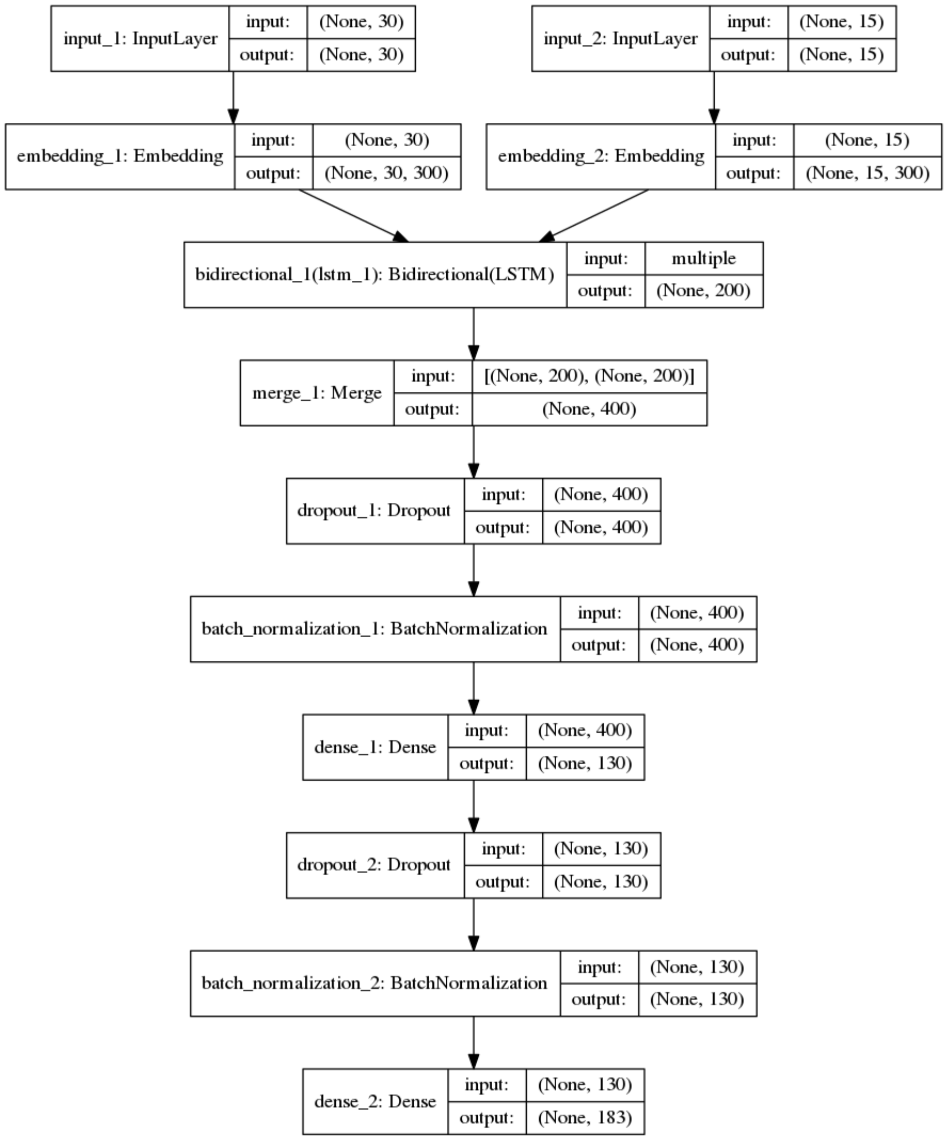


Figure 4.1 — Architectures of Bi-LSTM models with 100 units

As can be seen from the results of model training in Figure 4.2, categorical accuracy on the training set and validation set performs equally well: 0.7975 and 0.8203 respectively. Surprisingly, the results on train are slightly better than on training data.

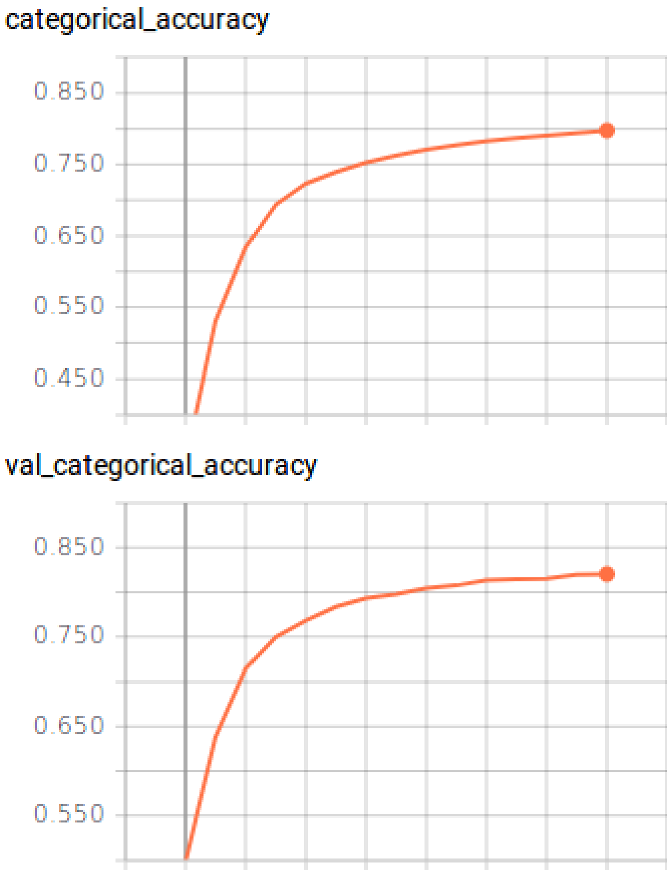


Figure 4.2 — Models train and validation categorical accuracy by epochs

Figure 4.3 shows that categorical cross entropy decrease on each epoch as it was expected. After 15 epochs results were the following: 0.8532 on the training set and 0.7478 on test.

The most important metric for this task was top categorical accuracy which can be seen in the Figure 4.4. On training data the model showed 0.9189 accuracy and on the test data - 0.9319.

Additional metrics which interested me in these experiments was time per epoch. These results can be seen in Figure 4.5.

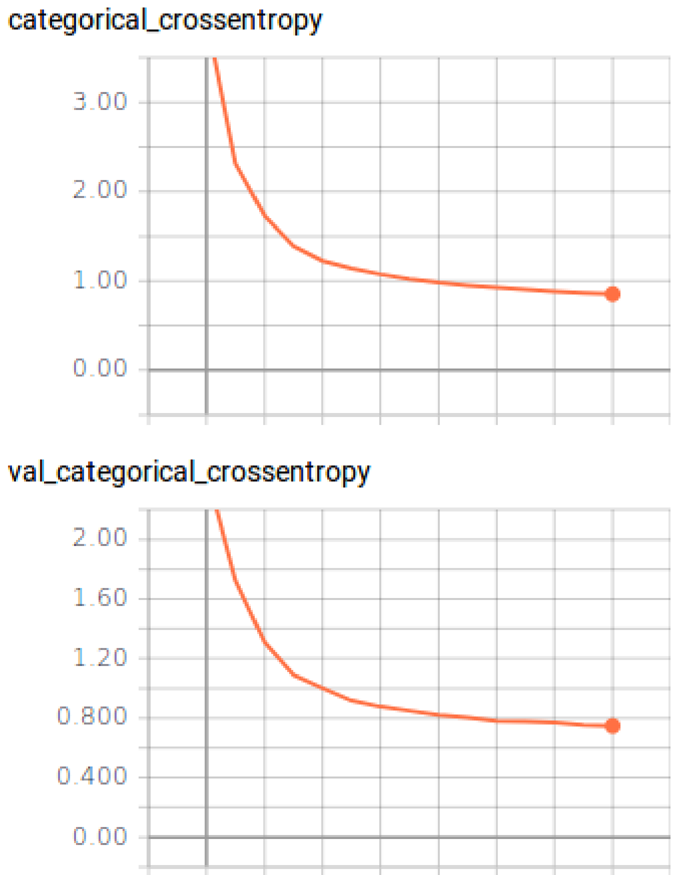


Figure 4.3 — Models train and validation category crossentropy by epochs

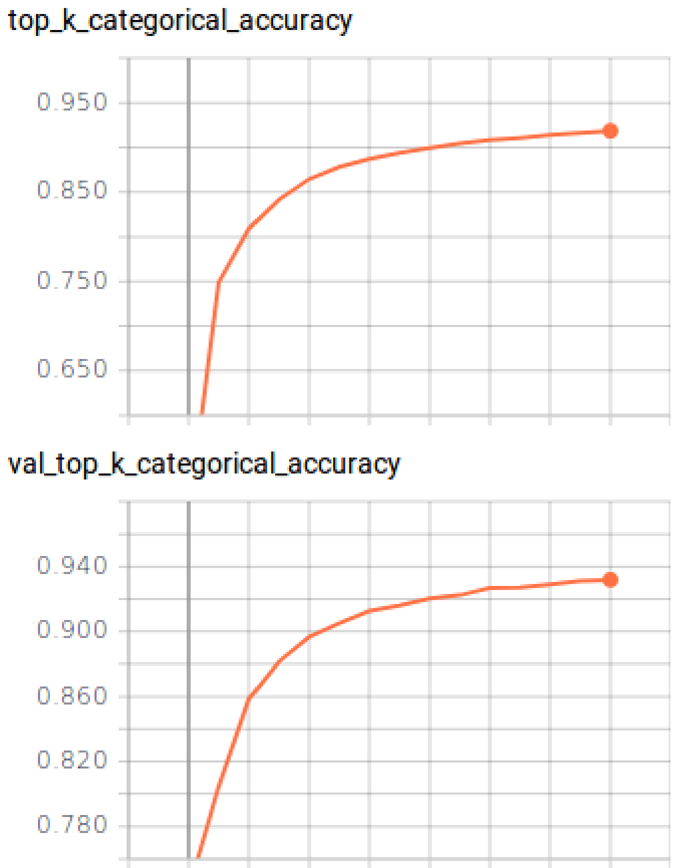


Figure 4.4 — Models train and validation top k accuracy by epochs

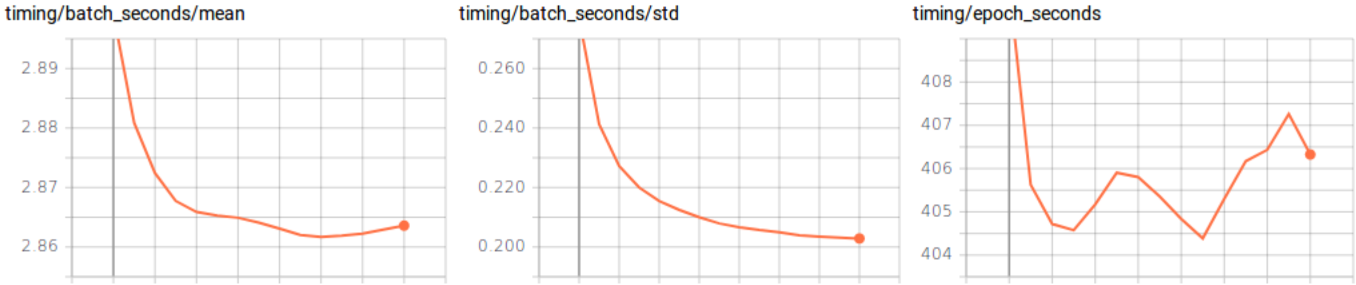
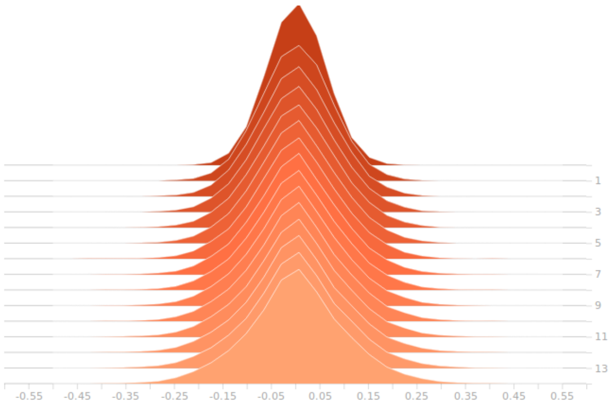
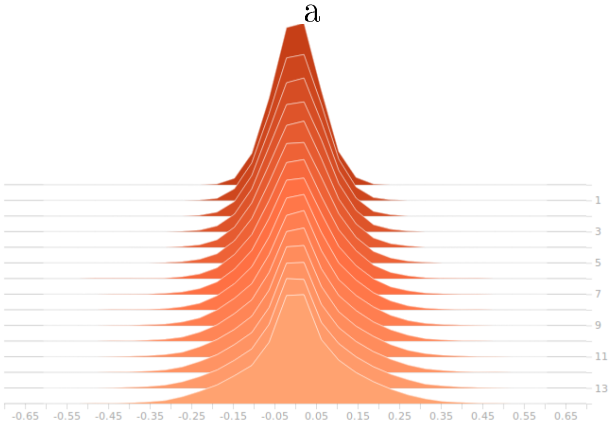


Figure 4.5 — Models batch time by epochs

Histograms give much information about what is happening with the network. I decided to pay attention to histograms which were output from the recurrent network - that show bi-LSTM behavior and weights for first layer of feedforward network. I did not find a descriptive explanation for how to interpret these kinds of graphics, so I followed basic knowledge from statistics.



a



b

Figure 4.6 — Bi-LSTM 100 units. Histogram of output from forward recurrent layers (a); histogram of weights from backward recurrent layers (b)

According to Figures 4.2, 4.3, 4.4 model was not overfitted. Histograms of outputs from feed forward and backward recurrent layers Figure 4.6 do not change significantly during training process. This can be explained by the fact that this layer does not train enough and I think that this network continues to learn thanks to FFNN part Figure 4.7. I think, that the best shape for the histogram of weights from first FFNN layer will be a normal distribution with higher variance, than after initialization. As we can see the FFNN layer has exactly such behavior which means that it learns some meaningful information. It is possible that more epochs will have a positive effect on the recurrent layer.

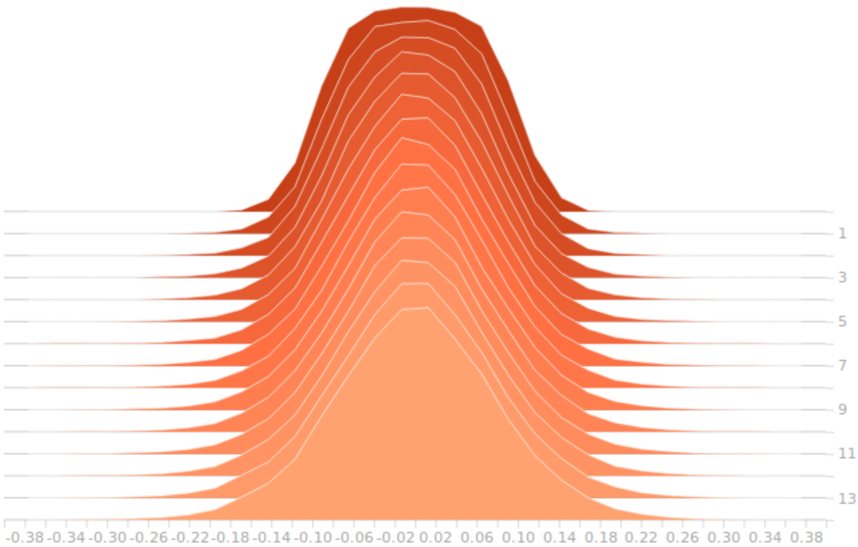


Figure 4.7 — Bi-LSTM 100 units. Histogram of weights from first FFNN layer.

The bi-LSTM model requires a significant amount of computational resources: especially RAM. Therefore, a batch size was chosen equal to 3048. We can see consumption of resources in Figure 4.8

Figure 4.8 — CPU resources which were used while training Bi-LSTM NN.

4.3 Convolution neural network

In this part, I would like to show that CNN can perform not worse than recurrent based neural networks. Therefore, I implemented CNN architecture Figure ?? which was described in the section 2.2.1. I build three model with different numbers of filters. The smallest one contains 128 filters, middle one - 256 and the biggest - 512. All models were trained with the same number of filter: 3, 4, 5 and dropout rate equals to 0.5. I did not use regularization for convolution layers, but according to my experience, big pooling window also prevent overfitting.Models were trained on 15 epochs.

Let us compare results which different models give.

Table 4.1 — Analysis of categorical accuracy

Number of filters	**Train**	**Test**
128	0.8165	0.8126
256	0.8532	0.8251
512	0.8885	0.8338

Table 4.2 — Analysis of category crossentropy

Number of filters	**Train**	**Test**
128	0.8060	0.8434
256	0.6340	0.7746
512	0.4731	0.7331

Table 4.3 — Analysis of top k accuracy

Number of filters	**Train**	**Test**
128	0.9259	0.9191
256	0.9495	0.9286
512	0.9696	0.9342

According to the results 4.9, 4.10, 4.11, 4.12, I can assume that models having more filters show better accuracy. However, it is noticeable that model with 256 and 512 filters have quite different results on train and test sets (3-5% difference). It can be interpreted as overfitting of these models.

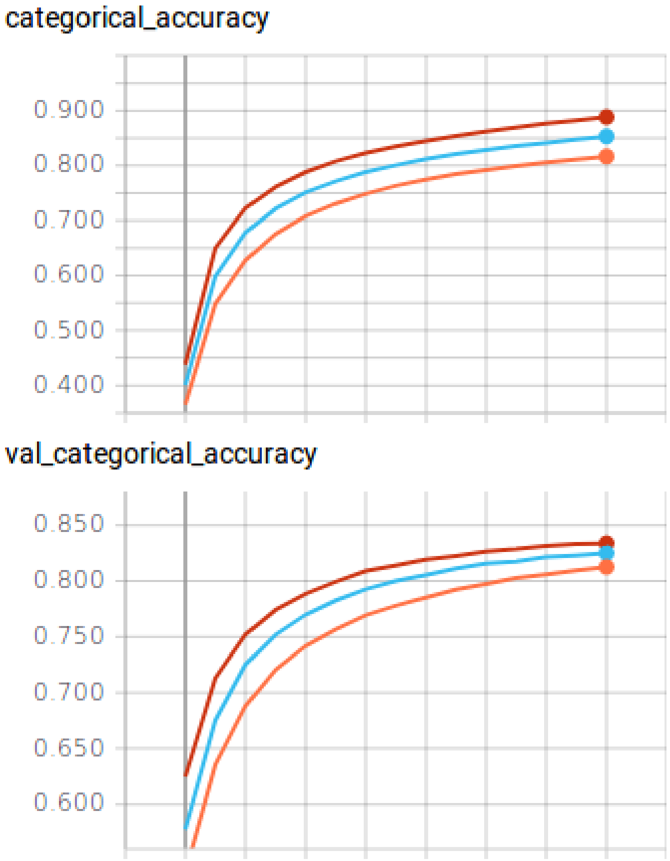


Figure 4.9 — Models train and validation categorical accuracy by epochs

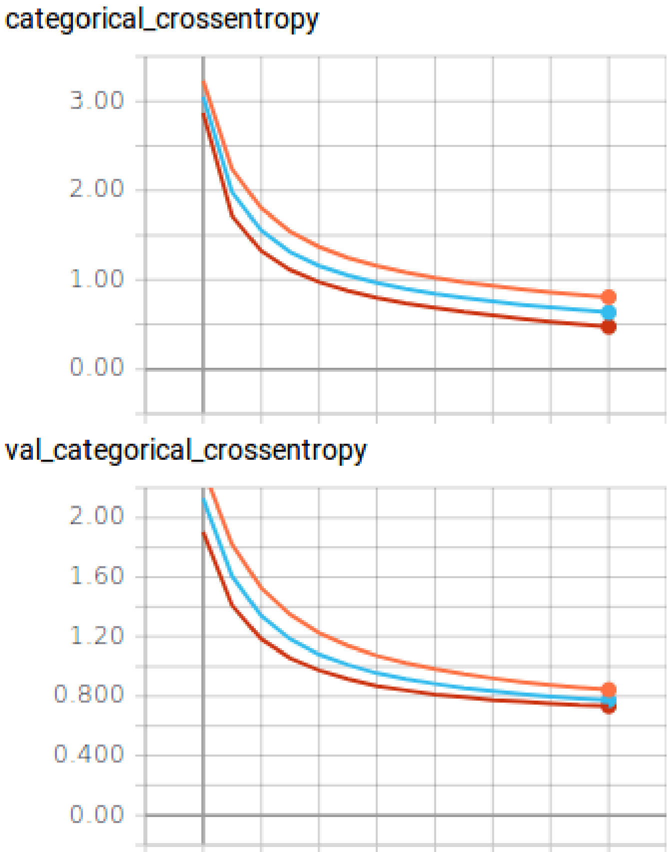


Figure 4.10 — Models train and validation category crossentropy by epochs

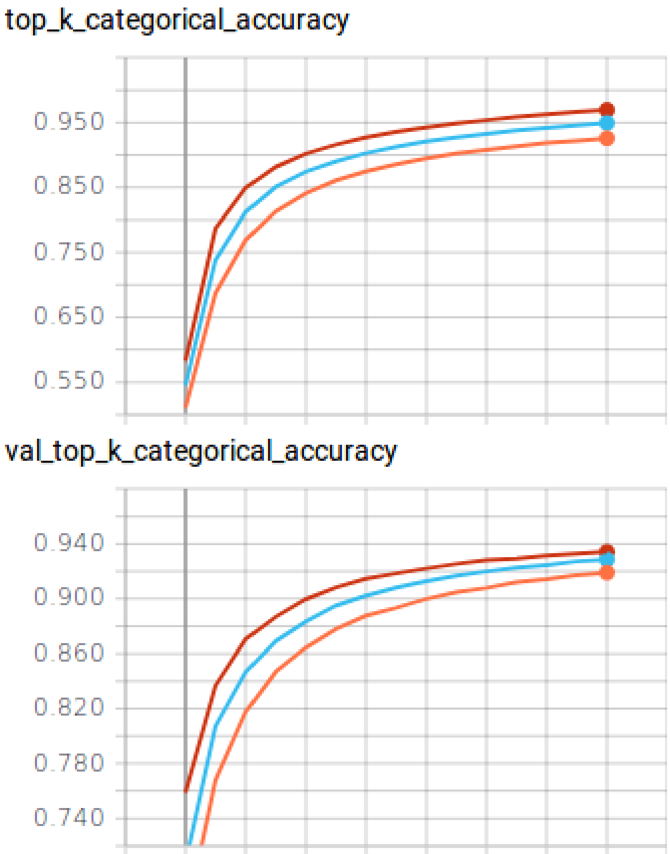


Figure 4.11 — Models train and validation top k accuracy by epochs

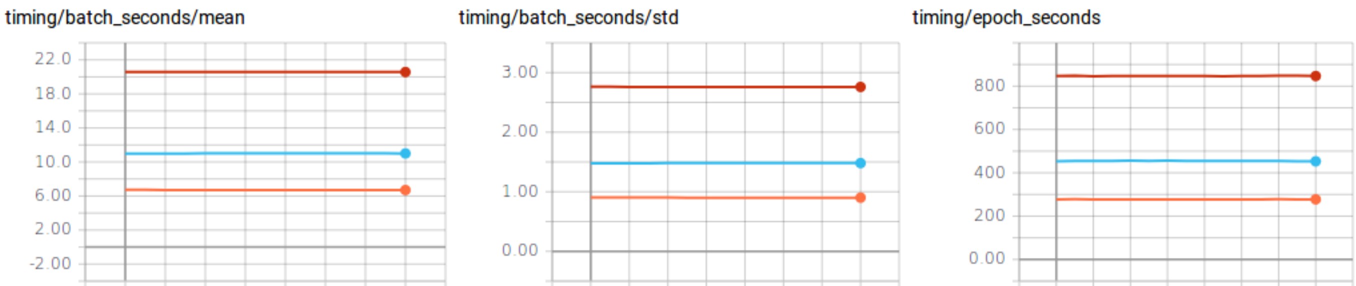
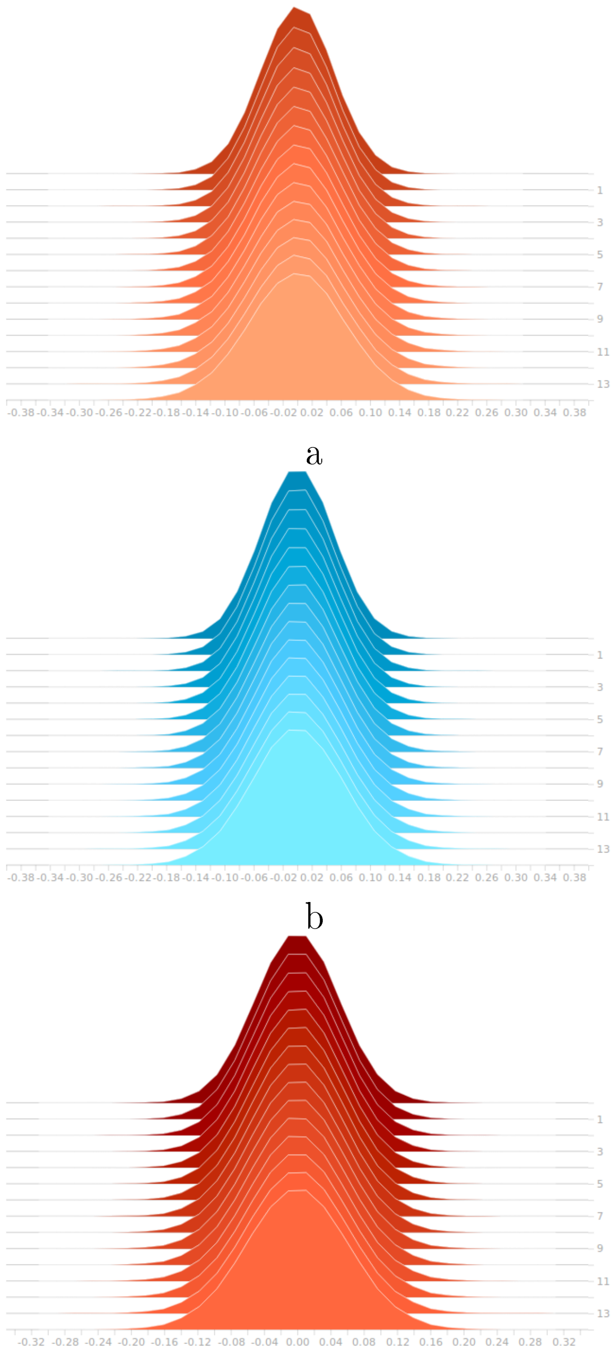


Figure 4.12 — Models batch time by epochs

Histograms of convolution layers Figure 4.13 look almost the same. They do not change significantly from the initial distribution. It can mean these layers do not learn much.



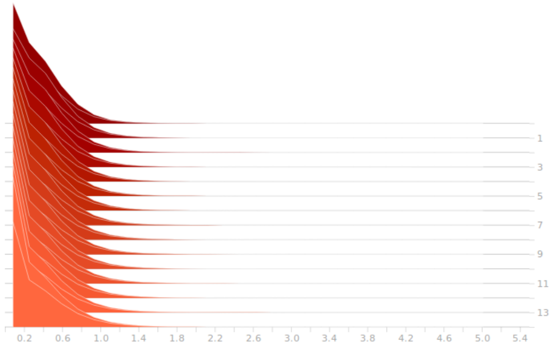
a

b

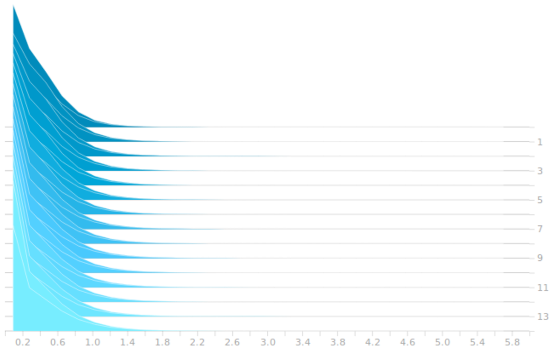
c

Figure 4.13 — Convolutional model (a) 128;(b) 256; (c) 512 filters for each sizes [3, 4, 5]. Histogram of convolution layers

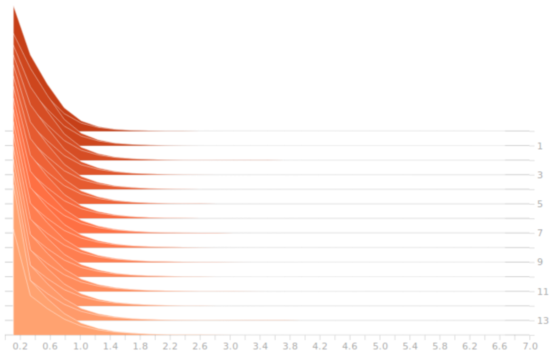
Merged layers Figure 4.14 are also very similar, because of convolution layers.



a



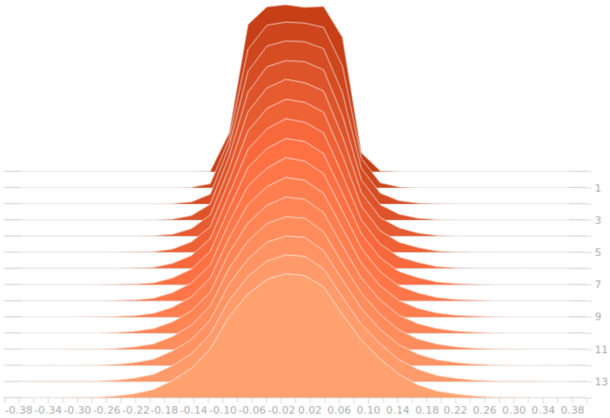
b



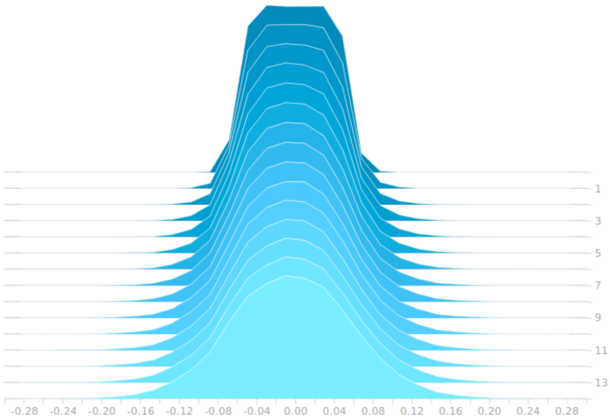
c

Figure 4.14 — Convolutional model (a) 128;(b) 256; (c) 512 filters for each sizes [3, 4, 5]. Histogram of merged layers

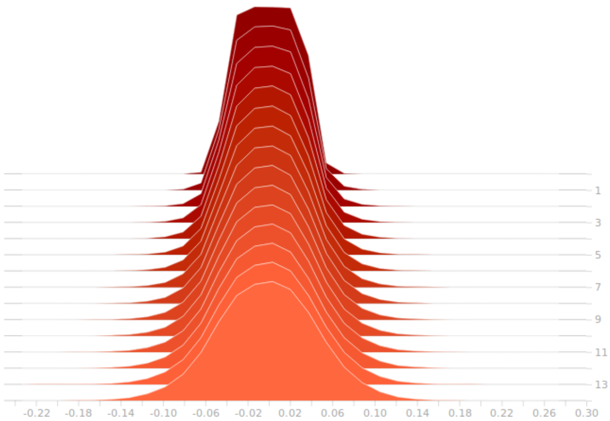
We can easily notice the difference in the shape of the distribution Figure 4.15 on the initial epoch and the final. The model with 512 filters has a normal distribution with lower variance comparing to the one with 128 filters. It is noticeable, that this the final distribution changed comparing to initial one, that means that layer learned some information.



a



b



c

Figure 4.15 — Convolutional model (a) 128;(b) 256; (c) 512 filters for each sizes [3, 4, 5]. Histogram of dense layers

The CNN model requires fewer resources for training than bi-LSTM. Therefore, I was able to use 5 times bigger batch size. We can see consumption of resources in the Figure 4.16

Figure 4.16 — CPU resources which were used while training CNN.

4.4 Convolution neural network with different regularization

In this section, I tried to use different regularization to get rid of overfitting. I tested all these approaches on the CNN with 512 filters. I made the following changes in the initial architecture of the model:

Modifications:

1. In the previous section the convolution layer did not train enough, so I thought it was caused because of big dropout rate. Therefore I decreased it in two times.
2. I also added the l2-regularization equals to 0.01 both for convolution layers and dense layer. Dropout equals to the rate 0.5 both for dense and convolution layers. Moreover, I decided to configure my training algorithm, so I used Adam with learning rate 1e-4.
3. l2-regularization equals to 0.001 for convolution layers and 0.01 for dense layer. Adam was with learning rate 1e-3. Dropout rate equals to 0.25 both for dense and convolution layers.
4. l2-regularization equals to 0.001 for convolution layers and 0.01 for dense layers. Dropout rate equals to 0.25 for convolution layers and 0.5 for dense layers.

Table 4.4 — Analysis of categorical accuracy

Modification type	**Train**	**Test**
1	0.8321	0.7993
2	0.7101	0.7146
3	0.9026	0.7882
4	0.8514	0.7449

Table 4.5 — Analysis of categorical cross entropy

Modification type	**Train**	**Test**
1	0.7894	0.9405
2	1.504	1.7040
3	0.4928	1.4280
4	0.7151	1.6890

Table 4.6 — Analysis of top k accuracy

Modification type	**Train**	**Test**
1	0.9681	0.9348
2	0.8333	0.8363
3	0.9704	0.9179
4	0.9453	0.8921

According to the results Figures 4.17, 4.18, 4.19, 4.20, it can be clearly seen that model with modifications number 3 and 4 were overfitted for sure. Model number 1 has lower overfit rate. The most stable model was number 2, which demonstrated the same results on test and train dataset.

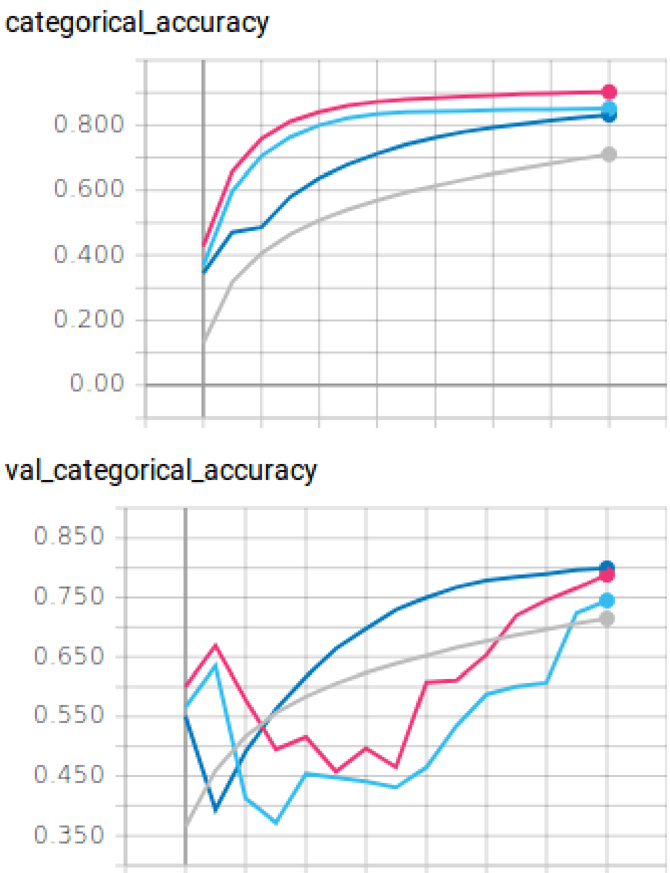


Figure 4.17 — Models train and validation categorical accuracy by epochs

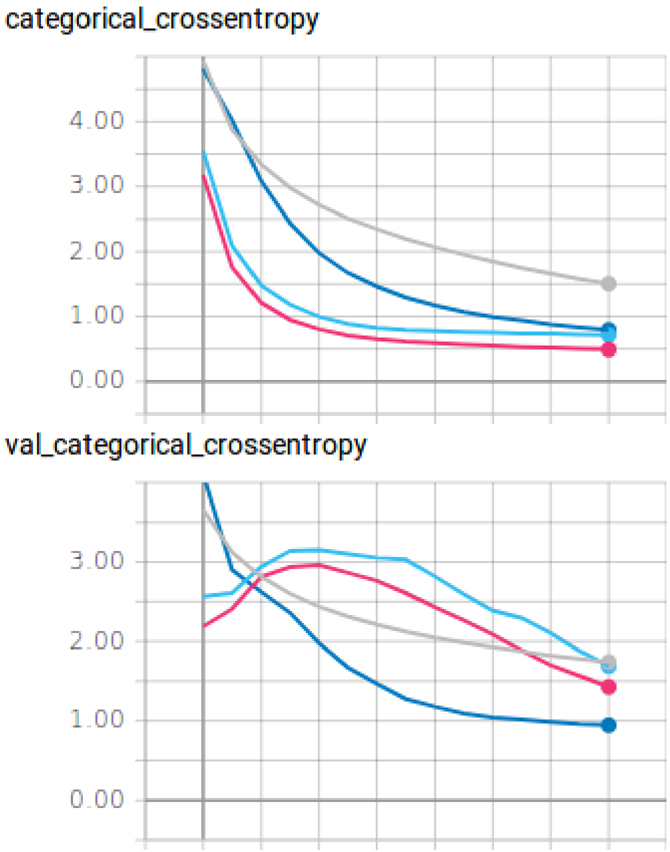


Figure 4.18 — Models train and validation category crossentropy by epochs

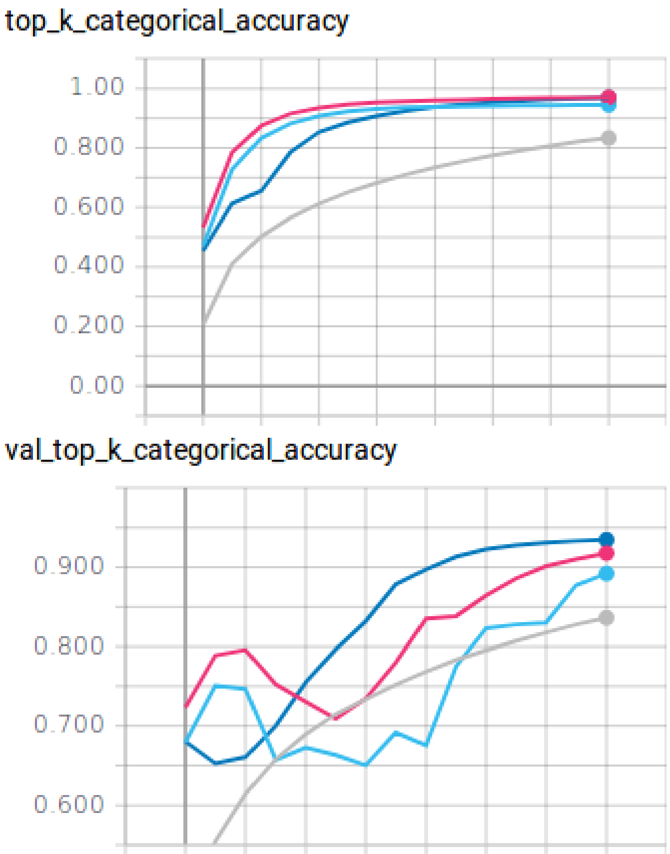


Figure 4.19 — Models train and validation top k accuracy by epochs

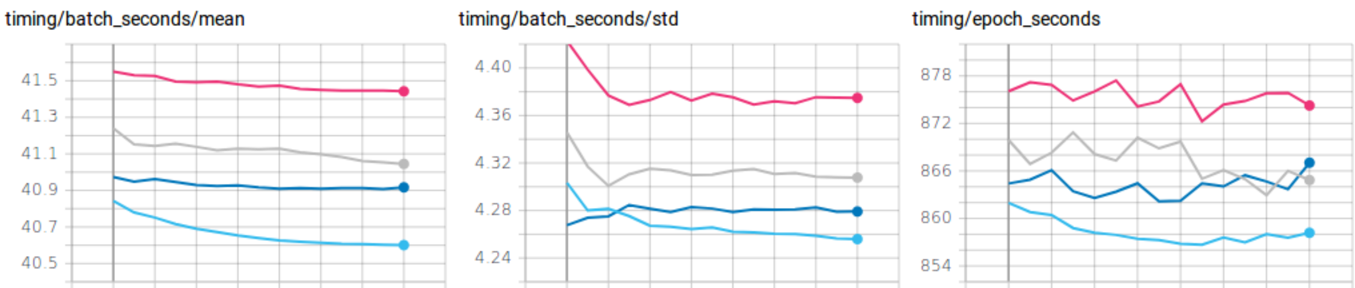
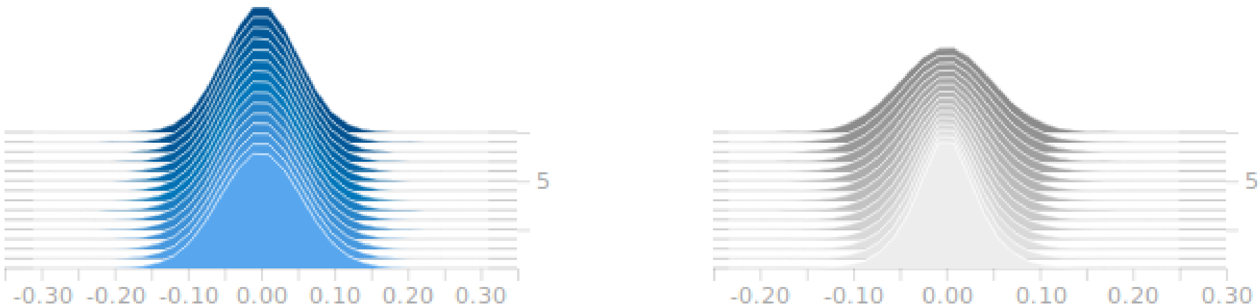


Figure 4.20 — Models batch time by epochs



a, b

c, d

Figure 4.21 — Convolutional models with modification (a) 1;(b) 2; (c) 3; (d) 4. Histogram of convolution layers

If we compare results of weights histograms Figures 4.21, 4.22 with the ones in Section 4.3 we can see a significant difference in distributions. Which is mostly caused by different types of regularizations. We can also see that convolution layers from model 1 and 3 change through epochs which means that they learn.

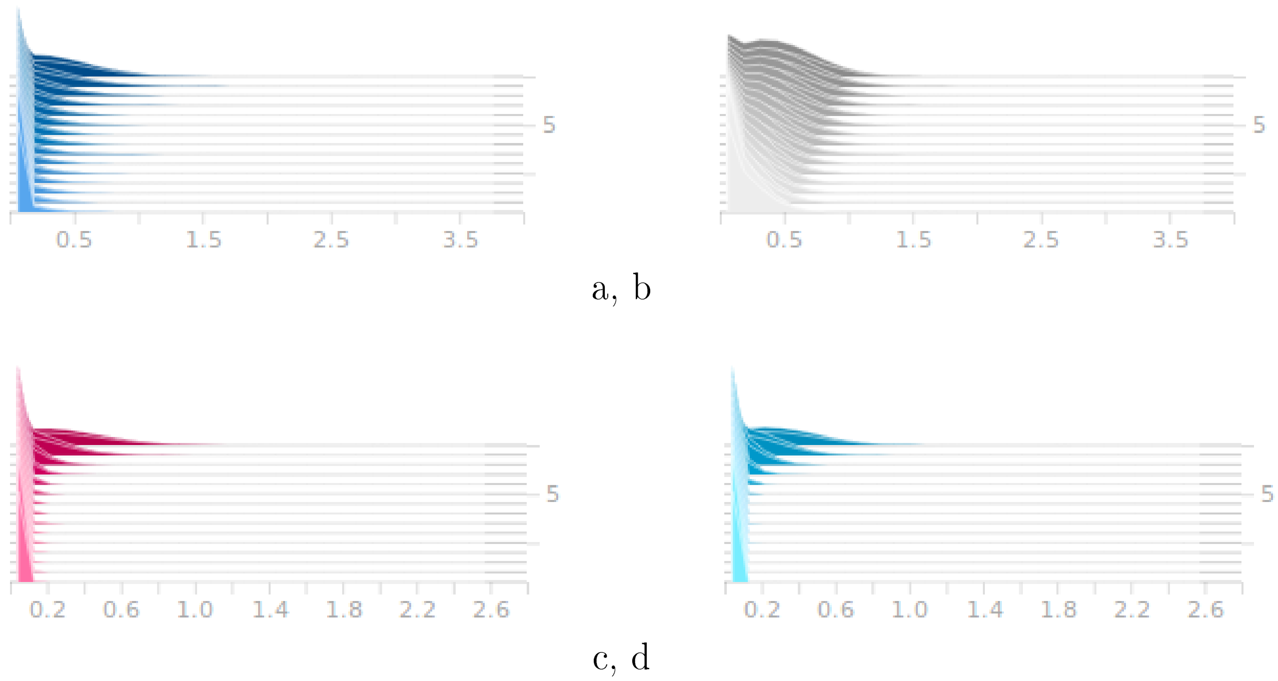


Figure 4.22 — Convolutional models with modification (a) 1;(b) 2; (c) 3; (d) 4. Histogram of merged layers

Similar situation we can see with dense layers Figure 4.23. The weights are put into ranges between -0.1 and 0.1 where l2-regularizations was used. The changes through epochs we can see only in first two models which are marked with blue and gray colors.

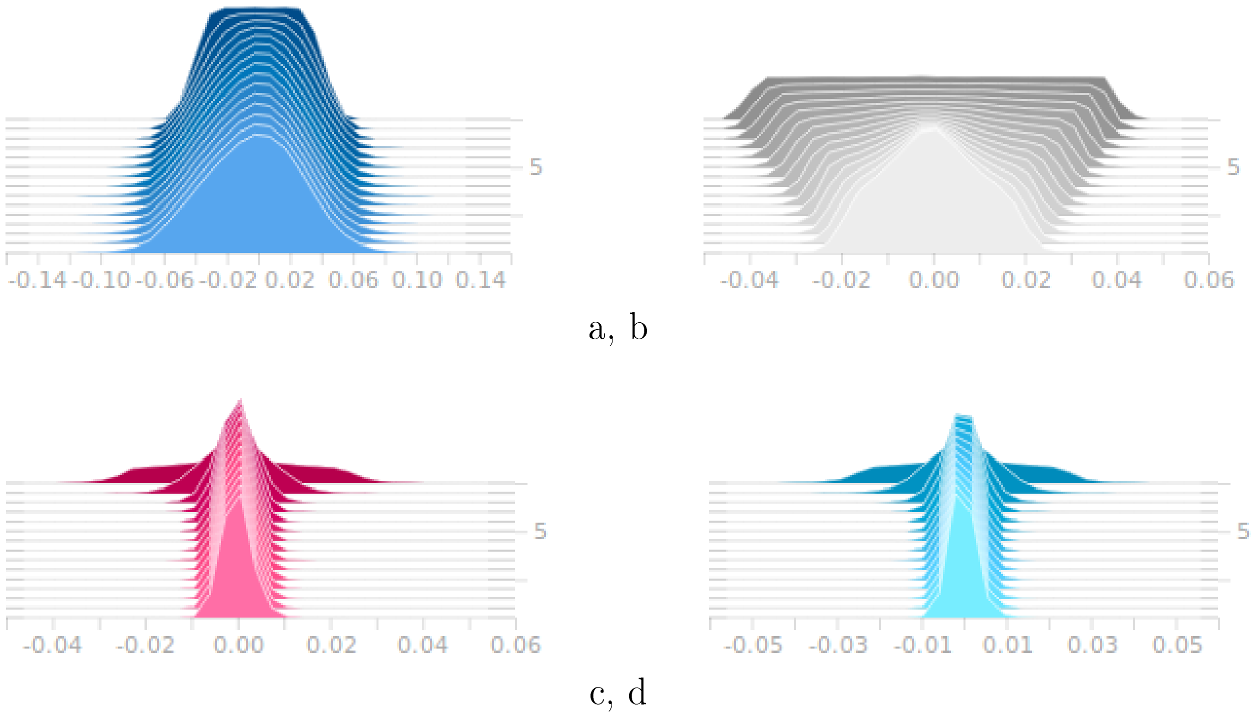


Figure 4.23 — Convolutional models with modification (a) 1;(b) 2; (c) 3; (d) 4. Histogram of dense layers

As we can see regularizations play the key role to avoid overfitting of the models. High dropout rate in combination with l2-regularization for dense and convolutions layers showed the best stability. Therefore, I chose model number 2 for the further training.

4.5 Final model

In this section I trained the best model from the Section 4.5 on 25 epochs. I has following parameters:

- 300 filters
- size of filter: 3, 4, 5
- l2-regularization equals to 0.01 both for convolution layers and dense layer.
- dropout equals to the rate 0.5 both for dense and convolution layers.
- learning algorithm Adam with learning rate 1e-4.

One more modification which was made: from the previous Section 4.5 according to many metrics model with these configurations trained slower than others because it has the lowest learning rate 1e-4. To speed up the training process I increased the learning rate in the beginning to be equal 1e-3. Then as the CNN slowly converges to its optimal value the learning was decreased to 1e-4 to slow down - otherwise it may overshoot the optimal value. Such modifications gave me improvement in speed and the stability of the model at the same time. The final results can be seen in Figures 4.24, 4.25, 4.26.

Table 4.7 — Final results

Metric	**Train**	**Test**
categorical accuracy	0.8250	0.8307
category crossentropy	0.5800	0.6612
top k accuracy	0.9545	0.9473

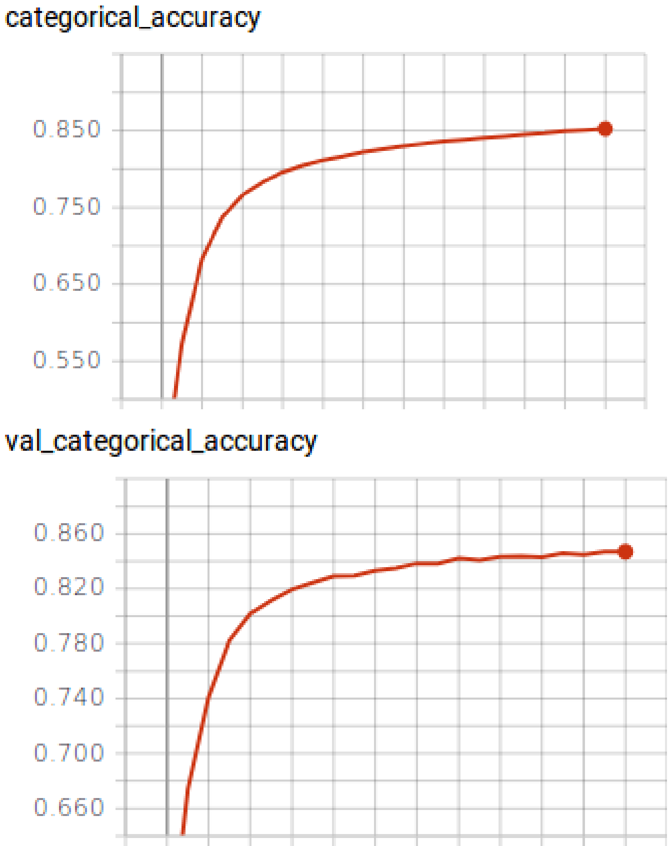


Figure 4.24 — Models train and validation categorical accuracy by epochs

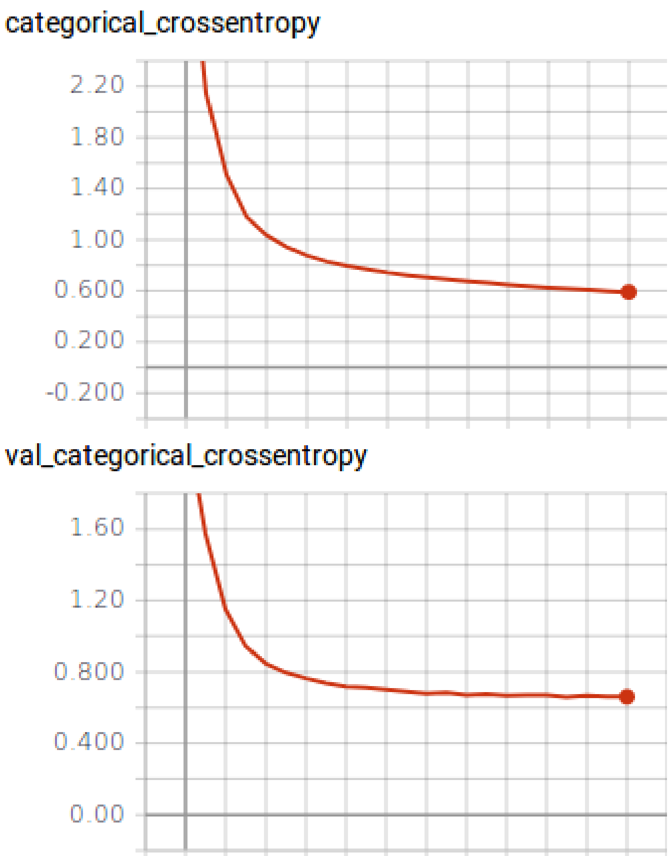


Figure 4.25 — Models train and validation category crossentropy by epochs

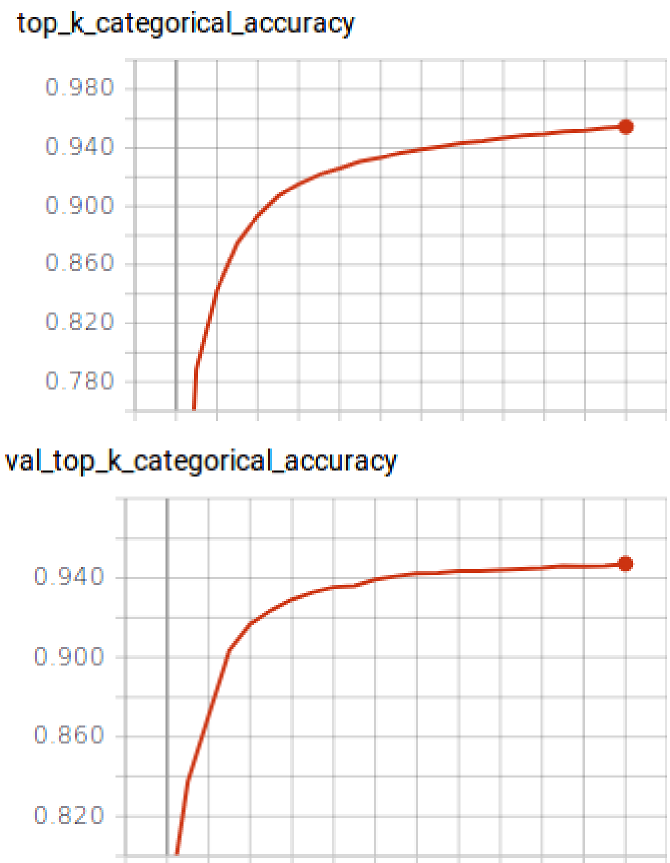


Figure 4.26 — Models train and validation top k accuracy by epochs

4.6 Summary of the section

In this section I have made a series of experiments with recurrent and convolutional neural networks built on top of word2vec. I did a little tuning of hyperparameters to achieve the highest scores. According to the results of experiments which CNN models achieved, are the same and in some components even better than the ones of Bi-LSTM models.I can make a conclusion that Convolution Neural Networks perform remarkably well for NLP related problems and specially for text classification.

CONCLUSION

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Jurafsky D. Speech and Language Processing / D. Jurafsky, M. James H. – New Jersey, 2008. – 1031 c. – (Pearson). – (ISBN: 0131873210). 2

2. Manning C. An Introduction to Information Retrieval / C. Manning, P. Raghavan, H. Schütze. – Cambridge, England: Online edition, 2009. – 544 p. – (Cambridge University Press). (ISBN: 0521865719). 3

3. Mohri M. Foundations of Machine Learning / M. Mohri, A. Rostamizadeh, A. Talwalkar. – Cambridge, Massachusetts, 2012. – 412 c. – (The MIT Press). – (ISBN: 9780262018258). 4

4. Géron A. Hands-On Machine Learning with Scikit- Learn and TensorFlow / Aurélien Géron. – Gravenstein Highway North, Sebastopol, CA, 2017. – 751 c. – (O'Reilly Media). – (ISBN: 8009989938). 5

5. Yang, Yiming, and Xin Liu. 1999. A re-examination of text categorization methods. In Proc. SIGIR, pp. 42–49. ACM Press. 2

6. McCallum, Andrew, and Kamal Nigam. 1998. A comparison of event models for Naive Bayes text classification. In AAAI/ICML Workshop on Learning for Text Categorization, pp. 41–48.

7. Rennie, Jason D., Lawrence Shih, Jaime Teevan, and David R. Karger. 2003. Tackling the poor assumptions of naive Bayes text classifiers. In Proc. ICML, pp. 616–623.

8. Tong, Simon, and Daphne Koller. 2001. Support vector machine active learning with applications to text classification. JMLR 2:45–66

9. https://arxiv.org/pdf/1301.3781.pdf

10. https://arxiv.org/pdf/1310.4546.pdf

11. Zavrel, Jakub, Peter Berck, and Willem Lavrijssen. 2000. Information extraction by text classification: Corpus mining for features. In Workshop Information Extraction Meets Corpus Linguistics. URL:

www.cnts.ua.ac.be/Publications/2000/ZBL00. Held in conjunction with LREC-2000.

12. Assignment 1[Course assignment]. (2016). Retrieved from http://cs224d.stanford.edu/assignment1/assignment1.pdf

13.

14.

15. Kalinin S. (2017, December, 7). Convolution neural networks with python.[Blog post]. Retrieved from https://habrahabr.ru/company/ods/blog/344008/

16. Raschka S. (2016, June, 11) Model evaluation, model selection, and algorithm selection in machine learning. [Blog post]. Retrieved from https://sebastianraschka.com/blog/2016/model-evaluation-selection-part1.html

17. P. Bojanowski*, E. Grave*, A. Joulin, T. Mikolov, Enriching Word Vectors with Subword Information

18. Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014), 1746–1751.

19. Kalchbrenner, N., Grefenstette, E., Blunsom, P. (2014). A Convolutional Neural Network for Modelling Sentences. Acl, 655–665.

20. http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

21. https://en.wikipedia.org/wiki/Recurrent_neural_network

22. http://colah.github.io/posts/2015-08-Understanding-LSTMs/

23. https://arxiv.org/pdf/1211.5063.pdf

LIST OF FIGURES

1.1 Classes, training set, and test set in text classification. 8
1.2 Supervised learning workflow. 9
1.3 Holdout method 11
1.4 Confusion matrix 12

2.1 Neural Network 17
2.2 The Skip-gram model architecture. 20
2.3 Words representation 22
2.4 The CBOW model architecture. 24
2.5 Convolution Neural Networks architecture for text classification . . . 25
2.6 Basic variables used in the convolution layer 25
2.7 ReLu activation function 27
2.8 Max pulling layer 27
2.9 Fully connected layer of CNN 28
2.10 Back propagation through max pulling layer 29
2.11 Back propagation through convolution layer 30
2.12 The structure of Recurrent neural network 31
2.13 The architecture of Recurrent neural network 32
2.14 The architecture of Long Short Term Memory neural network 32

3.1 Simplified event structure of data preprocessing 39
3.2 Build embeddings structure 41
3.3 4 42
3.4 Metrics which were logged 43

4.1 Architectures of Bi-LSTM models with 100 units 45
4.2 Models train and validation categorical accuracy by epochs 46
4.3 Models train and validation category crossentropy by epochs 47
4.4 Models train and validation top k accuracy by epochs 47
4.5 Models batch time by epochs 48
4.6 Bi-LSTM 100 units. Histogram of output from forward recurrent layers (a); histogram of weights from backward recurrent layers (b) . 48
4.7 Bi-LSTM 100 units. Histogram of weights from first FFNN layer. . . 49

4.8 CPU resources which were used while training Bi-LSTM NN. 49
4.9 Models train and validation categorical accuracy by epochs 51
4.10 Models train and validation category crossentropy by epochs 51
4.11 Models train and validation top k accuracy by epochs 52
4.12 Models batch time by epochs 52
4.13 Convolutional model (a) 128;(b) 256; (c) 512 filters for each sizes [3, 4, 5]. Histogram of convolution layers 53
4.14 Convolutional model (a) 128;(b) 256; (c) 512 filters for each sizes [3, 4, 5]. Histogram of merged layers 54
4.15 Convolutional model (a) 128;(b) 256; (c) 512 filters for each sizes [3, 4, 5]. Histogram of dense layers 55
4.16 CPU resources which were used while training CNN. 56
4.17 Models train and validation categorical accuracy by epochs 58
4.18 Models train and validation category crossentropy by epochs 59
4.19 Models train and validation top k accuracy by epochs 59
4.20 Models batch time by epochs 60
4.21 Convolutional models with modification (a) 1;(b) 2; (c) 3; (d) 4. Histogram of convolution layers 60
4.22 Convolutional models with modification (a) 1;(b) 2; (c) 3; (d) 4. Histogram of merged layers 61
4.23 Convolutional models with modification (a) 1;(b) 2; (c) 3; (d) 4. Histogram of dense layers 62
4.24 Models train and validation categorical accuracy by epochs 64
4.25 Models train and validation category crossentropy by epochs 64
4.26 Models train and validation top k accuracy by epochs 65
27 Architectures of CNN model 79

LIST OF TABLES

2.1 Feature vector . . . 15

3.1 The hierarchy of categories . . . 36
3.2 Structure of the data files . . . 37
3.3 Training set general information . . . 37
3.5 Information about second-level categories . . . 37
3.4 Information about first-level categories . . . 38
3.6 Information about categorical features . . . 38
3.7 Simplified event structure of data preprocessing . . . 40
3.8 Simplified event structure of data preprocessing . . . 41

4.1 Analysis of categorical accuracy . . . 50
4.2 Analysis of category crossentropy . . . 50
4.3 Analysis of top k accuracy . . . 50
4.4 Analysis of categorical accuracy . . . 57
4.5 Analysis of categorical cross entropy . . . 57
4.6 Analysis of top k accuracy . . . 58
4.7 Final results . . . 63
8 Classification report . . . 73
8 Classification report . . . 74
8 Classification report . . . 75
8 Classification report . . . 76
8 Classification report . . . 77
8 Classification report . . . 78

APPENDIX A

Table 8 — Classification report

category	precision	recall	f1-score	support
1	0	0	0	7
3	0	0	0	4
4	0	0	0	2
5	0	0	0	1
6	0	0	0	6
7	0	0	0	1
8	0	0	0	2
9	0	0	0	2
11	0.78	0.58	0.66	623
12	0.63	0.42	0.5	281
14	0.93	0.99	0.96	8070
15	0.82	0.68	0.74	362
16	0.93	0.95	0.94	1656
17	0.81	0.88	0.84	550
18	0.73	0.73	0.73	314
19	0.82	0.9	0.86	263
20	0.82	0.91	0.86	1151
21	0.91	0.41	0.56	76
22	0.82	0.85	0.83	830
23	0.44	0.56	0.49	482
24	0	0	0	4
25	0.75	0.9	0.82	1910
26	0.76	0.84	0.8	310
27	0	0	0	5
29	0.97	0.99	0.98	12346
30	0.86	0.61	0.72	270
31	0.85	0.52	0.64	224
33	0.9	0.95	0.92	434

Table 8 — Classification report

category	precision	recall	f1-score	support
34	0.5	0.04	0.07	26
35	0.09	0.04	0.06	24
36	0.16	0.15	0.16	67
37	0.89	0.66	0.76	197
38	0.68	0.21	0.33	183
40	0.81	0.68	0.74	678
42	0.89	0.9	0.89	1298
43	0.71	0.81	0.76	782
44	0.84	0.95	0.89	1184
45	1	0.03	0.06	30
46	0.58	0.31	0.41	181
47	0	0	0	1
51	0.72	0.85	0.78	591
53	0.59	0.72	0.65	793
55	0.84	0.95	0.89	2471
56	0.63	0.59	0.61	537
57	0.86	0.72	0.79	163
59	0	0	0	2
60	0.67	0.59	0.63	231
61	0.7	0.5	0.58	113
62	0.63	0.58	0.6	67
64	1	0.79	0.88	14
65	0.76	0.44	0.55	117
66	1	0.21	0.35	28
67	0.52	0.37	0.43	122
70	0	0	0	4
71	0	0	0	22
72	0.75	0.19	0.31	31
73	0	0	0	1
74	0.59	0.73	0.65	146
75	0	0	0	3

Table 8 — Classification report

category	precision	recall	f1-score	support
76	0	0	0	18
78	0.57	0.75	0.65	83
79	0.38	0.4	0.39	92
80	0.2	0.1	0.13	21
81	0	0	0	6
82	0.22	0.64	0.32	85
83	0.49	0.53	0.51	78
84	0.24	0.25	0.24	16
85	0.68	0.61	0.64	87
86	0.57	0.53	0.55	32
87	0.2	0.14	0.17	7
88	0	0	0	1
89	0.75	0.33	0.46	27
90	0.53	0.47	0.5	49
91	0.63	0.39	0.48	83
92	0.58	0.28	0.38	25
93	0.42	0.29	0.34	52
94	0	0	0	12
95	0.5	0.18	0.27	11
96	0.65	0.48	0.55	23
97	0	0	0	4
98	0.2	0.12	0.15	17
99	0	0	0	5
100	0.43	0.11	0.18	107
101	0.36	0.21	0.27	47
102	0.22	0.11	0.15	18
103	0	0	0	7
104	0.56	0.6	0.58	47
105	0.46	0.6	0.52	10
106	0	0	0	3
107	0.8	0.6	0.69	68

Table 8 — Classification report

category	precision	recall	f1-score	support
108	0.71	0.14	0.23	37
109	0.09	0.04	0.06	73
110	0	0	0	1
111	0.79	0.75	0.77	88
112	0.68	0.38	0.49	50
113	0.76	0.61	0.68	83
114	0.78	0.64	0.7	11
115	0.39	0.64	0.49	194
116	0.58	0.52	0.55	87
117	0.41	0.33	0.37	21
118	0.81	0.73	0.77	237
119	0.63	0.63	0.63	70
120	0.67	0.56	0.61	18
121	0	0	0	4
122	0.6	0.54	0.57	28
123	0.55	0.89	0.68	63
124	0.74	0.84	0.78	87
125	0.65	0.57	0.61	56
126	0.63	0.7	0.67	47
127	0.67	0.57	0.62	7
128	0.62	0.36	0.46	22
129	0.8	0.5	0.62	8
130	0	0	0	4
131	0.64	0.46	0.53	167
132	0.72	0.5	0.59	62
133	1	0.2	0.33	5
134	0.66	0.72	0.69	87
135	0.64	0.78	0.7	18
136	1	0.25	0.4	8
137	0.8	0.85	0.82	142
138	0.76	0.62	0.69	56

Table 8 — Classification report

category	precision	recall	f1-score	support
139	0.31	0.31	0.31	83
140	0	0	0	1
141	0.33	0.09	0.14	81
142	0.37	0.34	0.35	263
143	0	0	0	5
144	0.62	0.15	0.24	66
145	0.78	0.83	0.8	142
146	0.47	0.2	0.28	35
147	0.53	0.7	0.6	221
148	0.56	0.25	0.34	109
149	0.29	0.18	0.22	11
150	0	0	0	12
151	0	0	0	34
152	0	0	0	89
153	0	0	0	4
154	0.81	0.29	0.43	58
155	0	0	0	35
156	0.57	0.18	0.27	68
157	0.21	0.1	0.14	30
158	0.76	0.88	0.82	375
159	0.57	0.09	0.16	43
160	0.43	0.59	0.49	188
162	0.62	0.56	0.58	263
165	0.71	0.67	0.69	445
166	0.53	0.37	0.43	49
167	0	0	0	13
168	0.92	0.93	0.93	423
169	0.83	0.91	0.87	570
172	0.71	0.67	0.69	625
249	0.57	0.46	0.51	177
250	0.28	0.08	0.13	109

Table 8 — Classification report

category	precision	recall	f1-score	support
251	0.78	0.74	0.76	316
252	0.56	0.62	0.59	332
253	0.65	0.67	0.66	313
254	0.71	0.18	0.28	299
255	0.87	0.79	0.83	232
256	0.76	0.57	0.65	312
257	0.88	0.91	0.89	847
258	0.57	0.72	0.64	495
259	0.89	0.75	0.81	63
265	0.69	0.92	0.79	101
266	0.62	0.78	0.69	233
267	0.62	0.42	0.5	31
268	0.64	0.77	0.7	258
269	0	0	0	17
270	0.74	0.55	0.63	158
272	0.8	0.79	0.79	228
273	0.62	0.21	0.31	78
274	0.63	0.33	0.44	108
275	0.93	0.42	0.58	31
278	0.78	0.5	0.61	129
279	0.79	0.86	0.82	842
280	0.57	0.47	0.51	344
281	0.62	0.6	0.61	419
283	0	0	0	23
284	0.83	0.4	0.54	248
285	0.67	0.03	0.06	66
287	0.56	0.58	0.57	429
288	0	0	0	66
289	0	0	0	80
avg	0.81	0.82	0.81	55000

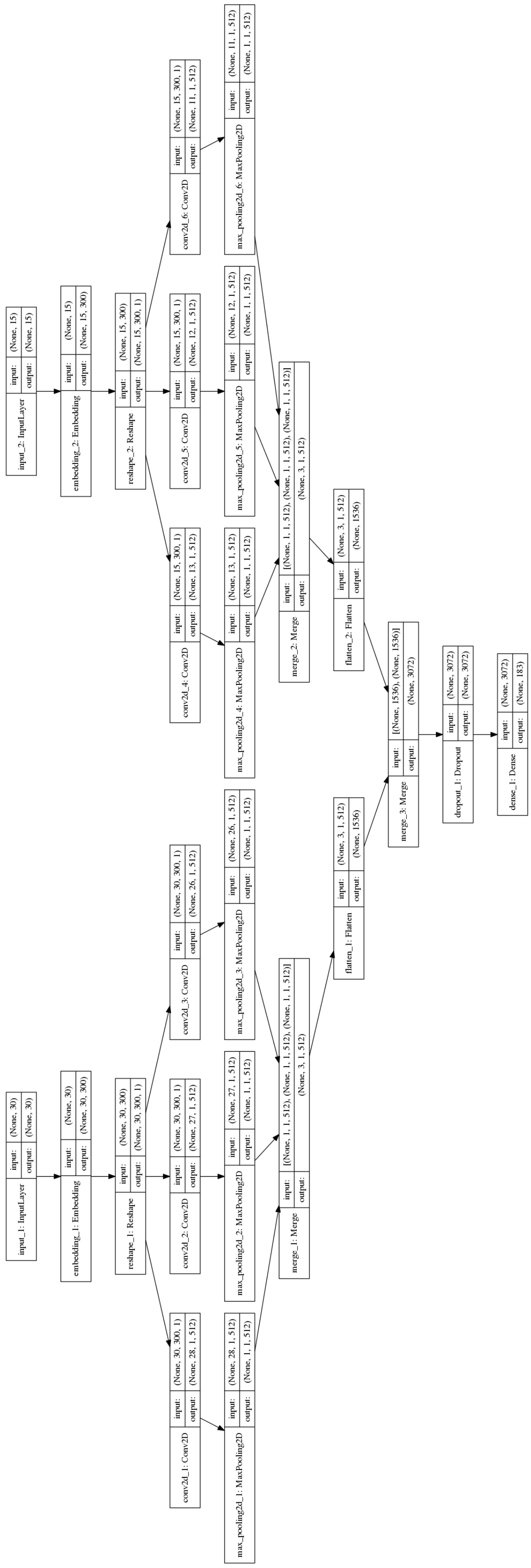


Figure 27 — Architectures of CNN model