НАЗВАНИЕ УЧРЕЖДЕНИЯ, В КОТОРОМ ВЫПОЛНЯЛАСЬ ДАННАЯ ДИССЕРТАЦИОННАЯ РАБОТА

На правах рукописи

УДК xxx.xxx

Фамилия Имя Отчество автора

НАЗВАНИЕ ДИССЕРТАЦИОННОЙ РАБОТЫ

Специальность XX.XX.XX —
«Название специальности»

Диссертация на соискание учёной степени
кандидата физико-математических наук

Научный руководитель:
уч. степень, уч. звание
Фамилия Имя Отчество

Город — 20XX

Contents

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

The work consists of five sections.

In the first section we consider the problem of text classification, an overview of the basic concepts, models and criteria used in solving such problems.

In the second section, we present the process of transforming text into features, formalizing classification models, and criteria for evaluating the obtained models.

The third section examines the software used, as well as a code with explanations and diagrams that reproduce the process of text into features, then putting features into neural nets and selecting the best model. There are also limitations to their use, advantages, and comparison of the two methods.

The fourth section contains the results of work and the accompanying description.

The fifth section provides a financial and economic analysis of the software product.

1. Text classification

1.1 Relevance of the problem

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

1.2 Statement of classification problem

Classification problem - the problem of identifying to which category a new observation belongs. The basic example can be situation when you receive a new email and 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 human defined 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:

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

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 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 on 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 an application. When we use the training set to learn a classifier for test data, we make the assumption that training data and test data are similar or from the same distribution. [2, p.256-257]

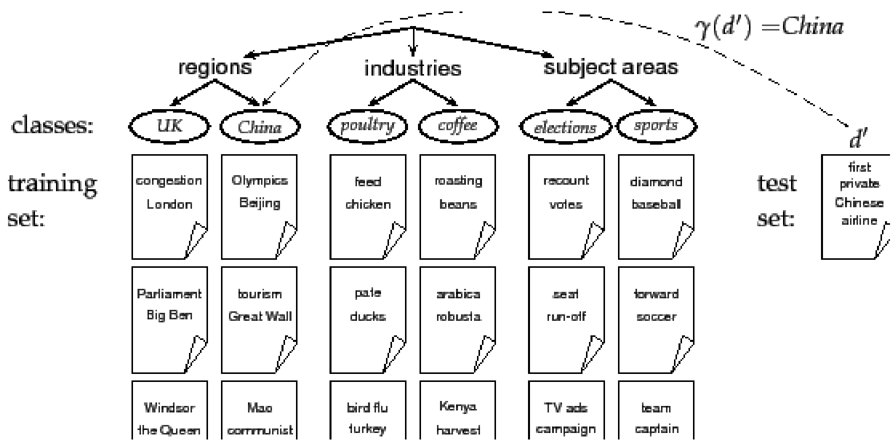


Figure 1.1 — Classes, training set, and test set in text classification.

1.3 Short review of existing mathematical models, which can be used to solve the classification problem

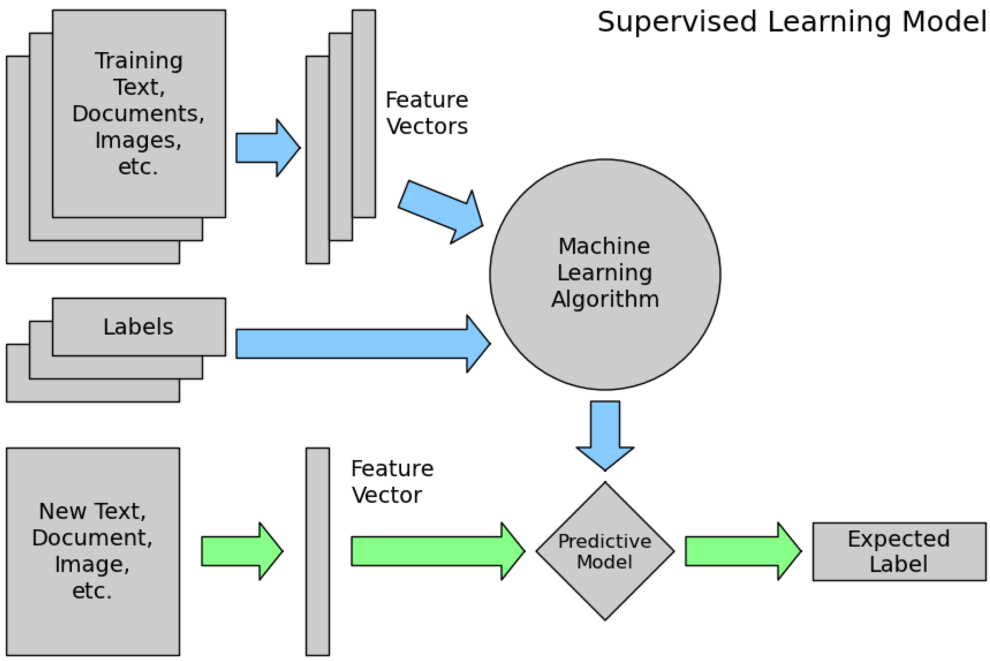


Figure 1.2 — Supervised learning work flow.

Supervised learning - the machine learning task of inferring a function from labeled 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 this 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 in 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]
4. Decision Trees and Random Forests . [2]
5. Neural networks . [2]

1.4 Model evaluation and validation

Machine learning pipeline does not finish 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 valuation 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. Overfit 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.

1.4.1 Model Evaluation Applications

Generalization performance - 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.

Model selection - we want 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 do 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 best one under the given circumstances. Therefore the natural need is to compare different algorithms to each other, often regarding predictive and computational performance. Nevertheless, these three sub-tasks have similarities in terms that we want to estimate the performance of a model, they all require different approaches.

Although these three sub-tasks have all in common that we want to estimate the performance of a model, they all require different approaches.

1.4.2 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.

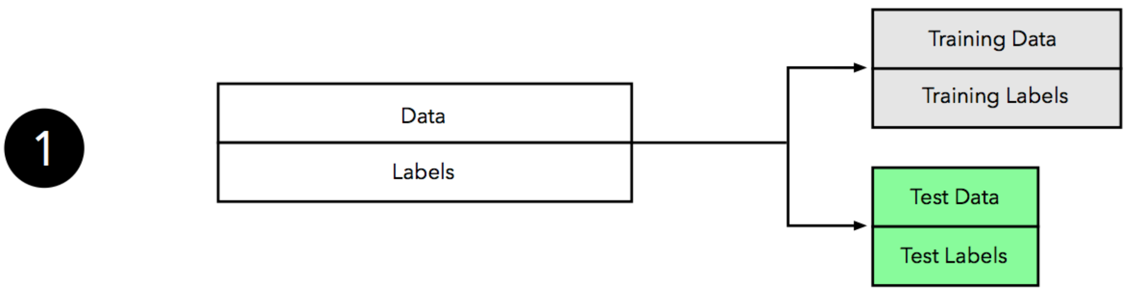


Figure 1.3 — .

Then, we fit a model to the training data and predict the labels of the test set.

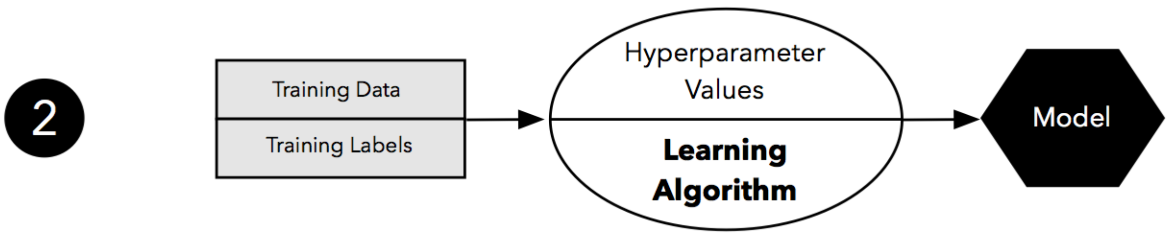


Figure 1.4 — .

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. [14]

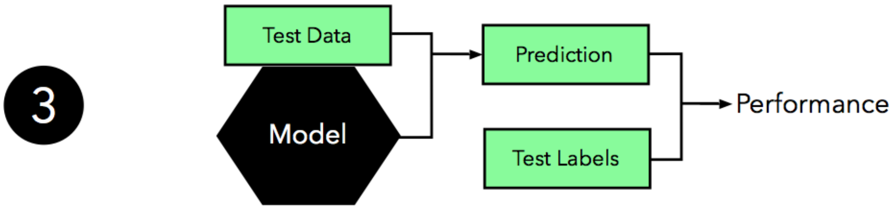


Figure 1.5 — .

1.5 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 for classification problem are:

Accuracy

Accuracy simply measures what percent of your predictions were 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.

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.

Confusion matrix allows you to compute various classification metrics.

Precision and Recall

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

Prediction outcome

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

Figure 1.6 — Confusion matrix

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

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

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

F1-score

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.

1.6 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.

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 ROC-AUC curve were selected.

2. Mathematical models and algorithms for text classification

2.1 Words representations

In supervised learning domain, to perform classification tasks, usually our goal is 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 specific numerical format, in another words it is necessary to form feature vectors. In Natural Language Processing automated feature extraction may be achieved in many ways.[тут вставить список литературы]

2.1.1 Bag-of-Words Approach

Bag-of-words - 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 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 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, demonstrate 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 to distinguish between sentences, built from the same words while having opposite meanings:"This paintings don't feel like ugly - buy them!"(positive) and "This 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.

1. **Word n-grams - catches 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 applies a weight to each overlapping word. The TF weight measures how many times the word occurs in 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 IDF content words is high. [1, p.647] Formaly it can be defined:

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

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: A word's meaning is given by the words that frequently appear close-by.

тут вставить информацию про работі проведенные в данной области. https://arxiv.org/pdf/1301.3781.pdf https://arxiv.org/pdf/1310.4546.pdf

1. **Skip-gram model**

To begin with 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:

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

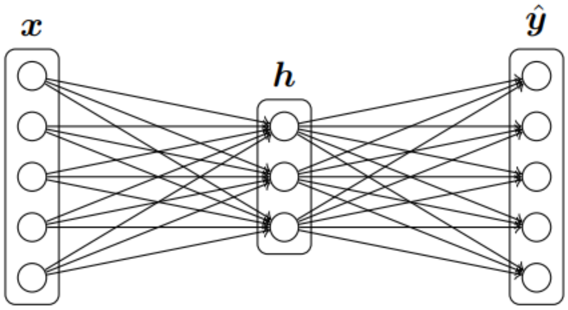


Figure 2.1 — Neural Network

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 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 the 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 [11]:

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,...,|W|) are the ‘output’ word vectors for all words in the vocabulary. Cross entropy cost is applied to this

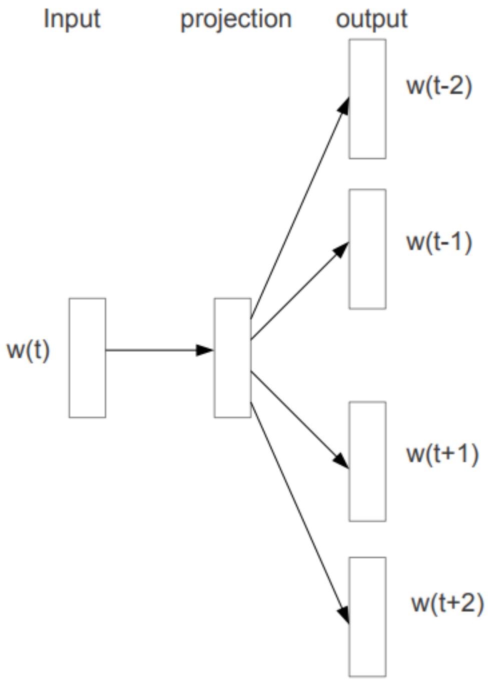


Figure 2.2 — The Skip-gram model architecture.

prediction and word o is the expected word (the o-th element of the one-hot label vector is one). $\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_{|W|}]$ 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,

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

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

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

We have calculated gradient for one particular word, now we will generalize this to a number of words. We have a set of context words [$\texttt{word}_{c-\mathbf{m}}$,...,$\texttt{word}_{c-\mathbf{1}}$,$\texttt{word}_c$,$\texttt{word}_{c+\mathbf{1}}$,...,$\texttt{word}_{c+\mathbf{m}}$], 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 $J(\boldsymbol{o}, \boldsymbol{v}_c, ...)$ cost functions.
Then we can rewrite cost function as follows:

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

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 [10],

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

and

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

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:

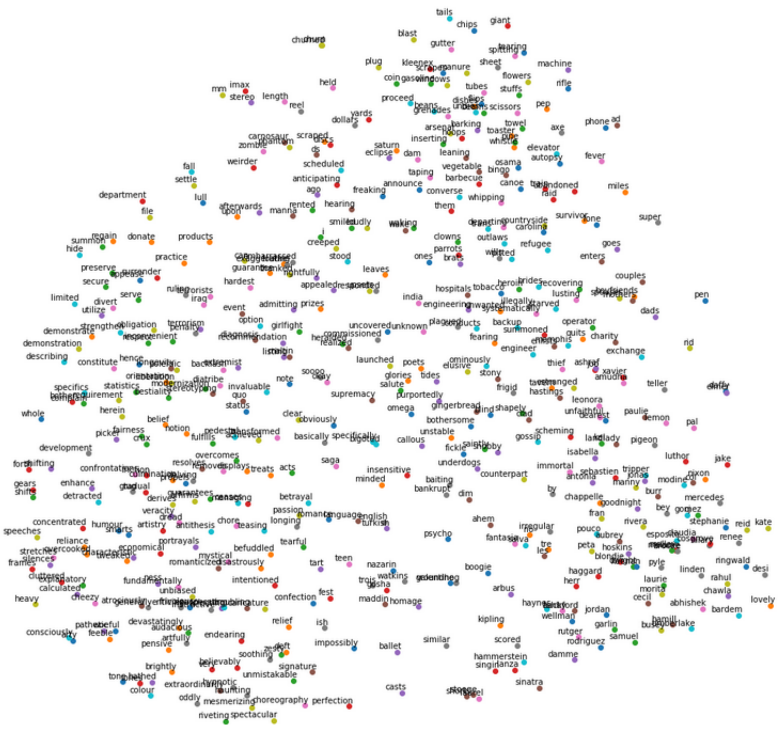


Figure 2.3 — Words representation

However, this type of architecture, where for each output we need to compute separate *softmax* function are 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

Negative Sampling technique

The only difference from 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 calculation not on whole vocabulary V, but only on part of it, which randomly generated each time.

Hierarchical Softmax
H-Softmax is an approximation which uses 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 thees have a maximum depth of $log_2(|V|)$, that means that in the worst case we need to calculate $log_2(|V|)$ nodes to find the necessary probability of certain word.

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

2. **CBOW model** This model is very similar to skip-gram, but CBOW predicts target word from the bag of words context. From the practical point of view: skip-gram works well with small amount of the training data and represents well even rare words or phrases. CBOW - several times faster to train than the skip-

gram, slightly better accuracy for the frequent words. This model presented in Figure 2.4 [11]:

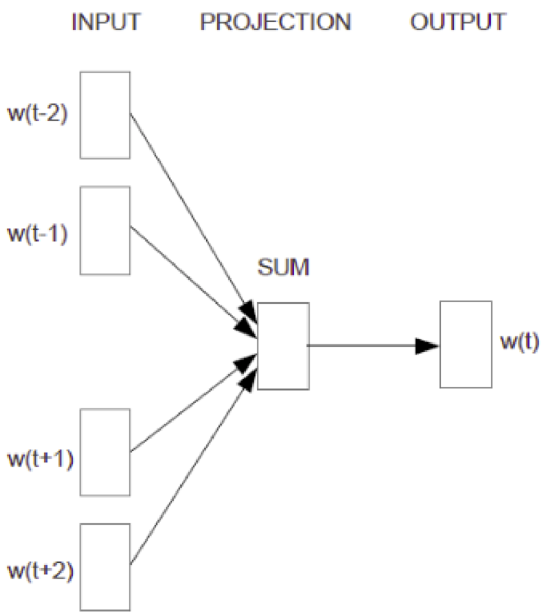


Figure 2.4 — The CBOW model architecture.

2.2 Deep learning algorithms for text classification

When people hear about NLP problems and neural networks in the one context they probably think about Recurrent neural networks or their modification. However, recently some papers which apply CNNs to problems in Natural Language Processing were introduced and they got some interesting results [16] [17]. 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 [12], 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 $x_{i:i+j}$ refer to the concatenation of words $\{x_i, x_{i+1}, ..., x_{i+j}\}$. [12]

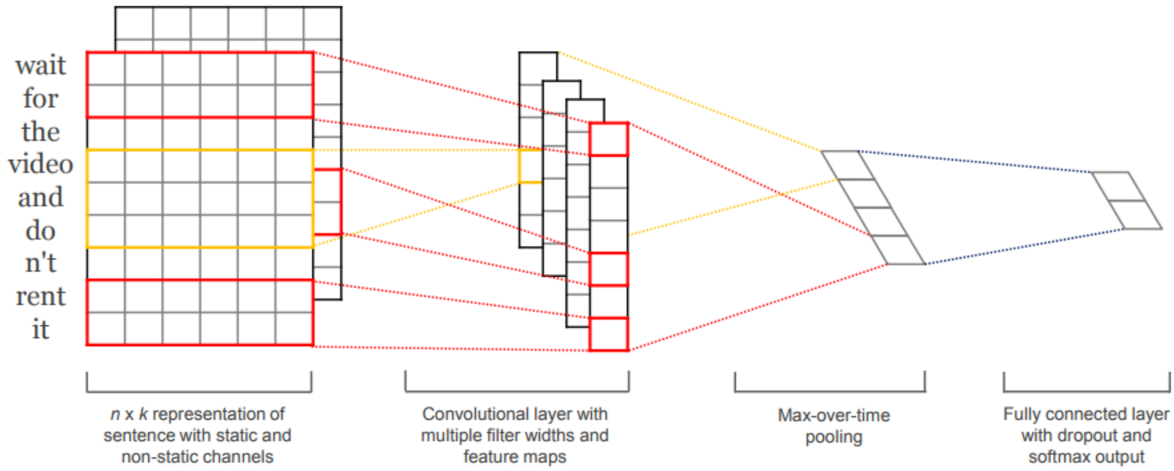


Figure 2.5 — Convolution Neural Networks architecture for text classification

Convolution

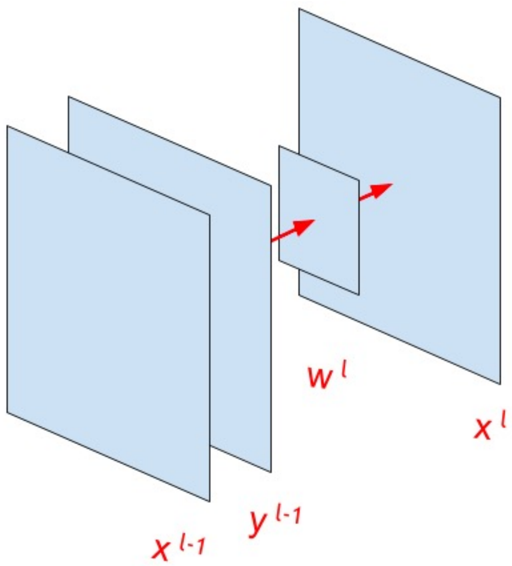


Figure 2.6 — Basic variables which are used in the convolution layer

In a 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 illustrated in the Figure 2.6 [13].

The following equation 2.35 describes 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 (N, M).

The important moment which I should put attention 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 transformation which has such general view $y^l = f(x^l)$. I do not cover all activations functions which exist, I chose only these which were used in current model.

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

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

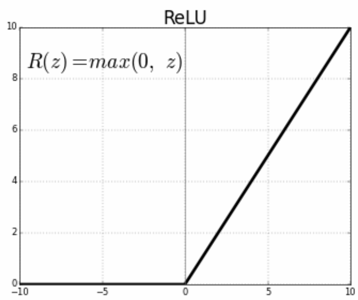


Figure 2.7 — ReLu activation function

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

Max pulling layer

This layer allows you 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: 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 we can use for classification.

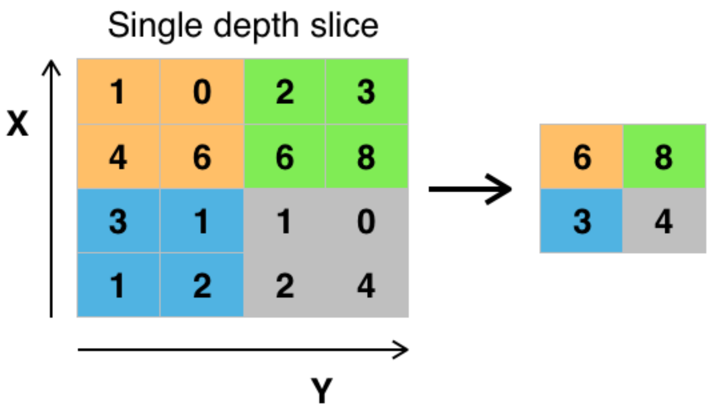


Figure 2.8 — Max pulling layer

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 2.1 describes this stage.

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

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.

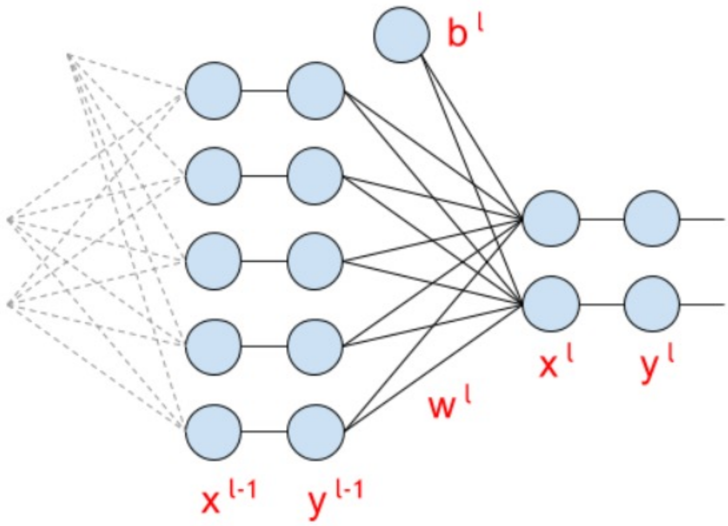


Figure 2.9 — Fully connected layer of CNN

Now after all components of CNN are known, we need to optimize weights for each layer. Therefore, it is necessary to derive of 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 following equation 2.39:

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

or

$$\begin{gathered}\left[\begin{array}{cccc}\frac{\partial J}{\partial x_0^l} & \frac{\partial J}{\partial x_1^l} & \ldots & \frac{\partial J}{\partial x_n^l}\end{array}\right] = \\ = \left[\begin{array}{cccc}(\frac{\partial J}{\partial y_0^l}\frac{\partial y_0^l}{\partial x_0^l} + \frac{\partial J}{\partial y_1^l}\frac{\partial y_1^l}{\partial x_0^l} + \ldots + \frac{\partial J}{\partial y_n^l}\frac{\partial y_n^l}{\partial x_0^l}) & (\frac{\partial J}{\partial y_0^l}\frac{\partial y_0^l}{\partial x_1^l} + \frac{\partial J}{\partial y_1^l}\frac{\partial y_1^l}{\partial x_1^l} + \ldots + \frac{\partial J}{\partial y_n^l}\frac{\partial y_n^l}{\partial x_1^l}) & \ldots & (\frac{\partial J}{\partial y_0^l}\frac{\partial y_0^l}{\partial x_n^l} + \frac{\partial J}{\partial y_1^l}\frac{\partial y_1^l}{\partial x_n^l} + \ldots + \frac{\partial J}{\partial y_n^l}\frac{\partial y_n^l}{\partial x_n^l})\end{array}\right] \\ = \left[\begin{array}{cccc}\frac{\partial J}{\partial y_0^l} & \frac{\partial J}{\partial y_1^l} & \ldots & \frac{\partial J}{\partial y_n^l}\end{array}\right] \left[\begin{array}{cccc}\frac{\partial y_0^l}{\partial x_0^l} & \frac{\partial y_0^l}{\partial x_1^l} & \ldots & \frac{\partial y_0^l}{\partial x_n^l} \\ \frac{\partial y_1^l}{\partial x_0^l} & \frac{\partial y_1^l}{\partial x_1^l} & \ldots & \frac{\partial y_1^l}{\partial x_n^l} \\ \ldots & \ldots & \ldots & \ldots \\ \frac{\partial y_n^l}{\partial x_0^l} & \frac{\partial y_n^l}{\partial x_1^l} & \ldots & \frac{\partial y_n^l}{\partial x_n^l}\end{array}\right]\end{gathered} \tag{2.40}$$

Next, we should update weight 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

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

Jquation 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. 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.

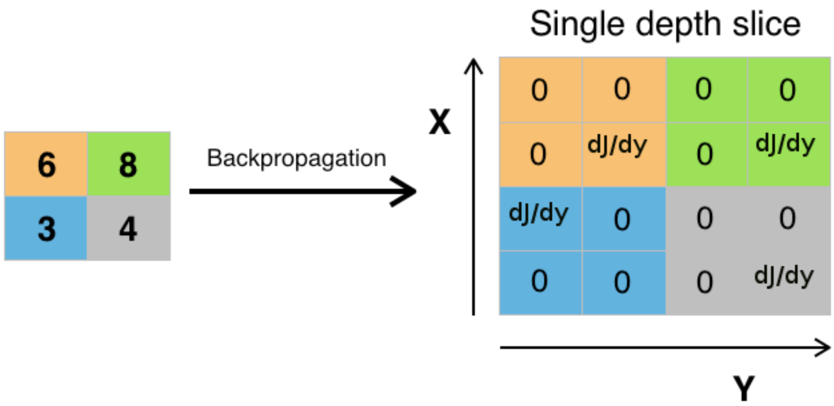


Figure 2.10 — Back propagation through max pulling layer

It is necessary to derive weights update for kernel 2.11.

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

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

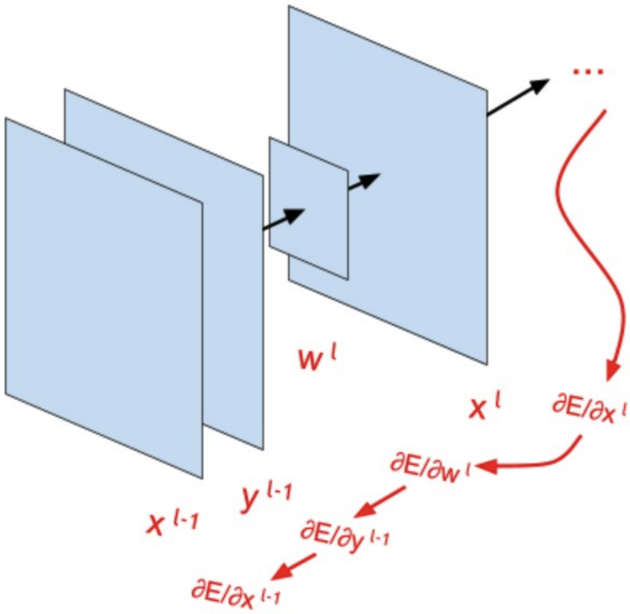


Figure 2.11 — Back propagation through convolution layer

Derivation of gradient for the bias element.

$$\frac{\partial J}{\partial b^l} = \sum_i \sum_j \frac{\partial J}{\partial y_{ij}^l} \frac{\partial y_{ij}^l}{\partial x_{ij}^l} \frac{\partial x_{ij}^l}{\partial b^l} = \sum_i \sum_j \frac{\partial J}{\partial y_{ij}^l} \frac{\partial y_{ij}^l}{\partial x_{ij}^l} \tag{2.45}$$

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

$$\frac{\partial J}{\partial y_{ij}^{l-1}} = \sum_{i'} \sum_{j'} \frac{\partial J}{\partial y_{i'j'}^l} \frac{\partial y_{i'j'}^l}{\partial x_{i'j'}^l} \cdot w_{(i-i's)(j-j's)}^l \tag{2.46}$$

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 tasks such as natural language processing. [19] Recurrent neural networks are networks with loops in them, allowing information to persist. Figure 2.12 [20] illustrates RNN where, A looks at some input x_t and outputs a value h_t. A loop allows information to be passed from one step of the network to the next.

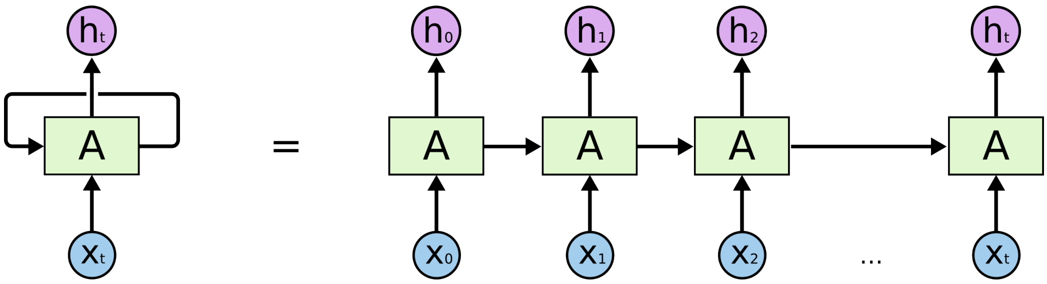


Figure 2.12 — The structure of Recurrent neural network

However, this type of NN has problems called "Vanishing Gradient and Gradient Explosion Problems". This problem was detailed explored in [21]. 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 [20] and Figure 2.14 [20].

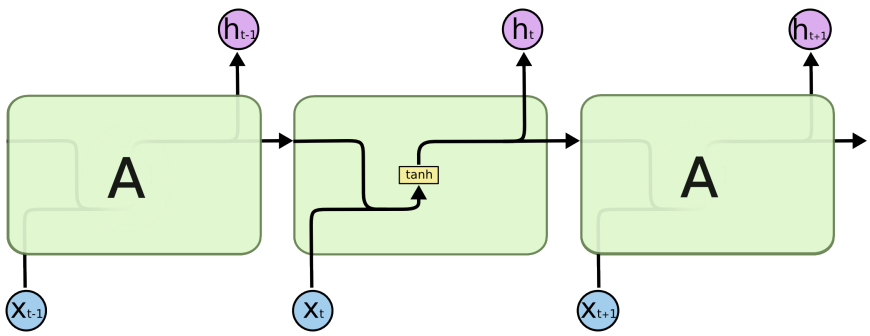


Figure 2.13 — The architecture of Recurrent neural network

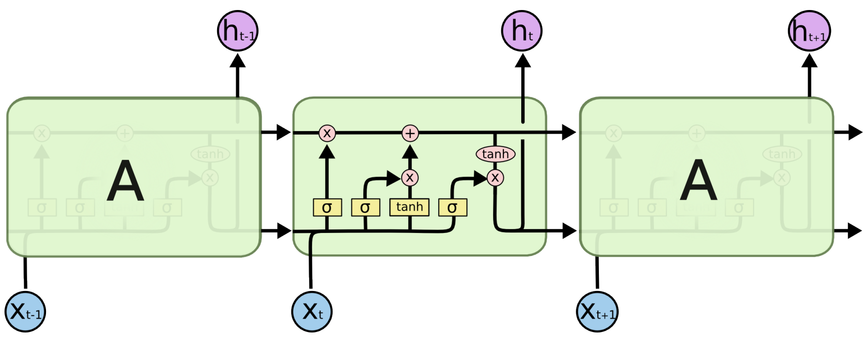


Figure 2.14 — The architecture of Long Short Term Memory neural network

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. It 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}.

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

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.

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

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

It's now time 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 the 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, we need to decide what we're going 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 [20] .

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

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

2.2.3 Advantages and drawbacks of different architectures

Recurrent Neural Networks have intuitive sense in NLP taks. They resemble how we process language : reading sequentially from left to right. In contrary, CNNs which widely 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 has important where in the sentence a word appears. 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 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 fast. Very fast. Convolutions are a central part of computer graphics and 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 needing to represent the whole vocabulary. It's completely reasonable to have filters of size larger than 5. [18] In the next chapter, I will implement both architectures and evaluate them.

2.2.4 Summary of the section

The second section provides different methods for textual information encoding such as Bag-of-words and embeddings. Deep analysis of architectures neural networks which are useful for text classification problem. In details gradients of each neural networks were calculated.

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 were made in the 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, 15×2 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 [15]. These vectors in dimension 300 were obtained using the skip-gram model with default parameters.

My main requirement for the framework for building deep neural network models were:

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

Among a wide range of frameworks which are 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 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 the Table 3.2. First, let us get familiar with data and how it distributes between categories. As it can be clearly seen from the Table 3.3 we have 2 categorical and 2 numerical variables and our data does not have any missing fields. That is good to start examine 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 a 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 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.

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.

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

3.3 Data preparation

1.Tokenize Text

The given text was split by spaces and then lemmatized.

2. Remove infrequent words

Words which appears 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 numbers which can be represented as price, year, mobile number. The telephone numbers occur really frequently in

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%

advertisements, but in most cases, they are unique because different people have their own telephone numbers - the information about whether a telephone number presents in 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.

4. Build training data matrices

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 3.6

Information about categorical features

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

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.

A metric is a function that is used to judge the performance of your model. categorical_accuracy top_k_categorical_accuracy batch_timer

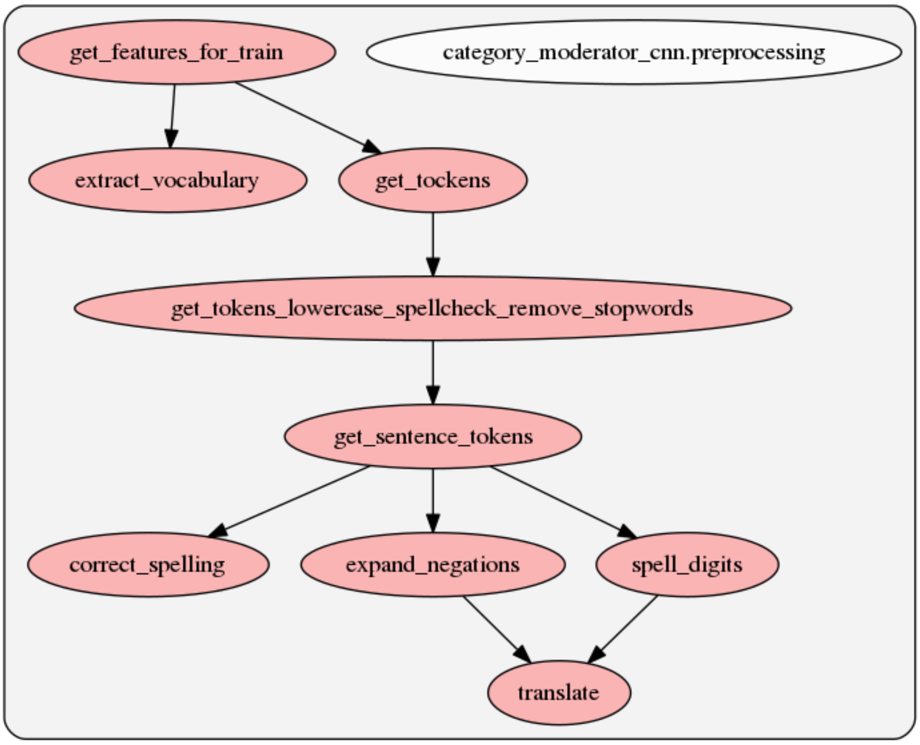


Figure 3.1 — Simplified event structure of data preprocessing

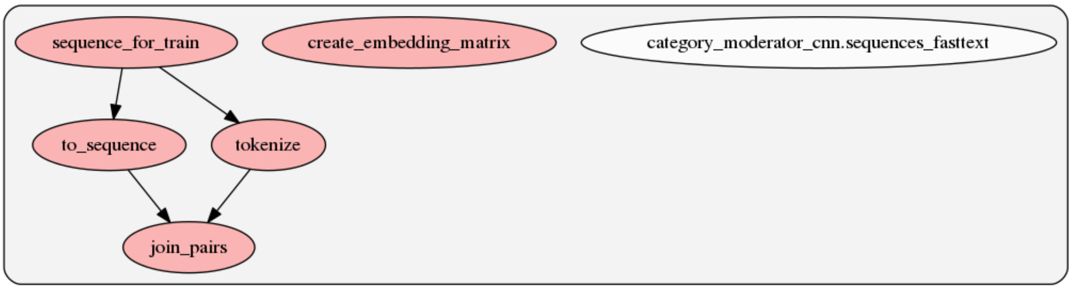


Figure 3.2 — 1

Table 3.7

Simplified event structure of data preprocessing

Functions	**Explanation**
get_features_for_train	Unifying function which upload raw data and call nested functions
extract_vocabulary	form vocabulary from unique words
get_tokens	Parallel batch execution of texts preprocessing which save and return preprocessed tokens for both test and train.
get_tokens_lowercase_spellcheck	wrap over get_sentence_tokens which set necessary flags for it
get_sentence_tokens	recieves single sentence breaks 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

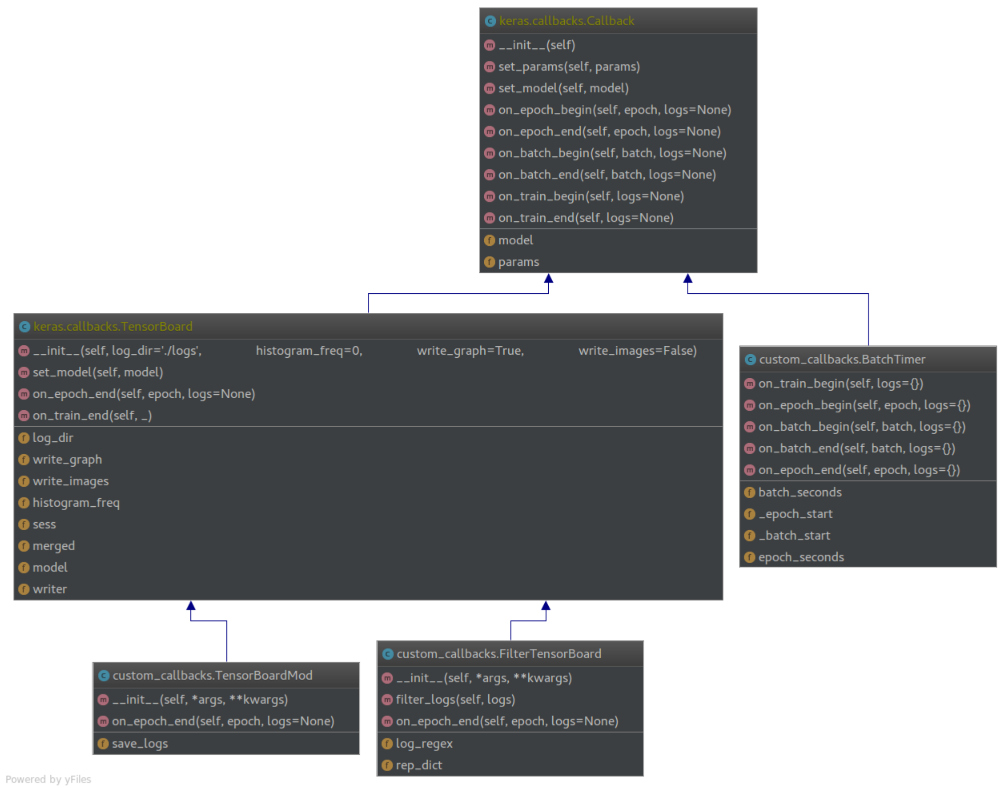


Figure 3.3 — 3

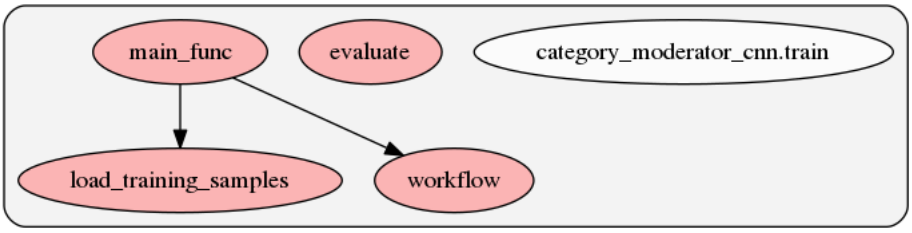

main_func
evaluate
category_moderator_cnn.train
load_training_samples
workflow

Figure 3.4 — 4

4. Classification results evaluation

4.1 Base line model

It is a common knowledge that RNN based neural networks shows the best performance for the text based problems. For my goal, I used Bidirectional Long-Short Term Memory (bi-LSTM) model with 100 ... HERE DESCRIBE MORE ABOUT PARAMETERS

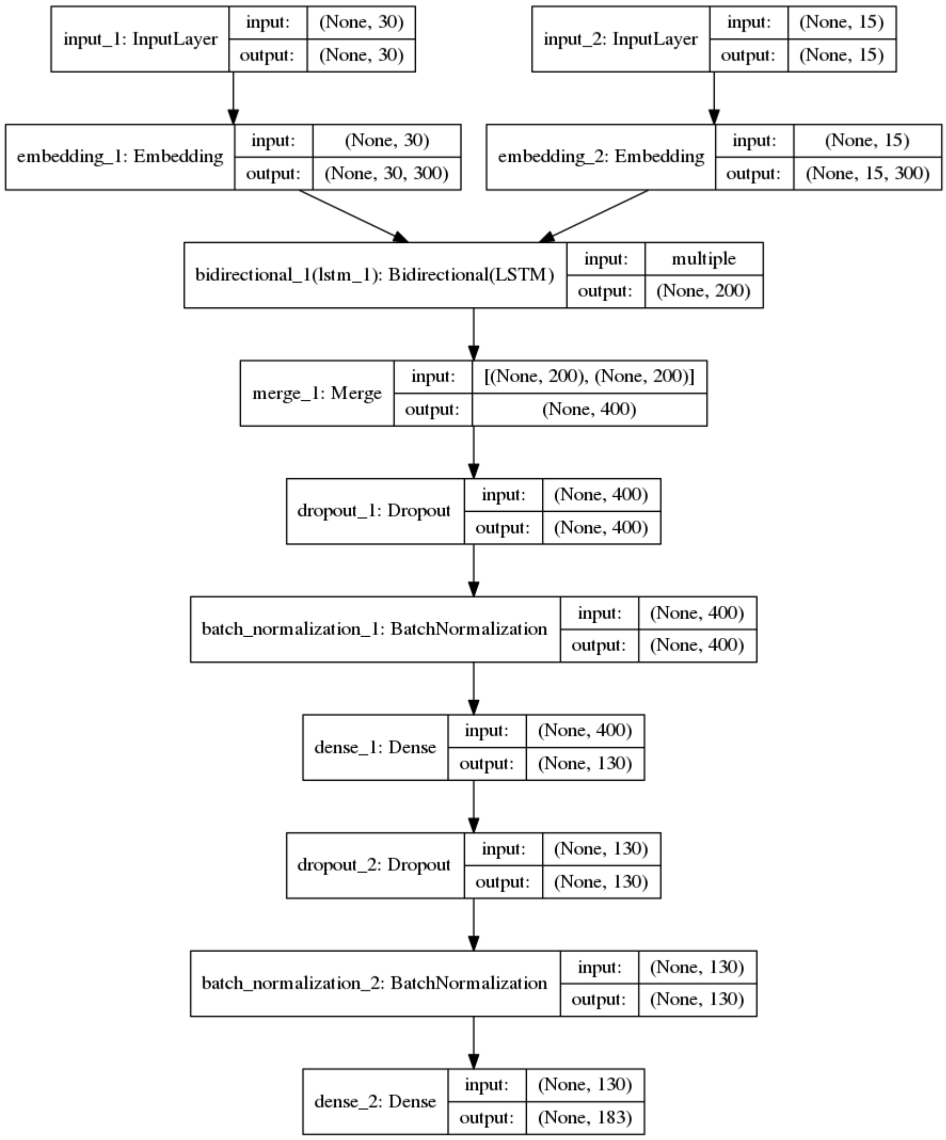


Figure 4.1 — Architectures of Bi-LSTM models with 100 units

top_k_categorical_accuracy

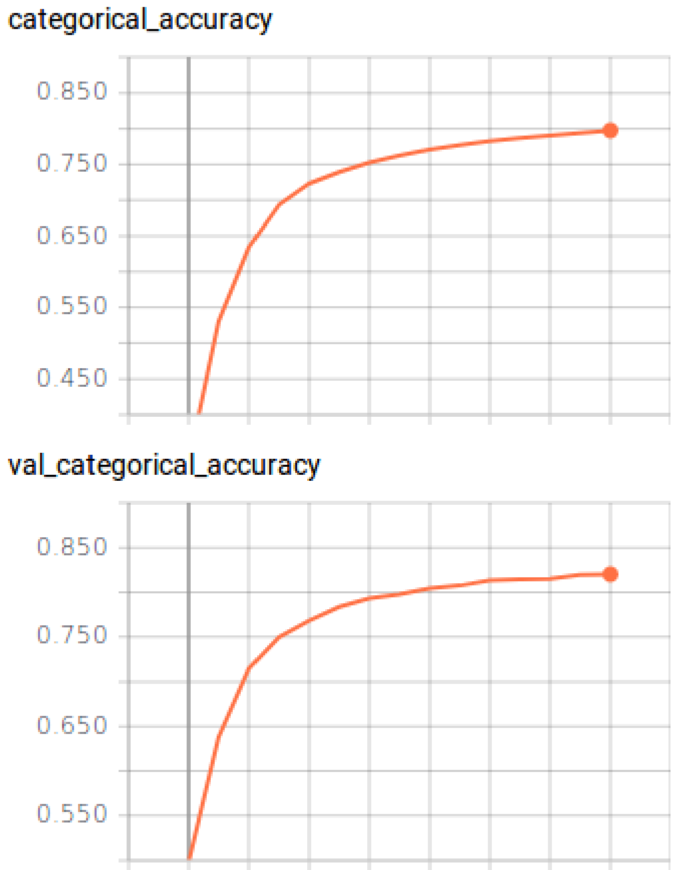


Figure 4.2 — Models train and validation categorical accuracy by epochs

4.2 Convolution neural network

top_k_categorical_accuracy

4.3 Convolution neural network with different regularization

4.4 Final model

4.5 Classification results

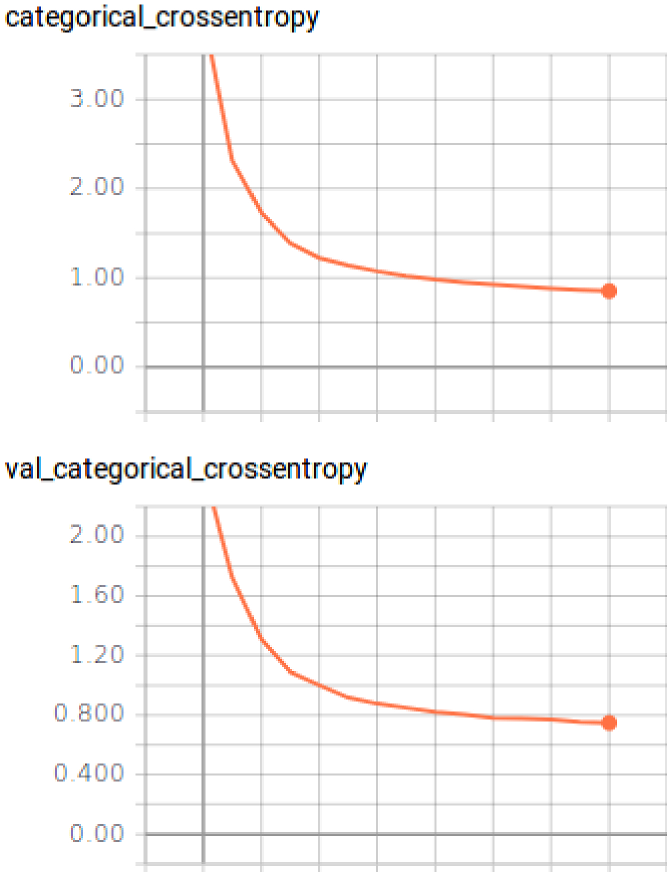


Figure 4.3 — Models train and validation category crossentropy by epochs

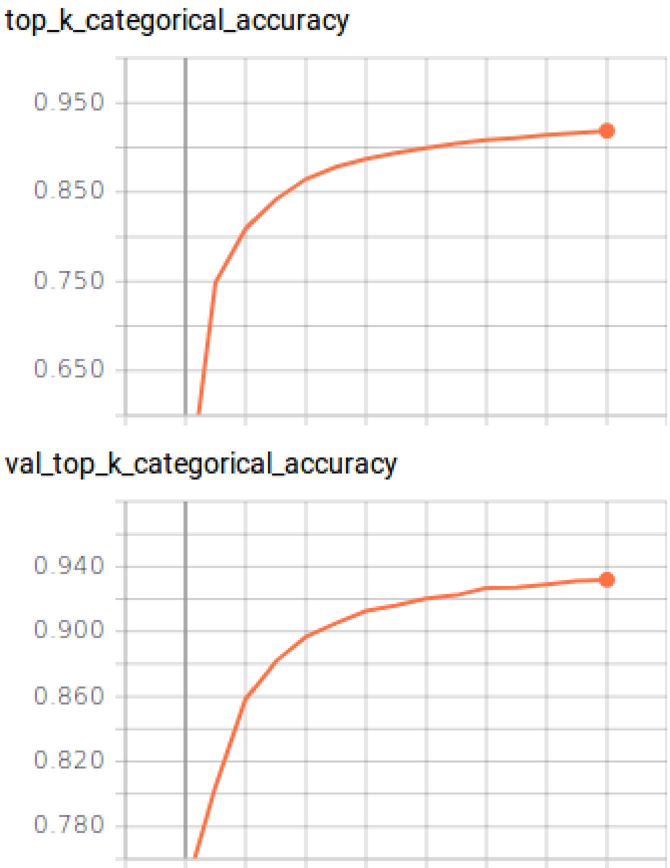


Figure 4.4 — Models train and validation top k accuracy by epochs

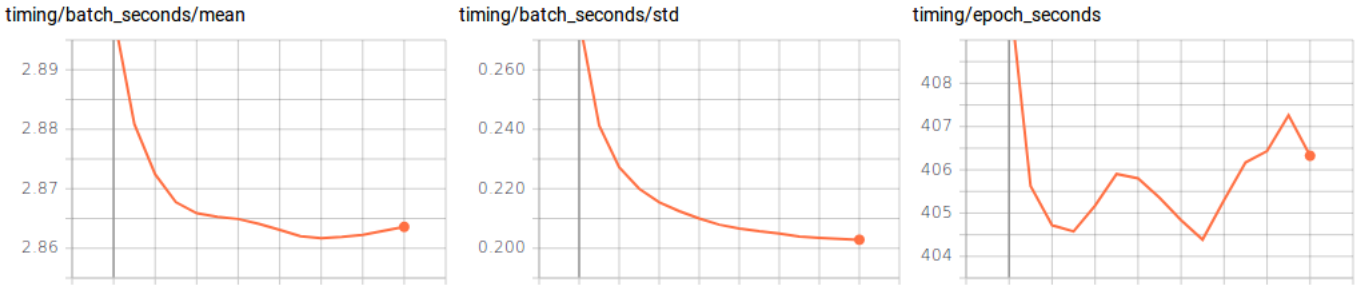
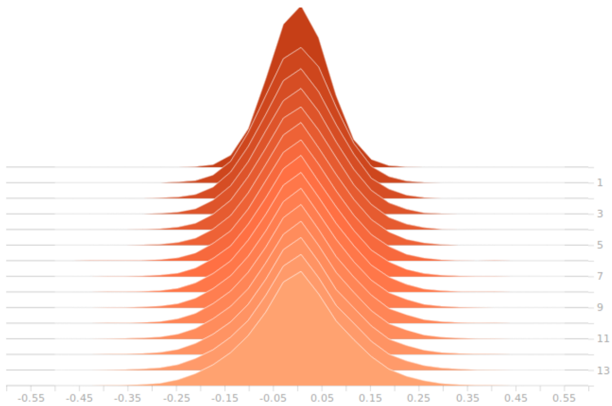
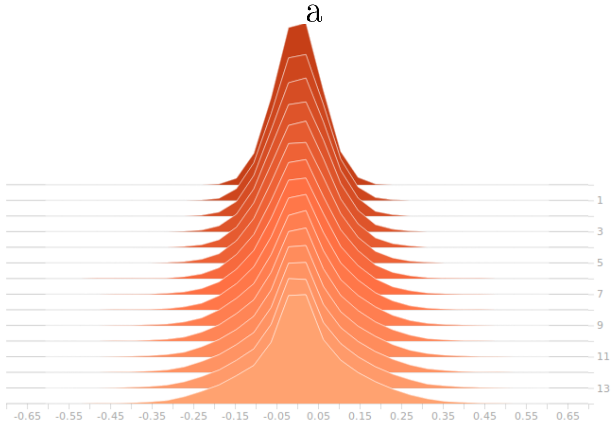


Figure 4.5 — Models batch time by epochs



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)

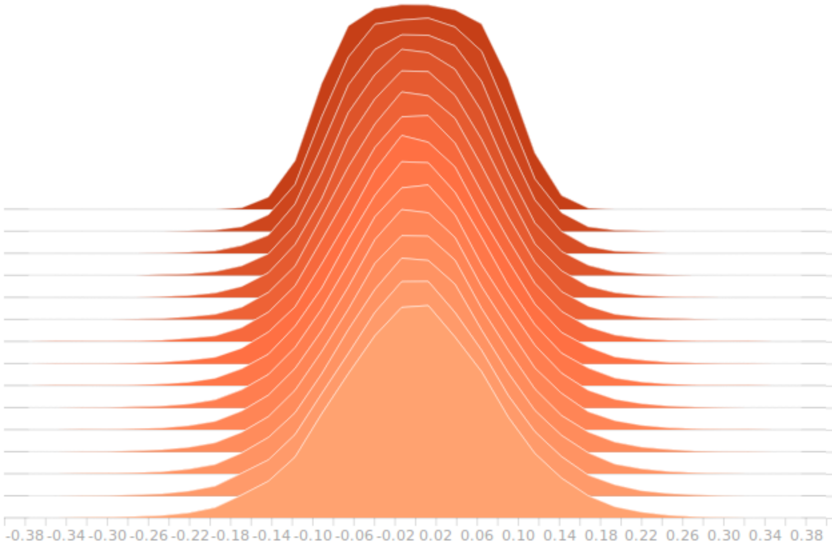


Figure 4.7 — Bi-LSTM 100 units. Histogram of weights from first FFNN layer.

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

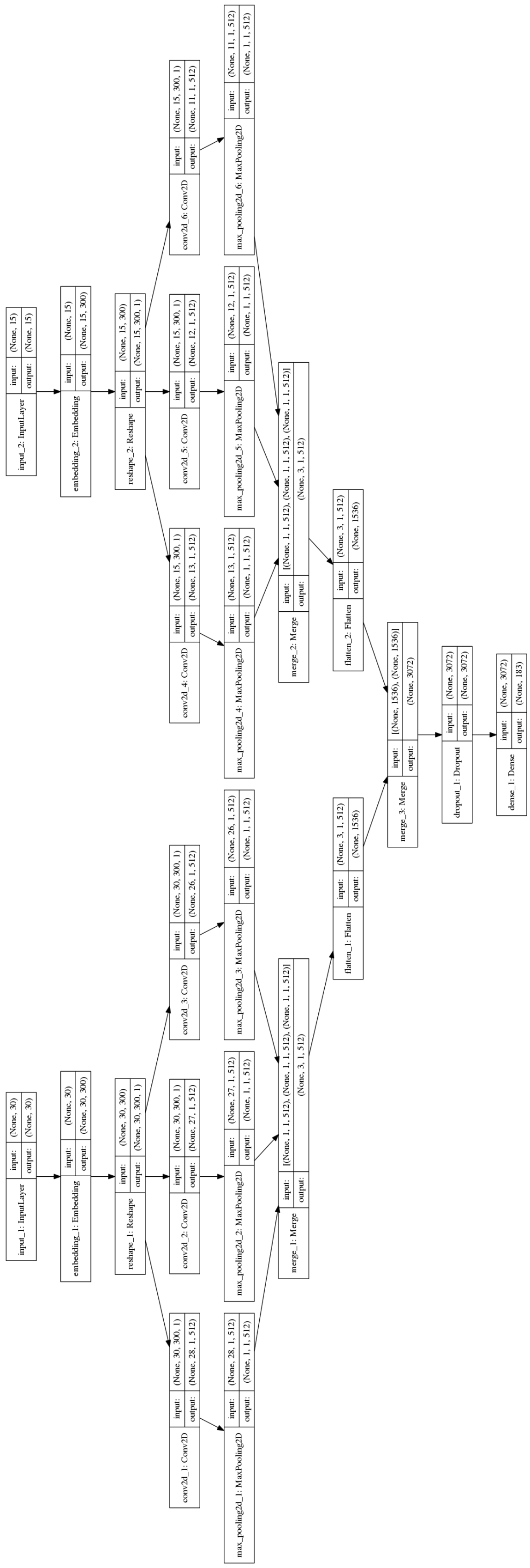


Figure 4.9 — Architectures of CNN model

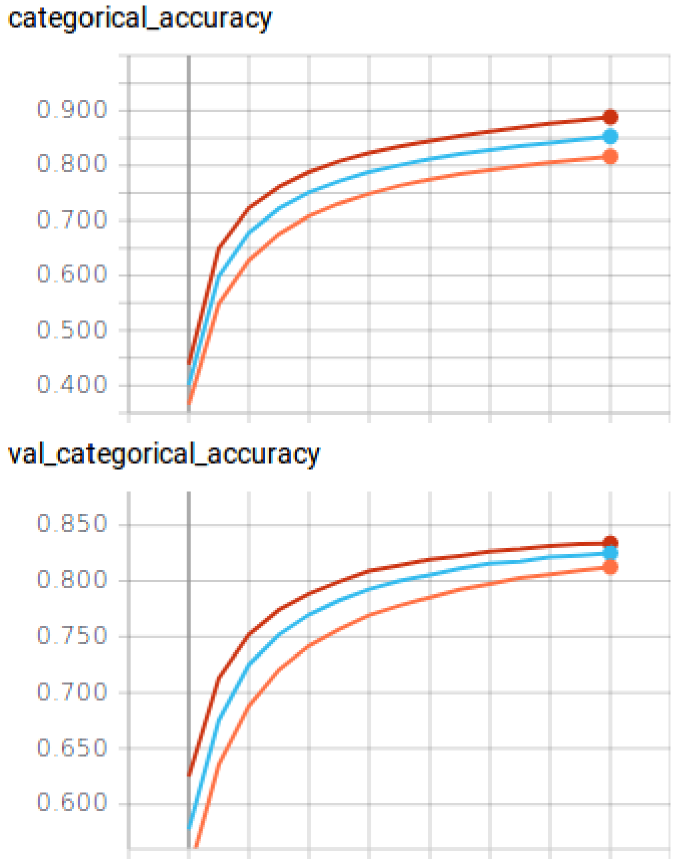


Figure 4.10 — Models train and validation categorical accuracy by epochs

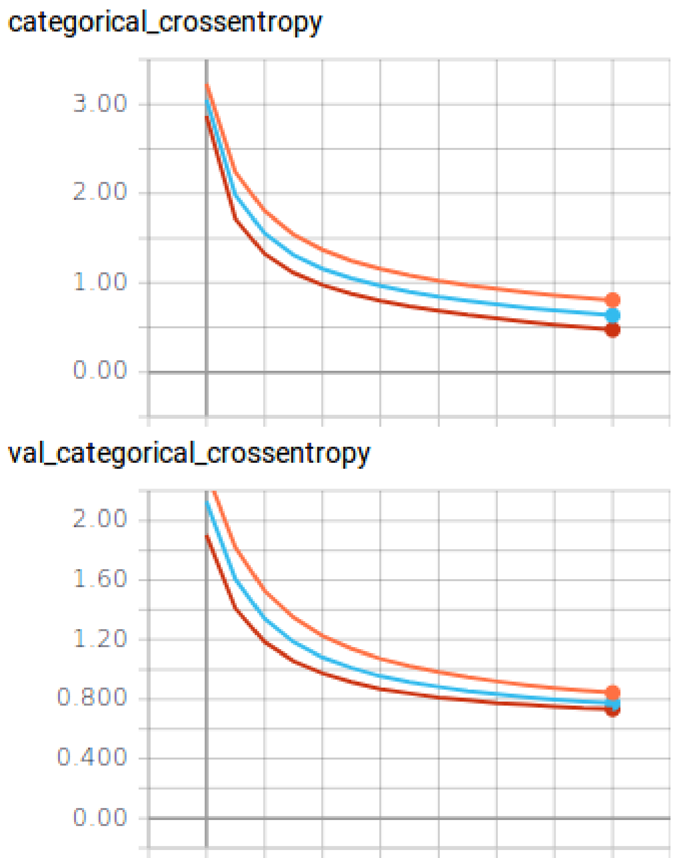


Figure 4.11 — Models train and validation category crossentropy by epochs

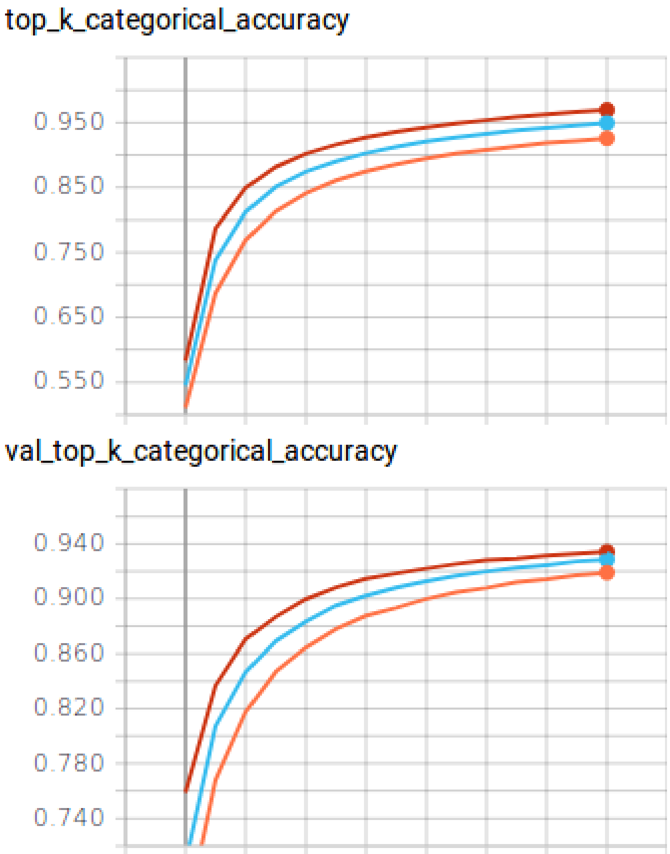


Figure 4.12 — Models train and validation top k accuracy by epochs

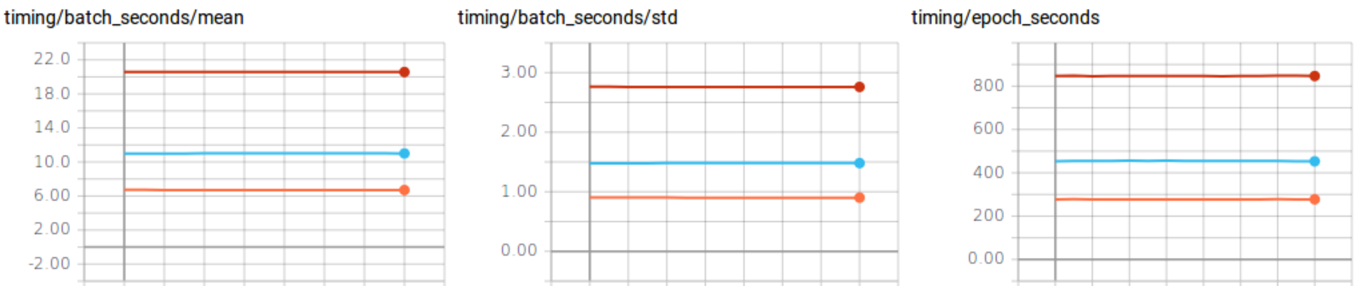
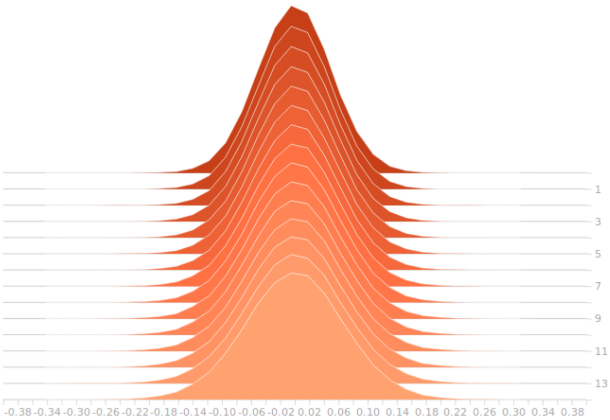
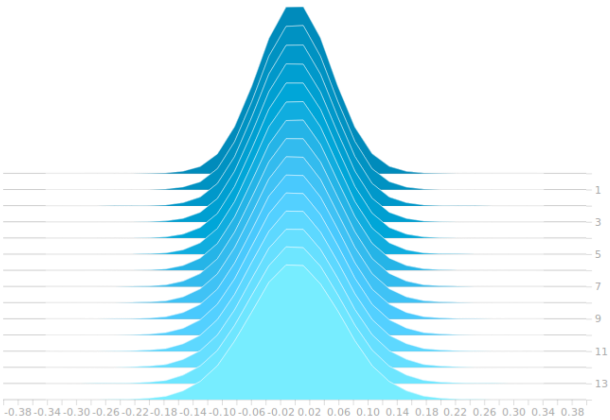


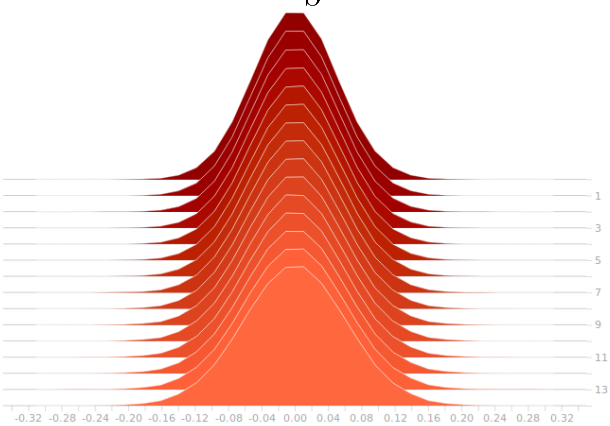
Figure 4.13 — Models batch time by epochs



a

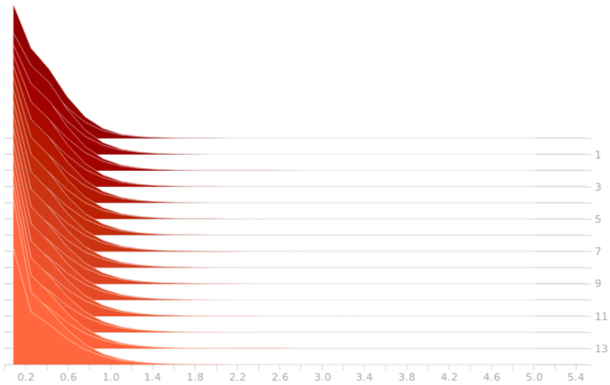


b

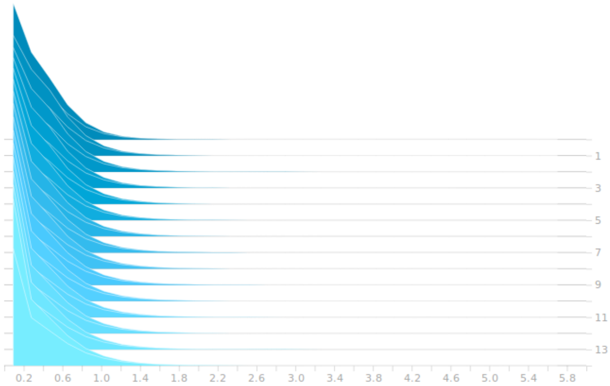


c

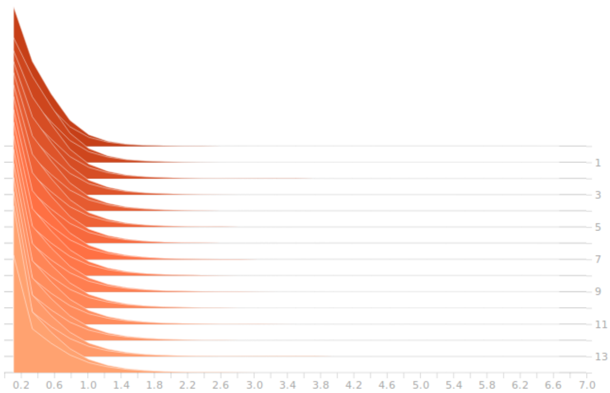
Figure 4.14 — Convolutional model (a) 128;(b) 256; (c) 512 filters for each sizes [3, 4, 5]. Histogram of convolution layers



a

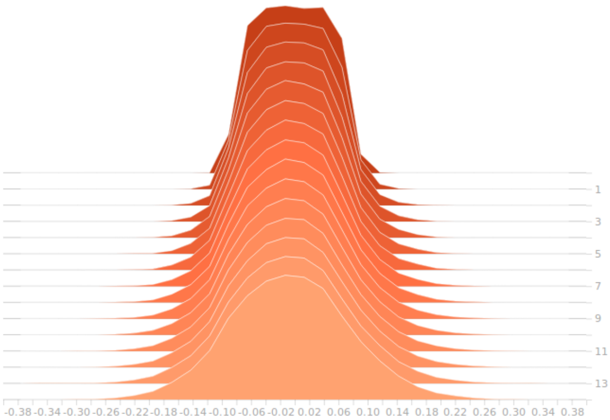


b

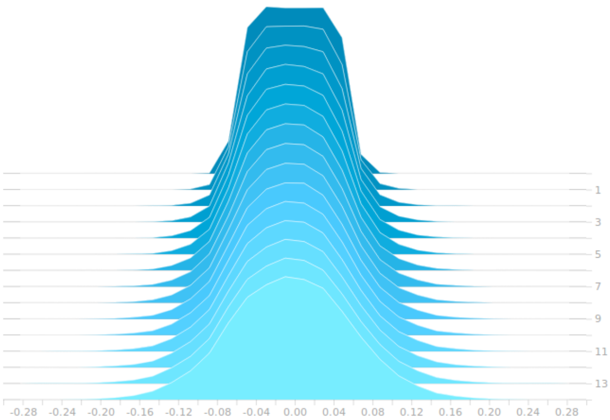


c

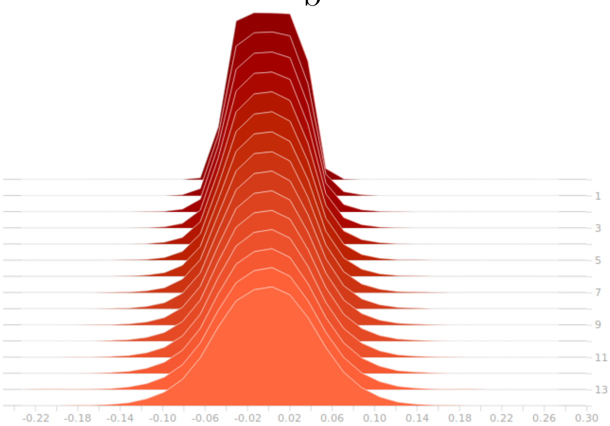
Figure 4.15 — Convolutional model (a) 128;(b) 256; (c) 512 filters for each sizes [3, 4, 5]. Histogram of merged layers



a



b



c

Figure 4.16 — Convolutional model (a) 128;(b) 256; (c) 512 filters for each sizes [3, 4, 5]. Histogram of dense layers

Figure 4.17 — CPU resources which were used while training CNN.

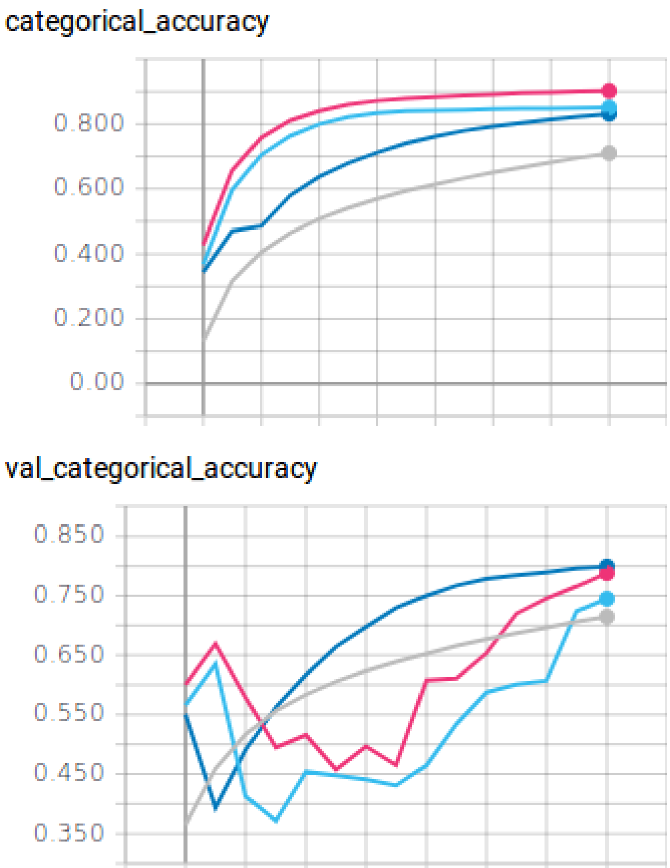


Figure 4.18 — Models train and validation categorical accuracy by epochs

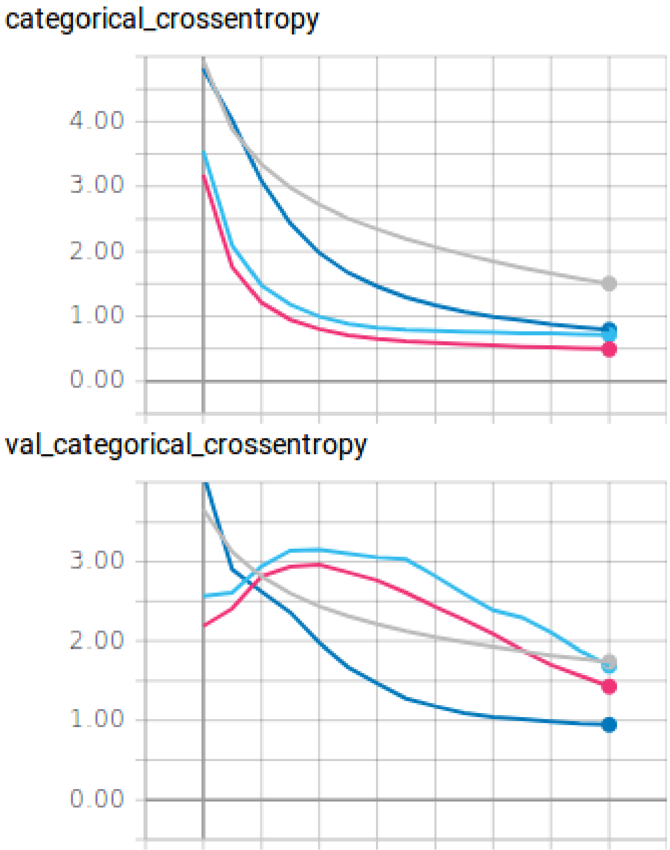


Figure 4.19 — Models train and validation category crossentropy by epochs

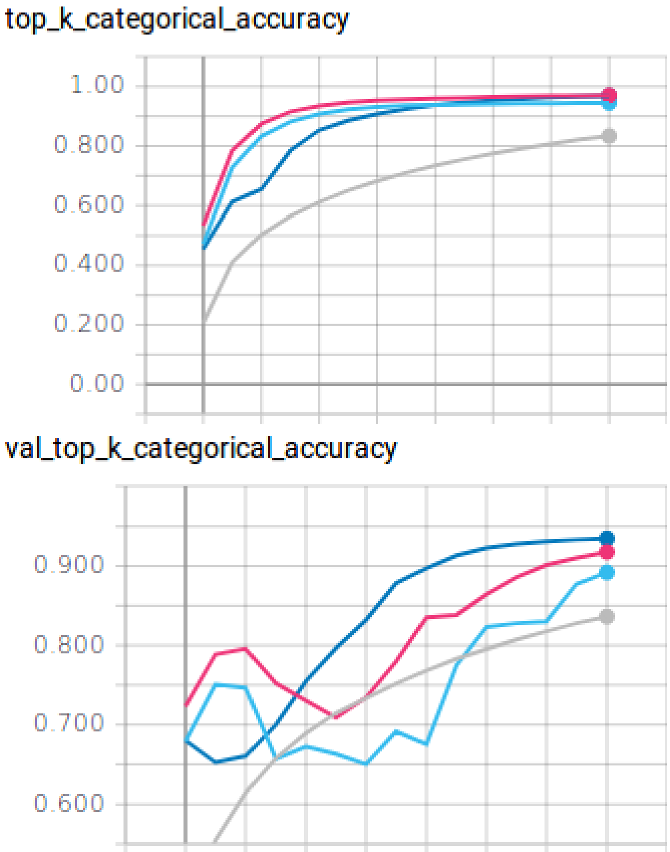


Figure 4.20 — Models train and validation top k accuracy by epochs

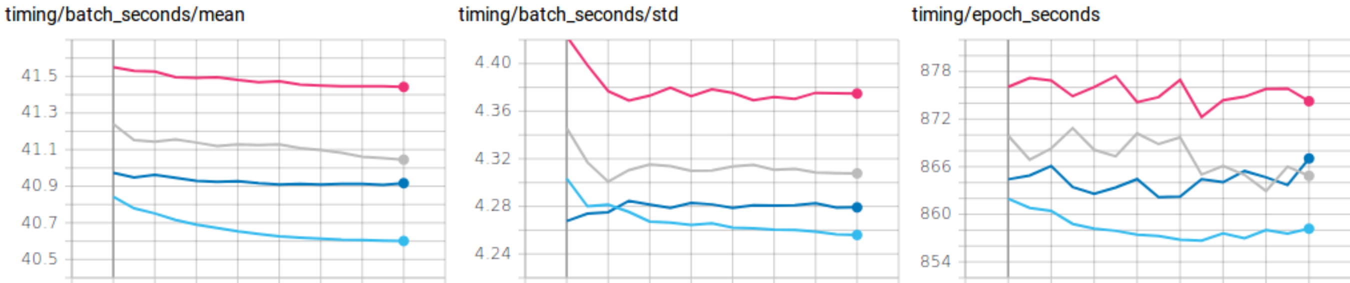
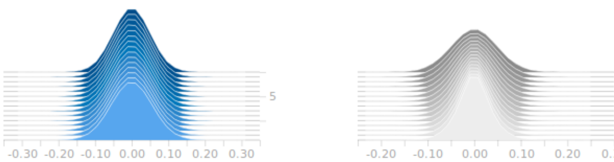


Figure 4.21 — Models batch time by epochs

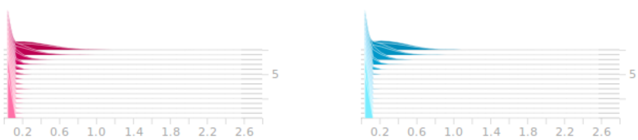


a

b

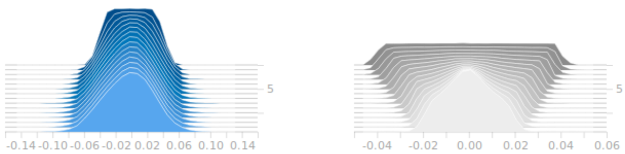
Figure 4.22 — Convolutional model (a) 128;(b) 256; (c) 512 filters for each sizes [3, 4, 5]. Histogram of convolution layers

a

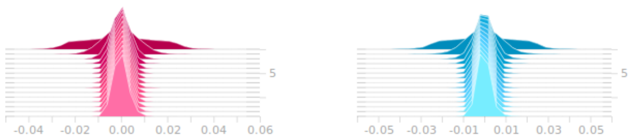


b

Figure 4.23 — Convolutional model (a) 128;(b) 256; (c) 512 filters for each sizes [3, 4, 5]. Histogram of convolution layers



a



b

Figure 4.24 — Convolutional model (a) 128;(b) 256; (c) 512 filters for each sizes [3, 4, 5]. Histogram of convolution layers

Conclusion

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. 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.

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

11.

12.

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

14. 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

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

16. 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.

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

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

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

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

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

List of figures

List of tables

Приложение А

Appendix

Table A.1

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

Table A.1

Classification report

category	precision	recall	f1-score	support
31	0.85	0.52	0.64	224
33	0.9	0.95	0.92	434
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

Table A.1

Classification report

category	precision	recall	f1-score	support
74	0.59	0.73	0.65	146
75	0	0	0	3
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

Table A.1

Classification report

category	precision	recall	f1-score	support
106	0	0	0	3
107	0.8	0.6	0.69	68
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

Table A.1

Classification report

category	precision	recall	f1-score	support
137	0.8	0.85	0.82	142
138	0.76	0.62	0.69	56
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

Table A.1

Classification report

category	precision	recall	f1-score	support
249	0.57	0.46	0.51	177
250	0.28	0.08	0.13	109
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

Table A.1

Classification report

category	precision	recall	f1-score	support
avg	0.81	0.82	0.81	55000