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

Оглавление

Стр.

Abbreviation

ANN (Artificial Neural Network) - штучна нейронна мережа

LSTM (Long Short Term Memory) - мережа з довгою коростроковою пам'яттю

BiLSTM (Bidirectional Long Short Term Memory) - двонапрямлена мережа з довгою коростроковою пам'яттю

CNN (Convolutional Neural Network) - згорткова нейронна мережа

CTC (Connectionist Temporal Classification) - нейромережева часова класифікація

ReLU (Rectified Linear Unit) - активаційна функція виправлений лінійний модуль

Введение

Обзор, введение в тему, обозначение места данной работы в мировых исследованиях и т. п.

Целью данной работы является ...

Для достижения поставленной цели необходимо было решить следующие задачи:

1. Исследовать, разработать, вычислить и т. д. и т. п.
2. Исследовать, разработать, вычислить и т. д. и т. п.
3. Исследовать, разработать, вычислить и т. д. и т. п.
4. Исследовать, разработать, вычислить и т. д. и т. п.

Основные положения, выносимые на защиту:

1. Первое положение
2. Второе положение
3. Третье положение
4. Четвертое положение

Научная новизна:

1. Впервые ...
2. Впервые ...
3. Было выполнено оригинальное исследование ...

Научная и практическая значимость ...

Степень достоверности полученных результатов обеспечивается ... Результаты находятся в соответствии с результатами, полученными другими авторами.

Апробация работы. Основные результаты работы докладывались на: перечисление основных конференций, симпозиумов и т. п.

Личный вклад. Автор принимал активное участие ...

Публикации. Основные результаты по теме диссертации изложены в XX печатных изданиях [?; ?; ?; ?], X из которых изданы в журналах, рекомендованных ВАК [?; ?], XX — в тезисах докладов [?; ?].

Объем и структура работы. Диссертация состоит из введения, четырёх глав, заключения и двух приложений. Полный объём диссертации состав-

ляет 71 страницу с 28 рисунками и 4 таблицами. Список литературы содержит 14 наименований.

Глава 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:

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

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 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]

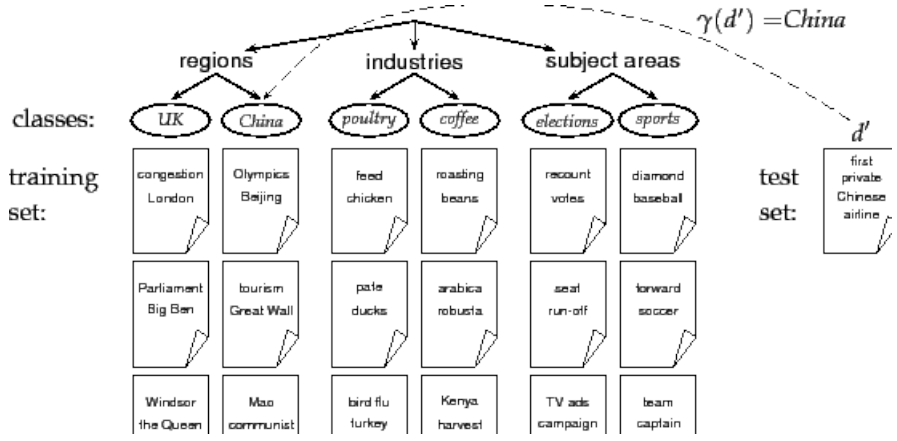

Figure 1.1 — Classes, training set, and test set in text classification.

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

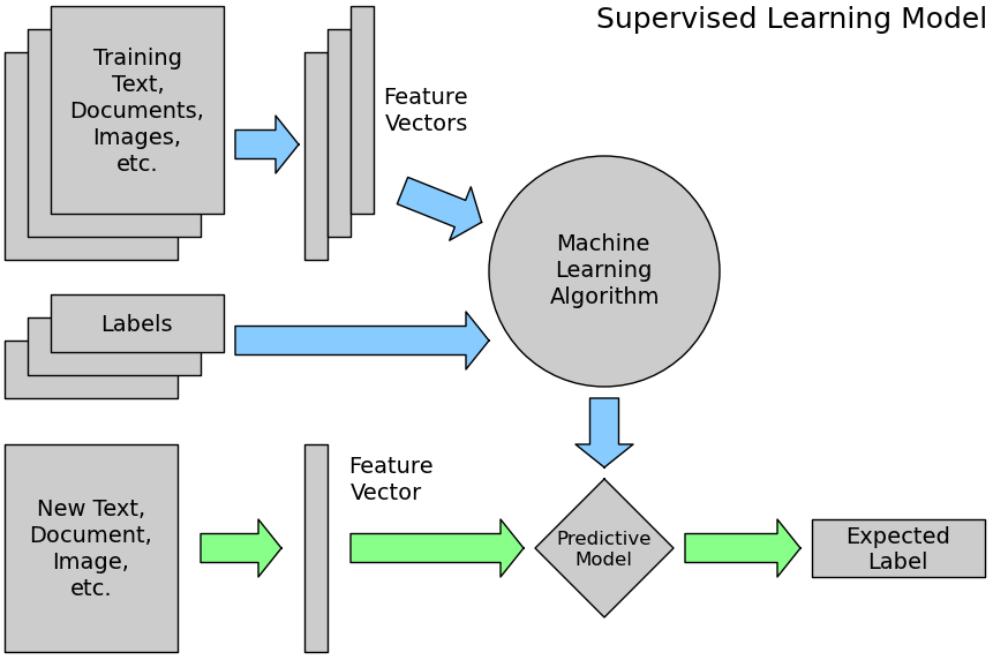

Figure 1.2 — Supervised learning work flow.

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. **Naive Bayes** - a probabilistic learning method. The probability of a document d being in class c is computed as

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c) \tag{1.3}$$

where $P(t_k|c)$ is the conditional probability of term t_k occurring in a document of class c. We interpret $P(t_k|c)$ as a measure of how much evidence t_k contributes that c is the correct class. $P(c)$ is the prior probability of a document occurring in class c. If a document's terms do not provide clear evidence for one class versus another, we choose the one that has a higher prior probability. $\langle t_1, t_2, \ldots, t_{n_d} \rangle$ are the tokens in d that are part of the vocabulary we use for classification and n_d is the number of such tokens in d.

In text classification, our goal is to find the best class for the document. The best class in NB classification is the most likely or maximum a posteriori (MAP) class c_{map}:

$$c_{map} = argmax_{c \in \mathbb{C}} \hat{P}(c|d) = argmax_{c \in \mathbb{C}} \hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c). \tag{1.4}$$

For the priors this estimate is

$$\hat{P}(c) = \frac{N_c}{N}, \tag{1.5}$$

where N_c is the number of documents in class c and N is the total number of documents.
We estimate the conditional probability $\hat{P}(t|c)$ as the relative frequency of term t in documents belonging to class c:

$$\hat{P}(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V}(T_{ct'} + 1)}, \tag{1.6}$$

where T_{ct} is the number of occurrences of t in training documents from class c, including multiple occurrences of a term in a document T_{ct} is a count of occurrences in all positions k in the documents in the training set, V - vocabulary To eliminate zeros, we use add-one or Laplace smoothing, which simply adds one to each count. [2, p.258-260]

```
TrainMultinomialNB(ℂ, 𝔻)
 1  V ← ExtractVocabulary(𝔻)
 2  N ← CountDocs(𝔻)
 3  for each c ∈ ℂ
 4  do N_c ← CountDocsInClass(𝔻, c)
 5     prior[c] ← N_c/N
 6     text_c ← ConcatenateTextOfAllDocsInClass(𝔻, c)
 7     for each t ∈ V
 8     do T_ct ← CountTokensOfTerm(text_c, t)
 9     for each t ∈ V
10     do condprob[t][c] ← (T_ct+1) / (Σ_t'(T_ct'+1))
11  return V, prior, condprob

ApplyMultinomialNB(ℂ, V, prior, condprob, d)
1  W ← ExtractTokensFromDoc(V, d)
2  for each c ∈ ℂ
3  do score[c] ← log prior[c]
4     for each t ∈ W
5     do score[c] += log condprob[t][c]
6  return arg max_{c∈ℂ} score[c]
```

Figure 1.3 — Naive Bayes algorithm (multinomial model): Training and testing.

2.**Logistic Regression**. For now, we will focus on the binary classification problem in which y can take on only two values, 0 and 1

1.4 Model evaluation

Model evaluation is not just the end point of our machine learning pipeline. Before we handle any data, we want to plan ahead and use techniques and metrics that are suited for our purposes.

1.4.1 Model Evaluation Applications

Generalization performance - We want to estimate the predictive performance of our model on future (unseen) data. - Ideally, the estimated performance of a model tells how well it performs on unseen data – making predictions on future data is often the main problem we want to solve.

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. - Typically, machine learning involves a lot of experimentation. Running a learning algorithm over a training dataset with different hyperparameter settings and different features will result in different models. Since we are typically interested in selecting the best-performing model from this set, we need to find a way to estimate their respective performances in order to rank them against each other.

Algorithm selection - We want to compare different ML algorithms, selecting the best-performing one. - We are usually not only experimenting with the one single algorithm that we think would be the "best solution" under the given circumstances. More often than not, we want to compare different algorithms to each other, oftentimes in terms of predictive and computational performance.

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 simplest model evaluation technique. We take our labeled dataset and split it randomly into two parts: A **training set** and a **test set**

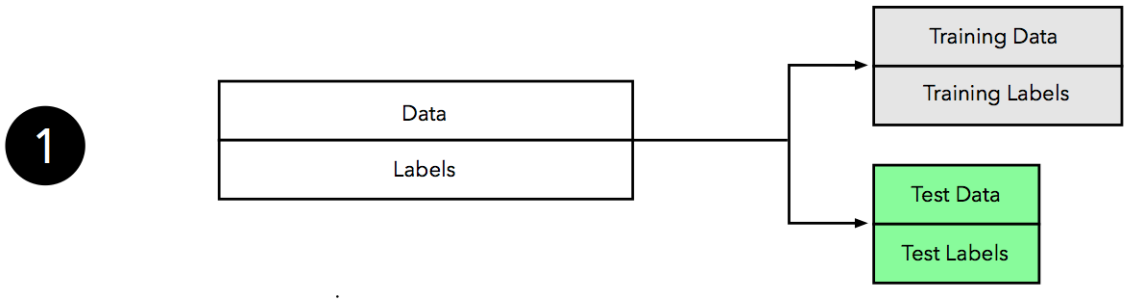

Figure 1.4 — .

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

And the fraction of correct predictions constitutes our estimate of the prediction accuracy.

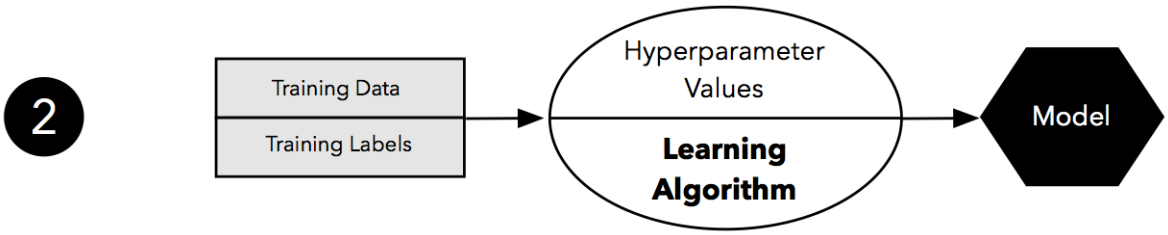

Figure 1.5 — .

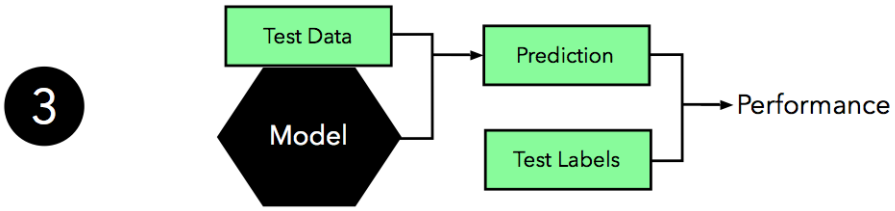

Figure 1.6 — .

We really don't want to train and evaluate our model on the same training dataset, since it would introduce **overfitting**. In other words, we can't tell whether the model simply memorized the training data or not, or whether it generalizes well to new, unseen data. [14]

1.5 Classification metrics

Classification problems are probably the most common type of ML problem and as such there are many metrics that can be used to evaluate predictions for these problems. We will review some of them.

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.

$$accuracy = \frac{correct}{predictions} \tag{1.7}$$

Accuracy is also the most misused metric. It is really **only suitable** when there are 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.

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

Confusion matrix allows you to compute various classification metrics, and these metrics can guide your model selection.

Precision and Recall

Precision and recall are actually two metrics. But they are often used together.

Precision answers the question: What percent of positive predictions were correct?

$$precision = \frac{\#\ true\ positive}{\#\ true\ positive + \#\ false\ positive} \tag{1.8}$$

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

$$recall = \frac{\#\ true\ positive}{\#\ true\ positive + \#\ false\ negative} \tag{1.9}$$

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:

$$F_1 = 2\frac{precision * recall}{precision + recall} \tag{1.10}$$

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.

Глава 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:

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

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']

2. **Character n-grams - helps to deal with misspelling**:
   - 4-grams: "The sun is yellow."→ ['The ', 'sun ', 'is y', 'ello' ...]
3. **Skip-n-gram - sequence of N basic tokens, having distance of $\leq K$ tokens between them**
   - 1-skip-2-grams:: "The sun is yellow."→ ['The is', 'sun 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 4.4 and sigmoid 4.5 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:

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

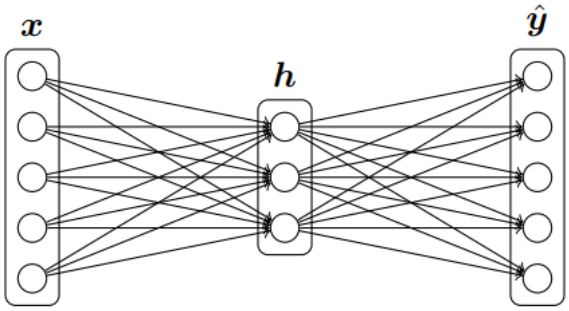

Figure 2.1 — Neural Network

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 4.16 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 4.8, 4.10, 4.11,

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

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

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

Which is equivalent to,

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

Forward propagation is as follows:

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

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

where $\mathrm{W_i}$ 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 4.7:

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

and

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

Sigmoid (σ) derivative 4.6:

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

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 4.19:

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

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

We have everything to update our weights:

Now, turn definitely to skip-gram model shown in Figure 4.2 [11]:

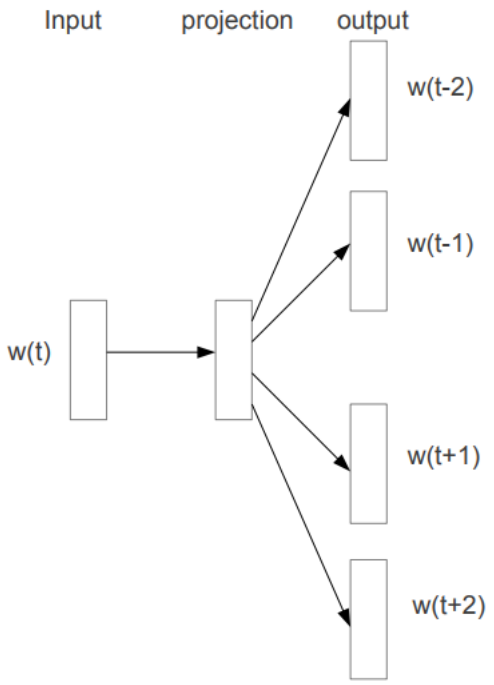

Figure 2.2 — The Skip-gram model architecture.

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 4.26 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 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],

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

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:

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

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 4.3:

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 4.4 [11]:

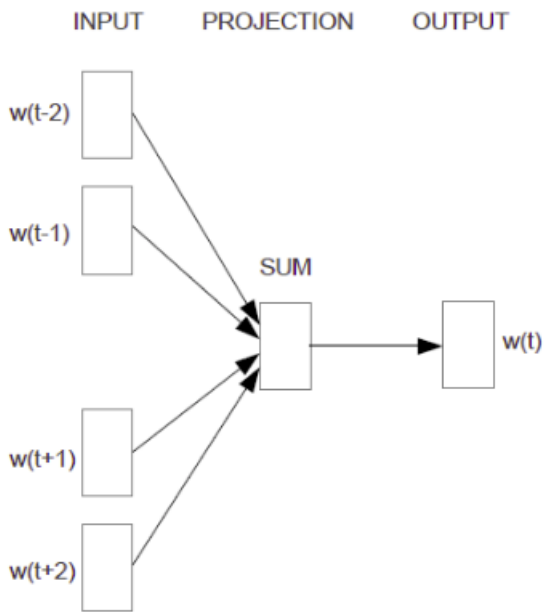

Figure 2.4 — The CBOW model architecture.

2.2 Deep learning algorithms for text classification

Nowadays, there are a great number of NN architectures which are used for text classification. In this section I would like to consider the most powerful and efficient ones.

2.2.1 Convolution Neural Networks

The model architecture, shown in Figure 4.5 [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]

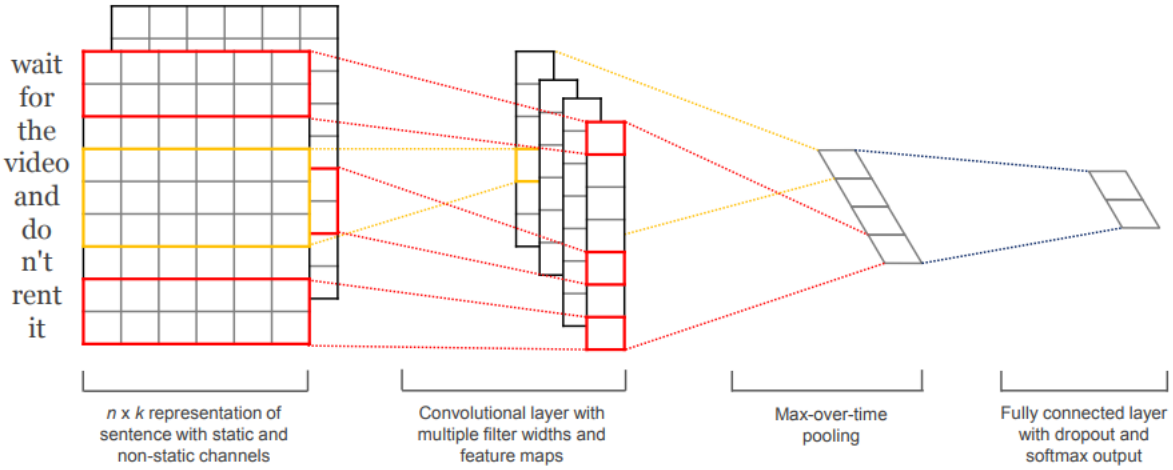

Figure 2.5 — Convolution Neural Networks architecture for text classification

Convolution

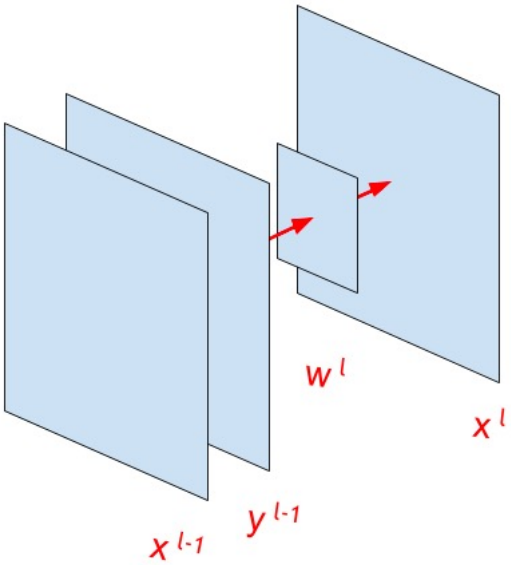

Figure 2.6 — Basic variables which are used in the convolution layer

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 4.6 [13].

The following equation 4.35 describes words above into mathematical way:

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

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** 4.36 - this activation function was used at Convolution layers. It has the following properties:

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

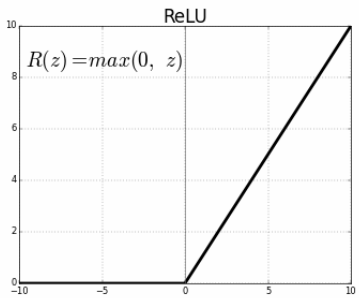

Figure 2.7 — ReLu activation function

2) **Softmax** 4.4 - 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.

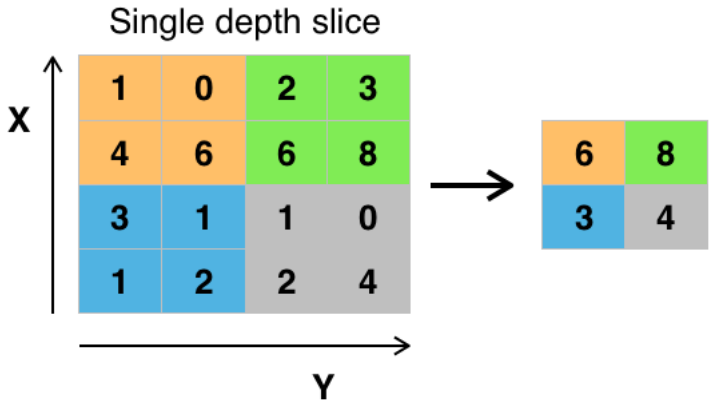

Figure 2.8 — Max pulling layer

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 4.1 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 4.7 described above.

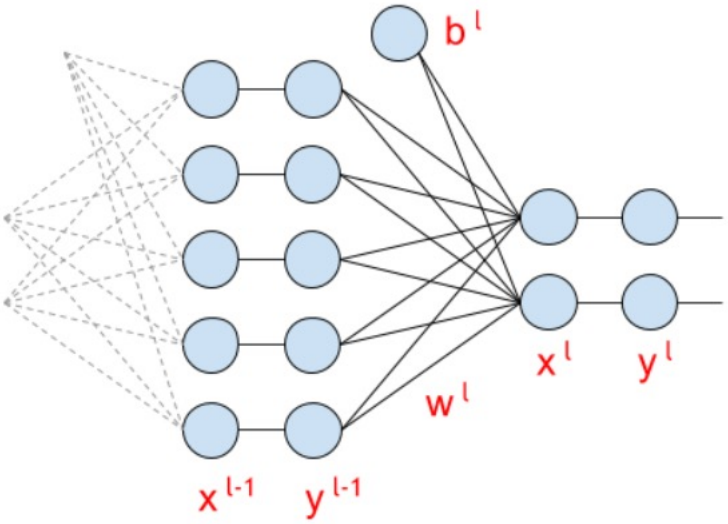

Figure 2.9 — Fully connected layer of CNN

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 4.10, 4.11, 4.12. Therefore, we have following equation 4.39:

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

or

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

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.

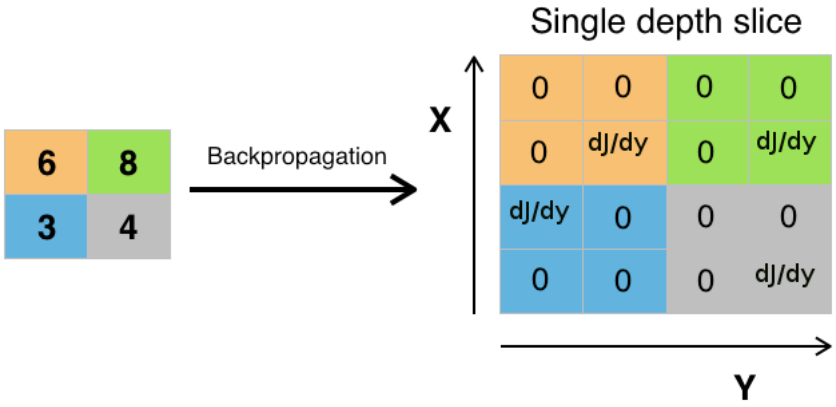

Figure 2.10 — Back propagation through max pulling layer

It is necessary to derive weights update for kernel 4.11.

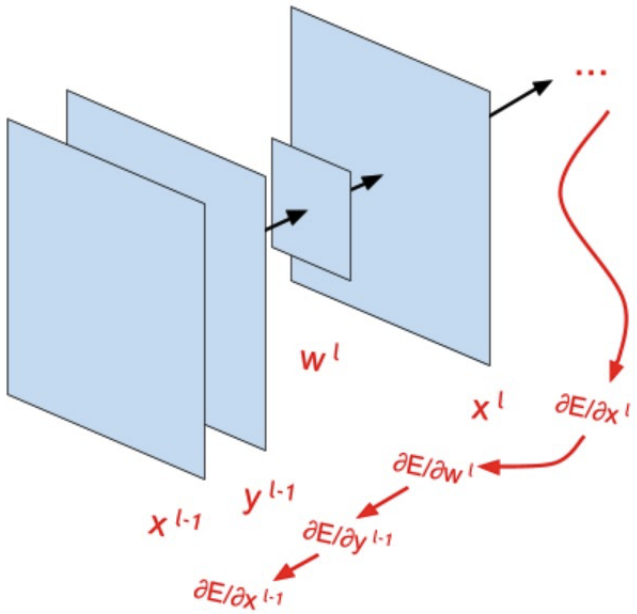

Figure 2.11 — Back propagation through convolution layer

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

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

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.

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

Глава 3. Вёрстка таблиц

3.1 Таблица обыкновенная

Глава 4. Mathematical models and algorithms for text classification

4.1 Words representations

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

$$A(X, \hat{w}) : A(X, \hat{w}) \simeq f(X) \Leftrightarrow A(X, \hat{w}) = \arg\min_{w} \|A(X, w) - f(X)\| \qquad (4.1)$$

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]

4.1.1 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 4.1

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']

2. **Character n-grams - helps to deal with misspelling**:
   - 4-grams: "The sun is yellow."→ ['The ', 'sun ', 'is y', 'ello' ...]
3. **Skip-n-gram - sequence of N basic tokens, having distance of $\leq K$ tokens between them**
   - 1-skip-2-grams:: "The sun is yellow."→ ['The is', 'sun yellow ']

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

4.1.2 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:

$$\begin{cases} TF(w,T) = n_{Tw} \\ IDF(w,T) = log\frac{N}{n_w} \end{cases} \implies TF\text{-}IDF(w,T) = n_{Tw}\ log\frac{N}{n_w} \quad \forall w \in W \tag{4.2}$$

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.

$$\lim_{n_w \to N} TF\text{-}IDF(w,T) = 0 \tag{4.3}$$

4.1.3 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 4.4 and sigmoid 4.5 functions,

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1} e^{x_j}} \tag{4.4}$$

$$\text{sigmoid} = \sigma(z) = \frac{1}{1+e^{-z}}. \tag{4.5}$$

The gradient of sigmoid function is follows:

$$\sigma'(z) = \sigma(z)(1-\sigma(z)) \tag{4.6}$$

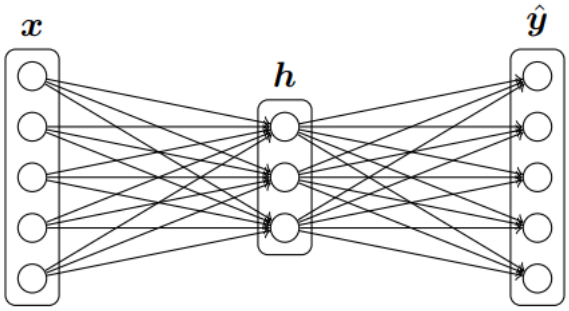

Figure 4.1 — Neural Network

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 4.16 is used.

$$\text{CE}(\mathbf{y},\hat{\mathbf{y}}) = -\sum_i y_i \log \hat{y}_i \tag{4.7}$$

Now, we will compute the gradient of cross entropy:

$$\frac{\partial(\text{CE})}{\partial \hat{y}_i} = -\frac{y_j}{\hat{y}_i} \tag{4.8}$$

That leads,

$$\frac{\partial(\text{CE})}{\partial \theta_k} = \frac{\partial(\text{CE})}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial \theta_k} = -\frac{y_j}{\hat{y}_i}\frac{\partial \hat{y}_i}{\partial \theta_k} \tag{4.9}$$

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:

$$\begin{aligned} \frac{\partial \hat{y}_i}{\partial \theta_i} &= \frac{\partial}{\partial \theta_i}\left(\frac{e^{\theta_i}}{\sum_{j=1} e^{\theta_j}}\right) = \\ &= \frac{e^{\theta_i}}{\sum_{j=1} e^{\theta_j}} - \left(\frac{e^{\theta_i}}{\sum_{j=1} e^{\theta_j}}\right)^2 = \\ &= \hat{y}_i \cdot (1 - \hat{y}_i) \end{aligned} \tag{4.10}$$

and (where $i \neq k$),

$$\begin{aligned} \frac{\partial \hat{y}_i}{\partial \theta_k} &= \frac{\partial}{\partial \theta_k}\left(\frac{e^{\theta_i}}{\sum_{j=1} e^{\theta_j}}\right) = \\ &= -\left(\frac{e^{\theta_i} e^{\theta_k}}{\sum_{j=1} e^{\theta_j}}\right) = -\hat{y}_i \hat{y}_k \end{aligned} \tag{4.11}$$

After combination of equations 4.8, 4.10, 4.11,

$$\frac{\partial(\text{CE})}{\partial \theta_k} = \begin{cases} -y_j(1 - \hat{y}_k) & \text{for } i = k \\ y_j \hat{y}_k & \text{for } i \neq k \end{cases} \tag{4.12}$$

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

$$\frac{\partial(\text{CE})}{\partial \theta_j} = \begin{cases} (\hat{y}_j - 1) & \text{for } i = j \\ \hat{y}_j & \text{for } i \neq j \end{cases} \tag{4.13}$$

Which is equivalent to,

$$\frac{\partial(\text{CE})}{\partial \boldsymbol{\theta}} = \hat{\mathbf{y}} - \mathbf{y} \tag{4.14}$$

Forward propagation is as follows:

$$\mathbf{h} = \text{sigmoid}(\mathbf{x}\mathrm{W}_\mathbf{1} + \mathbf{b_1}) \tag{4.15}$$

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{h}\mathrm{W}_\mathbf{2} + \mathbf{b_2}) \tag{4.16}$$

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:

$$\begin{aligned} \mathbf{z}_1 &\equiv \quad \mathbf{x}\mathrm{W}_\mathbf{1} + \mathbf{b_1} \\ \mathbf{z}_2 &\equiv \quad \mathbf{h}\mathrm{W}_\mathbf{2} + \mathbf{b_2} \end{aligned} \tag{4.17}$$

Starting with the results from 4.7:

$$\frac{\partial J}{\partial \mathbf{z}_2} = \hat{\mathbf{y}} - \mathbf{y} \tag{4.18}$$

and

$$\frac{\partial \mathbf{z}_2}{\partial \mathbf{h}} = \mathrm{W}_\mathbf{2}^\top \tag{4.19}$$

Sigmoid (σ) derivative 4.6:

$$\frac{\partial \mathbf{h}}{\partial \mathbf{z}_1} \equiv \sigma'(z_1) \tag{4.20}$$

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

$$\frac{\partial J}{\partial z_i} = (\hat{\mathbf{y}} - \mathbf{y})\mathrm{W}_\mathbf{2}^\top \cdot \sigma'(\mathbf{z_1}) \tag{4.21}$$

Finally, using the results from Equation 4.19:

$$\frac{\partial J}{\partial \mathbf{W}^{(1)}} = (\hat{\mathbf{y}} - \mathbf{y})\mathrm{W}_\mathbf{2}^\top \cdot \sigma'(\mathbf{z_1}) \cdot \mathrm{X}^\top \tag{4.22}$$

$$\frac{\partial J}{\partial \mathbf{W}^{(2)}} = (\hat{\mathbf{y}} - \mathbf{y})\mathrm{h}^\top \tag{4.23}$$

We have everything to update our weights:

Now, turn definitely to skip-gram model shown in Figure 4.2 [11]:

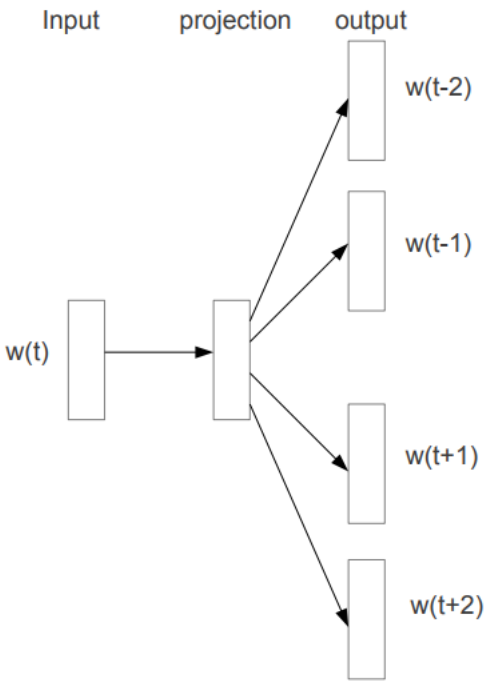

Figure 4.2 — The Skip-gram model architecture.

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:

$$\hat{\boldsymbol{y}}_{\boldsymbol{o}} = p(\ \boldsymbol{o} \mid \boldsymbol{c}\ ) = \frac{\exp\left(\boldsymbol{u}_o^\top \boldsymbol{v}_c\right)}{\sum_{j=1}^{|W|} \exp\left(\boldsymbol{u}_j^\top \boldsymbol{v}_c\right)} \tag{4.24}$$

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:

$$J = -\log p = -\boldsymbol{u}_o^\top \boldsymbol{v}_c + \log \sum_{j=1}^{|V|} \exp\left(\boldsymbol{u}_j^\top \boldsymbol{v}_c\right) \tag{4.25}$$

Let $z_j = \boldsymbol{u}_j^\top \boldsymbol{v}_c$, and δ_j^i 4.26 be the indicator function, then

$$\delta_j^i = \begin{cases} 1, & \text{for } i = j \\ 0, & \text{for } i \neq j \end{cases} \tag{4.26}$$

$$\frac{\partial J}{\partial z_k} = -\delta_k^i + \frac{\exp\left(\boldsymbol{u}_i^\top \boldsymbol{v}_c\right)}{\sum_{j=1}^{|V|} \exp\left(\boldsymbol{u}_j^\top \boldsymbol{v}_c\right)} \tag{4.27}$$

Now, using the chain rule, we can calculate,

$$\begin{aligned} \frac{\partial J}{\partial \boldsymbol{v}_c} &= \frac{\partial J}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{v}_c} = \\ &= \sum_{j=1}^{|V|} \boldsymbol{u}_j^\top \left(\frac{e^{z_j}}{\sum_{k=1}^{|V|} e^{z_k}} - 1 \right) = \\ &= \sum_{k=1}^{|V|} \boldsymbol{P}(\boldsymbol{u}_j | \boldsymbol{v}_c) \boldsymbol{u}_j - \boldsymbol{u}_j \end{aligned} \tag{4.28}$$

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

$$\begin{aligned} \frac{\partial J}{\partial \boldsymbol{u}_j} &= \frac{\partial J}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{u}_j} = \\ &= \boldsymbol{v}_c \left(\frac{\exp\left(\boldsymbol{u}_0^\top \boldsymbol{v}_c\right)}{\sum_{j=1}^{|V|} \exp\left(\boldsymbol{u}_j^\top \boldsymbol{v}_c\right)} - \delta_j^0 \right) \end{aligned} \tag{4.29}$$

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:

$$J = \sum_{-m \le j \le m, j \neq 0} F(\boldsymbol{w}_{c+j}, \boldsymbol{v}_c) \tag{4.30}$$

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],

$$\begin{aligned}\frac{\partial J}{\partial \boldsymbol{w}_k} &= \\ &= \frac{\partial}{\partial \boldsymbol{w}_k} \sum_{-m \le j \le m, j \neq 0} F(\boldsymbol{w}_{c+j}, \boldsymbol{v}_c) = \\ &= \sum_{-m \le j \le m, j \neq 0} \frac{\partial F}{\partial \boldsymbol{w}_{i+j}} \delta_k^{i+j}\end{aligned} \tag{4.31}$$

and

$$\frac{\partial J}{\partial \boldsymbol{v}_c} = \sum_{-m \le j \le m, j \neq 0} \frac{\partial F}{\partial \boldsymbol{v}_c} \tag{4.32}$$

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

$$\begin{aligned} w_k^{new} &= w_k^{old} - \eta \frac{\partial J}{\partial w_k} \\ v_c^{new} &= v_c^{old} - \eta \frac{\partial J}{\partial v_c} \end{aligned} \tag{4.33}$$

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 4.3:

Figure 4.3 — 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,

$$J(\boldsymbol{u}_o, \boldsymbol{v}_c, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)) \tag{4.34}$$

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 4.4 [11]:

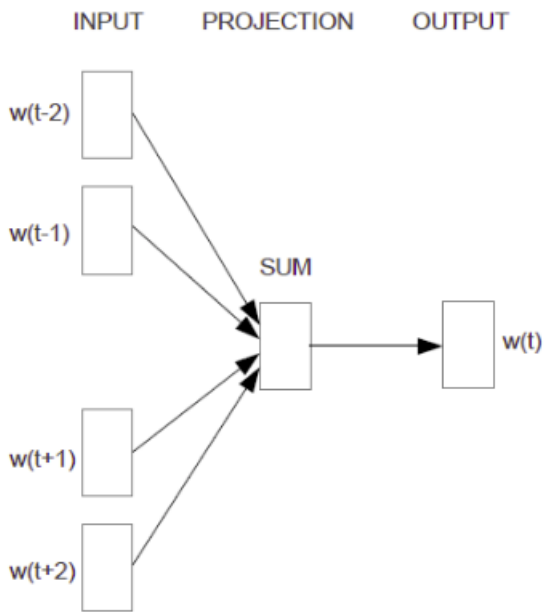

Figure 4.4 — The CBOW model architecture.

4.2 Deep learning algorithms for text classification

Nowadays, there are a great number of NN architectures which are used for text classification. In this section I would like to consider the most powerful and efficient ones.

4.2.1 Convolution Neural Networks

The model architecture, shown in Figure 4.5 [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]

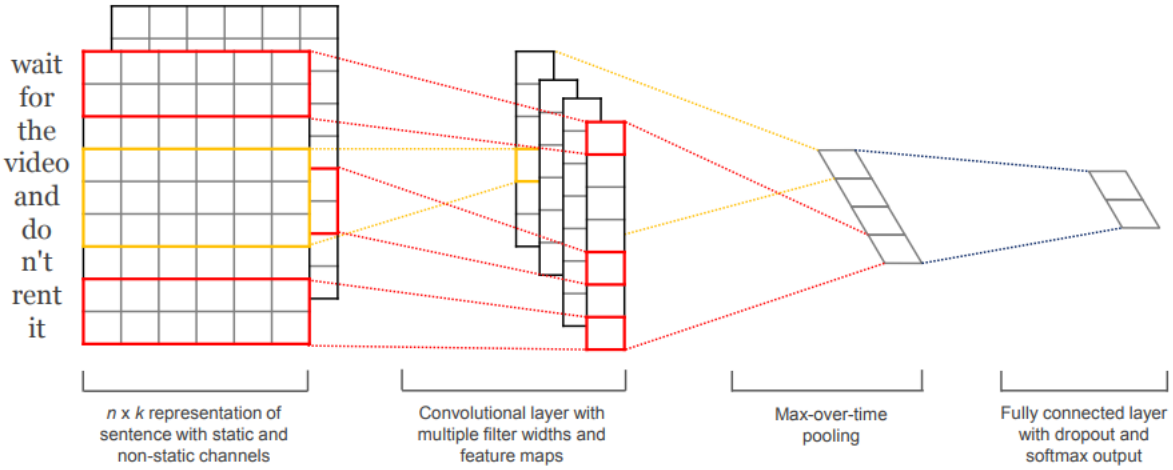

Figure 4.5 — Convolution Neural Networks architecture for text classification

Convolution

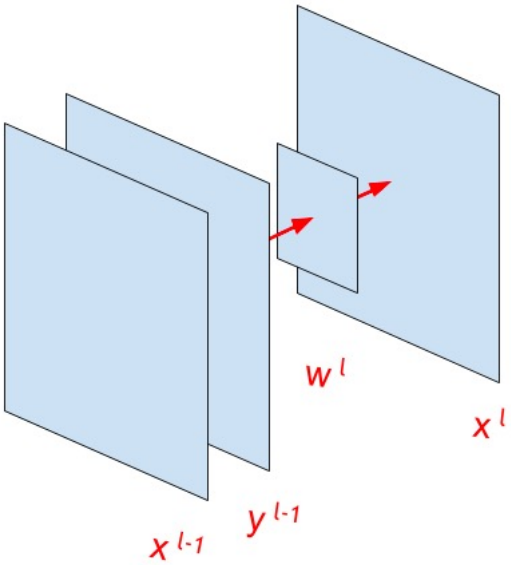

Figure 4.6 — Basic variables which are used in the convolution layer

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 4.6 [13].

The following equation 4.35 describes words above into mathematical way:

$$x_{ij}^{l} = \sum_{a=-\infty}^{+\infty} \sum_{b=-\infty}^{+\infty} w_{ab}^{l} \cdot y_{(i\cdot s-a)(j\cdot s-b)}^{l-1} + b^{l} \qquad \forall i \in (0,...,N) \ \forall j \in (0,...,M) \quad (4.35)$$

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** 4.36 - this activation function was used at Convolution layers. It has the following properties:

$$f_{ReLU} = max(0,x) \quad (4.36)$$

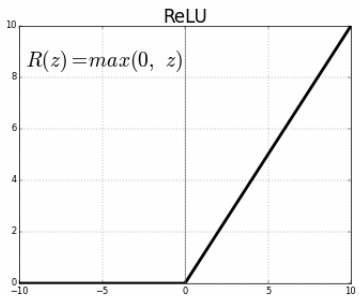

Figure 4.7 — ReLu activation function

2) **Softmax** 4.4 - 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.

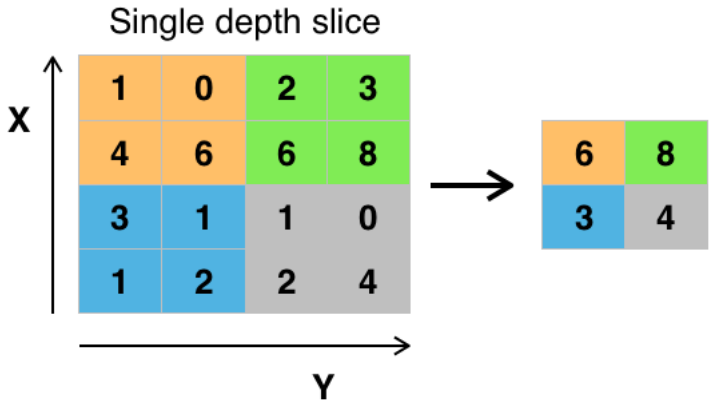

Figure 4.8 — Max pulling layer

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 4.1 describes this stage.

$$x_i^l = \sum_{k=0}^{m} w_{ki}^l y_k^{l-1} + b_i^l \qquad \forall i \in (0,...,n) \tag{4.37}$$

in matrix representation:

$$X^l = Y^{l-1} W^l + B_i^l \tag{4.38}$$

Loss function for the model is Cross Entropy 4.7 described above.

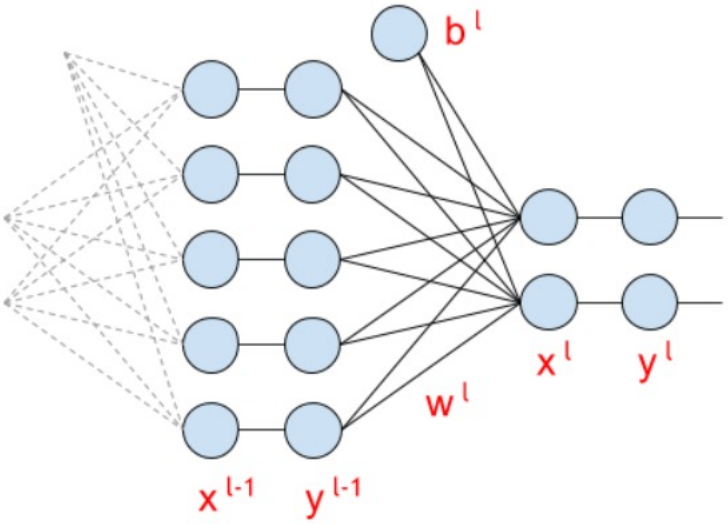

Figure 4.9 — Fully connected layer of CNN

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 4.10, 4.11, 4.12. Therefore, we have following equation 4.39:

$$\begin{aligned}\frac{\partial J}{\partial x_i^l} = \sum_{k=0}^{n} \frac{\partial J}{\partial y_k^l}\frac{\partial y_k^l}{\partial x_i^l} = \frac{\partial J}{\partial y_0^l}\frac{\partial y_0^l}{\partial x_i^l} + ... \\ + \frac{\partial J}{\partial y_1^l}\frac{\partial y_1^l}{\partial x_i^l} + ... + \frac{\partial J}{\partial y_n^l}\frac{\partial y_n^l}{\partial x_i^l} \qquad \forall i \in (0,...n)\end{aligned} \tag{4.39}$$

or

$$\begin{aligned}&\begin{bmatrix} \frac{\partial J}{\partial x_0^l} & \frac{\partial J}{\partial x_1^l} & \dots & \frac{\partial J}{\partial x_n^l} \end{bmatrix} = \\ &= \begin{bmatrix} (\frac{\partial J}{\partial y_0^l}\frac{\partial y_0^l}{\partial x_0^l} + \frac{\partial J}{\partial y_1^l}\frac{\partial y_1^l}{\partial x_0^l} + \dots + \frac{\partial J}{\partial y_n^l}\frac{\partial y_n^l}{\partial x_0^l}) & (\frac{\partial J}{\partial y_0^l}\frac{\partial y_0^l}{\partial x_1^l} + \frac{\partial J}{\partial y_1^l}\frac{\partial y_1^l}{\partial x_1^l} + \dots + \frac{\partial J}{\partial y_n^l}\frac{\partial y_n^l}{\partial x_1^l}) & \dots & (\frac{\partial J}{\partial y_0^l}\frac{\partial y_0^l}{\partial x_n^l} + \frac{\partial J}{\partial y_1^l}\frac{\partial y_1^l}{\partial x_n^l} + \dots + \frac{\partial J}{\partial y_n^l}\frac{\partial y_n^l}{\partial x_n^l}) \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial J}{\partial y_0^l} & \frac{\partial J}{\partial y_1^l} & \dots & \frac{\partial J}{\partial y_n^l} \end{bmatrix} \begin{bmatrix} \frac{\partial y_0^l}{\partial x_0^l} & \frac{\partial y_0^l}{\partial x_1^l} & \dots & \frac{\partial y_0^l}{\partial x_n^l} \\ \frac{\partial y_1^l}{\partial x_0^l} & \frac{\partial y_1^l}{\partial x_1^l} & \dots & \frac{\partial y_1^l}{\partial x_n^l} \\ \dots & \dots & \dots & \dots \\ \frac{\partial y_n^l}{\partial x_0^l} & \frac{\partial y_n^l}{\partial x_1^l} & \dots & \frac{\partial y_n^l}{\partial x_n^l} \end{bmatrix}\end{aligned} \tag{4.40}$$

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

$$\frac{\partial J}{\partial w^l} = \frac{\partial J}{\partial y^l}\frac{\partial y^l}{\partial x^l}\frac{\partial x^l}{\partial w^l} = \delta^l \cdot \frac{\partial x^l}{\partial w^l} = \left(y^{l-1}\right)^T \cdot \delta^l \tag{4.41}$$

and b^l

$$\frac{\partial J}{\partial b^l} = \delta^l \tag{4.42}$$

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

$$\frac{\partial J}{\partial y^{l-1}} = \delta^l \cdot \frac{\partial x^l}{\partial y^{l-1}} = \delta^l \cdot (w^l)^T = \delta^{l-1} \tag{4.43}$$

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.

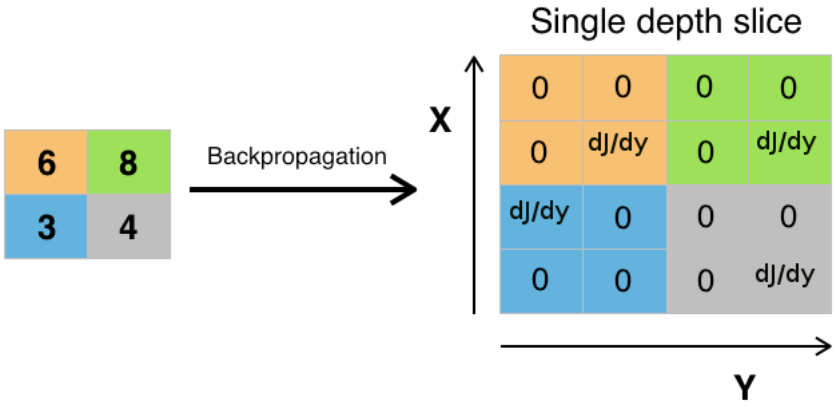

Figure 4.10 — Back propagation through max pulling layer

It is necessary to derive weights update for kernel 4.11.

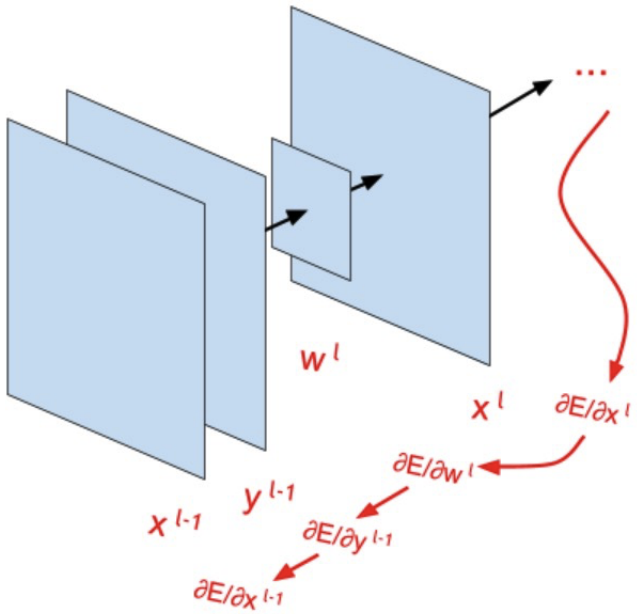

Figure 4.11 — Back propagation through convolution layer

$$\begin{aligned}
\frac{\partial J}{\partial w_{ab}^{l}} &= \sum_i \sum_j \frac{\partial J}{\partial y_{ij}^{l}} \frac{\partial y_{ij}^{l}}{\partial x_{ij}^{l}} \frac{\partial x_{ij}^{l}}{\partial w_{ab}^{l}} \\
&= {}^{(1)} \sum_i \sum_j \frac{\partial J}{\partial y_{ij}^{l}} \frac{\partial y_{ij}^{l}}{\partial x_{ij}^{l}} \cdot \frac{\partial \left(\sum_{a'=-\infty}^{+\infty} \sum_{b'=-\infty}^{+\infty} w_{a'b'}^{l} \cdot y_{(is-a')(js-b')}^{l-1} + b^{l} \right)}{\partial w_{ab}^{l}} \\
&= {}^{(2)} \sum_i \sum_j \frac{\partial J}{\partial y_{ij}^{l}} \frac{\partial y_{ij}^{l}}{\partial x_{ij}^{l}} \cdot y_{(is-a)(js-b)}^{l-1} \\
&\quad \forall a \in (-\infty, ..., +\infty) \;\; \forall b \in (-\infty, ..., +\infty)
\end{aligned} \tag{4.44}$$

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

Derivation of gradient for the bias element.

$$\frac{\partial J}{\partial b^{l}} = \sum_i \sum_j \frac{\partial J}{\partial y_{ij}^{l}} \frac{\partial y_{ij}^{l}}{\partial x_{ij}^{l}} \frac{\partial x_{ij}^{l}}{\partial b^{l}} = \sum_i \sum_j \frac{\partial J}{\partial y_{ij}^{l}} \frac{\partial y_{ij}^{l}}{\partial x_{ij}^{l}} \tag{4.45}$$

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

$$\frac{\partial J}{\partial y_{ij}^{l-1}} = \sum_{i'} \sum_{j'} \frac{\partial J}{\partial y_{i'j'}^{l}} \frac{\partial y_{i'j'}^{l}}{\partial x_{i'j'}^{l}} \cdot w_{(i-i's)(j-j's)}^{l} \tag{4.46}$$

Заключение

Основные результаты работы заключаются в следующем.

1. На основе анализа ...
2. Численные исследования показали, что ...
3. Математическое моделирование показало ...
4. Для выполнения поставленных задач был создан ...

И какая-нибудь заключающая фраза.

Список литературы

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. @ARTICLE2013arXiv1301.3781M, author = Mikolov, T. and Chen, K. and Corrado, G. and Dean, J. , title = "Efficient Estimation of Word Representations in Vector Space journal = ArXiv e-prints, archivePrefix = "arXiv eprint = 1301.3781, primaryClass = "cs.CL keywords = Computer Science - Computation and Language, year = 2013, month = jan, adsurl = http://adsabs.harvard.edu/abs/2013arXiv1301.3781M, adsnote = Provided by the SAO/NASA Astrophysics Data System

12. @ARTICLE2014arXiv1408.5882K, author = Kim, Y., title = "Convolutional Neural Networks for Sentence Classification journal = ArXiv e-prints, archivePrefix = "arXiv eprint = 1408.5882, primaryClass = "cs.CL keywords = Computer Science - Computation and Language, Computer Science - Neural and Evolutionary Computing, year = 2014, month = aug, adsurl = http://adsabs.harvard.edu/abs/2014arXiv1408.5882K, adsnote = Provided by the SAO/NASA Astrophysics Data System

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

Список рисунков

Список таблиц

Приложение А

Примеры вставки листингов программного кода

Для крупных листингов есть два способа. Первый красивый, но в нём могут быть проблемы с поддержкой кириллицы (у вас может встречаться в комментариях и печатаемых сообщениях), он представлен на листинге А.1.

Листинг А.1

Программа “Hello, world” на C++

```
#include <iostream>
using namespace std;

int main() //кириллица в комментариях при xelatex и lualatex и
   меет проблемы с пробелами
{
  cout << "Hello, world" << endl; //latin letters in
     commentaries
  system("pause");
  return 0;
}
```

Второй не такой красивый, но без ограничений (см. листинг А.2).

Листинг А.2

Программа “Hello, world” без подсветки

```
#include <iostream>
using namespace std;

int main() //кириллица в комментариях
{
    cout << "Привет, мир" << endl;
}
```

Можно использовать первый для вставки небольших фрагментов внутри текста, а второй для вставки полного кода в приложении, если таковое имеется.

Если нужно вставить совсем короткий пример кода (одна или две строки), то выделение линейками и нумерация может смотреться чересчур громоздко. В таких случаях можно использовать окружения `lstlisting` или `Verb` без `ListingEnv`. Приведём такой пример с указанием языка программирования, отличного от заданного по умолчанию:

```
fibs = 0 : 1 : zipWith (+) fibs (tail fibs)
```

Такое решение — со вставкой нумерованных листингов покрупнее и вставок без выделения для маленьких фрагментов — выбрано, например, в книге Эндрю Таненбаума и Тодда Остина по архитектуре

Наконец, для оформления идентификаторов внутри строк (функция `main` и тому подобное) используется `lstinline` или, самое простое, моноширинный текст (`\texttt`).

Пример A.3, иллюстрирующий подключение переопределённого языка. Может быть полезным, если подсветка кода работает криво. Без дополнительного окружения, с подписью и ссылкой, реализованной встроенным средством.

Листинг A.3

Пример листинга с подписью собственными средствами

```
## Caching the Inverse of a Matrix

## Matrix inversion is usually a costly computation and there
   may be some
## benefit to caching the inverse of a matrix rather than
   compute it repeatedly
## This is a pair of functions that cache the inverse of a
   matrix.

## makeCacheMatrix creates a special "matrix" object that can
   cache its inverse

makeCacheMatrix <- function(x = matrix()) {#кириллица в коммента
   риях при xelatex b lualatex имеет проблемы с пробелами
    i <- NULL
    set <- function(y) {
        x <<- y
        i <<- NULL
    }
    get <- function() x
    setSolved <- function(solve) i <<- solve
```

```
    getSolved <- function() i
    list(set = set, get = get,
    setSolved = setSolved,
    getSolved = getSolved)

}


## cacheSolve computes the inverse of the special "matrix"
   returned by
## makeCacheMatrix above. If the inverse has already been
   calculated (and the
## matrix has not changed), then the cachesolve should retrieve
   the inverse from
## the cache.

cacheSolve <- function(x, ...) {
    ## Return a matrix that is the inverse of 'x'
    i <- x$getSolved()
    if(!is.null(i)) {
        message("getting cached data")
        return(i)
    }
    data <- x$get()
    i <- solve(data, ...)
    x$setSolved(i)
    i
}
```

Листинг A.4 подгружается из внешнего файла. Приходится загружать без окружения дополнительного. Иначе по страницам не переносится.

Листинг A.4

Листинг из внешнего файла

```
# Analysis of data on Course Project at Getting and Cleaning
   data course of Data Science track at Coursera.

# Part 1. Merges the training and the test sets to create one
   data set.
# 3. Uses descriptive activity names to name the activities in
   the data set
```

```
# 4. Appropriately labels the data set with descriptive variable
    names.

if (!file.exists("UCI HAR Dataset")) {
    stop("You need 'UCI HAR Dataset' folder full of data")
}


library(plyr) # for mapvalues


#getting common data
features <- read.csv("UCI HAR Dataset/features.txt",sep=" ",
   header = FALSE,
                     colClasses = c("numeric","character"))
activity_labels <- read.csv("UCI HAR Dataset/activity_labels.txt
   ",sep="",
                            header = FALSE,colClasses = c("
                               numeric","character"))

#getting train set data
subject_train <- read.csv("UCI HAR Dataset/train/subject_train.
   txt",
                          header = FALSE,colClasses = "numeric",
                             col.names="Subject")
y_train <- read.csv("UCI HAR Dataset/train/y_train.txt", header
   = FALSE,
                    colClasses = "numeric")
x_train <- read.csv("UCI HAR Dataset/train/X_train.txt",sep="",
   header = FALSE,
                    colClasses = "numeric",col.names=features$V2
                       ,check.names = FALSE)

activity_train <- as.data.frame(mapvalues(y_train$V1, from =
   activity_labels$V1,
                                             to = activity_labels$
                                                V2))
names(activity_train) <- "Activity"


```

```
#getting test set data
subject_test <- read.csv("UCI HAR Dataset/test/subject_test.txt"
   ,
                         header = FALSE,colClasses = "numeric",
                            col.names="Subject")
y_test <- read.csv("UCI HAR Dataset/test/y_test.txt", header =
   FALSE,
                   colClasses = "numeric")
x_test <- read.csv("UCI HAR Dataset/test/X_test.txt",sep="",
   header = FALSE,
                   colClasses = "numeric",col.names=features$V2,
                      check.names = FALSE)

activity_test <- as.data.frame(mapvalues(y_test$V1, from =
   activity_labels$V1,
                                            to = activity_labels$V2
                                               ))
names(activity_test) <- "Activity"


# Forming full dataframe
data_train <- cbind(x_train,subject_train,activity_train)
data_test <- cbind(x_test,subject_test,activity_test)
data <- rbind(data_train, data_test)

# Cleaning memory
rm(features, activity_labels, subject_train, y_train, x_train,
   activity_train,
   subject_test, y_test, x_test, activity_test, data_train, data
      _test)


# Part 2. Extracts only the measurements on the mean and
   standard deviation for each measurement.

cols2match <- grep("(mean|std)",names(data))

# Excluded gravityMean, tBodyAccMean, tBodyAccJerkMean,
   tBodyGyroMean,
# tBodyGyroJerkMean, as these represent derivations of angle
   data, as
```

```
# opposed to the original feature vector.

# Subsetting data frame, also moving last columns to be first
Subsetted_data_frame <- data[ ,c(562, 563, cols2match)]

# Part 5. From the data set in step 4, creates a second,
   independent tidy data set
# with the average of each variable for each activity and each
   subject.

library(dplyr) # for %>% and summarise_each


tidydata <- Subsetted_data_frame %>% group_by(Subject,Activity)
   %>%
            summarise_each(funs(mean))

write.table(tidydata, "tidydata.txt", row.names=FALSE)
```

Приложение Б

Очень длинное название второго приложения, в котором продемонстрирована работа с длинными таблицами

Б.1 Подраздел приложения

Вот размещается длинная таблица:

Параметр	Умолч.	Тип	Описание
&INP			
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс

продолжение следует

(продолжение)			
Параметр	Умолч.	Тип	Описание
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			продолжение следует

(продолжение)			
Параметр	Умолч.	Тип	Описание
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
&SURFPAR			
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			продолжение следует

(продолжение)			
Параметр	Умолч.	Тип	Описание
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс

Б.2 Ещё один подраздел приложения

Нужно больше подразделов приложения!

Пример длинной таблицы с записью продолжения по ГОСТ 2.105

Table Б.2

Наименование таблицы средней длины

Параметр	Умолч.	Тип	Описание
&INP			
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно

Продолжение таблицы Б.2

Параметр	Умолч.	Тип	Описание
			экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно

Продолжение таблицы Б.2

Параметр	Умолч.	Тип	Описание
			экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно
			экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно
			экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно
			экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно
			экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно

Продолжение таблицы Б.2

Параметр	Умолч.	Тип	Описание
			экватора
mars	0	int	1: инициализация модели для планеты Марс
&SURFPAR			
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно
			экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно
			экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно
			экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно
			экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума

Продолжение таблицы Б.2

Параметр	Умолч.	Тип	Описание
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс
kick	1	int	0: инициализация без шума ($p_s = const$)
			1: генерация белого шума
			2: генерация белого шума симметрично относительно экватора
mars	0	int	1: инициализация модели для планеты Марс

Б.3 Очередной подраздел приложения

Нужно больше подразделов приложения!

Б.4 И ещё один подраздел приложения

Нужно больше подразделов приложения!