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:

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

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]

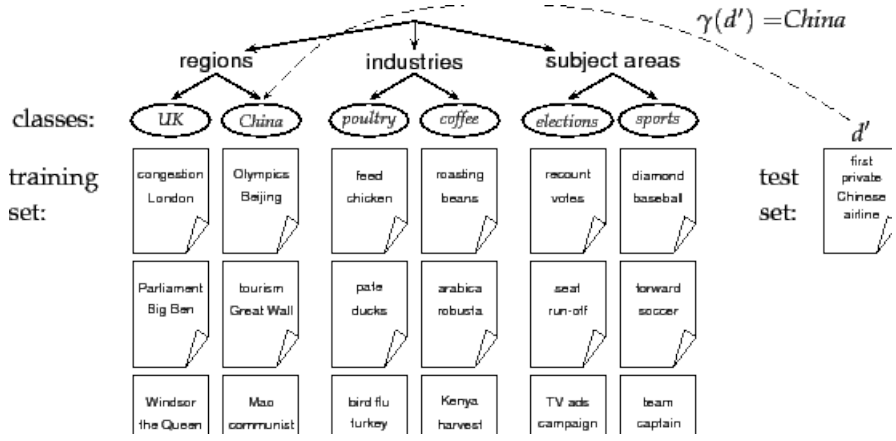

Figure 1.1 — Classes, training set, and test set in text classification.

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

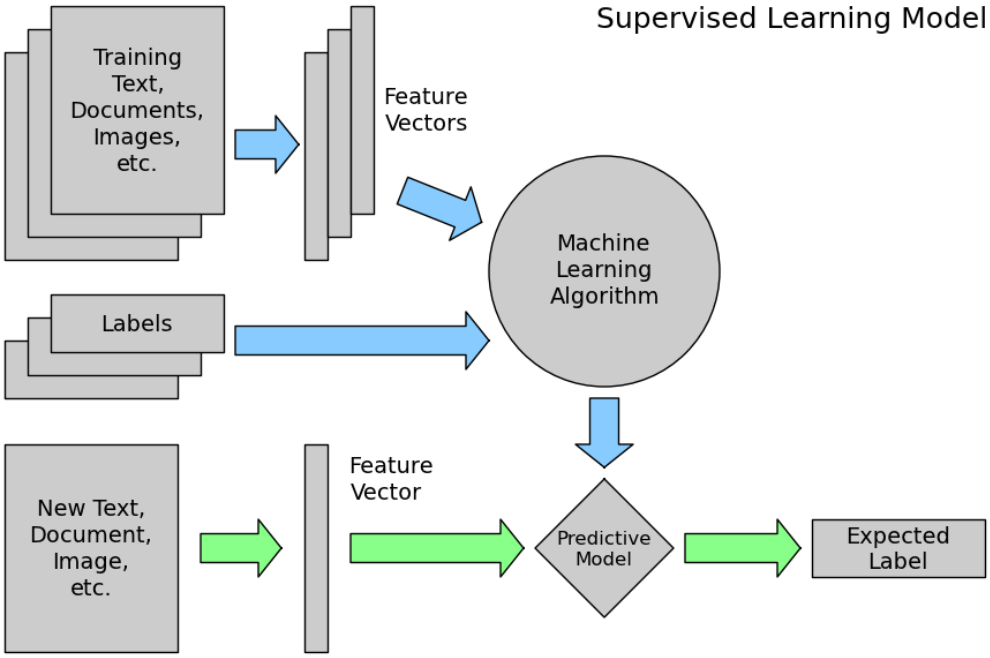

Figure 1.2 — Supervised learning work flow.

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 hyper parameter 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.

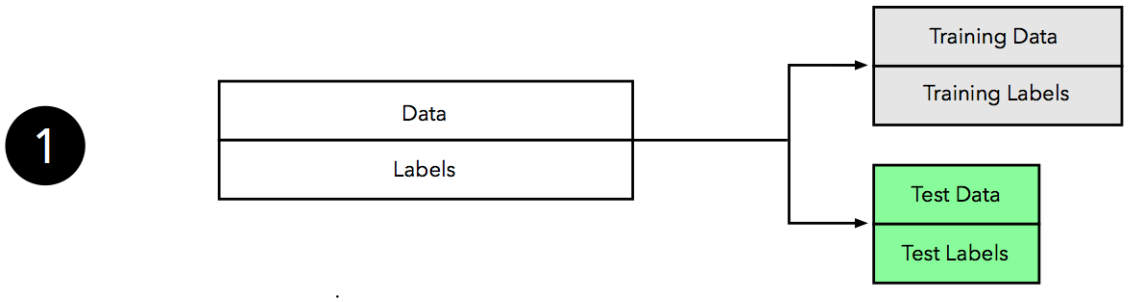

Figure 1.4 — .

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.

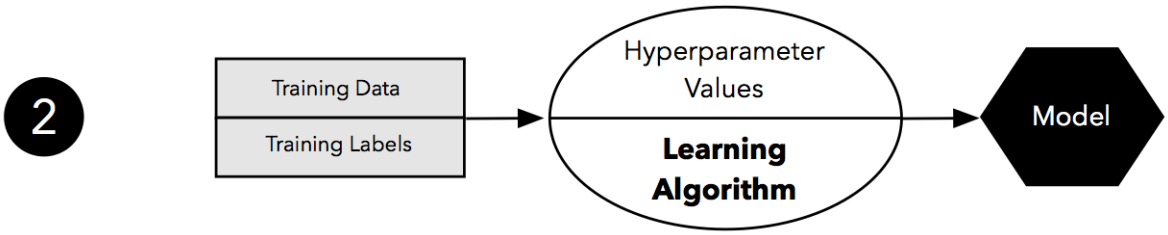

Figure 1.5 — .

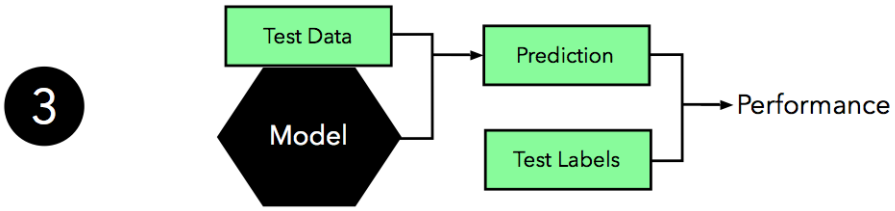

Figure 1.6 — .

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.

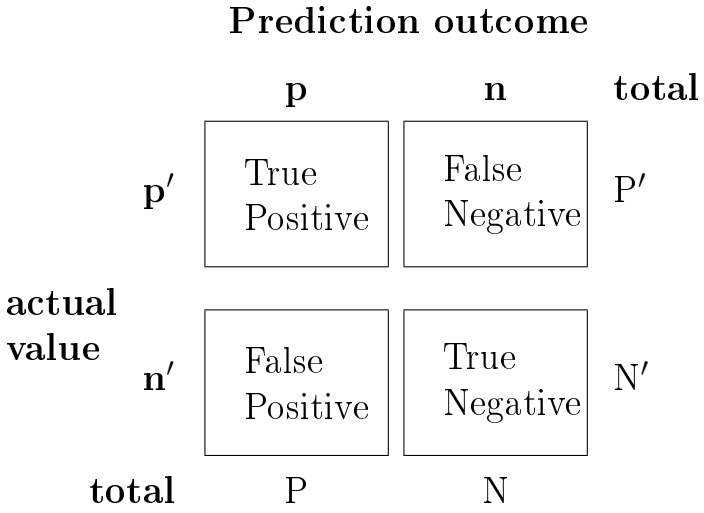

Figure 1.7 — Confusion matrix

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.

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:

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

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']

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:

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

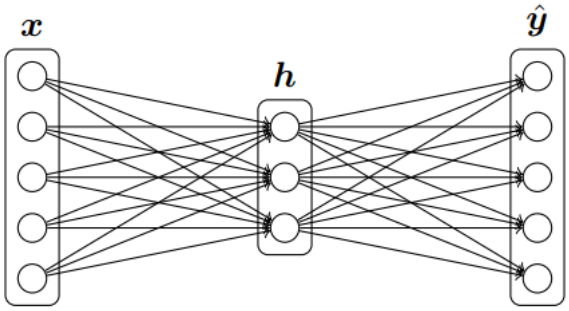

Figure 2.1 — Neural Network

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:

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

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

where $\mathrm{W}_\mathbf{i}$ and $\mathbf{b}_\mathbf{i}$ ($i \in \{1,2\}$) 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:

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

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

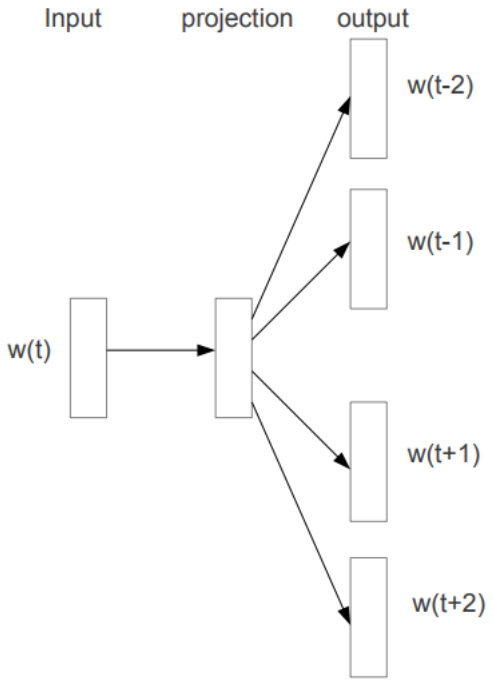

Figure 2.2 — The Skip-gram model architecture.

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

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

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

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

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

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

Now, using the chain rule, we can calculate,

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

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

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

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

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

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

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

The derivative of the loss has two terms, $\boldsymbol{w}_{c+j}$ and $\boldsymbol{v}_c$, which yields the following [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 2.3:

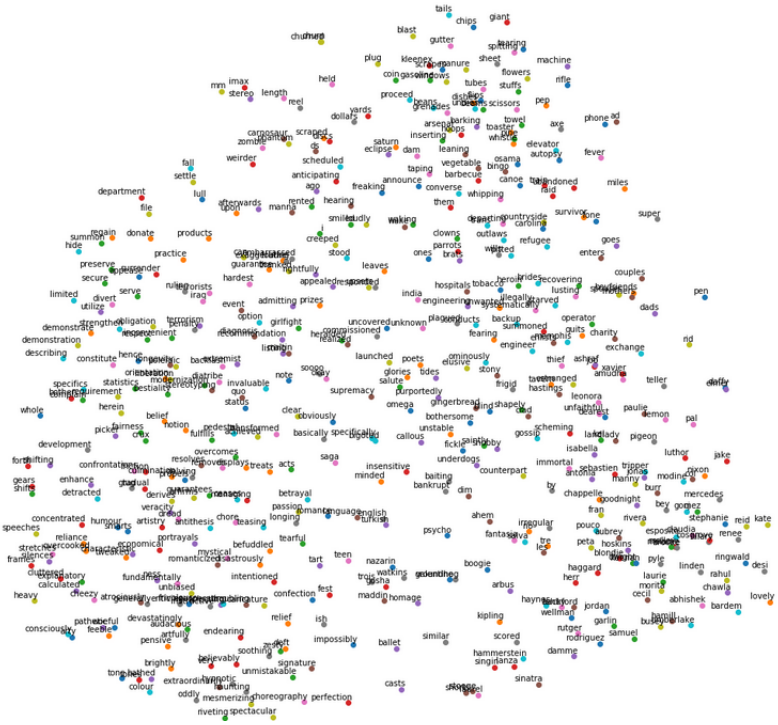

Figure 2.3 — Words representation

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]:

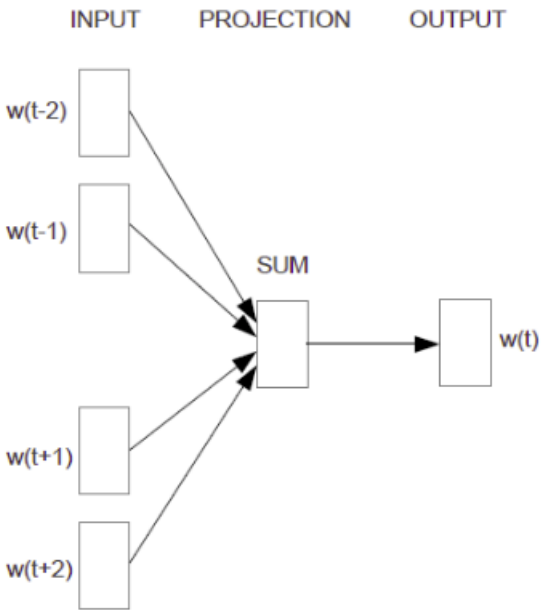

Figure 2.4 — The CBOW model architecture.

2.2 Deep learning algorithms for text classification

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 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]

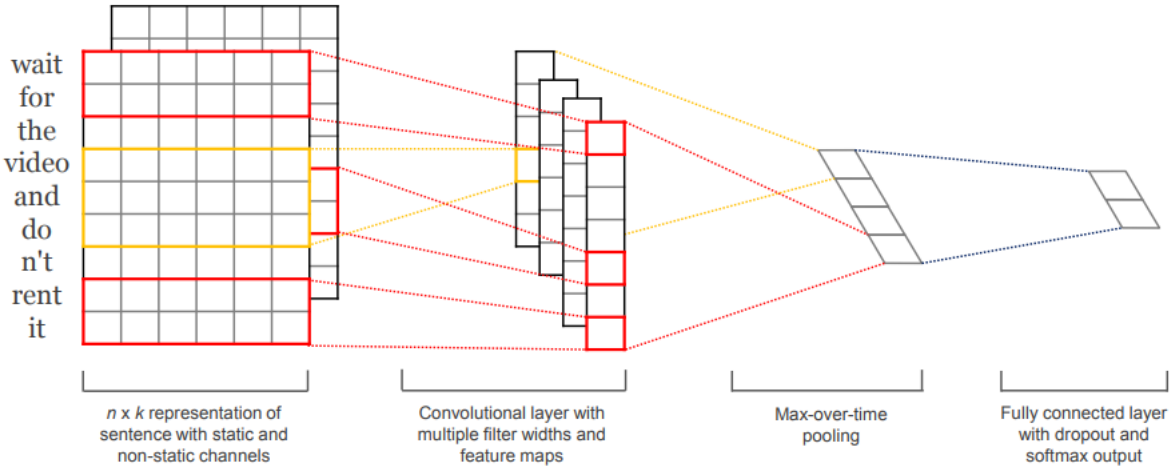

Figure 2.5 — Convolution Neural Networks architecture for text classification

Convolution

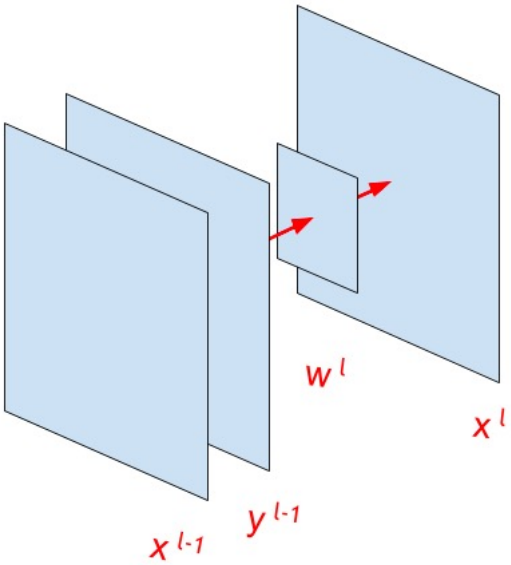

Figure 2.6 — Basic variables which are used in the convolution layer

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:

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

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

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

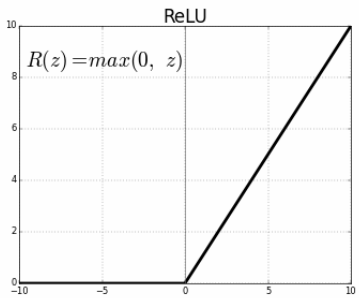

Figure 2.7 — ReLu activation function

2) **Softmax** 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.

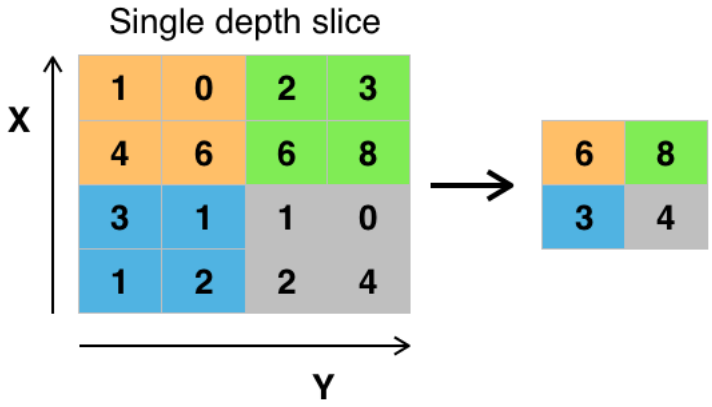

Figure 2.8 — Max pulling layer

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.

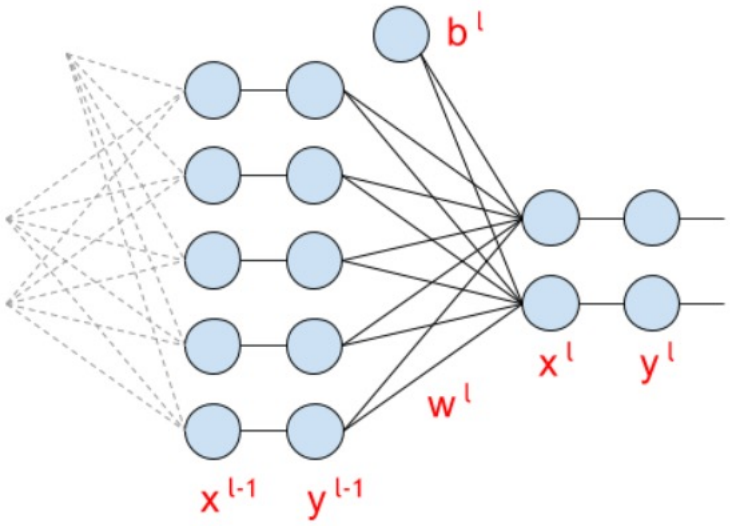

Figure 2.9 — Fully connected layer of CNN

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:

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

or

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

$$= \begin{bmatrix} \frac{\partial J}{\partial y_0^l} & \frac{\partial J}{\partial y_1^l} & \dots & \frac{\partial J}{\partial y_n^l} \end{bmatrix} \begin{bmatrix} \frac{\partial y_0^l}{\partial x_0^l} & \frac{\partial y_0^l}{\partial x_1^l} & \dots & \frac{\partial y_0^l}{\partial x_n^l} \\ \frac{\partial y_1^l}{\partial x_0^l} & \frac{\partial y_1^l}{\partial x_1^l} & \dots & \frac{\partial y_1^l}{\partial x_n^l} \\ \dots & \dots & \dots & \dots \\ \frac{\partial y_n^l}{\partial x_0^l} & \frac{\partial y_n^l}{\partial x_1^l} & \dots & \frac{\partial y_n^l}{\partial x_n^l} \end{bmatrix} \tag{2.40}$$

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.

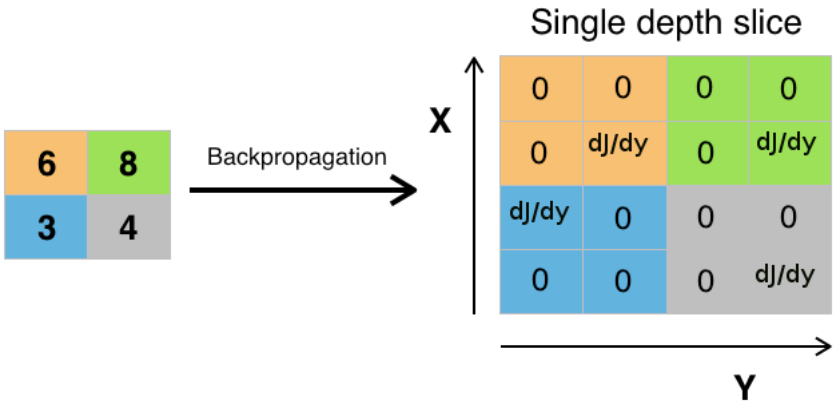

Figure 2.10 — Back propagation through max pulling layer

It is necessary to derive weights update for kernel 2.11.

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

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

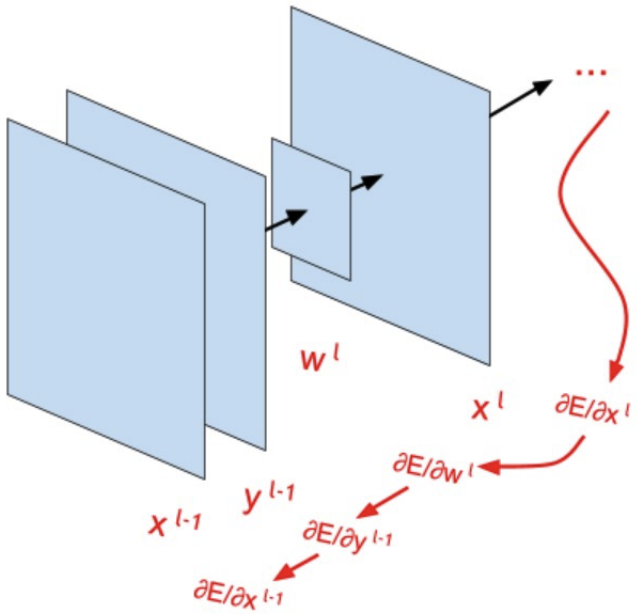

Figure 2.11 — Back propagation through convolution layer

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

In this section, you'll implement your first recurrent neural network (**RNN**) for building a language model.

Language modeling is a central task in `NLP`, and language models can be found at the heart of speech recognition, machine translation, and many other systems. Give $\boldsymbol{x}_1, ..., \boldsymbol{x}_t$, a language model prdicts the following word $\boldsymbol{x}_{t+1}$ by modeling:

$$P(\boldsymbol{x}_{t+1} = v_j | \boldsymbol{x}_1, ..., \boldsymbol{x}_t) \tag{2.47}$$

where v_j is a word in the vocabulary.

Your job is to implement a recurrent neural network language model, which use feedback informatoin in the hidden layer to model the 'history' $x_t, x_{t-1}, ..., x_1$. Formally, the model

$$\boldsymbol{e}^{(t)} = \boldsymbol{x}^{(t)}\boldsymbol{L} \tag{2.48}$$

$$\boldsymbol{h}^{(t)} = \sigma(\boldsymbol{h}^{(t-1)}\boldsymbol{H} + e^{(t)}\boldsymbol{I} + \boldsymbol{b}_1) \tag{2.49}$$

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}(\boldsymbol{h}^{(t)}\boldsymbol{U} + \boldsymbol{b}_2) \tag{2.50}$$

$$\bar{P}(\boldsymbol{x}^{(t)} = \boldsymbol{v}_j | \boldsymbol{x}_{(t)}, ..., \boldsymbol{x}_{(1)}) = \hat{\boldsymbol{y}}_j^{(t)} \tag{2.51}$$

where $\boldsymbol{h}^0 = \boldsymbol{h}_0 \in \mathbb{R}^{D_h}$ is some initialization vector for the hidden layer and $\boldsymbol{x}^{(t)}\boldsymbol{L}$ is the product of $\boldsymbol{L}$ with the one-hot row-vector $\boldsymbol{x}^{(t)}$ representing index of the current word. The parameters are:

$$\boldsymbol{L} \in \mathbb{R}^{|V| \times d} \tag{2.52}$$

where $\boldsymbol{L}$ is the embedding matrix, $\boldsymbol{I}$ input word representation matrix, $\boldsymbol{H}$ the hidden transformation matrix, and $\boldsymbol{U}$ is the output word representation matrix, $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are biases (d is the embedding dimension, |V| is the vocabulary size, and D_h is the hidden layer dimension).

The output vector $\hat{\boldsymbol{y}}^{(t)} \in \mathbb{R}^{|\text{V}|}$ is a probability distribution over the vocabulary, and we optimize the (unregularized) cross-entropy loss:

$$J(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{t=1}^{|\text{V}|} \boldsymbol{y}_i{}^{(t)} \log(\hat{\boldsymbol{y}}_i^{(t)} \tag{2.53}$$

where $\boldsymbol{y}^{(t)}$ is the one-hot vector corresponding to the target word (which here is equal to x_{t+1}). As in question , this is a point-wise loss, and we sum (or average) the cross-entropy loss across all examples in a sequence, across all sequences

$$2^J = 2^{-\sum_{j=1}^{|V|} y_j \log \hat{y}_j} \tag{2.54}$$
$$= \frac{1}{\prod_{j=1}^{|V|} 2^{y_j \log \hat{y}_j}} \tag{2.55}$$
$$= \frac{1}{\prod_{j=1}^{|V|} 2^{\log \hat{y}_j^{y_j}}} \tag{2.56}$$
$$= \frac{1}{\prod_{j=1}^{|V|} \hat{y}_j^{y_j}} \tag{2.57}$$
$$= \frac{1}{\sum_{j=1}^{|V|} y_j \cdot \hat{y}_j} \tag{2.58}$$
$$= PP\left(\hat{y}, y\right) \tag{2.59}$$

In the last step we used the fact that y_j is a one-hot vector and thus the sum and the product are identical. Since taking something to exponent of 2 is a monotonic transformation, minimizing the cross entropy also minimizes the perplexity.

Summed over all examples we can see that $2^{\frac{1}{|N|}\sum_t J^{(t)}} = \sqrt[N]{\prod_t PP\left(\hat{y}, y\right)}$. Thus, the arithmetic mean of the cross entropy is the geometric mean of the perplexity.

In the case of completely random predictions we would predict $\frac{1}{|V|}$ for each word. This would yield a cross entropy of $-\log_2 \frac{1}{|V|}$:

$$-\log_2 \frac{1}{2000} = 10.9657842847 \tag{2.60}$$
$$-\log_2 \frac{1}{10000} = 13.2877123795 \tag{2.61}$$

Let:

$$z_1^{(t)} = Hh^{(t-1)} + Lx^{(t)} \tag{2.62}$$
$$z_2^{(t)} = Uh^{(t)} \tag{2.63}$$

We also define the δ terms for a given time step t. Note that at time step t there will be a delta term with respect to all the previous time steps. For example, $\delta_1^{(t)}$ is the delta term at time t with respect to time step 1. Let:

$$\delta_x^{(t)} = \frac{\partial J^{(t)}}{\partial z_1^{(x)}} \tag{2.64}$$

Let's now derive the gradients for the model parameters:

$$\begin{aligned} \jmath dU &= \jmath dz_2^{(t)} \frac{\partial z_2^{(t)}}{\partial U} && (2.65)\\ &= (\hat{y}^{(t)} - y^{(t)}) \frac{\partial z_2^{(t)}}{\partial U} && (2.66)\\ &= (\hat{y}^{(t)} - y^{(t)}) h^{(t)T} && (2.67)\\ & && (2.68) \end{aligned}$$

By the backpropagation rule, let $\delta_t^{(t)} = ((U^T(\hat{y}^{(t)} - y^{(t)})) \circ \sigma'(z_1^{(t)}))$.

$$\begin{aligned} \frac{\partial J^{(t)}}{\partial L} &= \delta_t^{(t)} \left(x^{(t)}\right)^T && (2.69)\\ \frac{\partial J^{(t)}}{\partial H}\Big|_{(t)} &= \delta_t^{(t)} \left(h^{(t-1)}\right)^T && (2.70)\\ \frac{\partial J^{(t)}}{\partial h^{(t-1)}} &= H^T \delta_t^{(t)} && (2.71) \end{aligned}$$

Note that since $x^{(t)}$ is a one-hot vector, $\frac{\partial J^{(t)}}{\partial L_{x^{(t)}}}$ is simply $\delta_t^{(t)}$.
In order to calculate $\delta_{(t-1)}^{(t)}$ we can backpropagate as follows:

$$\delta_{(t-1)}^{(t)} = (H^T \delta_t^{(t)}) \circ \sigma'(z_1^{(t-1)}) \tag{2.72}$$

Calculating the gradients is easy now. We note that the gradients at time $(t-1)$ only depend on the above delta term and the previous hidden layer value. By backpropagation:

$$\frac{\partial J^{(t)}}{\partial L_{x^{(t-1)}}} = \delta^{(t)}_{(t-1)} \tag{2.73}$$

$$\frac{\partial J^{(t)}}{\partial H}\Big|_{(t-1)} = \delta^{(t)}_{(t-1)} \left(h^{(t-2)}\right)^T \tag{2.74}$$

Let's assume the cost of matrix multiplication for AB with $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$ is $O(mnp)$. We also assume that our vocabulary $|V|$ is much larger than the dimensionality of our hidden layers D_h.

Then, for forward propagation we get:

$$O(D_h^2 + |V|D_h) = O(|V|D_h) \tag{2.75}$$

Backpropagating one step in time:

$$O(2|V|D_h + D_h^2) = O(|V|D_h) \tag{2.76}$$

Backpropagating τ steps in time, just adds one more calculation for $\delta^{(t)}_{(t-1)}$ and a gradient of complexity D_h^2:

$$O(|V|D_h + \tau D_h^2) == O(|V|D_h) \tag{2.77}$$

$|V|D_h$ is the slowest operation in the above. Thus, the computation time is heavily depended on the size of the vocabulary.

We make use of the fact that the derivative of the cross entropy loss $J(\theta)$ for a softmax function is given by $\frac{\partial J^{(t)}}{\partial \theta} = (\hat{y} - y)$. We derived this fact in problem set 1.

$$\frac{\partial J^{(t)}}{\partial U} = \frac{\partial J^{(t)}}{\partial z3}\frac{\partial z_3}{\partial U} \tag{2.78}$$
$$= (a_3 - y)\frac{\partial z_3}{\partial U} \tag{2.79}$$
$$= (a_3 - y)a_2^T = (\hat{y} - y)a_2^T \tag{2.80}$$
$$= \delta_3 a_2^T \tag{2.81}$$

We confirm that $\frac{\partial J^{(t)}}{\partial U} \in \mathbb{R}^{5x100}$.

$$\frac{\partial J^{(t)}}{\partial b_2} = \frac{\partial J^{(t)}}{\partial z3}\frac{\partial z_3}{\partial b_2} \tag{2.82}$$
$$= (a_3 - y)\frac{\partial z_3}{\partial b_2} \tag{2.83}$$
$$= (a_3 - y) = (\hat{y} - y) \tag{2.84}$$
$$= \delta_3 \tag{2.85}$$

We confirm that $\frac{\partial J^{(t)}}{\partial b_2} \in \mathbb{R}^5$.

Let's remember that $\tanh'(x) = 1 - \tanh^2(x)$. From the backpropagation formula we know that:

$$\frac{\partial J^{(t)}}{\partial W} = \delta_2 a_1^T \tag{2.86}$$
$$= ((1 - \tanh^2(z_2)) \circ (U^T\delta_3)a_1^T \tag{2.87}$$
$$= ((1 - \tanh^2(z_2)) \circ (U^T(\hat{y} - y))a_1^T \tag{2.88}$$

We confirm that $\frac{\partial J^{(t)}}{\partial W} \in \mathbb{R}^{100x150}$.

$$\frac{\partial J^{(t)}}{\partial b_1} = \delta_2 \tag{2.89}$$
$$= ((1 - \tanh^2(z_2)) \circ (U^T\delta_3) \tag{2.90}$$
$$= ((1 - \tanh^2(z_2)) \circ (U^T(\hat{y} - y)) \tag{2.91}$$

We confirm that $\frac{\partial J^{(t)}}{\partial b_1} \in \mathbb{R}^{100}$

$$\frac{\partial J^{(t)}}{\partial L_1} = W^T \delta_2 \tag{2.92}$$

$$= W^T((1 - \tanh^2(z_2)) \circ (U^T(\hat{y} - y)) \tag{2.93}$$

We confirm that $\frac{\partial J^{(t)}}{\partial L_1} \in \mathbb{R}^{150}$.
Since $\frac{\partial J_{reg}}{\partial b_1} = \frac{\partial J_{reg}}{\partial b_2} = \frac{\partial J_{reg}}{\partial L_i} = 0$ the gradients for b_1, b_2 and L_i stay the same.

$$\frac{\partial J_{reg}}{\partial W} = \lambda W \tag{2.94}$$

$$\frac{\partial J_{reg}}{\partial U} = \lambda U \tag{2.95}$$

We simply add the above to $\frac{\partial J^{(t)}}{\partial W}$ and $\frac{\partial J^{(t)}}{\partial I}$ from section **??**.

Table 2.2 shows the top-10 word lists for the center word on 5 hidden layer neurons. We can see that some of the neurons try to encode high-level concepts such as countries, institutions or adverbs.

Neuron 2	malaysia, germany, berlin, britain, ocean, norway, ireland, pakistan, iraq, korea
Neuron 7	have, which, worker, help, measures, ;, what, sector, that, how
Neuron 24	suddenly, permanently, immediately, foster, there, forth, she, temporarily, behalf, thereby
Neuron 35	educational, salaries, membership, health, ministry, legacy, approval, assistance, credit, instruction
Neuron 67	admission, achievement, additional, initial, extra, arithmetic, example, answer, endorsement, instruction

Table 2.2

Top-10 word lists for the center word on 5 hidden layer neurons.

Table 2.3 shows the top-10 word lists for the center word on the model output. We see that the neurons learn general concepts related to locations, organizations and people. For example, the concept of people names for the PER label.

Table 2.4 shows the top-10 word lists for the first word on the model output. The models seems to learn concepts that commonly prefix the classes. For example directional modifiers for LOC, first names for PER and company prefixes for ORG.

LOC	lanka, england, korea, berlin, ireland, egypt, britain, iraq, pakistan, italy
MISC	german, turkish, danish, korean, brazilian, olympic, israeli, italian, belgian, english
ORG	zenith, inc., liverpool, cdu, microsoft, reuters, ajax, commons, inc, corp
PER	wept, jim, scott, stephen, martin, herb, trembling, sarah, roger, pat

Table 2.3

Top-10 word lists for the center word on the model output.

LOC	beneath, near, st., mount, lake, west, native, san, santa, cape
MISC	county, golf, oak, serie, exotic, fair, grand, lakes, tour, windows
ORG	la, cdu, f.c., securities, cooperation, enterprise, &, v, moody, dream
PER	stephen, matt, d., pat, m., john, a., michael, roger, peter

Table 2.4

Top-10 word lists for the first word on the model output.

Vanishing Gradients

Now that we gained intuition about the nature of the vanishing gradients problem and how it manifests itself in deep neural networks, let us focus on a simple and practical heuristic to solve these problems. To solve the problem of exploding gradients, Thomas Mikolov first introduced a simple heuristic solution that clips gradients to a small number whenever they explode. It shows the decision surface of a small recurrent neural network with respect to its W matrix and its bias terms, b. The model consists of a single unit of recurrent neural network running through a small number of timesteps; the solid arrows illustrate the training progress on each gradient descent step. When the gradient descent model hits the high error wall in the objective function, the gradient is pushed off to a far-away location on the decision surface. The clipping model produces the dashed line where it instead pulls back the error gradient to somewhere close to the original gradient landscape. To solve the problem of vanishing gradients, we introduce two techniques. The first technique is that instead of initializing randomly, start off from an identify matrix initialization.The second technique is to use the Rectified Linear Units (ReLU) in-

stead of the sigmoid function. The derivative for the ReLU is either 0 or 1. This way, gradients would flow through the neurons whose derivative is 1 without getting attenuated while propagating back through time-steps.

Long-Short-Term-Memories

Long-Short-Term-Memories are another type of complex activation unit that differ a little from GRUs. The motivation for using these is similar to those for GRUs however the architecture of such units does differ. Let us first take a look at the mathematical formulation of LSTM units before diving into the intuition behind this design:

$$
\begin{aligned}
i_t &= \sigma(W^{(i)}x_t + U^{(i)}h_{t-1}) && \text{(Input gate)} \\
f_t &= \sigma(W^{(f)}x_t + U^{(f)}h_{t-1}) && \text{(Forget gate)} \\
o_t &= \sigma(W^{(o)}x_t + U^{(o)}h_{t-1}) && \text{(Output/Exposure gate)} \\
\tilde{c}_t &= \tanh(W^{(c)}x_t + U^{(c)}h_{t-1}) && \text{(New memory cell)} \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t && \text{(Final memory cell)} \\
h_t &= o_t \circ \tanh(c_t)
\end{aligned}
$$

New memory generation:

1. This stage is analogous to the new memory generation stage we saw in GRUs. We essentially use the input c_t which includes aspects of the new word $x_(t)$.

2. Input Gate: We see that the new memory generation stage doesn't check if the new word is even important before generating the new memory – this is exactly the input gate's function. The input gate uses the input word and the past hidden state to determine whether or not the input is worth preserving and thus is used to gate the new memory. It thus produces it as an indicator of this information.

3. Forget Gate: This gate is similar to the input gate except that it does not make a determination of usefulness of the input word – instead it makes an assessment on whether the past memory cell is useful for the computation of the current memory cell. Thus, the forget gate looks at the input word and the past hidden state and produces f_t.

4. Final memory generation: This stage first takes the advice of the it takes the advice of the input gate it and accordingly gates the new memory c_t. It then sums these two results to produce the final memory c_t . 5. Output/Exposure Gate: This

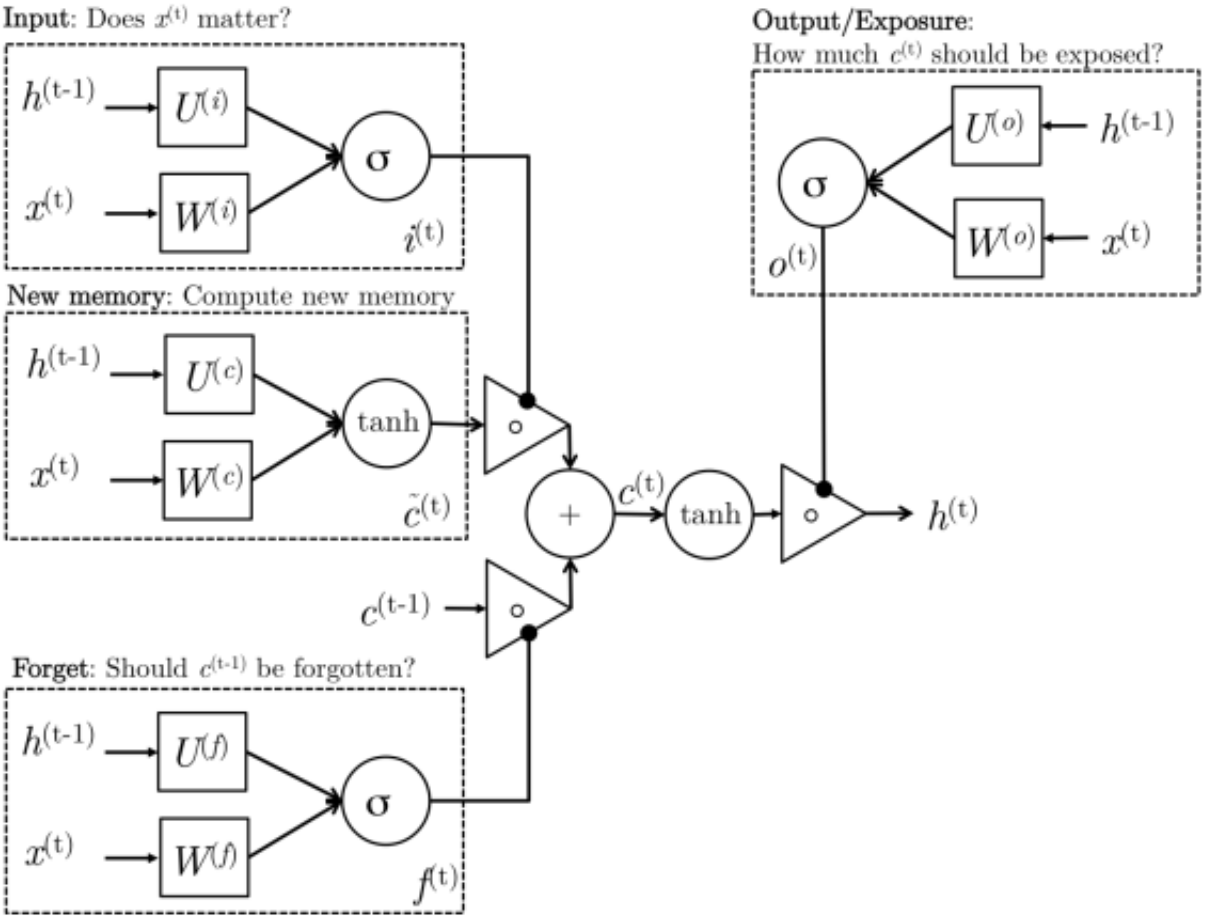

Figure 2.12 — The detailed internals of a LSTM

is a gate that does not explicitly exist in GRUs. It's purpose is to separate the final memory from the hidden state. The final memory ct contains a lot of information that is not necessarily required to be saved in the hidden state. Hidden states are used in every single gate of an LSTM and thus, this gate makes the assessment regarding what parts of the memory ct needs to be exposed/present in the hidden state h_t. The signal it produces to indicate this is ot and this is used to gate the point-wise tanh of the memory.

2.2.3 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

Table 3.1

My caption

lvl1	lvl2	titles	descriptions
6	29	Clean Toyota Camry 2008 Silver	Fairly used Toyota 08 Camry with no
5	25	Look Unique	Nice, quality, adorable,unique dress av
6	29	Mercedes Benz Ml 430 2001 Silver	mercedes benz ml430 , 2001 model in
5	25	Versace Shirt Dress	Adorable versace shirt dress, whatsapp
5	25	Addidas Jumpsuit	Nice quality addidas jumpsuit availab

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

Table 3.2

Training set general information

Number of variables	4
Numeric variables	2
Categorical variables	2
Number of observations	1000000
Total Missing (%)	2.4%
Total size in memory	57.7 MiB
Average record size in memory	48.0 B

The total number of first level categories is 16.

The total number of second level categories is 182.

3.3 Data preparation

Table 3.3

Information about first level categories

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

Table 3.4

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 3.5

Information about categorical features

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

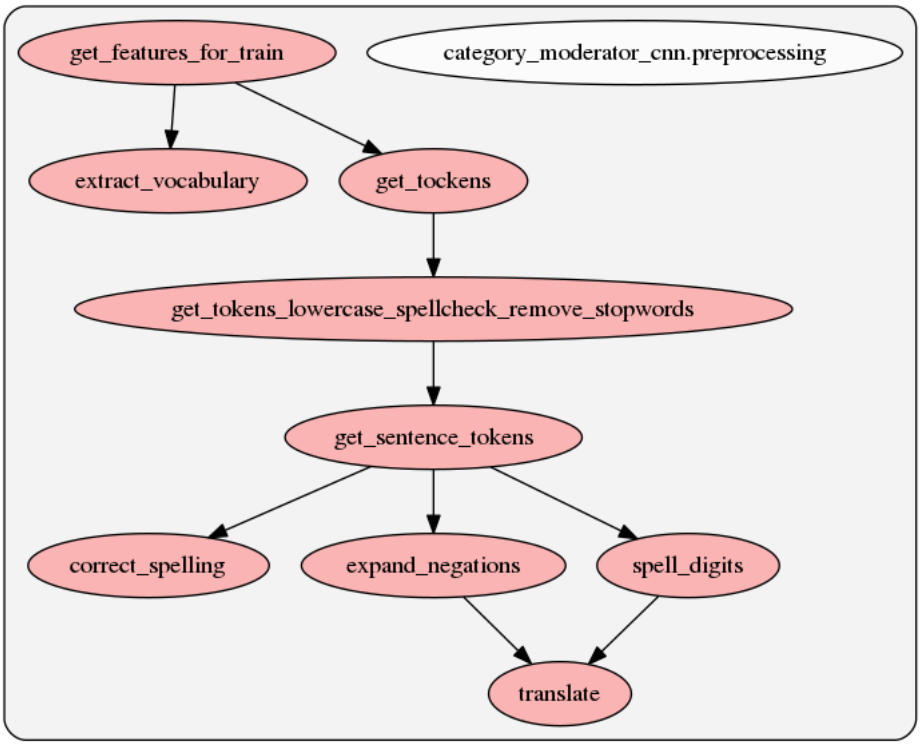

Figure 3.1 — The detailed internals of a LSTM

Table 3.6

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_remove_stopwords	wrapper function 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 → one . . .
translate	function which makes replacement of words

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

List of figures

List of tables