ТЕХНИЧЕСКИЕ НАУКИ
И ТЕХНОЛОГИИ

ИНФОРМАЦИОННЫЕ СИСТЕМЫ
INFORMATION SYSTEMS
АҚПАРАТТЫҚ ЖҮЙЕЛЕР

DOI 10.51885/1561-4212_2023_4_343
IRSTI 20.23.17

D.V. Sukharnikov[1], A.T. Bekishev[1], K.Y. Nursakitov[1], S.K. Kumargazhanova[1], A.M. Urkumbaeva[1], L.K. Bobrov[2]
[1]D.Serikbayev East Kazakhstan Technical University, Ust-Kamenogorsk, Kazakhstan
E-mail: afterallspace@gmail.com
E-mail: a.nomad.b@mail.ru
*E-mail: Nursakitov@bk.ru**
E-mail: skumargazhanova@gmail.com
E-mail: urkumbaeva@mail.ru
[2]Novosibirsk state university of economics and management, Novosibirsk, Russian Federation
E-mail: l.k.bobrov@edu.nsuem.ru

APPLICATION OF NEURAL NETWORKS FOR CYBERBULLYING DETECTION

КИБЕРБУЛЛИНГТІ АНЫҚТАЙТЫН НЕЙРОНДЫҚ ЖЕЛІЛЕРДІ ҚОЛДАНУ

ПРИМЕНЕНИЕ НЕЙРОННЫХ СЕТЕЙ ДЛЯ ОПРЕДЕЛЕНИЯ КИБЕРБУЛЛИНГА

Abstract. *Cyberbullying is a global problem affecting many people, especially children and teenagers. This phenomenon is common in many countries, causing psychological, emotional and social damage to its victims. Technologies such as social media and mobile apps exacerbate the problem by providing a platform for anonymous attacks. To resist cyberbullying, many organizations and governments are taking action, from zero-tolerance policies to legislative initiatives. Awareness raising and education also play a key role. At the forefront of the fight against cyberbullying is the development of neural networks that can automatically detect and block toxic content online. This article describes the main stages of training a neural network and creating an information technology based on it to protect people from such a phenomenon as cyberbullying.*

Key words: *natural language processing, sentiment analysis, machine learning, cyberbullying, neural network, TensorFlow.*

Аңдатпа. *Кибербуллинг көптеген адамдарға, әсіресе балалар мен жасөспірімдерге әсер ететін жаһандық мәселе. Бұл құбылыс көптеген елдерде жиі кездеседі, зардап шеккендерге психологиялық, эмоционалдық және әлеуметтік зиян келтіреді. Әлеуметтік медиа және мобильді қосымшалар сияқты технологиялар анонимді шабуылдарға арналған платформаны қамтамасыз ету арқылы мәселені ушықтырады. Кибербуллингпен күресу үшін көптеген ұйымдар мен үкіметтер нөлдік төзімділік саясатынан бастап заңнамалық бастамаларға дейін әрекет етуде. Ақпаратты арттыру мен білім беру де басты рөл атқарады. Кибербуллингпен күрестің алдыңғы қатарында улы мазмұнды желіде автоматты түрде анықтап, блоктай алатын нейрондық желілерді дамыту. Бұл мақалада нейрондық желіні оқытудың және оның негізінде кибербуллинг сияқты құбылыспен күресу үшін ақпараттық технологияны құрудың негізгі кезеңдері сипатталған.*

Түйін сөздер: *табиғи тілді өңдеу, сентимент талдау, машиналық оқыту, табиғи тілді өңдеу,, кибербуллинг, нейрондық желі, TensorFlow.*

Аннотация. *Кибербуллинг – глобальная проблема, затрагивающая многих, особенно детей и подростков. Это явление распространено во многих странах, причиняя жертвам психологический, эмоциональный и социальный ущерб. Технологии, такие как социальные сети и мобильные приложения, усугубляют проблему, предоставляя платформу для анонимных атак. Чтобы бороться с кибербуллингом, многие организации и правительства принимают меры: от политик нулевой терпимости до законодательных инициатив. Повышение осведомленности и образование также играют ключевую роль. На переднем крае борьбы с кибербуллингом стоит разработка нейронных сетей, которые могут автоматически определять и блокировать токсичный контент в сети. В данной статье описаны основные этапы обучения нейронной сети и создания информационной технологии на её основе по борьбе с таким явлением как кибербуллинг.*

Ключевые слова: *обработка естественного языка, сентимент-анализ, машинное обучение, кибербуллинг, нейронная сеть, TensorFlow.*

Introduction. The increase in Internet communication has led to the emergence and growth of one of the most acute social problems – cyberbullying. Cyberbullying can cause irreparable harm to the psychological and emotional health of people, cause depression and suicide[1]. A 2022 Pew Research study found that nearly half of all teens (49%) had experienced some form of cyberbullying (https://www.comparitech.com/internet-providers/cyberbullying-statistics/). At the same time, with the development of digital technologies and the spread of social networks, the number of cases of cyberbullying is becoming more and more. One of the effective ways to detect cyberbullying is to use artificial intelligence technologies[2]. In this paper, a neural network model has been developed, tools, patterns and architectural solutions for network training have been proposed; information technology for detecting cases of bullying in social networks.

Literature Review. This section examines the most current research related to the use of neural networks to determine cyberbullying.

Article[3] is devoted to the development of a program for determining the sentiment of a text. The article substantiates the relevance of protecting society from cyberbullying. Methods of counteracting cyberbullying are analyzed. An indicator of the negativity of site information has been introduced. The work of the site blocker is discussed in detail. The use of sentiment analysis, which is based on the use of neural networks, is justified. For sentiment analysis of information flows, a program was developed in the high-level Python programming language with the introduction of ready-made trained neural networks into it. A dictionary based on themes is used.

The main contribution of the authors of the work [4] to the study of the topic is the use of a method for identifying trolls in social networks, which is based on an analysis of the emotional state of network users and behavioral activity. To identify trolls, users were grouped into groups; this grouping is carried out by identifying a similar method of communication. The distribution of users is carried out automatically thanks to the use of a special type of neural networks, namely self-organizing Kohonen maps. The group number is also determined automatically. To determine the characteristics of users on the basis of which the distribution into groups is made, the number of comments, the average length of a comment and an indicator responsible for the emotional state of the user are used.

The article [5] discusses the features of the procedure for automatically detecting cyberbullying in English-language tweets based on the Logistic Regression machine learning model. It is noted that this model allows you to obtain an accurate result of identifying the means of implementing electronic bullying, as well as automatically replenish the database with new cyberbullying markers, including units derived from them.

This work [6] considered solving the problem of multi-class classification of toxic messages based on gradient boosting using the CatBoost library and neural networks using Keras. In particular, convolutional neural networks, neural networks with LSTM and GRU architecture were considered.

The authors[7] used a corpus linguistic-statistical research tool based on the use of relational-situational analysis, psycholinguistic indicators and dictionaries covering the vocabulary of emotional and rational assessment; to obtain data on the level of aggressiveness of the subjects, the Bass-Perry questionnaire was used; When processing the data, binary classification algorithms, support vector machine (SVM) and random forest (Random Forest), were used.

Thus, these works show the relevance of using machine learning methods to identify cyberbullying on social networks. However, further research is needed to develop more accurate and effective models for detecting cyberbullying on social media platforms.

*Materials and methods of research***.** Training a neural network for text processing includes several steps:

1. data collection. The first step in training a neural network is to collect enough data. In the case of working with text, this may include collecting text documents, testimonials, articles, or any other type of text content appropriate to the task;

2. text preprocessing. After collecting the data, it is necessary to pre-process the text. This includes removing unwanted characters, numbers, punctuation, converting text to lowercase, and other operations to clean up data and simplify subsequent analysis;

3. tokenization. The next step is to split the text into individual tokens or words. Tokenization helps convert textual data into a numeric form, allowing the neural network to work with it. Each word or character is assigned a unique identifier;

4. creating a model. After pre-processing the data, it is necessary to create a neural network model. The model defines the architecture of the network, including layers, links, and parameters. You can choose different architectures such as Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN) or Transformers depending on your task;

5. model training. After creating the model, you need to train it on the prepared data. This process includes feeding training examples to the model, calculating the error (loss), and optimizing the model parameters using the gradient descent algorithm. Training can include multiple epochs, where each epoch is a complete pass over all training data;

6. evaluation and tuning of the model. After completing the training of the model, it is required to evaluate its performance. This may include accuracy, recall, F1 score, or other metrics, depending on your task. If necessary, you can tweak the model settings or change its architecture to improve performance.

Each of these stages requires careful work and experimentation in order to achieve good results and ensure the efficient functioning of the neural network.

The development was carried out in Python, Keras and TensorFlow were used to train the neural network – these are two packages from Google that are widely used in the field of deep learning of neural networks [8].

Preparing a dataset for training recurrent networks is one of the most important steps in creating a model. The steps that are usually included in the process of preparing a dataset for training recurrent networks include the following:

1. data collection. It is important to have a large and diverse dataset for training recurrent networks. For this, various data sources can be used, such as social networks, news articles, forum posts, etc.;

2. data cleaning. Data obtained from various sources may contain many errors, typos and other inaccuracies that may adversely affect the quality of the model. Therefore, it is necessary to clean up the data, which includes removing unnecessary characters, correcting typos, etc.;

3. Transform data into a number format: To train recurrent networks, the data must be converted into a number format that can be used as input to the model. For example, for text data, character-to-number encoding can be used using methods such as one-hot encoding or word embedding;

4. splitting the data into training and test sets: after preparing the data and converting it into a numerical format, it is necessary to divide the data into training and test sets. The training set is used to train the model and the test set is used to check the quality of the model;

5. Creation of sequences: Recurrent networks work with sequential data, so the data must be converted into sequences. For example, for text data, you can break the text into sequences of a fixed length.

In general, preparing a dataset for training recurrent networks is a complex process that requires a lot of time and patience. However, the right approach will help to better train the final model and improve its performance in real use [11].

Morphological text analysis is an important step before training a neural network or other natural language processing model. It includes the analysis and analysis of words in the text to determine their lexical and grammatical characteristics[12].

The steps that can be performed in the process of morphological text analysis include the following concepts:

1. tokenization. The text is broken into individual tokens, which can be words or characters. Tokens serve as the basic units of analysis[9].

2. lemmatization. Reduction of words to their lemmas (basic form). Lemmatization allows you to reduce the various grammatical forms of a word to a single basic form[13]. For example, the words “run”, “ran”, “running” will be reduced to the lemma “run”;

3. stemming. The process of stripping affixes from a word to obtain its stem (stem). The result of stemming may be less precise than lemmatization, but it allows words to be reduced to their base form for further analysis[14]. For example, the words “run”, “ran”, “running” can be truncated to the common stem “run”;

4. extraction of syntactic relations. Analysis of connections and relationships between words in a sentence, such as subject and predicate, dependent and independent constructions[10]. This allows for a deeper understanding of sentence structure and the context in which each word is used.

Morphological text analysis helps to make the text understandable for neural network training, as it provides information about the structure and characteristics of words[16].

Text tokenization is the process of converting text data into sequences of numbers (tokens) that can be used to train neural networks[9]. When text is used as input to a neural network, it must be converted to a numeric format. This can be done by breaking the text into individual words (tokens) and then assigning each word a unique number.

A special dataset was used to train the neural network, which contains 14135 classified records in russian. The data structure is as follows: comment – user comment, toxic – comment toxicity score in the form of 0 and 1. You can find the full collection here: https://www.kaggle.com/datasets/blackmoon/russian-language-toxic-comments?resource=download.

The collected data collection should be brought to a form that the neural network learning algorithm understands, therefore, first, it is necessary to bring all the words into normal form, and the necessary tools already exist for this. The pymorphy2, re and nltk packages were used for this.

The following functions are used to convert sentences into the correct format:

```
import pymorphy2
import re
```

```
from nltk.corpus import stopwords

TOKEN_RE = re.compile(r'[а-яё]+')
347roblem_stopwords = stopwords.words("347roblem")
morph = pymorphy2.MorphAnalyzer(lang='ru')
def tokenize_text(txt, min_lenght_token=2):
    txt = txt.lower()
    all_tokens = TOKEN_RE.findall(txt)
    return [token for token in all_tokens if len(token) >= min_lenght_token]

def remove_stopwords(tokens):
    return list(filter(lambda token: token not in 347roblem_stopwords, tokens))

def lemmatizing(tokens):
    return [morph.parse(token)[0].normal_form for token in tokens]
def text_cleaning(txt):
    tokens = tokenize_text(txt)
    tokens = lemmatizing(tokens)
    tokens = remove_stopwords(tokens)
    return ' '.join(tokens)
```

This code implements several functions for preprocessing text in Russian:

1. The necessary libraries are imported: 'pymorphy2' for lemmatization, 're' for working with regular expressions and 'stopwords' from the NLTK library for removing stop words.
2. The regular expression TOKEN_RE is defined, which only searches for Russian words and ignores all other characters in the text.
3. A list of Russian stop words is created and the morphological analyzer pymorphy2 is loaded.
4. The tokenize_text function receives the text txt and the optional argument min_lenght_token (the minimum length of the token) and performs the following steps:

• Converts all text to lower case.
• Searches for all tokens (words) that match the regular expression TOKEN_RE.
• Returns a list of tokens whose length is greater than or equal to min_lenght_token.

5. The remove_stopwords function receives a list of tokens and returns a new list without stop words from the list of Russian stop words.
6. The lemmatizing function takes a list of tokens and returns a list of lemmas (normal forms of words) using pymorphy2.
7. The text_cleaning function receives the text txt and performs the following steps:

• Tokenizes text using the tokenize_text function.
• Lemmatizes tokens using the lemmatizing function.
• Removes stop words from tokens using the remove_stopwords function.
• Concatenates the remaining tokens into a string using spaces and returns the result.

Thus, the text_cleaning function performs pre-processing of the text, which can be used for subsequent analysis of text data.

The Keras package used in the project uses the Tokenizer class to tokenize text. It allows you to convert text into a sequence of numbers, where each word in the text has a unique index.

```
Df = pd.read_csv('vendors/ai/data.csv')
df['comment'] = df['comment'].apply(str)
train_features = df['comment']
```

```
train_labels = df['toxic']
vocab_size = 20000
max_seq_len = 20
vector_size = 300
tokenizer = Tokenizer(oov_token="<OOV>", num_words=vocab_size)
tokenizer.fit_on_texts(train_features)
sequences_train = tokenizer.texts_to_sequences(train_features)
padded_train = pad_sequences(sequences_train, padding='post', maxlen=max_seq_len)
```

The first lines of code load the data from the file "data.csv" into a DataFrame object, previously normalized, then a new "comment" column is created, in which each value is converted to a string to avoid errors.

Further, the data is divided into features and labels. The features are the "comment" column, which contains text comments, and the tags are the "toxic" column, which contains information about whether the comment is toxic or not.

About variables:

1. vocab_size (dictionary size) – the number of unique words in our corpus of texts that will be used to build word vectors;
2. max_seq_len (maximum sequence length) – the maximum length of the sequence of words that will be used to train the model;
3. vector_size (vector dimension) – the number of vector dimensions that each word in our dictionary will represent.

For text tokenization, an object of the Tokenizer class is used. The tokenizer parameters are defined in the class constructor: oov_token is a token that will be used for all words that were not taken into account during training; num_words is the maximum number of words that will be taken into account when training. In this case, the value is set to 20000.

Then, the fit_on_texts() method trains the tokenizer on the text data passed as an argument. Next, the texts_to_sequences() method converts text comments into sequences of numbers based on the trained tokenizer. Finally, the pad_sequences() method adjusts the length of each sequence to max_seq_len, padding it with zeros or truncating it if the length is greater than max_seq_len.

It is worth explaining in detail why to further limit the size of a previously optimized dictionary. The dictionary size limit when training a neural network has several reasons:

• First, large vocabularies result in a large number of parameters that must be trained. This requires more computational resources and time, which complicates the learning process and can lead to overfitting of the model.

• Secondly, there are often words in the text that appear very rarely or even only once. Their use in the model is inefficient, since there will not be enough examples for them to well evaluate their contribution to the prediction. The dictionary constraint allows you to remove such words from the model and improve its performance.

• Finally, limiting the dictionary size helps to reduce noise in the data[10]. Using a dictionary that is too large can cause the model to include random noise words that have nothing to do with the predicted class.

Thus, limiting the size of the dictionary is an important strategy when training neural networks to work with text data [17].

The process of creating a neural network model in Keras and selecting layers are very important steps in developing and training neural networks. Choosing the right architecture and hyperparameters for a neural network can significantly affect the quality of its work. With the wrong choice of layers and parameters, the network may not learn or give poor results [18].

Keras provides many layers and options that can be used to customize the model to suit the problem you are trying to solve. For example, using convolutional layers can be efficient for image processing, while using recurrent layers can be useful for processing sequential data such as texts and time series.

The right choice of layers and parameters can also increase the efficiency of neural network training[18]. For example, using the ReLU activation function instead of the sigmoid can speed up learning by calculating gradients quickly. Also, using Dropout and BatchNormalization layers can prevent overfitting and improve the quality of training.

In general, building an efficient neural network model is a process that requires careful data analysis and experimentation with different architectures and parameters.

After long attempts to get the best result of the neural network, the implementation of the following type of model was chosen:

```
def get_model():
    model = Sequential()
    model.add(Embedding(input_dim=vocab_size, output_dim=vector_size,
input_length=max_seq_len))
    model.add(Dropout(0.6))
    model.add(LSTM(max_seq_len, return_sequences=True))
    model.add(LSTM(6))
    model.add(Dense(1, activation='sigmoid'))
    return model

callbacks = [keras.callbacks.EarlyStopping(monitor="val_loss", patience=2, verbose=1,
mode="min", restore_best_weights=True
    )]
model = get_model()
```

First of all, the get_model() function is defined, which returns a serial model of the neural network.

The first layer of the model is the Embedding layer, which converts integers (word indices) into fixed-length dense vectors. The following parameters are set for this layer:

• input_dim=vocab_size is the size of the dictionary that specifies the input value for the layer;

• output_dim=vector_size — output space dimension, ie. The number of dimensions of the vector that is expected at the output;

• input_length=max_seq_len is the length of each sequence that is passed to the layer.

Next, a Dropout layer with a coefficient of 0.6 is added to the model, which prevents the model from retraining.

Then two LSTM layers are added and used to process the sequences. The first layer of the LSTM returns a sequence, while the second layer only returns the last output element, allowing the output to be dimensionally reduced.

Finally, a Dense layer is added with one output neuron and a sigmoid activation function, which allows texts to be classified into two classes.

The next line defines the callbacks array, which contains the callbacks that are executed during model training. In this case, EarlyStopping is used, which stops training when the val_loss metric does not improve for two epochs (used when compiling the model).

A model is created (see Fig. 1) by calling the get_model() function.

The structure of the developed model:

1. Sequential is the container used to create the model. This container contains one or more

layers that are added sequentially one after the other.

2. Embedding is a layer used to create vector representations of words. Each word in the input dataset is converted to a vector of the given dimension. These vectors are used to pass word information to the next layer of the model.

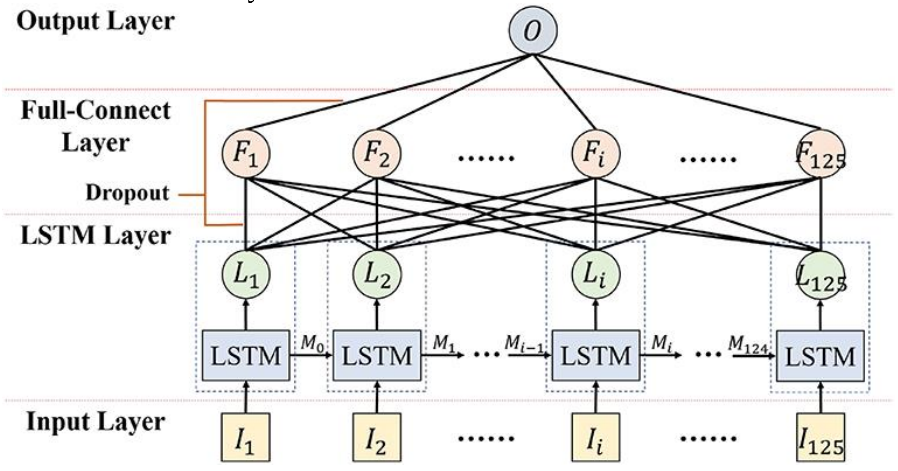

Output Layer
Full-Connect Layer
Dropout
LSTM Layer
Input Layer
O
F_1 F_2 ...... F_i ...... F_{125}
L_1 L_2 L_i L_{125}
LSTM M_0 LSTM M_1 ... M_{i-1} LSTM M_i ... M_{124} LSTM
I_1 I_2 ...... I_i ...... I_{125}

Figure 1. The developed neural network model

3. Dropout is the layer used to regularize the model. It helps prevent overfitting by removing randomly selected input data elements with a given probability. This allows the model to learn more general features.

4. LSTM is a recurrent layer used to parse sequential data such as text. It processes the sequence of inputs element by element and stores information about the previous elements in its memory.

5. Dense is a fully connected layer that takes input from previous layers and produces model predictions. In the code above, the layer has a single output and uses a sigmoid activation function to generate a binary output – the model's prediction about the offensiveness of the comment. A more complex classification is not required in this case.

The finished model needs to be trained:

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
tf.config.run_functions_eagerly(True)
history = model.fit(padded_train, train_labels, validation_split=0.33, callbacks=callbacks,
epochs=10)
```

The model.compile method is used to customize the model training process. Three important parameters are defined here:

1. optimizer is an optimizer that determines the algorithm for updating model weights during training. After much testing, the best result was obtained using the adam optimizer, which is a stochastic gradient descent optimization technique with variable size stochastic gradient steps.

2. loss is a loss function that measures how well the model performs on the training data. In this case, based on the requirements of the model, the binary_crossentropy loss function is used, which is usually used for binary classification.

3. metrics is a list of metrics used to evaluate the performance of the model. In the variant above, the accuracy metric was used, which measures the proportion of correct answers.

Next, calling tf.config.run_functions_eagerly puts TensorFlow into eager mode, which allows TensorFlow operations to execute immediately, as opposed to TensorFlow 1.x's compute graph and lazy execution.

Finally, model.fit starts the training process. Here, padded_train and train_labels represent the training dataset, and validation_split=0.33 means that 33% of the data will be used as the validation set to evaluate the performance of the model on data that it did not see during training.

Callbacks=callbacks is used to define additional callbacks to be executed during the learning process, such as early stopping. The array itself was mentioned earlier in the model design phase.

And epochs=10 specifies the number of epochs (full training cycles on the training dataset) that the model will train.

Here it is worth dwelling on the loss functions, the optimizer, and the choice of the number of training epochs in order to analyze in more detail why these parameter values were used.

The binary crossentropy loss function is used in binary classification problems (when there are only two classes), and it measures the difference between the real and predicted values between two classes. It is used to determine how well a neural network models class probabilities and is the most common loss function for binary classification problems.

The adam optimizer is a stochastic optimization technique that is used to tune model parameters based on loss function gradients[19]. Adam is an adaptive optimization method that adjusts the learning rate for each parameter individually, which allows the neural network to converge faster to optimal parameter values.

The number of epochs determines how many times the model will run through the entire training dataset. Each epoch consists of one or more training steps, during which the model receives training data as input, calculates the predictions and compares them with the true values, and then optimizes the parameters based on the loss function.

The number of epochs is chosen based on how quickly the model converges to optimal values, and what accuracy is achieved on the validation data set.

When choosing the number of epochs, it should be taken into account that too few epochs can lead to underfitting of the model, and too many epochs can lead to overfitting.

After running all the previously written code, the process of training the model starts, in this case the result was as follows:

```
    # Output:
# Epoch 1/10
# 296/296 [==============================] – 133s 449ms/step
# - loss: 0.4777 – accuracy: 0.7656
# - val_loss: 0.2748 – val_accuracy: 0.9016
# Epoch 2/10
# 296/296 [==============================] – 150s 507ms/step
# - loss: 0.2191 – accuracy: 0.9207
# - val_loss: 0.2618 – val_accuracy: 0.8928
# Epoch 3/10
# 296/296 [==============================] – 157s 530ms/step
# - loss: 0.1150 – accuracy: 0.9623
# - val_loss: 0.2914 – val_accuracy: 0.8939
# Epoch 4/10
# 296/296 [==============================] – ETA: 0s
# - loss: 0.0641 – accuracy: 0.9808
# Restoring model weights from the end of the best epoch: 2.
# 296/296 [==============================] – 132s 446ms/step
# - loss: 0.0641 – accuracy: 0.9808
# - val_loss: 0.3445 – val_accuracy: 0.8857
# Epoch 4: early stopping
```

As can be seen, in the first training epoch, the model achieved a loss of 0.4777 and an accuracy of 0.7656 on the training data, and a loss of 0.2748 and an accuracy of 0.9016 on the validation data.

In the second epoch, the model achieved a significant improvement in accuracy, reaching 0.9207 on the training data and 0.9020 on the validation data, with a loss of 0.2605.

In the third epoch, the model continued to improve, reaching a loss of 0.1150 and an accuracy of 0.9623 on the training data, but on the validation data it slightly worsened to a loss of 0.2914 and an accuracy of 0.8939.

At the end of the fourth epoch, the model retained the weights from the best epoch (epoch 2) because that epoch performed best on the validation data. And as a result, the model achieved a loss of 0.0641 and an accuracy of 0.9808 (98%) on the training data in this epoch. The training process ended ahead of schedule.

In conclusion about the training process, it can be noted that the model based on the LSTM architecture was trained on the classification dataset and showed good results, improving its accuracy at each training epoch.

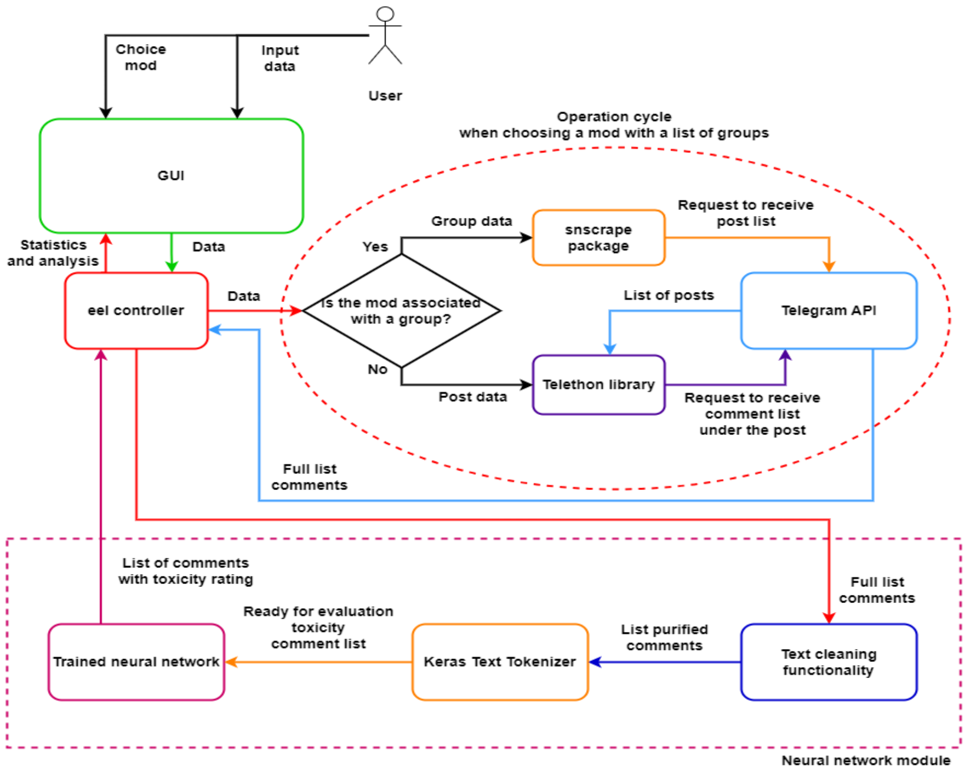
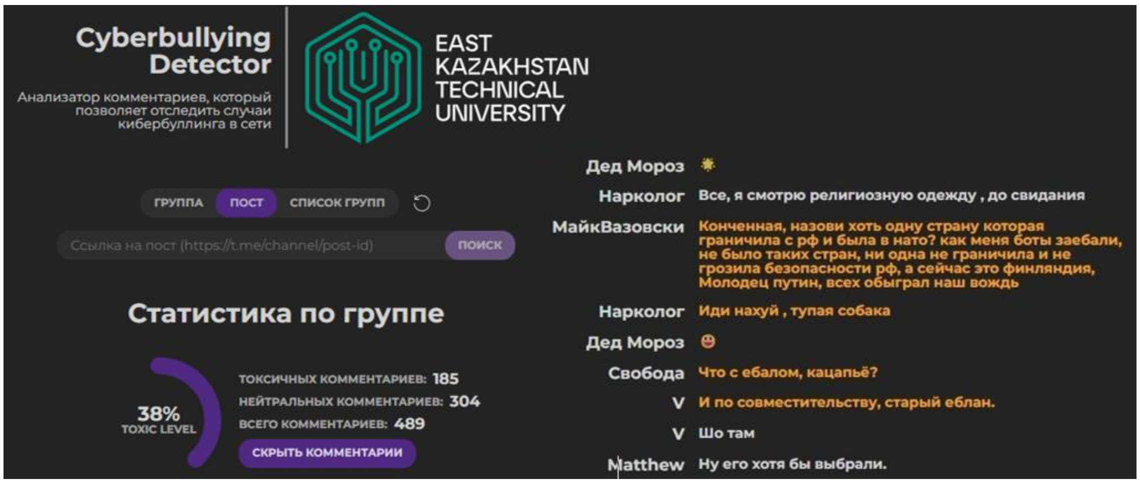


Figure 2. Bullying detection information technology

As an implementation of the trained neural network, an application with a graphical interface was written. The UI was developed using the Vue.js framework. To call the Python code, the eel library was included. Information technology looks like this (see Fig. 2).

Figure 2 shows the complete architecture of the entire application, based on which it is clear that all components are isolated and each works independently of other functionality, this approach avoids potential errors and simplifies the transfer of the desired component to another application, if there is such a need. The center of all software is a controller that connects the user interface, the logic of interaction between libraries for parsing data from Telegram, and the neural network module.


Cyberbullying Detector
Анализатор комментариев, который позволяет отследить случаи кибербуллинга в сети
EAST KAZAKHSTAN TECHNICAL UNIVERSITY
ГРУППА
ПОСТ
СПИСОК ГРУПП
Ссылка на пост (https://t.me/channel/post-id)
ПОИСК
Статистика по группе
38%
TOXIC LEVEL
ТОКСИЧНЫХ КОММЕНТАРИЕВ: 185
НЕЙТРАЛЬНЫХ КОММЕНТАРИЕВ: 304
ВСЕГО КОММЕНТАРИЕВ: 489
СКРЫТЬ КОММЕНТАРИИ

Дед Мороз 🌞
Нарколог Все, я смотрю религиозную одежду , до свидания
МайкВазовски Конченная, назови хоть одну страну которая граничила с рф и была в нато? как меня боты заебали, не было таких стран, ни одна не граничила и не грозила безопасности рф, а сейчас это финляндия, Молодец путин, всех обыграл наш вождь
Нарколог Иди нахуй , тупая собака
Дед Мороз 😃
Свобода Что с ебалом, кацапьё?
V И по совместительству, старый еблан.
V Шо там
Matthew Ну его хотя бы выбрали.

Figure 3. Application interface and assessing the toxicity of real user comments

The application interface (see Fig. 3) contains several mods, when selected, you can collect all comments under one post, under a certain number of posts of a specific group or an entire list of groups. Then statistics are collected based on the assessment of the toxicity of comments from the neural network. A corresponding diagram with calculations is displayed, which clearly shows the level of toxicity in the comments.

You can also expand the list of comments that were received by parsing posts. This list consists of the username and comment. Cleaned comments are submitted for evaluation of the neural network, but their original versions are displayed for final viewing. The interface highlights toxic comments in orange, while neutral ones are displayed in white. Thus, in a matter of seconds it is possible to collect a large amount of data and assess the level of toxicity of the audience of various groups. You can find the full source code for the project here: https://github.com/afteralls/ai-cyberbullying-detector.

Results and discussion. As a result of the work, a tool was created that will automatically recognize cases of insults and help in the fight against cyberbullying.

During the project the following tasks were completed: the theory of neural networks has been studied, a neural network model was created and compiled, a neural network model has been trained that can detect cases of online abuse, an application was designed to interact with the trained model.

Conclusion.

At the end of the article, the relevance of the problem associated with the growth of cyberbullying in the modern information society is considered, and an innovative solution based on artificial intelligence is proposed.

As a result of this work, a tool has been created that will automatically recognize instances of abuse and help in combating cyberbullying with a trained Keras model using the LSTM architecture.

Through trial and experimentation with model building, it was found that recurrent neural networks such as LSTM perform well when processing sequential data such as texts. Using this architecture allowed the model to identify text message characteristics indicative of cyberbullying.

Further research may be aimed at expanding the data set and improving the architecture of the model to improve its accuracy and efficiency in detecting cyberbullying. However, the results of this study demonstrate that recurrent neural networks can be an effective tool in the fight against online cyberbullying.

References

1. Mogunova M. M. Kiberbulling kak novaya opasnost' //Vestnik Severo-Kazahstanskogo Universiteta im. M. Kozybaeva. – 2022. – №. 2 (51). – S. 99-106.
2. Evseev V. L., Sadekova R. Sh. Countering cyberbullying in social networks // Security of information technologies. – 2021. – T. 28. – No. 3. – Pp. 92-102.
3. Pleshakova E. S. et al. Identification of cyberbullying using neural network methods // Security Issues. – 2022. – No. 3. – Pp. 28-38.
4. Ilyukovich T. S. Recognition of cyberbullying in English-language tweets using machine learning // Science today: problems and ways to solve them. – 2020. – P. 94-97.
5. Vasilchenko A. V. DEVELOPMENT OF A SYSTEM FOR AUTOMATIC DETECTION OF TOXIC COMMENTS IN SOCIAL NETWORKS //STUDENT SCIENCE: CURRENT ISSUES, ACHIEVEMENTS AND INNOVATIONS. – 2021. – pp. 35-38.
6. Kovalev A.K. et al. Methods for identifying psychological characteristics of the author from the text (using the example of aggressiveness) // Issues of cybersecurity. – 2019. – No. 4 (32). – Pp. 72-79.
7. Kisova V.V., Semenova E.A. Ispol'zovanie sovremennyh komp'yuternyh tekhnologij dlya opredeleniya verbal'noj reprezentacii kiberbullinga u podrostkov //Problemy sovremennogo pedagogicheskogo obrazovaniya. – 2022. – №. 77-4. – S. 381-383.
8. Géron A. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow. – " O'Reilly Media, Inc.", 2022.
9. Nikitin YU. V., Horoshilov A. A., Makarova A. E. Tokenizaciya tekstov na osnove metoda funkcional'nyh shablonov //Sistemy i sredstva informatiki. – 2022. – T. 32. – №. 4. – S. 59-68.
10. Platonov E. N. 354rob. Vyyavlenie i klassifikaciya toksichnyh vyskazyvanij metodami mashinnogo obucheniya //Modelirovanie i analiz dannyh. – 2022. – T. 12. – №. 1. – S. 27-48.
11. Sohinov D. YU., Kravchenko R. A., Logacheva O. V. REKOMENDACII PO PODGOTOVKE DATASET DLYA MASHINNOGO OBUCHENIYA. – 2023.
12. Zhubanov A. K. Ob avtomaticheskom morfologicheskom analize teksta prirodnogo (estestvennogo) yazyka //Tiltanym. – 2022. – №. 4. – S. 3-7.
13. Kovalevskij P. O. AVTOMATICHESKAYA OBRABOTKA TEKSTA (PROBLEMA LEMMATIZACII) //Yazyk, kul'tura, mental'nost': 354roblem i perspektivy filologicheskih issledovanij. – 2022. – S. 135-138.
14. KORYUKIN A. V. ISSLEDOVANIE VLIYANIYA STEMMINGA I LEMMATIZACII NA KACHESTVO BINARNOJ KLASSIFIKACII PO TONAL'NOSTI KRATKIH TEKSTOVYH KOMMENTARIEV //Aktual'nye issledovaniya. – 2021. – S. 10.
15. Pimeshkov V. K., Dikovickij V. V., Shishaev M. G. Izvlechenie otnoshenij tezaurusa iz tekstov na estestvennom yazyke s ispol'zovaniem statisticheskih i lingvisticheskih metodov //Trudy Kol'skogo nauchnogo centra RAN. – 2020. – T. 11. – №. 8-11. – S. 188-192.
16. Sidorova E. A. Kompleksnyj podhod k issledovaniyu leksicheskih harakteristik teksta //Vestnik SibGUTI. – 2019. – №. 3. – S. 80-88.
17. Nemchinova E. A., Plotnikova N. P., Fedosin S. A. Podgotovka i obrabotka normativno-spravochnoj tekstovoj informacii dlya klassifikacii s pomoshch'yu iskusstvennyh nejronnyh setej //Nelinejnyj mir. – 2019. – T. 17. – №. 2. – S. 27-33.
18. Semchenko R. V., Erovlev P. A. Programmirovanie nejronnyh setej v Python s ispol'zovaniem bibliotek keras i tensorflow //Postulat. – 2020. – №. 7 iyul'.
19. Zhang Z. Improved adam optimizer for deep neural networks //2018 IEEE/ACM 26th international symposium on quality of service (IWQoS). – Ieee, 2018. – C. 1-2.