Emotion Recognition from Tweets

Jakub Sydor[1], Szymon Cwynar[1]

[1]Faculty of Applied Mathematics, Silesian University of Technology, Kaszubska 23, 44100 Gliwice, POLAND

Abstract

These days we face more and more often internet bullying. Our goal was to develop software, which would recognize emotions from the bare text. Our project is based on Twitter posts, but it could be also used in every single platform, in which users communicate via text messages. We use a few solutions to make our program as accurate as it possibly could get. Firstly we picked a large database to get the biggest context, We used World2Vec to represent words as vectors, and lastly, we used a neural network to predict output from sentences beyond our database. Our article is mostly about different versions of the algorithm and comparison to choose the best approach to the problem. As we learned the biggest difference-maker was the amount of hidden layers and number of neurons inside each one of them, type of activation function, and training algorithm. We attach a big amount of plots to visualize each of our tries. In our article, we will try to show our approaches and data which is connected to those approaches. We created functions to monitor our error, the accuracy function to sum up our algorithm - how efficient it is, precision function - to diagnose what proportion of identifications was correct, recall - a fraction of relevant instances that were retrieved and f1 which combines precision and recall to make an average of it valued from 0 to 1.

Keywords

Artificial neural network, Word2vec, emotion, tweets

1. Introduction

The assumption of our project was to create an algorithm based on an artificial neural network. Its main goal was to recognize whether the entry is neutral, negative, or positive. The algorithm is learning on a base that contains 1.6m tweets using backpropagation algorithm. We decided to use neural network as classifiers as they have been reported in various interesting applications [1, 2, 3, 4].

In [5] neural networks are used in federated systems in which they resource information each other during training. Models of neural networks are also very efficient in detection threats over internet [6]. We can also find them as classifiers in images [7] and systems of IoT to detect position of people [8, 9, 10].

We got our database from Kaggle but it was full of unnecessary data like date or user. We cleared it and left only 2 columns - target and text, we got rid of columns that contained information like date, the user, or tweet id.

To make our algorithm work we needed to divide it into few subsections. The first of them is a section connected to a database. Firstly, as mentioned before, we dropped most of the columns, but secondly, we needed to make sure that our data is not containing unused data or data which possibly could make our algorithm less reliable. Therefore we cleared it out of stuff like Names, links, and mentions.

Next algorithm used in our program is Word2Vec which is responsible for translating our sentences and words into numbers. Every word is represented by a 10-dimensional vector. The algorithm which stands behind word2Vec is nothing else than an artificial neural network, which would be explained later. Because of the length of our one-word vector and the maximum length of twitter expression (280 words) we created an input layer which size is simply the result of the multiplication of those 2 values, which is 2800 neurons.

Then we move to the heart of our program - the artificial neural network. The whole structure is handwritten by us, we don't use any libraries. Its main functions are run and addlayer, which are responsible for adding layers and running the whole algorithm. The run function returns 2 output neurons, which represent, by using softmax, for the probability of label. The first neuron is the possibility of positive output and the second one for negative. We also add a function that check the absolute value of its difference, if it's small enough then the output is equal to neutral. The artificial network includes the input layer with 2800 neurons, a first hidden layer with 600 neurons, a second hidden layer with 200 neurons, third hidden layer with 20 neurons, and output layer which consists of 2 neurons.

2. Data Base

Our database consists of 1 600 000 tweets, each item is represented by 5 columns. One record includes the date

SYSTEM 2021 @ Scholar's Yearly Symposium of Technology, Engineering and Mathematics. July 27–29, 2021, Catania, IT

jakusyd988@student.polsl.pl (J. Sydor);
szymcwy664@student.polsl.pl (S. Cwynar)
https://github.com/Harasz/ (J. Sydor);
https://github.com/SzymCwy/ (S. Cwynar)

of the tweet, posting users nickname, the id of a tweet, the content of the tweet, and the label - if it's positive or negative. We needed to modify our database to contain only 2 of 5 columns because only text and labels will be used. Except that we needed to adapt our database to our use and clear it of meaningless text.

So firstly we loaded the database using 'pandas' and provided our data frame with labels to make the access easier. Also, we've implemented a function whose main task was to clear any irrelevant text, such as pronouns, conjunctions, links, and mentions, to make sure our algorithm will learn properly.

3. Algorithm overall

Our algorithms inputs are sentences that in the next steps are converted into words. Using Word2Vec each word in the sentence is converted into ten-dimensional vectors and those vectors are inserted into the array, each vector as a separate element of our array. Those words are easily available because of two mechanisms, word to id and id to word. Now using an artificial network, weighted sum and activation function the algorithm is filling every single neuron with proper values. In the output layer, we have 2 neurons that, at the end of the algorithm, return two values between 0 and 1. Because of softmax, those values can be identified as the probability of each label. When learning those two values are collated with expected outcomes. That way we get the distance between our result and the real label and we used that values in the algorithm of backward propagation to change all of the weights so that our program is getting more and more precise with each iteration.

4. Word2Vec algorithm

We are using the gensim library to implement the algorithm of Word2Vec. This part of our code lets us change words in the database to vectors, so they can be used in our calculations. The algorithm as input data takes the whole data frame with all sentences, each row represented as one sentence. Firstly sentences need to be divided into words. Nextly we count how many times each word occurs in the text and based on that information we create 2 dictionaries - word to id and id to word, which would make the conversion from text to id and inversely easier. In the built-in function, we need to specify the size of the vector, minimal number of occurrences, window, and source of words. In our example we set minimum occurrence to 1, size of vector to 10 and window to 7, to make sure our dictionary would be big, to connect big amounts word witch each other and also we needed 10-dimensional vector for every word so it would fit our input layer. Word2Vec is nothing else than an artificial neural network and it allows us to make mathematical operations on words. Gensims Wor2Vec implements two functions - Continuous Bag Of Words(CBOW) and Skip-Gram.

In the CBOW model surrounding words is combined to predict the word they surround, while in the Skip-Gram we use a word to predict the context.

5. Mathematical representation of Skip-Gram model

Mathematically, we can describe n-word sentence $w_1, \ldots, w_n$, using skip-gram as following formula:

$$SkipGram = w_{i_1}, w_{i_2}, \ldots, w_{i_n} | \sum_{j=1}^{n} i_j - i_{j-1} < k \tag{1}$$

where k is max skip-distance and n to subsequence length.

For example, when we have the sentence "I love to write scripts" and k is equal to 1 and n to 2, that means we will connect 2 words, which have a maximum of one word between them. Those connections would be: {I,love},{I, to},{love, to},{love, write},{to, write},{to, scripts},{write,scripts}.

6. Backward propagation

A backward propagation algorithm is used in our program to modify the weights of each neuron to get the best results. Our neurons have pregenerated weights from 0 to 1. To make our algorithm more precise by analyzing the errors backward propagation algorithm correct native weights starting from the end of our artificial neural network. As an input, it takes the probability of each label and the expected label. It calculates the error of each of the output neurons and those errors are propagated to previous layers. Each weight in our network is being modified based on the value of the error. This algorithm has its limit so you need to be careful while setting its iterations. After a few runs values are being modified to a lesser extent, so when those changes are minor, that's the sign to stop the algorithm.

7. Activation function

Activation function is inseparable element of every artificial neural network. We have lots of them available but each of them is different. We use s-shaped function - hiperbolic tangent as our activation function. It determines the output of artificial neural network. All of the output values are between 0 and -1. The advantage of

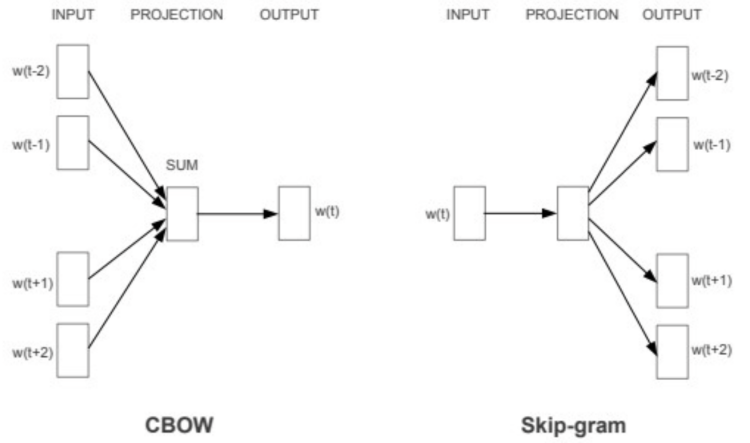

Figure 1: Graphical representation of the CBOW model and Skip-gram model [11].

Algorithm 1 Backpropagation Algorithm

procedure TRAIN
  $X \leftarrow$ Training Data Set of size mxn
  $y \leftarrow$ Labels for records in X
  $w \leftarrow$ The weights for respective layers
  $l \leftarrow$ The number of layers in the neural network, 1...L
  $D_{ij}^{(l)} \leftarrow$ The error for all l,i,j
  $t_{ij}^{(l)} \leftarrow 0$. For all l,i,j
  For $i = 1$ to m
    $a^l \leftarrow feedforward(x^{(i)}, w)$
    $d^l \leftarrow a(L) - y(i)$
    $t_{ij}^{(l)} \leftarrow t_{ij}^{(l)} + a_j^{(l)} \cdot t_i^{l+1}$
  if $j \neq 0$ **then**
    $D_{ij}^{(l)} \leftarrow \frac{1}{m} t_{ij}^{(l)} + \lambda w_{ij}^{(l)}$
  else
    $D_{ij}^{(l)} \leftarrow \frac{1}{m} t_{ij}^{(l)}$
    where $\frac{\partial}{\partial w_{ij}^{(l)}} J(w) = D_{ij}^{(l)}$

Figure 2: Pseudo-code of the back-propagation algorithm in training ANN [12].

our activation function is mapping, all positive and negative values will be presented as strong values and those which are close to 0 would be close to 0 on the tanh graph. We also have chosen tanh function because is strongly advised when neural network has only 2 outputs.

8. Maths behind activation function

Our activation function - Hyperbolic tangent might be represented as:

$$\tanh x = \frac{\sinh x}{\cosh x} \tag{2}$$

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3}$$

$$\frac{\partial}{\partial x}\tanh x = \frac{(e^x + e^{-x})(e^x + e^{-x}) - (e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2} \tag{4}$$

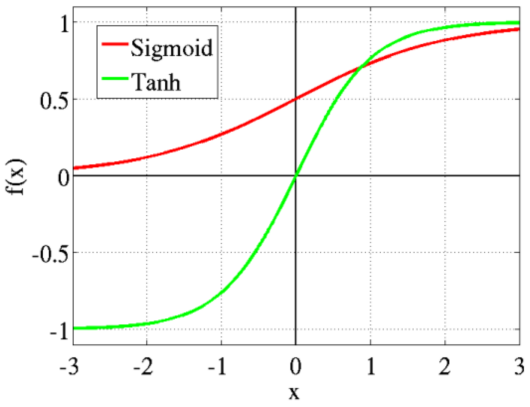

Figure 3: Comprehension between sigmoid and tanh activation functions. [13].

It's domain is range from -1 to 1. Its monotonic function, which derivative is non monotonic. Derivative of tanh:

$$\frac{\partial}{\partial x}\tanh x = 1 - \tanh^2 x \tag{5}$$

9. Artificial Neural Network

Our neural network algorithm is divided into to main classes. NeuralNetwork class and Neuron class.

NeuralNetwork has 2 variables - layers and weight. Which store respectively arrays of Neurons and theirs weights. First funtion is addlayer which was written to allow creating layers with specific number of neurons inside it given as an argument. When used it adds elements of Neuron class into the array of layers. Get size function is used to return number of neurons in whole artificial neural network, thanks to that function we are able to properly use generate weights. With the result of the previous function we use generate weights to create array with randomly generated numbers from 0 to 1. Load weights function is responsible for assigning weights to neurons. The last of them is run, it firstly checks if algorithm has the same amount of input neurons and inputs given by a user. If it returns true it starts to assign values to neurons.

The next class - Neuron, is responsible for calculating weighted sum and using activation function to assign values to each neuron.

10. Inference

Inference in our algorithm is simply choosing an option with a higher probability. Thanks to softmax we get on our output layer two neurons with probabilities for each label. Firstly we need to convert output as it is in a form that cannot be compared to label from our database. The output is a two-dimensional array, where first we got the probability of positive tweet and the second as the negative one. So we need to make a variable 'expected', so it would be represented as a two-dimensional array. Next we check if the absolute value of the subtraction is bigger than 0.1, if it's not we make our entry neutral. If one of the values is big enough we assign respectively the label. As we compare these two values we also calculate the accuracy of our algorithm. Below we represent the pseudocode of inference. [H] Input Data: sentence label *k*, array of vectors *j* (sentence represented as an array of vectors), Choosing the label of sentence

$k == 0: \quad expected = [0, 1] \quad expected = [1, 0]$

neo = Artificial neural net output as probability of each label

absolute = Absolute value of subtraction both output values

absolute < 0.1: Neutral *neo* [0] < *neo* [1] Positive Negative Inference Algorithm.

11. SoftMax

Softmax is exponential function, which normalizes values of our 2 output neurons to the sum of 1. We use that function in our program to represent both of our output neurons values as a probability of getting positive or negative label. Author names can have some kinds of marks and notes:

$$Softmax(x_i) = \frac{exp(x_i)}{\sum_j exp(x_j)} \tag{6}$$

Additionaly apart from calculating softmax, we need to get its derivative. Its used by backpropagation function during calculating difference between expected values and outputs from our net. We start by separately computing derivatives. First for the first neuron.

$$\frac{\partial S(z_1)}{\partial z_1} = \frac{\frac{\partial e^{z_1}}{\partial z_1} \cdot (e^{z_1} + e^{z_2}) - \frac{\partial}{\partial z_1}(e^{z_1} + e^{z_2}) \cdot e^{z_1}}{(e^{z_1} + e^{z_2})^2} \tag{7}$$

so we have:

$$\frac{\partial}{\partial z_1} S(z_1) = S(z_1) \times (1 - S(z_1)) \tag{8}$$

Now for the second one.

$$\frac{\partial S(z_2)}{\partial z_1} = \frac{\frac{\partial e^{z_2}}{\partial z_1} \cdot (e^{z_1} + e^{z_2}) - \frac{\partial}{\partial z_1}(e^{z_1} + e^{z_2}) \cdot e^{z_2}}{(e^{z_1} + e^{z_2})^2} \tag{9}$$

so we have:

$$\frac{\partial}{\partial z_1} S(z_2) = -S(z_1) \times S(z_2) \tag{10}$$

Conclusion for N outputs

$$S(z_1) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \tag{11}$$

General formula for softmax derivative for N outputs:

$$\frac{\partial}{\partial z_j} S(z_i) = \begin{cases} S(z_i) \times (1 - S(z_i)) & \text{if} \quad i = j \\ -S(z_i) \times S(z_j) & \text{if} \quad i \neq j \end{cases} \tag{12}$$

If we are computing $\frac{\partial}{\partial z_i} S(z_i)$ the output is always $S(z_i) \times (1 - S(z_i))$. However when we are computing $\frac{\partial}{\partial z_j} S(z_i)$ the output changes to $-S(z_i) \times S(z_j)$

12. Precision function

Precision is a function which shows the proportion of true positive identifications. If we analyse retrieval of information, precision is fraction of correct results divided by all returned results. We calculate precision from two variables: TP and FP. Which respectively stands for true positive and false positive. By the term true positive we mean outcome where the model correctly predicted positive class and false positive as incorrect prediction of positive class.

Precision is given by the following formula:

$$Precision = \frac{TP}{TP + FP} \tag{13}$$

When precision rate is equal to 1.0, that means that model produces no false positives.

Example of calculating Precision:

$$Precision = \frac{TP}{TP + FP} \tag{14}$$

$$Precision = \frac{10}{10 + 3} \tag{15}$$

$$Precision = 10/13 \approx 0.769 \tag{16}$$

True Positives: 10	False Positives: 3
False Negatives: 2	True Negatives: 15

13. Recall function

Recall function is very similar to precision function. The only difference is that we compare true positive values to false negatives (incorrect predictions - model predict incorrectly negative class). Our model is most efficient when recall factor is 1.0 - that means there are no false negatives.

The equation is also very similar:

$$Recall = \frac{TP}{TP + FN} \tag{17}$$

And when we calculate Recall one the same set of data as Precision that's our outcome:

True Positives: 10	False Positives: 3
False Negatives: 2	True Negatives: 15

$$Recall = \frac{TP}{TP + FN} \tag{18}$$

$$Recall = \frac{10}{10 + 2} \tag{19}$$

$$Recall = 10/12 \approx 0.833 \tag{20}$$

14. Comparison of recall and precision

The comparison of those 2 function is very difficult because of tension between them. That means if you improve one of them, the second one is reducing its precision. Using data above we got:
Precision ≈ 0.769
Recall ≈ 0.833

For data:

True Positives: 10	False Positives: 3
False Negatives: 2	True Negatives: 15

When we decrease number of FP and FN increases: We got:

True Positives: 10	False Positives: 1
False Negatives: 4	True Negatives: 15

Precision ≈ 0.91
Recall ≈ 0.71

And when we do the opposite thing, we decrease number of FN and increase number of FP: We got:

True Positives: 10	False Positives: 4
False Negatives: 1	True Negatives: 15

Precision ≈ 0.71
Recall ≈ 0.91

So we got to conclusion that they are not quite comparable, but there is another method which uses both of them in the calculations and it's named F1 score.

15. F1 score

Firstly the name of F1 score, also known as F-measure, is believed to refer to different F funcion, which was concluded in Van Rijsbegens Book, when introduced to the Fourth Message Understanding Conference.

F1 score is measurement of test's accuracy. It is calculated from the recall and precision. F-meause is the harmonic mean (the reciprocal of the arithmetic mean of the reciprocals of the given set of observations) of Precision and Recall. It can be modified by additional weights, valuing precision or recall more than other.

The highest value of F1 score is 1.0 which indicates the best precision and recall and 0 indicates that one of precision or recall is equal to 0.

F-measue is also known as Sørensen–Dice coefficient or Dice similarity coefficient (DSC).

$$F_1 = \frac{2}{recall^{-1} + precision^{-1}} \tag{21}$$

$$F_1 = 2 \times \frac{precision \times recall}{recall + precision} \tag{22}$$

$$F_1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{23}$$

Example of calculating F1 score:

True Positives: 10	False Positives: 3
False Negatives: 2	True Negatives: 15

$$F_1 = \frac{10}{10 + \frac{1}{2}(3 + 2)} \tag{24}$$

$$F_1 = \frac{10}{12.5} \tag{25}$$

$F_1 = 0.8$

F_β Score is used when we want recall to be considered β times more important than precision, where β is positive real factor.

$$F_\beta = (1 + \beta^2) \times \frac{precision \times recall}{(\beta^2 \times precision) + recall} \tag{26}$$

$$F_\beta = \frac{(1 + \beta^2) \times TP}{(1 + \beta^2) \times TP + \beta^2 * FN + FP} \tag{27}$$

When β is equal to 2 recall weights are higher than precision, however when its equal to 0.5 weights of precision are higher than recall. Example of calculating F_β score

True Positives: 10	False Positives: 3
False Negatives: 2	True Negatives: 15

with $\beta = 2$:

$$F_\beta = \frac{(1 + 4) \times 10}{(1 + 4) \times 10 + 4 * 2 + 3} \tag{28}$$

$$F_\beta = \frac{40}{55} \tag{29}$$

$F_\beta \approx 0.73$

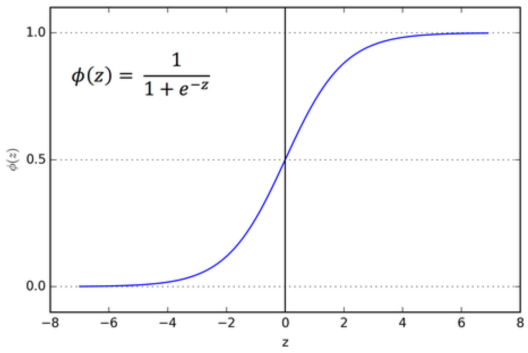

Figure 4: Sigmoid activation function [13].

16. Experiments

We've experimented with

- Activation Function
- Artificial neural network learning algorithms
- Structure of artificial neural network

We've chosen Hyperbolic tangent as our main activation function. We decided on that function after comparison of three functions - ReLU, Sigmoid and Tanh, as we thaught it would fit our algorithm best. We have tested all of them by accuracy and f1 score functions. Also we've

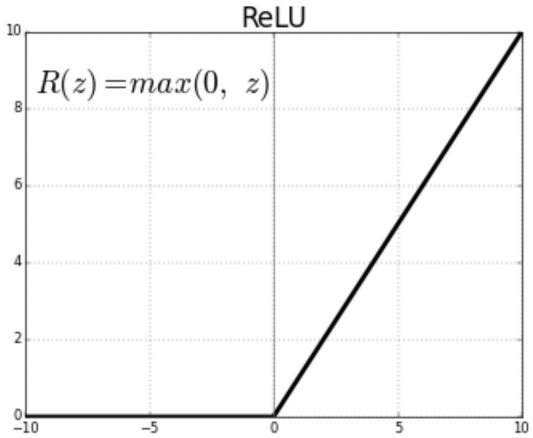

Figure 5: ReLU activation function [14].

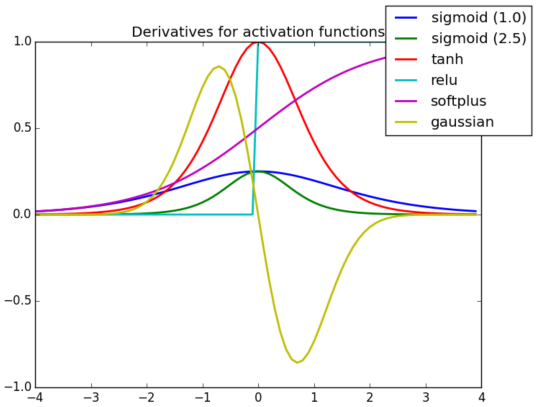

Figure 6: Comparison of activation functions [13].

read articles that proves that this type of function is the best to neural network with 2 neuron in output layer. Thing that outweigh the decision was its shape. Because of tanh function we are able to easily spot negative values and those which are close to 0.

We also have used softmax function to represent values on our output neurons as probability of each label. Apart from that we have used derivative of soft max in algorithm of back propagation to decrease the error.

We tried Particle Swarm Optimization and backward propagation as our learning algorithms. After reading articles and running some test, we decided to use backward propagation as it was easier to use with softmax and also was more efficient than PSO.

After many tries we ended our tests with 3 hidden layers: first - 600, second 200, third 10, as input and output layer number of neurons is constant - 2800 inputs and 2 outputs.

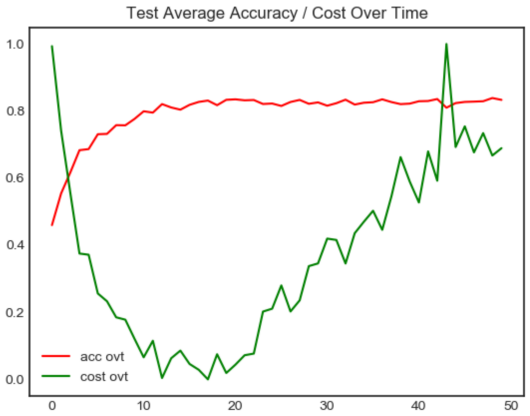

Figure 7: Accuracy/Cost for Test Over Time [15].

17. Conclusions

In our work we have tested application of neural networks to word processing purposes. We have used special library to work with tweets. Our idea was tested and results show we have good model which is able to work with tweets. in future works we will try to develop furhter our project to make it also compare tweets between various authors. We will also work to apply other models and ideas to compar them to this presented neural network.

References

[1] S. Brusca, G. Capizzi, G. Lo Sciuto, G. Susi, A new design methodology to predict wind farm energy production by means of a spiking neural network–based system, International Journal of Numerical Modelling: Electronic Networks, Devices and Fields 32 (2019). doi:10.1002/jnm.2267.

[2] G. Capizzi, C. Napoli, L. Paternò, An innovative hybrid neuro-wavelet method for reconstruction of missing data in astronomical photometric surveys, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 7267 LNAI (2012) 21–29. doi:10.1007/978-3-642-29347-4_3.

[3] G. Capizzi, F. Bonanno, C. Napoli, Hybrid neural networks architectures for soc and voltage prediction of new generation batteries storage, 2011. doi:10.1109/ICCEP.2011.6036301.

[4] C. Napoli, F. Bonanno, G. Capizzi, An hybrid neuro-wavelet approach for long-term prediction of solar wind, Proceedings of the International Astronomical Union 6 (2010) 153–155.

[5] D. Połap, M. Woźniak, Meta-heuristic as manager

in federated learning approaches for image processing purposes, Applied Soft Computing 113 (2021) 107872.
[6] M. Wozniak, J. Silka, M. Wieczorek, M. Alrashoud, Recurrent neural network model for iot and networking malware threat detection, IEEE Transactions on Industrial Informatics 17 (2021) 5583–5594.
[7] X. Liu, S. Chen, L. Song, M. Woźniak, S. Liu, Self-attention negative feedback network for real-time image super-resolution, Journal of King Saud University-Computer and Information Sciences (2021).
[8] G. Capizzi, C. Napoli, S. Russo, M. Woźniak, Lessening stress and anxiety-related behaviors by means of ai-driven drones for aromatherapy, volume 2594, 2020, pp. 7–12.
[9] M. Woźniak, M. Wieczorek, J. Siłka, D. Połap, Body pose prediction based on motion sensor data and recurrent neural network, IEEE Transactions on Industrial Informatics 17 (2020) 2101–2111.
[10] R. Avanzato, F. Beritelli, M. Russo, S. Russo, M. Vaccaro, Yolov3-based mask and face recognition algorithm for individual protection applications, in: CEUR Workshop Proc., 2020, pp. 41–45.
[11] T. Mikolov, Q. V. Le, I. Sutskever, Exploiting similarities among languages for machine translation, arXiv preprint arXiv:1309.4168 (2013).
[12] H. Guo, H. Nguyen, D.-A. Vu, X.-N. Bui, Forecasting mining capital cost for open-pit mining projects based on artificial neural network approach, Resources Policy (2019) 101474.
[13] S. Sharma, S. Sharma, A. Athaiya, Activation functions in neural networks, towards data science 6 (2017) 310–316.
[14] K. Sarkar, Relu: Not a differentiable function: Why used in gradient based optimization? and other generalizations of relu, Data Science Group, IITR (2018).
[15] J. D. Seo, Unfair back propagation with tensorflow [manual back propagation with tf], 2018.