UDC 621.91.01:004.032.26

O. Sokolova., Student, S. Vysloukh, PhD, Associate Professor
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

MODELING OF PARAMETERS OF THE MILLING OF PARTS FROM HEAT-RESISTANT STEEL

Annotation. The article discusses the problems of research of the machinability of alloy steels during their machining. The results of face milling of heat-resistant steel under different processing modes are presented. With using the TensorFlow platform tools, the technological process of milling is modeled. The results of predicting the processing power at different cutting modes are presented.

Keywords: machine learning, modeling, neural networks, TensorFlow platform, milling, heat-resistant steel.

INTRODUCTION

Heat-resistant steels and alloys are highly alloyed structural materials whose physical and mechanical characteristics are stable or only slightly change at high temperatures. These materials are highly resistant to chemical destroying in gaseous media, work well in unloaded and lightly loaded state [1].

The main features of the processing of heat-resistant steels and alloys are the high strengthening the material during deformation by cutting; low thermal conductivity of the processed material; the ability to maintain the original hardness and strength at elevated temperatures; reduce vibration resistance of the cutting movement [2].

The purpose of this research is to predict the machinability of heat-resistant steels during their milling under different cutting modes.

RESEARCH RESULTS

To achieve this objective, experimental researches of the milling process were carried out by determining the power spent on processing flat surfaces of workpieces made of heat-resistant steel 10X11H20T3P with milling cutters of various sizes, the cutting edge of which is made of BK8 hard alloy [2].

During research, the cutting depth t varied from 1 to 3 mm, the feed S - from 75 to 150 mm/min, and the cutting speed V – from 19.0 to 41.5 m/min. As a result of research, it was found that the power consumption during processing varied from 0.40 to 4.01 kW, depending on the cutting modes and the sizes of the milling cutter used. Processing of research results in order to model the milling process and subsequent prediction of cutting power was performed using the TensorFlow platform.

According to the using algorithm of TensorFlow [3], first we import the necessary libraries. The imported NumPy library is needed for representing data, and metrics – for computing errors.

```
import tensorflow as tf

import numpy as np

from sklearn import metrics
```

After that, a sequential model is created using the Keras add-in, which is a high-level neural network API written in Python and can work with TensorFlow, CNTK or Teano. It is designed with emphasis on quick experiments and the ability to go from idea to result with the least possible delay, which is the key to good research [4].

In Keras the model is usually represented as a graph of layers. The most common type of model is the layer stack: tf.keras.Sequential() model. Then layers adds (layers.Dense()). The construction of a simple multilayer perceptron is implemented in the following sequence:

```
model = tf.keras.Sequential()

model.add(tf.keras.layers.Dense(50, input_dim=5, activation='relu', bias_
initializer='glorot_normal'))

model.add(tf.keras.layers.Dense(50, input_dim=50, activation='relu', bias
_initializer='glorot_normal'))

model.add(tf.keras.layers.Dense(1, input_dim=50, activation='linear', bia
s_initializer='glorot_normal'))
```

Layers are configured as follows:

– activation. The activation function determines the output value of a neuron depending on the result of a weighted sum of inputs and a threshold value. In this example, ‘relu’ is used – this function returns x if x is positive, and 0 otherwise. ‘Relu’ is non-linear in nature and the combination of these activators is also non-linear (it is a good approximator, so any function can be approximated by a combination of Relu) [5]. ‘Linear’ is a function of linear activation. It uses inputs multiplied by the weights for each neuron and creates an output signal proportional to the input. This function is linear and the combination of these activators is also linear, therefore they are usually used alone [6];

– bias_initializer. Initialization schemes that create layer weights (core and shift). In our example, ‘glorot_normal’ was used – this is an initializer that draws samples from a truncated normal distribution centered at zero [7].

Next, configure the SGD optimizer – stochastic gradient descent. It takes three main parameters:

– learning_rate is a hyperparameter that is responsible for the speed of learning;

– momentum is a hyperparameter that accelerates SGD in the corresponding direction and dampens oscillations;

– nesterov is a parameter that indicates whether to use Nesterov's impulse, the main idea of which is that in case of an error, a correction is introduced.

```
sgd = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9, nesterov=
True)
```

After receiving the constructed model, the process of its learning is set up by calling the model.compile() method.

At the same time, model.compile() accepts three important arguments:

– optimizer. This object defines the training procedure. The instances of optimizer are passed to it. In our case, this is sgd, which is described earlier;

– loss. A function that is minimized in the learning process. One of the most common was used here: 'mean_squared_error' – mean square error;

– run_ragerly. A parameter that is passed to the compiler to be sure that the model is being trained and evaluated eagerly.

```
model.compile(optimizer=sgd, loss='mean_squared_error', run_eagerly=True)
```

Next, the user-defined callback function myCallback() was assigned to a variable. It is written to stop the learning process as soon as the desired accuracy is achieved.

```
callbacks=myCallback()
```

Model training was performed using the built-in fit() function. In addition to the input and target data, it takes two important arguments:

– epochs. Learning is broken down into epochs. The epoch is one iteration over all input data (performed in small batches);

– validation_data. When prototyping a model, it is possible track its performance on validation data. Passing a tuple of input data and labels with this argument allows the model to display the values of the loss function and metrics in output mode for the data being transmitted at the end of each era. Testing data is used, since training is not carried out on them.

```
model.fit(in_train, out_train, epochs=1200, callbacks=[callbacks], valida
tion_data=(in_test, true))
```

Next, data prediction is implemented using the predict() method, which takes input test data as an argument.

```
# make predictions on the testing set

out_pred = model.predict(in_test)
```

To compute the prediction errors, the user-defined function mean_error() was used, which takes as arguments the expected values and data that were predicted by the model at the same input.

```
mean_error(true, out_pred)
```

A fragment of the results of experimental research is presented in table 1. At the same time, the test data are marked in blue.

Table 1. A fragment of the results of experimental research.

D, mm	*B, mm*	*t, mm*	*S, mm/min*	*V, m/min*	*N, kW (research)*	*N, kW (predict)*
80	55	1	140	35	0,7	0.65
80	55	2	70	36	0,7	0.67
80	55	2	120	30,5	1,19	1.13
80	55	2	165	27,5	1,63	1.60
80	55	3	150	25	2,23	2.15
125	85	1	95	33	0,73	0.70
125	85	1	175	27	1,34	1.29
125	85	2	85	29	1,3	1.35
125	85	2	110	26	1,68	1.75
125	85	2	150	23,5	2,3	2.29
125	85	3	75	26,5	1,72	1.68
160	100	2	125	23	2,48	2.34
160	100	2	150	21	2,97	2.87
200	120	2	105	22	2,1	2.09
200	120	2	120	21	2,38	2.35
200	120	2	128	20	2,56	2.50
200	120	3	110	19	3,1	3.15

As a result of processing the research results using TensorFlow, the computing errors of the training data were determined:

MAE = 0.04457477154555144;

MSE = 0.0031432110126467815;

RMSE = 0.056064347072330926

and the computing errors of the test data:

MAE = 0.07362550377845761;

MSE = 0.006938523085845853;

RMSE = 0.08329779760501387.

CONCLUTIONS

The solution to the problem of modeling the process of milling flat surfaces of workpieces made of heat-resistant steel 10X11H20T3P with face mills of different diameters by measuring the power that is consumed during machining showed that creating a model using the TensorFlow platform based on the formation of artificial neural networks is an effective method for solving similar problems of modeling technological processes.

LIST OF USED REFERENCES

[1] Материаловедение и технология обработки конструкционных материалов в приборостроении: уч. пособ. / А.Н. Гормаков; Томский политехнический университет. – Томск: Издательство Томского политехнического университета, 2010. – 340 с.

[2] Соколова О. А. Моделювання процесу фрезерування деталей із жаростійкої сталі засобами штучних нейронних мереж. / О.А. Соколова, С.П. Вислоух // *Збірник наукових праць XV Всеукр. науково-практична конференція студентів, аспірантів та молодих вчених "Ефективність інженерних рішень в приладобудуванні"*, 10-11 грудня 2019 р. – К.: КПІ ім. Ігоря Сікорського. – 2019. – С. 172-175.

[3] Обзор Keras. — Режим доступа: www.URL:https://www.tensorflow.org/guide/keras/overview?hl=ru — 10.03.2020.

[4] Keras: The Python Deep Learning library. — Режим доступа:www.URL:https://keras.io/

[5] Функции активации нейросети: сигмоида, линейная, ступенчатая, ReLu, tahn. — Режим доступа:www.URL:https://neurohive.io/ru/osnovy-data-science/activation-functions/ — 29.11.2018.

[6] 7 Types of Neural Network Activation Functions: How to Choose? — Режим доступа:www.URL:https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/

[7] tf.keras.initializers.GlorotNormal — Режим доступа:www.URL:https://www.tensorflow.org/api_docs/python/tf/keras/initializers/GlorotNormal — 14.04.2020.

Academic adviser – PhD, associate professor S. Vysloukh