DOI: https://doi.org/10.33216/1998-7927-2019-253-5-87-91

UDC: 004.93'12

TOWARDS AN EMPIRICAL HYPERPARAMETERS OPTIMIZATION IN CNN

Siriak R.V., Skarga-Bandurova I.S., Biloborodova T.O.

ЗАБЕЗПЕЧЕННЯ ЕМПІРИЧНОЇ ОПТИМІЗАЦІЇ ГІПЕРПАРАМЕТРІВ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

Сіряк Р.В., Скарга-Бандурова І.С., Білобородова Т.О.

The necessity of creating a model of recognition of gestures based on convolutional neural network that effective not only in pattern recognition, but also in terms of learning speed and resource intensity, is substantiated. In this regard, the work solved the problem of optimization of hyperparameters and the selection of the best optimizer backpropagation errors. To implement the tasks, a model was created that can recognize hand gestures, both from a single image and from streaming video. When choosing an optimizer, two adaptive methods were tested - Adadelta and Adam. The experiments confirmed the high efficiency of Adadelta, however, when compared with Adam, it showed more than twice as long network training.
Keywords*: hyperparameter, convolutional neural network (CNN) , adaptive methods*

Introduction. Model optimization is one of the challenging tasks in the development and implementation of machine learning solutions. According to [1], hyperparameters are settings that can be configured to control the behavior of a machine learning algorithm. General speaking, hyperparameters are outside the model, but they are in a direct connection with it. A feature of the hyperparameters is that they are specific to the type of machine learning model that needs to be optimized. In some cases, the parameter is modeled as a hyperparameter, because it cannot be studied from the training set. A striking example of such a situation is the settings that control the capacity of the model (a set of functions that the model can represent). If the deep learning algorithm studies these parameters directly from the training set, then it will probably try to optimize this data set. As a result, it can lead to retraining the model. Examples of hyperparameters are the following: speed of model learning (a hyperparameter that can be used to optimize model capabilities); the number of hidden units (a key parameter for regulating the representative ability of the model); convolutional kernel width (the hyperparameter affects the number of parameters in the model, which, in turn, affects its throughput), etc.

The criteria for defining the hyperparameters are very abstract and flexible. Data science specialists usually spend a lot of time setting up hyperparameters to achieve the best performance for a particular model. In many cases, the hyperparameter optimization is considered as a global optimization of a black-box error function *f* whose evaluation is expensive [2]. Solving this problem is very challenging due to high complexity of the function *f* and depends on the current task.

In this context, our goal was to create a gesture recognition model based on a convolutional neural network (CNN), effective not only in pattern recognition but also in terms of learning speed and resource intensity. In this connection, the optimization problems of hyper-parameters and the selection of the best backpropagation optimizer were solved.

During the experiments, we used our own database containing six gestures: "fist", "one", "palm", "letterSH", "two", "zero." The total number of video files is 7674. The model was built using the Keras library and the TensorFlow framework. In the process of learning, preprocessing was carried out: the image of the hand was segmented and transferred to the grayscale format, after which the Canny edge detector was used to highlight the hand contours. The processed images were normalized and fed to the input of CNN with three coagulation layers and three fully connected. The last layer was the Softmax classifier whose number of neurons is equal to the number of object classes. When choosing an optimizer, two adaptive methods were tested - Adadelta and Adam.

Experimental Setup. CNN solves two problems: (1) determining the attributes of objects (2) determining the probability of the object belonging to a particular class.

In the convolutional core layer, a linear transformation is applied to each pixel of the image, revealing the characteristic features of this class of images. The kernels are matrices of weights of odd sizes, whose val-

ues are set so that any spatial or color features can be detected. At the output, we have feature maps, whose number is equal to the number of applied convolution kernels. Convolution can be presented as

$$(I * K)_{ij} = \sum_{a=0}^{m-1}\sum_{b=0}^{m-1} \omega_{ab} y_{(i+a)(j+b)}^{l-1}, \quad (1)$$

where ω is the core of size m × m, y are the inputs from the previous layer, f is the activation function of neurons.

The size of the output from the convolution layer is calculated by the formula:

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor, \quad (2)$$

where n_{in} is the size of the input data from the previous layer, p - padding, k is the size of the core, s is the step size with which the core is shifted in the image.

Convolution kernels form feature maps equal to their number. In our case, three convolutional layers were used, which yielded 16, 32, and 64 feature maps, respectively. The use of three convolutional layers seems to be optimal in terms of the quality of training to a small number of parameters. The number of parameters for a convolutional layer is defined as:

$$p = d * k * w + b, \quad (3)$$

where d is the depth of the input data (the number of channels), k is the core, w is the weights, b is bias. For example, for the first convolutional layer, this will look like, that is, 160 parameters. For the second layer, that is 4640 parameters.

The result of the convolution must be passed through the nonlinear activation function. Nonlinearity can be written as an expression:

$$y_{ij}^{l} = f(x_{ij}^{l}). \quad (4)$$

In our case, the activation function was the Rectified Linear Unit ReLU [3]:

$$f(x_i) = \begin{cases} x_i, & x_i > 0 \\ a_i x_i, & x_i \le 0 \end{cases}, \quad (5)$$

where x_i is the input data of the i-th channel, a_i is the coefficient controlling the slope at negative values. ReLU is significantly superior to other functions in the gradient damping resistance, and the learning rate according to [4] is six times faster than the hyperbolic tangent.

The feature maps formed on each layer are fed to the max-pooling sub-sampling layer.

The max-pooling layer with a 2x2 filter follows each convolution layer, enhances the features identified on the previous layer and reduces the dimension of the input data and, as a result, the number of parameters. Max-pooling is calculated as:

$$y^{l+1} = \max_{0 \le i < H, 0 \le j < W} x^{l}_{i^{l+1} \times H + i, j^{l+1} \times j, d}, \quad (6)$$

where H and W are the height and width of the downsampling filter, x is the input data.

The size of the output from the max-pooling layer is defined as:

$$n_{out} = \left\lfloor \frac{n_{in} - k}{s} \right\rfloor + 1. \quad (7)$$

Max-pooling has no options for training. In order to prevent retraining, in which the trained model too closely matches a given data set, losing the ability to generalize, a dropout is used in the developed neural network [5]. Dropout enables to exclude random neurons in a certain layer with a predetermined probability. The excluded neuron returns the value 0. The probability that a neuron of the current epoch will remain in the network is $q = 1 - p$.

This approach does not allow the neural network to simply remember the correct answers, but gives flexibility in recognizing patterns that are not included in any of the training datasets. Individual nodes are excluded from the network with an established probability of 0.25, that is, at each iteration 25 percent of random neurons will be excluded from the operation of the network.

The second part of the created network is a multilayered perceptron and is intended for training the hand gesture classifier on the identified features. Data from the last down sampling layer is fed to the Flatten layer, thereby transforming into a one-dimensional vector. Since 64 1313-size trait cards are received from the Flatten input, the first layer of the fully connected part of our network consists of 10816 neurons.

The next layer is fully connected and contains 256 weights, which is quite enough to solve the problem, without requiring a lot of resources. As a result, the number of parameters on this layer is 10816,256 + 256 = 2769152 parameters.

The calculation of the values of neurons for a fully connected layer occurs by the formula:

$$x_i^l = \sum_{k=0}^{m} w_{ki}^l y_k^{l-1} + b_i^l, \quad (8)$$

where w_{ki}^l is the weight from the k-th neuron of the l-1 layer to the i-th neuron of the current layer l, b is the offset of the current layer, y is the incoming data from the previous layer.

The last layer of the network with the number of outputs equal to the number of recognized categories, implements the softmax activation function.

Softmax assigns a value, represented by a non-negative real number, to each class, expressing the probability of belonging:

$$S_i = P(y = i \mid a), \tag{9}$$

where y is the output class numbered as 1...n, a is a vector of dimension n.

The sum of all output signals expressing the probability of belonging is equal to one. The output value of the i-th neuron corresponds to the probability that the correct answer is i:

$$S_i \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}, \tag{10}$$

where n is the number of classes. The value of S_i is always positive and is in the range (0, 1).

The created CNN consists of two parts: a) three blocks of alternating convolutional and subsampling layers that form the input feature vector for learning; b) three fully connected layers.

Convolutional layers apply a 3×3 convolution kernel and form 16, 32, and 64 feature maps, respectively. Feature maps pass through subsampling layers with a max-pooling of 2×2, each time halving the data dimension.

The classifying part of the network consists of a flatten layer, each node of which corresponds to one value of the feature vector, and two fully connected layers dense. The last layer is the output and implements the softmax function.

CNN training. Network training is based on the backpropagation algorithm. Backpropagation can be viewed as a differentiation of a complex function with the search for derived loss with respect to variables. The loss function is categorical cross-entropy [6]:

$$H(y, \hat{y}) = -\sum_i y_i \log \hat{y}_i = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}), \tag{11}$$

where y is the desired result, $\hat{y}$ is the actual result, between which it is necessary to minimize the cross entropy. For a training sample of size m, the minimization of the cross entropy between the predicted and real values is:

$$E(w) = -\frac{1}{m} \sum_{i=1}^{m} [y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)], \tag{12}$$

where m is the number of classes. In case the output signal is close to the expected one, the value of the loss function is close to zero:

$$\lim_{\hat{y} \to y} E = 0. \tag{13}$$

Depending on the results of calculating the loss function, the parameters change at each iteration as:

$$w_{ij}^{l} = w_{ij}^{l} - \alpha \frac{\partial}{\partial W_{ij}^{l}} E(W, b), \tag{14}$$

where α is the learning rate.

Changing the values of weights will be carried out through the gradient descent:

$$\Delta w_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}}. \tag{15}$$

Thus, the task of training the classifier S is to minimize the loss function in the space of weights:

$$\min_{W} E(S(X, W)M), \tag{16}$$

where X is the feature space, W is the network weights, M is the set of classes. Based on this, it is necessary to calculate the loss function gradient

$$\nabla E(S(X, W)M). \tag{17}$$

Choosing an optimizer. When choosing an optimizer, two adaptive methods were tested - Adadelta [8] and Adam [7].

Adadelta is essentially an extension of another AdaGrad optimizer [9], which had a problem with reducing learning speed. The problem arose due to the accumulation of the sum of squares of gradients, as can be seen from the AdaGrad formula itself:

$$w_{N+1} = w_N - \frac{\alpha}{\sqrt{G_N + \varepsilon}} g_N, \tag{18}$$

where w_N is the value of the parameter w in step N, G_N is the diagonal matrix containing the sum of squares of updates of the partial derivatives of the parameter w from the beginning of work to N, is the smoothing parameter for avoiding division by zero.

In Adadelta, instead of the total sum of updates, the averaged square of the gradient is used, for which the exponentially decaying running average is used. The exponentially average is calculated as:

$$E[g^2]_N = \gamma E[g^2]_{N-1} + (1 - \gamma) g_N^2, \tag{19}$$

where γ is the attenuation coefficient in time.

Due to the mean of the squares of the gradients looks like

$$RMS = \sqrt{E[g^2]_N + \varepsilon} \tag{20}$$

this rule of weights update for Adadelta takes the form:

$$w_{N+1} = w_N - \frac{RMS[\Delta w]_{N-1}}{RMS[g]_N} g_N. \quad (21)$$

The second tested optimizer of gradient descent was the Adam adaptive inertia method. It calculates adaptive learning rates based on the first and second gradient moment scores. The estimate of the first moment is calculated from the previously obtained partial derivative values, as a moving average of gradients. The estimate of the second moment is calculated on the basis of the squares of the gradients of the values of the latter values for the weight

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t,$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$$

where β_1 is the exponential decay rate for the first moment, β_2 is the exponential decay factor for estimating the second moment. Due to the fact that the moment estimates are initialized with zeros, and the values of the coefficients β_1 and β_2 are recommended by the authors close to one (0.9 and 0.999, respectively), the shift to zero is preserved. To prevent this, to change the parameters use:

$$\hat{m}_p = \frac{m_p}{1-\beta_1^p},$$
$$\hat{v}_p = \frac{v_p}{1-\beta_2^p}.$$

Recalculation of parameters is made according to the formula:

$$w_p = w_{p-1} - \frac{\alpha}{\sqrt{v_p + \varepsilon}} m_p, \quad (22)$$

where $\varepsilon = 10^{-8}$ is introduced to prevent possible division by zero. In our case, the learning rate $\alpha = 0.002$. A significant increase in speed reduced the effectiveness of network training and increased the number of incorrect answers. Decreasing speed made the network long-learning.

The experiments performed show a sufficiently high Adadelta performance, however, when compared with Adam, it showed more than twice the network's long learning ability.

Having error values, we calculate the partial derivative of the objective function E with respect to each output of the neuron. To calculate the loss derivatives for the variables in the embedded equation, the chain rule is applied:

$$\frac{\partial E}{\partial x_{ij}^l} = \frac{\partial E}{\partial y_{ij}^l} \frac{\partial y_{ij}^l}{\partial x_{ij}^l} = \frac{\partial E}{\partial y_{ij}^l} \frac{\partial}{\partial x_{ij}^l} (f(x_{ij}^l)) = \frac{\partial y_{ij}^l}{\partial x_{ij}^l} f(x_{ij}^l)$$

For a convolutional layer, the backpropagation procedure looks like:

$$\frac{\partial E}{\partial y_{ij}^{l-1}} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \frac{\partial E}{\partial x_{(i-1)(j-b)}^l} \frac{\partial_{(i-1)(j-b)}^l}{\partial y_{ij}^{l-1}} =$$
$$= \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \frac{\partial E}{\partial x_{(i-1)(j-b)}^l} w_{a,b}.$$

In the max-pooling layer, an error from the previous layer passes through a single maximum value. Since this layer does not train the network, the error passes through it unchanged.

For the softmax layer, it is necessary to differentiate the loss function J with respect to z_i:

$$\frac{\partial J}{\partial z_i} = -\sum_k y_k \frac{\partial \log \hat{y}_k}{\partial z_i} = -\sum_k y_k \frac{1}{\hat{y}_k} \frac{\partial \hat{y}_k}{\partial z_i} =$$
$$= -y_i(1-\hat{y}_i) - \sum_{k \neq i} y_k \frac{1}{\hat{y}_k} (-\hat{y}_k \hat{y}_i) = -y_i(1-\hat{y}_i) + \sum_{k \neq i} y_k (\hat{y}_i) =$$
$$= -y_i + y_i \hat{y}_i + \sum_{k \neq i} y_k (\hat{y}_i) = \hat{y}_i (\sum_k y_k) - y_i = \hat{y}_i - y_i$$

The experiments confirmed the high efficiency of Adadelta, however, when compared with Adam, it showed more than twice as long network training. As a result of experiments on a test subset with a size of 1,535 images, recognition accuracy of 94% was achieved.

Conclusion. In this work, we performed optimization hyperparameter manually that is exhausting and unexpanded. To address these challenges, the algorithms such as Grid Search and Random Search that automatically infer a potential set of hyperparameters and attempt to optimize them could be used. In the course of testing with gesture recognition, a high degree of correct network responses was obtained in real-time on a webcam, including people who did not participate in the creation of an image database. Future work will focus on the development of the effective method for automatic optimizing hyperparameters of deep learning algorithms.

References

1. Rodriues J. Understanding Hyperparameters Optimization in Deep Learning Models: Concepts and Tools https://towardsdatascience.com/understanding-hyperparameters-optimization-in-deep-learning-models-concepts-and-tools-357002a3338a. 2018.
2. Ilievski I., Ahkar T., Feng J., Shoemaker Ch.A. Efficient Hyperparameter Optimization of Deep Learning Algorithms Using Deterministic RBF Surrogates. Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17). P. 822-829. 2017.
3. Nair, Vinod, and Geoffrey E. Hinton. "Rectified linear units improve restricted boltzmann machines." Proceedings of the 27th international conference on machine learning (ICML-10). 2010.
4. Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural

networks." Advances in neural information processing systems. 2012.
5. Hinton, Geoffrey E., et al. "Improving neural networks by preventing co-adaptation of feature detectors." arXiv preprint arXiv:1207.0580 (2012).
6. Kullback, Solomon, and Richard A. Leibler. "On information and sufficiency." The annals of mathematical statistics 22.1 (1951): 79-86.
7. Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).
8. Zeiler, Matthew D. "ADADELTA: an adaptive learning rate method." arXiv preprint arXiv:1212.5701 (2012).
9. Duchi, John, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." Journal of Machine Learning Research 12.Jul (2011): 2121-2159.

Сіряк Р.В., Скарга-Бандурова І.С., Білобородова Т.О. Забезпечення емпіричної оптимізації гіперпараметрів згорткової нейронної мережі

Обґрунтовано необхідність створення моделі розпізнавання жестів на основі згорткової нейронної мережі, ефективною не тільки в точності розпізнаванні зображень, але й з точки зору швидкості навчання і використання обчислювальних ресурсів. У зв'язку з цим в статті представлено вирішення завдачі оптимізації гіперпараметрів і вибору найкращого оптимізатора помилок зворотного поширення. Для реалізації завдання створена модель, здатна розпізнавати жести рук як по одному зображенню, так і по потоковому відео. При виборі оптимізатора були протестовані два адаптивних методу - Adadelta і Adam. Експерименти підтвердили високу ефективність Adadelta, однак, у порівнянні з Adam, він показав вдвічі більший час навчання мережі.

Ключові слова: *гіперпараметр, згорткова нейронна мережа, адаптивні методи*

Сиряк Р.В., Скарга-Бандурова И.С., Белобородова Т.А. Обеспечение эмпирической оптимизации гиперпараметров сверточной нейронной сети

Обоснована необходимость создания модели распознавания жестов на основе сверточной нейронной сети, эффективной не только в точности распознавании изображений, но и с точки зрения скорости обучения и использования вычислительных ресурсов. В связи с этим в статье представлено решение задачи оптимизации гиперпараметров и выбора наилучшего оптимизатора ошибок обратного распространения. Для реализации задачи создана модель, способная распознавать жесты рук как по одному изображению, так и по потоковому видео. При выборе оптимизатора были протестированы два адаптивных метода - Adadelta и Adam. Эксперименты подтвердили высокую эффективность Adadelta, однако, по сравнению с Adam, он показал вдвое большее время обучение сети.

Ключевые слова: *гиперпараметр, сверточная нейронная сеть, адаптивные методы.*

Сіряк Ростислав Вікторович – здобув. кафедри комп'ютерних наук та інженерії Східноукраїнського національного університету імені Володимира Даля, e-mail: hashem.r@gmail.com

Скарга-Бандурова Інна Сергіївна – д.т.н., професор, зав. кафедри комп'ютерних наук та інженерії Східноукраїнського національного університету імені Володимира Даля, e-mail: skarga-bandurova@snu.edu.ua

Білобородова Тетяна Олександрівна – к.т.н., ст.викладач кафедри комп'ютерних наук та інженерії Східноукраїнського національного університету імені Володимира Даля, e-mail: beloborodova.t@gmail.com

Стаття подана: 05.07.2019.