Arcada Working Papers 3/2021
ISSN 2342-3064
ISBN 978-952-7365-12-0

Fusing Extreme Learning Machine with Convolutional Neural Network

Anton Akusok, Kaj-Mikael Björk

www.arcada.fi

Arcada Working Papers 3/2021
ISSN 2342-3064
ISBN 978-952-7365-12-0

Fusing Extreme Learning Machine with Convolutional Neural Network

Anton Akusok[i], Kai-Mikael Björk[i]

Sammandrag / Abstract

Practical image analysis with pre-trained networks faces a problem with the large size of the extracted image features that need to be saved as temporary training data. It reaches terabytes, complicating storage of movement of such data. Extreme Learning Machine classifier/regressor method has an internal formulation that is invariant to the number of training data samples.

This paper presents the theory and an example implementation of fusing Extreme Learning Machine processing into a pre-trained convolutional neural network. The resulting solution is compact, but still allows for tuning of the important hyperparameters being the number of hidden neurons and L2 regularization.

Keywords: extreme learning machine, convolutional neural network, mxnet

1 INTRODUCTION

Convolutional Neural Networks (CNN) are de-facto image feature extractors for any kind of automatic processing. Their pre-trained versions are indispensible in AI applications, as they allow researchers to quickly develop machine learning pipelines for solving various problems (Espinosa-Leal, 2021). These versions are shared at code sharing websites like Github, published in attachments to research papers, or are even included in software libraries like Keras pre-trained models.

Practical development with pre-trained CNNs at Arcada discovered a complication of very large data size for the extracted image features – due to their high dimensionality

[i] Arcada University of Applied Sciences, Helsinki, Finland, anton.akusok@arcada.fi

and dense random-like nature that prevents efficient compression. Temporary data size is often expressed in terabytes, causing problems with storing and moving such data across.

The purpose of the image features data is to build a specific classifier or regressor with image data as input. Extreme Learning Machines (Huang, 2006) (ELM) is a versatile classifier and regressor method, with a wide range of existing applications (Cambria, 2013). Most important, this method internally reduces training data to a temporary object whose size is limited and unaffected by the number of training samples.
This paper investigates the practical way of fusing the first part of ELM data processing into the pre-trained CNN. This would allow for saving of a relatively small data object in place of terabytes of temporary data; without affecting the numerical performance of the final classifier or regressor. Furthermore, due to ELM working, it is even possible to adjust L2-regularization or model size (Akusok, 2015), on the created temporary object.

2 METHODOLOGY

ELM method uses fixed random non-linear transformation, followed by a least squares linear system solver. The original problem is formulated as:

$$\mathbf{H}\beta = \mathbf{Y}$$

Here matrix **H** consists of hidden layer representation for all data samples. An alternative representation that gives equal solution is:

$$(\mathbf{H}^T\mathbf{H})\beta = \mathbf{H}^T\mathbf{Y}$$

Here matrix ($\mathbf{H}^T\mathbf{H}$) is a square matrix with size equal to the number of hidden neurons. Its size remains constant with any number of training data samples – partial matrices $(\mathbf{H}^T\mathbf{H})^*$ from new training data batches are added to the original one. The right-hand side matrix ($\mathbf{H}^T\mathbf{Y}$) has size of the number of hidden neurons times number of outputs, and normally is smaller than matrix ($\mathbf{H}^T\mathbf{H}$). It's size also does not change with the amount of training data.

The constant size of matrices ($\mathbf{H}^T\mathbf{H}$), ($\mathbf{H}^T\mathbf{Y}$) with respect to the amount of training data solves the problem of large intermediate data storage for the extracted image vectors. For this, the random projection layer of ELM and the outer product layer are added to the convolutional neural network:

1. $\mathbf{H} = tanh(\mathbf{X}\mathbf{W})$ where **W** is a fixed random projection matrix
2. $\mathbf{A}^* = \mathbf{H}^T\mathbf{H}$ partial output matrix **A** for the current data batch
3. $\mathbf{B}^* = \mathbf{H}^T\mathbf{Y}$ partial output matrix **B** for the current data batch
4. $\mathbf{A} \leftarrow \mathbf{A} + \mathbf{A}^*;\ \mathbf{B} \leftarrow \mathbf{B} + \mathbf{B}^*$ accumulate matrices from training data batches
5. $\mathbf{A}\beta = \mathbf{B}$ compute solution for ELM problem on whole dataset

Step (1) has straight-forward implementation as a dense CNN layer followed by a non-linear function layer. Steps (2) and (3) are matrix product layers multiplying the corresponding matrices. Step (4) can be implemented as an accumulator layer, or by manually accumulating batch results in memory.

The last step (5) is an ELM solution step that should be implemented separately to correctly handle L2-regularization term. This term is simply a constant added to the diagonal of an **A** matrix before the solution (Akusok, 2015). Different L2 parameter values give different solutions; the best value is found with a validation set.

Number of hidden neurons is another important parameter of an ELM. It is adjusted by taking the upper right corner of matrix **A** as a smaller matrix **A'** corresponding to fewer hidden neurons. A similar smaller matrix **B'** is taken as an upper part of whole matrix **B**.

Both L2-regularizaiton and the optimal smaller number of hidden neurons can be computed from the existing matrices **A** and **B** without re-creating training data. Effectively, this means the training procedure is feasible without ever storing the extracted image features that are found to take terabytes of space in practical experiments.

3 IMPLEMENTATION WITH MXNET FRAMEWORK

MXNet is one of the major Deep Learning frameworks used for training and running inference of CNNs. The particular model in the experiment is Inception21k, a pre-trained image classification model based on one of the best architectures called "Inception" but with 21,000 classes of images contrary to the typical 1,000 classes.

There are different ways of operating a CNN model in a framework; one of them is symbolic programming that produces very clean code. Image features of the pre-trained model are accessed as:

```
sym, arg_params, aux_params = mx.model.load_checkpoint('./wibc/wibc/data/Inception21k', 9)
X = sym.get_internals()['flatten_output']
```

Here symbol `X` refers to a vector of image features, reshaped ("flattened") from a 3D internal representation. We need two new symbols to fuse ELM computations with a pre-trained model:

```
w = mx.sym.Variable('W')
Y = mx.sym.Variable('Y')
```

With those symbols, steps (1), (2) and (3) of the methodology are added to the CNN model:

```
XW = mx.sym.FullyConnected(X, weight=w, num_hidden=L, no_bias=True)
H = mx.sym.tanh(XW)
HH = mx.sym.linalg_syrk(A=H, transpose=True, name='HtH')
HY = mx.sym.dot(lhs=H, rhs=Y, transpose_a=True)
```

One optimization is the use of symmetric rank-k update ("syrk") function instead of a dot product. Matrix ($\mathbf{H}^T\mathbf{H}$) is symmetric by definition, so computing every single element of it would waste computational resources. "Syrk" function computes only one triangle of a symmetric matrix, the other triangle is then copied from the computed one.

Model runtime object is created with two output targets, and with two inputs as well – now data labels **Y** are needed in addition to raw image pixel data.

```
modX = mx.mod.Module(symbol=mx.sym.Group([HH, HY]),
                     data_names=['data', 'Y'],
                     label_names=None, context=context)
```

Finally, fixed hidden matrix W is added to pre-trained model parameter, in correct format:

```
w_data = mx.nd.array(W)
arg_params['W'] = w_data
modX.set_params(arg_params, aux_params)
```

Finally, partial batch results are accumulated in MXNet array objects, because the symbolic interface does not seem to support between-batch accumulator operations. For this purpose, the model is wrapped in a Python class:

```
self.model.forward(Batch([mx.nd.array(data), mx.nd.array(y)]))
HH0, HY0 = self.model.get_outputs()
self.HtH += HH0
self.HtY += HY0
```

4 CONCLUSIONS

This paper proposes a solution to the practical problem of large temporary data size in image content analysis with pre-trained convolutional neural networks. The proposed solution uses the internal workings of an Extreme Learning Machine, in particular its formulation that is invariant to the number of training data samples.

A hands-on section presents a practical implementation of this method with MXNet deep learning framework, that is conseptually simple and requires little coding. The modified network accumulates data in two fixed-size matrices, instead of generating more image features with each training data batch. Furthermore, the system uses a more efficient operation for symmetric matrix computing, saving up to half of its runtime depending on internal implementation.

REFERENCES

1. Akusok, Anton, Kaj-Mikael Björk, Yoan Miche, and Amaury Lendasse. "High-performance extreme learning machines: a complete toolbox for big data applications." *IEEE Access* 3 (2015): 1011-1025.

2. Cambria, Erik, Guang-Bin Huang, Liyanaarachchi Lekamalage Chamara Kasun, Hongming Zhou, Chi Man Vong, Jiarun Lin, Jianping Yin et al. "Extreme learning machines [trends & controversies]." *IEEE intelligent systems* 28, no. 6 (2013): 30-59.

3. Espinosa-Leal L., Akusok A., Lendasse A., Björk KM. (2021) Extreme Learning Machines for Signature Verification. In: Cao J., Vong C.M., Miche Y., Lendasse A. (eds) Proceedings of ELM2019. ELM 2019. Proceedings in Adaptation, Learning and Optimization, vol 14. Springer, Cham.

4. Huang, Guang-Bin, Qin-Yu Zhu, and Chee-Kheong Siew. "Extreme learning machine: theory and applications." *Neurocomputing* 70, no. 1-3 (2006): 489-501.