Optimization on Integration Model of ConvNet and CasperNet for the EMNIST Classification

Hao He[1]

[1] ANU College of Engineering & Computer Science, Australian National University.
108 North Rd, Acton ACT 2601, Australia
{u6153769}@anu.edu.au

Abstract. The model of his paper integrates convolutional network and Casper Network for image classification of EMNIST dataset. The highest accuracy is 86.75% with 10 epochs and 15 hidden neurons. The performance of the model is better than the 78.02%±0.09% benchmark. The convolutional networks include LeNet-5 and VGG-16. One of them can be used while training. The Casper networks include Casper tower network and cascade fully connective mode. The two modes can be used separately or simultaneously during training. The Casper network can be substituted by fully connective layer. The test results show that Rprop is not applicable for the batch training. Also, the batch training is the constraints of the Casper network.

Keywords: Casper, Cascade Correlation, Convolutional Neural Network, EMNIST, Image Classification, Image Identification, Neural Network Optimisation

1 Introduction

A Cascade network algorithm employing Progressive RPROP (Casper) is introduced in [1]. It is similar to Cascade-Correlation (Cascor) architecture which is a supervised learning algorithm for artificial neural networks [2]. The Cascor starts from a simple network from inputs to outputs and adds one neuron at a time after the previously optimized weights fixed [2], while the Casper keeps updating the weights after the neurons are added [1]. Although some researches indicate that Cascor network may have problems in classification and regression work because of the frozen weights[3], the maximum correlation criterion that generates undesirable saturated hidden units [4] and the worse generalization of cascading of the hidden units [5], it can learn quickly and determine its own topology [2]. Due to its pros, it will have a broad application prospect in the future
The MNIST dataset, containing only handwritten digits of 10 classes, is regarded as a standard benchmark for learning and classification [6] which is derived from NIST Special Database 19 [7], while the Extended MNIST (EMNIST) consists of not only handwritten digits but also uppercase and lowercase letters. So the classification task of EMNIST is much more challenging. There are many methods about the classification of the MNIST dataset, and most of them achieved less than 1% of error rate, some published accuracies above 99.7% [8-15]. However, there is not much research on the classification of EMNIST so far. Compared with some other structures of neural networks, for example, fully connected artificial neural networks and convolutional neural network, Casper network structure does not seem to be so competitive for the image classification, and there are not many published works about the classification of MNIST and EMNIST by Casper network (CasperNet).
To optimize the performance of the model, deep learning is necessary. As is shown in [16] Fig. 3 and [17] Fig. 5, the distribution of the extracted features of MNIST, as well as EMNIST, are nonlinear. The deeper neural network can increase the nonlinearity. So the convolutional networks (ConvNets) are used to extract and discriminate features because of its great success in image recognition [18]. The comparison is conducted between the Fully-Connected (FC) layers and two kinds of Casper layers: one is tower mode illustrated in [1, 19] and the other is without cascading hidden neurons mode introduced in [14].

2 Method

The original model is a simple model with inputs, outputs, and an adaptive Casper tower. The optimized model greatly extends the original model. Different datasets, ConvNets, hidden layers are integrated into the model in order to better control the single variable to achieve the best contrast and to make the optimization more clear.

2.1 Datasets

In this paper, EMNIST Balanced Dataset and MNIST dataset are used for the classification task. EMNIST Balanced Dataset comprises 131,600 Samples, 47 Classes including ten digits and 37 merged characters of certain uppercase and lowercase letters, and each class has the same number of samples [6]. Each sample in EMNIST dataset has the same 28

×28 resolution as MNIST. MNIST is used for comparison because it is much smaller, easier and faster to train and test. The configuration of the model for MNIST is usually compatible with EMNIST. Therefore the optimization on MNIST is a good guide to EMNIST.

2.2 Original Model

The original model has followed the instruction of [1, 20]. It is a Casper network starts from a 28×28 features input layer and a 47 classes output layer with 28×28×47 adjustable weights and 47 adjustable biases. After the weights between the input and output layer are optimized, the network brings a candidate units connected with each input unit and each output unit. When all the weights of the candidate unit are optimized, the candidate unit is added to the network. Similarly, the network brings a new candidate unit that connects each input and output unit as well as the hidden unit added previously. In this way, it dynamically creates the hidden unit once at a time to determine the size and depth of the network. An illustration is shown in Fig. 1a, which is a Casper network model of the tower with three input units, two output units, a hidden neuron h_1, and a candidate neuron h_2.
For the optimization, weights between input units and output units, input units and candidate/hidden unit, candidate/hidden unit, and output units, candidate unit, and hidden unit are separately updated with different initial learning rates after each batch data training. The learning rates of candidate units are initialized relatively higher than the hidden units.

2.2.1 Architecture

The model starts from a minimal network without any hidden layer. 1×1 convolution filter is utilized which can be seen as a linear transformation of the input images. After maximized the correlation between inputs and outputs, Casper tower is created and extended by the correlation. The topology is shown in Fig. 1c.

2.2.2 Configurations

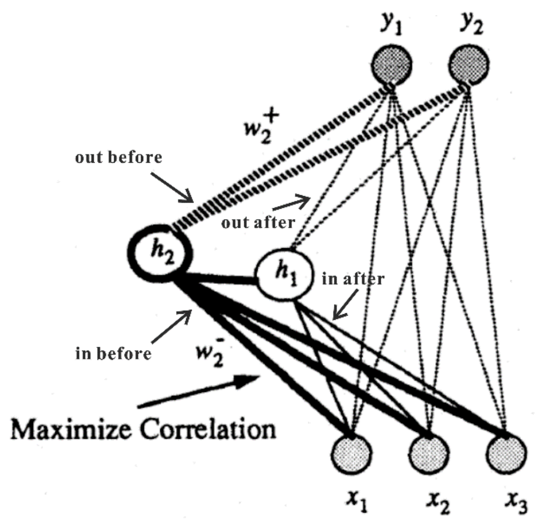
The activation function of the original model is Rectified Linear Units (ReLU). The optimizer is Rprop with 4 initial learning rates: Lr_in_before 0.01, Lr_out_before 0.005, Lr_in_after 0.001 and Lr_out_after 0.001 (see Fig. 1). The neuron is added to the net after a number of steps according to a threshold that is set empirically to get a maximized correlation, e.g., 2000.

2.3 Optimised Model

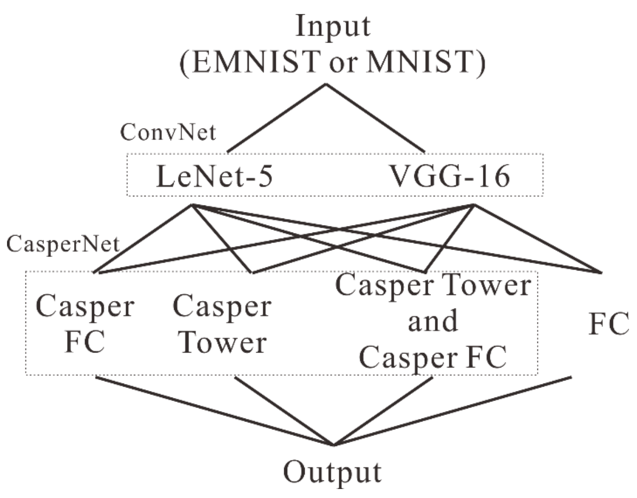
The optimized model is tested and changed from the original model step by step. The architecture of the original model is included in the optimized model, and the configurations, such as the activation function and optimizer, are different.

2.3.1 Architecture

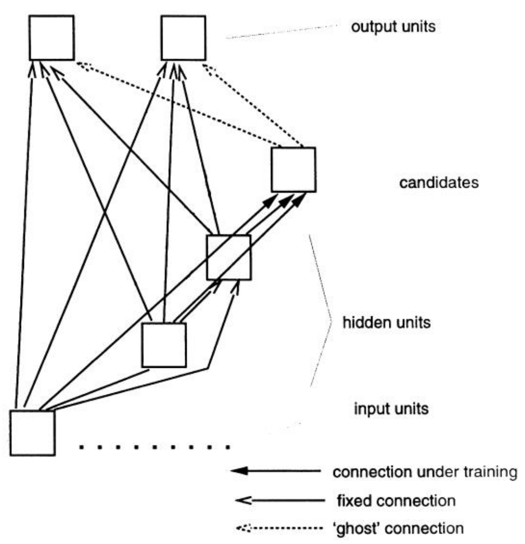
The optimized model consists of ConvNets and CasperNet (see Fig. 1b). There are two types of ConvNets, LeNet-5 and VGG-16. During the training, only one of them can be implemented. There are two types of CapserNets and, Casper tower (CT) shown in Fig. 1c and cascading without hidden neurons shown in Fig. 1d. The structure of cascading without hidden neurons can be considered as the fully connective hidden layer which can adaptively add new neurons, so in this paper, this type of CasperNet is denoted by Casper FC (CFC). These two types of CasperNet can be used separately or simultaneously. The optimized model allows adding several hidden neurons at one time to the Casper net. In addition, the CasperNet can be substituted for a fixed fully connective layer (FC).



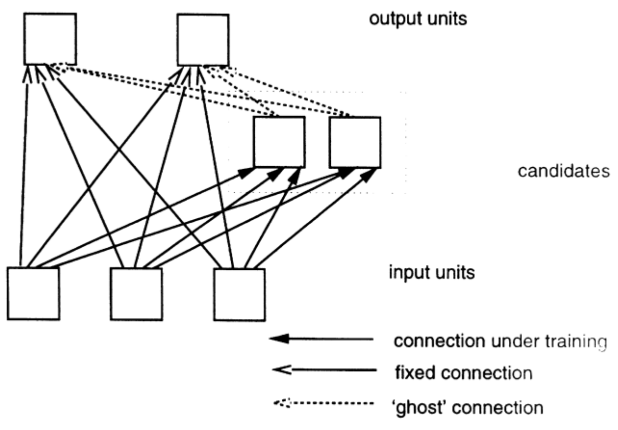
a



b



c



d

Fig. 1a. The network structure of Casper. The second unit h_2 is the candidate unit which is to be added after the optimization; the unit h_1 is the first hidden unit. The connection between inputs and neurons is denoted by "in", between neurons and outputs is denoted by "out", the connections of the neuron before added to the network is denoted by "before", and the connections of the neuron after added to the network is denoted by "after".This figure is modified from [4] Fig. 2. **Fig. 2b.** The topology of the optimized model including ConvNet and CasperNet. **Fig. 1c.** CT topology. With two hidden neurons and one candidate neutron in training. Each new neuron is connected to the previous one which increases the depth of the net. This figure is modified from [14] Fig. 4. **Fig. 1d.** CFC topology. The network is adding two neurons at a time. This figure is Modified from [14] Fig. 2.

2.3.2 Configurations

The ConvNet configurations in this paper are based on [15] and [18], which are outlined in Table 1. The configurations are nearly the same, except that fewer convolutional filters are used in the model to save time and memory, and the channel is only one instead of 3 channels of RGB. The width of conv3 layers starts from 8 and increasing by a factor of 2 until 64 instead of 64 to 512 because of the memory limitation of GPU. The max-pooling is applied only after the 2nd, 4th, 7th conv3 layer, because of the limited size of EMNIST and MNIST images. The convolution stride is fixed to 1 pixel, and spatial resolution is preserved after convolution by zero-padding.
The CasperNet configurations are different from the original model. The default stage is 3 with 5 neurons added at a time, because in this setting, the network is more stable than under the setting of some larger arguments. The default number of FC layer is 15 correspond to stage 3 of CT and CFC. Empirically, 800 neurons for FC is the best.

Activation Function

The ReLU is applied to each convolutional layers of ConvNet to keep consistent with [15] and [18], and the RReLU is applied to all layers of CasperNet, The improved version of ReLU can avoid the drawbacks of ReLU, such as the Parametric Rectified Linear Units (PReLU) and randomized leaky rectified linear units (RReLU). According to [21, 22], PreLU is better than LReLU and ReLU. However, RReLU is the best especially using the Rprop optimiser. In addition, some research indicates that the performance of RReLU is the best, which, to some extent, can also provide a regularization effect to prevent overfitting due to its characteristic of randomized [22].

Table 1. ConvNet configurations. The left column is LeNet-5 and the right is VGG-16. The convolutional layer parameters are denoted the same as [18] by "conv(receptive field size)-number of channels".

	LeNet-5	**VGG-16**
	2 weight conv layers	13 weight conv layers
	Input (28×28 image)	
ConvNet layers	conv5-32 maxpool, ReLU conv5-64 maxpool, ReLU	conv3-8 conv3-8 maxpool, ReLU conv3-16 conv3-16 maxpool, ReLU conv3-32 conv3-32 conv3-32 maxpool, ReLU conv3-64 conv3-64 conv3-64 ReLU conv3-64 conv3-64 conv3-64 ReLU
CasperNet layers	CT-5-stage3 and/or CFC-5-stage3 or FC-800	
	Log-soft-max	

Adaptive Optimiser

Adam optimizer is used in the optimized model instead of Adadelta and Rprop. The default initial four learning rates are 0.001, 0.001, 0.0005, 0.0005. Regularization is implemented in CasperNet with 10^{-5} weight decay rate for the candidate neurons, and 2×10^{-5} for the added neurons. Adam optimizer is used has a better performance than Adadelta and Rprop with much faster convergence [23].
Among different adaptive learning rate algorithms, e.g., AdaGrad, AdaDelta, RMSprop, Adam is the refined version for all of them. RMSprop is the upgraded version of Rprop. It gives each weight a mean square variable to record the average of the gradient squares of the first n times and then divides the nth gradient by the variable to determine the step [24]. It avoids the drawback of Rprop that determined the step merely by the sign of gradients. Adadelta is upgraded from Adagrad, which calculate different learning rates for each parameter. Adam is the refinement of RMSprop and Adadelta [23-26].

The Threshold for Adding a Neuron

In the model, 1800 is frequently used as the threshold. The neurons are added by the correlation, and the correlation is determined by the threshold.
For a small dataset, it is easy to set a number as a threshold for adding a neuron to the net, by just considering whether it is converged. If the results of loss function do not decrease for a certain number of times, it can be considered that the new neuron is ready for the net. Moreover, the number can be taken as a threshold. In contrast to the small dataset using batch type BPLN, sequential type BPLN (or as known as mini-batch), is applicable. Thus, the number as the threshold must be larger than the number of data divided by number of batches. The minimal threshold is indicated in formula (1). Because for each training of a batch, the loss varies. Only when the number of none decreases larger than the $\text{threshold}_{\min}$, it can be considered that the parameters are converged. So take EMNIST as an example, there are 131600 samples in total in the balanced dataset, with 112800 training data, and the batch size of Casper model is 64, so $\text{threshold}_{\min}$ is 112800 / 64 = 1762.5.

$$\text{threshold}_{\min} = \text{number of data} / \text{number of batches} \qquad \textbf{(1)}$$

If the threshold is set too small, the net will add the new unit very fast. So the network grows dramatically to be optimized. It may cost more time for the net to train. However, if the threshold is too large, it is easy to make the model overfitting. Thus, it is better to find an appropriate threshold for training. Simultaneously, arguments such as stage, epochs need to be balanced to achieve a good performance.

Batch Normalization

Batch normalization is applied for each Casper layer-in of this model to reduce internal covariate shift of weights and accelerate training [27]. The effect of batch normalization is not obvious in this optimized model, but can improve the results in the original model.

3 Results and Discussion

In [6], a benchmark result of EMINST balanced classification is provided, which is 78.02%±0.09% with more than 20 trials containing 10000 hidden layer neurons, by using a simple ELM network described in [28], based on Online Pseudo-Inverse Update method (OPIUM) for weights training illustrated in [29]. The results of the original Casper model on EMNIST are not as good as those benchmark results published in [6]. However, the results of the optimized model are better than the benchmark.
The results of the model are shown in Table 1. The optimized model inherits the refinements of the original model. Overall, the results of Rprop optimizer are the worst, the results of Adadelta are better, and all the best results are from Adam optimizer. For Rprop, activation functions and batch normalization have a relatively big effect on the improvement of the performance. However, their influence decreases while using Adadelta. For Adam optimizer, the effect of different activation functions and batch normalization is trivial. The ConvNet can dramatically improve the accuracy of the model. All the ConvNet results are better than the benchmark.

Table 2. Comparison of different results. The simple network only consists of inputs and outputs without any hidden layers is denoted by "miniNet". The arguments are denoted by abbreviation: e.g., s3 for stage 3, n15 for 15 hidden neurons, a5 for 5 neurons added at a time, "bn" for batch normalization.

No.	Network	Control Variable	Accuracy after Epoch1	Max Accuracy	Accuracy after Epoch10
1	miniNet	r, Rprop	35.90%	41.88%	41.88%
2	miniNet	p, Rprop	37.09%	43.42%	43.42%
3	miniNet	rr, Rprop	37.11%	43.61%	43.61%
4	miniNet-CT	s2, a5, r, Rprop,	35.90%	36.68%	21.61%
5	miniNet-CT	s2, a5, r, bn, Rprop,	35.90%	36.68%	25.30%
6	miniNet	r, Adadelta	43.70%	59.64%	59.64%
7	miniNet	p, Adadelta	44.44%	60.21%	60.21%
8	miniNet	rr, Adadelta	44.40%	60.18%	60.18%
9	miniNet-CT	s3, a5, r, Adadelta	43.70%	58.82%	58.82%
10	miniNet-CT	s3, a5, r, bn, Adadelta	43.70%	58.73%	58.73%
11	miniNet	r, Adam	62.30%	67.48%	66.48%
12	miniNet	p, Adam	62.57%	67.39%	66.33%
13	miniNet	rr, Adam	62.55%	67.39%	66.36%
14	miniNet-FC	n15, rr, Adam	58.33%	65.59%	65.23%
15	miniNet-FC	n800, rr, Adam	71.02%	82.49%	82.49%
16	miniNet-CT	s3, a5, r, Adam	62.30%	67.44%	66.45%
17	miniNet-CT	s3, a5, r, bn, Adam	62.30%	67.44%	66.44%
18	miniNet-CT	s3, a5, p, bn, Adam	62.57%	65.37%	64.04%
19	miniNet-CT	s3, a5, rr, bn, Adam	62.57%	65.55%	64.53%
20	miniNet-CT	s3, a2, rr, bn, Adam	62.49%	66.16%	65.65%
21	miniNet-CT	s5, a2, rr, bn, Adam	62.46%	65.95%	63.76%
22	miniNet-CFC	s3, a5, rr, bn, Adam	62.57%	66.01%	64.82%
23	miniNet-CT+CFC	s3, a5, rr, bn, Adam	62.57%	64.72%	60.69%
24	miniNet-CT+CFC	s3, a2, rr, bn, Adam	62.46%	64.96%	63.82%
25	LeNet-FC	n15, rr, bn, Adam	83.72%	87.55%	87.55%
26	LeNet- CT	s3, a5, rr, bn, Adam	85.83%	87.01%	86.30%
27	LeNet- CFC	s3, a5, rr, bn, Adam	85.77%	87.11%	86.75%
28	LeNet- CT+ CFC	s3, a5, rr, bn, Adam	85.96%	87.06%	84.90%
29	VGG- FC	n15, rr, bn, Adam	79.88%	86.43%	86.43%
30	VGG- CT	s3, a5, rr, bn, Adam	81.33%	86.14%	86.14%
31	VGG- CFC	s3, a5, rr, bn, Adam	78.68%	86.03%	84.98%
32	VGG- CT+ CFC	s3, a5, rr, bn, Adam	77.54%	86.05%	85.27%
No.	**Network**	**Control Variable**	**Accuracy after Epoch1**	**Max Accuracy**	**Accuracy after Epoch15**
33	LeNet- CT+ CFC	s7, a5, rr, bn, Adam	86.18%	87.11%	83.46%
34	LeNet- CT+ CFC	s5, a50, rr, bn, Adam	86.05%	86.89%	71.55%
35	VGG- CT+ CFC	s7, a5, rr, bn, Adam	77.65%	85.28%	84.36%

The results of No. 1–5 are from the original model. The resilient backpropagation is used according to [1]. It adapts learning rates on the sign of error gradient for each weight [30]. If the signs of the gradients are same, the learning step will be increased, otherwise, decreased. However, as the datasets are too large, batch training is implemented through the model. In [4], there are two types of backpropagation learning networks (BPLN's) that are batch and sequential. The

network model of this paper uses the sequential type, that updates the weights after each batch of data instead of after the complete training data. According to the formula (4) and (5) in [31], if using Rprop, the weights may have exponential growth or decrease because of the same sign of gradients of many batches, which causes the too huge or small magnitude of weights after several epochs, especially when many neurons or layers are added. During the tests, all the weights are initialized with the magnitude of 10^{-2}. After several epochs training, the magnitude of weights of the first hidden unit increase significantly to 10^3, then 10^5, and then 10^6, etc. The fluctuation of magnitude greatly reduces the performance of the model. Many problems arise from this issue, e.g., gradient explosion, saturated neurons, etc. Due to these problems, it is very easy for the neurons to be inactive and the weights cannot be updated if using ReLU. Similarly, sigmoid and tanh function are also easily get saturated. That is why the improved version of ReLUs and batch normalization have a positive influence on the model. For example, the effect of PReLU or RReLU can prevent the model from getting worse ,for they are more sensitive and active than ReLU, but they cannot solve these magnitude problems of weights.
So the RPROP algorithm is not good enough for the model, for it is not applicable for mini batch training. Therefore, RPROP optimizer is replaced with Adadelta optimizer for the refinement of the original model. Adadelta optimizer implements Adadelta algorithm proposed in [15]. However, after testing the Adam optimizer, the results indicate that Adam is obviously better than Adadelta. It coverages much faster than other algorithm and have a good performance on global optimization [25]. The quick coverage is very helpful for CasperNet, because the Casper network construction depends highly on the speed of coverage.
Comparing the results of 19 and 20, 33 and 34, it can be found that adding a small number of neurons at a time is better than a larger number of neurons. A large number neurons added to the net indicates a large number of new weights initialized. The initialized weights can greatly increase the loss and make it harder for the optimizer to reach a local optimum. For 20 and 21, the larger stage means the net keeps adding neurons, which will cause the similar problem demonstrated above.
The results of CT, CFC and CT + CFC indicate that the performance of CT + CFC is between CT and CFC, if they have the same scale of the network. The performance of CT is slightly better than the CFC in this model. However, some research shows that networks generated with CasCor can systematically have worse generalization than networks trained with the same method without cascading of the hidden units[5, 14]. The conflicts may be caused by the underfitting of the model, when there are not enough hidden neurons. Because of the settings with only 3 stages and 5 neurons added each time, most of the results are underfitting. However, for CT mode, there are a few more connections between hidden neurons, which comparatively reduces the problem of underfitting.
Although the model is improved dramatically, there are still many problems to solve. First, due to the mini batch training, the correlations of neurons in CasperNet are difficult to maximize, so the neurons are added quite slowly. Second, the number of neurons added each time, stages and threshold for the correlations need to be balanced. If the number of neurons added each time is small, the stage and threshold are large; it would cost too much time with many epochs to reach a suitable scale for the network. However, if adding too many neurons each time with lower stage and threshold, the network expands without maximizing the correlations of its neurons, which increases the difficulty for convergence. Therefore, the main problem is batch training.

4 Conclusion and Future Work

The model in this paper integrates the ConvNet (LeNet-5, VGG-16) with the CasperNet (CT, CFC). It is able to extract and discriminate features effectively, and is also able to self-determine its own depth and width for the Casper network. The test results reveal a better performance than that of [6]. It also indicates that Rprop is not applicable for the batch training. In addition, the maximization of correlations is also restricted by the batch training, which makes it hard to construct Casper network efficiently.
For the future work, weight normalization introduced in [32] may be tried. The Casper network can be revised by the inspiration of the cell structure [33]. The genetic evolutionary algorithm is a good way for the global optimization. The weights of each neuron can be extracted to form chromosomes for evolutionary computation (see Fig. 2). In addition, fuzzy logic may also be useful to deal with images with very similar features, for example, the handwritten 3, 5, 6 can be very similar, and the letter "O" and number 0.

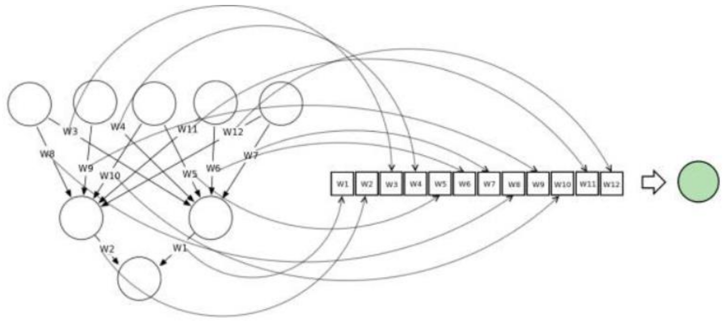


Fig. 2. Extracting weights to form a chromosome [34]

References

1. Treadgold, N.K., Gedeon, T.D.: A cascade network algorithm employing progressive RPROP. In: International Work-Conference on Artificial Neural Networks, pp. 733-742. Springer, 1997
2. Fahlman, S.E., Lebiere, C.: The cascade-correlation learning architecture. In: Advances in neural information processing systems, pp. 524-532. 1990
3. Adams, A., Waugh, S.: Function evaluation and the cascade-correlation architecture. In: IEEE International Conference on Neural Networks, pp. 942-946. 1995
4. Hwang, J.-N., You, S.-S., Lay, S.-R., Jou, I.-C.: The cascade-correlation learning: A projection pursuit learning perspective. IEEE Transactions on Neural Networks 7, 278-289 (1996)
5. Sjogaard, S.: A conceptual approach to generalisation in dynamic neural networks. Doctoral thesis, Aarhus University (1991)
6. Cohen, G., Afshar, S., Tapson, J., van Schaik, A.: EMNIST: an extension of MNIST to handwritten letters. arXiv preprint arXiv:1702.05373 (2017)
7. Grother, P.J.: NIST Special Database 19 Handprinted Forms and Characters Database. Visual Im.age Processing Group, Advanced Systems Division, National Institute of Standards and Technology (1995)
8. Kussul, E., Baidyk, T.: Improved method of handwritten digit recognition tested on MNIST database. Image and Vision Computing 22, 971-981 (2004)
9. Wan, L., Zeiler, M., Zhang, S., Le Cun, Y., Fergus, R.: Regularization of neural networks using dropconnect. In: International Conference on Machine Learning, pp. 1058-1066. 2013
10. Cireşan, D., Meier, U., Schmidhuber, J.: Multi-column deep neural networks for image classification. arXiv preprint arXiv:1202.2745 (2012)
11. Sato, I., Nishimura, H., Yokoi, K.: Apac: Augmented pattern classification with neural networks. arXiv preprint arXiv:1505.03229 (2015)
12. Chang, J.-R., Chen, Y.-S.: Batch-normalized maxout network in network. arXiv preprint arXiv:1511.02583 (2015)
13. Lee, C.-Y., Gallagher, P.W., Tu, Z.: Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In: Artificial Intelligence and Statistics, pp. 464-472. 2016
14. Prechelt, L.: Investigation of the CasCor family of learning algorithms. Neural Networks 10, 885-896 (1997)
15. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE 86, 2278-2324 (1998)
16. Ganin, Y., Lempitsky, V.: Unsupervised domain adaptation by backpropagation. arXiv preprint arXiv:1409.7495 (2014)
17. Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., Lempitsky, V.: Domain-adversarial training of neural networks. The Journal of Machine Learning Research 17, 2096-2030 (2016)
18. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
19. Treadgold, N., Gedeon, T.: Exploring architecture variations in constructive cascade networks. In: IEEE World Congress on Computational Intelligence, IEEE International Joint Conference on Neural Networks Proceedings, pp. 343-348. 1998
20. Treadgold, N.K., Gedeon, T.D.: Exploring constructive cascade networks. IEEE Transactions on Neural Networks 10, 1335-1350 (1999)
21. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision, pp. 1026-1034. 2015
22. Xu, B., Wang, N., Chen, T., Li, M.: Empirical evaluation of rectified activations in convolutional network. arXiv preprint arXiv:1505.00853 (2015)
23. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
24. Tieleman, T., Hinton, G.: Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning 4, 26-31 (2012)
25. Zeiler, M.D.: ADADELTA: an adaptive learning rate method. arXiv preprint arXiv:1212.5701 (2012)
26. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research 12, 2121-2159 (2011)
27. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)
28. Huang, G.-B., Zhu, Q.-Y., Siew, C.-K.: Extreme learning machine: theory and applications. Neurocomputing 70, 489-501 (2006)
29. van Schaik, A., Tapson, J.: Online and adaptive pseudoinverse solutions for ELM weights. Neurocomputing 149, 233-238 (2015)
30. Riedmiller, M.: Rprop-Description and Implementation Details. Technical Report (1994)
31. Riedmiller, M., Braun, H.: A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In: Neural Networks, 1993., IEEE International Conference on, pp. 586-591. IEEE, 1993
32. Salimans, T., Kingma, D.P.: Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In: Advances in Neural Information Processing Systems, pp. 901-909. 2016

33. Fritzke, B.: Growing cell structures—a self-organizing network for unsupervised and supervised learning. Neural networks 7, 1441-1460 (1994)
34. Ahire, J.B.: The Startup, Retrieved 20 July, 2018, from https://medium.com/swlh/artificial-neural-network-some-misconceptions-cb93e80b34bb