Neural Network Pruning Using Distinctiveness Measurement

Jiewei Qian

Research School of Computer Science, Australian National University
u6472740@anu.edu.au

Abstract. Feed-forward neural network with a few layers can be trained to solve varieties of problems. However, in order to train the network within a reasonable amount of time and to achieve a good result, the number of hidden units are usually increased beyond the point of what appears to be required. These units do not contribute to the final result and may be unused, or lead to incorrect generalization. There has been several researches on how to identify and remove such unnecessary nodes. In this paper, the effectiveness of one such pruning method based on distinctiveness measurement is investigated. It achieves 89.2% accuracy at 4.3% variance and outperforms conventional method (83.7% accuracy at 12.2% variance). This shows that this measurement is effective. In certain cases, the pruned network's accuracy can be further improved with a bit of extra training or tweaking. Additional experiments showed that this method may be used to estimate the minimum number of neurons required to achieve target accuracy by observing sharp drops in accuracy. The proposed method also outperforms evolutionary algorithm for neural network hyper-parameter selection in terms of speed (5 minutes vs. 10-15 hours).

Keywords: Neural Network · Optimization · Evolutionary Algorithm.

1 Introduction

In this paper, neural networks are assumed to consist three layers: input, hidden, and output. In each layer, neurons are fully connected to and only to their subsequent layer. There is no lateral, recurrent, backward or cross-layer connection. Each neuron has its own weight for previous layer neurons, and a bias. The performance of neural network is commonly measured by generalization performance, which is predominantly affected by three hyperparameters: number of training epochs, number of hidden neurons, and learning rate. They are usually determined by past experience or sometimes heuristically chosen. Usually, the number of hidden neurons are larger than what's minimally required. This could have adverse effect on generalization performance by "encouraging" the network to over-fit. Evolutionary algorithm can also be used to breed and evolve hyperparameters and neural networks[3] until they converge or achieve desired outcome though it requires significant longer time[13].

One method of reducing number of hidden neurons based on distinctiveness measurement was proposed by Gedeon and Harris in their work Network Reduction Techniques [6]. In this paper, a simplified but more general pruning method derived from the aforementioned work is presented. This method uses pair-wise neuron pruning and use iterative trainings to maintain accuracy. This paper also try to answer the following questions: 1) Is this method effective, specifically, whether it can identify and remove unnecessary neurons while maintaining accuracy; 2) What's the effect of different parameters of this method; 3) How does it compare with conventional training method in terms of consistency; 4) How does it compare with evolutionary algorithm.

The following of this paper presents in order: 1) the dataset and neural network parameters; 2) the definition and rationale of distinctiveness; 3) the design and implementation of pruning method; 4) the design of the evolutionary algorithm; 5) experiment result and analysis of Image Segmentation dataset under different thresholds, and lastly 6) conclusion of this method and future works.

2 Method

2.1 Dataset and Neural Network Structure

Image Segmentation dataset [4] from UCI machine learning repository is used to evaluate effectiveness of this method. The task is to classify inputs to seven classes representing different real-world objects. Each input represents a 3 by 3 region, and has 19 image related features encoded as real numbers. This dataset is chosen because image segmentation and classification fits typical neural network application [9] [2], contains sufficient but not excessive amount of data (2310 instances in total). This dataset is preprocessed and normalized, thus it only contains extracted image features instead of raw pixels, which simplifies discussion and implementation. The dataset contains 210 training instances, and 2100 testing instances. The experiments in this paper sticks to the provided training-testing split to ensure the results are comparable with other researchers' paper.

Other dataset can also be used, though they may not yield satisfactory results because the dataset does not fit typical neural network application. More information is given in section 3.

The neural network used to perform classification is modeled with 19 inputs and 7 outputs (the number of input attributes and the number of classes to classify), with various number of hidden units to test the pruning method. The number of hidden units is determined by rule of thumbs. The number of hidden neurons for initial training is set to an arbitrary amount that yields similar accuracy level of 90% when compared with other researcher's result for this dataset [12] [8]. The output is encoded as one-of-K, each neuron represents one class in the dataset. The prediction is the class represented by the output neuron with highest activation. The activation function for hidden units is sigmoid function, which lead to the decision of retraining for a few epochs after pruning. This is different from the original paper where the network does not need to be retrained. The cause and solution of this difference will be explained in section 3.

2.2 Definitely Useless Neurons

Neurons whose output is consistent for all inputs can be safely removed from the network. Recall the definition of neuron or perceptron, whose output is defined as the function of sum of weighted input plus bias.

$$y = f(\sum_{i=1}^{n} w_i \cdot x_i + b) \tag{1}$$

Neurons whose output always equal to a constant value can be safely removed by adding constant c to bias of subsequent layer neurons. This includes situations where neurons are always deactivated, always activated. Assume the output of n-th neuron is always a constant ($x_n * x_n \equiv c$), the previous equation can be rewritten as:

$$y = f(\sum_{i=1}^{n-1} w_i \cdot x_i + w_n * x_n + b) = f(\sum_{i=1}^{n-1} w_i \cdot x_i + (c + b)) = f(\sum_{i=1}^{n-1} w_i \cdot x_i + b') \tag{2}$$

2.3 Distinctiveness

Distinctiveness is determined by first computing neuron's output or activation over a given set of inputs. The result is a set of vectors with the size of input size. Each item in vector represents the neuron's output for the corresponding input. In other words, such vector reflects neuron's behavior under different circumstances. A neuron is distinct if its output is sufficiently different from other neurons. The extent of distinctiveness can be mathematically defined as a distance function over two neuron's output vector. It can be computed between two neurons, or among a group of neurons. For simplicity, this paper will focus on the two neurons situation.

There are several distance function candidates. In the original paper, angle between two normalized vectors are used to indicate distinctiveness. First, each element in the vector is subtracted with 0.5. The angle between two resulting vectors of length n, denoted as A and B, are computed according to definition of generalized cosine, given in formula 1. Two neurons are similar if the angle between them is close to 0°. Conversely, two neurons are complimentary (whose output are exact opposite) to each other if the angle between them is close to 180°. This measurement is also used in machine learning to determine similarity of inputs, and is considered to offer good performance [10] [14].

$$d_{angle}(A, B) = cos^{-1}(\frac{\sum_{i=1}^{n} A_i * B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}) \tag{3}$$

Total sum of squares (TSS) normalized over vector length might also be a good choice. It estimates how much output difference exists. The range of this measurement is $[0, 1]$. Values close to 0 means two neurons are identical, values close to 1 means two neurons are compliments.

$$d_{tss}(A, B) = \frac{(A_i - B_i)^2}{n} \tag{4}$$

The rationale of distinctiveness measurement is simple. In a minimum neural network, all neurons should have different function, for example, identify different decision boundaries, or pick up different features. Distinctiveness measurement aims to detect neurons that can either be removed or merged.

2.4 Pruning Method

First, neurons whose output are always 0 or 1 are removed. As discussed before, these types of neurons do not serve real functions, their weight can be added to bias of subsequent layer. This type of nodes are removed from computation graph without other processing.

For removal of compliments and merging of similar neurons, a threshold is defined. Pairs of neurons whose distinctiveness measurement is above (or below) the threshold will be processed. This paper focuses on angle distinctiveness measurement. For simplicity, this discussion will use a threshold of 15°, taken from the original paper. In other words, pairs of hidden units whose vector angle is above 165° are considered as compliments; pairs of hidden units whose vector angle is below 15° are considered to be similar enough for merging. Compliments are both removed without further processing.

For similar units, two neurons are merged. One is removed, while the other remains in the network. The remaining neuron takes over removed neurons' weight and bias. The resulting weights and bias are simple mathematics sums. It can be deduced from equation 1 that the resulting weights and bias yields the same result (before being processed by activation function) as combining two neurons.

The original paper states that the network does not require retraining. But in the proposed method, network is retrained for a small number of epochs (compared with initial training epochs). This is to adjust the network after pruning. There are two reasons: 1) during removal of constant activation neurons and compliments, bias is not added to subsequent layer. 2) during merging of similar neurons, output after activation function might change, because activation function may not be linear. Therefore, retraining is required to yield an equivalent network.

The retraining phase improves the original paper in terms of supporting non-linear activation functions. For linear activation functions, the retraining is not required because linear function satisfies $f(a) + f(b) = f(a + b)$ for any input a and b. However, non-linear activations do not have this property. The difference between left and right hand expression can be significant. Sigmoid function is used as an example here. Considering a scenario where the collected sum of $w_i * x_i + b$ (the value fed into activation function) for two identical neurons are both 1, it can be calculated that activation pre-merge and post-merge are: $sigmoid(1) + sigmoid(1) \approx 1.46$ and $sigmoid(1 + 1) \approx 0.88$. The difference is approximately 0.58, which is not trivial. This difference might degrade network's accuracy. Therefore, retraining is required to correct this type of merging errors. It is worth noting that non-continuous activation functions, for example ReLUs may require further improvement to weight merging strategy, because the merged weight could yield results on a different function segment, thus invalidating the underlying continuity assumption.

The pruning and retraining process is repeated for some iterations until the number of hidden units and accuracy become stable. This process will be illustrated in section 4.

2.5 Evolutionary Algorithm

Evolutionary algorithm is inspired by the how specie evolves, and has its root in biology and nature [1]. In its simplest form, a population are made up of some individuals. For each generation, individuals breed with each other and produce children by exchanging genetic information (crossover) from two or more parties. Each children has change of experiencing mutation, which change genetic information in some way, just like gene copy error in nature. Then the entire population is tested for fitness to survive. Individuals deemed unfit to survive are removed from population. Then this cycle starts again. Eventually, the population will reach a point that majority or all individuals are fit to survive.

In computer science, the aforementioned procedure in simulated numerically. Individuals are a set of parameters (binaries), the population is a group of such parameters. Crossover is defined as swapping parameters of two individuals. Mutation happens by changing parameters randomly within a small margin (or according to some probabilistic distribution). The fitness to survive is usually measured by fitness function, which is a formula assigning weights to different objective measurements. The initial population is usually a group of individuals whose parameters are uniformly sampled from their domain. This sample process ensures diversity and coverage of raw genetic material. So the algorithm can fully explore solution space.

Concretely, in neural network hyperparameter tuning, individuals are defined as a tuple of (training epochs, number of hidden neurons, learning rate). Crossover is defined as exchanging one or more tuple element across two such tuples. Mutation is defined as changing tuple element within its domain. Fitness is primarily defined as accuracy of neural network trained with parameters in such tuple, with penalty for large training epochs or number of hidden neurons.

$$fitness(x) = \sum_{i=1}^{n} w_i \cdot obj_i(x) \tag{5}$$

Because the neural network hyperparameters has specific domains. Generic mutation operators may not be a good choice. For example, the number of hidden neurons should be greater than 0, but less than some large number that obviously makes no sense (In the case of ImageSeg dataset, maybe 100). The learning rate should be greater than 0, but smaller than 1. Therefore, Gaussian mutators are used. They change the value being mutated by some amount sampled from Gaussian distribution. The change amount is computed relative to value's domain. This mutator offers three advantages: 1) the mutation is usually local, it will not cause huge change in value (for example, mutate 0.2 learning rate to 0.8); 2) the mutation still offer a small chance of radical changes; 3) the mutator is easy and efficient to implement. After mutation, the result is clipped (clamped) to its domain (if result is greater than upper bound, set it to upper bound. vice versa).

The mutator can be conveniently defined as the following equation, where y is the result, δ is a value sampled from Gaussian distribution $N(X, \sigma^2)$ with $\sigma = 0.05$, x^+ and x^- are the lower bound and the upper bound of value x. This effectively changes value by 15% of value's domain. For integers, the result is rounded.

To encourage the algorithm to find a set of hyperparameters that is quickest to train while giving similar results. A decay process is added to crossover process. When a crossover happens on epochs or hidden neurons, the result is reduced by 5%. When a crossover happens on learning rate, The result is increased by 5%. This encourage the algorithm to converge to the hyperparameters that takes least time to train (high learning rate, small epochs) and has least amount of hidden neurons (the objective of network pruning).

$$y' = (1 + \delta) * (x^+ - x^-) + x \tag{6}$$

$$y = max(x^-, min(x^+, y')) \tag{7}$$

At each generation, one parent is chosen based on its fitness. The fitter one individual is, the higher change of it being chosen as the parent. The chosen parent will then crossover with 50% of individuals in the population. Each crossover produces two children, each mutated with the aforementioned mutator. Then the worst 20% of parents are replaced with the same amount of children sampled according to their fitness. The resulting population is used to start the next generation.

2.6 Experiment setup

Experiments are conducted using PyTorch. Image Segmentation dataset is used without further processing (except modifying file headers into those accepted by csv standard). All attributes are read as float numbers, and class label encoded into decimal integers. Training testing split is 1:10, with 210 and 2100 instances respectively.. No specific reason is given by dataset contributor and other paper authors as why this odd ratio is chosen. In order to ensure our result is comparable with other researchers', this ratio is not changed. Otherwise, our model will be trained with more data than others, this invalidates the comparison process because our model would have more information to base its decision upon.

The network is constructed with 19 inputs each corresponding to one attribute, and 7 outputs each corresponding to one class. The amount of hidden units is determined by increasing the amount from 4 by 4 each time, until it consistently yields about 85% accuracy after 4000 epochs of training, using stochastic gradient descent optimizer and cross entropy loss. Initial training epoch is intentionally large to ensure network can reach a stable performance level, so change in accuracy after pruning is caused by pruning rather than insufficient initial training. The number of hidden units turns out to be 36. After pruning, the network is retrained for 500 epochs.

The 85% accuracy is chosen from the other researches using this dataset. Unfortunately, results of neural network methods can not be found, so the result is compared against the 91.4% accuracy from the moderated SVM classifier in this paper [8]. The target is rounded down 5% because our naive neural network solution is unlikely to outperform this advanced method. This target is on par with the achieved accuracy of 85%, 80% from MIPSVM and Naive Bayes in this paper [12].

For simplicity, the trained network is pruned and retrained for 10 iterations. The number of remaining hidden neurons and accuracy after each interaction is recorded to demonstrate pruning effectiveness. The experiment is run for 5 times. Ideally, all experiment should converge on the same number of hidden units and a consistent accuracy (or highest achieved accuracy across iterations).

A separate experiment is conducted to demonstrate the resulting network indeed contains only required amount of hidden neurons. The amount of hidden neurons is further reduced, and used to train a different network for 10 times under same conditions without pruning. If the proposed method is correct and effective, networks with fewer hidden units should struggle to achieve similar accuracy. It is observed that neurons encapsulates knowledge of the problem [7][5]. Removing neurons equivalently throws away useful knowledge, which naturally lead to degraded accuracy.

Neural network parameter used in the evolutionary algorithm are constrained to the following range. Values outside the range are unnecessary or unreasonable (such as very large number of hidden neurons and learning rate).

Table 1. Evolutionary Algorithm Parameter Constraints

	Lower bound	Upper bound	Crossover Decay
Epochs	500	6000	-5%
Hidden Neurons	1	40	-5%
Learning Rate	1e-6	0.5	+5%

Neural network training process depends on randomly initialized weights, one set of parameters will give different accuracy results. To mitigate such variance, the accuracy is measured on three training runs and averaged. This is analogous to giving a person multiple chances to prove themselves. Failing the exam once should not prove someone's incompetence, but failing multiple times should.

The evolutionary algorithm is configured to run with a population of 100 that evolves for 50 generations.

3 Result and Discussion

Table 2. Pruning result using angle distinctiveness

Threshold	Initial	Post-Pruning			
	Accuracy	N. Iter.	Neurons	Worst Accy. %	Best Accy. %
15°	88.8	1	24	89.4	91.0
	90.2	4	24	87.9	91.8
	91.1	3	22	90.4	92.3
	88.7	1	24	89.4	91.6
	90.5	1	23	89.0	91.7
30°	86.4	2	14	86.8	91.3
	90.5	2	13	79.7	89.4
	89.0	2	15	82.5	90.1
	89.4	2	15	87.5	90.3
	89.9	2	19	87.5	90.8

As is shown in table 2, N. Iter. represents the number of iterations before number of neurons become stable. A small number of iterations is required for the network to stabilize. This could be the result of forcing network to forget and further generalize learned knowledge. It could also be the result of pair-wise reduction fail to discover group of neurons who together has a stable activation.

Post-pruning accuracies are measured at the end of each iteration (after pruning and retraining). The best accuracy after pruning is not significantly or statistically different from pre-pruning accuracy. The worst accuracy is the result of first pruning, because this iteration removes the most number of neurons, which translates into forgetting most knowledge.

As is shown in figure 1, there is a drop in accuracy immediately after pruning. The drop is significant after first pruning (happened at 4000 epoch). The drop is noticeable smaller afterwards, where pruning removes fewer neurons (happened at 4500 epoch). No more neurons are removed after 5000 epoch. After pruning, the retraining phase restores accuracy to pre-pruning levels. The drop in accuracy is caused by the fact that sigmoid is not a linear function. This observation supports the previous discussion that retraining is required for the proposed method to work with non-linear or generic activation functions.

For comparison, several networks are trained with conventional method using the minimum number of neurons discovered in the process under the same condition. Table 3 shows that although it is possible to use convention method to achieve good accuracy, the result is inconsistent. High variance across trials is observed. The proposed method usually out-performs the convention method both in accuracy and consistency, but is still a little behind the baseline moderated SVM classifier, which is more complicated and achieves 92% accuracy [8].

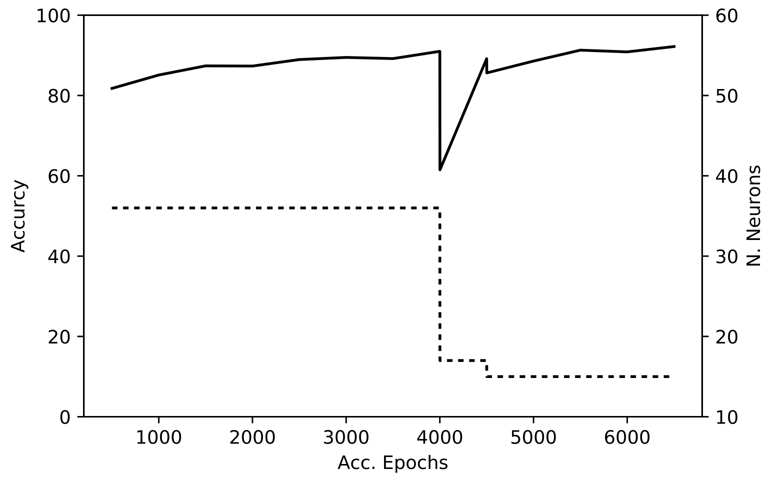

Fig. 1. Accuracy and neurons during training and pruning

Table 3. Comparison with conventional method

	Neurons	Worst %	Best %	Average %	Variance
Conventional Method	22	83.1	90.3	86.6	7.1
	13	75.3	83.7	80.3	12.2
Proposed Method	22	89.9	91.3	89.9	0.3
	13	84.6	89.2	84.6	4.3

Table 4. Comparison of pruning angle threshold

Threshold	N. Neurons	N. Retrain Epochs	Best Accy. %
15°	22	6	90.7
30°	13	5	90.3
35°	12	7	89.7
37°	11	8	91.3
40°	8	4	87.4
42°	5	6	73.0
45°	5	6	84.0
50°	5	8	65.0
60°	2	2	49.4

The experiment is also run with different pruning angles. Table 4 shows the best result achieved in 5 runs. It's obvious that aggressive pruning angles generally requires more training to restore the network to previous performance. The accuracy is bad immediately after training, but can be restored to original levels. Interestingly, thresholds below 30° yield a good neural network most of the time, but when threshold goes above 40 (or neurons goes below 10), there is a significant drop in accuracy. It might be worth investigating which threshold is good for general applications. The 15° proposed in the original paper is conservative. The drop in accuracy may be used an an indicator of minimum number of neurons required.

Table 5. Hyperparameters discovered by evolutionary algorithm

N. Epochs	N. Neurons	Learning Rate	Accuracy
5264	15	0.032	89.62%
5678	12	0.023	89.47%
6000	18	0.033	90.09%

It can be seen that the parameters discovered by evolutionary algorithm match with network pruning results. The number of hidden neurons are close to the 30° degree pruning result (1319 neurons). This supports the hypothesis that network pruning can produce the network that has minimal amount of neurons required. It can also be seen that evolutionary algorithm can produce neural networks with consistent accuracy. Though, it should be noted that running evolution algorithm is extremely time consuming. Running 50 generations takes about 10 hours on a GPU-accelerated machine. The produced result is still inferior to the proposed "train the prune" method, which usually takes 30 seconds to run one trial and yield equally good and consistent results.

Interestingly, on different datasets, the pruning method concludes that all hidden units should be pruned, resulting in a two-layer (input and output) network with the same performance. The resulting network is equivalent to a linear regression model. This is the case for default of credit card clients dataset [4] [15] and Soybean dataset [4]. This paper[15] provides comparison of several methods. It can be seen that decision tree and k-NN both give similar accuracy but with better efficiency.

4 Conclusion and Future Work

This paper explains the rationale and method of pruning neural network using distinctiveness measurement, presents experimental results to show this method is effective in reducing network size thus improving post-training performance. It's observed that 1) the proposed method yields more consistent performance than conventional training method; 2) aggressive pruning threshold might force the network to "forget" and generalize the problem better; 3) the proposed method produces comparable results with the ones from evolutionary algorithm, at significant cheaper time costs.

As discussed in section 2, several candidates exists for distance function. Vector angle is chosen because it is specified in the original paper. However, other function or combination of functions might perform as well or better. It worth investigating their performance, or they can be used to simplify detection of functionally stable groups as opposed to the banding and Gaussian vector pivot method proposed in the original paper.

As mentioned in section 3, the proposed method could prune all hidden neurons on certain datasets. Could this behavior be used to characterize problem? Could this behavior indicate that the problem is of linear nature, and can be solved by simpler linear regression (or derived) techniques?

The benefit (or lack of benefit) of input preprocessing to this pruning method is another interesting topic. Exposure of underlying problem structure could help pruning method in identifying excessive nodes.

Network reduction is the opposite of network expansion. Theoretically, reducing and simplifying network should yield functionally identical network which is expanded from scratch. Future inspirations may be drawn from techniques used in building cascade network (CasPer) [11], which yields excellent performance for tricky pattern spaces. It might be possible to implement an incremental pruning and training algorithm, the opposite way of CasPer, to yield good performance with shorted time compared with traditional training methods that rely on human experience.

References

1. Back, T.: Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford university press (1996)

2. Boland, M.V., Murphy, R.F.: A neural network classifier capable of recognizing the patterns of all major subcellular structures in fluorescence microscope images of hela cells. Bioinformatics **17**(12), 1213–1223 (2001)
3. Branke, J.: Evolutionary algorithms for neural network design and training. In: IN PROCEEDINGS OF THE FIRST NORDIC WORKSHOP ON GENETIC ALGORITHMS AND ITS APPLICATIONS. pp. 145–163 (1995)
4. Dheeru, D., Karra Taniskidou, E.: UCI machine learning repository (2017), http://archive.ics.uci.edu/ml
5. Gedeon, T., Catalan, J., Jin, J.: Image compression using shared weights and bidirectional networks. In: Proceedings 2nd International ICSC Symposium on Soft Computing (SOCO'97). pp. 374–381
6. Gedeon, T., Harris, D.: Network reduction techniques. In: Proceedings International Conference on Neural Networks Methodologies and Applications. vol. 1, pp. 119–126 (1991)
7. Gedeon, T., Harris, D.: Progressive image compression. In: Neural Networks, 1992. IJCNN., International Joint Conference on. vol. 4, pp. 403–407. IEEE (1992)
8. Kwok, J.Y.: Moderating the outputs of support vector machine classifiers. IEEE Transactions on Neural Networks **10**(5), 1018–1031 (1999)
9. Sharma, N., Ray, A.K., Sharma, S., Shukla, K., Pradhan, S., Aggarwal, L.M.: Segmentation and classification of medical images using texture-primitive features: Application of bam-type artificial neural network. Journal of medical physics/Association of Medical Physicists of India **33**(3), 119 (2008)
10. Steinbach, M., Karypis, G., Kumar, V., et al.: A comparison of document clustering techniques. In: KDD workshop on text mining. vol. 400, pp. 525–526. Boston (2000)
11. 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)
12. Tveit, A.: Empirical comparison of accuracy and performance for the mipsvm classifier with existing classifiers. Tech. rep., Technical report, Divison of Intelligent Systems, Department of Computer and Information Science, Norwegian University of Science and Technology (2003)
13. Yao, X., Liu, Y.: A new evolutionary system for evolving artificial neural networks. IEEE Transactions on Neural Networks **8**(3), 694–713 (May 1997). https://doi.org/10.1109/72.572107
14. Ye, J.: Cosine similarity measures for intuitionistic fuzzy sets and their applications. Mathematical and Computer Modelling **53**(1), 91 – 97 (2011). https://doi.org/https://doi.org/10.1016/j.mcm.2010.07.022, http://www.sciencedirect.com/science/article/pii/S0895717710003651
15. Yeh, I.C., Lien, C.h.: The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. Expert Systems with Applications **36**(2), 2473–2480 (2009)