2016 7th International Conference on Intelligent Systems, Modelling and Simulation

An Analysis of the Regularization between L_2 and Dropout in Single Hidden Layer Neural Network

Ekachai Phaisangittisagul
Department of Electrical Engineering, Faculty of Engineering
Kasetsart University
Bangkok, Thailand
Email: fengecp@ku.ac.th

***Abstract*—In supervised learning, one of the motivations and intuitions behind the design of the learning algorithm is to prevent an overfitting. Overfitting usually occurs when a learning model is excessively complex, especially when having too many parameters to adjust. A typical approach to alleviate this problem is to introduce a form of the penalty term in the objective function, which is called regularization, for preventing the network parameters from growing too large. Another proposed strategy known as dropout is introduced to prevent co-adaptation on each training data by randomly dropping some units out during the training process. In this paper, an analysis of different regularization techniques between L_2-norm and dropout in a single hidden layer neural networks are investigated on the MNIST dataset. In our experiment, both regularization methods are applied to the single hidden layer neural network with various scales of network complexity. The results show that dropout is more effective than L_2-norm for complex networks i.e., containing large numbers of hidden neurons. The results of this study are helpful to design the neural networks with suitable choice of regularization.**

Keywords–dropout; multilayer perceptrons; neural network; overfitting; regularization

I. Introduction

As we already know in supervised learning that the goal of training a learning model is not to learn an exact representation of the training data itself, but to build a model that is able to understand the process to generate the relation between inputs and outputs. In addition, the learning model should have good generalization to accurately predict an unseen input data. For a simple example of polynomial regression problem, a learning model is created to fit a set of m data points $\{x^{(i)}, y^{(i)}\}$ where $i = 1, \ldots, m$ by a technique to minimize the sum of squared error cost function. This problem can be considered a mapping function $y = f(x)$ that takes x as an input and produces y as an output. A poor generalization performance can occur if too few coefficients of the polynomial function are used in prediction. This type of poor performance is called underfitting. In contrast, if the polynomial function predicts the output with too many coefficients, poor generalization can also be obtained since the predictive function is excessively complex to the training data and this is named overfitting problem.

One of the most successful and widely used learning algorithms is a neural network inspired by a biological neural system. A simple form of this network is to feedforward from an input layer to zero or multiple hidden layers, and finally to an output layer and is known as Multilayer Perceptrons (MLPs) [1]. So, the MLPs allow signals to travel one way only from input layer to output layer. In each layer, it is composed of a large of highly interconnected simple linear or nonlinear processing elements called neurons. The MLPs are successfully applied to solve many problems such as handwritten recognition, face recognition, stock market prediction, etc. In the neural network design, the number of neurons in the input and the output layers can be identifed by a dimensionality of the input data and the target output. But the number of hidden layers and the number of neurons in each hidden layer are free parameters that can be chosen based on model selection approach corresponding to the balance between underfitting and overfitting. In practice, the number of hidden neurons controls the number of weight parameters or complexity of the networks. Unfortunately, the generalization performance of the networks is not a simple function of the hidden neurons. In particular, a complex network is usually used to solve for complex problems that tend to be more susceptible to overfitting. So, some researchers proposed a method to avoid overfitting problem by adding a penalty or regularization term for

2166-0670/16 $31.00 © 2016 IEEE
DOI 10.1109/ISMS.2016.14

controlling the magnitude of the model parameters in the error function. The well-known regularization method in machine learning community is known as an L_2 regularization or ridge regularization (more generally Tikhonov's regularization [2]). Another approach recently introduced for training a feedforward neural network by Hinton et al. [3] is called *dropout*. Basically, a dropout is a method that can be used to reduce an effect of overfitting by preventing co-adaptations among neurons on the training data. Its concept is very simple by randomly dropping neurons during training each example. Similar to Bagging [4], dropout trains a collection of randomly chosen models that draw from a subset of training data. The difference of dropout training from bagging is that each model is trained for only one epoch and all of the models share the same weight parameters.

In this study, we look to investigate how to achieve the benefit of the L_2 regularization and dropout training. An empirical analysis is performed to compare the performance between L_2 regularization and dropout. The paper is organized in the following way: section 2 will described the basic concepts of L_2 and dropout in brief detail. The details of the neural network architectures and experiment procedures including discussion will be provided in section 3. Finally, we will draw a conclusion in section 4.

II. L_2 REGULARIZATION AND DROPOUT

Regardless of many successes in applications of neural networks, overfitting still remains one of the major challenges to overcome in machine learning research community. Several approaches have been proposed to mitigate this problem. A widely used method is to add a regularization term to an error function so that the total error function is minimized in the following form [2],[5]:

$$\mathcal{L}_T(\mathbf{W}) = \mathcal{L}_{\mathbf{D}}(\mathbf{W}) + \lambda \mathcal{L}_{\mathbf{W}}(\mathbf{W}) \tag{1}$$

where $\mathbf{W}$ is neural network parameters. $\mathcal{L}_T(\mathbf{W})$ is a total loss function and λ is a regularization parameter that controls the tradeoff between $\mathcal{L}_D(\mathbf{W})$ and $\mathcal{L}_W(\mathbf{W})$. $\mathcal{L}_D(\mathbf{W})$ is a sum-of-squares error function between the target output, $y^{(i)}$ and the network output, $h(x^{(i)}; \mathbf{W})$ which can be computed by

$$\mathcal{L}_D(\mathbf{W}) = \frac{1}{2}\sum_{i=1}^{m}[y^{(i)} - h(x^{(i)}; \mathbf{W})]^2 \tag{2}$$

In the last term of (1), $\mathcal{L}_W(\mathbf{W})$ is a sum-of-squares of the weight parameters and can be determined by:

$$\mathcal{L}_W(\mathbf{W}) = \frac{1}{2}\mathbf{W}^T\mathbf{W} \tag{3}$$

This particular choice of regularization is known as an L_2 regularization (or weight decay). Its solution is able to find in a closed form since the total error function in (1) is simply a quadratic function.

Overfitting is not only a major concern to solve, but large neural networks are also difficult to train and use during the testing phase. As we know, combining several learning models and then averaging prediction results can enhance the overall performance [4]. However, the individual models must be trained on different dataset so that each of them has different hyperparameters to be omptimized. Dropout is a scheme that also aims to handle these issues. A motivation behind the dropout concept is brought from a mixability theory for the role of sex in evolution [6]. In sexual reproduction, an offspring is produced by combining half of the genes from one parent and another half from the other. To make the genes more robust, they have to learn to work well with another set of genes. The idea of dropout is to drop out units (neurons) in a neural network in which the dropping units are chosen by random with probability $q = 1 - p$. When any unit is dropped out, all its incoming and outgoing connections will be neglected as an illustration in Fig. 1. The objective of randomly dropping out neurons is to allow each neuron to learn something useful on its own without relying too much on other neurons to correct its shortcomings [7],[8]. Therefore, the dropout is equivalent to perform sub-sampling a collection of possible "thinned" neural networks. If a neural network has k neurons, the collection of the networks will have 2^k possible thinned networks. Then, the remaining weight parameters are trained by standard backpropagation algorithm [1],[9]. In each epoch of learning, each thinned network will be trained for each sample or minibatch with weight sharing. This strategy is able to reduce co-adaptation among the neurons in the networks. Training with dropout is equivalent to adding noise to the neural network which is similar to the context of Denoising Autoencoders [11]-[14]. After finishing training the networks, a collection of random 2^k thinned networks with shared weight parameters will be combined as a single neural network. At test time, a trained network is used without dropout procedure.

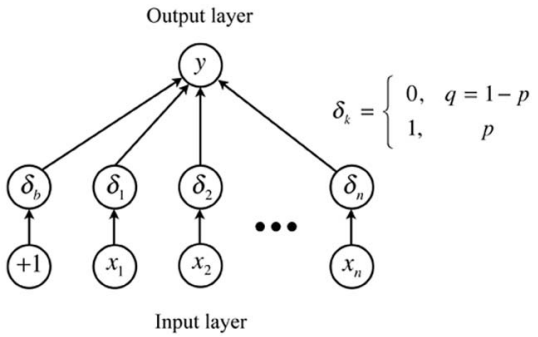

Figure 1. An example of neural network architecture trains with dropout. Each of hidden neuron is randomly dropped out with Bernoulli distribution (p).

Let's consider a neural network with L layers where $l \in \{0, \cdots, L-1\}$ is an index of the network layer. Index $l = 0$ and $l = L-1$ are represented as the input layer and output layer, respectively. Suppose $\mathbf{z}^{(l)}$ denote the input vector to the layer l and similarly, $\mathbf{y}^{(l)}$ express the output vector to the layer l. In addition, $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weight and biased parameters at layer l. Based on the operation of the MLP network, the input and output vectors at the hidden layers ($l = 1, \cdots, L-2$) can be computed by:

$$\mathbf{z}^{(l+1)} = \mathbf{W}^{(l+1)}\mathbf{y}^{(l)} + \mathbf{b}^{(l+1)} \quad (4)$$

$$\mathbf{y}^{(l+1)} = \mathbf{a}(\mathbf{z}^{(l+1)}) \quad (5)$$

where $\mathbf{a}(.)$ is an activation function. When dropout is applied, the operation becomes:

$$\delta_i^{(l)} \sim Bernoulli(p) \quad (6)$$

$$\tilde{\mathbf{y}}^{(l)} = \boldsymbol{\delta}^{(l)} \otimes \mathbf{y}^{(l)} \quad (7)$$

$$\mathbf{z}^{(l+1)} = \mathbf{W}^{(l+1)}\tilde{\mathbf{y}}^{(l)} + \mathbf{b}^{(l+1)} \quad (8)$$

$$\mathbf{y}^{(l+1)} = a(\mathbf{z}^{(l+1)}) \quad (9)$$

where $\otimes$ is an element-by-element multiplication. $\delta_i^{(l)}$ is a Bernoulli random variable of neuron i at layer l with probability (p_i) of being 1.

Consider a dropout for a single linear unit (no hidden layer) with an input $x^{(i)} \in \mathbb{R}^n$. The output can be calculated by a weighted sum of the input data.

$$y^{(i)} = \sum_{k=1}^{n} w_k \, x_k^{(i)} \, \delta_k + b\delta_b \quad (10)$$

In this case, Bernoulli random variables (δ_i) that will remove neuron i from the network with probability $P(\delta_i = 0) = q_i$ are assumed to be independent to each other. As a result, $P(\delta_i = 1) = 1 - q_i = p_i$ and we can apply the linearity property of the expectation to the output of the neuron.

$$\begin{aligned} E[y^{(i)}] &= \sum_{k=1}^{n} w_k \, x_k^{(i)} \, E[\delta_k] + bE[\delta_b] \\ &= \sum_{k=1}^{n} w_k \, x_k^{(i)} \, p_k + b \, p_b \end{aligned}$$

If the random variable is independent and identical distribution (IID), then $p_i = p$ and $p_b = p$.

$$\begin{aligned} E[y^{(i)}] &= \sum_{k=1}^{n} w_k \, x_k^{(i)} \, p + b \, p \\ &= p \cdot \Big[\sum_{k=1}^{n} w_k \, x_k^{(i)} + b \Big] \end{aligned} \quad (11)$$

Equation (11) represents that the weight parameters of this network must be simply multiplied by p at test time and the obtained output is the expected output of the network.

III. Experiments

This section will describe the experimental procedure and demonstrate comparison results between L_2 regularization and dropout. To measure the benefit of each method for training the neural networks, several single hidden layer network architectures with various complexity were used to classify the MNIST dataset of handwritten digit images. In addition, each single hidden layer network architecture of L_2 regularization and dropout were both set to the same learning rate parameter ($\alpha = 1$) and also the initial weight parameters were random with uniform distribution in the range of:

$$\boldsymbol{W}^{(l)} = \pm\sqrt{\frac{6}{N^{(l)} + N^{(l-1)} + 1}} \quad (12)$$

where $N^{(i)}$ is the number of neurons in layer i.

The MNIST database of handwritten digit ("0-9") gray-scale images consists of 60,000 examples for training and 10,000 examples for testing, respectively. Each handwritten

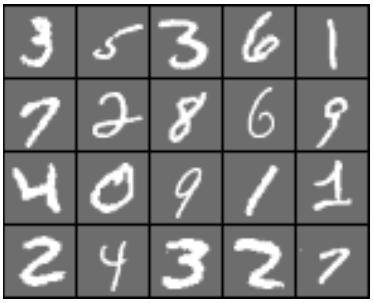

Figure 2. A visualization example of MNIST handwritten digit dataset is shown. This dataset consists of "0-9" handwritten digit of gray scale images with 28×28 pixels.

digit image has been size-normalized and centered in a fixed-size 28×28 image field as illustrated in Fig. 2. For effective implementation of neural networks as classifiers, a corresponding class label of the output $y^{(i)} \in \{1, 2, ..., 10\}$ has been employed in which "0" is mapped to digit 10. In our experiment, we hypothesize that the number of hidden neurons between 10 and 100 is considered not too complex, while the number of neurons between 100-1000 is relatively complex for MNIST classification problem. Note that the notation of network architecture used in this study represents the number of neurons in the order of input, hidden, and output layers.

Experimental results shown in Table I are based on the following settings. In L_2 regularization, the regularizer parameter (λ) is set equal to 1×10^{-4}. This value is imposed during optimization to balance between reconstruction error ($\mathcal{L}_D$) and sum squared of the weight parameters ($\mathcal{L}_W$) in the total loss function ($\mathcal{L}_T$). In dropout training, we decide to use probability of dropping the neuron(s) out with $p = 0.5$ (values between 0.4-0.7 used in practice). From Table I, we can observe that the accuracy performance of L_2 regularization increases in case of small network architecture and degrades when the network grows larger. Also, L_2 regularization achieves higher perdiction accuracy than dropout for small number of hidden neurons. However, a rate of change in performance is significantly decreased in larger networks. While the performance of dropout training is also reduced when the networks get larger, the rate of change is slightly slower and more robust than the L_2 regularization.

TABLE I
CLASSIFICATION PERFORMANCE BETWEEN L_2 REGULARIZATION AND DROPOUT ON MNIST DATASET WITH RANDOM INITIALIZATION OF WEIGHT PARAMETERS

	L_2 regularization		Dropout	
Network architecture	% accuracy	% Rate of change	% accuracy	% Rate of change
784-10-10	92.98	-	89.48	-
784-20-10	94.88	2.04	92.02	2.84
784-30-10	95.52	0.67	93.43	1.53
784-40-10	96.04	0.54	94.01	0.62
784-50-10	96.42	0.40	94.43	0.45
784-100-10	96.71	-	95.72	-
784-200-10	96.80	0.09	96.73	1.06
784-300-10	96.83	0.03	96.81	0.08
784-400-10	96.81	-0.02	97.00	0.20
784-500-10	96.74	-0.07	97.04	0.04
784-600-10	96.57	-0.18	97.09	0.05
784-700-10	96.65	0.08	97.09	0.00
784-800-10	96.59	-0.06	97.16	0.07
784-900-10	96.32	-0.28	97.20	0.04
784-1000-10	95.86	-0.48	97.10	-0.10

TABLE II
CLASSIFICATION PERFORMANCE BETWEEN L_2 REGULARIZATION AND DROPOUT ON MNIST DATASET WITH PRETRAINING OF WEIGHT PARAMETERS (SAE)

	L_2 regularization		Dropout	
Network architecture	% accuracy	% Rate of change	% accuracy	% Rate of change
784-100-10	96.97	-	96.51	-
784-200-10	97.02	0.05	97.49	1.02
784-300-10	97.05	0.03	97.82	0.34
784-400-10	97.05	0.00	97.97	0.15
784-500-10	96.90	-0.15	97.99	0.02
784-600-10	96.99	0.09	98.09	0.10
784-700-10	96.96	-0.03	98.19	0.10
784-800-10	96.86	-0.10	98.29	0.10
784-900-10	96.91	0.05	98.22	-0.07
784-1000-10	96.61	-0.31	98.17	-0.05

In addition, neural networks can be improved the performance by pretraining the network parameters with Stacked Auto-Encoder (SAE) [15], Restricted Boltzmann Machine (RBM) [16], etc. This pretraining step is then followed by fine-tuning step using standard back propagation approach. In practice, pretraining can be performed by using unlabelled data as an unsupervised feature learning. In this

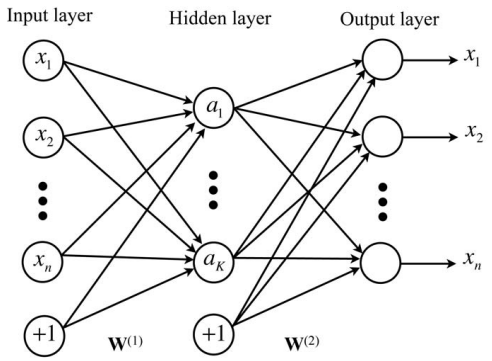

Figure 3. A stacked autoencoder neural network

study, we applied SAE as a pretraining method before fine-tuning with the backpropagation.

Conceptually, the SAE is learned by setting the target values ($y^{(i)}$) to be equal to the input values ($x^{(i)}$) as shown in fig.3. The network parameters are adjusted to minimize the following objective function.

$$J(W,b) = \frac{1}{2}\sum_{i=1}^{m} \|y^{(i)} - x^{(i)}\|_2^2 \tag{13}$$

The classification results of using SAE to initialize a network parameters are illustrated in Table II. The results show that pretraining can boost the performance of both L_2 regularization and dropout. However, dropout has more room of improvement and slowly drops the performance when the network is more complex. The reason that dropout is more robust than L_2 regularization is because the training process of dropout allows to explore various regions of the parameter space that it would have not encountered in regular training. As proposed in literatures, a good approach to improve the predictive performance is to integrate a very large number of different models. A typical way to do this scheme is to train many networks with different input data by using random selection, but computation resource and time for training and testing are very expensive. In dropout, each different network is trained with almost every training case using weight sharing. As a result, the dropout makes it possible to train a large number of different networks in a moderate time.

IV. Conclusion

Dropout is recently introduced a method for training neural networks to reduce overfitting. The main idea is to randomly drop neurons using Bernoulli gating variables during training so that the network is present with unreliable neurons to prevent co-adaptation among them. Therefore, no neurons depend excessively on other neurons to correct its mistakes and they must work well with other neurons in a wide variety of different situations. The major advantage of dropout allows a single network to be able to model a large number of different sub-networks in an inexpensive and simple means of both training and testing. Experimental results demonstrate that dropout training in a large network not only provides better performance improvement but it is more robust than L_2 regularization. In contrast, the L_2 regularization yields higher predictive accuracy than dropout in a small network since averaging learning model will enhance the overall performance when the number of sub-model is large and each of them must different from each other. The idea of dropout is not limited only to train the MLPs but it can be general to apply in many other learning models. In short, the dropout training is considered one of effective tools to reliably improve the performance of a large network.

Acknowledgment

The author would like to thank TRF (Thailand Research Funds) under grant research no. TRG5680074 and KURDI (Kasetsart University Research and Development Institute) for supporting this research.

References

[1] P. Baldi and K. Hornik. Learning in linear networks: a survey. IEEE Trans. on Neural Networks, 6(4): pp. 837–858 (1995)

[2] A. N. Tikhonov. On the stability of inverse problems. Doklady Akademii Nauk SSSR, 39(5): pp. 195–198 (1943)

[3] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. http://arxiv.org/abs/1207.0580 (2012)

[4] L. Breiman. Bagging predictors. Machine Learning, 24(2): pp. 123–140 (1994)

[5] C. M. Bishop. Training with noise is equivalent to Tikhonov regularization. Neural computation, 7(1): pp. 108–116 (1995)

[6] A. Livnat, C. Papadimitriou, N. Pippenger, and M. W. Feldman. Sex, mixability, and modularity. Proceedings of the National Academy of Sciences, 107(4): pp. 1452–1457 (2010)

[7] S. Wager, S. Wang, and P. Liang. Dropout training as adaptive regularization. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, Advances in Neural Information Processing Systems 26, pp. 351–359 (2013)

[8] P. Baldi and P. J. Sadowski. Understanding dropout. In Advances in Neural Information Processing Systems 26, pp. 2814–2822 (2013)

[9] D. Rumelhart, G. Hintont, and R. Williams. Learning representations by back-propagating errors. Nature, 323(6088): pp. 533–536 (1986)

[10] G. An. The effects of adding noise during backpropagation training on a generalization performance. Neural Computation, 8(3): pp. 643–674 (1996)

[11] Y. Raviv and N. Intrator. Bootstrapping with noise: An effective regularization technique. Connec- tion Science, 8(3-4): pp. 355–372 (1996)

[12] A. F. Murray and P. J. Edwards. Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training, IEEE Trans. on Neural Networks, 5(5): pp.792–802 (1994)

[13] K. Matsuoka. Noise injection into inputs in back-propagation learning. IEEE Trans. on Systems, Man and Cyberetics, 22(3): pp. 436–440 (1992)

[14] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. Journal of Machine Learning Research, 11: pp. 3371–3408 (2010)

[15] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. Science, 313(5786): pp. 504–507 (2006)

[16] P. Vincent, H. Larochelle, Y. Bengio, and P. A. Manzagol. Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th international conference on Machine learning, ICML 08, New York, NY, USA, pp. 1096–1103 (2008)