Adaptive Noise Level for Stacked Denoising Autoencoder

Qianjun Zhang\textsuperscript{1,a} and Lei Zhang\textsuperscript{2,b}

Machine Intelligence Laboratory, College of Computer Science, Sichuan University, Chengdu 610065, P. R. China
\textsuperscript{a}zqjblue@163.com, \textsuperscript{b}leizhang@scu.edu.cn

Keywords: Machine learning, Denoising autoencoder, Adaptive noise level, Annealing, Feature combination

Abstract. The stacked denoising autoencoder (SDAE) is a slight modification of stacked autoencoder, which is trained to reconstruct a clean version of an input from its corrupted version. It has been successfully used to learn new representations for an unsupervised framework. However, the noise level in the SDAE is determined by experience and remains fixed throughout the training process. To address this limitation, we present an adaptive stacked denoising autoencoder based on the principle of annealing (AdaptiveSDAE), a novel method of adaptively obtaining the noise level. This is achieved by first computing the average noise level for each epoch using a linear average noise level function based on the principle of annealing; and second calculating the noise level for each input neuron based on the average noise level, and the contribution of the input neuron to the activation of hidden neurons (which depend on the input neuron’s value and the weights). Thus, the network includes a combination of features at multiple scales. The experimental results show that our proposed AdaptiveSDAE performed better than SDAE and other unsupervised feature learning methods.

1. Introduction

Research in machine learning has increasingly focused on obtaining features using unsupervised or supervised learning; the performance of these methods is heavily dependent on the choice of features on which they are applied. The ability to learn multiple levels of feature representations in a hierarchical structure would make it easier to extract useful information when building classifiers or other predictors. A good feature is also one that is useful as input to a supervised predictor.

A denoising autoencoder (DAE)[1] is based on the concept that a good representation should contain enough information to reconstruct corrupted versions of the original input, thus making the autoencoder robust. Despite impressive practical results with denoising autoencoders in many studies[2,3,4], the noise level throughout the training process is determined by experience, and remains unchanged. For a digit recognition task, Vincent et al. (2010) [5] noticed that a low level of noise led to local blob detectors, while increasing noise resulted in the detectors of strokes or parts of digits. They also recognized that extreme levels of noise (either too low or too high) degrade the representation. The relationship between the level of noise and the spatial extent of the filters was also noted by Karklin and Simoncelli (2011) [6] for a different feature learning model. Therefore, our aim is to learn features at different levels of scale or at different levels of granularity.

Regarding the choice of noise level, researchers have proposed many new methods. Geras et al. (2014)[7] proposed a scheduled denoising autoencoder, which starts with a high level of noise that lowers as training progresses. Chandra et al. (2014)[8] proposed an adaptive stacked denoising autoencoder; the noise level of each input neuron computed based on the weights connecting the input neuron to the hidden layer.

In this paper, we propose an adaptive denoising autoencoder, in which the noise level for each input neuron is calculated independently. It is assumed that the activation value of the hidden neurons is determined by two factors: one is the input neuron value, and the other is the weight connected with the hidden neuron. At the same time, the principle of annealing[9] is used to compute the average
noise level of the input neurons at every epoch, which is kept high during the initial training phase, as training progresses, noise is slowly reduced.

Our contributions are as follows: i) We propose a linear average noise level function for computing the average noise level based on the principle of annealing; ii) We provide a calculation equation of adaptive noise level for each input neuron. This is based on the average noise level, and the analysis of the relationship between the noise level for each input neuron and the contribution of the input neuron to the activation value of the hidden neurons (which depend on the input neuron’s value and the weights).

Experimentally, we evaluated our algorithm on variants of MNIST dataset[10]. We made comparisons with SDAE, and other unsupervised feature learning methods. We found that AdaptiveSDAE learned a better representation. For classification tasks, the representation from AdaptiveSDAE yielded a lower test error than SDAE and the other unsupervised feature learning methods.

2. Background

A denoising autoencoder (DAE) [1] is an improved autoencoder, which is robust and can accurately reconstruct corrupted original input data. To reconstruct the original input, the hidden denoising autoencoder layer learns a good representation of input. Training a denoising autoencoder in a layer-wise manner by stacking is called a stacked denoising autoencoder (SDAE). The detailed process of DAE is as follows.

Encoder: The initial input \( x \in \mathbb{R}^d \) is corrupted into \( \tilde{x} \) by means of a stochastic mapping \( \tilde{x} = p(\tilde{x}|x,v) \). A nonlinear transfer function \( f \) that transforms the corrupted input vector \( x \) into hidden representation \( y \) is called the encoder, and is given by:

\[
y = f(W\tilde{x} + b), \tag{1}
\]

where \( f \) is the encoding function, the matrix \( W \in \mathbb{R}^{d' \times d} \) is the weights, the vector \( b \in \mathbb{R}^d \) is the biases, and \( W \) and \( b \) are the parameters of the network.

Decoder: A function \( g \) that "reconstructs" the input vector from the hidden representation is called the decoder, and is given by:

\[
z = g(W'y + b'), \tag{2}
\]

where \( g \) is the decoding function, matrix \( W' \in \mathbb{R}^{d' \times d} \) and vector \( b' \in \mathbb{R}^d \). Typically, \( W \) and \( W' \) are constrained by \( W' = W^T \), which has been justified theoretically by Vincent (2010)[5].

The typical training criterion for DAE is minimization of the reconstruction error \( L(x,z) \), with respect to some loss \( L \), typically either squared error or the binary cross-entropy[5]. For continuous \( x \), squared error can be used:

\[
L(x,z) = \|x - z\|^2. \tag{3}
\]

For binary \( x \), it is common to use the cross-entropy loss:

\[
L(x,z) = -\sum_{i=1}^{d} (x_i \log z_i + (1-x_i)\log(1-z_i)). \tag{4}
\]

After training the denoising autoencoder, activation values of the hidden layer give a good alternate representation of the input.

While training the denoising autoencoder, noise is introduced in the input, and the noise level is a hyper-parameter that is kept unchanged during the training process. Noise, as described by Vincent et al.(2008)[1], includes additive isotropic Gaussian noise, salt and pepper noise for gray-scale images, and masking noise; masking noise has been used in most of the simulations. For example, we get the corrupted version by defining a universal function representation of the noise distribution \( p(\tilde{x}|x,v) \), and the amount of corruption is controlled by a parameter \( v \), which is the
parameter that we want to tune. That is, for each \( i = 1, 2, \ldots, d \), we sample \( \tilde{x}_i \) independently as follows:

\[
\tilde{x}_i = \begin{cases} 
0, & p = v \\
x_i, & p = (1 - v)
\end{cases},
\]

(5)

where symbol \( p \) means probability. It means that the corrupted input version \( \tilde{x} \) is determined by the parameter \( v \). If \( v \) is small, then the input is slightly corrupted during training and \( \tilde{x} \) is similar to the input; if \( v \) is high, then the input is heavily corrupted.

3. Proposed Algorithm

3.1 Motivation of the Algorithm

In a denoising autoencoder, the noise level is determined by experience. During the overall training process it remains unchanged, which leads to the final result representing only the best result under the given noise level; it does not represent the global optimum result. In addition, we know that extremely low or high levels of noise degrade the representation; therefore, we wish to learn features at different levels of granularity.

Thus, using different noise levels is very important during the overall training process, particularly in determining the different noise levels. For solving this problem, we propose an improved algorithm.

3.2 Description of the Algorithm

Our algorithm is based on a standard stacked denoising autoencoder with different average noise levels, \( V_1 \geq V_2 \geq \cdots \geq V_E \), where \( E \) is the epoch of training. The goal is to learn a good representation that combines best aspects at the different levels of noise, with each noise level obtained by our adaptive rules. \( V_1 \) is the initial noise level and is the highest noise level that corrupts most of the input, \( V_E \) is the last and the lowest noise level, and \( V_i \) and \( V_E \) are computed by the classification error of the validation data.

In our algorithm, we first compute the average noise level based on the principle of annealing[9], which has been applied to clustering problems. Similar to the cooling function of the annealing process, the most commonly used average function is the linear function. The linear average noise level equation is as follows:

\[
V_e = V_i - (V_i - V_E) \times \frac{e - 1}{E - 1}.
\]

(6)

Then, based on the average noise level and the analysis of the relationship between the noise level for each input neuron and the contribution of the input neuron to the activation value of the hidden neurons, the noise level for each input neuron is calculated independently. In determining the activation function, activation values of hidden neurons depend on the input neurons value and the weights, the relationship is shown in Fig. 1.

![Diagram](https://via.placeholder.com/150)

Figure 1. The contribution of input neuron values and weights to the activation value of hidden neurons. \( c_j \) denotes the contribution of the \( j \) input neuron to the activation value of hidden neurons, which depend on the input neuron value and weights.
We obtain the calculation equation of the contribution of the $j^{th}$ input neuron to the activation value of hidden neurons, denoted by $c_j$.

$$c_j = \sum_i W_{ij}x_i.$$  \hspace{1cm} (7)

If $c_j$ is larger than the other values, the impact of the $j^{th}$ input neuron on the activation values of the hidden neurons is higher. This also means that the $j^{th}$ input neuron noticeably affects the impact of other input neurons. To enable other input neurons to play a greater role and make the algorithm learn more features, we allow the noise level of the $j^{th}$ neuron to be higher.

The calculation equation of the noise level of the $j^{th}$ input neuron at the $e^{th}$ epoch is as follows:

$$P_j = \begin{cases} 
\hat{c}_j \times V_e + V_e, & V_e \leq 0.5 \\
\hat{c}_j \times (1 - V_e) + V_e, & V_e \geq 0.5 
\end{cases}$$

\hspace{1cm} (8)

where $\hat{c}_j = \hat{c}_j - \bar{c}$, $\hat{c}_j = \frac{c_j}{\max c_j}$, $\bar{c} = \frac{\sum_j \hat{c}_j}{d}$, $d$ is the number of input neurons, the goal of $\hat{c}_j$ is to rescale the data along each data dimension so that the final data vectors lie in the range [0,1]; and $\bar{c}$ is the mean of $\hat{c}_j$. When $\hat{c}_j$ is larger than $\bar{c}$, then $\hat{c}_j$ is larger than zero, so the noise level of the $j^{th}$ input neuron $P_j$ is larger than the average noise level $V_e$; when $\hat{c}_j$ is equal to $\bar{c}$, $\hat{c}_j$ is equal to zero, the noise level of $j^{th}$ input neuron $P_j$ is equal to the average noise level $V_e$, otherwise $P_j$ is smaller than $V_e$. $V_e$ denotes the average noise level at the $e^{th}$ epoch, which is slowly reduced as the training progresses.

Our algorithm is summarized as follows:

|Algorithm 1 AdaptiveSDAE algorithm|
|---|
|1: for epoch $e = 1$ to $E$ do |
|2: \hspace{0.5cm} Calculate the average noise level $V_e$ using Eq. 6. |
|3: \hspace{0.5cm} for each input neuron $j$ do |
|4: \hspace{1cm} Calculate the noise level $P_j$ |
|5: \hspace{1cm} $P_j = \begin{cases} 
\hat{c}_j \times V_e + V_e, & V_e \leq 0.5 \\
\hat{c}_j \times (1 - V_e) + V_e, & V_e \geq 0.5 
\end{cases}$ |
|6: \hspace{1cm} Using noise level $P_j$ for the input neuron $j$. |
|7: \hspace{0.5cm} end for |
|8: \hspace{0.5cm} end for |

4. Experiments

In this section, we evaluate the performance of AdaptiveSDAE on MNIST[10] datasets.

4.1 Experimental setup

We used the MNIST handwritten digit dataset (which consists of handwritten digit images in gray-scale with 28×28 pixels) and its variants, a smaller subset of the original MNIST (basic), with added variations such as rotation (rot), and the addition of a background composed of random pixels (bg-rand) or patches extracted from a set of images (bg-img)[10]. We performed experiments on the same benchmark of classification problems as used in Vincent et al.(2008)[1]. The image datasets are detailed below; each of them has a training set (for tuning parameters), a validation set (for tuning hyper-parameters) and a test set (for reporting generalization performance). For a standard MNIST digit, we randomly separated every training set into 50,000 training cases and 10,000
validation cases, and 10,000 cases for testing; other variants were divided into a training, validation, and test set (10,000, 2,000, and 50,000 examples, respectively).

The network architecture in our experiment consisted of an input layer, three hidden layers and an output layer; the number of hidden neurons was 1,000, 1,000, and 1,000 for the MNIST dataset. A sigmoid nonlinear function was used in the encoder and decoder; masking noise was used for image corruption. For optimization, stochastic gradient descent (SGD) with mini-batches was used. Learning rate and additional hyper-parameters were selected through the error of the validation set.

The noise level $V_i$ and $V_E$ were chosen by the least validation error, the results are given in Table 1.

| Level | MNIST | basic | rot | bg-rand | bg-img |
|-------|-------|-------|-----|---------|--------|
| $V_i$  | 0.7   | 0.7   | 0.7 | 0.9     | 0.6    |
| $V_E$  | 0.1   | 0.1   | 0.4 | 0.4     | 0.4    |

### 4.2 Effect of noise level

To understand the effect of the noise level, we trained the SDAE with different noise level strategies on the MNIST dataset. We first fixed the noise levels and chose one of the following values during the training process: $v = 0.1, 0.2, \ldots, 0.9$; then we used our algorithm AdaptiveSDAE. Fig. 2 shows the visualization of the features of different noise level strategies.

![Figure 2](image)

From Fig. 2(b), we can see that for large noise levels (e.g., $v=0.7$), the algorithm learns global features. Nevertheless, as we decreased the noise level (e.g., $v=0.1$), the input was slightly corrupted and the features tended to be more local, as can be seen in Fig. 2(a). Finally, for AdaptiveSDAE, the algorithm learned a combination of the global and local features, as shown in Fig. 2(c).

### 4.3 Comparison of performance

To demonstrate the effectiveness of our algorithm and the quality of the features learned by our algorithm, we compared the test error of our algorithm with the other unsupervised feature learning methods.

We first compared the test error of our algorithm with a standard stacked denoising autoencoder (SDAE), which is the classical stacked denoising autoencoder with the same network architecture as our algorithm, the difference being the choice of noise level. In the SDAE, the noise level is unchanged during the whole training process. In our experiment, we fix the noise level and choose the following noise levels one by one during the training process: the noise level $v = 0.1, 0.2, \ldots, 0.9$. We call them SDAE(0.1), SDAE(0.2), \ldots, and SDAE(0.9), respectively.

To further validate the effectiveness of our algorithm, we compared it with established other unsupervised feature learning methods. We compared the results of dropout neural networks(DropoutNN)[11], SAE-3[5] and optimumSDAE[5]. OptimumSDAE is an optimum stacked denoising autoencoder, which is obtained using optimum hyper-parameters such as the
number of neurons in each hidden layer, the learning rate, and the number of epochs; to obtain the
optimum hyper-parameter, an enormous amount of processing time is required.

| Model          | MNIST  | basic | rot  | bg-rand | bg-img |
|----------------|--------|-------|------|---------|--------|
| AdaptiveSDAE   | 1.32   | 2.59  | 12.1 | 11.68   | 16.13  |
| SDAE(0.1)      | 2.01   | 2.98  | 13.45| 15.45   | 19.32  |
| SDAE(0.5)      | 1.75   | 3.01  | 14.25| 14.87   | 20.95  |
| DropoutNN[11]  | 1.35   | -     | -    | -       | -      |
| SAE-3[5]       | 1.40   | 3.46  | 10.30| 11.28   | 23.00  |
| OptimumSDAE[5] | 1.28(0.25) | 2.84(0.1) | 9.53(0.25) | 10.3(0.4) | 16.68(0.25) |

The results for the various datasets are compared in Table 2, the bold-faced values show the better performance for different algorithms. For clarity, we do not show the test error of SDAE (0.2), SDAE (0.3), SDAE (0.4), SDAE (0.6), SDAE (0.7), SDAE (0.8), and SDAE (0.9), which yield a higher test error than SDAE (0.5).

Table 2 provides the results for the MNIST dataset and its variants; the results show that our algorithm learned better representations and yielded a lower test error.

These results are comparable to standard SDAE. Our algorithm outperforms SDAE. It proved that scheduling noise level from high to low is feasible, and also proved the effectiveness of our algorithm.

These results are also comparable to the DropoutNN[11], SAE-3[5] and optimumSDAE[5]. DropoutNN achieved an error rate of about 1.35% and yielded higher test error rates than our algorithm. For some of the datasets (such as basic and bg-img), the performance of AdaptiveSDAE is better than the optimum SDAE and SAE-3. These further demonstrate the effectiveness of our algorithm.

From the results, we also see that the performance was strongly correlated to the noise level. By comparing the results of different noise level strategies, we confirmed that using the features learned with different noise levels indeed helps classification.

5. Discussion and Conclusion

In this work, we proposed a novel method called adaptive denoising autoencoder, which uses different noise levels for each input neuron at every epoch during training, and overcomes the limitations of SDAE in which the noise level is kept unchanged. The noise level for each input neuron depends on the contribution of the input neuron to the activation of hidden layers, which are quantized by the sum of the product of outgoing weights from the input neuron and its own value. We have demonstrated that AdaptiveSDAE can learn better features, and we can use the representations to achieve better classification results. The experiment shows that AdaptiveSDAE outperforms SDAE and the other unsupervised feature learning methods.

Acknowledgements

This work was supported by the National Key R & D Program of China (grant 2016YFC0801800); Fok Ying Tung Education Foundation (grant 151068); and National Natural Science Foundation of China (grant 61332002).

References

[1] P. Vincent, H. Larochelle, Y. Bengio and P.A. Manzagol, in: International Conference on Machine Learning, (2008), pp. 1096-1103.

[2] K. Cho: Eprint Arxiv (2013).
[3] M. Chen, Z. Xu, K. Weinberger, F. Sha: Computer Science (2012).

[4] X. Glorot, A. Bordes, Y. Bengio, in: Proceedings of the 28th International Conference on Machine Learning (2011).

[5] P. Vincent, H. Larochelle, Y. Bengio, P.A. Manzagol: The Journal of Machine Learning Research, Vol. 11 (2010), p.3371-3408.

[6] Y. Karklin, E. P. Simoncelli, in: Advances in Neural Information Processing Systems Vol. 24 (2011), pp. 999-1007.

[7] K. J. Geras, C. Sutton: Eprint Arxiv (2014).

[8] B. Chandra, R.K. Sharma, in: Neural Information Processing, Vol. 8834 (2014), pp. 535-542.

[9] K. Rose, in: Proceedings of the IEEE, Vol. 86(1998), pp. 2210-2239.

[10] Y. Bengio, Y. Lecun: Large-scale kernel machines, Vol. 34 (2007), pp. 321-359.

[11] N. Srivastava, G. Hinton, A Krizhevsky, I. Sutskever, R. Salakhutdinov: Journal of Machine Learning Research, Vol. 15(2014), pp. 1929-1958.