Effect of Additive Noise for Multi-Layered Perceptron with AutoEncoders

Motaz SABRI†(a), Nonmember and Takio KURITA†, Fellow

SUMMARY This paper investigates the effect of noises added to hidden units of AutoEncoders linked to multilayer perceptrons. It is shown that internal representation of learned features emerges and sparsity of hidden units increases when independent Gaussian noises are added to inputs of hidden units during the deep network training. It is also shown that the weights that connect the contaminated hidden units with the next layer have smaller values and outputs of hidden units tend to be more definite (0 or 1). This is expected to improve the generalization ability of the network through this automatic structuration by adding the noises. This network structuration was confirmed by experiments for MNIST digits classification via a deep neural network model.

key words: AutoEncoder, deep neural network, additive noise, gaussian noise, classification

1. Introduction

Since the neural network regularization has been introduced, many techniques have been applied to generalize networks beyond the training data including L1 and L2 regularization, dropout [1] and preventing the co-adaptation of detectors feature [2], [3]. Many regularizing methods focus on optimizing network target function during the training [1], [4]–[7]. One commonly used approach is introducing training criterion that discourages converging the network into building a complicated solution and suppresses the number of effective weights by directing the learning algorithm towards a solution with the least possible amount of zero weights such as weight decay [8]. Other approaches encourage less co-adapting between units and reduces overfitting such as Dropout [3].

One more powerful method of improving neural network generalization ability is noise injection. Kurita et al. [9] inspected the backpropagation learning behavior while injecting noises into the inputs of the hidden units during MLP training. They concluded that the connection from hidden units to output units holds smaller weights while the connections from input units to hidden units holds larger weights values. They confirmed the network ability of automatically structurize itself by simply adding the noises. They conducted pattern classification and logic Boolean function learning to confirm their results. Kartik et al. [10] introduced the Noisy convolutional neural network algorithm which utilizes noisy expectation maximization result to generate a hyperplane in noise space that extracts helpful noise. They showed that injecting noise into neural network boosts the backpropagation training of a neural network.

Poole et al. [11] injected multiple noise configuration into neural networks and studied the learning process. They also analyzed generalization ability after injecting noise into networks with de-noising, contractive, and sparse AutoEncoders. They showed that noisy learning outperforms denoising learning. They tested their noise scheme against MNIST, and CIFAR-10 and achieved comparable results. However, they haven’t concluded the place in which the noise is best to be injected and the reason why the noise is proven to enhance the training process.

Motivated by the need of expanding neural network generalization ability, we analyze a simple technique of selectively injecting Gaussian noise to multiple joints within a deep neural network. We inspect the relation between the network learning and noise injection mathematically. We also conduct pattern classification experiments that visualize the internal representation of the network and its hidden unit activation sparsity. Our mathematical modeling clarifies why learning process improves upon noise injection into specific joints within the network. Our experiments steadily result in lower loss and better accuracy upon injecting Gaussian noise into selected joints of neural network to classify MNIST [12] digits. Our experiments covered single layered AutoEncoder and Stacked AutoEncoder both connected to an MLP during separate experiments of classifying MNIST digits which come with different handwriting styles. This matches the finding of many researches that injecting noise to simple neural networks.

The contribution of this research is scoped towards showing improvement to the classification accuracy of deep network structures with standard gradient descent upon selective noise injection-point. To the best of our knowledge, our noise injection placement and tuning have not been used before in the training of deep neural networks. It shows better visualization for the hidden layer representation and more sparsity for the hidden unit’s activation when compared to noiseless case. We hope other researchers will utilize this method to achieve similar improvements in other research schemes.

This paper is organized as follows. In Sect. 2, relevant models, learning methods and noise injection techniques are reviewed as related work. The details of the approach used in this paper are explained in Sect. 3. The mathematical rep-
representation of the network for noiseless and noisy training is mentioned in Sects. 3.1 and 3.3 respectively. Details of the experiments are described in Sect. 4. Finally, conclusion is shown in Sect. 5.

2. Related Work

We analyze the average behavior of the backpropagation learning to noise injected into specific joints of AutoEncoders connected to multilayer perceptron (MLP). Optimizing MLP for learning is based on selecting the suitable architecture and the connection weights via the minimizing training error and a penalty term [13]. Many techniques have been proposed to extend MLP abilities by additional components. Kurita et al. introduced AutoEncoder with competitive hidden units with classifiers as its input [14]. This formed a mixture of classifier with ability of self-organizing in which one of the classifiers is selectively activated according to the input feature. Vincent et al. highlighted a method for building deep networks based on stacking layers of denoising AutoEncoders [15]. The AutoEncoders reduce the dimensionality of input data to a smaller-dimensional code space at the hidden layer. Such reduction drops features with less importance and highlights the feature with higher value. This increases the network ability to observe features that MLP alone wasn’t able to learn and ultimately improves learning and generalization ability [16], [17].

Interfering with the training process of a deep neural network has many shapes and has been shown to improve the performance of learning process [18]. Random Noise injection to the weights or the hidden units has been utilized in many neural network researches for many years [15], [19]–[21]. Kurita et al. [9] injected noise into the hidden layers of an MLP and showed the network ability to get automatically structured by simply adding the noises and therefore improving the generalization ability of the network. Adding more randomness gives higher potential for escaping local minima or passing through the early training phase in the most optimum phase.

Zeyer Et al. [20] introduced a low-overhead technique of adding gradient noise for neural network which is found to avoid overfitting and result in lower training loss. Their method allows a fully-connected, poorly initialized 20-layer deep network to be trained with standard gradient descent. They achieved improvements for many complex models. They were able to achieve a 72% reduction in error rate over a carefully-tuned baseline on a challenging question-answering task. In a parallel experiment, they were able to double the number of accurate binary multiplication models learned across 7,000 random restarts. The noise injection scheme recently started tackling modern deep networks and its advantages haven’t been fully documented in terms of mathematical representation and experiments variety [4], [20].

Guided by this fact we inspect a network with AutoEncoders and MLP behavior of learning update rules and noise injection mathematically. We assume the AutoEncoders ability to reduce dimensionality and highlight useful features will be strengthened due to noise injection. We aim to clarify why injecting noise into certain joints of network with AutoEncoders and MLP benefits the learning process. The contaminated neural network training proposed in this paper was based on well-known noise injection theory in the neural computing community [11], [15], [22]. The visualizations of network hidden layers weights and activation sparsity were obtained to show the feature representation considering the location of noise injection. Experiments on MNIST [12] highlight benefits of using noisy AutoEncoders by learning useful features for classification. The effect of adding random Gaussian noise are predicted theoretically and proven practically through network learning updated rules analysis and experiments. We assumed adding Gaussian noise effect will be comparable with well-known advantages of stochastic gradient descent as a learning algorithm by reducing overfitting.

3. Multi-Layered Noise Enabled Neural Network

In this paper, we theoretically and practically evaluate the effect of injecting noise into hidden and input layers of single layered and stacked AutoEncoders each connected to MLP neural network. The input data passes through an AutoEncoder first. The AutoEncoder learns detailed and useful features of the input data, in our case, the MNIST database [12]. The hidden layer of the AutoEncoder is connected to an MLP that decides which class each input data belongs to based on a teacher signal.

In order to enable such architecture to realize the variety of handwriting styles, the AutoEncoder must have the ability to select one of the classifier depending on a special feature within each input. The AutoEncoder is trained to reduce the dimension of the representation of the input data by considering the outputs of the hidden layer. Figure 1 shows the symbolic variables of the network. As referred by Kurita

![Fig. 1](Image) AutoEncoders have proven its ability to highlight features that can’t be identified by other neural network structure. Linking the classifier to AutoEncoder hidden layer allows the low dimension features to be extracted.
et al. this structure introduces competition among the units in the hidden layer [14] and their outputs can be regarded as the signal for selecting the classifiers. AutoEncoders main training criterion is minimizing the reconstruction error with respect to some loss. We will describe the loss functions we use in our experiment in the next section.

3.1 Network Modeling

Objects that lay within the same category can be classified based on special features within them. When objects have variation within the same feature representation then the training process becomes more complicated. Mutli-shaped representations can be encoded and decoded by AutoEncoders to highlight unclear features and be classified later by logistic units. Each classifier receives the same input vector from the AutoEncoder hidden layer and outputs the classification result as an output vector. The output of the total network is computed as the weighted sum of the outputs of the classifiers. This means that the last layer of the network works like a selector of classifiers. Our network classifies \( K \) classes defined in the set \( C = \{1, 2, \ldots, K\} \).

In our module definition \( x = [x_1 \ x_2 \ \ldots \ x_I]^T \) represents the input feature vector of the classifier and \( q = [q_1 \ q_2 \ \ldots \ q_H]^T \) is the bias vector of the hidden layer and \( r = [r_1 \ r_2 \ \ldots \ r_I]^T \) be the bias vector of the output layer of the AutoEncoder and \( s = [s_1 \ s_2 \ \ldots \ s_K]^T \) be the bias vector of the output layer of the network.

Let \( u_{hi} \) represents the weight of the connection between the \( i \)-th input unit and the \( h \)-th hidden unit therefore we define \( u_h = [u_{h1} \ u_{h2} \ \ldots \ u_{hi}]^T \). Similarly let \( w_{ih} \) represent the weight of the connection between the \( h \)-th hidden unit and the \( i \)-th output unit of the AutoEncoder therefore we define \( w_i = [w_{i1} \ w_{i2} \ \ldots \ w_{ih}]^T \). Let \( v_{kh} \) represent the weight of the connection between the \( h \)-th hidden unit and the \( k \)-th output unit of the network, therefore we define \( \upsilon_k = [\upsilon_{k1} \ \upsilon_{k2} \ \ldots \ \upsilon_{kh}]^T \).

Lets define the weights vectors as matrices to define the activation functions, the matrix \( U \) is given by:

\[
U = [u_1 \ u_2 \ \ldots \ u_H]^T
\]

the matrix \( W \) is given by:

\[
W = [w_1 \ w_2 \ \ldots \ w_I]^T
\]

and the matrix \( V \) is given by:

\[
V = [v_1 \ v_2 \ \ldots \ v_K]^T
\]

To define the activation function of the AutoEncoder hidden layer let’s define \( \eta_h \) as shown in Eq. (1):

\[
\eta_h = u_h^T x + q_h
\]

We define the hidden layer output vector \( g \) as function of input \( x \) as shown in Eq. (2):

\[
g(x) = [g_1(x) \ g_2(x) \ \ldots \ g_H(x)]^T = f(Ux + q)
\]

accordingly the hidden units have the sigmoid activation described by Eq. (3):

\[
g_h(x) = f(\eta_h) = \frac{1}{1 + e^{-\eta_h}}
\]

Similarly to define the activation function of the output layer of the AutoEncoder let’s define \( \xi_i = w_i^T g + r_i \).

Given the output vector of AutoEncoder \( z \) as function of input \( g \) as follows:

\[
z(g) = [z_1(g) \ z_2(g) \ \ldots \ z_I(g)]^T = f(Wg + r)
\]

accordingly the output units of the AutoEncoder have the linear activation of \( z_i(g) = f(\xi_i) = \xi_i \).

To define our activation function of the network output let’s define \( \rho_k = v_k^T g + s_k \).

We define the output vector \( y \) as function of AutoEncoder hidden layer output \( g \) as follows:

\[
y(g) = [y_1(g) \ y_2(g) \ \ldots \ y_K(g)]^T = f(Vg + s)
\]

accordingly the output units of the network have the softmax activation as shown in (4):

\[
y_k(g) = f(\rho_k) = \frac{e^{\rho_k}}{\sum_{k=1}^{K} e^{\rho_k}} \quad (k = 1, 2, 3, \ldots, K)
\]

Let \( t = [t_1 \ t_2 \ \ldots \ t_K]^T \in \{0, 1\}^K \) denote a binary vector composed of teacher signals in which \( t_k \) equals to 1 if the current input vector is classified under the \( k \)-th class and 0 otherwise.

The training data \( D = \{<x_p, t_p> \in C \mid p = 1 \ldots N\} \) where \( N \) is the number of samples. Each set of \( N \) input samples comes with its own output set and training signals, therefore the matrix \( X \) is given by:

\[
X = [x_1 \ x_2 \ \ldots \ x_N]^T
\]

The \( n \)-th \( \eta \) term is given as \( \eta_{nh} \) and outputs of the hidden layer given input \( x_n \) is given by \( g_n = g(x_n) \), therefore the Matrix \( G \) is given by:

\[
G = [g_1 \ g_2 \ \ldots \ g_N]^T
\]

outputs of the AutoEncoder given output of hidden layer \( g_n \) is given by \( z_n = z(g_n) \), therefore the matrix \( Z \) is given by:

\[
Z = [z_1 \ z_2 \ \ldots \ z_N]^T
\]

The \( n \)-th \( \xi \) term is given as \( \xi_{ni} \).

Outputs of the network given hidden layer vector \( g_n \) is given by \( y_n = y(g_n) \), therefore the Matrix \( Y \) is given by:

\[
Y = [y_1 \ y_2 \ \ldots \ y_N]^T
\]

The \( n \)-th teacher signal is given as \( t_n \) and the \( n \)-th \( \rho \) term is given as \( \rho_{nk} \).

We adjust the weights of the AutoEncoder to minimize the mean squared error (MSE) on training set which is given
via the form $\ell_1$ in Eq. (5):

$$\ell_1 = \frac{1}{2} \sum_{n=1}^{N} \sum_{i=1}^{I} (x_{ni} - z_{ni})^2$$  \tag{5}$$

The cost function of the MLP is given via cross entropy as $\ell_2$ and its shown in Eq. (6):

$$\ell_2 = -\sum_{n=1}^{N} \sum_{h=1}^{K} t_{nh} \log y_{nh}$$  \tag{6}$$

3.2 Weight Update Rules

The network learning process of this network contains updating 3 weights: $u_{hi}$, $w_{ih}$ and $v_{kh}$. We are interested in minimizing the error function with respect to the weights.

3.2.1 Update Rule for $u_{hi}$

While the error back-propagates from both outputs (AutoEncoder and MLP), the overall error function is given in Eq. (7):

$$L = \ell_1 + \lambda \ell_2$$  \tag{7}$$

therefore minimizing the error function with respect to the weight $u_{hi}$ is given in Eq. (8):

$$\frac{\partial L}{\partial u_{hi}} = \sum_{n=1}^{N} \omega_{nh} \delta_{mh} x_{ni} + \lambda \omega_{nh} \delta_{mh} y_{nh}$$  \tag{8}$$

Given that the term $\delta_{mh} = \sum_{i=1}^{I} (x_{ni} - z_{ni}) w_{ih}$ and $\delta_{mh} = \sum_{k=1}^{K} (y_{nk} - t_{nk}) u_{kh}$ that is the weighted delta and $\omega_{nh} = g_{nh}(1 - g_{nh})$. The updated weight due to the back propagation is given in Eq. (9):

$$\Delta u_{hi} = -\alpha \frac{\partial L}{\partial u_{hi}}$$  \tag{9}$$

were $\alpha$ is the learning rate, the Eq. (9) leads to the following conclusion:

$$u_{hi,new} = u_{hi,old} - \alpha \sum_{n=1}^{N} \omega_{nh} x_{ni}(\delta_{mh} + \lambda \delta_{mh})$$  \tag{10}$$

3.2.2 Update Rule for $w_{ih}$

This time we are interested in minimizing the error function with respect to the weight $w_{ih}$. Similarly the updated weight due to the back propagation is given by Eq. (11):

$$w_{ih,new} = w_{ih,old} - \alpha \sum_{n=1}^{N} \delta_{ni} y_{nh}$$  \tag{11}$$

Where $(x_{ni} - z_{ni})$ is the delta term, referenced as $\delta_{ni}$.

3.2.3 Update Rule for $v_{kh}$

While the error back-propagates from both outputs (AutoEncoder and MLP), Given that $\delta_{nk} = (y_{nk} - t_{nk})$ is the delta term, the updated weight due to the back propagation is given by Eq. (12):

$$v_{kh,new} = v_{kh,old} - \lambda \sum_{n=1}^{N} \delta_{nk} y_{nh}$$  \tag{12}$$

3.3 Additive Noise -Hidden Layer

The simplest form of neural network contamination can be evaluated by injecting noises to hidden units in Multilayer Perceptron (MLP) with one hidden layer [9]. Observing the expectation of the partial derivatives of the squared error with respect to the weights shows the weights from the hidden units to the output units tend to get smaller. On the other side the outputs of the hidden units tend to be 0 or 1 [23]. We extend this basic form to deep neural network as shown in Fig. 2.

Adding noise to the hidden layer contaminates the units by identical independent Gaussian noise with zero mean and variance of $\delta^2$.

Denote $\epsilon_h$ be the noise added to $\eta_h$, and the contaminated vector $\hat{\eta}_h = [\eta_1 + \epsilon_1 \cdots \eta_H + \epsilon_H]^T$, $\epsilon = [\epsilon_1 \cdots \epsilon_H]^T$ and the signal transmitted to hidden unit is given by Eq. (13):

$$\hat{\eta}_h = u_h^T x + q_h + \epsilon_h = \eta_h + \epsilon_h$$  \tag{13}$$

Given $\omega_{nh} = g_{nh}(1 - g_{nh})$ and $v_h = g_h(1 - g_h)(1/2 - g_h)$, the term $\bar{g}_{nh}$ is given as, the output vector $\bar{g}$ is given by Taylor expansion as in Eq. (14):

$$\bar{g}_h \approx g_h + \omega_{nh} \epsilon_h + v_{nh} \epsilon_h^2$$  \tag{14}$$

The i-th output of the AutoEncoder $\hat{z}_{ni}$ now becomes as shown in Eq. (15):

$$\hat{z}_{ni} \approx w_i^T \bar{g}_h + r_i = z_{ni} + \sum_{h=1}^{H} w_{ih}(\omega_{nh} \epsilon_h + v_{nh} \epsilon_h^2) = z_{ni} + \Delta z_{ni}$$  \tag{15}$$

On the other hand, the output units of the network are also affected by the noise injection. This is represented in Eq. (16):

$$\bar{p}_k \approx v_k^T \bar{g} + s_k = \rho_k + \sum_{h=1}^{H} v_{kh}(\omega_{nh} \epsilon_h + v_{nh} \epsilon_h^2) = \rho_k + \Delta \rho_k,$$  \tag{16}$$

and

$$\bar{y}_k = \frac{e^{\rho_k}}{\sum_{i=1}^{K} e^{\rho_i}}$$  \tag{17}$$
3.3.1 Update Rule for $u_{hi}$

We are interested in minimizing the error function with respect to the weight $u_{hi}$. To denote the loss function with noise injection, we denote $\tilde{\ell}_1 = \frac{1}{2} \sum_{n=1}^{N} \sum_{i=1}^{I} (\tilde{x}_{ni} - z_{ni})^2$ for AutoEncoder, and $\tilde{\ell}_2 = - \sum_{n=1}^{N} \sum_{k=1}^{K} t_k \log \tilde{y}_k$ for the softmax classifier, given that $\tilde{L} = \tilde{\ell}_1 + \lambda \tilde{\ell}_2$ then

$$\frac{\partial \tilde{L}}{\partial u_{hi}} = \sum_{n=1}^{N} \omega_{nh} \delta_{nh} x_{ni} + \sum_{n=1}^{N} \sum_{j=1}^{J} w_{jh} \Delta z_{nj} \omega_{nh} x_{ni} + \sum_{n=1}^{N} \sum_{j=1}^{J} (1 - 2g_{nh}) \epsilon_{h} + \frac{1}{2} \chi_{nh} \epsilon_{h}^2 \frac{\partial \ell}{\partial u_{hi}}$$

$$\frac{\partial \tilde{L}}{\partial u_{hi}} = \sum_{n=1}^{N} \omega_{nh} \delta_{nh} x_{ni} + \sum_{n=1}^{N} \sum_{j=1}^{J} w_{jh} \Delta z_{nj} \omega_{nh} x_{ni} + \sum_{n=1}^{N} \sum_{j=1}^{J} (1 - 2g_{nh}) \epsilon_{h} + \frac{1}{2} \chi_{nh} \epsilon_{h}^2 \frac{\partial \ell}{\partial u_{hi}}$$

where $\chi_{nh} = 1 - 6g_{nh} + 6g_{nh}^2$. The detailed derivation of Eq. (18) can be found at the appendix.

The expectation value of $\frac{\partial \tilde{L}}{\partial u_{hi}}$ is given in Eq. (19):

$$\mathbb{E} \left[ \frac{\partial \tilde{L}}{\partial u_{hi}} \right] = \mathbb{E} \left[ \frac{\partial \tilde{\ell}_1}{\partial u_{hi}} \right] + \mathbb{E} \left[ \lambda \frac{\partial \tilde{\ell}_2}{\partial u_{hi}} \right]$$

Where $\mathbb{E} \left[ \frac{\partial \tilde{\ell}_1}{\partial u_{hi}} \right]$ equals to:

$$\mathbb{E} \left[ \frac{\partial \tilde{\ell}_1}{\partial u_{hi}} \right] = \frac{\partial \tilde{\ell}_1}{\partial u_{hi}} \left( 1 + \chi_{nh} \sigma^2 \right) + \sum_{n=1}^{N} \sum_{j=1}^{J} w_{jh} \omega_{nh} v_{nh} x_{ni} \sigma^2$$

From Eq. (20), we can see that the term $\chi_{nh} \sigma^2 / 2$ takes negative values when the output $g_{nh}$ is in the range $1/2 - \sqrt{3}/6 < g_{nh} < 1/2 + \sqrt{3}/6$ while it takes positive values when the output $g_{nh}$ approaches to 0 or 1. Therefore, the learning rate is suppressed when $y$ is near 1/2, an uncertain value, and is accelerated when $g_{nh}$ is near 0 or 1, a definite value.

The expectation value of $\frac{\partial \tilde{\ell}_2}{\partial u_{hi}}$ is estimated by Jensen’s inequality as shown in Eq. (21):

$$\mathbb{E} \left[ \log \sum_{n=1}^{N} \sum_{j=1}^{K} e^{\phi_{nj}} \right] \leq \log \left( \mathbb{E} \left[ \sum_{n=1}^{N} \sum_{j=1}^{K} e^{\phi_{nj}} \right] \right)$$

$$\mathbb{E} \left[ \log \sum_{n=1}^{N} \sum_{j=1}^{K} e^{\phi_{nj}} \right] \leq \log \left( \mathbb{E} \left[ \sum_{n=1}^{N} \sum_{j=1}^{K} e^{\phi_{nj}} \right] \right)$$

where $\mathbb{E} \left[ \log \tilde{y}_{nk} \right] = \rho_{nk} + \left( \sum_{k=1}^{H} v_{kh} \sqrt{\nu} \right) \sigma^2 - \mathbb{E} \left[ \log \sum_{n=1}^{N} \sum_{k=1}^{K} e^{\phi_{nk}} \right]$.

The detailed derivation of Eqs. (21) and (22) can be found at the appendix.

From Eq. (22), we can see that the term $\nu_{nh}$ is close to 0 when the output $g_{nh}$ approaches to 0 or 1, thus makes the second term be closer to 0 in this case. Similarly, when $g_{nh}$ is close to 0 or 1, $\nu_{nh}$ is also close to 0, and makes the variance $\mathbb{V}[\tilde{y}_{nk}]$ be closer to 0. In summary, when $g_{nh}$ becomes a definite value, $\mathbb{E} \left[ \log \tilde{y}_{nk} \right]$ is close to $\rho_{nk}$ and that makes $\mathbb{E} \left[ \tilde{y}_{nk} \right]$ is close to $y_{nk}$. Since $\chi_{nh}$ is positive when $g_{nh}$ is close to 0 or 1, we have

$$\mathbb{E} \left[ (\tilde{y}_{nk} - t_{nk}) \frac{\partial \tilde{\ell}_2}{\partial u_{hi}} \right] \leq (y_{nk} - t_{nk}) \frac{\partial \tilde{\ell}_2}{\partial u_{hi}}$$

When the outputs of hidden layer units are uncertain, $g_{nk} \approx 1/2$ for all $h$, the gap between two sides of Jensen’s inequality is almost zero. And also, $\nu_{nh} = 0$ when $g_{nh} = 1/2$. Therefore in Eq. (24) we have

$$\log \mathbb{E} \left[ \tilde{y}_{nk} \right] \approx \rho_{nk} - \log \left( \mathbb{E} \left[ \sum_{n=1}^{N} \sum_{k=1}^{K} e^{\phi_{nk} \mathbb{V}[\tilde{y}_{nk}] / 2} \right] \right)$$

In this case we have $\log \mathbb{E} \left[ \tilde{y}_{nk} \right] \leq \rho_{nk}$, which implies that $\mathbb{E} \left[ \tilde{y}_{nk} \right] \leq y_{nk}$. Since $\chi_{nh} < 0$ when $g_{nh}$ is close to 1/2, we can conclude that the learning rate is suppressed.

3.3.2 Update rule for $v_{kh}$

The expectation value of $\frac{\partial \tilde{\ell}_2}{\partial v_{kh}}$ is given in (25):

$$\mathbb{E} \left[ \frac{\partial \tilde{\ell}_2}{\partial v_{kh}} \right] \approx \frac{\partial \tilde{\ell}_2}{\partial v_{kh}} + (y_{nk} - t_{nk}) v_{nh} \sigma^2.$$
mean and variance of $\sigma^2$. The contaminated vector $\tilde{x} = [x + \epsilon_1 \cdots x + \epsilon_l]'$, $\epsilon = [\epsilon_1 \cdots \epsilon_l]'$ and the signal transmitted to hidden unit is

$$\tilde{y}_h = u^2_1\tilde{x} + q_h = u^2_1 x + u^2_1\epsilon + q_h := \eta_h + \epsilon_h,$$

where $\epsilon_h := u^2_1\epsilon$, similarly we obtain the expectation values of gradients referred to as $E[\frac{\partial \ell_1}{\partial u_{hl}}]$ and $E[\frac{\partial \ell_2}{\partial u_{hl}}]$.

$$E[\frac{\partial \ell_1}{\partial u_{hl}}] = \frac{\partial \ell_1}{\partial u_{hl}} \left(1 + \frac{\chi_{nh}\|u_h\|^2_2\sigma^2}{2}\right)$$

$$+ \sum_{n=1}^{N} \sum_{j=1}^{l} w_{nj}\omega_{nh}\chi_m\sigma^2 \left(\sum_{h' = 1}^{H} w_{nh'}\nu_{nh'}\|u_{h'}\|^2_2\right)$$

$$+ 3\sum_{n=1}^{N} \sum_{j=1}^{l} w_{nj}\omega_{nh}\nu_{nh}\chi_m\sigma^2\|u_h\|^2_2$$

$$+ \sum_{j=1}^{l} w_{nj}\nu_{nh}\chi_m\sigma^2 (26)$$

From Eq. (26), we can see that the term $\chi_{nh}\sigma^2\|u_{h}\|^2_2/2$ takes negative values when the output $g_{nh}$ is in the range $1/2 - \sqrt{3}/6 < g_{nh} < 1/2 + \sqrt{3}/6$ while it takes positive values when the output $g_{nh}$ approaches to 0 or 1. Therefore, the learning rate is suppressed when $y$ is near 1/2, an uncertain value, and is accelerated when $g_{nh}$ is near 0 or 1, a definite value.

The expectation value $E[\frac{\partial \ell_2}{\partial u_{hl}}]$ is described in Eq. (26):

$$E[\frac{\partial \ell_2}{\partial u_{hl}}] =$$

$$\left(\sum_{n=1}^{N} \sum_{k=1}^{K} v_{kh}\|y_{nk} - t_{nk}\| \omega_{nh}\chi_m \left(1 + \frac{\chi_{nh}\sigma^2\|u_h\|^2_2}{2}\right)\right)$$

$$+ 2\left(\sum_{n=1}^{N} \sum_{k=1}^{K} E[\|y_{nk} - t_{nk}\|\nu_{nh}\nu_{nl}u_{hl}]\right)$$

Recall that

$$\tilde{\rho}_{nk} = \frac{e^{\tilde{\theta}_{nk}}}{\sum_{m=1}^{N} \sum_{j=1}^{K} e^{\tilde{\theta}_{mj}}}$$

The softmax operator is to normalize the values $e^{\tilde{\theta}_{nk}}$ in the output units. We now analyze the influence of noise injection on $e^{\tilde{\theta}_{nk}}$ through Eqs. (29) and (30).

$$e^{\tilde{\theta}_{nk}} = e^{\tilde{\theta}_{nk}} \cdot \sum_{m=1}^{N} \sum_{k=1}^{K} e^{\tilde{\theta}_{mk}} \cdot e^{\tilde{\theta}_{nk}\nu_{nk}\epsilon_n^2}$$

$$E[e^{\tilde{\theta}_{nk}}] = e^{\tilde{\theta}_{nk}} E\left[\sum_{m=1}^{N} \sum_{k=1}^{K} e^{\tilde{\theta}_{mk}} \cdot e^{\tilde{\theta}_{nk}\nu_{nk}\epsilon_n^2}\right]$$

When the variance of noise is small, the amplitude of $\epsilon_n^2$ can be negligible. In this case, we can simplify the expectation $E[e^{\tilde{\theta}_{nk}}]$ as shown in Eq. (31):

$$E[e^{\tilde{\theta}_{nk}}] = e^{\tilde{\theta}_{nk}} + \sum_{m=1}^{N} \sum_{k=1}^{K} \nu_{nk}\epsilon_n^2\|u_{h}\|^2_2/2$$

Since the term $\sum_{m=1}^{N} \sum_{k=1}^{K} \nu_{nk}\epsilon_n^2\|u_{h}\|^2_2$ is always nonnegative, it is clear that $E[e^{\tilde{\theta}_{nk}}] \geq e^{\tilde{\theta}_{nk}}$. Thus, the normalized value $\tilde{y}_{nk}$ will also be increased when noise is injected. The variance value of noise is small when $g_{nh}$ is close to 0 or 1, therefore the normalized value $\tilde{y}_{nk}$ will be increased when noise is injected. Since the term $\nu_{nh}\sigma^2\|u_{h}\|^2_2 > 0$, we can conclude that the learning is accelerated when updating $\tilde{y}_{nk}$ by injecting slight noise.

### 3.4.1 Update Rule for $v_{kh}$

Now we proceed to find updating rules for $v_{kh}$. Its gradient expectation is given by Eq. (32):

$$E\left[\frac{\partial \ell_2}{\partial v_{kh}}\right] = \nu_{nh}E[\epsilon_n^2] = \nu_{nh}\sigma^2\|u_{h}\|^2_2$$

From above analysis, we can see that when the variance of noise is small, the normalized value $\tilde{y}_{nk}$ will be increased when noise is injected. Since the term $\nu_{nh}\sigma^2\|u_{h}\|^2_2 > 0$, we can conclude that the learning is accelerated when updating $\tilde{y}_{nk}$ by injecting slight noise.

### 4. Experiment

In our experiment we train and evaluate classification networks using MNIST dataset [12]. Initially we only use the MLP to conduct the basic classification experiment. Later in our experiment we consider 2 cases, stacked and non-stacked AutoEncoder, each is connected to the MLP. Two stages per case is considered: noiseless and noisy training. In all experiments, Images with 28 by 28 pixels per is used which defines the input size of the network. We compare the network accuracy and overall loss for every case.

#### 4.0.1 Basic Case: Classifying with an MLP

Logistic regression is a linear classifier. It is described in terms of its weight matrix and a bias vector. Having certain number of classes, the classification is performed by projecting an input vector onto a set of planes, each of which corresponds to a class. The Learning process aims to build an optimal model parameters. This process involves minimizing a loss function. In the case of multi-class logistic regression. A common loss function is the negative log-likelihood. The evaluation criteria of this experiment is the accuracy of classification and the overall loss evaluated by loss functions. We used 16,000 MNIST images, 6,000 for training and 10,000 for testing. The training took 1,000 Epochs with mini-batch size of 100. The loss function is the cross entropy, during this step we evaluated noiseless training and training with noise. The classification detection rate was 98.00% while the loss was 0.0023 without noise. Table 1 shows the basic case results in its last 2 rows.
4.0.2 Nested Case: Classifying with AutoEncoder and an MLP

Secondly an AutoEncoder is added to the MLP. The experiment used 16,000 images of MNIST, 6,000 of those images were used for training and 10,000 images for testing. The training took 1,000 Epochs with mini-batch size of 100. The loss function \((\lambda)CE + (1 - \lambda)MSR\) is balanced by \(\lambda = 0.6\) based on multiple experiments. ADAM optimizer is used for the training without weight decays. Adaptive Moment Estimation (Adam) \([24]\) is one of the techniques to find adaptive learning rates for each parameter within the network. Figure 2 shows a rough description for the network and how the AutoEncoder hidden units is directly connected to MLP at one case and stacked with another AutoEncoder in the second case. The number of units in a AutoEncoder hidden layer is varied to cover 10, 50, 100, 500, 700 and 1,000 hidden units. Increasing the hidden units increases the accuracy and lowers the loss. At this stage noise injection is not enabled.

On the other hand stacking the AutoEncoder has useful effects on the network performance. It tends to learn features like edges in an image. Such features are called first-order features. The second layer of a stacked AutoEncoder has the ability to understand more features detected from the first layer such as which edges tend to occur together to form contour or corner detectors. Considering the modules we discussed we designed a multi-layer AutoEncoder that is connected to MLP through its hidden units. The effect of increasing the hidden units is similar to the single layered AutoEncoder. Increasing the hidden units increases the accuracy, lowers the loss.

In both cases the number of hidden units per layer is varied, the accuracy of classification and the loss are evaluated. Less number of hidden units tend to give less accuracy and higher loss. This is natural while the ability to capture features becomes limited.

### Table 1

<table>
<thead>
<tr>
<th>Design</th>
<th>Noise Position</th>
<th>Accuracy (Train)</th>
<th>Accuracy (Test)</th>
<th>Loss (Train)</th>
<th>Loss (Test)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AE + MLP</td>
<td>None</td>
<td>84.79% (± 3.23)</td>
<td>85.62% (± 4.01)</td>
<td>0.0092</td>
<td>0.0079</td>
</tr>
<tr>
<td>AE + MLP</td>
<td>Input</td>
<td>86.96% (± 2.96)</td>
<td>88.20% (± 2.59)</td>
<td>0.0082</td>
<td>0.0067</td>
</tr>
<tr>
<td>AE + MLP</td>
<td>Hidden</td>
<td>97.09% (± 2.54)</td>
<td>98.51% (± 1.36)</td>
<td>0.00023</td>
<td>0.0001</td>
</tr>
<tr>
<td>AE + MLP + Hidden</td>
<td>Input + Hidden</td>
<td>90.91% (± 2.3)</td>
<td>91.39% (± 1.52)</td>
<td>0.0084</td>
<td>0.0069</td>
</tr>
<tr>
<td>MLP</td>
<td>None</td>
<td>96.25% (± 2.96)</td>
<td>97.60% (± 1.98)</td>
<td>0.0030</td>
<td>0.003</td>
</tr>
<tr>
<td>MLP</td>
<td>Hidden</td>
<td>97.08% (± 1.03)</td>
<td>98.00% (± 0.76)</td>
<td>0.0031</td>
<td>0.003</td>
</tr>
</tbody>
</table>

Fig. 3 Injecting noise into the hidden units of the single layered AutoEncoder improves the accuracy (top) and reduces the loss (bottom). The effect becomes clearer during the increase of hidden units number. Both noiseless and noisy MLP use 1000 hidden units for evaluation.

Fig. 4 Noise within the hidden units of and MLP with stack layered AutoEncoder improves the accuracy (top) and reduces the loss (bottom). The effect becomes clearer during the increase of hidden units number. Both noiseless and noisy MLP use 1000 hidden units for evaluation.

4.0.3 Noisy Case: Contaminating Nested Network

To confirm the effectiveness of noise contamination we inject the Gaussian noise which comes with a mean of 0 and the standard deviation \(\sigma = 0.1\) based on multiple experiments into hidden and input units of AutoEncoder (Single
Table 2 Networks with stacked AutoEncoders have better ability to extract important features. Injecting noise within their hidden units clearly improves the performance. A 1000 hidden unit stacked AutoEncoders is used and noise injection results are shown.

<table>
<thead>
<tr>
<th>Design</th>
<th>Noise Position</th>
<th>Accuracy (Train)</th>
<th>Accuracy (Test)</th>
<th>Loss (Train)</th>
<th>Loss (Test)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AE + MLP</td>
<td>None</td>
<td>85.22% (± 7.26)</td>
<td>86.14% (± 5.01)</td>
<td>0.0056</td>
<td>0.005</td>
</tr>
<tr>
<td>AE + MLP</td>
<td>Input</td>
<td>89.52% (± 3.56)</td>
<td>91.70% (± 2.55)</td>
<td>0.0042</td>
<td>0.0037</td>
</tr>
<tr>
<td>AE + MLP</td>
<td>Hidden</td>
<td>98.09% (± 0.29)</td>
<td>99.65% (± 0.06)</td>
<td>0.000039</td>
<td>0.000025</td>
</tr>
<tr>
<td>AE + MLP</td>
<td>Input and Hidden</td>
<td>93.52% (± 2.5)</td>
<td>94.00% (± 1.9)</td>
<td>0.0056</td>
<td>0.004</td>
</tr>
</tbody>
</table>

Fig. 5 AutoEncoder often captures a useful “hierarchical grouping” of the input. Visualizations for the hidden layer output shows stronger features of the digits after noise injection (left) compared to noiseless case (right).

Fig. 6 Upon giving certain input to the network the activation of hidden units has higher density around certain value that represents better sparsity.

Fig. 7 The t-SNE clearly shows better sparsity in case of noisy training (top) while inputs representations in hidden layer are clearly not overlapping. On the other hand the t-SNE chart shows more overlapping in the case of injecting noise at the input (middle). Finally overlapping among activation is the highest when noiseless training is performed for the same network (bottom).

To add experimental understanding to our work, AutoEncoder’s hidden layer output is visualized for noisy and noiseless schemes under hidden units number variation. We observe stronger feature representation when noise is injected into any network size and depth. Figure 5 shows how the internal representation of the features improves due to noise injection.
On the same hand the sparsity of hidden units activation increases as a measure of better feature representation as shown in the histogram chart in Fig. 6.

Figure 7 illustrates sparsity through the “t-SNE” [25] diagram that visualizes high-dimensional data by giving each hidden unit activation a location in a two dimensional map.

The t-SNE shows the hidden unit activation across all input cases. In the noiseless case there is considerable overlap while in noisy training case the overlap is almost zero which gives another measure of how noise injection improves the network performance.

5. Conclusion

In this paper, we investigated the effect of injecting noise on enhancing learning and improving generalization of neural networks made of AutoEncoders connected to MLP. We proposed mathematical clarification on why injecting noise at certain locations increases the network performance. Experimentally, injecting noise at those locations is able to obtain a better performance for both deep and shallow networks.

The proposed method was demonstrated through classification problem of digits under variety of handwriting. The experimental findings match our hypothesis which is supported by mathematical evidence.

Considering our findings, many other schemes of noise injection can be proposed. High research potential lies ahead in investing this design space to enhance existing networks performance and generalization ability.

Acknowledgments

This work was supported by JSPS KAKENHI Grant number 16K00239 and 16H01430.

References


Appendix:

This appendix contains derivations of equations that were
considered too lengthy or not crucial to the body of the paper.

A.1 Additive Noise Update rule for $u_{hi}$ - Eq. (18)

\[
\tilde{\ell}_1 = \frac{1}{2} \sum_{n=1}^{N} \sum_{j=1}^{I} (\tilde{z}_{nj} - z_{nj})^2 \\
= \ell_1 + \sum_{n=1}^{N} \sum_{j=1}^{I} (\Delta z_{nj} \delta_{nj} + \frac{\Delta z_{nj}^2}{2})
\]

(A-1)

where $\delta_{nj} := z_{nj} - \tilde{z}_{nj}$. Thus

\[
\frac{\partial \tilde{\ell}_1}{\partial u_{hi}} = \frac{\partial \ell_1}{\partial u_{hi}} + \frac{\partial}{\partial u_{hi}} \sum_{n=1}^{N} \sum_{j=1}^{I} (\delta_{nj} \Delta z_{nj} + \frac{\Delta z_{nj}^2}{2})
\]

(A-2)

For simplicity we approximate $\Delta z_{nj}^2$ to be $\Delta z_{nj}^2 \approx \sum_{h=1}^{H} (w_{jh} \omega_{nh} \epsilon_h)^2$ then we have

\[
\frac{\partial \Delta z_{nj}^2}{\partial u_{hi}} = 4w_{jh}^2 \sigma_h^2 \omega_{nh} \nu_{nh} x_{ni}
\]

(A-3)

\[
\frac{\partial \Delta z_{nj} \delta_{nj}}{\partial u_{hi}} = \\
(\nu_{jh} \Delta z_{nj} + ((1 - 2 \nu_{gh}) \epsilon_h + \frac{1}{2} \sigma_h \sum_{h=1}^{H} (w_{jh} \omega_{nh} \nu_{nh} x_{ni})
\]

(A-4)

is obtained through chain rule. The partial derivative

\[
\frac{\partial}{\partial u_{hi}} (\Delta z_{nj} \delta_{nj} + \frac{\Delta z_{nj}^2}{2}) = \\
w_{jh} \Delta z_{nj} \nu_{nh} x_{ni} + 2w_{jh}^2 \sigma_h^2 \omega_{nh} \nu_{nh} x_{ni}
\]

\[
+ ((1 - 2 \nu_{gh}) \epsilon_h + \frac{1}{2} \sigma_h \sum_{h=1}^{H} (w_{jh} \omega_{nh} \nu_{nh} x_{ni})
\]

(A-5)

\[
\frac{\partial \ell_2}{\partial u_{hi}} = \sum_{n=1}^{N} \sum_{k=1}^{K} (\bar{y}_{nk} - t_{nk}) \left( \frac{\partial \nu_{nk}}{\partial u_{hi}} + \frac{\partial \Delta \nu_{nk}}{\partial u_{hi}} \right)
\]

(A-6)

\[
\frac{\partial \Delta \nu_{nk}}{\partial u_{hi}} = 2 \nu_{nh} \nu_{nh} x_{ni} \epsilon_h + \frac{1}{2} \nu_{nh} \nu_{nh} \sigma_h^2
\]

(A-7)

\[
\frac{\partial \nu_{nk}}{\partial u_{hi}} = \nu_{nh} \nu_{nh} x_{ni}
\]

(A-8)

is found through chain rule, we have

\[
\frac{\partial \nu_{nk}}{\partial u_{hi}} + \frac{\partial \Delta \nu_{nk}}{\partial u_{hi}} = \\
\nu_{nh} \nu_{nh} x_{ni} \left( 1 + \sigma_h \epsilon_h^2 \right) + 2 \nu_{nh} \nu_{nh} x_{ni} \epsilon_h^2
\]

(A-9)

(A-5) and (A-9) together with the summations given in (A-1) and (A-6) form the Eq. (18). Equation (27) is derived similarly.

A.2 Expectation of Equation $\frac{\partial \tilde{\ell}_1}{\partial u_{hi}}$ - Eq. (20)

\[
\mathbb{E} \left[ \frac{\partial \tilde{\ell}_1}{\partial u_{hi}} \right] = \frac{\partial \ell_1}{\partial u_{hi}} + \sum_{n=1}^{N} \sum_{j=1}^{I} w_{jh} \omega_{nh} \nu_{nh} x_{ni} \sigma_h^2 \sum_{h=1}^{H} w_{jh} \nu_{nh} \nu_{nh} x_{ni} \sigma_h^2
\]

(A-10)

\[
+ \frac{\partial \ell_1}{\partial u_{hi}} \left( 1 + \sigma_h \epsilon_h^2 \right) + \sum_{n=1}^{N} \sum_{j=1}^{I} w_{jh} \omega_{nh} \nu_{nh} x_{ni} \sigma_h^2
\]

(A-11)

\[
\mathbb{E} \left[ \frac{\partial \ell_2}{\partial u_{hi}} \right] = \mathbb{E} \left[ \bar{y}_{nk} - t_{nk} \right] \frac{\partial \nu_{nk}}{\partial u_{hi}} \left( 1 + \sigma_h \epsilon_h^2 \right)
\]

(A-12)

In order to find $\mathbb{E} [\bar{y}_{nk} - t_{nk}]$, Jensen’s inequality is used:

\[
\mathbb{E} [\log \bar{y}_{nk}] \leq \log \mathbb{E} [\bar{y}_{nk}]
\]

(A-13)

\[
\mathbb{E} [\log \bar{y}_{nk}] = \mathbb{E} [\bar{y}_{nk}] - \mathbb{E} \left[ \log \sum_{n=1}^{N} \sum_{j=1}^{K} e_{nj} \right]
\]

(A-14)

By Jensen’s inequality again leads to Eqs. (21) and (22).

Motaz Sabri  he received a master degree in computer engineering from Hiroshima University. Presently he is a PhD. student at Hiroshima University. He is enrolled in big data laboratory. His main research interests are features classification, pattern recognition and its applications to image understanding via wavelets and neural networks.
Takio Kurita received the B.Eng. degree from Nagoya Institute of Technology and the Dr. Eng. degree from the University of Tsukuba, in 1981 and in 1993, respectively. He joined the Electrotechnical Laboratory, AIST, MITI in 1981. From 1990 to 1991 he was a visiting research scientist at Institute for Information Technology, National Research Council Canada. From 2001 to 2009, he was a deputy director of Neurosciences Research Institute, National Institute of Advanced Industrial Science and Technology (AIST). Also he was a Professor at Graduate School of Systems and Information Engineering, University of Tsukuba from 2002 to 2009. He is currently a Professor at Hiroshima University. His current research interests include statistical pattern recognition and its applications to image recognition. He is a member of the IEEE, the IPSJ, the IEICE of Japan, Japanese Neural Network Society, The Japanese Society of Artificial Intelligence.