Autoencoder with New Loss Functions

Takuma Haruguchi s1230038 Supervised by Prof. Shuxue Ding

Abstract

In general, the symmetrical functions such as squared error are used as loss function for autoencoder. This study employed I-divergence (Information-divergence, or generalized Kullback-Leibler Divergence) as loss function, which is asymmetrical loss function. Using the I-divergence can be expected to work as well as the squared error does because both are siblings of β-divergence. This research demonstrated that I-divergence loss function can be calculated mathematically with backpropagation. In addition, it implemented the autoencoder that used the I-divergence. As a result, evaluating with SSIM (Structural Similarity) of output images, it was found that the autoencoder of using the I-divergence worked as well as using the squared error did.

1 Introduction

1.1 Overview of Autoencoder

An autoencoder is a neural network that generates output data to reproduce input data as closely as possible. The aim of the autoencoder is to learn a compressed representation of the input, which is called "code". To achieve it, the number of the code units is less than the input and output ones while the input units are same as the output units. These days, practical examples of using autoencoder are data denoising and dimensionality reduction for data visualization [1].

When learning to get the code, the autoencoder calculates and minimizes a value of a function that indicates distance between the input and the output, which is called "loss function".

1.2 Asymmetrical Loss Function based on I-divergence

Generally speaking, conventional loss functions are symmetrical such as the squared error. In fact, there is no mathematical proof that symmetrical functions are the best in autoencoder. Therefore, the aim of this study is to propose alternative loss functions based on I-divergence which is asymmetrical.

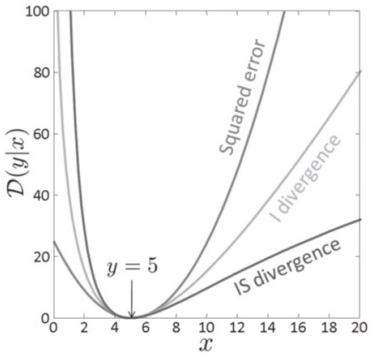

Figure 1: β-divergence Siblings [2]

It can be expected that these alternative functions can work as well as the squared error does because I-divergence is a sibling of β-divergence as in Fig. 1, which includes the squared error. As mentioned above, the squared error is one type of conventional loss function, and the graphs of I-divergence are similar to the squared error.

1.3 Definitions of Autoencoder

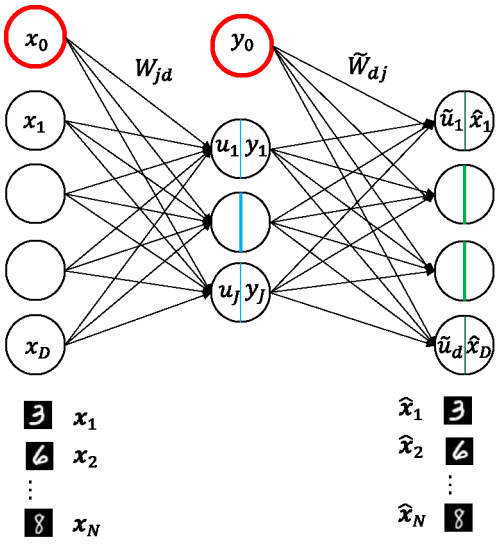

Figure 2: Network of Autoencoder

In this study, autoencoder loads a numerical image into the input units, transmits them to the code units in order to compress the image, and lastly generates the output units from the code units to reconstruct the image. Looking at Fig. 2, let us introduce the definitions of the autoencoder.

Input units can be defined as below.

$$\mathbf{x} = \begin{bmatrix} x_0 & x_1 & \dots & x_D \end{bmatrix}^{\mathrm{T}} , \; x_0 \equiv 1 , \; \mathbf{x} \in TR \tag{1}$$

x_0 is for the bias, the training data set TR can be defined as below.

$$TR = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\} \tag{2}$$

Code units can be defined as below.

$$\mathbf{y} = \begin{bmatrix} y_0 & y_1 & \dots & y_J \end{bmatrix}^{\mathrm{T}} , \; y_0 \equiv 1 \\ \mathbf{y} \in CD , \; D > J \tag{3}$$

y_0 is for the bias.

The set of code data can be defined as below.

$$CD = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N\} \tag{4}$$

Output units can be defined as below.

$$\hat{\mathbf{x}} = \begin{bmatrix} \hat{x}_1 & \hat{x}_2 & \dots & \hat{x}_D \end{bmatrix}^{\mathrm{T}} , \; \hat{\mathbf{x}} \in OP \tag{5}$$

Output data set can be defined as below.

$$OP = \{\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \dots, \hat{\mathbf{x}}_N\} \tag{6}$$

Weight and bias of the encoding can be defined as below.

$$W = \begin{bmatrix} W_{10} & \dots & W_{1D} \\ \vdots & \ddots & \vdots \\ W_{J0} & \dots & W_{JD} \end{bmatrix} \tag{7}$$

$$\mathbf{b} = \begin{bmatrix} b_1 & b_2 & \dots & b_J \end{bmatrix}^{\mathrm{T}} , \; W_{j0} = b_j \tag{8}$$

Weight and bias of the decoding can be defined as below.

$$\tilde{W} = \begin{bmatrix} \tilde{W}_{10} & \dots & \tilde{W}_{1J} \\ \vdots & \ddots & \vdots \\ \tilde{W}_{D0} & \dots & \tilde{W}_{DJ} \end{bmatrix} \tag{9}$$

$$\tilde{\mathbf{b}} = \begin{bmatrix} \tilde{b}_1 & \tilde{b}_2 & \dots & \tilde{b}_D \end{bmatrix}^{\mathrm{T}} , \; \tilde{W}_{d0} = \tilde{b}_d \tag{10}$$

Total input $\mathbf{u}$ for the encoder can be defined as below.

$$u_j = W_{j0}x_0 + W_{j1}x_1 + \dots + W_{jD}x_D \Leftrightarrow \mathbf{u} = \mathbf{W}\mathbf{x} \\ (j = 1, 2, \dots, J) \tag{11}$$

Encoder activation f can be defined as below.

$$y_j = f(u_j) \Leftrightarrow \mathbf{y} = \mathbf{f}(\mathbf{u}) = \begin{bmatrix} f(u_1) & f(u_2) & \dots & f(u_J) \end{bmatrix}^{\mathrm{T}} \tag{12}$$

Total input $\tilde{\mathbf{u}}$ for the decoder can be defined as below.

$$\tilde{u}_d = \tilde{W}_{d0}y_0 + \tilde{W}_{d1}y_1 + \dots + \tilde{W}_{dJ}y_J \Leftrightarrow \tilde{\mathbf{u}} = \tilde{\mathbf{W}}\mathbf{y} \\ (d = 1, 2, \dots, D) \tag{13}$$

Decoder activation $\tilde{f}$ can be defined as below.

$$\hat{x}_d = \tilde{f}(\tilde{u}_d) \Leftrightarrow \hat{\mathbf{x}} = \tilde{\mathbf{f}}(\tilde{\mathbf{u}}) = \begin{bmatrix} \tilde{f}(\tilde{u}_1) & \tilde{f}(\tilde{u}_2) & \dots & \tilde{f}(\tilde{u}_D) \end{bmatrix}^{\mathrm{T}} \tag{14}$$

Let us call both f and $\tilde{f}$ activations.

All weights in the network can be defined as below.

$$\mathbf{w} = \mathbf{vec}([\mathbf{W}, \tilde{\mathbf{W}}]) = \begin{bmatrix} w_1 & w_2 & \dots & w_M \end{bmatrix}^{\mathrm{T}} \tag{15}$$

The $\mathbf{w}$ is a vector which has all matrix elements of both $\mathbf{W}$ and $\tilde{\mathbf{W}}$ as the element of $\mathbf{w}$ through the vectorization.

Loss function $E(\mathbf{w})$ is scalar. It will be described in detail in Sec. 2.3.

Shorthand notation can be defined as below.

$$\hat{\mathbf{x}} \equiv \hat{\mathbf{x}}(\mathbf{x}) \equiv \hat{\mathbf{x}}(\mathbf{x}, \mathbf{w}) \equiv \tilde{\mathbf{f}}(\tilde{\mathbf{W}}\mathbf{f}(\mathbf{W}\mathbf{x})) \tag{16}$$

$$x_d(\mathbf{x_n}) \equiv x_d \tag{17}$$

$$\hat{x}_d(\mathbf{x_n}) \equiv \hat{x}_d(\mathbf{x_n}, \mathbf{w}) \equiv \hat{x}_d \tag{18}$$

$$\mathbf{y}(\mathbf{x_n}) \equiv \mathbf{y} \equiv \mathbf{f}(\mathbf{u}) \tag{19}$$

$$y_j(\mathbf{x_n}) \equiv y_j \tag{20}$$

$$\tilde{u}_d(\mathbf{x_n}) \equiv \tilde{u}_d(\tilde{\mathbf{W}}, \mathbf{y}(\mathbf{x_n})) \tag{21}$$

1.4 Learning: SGD

Theoretically, the final goal of the learning in a neural network is to get $\mathbf{w} = argmin_{\mathbf{w}} E(\mathbf{w})$, which is called global minimum $\mathbf{w}$. However, it is almost always impossible to get it because there are numerous local minimums $\mathbf{w}$ in $E(\mathbf{w})$ in many cases. Instead, let us introduce Stochastic Gradient Descent(SGD) with minibatch to get the local minimum $\mathbf{w}$ which is sufficiently small [3].

The steps of SGD with minibatch:

Step 1 Calculating loss function

$$E_t(\mathbf{w}) = \frac{1}{N_t} \sum_{n \in D_t} E_n(\mathbf{w}) \tag{22}$$

- $E_n(\mathbf{w})$ is the result of calculating for one training data $\mathbf{x_n}$ in (2).
- D_t is a minibatch, which is a small set of the training data set as (2). $N_t = |D_t|$
- The t is updating number, which is called epoch. In the t^{th} updating, the elements of D_t will be replaced with other elements in the training data set in (2).

Step 2 Calculating gradient

$$\nabla \mathbf{E}_t \equiv \frac{\partial E_t}{\partial \mathbf{w}} = [\frac{\partial E_t}{\partial w_1} \frac{\partial E_t}{\partial w_2} \cdots \frac{\partial E_t}{\partial w_M}]^{\mathrm{T}} \tag{23}$$

- The $\mathbf{w}$ is all weights as (15).

Step 3 Updating weight

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \epsilon \nabla \mathbf{E}_t \tag{24}$$

- The ϵ is the learning rate, which is a hyper parameter.

Step 4 Incrementing epoch

- $t \leftarrow t + 1$, and go to Step 1 until the t equals to max_epoch which is maximum of this repetition.

1.5 Optimizer

The explanations of Sec. 1.4 are conceptual. For example, that section doesn't consider deciding the values of hyper parameters a researcher needs to set up.

When implementing the autoencoder, we must specify an optimizer to decide them automatically. Comparing the results of autoencoders that uses each optimizer, this study employed Adagrad, Adam, AdaDelta, Nadam, and Adamax as optimizer [4].

1.6 Calculating the Gradient: Backpropagation

In terms of implementation, the more layers a neural network has, the more explosive increase the computational complexity of the calculating gradient in SGD becomes. Practically, we must calculate it with the backpropagation. In Sec. 2.4, the backpropagation will be described.

1.7 The Goal of This Study

The goal of this study contains two points. The first one is to verify whether the alternative loss functions can be mathematically calculated with backpropagation. It will be discussed in Sec. 2.4. The second is to implement the autoencoder with the alternative functions to examine whether they can work as well as using the squared error does. Sec. 2.5 and 3 will describe this point.

2 Method

2.1 Concrete Values and Input Restriction

In this study, the concrete values for the variables about (1), (2), (3), (22) and Step 4 in SGD are as below.

$$\begin{aligned} D = 784, N = 60000, J = 32, \\ N_t = 256, max\_epoch = 50 \end{aligned} \tag{25}$$

The 784 is based on the pixel size of a image: 28×28

In addition, we will restrict domain of definition of the input units to the below expression with normalization such as $x_d/255$. Note that each pixel of a original image ranges from 0 to 255.

$$\{x_d \in \mathbb{R} \mid 0 \leq x_d \leq 1\} \tag{26}$$

2.2 Activation

The combinations of activation in this study are as below.

$$f(u_j) = max(u_j, \varepsilon)\ ,\ \tilde{f}(\tilde{u}_d) = max(\tilde{u}_d, \varepsilon) \tag{27}$$

$$f(u_j) = max(u_j, \varepsilon)\ ,\ \tilde{f}(\tilde{u}_d) = \frac{1}{1 + \exp(\tilde{u}_d)} \tag{28}$$

$$\varepsilon \to 0$$

(27) and (28) will be calculated with backpropagation and implemented individually. In general $max(x, 0)$ is called ReLU (Rectified Linear Units). However, let us employ $max(x, \varepsilon)$ as ReLU in this study to prevent the denominator in (58) from becoming zero. The $\frac{1}{1+\exp(\tilde{u}_d)}$ is called logistic sigmoid function. Let us call (27) ReLU-ReLU and (28) ReLU-Sigmoid in this study.

2.3 Loss Function from β-divergence

Let us define the alternative loss functions. Firstly, I will derive the squared error and I-divergence from

β-divergence to show that both are siblings. The definition of β-divergence is as below.

$$D_\beta(y,x) := \frac{y^\beta}{\beta(\beta-1)} + \frac{x^\beta}{\beta} + \frac{yx^{\beta-1}}{\beta-1} \tag{29}$$

plugging 2 or 1 for the variable β into (29), we will obtain the below [5].

Squared Error:

$$\beta = 2: \quad D_\beta(y,x) := \frac{1}{2}(y-x)^2 \tag{30}$$

I-divergence:

$$\beta \to 1 : D_\beta(y,x) = \lim_{\beta\to 1} \frac{y^\beta}{\beta(\beta-1)} + x - \frac{yx^{\beta-1}}{\beta-1}$$

Using L'Hôpital's Rule,

$$\lim_{\beta\to 1} \frac{\frac{d}{d\beta}y^\beta}{\frac{d}{d\beta}\beta(\beta-1)} = \lim_{\beta\to 1} \frac{y^\beta \log y}{2\beta-1} = y\log y$$

$$\lim_{\beta\to 1} \frac{\frac{d}{d\beta}yx^{\beta-1}}{\frac{d}{d\beta}(\beta-1)} = \lim_{\beta\to 1} \frac{yx^\beta \log x \times 1}{1} = \lim_{\beta\to 0} yx^\beta \log x = y\log x$$

$$\therefore \beta \to 1: \; D_\beta(y,x) \to \; ylog\frac{y}{x} + x \tag{31}$$

Next, in order to fit the I-divergence vertex into the squared error one, change (31) into the below.

$$D_I(y|\,x) = ylog\frac{y}{x} - y + x \tag{32}$$

Lastly, from (32) we have obtained the two loss functions for one pair ($\hat{x}_d,\ x_d$) as below: I-divergence 1 and I-divergence 2.

$$E_{I1}(\mathbf{w}) = \sum_{d=1}^{D} \hat{x}_d log\frac{\hat{x}_d}{x_d} - \hat{x}_d + x_d \tag{33}$$

$$E_{I2}(\mathbf{w}) = \sum_{d=1}^{D} x_d log\frac{x_d}{\hat{x}_d} - x_d + \hat{x}_d \tag{34}$$

See (18) about the relationship between $\hat{x}_d$ and $\mathbf{w}$.

2.4 Backpropagation by Hand [6]

This section will demonstrate that the alternative loss functions can be calculated mathematically with backpropagation. It means that the implementation of using the functions can be expected to perform as well as using the squared error functions does.

Calculating the backpropagation consists of five steps. The input of it is one pair ($\hat{\mathbf{x}}_n,\ \mathbf{x}_n$) and the final outputs are $\frac{\partial E_n}{\partial W_{jd}}$ and $\frac{\partial E_n}{\partial \tilde{W}_{dj}}$ which mean the gradient of each layer. Step 2 through 4 are different for each loss function, ReLU-ReLU, and ReLU-Sigmoid.

2.4.1 Preparation for Differentiation of ReLU

When $u = 0$, ReLU $f(u)$ is indifferentiable because left derivative and right derivative are not equal. Introducing subderivative, let us define the differentiation of ReLU which includes $u = 0$.

$$\frac{d}{du}f(u) = \begin{cases} 1, & if\ u > 0 \\ 0, & otherwise \end{cases} \tag{35}$$

In addition, we must define the below sets to deal with the differentiation.

$$I = \{i \mid \tilde{u}_i > 0\} \tag{36}$$

$$H = \{h \mid \tilde{u}_h \le 0\} \tag{37}$$

2.4.2 Step 1 Forward Propagation

In this step, each unit in each layer is calculated in the forward direction. The number in a parenthesis of a superscript represents the order of a layer.

$$\mathbf{z}^{(1)} = \mathbf{x}_n \tag{38}$$

$$\mathbf{z}^{(2)} = \mathbf{y}_n = \mathbf{f}(\mathbf{W}\mathbf{x}_n) \tag{39}$$

$$\mathbf{z}^{(3)} = \hat{\mathbf{x}}_n = \tilde{\mathbf{f}}\left(\tilde{\mathbf{W}}\mathbf{y}_n\right) \tag{40}$$

Shorthand notations are as below.

$$\begin{gathered} z_d^{(1)} = x_d \quad z_j^{(2)} = y_j \quad z_d^{(3)} = \hat{x}_d \\ d = 1,2,\ldots,D, \quad j = 1,2,\ldots,J \end{gathered} \tag{41}$$

2.4.3 Step 2 Calculating Delta of Output ($E_{I1}(\mathbf{w})$ in ReLU-ReLU)

The delta of the output units is as below.

$$\delta_d^{(3)} \equiv \frac{\partial E_n}{\partial \tilde{u}_d} \tag{42}$$

By applying (33) and (14) to (42), we obtain the following expression.

$$\begin{aligned} \delta_d^{(3)} &= \frac{\partial}{\partial \tilde{u}_d} E_{I1}(\mathbf{w}) = \frac{\partial}{\partial \tilde{u}_d}\left\{\sum_{i=1}^{D} \hat{x}_i log\frac{\hat{x}_i}{x_i} - \hat{x}_i + x_i\right\} \\ &= \frac{\partial}{\partial \tilde{u}_d}\left\{\sum_{i=1}^{D} \tilde{f}(\tilde{u}_i)\, log\frac{\tilde{f}(\tilde{u}_i)}{x_i} - \tilde{f}(\tilde{u}_i) + x_i\right\} \end{aligned} \tag{43}$$

Applying (27) to (43), when $\tilde{u}_d > 0$,

$$\begin{aligned}\delta_d^{(3)} &= \frac{\partial}{\partial \tilde{u}_d}\left\{\sum_{i=1}^{D}\tilde{u}_i log\frac{\tilde{u}_i}{x_i} - \tilde{u}_i + x_i\right\} \\ &= log\frac{\tilde{u}_d}{x_d} + \tilde{u}_d\frac{1}{\frac{\tilde{u}_d}{x_d}}\frac{1}{x_d} - 1 = log\frac{\tilde{u}_d}{x_d}\end{aligned} \tag{44}$$

When $\tilde{u}_d \leq 0$,

$$\begin{aligned}\delta_d^{(3)} = \{\frac{\partial}{\partial \tilde{u}_d}\tilde{f}(\tilde{u}_i)\}log\frac{\tilde{f}(\tilde{u}_i)}{x_d} + \tilde{f}(\tilde{u}_i)\frac{1}{\frac{\tilde{f}(\tilde{u}_i)}{x_d}}\frac{\frac{\partial}{\partial \tilde{u}_d}\tilde{f}(\tilde{u}_i)}{x_d} \\ -\frac{\partial}{\partial \tilde{u}_d}\tilde{f}(\tilde{u}_i) = 0 \\ \because \frac{\partial}{\partial \tilde{u}_d}\tilde{f}(\tilde{u}_i) = 0\end{aligned} \tag{45}$$

2.4.4 Step 3 Backpropagation (E_{I1}(**w**) in ReLU-ReLU)

The delta of the code units is as below.

$$\delta_j^{(2)} = \sum_{k=1}^{D}\delta_k^{(3)}\left\{\tilde{W}_{kj}\frac{\partial}{\partial u_j}f\left(u_j\right)\right\} \tag{46}$$

By applying (36) and (37) to (46), we obtain the following expression.

$$\begin{aligned}\delta_j^{(2)} = \sum_{k\in I}\delta_k^{(3)}\left\{\tilde{W}_{kj}\frac{\partial}{\partial u_j}f\left(u_j\right)\right\} + \\ \sum_{k\in H}\delta_k^{(3)}\left\{\tilde{W}_{kj}\frac{\partial}{\partial u_j}f\left(u_j\right)\right\}\end{aligned} \tag{47}$$

Applying (27) to (47), when $u_j > 0$,

$$\begin{aligned}(47) \Leftrightarrow \delta_j^{(2)} &= \sum_{k\in I}\tilde{W}_{kj}log\frac{\tilde{u}_k}{x_k} \\ &\because \delta_k^{(3)} = 0\ (k \in H)\end{aligned} \tag{48}$$

When $u_j \leq 0$,

$$\begin{aligned}(47) \Leftrightarrow \delta_j^{(2)} &= 0 \\ \because \frac{\partial}{\partial u_j}f\left(u_j\right) &= 0\end{aligned} \tag{49}$$

2.4.5 Step 4 Calculating Gradients of Each Layer (E_{I1}(**w**) in ReLU-ReLU)

Gradients of each layer are as below.

$$\frac{\partial E_n}{\partial W_{jd}} = \delta_j^{(2)}z_d^{(1)} \tag{50}$$

$$\frac{\partial E_n}{\partial \tilde{W}_{dj}} = \delta_d^{(3)}z_j^{(2)} \tag{51}$$

From (50) and (51), we can obtain the following expressions.

$$\frac{\partial E_n}{\partial W_{jd}} = \delta_j^{(2)}z_d^{(1)} = \left\{\sum_{k\in I}\tilde{W}_{kj}log\frac{\tilde{u}_k}{x_k}\right\}x_d \quad (u_j > 0) \tag{52}$$

$$\frac{\partial E_n}{\partial W_{jd}} = \delta_j^{(2)}z_d^{(1)} = 0 \quad (u_j \leq 0) \tag{53}$$

$$\frac{\partial E_n}{\partial \tilde{W}_{dj}} = \delta_d^{(3)}z_j^{(2)} = \{log\frac{\tilde{u}_d}{x_d}\}y_j \quad (\tilde{u}_d > 0) \tag{54}$$

$$\frac{\partial E_n}{\partial \tilde{W}_{dj}} = \delta_d^{(3)}z_j^{(2)} = 0 \quad (\tilde{u}_d \leq 0) \tag{55}$$

2.4.6 Step 2 Calculating Delta of Output (E_{I2}(**w**) in ReLU-ReLU)

By applying (34) and (14) to (42), we obtain the following expression.

$$\begin{aligned}\delta_d^{(3)} = \frac{\partial}{\partial \tilde{u}_d}E_{I2}(\mathbf{w}) &= \frac{\partial}{\partial \tilde{u}_d}\left\{\sum_{i=1}^{D}x_i log\frac{x_i}{\hat{x}_i} - x_i + \hat{x}_i\right\} \\ &= \frac{\partial}{\partial \tilde{u}_d}\left\{\sum_{i=1}^{D}x_i log\frac{x_i}{\tilde{f}(\tilde{u}_i)} - x_i + \tilde{f}(\tilde{u}_i)\right\}\end{aligned} \tag{56}$$

Applying (27) to (56), when $\tilde{u}_d > 0$,

$$\begin{aligned}\delta_d^{(3)} &= \frac{\partial}{\partial \tilde{u}_d}\left\{\sum_{d=1}^{D}x_i log\frac{x_i}{\tilde{u}_i} - x_i + \tilde{u}_i\right\} \\ &= x_d\frac{1}{\frac{x_d}{\tilde{u}_d}}\frac{-x_d}{\tilde{u}_d^2} + 1 = -\frac{x_d}{\tilde{u}_d} + 1\end{aligned} \tag{57}$$

When $\tilde{u}_d \leq 0$,

$$\begin{aligned}\delta_d^{(3)} &= x_d\frac{1}{\frac{x_d}{\tilde{f}(\tilde{u}_d)}}\left\{\frac{-x_d\frac{\partial}{\partial \tilde{u}_d}\tilde{f}(\tilde{u}_d)}{\tilde{f}(\tilde{u}_d)^2}\right\} \\ &= \frac{-x_d\frac{\partial}{\partial \tilde{u}_d}\tilde{f}(\tilde{u}_d)}{\tilde{f}(\tilde{u}_d)} = 0\end{aligned} \tag{58}$$

2.4.7 Step 3 Backpropagation (E_{I2}(**w**) in ReLU-ReLU)

Applying (27) to (47), we obtain the following expressions.

When $u_j > 0$,

$$\delta_j^{(2)} = -\sum_{k\epsilon I}\tilde{W}_{kj}\left\{\frac{x_k}{\tilde{u}_k} - 1\right\} \tag{59}$$

When $u_j \leq 0$,

$$\delta_j^{(2)} = 0 \tag{60}$$

2.4.8 Step 4 Calculating Gradients of Each Layer ($E_{I2}(\mathbf{w})$ in ReLU-ReLU)

From (50) and (51), we can obtain the following expressions.

$$\frac{\partial E_n}{\partial W_{jd}} = \delta_j^{(2)} z_d^{(1)} = \{-\sum_{k\epsilon I} \tilde{W}_{kj} \left\{ \frac{x_k}{\tilde{u}_k} - 1 \right\} \} x_d \quad (u_j > 0) \tag{61}$$

$$\frac{\partial E_n}{\partial W_{jd}} = \delta_j^{(2)} z_d^{(1)} = 0 \quad (u_j \leq 0) \tag{62}$$

$$\frac{\partial E_n}{\partial \tilde{W}_{dj}} = \delta_d^{(3)} z_j^{(2)} = \{-\frac{x_d}{\tilde{u}_d} + 1\} y_j \quad (\tilde{u}_d > 0) \tag{63}$$

$$\frac{\partial E_n}{\partial \tilde{W}_{dj}} = \delta_d^{(3)} z_j^{(2)} = 0 \quad (\tilde{u}_d \leq 0) \tag{64}$$

2.4.9 Step 2 Calculating Delta of Output ($E_{I1}(\mathbf{w})$ in ReLU-Sigmoid)

By applying (28) to (43), we obtain the following expression.

$$\begin{aligned} \delta_d^{(3)} &= \tilde{f}(\tilde{u}_d)\{1 - \tilde{f}(\tilde{u}_d)\} log \frac{\tilde{f}(\tilde{u}_d)}{x_d} \\ &+ \tilde{f}(\tilde{u}_d) \frac{1}{\frac{\tilde{f}(\tilde{u}_d)}{x_d}} \frac{\tilde{f}(\tilde{u}_d)\{1 - \tilde{f}(\tilde{u}_d)\}}{x_d} - \tilde{f}(\tilde{u}_d)\{1 - \tilde{f}(\tilde{u}_d)\} \\ &= \tilde{f}(\tilde{u}_d)\left\{1 - \tilde{f}(\tilde{u}_d)\right\} log \frac{\tilde{f}(\tilde{u}_d)}{x_d} \\ &\because \frac{d}{d\tilde{u}_d} \tilde{f}(\tilde{u}_d) = \tilde{f}(\tilde{u}_d)\left\{1 - \tilde{f}(\tilde{u}_d)\right\} \end{aligned} \tag{65}$$

2.4.10 Step 3 Backpropagation ($E_{I1}(\mathbf{w})$ in ReLU-Sigmoid)

Applying (28) to (46), we obtain the following expressions.

when $u_j > 0$,

$$\delta_j^{(2)} = \sum_k \tilde{W}_{kj} \tilde{f}(\tilde{u}_k)\left\{1 - \tilde{f}(\tilde{u}_k)\right\} log \frac{\tilde{f}(\tilde{u}_k)}{x_k} \tag{66}$$

when $u_j \leq 0$,

$$\delta_j^{(2)} = 0 \tag{67}$$

2.4.11 Step 4 Calculating Gradients of Each Layer ($E_{I1}(\mathbf{w})$ in ReLU-Sigmoid)

From (50) and (51), we can obtain the following expressions.

$$\frac{\partial E_n}{\partial W_{jd}} = \{\sum_k \tilde{W}_{kj} \tilde{f}(\tilde{u}_k)\left\{1 - \tilde{f}(\tilde{u}_k)\right\} log \frac{\tilde{f}(\tilde{u}_k)}{x_k}\} x_d \quad (u_j > 0) \tag{68}$$

$$\frac{\partial E_n}{\partial W_{jd}} = 0 \quad (u_j \leq 0) \tag{69}$$

$$\frac{\partial E_n}{\partial \tilde{W}_{dj}} = \{\tilde{f}(\tilde{u}_d)\left\{1 - \tilde{f}(\tilde{u}_d)\right\} log \frac{\tilde{f}(\tilde{u}_d)}{x_d}\} y_j \tag{70}$$

2.4.12 Step 2 Calculating Delta of Output ($E_{I2}(\mathbf{w})$ in ReLU-Sigmoid)

By applying (28) to (56), we obtain the following expression.

$$\begin{aligned} \delta_d^{(3)} &= x_d \frac{1}{\frac{x_d}{\tilde{f}(\tilde{u}_d)}} \frac{-x_d \tilde{f}(\tilde{u}_d)\{1 - \tilde{f}(\tilde{u}_d)\}}{\tilde{f}(\tilde{u}_d)^2} \\ &+ \tilde{f}(\tilde{u}_d)\{1 - \tilde{f}(\tilde{u}_d)\} \\ &= \left\{1 - \tilde{f}(\tilde{u}_d)\right\}\{\tilde{f}(\tilde{u}_d) - x_d\} \end{aligned} \tag{71}$$

2.4.13 Step 3 Backpropagation ($E_{I2}(\mathbf{w})$ in ReLU-Sigmoid)

Applying (28) to (46), we obtain the following expressions.

when $u_j > 0$,

$$\delta_j^{(2)} = \sum_k \tilde{W}_{kj}\left\{1 - \tilde{f}(\tilde{u}_k)\right\}\{\tilde{f}(\tilde{u}_k) - x_k\} \tag{72}$$

when $u_j \leq 0$,

$$\delta_j^{(2)} = 0 \tag{73}$$

2.4.14 Step 4 Calculating Gradients of Each Layer ($E_{I2}(\mathbf{w})$ in ReLU-Sigmoid)

From (50) and (51), we can obtain the following expressions.

$$\frac{\partial E_n}{\partial W_{jd}} = \{\sum_k \tilde{W}_{kj}\left\{1 - \tilde{f}(\tilde{u}_k)\right\}\{\tilde{f}(\tilde{u}_k) - x_k\}\} x_d \quad (u_j > 0) \tag{74}$$

$$\frac{\partial E_n}{\partial W_{jd}} = 0 \quad (u_j \leq 0) \tag{75}$$

$$\frac{\partial E_n}{\partial \tilde{W}_{dj}} = \{\left\{1 - \tilde{f}(\tilde{u}_d)\right\}\{\tilde{f}(\tilde{u}_d) - x_d\}\} y_j \tag{76}$$

2.4.15 Step 5 Applying Minibatch

When using minibatch, the gradient E is derived from $E = \sum_{n \in D_t} E_n$ for each layer.
The final output is as below.

$$\frac{\partial E}{\partial W_{jd}} = \sum_{n \in D_t} \frac{\partial E_n}{\partial W_{jd}} = \sum_{\mathbf{x}_n \in D_t} \frac{\partial E(\mathbf{x}_n)}{\partial W_{jd}} \tag{77}$$

$$\frac{\partial E}{\partial \tilde{W}_{dj}} = \sum_{n \in D_t} \frac{\partial E_n}{\partial \tilde{W}_{dj}} = \sum_{\mathbf{x}_n \in D_t} \frac{\partial E(\mathbf{x}_n)}{\partial \tilde{W}_{dj}} \tag{78}$$

See (22) about D_t and $\mathbf{x}_n$.

2.5 Overview of Experiment

In this study, the autoencoder is implemented as below. See [7] about source code.

Dataset	MNIST
Language	Python
Framework	Keras (backend: TensorFlow)
Loss Function	MSE, $E_{I1}(\mathbf{w})$, $E_{I2}(\mathbf{w})$
Activations	ReLU-ReLU, ReLU-Sigmoid

Note that Squared Error was implemented as the Mean Squared Error (MSE) because of considering minibatch for Keras. The same configuration also applied to other loss functions.

This study employed SSIM (Structural Similarity) as evaluation index of the output image that the autoencoder generated. The definition of SSIM is as below [8].

$$\mathrm{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

The x and y are input image and output image respectively. The μ_x is the average of x and σ_x^2 is the variance of x. The μ_y and the σ_y^2 are similar to them. The σ_{xy} is the covariance of x and y. The C_1 is $(0.01 \times 255)^2$ and The C_2 is $(0.03 \times 255)^2$ when evaluating the images of 8 bit grayscale.

The higher SSIM value is, the more similar to the input image the output image is. The range of it is $[0, 1]$.

3 Results

3.1 SSIM

The result of the experiment about SSIM and optimizer was as below. Each optimizer in the tables meant showing the best SSIM value for each loss function among all optimizers.
ReLU-ReLU:

Loss Function	MSE	$E_{I1}(\mathbf{w})$	$E_{I2}(\mathbf{w})$
SSIM	0.795	0.526	0.751
Optimizer	Adagrad	Adamax	AdaDelta

ReLU-Sigmoid:

Loss Function	MSE	$E_{I1}(\mathbf{w})$	$E_{I2}(\mathbf{w})$
SSIM	0.855	0.632	0.880
Optimizer	Adam	Adamax	Nadam

For details about arguments of each optimizer, see the source code [7].

3.2 Output Images

Input image was as below.

Figure 3: Input Image

The results as output image were as below.

Figure 4: The output image of MSE in ReLU-ReLU

Figure 5: The output image of I-divergence 1 in ReLU-ReLU

Figure 6: The output image of I-divergence 2 in ReLU-ReLU

Figure 7: The output image of MSE in ReLU-Sigmoid

Figure 8: The output image of I-divergence 1 in ReLU-Sigmoid

Figure 9: The output image of I-divergence 2 in ReLU-Sigmoid

3.3 The Loss History

The histories of the value which the loss function returned were as below.

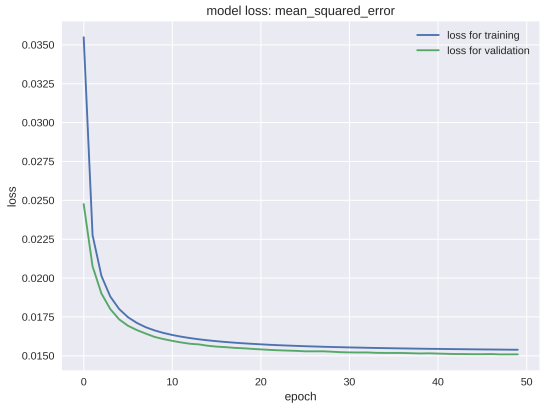

Figure 10: Loss History of MSE in ReLU-ReLU

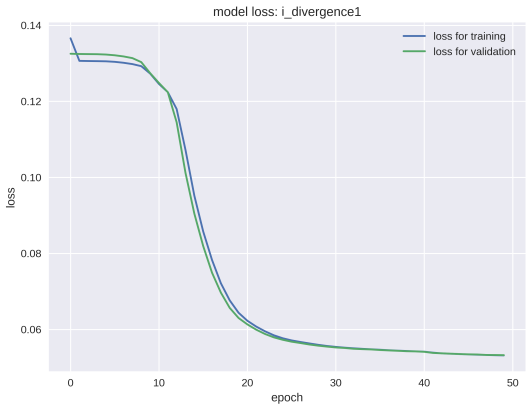

Figure 11: Loss History of I-divergence 1 in ReLU-ReLU

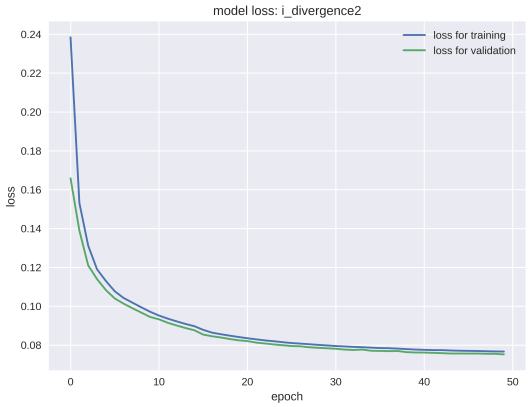

Figure 12: Loss History of I-divergence 2 in ReLU-ReLU

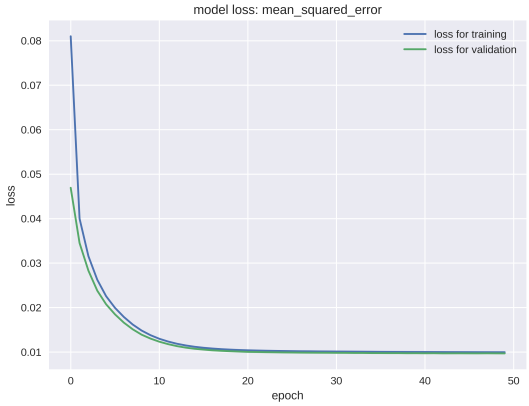

Figure 13: Loss History of MSE in ReLU-Sigmoid

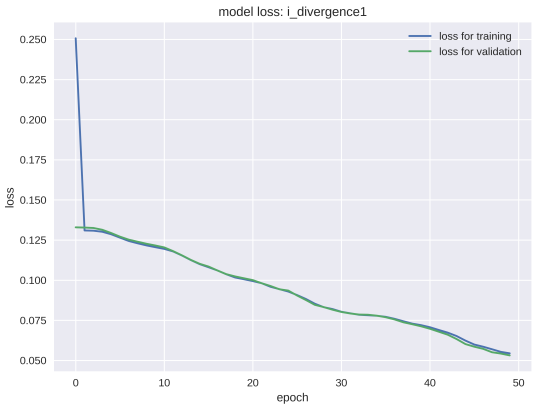

Figure 14: Loss History of I-divergence 1 in ReLU-Sigmoid

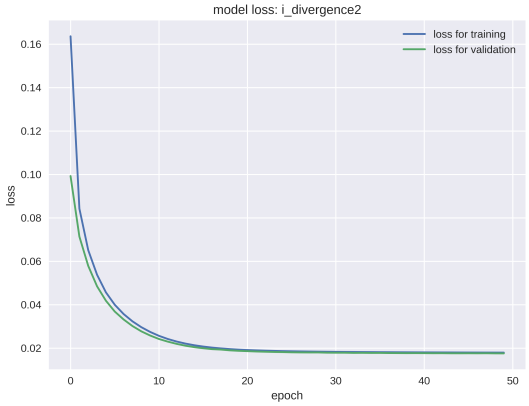

Figure 15: Loss History of I-divergence 2 in ReLU-Sigmoid

4 Discussion

About accuracy, the results of SSIM value and output images suggested that I-divergence loss functions can work as well as the squared error functions did. It indicated that the SSIM of the ReLU-Sigmoid combination was better than the ReLU-ReLU combination. In particular, the SSIM of I-divergence 2 in the ReLU-Sigmoid was as good as MSE.

About the loss history, if the gap between the training and the validation occurs, it means that the overfitting happens. The results of the loss history indicated that all combinations of loss function and activation had no overfitting.

The thesis indicated that the performance of I-divergence, which is one of β-divergence, was as good as the squared error. However, this study did not achieve the further conclusion for the I-divergence and the squared error: Which is more suitable for autoencoder? Therefore, further studies are needed in order to investigate which β-divergence has the best performance as the loss function of autoencoder.

References

[1] Francois Chollet. Building autoencoders in keras. `https://web.archive.org/web/20190107073638/https://blog.keras.io/building-autoencoders-in-keras.html`. Accessed: 2019-01-07.

[2] KAMEOKA Hirokazu. 非負値行列因子分解 [Non-negative Matrix Factorization]. 計測と制御 *[Journal of the Society of Instrument and Control Engineers]*, 51(9):837, 2012.

[3] OKATANI Takayuki. 深層学習 *[Deep Learning]*, pages 24–27. 機械学習プロフェッショナルシリーズ [Machine Learning Professional Series]. 講談社 [Kodansha Ltd.], 2015.

[4] Keras documentation - optimizers. `https://web.archive.org/web/20190117110324/https://keras.io/optimizers/`. Accessed: 2019-01-17.

[5] KAMEOKA Hirokazu. スパース表現に基づく音響信号のモデル化と分析 [The Modeling and Analysis of Acoustic Signal Based on Sparse Representation]. In データ生成モデルを活用するデータマイニング技術 *[Data Mining Technology Utilizing The Data Generation Model]*. 第６０回人工知能セミナー [The 60th Artificial Intelligence Seminar].

[6] OKATANI Takayuki. 深層学習 *[Deep Learning]*, page 48. 機械学習プロフェッショナルシリーズ [Machine Learning Professional Series]. 講談社 [Kodansha Ltd.], 2015.

[7] HARUGUCHI Takuma. Autoencoder for thesis. `https://github.com/s1230038/autoencoder/blob/master/AutoencoderForThesis.ipynb`. Accessed: 2019-01-23.

[8] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.