Study of Noise Injection and Gradient Regularization Toward Better Generalization

16B00133

Napat Thumwanit

Transdisciplinary Science and Engineering
School of Social and Environment
Tokyo Institute of Technology

Supervisor: Yukihiko YAMASHITA

January 2020

Contents

Abstract

The neural network has been prevalently applied to various fields, such as computer vision, natural language processing and so on. Despite its popularity, neural networks are prone to overfitting especially when the available data is scarce. In this research, we will focus on gradient regularization; several pieces of research have shown its ability to regularize the neural network in the situation where data are limited. We do an analysis on noise injection of its effect toward the loss, and based on that analysis, we propose new gradient regularization methods named Hessian-scaled Frobenius regularizer. Quantitative results show that our proposed methods can outperform other gradient regularization methods in some experiment cases which might be a potential regularizer with the need for further mathematical analysis regarding the effect of regularization as future work.

1 Introduction

Since the upcoming of neural network, many forms of models are created for tackling many kinds of tasks such as regression and classification. With the non-linearity introduced in the neural network, it holds a large capacity for expressing a complex task such as image classification or non-linear regressions. However, the high capacity might lead to overfitting or failure to generalize in unseen data which should be controlled with the process called regularization. Before neural network, there are several ways to regularize the linear regression such as ridge or Least Absolute Shrinkage and Selection Operator (LASSO). Likewise, regularization in neural network can be performed in many ways like data augmentation, for example, reflecting, shearing or bending the image, exposing the model with more variety of data and directly controlling the structure or parameters of the model such as weight decay or dropout.

1.1 Objective

The purpose of this thesis is to propose new regularization methods and compare the accuracy for a classification task with respect to the test data of the dataset by several regularization methods both existed and our proposed. Several papers of using gradients to regularize the model, neural network, have been purposed such as Double backpropagation [13] which penalizes the loss function by the gradient of the loss with respect to the input in order to suppress the change occurred by small change in the input. Even though this method was purposed since 1998, this method can perform considerably well in some dataset such as MNIST or CIFAR-10 [16]. On the other hand, there are several papers observing the regularization by injecting noise into the input or a layer of neural network. Based on the Taylor expansion, we can derive the relationship between the noise injection and gradient regularization. Despite knowing the fact, researches comparing the noise injection and gradient regularization are scarce. Therefore, this paper will be mainly discussed the evaluation of the results performed by two categories of methods that are gradient regularization and noise injection especially Gaussian's noise injection into input space. Furthermore, our proposed methods are based on the Taylor expansion of the perturbed loss and they are compared with other methods.

1.2 Organization

The structure of this paper is shown as follows,

In chapter 2, the fundamental of machine learning and deep learning will be explained, especially the composition of the neural network which is the main theme of this paper. It includes feed-forward neural network, activation function, convolution neural network and back propagation.

In chapter 3, the basic ideas of regularization for classical linear regression and neural networks will be introduced.

In chapter 4, the concept of the large margin classifier especially SVM will be explained as the basic concept for the related work.

In chapter 5, the researches and works related to gradient regularization and noise injection will be introduced. The methods derived from these works will be used in the experiments.

In chapter 6, the neural networks that will be used in the experiment will be described. Furthermore, the datasets and its usage will be introduced.

In chapter 7, our proposed regularization methods and its derivation will be explained. Then, the experimental methods and their results will be shown.

In chapter 8, the results will be summarized then discussed for further improvement in the future.

1.3 Research ethics and compliance to the codes of conducting research

Machine learning is one of the fields that enables the AI (artificial intelligence) to move forward and there are a lot of intense studies in recent years. Both the academic side and business side start utilizing AI for many kinds of applications such as image recognition, text analyzing, and so on.

Machine learning applications can bring convenience to many people's lives, for example, driverless cars, automation in the manufacturing line, face detection for authentication or personal assistant as a chatbot. However, no matter how technologies advance, many machine learning models are prone to unseen samples or, in the worst case, adversarial samples which neatly crafted by malicious attackers. This is crucial to the development of the system relying on machine learning because a small mistake by the machine may cause a catastrophic incident and lead to casualties.

This research is conducted focusing on improving the robustness of the neural network, one of the machine learning models, and comparing many methods to observe the robustness gained from them. This significantly improves the reliability of the machine learning models so that they are more capable to be implemented in the real-world system. In short, the reliability of machine learning will noticeably increase and become more practical for implementation.

For the ethics in this experiment, the data used in the research are cited and trained not for any commercial usage. Furthermore, the evaluation results are shown directly as a result of the experiment.

2 Convolution Deep Neural Network

2.1 Feed-Forward Neural Network

2.1.1 Neuron Node

The neuron node represents the linear operation of input vector $\boldsymbol{x} \in \mathbb{R}^h$ with weight vector and bias scalar $\boldsymbol{w} \in \mathbb{R}^h, b \in \mathbb{R}$ and then it is applied by an non-linear activation function $f_{\text{act}} : \mathbb{R} \to \mathbb{R}$ which the result is the output of the neuron node. It can be written as the following equation.

$$y = f(\boldsymbol{w}^\top \boldsymbol{x} + b)$$

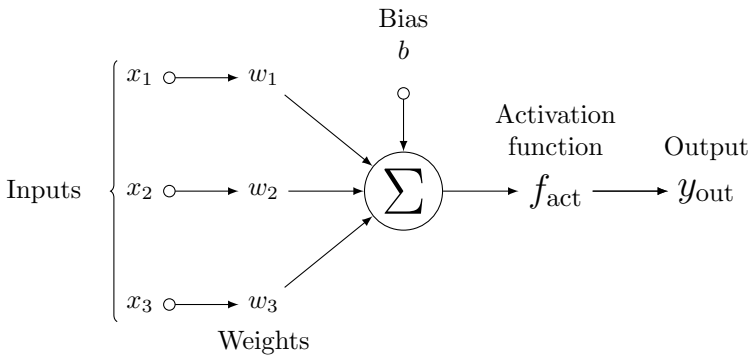


Figure 1: Diagram of a neuron node

2.1.2 Activation Functions

Biologically inspired, the activation function is represented as a gate which the incoming signal to a neuron node, in this case $\boldsymbol{w}^\top \boldsymbol{x} + b$, should be fired out to the next node or not. In the neural network context, the output values may vary infinitely on the range of $\mathbb{R}$, so that the activation function can determine the activated values that are the inputs for the next neuron nodes. The usage and meaning might differ on the types of activation function.

The important property of the activation function is it needs to be a non-linear function. It is important to introduce non-linearity into the neural network in order to increase its expressive power. In other words, the activation function helps the neural network to adapt better to arbitrary data.

Most of the activation functions likely to be differentiable so that the neural network can conduct backpropagation easily (that will be mentioned in section 2.3). The most widely used nowadays are Tanh, sigmoid, Relu, LeakyRelu, or softmax and vice versa. The following items are the formulas for each activation function with respect to $x \in \mathbb{R}$

1. Tanh: $\dfrac{e^{2x} - 1}{e^{2x} + 1}$.

2. Sigmoid: $\dfrac{1}{1+e^{-x}}$.

3. Relu (Rectified linear unit): $\max(0, x)$.

4. Leaky Relu: $\begin{cases} x, & \text{if } x > 0 \\ ax, & \text{otherwise} \end{cases}$ (where a is a small constant).

For the softmax, it is employed to determine the probability of each element in $\boldsymbol{x} \in \mathbb{R}^h$. For any $\boldsymbol{x}$, it will constrain the sum of the element in the output vector to 1 which represent the probability of each element.

$$[\text{softmax}(\boldsymbol{x})]_i = \frac{\exp(x_i)}{\sum_{j=1}^{h} \exp(x_j)}$$

, Note that $x_i \geq x_j \iff [\text{softmax}(\boldsymbol{x})]_i \geq [\text{softmax}(\boldsymbol{x})]_j$.

2.1.3 Neural Network

Trying to resemble the biological neural network system, neural network is a connection of many-to-many neuron nodes to pass the values through the layers of neurons. It can be illustrated as shown in Figure 2 in which the neural network is composed of input nodes, nodes in one hidden layer, and the output nodes.

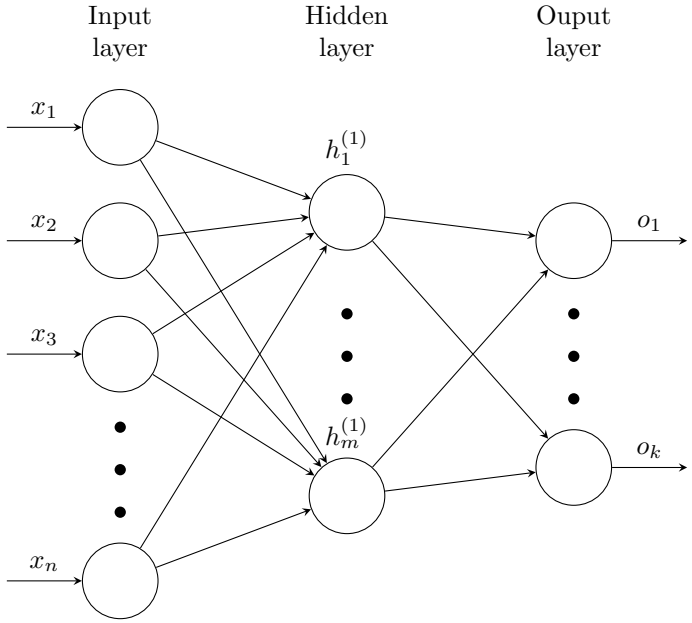


Figure 2: Neural network with one hidden layer

Similar to the neuron node, the computation of the neural network can be presented using the output from the previous layer $\boldsymbol{h}^{(i-1)} \in \mathbb{R}^m$ to the input for the next layer $\boldsymbol{h}^{(i)} \in \mathbb{R}^n$ by the weight matrix $\boldsymbol{W}^{(i)} \in \mathbb{R}^{m \times n}$, bias vector $\boldsymbol{b}^{(i)} \in \mathbb{R}^n$ and activation function $f^{(i)}$ through the following equations.

$$\boldsymbol{h}^{(i)} = f({\boldsymbol{W}^{(i)}}^\top \boldsymbol{h}^{(i-1)} + \boldsymbol{b}^{(i)}). \tag{1}$$

For the neural network in Figure 2, the input layer is $\boldsymbol{h}^{(0)} = \boldsymbol{x}$ and the output layer is $\boldsymbol{o} = \boldsymbol{h}^{(2)}$

2.1.4 Loss Function

While training the neural network, we need to evaluate how well the neural network is performing in order to update the parameters, weights and biases, through backpropagation. The losses we will use in the experiment are mean squared error (MSE), binary-entropy loss, and cross-entropy loss. The formulas are as shown below with respect to the output $\boldsymbol{y} \in \mathbb{R}^n$ and the target $\hat{\boldsymbol{y}} \in \mathbb{R}^n$ where n is the number of samples.

1. MSE:

$$\mathcal{L}_{mse} = \frac{1}{n} \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2 = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2. \tag{2}$$

2. Binary-entropy loss:

$$\mathcal{L}_{binary} = \frac{1}{n} \sum_{i=1}^{n} (y_i (\log \hat{y}_i) + (1 - y_i)(\log(1 - \hat{y}_i))). \tag{3}$$

Cross-entropy loss is employed for multi-label classification. Let the output is $\boldsymbol{y} \in \mathbb{R}^{n \times c}$ and the target $\hat{\boldsymbol{y}} \in \mathbb{R}^{n \times c}$ where n, c are the number of samples and classes respectively. Here we will assume that the output was activated by softmax.

1. Cross-entropy loss:

$$\mathcal{L}_{cross} = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{c} \hat{y}_{ij} (\log y_{ij}). \tag{4}$$

It might be useful to note that the losses above are all convex with respect to the output.

2.2 Convolutional Neural Network

Convolutional neural networks (LeCun, 1989), or as known as convolutional neural networks (CNNs) is a kind of network for processing data with grid-like topology. For instance, time-series data which can be taken as a 1-D grid from

each section of the data or, in this experiment, image data witch a group of 2-D grid pixel can be processed. The neuron in CNNs has a reception field adjusted by the window size and connects to other neuron overs the entire image. It can be interpreted that each neuron is detecting different features of the image, such as lines, curves or recognizing more complex features such as faces, texts and vice versa.

The very first CNNs [13] can be seen in Figure 3. It succeeded in recognizing handwritten zip code and, later on, with higher computational power, a deeper or larger architectures were employed for many kind of image recognition tasks.

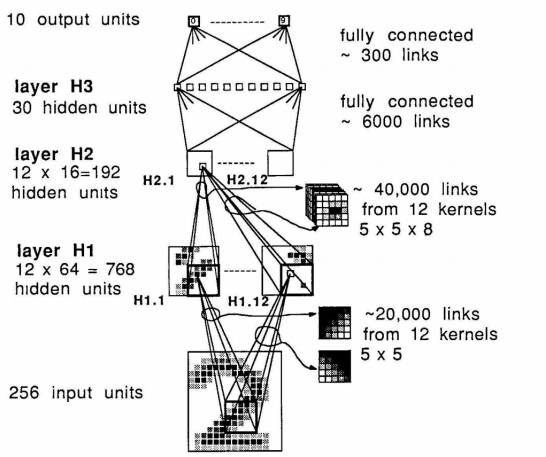


Figure 3: CNNs from the paper "Backpropagation Applied to Handwritten Zip Code Recognition"

2.2.1 Convolutional Layer

Convolutional layers are the main component of the CNNs which can be interpreted as feature extraction layers similarly to its original application in computer vision. In image processing, the convolution flips the kernel then conduct dot product on the reception field. The reason that the convolution need to be flipped is to conserve the commutative properties. Let I is the image and K is the kernel, we can write the convolution as the following

$$\begin{aligned} S(i,j) \coloneqq (I * K)(i,j) &= \sum_m \sum_n I(m,n)K(i-m, j-n), \\ &= (K * I)(i,j) = \sum_m \sum_n K(m,n)I(i-m, j-n). \end{aligned}$$

However, in neural network the commutative properties are unnecessary, instrad, for convenience, most libraries implement the **cross-correlation** instead

$$S(i,j) = (K \otimes I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n).$$

In CNNs, each layer with input sized $H \times W \times C$ consists the D kernels sized $h \times w \times C$ and bias sized D for each output channel will output a tensor with size $H_2 \times W_2 \times D$ which can be determined by the following formula

$$W_2 = \frac{W - w + 2P}{S} + 1,$$
$$H_2 = \frac{H - h + 2P}{S} + 1,$$

where P is padding size, S is stride value.

Mathematically, for a kernel $\boldsymbol{W} \in \mathbb{R}^{h \times w \times C_2 \times C_1}$, bias $\boldsymbol{b} \in \mathbb{R}^{C_2}$ and padded input $\boldsymbol{x'} \in \mathbb{R}^{H \times W \times C_1}$ which the output $\boldsymbol{y} \in \mathbb{R}^{H' \times W' \times C_2}$ where s is the stride, we can formulate the convolution as the following equation

$$y_{ijc} = \sum_{k=1}^{h} \sum_{l=1}^{w} \boldsymbol{w}_{klc}^{\top} \boldsymbol{x'}_{si+k-1, sj+l-1} + b_c, \tag{5}$$

and the equation actually can be understood intuitively as Figure 4. After the convolutional layer, it is usually followed by an activation function such as ReLU.

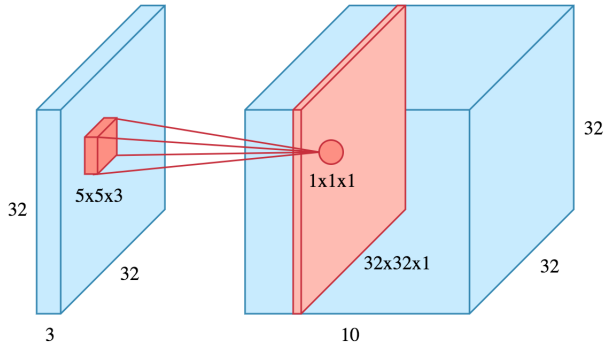


Figure 4: Illustration of a convolutional layer

2.2.2 Pooling Layer

After the convolutional layer and the non-linear activation function, we introduce further a layer to modify the output called **pooling**. A pooling layer replaces the output with the summary statistics of the nearby output. For example, replace the 2×2 output field with its maximum element, max pooling,

or with the average of all 4 elements, average pooling. Similar to convolutional layer, in this layer, mainly 2 parameters to control are the window size and the stride which the most usual combination is window size 2×2 and stride 2 as suggested in Figure 5.

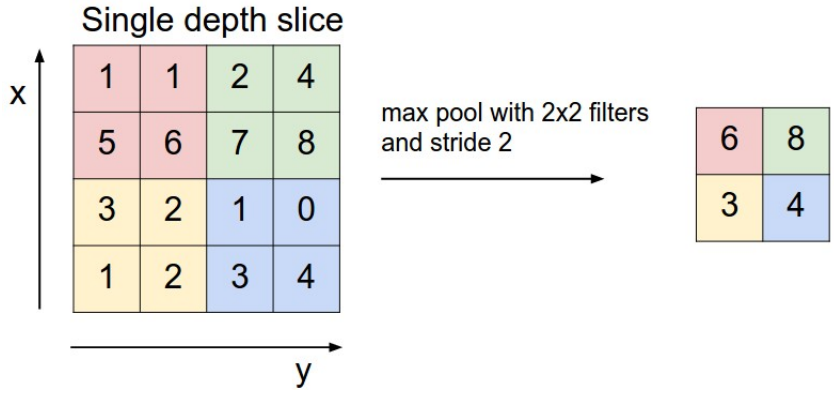


Figure 5: 2×2 max pooling with 2 stride

2.3 Backpropagation

In a simple linear regression such as $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$ where the loss function is $\mathcal{L}(\boldsymbol{x}, y) = (f(\boldsymbol{x}) - y)^2$, the differentials of the loss with respect to the variable $\boldsymbol{w}, b$ are easily calculated by the formulas

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = 2(f(\boldsymbol{x}) - y)\boldsymbol{x},$$
$$\frac{\partial \mathcal{L}}{\partial b} = 2(f(\boldsymbol{x}) - y)$$

which can be applied to optimization method such as gradient descent.

However, this is not the case for a neural network where there are a lot of layers stacking on each other and can be represented as the following equation

$$f(\boldsymbol{x}, \theta) = f_n(f_{n-1}(\cdots f_2(f_1(\boldsymbol{x}, \theta_1), \theta_2) \cdots, \theta_{n-1}), \theta_n),$$

and most of the layers are non-linear in which the formula cannot be derived analytically. To alleviate this problem, rather than deriving the formula, instead exploiting the chain rule property is convenient especially for a layer-like structure such as a neural network. The methodology will be explained through the computational graph and, later on, focus on each type of layer in the neural network.

2.3.1 Computational Graph

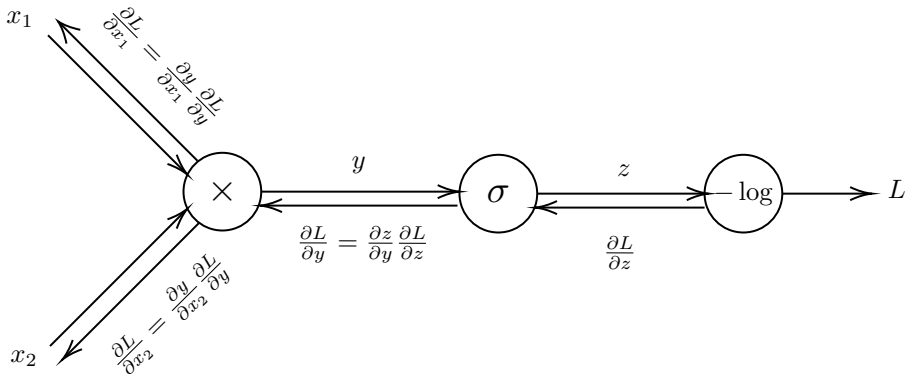


Figure 6: Simple Computational Graph

The computational graph represents the flow of calculation. As can be seen in Figure 6, the nodes represent the operation and the arrows from left to right represent the forward calculation, or forward pass, while the opposite direction arrows are what will be explained later, backpropagation. The equation of the sample computational graph can be written as follow:

$$
\begin{aligned}
y &= x_1 x_2, \\
z &= \frac{1}{1+\exp(-y)}, \\
L &= -\log(z).
\end{aligned}
$$

In order to calculate $\partial\mathcal{L}/\partial x_1$ and $\partial\mathcal{L}/\partial x_2$, backpropagation can exploit the chain rule rather than directly derive a complex equation. This method might compute the differentiation, or gradient, with respect to variable which the complexity be as same as the forward pass. The derivation of the backward pass can be aligned as the following equations.

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial z} &= -\frac{1}{z} \\
\frac{\partial \mathcal{L}}{\partial y} &= \frac{\partial z}{\partial y}\frac{\partial \mathcal{L}}{\partial z} = z(1-z)\left(-\frac{1}{z}\right) = z-1 \\
\frac{\partial \mathcal{L}}{\partial x_1} &= \frac{\partial y}{\partial x_1}\frac{\partial \mathcal{L}}{\partial y} = x_2(z-1) \\
\frac{\partial \mathcal{L}}{\partial x_2} &= \frac{\partial y}{\partial x_2}\frac{\partial \mathcal{L}}{\partial y} = x_1(z-1)
\end{aligned}
$$

It can be observed that the inputs and outputs should be stored in order to compute the derivation for each step. Therefore, this method can be memory consuming for a larger neural network.

2.3.2 Backpropagation for Feed Forward Layer

Considering a single feed forward layer as equation 1, let assume that the activation function is ReLU, the feed forward layer can be written step-by-step by the following equations. $\boldsymbol{h}^{(i-1)} \in \mathbb{R}^m, \boldsymbol{h}^{(i)} \in \mathbb{R}^n, \boldsymbol{W}^{(i)} \in \mathbb{R}^{m \times n}$, and $\boldsymbol{b}^{(i)} \in \mathbb{R}^n$

$$\begin{aligned} \boldsymbol{h}'^{(i)} &= \boldsymbol{W}^{(i)^\top} \boldsymbol{h}^{(i-1)} + \boldsymbol{b}^{(i)}, \\ \boldsymbol{h}^{(i)} &= \mathrm{ReLU}(\boldsymbol{h}'^{(i)}). \end{aligned}$$

The differentiation of ReLU can be written case-separately,

$$\mathrm{ReLU}(x) = \max(0, x) \Rightarrow \frac{\partial \mathrm{ReLU}(x)}{\partial x} = (x > 0) := \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

though the differentiation at $x = 0$ is undefined. The experiment conducted in this thesis based on the library Tensorflow which treats it as 0.

For the differentiation $\partial \mathcal{L}/\partial \boldsymbol{W}^{(i)}, \partial \mathcal{L}/\partial \boldsymbol{b}^{(i)}$ and $\partial \mathcal{L}/\partial \boldsymbol{h}^{(i-1)}$ can be calculated on matrix multiplication.

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{(i)}} &= \left(\frac{\partial \boldsymbol{h}'^{(i)}}{\partial \boldsymbol{W}^{(i)}} \right)^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}'^{(i)}}, \\ \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^{(i)}} &= \left(\frac{\partial \boldsymbol{h}'^{(i)}}{\partial \boldsymbol{b}^{(i)}} \right)^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}'^{(i)}}, \\ \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}^{(i-1)}} &= \left(\frac{\partial \boldsymbol{h}'^{(i)}}{\partial \boldsymbol{h}^{(i-1)}} \right)^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}'^{(i)}}, \end{aligned}$$

where the term $\partial \boldsymbol{h}'^{(i)}/\partial \boldsymbol{W}^{(i)}$ becomes 3-D tensor and others term that can be simplified for efficient gradient calculation as the following:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{(i)}} &= \left(\boldsymbol{h}^{(i-1)} \right)^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}'^{(i)}}, \\ \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^{(i)}} &= \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}'^{(i)}}, \\ \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}^{(i-1)}} &= \left(\boldsymbol{W}^{(i)} \right)^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}'^{(i)}}, \end{aligned}$$

where $\partial \boldsymbol{h}'^{(i)}/\partial \boldsymbol{b}^{(i)} = \boldsymbol{I}, \partial \boldsymbol{h}'^{(i)}/\partial \boldsymbol{h}^{(i-1)} = \boldsymbol{W}^{(i)}$ and $\partial \mathcal{L}/\partial \boldsymbol{h}'^{(i)}$ can be calculated from the gradient with respect to the output $\boldsymbol{h}^{(i)}$ as the differentiation of ReLU in (6).

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}'^{(i)}} = \left(\boldsymbol{h}'^{(i)} > 0\right) \odot \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}^{(i)}}.$$

2.3.3 Backpropagation for Convolution Layer

From (5), we can see that convolution can be written similarly to a feed forward layer; consequently, the differentiation also turned out to be similar. It might be easier to accumulate the differentiation with respect to each element.

$$\begin{aligned}
\frac{\partial y_{ijc}}{\partial \boldsymbol{x}'_{si+k-1,sj+l-1}} &= \boldsymbol{w}_{klc}, \\
\frac{\partial y_{ijc}}{\partial \boldsymbol{w}_{klc}} &= \boldsymbol{x}'_{si+k-1,sj+l-1}, \\
\frac{\partial y_{ijc}}{\partial b_c} &= 1.
\end{aligned}$$

This operation can be furthermore vectorized by the operation **im2col**, an operation to unfold the input and kernel for conducting matrix multiplication. After the operation, only a simple matrix multiplication is needed which greatly enhances the performance especially parallel processor such as GPU.

$$\boldsymbol{y} = (\hat{\boldsymbol{W}})^\top \hat{\boldsymbol{x}}' + \boldsymbol{b}.$$

The backpropagation can be firstly done on $\hat{\boldsymbol{W}}$ and, later, fold it back to $\boldsymbol{W}$ by the inverse operation **col2im**.

2.3.4 Optimization

Optimization is an algorithm to minimize or maximize a function with respect to the inputs or parameters. In the neural network, we will minimize the loss function $\mathcal{L}(\theta; \boldsymbol{x}, \boldsymbol{y})$ where θ is the trainable parameters for the neural network $f(\boldsymbol{x}, \theta)$. Due to the accessibility to the differentiation of the neural network, we usually apply first-order optimization or occasionally second-order one. Some example of algorithm that will be introduced here are Stochastic Gradient Descent (SGD), SGD with momentum, Nesterov Accelerated Gradient, Adaptive gradient (AdaGrad), Root Mean Square Propagation (RMSprop), and Adaptive Moment estimation (Adam). Note that all the algorithm here might converge into the local minimum instead of global minimum.

1. Steepest Descent Method is the simplest method using the gradient as the steepest direction to decrease the value of the function within the neighborhood. For any function $L(\theta)$, we can minimize using this method with θ_0 as an initial points by the following update rule.

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta f(\theta_k). \tag{7}$$

where α is the learning rate.

2. Stochastic Gradient Descent (SGD) is a method based on Steepest Descent Method. We train the parameter θ by randomly selecting training samples $\boldsymbol{x}^{(t)} \sim \boldsymbol{X}$ with training labels $\boldsymbol{y}^{(t)} \sim \boldsymbol{Y}$ then apply the Steepest Descent Method with respect to the parameters.

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta \mathcal{L}_{(\boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)}) \sim (\boldsymbol{X}, \boldsymbol{Y})}(\theta_t; \boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)}). \tag{8}$$

For the readability, the symbol represented sampling training data $(\boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)}) \sim (\boldsymbol{X}, \boldsymbol{Y})$ will be omitted.

3. SGD with Momentum [14] employs the concept of momentum to make the direction of gradient change smoothly which empirically proved that it increases the speed of convergence.

$$\boldsymbol{v}_{t+1} = \beta \boldsymbol{v}_t + (1 - \beta) \nabla_\theta \mathcal{L}(\theta_t; \boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)}), \tag{9}$$

$$\theta_{t+1} = \theta_t - \alpha \boldsymbol{v}_{t+1} \tag{10}$$

for some $0 < \beta < 1$.

4. Adaptive Gradient Algorithm (AdaGrad) [5] accumulates the sum of the squared gradient to adjust the step length. When the magnitude of the gradient becomes large, by dividing with the accumulated squared gradient, it will reduce the step size so that it is less punishing for mistaken direction.

$$\boldsymbol{\Sigma}_{t+1} = \boldsymbol{\Sigma}_t + (\nabla_\theta \mathcal{L}(\theta_t))^2, \tag{11}$$

$$\theta_{t+1} = \theta_t - \frac{\alpha \nabla_\theta \mathcal{L}(\theta_t)}{\sqrt{\boldsymbol{\Sigma}_{t+1}} + \epsilon}. \tag{12}$$

5. Root Mean Square Propagation (RMSprop)[1] is similar to Adagrad, but introduces momentum update to the accumulated squared gradient.

$$\boldsymbol{\Sigma}_{t+1} = \beta \boldsymbol{\Sigma}_t + (1 - \beta)(\nabla_\theta \mathcal{L}(\theta_t))^2, \tag{13}$$

$$\theta_{t+1} = \theta_t - \frac{\alpha \nabla_\theta \mathcal{L}(\theta_t)}{\sqrt{\boldsymbol{\Sigma}_{t+1}} + \epsilon}. \tag{14}$$

6. ADAM optimizer [12] is combining AdaGrad, and the concept of momentum update to determine the directions of the step. This method is one of the most popular optimization method for neural network.

[1] http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

$$\boldsymbol{m}_{t+1} = \beta_1 \boldsymbol{m}_t + (1-\beta_1)\nabla_\theta \mathcal{L}(\theta_t), \tag{15}$$

$$\boldsymbol{v}_{t+1} = \beta_2 \boldsymbol{v}_t + (1-\beta_2)(\nabla_\theta \mathcal{L}(\theta_t))^2, \tag{16}$$

$$\hat{\boldsymbol{m}}_{t+1} = \frac{\boldsymbol{m}_{t+1}}{(1-\beta_1^t)}, \tag{17}$$

$$\hat{\boldsymbol{v}}_{t+1} = \frac{\boldsymbol{v}_{t+1}}{(1-\beta_2^t)}, \tag{18}$$

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{\boldsymbol{m}}_{t+1}}{\sqrt{\hat{\boldsymbol{v}}_{t+1}} + \epsilon}. \tag{19}$$

2.4 Batch Normalization

In order to alleviate the vanishing gradient problem, the batch normalization scales the input by its empirical mean and variance. It was claimed that the batch normalization can reduce overfitting due to the usage of data variation and also a larger learning rate can be used without exploding the gradients [11]. The normalization for a batch with m samples $\boldsymbol{x} \in \mathbb{R}^m$ is shown as the following equation.

$$\mu_b = \frac{1}{m}\sum_{i=1}^{m} x_i, \tag{20}$$

$$\sigma_b^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_b)^2, \tag{21}$$

$$\hat{x_i} = \frac{x_i - \mu_b}{\sqrt{\sigma_b^2 + \epsilon}}, \tag{22}$$

$$y_i = \gamma \hat{x_i} + \beta, \tag{23}$$

where μ_b, σ_b are batch mean and variance, $\hat{x_i}$ is normalized value of x_i, ϵ is a small constant to prevent from numerical instability, γ and β are parameters for scaling the normalized input which both of them are trainable.

In the test time, the empirical mean and variance are intractable. In order to tackle that problem, while training, the running mean and running variance $\hat{\mu}, \hat{\sigma}$, which are used in place of empirical ones, should be adjust by using momentum update as the following:

$$\hat{\mu} := \alpha\hat{\mu} + (1-\alpha)\mu_b, \tag{24}$$

$$\hat{\sigma} := \alpha\hat{\sigma} + (1-\alpha)\sigma_b \tag{25}$$

3 Regularization

In machine learning, the model is trained in order to generalize well to the unseen data so it should be regulated by some rule forcing the model not to get too "used to" the training data. In this section, the regularization for linear regression and neural network will be introduced.

3.1 Regularization in Linear Regression

Briefly, the linear regression is the problem to minimize the MSE from n-sample target $\boldsymbol{y} \in \mathbb{R}^n$ with respect to the n-samples d-dimensional inputs $\boldsymbol{X} \in \mathbb{R}^{n \times (d+1)}$ by adjusting the weight $\boldsymbol{w} \in \mathbb{R}^{d+1}$, which an extra dimension is for bias. The equation

$$\mathcal{L}(\boldsymbol{w}, b; \boldsymbol{X}, y) = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2 . \tag{26}$$

can be solved analytically or using gradient descent. However, directly optimizing the loss in equation 26 might lose generality due to the outliers or bad features. Usually, it can be considered that the model is more expressive when the magnitude of $\boldsymbol{w}$ become large. This idea leads to the following regularization methods, Least absolute shrinkage and selection operator (LASSO) and ridge.

3.1.1 LASSO

LASSO or as known as $L1$-regularizer penalizes the weight by it $L1$-norm by a parameter $\lambda \in \mathbb{R}$ which control the penalty due to the regularization. Instead, We can minimize the following loss function.

$$\mathcal{L}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}, \lambda) = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda \|\boldsymbol{w}\|_1 . \tag{27}$$

The analytical solution is fairly complex so it will not be covered here. LASSO is a regularizer which encourages sparse weights [7].

3.1.2 Ridge

Similar to LASSO, Ridge regression penalize the $L2$-norm of the weight and lead us to the optimization of the following loss function.

$$\mathcal{L}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}, \lambda) = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda \|\boldsymbol{w}\|_2^2 \tag{28}$$

in which this loss function can be solved analytically as below

$$\hat{\boldsymbol{w}} = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I}) \boldsymbol{X}^\top \boldsymbol{y}. \tag{29}$$

Contrasting with LASSO, Ridge encourages smoother weights with small magnitude [7].

3.2 Regularization in Neural Network

Neural network works well with non-linear data, for example, nonlinear regression or non-linearly separable classification, because of its high expressive power due to the non-linear activation layers. However, it might cause the overfitting to the train data because of its own expressive power. As a regularizer, two main methods will be introduced which are weight decay and dropout. Some other methods to reduce overfitting might not be count as regularization, for instance, data augmentation which tries to expose the model to more data modified in a feasible range.

3.2.1 Weight Decay

Let consider a layer with weight $\boldsymbol{w} \in \mathbb{R}^d$. It can be penalized the weight similarly to the ridge regression as the following loss function.

$$\mathcal{L}_{decay}(\theta; \boldsymbol{X}, \boldsymbol{y}) = \mathcal{L}(\theta; \boldsymbol{X}, \boldsymbol{y}) + \lambda \left\|\boldsymbol{w}\right\|_2^2 . \tag{30}$$

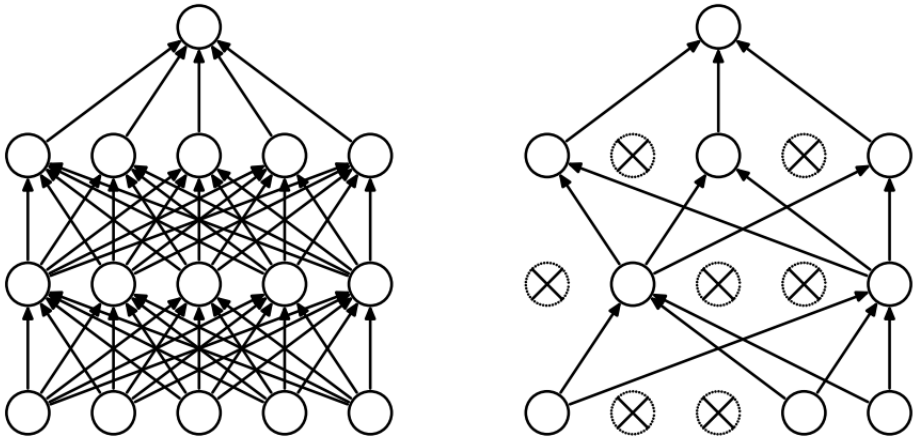
Assume that we apply SGD to update the parameter. Hence, the update with respect to the weight $\boldsymbol{w}$ will be given by

$$\begin{aligned} \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \alpha \nabla_{\boldsymbol{w}} \mathcal{L} - 2\alpha\lambda \boldsymbol{w}_t \\ &= (1 - 2\alpha\lambda)\boldsymbol{w}_t - \alpha \nabla_{\boldsymbol{w}} \mathcal{L}. \end{aligned}$$

Then, weight $\boldsymbol{w}$ is shrinking proportionally. Therefore, this method is called "weight decay" or to reduce the magnitude of the weight so that it is less prone to overfitting problem [7]. The factor λ can be adjusted appropriately for each layer.

3.2.2 Dropout

Dropout is a method to drop some units, or values, to zero. It was claimed that this method prevents the neural network from adapting or relying on a single node too much and encourages ensemble from averaging multiple units together when test time [15]. Dropout can be illustrated as shown in Figure 7



(a) Standard Neural Net (b) After applying dropout.

Figure 7: Dropout illustration

Consider a hidden layer l. Let $\boldsymbol{z}^{(l)}$ denote the vector of input into layer l, $\boldsymbol{y}^{(l)}$ denote the vector of output from layer l. Let $\boldsymbol{W}^{(l)}$ and $\boldsymbol{b}^{(l)}$ denote weight and bias of layer l, a standard feed-forward layer can be describe as

$$
\begin{aligned}
z_i^{l+1} &= \boldsymbol{w}_i^{l+1}\boldsymbol{y}^{(l)} + b_i^{(l)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}).
\end{aligned}
$$

By applying dropout, a mask from Bernoulli distribution with probability p to be 1 will be generated.

$$
\begin{aligned}
r_j^{(l)} &\sim \text{Bernoulli}(p), \\
\widetilde{\boldsymbol{y}}^{(l)} &= \boldsymbol{r}^{(l)} \odot \boldsymbol{y}^{(l)}, \\
z_i^{l+1} &= \boldsymbol{w}_i^{l+1}\widetilde{\boldsymbol{y}}^{(l)} + b_i^{(l)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)})
\end{aligned}
$$

4 Support Vector Machine (SVM)

For simplicity, the classification problem to be discussed will return to two-class classification problem which can be written using linear models as the form

$$y(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}) + b, \tag{31}$$

where $\boldsymbol{\phi}(\boldsymbol{x})$ denotes a fixed feature-space transformation, such as Gaussian kernel $\exp(-x^2)$. The training dataset comprises N input vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ with their corresponding targets $t_1, \ldots, t_N$ where $t_n \in \{-1, 1\}$ and test data $\boldsymbol{x}$, new data points, will be classified based on the sign of $y(\boldsymbol{x})$.

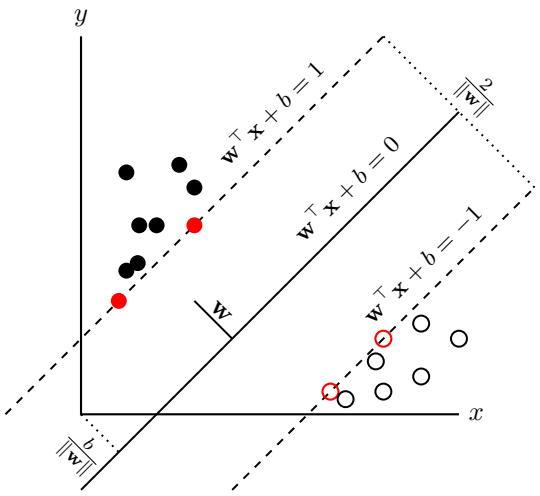


Figure 8: Decision boundary and its margin

In fact, there are many solutions to approach this problem that guarantee to fit the training data with finite iterations though the results might depend on the initial value of parameters $\boldsymbol{w}$ and b. To find the solution which generalizes the best, the support vector machine (SVM) tackles this problem through the concept of the margin, which is defined to be the smallest distance between the decision boundary and the samples as shown in figure 8.

The perpendicular distance of an arbitrary point $\boldsymbol{x}$ from a hyperplane $y(\boldsymbol{x}) = 0$ is given by $|y(\boldsymbol{x})| / \|\boldsymbol{w}\|$. If the case which all samples are classified correctly is considered, it can be easily shown that $t_n y(\boldsymbol{x}_n) > 0$. Therefore, the distance of a point $\boldsymbol{x}_n$ to the decision surface is

$$\frac{t_n y(\boldsymbol{x})}{\|\boldsymbol{w}\|} = \frac{t_n(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}) + b)}{\|\boldsymbol{w}\|}. \tag{32}$$

The margin is given by the smallest distance while we would like to maximize the margin. Thus the problem can be formulated as an optimization of $\boldsymbol{w}$ and

b to maximize the distance

$$\underset{\boldsymbol{w},b}{\operatorname{argmax}} \left\{ \frac{1}{\|\boldsymbol{w}\|} \min_n \left[t_n(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_n) + b) \right] \right\}. \tag{33}$$

However, to solve (33) directly is complicated and further simplification can be done. From an observation, the distance from any point $\boldsymbol{x}_n$ is unchanged for any rescaling $\boldsymbol{w} \to \tau \boldsymbol{w}$ and $b \to \tau b$ so it can be used to freely set

$$t_n(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_n) + b) = 1 \tag{34}$$

for the point that is closest to the boundary and for all data points, they satisfy

$$t_n(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_n) + b) \geq 1. \tag{35}$$

Furthermore, the optimization in (33) can be simplified into

$$\underset{\boldsymbol{w}}{\operatorname{argmax}} \frac{1}{\|\boldsymbol{w}\|} \iff \underset{\boldsymbol{w}}{\operatorname{argmin}} \frac{1}{2} \|\boldsymbol{w}\|^2, \tag{36}$$

where the factor $1/2$ is included for later convenience. The problem is turned into a quadratic programming problem where an equation is optimized while there is a set of inequality constraints. Even the variable b disappears from the optimization, it is implicitly adjusted by the constraints. In order to solve the problem, the Lagrange multipliers $a_n \geq 0$ are introduced giving Lagrangian function

$$L(\boldsymbol{w}, b, \boldsymbol{a}) = \frac{1}{2} \|\boldsymbol{w}\|^2 - \sum_{n=1}^{N} a_n \left\{ t_n(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_n) + b) - 1 \right\}. \tag{37}$$

By taking partial derivative with respect to $\boldsymbol{w}$ and b and let them be zeros, two conditions are obtained

$$\partial L / \partial \boldsymbol{w} = 0 \quad (\Rightarrow) \quad \boldsymbol{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\boldsymbol{x}_n), \tag{38}$$

$$\partial L / \partial b = 0 \quad (\Rightarrow) \quad 0 = \sum_{n=1}^{N} a_n t_n. \tag{39}$$

Next, the variable $\boldsymbol{w}$ in (37) can be eliminated by substituting $\boldsymbol{w}$ from (38) and b can be eliminated because its coefficient is exactly the same as RHS of (39). Therefore, we derive an equation relying only on $\boldsymbol{a}$ as called as **dual representation**

$$\tilde{L}(\boldsymbol{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m \boldsymbol{\phi}(\boldsymbol{x}_n)^\top \boldsymbol{\phi}(\boldsymbol{x}_m) \tag{40}$$

with the following constraints

$$a_n \geq 0, \tag{41}$$

$$\sum_{n=1}^{N} a_n t_n = 0. \tag{42}$$

This problem, a_n, can be solved by quadratic programming which will be omitted. Instead of solving $\boldsymbol{w}$ and b, we solve a_n which can be used for deriving $\boldsymbol{w}$ and b in the later steps. By substituting $\boldsymbol{w}$ in (38) into (31), the linear model will be represented in the form of a_n and b as

$$y(\boldsymbol{x}) = \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}_n) + b = \sum_{n=1}^{N} a_n t_n k(\boldsymbol{x}, \boldsymbol{x}_n) + b, \tag{43}$$

where $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}')$ is called **kernel function**. Here is the list of most used kernel functions.

Table 1: List of typical kernel functions

Kernel name	Formula
Linear	$\boldsymbol{x}'^\top \boldsymbol{x}$
RBF (Gaussian)	$\exp(-\left\|\boldsymbol{x} - \boldsymbol{x}'\right\|^2 / 2\sigma^2)$
Polynomial	$(1 + \boldsymbol{x}^\top \boldsymbol{x})^d$

From the Karush–Kuhn–Tucker conditions (KKT), the complementary slackness is derived which the following condition should be satisfied.

$$a_n \{t_n y(\boldsymbol{x}_n) - 1\} = 0. \tag{44}$$

For any data point $\boldsymbol{x}_n$ such that $t_n y(\boldsymbol{x}_n) - 1 > 0$, a_n should be zero. Therefore, $a_n \neq 0$ if and only if $t_n y(\boldsymbol{x}_n) - 1 = 0$ or only on the data points which lying on the maximum margin hyperplane. From this fact, only data points that come into consideration are only a few points on the maximum margin hyperplane and the set S is defined as the indices of the points that are "active" and it is called **support vector**. Here, we can rearrange (43) as the following for less computational complexity

$$y(\boldsymbol{x}) = \sum_{n \in S} a_n t_n k(\boldsymbol{x}, \boldsymbol{x}_n) + b. \tag{45}$$

For any support vector $\boldsymbol{x}_n$, it is satisfied $t_n y(\boldsymbol{x}_n) = 1$ then substitute by (43), we yield

$$t_n \left(\sum_{m \in S} a_m t_m k(\boldsymbol{x}_m, \boldsymbol{x}_n) + b \right) = 1. \tag{46}$$

To determine the value of variable b, it is possible to pick any support vector $\boldsymbol{x}_n$ and solve (46) directly. However, that might cause numerical instability; hence, instead, we multiply (46) by t_n, which $t_n^2 = 1$, and average them over all support vector then the answer for b is

$$b = \frac{1}{N_s} \sum_{n \in S} \left(t_n - \sum_{m \in S} a_m t_m k(\boldsymbol{x}_m, \boldsymbol{x}_n) \right), \tag{47}$$

where N_s is the number of support vector ($N_s = |S|$).

5 Related Work

As introduced in the previous sections, there are number of ways to regularize the neural network such as weight decay and dropout. Meanwhile, another idea of regularizing the gradient instead to suppress large change with respect to a small change in the input space or directly inject the noise to the input space or hidden layer.

5.1 Gradient Regularization

Gradient regularization is introduced by many interpretations, such as encouraging smoothness of the output space or decision boundary, or encouraging a large margin decision boundary which are introduced in this section. For convenience, we will define our model, the neural network, and the loss as the following relation.

$$\mathcal{L}(\boldsymbol{x}, y, \theta) = M(f(\boldsymbol{x}, \theta), y) = M(\text{softmax}(g(\boldsymbol{x}, \theta), y)). \tag{48}$$

5.1.1 Double Backpropagation

This idea is the very firstly purposed gradient regularization in 1991 by Y. LeCun [4]. It penalizes the loss by the gradient of the loss with respect to the input as

$$\mathcal{L}_{DG}(\boldsymbol{x}, y; \theta) = \mathcal{L}(\boldsymbol{x}, y; \theta) + \lambda \left\| \nabla_{\boldsymbol{x}} \mathcal{L} \right\|_2^2 . \tag{49}$$

The loss function explicitly pushes the norm of the gradient to zeros which encourages the minimum to be broader and increases generalization. It has been proved to be succeed on a handwritten numbers dataset.

5.1.2 Jacobian Regularization

1. **Jacobian Regularizer (JacReg)** [6] Penalizes the squared Frobenius norm of the Jacobian of the softmax output with respect to the input:

$$\mathcal{L}_{JacReg}(\boldsymbol{x}, y; \theta) = \mathcal{L}(\boldsymbol{x}, y; \theta) + \lambda \left\| J_f \right\|_F^2 . \tag{50}$$

Inspired by large margin classifier, they defined the *generalization error* as the difference between training set and test set as

$$\text{GE}(g) = |l_{\text{exp}}(g) - l_{\text{emp}}(g)|, \tag{51}$$

where l_{exp} and l_{emp} is the loss on the training set and test set respectively. Furthermore, they defined the terms **score** ($o(s_i)$) and **margin** ($\gamma^d(s_i)$) as the followings.

$$o(s_i) = \min_{j \neq y_i} \sqrt{2}(\boldsymbol{\delta}_{y_i} - \boldsymbol{\delta}_j)^\top f(\boldsymbol{x}_i), \tag{52}$$

$$\gamma^d(s_i) = \sup\{a : d(\boldsymbol{x}_i, \boldsymbol{x}) \leq a \Rightarrow g(\boldsymbol{x}) = y_i \ \forall \boldsymbol{x}\}, \tag{53}$$

where s_i is the pair of training sample and its label $(\boldsymbol{x}_i, y_i)$, $\boldsymbol{\delta}_i$ is the Kronecker delta vector with $(\boldsymbol{\delta}_i)_i = 1$, and $(\boldsymbol{\delta}_i)_j = 0$ where $i \neq j$. The authors of this paper claimed the following lower bound for the margin

$$\gamma^d(s_i) \geq \frac{o(s_i)}{\sup_{\boldsymbol{x}:\|\boldsymbol{x}-\boldsymbol{x}_i\|_2 \leq \gamma^d(s_i)} \|\boldsymbol{J}(\boldsymbol{x})\|_2}, \tag{54}$$

where $\boldsymbol{J}(\boldsymbol{x})$ is the Jacobian of the probabilities with respect to the input; therefore, its dimension is $D \times C$ where D is the dimension of the input and C is the number of classes.

As the lower bound (54) suggests, the margin can be confirmed to be large if the tractable term $\|\boldsymbol{J}(\boldsymbol{x})\|_2$ can be minimize. Hence, this inspired the idea of regularizing the Jacobian. However, calculating the spectral norm of a matrix can be expensive. To circumvent this problem, the authors of this paper suggests penalizing the Frobenious norm instead based on the following relationship between spectral norm and Frobenious norm.

$$\frac{1}{\operatorname{rank}(\boldsymbol{J}(\boldsymbol{x}_i))} \|\boldsymbol{J}(\boldsymbol{x}_i)\|_F^2 \leq \|\boldsymbol{J}(\boldsymbol{x}_i)\|_2^2 \leq \|\boldsymbol{J}(\boldsymbol{x}_i)\|_F^2 . \tag{55}$$

2. **Frobenius Regularizer (FrobReg)** [16] Penalizes the squared Frobenius norm of the Jacobian of the logits with respect to the input. The only difference from JacReg is that it penalizes the logits instead of probabilities.

$$\mathcal{L}_{\text{FrobReg}}(\boldsymbol{x}, y; \theta) = \mathcal{L}(\boldsymbol{x}, y; \theta) + \lambda \|J_g\|_F^2 \tag{56}$$

5.2 Noise Injection

Noise injection consists in adding noise into the neural network during the training, or possibly testing. Several experimental results show the improvement of generalization ability [1]. However, the justification of the improvement remains unclear and the theoretical expected value of the perturbed cost cannot be calculated analytically. By performing Taylor expansion, we can estimate the deterministic perturbed cost which the formula is lead to gradient penalization. One of the simplest form of noise injection is injecting Gaussian noise $\epsilon \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. The perturbed cost due to the noise can be defined as

$$\mathcal{L}_{\text{NI}}(\boldsymbol{x}, y, \lambda; \theta) = \mathcal{L}(\boldsymbol{x} + \lambda\boldsymbol{\epsilon}, y; \theta) \tag{57}$$

6 Proposed Regularization Methods

6.1 Motivation

In order to analyse the stability of the network against the perturbation, usually the Taylor expansion is utilized. Based on (57), we can perform the first order Taylor approximaiton as

$$\begin{aligned}\mathcal{L}_{\mathrm{NI}}(\boldsymbol{x}, y, \lambda; \theta) &= \mathcal{L}(\boldsymbol{x} + \lambda\boldsymbol{\epsilon}, y; \theta) \\ &\approx \mathcal{L}(\boldsymbol{x}, y; \theta) + \lambda \nabla_{\boldsymbol{x}} \mathcal{L}^\top \boldsymbol{\epsilon}.\end{aligned}$$

Then we define the approximation of the difference

$$\Delta \coloneqq \mathcal{L}_{\mathrm{NI}}(\boldsymbol{x}, y, \lambda; \theta) - \mathcal{L}(\boldsymbol{x}, y; \theta) \approx \lambda \nabla_{\boldsymbol{x}} \mathcal{L}^\top \boldsymbol{\epsilon}. \tag{58}$$

If the expectation of the squared difference is evaluated, we can derive the form

$$\begin{aligned}\mathbb{E}[\Delta^2] &= \lambda^2 \, \mathbb{E}[(\nabla_{\boldsymbol{x}} \mathcal{L}^\top \boldsymbol{\epsilon})^2] \\ &= \lambda^2 \left\| \nabla_{\boldsymbol{x}} \mathcal{L} \right\|_2^2\end{aligned}$$

which arrives the same formula as the double backpropagation. Based on the difference of the loss and the perturbed loss, the following observation is hypothesized.

Observation 1. *The noise injection lead to increase of the loss when the loss function is convex.*

Proof. Consider a neural network defined as $f(\boldsymbol{x}; \theta)$ where $\boldsymbol{x}$ is the input as a vector. The noise is sampled from the Gaussian distribution with variance $\lambda \boldsymbol{I}$ as $\epsilon \sim \mathcal{N}(\boldsymbol{0}, \lambda \boldsymbol{I})$. Next, the loss function is defined as $\mathcal{L}(\boldsymbol{x}; \theta) = M(f(x; \theta))$, for example, where M can be mean squared error or cros-entropy loss. Firstly, the output of the neural network with respect to the perturbed by the gaussian noise is estimated using the first order of the Taylor expansion.

$$f(\boldsymbol{x} + \boldsymbol{\epsilon}; \theta) \approx f(\boldsymbol{x}; \theta) + \nabla_{\boldsymbol{x}} f^\top \boldsymbol{\epsilon}. \tag{59}$$

Next, the second order of the Taylor's expansion on the loss function M is considered.

$$\begin{aligned}M(f(\boldsymbol{x} + \boldsymbol{\epsilon}; \theta)) \approx & M(f(\boldsymbol{x}; \theta) + \nabla_{\boldsymbol{x}} f^\top \boldsymbol{\epsilon}) \\ \approx & M(f(\boldsymbol{x}; \theta)) + \left(\frac{\partial M}{\partial f} \right)^\top (\nabla_{\boldsymbol{x}} f^\top \boldsymbol{\epsilon}) \\ & + \frac{1}{2} \boldsymbol{\epsilon}^\top \nabla_{\boldsymbol{x}} f \left(\frac{\partial^2 M}{\partial f^2} \right) \nabla_{\boldsymbol{x}} f^\top \boldsymbol{\epsilon}.\end{aligned} \tag{60}$$

Based on the estimation in (60), the expectation value can be described as the following,

$$\begin{aligned}\mathbb{E}[M(f(\boldsymbol{x}+\boldsymbol{\epsilon};\theta))] \approx& \mathbb{E}[M(f(\boldsymbol{x};\theta))] + \mathbb{E}\left[\left(\frac{\partial M}{\partial f}\right)^\top (\nabla_{\boldsymbol{x}} f^\top \boldsymbol{\epsilon})\right] \\ &+ \mathbb{E}\left[\frac{1}{2}\boldsymbol{\epsilon}^\top \nabla_{\boldsymbol{x}} f \left(\frac{\partial^2 M}{\partial f^2}\right) \nabla_{\boldsymbol{x}} f^\top \boldsymbol{\epsilon}\right] \\ \approx& M(f(\boldsymbol{x};\theta)) + \frac{\lambda^2}{2} \operatorname{Tr}\left(\nabla_{\boldsymbol{x}} f \left(\frac{\partial^2 M}{\partial f^2}\right) \nabla_{\boldsymbol{x}} f^\top\right). \end{aligned} \tag{61}$$

For most of the loss functions, they are convex; therefore, the hessian of the loss function with respect to the logits $\dfrac{\partial^2 M}{\partial f^2}$ is positive definite. If $\nabla_{\boldsymbol{x}} f$ is rewritten as a row vector

$$\nabla_{\boldsymbol{x}} f = \begin{bmatrix} -v_1- \\ -v_2- \\ \vdots \\ -v_n- \end{bmatrix}$$

it can be implied that

$$\operatorname{Tr}\left(\nabla_{\boldsymbol{x}} f \left(\frac{\partial^2 M}{\partial f^2}\right) \nabla_{\boldsymbol{x}} f^\top\right) = \sum_{i=1}^{n} v_i \left(\frac{\partial^2 M}{\partial f^2}\right) v_i^\top \geq 0.$$

Hence, for a sufficiently small perturbation by Gaussian noise, it will lead to a perturbed sample which causes increase in loss. This might imply that sampling noises from Gaussian distribution can penalize the loss function.

□

For examples,

- MSE loss - considering a regression task, the gradient $\nabla_{\boldsymbol{x}} f$ is a vector and $\dfrac{\partial^2 M}{\partial f^2} = 2$ so that the estimation of (61) will be

$$\mathbb{E}[\mathcal{L}(\boldsymbol{x}+\boldsymbol{\epsilon};\theta)] \approx M(f(\boldsymbol{x};\theta)) + \lambda^2 \left\|\nabla_{\boldsymbol{x}} f\right\|_2^2.$$

- Cross-entropy loss - Let the loss can be calculated as the following order.

$$M(f(\boldsymbol{x};\theta)) = \text{cross-entropy}(\text{softmax}(f(\boldsymbol{x};\theta))).$$

  The hessian $\dfrac{\partial^2 M}{\partial f^2}$ is a matrix with the following property,

$$H_{ij} \coloneqq \left(\frac{\partial^2 M}{\partial f^2}\right)_{ij} = \begin{cases} z_i - z_i^2 & \text{if } i = j \\ -z_i z_j & \text{otherwise} \end{cases},$$

  where $\boldsymbol{z}$ is the probability output of the softmax. It is useful to note that the hessian $\boldsymbol{H}$ is positive definite. Analogously, the estimation of (61) will be given by

$$\mathbb{E}[\mathcal{L}(\boldsymbol{x}+\boldsymbol{\epsilon};\theta)] \approx M(f(\boldsymbol{x};\theta)) + \frac{\lambda^2}{2} \operatorname{Tr}\left(\nabla_{\boldsymbol{x}} f \boldsymbol{H} \nabla_{\boldsymbol{x}} f^\top\right). \tag{62}$$

6.2 Proposed Methods

From (62) on the right-hand term that represents the expectation of the estimation of the difference between non-perturbed and perturbed loss, new penalizing methods can be purposed using that term directly and the list below is the variants of the regularizer.

- **Hessian-scaled Frobenius Regularizer**

$$\mathcal{L}_{\text{hsfob}}(\boldsymbol{x};\theta) = \mathcal{L}(\boldsymbol{x};\theta) + \lambda \operatorname{Tr}\left(\nabla_{\boldsymbol{x}} f \boldsymbol{H} \nabla_{\boldsymbol{x}} f^{\top}\right) \tag{63}$$

- **Hessian-scaled Frobenius Regularizer with Double Backpropagation**

$$\mathcal{L}_{\text{hsfob}-\text{db}}(\boldsymbol{x};\theta) = \mathcal{L}(\boldsymbol{x};\theta) + \lambda_1 \operatorname{Tr}\left(\nabla_{\boldsymbol{x}} f \boldsymbol{H} \nabla_{\boldsymbol{x}} f^{\top}\right) + \lambda_2 \left\|\nabla_{\boldsymbol{x}} \mathcal{L}\right\|_2^2 \tag{64}$$

For simplicity, it is assumed that $\lambda_1 = \lambda_2$. Furthermore, we propose 2 variants for (64) as

- **hsfob-db1**

$$\mathcal{L}_{\text{hsfob}-\text{db1}}(\boldsymbol{x};\theta) = \mathcal{L}(\boldsymbol{x};\theta) + \lambda \left[\operatorname{Tr}\left(\nabla_{\boldsymbol{x}} f \boldsymbol{H} \nabla_{\boldsymbol{x}} f^{\top}\right) + \left\|\nabla_{\boldsymbol{x}} \mathcal{L}\right\|_2^2\right] \tag{65}$$

- **hsfob-db2**

$$\mathcal{L}_{\text{hsfob}-\text{db2}}(\boldsymbol{x};\theta) = \mathcal{L}(\boldsymbol{x};\theta) + \lambda \left[\operatorname{Tr}\left(\nabla_{\boldsymbol{x}} f \boldsymbol{H} \nabla_{\boldsymbol{x}} f^{\top}\right)^2 + \left\|\nabla_{\boldsymbol{x}} \mathcal{L}\right\|_2^2\right] \tag{66}$$

7 Experimental Methods and Results

7.1 Network Architecture

For uniformity, we employ two neural network models in this experiment.

7.1.1 Small model

Small model is composed of 3 convolution layers and followed up with dense layers. All layers are connected linearly; the output of a layer is the input of the next layer. The structure of the model is as the following:

- **Convolution Layers**
  - 3×3 kernel, 32 filters, 1 stride, activated by ReLU and followed with 2×2 max-pooling layer.
  - 3×3 kernel, 64 filters, 1 stride, activated by ReLU and followed with 2×2 max-pooling layer.
  - 3×3 kernel, 64 filters, 1 stride, activated by ReLU and followed with flatten layer; it unfolds the tensor into vector.
- **Fully-connected layer**
  - Hidden layer with 64 nodes, activated by ReLU.
  - logits layer output 10 nodes, the number of classes, activated by softmax.

7.1.2 ResNet

ResNet [10] or, as known as, Residual Network is a neural network that introduces a non-linear architecture. The architecture is built on the block of "shortcut connection" which just performs an identity mapping and adds it to the output from the activation layer. It was claimed by the authors of the paper that the shortcut connection lets the a very deep neural network to enjoy a better training and validation result.

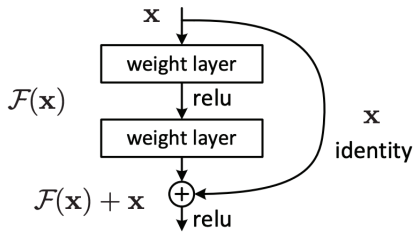


Figure 9: Residual Learning

In this experiment, ResNet is used for training on CIFAR-10 dataset and, based on shortcut connection, the architecture to be implemented is built as the following layers.

- **Convolution Layers**
  - Shortcut connection, 3×3 kernel, 16 filters, 1 stride, activated by ReLU and normalized by batch normalization.
  - Shortcut connection, 3×3 kernel, 32 filters, 2 stride, activated by ReLU and normalized by batch normalization.
  - Shortcut connection, 3×3 kernel, 64 filters, 2 stride, activated by ReLU and normalized by batch normalization.
  - 8×8 average-pooling layer and then flatten.
- **Fully-connected layer**
  - logits layer output 10 nodes, the number of classes, and activated by softmax.

7.2 Dataset

The experiment is conducted on the following 2 datasets.

- **MNIST**
  A dataset of handwritten digits from 0 to 9. There are 60,000 samples for the training set and 10,000 samples for the test set. The size of all images is 28×28 and the digit has been centered. In this experiment, the neural network will be trained on 50, 100, 200, 1,000 data points per class from the training set and test on all images from the test set.
- **CIFAR-10**
  A dataset of 10 object classes in color images. There are 50,000 samples for the training set and 10,000 samples for the test set. The size of all images is 32×32. In this experiment, the neural network will be trained on 50, 100, 200, 1,000 data points per class and all train data for ResNet, then test on all images from the test set

7.3 Training

For each model, it is trained with the learning rate 0.001 for 100 epochs. In addition, for each regularization method, the strength of the penalizing term is ranged as $\lambda \in [0.000001, 3.]$ by log-scale and the one that yields the best test result will be compared. The training will be done 10 times on different seeds which can be set in Numpy. This assures that the neural network weights are initialized and the training data is sampled exactly the same for each method. The loss used for this experiment is cross-entropy loss.

7.4 Results

The following results are the accuracy testing on the test set of each dataset by different methods. The value in the parenthesis is the standard variation of 10 trials. In addition, the methods under the horizontal line in each table are purposed methods and the baseline is the training without any regularizer.

Table 2: Recognition rate (%) on MNIST using small model

	50	100	200	1000
db	**92.273** (0.49)	95.010 (0.47)	96.725 (0.19)	98.378 (0.10)
jac	91.010 (0.44)	94.215 (0.31)	96.458 (0.17)	98.258 (0.09)
fob	90.828 (0.36)	94.274 (0.40)	96.433 (0.21)	98.311 (0.03)
NI	91.283 (0.64)	94.180 (0.33)	96.475 (0.24)	98.260 (0.08)
baseline	90.195 (0.45)	93.528 (0.48)	96.017 (0.28)	98.015 (0.14)
hsfob	90.881 (0.34)	94.106 (0.28)	96.479 (0.18)	98.256 (0.09)
hsfob-db1	92.239 (0.49)	**95.090** (0.49)	**96.862** (0.19)	**98.389** (0.13)
hsfob-db2	92.139 (0.34)	95.012 (0.41)	96.849 (0.20)	98.380 (0.08)

Table 3: Recognition rate (%) on CIFAR-10 using small model

	50	100	200	1000
db	39.761 (0.78)	46.467 (0.60)	52.352 (0.83)	65.644 (0.65)
jac	**39.976** (1.06)	46.107 (0.82)	52.348 (0.49)	63.257 (0.44)
fob	39.589 (1.04)	46.114 (0.74)	**52.426** (0.56)	63.268 (0.44)
NI	39.614 (1.15)	45.831 (0.55)	52.217 (0.77)	63.168 (0.51)
baseline	38.818 (1.44)	45.270 (0.83)	51.267 (0.80)	62.292 (0.56)
hsfob	39.684 (1.08)	46.361 (1.06)	52.255 (0.63)	63.648 (0.41)
hsfob-db1	39.781 (0.77)	46.342 (0.70)	52.401 (0.72)	66.031 (0.57)
hsfob-db2	39.773 (0.74)	**46.621** (0.65)	52.207 (0.79)	**66.208** (0.87)

Table 4: Recognition rate (%) on CIFAR-10 using ResNet

	50	100	200	1000
db	36.599 (0.87)	**42.156** (1.77)	**49.245** (1.43)	71.968 (0.86)
jac	36.063 (1.01)	40.040 (0.73)	45.720 (0.65)	64.128 (1.13)
fob	36.318 (0.73)	40.861 (1.35)	45.674 (0.78)	64.207 (1.13)
NI	35.504 (0.64)	39.302 (1.10)	45.837 (2.01)	68.628 (0.70)
baseline	31.451 (3.02)	31.992 (4.42)	39.295 (2.87)	64.893 (2.18)
hsfob	35.791 (0.47)	40.786 (0.61)	46.016 (0.25)	64.465 (1.22)
hsfob-db1	36.191 (0.86)	41.985 (1.07)	48.931 (1.44)	71.852 (1.20)
hsfob-db2	**37.021** (1.40)	42.089 (1.05)	47.908 (1.37)	**72.247** (0.67)

7.5 Discussion

For the experiment on the MNIST, all of the methods clearly perform outperform baseline. An interesting point is that in each experiment on different size of training data, the methods that include the term from the double backpropagation method enjoy better accuracy over other methods that do not include it. If the methods included double backpropagation terms are removed, it can be observed that the noise injection can achieve high accuracy while it enjoys less complexity compared to the other methods.

For the experiment on the CIFAR-10, starting from the small model results, the second variant of the hessian-scaled Frobenius regularizer with double backpropagation, as written as hsfob-db2, outperforms the hsfob-db1 in some experiment suggesting that both variants should be investigated. The methods involved with double backpropagation do not show any significant difference until the training size is increased to 1000 points per class.

In the case of training with ResNet, it can be clearly seen that methods involved with double backpropagation outperform the other methods. This suggests that the change of architecture can affect the result of each regularizer. Another interesting point is the experiment on 1000 data points per class in which the noise injection can clearly outperform many other methods with even lower variation, 0.70, of the accuracy. This implies that noise injection also is a potential regularizer that can raise the robustness of the model despite its simplicity.

8 Conclusion

8.1 Conclusion

The study focuses on comparing various methods of gradient regularization, both existed and proposed one, and the noise injection. The proposed methods derived from the observation of the difference occurred made by the perturbed input. The experiments show that the proposed methods can increase the robustness of the neural network even in the situation where data is scarce. The results also suggest that the methods included the term in the double backpropagation mostly yield better accuracy on the test set. However, it cannot be said clearly that from the methods we proposed, hsfob, hsfob-db1, and hdfob-db2, which one can perform the best because the results varying a lot on the experiment. Finally, even not the best, noise injection shows a noticeable performance in spite of its simplicity of implementation and computation.

8.2 Future Work

The future work can be more focused on the mathematical analysis of the effect due to the noise injection and gradient regularization or their relationship with other regularizations methods such as weight decay or dropout. For the experiment, not only the test accuracy is considered to determine the robustness of the neural network, but also it should be tested on the adversarial samples, for example, Fast Gradient Sign Method (FGSM) [8] and CW attack [3]. Because these methods of generating adversarial samples based on the gradients, it might be possible to withstand these attacks by reducing the magnitude of the gradients with respect to the input space.

Acknowledgement

I would like to express my appreciation to everyone who has supported me to finalized this thesis for the independent research project along with my study at the Tokyo Institute of Technology.

To begin with, I would like to express my gratitude to my laboratory advisor, Prof. Yukihiko Yamashita, who gives me an opportunity to conduct research and deepen my understanding of machine learning and image processing. Even I have knowledge about programming, the lecture "Applied programming and numerical analysis" that given by him still intrigued me with image processing which is surely going to be useful for my future researches or works. I also appreciate his kind support in this research project, including computer environment setup, technical assistant and mathematical correctness both in the project and seminars.

The next group of people who I would like to thank is my lab members, Mr. Haruki Ikenoue, Mr. Takuma Takezawa, Mr. Chawit Chaijirawiwat, Mr. Zhou Lijun who offer me supports in computer setup, technical problems, and miscellaneous knowledge.

Furthermore, I would like to thank the Tokyo Institute of Technology for the advance facilities, lectures and the opportunities allowing me to widen my perspective toward engineering fields, and special thanks to GSEP program which brings those amazing faculties and friends who always support me.

Finally, I would like to thank my family for allowing me to study abroad at the Tokyo Institute of Technology and consistently cheering me up when I feel lost in life.

References

[1] Chris M. Bishop. Training with noise is equivalent to tikhonov regularization. *Neural Computation*, 7(1):108–116, 1995.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer New York, 2016.

[3] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. *CoRR*, abs/1608.04644, 2016.

[4] Harris Drucker and Yann LeCun. Double backpropagation increasing generalization performance. In Anon, editor, *Proceedings. IJCNN - International Joint Conference on Neural Networks*, pages 145–150. Publ by IEEE, 1992.

[5] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7):2121–2159, 2011.

[6] Gamaleldin F. Elsayed, Dilip Krishnan, Hossein Mobahi, Kevin Regan, and Samy Bengio. Large margin deep networks for classification. In *Proceed-*

ings of the 32nd International Conference on Neural Information Processing Systems, NIPS'18, pages 850–860, Red Hook, NY, USA, 2018. Curran Associates Inc.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[8] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[9] Yves Grandvalet, Stéphane Canu, and Stéphane Boucheron. Noise injection: Theoretical prospects. *Neural Computation*, 9:1093–1108, 1997.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[12] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, December 2014.

[13] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

[14] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Netw.*, 12(1):145–151, January 1999.

[15] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[16] Dániel Varga, Adrián Csiszárik, and Zsolt Zombori. Gradient regularization improves accuracy of discriminative models. *CoRR*, abs/1712.09936, 2017.