UNIVERSITAS INDONESIA

LEAST SQUARE ADVERSARIAL AUTOENCODER DAN APLIKASINYA UNTUK REKONSTRUKSI DAN PEMBANGKITAN CITRA

TUGAS AKHIR

MARSHAL ARIJONA SINAGA
1606917525

FAKULTAS ILMU KOMPUTER
PROGRAM STUDI ILMU KOMPUTER
DEPOK
JULI 2020

UNIVERSITAS INDONESIA

LEAST SQUARE ADVERSARIAL AUTOENCODER DAN APLIKASINYA UNTUK REKONSTRUKSI DAN PEMBANGKITAN CITRA

TUGAS AKHIR
Diajukan sebagai salah satu syarat untuk memperoleh gelar
Sarjana Ilmu Komputer

MARSHAL ARIJONA SINAGA
1606917525

FAKULTAS ILMU KOMPUTER
PROGRAM STUDI ILMU KOMPUTER
DEPOK
JULI 2020

HALAMAN PERSETUJUAN

Judul : *Least Square Adversarial Autoencoder* dan Aplikasinya untuk Rekonstruksi dan Pembangkitan Citra
Nama : Marshal Arijona Sinaga
NPM : 1606917525

Laporan Tugas Akhir ini telah diperiksa dan disetujui.

21 Juli 2020

Lim Yohanes Stefanus, Ph.D.
Pembimbing Tugas Akhir

HALAMAN PERNYATAAN ORISINALITAS

Tugas Akhir ini adalah hasil karya saya sendiri,
dan semua sumber baik yang dikutip maupun dirujuk
telah saya nyatakan dengan benar.

Nama	:	**Marshal Arijona Sinaga**
NPM	:	**1606917525**
Tanda Tangan	:	
Tanggal	:	**21 Juli 2020**

HALAMAN PENGESAHAN

Tugas Akhir ini diajukan oleh	:	
Nama	:	Marshal Arijona Sinaga
NPM	:	1606917525
Program Studi	:	Ilmu Komputer
Judul Tugas Akhir	:	*Least Square Adversarial Autoencoder* dan Aplikasinya untuk Rekonstruksi dan Pembangkitan Citra

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Ilmu Komputer pada Program Studi Ilmu Komputer, Fakultas Ilmu Komputer, Universitas Indonesia.

DEWAN PENGUJI

Pembimbing	:	Lim Yohanes Stefanus, Ph.D.	( )
Penguji 1	:	Nilam Fitriah, S.Kom., M.Si.	( )
Penguji 2	:	Fariz Darari S.Kom, M.Sc., Ph.D	( )

Ditetapkan di	:	Depok
Tanggal	:	21 Juli 2020

KATA PENGANTAR

Segala puji dan syukur saya panjatkan kepada Tuhan Yesus Kristus karena kasih dan anugerah-Nya saya dapat menyelesaikan tugas akhir ini.

Penulisan tugas akhir ini ditujukan untuk memenuhi salah satu syarat untuk menyelesaikan pendidikan Program Sarjana Ilmu Komputer, Universitas Indonesia. Penyelesaian tugas akhir ini tidak terlepas dari dukungan beberapa pihak. Karenanya, saya ingin mengucapkan terimakasih kepada pihak - pihak berikut:

1. Bapak Lim Yohanes Stefanus, Ph.D. sebagai dosen pembimbing akademis dan dosen pembimbing tugas akhir saya yang selalu sabar membimbing saya dalam menyelesaikan tugas akhir ini.
2. Dosen penguji yang telah memberikan masukan untuk membuat skripsi ini menjadi lebih baik.
3. Orang tua dan adik saya yang memberi dukungan, semangat, dan doa kepada saya dalam menyelesaikan tugas akhir ini.
4. Teman - teman pengurus PO Fasilkom UI periode 2018 dan 2019 yang menjadi teman dalam melayani dan menjalani kepengurusan di PO Fasilkom.
5. Kelompok kecil saya selama menempuh pendidikan di Fasilkom UI.
6. Benny dan Frederic, teman saya dalam menertawakan kehidupan dan melepas penat.

Saya juga ingin mengucapkan terimakasih kepada setiap pihak yang tidak bisa disebutkan satu per satu yang turut membantu saya menyelesaikan tugas akhir ini. Akhir kata, saya berharap tugas akhir ini dapat memberi kontribusi pada perkembangan ilmu pengetahuan, khususnya di bidang jaringan saraf tiruan. *Soli Deo gloria*.

Depok, 21 Juli 2020

Marshal Arijona Sinaga

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini:

Nama	:	Marshal Arijona Sinaga
NPM	:	1606917525
Program Studi	:	Ilmu Komputer
Fakultas	:	Ilmu Komputer
Jenis Karya	:	Tugas Akhir

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif (Non-exclusive Royalty Free Right)** atas karya ilmiah saya yang berjudul:

Least Square Adversarial Autoencoder dan Aplikasinya untuk Rekonstruksi dan Pembangkitan Citra

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/formatkan, mengelola dalam bentuk pangkalan data (database), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyatan ini saya buat dengan sebenarnya.

Dibuat di : Depok
Pada tanggal : 21 Juli 2020
Yang menyatakan

(Marshal Arijona Sinaga)

ABSTRAK

Nama	:	Marshal Arijona Sinaga
Program Studi	:	Ilmu Komputer
Judul	:	*Least Square Adversarial Autoencoder* dan Aplikasinya untuk Rekonstruksi dan Pembangkitan Citra

Tugas Akhir ini menelaah *least square adversarial autoencoder* yang menggunakan *least square generative adversarial network* sebagai diskriminatornya. Diskriminator tersebut meminimalkan fungsi Pearson χ^2 *divergence* antara distribusi variabel laten dan suatu distribusi apriori. Adanya diskriminator memungkinkan *autoencoder* untuk membangkitkan data yang memiliki karakteristik yang menyerupai sampel pembelajarannya. Penelitian ini dilakukan dengan membuat program yang memodelkan *least square adversarial autoencoder*. Program memodelkan dua jenis *autoencoder* yaitu *unsupervised least square adversarial autoencoder* dan *supervised least square adversarial autoencoder* dengan memanfaatkan *dataset* MNIST dan FashionMNIST. *Unsupervised least square adversarial autoencoder* menggunakan variabel laten berdimensi 20 sementara *supervised least square adversarial autoencoder* menggunakan variabel laten masing-masing berdimensi 2, 3, 4, dan 5. Program diimplementasikan menggunakan *framework* PyTorch dan dieksekusi menggunakan Jupyter Notebook. Seluruh aktivitas pemrograman dilakukan pada *environment cloud* yang disediakan oleh Floydhub dan Tokopedia-UI AI Center yang masing-masing menggunakan GPU NVIDIA Tesla K80 dan NVIDIA Tesla V100 sebagai perangkat komputasinya. Proses pembelajaran pada *unsupervised least square adversarial autoencoder* berlangsung selama dua jam sementara pada *supervised least square adversarial autoencoder* berlangsung selama enam jam. Berdasarkan hasil eksperimen, nilai *mean squared error unsupervised least square adversarial autoencoder* untuk masing-masing *dataset* MNIST dan FashionMNIST adalah 0.0063 dan 0.0094. Sementara itu, nilai *mean squared error supervised least square adversarial autoencoder* pada *dataset* MNIST sebesar 0.0033. Selanjutnya, nilai Fréchet Inception Distance *unsupervised least square adversarial autoencoder* untuk masing-masing *dataset* MNIST dan FashionMNIST adalah 15.7182 dan 38.6967. Sementara itu, nilai Fréchet Inception Distance *supervised least square adversarial autoencoder* pada *dataset* MNIST sebesar 62.512. Hasil tersebut menunjukkan bahwa *least square adversarial autoencoder* mampu merekonstruksi citra dengan baik, namun kurang mampu membangkitkan citra dengan kualitas sebaik sampel pembelajarannya.

Kata kunci:
jaringan saraf tiruan, *autoencoder*, *least square generative adversarial network*, *regularisasi*, model pembangkit

ABSTRACT

Name : Marshal Arijona Sinaga
Program : Ilmu Komputer

Title : Least Square Adversarial Autoencoder and Its Application for Image Reconstruction and Image Generation

This Final Project (Tugas Akhir) investigates the least square adversarial autoencoder that uses least square generative adversarial network as its discriminator. The discriminator minimizes the Pearson χ^2 divergence between the latent variable distribution and the prior distribution. The presence of discriminator allows the autoencoder to generate data that has characteristics that resemble the original data. Python programs were developed to model the least square adversarial autoencoder. This programs try to model two types of autoencoder namely unsupervised least square adversarial autoencoder and supervised least square adversarial autoencoder by utilizing MNIST dataset and FashionMNIST dataset. The unsupervised least square adversarial autoencoder uses latent variables of dimension 20 while the supervised least square adversarial autoencoder uses latent variables with dimensions of 2, 3, 4, and 5, respectively. This programs were implemented using PyTorch and executed using Jupyter Notebook. All of the programming activities are carried out in the cloud environment provided by Floydhub and Tokopedia-UI AI Center, respectively using NVIDIA Tesla K80 GPU and NVIDIA Tesla V100 GPU as their computing resource. Training time in unsupervised least square adversarial autoencoder lasts for two hours while in supervised least square adversarial autoencoder lasts for six hours. The Results of experiments show that the mean squared error of unsupervised least square adversarial autoencoder for MNIST dataset and FashionMNIST dataset are 0.0063 and 0.0094, respectively. Meanwhile, the mean squared error of supervised least square adversarial autoencoder for MNIST dataset is 0.0033. Furthermore, the Fréchet Inception Distance scores of unsupervised least square adversarial autoencoder for MNIST dataset and FashionMNIST dataset are 15.7182 and 38.6967, respectively. Meanwhile, the value of Fréchet Inception Distance score of supervised least square adversarial autoencoder in MNIST dataset is 62.512. These results indicate that the least square adversarial autoencoder is able to reconstruct the image properly, but is less able to generate images with the same quality as the learning sample.

Key words:
artificial neural network, autoencoder, least square generative adversarial network, regularization, generative model

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE PROGRAM

BAB 1
PENDAHULUAN

Bab ini berisi latar belakang, permasalahan, tujuan, tahapan, dan sistematika penulisan dari Tugas Akhir yang dilakukan.

1.1 Latar Belakang

Pembelajaran representasi merupakan salah satu topik mendasar yang belum terselesaikan sepenuhnya di bidang kecerdasan buatan. Pembelajaran representasi bertujuan untuk memodelkan distribusi (representasi) dari suatu kelompok data. Hal ini didorong oleh kebutuhan akan data yang terstruktur ketika akan melakukan pembelajaran pada *machine learning*. Kebanyakan model *machine learning* yang ada memanfaatkan data yang diolah secara manual oleh manusia (*feature engineering*). Penelitian terhadap pembelajaran representasi berusaha untuk melakukan otomatisasi terhadap agen kecerdasan buatan untuk dapat mengekstraksi fitur utama yang dapat digunakan sebagai fitur untuk suatu model prediktor [Bengio et al., 2013]. Beberapa model yang dikembangkan untuk menyelesaikan permasalahan pembelajaran representasi diantaranya *restricted Boltzman machine* [Hinton and Salakhutdinov, 2006], *autoencoder* [Bengio et al., 2013], dan *principal component analysis* [Addi and Williams, 2010].

Autoencoder merupakan salah satu model yang paling banyak digunakan untuk menyelesaikan permasalahan pembelajaran representasi. *Autoencoder* memetakan data ke dalam suatu representasi yang disebut sebagai variabel laten. Variabel laten mampu mengestimasi distribusi data berdasarkan sampel pembelajaran yang diberikan. Variabel laten yang didapat kemudian dipergunakan untuk merekonstruksi sampel data yang asli. Kemampuan rekonstruksi *autoencoder* dimanfaatkan diantaranya untuk reduksi dimensi [Goodfellow et al., 2016], aplikasi pada perolehan informasi [Salakhutdinov and Hinton, 2009], restorasi citra [Mao et al., 2016] dan penerjemahan teks antar bahasa [Cho et al., 2014]. Namun, kemampuan *autoencoder* biasa masih belum mampu menyelesaikan permasalahan pembangkitan data.

Pembangkitan data dibutuhkan untuk menghasilkan data sintetis yang memiliki karakteristik yang menyerupai data asli. Pembangkitan data terutama berguna untuk data yang jumlahnya terbatas. Data sintetis dapat dimanfaatkan untuk keperluan penelitian. Salah satu contoh pengaplikasan pembangkitan data adalah pembangkitan citra medis dengan memanfaakan *generative adversarial autoencoder* [Shin et al., 2018]. Citra

sintetis tersebut dimanfaatkan untuk melakukan penelitian dalam mendeteksi tumor otak. Citra sintetis dimanfaatkan karena terbatasnya jumlah citra asli yang tersedia. Aplikasi lainnya adalah di bidang pembangkitan teks [Subramanian et al., 2018, Xie, 2017].

Autoencoder mampu mengestimasi distribusi variabel laten dari sampel pembelajaran tetapi distribusi tersebut tidak dapat diatur dan diadaptasi untuk dimanfaatkan sebagai model pembangkit. Model pembangkit seharusnya memiliki kemampuan untuk menghasilkan data yang bermakna dari suatu distribusi apriori. Berangkat dari permasalahan tersebut, beberapa penelitian dilakukan sehingga *autoencoder* dapat berperan sebagai model pembangkit. Beberapa penelitian yang paling terkenal diantaranya *variational autoencoder* [Kingma and Welling, 2013] dan *adversarial autoencoder* [Makhzani et al., 2015] yang memanfaatkan *generative adversarial network* (GAN). Jenis *autoencoder* ini dapat dimanfaatkan untuk merekonstruksi suatu data sekaligus juga membangkitkan data yang mirip dengan sampel pembelajarannya. Salah satu aplikasinya adalah di bidang *fraud detection* pada kartu ATM. Kemampuan rekonstruksi *autoencoder* dapat digunakan untuk mendeteksi *fraud* pada kartu ATM. Sementara itu, kemampuan untuk membangkitkan data dapat digunakan untuk menghasilkan sampel kartu ATM yang cacat. Sampel ini kemudian dapat dimanfaatkan sebagai sampel pembelajaran untuk model *machine learning*.

Adversarial autoencoder merupakan *autoencoder* yang memiliki kemampuan untuk melakukan pembangkitan data dari distribusi apriori. Kemampuan tersebut didapat dengan memanfaatkan *generative adversarial network*. *Generative adversarial network* merupakan suatu model yang mampu menghasilkan data yang bermakna dari suatu distribusi apriori. *Generative adversarial network* memiliki banyak varian, salah satunya adalah *least square generative adversarial network* (LSGAN)[Mao et al., 2017]. Pemilihan LSGAN didasari oleh dua sifat LSGAN yang lebih baik dibandingkan *vanilla* GAN (varian GAN yang paling awal). Sifat yang pertama, LSGAN mampu membangkitkan citra yang cenderung lebih mirip dengan sampel pembelajarannya dibandingkan dengan *vanilla* GAN [Mao et al., 2017]. Sifat yang kedua, proses pembelajaran LSGAN lebih stabil dibandingkan dengan *vanilla* GAN [Mao et al., 2017]. Penelitian mengenai *adversarial autoencoder* saat ini hanya menggunakan *vanilla* GAN [Goodfellow et al., 2014] dan *Wasserstein* GAN [Arjovsky et al., 2017].

Berdasarkan uraian di atas, penelitian Tugas Akhir ini dilakukan untuk menelaah bagaimana cara menggabungkan LSGAN dan *autoencoder* (selanjutnya disebut sebagai *least square adversarial autoencoder*), mengimplementasikan *least square adversarial autoencoder*, serta mengetahui kualitas rekonstruksi citra dan kualitas pembangkitan citra oleh *least square adversarial autoencoder*. Penelitian dilakukan dengan mengembangkan suatu program Python yang memodelkan *least square adversarial autoencoder* dengan

memanfaatkan data berupa citra.

1.2 Permasalahan

Bagian ini berisi definisi permasalahan yang ingin diteliti dan batasan-batasan yang ditetapkan untuk menyelesaikan permasalahan tersebut.

1.2.1 Definisi Permasalahan

Berikut ini adalah rumusan permasalahan dari penelitian yang dilakukan:

- Bagaimana menggabungkan *least square generative adversarial network* dan *adversarial autoencoder* menjadi *least square adversarial autoencoder*?
- Bagaimana mengimplementasikan *least square adversarial autoencoder* ?
- Bagaimana kualitas hasil rekonstruksi citra menggunakan *least square adversarial autoencoder* ?
- Bagaimana kualitas citra yang dibangkitkan dari hasil *sampling* variabel laten menggunakan *least square adversarial autoencoder* ?

1.2.2 Batasan Permasalahan

Berikut ini adalah asumsi yang digunakan sebagai batasan penelitian ini:

- Penelitian ini menyelidiki model *unsupervised least square adversarial autoencoder* dan *supervised least square adversarial autoencoder*.
- *Dataset benchmark* yang digunakan adalah MNIST dan FashionMNIST yang merupakan citra *grayscale*.
- Arsitektur *encoder* dan *decoder* pada *unsupervised least square adversarial autoencoder* diimplementasi menggunakan jaringan saraf konvolusional sementara arsitektur *encoder* dan *decoder* pada *supervised least square adversarial autoencoder* diimplementasi menggunakan arsitektur *fully connected*.
- Arsitektur diskriminator pada *unsupervised least square adversarial autoencoder* dan *supervised least square adversarial autoencoder* diimplementasi menggunakan arsitektur *fully connected*.
- Fungsi objektif pada *decoder*, *autoencoder*, dan diskriminator menggunakan *mean squared error*.

1.3 Tujuan Penelitian

Penelitian ini bertujuan untuk mempelajari sifat-sifat dan mengimplementasikan *least square adversarial autoencoder* dalam melakukan rekonstruksi citra dan pembangkitan citra dalam kerangka pembelajaran representasi untuk menunjang sistem *machine learning* yang handal. *Novelty* dari penelitian ini adalah menggabungkan *adversarial autoencoder* dan *least square generative adversarial network*.

1.4 Tahapan Penelitian

Berikut ini adalah langkah-langkah penelitian yang telah dilakukan:

1. **Studi literatur**
   Pada tahap ini, dilakukan kajian literatur mengenai konsep jaringan saraf tiruan [Goodfellow et al., 2016, Shalev-Shwartz and Ben-David, 2014, Liu, 2019], *stochastic gradient descent* [Liu, 2019], regularisasi [Goodfellow et al., 2016], jaringan saraf konvolusional [Goodfellow et al., 2016], *autoencoder* [Shalev-Shwartz and Ben-David, 2014, Goodfellow et al., 2016], dan *generative adversarial network* [Goodfellow et al., 2014, Langr and Bok, 2019] yang menjadi landasan teori pada penelitian ini. Selain itu, dilakukan pula kajian mengenai PyTorch [PyTorch, 2020] yang digunakan sebagai *framework* dalam penelitian ini. Sumber literatur berasal dari buku, jurnal penelitian, dan internet.

2. **Pengumpulan *Dataset***
   Pada tahap ini, dilakukan pengumpulan *dataset* yang digunakan pada eksperimen. *Dataset* yang dipergunakan yaitu MNIST [LeCun and Cortes, 2020] dan FashionMNIST [Xiao et al., 2017]. Kedua *dataset* tersebut diakses menggunakan *PyTorch*. Selanjutnya, dilakukan *preprocessing* pada *dataset* yang digunakan. *Dataset* MNIST dan FashionMNIST masing-masing berukuran 11.59 MegaByte dalam kondisi terkompres.

3. **Perancangan Model dan Algoritma Pembelajaran**
   Pada tahap ini, dilakukan perancangan model *unsupervised least square adversarial autoencoder* dan model *unsupervised least square adversarial autoencoder*. Selanjutnya, dilakukan perancangan algoritma pembelajaran pada kedua model tersebut.

4. **Implementasi Model dan Algoritma**
   Dilakukan implementasi terhadap rancangan model dan algoritma. Implementasi dilakukan dengan menggunakan *framework* PyTorch. Model yang merupakan hasil

implementasi kemudian dilatih menggunakan algoritma pembelajaran yang sudah diimplementasi. Proses pembelajaran dilakukan terhadap *dataset* yang didapat pada tahap kedua.

5. **Percobaan terhadap Model dan Analisis Hasil Percobaan**
   Dilakukan pengujian terhadap model yang sudah melewati proses pembelajaran. Selanjutnya, dilakukan pengolahan dan penyajian terhadap data hasil uji model. Data hasil uji kemudian dianalisis untuk memperoleh kesimpulan penelitian.

1.5 Sistematika Penulisan

Sistematika penulisan laporan adalah sebagai berikut:

- Bab 1 PENDAHULUAN
  Bab ini berisi latar belakang, tujuan dan manfaat, tahapan, ruang lingkup, dan sistematika penulisan laporan dari penelitian yang dilakukan.

- Bab 2 JARINGAN SARAF TIRUAN DAN REGULARISASI PEMBELAJARAN
  Bab ini membahas mengenai landasan teori mengenai jaringan saraf tiruan. Landasan teori mencakup konsep jaringan saraf tiruan secara umum, *gradient descent*, regularisasi, dan jaringan saraf konvolusional.

- Bab 3 LEAST SQUARE GENERATIVE ADVERSARIAL NETWORK
  Bab ini membahas konsep dan teori mengenai *generative adversarial network* dan *least square generative adversarial network*.

- Bab 4 LEAST SQUARE ADVERSARIAL AUTOENCODER
  Bab ini berisi penjelasan mengenai *autoencoder*, *adversarial autoencoder*, *least square adversarial autoencoder*, dan standar pengukuran pada *least square adversarial autoencoder*.

- Bab 5 EKSPERIMEN DAN ANALISIS HASIL EKSPERIMEN
  Bab ini berisi deskripsi teknologi yang digunakan, deskripsi *dataset* yang digunakan, hasil eksperimen, dan analisis hasil eksperimen.

- Bab 6 KESIMPULAN DAN SARAN
  Bab ini mencakup kesimpulan akhir penelitian dan saran untuk pengembangan berikutnya.

BAB 2
JARINGAN SARAF TIRUAN DAN REGULARISASI PEMBELAJARAN

Bab ini menjelaskan teori-teori yang diperlukan dalam memahami jaringan saraf tiruan yang meliputi model jaringan saraf tiruan, *gradient descent*, regularisasi, dan jaringan saraf konvolusi.

2.1 Model Jaringan Saraf Tiruan

Jaringan saraf tiruan merupakan salah satu model *machine learning* yang memanfaatkan model *perceptron* (selanjutnya ditulis sebagai perseptron) yang terinspirasi oleh model jaringan saraf pada manusia [Chen et al., 2019]. Jaringan saraf tiruan mampu melakukan tugas yang spesifik berdasarkan sampel pembelajaran yang diberikan. Proses pembelajaran pada jaringan saraf tiruan meminimalkan suatu fungsi objektif (dapat pula disebut sebagai *error function* atau *cost function*) yang digunakan untuk mengukur performa jaringan saraf tiruan [Russel et al., 2013]. Model jaringan saraf tiruan yang telah melewati proses pembelajaran dapat digunakan untuk mengaproksimasi target dari *test set*. *Test set* merupakan sampel data yang tidak digunakan pada proses pembelajaran. Performa jaringan saraf tiruan terhadap *test set* dapat diukur sebagai *generalization error* [Data et al., 2012]. Pada bagian ini dijelaskan mengenai perseptron dan arsitektur *feedforward neural network* yang merupakan arsitektur pada jaringan saraf tiruan.

2.1.1 Perseptron

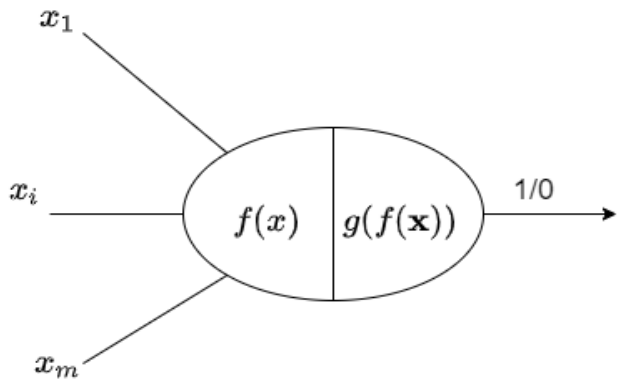
Perseptron merupakan model yang digunakan pada klasifikasi biner. Klasifikasi biner merupakan sebuah fungsi yang menentukan apakah suatu masukan (direpresentasikan dalam vektor) tergolong dalam sebuah kelas atau tidak. Model perseptron pertama kali diperkenalkan oleh McCulloch dan Pitts pada tahun 1943 [Piccinini, 2004]. Model ini kemudian dikembangkan oleh Frank Rosenblatt pada tahun 1958 dan Minsky dan Papert pada tahun 1969 [Rosenblatt, 1958, Minsky and Papert, 1969].

Perseptron terdiri dari dua buah fungsi yaitu $f(\boldsymbol{x})$ dan $g(f(\boldsymbol{x}))$. Fungsi $f(x)$ dan $g(x)$ pada perseptron dapat dituliskan sebagai:

$$f(\mathbf{x}) = \sum_{i=0}^{m} (w_i x_i) + \mathbf{b} \tag{2.1}$$

$$g(f(\mathbf{x})) = \begin{cases} 1 & \text{jika } f(\mathbf{x}) > 0 \\ 0 & \text{jika } f(\mathbf{x}) \leq 0 \end{cases} \tag{2.2}$$

Notasi $\boldsymbol{w}$ menyatakan vektor *weight* yang menjadi salah satu operan pada operasi dot terhadap vektor masukan $\boldsymbol{x}$ yang berdimensi m. Notasi b menyatakan bias yaitu suatu variabel yang dijumlahkan dengan hasil operasi dot $\boldsymbol{w}.\boldsymbol{x}$. **Gambar 2.1** menunjukkan model perseptron yang dikembangkan oleh McCulloch-Pitt [Piccinini, 2004].



Gambar 2.1: Model Perseptron yang Dikembangkan oleh McCulloch-Pitt [Piccinini, 2004]

Fungsi $g(\boldsymbol{x})$ disebut juga sebagai fungsi aktivasi. Fungsi aktivasi menentukan apakah suatu masukan $\boldsymbol{x}$ tergolong dalam suatu kelas atau tidak. Nilai satu menyatakan bahwa masukan tersebut tergolong dalam suatu kelas, sementara nilai nol menunjukkan hal sebaliknya. Penelitian mengenai perseptron kemudian berkembang dan mendasari penelitian mengenai jaringan saraf tiruan. Penelitian dilakukan untuk mengembangkan fungsi aktivasi agar tidak terbatas pada fungsi biner saja. Saat ini, terdapat beberapa fungsi aktivasi selain fungsi biner yang berupa fungsi nonlinear, diantaranya fungsi ReLU, fungsi Sigmoid, dan fungsi Tanh [Nwankpa et al., 2018].

2.1.2 *Feedforward Neural Network*

Bagian ini menjelaskan arsitektur jaringan perseptron satu lapis dan arsitektur jaringan perseptron berlapis banyak yang merupakan bagian dari *feedforward neural network*, serta algoritma *feedforward propagation* dan *back propagation* yang merupakan algoritma yang digunakan pada pembelajaran *feedforward neural network*.

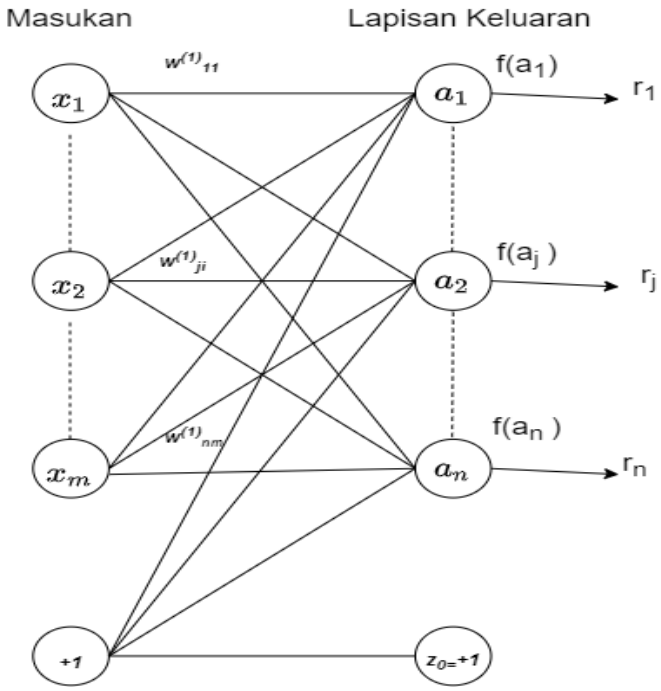
2.1.2.1 Jaringan Perseptron Satu Lapis

Jaringan perseptron satu lapis merupakan arsitektur jaringan saraf tiruan yang memiliki sebuah lapisan yang terdiri dari sekumpulan unit perseptron. Lapisan ini disebut juga sebagai lapisan keluaran. Jaringan perseptron satu lapis terdiri dari masukan dan lapisan keluaran. Masukan yang berupa vektor $\boldsymbol{x}$ diteruskan menuju lapisan keluaran untuk dilakukan operasi dot dengan *weight* $\boldsymbol{w}$. *Weight* $\boldsymbol{w}$ merupakan koneksi yang menghubungkan perseptron dan komponen dari vektor masukan. Selain itu, terdapat bias b yang dijumlahkan dengan hasil operasi dot $\boldsymbol{w}.\boldsymbol{x}$. Penggunaan bias bersifat opsional sehingga bisa saja tidak digunakan. Hasil penjumlahan operasi dot dan bias (opsional) disebut sebagai aktivasi. Nilai aktivasi kemudian menjadi masukan bagi fungsi aktivasi. Hasil dari perhitungan fungsi aktivasi menghasilkan keluaran. Proses perhitungan masukan pada model jaringan perseptron satu lapis dapat diformulasikan sebagai:

$$a_j = \sum_{i=1}^{m} (w_{ji} x_i) + b \tag{2.3}$$

$$r_j = f(a_j) \tag{2.4}$$

Notasi x_i menyatakan komponen ke-i dari vektor masukan. Notasi w_{ji} menyatakan koneksi *weight* antara perseptron ke-j dan komponen ke-i dari vektor masukan [Abbod et al., 2007]. Notasi b menyatakan bias [Liu, 2019]. Selanjutnya, notasi a_j menyatakan nilai aktivasi dan r_j menyatakan nilai hasil perhitungan fungsi aktivasi. **Gambar 2.2** menunjukkan arsitektur jaringan perseptron satu lapis.



Gambar 2.2: Arsitektur Jaringan Perseptron Satu Lapis.

Proses pembelajaran pada jaringan saraf tiruan satu lapis adalah dengan memperbarui nilai parameter w_{ji}. Misalkan t_j merupakan nilai yang benar untuk komponen keluaran ke-j dari vektor keluaran dan r_j merupakan hasil prediksi jaringan. Nilai fungsi objektif antara kedua nilai tersebut dapat dituliskan sebagai:

$$\Delta r_j = r_j - t_j \tag{2.5}$$

Jika nilai r_j sama dengan nilai t_j, maka nilai fungsi objektif bernilai nol, artinya tidak diperlukan pembaruan terhadap parameter w_{ji}. Jika nilai fungsi objektif lebih besar dari nol, koneksi *weight* w_{ji} diperbarui untuk meminimalkan fungsi objektif [Liu, 2019]. Besarnya nilai untuk memperbarui w_{ji} dapat dituliskan sebagai:

$$\Delta w_{ji} = -\varepsilon \Delta r_j = -\varepsilon(r_j - t_j) \tag{2.6}$$

dengan ε menyatakan *learning rate* yang berfungsi mengatur skala perubahan.

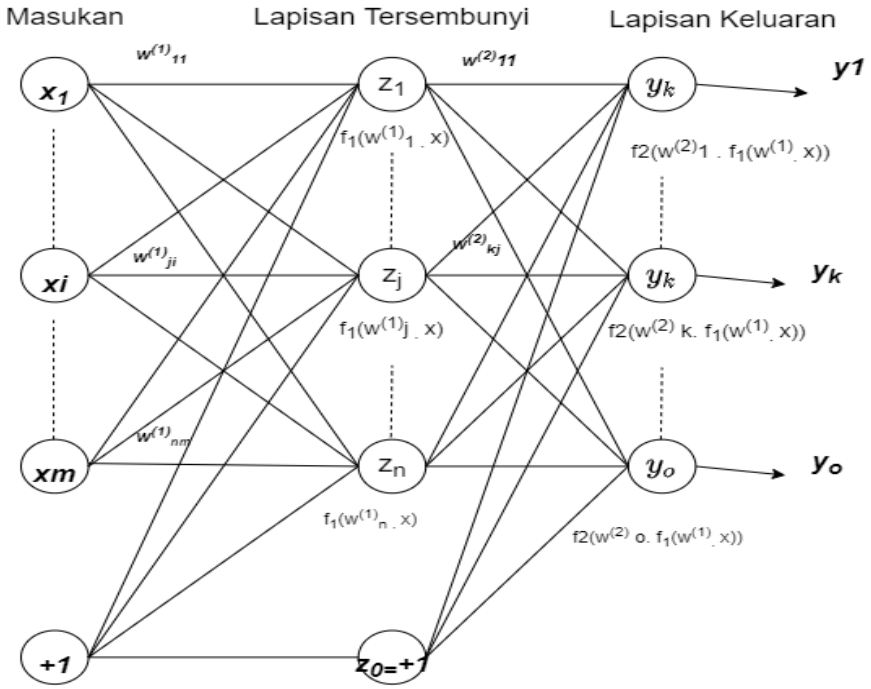
2.1.2.2 Jaringan Perseptron Berlapis Banyak

Jaringan perseptron berlapis banyak (JPBB) merupakan jaringan saraf tiruan yang memiliki lebih dari satu lapisan perseptron. Jaringan perseptron berlapis banyak terdiri dari tiga bagian yaitu masukan, lapisan tersembunyi, dan lapisan keluaran, dengan lapisan tersembunyi dapat berjumlah lebih dari satu [Liu, 2019]. Terdapat dua koneksi *weight*

pada model JPBB yaitu [Liu, 2019]:

- $w_{ji}^{(1)}$ yang menghubungkan antara komponen vektor masukan x_i dan salah satu perseptron z_j pada lapisan tersembunyi.
- $w_{kj}^{(2)}$ yang menghubungkan antara salah satu perseptron z_j pada lapisan tersembunyi dan salah satu perseptron y_k pada lapisan keluaran.

Gambar 2.3 menunjukkan arsitektur JPBB dengan satu lapisan tersembunyi.



Gambar 2.3: Arsitektur Jaringan Perseptron Berlapis Banyak.

Proses meneruskan vektor masukan $\boldsymbol{x}$ pada JPBB dapat dituliskan sebagai:

$$y_k(\mathbf{x},\mathbf{w}) = f_2(\sum_{j=1}^{n} w_{kj}^{(2)} f_1[\sum_{i=1}^{m} w_{ji}^{(1)} x_i + b_1] + b_2) \tag{2.7}$$

Notasi f_1 dan f_2 menyatakan fungsi aktivasi yang sifatnya non linear dan *differentiable*. Notasi b_1 dan b_2 menyatakan bias pada lapisan tersembunyai dan bias pada lapisan keluaran.

2.1.2.3 *Forward Propagation* dan *Back Propagation*

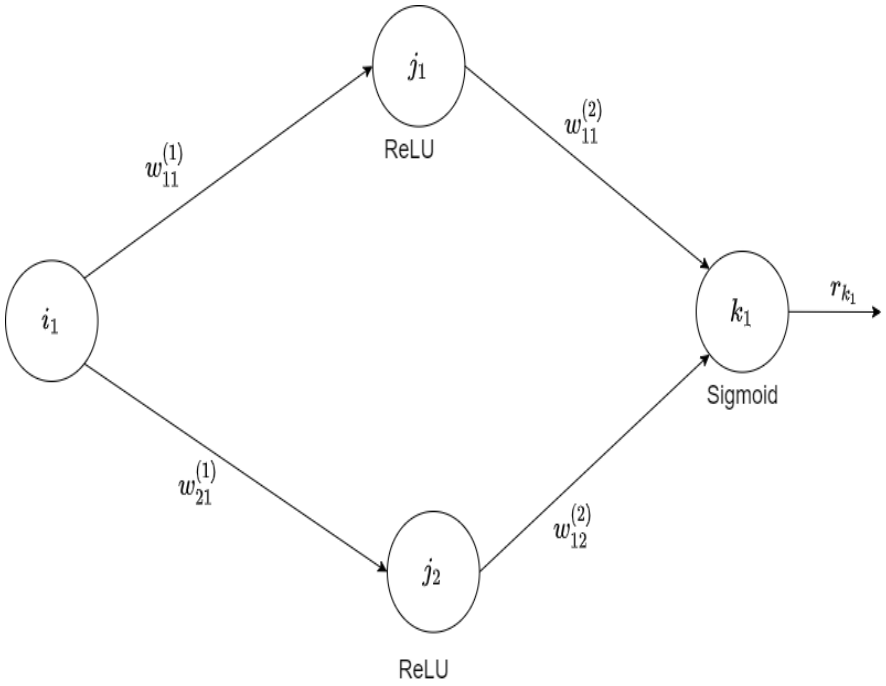
Proses pembelajaran pada jaringan saraf tiruan terdiri dari dua tahapan. Tahapan yang pertama disebut sebagai *forward propagation*. Pada tahapan ini, sampel data dilewatkan terhadap lapisan berisi perseptron pada jaringan saraf tiruan. Proses ini

melibatkan perhitungan sampel data terhadap parameter berupa *weight* pada jaringan saraf tiruan[Liu, 2019]. Hasil keluaran dari proses *feedforward propagation* disebut sebagai nilai prediksi. Nilai prediksi kemudian dibandingkan dengan target dari sampel data menggunakan sebuah fungsi yang disebut sebagai fungsi objektif. Fungsi objektif digunakan untuk mengukur performa jaringan saraf tiruan dalam memprediksi nilai target dari sampel data [Liu, 2019].

Tahap kedua dari proses pembelajaran pada jaringan saraf tiruan disebut sebagai *back propagation*. *Back propagation* bertujuan untuk memperbarui nilai *weight* yang merupakan parameter pada jaringan saraf tiruan berdasarkan nilai dari fungsi objektif yang didapat [Liu, 2019]. Proses ini bertujuan untuk meminimalkan fungsi objektif. Langkah-langkah *back propagation* adalah sebagai berikut:

- Hitung gradien fungsi objektif terhadap semua parameter yang ada dengan mencari turunan parsial dari fungsi tersebut. Turunan parsial dapat dihitung menggunakan aturan rantai.
- Lakukan pembaruan terhadap semua nilai parameter (*weight* dan bias) berdasarkan nilai gradien yang didapat. Semua parameter diperbarui menggunakan *gradient descent*. Pembahasan mengenai *gradient descent* dijelaskan pada subbab selanjutnya.

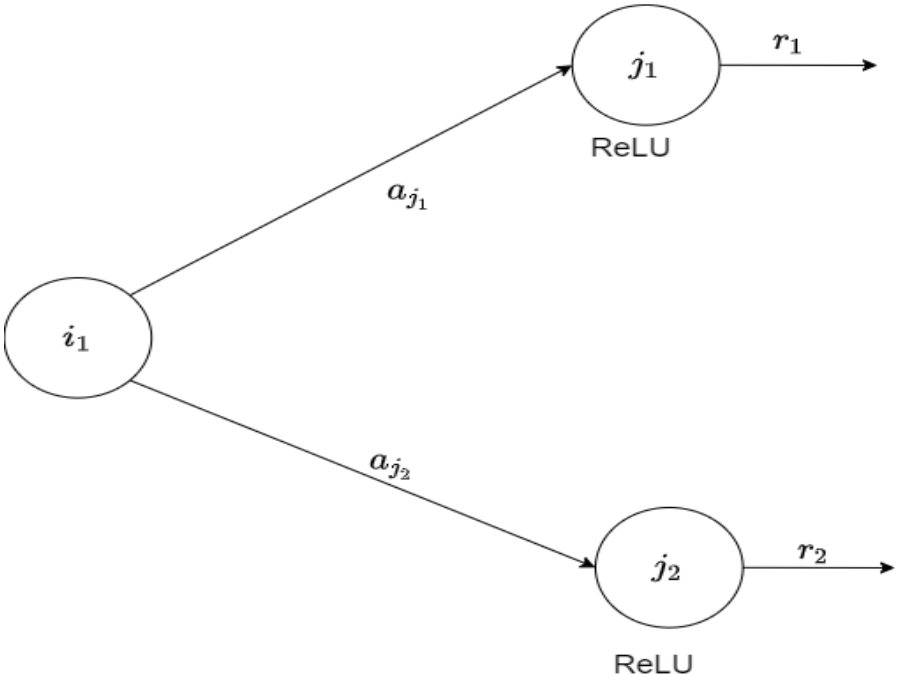
Misalkan terdapat suatu jaringan saraf tiruan yang terdiri dari satu lapisan tersembunyi dan satu lapisan keluaran. Lapisan tersembunyi menggunakan fungsi ReLU sebagai fungsi aktivasi, sedangkan lapisan keluaran menggunakan fungsi Sigmoid. **Gambar 2.4** menunjukkan model jaringan saraf tersebut.



Gambar 2.4: Contoh Jaringan Saraf Tiruan yang Terdiri dari Dua Lapisan Perseptron.

Berdasarkan gambar tersebut, terdapat dua *weight* antara masukan dan lapisan tersembunyi ($w_{11}^{(1)}$ dan $w_{21}^{(1)}$), begitu pula dengan jumlah *weight* antara lapisan tersembunyi dan lapisan keluaran ($w_{11}^{(2)}$ dan $w_{12}^{(2)}$). Berdasarkan jumlah tersebut, terdapat empat parameter yang harus diperbarui untuk meminimalkan fungsi objektif. Selanjutnya, dilakukan *forward propagation* dan *back propagation* terhadap jaringan tersebut menggunakan suatu sampel data.

Fordward propagation berdasarkan jaringan saraf tiruan yang ditunjukkan oleh **Gambar 2.4** melewati dua lapisan. Pertama, data diteruskan melalui lapisan tersembunyi. **Gambar 2.5** menunjukkan *forward propagation* antara masukan dan lapisan tersembunyi.



Gambar 2.5: *Forward Propagation* antara Masukan dan Lapisan Tersembunyi.

Ketika melalui lapisan tersembunyi terjadi operasi dot antara komponen masukan i_1 dan *weight* yang menghubungkan komponen masukan i_1 dan perseptron pada lapisan tersembunyi (j_1 dan j_2). Hasil perhitungan tersebut disebut sebagai aktivasi a_{j_1} dan a_{j_2}. Secara matematis, proses tersebut dapat dituliskan sebagai:

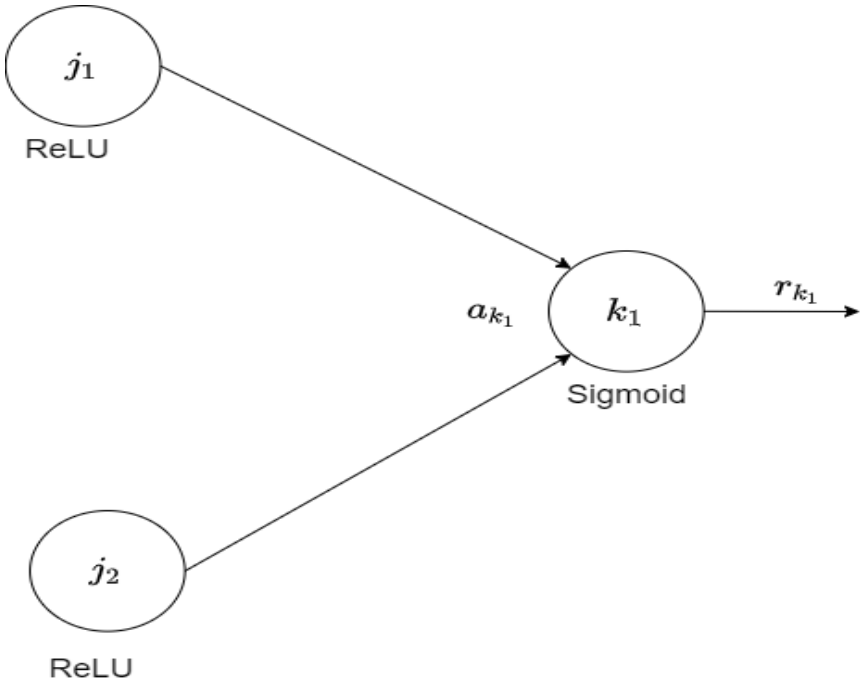
$$a_{j_1} = \sum_{m=1}^{1} w_{1m}^{(1)} i_m$$
$$a_{j_2} = \sum_{m=1}^{1} w_{2m}^{(1)} i_m.$$

Nilai aktivasi tersebut kemudian menjadi masukan bagi fungsi aktivasi. Berdasarkan **Gambar 2.4**, fungsi aktivasi yang digunakan adalah ReLU ($f(x) = \max(x, 0)$). Secara matematis, perhitungan fungsi aktivasi dapat dituliskan sebagai:

$$r_{j_1} = \max(0, a_{j_1})$$
$$r_{j_2} = \max(0, a_{j_2}).$$

Selanjutnya, data diteruskan dari lapisan tersembunyi menuju lapisan keluaran. Ketika melalui lapisan keluaran terjadi operasi dot antara hasil perhitungan pada lapisan tersembunyi (r_{j_1} dan r_{j_2}) dan *weight* yang menghubungkan perseptron pada lapisan tersembunyi dan perseptron pada lapisan keluaran. Hasil perhitungan tersebut disebut sebagai aktivasi a_{k_1}. **Gambar 2.6** menunjukkan *forward propagation* antara lapisan

tersembunyi dan lapisan keluaran.



Gambar 2.6: *Forward Propagation* antara Lapisan Tersembunyi dan Lapisan Keluaran.

Perhitungan a_{k_1} pada lapisan keluaran dapat dituliskan sebagai:

$$a_{k_1} = \sum_{n=1}^{2} w_{1n}^{(2)} j_n$$

Selanjutnya, nilai keluaran dari fungsi aktivasi r_{k_1} dapat dituliskan sebagai:

$$Sigmoid \longrightarrow f(x) = \frac{1}{1+e^{-x}}$$
$$r_{k_1} = \frac{1}{1+e^{-a_{k_1}}}$$

Nilai r_{k_1} merupakan hasil prediksi dari jaringan saraf tiruan tersebut untuk suatu masukan i_1. Selanjutnya, dihitung fungsi objektif antara hasil prediksi r_{k_1} dan target dari masukan i_1, yang dimisalkan sebagai t_{k_1}. Misalkan fungsi objektif yang digunakan adalah *mean*

squared error, maka fungsi objektif dapat dituliskan sebagai:

$$E = \frac{1}{2}(target - prediksi)^2$$
$$E = \frac{1}{2}(t_{k_1} - r_{k_1})^2.$$

Selanjutnya dilakukan *back propagation* berdasarkan fungsi objektif E.

Proses *back propagation* dilakukan dengan mencari turunan parsial fungsi objektif terhadap parameter - parameter yang dioptimisasi. Misalkan dilakukan *back propagation* pada parameter $w_{11}^{(2)}$ dan $w_{12}^{(2)}$. Proses tersebut dapat dituliskan sebagai:

$$\frac{\partial E}{\partial w_{11}^{(2)}} = \frac{\partial E}{\partial r_{k_1}} \frac{\partial r_{k_1}}{\partial a_{k_1}} \frac{\partial ak_1}{\partial w_{11}^{(2)}} \tag{2.8}$$

$$\frac{\partial E}{\partial w_{12}^{(2)}} = \frac{\partial E}{\partial r_{k_1}} \frac{\partial r_{k_1}}{\partial a_{k_1}} \frac{\partial a_{k_1}}{\partial w_{12}^{(2)}}. \tag{2.9}$$

Pertama, dicari turunan parsial fungsi objektif E terhadap r_{k_1}.

$$E = \frac{1}{2}(t_{k_1} - r_{k_1})^2$$
$$\frac{\partial E}{\partial r_{k_1}} = \frac{\partial(\frac{1}{2}(t_{k_1} - r_{k_1})^2)}{\partial r_{k_1}}$$
$$\frac{\partial E}{\partial r_{k_1}} = -1 \times 2 \times \frac{1}{2}(t_{k_1} - r_{k_1})$$
$$\frac{\partial E}{\partial r_{k_1}} = r_{k_1} - t_{k_1}.$$

Selanjutnya, dicari gradien r_{k_1} terhadap nilai a_{k_1}.

$$r_{k_1} = \frac{1}{1 + e^{-a_{k_1}}}$$
$$\frac{\partial r_{k_1}}{\partial a_{k_1}} = \frac{\partial(\frac{1}{1+e^{-a_{k_1}}})}{\partial a_{k_1}}$$
$$\frac{\partial r_{k_1}}{\partial a_{k_1}} = \frac{1}{1 + e^{-a_{k_1}}} \times (1 - \frac{1}{1 + e^{-a_{k_1}}}).$$

Setelah itu, dicari gradien a_{k_1} terhadap $w_{11}^{(2)}$ dan $w_{12}^{(2)}$.

$$a_{k_1} = \sum_{n=1}^{2} w_{1n}^{(2)} j_n$$
$$\frac{\partial a_{k_1}}{\partial w_{11}^{(2)}} = \frac{\partial (\sum_{n=1}^{2} w_{1n}^{(2)} j_n)}{\partial w_{11}^{(2)}}$$
$$\frac{\partial a_{k_1}}{\partial w_{12}^{(2)}} = \frac{\partial (\sum_{n=1}^{2} w_{1n}^{(2)} j_n)}{\partial w_{12}^{(2)}}.$$

Setelah mendapatkan $\frac{\partial E}{\partial r_{k_1}}$, $\frac{\partial r_{k_1}}{\partial a_{k_1}}$, $\frac{\partial a_{k_1}}{\partial w_{11}^{(2)}}$, dan $\frac{\partial a_{k_1}}{\partial w_{12}^{(2)}}$, dicari nilai gradien fungsi objektif E terhadap *weight* $w_{11}^{(2)}$ dan $w_{12}^{(2)}$ dengan menggunakan aturan rantai, seperti yang ditunjukkan oleh **Persamaan 2.8** dan **Persamaan 2.9**. Nilai gradien tersebut kemudian digunakan untuk melakukan optimisasi menggunakan algoritma *gradient descent* (dibahas pada subbab selanjutnya). *Back propagation* pada $w_{11}^{(1)}$ dan $w_{21}^{(1)}$ dilakukan dengan mencari turunan parsial $\frac{\partial E}{\partial w_{11}^{(1)}}$ dan $\frac{\partial E}{\partial w_{21}^{(1)}}$ menggunakan aturan rantai.

2.2 *Gradient Descent*

Algoritma pada jaringan saraf tiruan memanfaatkan optimisasi. Optimisasi merupakan suatu upaya untuk memaksimalkan atau meminimalkan suatu fungsi *f(x)* dengan cara mengubah nilai *x*. Fungsi yang dioptimisasi dapat disebut sebagai fungsi objektif [Goodfellow et al., 2016]. Nilai *x* yang meminimalkan fungsi objetif dinotasikan sebagai $x^* = \text{argmin } f(x)$.

Misalkan terdapat suatu fungsi $y = f(x)$, x dan y merupakan bilangan *real*. Turunan dari fungsi tersebut dinotasikan sebagai *f'(x)* atau $\frac{dy}{dx}$. Turunan *f'(x)* menunjukkan kemiringan fungsi *f(x)* pada titik *x*. Turunan fungsi menunjukkan efek perubahan nilai masukan fungsi terhadap perubahan nilai keluaran pada fungsi, yang secara matematis ditulis sebagai $f(x+\varepsilon) \equiv f(x) + \varepsilon f'(x)$.

Turunan fungsi berguna untuk meminimalkan fungsi tersebut karena turunan mampu menunjukkan besarnya perubahan nilai *x* yang dibutuhkan untuk memberikan perubahan dengan nilai tertentu pada *y*. Misalkan diketahui $f(x - \varepsilon f'(x))$ lebih kecil dari $f(x)$ untuk suatu nilai ε yang cukup kecil. Nilai *f(x)* dapat dikurangi dengan cara menggeser nilai *x* dengan nilai yang cukup kecil dengan arah yang berlawanan terhadap tanda turunan fungsi tersebut [Goodfellow et al., 2016]. Teknik tersebut dinamakan *gradient descent*.

Ketika $f'(x) = 0$, turunan fungsi tidak memberikan informasi mengenai arah

pergerakan nilai x. Nilai x yang menyebabkan $f'(x) = 0$ disebut sebagai titik kritis atau titik stasioner. Titik minimum lokal merupakan titik atau nilai x yang menyebabkan fungsi $f(x)$ memiliki nilai minimum dibandingkan nilai lain di sekitarnya. Titik maksimum lokal merupakan titik atau nilai x yang menyebabkan fungsi $f(x)$ memiliki nilai maksimum dibandingkan nilai sekitarnya. Titik pelana merupakan nilai x yang menyebabkan fungsi $f(x)$ tidak terkategori sebagai titik minimum lokal ataupun titik maksimum lokal. Titik minimum global merupakan nilai x yang menyebabkan fungsi $f(x)$ paling minimum dibandingkan dengan nilai lainnya. Titik minimum lokal bisa saja bukan merupakan titik minimum global[Goodfellow et al., 2016].

Pada model jaringan saraf tiruan, proses optimisasi melibatkan fungsi yang memiliki banyak nilai minimum lokal yang kurang optimal dan juga banyak nilai pelana. Kondisi tersebut menyebabkan proses optimisasi menjadi sulit, terutama untuk masukan multidimensional. Proses optimisasi pada model jaringan saraf tiruan biasanya berusaha menemukan nilai yang cukup minimal namun bukan nilai minimum global [Goodfellow et al., 2016].

Secara matematis, *gradient descent* dapat ditulis sebagai:

$$\boldsymbol{x'} = \boldsymbol{x} - \varepsilon \nabla_{\boldsymbol{x}} f(\mathbf{x}) \tag{2.10}$$

Nilai $\boldsymbol{x}$ dan $\boldsymbol{x'}$ berupa vektor yang menunjukkan bahwa fungsi $f(\boldsymbol{x})$ menerima masukan multivariabel. Vektor $\boldsymbol{x}$ menunjukkan nilai masukan awal, sementara vektor $\boldsymbol{x'}$ menunjukkan nilai $\boldsymbol{x}$ yang sudah dioptimalkan melalui *gradient descent*. Karena masukan $\boldsymbol{x}$ bersifat multivariabel, maka proses penurunan fungsi menggunakan turunan parsial. Turunan parsial $\frac{\partial}{\partial x_i} f(\boldsymbol{x})$ menunjukkan besarnya perubahan nilai $f(\boldsymbol{x})$ terhadap besarnya perubahan nilai x_i yang berada pada titik $\boldsymbol{x}$. Gradien f merupakan vektor yang komponennya merupakan turunan parsial $f(\boldsymbol{x})$ terhadap semua x_i, yang dinyatakan dengan notasi $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$. Simbol ε merupakan *learning rate*, suatu nilai skalar yang menentukan seberapa besar perubahan nilai $\boldsymbol{x}$. Pemilihan nilai ε dapat dilakukan dengan beberapa cara. Cara yang pertama adalah dengan menetapkan nilai konstan yang cukup kecil pada ε. Metode lainnya adalah dengan memanfaatkan metode yang bersifat dinamis seperti *line search*.

Terdapat beberapa varian *gradient descent* diantaranya *stochastic gradient descent*, metode momentum, metode momentum Nesterov, metode Adagrad, metode RMSProp, dan metode *adaptive moment* (adam). Bagian ini fokus menjelaskan *stochastic gradient descent*, metode momentum, dan metode adam.

2.2.1 *Stochastic Gradient Descent*

Stochastic gradient descent merupakan modifikasi dari *gradient descent*. Permasalahan utama pada saat proses belajar suatu model *machine learning* adalah dibutuhkannya banyak data. Namun data yang banyak menyebabkan biaya komputasi yang mahal. Fungsi objektif yang biasa dipergunakan pada model *machine learning* adalah penjumlahan fungsi objektif sampel data pada pembelajaran. Sebagai contoh, untuk fungsi objektif berupa *negative conditional log-likelihood* dari sampel pada pembelajaran dapat ditulis sebagai:

$$J(\theta) = \mathbb{E}_{\mathbf{x},\mathbf{y}\sim p_{data}} L(\boldsymbol{x},\boldsymbol{y},\theta) = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\theta),\boldsymbol{y}^{(i)}) \tag{2.11}$$

dengan fungsi objektif per sampel $L(f(\boldsymbol{x};\ \theta),\ \boldsymbol{y})$ = - log $p(\boldsymbol{y} \mid \boldsymbol{x};\ \theta)$. *Gradient descent* memperhitungkan gradien fungsi objektif yang ditulis sebagai:

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta L(f(\boldsymbol{x}^{(i)};\theta),\boldsymbol{y}^{(i)}) \tag{2.12}$$

Konsep *stochastic gradient descent* adalah dengan menganggap gradien sebagai nilai ekspektasi. Nilai ekspektasi dapat diestimasi menggunakan sejumlah sampel. Secara lebih detail, untuk setiap iterasi pada saat pembelajaran, dilakukan *sampling minibatch* terhadap data sampel $\mathbb{B} = \{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(n)}\}$ dengan n lebih kecil dibandingkan dengan jumlah seluruh sampel pembelajaran m. Estimasi gradien dapat dituliskan sebagai:

$$\boldsymbol{g} = \frac{1}{n}\nabla_\theta \sum_{i=1}^{n} L(f(\boldsymbol{x}^{(i)};\theta),\boldsymbol{y}^{(i)}) \tag{2.13}$$

menggunakan sampel dari *minibatch* $\mathbb{B}$. Secara matematis, *stochastic gradient descent* dapat dituliskan sebagai:

$$\theta \leftarrow \theta - \varepsilon \boldsymbol{g} \tag{2.14}$$

dengan ε merupakan *learning rate*. Algoritma *stochastic gradient descent* ditunjukkan sebagai berikut:

Algoritma 1 *Stochastic gradient descent* pada iterasi / *epoch* ke-*k*

Require:

- Nilai awal parameter θ, *learning rate* ε

while belum mencapai kriteria belajar **do**

- *Sampling* minibatch n dari sampel pembelajaran $\{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(n)}\}$ dengan target $\boldsymbol{y}^{(i)}$
- Hitung gradien $\boldsymbol{g} \leftarrow \frac{1}{n}\nabla_\theta \sum_i L(f(\boldsymbol{x}^{(i)};\theta), \boldsymbol{y}^{(i)})$
- Lakukan pembaruan: $\theta = \theta - \varepsilon \boldsymbol{g}$

2.2.2 Metode Momentum

Optimisasi menggunakan *stochastic gradient descent* terkadang membutuhkan waktu yang lama. Optimisasi menggunakan momentum dapat menyelesaikan permasalahan tersebut karena dapat mempercepat proses belajar pada jaringan saraf tiruan. Penggunaan momentum efektif pada fungsi objektif yang memiliki kelengkungan permukaan yang tinggi namun memiliki nilai gradien yang konsisten di sepanjang kurvanya. Metode Momentum bekerja dengan cara mengakumulasi rata-rata gradien fungsi objektif proses belajar pada iterasi sebelumnya dan berlanjut sesuai dengan arahnya. Optimisasi momentum umumnya dikombinasikan dengan metode *stochastic gradient descent* [Goodfellow et al., 2016].

Momentum memiliki variabel $\boldsymbol{v}$ yang berfungsi seperti kecepatan pada fenomena fisika. Variabel $\boldsymbol{v}$ menentukan nilai dan arah pergerakan parameter pada ruang parameter. Nilai $\boldsymbol{v}$ didapat dari nilai rata-rata gradien proses belajar pada iterasi sebelumnya. Nilai gradien pada proses belajar akan mengecil secara eksponensial seiring bertambahnya iterasi. Penggunaan terminologi momentum terinspirasi dari fenomena momentum dalam fisika. Variabel $\boldsymbol{v}$ berfungsi seperti momentum dengan massa satuan. Metode momentum membutuhkan hyperparameter α yang berfungsi untuk menentukan kecepatan mengecilnya kontribusi gradien-gradien sebelumnya. Secara matematis, proses optimisasi menggunakan momentum dapat ditulis sebagai:

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \varepsilon \nabla_\theta \left(\frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\theta), \boldsymbol{y}^{(i)})\right) \tag{2.15}$$

$$\theta \leftarrow \theta + \boldsymbol{v} \tag{2.16}$$

Variabel $\boldsymbol{v}$ mengakumulasi gradien $\nabla_\theta(\frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\theta), \boldsymbol{y}^{(i)}))$. Semakin besar nilai *hyperparameter* α terhadap ε, maka nilai gradien pada iterasi sebelumnya semakin berpengaruh dalam perhitungan gradien pada iterasi saat itu. Algoritma *stochastic*

gradient descent dengan menggunakan metode momentum adalah sebagai berikut:

Algoritma 2 *Stochastic gradient descent* menggunakan metode momentum

Require:

- *Learning rate* ε, parameter momentum α
- Nilai awal parameter θ, nilai awal kecepatan $\boldsymbol{v}$

while belum mencapai kriteria berhenti **do**

- *Sampling* minibatch n dari sampel pembelajaran $\{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(n)}\}$ dengan target $\boldsymbol{y}^{(i)}$
- Hitung gradien $\boldsymbol{g} \leftarrow \frac{1}{n}\nabla_\theta \sum_i L(f(\boldsymbol{x}^{(i)};\theta), \boldsymbol{y}^{(i)})$
- Hitung nilai $\boldsymbol{v}$ yang baru: $\boldsymbol{v} \leftarrow \alpha\boldsymbol{v} - \varepsilon\boldsymbol{g}$
- Lakukan pembaruan: $\theta = \theta - \varepsilon\boldsymbol{g}$

2.2.3 Metode *Adaptive Moment*

Metode *adaptive moment* (adam) merupakan metode optimisasi yang memiliki kemampuan untuk mengatur nilai *learning rate* secara dinamis. Metode adam dapat dikatakan sebagai kombinasi antara metode momentum dan RMSProp [Goodfellow et al., 2016]. Ide utama dari algoritma adam adalah melakukan *rescaling* terhadap nilai rata-rata gradien. Nilai rata-rata gradien didapat dengan mengadaptasi metode momentum. Proses *rescaling* rataan gradien dilakukan dengan mengadaptasi metode RMSProp [Kingma and Ba, 2014]. Algoritma adam adalah sebagai berikut:

Algoritma 3 Algoritma adam

Require:

- *Step size* ε (nilai ideal: 0.001)
- Laju luruh konstanta eksponensial dari momen ρ_1 dan ρ_2 pada selang [0, 1) (nilai ideal: 0.9 untuk ρ_1 dan 0.999 untuk ρ_2)
- Konstanta δ untuk menjaga stabilitas numerik (nilai ideal: 10^{-8})
- Nilai awal terhadap parameter θ
- Nilai awal variabel momen $\boldsymbol{s} = 0, \boldsymbol{r} = 0$
- Nilai awal *time step t = 0*

while belum mencapai kriteria berhenti **do**

- *sampling* minibatch n dari sampel pembelajaran $\{\boldsymbol{x}^{(i)}, ..., \boldsymbol{x}^{(n)}\}$ dengan target $\boldsymbol{x}^{(i)}$
- hitung gradien $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_\theta \sum_i L(f(\boldsymbol{x}^{(i)};\theta),\boldsymbol{y}^{(i)})$
- $t \leftarrow t+1$
- $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1-\rho_1)\boldsymbol{g}$
- $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1-\rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
- $\boldsymbol{s'} = \frac{\boldsymbol{s}}{1-\rho_1^t}$
- $\boldsymbol{r'} = \boldsymbol{r} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$
- $\Delta\theta = -\varepsilon\frac{\boldsymbol{s'}}{\sqrt{\boldsymbol{r'}}+\delta}$ (operasi hadamard (*element-wise*))
- $\theta = \theta + \Delta\theta$

2.3 Regularisasi

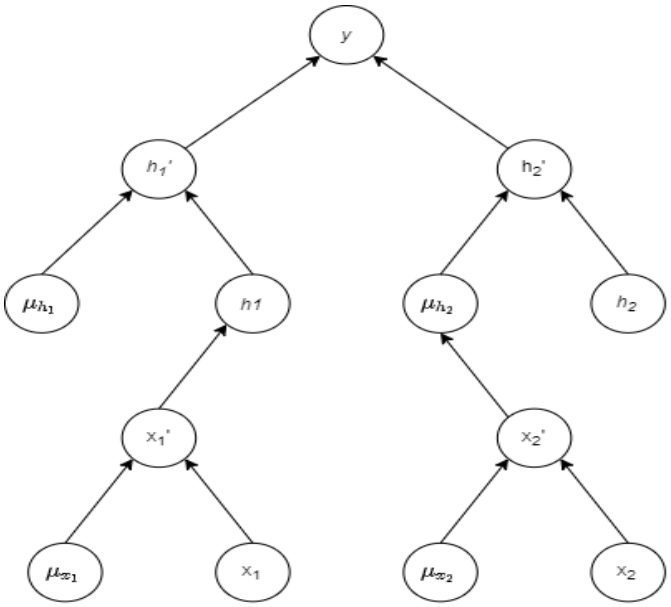
Regularisasi merupakan segala bentuk modifikasi yang dilakukan pada algoritma *machine learning* yang bertujuan untuk mengurangi *generalization error* tanpa mengurangi nilai fungsi objektif pada saat pembelajaran [Goodfellow et al., 2016]. Metode regularisasi bersifat unik untuk setiap algoritma *machine learning*, sehingga diperlukan pemilihan metode regularisasi yang sesuai untuk tugas spesifik yang dikerjakan oleh suatu algoritma *machine learning*. Beberapa metode regularisasi yang umum diantaranya penerapan *norm penalty* pada parameter *machine learning*, *data augmentation*, penggunaan *noise*, *early stopping*, *semi supervised learning*, *multi-tasking learning*, *parameter tying and parameter sharing*, *sparse representation*, *ensemble method*, *drop out*, *adversarial training*, *tangent distance, tangent prop*, dan *manifold tangent classifier*.

2.3.1 *Dropout*

Dropout merupakan teknik regularisasi yang mengubah konsep belajar pada jaringan saraf tiruan yang menggunakan seluruh *weight* pada perseptron menjadi sebagian dari

weight untuk setiap iterasi selama proses belajar. Metode *dropout* menyelesaikan permasalahan *overfitting* pada arsitektur jaringan saraf tiruan yang besar dan kompleks yang sebelumnya tidak dapat diselesaikan dengan teknik regularisasi *norm penalty* [Goodfellow et al., 2016]. *Overfitting* merupakan kondisi jaringan saraf tiruan yang memiliki *generalization error* yang tinggi meskipun fungsi objektif yang dihasilkan pada saat pembelajaran rendah [Goodfellow et al., 2016]. Permasalahan *overfitting* disebabkan oleh *coadaptation*. *Coadaptation* merupakan adanya kecenderungan beberapa koneksi *weight* pada perseptron memiliki kapasitas aproksimasi yang jauh lebih besar dibandingkan koneksi *weight* yang lainnya [Srivastava et al., 2014]. Akibatnya, terdapat koneksi *weight* yang diabaikan ketika melakukan proses aproksimasi. Teknik regularisasi *norm penalty* tidak dapat menyelesaikan permasalahan *coadaptation* karena *norm penalty* tidak melakukan regularisasi terhadap *weight*, namun terhadap hasil aproksimasi [Srivastava et al., 2014].

Proses belajar dengan regularisasi *dropout* memanfaatkan algoritma *minibatch-based learning*, seperti *stochastic gradient descent*. Setiap kali memasukkan sampel data pada *minibatch*, dilakukan *sampling mask* yang dapat diterapkan pada lapisan tersembunyi maupun lapisan masukan. *Mask* merupakan suatu vektor biner, yaitu vektor yang komponennya bernilai satu atau nol, yang berukuran sama dengan lapisan perseptron atau masukan. *Mask* berfungsi untuk menentukan perseptron yang aktif ataupun komponen vektor masukan yang digunakan (masukan multidimensional) ketika proses pembelajaran. Komponen *mask* yang bernilai satu menunjukkan bahwa perseptron pada lapisan tersembunyi atau komponen vektor masukan yang bersesuaian terhadap *mask* aktif pada saat proses belajar. Komponen *mask* yang bernilai nol bekerja sebaliknya. Proses *sampling* pada *mask* dilakukan secara independen antar unit pada lapisan tersembunyi maupun pada lapisan masukan. Peluang *sampling* pada mask merupakan *hyperparameter* yang nilainya tetap. Penentuan *hyperparameter* dilakukan sebelum pembelajaran. Secara formal, misalkan vektor *mask* μ menentukan unit yang akan terlibat dalam proses belajar pada *batch* tertentu, dan $L(\theta, \mu)$ menyatakan nilai fungsi objektif model jaringan saraf tiruan yang terparameterisasi θ dan *mask* μ. Proses belajar pada jaringan saraf tiruan dengan menggunakan regularisasi *dropout* bertujuan untuk meminimalkan $\mathbb{E}_{\mu} L(\theta, \mu)$ [Goodfellow et al., 2016]. **Gambar 2.7** menunjukkan model jaringan saraf tiruan yang menggunakan *dropout*.



Gambar 2.7: Model jaringan saraf tiruan yang dilatih menggunakan *dropout*.

Gambar tersebut menunjukkan jaringan saraf tiruan yang memiliki dua lapis. Setiap *neuron* (x_1, x_2, h_1, h_2) memiliki peluang bernilai p untuk di-*masking* $(\mu x_1, \mu x_2, \mu h_1, \mu h_2)$ atau tidak. *Neuron* yang sudah ditentukan untuk di-*masking* atau tidak menentukan keterlibatan mereka dalam proses pembelajaran (x'_1, x'_2, h'_1, h'_2). *Neuron* yang dimasking akan menghasilkan nilai 0, sementara *neuron* yang tidak di-*masking* menghasilkan keluaran bernilai *real*. Keluaran yang dihasilkan oleh x'_1 dan x'_2 menjadi masukan bagi h_1 dan h_2.

Penelitian menunjukkan bahwa dropout bekerja lebih efektif dibandingkan dengan teknik regularisasi lain seperti *weight decay*, *filter norm constraint*, dan *sparse regularization*. Penelitian juga menunjukkan bahwa metode *dropout* dapat dikombinasikan dengan metode regularisasi lainnya [Srivastava et al., 2014].

Terdapat dua keuntungan melakukan regularisasi menggunakan tekanik *dropout*. Keuntungan yang pertama adalah biaya komputasi yang murah. Regularisasi menggunakan *dropout* membutuhkan komputasi sebesar *O(n)* dikali jumlah lapisan yang menggunakan *dropout* per sampel, dengan n menyatakan dimensi dari vektor *mask* [Goodfellow et al., 2016]. Komputasi yang dilakukan adalah proses *sampling* terhadap komponen-komponen vektor *mask*. Keuntungan yang kedua adalah *dropout* tidak terbatas pada model jaringan saraf tiruan tertentu. Penelitian menunjukkan bahwa *dropout* dapat diterapkan pada model probabilistik seperti *restricted Boltzman machine* [Srivastava et al., 2014] dan pada arsitektur *recurrent neural network* [Pascanu et al., 014a] [Bayer and Osendorfer, 2014].

Proses *sampling* pada *dropout* yang dilakukan pada setiap iterasi dapat disimplifikasi dengan cara mengestimasi seluruh submodel dari arsitektur jaringan saraf tiruan. Teknik

aproksimasi ini disebut sebagai *fast dropout* [Wang and Manning, 2013]. *Fast dropout* memiliki keuntungan yaitu waktu yang dibutuhkan untuk proses pembelajaran lebih cepat dan memberikan hasil yang kurang lebih sama dengan metode *dropout* yang biasa.

2.4 Jaringan Saraf Konvolusional

Jaringan saraf konvolusional (selanjutnya dapat juga disebut sebagai JSK) merupakan jaringan saraf tiruan khusus yang diterapkan pada data yang memiliki topologi seperti *grid* [LeCun et al., 1995, Goodfellow et al., 2016]. Beberapa contoh data yang memiliki topologi *grid* diantaranya adalah data *time series* dan data berupa citra. Data *time series* dapat dipandang sebagai *grid* satu dimensi dengan interval waktu tertentu. Sementara itu, data citra dapat dipandang sebagai *grid* dua dimensi. Jaringan saraf konvolusional menggunakan operasi konvolusi sebagai ganti operasi perkalian *weight* $\boldsymbol{w}$ pada setidaknya salah satu lapisan pada jaringan saraf tiruan.

Operasi konvolusi merupakan operasi matematika yang melibatkan dua buah fungsi. Misalkan dua buah fungsi tersebut adalah x dan w. Konvolusi x dan w didefinisikan sebagai berikut:

$$s(t) = \int x(a)\, w(t-a) da \tag{2.17}$$

Fungsi konvolusi dilambangkan dengan simbol *asterisk* (*) dan dituliskan sebagai:

$$s(t) = (x * w)(t) \tag{2.18}$$

Dalam konteks jaringan saraf tiruan, fungsi x menyatakan masukan pada jaringan saraf tiruan dan fungsi w menyatakan kernel. Keluaran dari operasi konvolusi masukan dan kernel disebut sebagai *feature map* [Goodfellow et al., 2016].

Operasi konvolusi yang digunakan pada JSK merupakan operasi yang bersifat diskret karena hampir semua data yang diolah oleh komputer bersifat diskret. Masukan pada jaringan konvolusi berupa *array* multidimensional yang berisi data dan kernel merupakan *array* multidimensional yang berisi parameter yang dapat dioptimisasi. *Array* multidimenisonal ini dapat juga disebut sebagai tensor. Kernel berfungsi untuk mendapatkan fitur penting dari masukan. Masukan pada jaringan konvolusional dapat dianggap berukuran tak hingga, dengan nilai selain masukan yang sebenarnya bernilai nol [Goodfellow et al., 2016]. Berdasarkan syarat tersebut, operasi konvolusi pada JSK

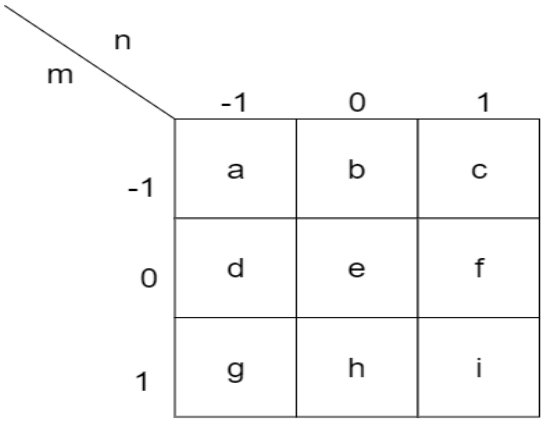
dapat dituliskan kembali menjadi:

$$s(t) = \sum_{-\infty}^{\infty} x(a)w(t-a) \tag{2.19}$$

Sebagai contoh, misalkan *I* merupakan masukan berupa citra (masukan dua dimensi) dan *K* merupakan suatu kernel dua dimensi, maka operasi konvolusi *S* baris ke-*i* dan kolom ke-*j* dapat ditulis sebagai:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.20}$$

Notasi *i* dan *j* menyatakan indeks kolom dan baris dari nilai operasi konvolusi yang dicari, sedangkan notasi *m* dan *n* menyatakan dimensi dari kolom dan baris dari masukan *I*. Kernel pada operasi tersebut umumnya berpusat di tengah, artinya indeks K(0, 0) berada di tengah kernel. Misalkan jika suatu kernel berukuran 3x3, maka indeks kernel tersebut adalah -1, 0, 1, dengan titik pusat (0, 0) berada di tengah kernel. **Gambar 2.8** menunjukkan contoh kernel pada operasi konvolusi.

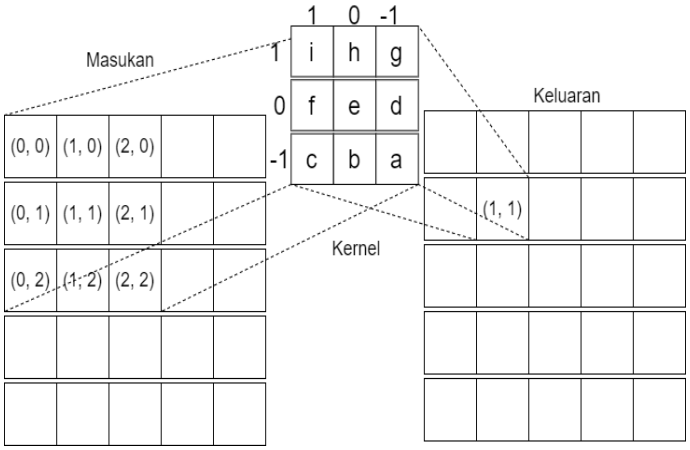


Gambar 2.8: Contoh kernel pada operasi konvolusi.

Selanjutnya, misalkan terdapat masukan citra berukuran 5x5 dan kernel berukuran 3x3, maka operasi konvolusi pada indeks (1, 1) dapat dituliskan sebagai:

$$\begin{aligned} S(1,1) &= \sum_m \sum_n I(m,n)K(1-m,1-n) \\ &= I(0,0)*K(1,1) + I(1,0)*K(0,1) + I(2,0)*K(-1,1) + \\ &\quad I(0,1)*K(1,0) + I(1,1)*K(0,0) + I(2,1)*K(-1,0) + \\ &\quad I(0,2)*K(1,-1) + I(1,2)*K(0,-1) + I(2,2)*K(-1,-1) \end{aligned}$$

Gambar 2.9 menunjukkan operasi konvolusi tersebut.



Gambar 2.9: Operasi Konvolusi pada Indeks (1,1) dengan Masukan Berukuran 5x5 dan Kernel Berukuran 3x3.

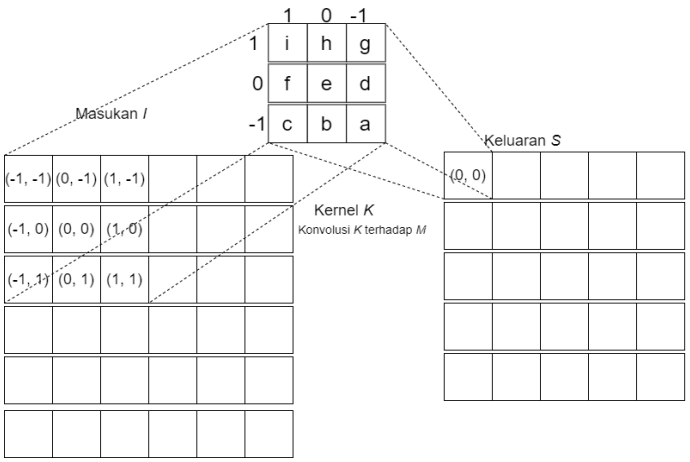
Pada operasi konvolusi, posisi kernel dirotasi sebesar 180° relatif terhadap posisi masukan. Operasi konvolusi dengan indeks masukan yang berada di bagian tepi dapat dilakukan karena dimensi masukan yang dianggap tak berhingga, dengan nilai masukan pada indeks di luar batas indeks sebenarnya bernilai 0. Misalkan dilakukan kembali operasi konvolusi terhadap masukan berukuran 5x5. Operasi konvolusi dilakukan pada indeks (0, 0). Operasi konvolusi tersebut dituliskan sebagai:

$$\begin{aligned} S(0,0) &= \sum_m \sum_n I(m,n)K(1-m,1-n) \\ &= I(-1,-1)*K(1,1)+I(0,-1)*K(0,1)+I(1,-1)*K(-1,1)+ \\ &\quad I(-1,0)*K(1,0)+I(0,0)*K(0,0)+I(1,0)*K(-1,0)+ \\ &\quad I(-1,1)*K(1,-1)+I(0,1)*K(0,-1)+I(1,1)*K(-1,-1) \end{aligned}$$

Karena *I(-1, -1), I(0, -1), I(1, -1), I(-1, 0)*, dan *I(-1, 1)* bernilai 0, maka:

$$S(0,0) = I(0,0)*K(0,0)+I(1,0)*K(-1,0)+I(0,1)*K(0,-1)+I(1,1)*K(-1,-1)$$

Gambar 2.10 menunjukkan operasi konvolusi tersebut.



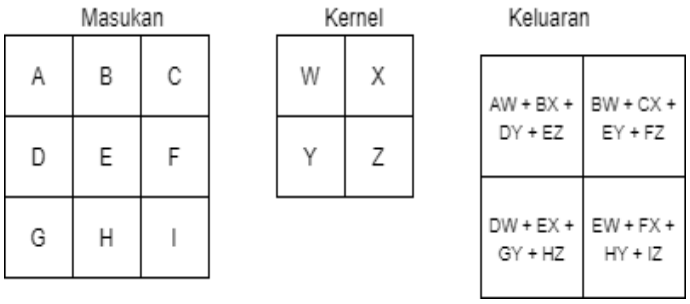
Gambar 2.10: Operasi Konvolusi pada Indeks (0, 0) dengan Masukan Berukuran 5x5 dan Kernel Berukuran 3x3.

Terdapat operasi yang mirip dengan operasi konvolusi yang dinamakan operasi *cross-correlation*. Operasi *cross-correlation* bekerja seperti operasi konvolusi, hanya saja tanpa memutar kernel relatif terhadap masukan. Operasi *cross-correlation* dapat ditulis sebagai:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \tag{2.21}$$

Implementasi operasi konvolusional pada kebanyakan *library* jaringan saraf tiruan menggunakan operasi *cross-correlation* [Goodfellow et al., 2016]. Meskipun diimplementasi menggunakan *cross-correlation*, operasi tersebut tetap dinamakan operasi konvolusi.

Operasi konvolusi pada JSK dapat dianggap sebagai perkalian matriks berbasis indeks. Ukuran matriks kernel pada umumnya lebih kecil dibandingkan dengan ukuran masukan. Jumlah kernel yang digunakan pada suatu lapisan JSK dapat lebih dari satu. Matriks masukan juga dapat berjumlah lebih dari satu. Data citra RGB (Red Green Blue) merupakan contoh masukan yang bila direpresentasikan dalam matriks berjumlah lebih dari satu (memilik tiga *channel*). Jumlah operasi perkalian matriks pada suatu lapisan JSK sejumlah kernel yang digunakan pada lapisan tersebut. **Gambar 2.11** menunjukkan operasi perkalian satu matriks kernel dan satu matriks masukan.



Gambar 2.11: Operasi konvolusi pada JSK.

Operasi perkalian matriks pada JSK dilakukan terhadap elemen pada posisi yang bersesuaian. Matriks kernel akan bergerak di sepanjang matriks masukan dan mengalikan elemen yang bersesuaian posisinya. Misalkan nilai *A, B, C, D, E, F, G, H*, dan *I* masing-masing adalah 1. Lalu, nilai *W, X, Y,* dan *Z* masing-masing adalah 2. Operasi konvolusi yang dilakukan oleh matriks kernel *WXYZ* terhadap matriks masukan *ABCDEFGHI* menghasilkan matriks keluaran berukuran 2×2. Berdasarkan operasi konvolusi, nilai keluaran pada setiap sel matriks adalah 8. Matriks kernel pada JSK merupakan parameter yang dioptimisasi untuk mencapai konvergensi pada proses pembelajaran. Proses optimisasi dilakukan melalui *gradient descent* dengan terlebih dahulu melakukan *backpropagation* terhadap matriks keluaran, matriks kernel, dan matriks masukan.

BAB 3
LEAST SQUARE GENERATIVE ADVERSARIAL NETWORK

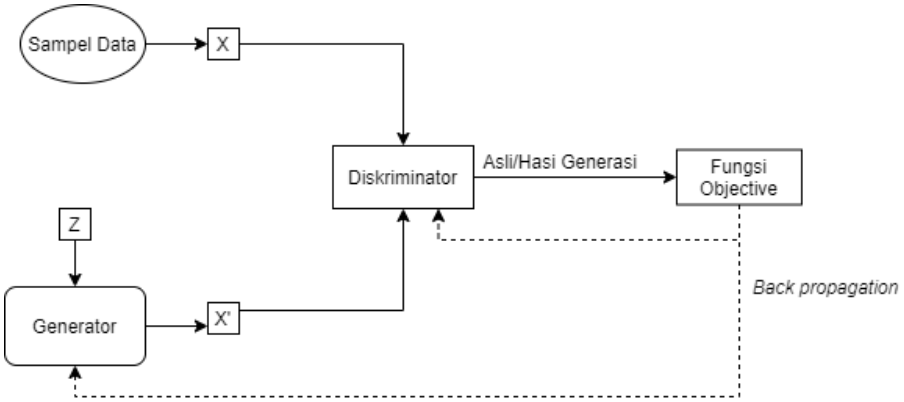
Bab ini menjelaskan *generative adversarial network* (GAN) sebagai model pembangkit dan *Least Square Generative Adversarial Network* (LSGAN) yang merupakan varian arsitektur GAN.

3.1 Generative Adversarial Network

Generative Adversarial Network merupakan model jaringan saraf tiruan yang terdiri dari dua jaringan yaitu generator G dan diskriminator D [Langr and Bok, 2019]. Generator berfungsi untuk mempelajari distribusi data berdasarkan sampel pembelajaran sedangkan diskriminator berfungsi untuk mengestimasi probabilitas data berasal dari sampel pembelajaran dan bukan berasal dari generator G [Goodfellow et al., 2014]. Generator dan diskriminator bekerja secara adversarial (berlawanan) dengan generator G akan memaksimalkan probabilitas diskriminator D gagal mengestimasi probabilitas data berasal dari sampel pembelajaran. Diskriminator D bekerja sebaliknya, yaitu dengan meminimalkan probabilitas dirinya gagal mengestimasi probabilitas data berasal dari sampel pembelajaran. Pembelajaran pada jaringan generator bertujuan menghasilkan data yang memiliki karakteristik yang mirip dengan sampel pembelajaran. Pembelajaran pada jaringan diskriminator bertujuan membedakan data yang berasal dari sampel pembelajaran dan data yang dihasilkan oleh generator.

Jaringan generator membangkitkan data dengan cara melakukan *sampling* ($\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})$) terhadap suatu variabel acak $\boldsymbol{z}$ yang disebut sebagai *noise*, dari suatu distribusi apriori $p_{\boldsymbol{z}}$. Variabel ini kemudian dipetakan oleh fungsi generator $G(\boldsymbol{z};\ \theta_g)$ untuk menghasilkan data $\boldsymbol{x}'$ yang memiliki karakteristik yang mirip dengan sampel pembelajaran [Goodfellow et al., 2014]. Fungsi $G(\boldsymbol{z};\ \theta_g)$ dioptimisasi berdasarkan parameter θ_g, yang merepresentasikan *weight* pada jaringan generator. Data yang dihasilkan kemudian menuju jaringan diskriminator. Selanjutnya, data yang dihasilkan jaringan generator menjadi masukan bagi fungsi diskriminator untuk dibedakan terhadap data $\boldsymbol{x}$ yang berasal dari sampel pembelajaran ($\boldsymbol{x} \sim p_{data}(\boldsymbol{x})$). Terdapat dua proses pemetaan pada model diskriminator. Pemetaan pertama adalah $D(\boldsymbol{x};\ \theta_d)$ dengan masukan berupa data dari sampel pembelajaran. Pemetaan kedua adalah $D(G(\boldsymbol{z};\ \theta_g);\ \theta_d)$ dengan masukan berupa data yang dihasilkan oleh jaringan generator [Goodfellow et al., 2014]. Fungsi pada jaringan diskriminator dioptimisasi berdasarkan

parameter θ_d, yang merepresentasikan *weight* pada jaringan diskriminator. **Gambar 3.1** menunjukkan skema GAN.



Gambar 3.1: Skema *Generative Adversarial Network*.

Model *generative adversarial network* mengoptimisasi fungsi objektif V(D, G), yang dituliskan sebagai:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1-D(G(\boldsymbol{z})))] \tag{3.1}$$

dengan fungsi $D(\boldsymbol{x};\theta_d)$ dan fungsi $D(\boldsymbol{x})$ merupakan fungsi yang sama, begitu pula dengan fungsi $G(\boldsymbol{z};\theta_g)$ dan fungsi $G(\boldsymbol{z})$. Fungsi tersebut disebut adversarial karena berusaha memaksimalkan nilai $D(\boldsymbol{x})$ dan meminimalkan nilai $G(\boldsymbol{z})$ disaat yang bersamaan.

Algoritma *training* pada GAN [Goodfellow et al., 2014] adalah sebagai berikut:

Algoritma 4 *Minibatch stochastic gradient descent training* pada GAN.

for jumlah iterasi pada *training* **do**

- *Sampling minibatch* pada *noise* z sebanyak m $(\boldsymbol{z}^{(1)},...,\boldsymbol{z}^{(m)})$ dari distribusi apriori $p_g(\boldsymbol{z})$
- *Sampling minibatch* pada *data* $\boldsymbol{x}$ sebanyak m $(\boldsymbol{x}^{(1)},...,\boldsymbol{x}^{(m)})$ dari distribusi sampel pembelajaran $p_{data}(\boldsymbol{x})$
- Perbarui parameter pada diskriminator menggunakan *stochastic gradient*:
  $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [\log \mathrm{D}(\boldsymbol{x}^{(i)}) + \log(1-D(G(\boldsymbol{z}^{(i)})))]$
- *Sampling minibatch* pada *noise* $\boldsymbol{z}$ sebanyak m $(\boldsymbol{z}^{(1)},...,\boldsymbol{z}^{(m)})$ dari distribusi apriori $p_g(\boldsymbol{z})$
- Perbarui parameter pada generator menggunakaan *stochastic gradient*:
  $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1-D(G(\boldsymbol{z}^{(i)})))$

Pada algoritma tersebut terdapat dua optimisasi. Optimisasi pertama dilakukan pada diskriminator. Optimisasi tersebut melibatkan data yang berasal dari sampel

pembelajaran dan data yang dihasilkan oleh jaringan generator. Optimisasi dilakukan untuk memaksimalkan kemampuan diskriminator membedakan kedua jenis data tersebut. Optimisasi kedua dilakukan terhadap generator. Optimisasi pada generator meminimalkan kemampuan diskriminator untuk membedakan data yang dihasilkan oleh jaringan generator dan data yang berasal dari sampel pembelajaran. Terdapat berbagai varian arsitektur GAN, diantaranya *Least Square* GAN [Mao et al., 2017], *Wasserstein*-GAN [Arjovsky et al., 2017], *Wasserstein*-GAN *gradient penalized* [Gulrajani et al., 2017], *Conditional* GAN [Mirza and Osindero, 2014], *Info* GAN [Chen et al., 2016], *Auxilary Classifier* GAN [Odena et al., 2017], *Deep Regret Analytic* GAN [Kodali et al., 2017], *Energy Based* GAN [Zhao et al., 2016], dan *Boundary Equilibrium* GAN [Berthelot et al., 2017].

Generative adversarial network merupakan model pembangkit. Model pembangkit mampu menghasilkan data yang memiliki makna dari suatu distribusi apriori. Pada saat proses belajar, generator pada GAN memetakan sampel pembelajaran mengikuti distribusi aprori. Ketika proses belajar selesai, GAN mampu menghasilkan keluaran yang memiliki karakteristik yang mirip dengan sampel pembelajaran (dihasilkan oleh generator). Kemampuan GAN sebagai model pembangkit banyak dimanfaatkan di bidang pengolahan citra dan pengolahan audio untuk menghasilkan sampel citra maupun audio yang baru dan bermakna.

3.1.1 Teori *Generative Adversarial Network*

Generator G secara implisit mengestimasi distribusi data hasil pembangkitan p_g berdasarkan fungsi $G(\boldsymbol{z})$ yang melibatkan *noise* $\boldsymbol{z}$, dengan *noise* didapat dari hasil *sampling* $\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})$. **Algoritma 4** bertujuan untuk mencapai titik yang konvergen sedemikian sehingga mampu menjadi estimator terhadap distribusi p_{data}. Pada bagian ini ditunjukkan bahwa terdapat nilai optimum sehingga $p_g = p_{data}$. Kemudian akan diperlihatkan bahwa **Algoritma 4** dapat mengoptimisasi **Persamaan 3.1** [Goodfellow et al., 2014].

3.1.1.1 Nilai Optimal Distribusi $p_{data} = p_g$

Misalkan terdapat suatu diskriminator D yang optimal terhadap suatu generator G. **Proposisi 1** [Goodfellow et al., 2014] *Untuk suatu generator G, nilai optimal diskriminator D adalah*

$$D_G^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \tag{3.2}$$

Fungsi objektif diskriminator D dapat dianggap sebagai upaya memaksimalkan nilai *log-likelihood* untuk mengestimasi probabilitas bersyarat $P(Y = y|x)$, dengan Y memberi informasi apakah $\boldsymbol{x}$ berasal dari p_{data} $(y = 1)$ atau p_g $(y = 0)$ [Goodfellow et al., 2014]. Berdasarakan **Proposisi 1**, **Persamaan 3.1** dapat dituliskan kembali sebagai:

$$\begin{aligned} C(G) &= \max_D V(G,D) \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D^*_G(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_z}[\log(1 - D^*_G(G(\boldsymbol{z})))] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D^*_G(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(1 - D^*_G(\boldsymbol{x}))] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log \frac{p_g(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}] \end{aligned} \tag{3.3}$$

Teorema 1 [Goodfellow et al., 2014] *Fungsi $C(G)$ mencapai nilai minimum ketika $p_{data} = p_g$. Nilai minimum yang dicapai adalah* - log *4.*

Berdasarkan **Proposisi 1** dan **Teorema 1**, fungsi objektif pada GAN meminimalkan fungsi Jensen-Shannon Divergence [Goodfellow et al., 2014]. Fungsi tersebut dituliskan sebagai berikut:

$$\begin{aligned} C(G) &= -\log(4) + KL(p_{data}(\boldsymbol{x})||\frac{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}) + KL(p_g(\boldsymbol{x})||\frac{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}) \\ &= -\log(4) + 2\,JSD\,(p_{data}(\boldsymbol{x}) \,||\, p_g(\boldsymbol{x})) \end{aligned} \tag{3.4}$$

dengan $KL(p_{data}(\boldsymbol{x})||\frac{p_{data}(\boldsymbol{x})+p_g(\boldsymbol{x})}{2})$ menyatakan fungsi Kullback–Leibler Divergence dari distribusi $\frac{p_{data}(\boldsymbol{x})+p_g(\boldsymbol{x})}{2}$ menuju distribusi $p_{data}(\boldsymbol{x})$, $KL(p_g(\boldsymbol{x})||\frac{p_{data}(\boldsymbol{x})+p_g(\boldsymbol{x})}{2})$ menyatakan fungsi Kullback-Leibler Divergence dari distribusi $\frac{p_{data}(\boldsymbol{x})+p_g(\boldsymbol{x})}{2}$ menuju distribusi $p_g(\boldsymbol{x})$, dan $JSD\,(p_{data}(\boldsymbol{x}) \,||\, p_g(\boldsymbol{x}))$ menyatakan fungsi Jensen-Shannon Divergence antara distribusi $p_{data}(\boldsymbol{x})$ dan $p_g(\boldsymbol{x})$.

3.1.1.2 Konvergensi Algoritma *Generative Adversarial Network*

Proposisi 2 [Goodfellow et al., 2014] Jika:

- Generator G dan diskriminator D memiliki kapasitas yang cukup
- Pada setiap iterasi **Algoritma 4** diskriminator mampu mencapai titik optimum untuk suatu generator G
- Nilai p_g diperbarui untuk mengoptimisasi fungsi objektif

$$\mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D^*_G(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(1 - D^*_G(\boldsymbol{x}))] \tag{3.5}$$

maka p_g akan konvergen terhadap p_{data}.

Dalam praktiknya, GAN hanya mampu mengestimasi p_g secara terbatas melalui fungsi $G(\mathbf{z};\theta_g)$ sehingga GAN tidak mengoptimisasi p_g, namun mengoptimisasi θ_g [Goodfellow et al., 2014].

3.2 Least Square Generative Adversarial Network

Least square generative adversarial network (selanjutnya dapat pula ditulis sebagai LSGAN) merupakan model *generative adversarial network* yang mengadopsi fungsi objektif pada *least square* sebagai fungsi objektif pada diskriminator dan generatornya [Mao et al., 2017]. Fungsi objektif *least square* yang umum digunakan adalah *mean squared error* (MSE), yang mengukur rata - rata dari L^2 *loss* [Charnes et al., 1976]. *Mean squared error* merupakan fungsi yang menghitung rata - rata dari penjumlahan hasil kudrat dari pengurangan nilai hasil prediksi terhadap nilai target. Fungsi objektif pada metode *least square* dapat menggeser data yang dihasilkan oleh generator G menuju fungsi batas *least square*. Proses tersebut dilakukan dengan memberi penalti pada data generator yang berada cukup jauh dari fungsi batas [Mao et al., 2017]. Secara matematis, fungsi objektif dari diskriminator dan generator pada LSGAN adalah:

$$\min_D V_{LSGAN}(D) = \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[(D(\boldsymbol{x})-b)^2] + \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{data}(\boldsymbol{z})}[(D(G(\boldsymbol{z}))-a)^2] \tag{3.6}$$

$$\min_G V_{LSGAN}(G) = \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{data}(\boldsymbol{z})}[(D(G(\boldsymbol{z}))-c)^2] \tag{3.7}$$

dengan a dan b merupakan *hyperparameter* untuk sampel data yang dihasilkan generator dan sampel data yang digunakan selama pembelajaran. Nilai c merupakan batas nilai yang digunakan oleh generator G untuk memaksimalkan fungsi objektif diskriminator D. Penjelasan mengenai nilai a, b, dan c dijelaskan pada salah satu bagian dari subbab ini.

3.2.1 Hubungan LSGAN terhadap *F-Divergence*

Menurut teori probabilitas, *f-divergence* merupakan sebuah fungsi $D_f(P||Q)$ yang mengukur perbedaan distribusi probabilitas antara P dan Q [Nielsen and Nock, 2013]. Terdapat beberapa fungsi yang dapat digunakan untuk mengukur *f-divergence* diantaranya Kullback-Leibler Divergence [Kullback and Leibler, 1951, Kullback, 1997], Pearson χ^2 Divergence [Pearson, 1900], dan Jensen-Shannon Divergence [Manning et al., 1999, Dagan et al., 1997]. Pada bagian sebelumnya dijelaskan bahwa fungsi objektif pada Vanilla GAN (penelitian awal) ekuivalen dengan meminimalkan fungsi Jensen-Shannon

Divergence (**Persamaan 3.5**), yaitu:

$$C(G) = -\log(4) + KL(p_{data}(\boldsymbol{x})||\frac{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}) + KL(p_g(\boldsymbol{x})||\frac{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2})$$

Bagian ini menunjukkan hubungan antara LSGAN dan *f-divergence*. **Persamaan 3.6** dan **Persamaan 3.7** dapat dituliskan sebagai:

$$\min_D V_{LSGAN}(D) = \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[(D(\boldsymbol{x}) - b)^2] + \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{data}(\boldsymbol{z})}[(D(G(\boldsymbol{z})) - a)^2]$$

$$\min_G V_{LSGAN}(G) = \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[(D(\boldsymbol{x}) - c)^2] + \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{data}(\boldsymbol{z})}[(D(G(\boldsymbol{z})) - c)^2] \tag{3.8}$$

Penambahan nilai $\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[(D(x) - c)^2]$ pada fungsi objektif generator tidak mengubah nilai optimum fungsi objektif karena tidak mengandung parameter G [Mao et al., 2017]. Selanjutnya, nilai optimal diskriminator D terhadap generator G adalah:

$$D^*_G(\boldsymbol{x}) = \frac{b\, p_{data}(\boldsymbol{x}) + a\, p_g(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \tag{3.9}$$

Persamaan 3.8 yang dikalikan dengan 2 dapat dituliskan kembali menjadi:

$$\begin{aligned}
2C(G) &= \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[(D^*(\boldsymbol{x}) - c)^2] + \mathbb{E}_{\boldsymbol{x}\sim p_g(\boldsymbol{x})}[(D^*(\boldsymbol{x}) - c)^2] \\
&= \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[(\frac{b\, p_{data}(\boldsymbol{x}) + a\, p_g(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} - c)^2] + \mathbb{E}_{\boldsymbol{x}\sim p_g(\boldsymbol{x})}[(\frac{b\, p_{data}(\boldsymbol{x}) + a\, p_g(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} - c)^2] \\
&= \int_{\boldsymbol{x}} p_{data}(\boldsymbol{x})(\frac{(b-c)\, p_{data}(\boldsymbol{x}) + (a-c)\, p_g(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})})^2 d\boldsymbol{x} + \\
&\quad \int_{\boldsymbol{x}} p_g(\boldsymbol{x})(\frac{(b-c)\, p_{data}(\boldsymbol{x}) + (a-c)\, p_g(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})})^2 \\
&= \int_{\boldsymbol{x}} p_{data}(\boldsymbol{x}) \frac{((b-c)\, p_{data}(\boldsymbol{x}) + (a-c)\, p_g(\boldsymbol{x}))^2}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} d\boldsymbol{x} \\
&= \int_{\boldsymbol{x}} p_{data}(\boldsymbol{x}) \frac{((b-c)\,(p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})) - (b-a)\, p_g(\boldsymbol{x}))^2}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} d\boldsymbol{x}
\end{aligned} \tag{3.10}$$

dengan *a, b,* dan *c* memenuhi persamaan *b* - *c* = 1 dan *b* - *a* = 2, maka:

$$\begin{aligned}
2C(G) &= \int_{\boldsymbol{x}} \frac{(2\, p_g(\boldsymbol{x}) - (p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})))^2}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} d\boldsymbol{x} \\
&= \chi^2_{Pearson}(p_{data} + p_g || 2\, p_g)
\end{aligned} \tag{3.11}$$

dengan $\chi^2_{Pearson}$ merupakan nilai Pearson χ^2 Divergence. Meminimalkan **Persamaan 3.8** sama dengan meminimalkan nilai Pearson χ^2 Divergence antara $p_g(\boldsymbol{x}) + p_{data}(\boldsymbol{x})$ dengan $2\,p_g(\boldsymbol{x})$ jika *a,b,* dan c memenuhi $b-c=1$ dan $b-a=2$. **Persamaan 3.11** yang menunjukkan bahwa *least square* GAN dapat meminimalkan fungsi *f-divergence* memberi justifikasi bahwa LSGAN dapat melakukan pembelajaran sebagai model pembangkit. Pemilihan nilai a dan b dijelaskan pada bagian selanjutnya.

3.2.2 Pemilihan Parameter pada Least Square Generative Adversarial Network

Pemilihan parameter berfokus pada pemilihan parameter a, b, dan c yang dinyatakan pada fungsi objektif LSGAN. Salah satu cara untuk mendapatkan nilai *a, b,* dan c adalah dengan menyelesaikan persamaan $b-c=1$ dan $b-a=2$ yang berimplikasi meminimalkan **Persamaan 3.6** dan **Persamaan 3.7**. Solusi yang umum digunakan adalah dengan memilih $a=-1$, $b=1$, dan $c=0$ yang menghasilkan fungsi objektif sebagai berikut:

$$\min_D V_{LSGAN}(D) = \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[(D(\boldsymbol{x})-1)^2] + \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{data}(\boldsymbol{z})}[(D(G(\boldsymbol{z}))+1)^2] \tag{3.12}$$

$$\min_G V_{LSGAN}(G) = \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{data}(\boldsymbol{z})}[(D(G(\boldsymbol{z})))^2] \tag{3.13}$$

Cara lainnya adalah dengan memilih nilai $b=c$. Misalnya dengan memilih nilai $a=0$, didapat $b=c=1$, sehingga:

$$\min_D V_{LSGAN}(D) = \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[(D(\boldsymbol{x})-1)^2] + \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{data}(\boldsymbol{z})}[(D(G(\boldsymbol{z})))^2] \tag{3.14}$$

$$\min_G V_{LSGAN}(G) = \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{data}(\boldsymbol{z})}[(D(G(\boldsymbol{z}))-1)^2] \tag{3.15}$$

BAB 4
LEAST SQUARE ADVERSARIAL AUTOENCODER

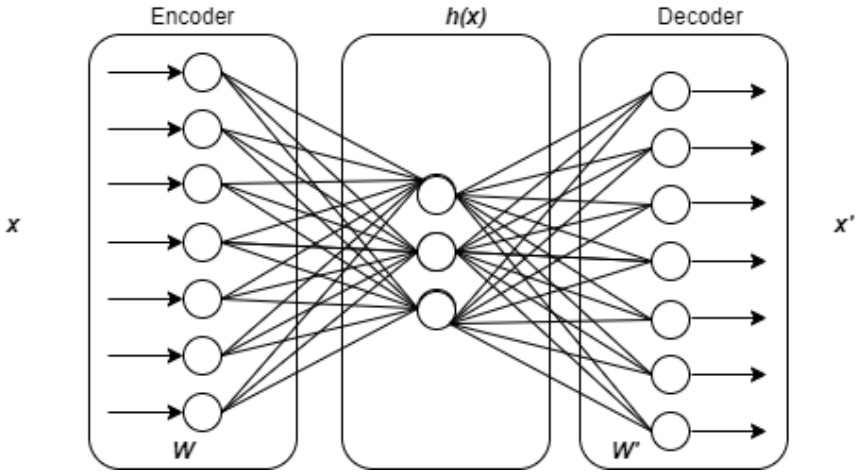
Bab ini menjelaskan *autoencoder* secara umum, *adversarial autoencoder* sebagai salah satu model *autoencoder*, *least square adversarial autoencoder*, dan standar pengukuran *least square adversarial autoencoder* (LSAA). Pembahasan mengenai LSAA dibagi menjadi dua yaitu *unsupervised* LSAA dan *supervised* LSAA.

4.1 *Autoencoder*

Pada bagian ini dijelaskan mengenai pengertian *autoencoder* dan model *undercomplete autoencoder*.

4.1.1 Pengertian Autoencoder

Autoencoder merupakan suatu model jaringan saraf tiruan yang dilatih sehingga mampu mentranformasi masukan ke dalam suatu representasi dan merekonstruksi masukan yang diberikan berdasarkan represntasi tersebut. *Autoencoder* memiliki lapisan tersembunyi $\boldsymbol{h}$ yang berfungsi untuk menyimpan variabel laten dari masukan. Variabel laten merupakan variabel yang tidak dapat diamati secara langsung melainkan disimpulkan berdasarkan distribusi variabel yang lain. *Autoencoder* dapat dipandang sebagai dua buah komponen yaitu *encoder* dan *decoder*. *Encoder* merupakan suatu fungsi yang memetakan masukan $\boldsymbol{x} \in \mathbb{R}^d$ menuju suatu keluaran $\boldsymbol{h} \in \mathbb{R}^k$ ($\boldsymbol{h} = f(\boldsymbol{x})$). Sementara itu, *decoder* merupakan fungsi yang memetakan masukan $\boldsymbol{h} \in \mathbb{R}^k$ menuju keluaran $\boldsymbol{r} \in \mathbb{R}^d$ ($\boldsymbol{r} = g(\boldsymbol{h})$) [Shalev-Shwartz and Ben-David, 2014]. Keluaran yang dihasilkan oleh *encoder* merupakan fitur yang berguna untuk meningkatkan efisiensi proses belajar. **Gambar 4.1** menunjukkan skema dari sebuah autoencoder yang terdiri dari *encoder* dan *decoder*.



Gambar 4.1: Skema *autoencoder*.

4.1.2 *Undercomplete* Autoencoder

Undercomplete autoencoder merupakan model *autoencoder* dimana dimensi lapisan tersembunyi lebih kecil dibandingkan dimensi masukannya. Hal ini dimaksudkan agar *autoencoder* tersebut hanya mempelajari fitur yang paling penting dari distribusi data yang diberikan. Proses belajar pada model ini adalah dengan meminimalkan nilai fungsi objektif

$$\boldsymbol{L}(\boldsymbol{x}, g(\boldsymbol{x})) \tag{4.1}$$

Fungsi objektif berfungsi untuk mendapatkan nilai yang menyatakan *distance* antara data $\boldsymbol{x}$ dan keluaran yang dihasilkan oleh *undercomplete autoencoder* $g(f(\boldsymbol{x}))$. Dimensi variabel laten yang lebih kecil daripada dimensi data masukan memberi kemampuan pada *autoencoder* untuk mengestimasi distribusi sampel pembelajaran. Dimensi variabel laten yang terlalu tinggi menyebabkan proses rekonstruksi pada *autoencoder* bekerja seperti fungsi identitas, meskipun rekonstruksi datanya baik[Goodfellow et al., 2016].

4.2 Adversarial Autoencoder

Adversarial autoencoder (AA) merupakan *autoencoder* yang memiliki jaringan diskriminator. Jaringan diskriminator berfungsi untuk membedakan antara variabel laten yang merupakan hasil *encoding* dan sampel *noise* yang berasal dari distribusi apriori. Distribusi apriori dapat berupa distribusi apapun [Makhzani et al., 2015]. Jaringan diskriminator pada AA bekerja secara adversarial dengan *encoder*. Karena variabel laten yang dihasilkan oleh *encoder* menjadi masukan bagi diskriminator, maka *encoder* pada AA dapat pula disebut sebagai generator. Jaringan *decoder* tidak dipergunakan sebagai

generator karena dimensi keluaran yang dihasilkan tidak sesederhana dimensi variabel laten (dimensinya sama dengan dimensi masukan). Jaringan diskriminator dan generator yang bekerja secara adversarial memberi kemampuan bagi *autoencoder* sebagai model pembangkit. Jaringan diskriminator dan jaringan *encoder* yang bekerja secara adversarial dapat dipandang sebagai GAN. *Adversarial autoencoder* yang telah selesai belajar dapat digunakan untuk merekonstruksi data dan melakukan pembangkitan data yang memiliki karakteristik yang mirip dengan sampel pembelajaran. Proses pembangkitan pada AA dilakukan dengan melakukan *sampling* terhadap *noise* yang kemudian diteruskan menuju *decoder*.

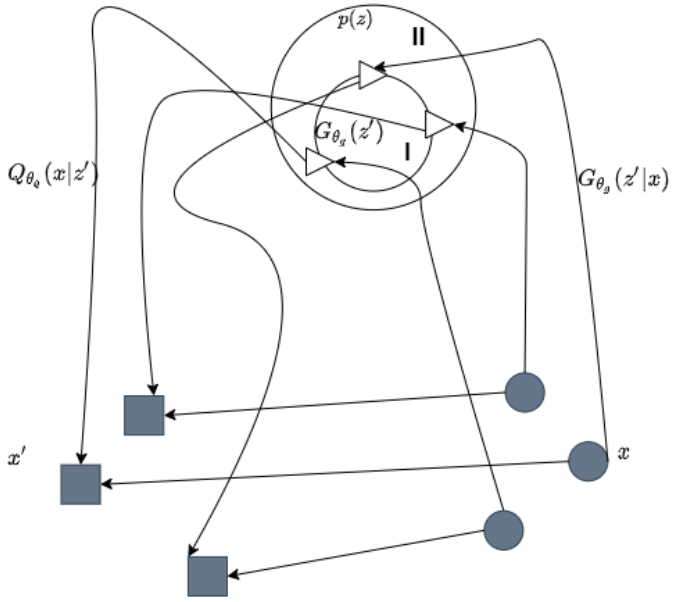
Adversarial autoencoder memiliki tiga jaringan yaitu *encoder* / generator G_{θ_g}, *decoder* Q_{θ_q}, dan diskriminator D_{θ_d}, dengan θ_g, θ_q, dan θ_d masing-masing merupakan parameter pada *encoder*, *decoder*, dan diskriminator. Masing-masing parameter tersebut merepresentasikan *weight* pada jaringan *encoder*, *decoder*, dan diskriminator. Proses pembelajaran pada AA dilakukan dengan mengoptimisasi nilai parameter pada ketiga jaringan tersebut. Proses pembelajaran *encoder*, *decoder*, dan diskriminator pada AA adalah sebagai berikut.

- Pembelajaran yang pertama dilakukan terhadap *encoder* G_{θ_g} dan *decoder* Q_{θ_q}. Pembelajaran dilakukan untuk meminimalkan *reconstruction error* antara data asli $\boldsymbol{x}$ dan keluaran yang dihasilkan oleh *decoder*.

- Pembelajaran yang kedua dilakukan terhadap diskriminator D_{θ_d}. Pembelajaran pada diskriminator meminimalkan probabilitas gagal untuk mengestimasi probabilitas data berasal dari distribusi apriori $p_{\boldsymbol{z}}(\boldsymbol{z})$. Optimisasi dilakukan dengan melibatkan sampel *noise* $\boldsymbol{z}$ dari distribusi apriori dan sampel variabel laten $\boldsymbol{z'}$ yang berasal dari generator ($G_{\theta_g}(\boldsymbol{x})$). Optimisasi dilakukan untuk memaksimalkan kemampuan diskriminator membedakan kedua data tersebut.

- Pembelajaran yang terakhir dilakukan terhadap generator G_{θ_g}. Pembelajaran pada generator memaksimalkan probabilitas diskriminator gagal mengestimasi probabilitas data yang berasal dari distribusi apriori $p_{\boldsymbol{z}}(\boldsymbol{z})$. Optimisasi dilakukan berdasarkan variabel laten $\boldsymbol{z'}$ yang merupakan hasil *encoding* sampel pembelajaran ($G_{\theta_g}(\boldsymbol{z'}|\boldsymbol{x})$). Optimisasi dilakukan untuk meningkatkan kemampuan generator menghasilkan keluaran yang memiliki karakteristik yang mirip dengan sampel yang dihasilkan oleh distribusi apriori $p_{\boldsymbol{z}}(\boldsymbol{z})$.

Gambar 4.2 menunjukkan skema AA [Tolstikhin et al., 2017].



Gambar 4.2: Skema *Adversarial Autoencoder*.

Data sampel $\boldsymbol{x}$ dilambangkan oleh lingkaran hitam. Proses *encoding* data $\boldsymbol{x}$ melalui fungsi *encoder* $G_{\theta_g}(\boldsymbol{z'}|\boldsymbol{x})$ menghasilkan variabel laten $\boldsymbol{z'}$ yang mengikuti distribusi $G_{\theta_g}(\boldsymbol{z'})$. Distribusi $G_{\theta_g}(\boldsymbol{z'})$ dilambangkan oleh lingkaran I. Distribusi apriori $p_{\boldsymbol{z}}(\boldsymbol{z})$ dilambangkan oleh lingkaran II. Diskriminator berusaha meminimalkan *f-divergence* dari kedua distribusi tersebut. Proses *decoding* variabel laten $\boldsymbol{z'}$ melalui fungsi *decoder* $Q_{\theta_q}(\boldsymbol{x}|\boldsymbol{z'})$ menghasilkan data rekonstruksi $\boldsymbol{x'}$. Hasil rekonstruksi $\boldsymbol{x'}$ dilambangkan oleh persegi hitam. *Adversarial autoencoder* memiliki kemiripan dengan *variational autoencoder* yaitu dengan meregularisasi variabel laten. *Variational autoencoder* meregularisasi variabel laten dengan menggunakan metode inferensi variasional [Kingma and Welling, 2013]. *Variational autoencoder* memiliki kemampuan untuk melakukan rekonstruksi dan membangkitkan citra.

Adversarial autoencoder memiliki tiga jenis arsitektur [Makhzani et al., 2015], yaitu sebagai berikut.

- ***Unsupervised AA***
  Arsitektur ini tidak memanfaatkan label atau target dari data sampel ketika proses belajar berlangsung.

- ***Supervised AA***
  Arsitektur *Supervised* AA memanfaatkan label dari data sampel ketika proses belajar berlangsung.

- ***Semisupervised adversarial autoencoder***
  Arsitektur ini merupakan kombinasi dari *supervised* AA dan *unsupervised* AA.

Sebagian sampel pada *semisupervised* AA diberi label atau target dan sebagian lagi tidak.

Adversarial autoencoder dapat dimanfaatkan untuk melakukan reduksi dimensi, visualisasi data, klasifikasi berbasis *semisupervised*, maupun menguraikan data pada ruang *manifold* yang berdimensi lebih rendah [Makhzani et al., 2015].

Penelitian ini mencoba mengetahui efek penggunaan *LSGAN* pada jaringan adversarial autoencoder (jaringan diskriminator). Selanjutnya, *adversarial autoencoder* tersebut dinamakan sebagai *least square adversarial autoencoder* (LSAA). Penelitian ini dilatarbelakangi bahwa penelitian mengenai GAN memiliki beberapa arsitektur. Beberapa varian pada GAN diantaranya adalah Wasserstein GAN [Arjovsky et al., 2017], *Energy Based* GAN [Zhao et al., 2016], dan LSGAN [Mao et al., 2017]. Selanjutnya, terdapat penelitian yang mencoba mengetahui efek penggunaan arsitektur GAN yang berbeda pada AA. Salah satu penelitian yang dilakukan adalah dengan menggunakan Wasserstein *autoencoder* [Tolstikhin et al., 2017]. Pada penelitian ini, digunakan LSGAN pada dua jenis arsitektur AA yaitu arsitektur *unsupervised* AA dan arsitektur *supervised* AA.

4.3 *Unsupervised Least Adversarial Autoencoder*

Unsupervised LSAA merupakan model AA yang tidak mempergunakan label atau target dari sampel pada saat proses pembelajarannya. Model ini terdiri dari tiga bagian yaitu *encoder* yang juga bertugas sebagai generator G_{θ_g}, diskriminator D_{θ_d}, dan *decoder* Q_{θ_q}. Jaringan diskriminator pada LSAA menggunakan fungsi objektif yang sama dengan LSGAN.

Proses pembelajaran pada jaringan *encoder, decoder,* dan diskriminator adalah sebagai berikut.

- **Diskriminator** D_{θ_d}

  Proses pembelajaran pada diskriminator meminimalkan fungsi objektif berupa nilai MSE dari sampel noise $\boldsymbol{z}$ dan *hyperparameter* a ditambah dengan nilai MSE dari hasil *encoding* data sampel (variabel laten) $G_{\theta_g}(\boldsymbol{z'}|\boldsymbol{x})$ dan *hyperparameter* b. *Hyperparameter* a dan b merupakan hyperparameter yang berfungsi untuk membedakan data yang berasal dari distribusi apriori dan data yang merupakan variabel laten. Distribusi apriori yang digunakan adalah distribusi normal dengan *mean* bernilai 0 dan standar deviasi bernilai 1. Pemilihan nilai *hyperparameter* a dan b menggunakan cara yang sama dengan pemilihan *hyperparameter* pada LSGAN. Pada penelitian ini dipilih nilai $a = 1$ dan $b = 0$. Selanjutnya dilakukan pemberian penalti pada *hyperparameter* a dan b sehingga $a = 0.9$ dan $b = 0.1$.

Pemberian penalti dilakukan untuk menghindari kemungkinan terjadinya *vanishing gradient* [Salimans et al., 2016]. Selain itu, terdapat *hyperparameter* λ yang menentukan skala dari fungsi objektif pada diskriminator.

- **Generator / *Encoder*** G_{θ_g}
  Generator sekaligus *encoder* G_{θ_g} memiliki dua proses pembelajaran. Proses pembelajaran yang pertama adalah untuk mendapatkan variabel laten $\boldsymbol{z'}$ dari hasil *encoding* data sampel $\boldsymbol{x}$ ($G_{\theta_g}(\boldsymbol{z'}|\boldsymbol{x})$) sedemikian sehingga nilai (*reconstruction error*) antara data $\boldsymbol{x}$ dan hasil rekonstruksinya menjadi minimum. Proses pembelajaran pada *encoder* mengoptimisasi fungsi objektif yang sama dengan *decoder*. Proses pembelajaran yang kedua adalah untuk memaksimalkan fungsi objektif diskriminator D_{θ_d}. Proses pembelajaran pada generator ekuivalen dengan meminimalkan fungsi Pearson χ^2 Divergence antara distribusi variabel laten $G_{\theta_g}(\boldsymbol{z'})$ dan distribusi apriori $p_{\boldsymbol{z}}(\boldsymbol{z})$. Fungsi objektif pada generator G_{θ_g} berupa nilai MSE antara variabel laten $\boldsymbol{z'}$ dan *hyperparameter* c. Pemilihan *hyperparameter* c sama dengan pemilihan *hyperparameter* c pada LSGAN. Pada penelitian ini, dipilih $c = 0$ dengan pemberian penalti sehingga $c = 0.9$. Selain itu, terdapat *hyperparameter* λ yang menentukan skala dari fungsi objektif pada generator.

- ***Decoder*** Q_{θ_q}
  Pembelajaran pada *decoder* Q_{θ_q} meminimalkan fungsi *reconstruction error* antara hasil *decoding* terhadap data asli. Penelitian ini menggunakan MSE antara data asli $\boldsymbol{x}$ dan hasil rekonstruksi $\boldsymbol{x'}$ ($Q_{\theta_q}(\boldsymbol{x'}|\boldsymbol{z})$) sebagai fungsi objektif pada *decoder*. Jaringan *encoder* dan *decoder* mengoptimisasi fungsi objektif tersebut.

Algoritma 5 menunjukkan proses pembelajaran pada *unsupervised* LSAA:

Algoritma 5 Algoritma Pembelajaran pada *Unsupervised* LSAA

Require:

- *Hyperparameter* $\lambda > 0$
- Nilai awal parameter θ_g, parameter θ_d, dan parameter θ_q

while (θ_q, θ_g, θ_d) belum mencapai titik konvergen **do**

- *Sampling minibatch* pada data $\boldsymbol{x}$ sebanyak n $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, ..., \boldsymbol{x}^{(n)}\}$ dari sampel pembelajaran $p_{data}(\boldsymbol{x})$
- *Sampling minibatch* pada *noise* $\boldsymbol{z}$ sebanyak n $\{\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, ..., \boldsymbol{z}^{(n)}\}$ dari distribusi apriori $p_z(\boldsymbol{z})$
- Cari variabel laten $\boldsymbol{z'}^{(i)}$ dari $G_{\theta_g}(\boldsymbol{z'}^{(i)}|\boldsymbol{x}^{(i)})$ untuk i = 1, 2, ..., n
- Perbarui nilai θ_d dengan meminimalkan fungsi berikut (menggunakan metode adam):

$$\frac{\lambda}{n}\sum_{i=1}^{n}(D_{\theta_d}(\boldsymbol{z}^{(i)}) - a)^2 + \frac{\lambda}{n}\sum_{i=1}^{n}(D_{\theta_d}(\boldsymbol{z'}^{(i)}) - b)^2$$

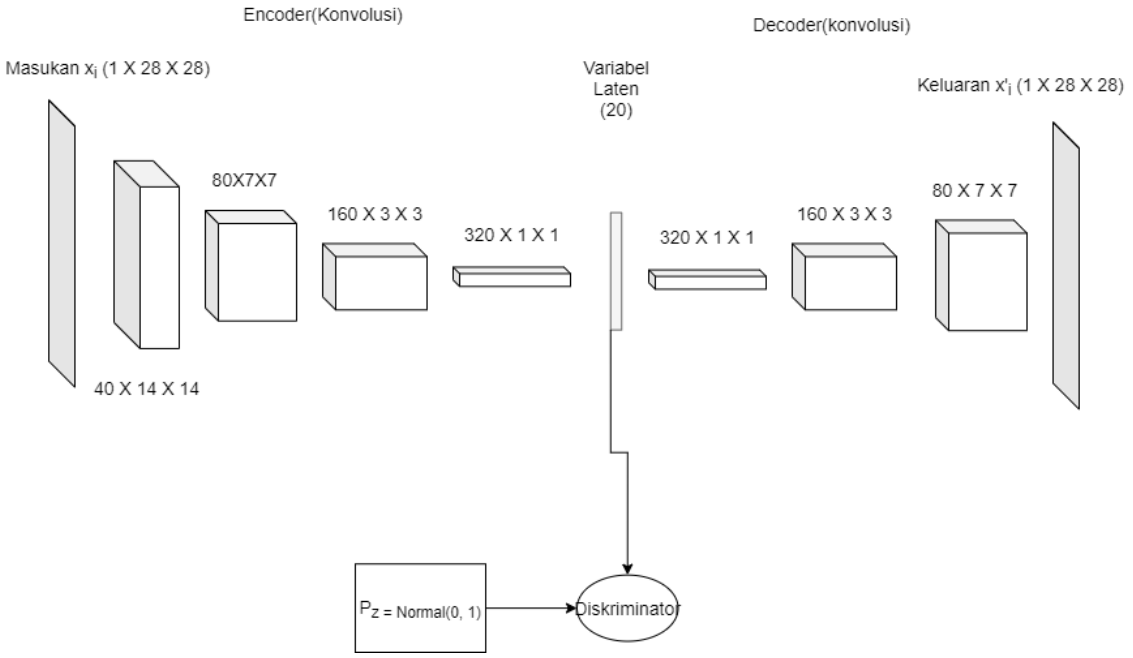
- Perbarui parameter θ_g dengan meminimalkan fungsi berikut (menggunakan metode adam):

$$\frac{\lambda}{n}\sum_{i=1}^{n}(G_{\theta_g}(\boldsymbol{z'}^{(i)}) - c)^2$$

- Perbarui parameter θ_q (parameter θ_g juga ikut diperbarui) dengan meminimalkan fungsi berikut (menggunakan metode adam):

$$\frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}^{(i)} - Q_{\theta_q}(w^{(i)}))^2$$

Pada penelitian ini, *unsupervised* LSAA diimplementasi menggunakan jaringan konvolusional. Jaringan konvolusional mampu mempelajari fitur secara lebih akurat untuk data yang memiliki topologi seperti *grid*. Jaringan konvolusional diharapkan dapat melakukan proses *encoding* dan *decoding* dengan baik meskipun tanpa menggunakan label. Jaringan konvolusional diterapkan pada *encoder* dan *decoder*. Sementara untuk jaringan diskriminator menggunakan jaringan saraf tiruan biasa. Diskriminator menggunakan jaringan saraf tiruan biasa karena variabel laten memiliki dimensi yang jauh lebih rendah bila dibandingkan dengan dimensi masukan. **Gambar 4.3** menunjukkan arsitektur jaringan *unsupervised* LSAA



Gambar 4.3: Arsitektur Jaringan *Unsupervised* LSAA.

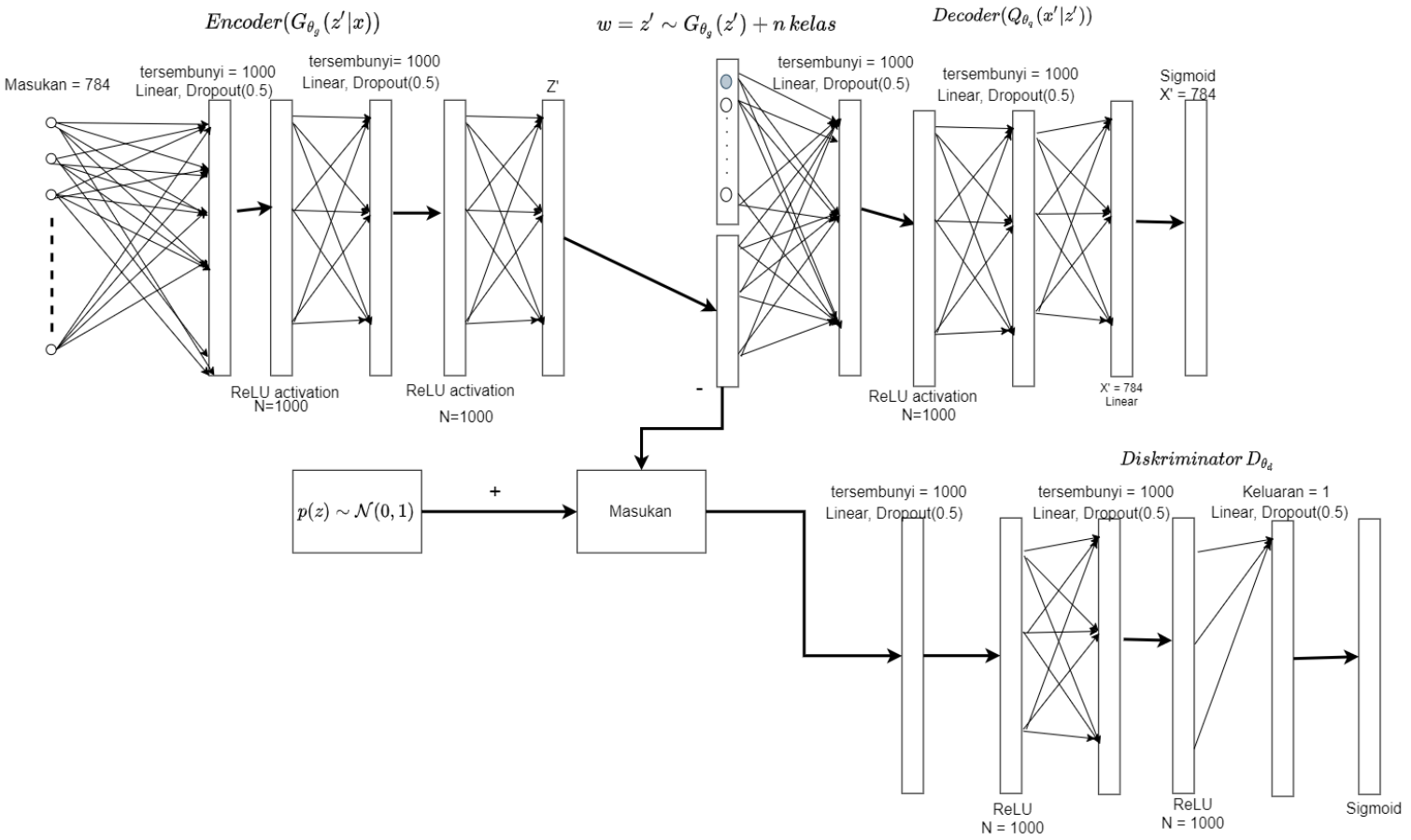
Jaringan *encoder* diimplementasi menggunakan empat buah lapisan konvolusional yang masing-masing menggunakan kernel berukuran 4×4 sejumlah 40 kernel, 4×4 sejumlah 80 kernel, 4×4 sejumlah 160 kernel dan 4×4 sejumlah 320 kernel. Ukuran masukan pada tiap lapisan konvolusional mengecil seiring dengan operasi konvolusi yang dilakukan. Jaringan *decoder* diimplementasikan menggunakan tiga buah lapisan konvolusional yang masing-masing menggunakan kernel sejumlah 4×4 sejumlah 320 kernel, 4×4 sejumlah 160 kernel, dan 4×4 sejumlah 80 kernel. Ukuran masukan pada tiap lapisan konvolusional membesar seiring dengan operasi konvolusi yang dilakukan. Hal tersebut dikarenakan terjadi proses interpolasi pada data untuk merekonstruksi data asli. Implementasi lengkap mengenai *unsupervised AA* dapat dilihat pada Lampiran 1.

4.4 *Supervised Least Square Adversarial Autoencoder*

Supervised LSAA merupakan jaringan LSAA yang memanfaatkan label dari data $\boldsymbol{x}$, yang disebut $\boldsymbol{y}$, dengan $\boldsymbol{x}$ merupakan data yang berasal dari sampel pembelajaran. Label pada data dimanfaatkan untuk membantu proses rekonstruksi pada sampel. Label pada data sampel dikonkatenasi dengan variabel laten $\boldsymbol{z'}$ menghasilkan representasi $\boldsymbol{w}$, dengan variabel laten $\boldsymbol{z'}$ merupakan hasil *encoding* data sampel. Representasi $\boldsymbol{w}$ menjadi masukan bagi *decoder* untuk merekonstruksi data sampel. Label pada data sampel direpresentasikan dalam bentuk *one hot vector*. Misalkan jumlah label atau target pada sampel pembelajaran adalah m dan dimensi variabel laten pada *autoencoder* berukuran n. Representasi label dalam *one hot vector* merupakan vektor berukuran m. Ukuran dimensi

setelah proses konkatenasi antara variabel laten $\boldsymbol{z'}$ dan label $\boldsymbol{y}$ ($\boldsymbol{w} = concat(\boldsymbol{z'}, \boldsymbol{y})$) adalah $m+n$. Secara intuitif, komponen label pada vektor laten yang baru dapat membedakan kelas antar sampel.

Arsitektur *supervised* LSAA pada penelitian ini menggunakan jaringan saraf tiruan *fully connected* (jaringan saraf tiruan biasa). Arsitektur tidak menggunakan jaringan konvolusional untuk mengetahui efek penggabungan label dengan variabel laten. *supervised* LSAA terdiri dari tiga jaringan yaitu *encoder* yang juga berperan sebagai generator G_{θ_g}, *decoder* Q_{θ_q}, dan diskriminator D_{θ_d}, dengan θ_g, θ_q, dan θ_d masing-masing merupakan parameter pada *encoder*, *decoder*, dan diskriminator. *Encoder*, *decoder*, dan diskriminator terdiri dari tiga lapisan tersembunyi. Lapisan tersembunyi pada *encoder* masing-masing memiliki 1000 perseptron, 1000 perseptron, dan 2 perseptron. Lapisan tersembunyi pada *decoder* masing -masing memiliki 1000 perseptron, 1000 perseptron, dan 784 perseptron. Diskriminator memiliki lapisan tersembunyi yang masing-masing memiliki 1000 perseptron, 1000 perseptron, dan 1 perseptron. Semua lapisan tersembunyi pada *encoder*, *decoder*, dan diskriminator menggunakan fungsi aktivasi ReLU, kecuali pada lapisan terakhir *decoder* dan diskriminator yang menggunakan fungsi sigmoid. Sebelum melalui fungsi aktivasi, nilai keluaran pada lapisan tersembunyi diregularisasi menggunakan *dropout*. **Gambar 4.4** menunjukkan arsitektur *supervised* LSAA.



Gambar 4.4: Arsitektur *Supervised* LSAA.

Perbedaan proses pembelajaran pada *supervised* LSAA dengan pembelajaran pada *unsupervised* LSAA terletak pada representasi variabel laten. *Supervised* LSAA memanfaatkan label data sampel pada representasi variabel latennya sedangkan *unsupervised* LSAA tidak. **Algoritma 6** menunjukkan proses pembelajaran pada *supervised* LSAA.

Algoritma 6 Algoritma Pembelajaran pada *Supervised* LSAA

Require:

- *Hyperparameter* $\lambda > 0$, nilai awal parameter *encoder* / generator G_{θ_g}, parameter diskriminator D_{θ_d}, dan parameter *decoder* Q_{θ_q}

while (θ_q, θ_g, θ_d) belum mencapai titik konvergen **do**

- *Sampling minibatch* pada data $\boldsymbol{x}$ beserta dengan label $\boldsymbol{y}$ sebanyak n $\{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), (\boldsymbol{x}^{(2)}, \boldsymbol{y}^{(2)}), ..., (\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)})\}$ dari sampel pembelajaran $p_{data}(\boldsymbol{x})$
- *Sampling minibatch* pada *noise* $\boldsymbol{z}$ sebanyak n $\{\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, ..., \boldsymbol{z}^{(n)}\}$ dari distribusi apriori $p_{\boldsymbol{z}}(\boldsymbol{z})$
- Cari variabel laten $\boldsymbol{z'}^{(i)}$ dari $G_{\theta_g}(\boldsymbol{z'}^{(i)}|\boldsymbol{x}^{(i)})$ untuk i = 1, 2, ..., n
- Cari $\boldsymbol{w}^{(i)}$ dengan melakukan konkatenasi antara $\boldsymbol{z'}^{(i)}$ dan $\boldsymbol{y}^{(i)}$; $\boldsymbol{w}^{(i)} = concat(\boldsymbol{z'}^{(i)}, \boldsymbol{y}^{(i)})$ untuk i = 1, 2, ..., n
- Perbarui parameter θ_q (parameter θ_g juga ikut diperbarui) dengan meminimalkan fungsi berikut (menggunakan metode adam):

$$\frac{1}{n} \sum_{i=1}^{n} (\boldsymbol{x}^{(i)} - Q_{\theta_q}(\boldsymbol{w}^{(i)}))^2$$

- Perbarui parameter θ_d dengan meminimalkan fungsi berikut (menggunakan metode adam):

$$\frac{\lambda}{n} \sum_{i=1}^{n} (D_{\theta_d}(\boldsymbol{z}^{(i)}) - a)^2 + \frac{\lambda}{n} \sum_{i=1}^{n} (D_{\theta_d}(\boldsymbol{z'}^{(i)} - b))^2$$

- Perbarui parameter θ_g dengan meminimalkan fungsi berikut (menggunakan metode adam):

$$\frac{\lambda}{n} \sum_{i=1}^{n} (G_{\theta_g}(\boldsymbol{z'}^{(i)}) - c)^2$$

BAB 5
EKSPERIMEN DAN ANALISIS HASIL EKSPERIMEN

Bab ini menjelaskan mengenai teknologi yang digunakan, *dataset* yang digunakan, eksperimen yang dilakukan, dan analisis terhadap eksperimen yang telah dilakukan. Penjelasan mengenai eksperimen dibagi menjadi dua bagian yaitu *unsupervised* LSAA dan *supervised* LSAA.

5.1 Teknologi

Seluruh eksperimen pada penelitian ini dilakukan pada *environment cloud* Floydhub dan Tokopedia-UI AI Center. Layanan *cloud* mendukung sumber daya komputasi yang tinggi yang dibutuhkan dalam penelitian ini. Sumber daya komputasi layanan tersebut berupa GPU yang memungkinkan proses komputasi dilakukan secara paralel sehingga dapat mempersingkat waktu komputasi. Berikut adalah *software tools* dan perangkat komputasi yang dipergunakan pada penelitian ini.

- PyTorch
  PyTorch merupakan *framework* yang berguna untuk membangun arsitektur *machine learning* [PyTorch, 2020]. *Framework* ini menggunakan bahasa pemrograman Python sebagai basis kodenya [Kuhlman, 2012]. Struktur data pada Pytorch direpresentasikan dalam bentuk *tensor*. PyTorch memiliki fitur *autograd* yang berfungsi menghitung gradien secara otomatis. PyTorch juga mendukung komputasi dengan menggunakan GPU. Perhitungan dengan GPU pada PyTorch memanfaatkan cuda, yaitu *application programming interface* yang dikembangkan oleh NVIDIA. Penelitian ini menggunakan PyTorch versi 1.4.

- Jupyter Notebook
  Jupyter Notebook merupakan aplikasi *open source* interaktif berbasis web untuk membuat file *notebook* [Notebook, 2020]. File *notebook* merupakan file berbentuk *javascript object notation* (JSON) yang berisi daftar masukan dan keluaran. Masukan dan keluaran tersebut dapat berupa kode program, teks *markdown*, grafik, dan hasil perhitungan matematika. Jupyter Notebook bekerja menggunakan IPython, sebuah *command shell* yang awalnya digunakan untuk melakukan komputasi secara interaktif menggunakan bahasa pemrograman Python. Saat ini, IPython telah memiliki kernel yang mendukung beberapa bahasa pemrograman diantaranya Julia, R, dan Heskell [Notebook, 2020].

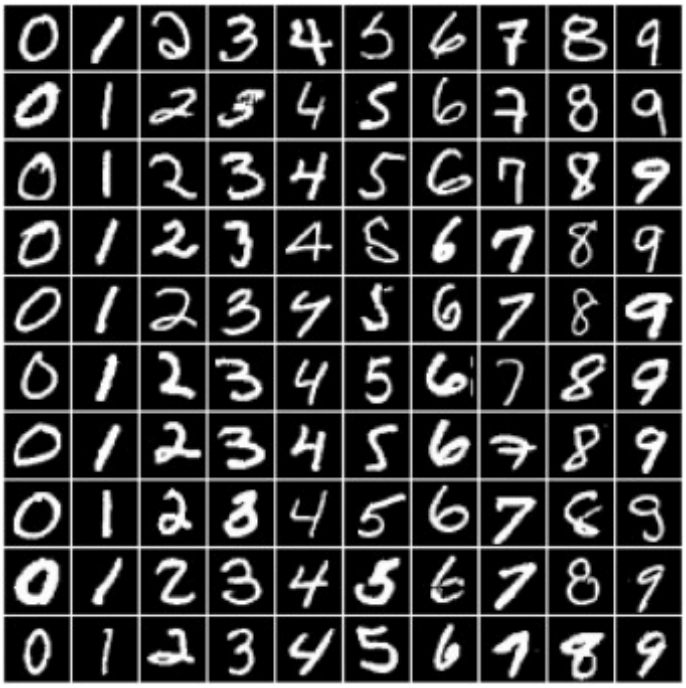
- GPU Tesla K80 dan GPU Tesla V100
  Penelitian ini menggunakan GPU untuk mendukung komputasi secara paralel· Spesifikasi GPU yang digunakan adalah NVIDIA Tesla K80 dan NVIDIA Tesla V100[NVIDIA, 2020]. Sumber daya komputasi tersebut disediakan oleh layanan *cloud* Floydhub dan UI AI Center. Pada awalnya penelitian dilakukan hanya dengan menggunakan GPU Tesla K80 yang disediakan oleh Floydhub. Penggunaan GPU tersebut dikenakan tarif sebesar 12$ untuk setiap 10 jam pemakaian. Seluruh *source code* dan *dataset* tersimpan di *server* Floydhub. Setelah mendapatkan akses dari UI AI Center, penelitian kemudian dialihkan menggunakan GPU Tesla V100. Akses terhadap GPU Tesla V100 tidak dikenakan biaya. Seluruh *source code* dan *dataset* juga disimpan pada server UI AI center. *Source code* dan *dataset* yang digunakan sama dengan yang tersimpan pada server Floydhub. Waktu komputasi menggunakan GPU Tesla V100 berlangsung lebih cepat bila dibandingkan dengan GPU Tesla K80. Sebagai perbandingan, waktu komputasi pada pembelajaran *unsupervised* LSAA untuk satu *epoch* berlangsung selama kurang lebih 20 detik. Sementara itu, waktu komputasi pada pembelajaran *supervised* LSAA untuk satu *epoch* berlangsung selama kurang lebih 60 detik.

5.2 *Dataset*

Terdapat dua *dataset* yang digunakan pada penelitian ini yaitu MNIST dan FashionMNIST. Kedua *dataset* diperoleh menggunakan library *torchvision* [Torch, 2020]. Pada penelitian ini, setiap *dataset* dinormalisasi dalam skala [0.0, 1.0]. Normalisasi terhadap data diharapkan dapat mempercepat konvergensi dalam proses pembelajaran.

5.2.1 MNIST

Dataset MNIST merupakan kumpulan citra digit sintetis yang diciptakan oleh National Institute of Standard and Technology. Citra bersifat *grayscale* dan berukuran 28×28. Keseluruhan *dataset* berjumlah 70000 dengan 60000 sampel pembelajaran dan 10000 sampel *test set* [LeCun and Cortes, 2020]. *Dataset* ini memiliki sepuluh label yang menyatakan digit nol sampai sembilan. Pada penelitian ini citra MNIST dipergunakan pada eksperimen *unsupervised* LSAA dan *supervised* LSAA. **Gambar 5.1** menunjukkan sampel gambar dari *Dataset* MNIST.



Gambar 5.1: Sampel Data MNIST.

5.2.2 FashionMNIST

FashionMNIST merupakan *dataset* berupa citra pakaian yang diambil dari Artikel Zalando. *Dataset* ini juga dibuat oleh NIST. *Dataset* FashionMNIST berupa citra *grayscale* berukuran 28 $\times$ 28. *Dataset* FashionMNIST memiliki 10 label atau kelas. Kesepuluh kelas tersebut yaitu kaos, celana panjang, *pullover*, gaun, mantel, sandal, kemeja, *sneaker*, tas, dan *ankle boot*. *Dataset* berjumlah 70000 sampel yang terdiri dari 60000 sampel pembelajaran dan 10000 sampel *test set* [Xiao et al., 2017]. *Dataset* ini digunakan pada eksperimen *unsupervised* LSAA. **Gambar 5.2** menunjukkan sampel *Dataset* FashionMNIST.

Gambar 5.2: Sampel Data FashionMNIST.

5.3 Standar Pengukuran *Least Square Adversarial Autoencoder*

Pada penelitian ini terdapat tiga standar pengukuran yang digunakan untuk menguji hasil eksperimen. Ketiga standar pengukuran tersebut adalah *reconstruction error*, akurasi klasifikasi antara data asli dan data hasil rekonstruksi, serta nilai Fréchet Inception Distance (nilai FID). *Mean squared error* dan akurasi klasifikasi mengukur hasil rekonstruksi LSAA. *Mean squared error* mengukur *quadratic* distance antara citra asli dan citra hasil rekonstruksi. Akurasi klasifikasi digunakan untuk mengukur akurasi klasifikasi data yang direkonstruksi terhadap data asli berdasarkan label atau target data tersebut [Liu, 2019]. Frechét Inception Distance score (FID score) digunakan untuk mengukur perbedaan statistik dan diversitas data yang dibangkitkan [Heusel et al., 2017].

Reconstruction error berguna untuk mengukur perbedaan antara data asli dan data hasil rekonstruksi yang dihasilkan oleh *autoencoder*. Penelitian ini menggunakan *mean squared error* (MSE) untuk mengukur *reconstruction error*. *Mean squared error* bekerja dengan cara menghitung rata-rata nilai kuadrat dari hasil pengurangan data asli terhadap data hasil rekonstruksi (L^2 *loss*) [Lehmann and Casella, 2006]. Nilai MSE yang semakin kecil menunjukkan bahwa kedua data cenderung semakin identik. Sebaliknya, nilai MSE yang semakin besar menunjukkan bahwa kedua data tersebut cenderung semakin berbeda. Nilai minimum MSE adalah nol, yang berarti bahwa kedua data tersebut adalah identik atau sama. Sementara itu, nilai maksimum MSE relatif terhadap nilai maksimum dan minimum data. Misalkan terdapat dua kelompok data D_1 dan D_2 yang berjumlah n. Nilai

MSE dari kedua kelompok data tersebut dapat ditulis sebagai:

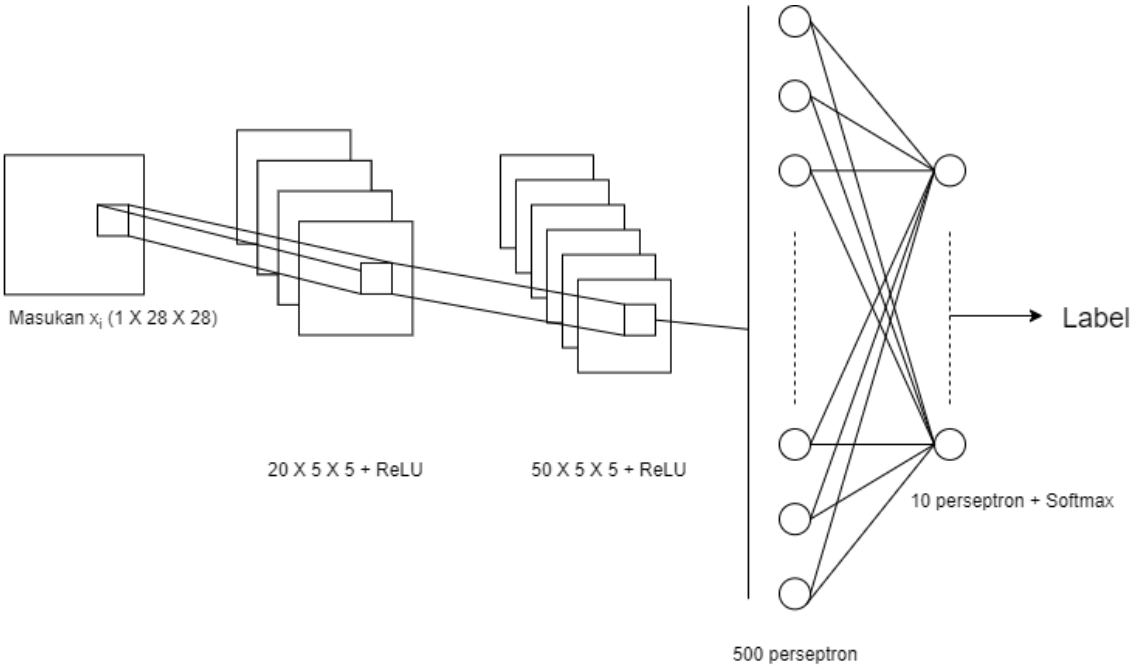
$$MSE(D_1, D_2) = \frac{1}{n}\sum_{i=1}^{n}(D_1^{(i)} - D_2^{(i)})^2 \tag{5.1}$$

Akurasi klasifikasi mengukur presentase hasil prediksi label pada data hasil rekonstruksi *autoencoder* terhadap label data asli. Standar pengukuran ini dipakai untuk mendukung hasil pengukuran *reconstruction error*. Pengukuran dilakukan untuk mengetahui apakah *classifier* mengklasifikasikan data asli dan data hasil rekonstruksi dengan label atau target yang sama. Apabila *autoencoder* bekerja secara ideal, maka data hasil rekonstruksi akan identik dengan data asli sehingga memiliki label yang sama ketika diklasifikasi. Misalkan terdapat kelompok data $D_1 = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), ..., (\boldsymbol{x}_n, \boldsymbol{y}_n)\}$ dengan $\boldsymbol{x}_i$ menyatakan data sampel dan $\boldsymbol{y}_i$ menyatakan label pada sampel. Lalu misalkan terdapat hasil rekonstruksi D_1 yaitu $D_2 = \{(\boldsymbol{x}'_1, \boldsymbol{y}'_1), (\boldsymbol{x}'_2, \boldsymbol{y}'_2), ..., (\boldsymbol{x}'_n, \boldsymbol{y}'_n)\}$ dengan $\boldsymbol{x}'_i$ menyatakan data hasil rekonstruksi dan $\boldsymbol{y}'_i$ menyatakan label hasil klasifikasi $\boldsymbol{x}'_i$. Akurasi klasifikasi hasil rekonstruksi data tersebut dapat dinyatakan sebagai:

$$akurasi(D_2) = \frac{100}{n}\sum_{i=1}^{n}(g(y^{(i)}, y'^{(i)})) \tag{5.2}$$

$$g(y^{(i)}, y'^{(i)}) = \begin{cases} 1 & \text{jika } y^{(i)} = y'^{(i)} \\ 0 & \text{jika } y^{(i)} \neq y'^{(i)} \end{cases} \tag{5.3}$$

Nilai akurasi yang tinggi menunjukkan bahwa kedua data memiliki kemiripan yang tinggi. Nilai maksimum akurasi adalah seratus persen (100%), yang berarti bahwa seluruh label pada data asli dan data hasil rekonstruksi sama. Sebaliknya, jika seluruh label pada data asli berbeda dengan seluruh label data hasil rekonstruksi, maka akurasi klasifikasi bernilai nol persen (0%). Pada penelitian ini, dikembangkan suatu jaringan saraf tiruan konvolusional untuk melakukan klasifikasi terhadap data yang dicobakan. Jaringan tersebut terdiri dari dua lapisan konvolusional berukuran 5×5 yang masing-masing berjumlah 20 kernel dan 50 kernel, serta 2 lapisan *fully connected*, yang masing-masing memiliki 500 perseptron dan 10 perseptron. Setiap lapisan konvolusional menggunakan fungsi aktivasi ReLU. Lapisan terakhir (*fully connected*) pada jaringan tersebut diaktivasi menggunakan fungsi softmax, untuk kemudian memberi prediksi mengenai label dari data masukan. **Gambar 5.3** menunjukkan arsitektur *classifier*. Program Python dari *classifier* dapat dilihat pada bagian Lampiran 3.



Gambar 5.3: Arsitektur *Classifier*.

Fréchet Inception Distance score (FID score) merupakan metode yang digunakan untuk mengukur kemiripan statistik dan diversitas antara data yang dibangkitkan dan data asli. Metode FID diterapkan pada data citra. Metode ini membandingkan komponen statistik antara citra asli dan citra yang dibangkitkan [Heusel et al., 2017]. Misalkan terdapat sampel citra $\boldsymbol{x}_r$ yang di-*sampling* dari *test set* ($\boldsymbol{x}_r \sim \mathcal{N}(\mu_r,\ \sum_r)$) dan sampel citra $\boldsymbol{x}_g$ yang diperoleh dari hasil pembangkitan ($\boldsymbol{x}_g \sim \mathcal{N}(\mu_g, \sum_g)$). Notasi μ menyatakan *mean*, sementara notasi $\sum$ menyatakan matriks kovarians. Secara matematis, nilai FID antara data asli $\boldsymbol{x}_r$ dan data hasil pembangkitan $\boldsymbol{x}_g$ dapat ditulis sebagai:

$$FID = |\mu_r - \mu_g|^2 + Tr(\sum_r + \sum_g - 2(\sum_r \sum g)^{\frac{1}{2}}) \tag{5.4}$$

dengan Tr menyatakan penjumlahan diagonal matriks kovarians. Nilai FID yang rendah menunjukkan bahwa dua distribusi data memiliki karakteristik statistik yang hampir sama. Nilai ideal FID adalah 0, yang berarti bahwa kedua distribusi adalah sama. Metode FID menggunakan arsitektur jaringan saraf tiruan konvolusional yang diberi nama *inception* [Szegedy et al., 2015]. Jaringan tersebut digunakan untuk mengekstraksi fitur dari citra asli dan citra hasil pembangkitan. Fitur tersebut kemudian dimodelkan menggunakan distribusi normal multivariatif dengan *mean* μ dan kovarians $\sum$. Terdapat metode lain yang dapat digunakan untuk mengukur kemiripan statistik dan diversitas data, yaitu *inception score* (IS) [Barratt and Sharma, 2018]. Namun metode ini dianggap kurang baik karena kurang resisten terhadap *noise* [Heusel et al., 2017].

5.4 Eksperimen pada *Unsupervised LSAA*

Eksperimen pada *unsupervised* LSAA menggunakan *dataset* MNIST dan FashionMNIST. Hasil *reconstruction error* dari *test set* diukur menggunakan MSE dan akurasi klasifikasi antara citra asli dan citra hasil rekonstruksi. Akurasi dihitung berdasarkan jumlah citra asli yang memiliki label atau target yang sama dengan citra hasil rekonstruksi ketika diklasifikasi oleh *classifier*. Kualitas hasil pembangkitan citra pada *dataset* MNIST dan FMNIST diukur menggunakan nilai FID.

5.4.1 *Hyperparameter* pada Model *Unsupervised* LSAA

Model ini diimplementasi menggunakan *framework* PyTorch dengan menggunakan beberapa *hyperparameter* yang ditunjukkan pada **Tabel 5.1**. Proses pembelajaran berlangsung selama kurang lebih 2 jam. Program Python dari *unsupervised* LSAA dapat dilihat pada bagian Lampiran 1.

Parameter	Nilai	Deskripsi
lambda (λ)	0.01 dan 0.02	*Hyperparameter* yang mengatur bobot fungsi objektif pada diskriminator dan generator
lr	0.0002	*learning rate* pada optimator adam
sigma	1.0	Nilai standar deviasi pada distribusi apriori (distribusi normal dengan *mean* = 0.0)
z_dimension	20	Ukuran dimensi pada variabel laten
batch_size	500	Ukuran *minibatch* yang digunakan selama proses pembelajaran
epochs	500	Jumlah iterasi yang dilakukan untuk melakukan pembelajaran

Tabel 5.1: Tabel Parameter *Unsupervised* LSAA.

Seluruh nilai *hyperparameter* tersebut didapat dengan mencoba beberapa nilai sampai mendapatkan hasil yang baik. Berikut adalah beberapa catatan mengenai penentuan *hyperparameter* tersebut.

- Pemilihan nilai *hyperparameter* λ berpengaruh terhadap hasil rekonstruksi dan hasil pembangkitan citra. Penggunaan nilai λ yang kurang optimal dapat menyebabkan kemampuan rekonstruksi yang lebih buruk atau kemampuan pembangkitan yang buruk. Nilai λ yang terlalu besar menyebabkan nilai *reconstruction error* menjadi besar dan sulit mencapai titik konvergen. Pemilihan

nilai λ yang terlalu kecil memberi dampak yang tidak signifikan pada fungsi objektif diskriminator dan generator. Akibatnya, meskipun nilai MSE *reconstruction error* bernilai kecil, LSAA tidak mampu berperan sebagai model pembangkit. Pemilihan nilai λ merupakan *tradeoff* untuk mendapatkan hasil rekonstruksi yang baik dan pembangkitan yang baik. Pada penelitian ini, *lambda* dengan nilai 0.01 memberi nilai MSE, akurasi klasifikasi, dan FID yang baik.

- Pemilihan nilai *epoch* berdasarkan kondisi pembelajaran dimana nilai *reconstruction error* (MSE) yang sudah mengalami osilasi mulai dari *epoch* ke-35. Osilasi merupakan kondisi dimana nilai fungsi objektif yang menaik dan menurun dalam rentang nilai yang tetap seiring bertambahnya *epoch*. Nilai *epoch* sebesar 50 kemudian ditetapkan sebagai batas atas nilai *epoch*.

5.4.2 Hasil *Reconstruction Error* pada *Unsupervised* LSAA

Pengukuran *reconstruction error* dilakukan terhadap 10000 sampel *test set* (MNIST dan FashionMNIST) yang sebelumnya tidak digunakan ketika model melakukan proses pembelajaran. Hasil pengukuran *reconstruction error* (MSE dan akurasi klasifikasi antara citra asli dan citra hasil rekonstruksi) pada kedua *dataset* dapat dilihat pada **Tabel 5.2**

Datasets	MSE	Akurasi
MNIST	0.006302	98.54%
FashionMNIST	0.009447	89.77%

Tabel 5.2: Tabel *Reconstruction Error* dan Akurasi Klasifikasi pada *Unsupervised* LSAA.

Kemampuan rekonstruksi LSAA dibandingkan dengan *variational autoencoder* (VAE) dan *Wasserstein autoencoder* (WAE), berdasarkan percobaan yang dilakukan Dapello et al.,(2018). Percobaan tersebut membandingkan kemampuan rekonstruksi dari tiga model yaitu *autoencoder* biasa, VAE dan WAE, dengan menggunakan *dataset* FashionMNIST dan MNIST. [Tolstikhin et al., 2017, Dapello et al., 2020]. Percobaan tersebut menggunakan jumlah data yang sama dengan penelitian yang dilakukan saat ini, dengan 60.000 sampel digunakan untuk pembelajaran dan 10000 sampel untuk *testing*. **Tabel 5.3** menunjukkan nilai MSE dari VAE dan WAE berdasarkan percobaan yang dilakukan Dapello et al.,(2018).

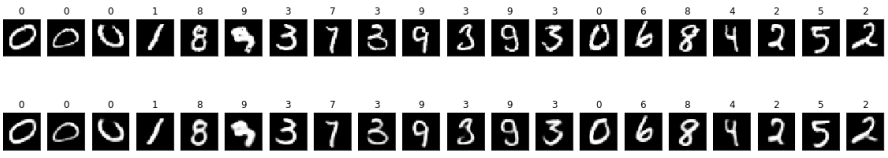
Datasets	MSE VAE	MSE WAE
MNIST	0.0095	0.0071
FashionMNIST	0.0137	0.0093

Tabel 5.3: Tabel MSE dari VAE dan WAE berdasarkan percobaan yang dilakukan oleh Dapello, et al.,(2018).

Berdasarkan nilai MSE yang ditunjukkan oleh **Tabel 5.2** dan **Tabel 5.3**, *unsupervised* LSAA memberi hasil rekonstruksi yang lebih baik dibandingkan dengan VAE. Pada *dataset* MNIST, nilai MSE VAE sebesar 0.0095 sementara nilai MSE *unsupervised* LSAA sebesar 0.006302. Pada *dataset* FashionMNIST, nilai MSE *unsupervised* LSAA sebesar 0.009447 sementara nilai MSE VAE sebesar 0.0137. *Unsupervised* LSAA juga memiliki nilai MSE yang lebih baik bila dibandingkan dengan WAE yang mencatatkan nilai MSE sebesar 0.0071 untuk *dataset* MNIST. Namun, *Wasserstein autoencoder* mencatat nilai MSE yang lebih baik untuk *dataset* FashionMNIST yaitu sebesar 0.0093.

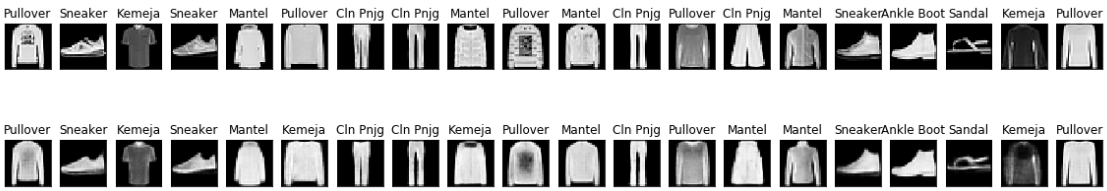
Hasil pengukuran akurasi menunjukkan terdapat 146 sampel *test set* MNIST dan 1023 sampel *test set* FashionMNIST yang salah diprediksi oleh *classifier* dari total 10000 data yang diuji. Jumlah kesalahan klasifikasi pada *test set* FashionMNIST yang banyak disebabkan adanya beberapa citra memiliki label yang berbeda namun memiliki morfologi yang mirip. Sebagai contoh, citra kemeja, mantel, dan *pullover* memiliki morfologi yang mirip. Proses rekonstruksi memberi perubahan pada morfologi citra. Perubahan tersebut menyebabkan *classifier* memberi nilai prediksi klasifikasi yang berbeda pada citra hasil rekonstruksi.

Selanjutnya, ditunjukkan contoh hasil rekonstruksi citra MNIST. **Gambar 5.4** menunjukkan contoh hasil rekonstruksi citra MNIST.



Gambar 5.4: Contoh Hasil Rekonstruksi *Dataset* MNIST Menggunakan *Unsupervised* LSAA.

Citra digit pada baris pertama menunjukkan citra yang berasal dari *test set*. Citra digit pada baris kedua merupakan hasil rekonstruksi terhadap citra pada baris pertama. Setiap digit pada gambar tersebut diklasifikasi oleh *classifier*. Hasil klasifikasi ditunjukkan oleh label yang berada di atas setiap citra. Berdasarkan gambar tersebut, setiap citra pada baris pertama memiliki label yang sama dengan setiap citra pada baris kedua. Selanjutnya, **Gambar 5.5** menunjukkan contoh hasil rekonstruksi pada *dataset* FashionMNIST.



Gambar 5.5: Contoh Hasil Rekonstruksi *Dataset* FashionMNIST Menggunakan *Unsupervised* LSAA.

Citra pakaian pada baris pertama menunjukkan citra yang berasal dari *test set*. Citra pakaian pada baris kedua merupakan hasil rekonstruksi terhadap citra pada baris pertama. Berdasarkan gambar tersebut, terdapat tiga citra asli yang memiliki tiga label yang berbeda dengan label citra hasil rekonstruksinya. Ketiga citra yang berbeda adalah citra *pullover* (urutan keenam dari sebelah kiri), citra mantel (urutan kesembilan dari sebelah kiri), dan citra celana panjang (urutan empat belas dari sebelah kiri). Meskipun mempunyai label yang berbeda, namun ketiga hasil rekonstruksi citra masih memiliki kemiripan morfologi dengan citra asli.

Berdasarkan hasil eksperimen, model *unsupervised* LSAA mampu melakukan rekonstruksi terhadap *dataset* MNIST dan FashionMNIST (model *unsupervised* LSAA). Hal ini ditunjukkan oleh nilai MSE yang mendekati 0. Nilai MSE yang semakin mendekati 0 menunjukkan citra hasil rekonstruksi semakin mirip dengan citra asli. Berikutnya, nilai akurasi klasifikasi antara citra asli dan citra hasil rekonstruksi pada model *unsupervised* LSAA menunjukkan bahwa hampir seluruh citra hasil rekonstruksi memiliki label sama dengan citra asli. Hasil rekonstruksi citra pada model *unsupervised* LSAA mampu mengembalikan morfologi citra yang asli. Namun, citra hasil rekonstruksi kehilangan beberapa detail bila dibandingkan dengan citra asli. Hal ini disebabkan hilangnya beberapa informasi pada saat proses *encoding*. Proses *encoding* memetakan data menuju dimensi yang lebih kecil sehingga terdapat detail informasi yang hilang.

5.4.3 Hasil Pembangkitan pada *Unsupervised* LSAA

Nilai FID hasil pembangkitan *supervised* LSAA membandingkan 10000 sampel citra yang berasal dari *test set* dan 10000 sampel citra hasil pembangkitan. Citra dibangkitkan dengan melakukan *decoding* terhadap *noise* $\boldsymbol{z}$ yang di-*sampling* dari distribusi apriori ($G_{\theta_g}(\boldsymbol{z})$). Distribusi apriori yang digunakan adalah distribusi normal dengan *mean* bernilai 0 dan standar deviasi bernilai 1. **Tabel 5.4** menunjukkan nilai FID *dataset* MNIST dan FashionMNIST.

Datasets	FID
MNIST	15.7182
FashionMNIST	38.69675

Tabel 5.4: Tabel nilai FID Hasil Pembangkitan Citra Menggunakan *Unsupervised* LSAA.

Nilai FID yang ditunjukkan oleh **Tabel 5.4** dibandingkan dengan nilai FID dari beberapa model pembangkit, berdasarkan penelitian yang dilakukan oleh Lucic et al.,(2018). Pada penelitian tersebut, dilakukan pengukuran nilai FID terhadap beberapa model pembangkit diantaranya *variational autoencoder*, LSGAN, DCGAN, BEGAN, dan Wasserstein GAN. Nilai FID masing-masing model pembangkit dihitung berdasarkan 10000 sampel *test set* dan 10000 sampel citra hasil pembangkitan. Pengukuran dilakukan sebanyak 50 kali sehingga didapatkan nilai rata-rata dari perhitungan FID. Penelitian tersebut menggunakan MNIST dan FashionMNIST sebagai *dataset*. **Tabel 5.5** menunjukkan nilai rata-rata FID dari beberapa model pembangkit berdasarkan penelitian yang dilakukan oleh Lucic, et al.,(2018).

Model	FID MNIST	FID FashionMNIST
Vanilla GAN	9.8	29.6
Non saturating GAN	6.8	26.5
Least square GAN	7.8	30.7
Wasserstein GAN	6.7	21.5
Wasserstein GAN *gradient penalty*	20.3	24.5
Deep Regret Analytic GAN	7.6	27.7
Boundary Equilibrium GAN	13.1	22.9
Variational autoencoder	23.8	58.7

Tabel 5.5: Tabel rata-rata 50 kali pengukuran FID dari beberapa model pembangkit berdasarkan percobaan yang dilakukan oleh Lucic, et al.,(2018).

Penelitian tersebut menunjukkan nilai rata-rata 50 pengukuran FID dari VAE adalah sebesar 23.8 untuk *dataset* MNIST dan 58.7 untuk *dataset* FashionMNIST [Lucic et al., 2018]. Pada penelitian tersebut, VAE mencatat nilai FID yang paling rendah dibandingkan dengan model pembangkit lainnya. Nilai FID yang paling tinggi dicatatkan oleh Wasserstein GAN, dengan nilai rata-rata FID sebesar 6.7 untuk *dataset* MNIST dan 21.5 untuk *dataset* FashionMNIST. Berdasarkan nilai FID tersebut, *unsupervised* LSAA menjadi model pembangkit yang lebih baik dibandingkan *variational autoencoder*. Hasil

ini masih belum maksimal bila dibandingkan dengan model pembangkit lain yang memberi nilai FID yang lebih rendah.

Gambar 5.6 menunjukkan hasil pembangkitan citra MNIST ($z \sim G_{\theta_g}(z)$).

Gambar 5.6: Contoh Hasil Pembangkitan Citra MNIST menggunakan *Unsupervised* LSAA.

Berdasarkan gambar tersebut, *unsupervise* LSAA mampu membangkitkan sebagian citra digit dengan morfologi yang menyerupai citra pembelajaran. Namun, terdapat beberapa citra yang memiliki morfologi yang berbeda dengan morfologi citra pembelajaran.

Hasil pembangkitan citra FashionMNIST dari *noise* ($z \sim G_{\theta_g}(z)$) dapat dilihat pada **Gambar 5.7**.

Gambar 5.7: Contoh Hasil Pembangkitan Citra FashionMNIST menggunakan *Unsupervised* LSAA

Berdasarkan gambar tersebut, model *unsupervised* LSAA mampu membangkitkan citra yang memiliki morfologi yang menyerupai *dataset* FashionMNIST. Namun, citra yang dihasilkan kurang memiliki detail, terutama untuk citra yang menyerupai kemeja atau kaos. Selain itu, terdapat citra yang menyerupai tas yang memiliki morfologi yang kurang sempurna. Hal ini menunjukkan bahwa LSAA kurang maksimal dalam melakukan pembangkitan citra. Dugaan sementara, *f-divergence* antara distribusi variabel laten dan distribusi apriori masih belum minimum secara sempurna.

5.5 Eksperimen pada *Supervised* LSAA

Penelitian *supervised* LSAA menggunakan *dataset* MNIST dan *dataset* FashionMNIST. Standar pengukuran *supervised* LSAA sama dengan pengukuran yang dilakukan terhadap model *unsupervised* LSAA. Pada eksperimen ini digunakan empat dimensi variabel laten berbeda yang masing-masing berukuran 2, 3, 4, dan 5.

5.5.1 *Hyperparameter* pada Model *Supervised* LSAA

Sama seperti *unsupervised* LSAA, model ini diimplementasi menggunakan *framework* PyTorch dengan menggunakan beberapa *hyperparameter* yang ditunjukkan pada **Tabel**

5.6. Proses pembelajaran berlangsung selama kurang lebih 1 jam untuk masing-masing model dengan dimensi variabel laten yang berbeda. Program Python model dan algoritma pembelajaran *supervised* LSAA dapat dilihat pada bagian Lampiran 2.

Parameter	Nilai	Deskripsi
sigma	5.0	Nilai standar deviasi pada distribusi priori (distribusi normal dengan *mean* = 0.0)
z_dimension	2, 3, 4, 5	Ukuran dimensi variabel laten
batch_size	100	Ukuran *minibatch* yang digunakan selama proses pembelajaran
epochs	500	Jumlah iterasi yang dilakukan untuk melakukan pembelajaran

Tabel 5.6: Tabel Parameter *Supervised* LSAA.

Seluruh *hyperparameter* tersebut didapatkan dengan mencoba beberapa nilai sampai mencapai hasil yang terbaik. Dimensi variabel laten yang digunakan bernilai 2,3,4, dan 5. Hal tersebut bertujuan untuk mengetahui efek dimensi dari variabel laten terhadap hasil rekonstruksi dan pembangkitan citra. Nilai *epoch* diatur cukup besar dengan tujuan memaksimalkan pembelajaran pada *supervised* LSAA.

5.5.2 Hasil Rekonstruksi pada *Supervised* LSAA

Pengukuran *reconstruction error* dilakukan terhadap 10000 sampel citra baru (MNIST) yang sebelumnya tidak digunakan ketika model melakukan proses pembelajaran. Hasil pengukuran MSE dan akurasi klasifikasi untuk *dataset* MNIST ditunjukkan oleh **Tabel 5.7**.

Dimensi Variabel Laten	MSE	Akurasi Klasifikasi (Persen)	Jumlah Misklasifikasi Data
2	0.003304	98.79%	121
3	0.002823	98.65%	135
4	0.002517	98.62%	138
5	0.002225	98.59%	141

Tabel 5.7: Tabel MSE dan Akurasi Klasifikasi pada *Supervised* LSAA untuk *Dataset* MNIST.

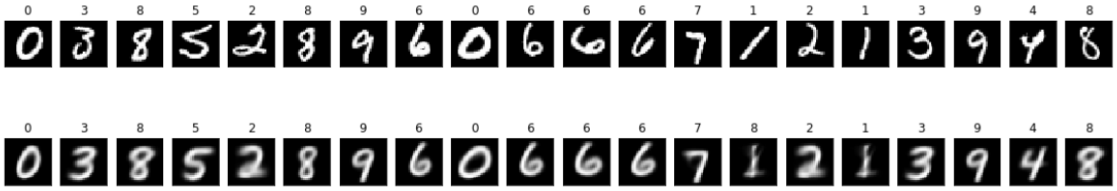
Kolom misklasifikasi data menunjukkan jumlah citra dari *test-set* yang memiliki label yang berbeda dengan hasil rekonstruksinya ketika diklasifikasi oleh *classifier*. Nilai MSE

pada *supervised* LSAA tergolong baik karena nilainya yang mendekati 0. Nilai MSE dari *supervised* LSAA semakin kecil seiring bertambahnya dimensi dari variabel laten. Secara teori, hal ini dikarenakan jumlah informasi yang hilang selama proses *encoding* semakin sedikit seiring bertambahnya dimensi dari variabel laten. Akibatnya, *distance* antara citra asli dan citra hasil rekonstruksi semakin kecil. Pada ekseperimen ini, nilai akurasi berkurang seiring dengan bertambahnya dimensi dari variabel laten. Nilai akurasi berkurang seiring dengan bertambahnya jumlah citra yang memiliki label yang berbeda dengan hasil rekonstruksinya. Secara keseluruhan nilai MSE dan akurasi klasifikasi antara citra asli dan hasil rekonstruksi dari *supervised* LSAA lebih baik dibandingkan dengan *unsupervised* LSAA. Nilai MSE *supervised* LSAA juga lebih baik dibandingkan nilai MSE VAE dan WAE berdasarkan percobaan yang dilakukan oleh Dapello, et al.,(2018) (**Tabel 5.3**). Lalu, hasil pengukuran MSE dan akurasi klasifikasi untuk *dataset* FashionMNIST ditunjukkan oleh **Tabel 5.8**.

Dimensi Variabel Laten	MSE	Akurasi Klasifikasi (Persen)	Jumlah Misklasifikasi Data
2	0.02578	88.1%	1190
3	0.02114	89.53%	1047
4	0.01819	88.44%	1156
5	0.01716	87.67%	1233

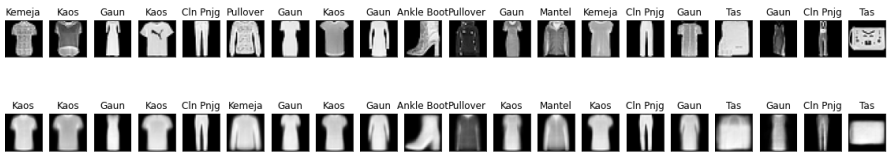
Tabel 5.8: Tabel MSE dan Akurasi Klasifikasi pada *Supervised* LSAA untuk *Dataset* FashionMNIST.

Nilai MSE pada *supervised* LSAA untuk *dataset* FashionMNIST tidak lebih baik dibandingkan dengan *unsupervised LSAA*. Hasil ini juga tidak lebih baik bila dibandingkan dengan nilai MSE dari VAE dan WAE berdasarkan percobaan yang dilakukan oleh Dapello, et al.,(2018) (**Tabel 5.3**). Nilai MSE *supervised* LSAA berkurang seiring dengan bertambahnya dimensi variabel laten. Nilai akurasi klasifikasi meningkat dari variabel laten berdimensi 2 menuju variabel laten berdimensi 3. Namun, akurasi menurun dari variabel laten berdimensi 3 menuju variabel laten berdimensi 4 dan dari variabel laten berdimensi 4 menuju variabel laten berdimensi 5. Selanjutnya, **Gambar 5.8** menunjukkan contoh hasil rekonstruksi citra MNIST untuk *supervised* LSAA dengan variabel laten berdimensi 2.



Gambar 5.8: Contoh Hasil Rekonstruksi *Dataset* MNIST Menggunakan *Supervised* LSAA dengan Variabel Laten Berdimensi 2.

Citra digit pada baris pertama menunjukkan citra yang berasal dari *test set*. Citra digit pada baris kedua merupakan hasil rekonstruksi terhadap citra pada baris pertama. Setiap digit pada gambar tersebut diklasifikasi oleh *classifier*. Hasil klasifikasi ditunjukkan oleh label yang berada di atas setiap citra. Berdasarkan gambar tersebut, terdapat satu citra pada baris pertama memiliki label yang berbeda dengan citra hasil rekonstruksinya (urutan ketujuh dari kanan). Selanjutnya, **Gambar 5.9** menunjukkan contoh hasil rekonstruksi citra FashionMNIST untuk *supervised* LSAA dengan variabel laten berdimensi 2.



Gambar 5.9: Contoh Hasil Rekonstruksi *Dataset* FashionMNIST menggunakan *Supervised* LSAA dengan Variabel Laten Berdimensi 2.

Citra pada baris pertama merupakan citra asli, sementara citra pada baris kedua merupakan citra hasil rekonstruksi. Berdasarkan gambar tersebut, terdapat dua citra yang memiliki label yang berbeda dengan citra hasil rekonstruksinya.

Penggunaan label pada proses pembelajaran menyebabkan model melakukan pembelajaran diskriminatif. Proses pembelajaran LSAA dengan menggunakan label dapat dipandang sebagai upaya untuk melakukan regresi untuk memprediksi hasil rekonstruksi citra asli.

Meski memberi hasil MSE yang baik, ketajaman citra hasil rekonstruksi *supervised* LSAA relatif lebih buruk dibandingkan dengan model *unsupervised* LSAA. Dimensi variabel laten yang jauh lebih sedikit diduga menjadi faktor utama (dimensi variabel laten pada *supervised* adalah 2). Semakin kecil dimensi variabel yang digunakan, maka semakin banyak informasi yang hilang dari data asli. Akibatnya, meskipun model *supervised* AA mampu merekonstruksi morfologi citra digit yang asli, detail informasi

yang hilang lebih banyak sehingga citra hasil rekonstruksi pada model *supervised* AA terlihat lebih *blur*.

5.5.3 Hasil Pembangkitan Citra pada *Supervised* LSAA

Nilai FID hasil pembangkitan pada *supervised* LSAA membandingkan 10000 sampel citra yang berasal dari *test set* dan 10000 sampel citra hasil pembangkitan. Citra dibangkitakan dengan melakukan *decoding* terhadap *noise* $\boldsymbol{z}$ yang di-*sampling* dari distribusi apriori ($\boldsymbol{z} \sim G_{\theta_g}(\boldsymbol{z})$). Distribusi apriori yang digunakan adalah distribusi normal dengan *mean* bernilai 0 dan standar deviasi bernilai 5. **Tabel 5.9** menunjukkan nilai FID *supervised* LSAA untuk *dataset* MNIST.

Dimensi Variabel Laten	FID
2	62.512
3	53.782
4	56.482
5	53.268

Tabel 5.9: Tabel FID pada *Supervised* LSAA untuk *Dataset* MNIST.

Nilai FID *supervised* LSAA relatif menurun seiring bertambahnya dimensi variabel laten. Anomali terjadi untuk perubahan dari dimensi laten berdimensi 3 menuju variabel laten berdimensi 4 dimana nilai FID mengalami kenaikan. Nilai FID *supervised* LSAA lebih tinggi dibandingkan dengan *unsupervised* LSAA (**Tabel 5.4**). Nilai FID *supervised* LSAA juga lebih tinggi dibandingkan seluruh model pembangkit yang dicobakan oleh Lucic, et al.,(2018) (**Tabel 5.5**). Selanjutnya, **Tabel 5.10** menunjukkan nilai FID *supervised* LSAA untuk *dataset* FashonMNIST.

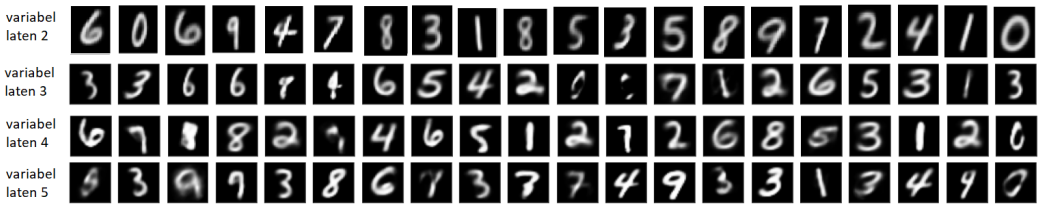
Dimensi Variabel Laten	FID
2	115.463
3	100.986
4	90.631
5	79.902

Tabel 5.10: Tabel FID pada *Supervised* LSAA untuk *Dataset* FashionMNIST.

Nilai FID *supervised* LSAA menurun seiring bertambahnya dimensi variabel laten. Nilai FID tersebut lebih tinggi dibandingkan dengan *unsupervised* LSAA

dan seluruh model pembangkit yang dicobakan oleh Lucic, et al.,(2018). Terdapat dua dugaan mengenai penyebab tingginya nilai FID pada *supervised* LSAA. Dugaan yang pertama adalah dimensi variabel laten yang kurang besar menyebabkan citra yang dibangkitkan kurang mirip dengan citra asli. Representasi data dalam dimensi yang lebih kecil memungkinkan adanya informasi yang hilang sehingga citra yang dibangkitkan kehilangan detil citra asli yang berkontribusi terhadap tingginya nilai FID. Dugaan kedua adalah *f-divergence* antara distribusi variabel laten dan distribusi apriori yang masih terlalu besar. Pembelajaran pada jaringan diskriminator yang belum mencapai titik ideal menyebabkan distribusi variabel laten tidak sama dengan distribusi apriori [Goodfellow et al., 2014]. Akibatnya, terdapat kemungkinan bahwa data yang dibangkitkan dari distribusi apriori bersifat *outlier* yang sama sekali berbeda dibandingkan dengan data pada sampel pembelajaran.

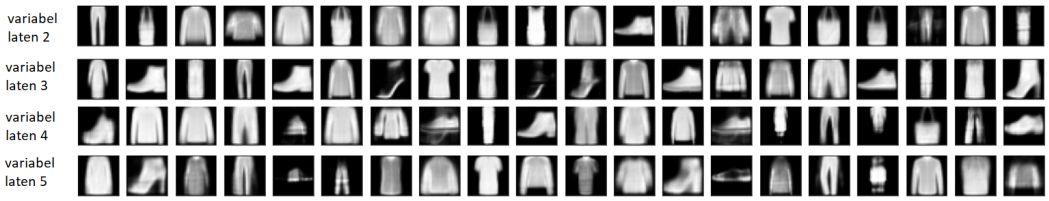
Gambar 5.10 menunjukkan hasil pembangkitan citra dari vektor *noise* $\boldsymbol{z}$ ($\boldsymbol{z} \sim G_{\theta_g}(\boldsymbol{z})$) pada *dataset* MNIST.



Gambar 5.10: Hasil Pembangkitan Citra MNIST menggunakan *Supervised* LSAA.

Baris pertama, kedua, ketiga, dan keempat masing-masing merupakan citra hasil pembangkitan *supervised* LSAA dengan variabel laten berdimensi 2, 3, 4, dan 5. Model *supervised* LSAA mampu membangkitkan beberapa citra yang memiliki morfologi yang menyerupai *dataset* MNIST. Namun, terdapat beberapa citra lain yang sama sekali tidak mirip dengan citra angka. Munculnya citra *outlier* hasil pembangkitan, diduga berkontribusi terhadap tingginya nilai FID pada *supervised* LSAA.

Gambar 5.11 menunjukkan hasil pembangkitan citra dari vektor *noise* $\boldsymbol{z}$ ($\boldsymbol{z} \sim G_{\theta_g}(\boldsymbol{z})$) untuk *dataset* FashionMNIST.



Gambar 5.11: Hasil Pembangkitan Citra FashionMNIST menggunakan *Supervised* LSAA.

Baris pertama, kedua, ketiga, dan keempat masing-masing merupakan citra hasil pembangkitan *supervised* LSAA dengan variabel laten berdimensi 2, 3, 4, dan 5. Model *supervised* LSAA mampu membangkitkan beberapa citra yang memiliki morfologi yang menyerupai *dataset* FashionMNIST. Namun, terdapat beberapa citra lain yang sama sekali tidak mirip dengan citra angka. Selain itu, citra hasil pembangkitan tidak memiliki detail seperti citra pembelajarannya.

BAB 6
KESIMPULAN DAN SARAN

Bab ini berisi kesimpulan dari penelitian yang telah dilakukan dan saran pengembangan untuk penelitian selanjutnya.

6.1 Kesimpulan

Tugas Akhir ini menelaah *least square adversarial autoencoder*, jenis *autoencoder* yang mampu merekonstruksi data sekaligus membangkitkan data dengan melakukan *decoding* pada *noise* yang di-*sampling* dari distribusi apriori. *Least square adversarial autoencoder* menggabungkan *adversarial autoencoder* dan *least square generative adversarial network*. Jaringan diskriminator pada *adversarial autoencoder* diimplementasi menggunakan *least square generative adversarial network*. Jaringan *least square generative adversarial network* pada diskriminator meminimalkan fungsi Pearson χ^2 *Divergence* antara distribusi variabel laten dan distribusi apriori. Terdapat dua model *least square adversarial autoencoder* yang diteliti yaitu *unsupervised least square adversarial autoencoder* dan *supervised least square adversarial autoencoder*. *Unsupervised least square adversarial autoencoder* tidak menggunakan label data dalam proses pembelajarannya, sedangkan *supervised least square adversarial autoencoder* menggunakan label data.

Pada Tugas Akhir ini, dibuat dua program Python yang memodelkan *least square adversarial autoencoder*. Program yang pertama memodelkan *unsupervised least square adversarial autoencoder*. Program tersebut memodelkan *encoder* dan *decoder* sebagai jaringan saraf konvolusional, sementara diskriminator sebagai jaringan saraf tiruan biasa. Program yang kedua memodelkan *supervised least square adversarial autoencoder*, dengan *encoder*, *decoder* dan diskriminator dimodelkan sebagai jaringan saraf tiruan biasa. *Unsupervised least square adversarial autoencoder* menggunakan variabel laten berdimensi 20 sementara *supervised least square adversarial autoencoder* menggunakan variabel laten masing-masing berdimensi 2, 3, 4, dan 5. Kedua program tersebut diimplementasikan menggunakan *framework* PyTorch dan dieksekusi menggunakan Jupyter Notebook. Seluruh aktivitas pemrograman dilakukan pada *environment cloud* yang disediakan oleh Floydhub dan Tokopedia-UI AI Center yang masing-masing menggunakan GPU NVIDIA Tesla K80 dan NVIDIA Tesla V100 sebagai perangkat komputasinya. Program yang sudah diimplementasi dilatih menggunakan

sampel pembelajaran dari *dataset* MNIST dan FashionMNIST yang berupa citra *grayscale*. Proses pembelajaran pada *unsupervised least square adversarial autoencoder* berlangsung selama kurang lebih dua jam sementara proses pembelajaran pada *supervised least square adversarial autoencoder* berlangsung kurang lebih selama enam jam. Program diuji dengan mengukur hasil rekonstruksi citra dan hasil pembangkitan citra.

Hasil eksperimen terhadap 10000 sampel *test set* MNIST dan 10000 sampel *test set* FashionMNIST menunjukkan *least square adversarial autoencoder* mampu merekonstruksi citra asli yang diberikan. Dalam penelitian ini, nilai *mean squared error* hasil rekonstruksi *unsupervised least square autoencoder* pada *dataset* MNIST sebesar 0.006302 dan pada *dataset* FashionMNIST sebesar 0.009447. Sementara itu, nilai *mean squared error* hasil rekonstruksi *least square adversarial autoencoder* pada *dataset* MNIST sebesar 0.003304. Selain itu, diukur pula nilai akurasi klasifikasi antara citra asli dan citra hasil rekonstruksinya. Pada model *unsupervised least square adversarial autoencoder* nilai akurasi pada *dataset* MNIST mencapai 98.54% pada *dataset* MNIST dan 89.77% pada *dataset* FashionMNIST. Sementara itu, nilai akurasi model *supervised least square adversarial autoencoder* pada *dataset* MNIST mencapai 98.79%.

Hasil eksperimen menunjukkan bahwa *least square adversarial autoencoder* mampu membangkitkan citra dengan morfologi yang menyerupai *dataset* yang digunakan pada proses pembelajaran. Hasil pengukuran FID pada model *unsupervised least square adversarial autoencoder* pada *dataset* MNIST mencapai 15.7182 dan pada *dataset* FashionMNIST mencapai 38.6967. Sementara itu, hasil pengukuran FID model *supervised least square adversarial autoencoder* pada *dataset* MNIST mencapai 62.512. Hasil FID kedua model tersebut menunjukkan kualitas hasil pembangkitan citra tidak sebagus kualitas citra *test set*.

Terdapat dua isu dalam penelitian ini. Isu yang pertama adalah sulitnya menentukan *hyperparameter* yang digunakan pada proses pembelajaran *least square adversarial autoencoder*. Sampai saat ini, belum ditemukan *state of the art* untuk menentukan *hyperparameter* yang optimal untuk mendukung proses pembelajaran pada *generative adversarial network* maupun *adversarial autoencoder*. Dibutuhkan percobaan berulang kali sampai akhirnya ditemukan *hyperparameter* yang ideal untuk mendukung proses pembelajaran. Isu yang kedua adalah mengenai *explainability*.

6.2 Saran

Berdasarkan hasil penelitian ini, terdapat beberapa saran untuk penelitian selanjutnya, yaitu:

1. Penelitian ini dilakukan pada data berupa citra *grayscale*. Penelitian dapat

dikembangkan untuk *dataset* yang berbeda seperti citra RGB (*Red, Green, Blue*), audio, teks, dan jenis *dataset* lainnya.

2. Penggunaan metode optimisasi pada proses pembelajaran *generative adversarial network* seperti *feature matching*, *batch discrimination*, dan *averaging history* [Salimans et al., 2016]. Metode tersebut dapat diterapkan pada jaringan diskriminator yang menggunakan *least square generative adversarial network*. Metode optimisasi tersebut dapat meningkatkan laju konvergensi dan mendapatkan parameter diskriminator yang lebih optimal.

3. *Autoencoder* dapat memetakan suatu distribusi data pada dimensi yang lebih kecil yang disebut sebagai variabel laten. Penelitian lebih lanjut dapat dilakukan untuk mempelajari variabel laten pada *least square adversarial autoencoder*. Variabel laten sendiri dapat dimanfaatkan untuk melakukan *transfer learning* dan *multi-task learning* [Baxter, 1998].

4. Variabel laten dapat digunakan untuk mengembangkan *explainable artificial intelligence* (*explainable* AI). Saat ini merupakan era *explainable* AI sehingga dibutuhkan suatu penjelasan atau interpretasi dari prediksi yang dihasilkan oleh agen cerdas (*right to explanation*). *Least square adversarial autoencoder* dapat berkontribusi terhadap *explainable* AI (khususnya *machine learning*) dengan memberi masukan dengan representasi yang lebih sederhana (variabel laten) terhadap model *explainable* AI. Variabel laten yang dihasilkan oleh *least square adversarial autoencoder* dapat dikombinasikan dengan model *explainable* AI yang menghasilkan *saliency map* [Tjoa and Guan, 2019]. *Saliency map* memberi interpretasi hasil prediksi dari suatu model *machine learning* dengan menunjukkan beberapa fitur masukan yang memiliki kontribusi paling besar terhadap hasil prediksi. Representasi masukan yang lebih sederhana mengurangi kompleksitas model *explainable* AI sehingga *saliency map* yang dihasilkan lebih sederhana.

DAFTAR PUSTAKA

[Abbod et al., 2007] Abbod, M. F., Catto, J. W., Linkens, D. A., and Hamdy, F. C. (2007). Application of artificial intelligence to the management of urological cancer. *The Journal of urology*, 178(4):1150–1156.

[Addi and Williams, 2010] Addi, H. and Williams, L. J. (2010). Principal component analysis. *Wiley Interdisciplinary Reviews: Comput-ational Statistics*, 2(4):433–459.

[Arjovsky et al., 2017] Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein gan. *arXiv preprint arXiv:1701.07875*.

[Barratt and Sharma, 2018] Barratt, S. and Sharma, R. (2018). A note on the inception score.

[Baxter, 1998] Baxter, J. (1998). Theoretical models of learning to learn. In *Learning to learn*, pages 71–94. Springer.

[Bayer and Osendorfer, 2014] Bayer, J. and Osendorfer, C. (2014). Learning stochastic recurrent networks. *arXiv preprint arXiv:1411.7610*.

[Bengio et al., 2013] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828.

[Berthelot et al., 2017] Berthelot, D., Schumm, T., and Metz, L. (2017). Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*.

[Charnes et al., 1976] Charnes, A., Frome, E. L., and Yu, P.-L. (1976). The equivalence of generalized least squares and maximum likelihood estimates in the exponential family. *Journal of the American Statistical Association*, 71(353):169–171.

[Chen et al., 2016] Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. (2016). Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180.

[Chen et al., 2019] Chen, Y.-Y., Lin, Y.-H., Kung, C.-C., Chung, M.-H., Yen, I., et al. (2019). Design and implementation of cloud analytics-assisted smart power

meters considering advanced artificial intelligence as edge analytics in demand-side management for smart homes. *Sensors*, 19(9):2047.

[Cho et al., 2014] Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

[Dagan et al., 1997] Dagan, I., Lee, L., Pereira, F., and Pereira, F. (1997). Similarity-based methods for word sense disambiguation. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 56–63. Association for Computational Linguistics.

[Dapello et al., 2020] Dapello, J., Sedelmeyer, M., and Yan, W. (2018 (accessed June 1, 2020)). An introduction to the wasserstein auto-encoder. https://github.com/sedelmeyer/wasserstein-auto-encoder/blob/master/Wasserstein-auto-encoder_tutorial.ipynb.

[Data et al., 2012] Data, L. F., Course, A. S., Abu-Mostafa, Y. S., Magdon-Ismail, M., and Lin, H. (2012). Scholarly activity. In *Proceedings of the 12th Conference on Electronic Commerce (EC), Valencia, Spain*, pages 4–8.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[Gulrajani et al., 2017] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017). Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777.

[Heusel et al., 2017] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems*, pages 6626–6637.

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

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

[Kodali et al., 2017] Kodali, N., Abernethy, J., Hays, J., and Kira, Z. (2017). On convergence and stability of gans. *arXiv preprint arXiv:1705.07215*.

[Kuhlman, 2012] Kuhlman, D. (2012). A python book: Beginning python. *Advanced Python, and Python Exercises*.

[Kullback, 1997] Kullback, S. (1997). *Information theory and statistics*. Courier Corporation.

[Kullback and Leibler, 1951] Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

[Langr and Bok, 2019] Langr, J. and Bok, V. (2019). *GANs in Action: Deep learning with Generative Adversarial Networks*. Manning Publications.

[LeCun et al., 1995] LeCun, Y., Bengio, Y., et al. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995.

[LeCun and Cortes, 2020] LeCun, Y. and Cortes, C. (2020 (accessed June 1, 2020)). The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

[Lehmann and Casella, 2006] Lehmann, E. L. and Casella, G. (2006). *Theory of point estimation*. Springer Science & Business Media.

[Liu, 2019] Liu, H. (2019). *Machine Learning A Quantitative Approach*. PerfMath.

[Lucic et al., 2018] Lucic, M., Kurach, K., Michalski, M., Gelly, S., and Bousquet, O. (2018). Are gans created equal? a large-scale study. In *Advances in neural information processing systems*, pages 700–709.

[Makhzani et al., 2015] Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., and Frey, B. (2015). Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*.

[Manning et al., 1999] Manning, C. D., Manning, C. D., and Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT press.

[Mao et al., 2017] Mao, X., Li, Q., Xie, H., Lau, R. Y., Wang, Z., and Paul Smolley, S. (2017). Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802.

[Mao et al., 2016] Mao, X.-J., Shen, C., and Yang, Y.-B. (2016). Image restoration using convolutional auto-encoders with symmetric skip connections. *arXiv preprint arXiv:1606.08921*.

[Minsky and Papert, 1969] Minsky, M. and Papert, S. (1969). Perceptrons cambridge. *MA: MIT Press. zbMATH*.

[Mirza and Osindero, 2014] Mirza, M. and Osindero, S. (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

[Nielsen and Nock, 2013] Nielsen, F. and Nock, R. (2013). On the chi square and higher-order chi distances for approximating f-divergences. *IEEE Signal Processing Letters*, 21(1):10–13.

[Notebook, 2020] Notebook, J. (2020 (accessed June 1, 2020)). The jupyter notebook. https://jupyter.org/.

[NVIDIA, 2020] NVIDIA (2020 (accessed June 1, 2020)). Nvidia tesla k80. https://www.nvidia.com/en-gb/data-center/tesla-k80/.

[Nwankpa et al., 2018] Nwankpa, C., Ijomah, W., Gachagan, A., and Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*.

[Odena et al., 2017] Odena, A., Olah, C., and Shlens, J. (2017). Conditional image synthesis with auxiliary classifier gans. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2642–2651. JMLR. org.

[Pascanu et al., 014a] Pascanu, R., Gülçehre, , Cho, K., and Bengio, Y. (2014a). How to construct deep recurrent neural networks.

[Pearson, 1900] Pearson, K. (1900). X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 50(302):157–175.

[Piccinini, 2004] Piccinini, G. (2004). The first computational theory of mind and brain: a close look at mcculloch and pitts's "logical calculus of ideas immanent in nervous activity". *Synthese*, 141(2):175–215.

[PyTorch, 2020] PyTorch (2020 (accessed June 1, 2020)). Pytorch. https://github.com/pytorch/pytorch#releases-and-contributing.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

[Russel et al., 2013] Russel, S., Norvig, P., et al. (2013). *Artificial intelligence: a modern approach*. Pearson Education Limited.

[Salakhutdinov and Hinton, 2009] Salakhutdinov, R. and Hinton, G. (2009). Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978.

[Salimans et al., 2016] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242.

[Shalev-Shwartz and Ben-David, 2014] Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding Machine Learning From Theory to Algorithms*. Cambridge University Press.

[Shin et al., 2018] Shin, H.-C., Tenenholtz, N. A., Rogers, J. K., Schwarz, C. G., Senjem, M. L., Gunter, J. L., Andriole, K. P., and Michalski, M. (2018). Medical image synthesis for data augmentation and anonymization using generative adversarial networks. In *International workshop on simulation and synthesis in medical imaging*, pages 1–11. Springer.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Khrizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). A simple way to prevent neural networks from overfitting.

[Subramanian et al., 2018] Subramanian, S., Mudumba, S. R., Sordoni, A., Trischler, A., Courville, A. C., and Pal, C. (2018). Towards text generation with adversarially learned neural outlines. In *Advances in Neural Information Processing Systems*, pages 7551–7563.

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

[Tjoa and Guan, 2019] Tjoa, E. and Guan, C. (2019). A survey on explainable artificial intelligence (xai): towards medical xai. *arXiv preprint arXiv:1907.07374*.

[Tolstikhin et al., 2017] Tolstikhin, I., Bousquet, O., Gelly, S., and Schoelkopf, B. (2017). Wasserstein auto-encoders. *arXiv preprint arXiv:1711.01558*.

[Torch, 2020] Torch, K. (2019 (accessed June 1, 2020)). Torchvision.datasets. https://pytorch.org/docs/stable/torchvision/datasets.html.

[Wang and Manning, 2013] Wang, S. and Manning, C. (2013). Fast dropout training.

[Xiao et al., 2017] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

[Xie, 2017] Xie, Z. (2017). Neural text generation: A practical guide. *arXiv preprint arXiv:1711.09534*.

[Zhao et al., 2016] Zhao, J., Mathieu, M., and LeCun, Y. (2016). Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126*.

LAMPIRAN

LAMPIRAN 1: *UNSUPERVISED LEAST SQUARE ADVERSARIAL AUTOENCODER*

Implementasi *Encoder* pada *Unsupervised Least Square Adversarial Autoencoder*

```
'''
Implementation of encoder on CNN Least Square Adversarial Autoencoder
The network consists of:
    * 4 convolutional layer (input_channel, output_channel):
        - (x-dimension, 40)
        - (40, 80)
        - (80, 160)
        - (160, 320)
    * 1 fully connected layer (input_dimension, output_dimension):
        - (320, 20)
    * 3 batch normalization layer (input_channel):
        - 80
        - 160
        - 320
    * parameters:
        - z_dimension = 20; dimensionality of latent vector
        - x_channel = 1; number of input's channel (grayscale)

The output of network is an 20-dimension of latent vector (G(Z|X))

The non-linearity of the output of each convolutional layer is controlled
by LeakyRelu function with negative slope = 0.2
'''


x_channel = 1
z_dimension = 20

class Encoder_net(nn.Module):
    def __init__(self):
        super(Encoder_net, self).__init__()
        self.conv1 = nn.Conv2d(x_channel, 40, 4, stride=2, padding=1, bias=False)
        self.conv2 = nn.Conv2d(40, 80, 4, stride=2, padding=1, bias=False)
        self.conv3 = nn.Conv2d(80, 160, 4, stride=2, padding=1, bias=False)
        self.conv4 = nn.Conv2d(160, 320, 4, stride=2, padding=1, bias=False)

        self.bn1 = nn.BatchNorm2d(80)
        self.bn2 = nn.BatchNorm2d(160)
        self.bn3 = nn.BatchNorm2d(320)

        self.linear1 = nn.Linear(320, z_dimension)

    def forward(self, x):
```

```
        x = self.conv1(x)
        x = F.leaky_relu(x, negative_slope=0.2, inplace=True)
        x = self.conv2(x)
        x = self.bn1(x)
        x = F.leaky_relu(x, negative_slope=0.2, inplace=True)
        x = self.conv3(x)
        x = self.bn2(x)
        x = F.leaky_relu(x, negative_slope=0.2, inplace=True)
        x = self.conv4(x)
        x = self.bn3(x)
        x = F.leaky_relu(x, negative_slope=0.2, inplace=True)
        x = x.squeeze()

        z = self.linear1(x)
        return z
```

Kode 1: Implementasi Kode *Encoder* pada *Unsupervised Least AAE*

Implementasi *Decoder* pada *Unsupervised Least Square Adversarial Autoencoder*

```
'''
Implementation of deccoder on CNN Least Square Adversarial Autoencoder
The network consists of:
    * 3 convolutional layer (input_channel, output_channel):
        - (320, 160)
        - (160, 80)
        - (80, 1)
    * 1 fully connected layer (input_dimension, output_dimension):
        - (z_dimension, 40 * 8 * 7 * 7)
    * 3 batch normalization layer (input_channel):
        - 160
        - 80
    * parameters:
        - z_dimension = 20; dimensionality of latent vector
        - x_channel = 1; number of input's channel (grayscale)

The output of network is an 784-dimension of reconstructed input vector (Q(X|Z))

The non-linearity of the output of each convolutional layer is controlled
by LeakyRelu function with negative slope = 0.2, except the last layer of convolutional layer
(using sigmoid instead)
'''

z_dimension = 20
class Decoder_net(nn.Module):
    def __init__(self):
        super(Decoder_net, self).__init__()
        self.layer1 = nn.Linear(z_dimension, 40 * 8 * 7 * 7)

        self.conv1 = nn.ConvTranspose2d(320, 160, 4, stride=1)
        self.conv2 = nn.ConvTranspose2d(160, 80, 4, stride=1)
        self.conv3 = nn.ConvTranspose2d(80, 1, 4, stride=2)

        self.bn1 = nn.BatchNorm2d(160)
        self.bn2 = nn.BatchNorm2d(80)


    def forward(self, x):
        x = self.layer1(x)
        x = x.view(-1, 320, 7, 7)
        x = self.conv1(x)
        x = self.bn1(x)
        x = F.leaky_relu(x)
        x = self.conv2(x)
        x = self.bn2(x)
        x = F.leaky_relu(x)
        x = self.conv3(x)
        x = F.sigmoid(x)
        return x
```

Kode 2: Implementasi Kode *Decoder* pada *Unsupervised Least AAE*

Implementasi Diskriminator pada *Unsupervised Least Square Adversarial Autoencoder*

```
'''
Implementation of discriminator on CNN Least Square Adversarial Autoencoder
The network consists of:
    * 5 fully connected layer (input_dimension, output_dimension):
        - (z_dimension, 160)
        - (160, 160)
        - (160, 160)
        - (160, 160)
        - (160, 1)
    * parameters:
        - z_dimension = 20; dimensionality of latent vector

The output of network is an 1-dimension of discrimination label

The non-linearity of the output of each convolutional layer is controlled
by LeakyRelu function with negative slope = 0.2, except the last layer of fully connected layer
(using sigmoid instead)
'''

z_dimension = 20
class Discriminator_net_gauss(nn.Module):
    def __init__(self):
        super(Discriminator_net_gauss, self).__init__()
        self.linear1 = nn.Linear(z_dimension, 160)
        self.linear2 = nn.Linear(160, 160)
        self.linear3 = nn.Linear(160, 160)
        self.linear4 = nn.Linear(160, 160)
        self.linear5 = nn.Linear(160, 1)

    def forward(self, x):
        x = self.linear1(x)
        x = F.leaky_relu(x, negative_slope=0.2, inplace=True)
        x = self.linear2(x)
        x = F.leaky_relu(x, negative_slope=0.2, inplace=True)
        x = self.linear3(x)
        x = F.leaky_relu(x, negative_slope=0.2, inplace=True)
        x = self.linear4(x)
        x = F.leaky_relu(x, negative_slope=0.2, inplace=True)
        x = self.linear5(x)
        x = F.sigmoid(x)

        return x
```

Kode 3: Implementasi Kode *Diskriminator* pada *Unsupervised Least AAE*

Implementasi Pembelajaran *Unsupervised Least Square Adversarial Autoencoder*

```
'''
Algorithm to train CNN least square Adversarial Autoencoder for one epoch
The sample is shuffled in several minibatches

encoder, decoder, and discriminator trained using mean square error

parameters:
TINY_ERROR = 1e-8; adding error value to avoid
TRAIN_BATCH_SIZE = 200; number of sample on one minibatch
learning rate for encoder / generator, decoder, and discriminator = 0.0002 using Adam Optimizer
prior distribution = N(0, 1)
lambda (weight loss) for discriminator and generator = 0.01

training method using free_params and frozen_params method as helper method
free_params method turn the parameter on during training
frozen_params method turn the parameter off during training
'''

TRAIN_BATCH_SIZE = 200
TINY_ERROR = 1e-8
Tensor = torch.cuda.FloatTensor
cuda = torch.device('cuda:0')

def free_params(module: nn.Module):
    for p in module.parameters():
        p.requires_grad = True

def frozen_params(module: nn.Module):
    for p in module.parameters():
        p.requires_grad = False

def train_one_epoch(decoder, encoder, discriminator_gauss, decoder_optimizer,
                    encoder_optimizer, discriminator_optimizer, train_loader):
    encoder.train()
    decoder.train()
    discriminator_gauss.train()

    reconstruction_loss = None
    real_loss = None
    fake_loss = None
    generator_loss = None

    for X, target in train_loader:
        encoder.zero_grad()
        decoder.zero_grad()
        discriminator_gauss.zero_grad()

        valid = Variable(Tensor(TRAIN_BATCH_SIZE, 1).fill_(0.9), requires_grad=False)
        fake = Variable(Tensor(TRAIN_BATCH_SIZE, 1).fill_(0.1), requires_grad=False)

        X = X.resize(TRAIN_BATCH_SIZE, 1, 28, 28)
        X, target = Variable(X), Variable(target)
```

```
53
54          if is_cuda:
55              X, target = X.cuda(cuda), target.cuda(cuda)
56
57
58          mse_loss = torch.nn.MSELoss()
59
60          frozen_params(decoder)
61          frozen_params(encoder)
62          free_params(discriminator_gauss)
63
64          z_real = torch.randn(TRAIN_BATCH_SIZE, z_dimension) * 1.0
65          z_real = z_real.cuda(cuda)
66          real_value = discriminator_gauss(z_real)
67
68          z_fake = encoder(X)
69          fake_value = discriminator_gauss(z_fake)
70
71          real_loss = 0.01 * mse_loss(real_value + TINY_ERROR, valid)
72          fake_loss = 0.01 * mse_loss(fake_value + TINY_ERROR, fake)
73
74          real_loss.backward()
75          fake_loss.backward()
76
77          discriminator_optimizer.step()
78
79          free_params(decoder)
80          free_params(encoder)
81          frozen_params(discriminator_gauss)
82
83          z = encoder(X)
84          x_hat = decoder(z)
85
86          z_2 = encoder(Variable(X.data))
87          z_2_dis = discriminator_gauss(z_2)
88
89          x_hat_resize = x_hat.view(-1, 784)
90          X_resize = X.view(-1, 784)
91
92          reconstruction_loss = mse_loss(x_hat_resize + TINY_ERROR, X_resize)
93          generator_loss = 0.01 * mse_loss(z_2_dis + TINY_ERROR, valid)
94
95          reconstruction_loss.backward()
96          generator_loss.backward()
97
98          encoder_optimizer.step()
99          decoder_optimizer.step()
100
101
102
103     return generator_loss, real_loss + fake_loss, reconstruction_loss
```

Kode 4: Implementasi Kode Pembelajaran pada *Unsupervised Least AAE*

LAMPIRAN 2: *SUPERVISED LEAST SQUARE ADVERSARIAL AUTOENCODER*

Implementasi *Encoder* pada *Supervised Least Square Adversarial Autoencoder*

```
'''
Implementation of encoder on Supervised Least Square Adversarial Autoencoder
The network consists of:
    * 3 fully connected layer (input_dimension, output_dimension):
        - (z_dimension, N)
        - (N, N)
        - (N, z_dimension)
    * parameters:
        - N = 1000; dimensionality of hidden layer vector
        - z_dimension = 2; dimensionality of latent_vector

The output of network is an 2-dimension of latent vector (G(Z|X))

The non-linearity of the output of each convolutional layer is controlled
by Relu function, except the last layer

Each layer is regularized using dropout, except the last layer with bernouli probability = 0.5
'''

X_dimension = 784
N = 1000
z_dimension = 2

class Encoder_net(nn.Module):
    def __init__(self):
        super(Encoder_net, self).__init__()
        self.layer1 = nn.Linear(X_dimension, N)
        self.layer2 = nn.Linear(N, N)
        self.layer3 = nn.Linear(N, z_dimension)

    def forward(self, x):
        x = F.dropout(self.layer1(x), p=0.5, training=self.training)
        x = F.relu(x)
        x = F.dropout(self.layer2(x), p=0.5, training=self.training)
        x = F.relu(x)
        x = self.layer3(x)

        return x
```

Kode 5: Implementasi Kode *Encoder* pada *Unsupervised Least AAE*

Implementasi *Decoder* pada *Supervised Least Square Adversarial Autoencoder*

```
'''
Implementation of decoder on Supervised Least Square Adversarial Autoencoder
The network consists of:
    * 3 fully connected layer (input_dimension, output_dimension):
        - (z_dimension + n_classes, N)
        - (N, N)
        - (N, X_dimension)
    * parameters:
        - N = 1000; dimensionality of hidden layer vector
        - z_dimension = 2; dimensionality of latent_vector
        - n_classes = 10; dimensionality of sample's target, represented as one hot vector

The output of network is an 784-dimension of reconstructed input vector (Q(X|Z))

The non-linearity of the output of each convolutional layer is controlled
by Relu function, except the last layer
(using sigmoid instead)

Each layer is regularized using dropout, except the last layer with bernouli probability = 0.5
'''

n_classes = 10
z_dimension = 2
N = 1000
X_dimension = 784

class Decoder_net(nn.Module):
    def __init__(self):
        super(Decoder_net, self).__init__()
        self.layer1 = nn.Linear(z_dimension + n_classes, N)
        self.layer2 = nn.Linear(N, N)
        self.layer3 = nn.Linear(N, X_dimension)

    def forward(self, x):
        x = self.layer1(x)
        x = F.dropout(x, p=0.5, training=self.training)
        x = F.relu(x)
        x = self.layer2(x)
        x = F.dropout(x, p=0.5, training=self.training)
        s = F.relu(x)
        x = self.layer3(x)
        return F.sigmoid(x)
```

Kode 6: Implementasi Kode *Decoder* pada *Supervised Least AAE*

Implementasi Diskriminator pada *Supervised Least Square Adversarial Autoencoder*

```
'''
Implementation of discriminator on Supervised Least Square Adversarial Autoencoder
The network consists of:
    * 3 fully connected layer (input_dimension, output_dimension):
        - (z_dimension, N)
        - (N, N)
        - (N, 1)
    * parameters:
        - N = 1000; dimensionality of hidden layer vector
        - z_dimension = 2; dimensionality of latent_vector
        - n_classes = 10; dimensionality of sample's target, represented as one hot vector

The output of network is an 1-dimension of discrimination vector

The non-linearity of the output of each convolutional layer is controlled
by Relu function, except the last layer

Each layer is regularized using dropout, except the last layer with bernouli probability = 0.5
'''

z_dimension = 2
N = 1000

class Discriminator_net_gauss(nn.Module):
    def __init__(self):
        super(Discriminator_net_gauss, self).__init__()
        self.layer1 = nn.Linear(z_dimension, N)
        self.layer2 = nn.Linear(N, N)
        self.layer3 = nn.Linear(N, 1)

    def forward(self, x):
        x = F.dropout(self.layer1(x), p=0.5, training=self.training)
        x = F.relu(x)
        x = F.dropout(self.layer2(x), p=0.5, training=self.training)
        x = F.relu(x)

        return self.layer3(x)
```

Kode 7: Implementasi Kode *Diskriminator* pada *Supervised Least AAE*

Implementasi Pembelajaran *Supervised Least Square Adversarial Autoencoder*

```
'''
Algorithm to train supervised least square Adversarial Autoencoder for one epoch
The sample is shuffled in several minibatches

encoder, decoder, and discriminator trained using mean square error

parameters:
TINY_ERROR = 1e-8; adding error value to avoid
TRAIN_BATCH_SIZE = 100; number of sample on one minibatch
learning rate for encoder / generator, decoder, and discriminator = 0.0002 using Adam Optimizer
prior distribution = N(0, 5)

'''

X_dimension = 784
TRAIN_BATCH_SIZE = 100
cuda = torch.device('cuda:0')

def train_one_epoch(decoder, encoder, discriminator_gauss, decoder_optimizer,
                    encoder_optimizer, generator_optimizer, discriminator_optimizer,
                    data_loader):

    encoder.train()
    decoder.train()
    discriminator_gauss.train()

    for X, target in data_loader:
        X = X * 0.3081 + 0.1307
        X.resize_(TRAIN_BATCH_SIZE, X_dimension)
        X, target = Variable(X), Variable(target)
        if cuda:
            X, target = X.cuda(cuda), target.cuda(cuda)

        # Init gradients
        decoder.zero_grad()
        encoder.zero_grad()
        discriminator_gauss.zero_grad()


        z_gauss = encoder(X)

        category = np.array(target.data.tolist())
        category = np.eye(n_classes)[category].astype('float32')
        category = torch.from_numpy(category)
        z_category = Variable(category)

        if cuda:
            z_category = z_category.cuda(cuda)

        z_sample = torch.cat((z_category, z_gauss), 1)

        X_sample = decoder(z_sample)
```

```
        compared_with_original = X.resize(TRAIN_BATCH_SIZE, X_dimension)
        mse_loss = torch.nn.MSELoss()
        reconstruction_loss = mse_loss(X_sample + TINY_ERROR, compared_with_original + TINY_ERROR)

        reconstruction_loss.backward()
        decoder_optimizer.step()
        encoder_optimizer.step()

        decoder.zero_grad()
        encoder.zero_grad()
        discriminator_gauss.zero_grad()

        # Discriminator
        encoder.eval()
        z_real_gauss = Variable(torch.randn(TRAIN_BATCH_SIZE, z_dimension) * 5.)
        if cuda:
            z_real_gauss = z_real_gauss.cuda(cuda)

        z_fake_gauss = encoder(X)

        discriminator_real_gauss = discriminator_gauss(z_real_gauss)
        discriminator_fake_gauss = discriminator_gauss(z_fake_gauss)

        discriminator_loss = 0.5 * (torch.mean((discriminator_real_gauss + TINY_ERROR - 1)**2) + torch.mean

        discriminator_loss.backward()
        discriminator_optimizer.step()

        decoder.zero_grad()
        encoder.zero_grad()
        discriminator_gauss.zero_grad()

        # Generator
        encoder = encoder.train()
        z_fake_gauss = encoder(X)

        generator_fake_gauss = discriminator_gauss(z_fake_gauss)
        generator_loss = 0.5 * torch.mean((generator_fake_gauss + TINY_ERROR - 1)**2)

        generator_loss.backward()
        generator_optimizer.step()

        decoder.zero_grad()
        encoder.zero_grad()
        discriminator_gauss.zero_grad()

    return discriminator_loss, generator_loss, reconstruction_loss
```

Kode 8: Implementasi Kode Pembelajaran pada *Supervised Least AAE*

LAMPIRAN 3: *CONVOLUTIONAL CLASSIFIER PADA MNIST*

Implementasi Arsitektur *Convolutional Classifier* pada MNIST

```
'''
Implementation of CNN classifier
The network consists of:
    * 2 convolutional layer (input_channel, output_channel):
        - (x_channel, 20)
        - (20, 50)
    * 2 fully connected layer (input_dimension, output_dimension):
        - (4 * 4 * 50, 500)
        - (500, 10)
    * 2 max-pool layer (mask size)
        - (2, 2)
        - (2, 2)
    * parameters:
        - x_channel = 1; number of input's channel (grayscale)

The output of network is an 10-dimension of output vector

The non-linearity of the output of each convolutional layer is controlled
by Relu function except last layer (using softmax instead)
'''

x_channel = 1
class Convolutional_Net(nn.Module):
    def __init__(self):
        super(Convolutional_Net, self).__init__()
        self.convolutional1 = nn.Conv2d(x_channel, 20, 5, 1)
        self.convolutional2 = nn.Conv2d(20, 50, 5 , 1)
        self.linear1 = nn.Linear(4*4*50, 500)
        self.linear2 = nn.Linear(500, 10)

    def forward(self, x):
        x = F.relu(self.convolutional1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.convolutional2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 4*4*50)
        x = F.relu(self.linear1(x))
        x = self.linear2(x)
        return F.log_softmax(x, dim=1)
```

Kode 9: Implementasi Kode *Convolutional Classifier* pada MNIST

Implementasi Pembelajaran *Convolutional Classifier pada MNIST*

```
'''
Algorithm to train convolutional classifier
The sample is shuffled in several minibatches

encoder, decoder, and discriminator trained using mean square error

parameters:
TINY_ERROR = 1e-8; adding error value to avoid
TRAIN_BATCH_SIZE = 100; number of sample on one minibatch
learning rate = 0.001 using Stochastic Gradient Descent Optimization
momentum = 0.5

'''

cuda = torch.device('cuda:0')

def train(model, train_loader, optimizer, epoch):
    model = model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.cuda(cuda), target.cuda(cuda)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
```

Kode 10: Implementasi Kode Pembelajaran *Convolutional Classifier* pada MNIST