

Deep Reinforcement Learning

Self Organising Map (SOM)

Envolving Deep Neural Network

*DEEP LEARNING*

Tugas 2

**YAYASAN MEMAJUKAN ILMU DAN KEBUDAYAAN**

**UNIVERSITAS SIBER ASIA**

Kampus Menara, Jl. RM. Harsono, Ragunan - Jakarta Selatan.Daerah Khusus Ibukota Jakarta 12550.

Telp. (+6221) 27806189. [asiacyberuni@acu.ac.id. www.unsia.ac.id](mailto:asiacyberuni@acu.ac.id)

Mata Kuliah : Deep Learning

Kelas : IT501

Prodi : PJJ Informatika

Nama Mahasiswa : Hendro Gunawan

NIM : 200401072103

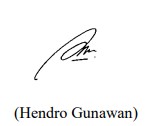
Dosen : Catur Nogroho, S.Kom.,M.Kom.

Kata Pengantar

Dalam pertemuan kali ini saya akan membahas tentang tiga hal penting dalam pembelajaran *Deep Learning* yaitu materi yang telah disampaikan oleh dosen saya Bapak Catur Nugroho S.Kom., M.Kom selaku dosen yang mengajar mata kuliah *Deep Learning* di kampus UNSIA. Pada makalah ini saya menggunakan metode penelitian yang bersumber dari materi perkuliahan pertemuan kesatu sampai dengan pertemuan keenambelas dan dari browsing di internet. Kami telah diberikan materi pada sesi ke satu yang membahas tentang Pengenalan *Deep Learning* (DL), kemudian materi pada pertemuan kedua yang membahas tentang Jaringan Syaraf Tiruan (JST), lalu pada pertemuan ketiga membahas tentang *Convolution Neural Network*, kemudian pada pertemuan keempat yang membahas tentang konsep *Deep Unsupervised Learning* (DUL), pada pertemuan kelima membahas tentang konsep *Back Propagations Algorithm*, selanjutnya pada pertemuan keenam yang membahas tentang *Capsule Networking*, terus ada lagi pertemuan ketujuh yang membahas *tentang Recurrent Neural Networks* (RNNs), kemudian pada pertemuan kesembilan yang membahas tentang *Reinforcement Learning* (RL), dilanjutkan pertemuan kesepuluh yaitu membahas tentang *Self Organizing MAP* (SOM), dan pertemuan kesebelas membahas tentang *Envolving Deep Neural Network* (EDNN). Di sini saya hanya akan membahas tiga bab saja yaitu materi pertemuan kesembilan, kesepuluh dan kesebelas. Semoga dengan adanya makalah ini kita bisa menambah wawasan tentang mata kuliah *Deep Learning*. Koreksi, komentar, kritik, dan saran dapat disampaikan melalui surel (*email*) ke alamat: [hendro.gnwn@gmail.com](mailto:hendro.gnwn@gmail.com), [hendro.gnwn@ymail.com](mailto:hendro.gnwn@ymail.com), atau [hendro.gnwn@outlook.com](mailto:hendro.gnwn@outlook.com). Semoga mendapat manfaat dari makalah ini.

Gresik, 26 Juni 2023

Penulis



Daftar Isi

Kata Pengantar..................................................................................................i

Daftar Isi..........................................................................................................ii

BAB I PENDAHULUAN................................................................................1

BAB II TINJAUAN PUSTAKA……………………………………………..2

BAB III PEMBAHASAN……………………………………………………4

BAB IV PENUTUP…………………………………………………………36

DAFTAR PUSTAKA………………………………………………………..37

**BAB I**

**PENDAHULUAN**

Pada kompetisi citra *ImageNet Large Scale Visual Recognition Competition* (ILSVRC) tahun 2012, dunia komputasi dikejutkan oleh munculnya *deep learning* (pembelajaran mendalam). Teknik ini menjadi pemenang pertama dengan akurasi yang jauh lebih tinggi dibanding semua teknik machine learning yang ada selama ini, yang disebut *shallow learning* (pembelajaran dangkal) (Lab, 2018). *Deep learning* menjadi teknik disruptif yang mengubah segala paradigma lama dalam *machine learning* konvensional. Beragam model *deep learning*, yang sebenarnya adalah pembaruan model-model *machine learning* konvensional (*shallow learning*), mampu memberikan performansi sangat tinggi dalam menangani bermacam himpunan data, khususnya himpunan data yang sangat besar. Jadi, *deep learning* bisa dikatakan sebagai modernisasi *machine learning* untuk menangani *big data*.

* 1. **Latar Belakang Masalah**
     1. **Mengapa Deep Learning?**

Semua teknik dan metode machine learning konvensional, mulai dari regresi, decision tree learning, naïve bayes, artificial neural network, hingga ensemble learning, sering kali disebut shallow learning. Teknik shalow learning umumnya mengharuskan adanya metode ekstraksi fitur yang tepat, yang sering kali sangat rumit, untuk menghasilkan akurasi tinggi. Hal ini membuatnya kurang praktis. Masalah lainnya adalah penambahan data latih umumnya tidak memberikan peningkatan akurasi yang sebanding dengan usaha untuk pencarian data tersebut.

Untuk itu, para ahli berusaha menemukan teknik pembelajaran lain yang lebih efisien dan memberikan peningkatan akurasi yang signifikan sebanding dengan penambahn jumlah data latih. Akhirnya pada tahun 2006 lahirlah suatu teknik pembelajaran baru yang disebut deep learning ( DL), dengan kemampuan featur engineering yang dapat merekayasa fitur secara otomatis. Selain itu, deep learning terbukti mampu memberikan peningkatan akurasi yang sebanding dengan penambahan jumlah data.

DL pertama kali diperkenalkan oleh Geoffrey Hinton pada tahun 2006, dengan sebuah jaringan saraf yang disebut deep belief nets. Perkembangan DL semakin cepat setelah ditemukan teeknik implementasi yang lebih praktis menggunakan graphic processing unit (GPU) pada tahun 2009. GPU adalah sebuah prosesor khusus untuk pengolahan grafis yang banyak digunakan pada aplikasi-aplikasi games. Proses pembelajaran DL dapat dilakukan secara paralel dengan memanfaatkan GPU sehingga jauh lebih cepat dibanding secara sekuensial hanya mengandalkan prossor CPU.

DL telah menjadi salah satu bidang yang paling banyak diteliti dan diaplikasikan. Hampir setiap bulan muncul konsep dan arsitektur baru yang lebih matang. Di tahun 2019, DL semakin banyak diaplikasikan di berbagai bidang. Hal ini karena SDL memilikibanyak kelebihan, diantaranya adalah universal, tahan terhadap variasi data, serta memiliki tingkat generalisasi dan skapabilitas yang tinggi

* + 1. **Definisi Deep Learning**

Definisi DL yang bisa berlaku bagi banyak orang agak sulit ditentukan karena telah berubah bentuk selama beberapa tahun terakhir. Salah satu definisi yang sederhana: “*DL adalah setiap jaringan saraf dengan lebih dari dua lapisan.*”

Definisi tersebut secara implisit menyatakan bahwa DL seolah-olah telah ada sejak tahun 1980-an. Namun, ada empat hal dalam evolusi jaringan saraf ini: lebih banyak neuron daripada jaringan sebenarnya, cara yang lebih kompleks untuk menghubungkan lapisan, ledakan dalam jumlah daya komputasi yang tersedia untuk melatih, dan ekstraksi fitur otomatis.

Ide-ide DL ini sebenarnya telah ada sejak lama. Sebagai contoh, *Restricted Boltzmann Machine* (RBM) merupakan pembaruan dari konsep jaringan saraf Hopfield. Sejumlah RBM bisa ditumpuk menjadi beberapa lapisan. Penumpukan RBM ini ternyata memberikan peningkatan performansi yang sangat signifikan. Ide inilah yang menjadi titik awal munculnya DL. Selanjutnya para pakar semakin bersemangat membangun model-model DL yang lebih andal, mulai dari *Konvolutional Neural Networks* (CNN) *Capsule networks* atau *CapNet*, *Deep Recurrent Networks*, hingga *Deep Reinforcement Learning* (DRL). Jadi, DL bisa dikatakan sebagai modernisasi machine learning untuk menangani *big data*.

**1.2 Rumusan Masalah**

Dengan melihat latar belakang masalah yang telah dikemukakan maka, beberapa masalah yang dapat penulis rumuskan dan akan dibahas dalam laporan ini adalah:

1. Apakah pengertian Deep Reinforcement Learning itu?

2. Apakah pengertian Self Organizing MAP (SOM) itu?

3. Apakah pengertian Envolving Deep Neural Network itu?

**1.3 Tujuan Dan Manfaat**

Tujuan dan manfaat penelitian yang ingin dicapai adalah:

1. Mengetahui bagaimana algoritma Deep Reinforcement Learning?

2. Mengetahui bagaimana konsep dasar Self Organising MAP (SOM)?

3. Mengetahui bagaimana metode Envolving Deep Neural Network?

**1.4 Metode penelitian**

Metode yang digunakan oleh penulis dalam menyusun makalah ini yaitu dengan mengumpulkan informasi dari berbagai sumber buku dan browsing di internet.

**1.5 Estetika Penulisan**

Dalam penyusunan makalah ini terdiri dari hal-hal yang saling berkaitan antara bab I sampai dengan bab IV yang memuat beberapa isi sebagai berikut:

**BAB I Pendahuluan**

Membahas tentang latar belakang masalah, rumusan masalah, tujuan penulisan dan sistematika penulisan.

**BAB II Tinjauan Pustaka**

Membahas tentang tinjauan tentang algoritma Deep Reinforcement Learning, konsep dasar Self Organising MAP (SOM) dan metode Envoving Deep Neural Network.

**BAB III Pembahasan**

Membahas tentang apakah pengertian Deep Reinforcement Learning?, apakah pengertian Self Organising MAP (SOM)?, apakah pengertian Envolving Deep Neural Network?

**BAB IV Penutup**

Membahas tentang kesimpulan, saran, dan daftar pustaka.

**BAB II**

**TINJAUAN PUSTAKA**

**2.1 Pengertian Deep Reinforcement Learning**

*Reinforcement Learning* (RL) adalah jenis algoritma machine learning yang berada diantara supervised dan unsupervised. *Reinforcement Learning* (RL) tidak dapat diklasifikasikan sebagai supervised learning karena tidak hanya mengandalkan kumpulan data pelatihan berlabel, tetapi juga bukan unsupervised learning dalam pembelajaran ini digunakan untuk memaksimalkan tindakan yang “benar” untuk diambil dalam berbagai skenario. Untuk memahami *Deep Reinforcement Learning* dengan lebih baik, bayangkan computer-mu ingin bermain catur denganmu. Pertanyaan pertama yang harus ditanyakan adalah: Apakah mungkin jika mesin dilatih dengan cara yang supervised? Secara teori, iya bisa, tetapi ada dua kelemahan yang perlu jadi pertimbangan. Pertama, untuk maju dengan supervised learning, diperlukan kumpulan data yang relevan. Kedua, jika kita melatih mesin untuk meniru perilaku manusia dalam permainan catur, mesin tidak akan pernah lebih baik dari manusia, karena hanya mereplikasi perilaku yang sama. Jadi menurut definisi, kita tidak dapat menggunakan supervised learning untuk melatih mesin. *Reinforcement Learning* adalah jenis machine learning algorithm yang belajar untuk memecahkan masalah multi-level dengan coba-coba. Mesin dilatih pada skenario kehidupan nyata untuk membuat urutan keputusan. Yang dimaksud dengan *Deep Reinforcement Learning* adalah beberapa lapisan Jaringan Syaraf Tiruan (*Artificial Neural Network*) yang ada dalam arsitektur untuk mereplikasi kerja otak manusia.

**2.2 Pengertian Self Organising MAP (SOM)**

*Self Organizing Map* (SOM) adalah salah satu tool yang sangat baik dalam penanganan data. Sering disebut topology-preserving map pertama kali diperkenalkan oleh Teuvo Kohonen tahun 1996. SOM merupakan salah satu teknik dalam Neural Network yang bertujuan untuk melakukan visualisasi data dengan cara mengurangi dimensi data melalui penggunaan self-organizing neural networks sehingga manusia dapat mengerti high-dimensional data yang dipetakkan dalam bentuk low-dimensional data. *Self Organizing Map* (SOM) adalah salah jenis algoritma pembelajaran tak terawasi (unsupervised learning) dalam bidang kecerdasan buatan (artificial inteligent) dan pemrosesan data. SOM digunakan untuk memetakkan data multidimensi ke dalam ruang dua dimensi atau tiga dimensi dengan tujuan menggambarkan hubungan atau struktur yang tersembunyi dalam data tersebut.

Prinsip dasar SOM adalah mengorganisasi dan mengklasifikasikan data ke dalam unit-unit yang terorganisir dalam bentuk grid atau jaringan. Setiap unit dalam jaringan mewakili neuron atau titik yang berfungsi untuk menangkap dan menggambarkan pola atau karakteristik dari data yang diberikan. SOM menggunakan metode pembelajaran yang kompetitif dan kooperatif dimana setiap unit berkompetisi untuk menjadi pemenang atau unit yang paling responsif terhadap input data tertentu, dan kemudian unit-unit tetangga bekerja sama untuk mengatur dan mengatur diri mereka sendiri agar lebih sesuai dengan data tersebut.

**2.3 Pengertian Envolving Deep Neural Network**

*Envolving Deep Neural Networks* (EDNN) adalah konsep yang mengacu pada pengembangan atau evolusi jaringan saraf dalam arsitektur *Deep Neural Network* (DNN). EDNN melibatkan penggunaan algoritma evolusi untuk mengoptimalkan struktur dan parameter jaringan saraf secara otomatis.

Pada dasrnya, EDNN menggunakan prinsip dasar algoritma genetika atau algoritma evolusi lainnya untuk menghasilkan variasi dalam arsitektur jaringan saraf dan mengevaluasi kinerja setiap variasi berdasarkan kriteria tertentu. Variasi yang memiliki kinerja yang lebih baik kemudian dipilih dan diteruskan ke generasi berikutnya, sedangkan variasi yang kurang baik dapat dieliminasi.

Tujuan utama dari EDNN adalah untuk secara otomatis mengoptimalkan arsitektur dan parameter jaringan saraf untuk tugas pembelajaran tertentu. Dengan melakukan evolusi jaringan saraf, EDNN memungkinkan adaptasi dan peningkatan jaringan saraf seiring dengan kemajuan iteratif melalui generasi.

Secara keseluruhan, Envolving Deep Neural Network (EDNN) adalah pendekatan yang menggabungkan algoritma evolusi dengan jaringan saraf dalam arsitektur Deep Neural Network (DNN) untuk mencapai optimasi otomatis dan peningkatan kinerja jaringan saraf dalam tugas pembelajaran mesin.

**BAB III**

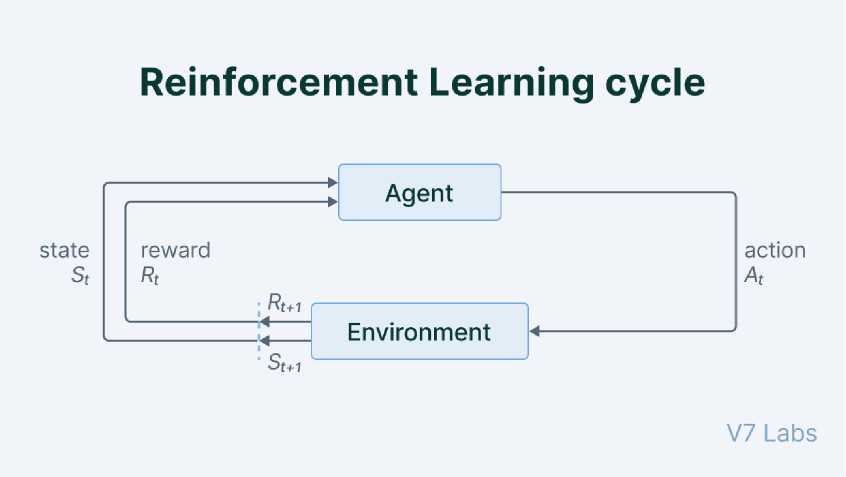
**PEMBAHASAN**

**3.1 Deep Reinforcement Learning**

*Deep Reinforcement Learning* (DRL) merupakan temuan penting dalam deep learning. Teknik ini merupakan gabungan *Deep Learning* dengan *Reinforcement Learning*. Pada bab ini kita akan mendiskusikan DRL, sebuah teknik pembelajaran DL yang memungkinkan suatu program mampu mempelajari lingkungan secara langsung tanpa terlebih dahulu melakukan proses *learning* yang umumnya menggunakan banyak sekali data latih. Program berbasis DRL tersebut mempelajari lingkungan berdasarkan *reward* (hadiah atau imbalan) positif dan negatif dengan cara memaksimalkan *cumulative reward* (hadiah atau imbalan yang berhasil dikumpulkan secara kumulatif). Dengan kata lain, program berbasis DRL **belajar dari interaksi** dengan lingkungannya. Dalam bukunya, Richard S. Sutton dan Andrew G. Barto menyatakan: “Belajar dari interaksi merupakan ide mendasar untuk sebagian besar teknik pembelajaran dan kecerdasan” (Sutton and Barto, 2017).

**3.1.1. Ide Dasar Reinforcement Learning**

Pendekatan machine learning yang terkenal adalah supervised learninmg dan unsupervised learning. Pada supervised learning, sistem diberikan data yang lengkap dengan label. Proses pembelajaran pada supervised learning adalah berusaha menemukan fungsi yang memetakkan data dengan label. Tujuan unsupervised learning adalah menemukan pola intrinsik dari derajat kesamaan (similarity) antardata. Sedangkan Reinforcement Learning (RL) memiliki pendekatan yang berbeda. Pada RL, agent berinteraksi dengan lingkungan yang menyediakan reward. Setiap aksi yang dilakukan oleh agent akan memberikan perubahan pada lingkungan dan akan mendapatkan reward berdasarkan aksi yang dilakukan. Tujuan dari RL adalah menemukan urutan aksi yang dapat memaksimalkan total reward yang didapatkan.



**Gambar 3.1 Inetraksi Agent dan Environment**

Pada RL, agent berinteraksi dengan lingkungan (*environment*). Setiap langkah, environment akan memberikan respon kepada agent dalam bentuk *state*. *Agent* melakukan aksi untuk merespon state tersebut. Kemudian dari aksi yang dilakukan oleh *agent*, *environment* akan merespon dengan *next state* berdasarkan aksi yang dilakukan, serta memberikan *reward* dari aksi yang dilakukan oleh *agent* tersebut.

Tujuan dari *agent* adalah untuk mempelajari *optimal policy* (*control startegy*) yang memaksimalkan total *reward* yang diharapkan (baik cumulative maupun *discounted reward*). Tantangan pada RL adalah *agent* dituntut untuk melakukan langkah trial-and-error untuj mencapai tujuan memaksimalkan total reward tersebut. Secara formal, RL dapat digambarkan sebagai *Markov Decision Process* (MDP) yang terdiri dari :

* Serangkaian state S,
* Serangkaian aksi A,
* Transisi T(| , ) yang memetakkan state berikutnya jika diberikan stste saat ini dan aksi yang dilakukan,
* Fungsi reward R(,,),
* Faktor diskon dengan nilai antara 0 sampai 1.

Secara umum, jika MDP bersifat episodik (state selalu di-reset setiap kali mencapai rangkaian state, aksi, state sepanjang T), maka rangkaian state, aksi, stste dalam satu episode merupakan sebuah lintasan dari sebuah policy. Setiap satu lintasan dari policy mengakumulasikan reward yang diperoleh dari environment dengan fungsi R.

*R* = (3-1)

Secara metematis, optimal policy dapat dituliskan sebagai berikut.

= arg max R| (3-2)

**3.1.2 Algoritma Pembelajaran RL**

Untuk menyelesaikan problem pada RL dapat menggunakan tiga pendekatan yaitu berbasis policy, pendekatan berbasis value function, dan pendekatan berbasis model (Sutton and Barto, 2017).

1. Pendekatan berbasis policy: fungsi yang memetakkan aksi dengan parameter state.
2. Pendekatan berbasis value function: fungsi yang menggambarkan seberapa baik setiap aksi atau state.
3. Pendekatan berbasis model: agent memiliki model yang merupakan representasi dari environment dari sudut pandang agent.

Contohnya pada kasus berikut ini. Diberikan sebuah labirin. Agent diminta untuk menemukan rute dari start ke goal.

Start

Goal

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

**Gambar 3.2 Contoh Problem RL**

Aksi yang dapat dilakukan seorang agent adalah bergerak ke Atas, Bawah, Kiri dan Kanan. Maka penyelesaian problem menggunakan RL untuk masing-masing pendekatantersebut adalah sebagai berikut.

|  |  |
| --- | --- |
| Pendekatan berbasis *policy* | Pendekatan berbasis *value function* |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |   Start | Goal  -22  -22  -21  -2  Goal   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  | | Start |  |  |  |  |  |  |  | |  | -14 | -13 | -12 | -11 | -10 | -9 |  | | -16 | -15 |  |  | -12 |  | -8 |  | |  | -16 | -17 |  |  | -6 | -7 |  | |  |  | -18 | -19 |  | -5 |  |  | |  | -24 |  | -20 |  | -4 | -3 |  | |
|  | -23  -1 |

-1

|  |
| --- |
| Pendekatan berbasis *model* |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  | -1 |  | | Start |  |  |  | -1 |  | -1 |  | |  | -1  -1 | -1 | -1 | -1 | -1 |  |  | | -1 | -1 |  |  |  |  |  |  | |  |  |  |  |  | -1 | -1 |  | |  |  |  |  |  | -1 |  | -1 | |  |  |  |  |  | -1 | -1 |  |   Goal |

**Gambar 3.3 Perbedaan Tiga Pendekatan RL**

Perhatikan perbedaan antara ketiga pendekatan tersebut. Pada pendekatan berbasis policy, RL agent diberikan serangkaian rekomendasi aksi yang dapat dipilih pada setiap state. Pada pendekatan berbasis value function, RL agent diberikan serangkaian nilai yang merupakan perkiraan total reward dari masing-masing state. Secara matematis value function dapat dituliskan sebagai berikut.

(s) = E [ R|s,**]** (3-3)

Dan optimal policy dapat diformulasikan sebagai berikut.

(s) =  (s), s (3-4)

Pada pendekatan berbasis model, agent memiliki representasi internal terhadap *environment*. Representasi tersebut bisa didapatkan dari beberapa kali percobaan. Walaupun model yang dibangun oleh agent tidak sempurna, namun dapat membantu agent dalam menemukan optimal *policy*. Selain *value function*, RL agent biasanya menggunakan fungsi yang menerima pasangan state dan aksi yang disebut sebagai *Q-function*. Q-function merupakan fungsi yang mengembalikan perkiraan total reward (expected return) pada sebuah state s dengan aksi a. (*s*,*a*) = [| = s, = a, ] (3-5)

Untuk memaksimalkan nilai dari (*s*,*a*), kita dapat mengasumsikan bahwa nilai sebuah Q-function didapatkan dengan menambahkan nilai reward pada suatu state dengan nilai Q-function pada stste berikutnya.Atau secara formal dapat kita tuliskan bahwa:

(, ) = [, + (, (3-6)

Perhatikan bahwa bentuk Q-function adalah bentuk rekursif . Artinya, kita dapat menggunakan nilai dari sebuah Q-function saat ini, untuk mengoptimalkan nilai Q-function berikutnya. Algoritma untuk mengoptimalkan nilai dari Q-function disebut algoritma Q-learning. Berikut adalah algoritma Q-learning (Sutton and Barto, 2017).

Algoritma Q-Learning

Inisialisasi Q(s,a)

**Repeat** untuk setiap episode

*Inisialisasi state s*

**Repeat** untuk setiap langkah dalam episode

Pilih a dari s menggunakan policy yang diturunkan dari Q (misalnya

Ambil aksi a, observasi R dan s’

Q(s,a) 🡨 Q(s,a) + max Q(s’, a’) – Q(s,a)]

S 🡨 s’

**Util** **s** adalah terminal state

Nilai adalah *learning* rate. Dalam implementasinya, fungsi Q(s,a)diimplementasikan dalam bentuk tabel untuk menyederhanakan kompleksitas dari algoritma *Q-learning*.

**3.1.2 Deep Q Networks**

Jika representasi dari state dan aksi yang dapat dilakukan adalah sangat banyak, maka algoritma *Q-learning* konvensional akan menjadi kompleks. Dalam kasus seperti ini, dibutuhkan pendekatan berbasis *Deep Learning* (DL) untuk meningkatkan kemampuan dari Q learning agar dapat memproses kasus dengan jumlah stste dan jumlah aksi yang sangat banyak. Penggabungan *Q-learning* dengan skema DL menghasilkan algoritma pembelajaran baru yang disebut *Deep Q Networks* (DQN) (Arulkumaran et al., 2017a). Mekanisme kerja dari DQN adalah dengan mengganti tabel Q(s,a) yang ada pada algoritma *Q-learning* konvensional dengan sebuah *neural network* yang mencoba untuk memperkirakan nilai dari Q. Neural Network ini berperan sebagai fungsi aproksimasi yang dinotasikan dengan Q(s,a;) dengan adalah bobot dari *neural network* tersebut. *Loss* function pada DQN dapat didefinisikan sebagai berikut.

= (3-7)

Dengan nilai disebut sebagai nilai target didefinisikan sebagai berikut.

a

= r + max Q(s’, a’; )

Dan proses update bobot pada DQN dapat didefinisikan menggunakan rumus berikut.

= + (- Q(s, a ; )) ∇ Q(s, a; ) (3-8)

Dengan adalah *learning rate*. Karena nilai target pada DQN harus menghitung nilai Q yang maksimal di masa depan, maka DQN harus menghasilkan beberapa prediksi nilai Q sekaligus untuk menghitung nilai target tersebut. Maka arsitektur dari DQN dapat digambarkan sebagai berikut.

**Q-Value 1**

**Network**

**S**

**State**

**Q-Value 2**

**Q-Value 3**

Input dari DQN adalah state dan outputnya adalah beberapa nilai Q berdasarkan aksi yang mungkin dilakukan pada stste tersebut.

**3.1.3 Policy Gradient**

Algoritma *Q-learning* dapat berjalan dengan baik dalam kondisi policy yang bersifat deterministik. Namun hal ini dapat berlaku dalam kondisi policy yang bersifat probabilistik. *Probabilistik policy* didefinisikan sebagai fungsi (*s*,*a*): S x A 🡪 [0,1] dengan (*s*,*a*) menggambarkan probailitas aksi *a* dilakukan jika diberikan state *s*.

Algoritma Policy Gradient (PG) menggunakan pendekatan berbasis policy untuk menyelesaikan problem RL dengan cara langsung mencari optimal policy tanpa menggunakan value seperti pada Q-learning. Sama seperti DQN, algoritma PG memberikan parameter bobot pada fungsi policy sebagai (s,a,) = P(a|s, ). Formulasi dari PG dapat bekerja dengan baik pada kondisi ruang aksi yang kontinu. Misalkan pada kendali sebuah kendaraan yang tidak hanya mengatur arah gerak, namun juga seberapa cepat kendaraan bergerak dan seberapa besar putaran dari kemudi kendaraan.

PG mendefinisikan fungsi J() sebagai fungsi objektif dari PG yang memaksimalkan total reward dari policy. Algoritma PG mencari nilai maksimum pada J() dengan pendekatan gradien descent terhadap nilai parameter . PG diimplementasikan dalam fungsi REINFORCE berikut ini (Ammar et al., 2014).

Algoritma Function REINFORCE

Inisialisasi parameter

Repeat untuk setiap episode

Inisialisasi state s

Repeat untuk setiap langkah dalam episode

🡨 log(|)

🡨 +

return

**3.1.4 Implementasi Deep Reinforcement Learning dengan Python**

**3.1.4.1 Import Library**

import gym

import torch

import torch.nn as nn

import torch.optim as optim

import torch.nn.functional as F

**3.1.4.2 Tentukan Arsitektur Jaringan Saraf Untuk Agen**

class DQN(nn.Module):

    def \_\_init\_\_(self, input\_dim, output\_dim):

        super(DQN, self).\_\_init\_\_()

        self.fc1 = nn.Linear(input\_dim, 64)

        self.fc2 = nn.Linear(64, 64)

        self.fc3 = nn.Linear(64, output\_dim)

    def forward(self, x):

        x = F.relu(self.fc1(x))

        x = F.relu(self.fc2(x))

        x = self.fc3(x)

        return x

**3.1.4.3 Inisialisasi Lingkungan**

env = gym.make('CartPole-v1')

input\_dim = env.observation\_space.shape[0]

output\_dim = env.action\_space.n

**3.1.4.4 Inisialisasi Jaringan Saraf Agen dan Optimizer**

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

agent = DQN(input\_dim, output\_dim).to(device)

optimizer = optim.Adam(agent.parameters(), lr=0.001)

**3.1.4.5 Definisikan Fungsi untuk Memilih Tindakan Berdsarkan Kebijakan Epsilon-Greedy**

def select\_action(state, epsilon):

    if torch.rand(1) < epsilon:

        return torch.tensor([env.action\_space.sample()], device=device, dtype=torch.long)

    else:

        with torch.no\_grad():

            q\_values = agent(state)

            return torch.argmax(q\_values).view(1, 1)

**3.1.4.6 Tentukan Fungsi Untuk melatih Agen Menggunakan Algoritma Q-Learning**

def train\_agent(num\_episodes, epsilon):

    for episode in range(num\_episodes):

        state = env.reset()

        done = False

        total\_reward = 0

        while not done:

            state = torch.tensor(state, device=device, dtype=torch.float32).unsqueeze(0)

            action = select\_action(state, epsilon)

            next\_state, reward, done, \_ = env.step(action.item())

            next\_state = torch.tensor(next\_state, device=device, dtype=torch.float32).unsqueeze(0)

            total\_reward += reward

            if done:

                next\_state = None

            agent.train()

            optimizer.zero\_grad()

            q\_values = agent(state).gather(1, action)

            if next\_state is None:

                target = torch.tensor([reward], device=device, dtype=torch.float32)

            else:

                agent.eval()

                with torch.no\_grad():

                    target = reward + torch.max(agent(next\_state)).item()

            loss = F.mse\_loss(q\_values, target.unsqueeze(1))

            loss.backward()

            optimizer.step()

            state = next\_state

        print("Episode:", episode+1, "Total Reward:", total\_reward)

**3.1.4.7 Panggil Fungsi Train\_Agent () Untuk memulai Pelatihan Agen**

num\_episodes = 100

epsilon = 0.1

train\_agent(num\_episodes, epsilon)

Dalam contoh di atas, agen menggunakan algoritma Q-Learning untuk belajar memainkan permainan Cart-Pole-v1. Dengan melakukan iterasi melalui serangkaian episode, agen secara interaktif berinteraksi dengan lingkungan, memilih tindakan berdasarkan kebijakan epsilon-greedy, dan menggunakan fungsi kerugian Mean Square Error (MSE) untuk memperbarui jaringan saraf.

**3.2 Self Organiszing MAP (SOM)**

**6. Implementasi CNN dengan Python**

**6.1 Import Tensor Flow**

****

**6.2 Import Sequential, Dense, Conv2D, MaxPooling2D, Dropout, Flatten, Callback, np\_utils, dan Files**

****

**6.3 Download dan Siapkan CIFAR10 Dataset**

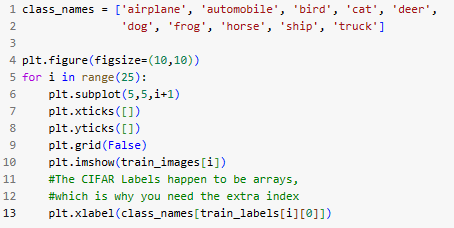
CHIFAR10 dataset berisi 60.000 gambar berwarna dalam 10 class, dengan 6.000 gambar dalam kelass lain. Dataset adalah terdefinisi ke dalam 50.000 gambar training dan 10.000 gambar testing. Kelas-kelas saling eksklusif dan tidak ada tumpang tindih di antara mereka

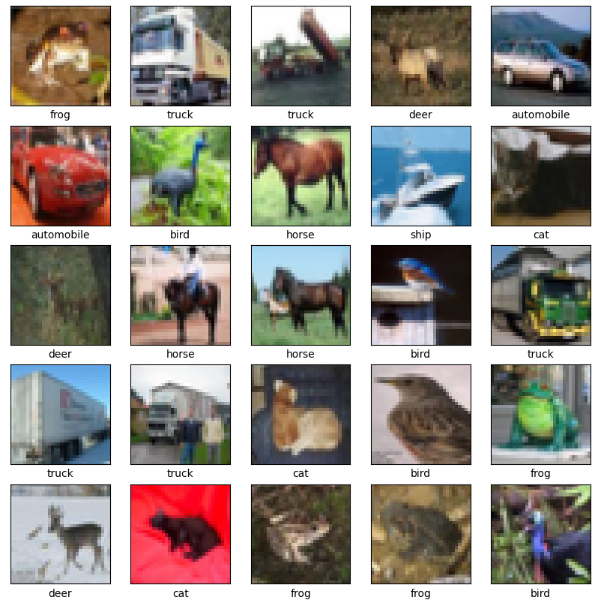
****

****

**6.4 Verifikasi Data**

Untuk memverifikasi bahwa dataset terlihat benar, mari plot 25 gambar pertama dari set pelatihan dan tampilkan nama kelas di bawah setiap gambar.

****



***Gambar 28. Gambar Datasets***

**6.5 Buat Basis Konvolusional**

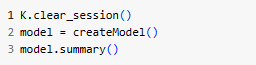
6 baris kode di bawah menentukan basis konvolusi menggunakan pola umum: setumpuk Conv2D (<https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D>) dan MaxPooling2D (<https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool2D>)

lapisan.

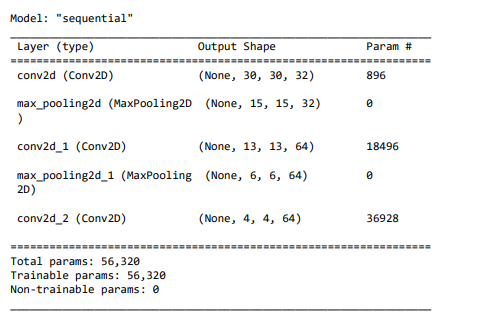
Sebagai input, CNN mengambil bentuk tensor (image\_height, image\_width, color\_channels).

Mengabaikan ukuran batch. Jika Anda baru mengenal dimensi ini, color\_channels mengacu pada (R, G, B). Dalam contoh ini, akan mengonfigurasi CNN Anda untuk memproses input bentuk (32, 32, 3), yang merupakan format gambar CIFAR. Anda dapat melakukan ini dengan meneruskan bentuk masukan argumen ke lapisan pertama Anda.

Mari tampilkan arsitektur atau model Anda sejauh ini



***Tabel 1. Model sequential***



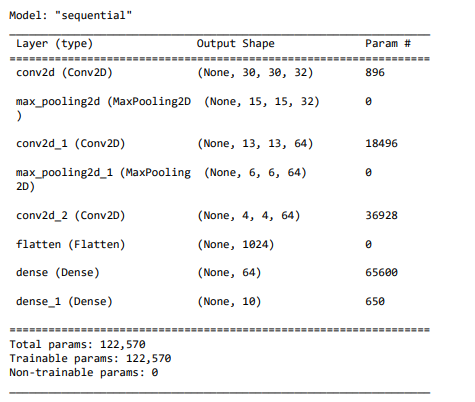
Di atas, Anda dapat melihat bahwa keluaran dari setiap lapisan Conv2D dan MaxPooling2D adalah bentuk tensor 3D (tinggi, lebar, saluran). Dimensi lebar dan tinggi cenderung menyusut saat Anda masuk lebih dalam ke jaringan. Jumlah saluran keluaran untuk setiap lapisan Conv2D dikontrol oleh argumen pertama (mis., 32 atau 64). Biasanya, saat lebar dan tinggi menyusut, Anda mampu (secara komputasi) menambahkan saluran keluaran di setiap lapisan Conv2D.

**6.7 Tambahkan Lapisan Padat di Atas**

Untuk menyelesaikan model, Anda akan mengumpankan tensor keluaran terakhir dari basis konvolusional (bentuk (8, 8, 64)) ke dalam satu lapisan Dense lagi untuk melakukan klasifikasi. Lapisan padat mengambil vektor sebagai masukan (yaitu 1D), sedangkan keluaran saat ini adalah tensor 3D. Pertama, Anda akan meratakan (atau membuka gulungan) keluaran 3D menjadi 1D, lalu menambahkan satu lapisan Padat lagi di atasnya. CIFAR memiliki 10 kelas keluaran, jadi Anda menggunakan layer Dense akhir dengan 10 keluaran.

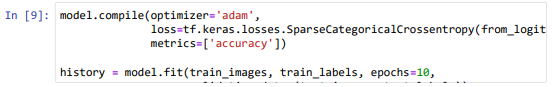


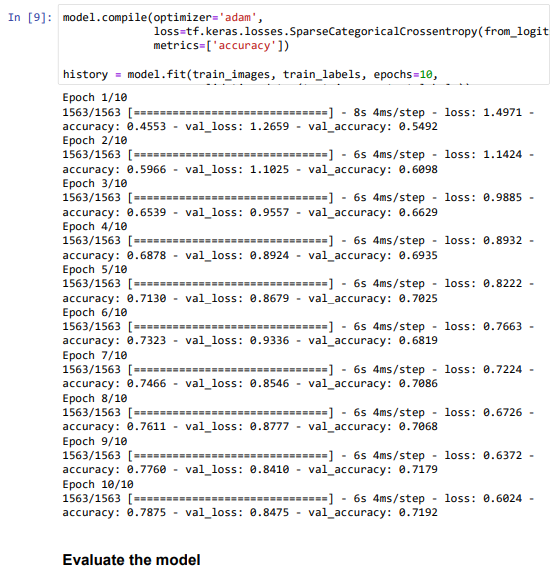
***Tabel 2. Model sequential***



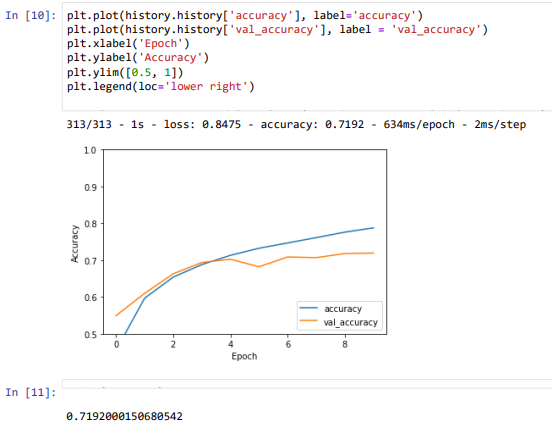
Ringkasan jaringan menunjukkan bahwa (4, 4, 64) keluaran diratakan menjadi vektor bentuk (1024) sebelum melalui dua lapisan Padat.

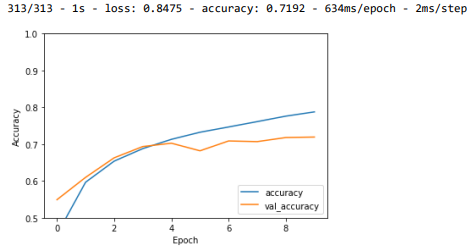
**6.8 Kompilasi dan Latih Model**

****



**6.9 Evaluasi Model**





**6.10 Akurasi**

0.7192000150680542

**C. Deep Unsupervised Learning?**

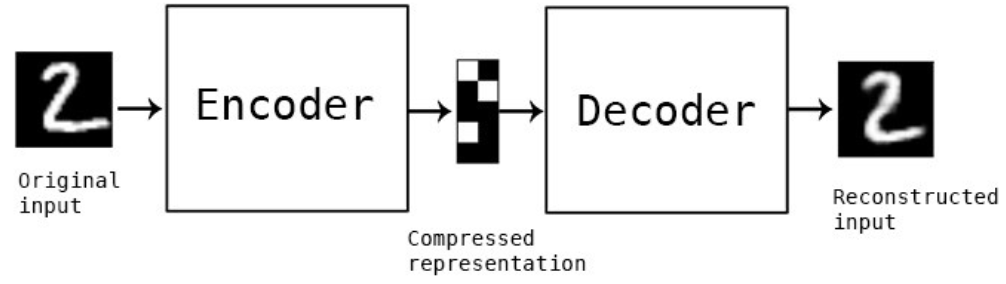
Pendekatan *Deep Unsupervised Learning* ini memodelkan sekumpulan input secara otomatis tanpa panduan. Artinya, data-data yang dipelajari hanya berupa input tanpa label kelas. DL jenis ini digunakan untuk masalah klasterisasi. Jika diberikan sekumpulan data masukan, DL mampu secara otomatis membagi data tersebut ke dalam sejumlah klaster berdasarkan, misalnya, tingkat kemiripan dalam suatu kelas. Pada bab ini, kita akan membahas sejumlah model, yaitu: *Autoencoders* (AE), *Restricted Boltzmann Machines* (RBM).

Jika Anda mempunyai data berdimensi sangat tinggi (jumlah fitur sangat banyak), mungkin saja setiap fitur tersebut pada masing-masing dimensi sehingga setiap data terlihat sangat berbeda. Untuk mengatasi masalah ini, Anda bisa melakukan rekayasa fitur dengan dua cara: seleksi fitur atau reduksi fitur. Rekayasa fitur digunakan untuk membuat sebaran data menjadi lebih sederhana. Data yang tadinya tersebar acak dan tumpang tindih dapat dikelompokkan secara teratur dengan sedikit (bahan tidak ada) tumpang tindih lagi, sehingga lebih mudah dianalisis. Selain itu, rekayasa fitur juga digunakan untuk memvisualisasikan data sehingga sebaran lebih mudah dipahami.

**1. Autoencoder**

*Autoencoder* adalah model neural network yang memiliki input dan output yang sama. *Autoencoder* mempelajari data input dan berusaha untuk melakukan rekonstruksi terhadap data input tersebut.

Lalu apa gunanya *Autoencoder*? Autoencoder biasa digunakan untuk mengurangi dimensi dari features (*Dimensionality Reduction*). Jika kita mempunyai data yang mempunyai dimensi yang sangat tinggi (data dengan jumlah features yang sangat banyak) bisa jadi tiap features yang ada tersebar pada setiap dimensi dari data sehingga setiap data yang ada terlihat sangat berbeda. Untuk mengatasi masalah tersebut kita bisa menggunakan PCA, t-SNE ataupun *Autoencoder*.



***Gambar 29. Autoencoder***

*Autoencoder* terdiri dari dua bagian utama yaitu *encoder* dan *decoder*. Diantara *encoder* dan *decoder* terdapat code layer atau juga bisa dibilang target layer (istilah saya sendiri). Jumlah neuron pada code layer adalah jumlah dimensi yang kita harapkan untuk mengurangi dimensi data kita.

Untuk contoh implementasi, kita akan melakukan dimensionality reduction terhadap MNIST data. MNIST adalah dataset yang terdiri dari angka 0 sampai 9 yang ditulis oleh tangan.

**2. Dependency**

Dependency yang dibutuhkan pada contoh autoencoder kali ini hampir sama dengan contoh pada part-part sebelumnya. Hanya saja kali ini kita akan load MNIST data dari package yang sudah disediakan oleh keras. Kita juga mau coba optimizer baru yaitu ADAM. ADAM adalah varian dari algoritma gradient descent yang lain, untuk lebih jelasnya mungkin bisa di baca sendiri di paper <https://arxiv.org/abs/1412.6980>

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from keras.models import Model

from keras.layers import Input, Activation, Dense

from keras.optimizers import Adam

from keras.utils.np\_utils import to\_categorical

from keras.datasets import mnist

**2.1 Data Preparation**

Dari data MNIST ini adalah *grayscale* image dengan range daro 0 hingga 255. Range data seperti ini “terlalu besar” untuk model kita, apalagi dengan learning rate yang cukup kecil, sehingga kita perlu melakukan scaling dengan membaginya dengan 255. Sehingga kita dapatkan range data baru antara 0 dan 1. Setelah itu merubahnya menjadi 784 vector yang dijadikan sebagai input.

# Download and Save MNIST Dataset

(train\_x, train\_y), (test\_x, test\_y) = mnist.load\_data()

# Scale from 0 to 1

train\_x = train\_x.astype('float32') / 255.

test\_x = test\_x.astype('float32') / 255.

# Reshape from 28x28 matrix to 784 vector

train\_x = np.reshape(train\_x, (len(train\_x), np.prod(train\_x.shape[1:])))

test\_x = np.reshape(test\_x, (len(test\_x), np.prod(test\_x.shape[1:])))

**2.2 Create the Autoencoder**

Seperti yang telah dijelaskan di atas, autoencoder terdiri dari *encoder* dan *decoder*. Jumlah neuron pada tiap layer dari encoder adalah 784-256-128-64-32-16 dan untuk *decoder* adalah 16-32-64-128-256-784. Optimizer yang digunakan adalah ADAM dengan learning rate sebesar 0.001 dan loss function yang akan digunakan adalah binary\_crossentropy. Kenapa *cross entropy*? Data yang kita miliki mempunyai rentang dari 0 sampai 1, *activation function* pada output layer juga sigmoid yang mempunyai rentang dari 0 sampai 1. Itu semua identik dengan probability distribution, sehingga kita bisa anggap ini adalah permasalahan klasifikasi dan kita bisa gunakan *binary cross entropy*.

Sebenarnya kita juga bisa anggap ini adalah sebuah regresi dengan menggunakan linear activation pada output layer, tapi loss function yang kita gunakan harus MSE (*Mean Square Error*). Nantinya hasilnya akan dicompare di akhir part ini. Perlu diingat juga kalau autoencoder berusaha melakukan rekonstruksi terhadap inputdata, jadi target yang kita gunakan adalah input itu sendiri (train\_x dan test\_x)

# Target Dimension

TARGET\_DIM = 16

# Encoder

inputs = Input(shape=(784,))

h\_encode = Dense(256, activation='relu')(inputs)

h\_encode = Dense(128, activation='relu')(h\_encode)

h\_encode = Dense(64, activation='relu')(h\_encode)

h\_encode = Dense(32, activation='relu')(h\_encode)

# Coded

encoded = Dense(TARGET\_DIM, activation='relu')(h\_encode)

# Decoder

h\_decode = Dense(32, activation='relu')(encoded)

h\_decode = Dense(64, activation='relu')(h\_decode)

h\_decode = Dense(128, activation='relu')(h\_decode)

h\_decode = Dense(256, activation='relu')(h\_decode)

outputs = Dense(784, activation='sigmoid')(h\_decode)

# Autoencoder Model

autoencoder = Model(inputs=inputs, outputs=outputs)

# Encoder Model

encoder = Model(inputs=inputs, outputs=encoded)

# Optimizer / Update Rule

adam = Adam(lr=0.001)

# Compile the model Binary Crossentropy

autoencoder.compile(optimizer=adam, loss='binary\_crossentropy')

# Train and Save weight

autoencoder.fit(train\_x, train\_x, batch\_size=256, epochs=100, verbose=1, shuffle=True, validation\_data=(test\_x, test\_x))

autoencoder.save\_weights('weights.h5')

**2.3 Linear & Mean Square Error**

TARGET\_DIM = 16

# Encoder

inputs = Input(shape=(784,))

h\_encode = Dense(256, activation='relu')(inputs)

h\_encode = Dense(128, activation='relu')(h\_encode)

h\_encode = Dense(64, activation='relu')(h\_encode)

h\_encode = Dense(32, activation='relu')(h\_encode)

# Coded

encoded = Dense(TARGET\_DIM, activation='relu')(h\_encode)

# Decoder

h\_decode = Dense(32, activation='relu')(encoded)

h\_decode = Dense(64, activation='relu')(h\_decode)

h\_decode = Dense(128, activation='relu')(h\_decode)

h\_decode = Dense(256, activation='relu')(h\_decode)

outputs = Dense(784, activation='linear')(h\_decode)

# Autoencoder Model

autoencoder = Model(inputs=inputs, outputs=outputs)

# Encoder Model

encoder = Model(inputs=inputs, outputs=encoded)

# Optimizer / Update Rule

adam = Adam(lr=0.001)

# Compile the model Binary Crossentropy

autoencoder.compile(optimizer=adam, loss='mse')

autoencoder.fit(train\_x, train\_x, batch\_size=256, epochs=100, verbose=1, shuffle=True, validation\_data=(test\_x, test\_x))

autoencoder.save\_weights('weightsLinear.h5')

**2.4 Reconstruction Result**

Setelah training selesai, kita akan mencoba untuk melakukan rekonstruksi untuk digit 0 hingga 9 pada test data. Kita juga bisa mengambil output dari encoder yang merupakan features yang dimensinya telah direduksi. Feature ini nantinya bisa digunakan untuk klasifikasi untuk melihat bagaimana kualitasnya.

# Encoded Data

encoded\_train = encoder.predict(train\_x)

encoded\_test = encoder.predict(test\_x)

# Reconstructed Data

reconstructed = autoencoder.predict(test\_x)

n = 10

plt.figure(figsize=(20, 4))

for i in range(n):

count = 0

while True:

if i == test\_y[count]:

# Original

ax = plt.subplot(2, n, i + 1)

plt.imshow(test\_x[count].reshape(28, 28))

plt.gray()

ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

# Reconstructed

ax = plt.subplot(2, n, i + 1 + n)

plt.imshow(reconstructed[count].reshape(28, 28))

plt.gray()

ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

break;

count += 1

plt.show()

Untuk linear MSE, setelah 100 epoch kita mendapatkan loss sebesar 0.0140/0.0145 MSE loss.



***Gambar 30. Linear-MSE Reconstruction***

Sedangkan sigmoid cross entropy, setelah 100 epoch, kita mendapatkan loss/validation loss sebesar 0.0923/0.0950 cross-entropy loss.



***Gambar 31. Kualitas rekonstruksi dari sigmoid***

Namun jika hasil prediksi diubah menjadi image, kualitas rekonstruksi dari sigmoid cross entropy mempunyai kualitas yang lebih bagus.

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from keras.models import Model

from keras.layers import Input, Activation, Dense

from keras.optimizers import Adam

from keras.utils.np\_utils import to\_categorical

from keras.datasets import mnist

# Download and Save MNIST Dataset

(train\_x, train\_y), (test\_x, test\_y) = mnist.load\_data()

# Scale from 0 to 1

train\_x = train\_x.astype('float32') / 255.

test\_x = test\_x.astype('float32') / 255.

# Reshape from 28x28 matrix to 784 vector

train\_x = np.reshape(train\_x, (len(train\_x), np.prod(train\_x.shape[1:])))

test\_x = np.reshape(test\_x, (len(test\_x), np.prod(test\_x.shape[1:])))

# Target Dimension

TARGET\_DIM = 16

# Encoder

inputs = Input(shape=(784,))

h\_encode = Dense(256, activation='relu')(inputs)

h\_encode = Dense(128, activation='relu')(h\_encode)

h\_encode = Dense(64, activation='relu')(h\_encode)

h\_encode = Dense(32, activation='relu')(h\_encode)

# Coded

encoded = Dense(TARGET\_DIM, activation='relu')(h\_encode)

# Decoder

h\_decode = Dense(32, activation='relu')(encoded)

h\_decode = Dense(64, activation='relu')(h\_decode)

h\_decode = Dense(128, activation='relu')(h\_decode)

h\_decode = Dense(256, activation='relu')(h\_decode)

outputs = Dense(784, activation='sigmoid')(h\_decode)

# Autoencoder Model

autoencoder = Model(inputs=inputs, outputs=outputs)

# Encoder Model

encoder = Model(inputs=inputs, outputs=encoded)

# Optimizer / Update Rule

adam = Adam(lr=0.001)

# Compile the model Binary Crossentropy

autoencoder.compile(optimizer=adam, loss='binary\_crossentropy')

# Train and Save weight

autoencoder.fit(train\_x, train\_x, batch\_size=256, epochs=100, verbose=1, shuffle=True, validation\_data=(test\_x, test\_x))

autoencoder.save\_weights('weights.h5')

# Encoded Data

encoded\_train = encoder.predict(train\_x)

encoded\_test = encoder.predict(test\_x)

# Reconstructed Data

reconstructed = autoencoder.predict(test\_x)

n = 10

plt.figure(figsize=(20, 4))

for i in range(n):

count = 0

while True:

if i == test\_y[count]:

# Original

ax = plt.subplot(2, n, i + 1)

plt.imshow(test\_x[count].reshape(28, 28))

plt.gray()

ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

# Reconstructed

ax = plt.subplot(2, n, i + 1 + n)

plt.imshow(reconstructed[count].reshape(28, 28))

plt.gray()

ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

break;

count += 1

plt.show()

**3. Restricted Boltzmann Machine**

*Restricted Boltzmann Machine* (RBM) adalah model graph tidak berarah yang memainkan peran penting dalam bidang DL belakangan ini. Diciptakan oleh Geoffrey Hilton pada tahun 1985, *Restricted Boltzmann Machine* termasuk dalam kategori algoritma *unsupervised learning algorithms* adalah jaringan unit mirip neuron yang terhubung secara simetris yang membuat keputusan stokastik (menggambarkan hasil yang didasarkan pada hubungan pasti) Algoritma deep learning ini menjadi sangat populer setelah Kompetisi Netflix di mana RBM digunakan sebagai teknik penyaringan kolaboratif untuk memprediksi peringkat pengguna untuk film dan mengalahkan sebagian besar pesaingnya. Ini berguna untuk regresi, klasifikasi, pengurangan dimensi, pembelajaran fitur, pemodelan topik, dan penyaringan kolaboratif. Restricted Boltzmann Machine adalah jaringan saraf dua lapis stokastik yang termasuk dalam kategori model berbasis energi yang dapat mendeteksi pola bawaan secara otomatis dalam data dengan merekonstruksi masukan. Mereka memiliki dua lapisan yang terlihat dan tersembunyi. Lapisan terlihat memiliki node input (node yang menerima data input) dan lapisan tersembunyi dibentuk oleh node yang mengekstrak informasi fitur dari data dan output pada lapisan tersembunyi adalah jumlah tertimbang dari lapisan input. Ini mungkin tampak aneh tetapi mereka tidak memiliki node keluaran dan mereka tidak memiliki keluaran biner yang khas di mana pola dipelajari. Proses belajar terjadi tanpa adanya kemampuan yang membuat mereka berbeda. Kami hanya menangani node input dan tidak khawatir tentang node tersembunyi. Setelah input diberikan, RBM secara otomatis menangkap semua pola, parameter, dan korelasi antara data. Itulah keindahan mesin *Boltzmann*. Ada banyak kasus penggunaan bisnis waktu nyata dimana RBM digunakan seperti:

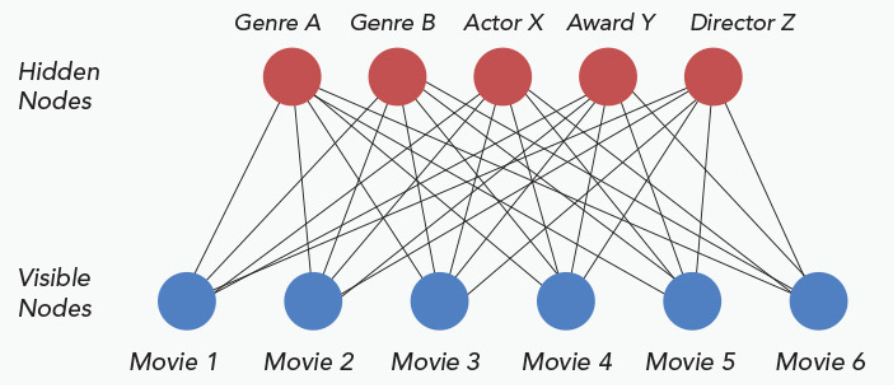
* *Pattern Recognition* (Pengenalan pola): RBM digunakan untuk ekstraksi fitur dalam masalah pengenalan pola di mana tantangannya adalah memahami teks tulisan tangan atau pola acak.
* *Recommendation Engine* (Mesin Rekomendasi): RBM banyak digunakan untuk kolaborasi teknik penyaringan di mana digunakan untuk memprediksi apa yang harus direkomendasikan kepada pengguna akhir sehingga pengguna senang menggunakan aplikasi atau platform tertentu. Contoh: Rekomendasi film, Rekomendasi buku.
* *Radar Target Recognition* (Pengenalan target Radar): di sini RBM digunakan untuk mendeteksi pulsa intra dalam sistem radar yang memiliki SNR sangat rendah dan noise tinggi.

**3.1 Fitur Restricted Boltzmann Machine**

Beberapa fitur penting dari Mesin Boltzmann yaitu:

* Mereka menggunakan struktur berulang dan simetris.
* RBM dalam proses pembelajaran mereka mencoba untuk mengasosiasikan probabilitas tinggi dengan keadaan energi rendah dan sebaliknya.
* Tidak ada koneksi intra layer.
* Ini adalah *unsupervised learning algorithm*, yaitu membuat kesimpulan dari data input tanpa respon berlabel.

Bagaimana memahami *Restricted Boltzmann Machine* berbeda dari Boltzmann Machine. Kedua algoritma memiliki dua lapisan yang terlihat dan tersembunyi. Dalam *Boltzmann Machine* setiap neuron di lapisan yang terlihat terhubung ke setiap neuron di lapisan tersembunyi serta semua neuron terhubung di dalam lapisan. Namun RBM adalah kasus khusus dari *Boltzmann Machine* dengan batasan waktu neuron di dalam lapisan tidak terhubung yaitu, tidak ada komunikasi intra layer yang membuatnya independen dan lebih mudah diimplementasikan karena independensi bersyarat berarti kita hanya perlu menghitung margin probabilitas yang lebih mudah untuk dihitung. Diagram di bawah ini akan membantu kita memahami hal yang sama:



***Gambar. 32. Pengenalan faktor-faktor laten dalam data movie***

**3.2 Training RBM**

RBM dilatih dengan menggunakan *Gibbs* Sampling dan *Contrastive Divergence*. Dalam statistik, pengambilan sampel *Gibbs* adalah algoritma rantai Markov Carlo untuk memperoleh urutan pengamatan yang kira-kira dari distribusi probabilitas multivariat tertentu, ketika pengambilan sampel langsung sulit dilakukan. Jika input diwakili oleh *v* dan nilai tersembunyi oleh h maka, *P* () adalah prediksi. Mengetahui nilai tersembunyi, *P* () digunakan untuk prediksi nilai input yang diregenerasi. Katakanlah proses ini diulang *k* kali dan setelah k iterasi *v\_k* diperoleh dari nilai input awal *v\_0*. Divergensi kontrastif adalah perkiraan algoritma pembelajaran kemungkinan. Maksimum untuk memperkirakan kemiringan grafis yang mewakili hubungan antara bobot jaringan dan kesalahannya, yang disebut gradien. Ini digunakan dalam situasi di mana kita tidak dapat mengevaluasi fungsi atau serangkaian probabilitas secara langsung, beberapa bentuk model inferensi diperlukan untuk memperkirakan gradien pembelajaran algoritma dan memutuskan arah mana yang akan dituju.

Dalam CD, bobot sedang diperbarui. Gradien pertama dihitung dari input yang direkonstruksi dan kemudian delta ditambahkan ke bobot lama untuk mendapatkan bobot baru.

**3.3 Kelebihan dan Kekurangan RBM**

Kelebihan:

* Cukup ekspresif untuk mengkodekan distribusi apapun dan efisien secara kumputasi.
* Lebih cepat dari Mesin Boltzmann tradisional karena pembatasan dalam hal koneksi antar node.
* Aktivasi lapisan tersembunyi dapat digunakan sebagai input ke model lain sebagai fitur yang berguna untuk meningkatkan kinerja.

Kekurangan:

* Pelatihan lebih sulit karena sulit untuk menghitung fungsi gradien energi.
* Algoritma CD-k yang digunakan dalam RBM tidak begitu familiar dengan algoritma *backpropagation*.
* Penyesuaian berat.

**3.4 Aplikasi RBM**

Pengenalan digit tulisan tangan adalah masalah yang sangat umum akhir-akhir ini dan digunakan dalam berbagai aplikasi seperti bukti kriminal, komputerisasi kantor, verifikasi cek, dan aplikasi entri data. Itu juga dilengkapi dengan tantangan seperti gaya penulisan yang berbeda,, variasi bentuk dan ukuran serta noise gambar, yang mengarah pada perubahan topologi numerik. Dalam hal ini metodologi RBM-CNN hybrid digunakan untuk pengenalan digit. Pertama, fitur diekstraksi menggunakan algoritma deep learning RBM. Kemudian fitur yang diekstraksi diumpankan ke algoritma deep learning CNN untuk klasifikasi. RBM sangat mampu untuk mengekstraksi fitur dari data input. Ini dirancang sedemikian rupa sehingga dapat mengekstrak fitur diskriminatif dari kumpulan data yang besar dan kompleks dengan memperkenalkan unit tersembunyi dengan cara yang tidak diawasi.

**BAB IV**

**PENUTUP**

Kita telah mendiskusikan banyak model dan arsitektur *Deep Learning*. Sebagai disiplin ilmu baru, *Deep Learning* mengalami perkembangan sangat cepat dengan lompatan-lompatan yang luar biasa. Dalam beberapa tahun terakhir, DL telah meletakkan sejumlah pijakan teori yang lebih kuat, lebih stabil, dan lebih mapan. Banyak model baru bermunculan, yang merupakan modernisasi model-model lama, bukan kuno, yang telah ada sejak puluhan tahun lalu. Berbagai model modern terbaru telah berhasil menyelesaikan banyak masalah dunia nyata dan sebagian mulai diterapkan dalam aplikasi riil. Bagaimanapun, DL tetap memiliki sejumlah kekurangan dan kelemahan. Sebagaimana semua bidang ilmu sains dan rekayasa, Deep learning juga memerlukan kritik yang tajam untuk terus dilakukan perbaikan. Sebagai penutup, mari kita bersama-sama merangkum seluruh diskusi kita.

**A. Kesimpulan**

Dari penelitian dan eksperimen yang telah dilakukan di atas dapat disimpulkan bahwa, Deep Learning telah menjadi salah satu bidang yang paling banyak diteliti dan diaplikasikan. Hampir setiap bulan muncul konsep dan arsitektur baru yang lebih matang. Di tahun 2019 ini, *Deep Learning* semakin banyak diaplikasikan di berbagai bidang. Hal ini karena DL memiliki banyak kelebihan, diantaranya adalah universal, tahan terhadap variasi data, memiliki tingkat generalisasi dan kapabilitas yang tinggi.

***Universal***. Deep Learning dapat diimplementasikan pada hampir setiap ranah aplikasi. Hal ini membuat DL disebut memiliki kemampuan universal learning.

***Robust***. Deep Learning bersifat tahan terhadap berbagai variasi data yang secara alami memang sesuai fakta yang ada. DL tidak memerlukan fitur yang didesain secara artifisial. Bahkan, fitur optimal dapat dipelajari secara otomatis.

***Generalization***. Deep Learning (DL) bisa digunakan pada berbagai aplikasi dan tipe data karena DL memiliki kemampuan transfer learning. Hal ini sangat berguna ketika tidak tersedia data yang cukup untuk suatu masalah yang akan diselesaikan.

***Scalability***. *Deep Learning* skalabilitas yang tinggi. Sejumlah jaringan *RestNet* yang dibuat oleh Microsoft terdiri atas 1.202 lapisan dan diimplementasikan pada sebuah supercomputer (K He et al., 2016). Namun, *RestNet* dapat diimplementasikan pada sistem paralel dengan ribuan komputer, seperti yang dilakukan oleh *Lawrence Livermore National Laboratory* (Van Essen et al., 2015).

**B. Saran**

Dalam mempelajari Deep learning yang meliputi *Artificial Neural Network* (ANN), *Convolution Neural Network* (CNN), dan *Deep Unsupervised Learning* (DUL) kita harus dapat memahami secara mendalam dan mengkaji sejauh mana capaian yang telah didapat serta apa saja permasalahan yang dihadapi dalam mempelajari deep learning (DL). Kita juga harus dapat memprediksi bagaimana masa depan Deep Learning serta tantangan yang dihadapinya di masa yang akan datang bagi kehidupan manusia.

**DAFTAR PUSTAKA**

1. Deep Learning . (2023). Dalam M. Catur Nugroho S.Kom., *Sesi-1 Pengenalan* (hal. 1-19). Jakarta: UNSIA.

2. Deep Learning. (2023). Dalam M. Catur Nugroho S.Kom., *Pertemuan 2 Jaringan Saraf Tiruan* (hal. 1-33). Jakarta: UNSIA.

3. Deep Learning. (2023). Dalam S. M. Catur Nugroho, *Sesi 3 Convolutional Neural Network* (hal. 1-12). Jakarta: UNSIA.

4. Deep Learning. (2023). Dalam S. M. Catur Nugroho, *Sesi 4 Deep Unsupervised Learning* (hal. 1-11). Jakarta: UNSIA.

5. Dr. Suyanto, S. M. (2019). *Deep Learning Modernisasi Machine Learning untuk Big Data.* Bandung: INFORMATIKA.

6. Mulyawan, R. (16 Mei 2023). *Pengertian ANN (Artificial Neural Network): Cara Kerja, Fungsi dan Jenisnya*. Diambil kembali dari RifkiMulyawan.com: <https://rifqimulyawan.com/blog/pengertian-ann/>. Diakses 16 Mei 2023.

7. Adi, A. S. (2021, Maret 5). *Konsep Dasar Convolutional Neural Network (CNN)*. Diambil kembali dari Warung Sains Teknologi: <https://warstek.com/convolutional-neural-network/>. Diakses 16 mei 2023.

****

|  |  |  |
| --- | --- | --- |
| **Nilai** | **Tanda Tangan Dosen Pengampu** | **Tanda Tangan**  **Mahasiswa** |
|  | **(Catur Nugroho, S.Kom,. M.Kom)** | **(Hendro Gunawan)** |
| Diseahkan pada Tanggal : | | Tanggal Mengumpulkan : |
|  | | 21/05/2023 |