**BAB II**

**LANDASAN TEORI**

1. **Citra Digital**

Citra atau gambar adapat didefinisikan sebagai sebuah fungsi dua dimensi, , dimana dan adalah koordinat bidang datar, dan harga fungsi di setiap pasangan koordinat disebut intensitas atau level keabuan (*grey level*) dari gambar di titik itu (Hermawati, 2013).

Jika dan semuanya berhingga (*finite*), dan nilainya diskrit, maka gambarnya disebut citra digital (gambar digital).

Sebuah citra digital terdiri dari sejumlah elemen yang berhingga, di mana masing-masing mepunyai lokasi dan nilai tertentu. Elemen-elemen ini disebut sebagai picture element, image element, pels atau pixels.

1. **Pengolahan Citra Digital**

Pengolahan citra digital merupakan suatu disiplin ilmu yang mempelajari hal-hal yang berkaitan dengan perbaikan kualitas gambar (peningkatan kontras, transformasi warna, restorasi citra), transformasi gambar (rotasi, translasi, transformasi geometrik, skala), agar mudah diinterpretasi oleh manusia/mesin (komputer). Masukannya adalah citra dan keluarannya juga citra tapi dengan kualitas lebih baik daripada citra masukan missal citra warna kurang tajam, kabur (*blurr*), dan mengandung *noise* (missal bintik-bintik putih) sehingga perlu ada pemrosesan untuk memperbaiki citra karena citra tersebut menjadi sulit diinterpretasikan karena informasi yang disampaikan menjadi berkurang (Universitas Sumatra, 2014, p. 16).

1. **Machine Learning**

*Machine Learning* adalah salah satu terapan bedang Kecerdasan Buatan/*Artificial Intelligence* (AI) yang mampu secara otomatis mempelajari dan berkembang dari pengalaman tanpa harus diprogram secara khusus. *Machine Learning* berfokus pada pengembangan program komputer melalui akses terhadap data lalu menggunakan data tersebut untuk mengajari dirinya

sendiri (Expert System, 2017). Beberapa macam pembelajaran pada *machine learning* adalah sebagai berikut:

* 1. **Supervised Learning**

*Supervised Learning* adalah metode pembelajaran *machine learning* yang dapat menerapkan hasil pembelajaran pada data baru. Hasil pembelajaran pada *supervised learning* diperoleh dari hasil latih pada data yang sudah diberi label sebelumnya, sehingga bisa memprediksi kesimpulan dari data baru yang ada pada masa mendatang.

Metode pembelajaran *supervised learning* dimulai dari menganalisa data yang telah diberi label (dataset), lalu algoritma pembelajaran yang digunakan akan menghasilkan pengetahuan yang bisa digunakan untuk membuat prediksi mengenai label data pada dataset yang baru setelah algoritma dilatih dengan cukup. Hasil dari algoritma pembelajaran ini juga dapat dibandingkan dengan label sebenarnya lalu kesalahan prediksi dari label sebenarnya bisa digunakan untuk mengubah model sesuai kesalahan yang dipelajari.

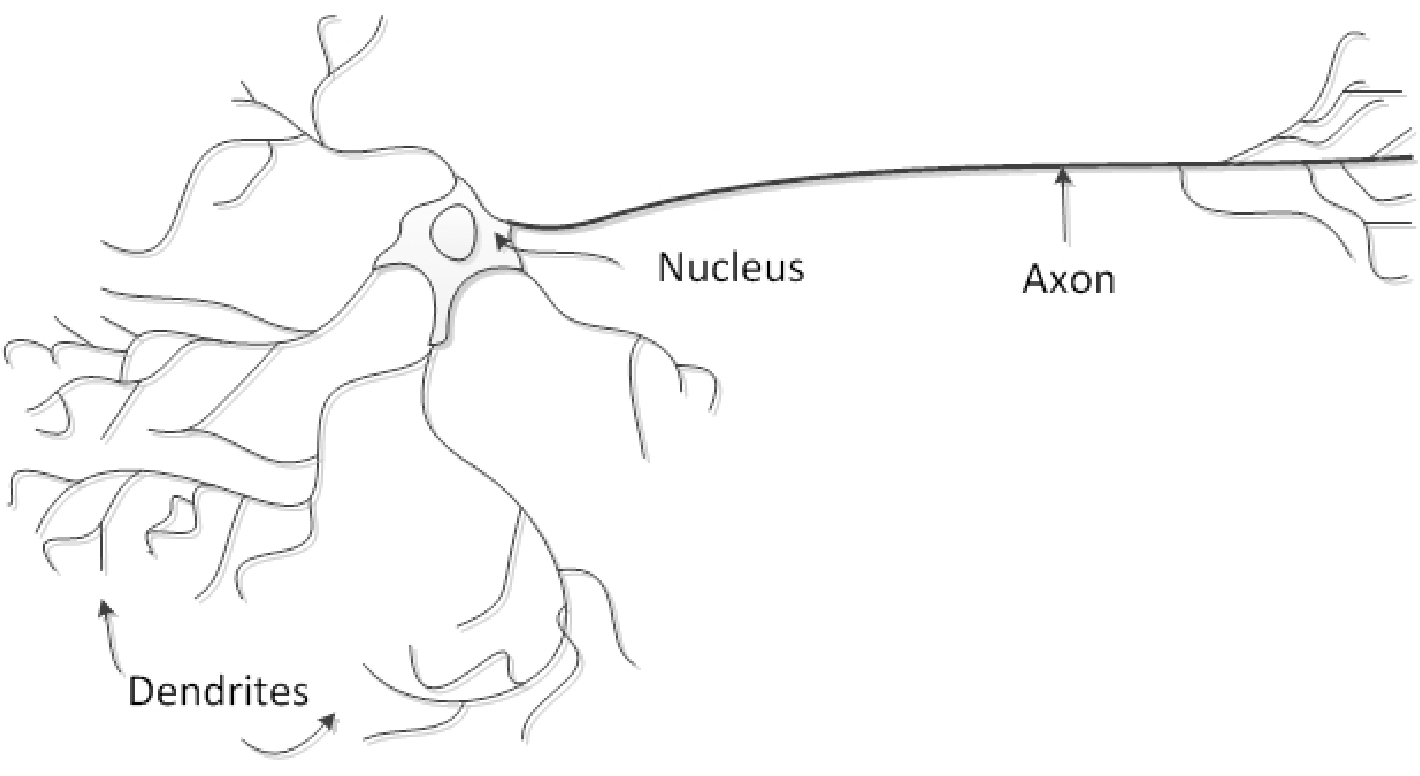
* 1. **Unsupervised Learning**

Algoritma *unsupervised learning* digunakan ketika data yang digunakan untuk mengajari algoritma pembelajaran tidak diberi label dan tidak juga telah diklasifikasi. Algoritma *unsupervised learning* mempelajari bagaimana mendapatkan pengetahuan dari data yang tidak memiliki label dan tidak terstruktur. Algoritma *unsupervised learning* tidak memberi output secara langsung melainkan memberikan gambaran dan struktur dari sebuah data yang tidak terstruktur.

1. **Artificial Neural Network (ANN)**

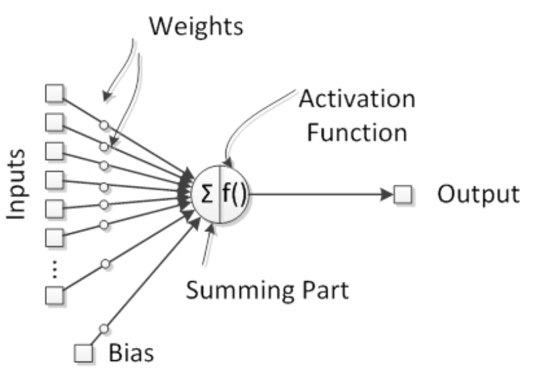
*Artificial Neural Network* atau Jaringan Syaraf Tiruan adalah sebuah sistem komputasi yang dirancang dengan menirukan cara otak manusia menganalisa dan memproses informasi. Sistem ANN memiliki kemampuan untuk belajar secara mandiri dimana hasil pembelajaran dipengaruhi oleh jumlah data yang digunakan dalam belajar (Jake Frankenfield, 2018).

Adapun cara ANN bekerja adalah dengan menirukan cara otak manusia bekerja, dimana ANN terbentuk dari satuan unit komputasi yang kecil-kecil yang disebut *node* yang saling terhubung membentuk sebuah jaringan. Otak manusia memiliki ratusan miliar *node* yang disebut dengan *neuron* dimana setiap *neuron* terdiri dari *dendrites* sebagai *input, nucleus* sebagai pemroses dan *axon* sebagai penyalur informasi yang telah diproses, dimana *axon* akan tersambung lagi ke *dendrites* dari *neuron* yang lain (Soares & Souza, 2016).



Gambar 2.1 Struktur *Neuron* Pada Otak Manusia

Menirukan sistem *neuron* maka pada ANN terdapat unit komputasi *neuron* terkecil yang disebut dengan *perceptron* yang memiliki cara kerja seperti *neuron* pada otak manusia.



Gambar 2.2 Struktur *Perceptron* Pada ANN

Gambar 2.2 diatas dapat diekspresikan dengan persamaan matematis seperti dibawah:

Dimana adalah *output,* adalah fungsi aktivasi, adalah *weight,* adalah *input* dan adalah bias, setiap komponen diatas memiliki penjelasan sebagai berikut:

1. **Input**

*Input* adalah data yang masuk yang akan diproses, dimana pada proses pembelajaran (*training*) *perceptron* akan mempelajari data *input* dengan melatih *lernable paramters (weight* dan *bias)* agar mampu mencapai tujuan pembelajaran yang diinginkan. Pada saat *testing perceptron* akan membuat prediksi dari *input* dengan *weight* dan *bias* yang sudah dilatih pada proses *training.*

1. **Weight**

*Weight* adalah salah satu komponen *learnable parameter* pada *perceptron,* dalam *perceptron* *weight* memiliki peran sebagai penyambung antar *input* dengan *neuron* dan memiliki kemampuan untuk menguatkan atau melemahkan signal yang diterima (*input*) dengan cara melakukan operasi perkalian dengan nilai *weight* (bobot). Dimana semakin besar nilai bobot yang digunakan maka signal input akan semakin menguat begitu juga sebaliknya (Soares & Souza, 2016). Pada proses *training* kekuatan *weight* (bobot) akan disesuaikan (dilatih) agar nilai *weight* dapat sesuai dengan tujuan yang ingin dicapai. Sebagai contoh persamaan linear sederhana dibawah.

Dalam kasus persamaan linear sederhana diatas tujuan yang ingin dicapai adalah 8 dan *input* yang akan diproses adalah 2 dan *weight* yang akan dipelajari agar mampu memenuhi tujuan adalah variabel . Dimana *perceptron* akan melatih variabel agar mampu mendekati angka 4 sehingga tujuan pembelajaran dapat dimaksimalkan.

1. **Bias**

Bias juga adalah komponen *learnable parameter* pada *perceptron.* Dalam struktur *perceptron* bias memiliki kemampuan untuk mengubah hasil proses variabel *weight.* Kemampuan bias dalam mengubah hasil proses *weight* sangat diperlukan untuk menggenapkan kekurangan hasil proses variabel *weight.* Sebagai contoh persamaan linear yang sama:

Dalam kasus linear diatas sebagai contoh nilai *w* yang berhasil dipelajari adalah 3.5 tanpai variabel hasil dari , dalam kasus ini perceptron akan melatih variabel agar bisa menggenapkan hasil proses agar dapat memenuhi nilai 8, dengan memaksimalkan variabel mendekati nilai 1 maka tujuan pembelajaran dapat dimaksimalkan.

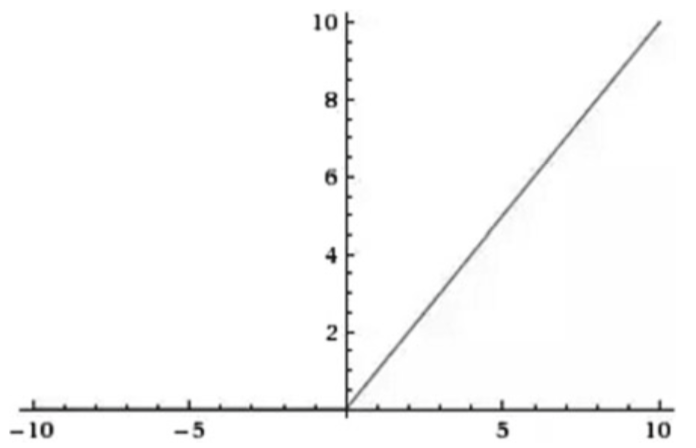
1. **Aktivasi**

*Output* dari unit *neuron* dihasilkan dari sebuah fungsi aktivasi. Fungsi aktivasi dalam *perceptron* berfungsi untuk menghasilkan nilai *nonlinear* dimana ini dibutuhkan karena secara alami *neuron* pada otak manusia juga berperilaku secara *nonlinear.* Fungsi aktivasi biasanya dibatasi antar dua rentang nilai yang menyebabkan hasilnya *nonlinear,* namun ada beberapa fungsi aktivasi yang menghasilkan fungsi linear (Soares & Souza, 2016). Ada banyak fungsi aktivasi yang bisa digunakan dalam ANN, beberapa contoh fungsi aktivasi yang bisa digunakan adalah sebagai berikut:

* + 1. ReLU

*Rectified Linear Unit* (ReLU) adalah fungsi aktivasi linear yang membatasi nilai output minimal pada nilai 0 (Wani, Bhat, Afzal, & Khan, 2018). Bentuk fungsi aktivasi ini bisa dilihat seperti dibawah.

Jika dibentuk dalam grafik output dari ReLU terlihat seperti gambar 2.3 dibawah.

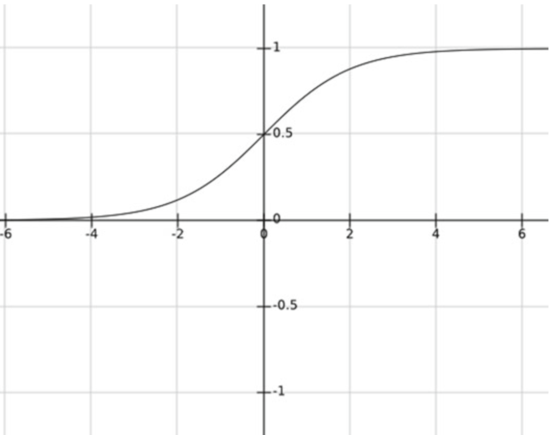


Gambar 2.3 Grafik Aktivasi ReLU

* + 1. Sigmoid

*Sigmoid* adalah fungsi nonlinear yang juga dikenal dengan istilah *logistic function.* Fungsi *Sigmoid* memiliki *output* dalam rentang [0,1] (Wani et al., 2018). Bentuk fungsi *Sigmoid* terlihat dalam persamaan dibawah

Persamaan diatas jika dibentuk dalam grafik akan terlihat seperti gambar 2.4 dibawah.

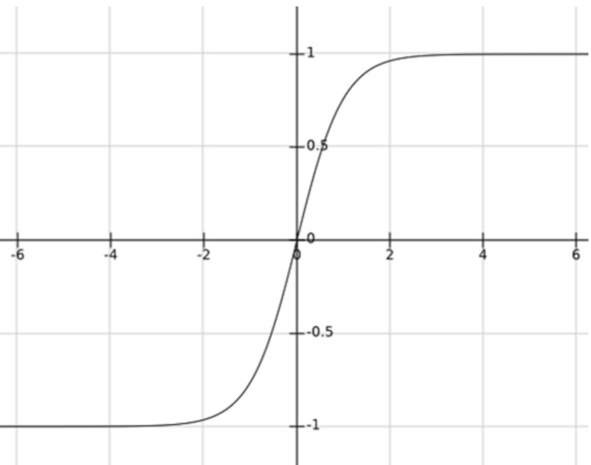


Gambar 2.4 Grafik Aktivasi *Sigmoid*

* + 1. ﻿Hyperbolic Tangent (Tanh)

Fungsi aktivasi Hyperbolic Tangent atau dikenal juga dengan nama fungsi Tanh adalah fungsi nonlinear seperti *Sigmoid* namun memetakan output dalam rentang [-1,1]. Kelebihan fungsi Tanh dari Sigmoid adalah nilai input negatif akan ditekan dengan kuat menjadi nilai negatif dan nilai input nol akan dipetakan menjadi nilai mendekati 0 (Wani et al., 2018). Bentuk matematis dari fungsi Tanh terlihat seperti dibawah.

Persamaan diatas jika dipetakan dalam grafik akan terlihat seperti gambar 2.5 dibawah.



Gambar 2.5 Grafik Fungsi Tanh

* + 1. ﻿Softmax

Fungsi aktivasi Softmax adalah fungsi yang banyak digunakan dalam kasus klasifikasi. Bentuk aktivasi ini terlihat seperti dibawah.

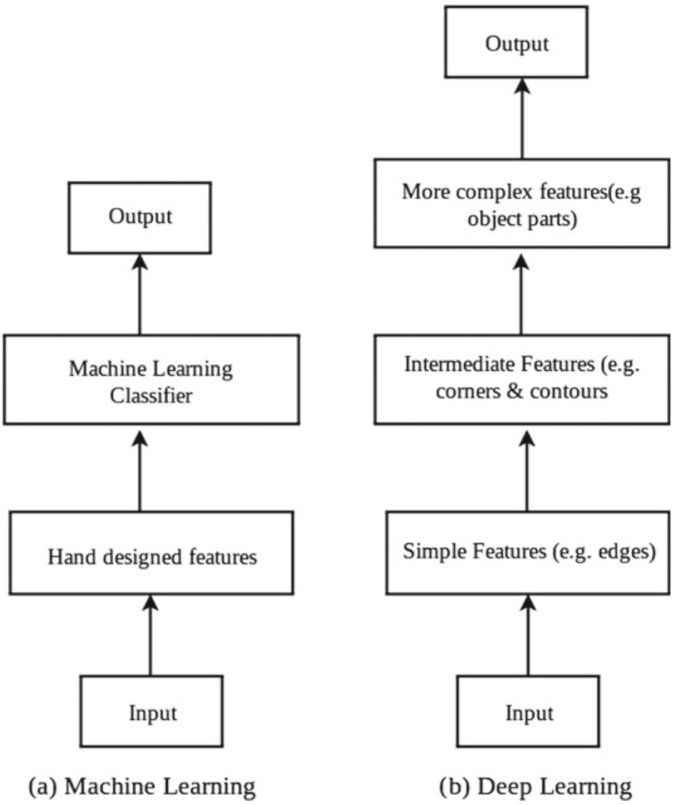
1. **Deep Learning**

*Deep Learning* adalah area baru dari *machine learning* yang semakin populer pada dekade ini. Istilah *Deep Learning* merujuk pada arsitektur ANN yang terdiri dari beberapa *hidden layers* (*Deep Networks*) yang berfungsi untuk mengenali fitur-fitur yang berbeda pada *input*. Algoritma *Deep Learning* bertujuan untuk mencari struktur yang tidak diketahui untuk mendapatkan representasi yang baik dari *input* (Wani et al., 2018).

Metode *machine learning* yang konvensional terbatas dalam cara memproses data mentah. Dalam waktu yang lama untuk dapat mengenal pola menggunakan metode *machine learning* membutuhkan keahlian khusus dalam bidang-bidang tertentu dan membutuhkan keahlian rekayasa yang sangat hati-hati sesuai kasus yang dihadapi untuk dapat mengekstraksi fitur dari sebuah *input*.

Namun dengan *Deep Learning* proses pengenalan pola dan rekayasa secara manual yang membutuhkan keahlian dalam domain tertentu tersebut dapat dilakukan secara otomatis selama proses pembelajaran. Dalam proses pembelajaran (*training*) metode *Deep Learning* dapat mengenali fitur-fitur yang ada pada data mentah secara otomatis dengan hasil yang lebih baik dari cara manual (Wani et al., 2018).

Metode pembelajaran fitur pada Deep Learning dilakukan dengan membentuk layer hirarki dimana setiap layer dibangun diatas layer yang lain. Layer yang paling bawah dari model ini bertanggung jawab untuk mempelajari representasi dasar dari masalah, dan layer-layer diatasnya bertanggung jawab untuk membentuk konsep yang lebih kompleks dari data. Sebagai contoh dalam kasus pengenalan wajah pada gambar, setiap pixel yang ada pada gambar akan dimasukkan ke hirarki layer. Setiap hidden layer pada struktur hirarki ini kemudian akan mengekstrak fitur-fitur dari gambar input. Pada layer pertama hirarki ini akan mendeteksi tepi-tepi dari wajah, kemudian pada pada layer kedua akan dipelajari garis-garis pipi, alis, cekungan pada dagu, pada layer berikutnya garis-garis yang telah dipelajari akan membentuk gambaran yang lebih abstrak dari gambar. Semua pembelajaran yang dilakukan pada setiap layer ini dilakukan secara otomatis tanpa campur tangan manusia. Secara visual perbedaan tehnik *machine learning* konvesional dan *deep learning* bisa dilihat pada gambar 2.6 dibawah.

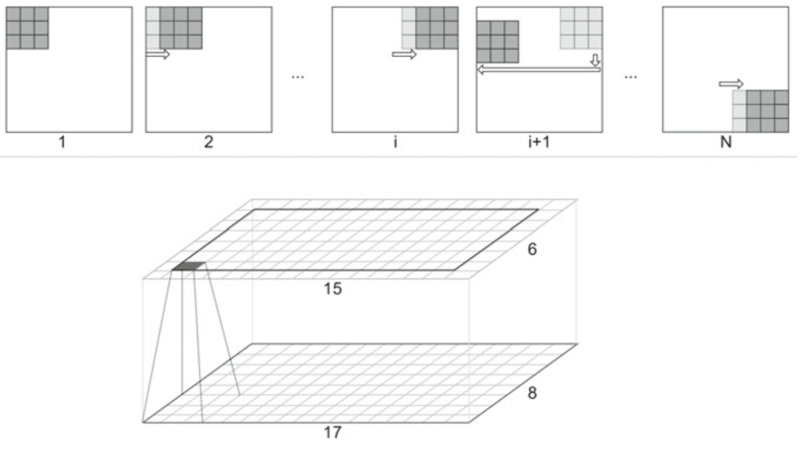


Gambar 2.6 Perbedaan *Machine Learning* dan *Deep Learning*

Adapun layer ekstraksi fitur yang bisa digunakan pada arsitektur *Deep Learning* adalah sebagai berikut.

1. **Layer Konvolusi**

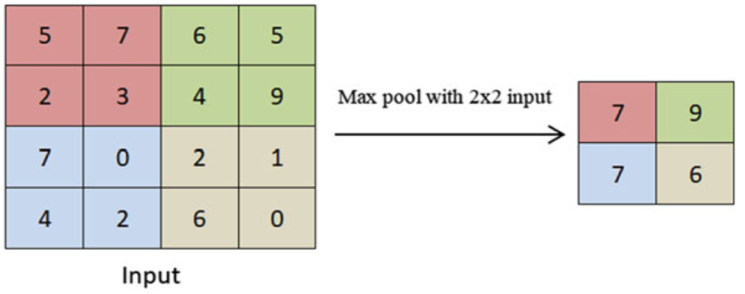
LayerKonvolusi (*Convolutional Layer*)adalah layer operasi konvolusi 2D, dimana dengan operasi konvolusi setiap blok wilayah pada gambar dapat dipelajari dan menjadi fitur dari gambar tersebut. Komponen dari operasi konvolusi adalah matriks persegi yang disebut filter/kernel umumnya berdimensi 3x3, filter/kernel pada operasi konvolusi berfungsi untuk mengekstrak fitur setiap wilayah yang ada pada gambar. Dalam proses *training* filter/kernel akan menjadi *learnable parameter* yaitu setiap nilai matriks kernel akan dipelajari pada saat *training*. Pada setiap layer *Convolutional* jumlah kernel/filter bisa lebih dari satu sesuai kompleksitas dan jumlah fitur yang ingin dipelajari dari input gambar. Sebagai contoh jika pada sebuah gambar dengan satu kanal warna ingin dipelajari 64 jenis fitur maka pada layer *convolutional* bisa memiliki 64 jenis kernel/filter dengan dimensi sehingga hasil dari operasi layer *convolutional* adalah 64 gambar atau juga disebut 64 kanal (Skansi, 2018). Ilustrasi proses konvolusi bisa dilihat pada gambar 2.7 dibawah.



Gambar 2.7 Ilustrasi Proses Konvolusi

1. **Max Pool**

Layer *Max Pool* pada lapisan jaringan ANN berfungsi untuk mereduksi ukuran gambar dengan metode *down sampling.* Layer *Max Pool* akan menyimpulkan nilai tertinggi pada gambar dalam ukuran region tertentu (Wani et al., 2018). Sebagai contoh gambar 2.8 dibawah operasi *Max Pool* akan mengambil nilai tertinggi pada gambar dalam setiap 2x2 wilayah pada gambar.



Gambar 2.8 Ilustrasi Operasi *Max Pool*

1. **Batch Normalization**

Proses *Batch Normalization* (BN)pada lapisan ANN berfungsi untuk menormalisasi skala nilai *input*. Menurut Sergey Ioffe (2015) dalam proses *training* menambahkan lapisan BN dapat meningkatkan generalisasi pada model sehingga model dapat belajar dengan baik tanpa *overfit* (Ioffe & Szegedy, 2015)*.* Proses BN bisa dilihat dalam persamaan dibawah (Wani et al., 2018).

**Dimana:**

adalah hasil proses BN

adalah data yang akan dinormalisasi

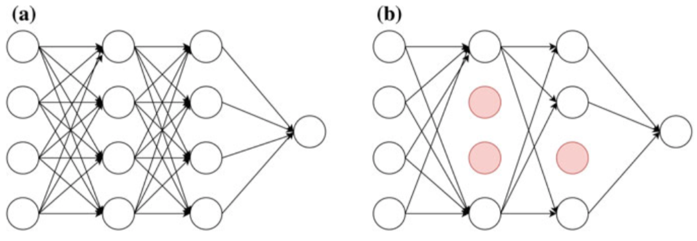
adalah nilai rata-rata pada *batch*

adalah nilai variansi dalam batch

Menumpuk layer Konvolusi dengan *Max Pool* menjadi beberapa layer disebut dengan CNN (*Convolutional Neural Network*).

1. **Dropout**

Lapisan jaringan ANN yang terdiri dari beberapa lapis yang kompleks dapat membuat model mempelajari fitur yang rumit pada data, setiap fitur yang dipelajari akan tersambung ke sejumlah *neuron* atau disebut *Fully Connected Layer* (FC) untuk membuat kesimpulan dari fitur yang didapat.LayerFCakan saling terhubung dengan layer FC yang lain dan sangat mudah mengalami *overfit. Overfit* adalah keadaan dimana model mempelajari data *training* dengan sangat baik namun gagal dalam memprediksi kesimpulan dari data baru, untuk mengatasi hal ini layer *dropout* dapat ditambahkan pada lapisan ANN. Layer *dropout* akan memutus secara acak sambungan FC pada saat *training* sehingga hanya sambungan yang masih terhubung yang akan dilatih pada saat *training* (Wani et al., 2018)*.* Ilustrasi proses *dropout* bisa dilihat pada gambar 2.9 dibawah.



Gambar 2.9 Ilustrasi Proses *Dropout*

1. **Training**

*Training* dalam *Deep Learning* adalah proses melatih model untuk mempelajari data untuk mendapatkan hasil yang diinginkan. Dalam proses *training* ada beberapa komponen yaitu:

1. **Epoch**

*Epoch* dalam *training* merujuk pada berapa kali semua data *training* akan digunakan untuk belajar. Menggunakan data *training* dalam melatih model tidak cukup hanya sekali, dibutuhkan beberapa kali agar model dapat belajar dengan baik.

1. **Batch Size**

*Batch Size* dalam *training* merujuk pada jumlah data yang akan diproses untuk latihan dalam satu waktu. Metode *deep learning* pada umumnya membutuhkan data dalam jumlah besar agar dapat belajar dengan baik. Dalam proses belajar, memasukkan keseluruhan data ke dalam jaringan dalam waktu bersamaan akan memakan banyak sumber daya memory dan operasi komputasi. Agar dapat diproses dengan baik data yang dalam jumlah banyak akan dipecah menjadi beberapa bagian yang disebut dengan *mini batch* dimana jumlah data setiap *mini batch* disebut dengan *batch size* dan pemrosesan setiap *batch* disebut dengan iterasi*.* Umumnya ukuran *batch size* yang digunakan adalah kelipatan 2 misal 32, 64, 128 jumlah data.

1. **Backpropagation**

*Backpropagation* adalah algoritma yang sangat banyak digunakan untuk melatih model *deep learning.* Algoritma *backpropagation* akan mengubah nilai *learnable parameters (weight* dan *bias)* dimulai dari layer terakhir hingga layer pertama berdasarkan selisih (*error/cost/loss*) dari nilai prediksi dengan nilai yang diinginkan hingga nilai selisih (*error*) dapat diminimalisir.

Dalam ANN yang terdiri dari satu layer (*perceptron*) mengubah nilai *weight* sangat mudah dilakukan dengan algoritma *Gradient Descent* dimana perubahan nilai weight baru bisa diberikan dengan persamaan dibawah (Sebastian Raschka, 2015).

Sehingga:

adalah nilai perubahan *weight* yang baru yang didefinisikan melalui turunan parsial hingga mendapatkan persamaan akhir yaitu

**Dimana:**

adalah konstanta *learning rate*

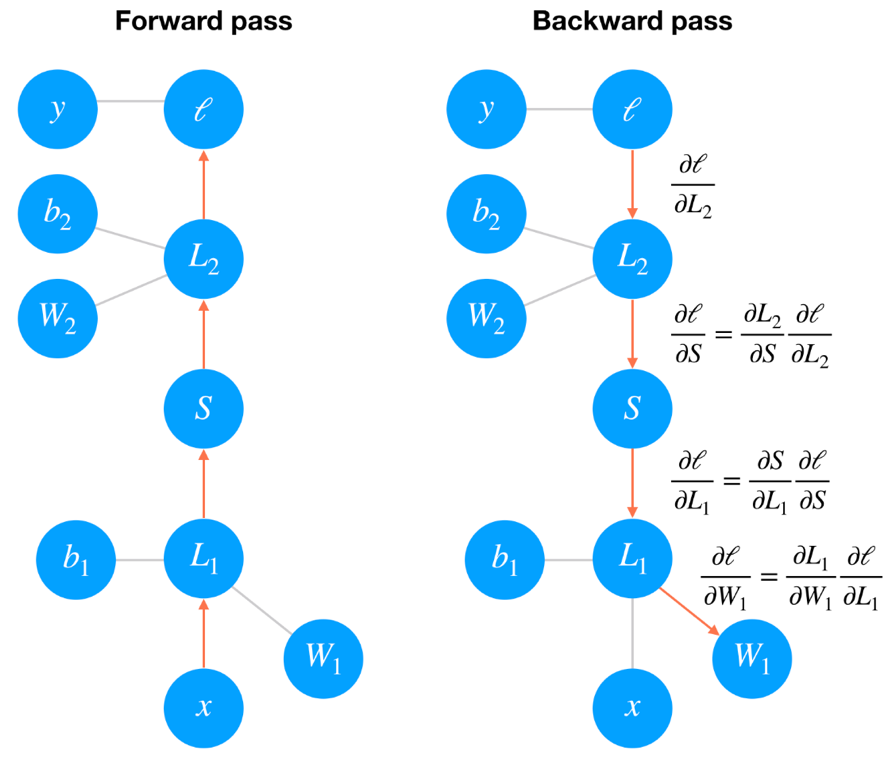
adalah nilai target yang diinginkan

adalah nilai prediksi

adalah data *training*

Dengan persamaan diatas perubahan nilai *weight* akan ditekan sesuai selisih dari target dan prediksi, jika nilai target dan prediksi sama maka nilai artinya tidak akan terjadi perubahan nilai *weight,* perubahan yang dilakukan dalam algoritma *gradient descent* adalah untuk mencari *local/global minima*.

Berbeda dengan kasus *single layer* diatas, jaringan yang memiliki banyak layer cukup sulit untuk dilatih dengan metode *Gradient Descent* biasa, disinilah peran *backpropagation* (Udacity Course, 2018)*. Backpropagation* digunakan untuk melatih jaringan dengan banyak layer, menurut Jeremy Howard dalam penerapannya *backpropagation* adalah terapan dari *chain rule* pada kalkulus. Sebagai contoh gambar 2.10 dibawah.



Gambar 2.10 Ilustrasi *Forward* dan *Backward Pass*

Dalam gambar *Forward pass* diatas data akan dimasukkan dari bawah hingga atas. Data akan diproses pada layer dengan *weight*  dan bias lalu hasil proses akan ditransformasi pada fungsi aktivasi Sigmoid lalu terdapat operasi pada layer kedua dengan *weight*  dan bias kemudian akan dihitung kesalahan prediksi dari nilai target yang diinginkan menggunakan fungsi *loss* , setelah mendapatkan nilai *loss* (*error*) akan dilakukan proses pengubahan nilai hingga nilai *loss* (*error*)dapat diminimalisir. Agar dapat melatih jaringan diatas menggunakan *gradient descent,* akan dihitung nilai *gradient* dari *loss* lalu disebarkan (*propagate*) hingga layer pertama pada jaringan. Setiap operasi memiliki *gradient* antara *input* dan *output,* pada saat proses penyebaran (*propagate*) *gradient* ke layer pertama (*backward*) akan dilakukan proses perkalian antara nilai *gradient* layer sebelumnya dengan nilai *gradient* pada operasi layer saat ini,seperti terlihat pada gambar *backward pass* gambar 2.10 diatas. Secara matematis, proses ini akan menghitung nilai *gradient* dari *loss* sesuai nilai *weight* dengan aturan berantai (*chain rule*).

Nilai *weight* baru adalah hasil dari perhitungan *gradient* diatas dengan perkalian *learning rate* sehingga nilai weight baru bisa dihitung dengan persamaan dibawah.

Nilai konstanta *learning rate* diberikan sehingga perubahan nilai *weight* dapat bergerak dengan cukup pada proses iterasi agar nilai *loss* tetap berada pada area minimum.