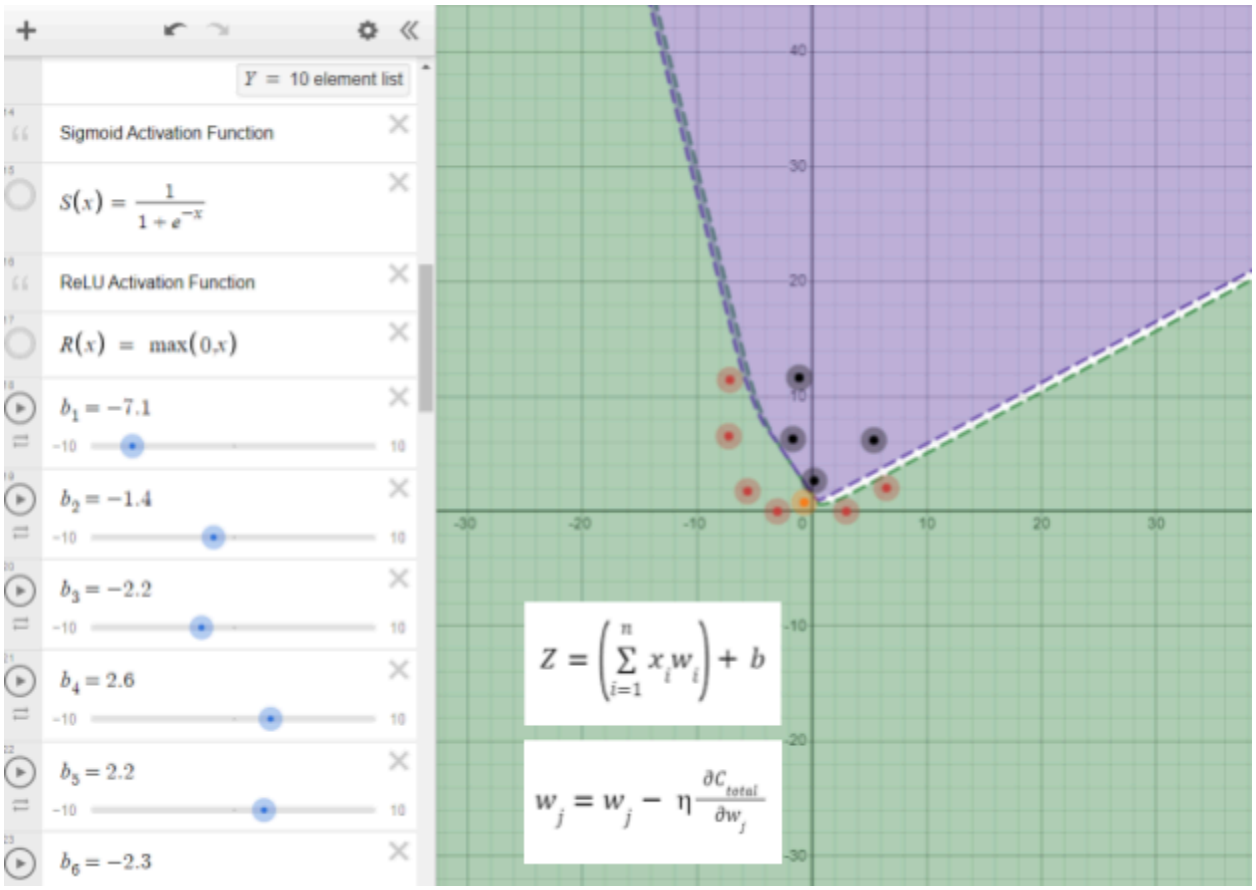
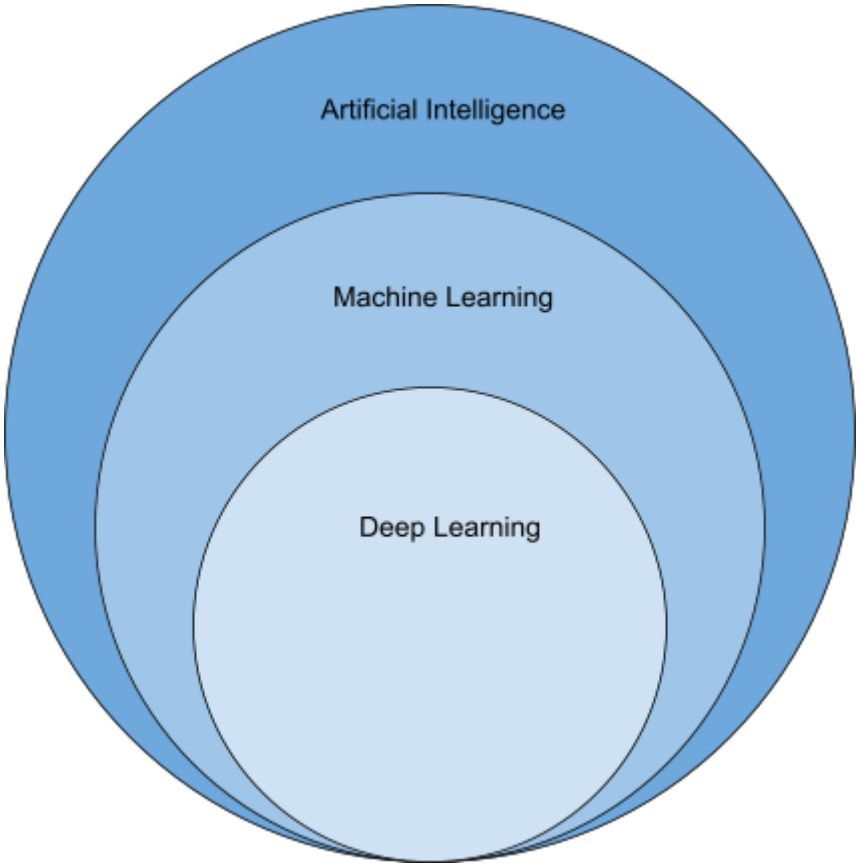
Deep Learning, Calculus, Linear Algebra dan Javascript

Oleh Wuriyanto

http://wuriyan.to



Artificial Neural Network, salah satu **Machine Learning Model** di belakang semua *advance tech* yang mungkin anda gunakan setiap hari. Sebut saja **Google translate**, **Amazon Alexa**, **ChatGPT**, **Dall-E**, dan banyak lagi. Tidak bisa dipungkiri, *advance tech* tersebut sudah menjadi bagian dari hidup kita.



Sebagian dari kita mungkin kadang lupa dan sulit membedakan istilah **AI**, **Machine Learning** dan **Deep Learning**. Sebenarnya tiga istilah tersebut adalah *subset* satu sama lain.

1. **AI** => kemampuan mesin (komputer) meniru tingkah laku manusia.
2. **Machine Learning** => Aplikasi atau program Komputer yang dapat belajar dengan sendirinya secara otomatis.
3. **Deep Learning** => Aplikasi atau program **Machine Learning** yang memanfaatkan **Algoritma Neural Network** dengan arsitektur layer yang kompleks dan dalam (*deep*).

Dalam buku ini, kita akan melakukan eksplorasi **Matematika** dibelakang **Deep Learning**. **Matematika** yang akan kita pelajari adalah **Calculus**, fokus *core* **Calculus** yang kita pelajari adalah **Derivative (Turunan)**. **Derivative (Turunan)** akan membantu kita dalam mengoptimasi fungsi dan parameter **Artificial Neural Network** yang akan kita bangun. Kita juga akan belajar **Linear Algebra** (Aljabar Linier). **Linear Algebra** (Aljabar Linier) akan membantu kita pada saat mengabstraksi kebutuhan data dan kumpulan data (*Datasets*). Kita akan mempelajari apa itu *Vector*, *Matrix* dan operasinya.

Pada akhir buku ini, kita akan membangun *mini project* **Artificial Neural Network** hanya dengan **Plain/Pure Javascript** dengan formula **Matematika** yang telah kita pelajari.

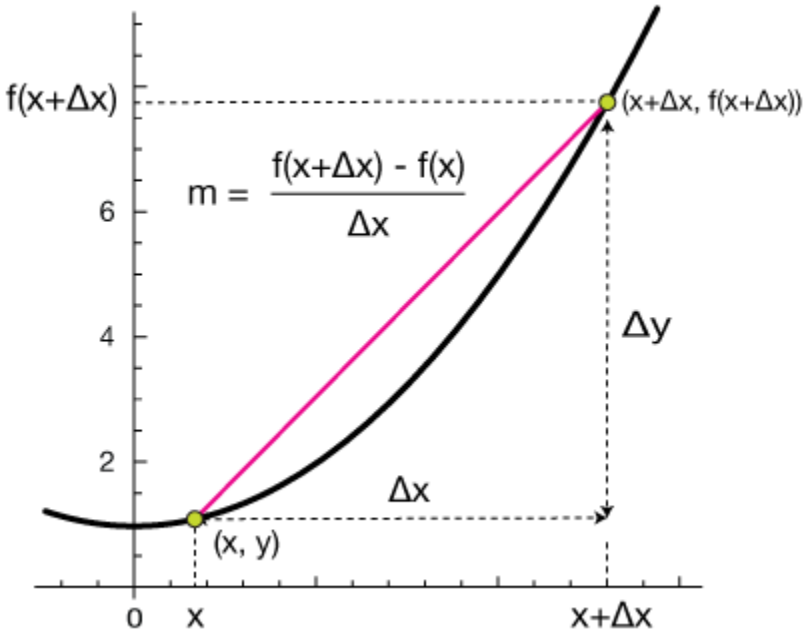
Catatan: Sebagian besar istilah dari buku ini akan menggunakan istilah asli dalam Bahasa Inggris. Tujuannya adalah agar anda bisa dengan mudah melanjutkan ke pembahasan diluar buku ini yang lebih *advance*, dan terbiasa dengan istilah-istilah yang ada dalam buku ini.

Calculus Derivative (Turunan) & Rate of Change (Tingkat Perubahan)

Mungkin anda bertanya-tanya, kenapa **Calculus** menjadi *prerequisite* untuk memahami alur bagaimana melakukan training terhadap **Artificial Neural Network(ANN)**. **Calculus** menjadi *prerequisite,* sebab tujuan akhir dari melatih **ANN** adalah meminimalkan fungsi **Loss/Error** dengan cara mengamati tingkat perubahan setiap parameter dan outputnya. Memahami **Derivative(Turunan)** akan membantu kita dalam proses *minimum optimization* pada fungsi **Loss/Error**.

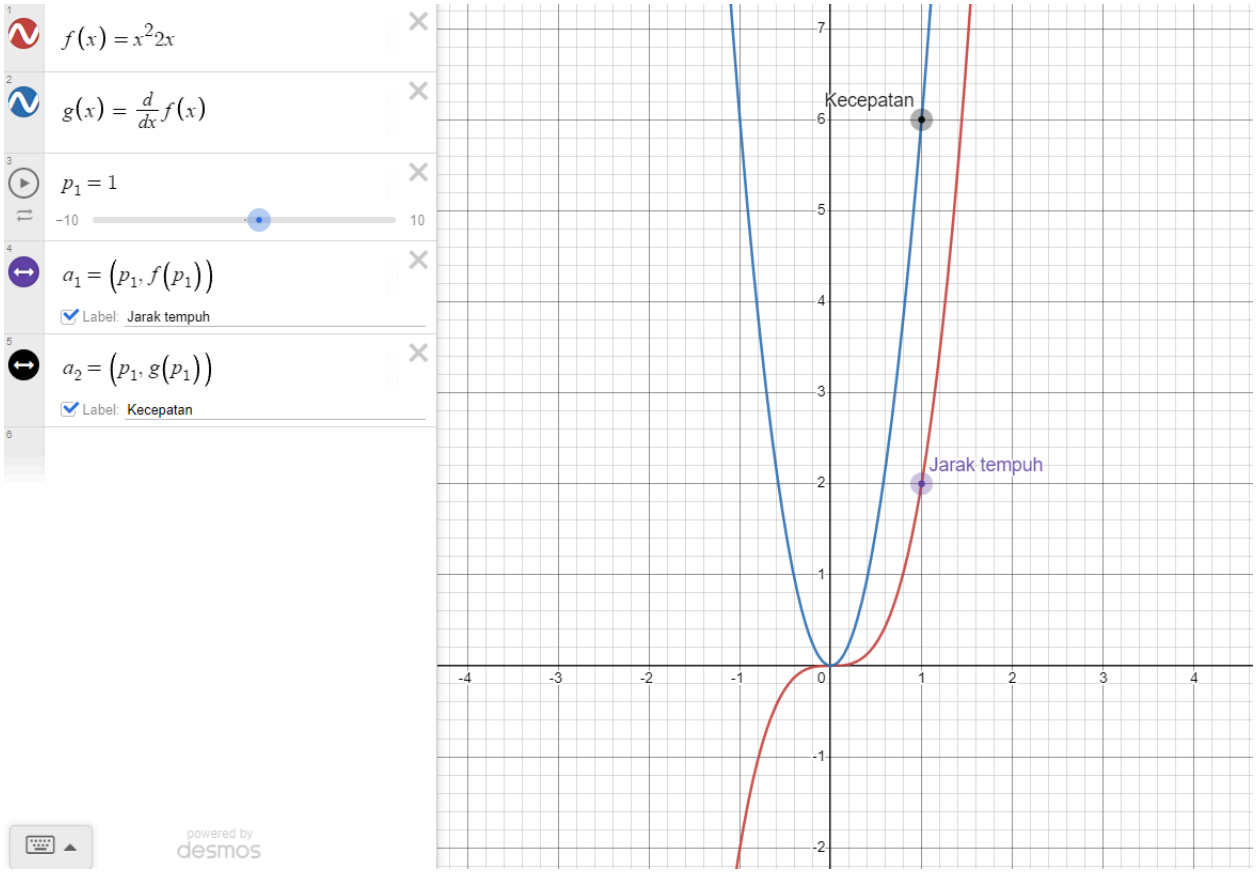
Bentuk umum Derivative *(Definition of Derivative/ First Principle)*

$f'(x) = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \frac{f(x+\Delta x) - f(x)}{\Delta x}$ Notasi $f'(x)$ atau $\frac{\Delta y}{\Delta x}$ atau $\frac{dy}{dx}$ atau $\frac{\partial y}{\partial x}$ $(Partial\ Derivative)$



Pada dasarnya notasi diatas dapat kita interpretasikan "bagaimana Δy berubah, ketika kita ubah sedikit nilai Δx". Δy = *delta* dari y, dan Δx *delta* dari x. Sedangkan notasi $\lim_{\Delta x \to 0}$ memberi tahu ketika dua titik pada *Secant line* semakin dekat, sehingga nilai Δx (Delta x) mendekati 0, tetapi tidak 0. Karena semakin kecilnya Δx (*Delta* x), kemudian *Secant line* tersebut berubah menjadi *Tangent line*. Dengan kata lain, semakin kecil Δx (*Delta* x) maka semakin akurat juga nilai perkiraan (*approximation*) dari *average rate*-nya.

Derivative adalah kebalikan dari *Integral*. Dengan *Derivative*, kita bisa mengukur tingkat perubahan sebuah fungsi pada titik tertentu. Sebagai contoh, kita bisa memperkirakan kecepatan sebuah mobil pada titik jarak dan waktu tertentu (sumbu x merepresentasikan waktu, dan sumbu y merepresentasikan jarak). Dengan catatan kita mengetahui *distance function* dan time *function*-nya. Sedangkan *Integral* adalah kebalikannya. Kita akan melakukan *Integration(integral)* ketika kita ingin memperkirakan jarak yang ditempuh pada kecepatan dan waktu tertentu (sumbu x merepresentasikan waktu, dan sumbu y merepresentasikan kecepatan). Dengan catatan kita mengetahui *velocity function* dan *time function*-nya. Sebagai contoh ketika kita mengendarai Supercar dengan data yang sudah kita analisa menghasilkan *Distance function* $f(x) = x^2 2x$.



Distance function: $f(x) = x^2 2x$

Kita bisa memperkirakan kecepatan *supercar* yang kita kendarai pada titik jarak dan waktu tertentu dengan mencari *Derivative* (menurunkan) *Distance function* diatas.

Dengan menggunakan bentuk umum *Derivative*

$f'(x) = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \frac{f(x+\Delta x) - f(x)}{\Delta x}$

Derivative dari fungsi $f(x) = x^2 2x$:

$f'(x) = \frac{(x+\Delta x)^2 2(x+\Delta x) - x^2 2x}{\Delta x}$

$= \frac{((x+\Delta x)(x+\Delta x))(2x+2\Delta x) - x^2 2x}{\Delta x}$

$= \frac{(x^2+x\Delta x+x\Delta x+\Delta x^2)(2x+2\Delta x) - x^2 2x}{\Delta x}$

$= \frac{(x^2+2x\Delta x+\Delta x^2)(2x+2\Delta x) - x^2 2x}{\Delta x}$

$= \frac{2x^3+2x^2\Delta x+4x^2\Delta x+4x\Delta x^2+2x\Delta x^2+2\Delta x^3-2x^3}{\Delta x}$

$= \frac{6x^2\Delta x+6x\Delta x^2+2\Delta x^3}{\Delta x}$ Bagi *numerator* term yang terdapat Δx dengan *denominator* Δx

$= 6x^2 + 6x\Delta x + 2\Delta x^2$

Subtitusikan $\lim_{\Delta x \to 0}$ ke dalam Δx

$= 6x^2 + 6x * 0 + 2 * 0^2$

$= 6x^2$

Dengan demikian, ketika x(waktu) bernilai 1, maka *velocity* dari mobil tersebut adalah $6 * 1^2 = 6\ unit$. Ketika x(waktu) bernilai 2, maka *velocity* dari mobil tersebut adalah $6 * 2^2 = 24\ unit$. Dan seterusnya.

Contoh Derivative 1:

$f(x) = 2x + 2$

Untuk fungsi diatas, kita bisa langsung substitusikan fungsi diatas ke dalam bentuk umum *Derivative*:

$f'(x) = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \frac{2(x+\Delta x) + 2 - (2x+2)}{\Delta x}$

Expand, dan kalikan *Negative Sign* dengan fungsi $-(2x + 2)$, sehingga menjadi:

$$= \frac{2x + 2\Delta x + 2 \ - \ 2x - 2}{\Delta x}$$

Cancel $2x$ dan 2

$$= \frac{2x + 2\Delta x + 2 \ - \ 2x - 2}{\Delta x}$$

$$= \frac{2\Delta x}{\Delta x}$$

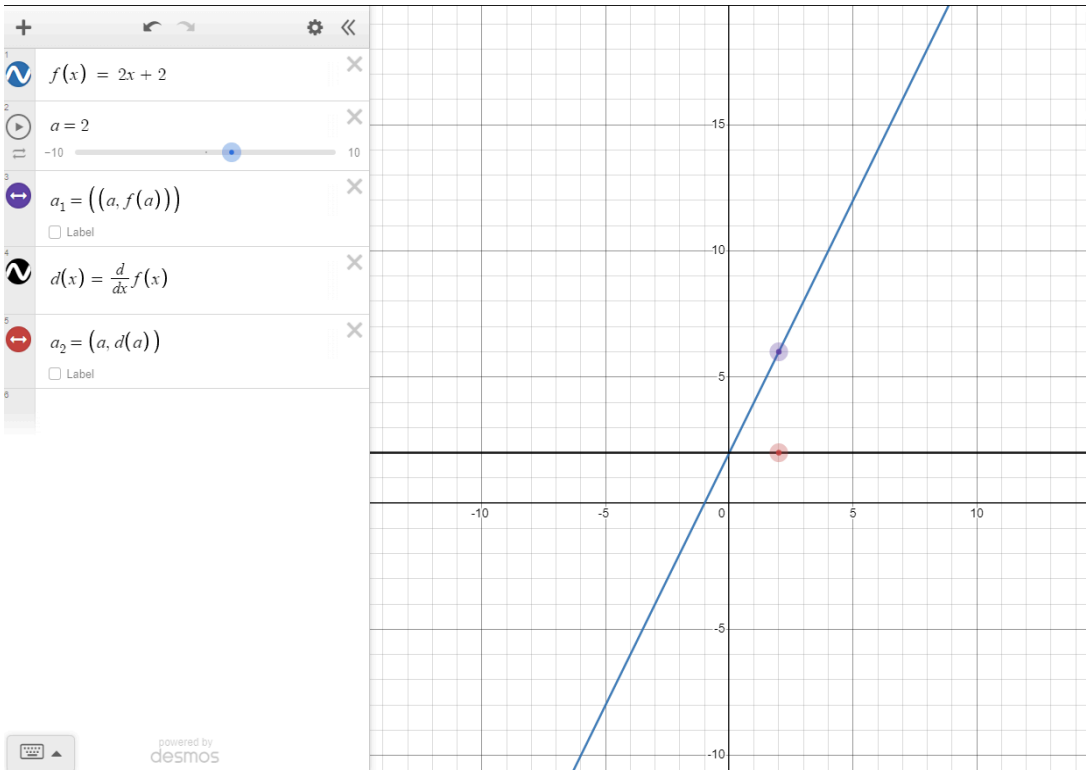
$$= \frac{\Delta x(2)}{\Delta x}$$

Cancel Δx, dengan cara membagi. Sehingga menjadi:

$$= \ 2$$

$$f'(x) \ = \ 2x \ + \ 2 \ = \ 2$$

Plot $f(x) \ = \ 2x \ + \ 2$ **dan** $\frac{\Delta y}{\Delta x} \ 2x \ + \ 2$



Bisa kita lihat pada fungsi garis lurus diatas, *Derivative* dari fungsi garis lurus selalu *coefficient* dari x. Artinya dimanapun kita menempatkan posisi x, derivative pada fungsi $f(x) = 2x + 2$ selalu menghasilkan *value* 2.

Contoh Derivative 2:

$f(x) = x^2$

Untuk fungsi diatas, kita bisa langsung substitusikan fungsi diatas ke dalam bentuk umum *Derivative*:

$$f'(x) = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \frac{(x + \Delta x)^2 - x^2}{\Delta x}$$

Expand:

$$= \frac{(x + \Delta x)\ (x + \Delta x) - x^2}{\Delta x}$$

Expand lebih lanjut

$$= \frac{x^2 + x\Delta x + x\Delta x + \Delta x^2 - x^2}{\Delta x}$$

$$= \frac{x^2 + 2x\Delta x + \Delta x^2 - x^2}{\Delta x}$$

Cancel x^2

$$= \frac{2x\Delta x + \Delta x^2}{\Delta x}$$

Sederhanakan $2x\Delta x + \Delta x^2$ dengan *Distributive Property*, $a(b + c) = ab + ac$ dengan *Common factor* Δx, sehingga menjadi

$$= \frac{\Delta x(2x + \Delta x)}{\Delta x}$$

Cancel Δx, sehingga menjadi

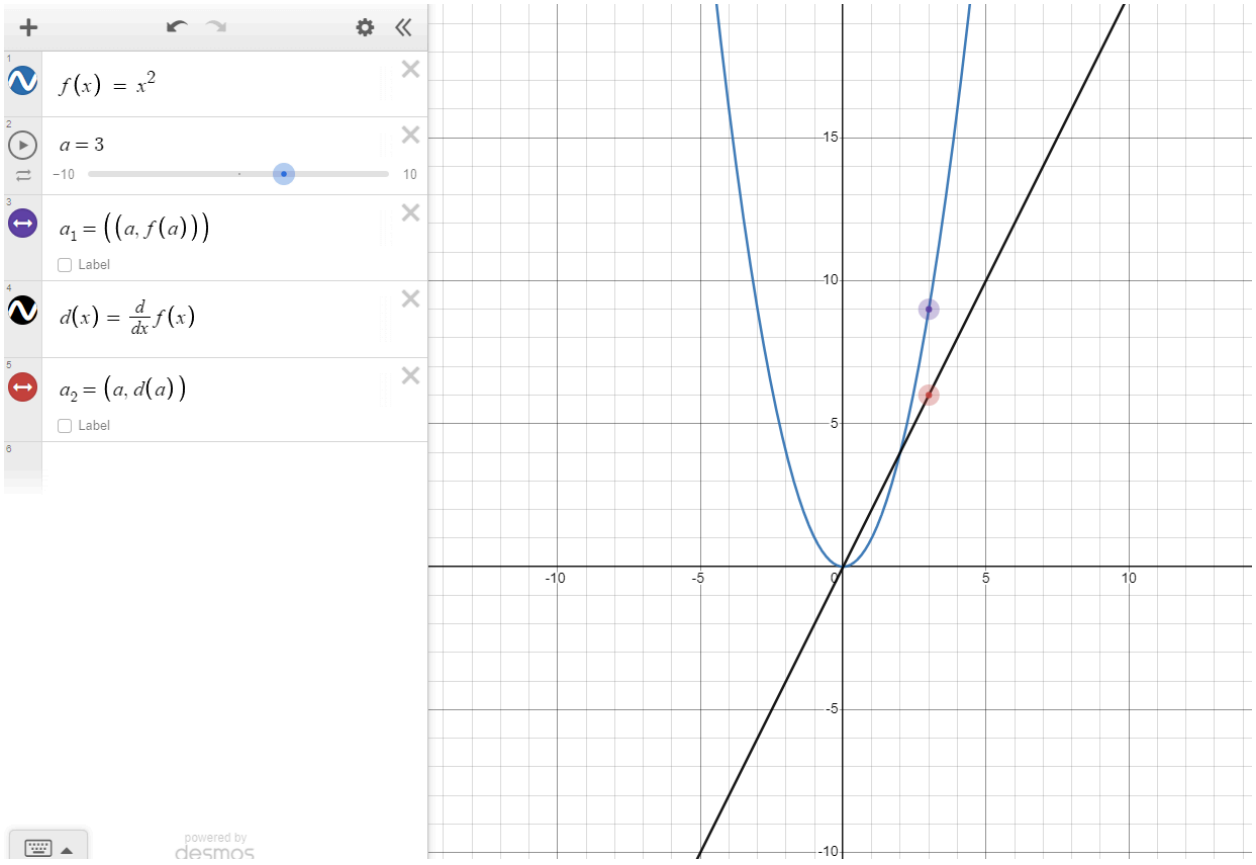
$= 2x + \Delta x$

Substitusikan 0 ke dalam Δx,

$$f'(x) = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = 2x + 0$$

$$f'(x) = \lim_{\Delta x \to 0} x^2 = 2x$$

Plot $f(x) = x^2$ dan $\frac{\Delta y}{\Delta x} x^2$



Graph di atas bisa kita artikan: setiap posisi x pada fungsi $f(x) = x^2$, tingkat perubahannya adalah $2x$.

Derivative Rule

Pada saat kita mencari *Derivative* sebuah fungsi, pada dasarnya kita jarang menggunakan bentuk umum:

$$f'(x) = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

Seringnya, untuk membantu mempercepat mencari *Derivative* (Turunan) sebuah fungsi, kita menggunakan *Derivative Rule*.

1. *Constant*
   Pada fungsi, $f(x) = 2x + 5$, ketika kita mencari $\frac{\Delta y}{\Delta x}$ kita memperlakukan 5 sebagai *Constant*. Pada *Derivative Rule*, *Constant* selalu bernilai 0. Sehingga $\frac{\Delta y}{\Delta x}\ 2x + 5 = 2 + 0 = 2$

2. *Exponential*
   $e^x = e^x$ dimana e adalah **Euler Number** = 2.718

3. *Logarithms*
   $ln(x) = \frac{1}{x}$, dimana $ln(x)$ adalah **Natural Logarithms**.

4. *Trigonometry*
   $sin(x) = cos(x)$
   $cos(x) = -sin(x)$
   $tan(x) = sec^2(x)$
   $arcsin(x) = \frac{1}{\sqrt{1-x^2}}$
   $arccos(x) = \frac{-1}{\sqrt{1-x^2}}$
   $arctan(x) = \frac{1}{\sqrt{1+x^2}}$
5. *Power Rule*
   $x^n = nx^{n-1}$, contoh $x^2 = 2x^{2-1} = 2x$

6. *Sum Rule*
   $f(a) + g(b) = f'(a) + g'(b)$

7. *Difference Rule*
   $f(a) - g(b) = f'(a) - g'(b)$

8. *Product Rule*
   $f(a)g(b) = f(a)g'(b) + f'(a)g(b)$

   Contoh *Product Rule* pada fungsi $f(x) = x^2 2x$:
   $f(a) = a^2$ dan $g(b) = 2b$
   $= x^2 * 2 + 2x * 2x$
   $f'(x) = x^2 2x = 6x^2$

9. *Quotient Rule*
   $\frac{f(a)}{g(b)} = \frac{f'(a)g(b) - f(a)g'(b)}{g(b)^2}$

10. *Reciprocal Rule*
    $\frac{1}{f(x)} = \frac{-f'(x)}{f(x)^2}$

11. *Chain Rule*
    $Notasi\ 1$: $\frac{\Delta y}{\Delta x} = \frac{\Delta y}{\Delta u} * \frac{\Delta u}{\Delta x}$
    $Notasi\ 2$: $f(g(x)) = f'(g(x))g'(x)$

    Chain Rule dibutuhkan ketika kita mencari *Derivative* pada *Composite Function*. *Composite Function* adalah operasi pada dua buah fungsi atau lebih yang saling terkomposisi.

    Jika:
    $y = f(u)$
    $u = g(x)$

    Bentuk dalam *Composite Function* menjadi
    $y = f(g(x))$

    Pada fungsi diatas, untuk mencari seberapa sensitif x terhadap y, kita menggunakan *Chain Rule*:

$$\frac{\Delta y}{\Delta x} = \frac{\Delta y}{\Delta u} * \frac{\Delta u}{\Delta x} = \frac{f'(u)}{\Delta u} * \frac{g'(x)}{\Delta x}$$

Contoh *Chain Rule*:

Jika kita mempunyai 2 buah fungsi, dan kedua fungsi tersebut adalah fungsi komposisi.

$y = f(u) = u2$
$u = g(x) = x + 1$

Sehingga untuk operasi *Composite*-nya adalah

$f(g(x)) = (x + 1) * 2$

Untuk mencari seberapa sensitif perubahan x terhadap y, atau dalam notasi *Derivative* $\frac{\Delta y}{\Delta x}$, maka kita perlu menggunakan *Chain Rule* untuk menyelesaikan permasalahan diatas.
Jika $x = 2$,
$f(g(2)) = (2 + 1) * 2$

$y = 6$

Dengan *Chain Rule*, kita akan mengetahui seberapa sensitif ketika $x = 2$, $x = 3$ dan seterusnya.

$$\frac{\Delta y}{\Delta x} = \frac{\Delta y}{\Delta u} * \frac{\Delta u}{\Delta x}$$

$$= \frac{f'(u)}{\Delta u} * \frac{g'(x)}{\Delta x}$$

Sebelumnya, kita perlu mencari *Derivative* kedua fungsi diatas.
$f(u) = u2$

$\frac{f'(u)}{\Delta u} = 2$, *Derivative* dari fungsi $u2$ adalah 2. Dengan *Power Rule*, u memiliki bentuk lain u^1,

maka

$$u^1 = u^{1-1} = u^0 = 1$$

Kita tau, sesuatu yang dipangkatkan ke 0 bernilai 1.

Sehingga

$f(u) = u2$ Derivative-nya adalah $\frac{f'(u)}{\Delta u} = 1 * 2 = 2$
Derivative untuk fungsi kedua.
$g(x) = x + 1$

$\frac{g'(x)}{\Delta x} = 1$, Derivative dari fungsi $x + 1$ adalah 1. Anda masih ingat *Constant Rule*. Semua constant bernilai 0. Untuk *Derivative* x , dengan *Power Rule* maka

$$x^1 = x^{1-1} = x^0 = 1$$

Sehingga
$g(x) = x + 1$ *Derivative*-nya adalah $\frac{g'(x)}{\Delta x} = 1 + 0 = 1$

$$\frac{\Delta y}{\Delta x} = \frac{\Delta y}{\Delta u} * \frac{\Delta u}{\Delta x} = \frac{f'(u)}{\Delta u} * \frac{g'(x)}{\Delta x} = 2 * 1 = 2$$

Sehingga dapat disimpulkan sensitivitas x terhadap y bernilai 2. Dengan kata lain, setiap penambahan 1 unit pada x akan berefek 2 unit penambahan pada y.

Pada saat melakukan *training* terhadap **Artificial Neural Network** (tahap **Backpropagation**) *Derivative Rule* akan sangat sering kita gunakan. Sehingga, kita akan sering membahas bagian ini pada bagian-bagian selanjutnya.

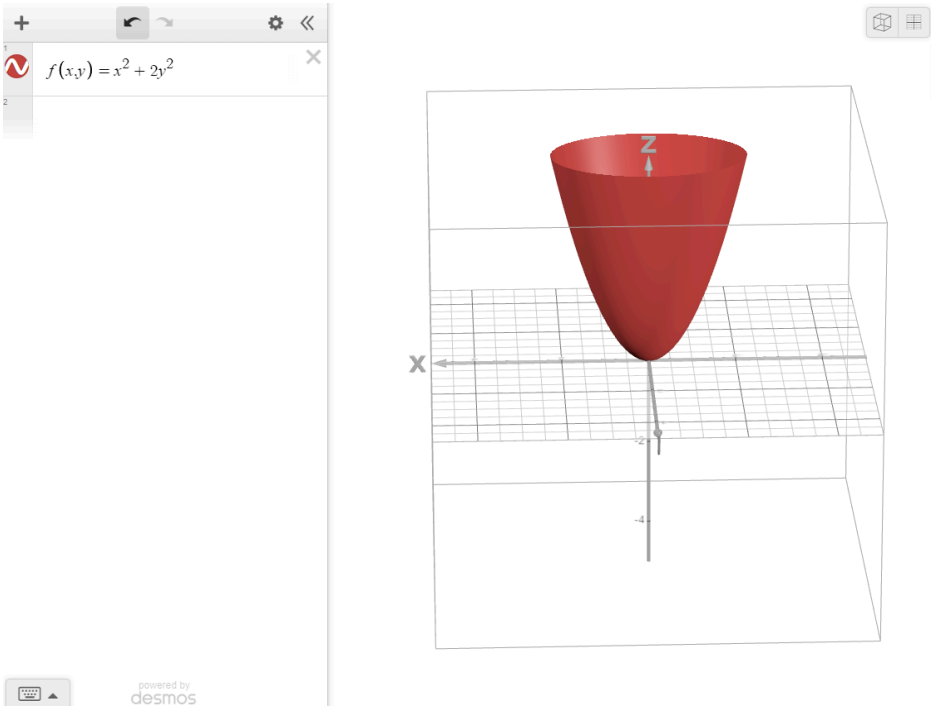
Multivariable Calculus dan Partial Derivative

$$f\colon R^n \rightarrow R$$

Multivariable Calculus adalah perluasan dari *Calculus* untuk menyelesaikan permasalahan fungsi yang mempunyai lebih dari satu parameter dan fungsi pada koordinat 3 dimensi. Pada beberapa pembahasan *Derivative* diatas, terutama pada bagian fungsi, kita hanya berfokus pada fungsi dengan 1(*single*) parameter saja. Pada implementasi yang sebenarnya, kita akan sering menjumpai fungsi dengan lebih dari satu parameter. Spesifik pada konteks **Artificial Neural Network**, fungsi yang kita gunakan seringnya mempunyai beberapa parameter sekaligus pada setiap *Neuron*-nya. Sehingga kita harus mencari sensitivitas setiap variabel atau parameter yang ada terhadap fungsinya dengan *Partial Derivative*.

Contoh *Parabolic* function dibawah ini. Fungsi tersebut mempunyai 2 buah parameter, x dan y, maka kita perlu mencari tahu bagaimana fungsi f berubah ketika x berubah, dan juga mencari tahu bagaimana fungsi f berubah ketika y berubah.

$$f(x,y) = x^2 + 2y^2$$



Untuk mencari *Derivative* dari fungsi:

$f(x, y) = x^2 + 2y^2$

Kita akan mencari masing-masing *Derivative* f terhadap variabel x dan variabel y dengan *Partial Derivative*.

Derivative f terhadap variabel x
$\frac{\partial f}{\partial x} = 2x$

Tentu anda masih ingat *Power Rule*, dengan *Power Rule* $x^2 = 2x^{2-1} = 2x$, sedangkan untuk y, kita perlakukan menjadi *Constant*. $2y^2 = 2 * 2 * 0^{2-1} = 0$.

Derivative f terhadap variabel y
$\frac{\partial f}{\partial y} = 4y$

dengan *Power Rule* $2y^2 = 2 * 2y^{2-1} = 4y$, sedangkan untuk x, kita perlakukan menjadi *Constant*. $x^2 = 2 * 0^{2-1} = 0$.

Sehingga *Derivative* dari fungsi

$f(x, y) = x^2 + 2y^2$ adalah $\frac{\partial f}{\partial (x,y)} = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right] = [2x, 4y]$

Derivative hubungannya dengan Deep Learning

Esensi dari melatih **Artificial Neural Network** adalah menemukan dan meminimalisir **Loss function**. **Loss function** adalah fungsi untuk mengukur seberapa optimal **Learnable Parameter** yang kita inputkan ke dalam **Neural Network Model** yang kita bangun. Parameter-parameter tersebut adalah *weights* dan *biases*. Semakin besar nilai dari **Loss function**, menandakan parameter yang kita inputkan (*weights* dan *biases*) belum optimal. Semakin kecil nilai dari **Loss function**, menandakan parameter yang kita inputkan sudah optimal dan semakin baik juga model yang kita bangun. Derivative dari **Loss Function** *with respect to Learnable parameters* $\frac{\Delta C}{\Delta \theta_i}$, notasi tersebut bisa kita interpretasikan menjadi “bagaimana perubahan total nilai ΔC, ketika kita ubah sedikit

dari nilai $\Delta\theta$ (*Delta* θ)". Bisa kita artikan juga dengan kalimat lain, "seberapa sensitif $\Delta\theta$ terhadap **Loss Function** ΔC". **Loss Function** selanjutnya akan kita bahas lebih detail pada bagian **Backpropagation**.

Derivative membantu kita mengoptimalkan model yang kita bangun dengan cara meminimumkan fungsi **Loss/ Error**.

Linear Algebra (Aljabar Linier)

Melatih **AI**, **Machine Learning**, dan **Deep Learning** tidak lepas dari kebutuhan Data. Sedangkan sumber data yang ada pada dunia nyata sangatlah beragam. Contoh ketika anda melatih **AI** untuk kebutuhan pengenalan wajah, anda membutuhkan data foto. Ketika anda melatih **AI** untuk kebutuhan pengenalan suara, anda membutuhkan data audio file seperti *wav* dan *mp3*. Ketika anda melatih **AI** untuk kebutuhan **Natural Language Processing(NLP)**, anda membutuhkan data *text*. Dengan Kebutuhan jenis data yang beragam dan kita tau, Komputer hanya bisa mengenali angka. Kita membutuhkan mekanisme yang bisa mengabstraksi berbagai jenis data tersebut dan meng-*encode*-nya menjadi deretan angka.

Linear Algebra (Aljabar Linier) adalah cabang studi **Matematika** yang mempelajari *Linear Equation(Persamaan Linier)*, dan representasinya pada *Vector Space* dan *Matrix*. Pengertian singkatnya, **Linear Algebra** adalah cabang studi **Matematika** yang berfokus pada *Vector*, *Matrix* dan operasinya.

Linear Algebra pada **AI** secara umum digunakan untuk mengabstraksi data yang akan digunakan untuk proses *training* maupun *inference* dengan cara meng-*encode*-nya menjadi deretan angka. Data-data seperti *image file*, *audio file*, dan *text file* akan di *encode* terlebih dahulu ke sebuah *Vector* maupun *Matrix*.

Linear Equation dan Linear System Equation

Secara umum, pengertian **Linear Equation** adalah persamaan yang setiap variabelnya memiliki pangkat tertinggi 1. Sedangkan **Linear System Equation** adalah dua atau lebih **Linear Equation** yang memiliki variabel yang sama (dari sisi jumlah variabel dan nama variabel).

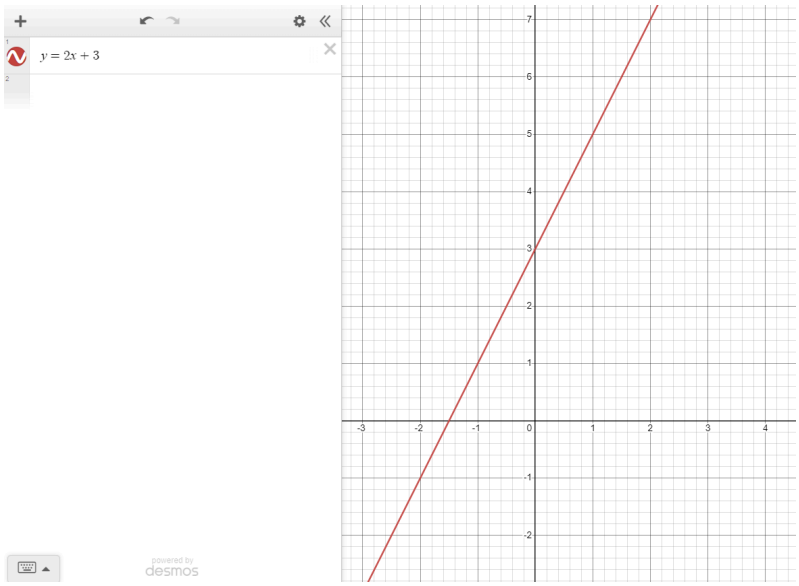
Bentuk umum **Linear Equation**(Persamaan ini disebut juga *Slope-Intercept form*):
$y = wx + b$

$w = slope\ (mengatur\ steepness/kecuraman\ dari\ garis\ yang\ kita\ buat)$
$b = intercept(titik\ dimana\ garis\ memotong\ sumbu\ y\ (pada\ saat\ x = 0))$

Misal kita mempunyai fungsi:
$y = 2x + 3$

Dimana:
$2 = Coefficient$
$x = Variable$
$3 = Constant$

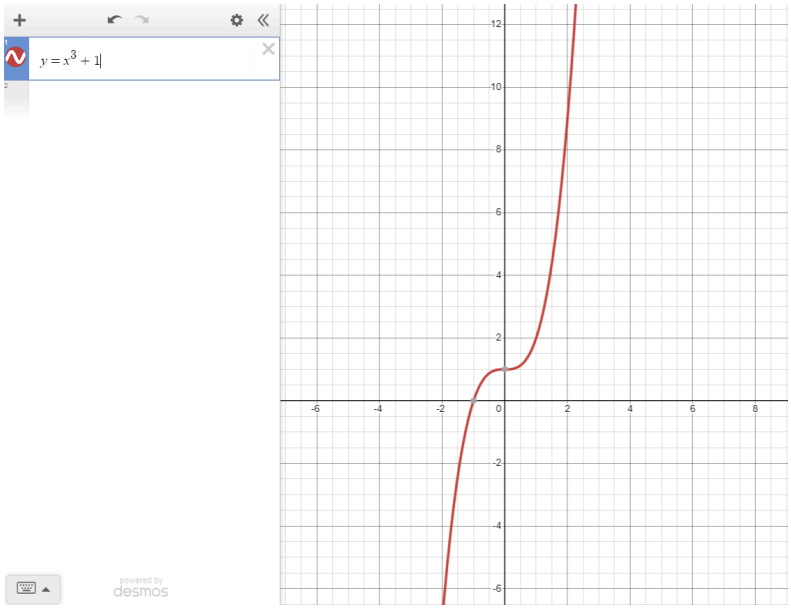
Plot $y = 2x + 3$



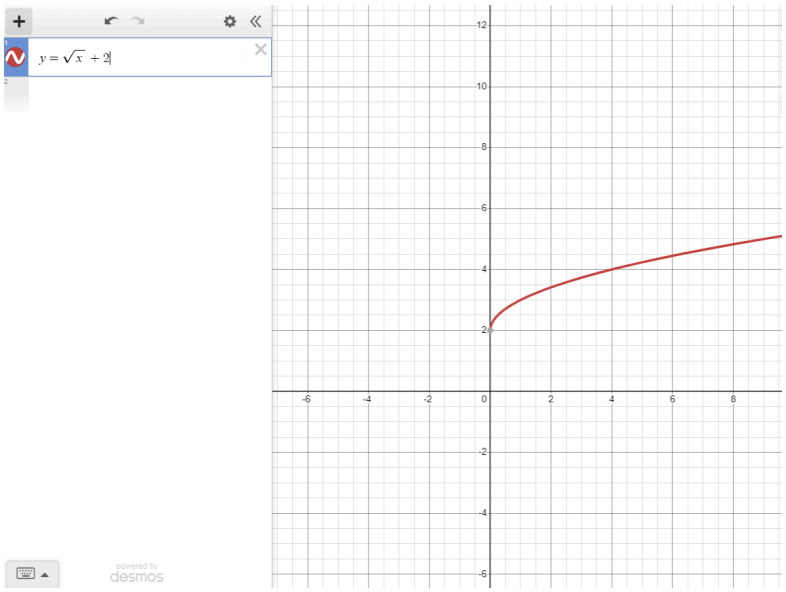
Dengan pangkat tertinggi pada setiap variabelnya adalah 1. Itulah kenapa, *Plot/Graph* dari **Linear Equation (persamaan linier)** selalu menghasilkan **garis lurus** seperti pada gambar diatas.

Equation(persamaan) seperti $y = x^3 + 1$ bukan termasuk **Linear Equation**. Karena salah satu variabelnya memiliki pangkat lebih dari satu, dan hasil dari *Plot/Graph*-nya tidak menghasilkan garis lurus. Contoh lain $y = \sqrt{x} + 2$ juga bukan **Linear Equation**. Karena $\sqrt{x}$ sama saja dengan memangkatkan x dengan $\frac{1}{2}$.

Plot $y = x^3 + 1$



Plot $y = \sqrt{x} + 2$



Bisa kita lihat pada dua contoh diatas, **Non Linear Equation** tidak pernah menghasilkan garis lurus.

Memecahkan Linear Equation

Contoh 1:
$5x + 10 = 30$

Untuk memecahkan **Linear Equation** di atas, kita akan isolasi variabel x tetap berada disisi kiri.

$5x = 30 - 10$
$5x = 20$
$x = \frac{20}{5}$
$x = 4$

Contoh 2 (**Linear Equation** dengan 2 variabel):
$8x + 4y + 24 = 0$
$8x + 4y = -24$
$4y = -24 - 8x$
$y = \frac{-24-8x}{4}$
$y = -2x - 6$

Memecahkan Linear System Equation

Contoh 1:
$x - y = 4$
$2x + y = 2$

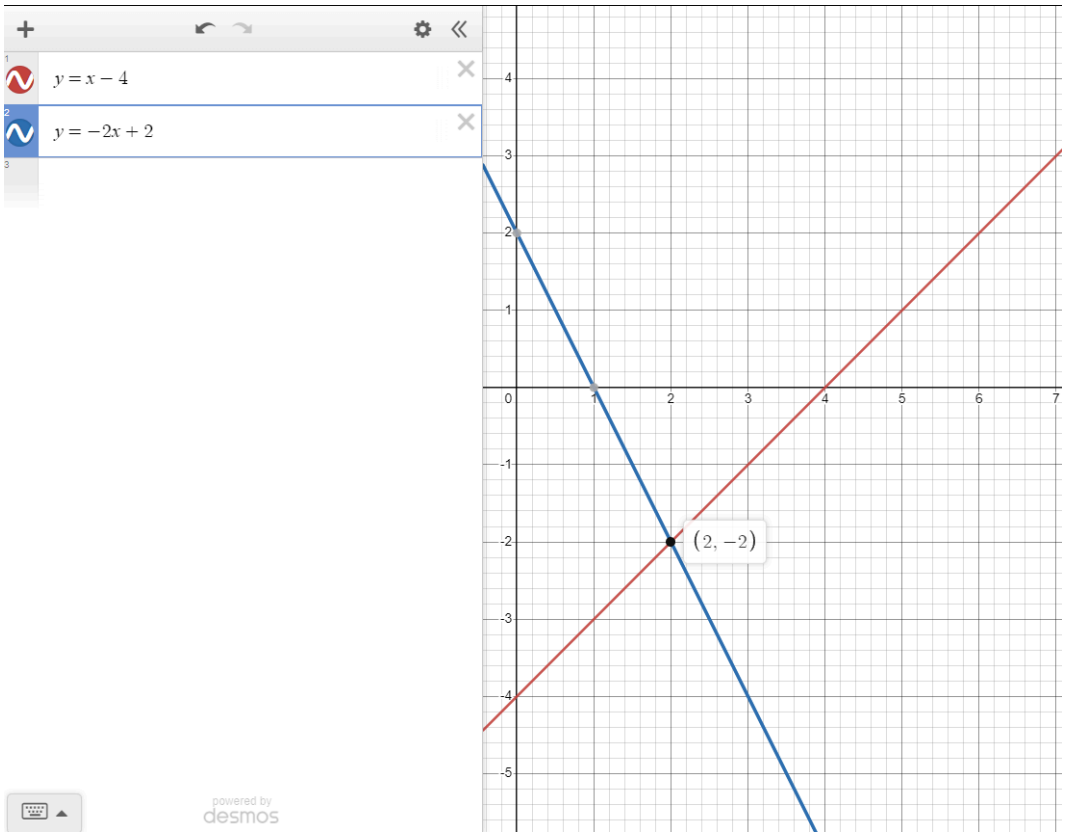
Memecahkan dengan cara *Plot/Graphing*. Pertama ubah terlebih dahulu ke dalam bentuk $y = wx + b$.

Persamaan pertama
$x - y = 4$
$-y = -x + 4$
$y = x - 4$

Persamaan kedua
$2x + y = 2$

$y =- 2x + 2$

Plot $y = x - 4$ dan $y =- 2x + 2$



Perhatikan garis yang saling memotong(*intersect*) dari dua persamaan diatas. Garis yang memotong ada pada koordinat $(2, \ -2)$. Sehingga solusi dari **Linear System Equation** diatas adalah $(2, \ -2)$, $x = 2$ dan $y =- 2$.

Memecahkan dengan cara Substitusi. Pertama isolasi variabel x pada persamaan pertama.

$x - y = 4$

$x = 4 + y$

Substitusikan hasil isolasi x diatas ke persamaan kedua.

$2x + y = 2$

$2(4 + y) + y = 2$

$8 + 2y + y = 2$

$8 + 3y = 2$

$3y = 2 - 8$

$3y = -6$

$y = \frac{-6}{3}$

Sehingga $y = -2$

Untuk mencari x, substitusikan $y = -2$ ke persamaan mana saja. Untuk contoh ini, kita substitusikan $y = -2$ ke persamaan $x - y = 4$.

$x - (-2) = 4$

$x + 2 = 4$

$x = 4 - 2$

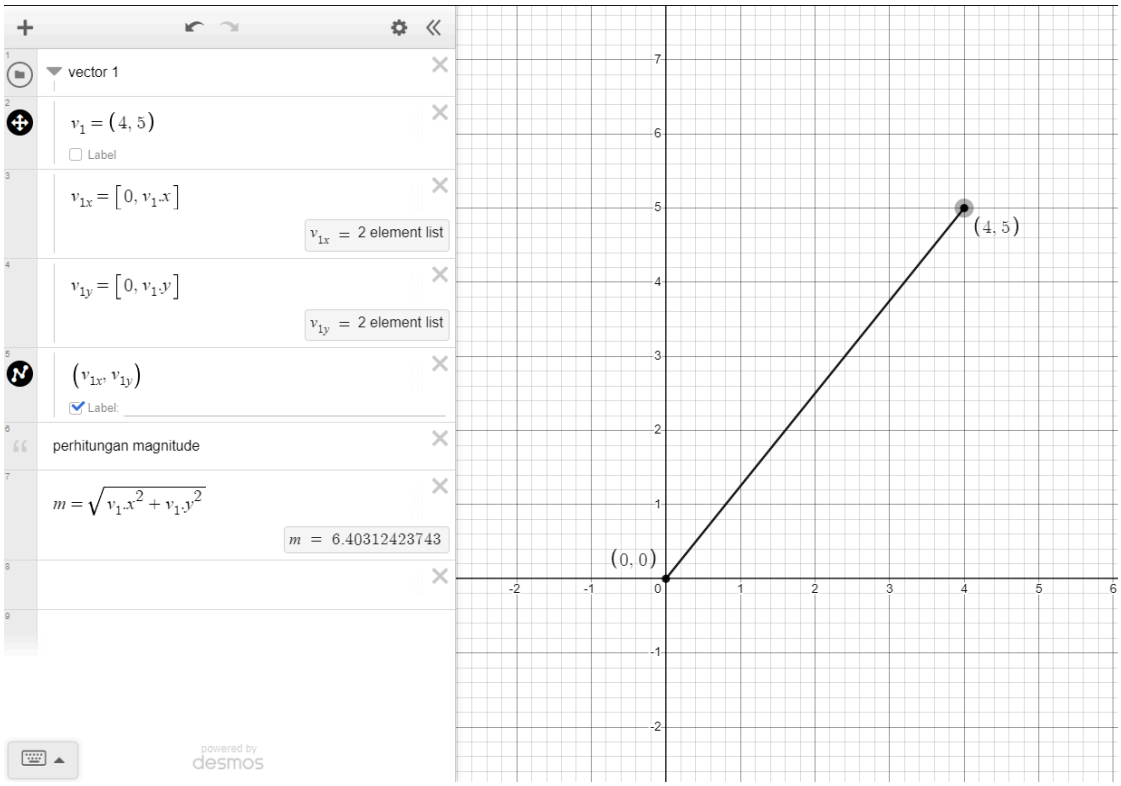
Sehingga $x = 2$

Sehingga solusi dari **Linear System Equation** diatas adalah $(2, -2)$, $x = 2$ dan $y = -2$. Hasilnya sama dengan cara *Plot/Graphing* yang dilakukan sebelumnya.

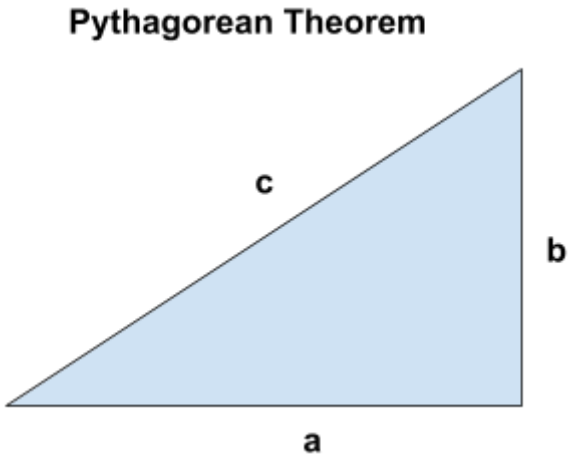
Vector

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

Pengertian *vector* adalah deretan nomor yang menyimpan sebuah informasi. Misalnya kita butuh menyimpan informasi warna hijau pada sebuah *vector*. $[50,\ 168,\ 82]$, list atau kumpulan nomor tersebut bisa kita gunakan untuk merepresentasikan warna hijau. Contoh lain, kita butuh menyimpan informasi spesifikasi sebuah rumah. Rumah tersebut mempunyai harga $Rp.\ 400.000.000$ dan luas $123\ m^2$. Maka informasi tersebut juga bisa kita simpan pada sebuah *vector* $[400.000.000,\ 123]$. Umumnya, *vector* biasa direpresentasikan dengan *Array* atau *List* pada bahasa pemrograman. *Vector* bisa kita gambarkan pada sebuah *coordinate system* seperti dibawah ini. *Vector* mempunyai *magnitude* (panjang) dan *direction*(arah) seperti pada gambar dibawah ini.



Vector diatas mempunyai koordinat $(4,\ 5)$. Dengan $x = 4$ dan $y = 5$. Sehingga bisa ditarik garis lurus dari koordinat *origin*, yaitu $(0, 0)$ ke koordinat $(4,\ 5)$. Garis yang menghubungkan koordinat *origin* dan $(4,\ 5)$ disebut dengan *magnitude*. Untuk menghitung *magnitude*, kita bisa menggunakan **Pythagorean Theorem**.

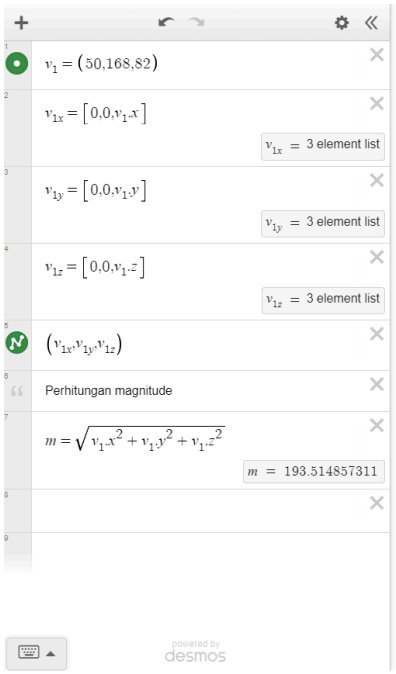


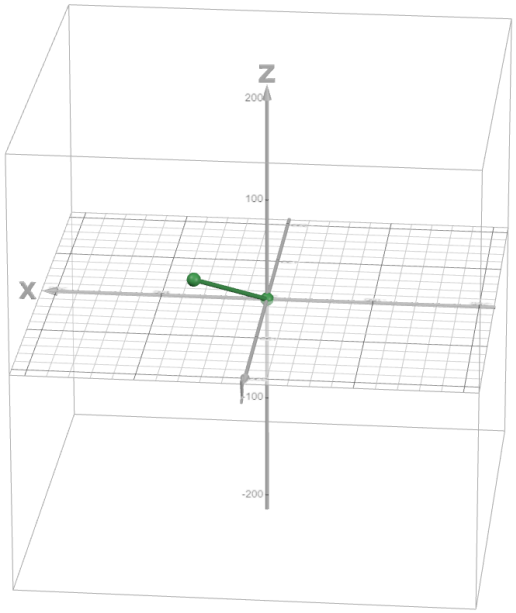
$$c^2 = a^2 + b^2$$
$$c = \sqrt{a^2 + b^2}$$

Dimana:
$a = x$
$b = y$

Sehingga
$$m^2 = x^2 + y^2$$
$$m = \sqrt{x^2 + y^2}$$
$$m = \sqrt{4^2 + 5^2}$$
$$m = 6.4$$

Vector pada sistem koordinat 3 dimensi





Vector pada sistem koordinat 3 dimensi mirip dengan pada koordinat 2 dimensi. Bedanya pada koordinat 3 dimensi, *vector* mempunyai tambahan informasi yaitu titik pada sumbu z. Pada contoh *vector* 3 dimensi di atas, *vector* tersebut mempunyai koordinat $[50, 168, 82]$. Untuk menghitung *magnitude* pada *vector* 3 dimensi, kita juga bisa menggunakan **Pythagorean Theorem**.

Menghitung *magnitude* vector 3 dimensi:

$$d^2 = a^2 + b^2 + c^2$$

Atau dengan bentuk umum untuk $n\ dimensions$ vector:

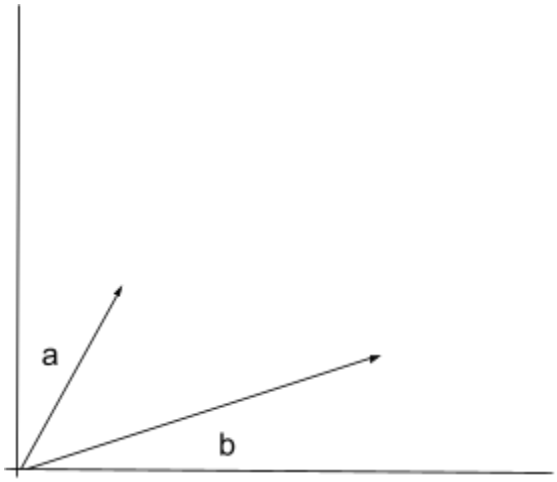
$$|m| = \sqrt{\sum_{i=1}^{n} d_i^2}$$

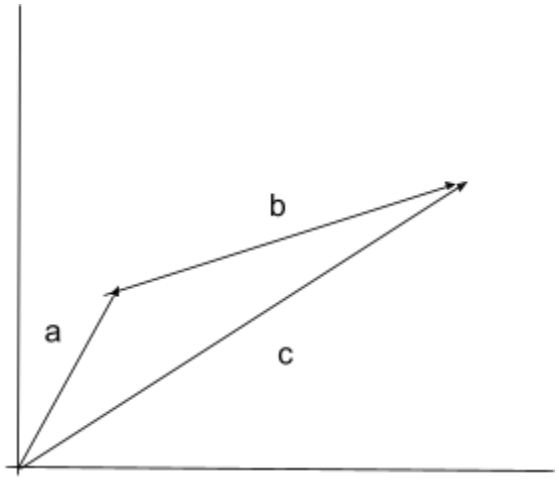
Dimana:

$d = dimensi$

Persamaan di atas digunakan untuk menghitung *magnitude vector* dengan $n\ dimensions$.

Vector Addition (Penjumlahan Vector)

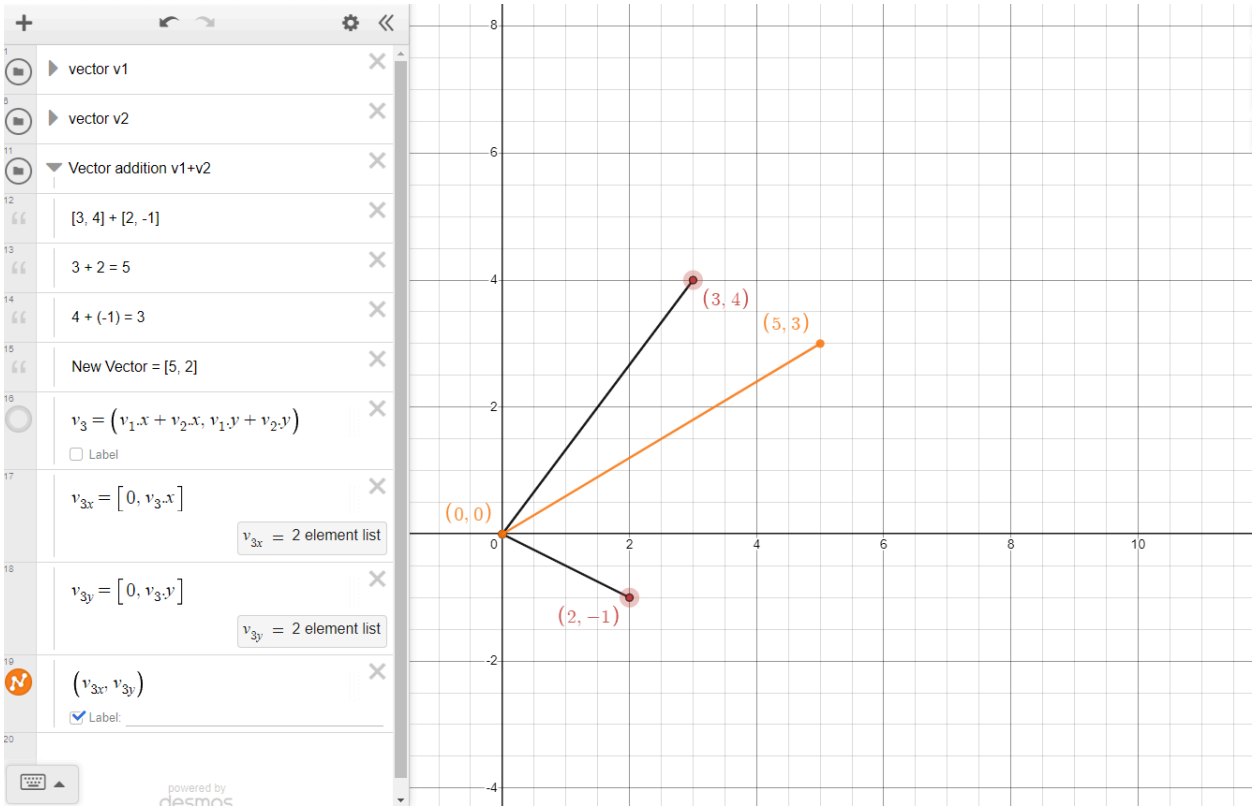




a + b = c

Menambahkan 2 buah *vector* cukup mudah. Dengan catatan 2 *vector* tersebut mempunyai dimensi yang sama. Seperti gambar di atas, *vector* a ditambahkan dengan *vector* b akan menghasilkan *vector* c. Menjumlahkan satu *vector* dengan *vector* lainnya akan merubah *magnitude* dan *direction* dari *vector* tersebut.

Contoh:
$a = [3, 4]$
$b = [2, -1]$



Vector c berwarna *orange* adalah hasil penjumlahan *vector* a dan *vector* b.

$$c.x = a.x + b.x$$
$$c.y = a.y + b.y$$

$$c.x = 3 + 2 = 5$$
$$c.y = 4 + (-1) = 3$$

Sehingga menghasilkan *vector* baru, yaitu *vector* $c = (5, 3)$.

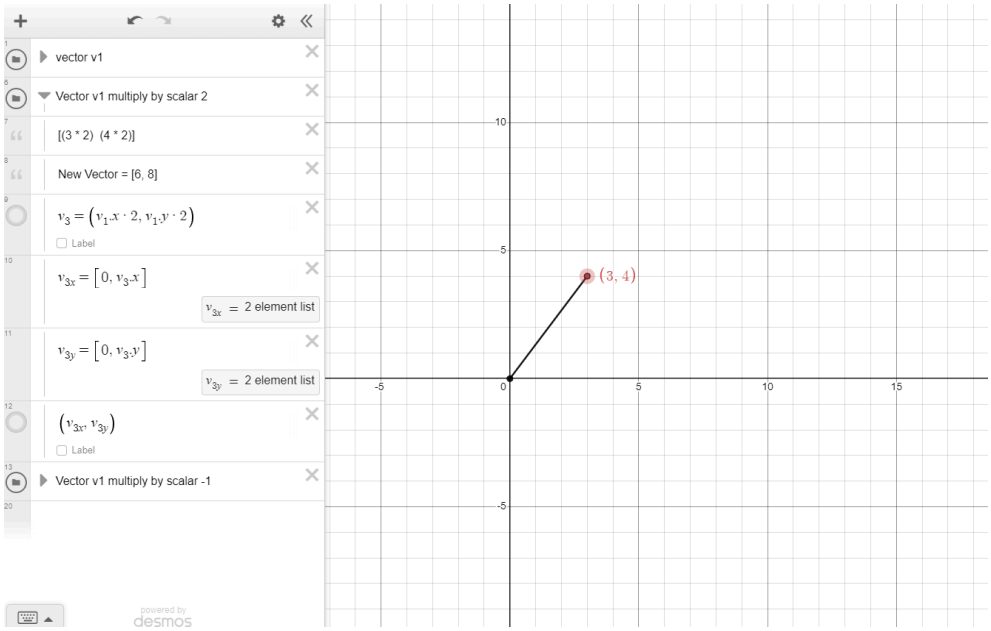
Vector Multiplication (Perkalian Vector) dengan Scalar

Perkalian *vector* dengan *scalar* biasanya bertujuan untuk memperbesar atau memperkecil *magnitude* dan membalik (*flipping*) suatu *vector*.

Contoh mengalikan *vector* dengan positif *scalar* 2:

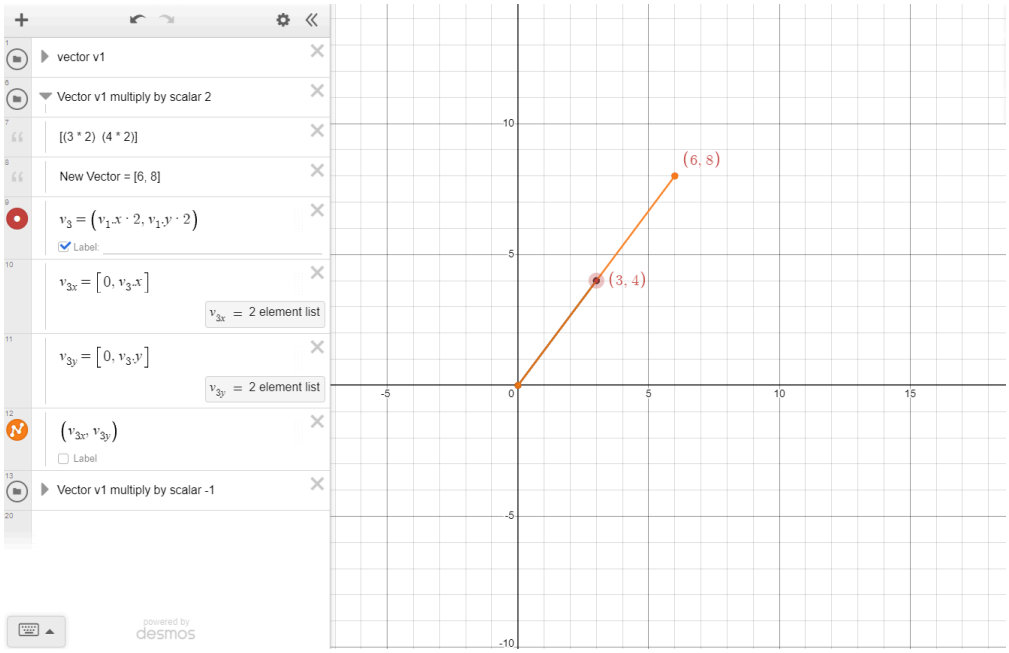
$$a = [3, 4]$$
$$scalar = 2$$

Mengalikan *vector* a dengan *scalar* 2 cukup mudah. Kalikan setiap elemennya dengan *scalar* 2.



$b.x = 3 * 2$
$b.y = 4 * 2$

Sehingga menghasilkan *vector* baru, yaitu *vector* b.



Vector $b = (6, 8)$.

Contoh mengalikan *vector* dengan negative scalar -1 (untuk membalik *direction* sebuah *vector*):

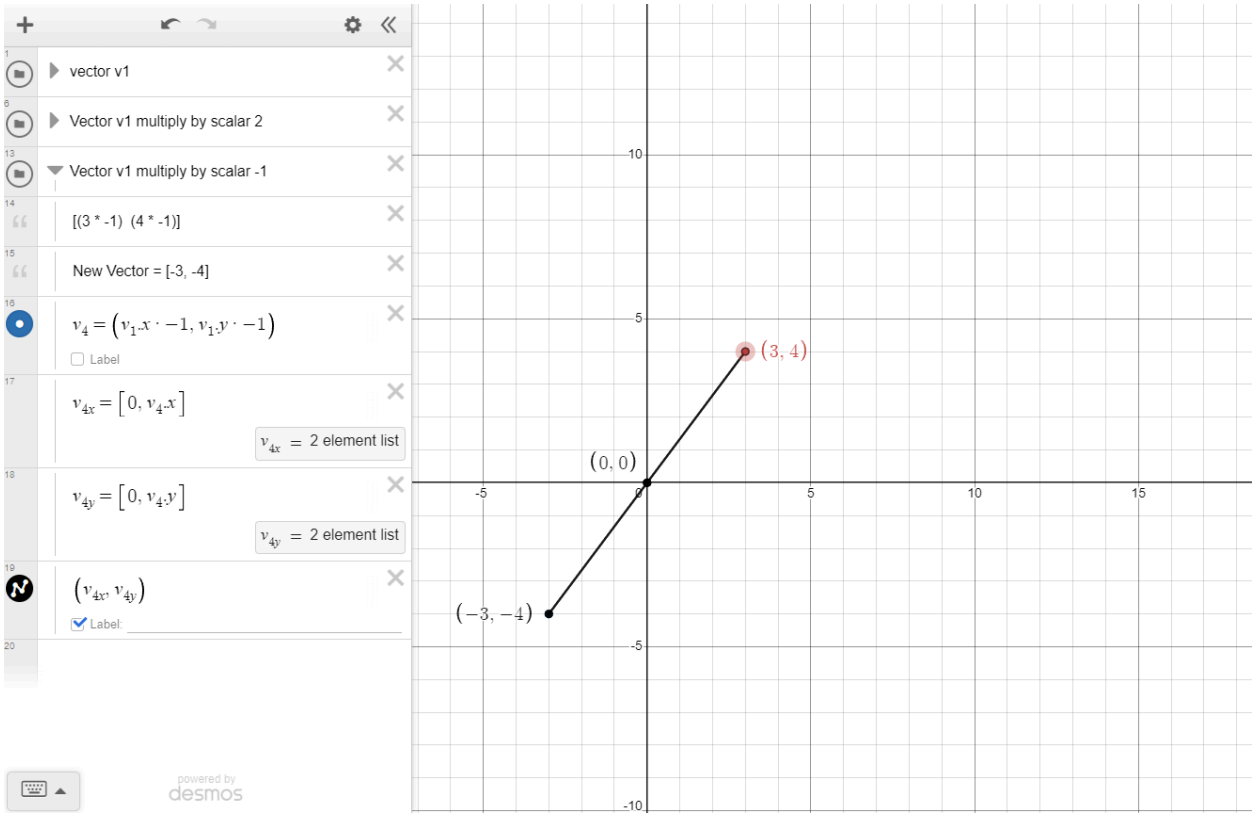
$a = [3, 4]$

$scalar = -1$

$b.x = 3 * (-1)$

$b.y = 4 * (-1)$

Sehingga menghasilkan *vector* baru, yaitu *vector* b.



Vector $b = (-3, -4)$.

Perkalian vector dengan vector (Dot Product)

Perkalian *vector* dengan *vector* (**Dot Product**) menghasilkan sebuah *scalar*, sehingga disebut juga *scalar product*. **Dot Product** digunakan untuk mengukur kemiripan dua *vector*. Untuk menghitung *Dot Product* ada dua cara yang bisa dilakukan.

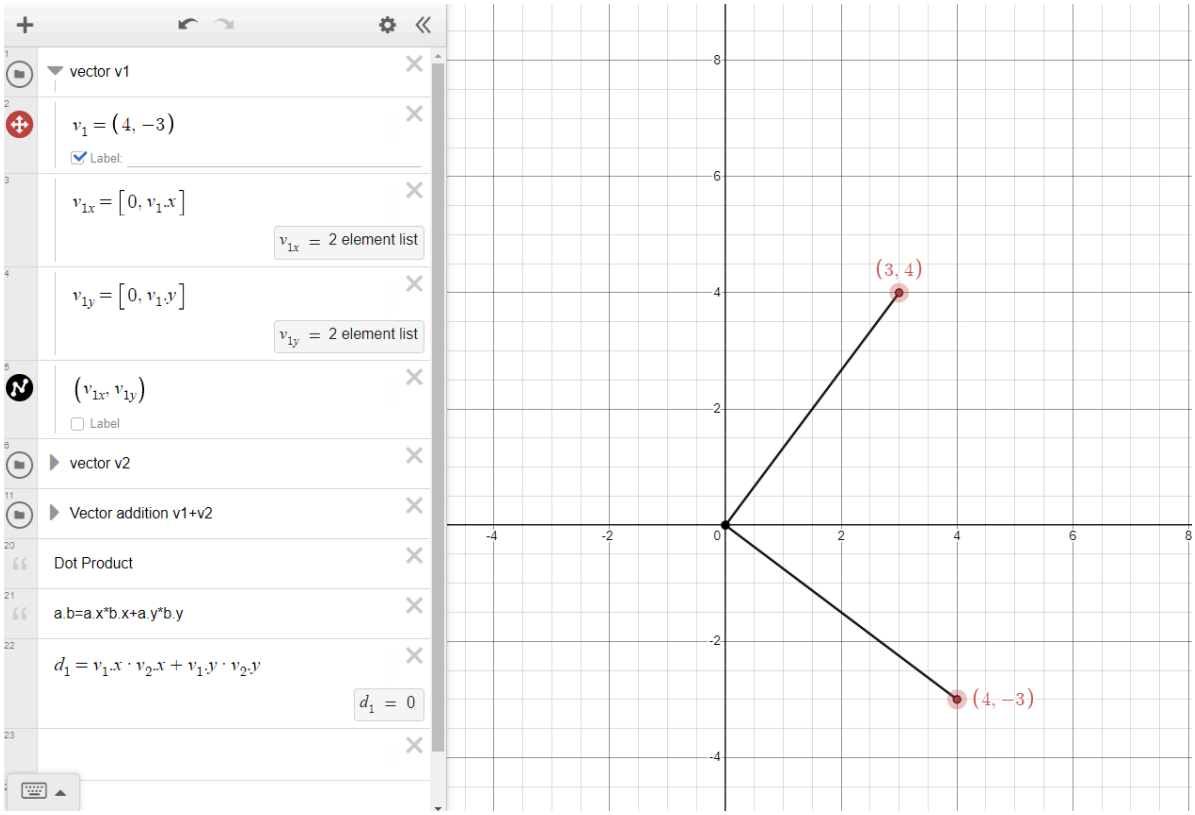
Cara pertama kita bisa mengalikan *magnitude* dua *vector* tersebut dengan nilai $cos(\theta)$, dimana θ adalah sudut antara dua *vector*.
$a \cdot b = |a|\ |b|\ cos(\theta)$

Cara kedua kita bisa menjumlahkan *product* dari setiap elemennya.
$a \cdot b = a.x * b.x + a.y * b.y$

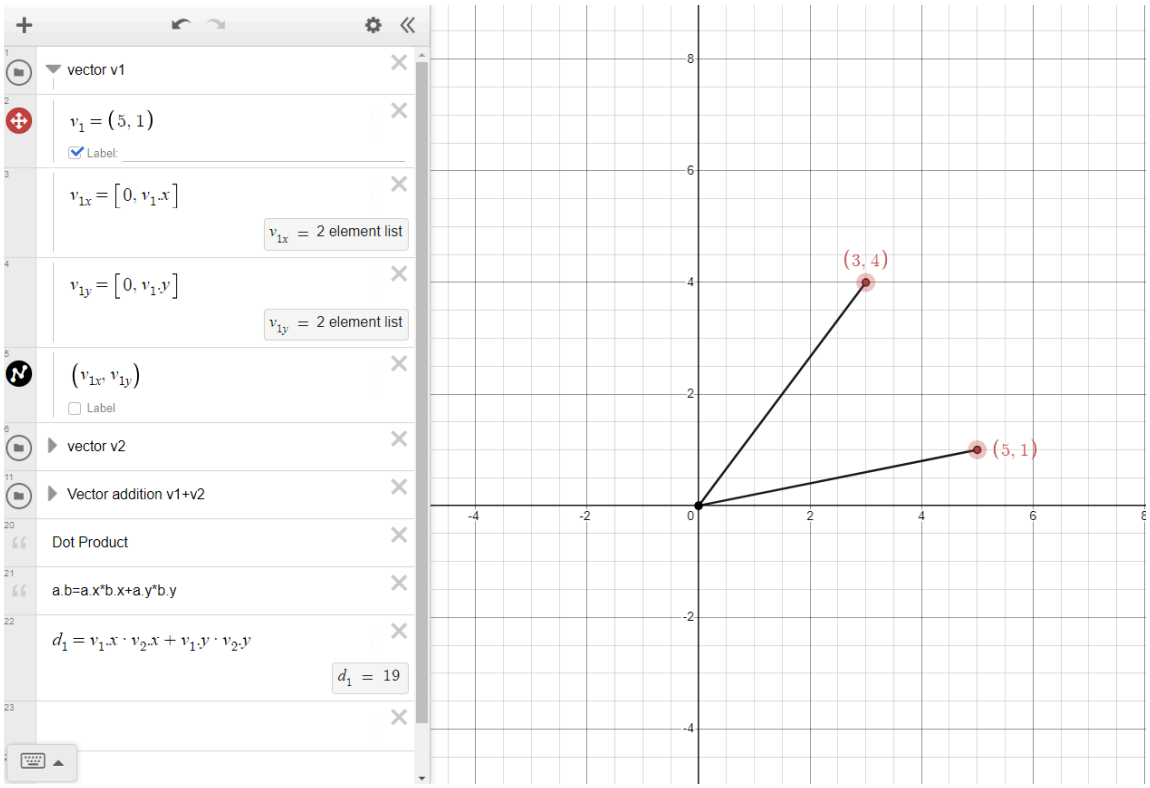
Contoh:
$a = [3, 4]$
$b = [2, -1]$
$a \cdot b = 3 * 2 + 4 * (-1)$
$a \cdot b = 2$

Sifat-sifat **Dot Product**

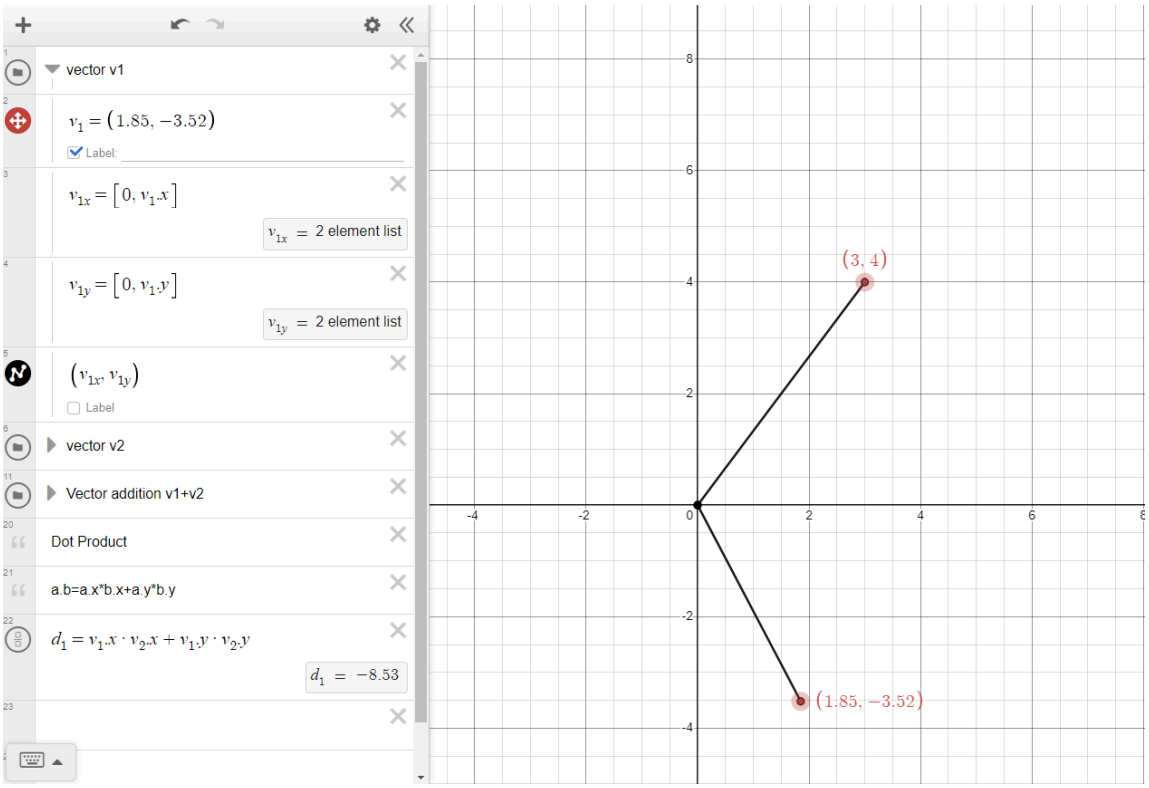
- **Dot Product** bernilai 0 ketika sudut antara 2 *vector* 90°



- **Dot Product** lebih dari > 0 (*positive*) ketika sudut antara 2 *vector* $< 90 \circ$



- **Dot Product** kurang dari < 0 (*negative*) ketika sudut antara 2 *vector* $> 90 \circ$



Perkalian Vector Silang (Cross Product)

Perkalian *vector* silang menghasilkan *vector* baru, tidak seperti **Dot Product**, yang menghasilkan sebuah *scalar*. **Cross Product** hanya bisa dilakukan pada sistem koordinat 3 dimensi.

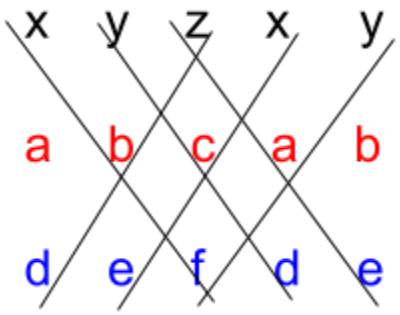
Untuk menghitung **Cross Product** ada dua cara yang bisa dilakukan.

Cara pertama kita bisa mengalikan *magnitude* dari dua vector tersebut dengan nilai $sin(\theta)$, dimana θ adalah sudut antara dua *vector* dan n adalah *unit vector*. ***Unit vector*** **mempunyai** ***magnitude*** 1.

$$a \times b = |a|\,|b|\,sin(\theta)n$$

Cara kedua kita bisa mengalikan silang dari setiap elemennya.

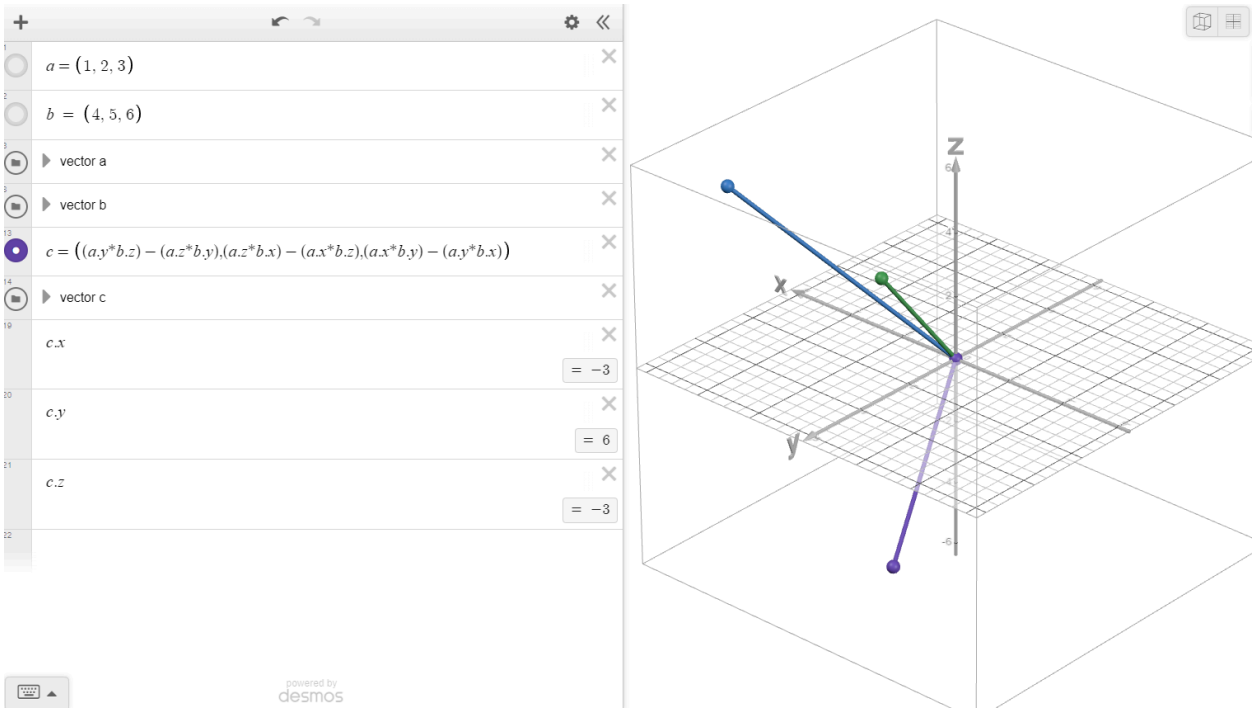
A = [a, b, c] B = [d, e, f]



A x B = [(bf - ce), (cd - af), (ae - bd)]

$a = [1,\ 2,\ 3]$
$b = [4,\ 5,\ 6]$

$a \times b = [(2 * 6) - (3 * 5), (3 * 4) - (1 * 6), (1 * 5) - (2 * 4)]$
$a \times b = [12 - 15, 12 - 6, 5 - 8]$
$a \times b = [-3, 6, -3]$



Euclidean Distance

Mengukur kemiripan dari 2 *vector* atau koordinat ada beberapa formula dan algoritma yang bisa kita gunakan. Salah satunya adalah **Euclidean Distance**. Formula ini memanfaatkan **Pythagorean Theorem**, sehingga **Euclidean Distance** mempunyai nama lain **Pythagorean Distance**.

Formula **Euclidean Distance**:

$$d(p,\ q) = \sqrt{\sum_{i=1}^{n} (p.x_i - q.x_i)^2 + (p.y_i - q.y_i)^2}$$

Melihat formula diatas, **Euclidean Distance** tidak ada batasan dimensi dari sisi data. Formula ini bisa diterapkan pada berbagai dimensi. 3 dimensi, 4 dimensi dan seterusnya.

Contoh:
Katakanlah kita mempunyai 2 koordinat pada *dataset* yang kita miliki.
$b = (12,\ 11)$
$c = (10,\ 0.5)$

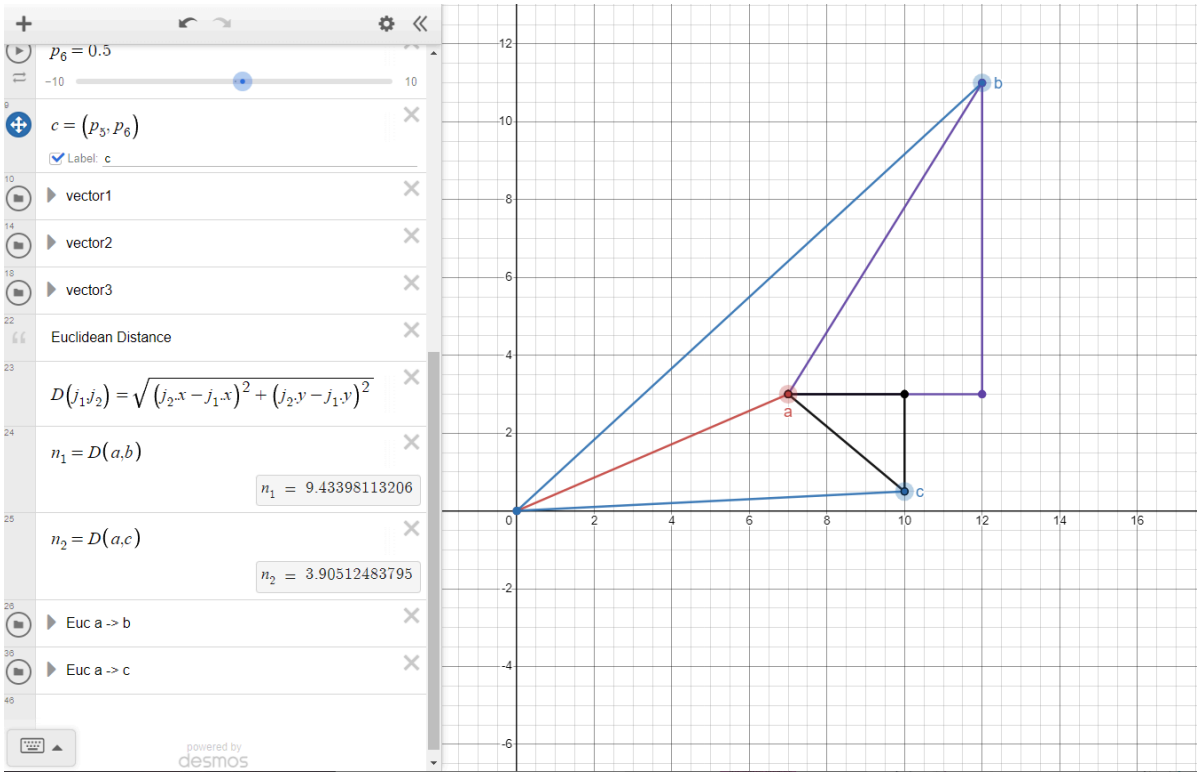
Kemudian kita mempunyai data koordinat baru, yaitu $a = (7, 3)$. Kita akan ukur seberapa dekat data-data yang ada di *dataset* dengan data baru a menggunakan **Euclidean Distance**.

Jarak dari a ke b.

$$dab = \sqrt{(12 - 7)^2 + (11 - 3)^2}$$
$$dab = 9.43$$

Jarak dari a ke c

$$dac = \sqrt{(10 - 7)^2 + (0.5 - 3)^2}$$
$$dac = 3.90$$



Proses sebelumnya coba divisualisasikan pada koordinat sistem. Sehingga bisa disimpulkan, jarak a ke c lebih dekat dari jarak a ke b.

Cosine Distance

Pada bagian sebelumnya kita sudah menggunakan **Euclidean Distance** untuk mengukur seberapa mirip/dekat dari dua buah *vector* maupun koordinat. **Euclidean Distance** menggunakan perbandingan jarak dari 2 koordinat *vector*, sedangkan **Cosine Distance** menggunakan *angle*/sudut untuk membandingkan seberapa mirip 2 buah *vector*.

Formula **Cosine Distance**:

$$cos(\theta) = \frac{A \cdot B}{||A||\,||B||}$$

$$cos(\theta) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Dimana
$A \cdot B = Dot\ Product\ dari\ vector\ A\ dan\ B$
$||A||\,||B|| = Magnitude\ dari\ vector\ A\ dan\ B$

Contoh:
Katakanlah kita mempunyai 2 koordinat pada dataset yang kita miliki.
$b = (12,\ 11)$
$c = (10,\ 0.5)$

Kemudian kita mempunyai data koordinat baru, yaitu $a = (8,\ 2)$. Kita akan ukur seberapa dekat *angle*/sudut dari data-data yang ada di *dataset* dengan data baru a menggunakan **Cosine Distance**.

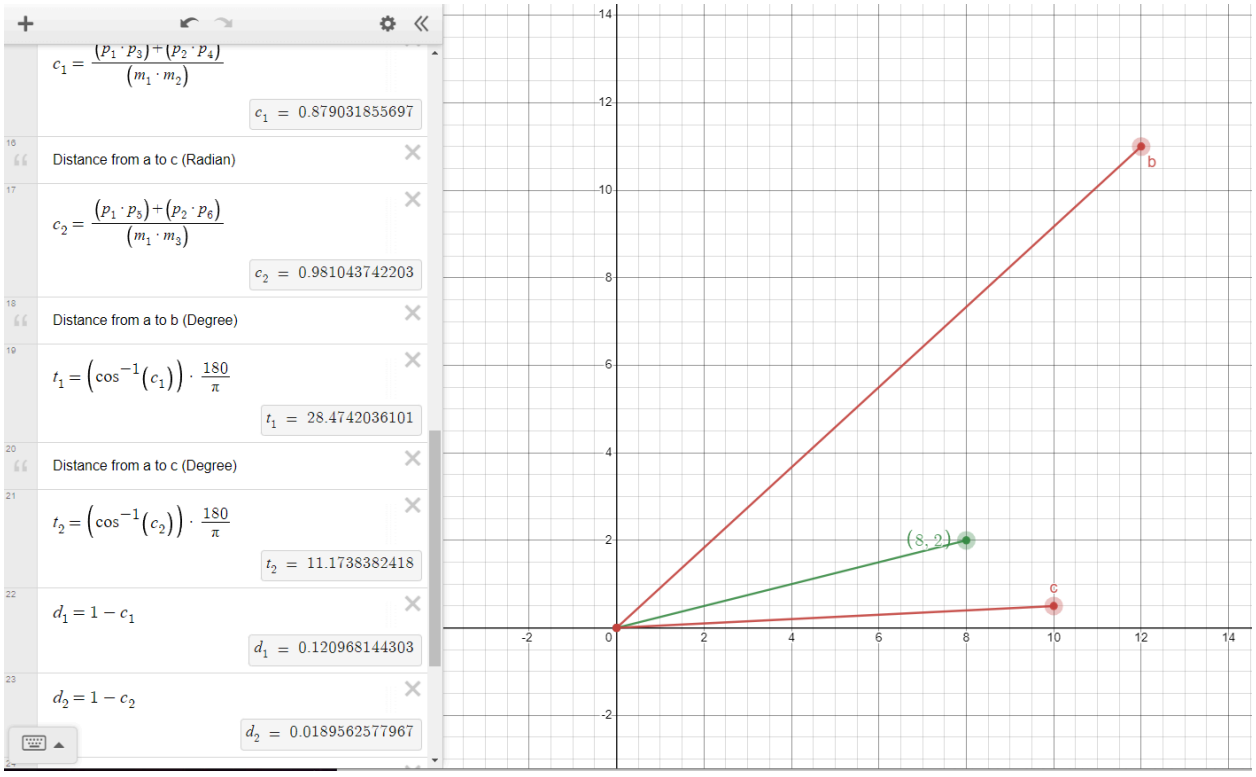
Sudut a ke b.
$$Aab = \frac{(8\times12)+(2\times11)}{\left(\sqrt{8^2+2^2}\right)\left(\sqrt{12^2+11^2}\right)}$$
$Aab = 0.879\ radian$
$cosine\ distance\ a \rightarrow b = 1 - 0.879 = 0.121$

Sudut a ke c.

$$Aac = \frac{(8\times10)+(2\times0.5)}{\left(\sqrt{8^2+2^2}\right)\left(\sqrt{10^2+0.5^2}\right)}$$

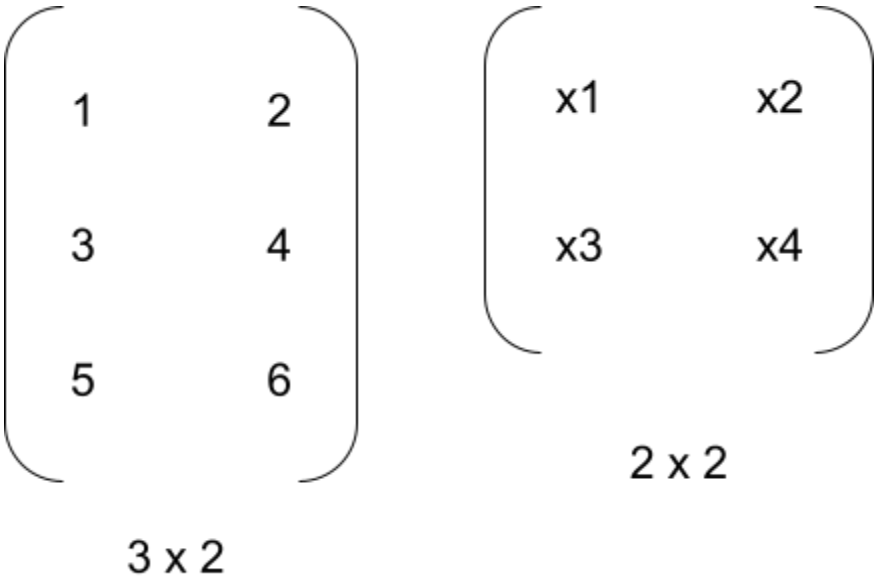
$$Aac = 0.981\, radian$$

$$cosine\ distance\ a \rightarrow c = 1 - 0.981 = 0.019$$



Proses sebelumnya coba divisualisasikan pada koordinat sistem seperti pada gambar diatas. Sehingga bisa disimpulkan, **Cosine Distance** a ke c lebih kecil dari **Cosine Distance** a ke b.

Matrix

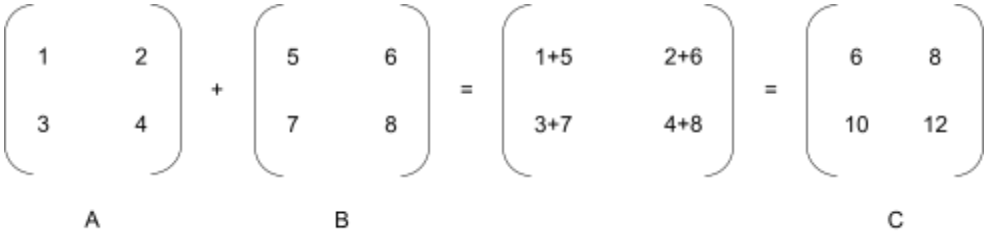


Matrix adalah sekumpulan bilangan, simbol ataupun *expressions* yang tersusun pada baris dan kolom seperti gambar diatas. Dalam sebuah bahasa pemrograman, *matrix* bisa kita representasikan dengan *Array* multi dimensi. Sama seperti *vector*, *matrix* dalam konteks **AI/ML/Deep Learning** juga kita gunakan untuk merepresentasikan atau mengabstraksi sebuah data.

Untuk mengidentifikasi sebuah *matrix*, kita mulai dari baris kemudian kolom. Seperti pada gambar diatas, *matrix* pertama mempunyai 3 baris dan 2 kolom (3x2 *matrix*). Sedangkan pada *matrix* kedua mempunyai 2 baris dan 2 kolom (2x2 *matrix*).

Matrix Addition (Penjumlahan Matrix)

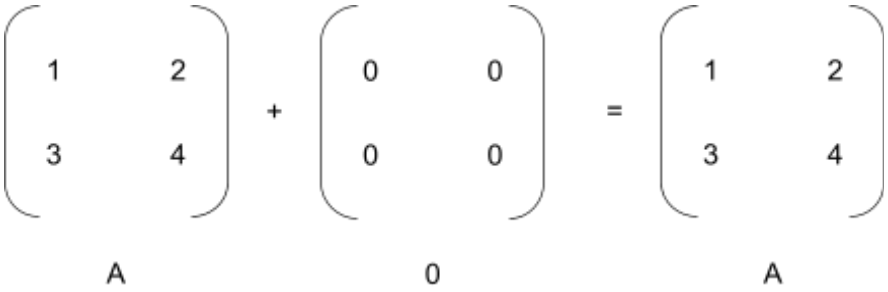
Penjumlahan *Matrix* adalah operasi aritmatika penjumlahan pada 2 buah *matrix*. Penjumlahan *Matrix* mempunyai satu rule utama yang harus dipenuhi. Yaitu harus mempunyai dimensi yang sama. Contoh *matrix* 2x2 bisa dijumlahkan dengan *matrix* 2x2. Tetapi matrix 2x2 bisa dijumlahkan dengan *matrix* 4x4.



Pada matrix diatas dimensi antara *matrix* A dan *matrix* B sama, yaitu sama-sama mempunyai dimensi 2x2.

Beberapa Properti dari Penjumlahan *Matrix*:

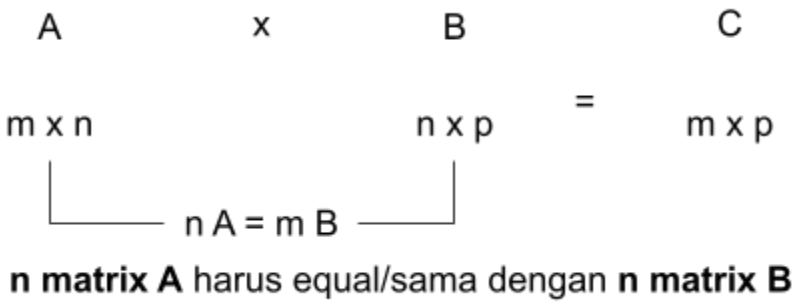
- *Commutative*: A + B = B + A
- *Associative*: A + (B + C) = (A + B) + C
- *Additive Identity*: A + 0 = A



- Additive Inverse A + (-A) = 0

Matrix Multiplication (Perkalian Matrix)

Penjumlahan *Matrix* adalah operasi aritmatika perkalian pada 2 buah *matrix*. Sama seperti penjumlahan *matrix*, perkalian *matrix* juga mempunyai satu *rule* utama yang harus dipenuhi.



n matrix A harus equal/sama dengan **n matrix B**

Setelah proses perkalian, **m matrix A** dan **n matrix B** akan menjadi dimensi baru **m x p** pada matrix hasil perkalian

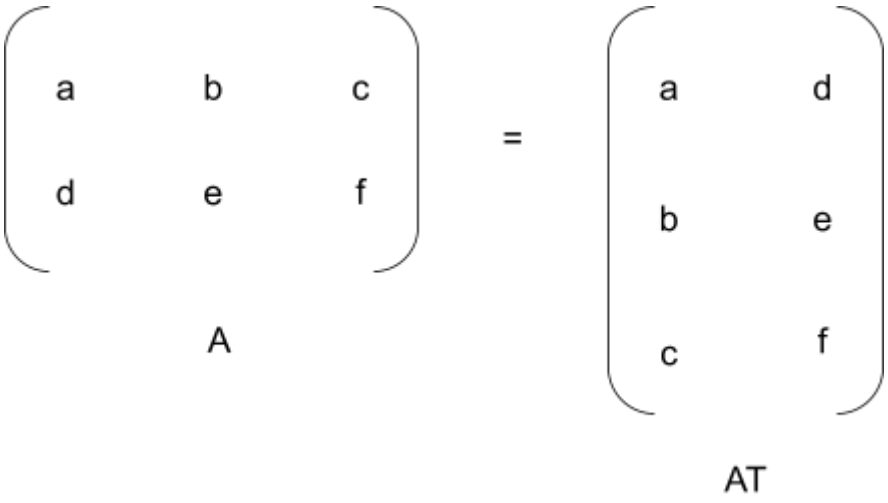
Contoh:

Beberapa property dari Perkalian *Matrix*:

- *Commutative*: A x B ≠ B x A
- *Associative*: (AB)C = A(BC)
- *Distributive*: A(B + C) = AB + AC
  (B + C)A = BA + CA
- *Multiplicative*: A x 0 = 0

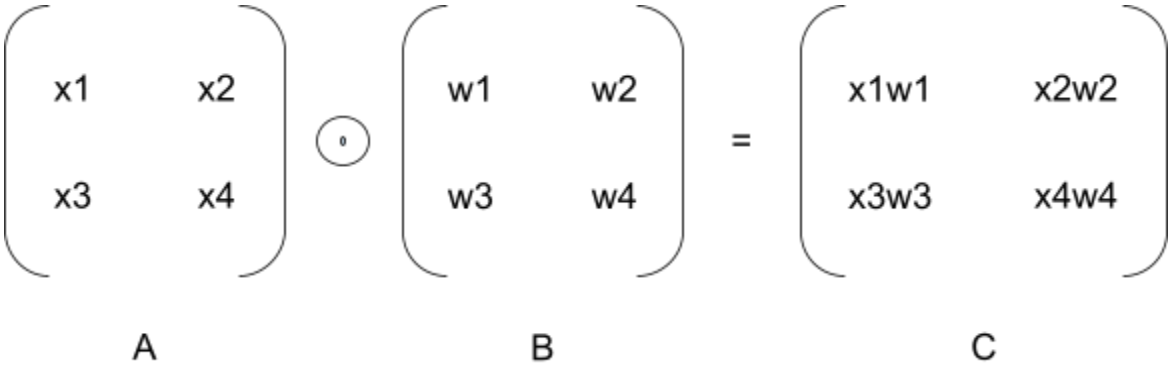
Transpose Matrix

Operasi *transpose* adalah operasi *matrix* yang merubah baris menjadi kolom.



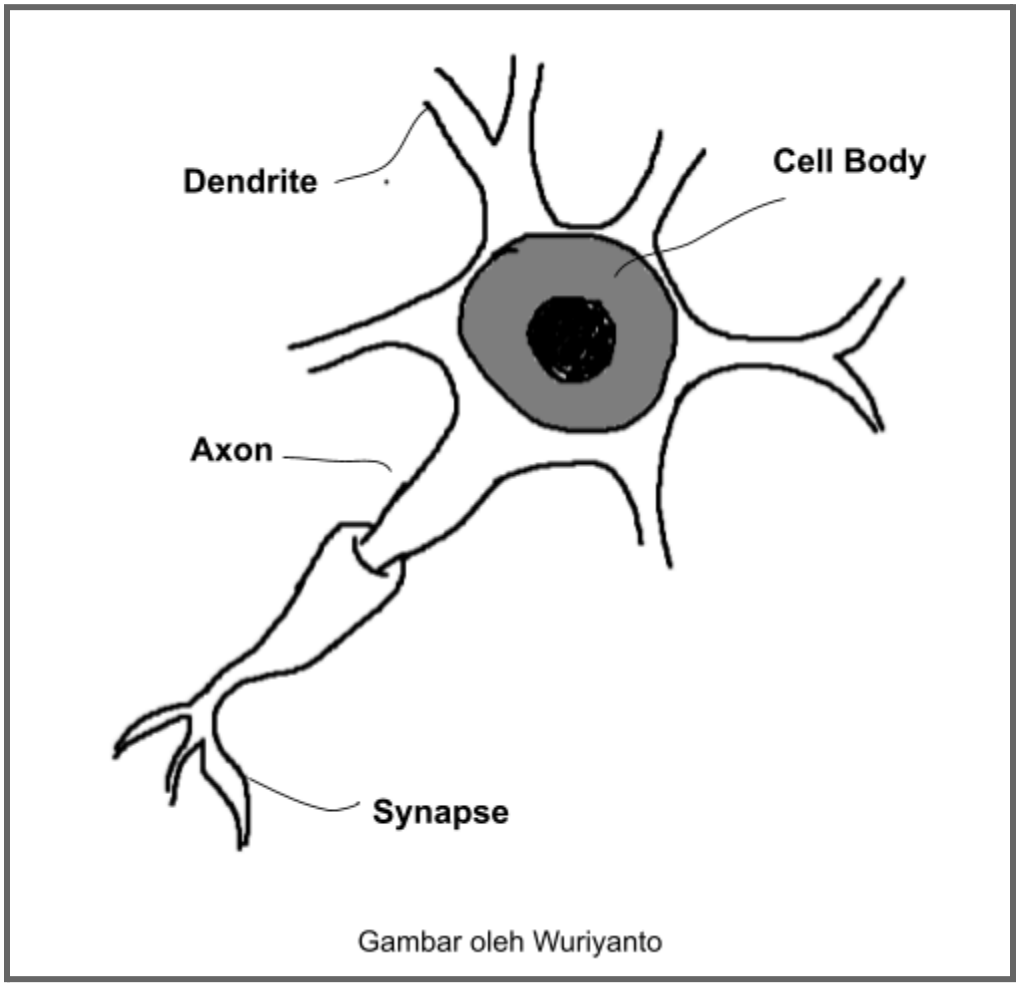
Hadamard Product of Matrix (Element Wise Multiplication)

Hadamard Product adalah operasi perkalian pada *matrix* dengan dimensi yang sama. Operasi **Hadamard Product** mirip dengan penjumlahan *matrix*, hanya saja kali ini operasi aritmatika yang digunakan adalah perkalian.



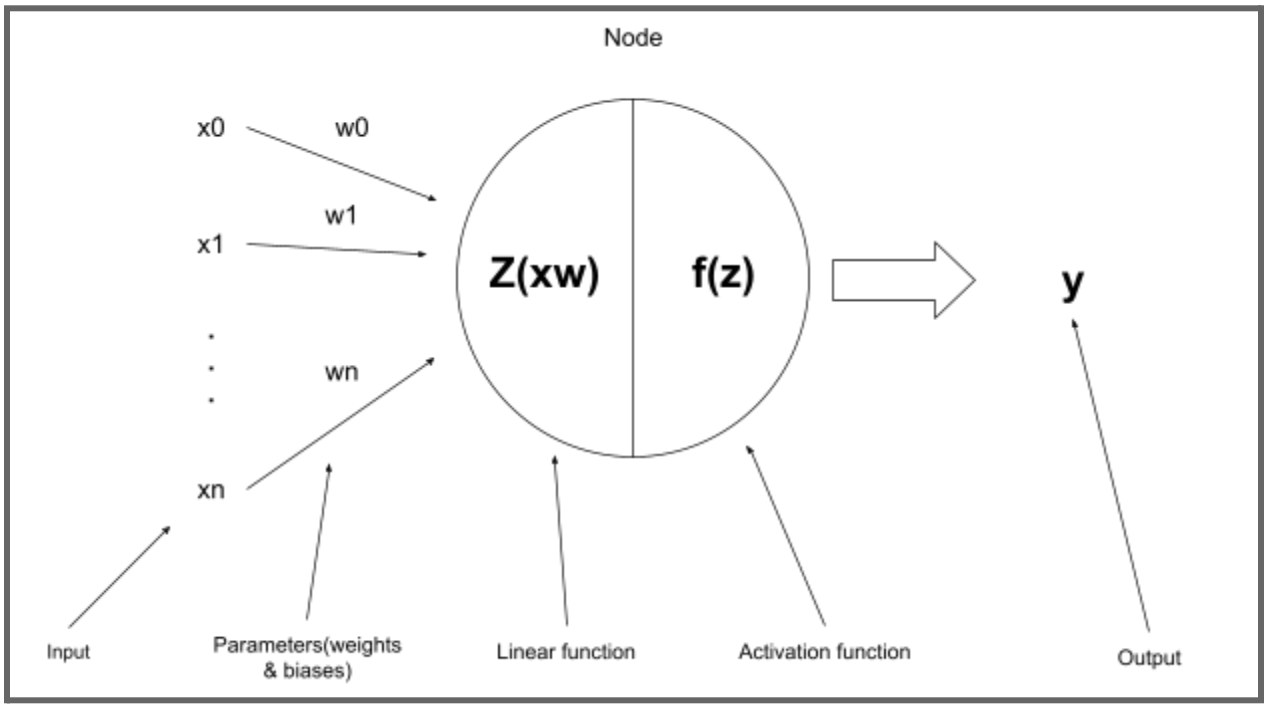
Artificial Neural Network (Jaringan Saraf Tiruan)

Artificial Neural Network (ANN) adalah salah satu pemodelan **Machine Learning** yang terinspirasi dari struktur dan cara kerja otak manusia. Setiap jaringan saraf (*Neuron*) pada manusia terdiri dari beberapa komponen utama seperti *Dendrite*, *Cell Body* dan *Axon*. *Dendrite* menerima *input signal*(setiap *input signal* yang masuk akan dipengaruhi oleh *Synapses/weights*. Semakin kuat *Synapses/wights*, maka semakin kuat koneksi antar *neuron*-nya), *Cell Body* bertugas melakukan komputasi, dan *Axon* bertugas mengirimkan *response* (*output*).



Gambar oleh Wuriyanto

Dalam **Artificial Neural Network**, satu *neuron* adalah satuan fungsi yang ada di dalam sebuah *node* yang menerima input eksternal atau *input* dari *neuron* lain, kemudian *output* dari *neuron* tersebut akan diaktifkan melalui *activation function*. *Neuron* yang aktif akan mempunyai *value* yang besar, sedangkan *neuron* yang tidak aktif akan mempunyai nilai yang kecil.



Perbandingan istilah pada **Biologi** dan **Artificial Neural Network**:

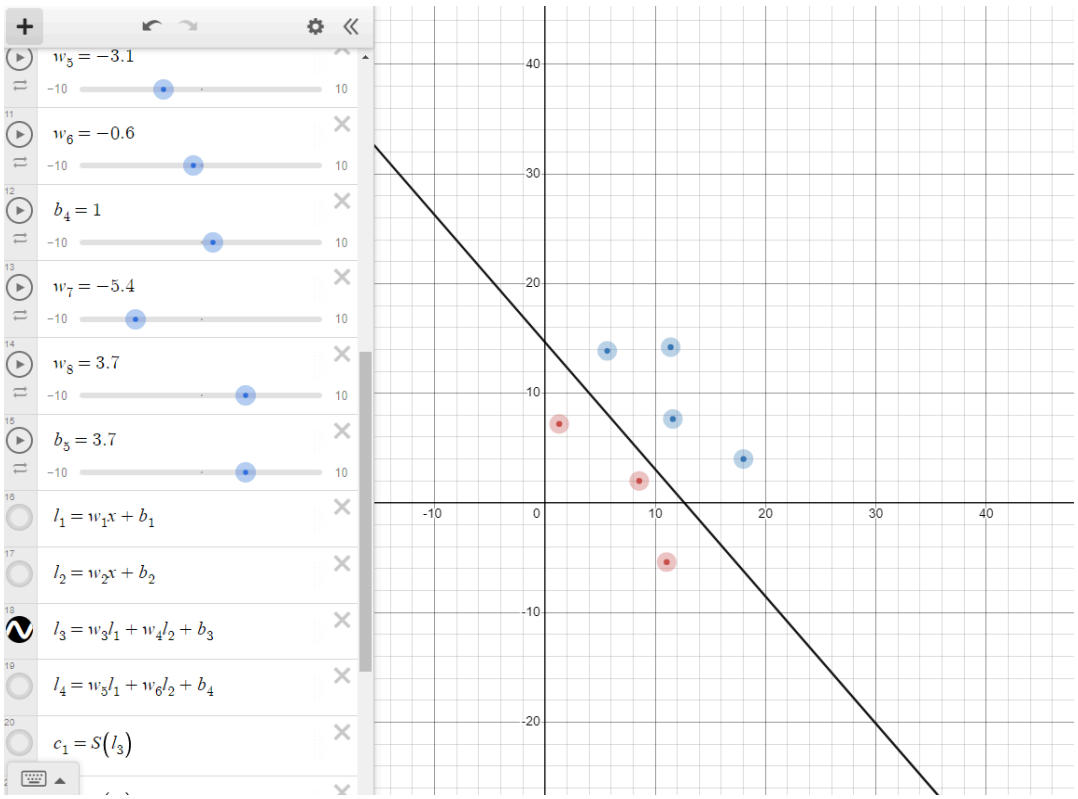
Biologi	**ANN**
Dendrite	*Input*
Cell Body	*Node*
Synapse	*Parameters(weights & biases)*
Axon	*Output*

Linear Function dan Classification Problem

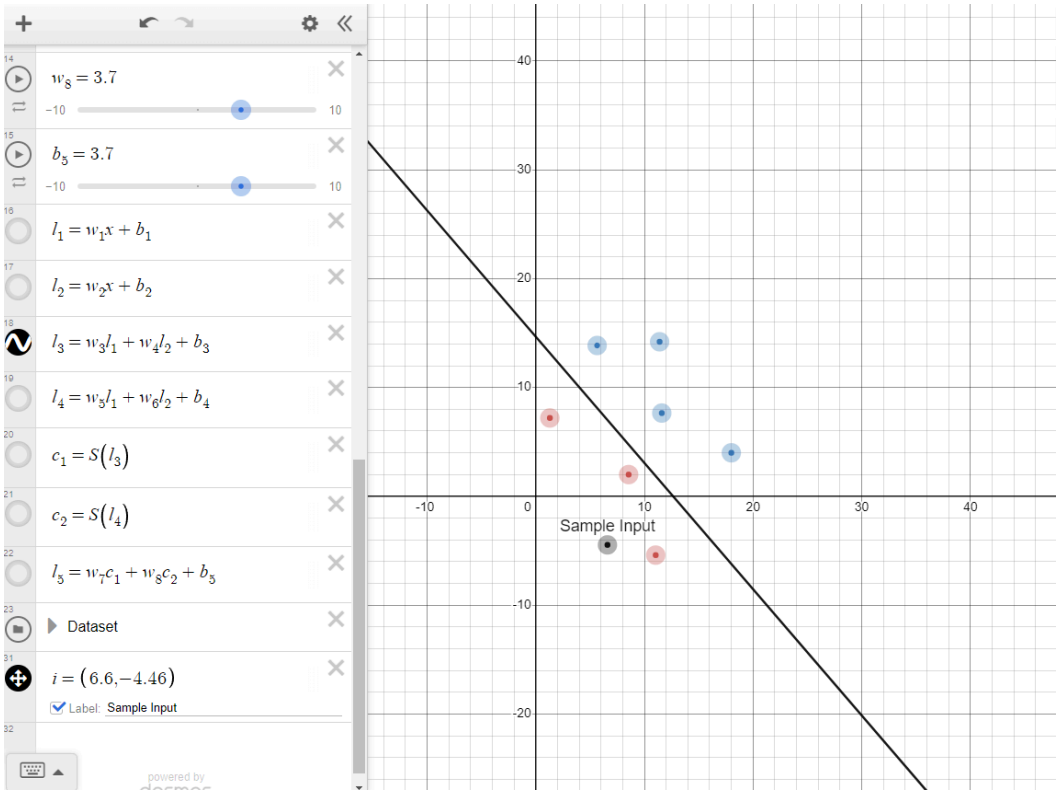
Pada bagian pembahasan **Linear Algebra**, kita tahu bahwa **Linear function** $y = wx + b$ akan menghasilkan garis lurus. Jadi dengan parameter (*weight/slope* dan *bias/intercept)* yang sesuai, kita bisa memanfaatkan **Linear Function** untuk menyelesaikan problem klasifikasi data.

Data *classification* dengan **Linear Function**

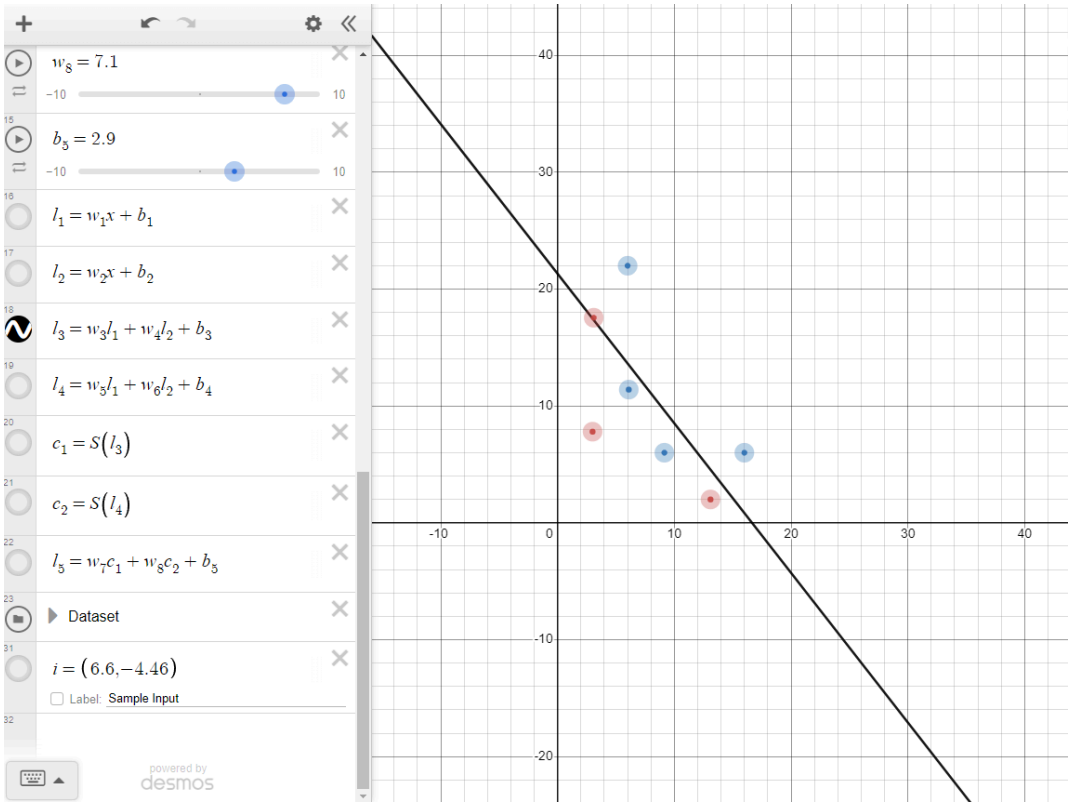
$y = wx + b$



Pada *graph* diatas, kita bisa dengan mudah membuat garis lurus (*decision boundary*) untuk mengklasifikasi data merah dan data biru. Sehingga pada saat ada data baru yang kita gunakan sebagai *input* seperti gambar dibawah ini.



Kita dengan mudah bisa langsung mengklasifikasikan data "*Sample Input*" masuk ke dalam data kategori data merah.

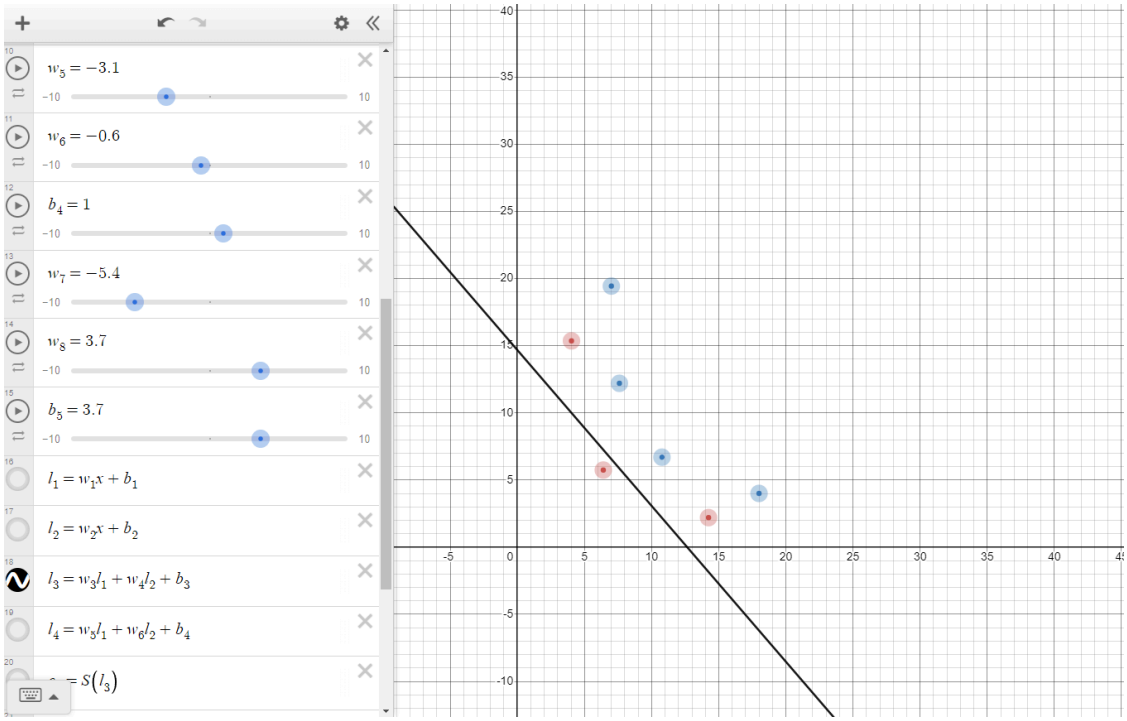


Masalah timbul pada saat data yang ada tidak *Linear* seperti pada gambar diatas. Sebanyak apapun kita menambah parameter dan *Linear function*-nya, garis yang terbentuk tetaplah garis lurus. Sebab, kita hanya menambahkan *Linear function* satu dengan *Linear function* lainnya. Bisa kita simpulkan, *Linear function* tidak bisa membantu kita menyelesaikan masalah data *Non linear* seperti gambar diatas.

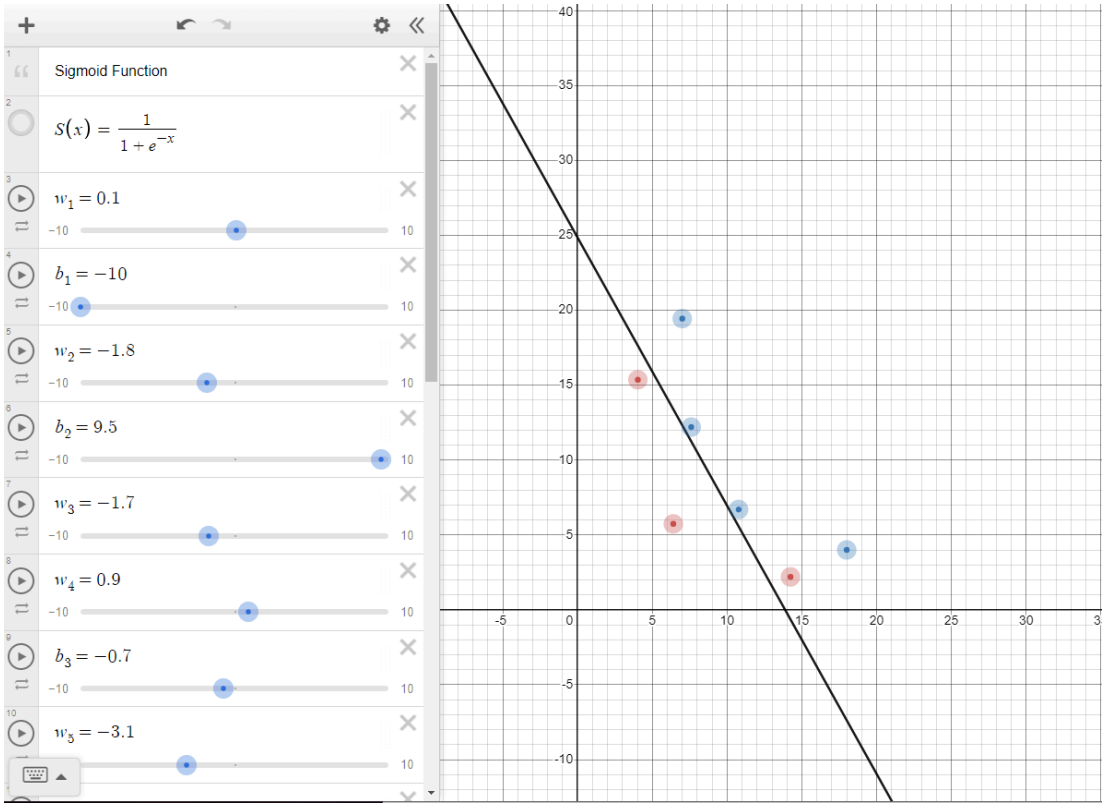
Non Linearity

Pada *Linear Model* bagian sebelumnya, relasi antara *input* dan *output* data bisa direpresentasikan dalam *Linear function* dan dengan garis lurus. Disisi lain, untuk menangani kumpulan data yang kompleks seperti pada bagian terakhir diatas, *Linear Model* tidak bisa menangkap relasi antara data *input* dan *output*. Sebab pada *Linear Model* kita hanya menambahkan Linear function satu dengan *Linear function* lainnya, sehingga hanya *Linear Function* lain yang akan terbentuk. Untuk menangkap kompleksitas data seperti dibawah ini, perlu menggunakan pendekatan yang *Non Linear*, atau *Non Linearity*. Sebaik dan sebanyak apapun parameter yang kita optimalkan, *Linear function* tidak bisa menangkap kompleksitas data *Non linear*.

Percobaan optimasi parameter ke 1.



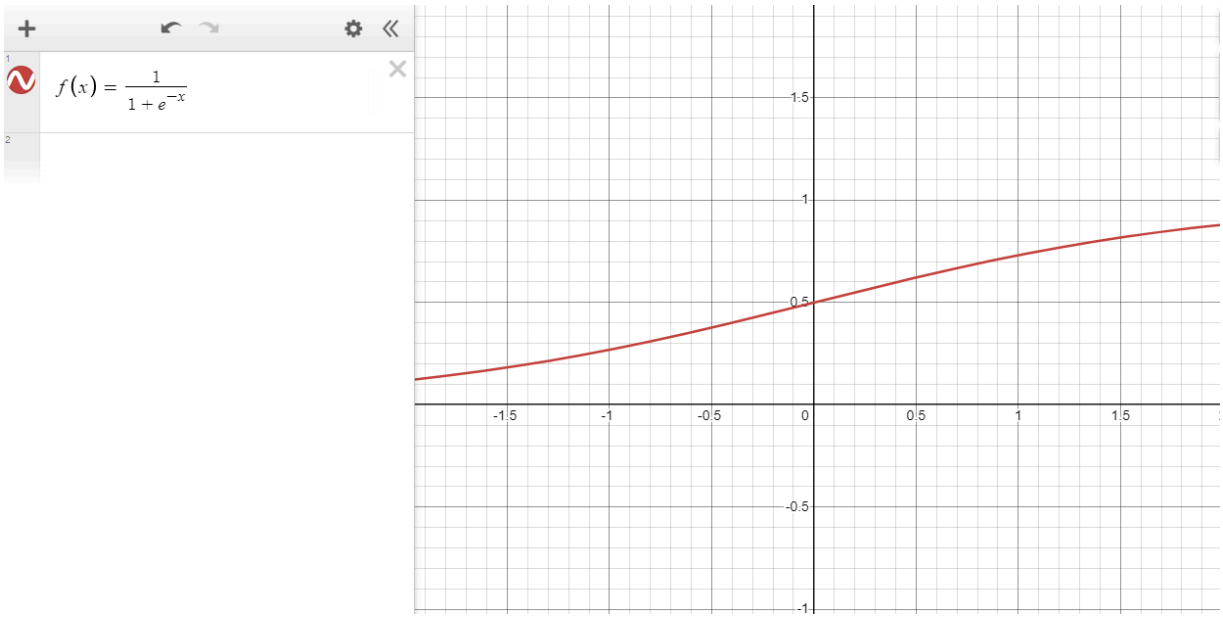
Percobaan optimasi parameter ke 2.



Sigmoid Function

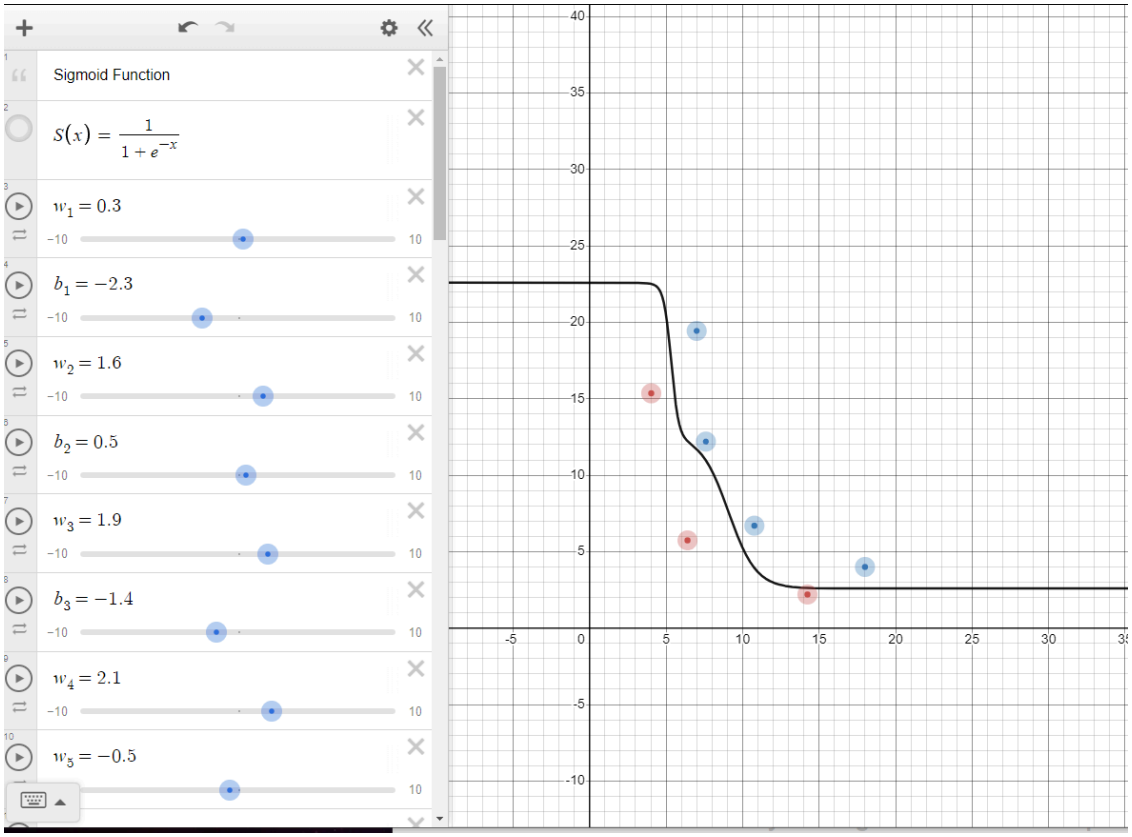
Menangani data *Non Linear* kita membutuhkan garis yang bisa melengkung. Dengan kata lain kita perlu mengubah *Linear function* menjadi *Non Linear function*. Dengan bantuan **Sigmoid function**, kita bisa mengubah *Linear function* menjadi *Non Linear function*. **Sigmoid function** adalah salah satu **Activation function** yang banyak dipakai di area **Machine Learning** untuk melakukan kompresi (*Squashing function*) parameternya menjadi *value* antara 0 - 1 dan menghasilkan value probabilitas.

$$f(x) = \frac{1}{1+e^{-x}}$$



Dengan bentuknya yang melengkung, sehingga **Sigmoid function** juga mempunyai nama lain **S curve** dan **Logistic Curve**.

Kembali pada masalah sebelumnya, kita akan coba menambahkan **Sigmoid function** pada *Linear Model* yang sebelumnya kita buat.



Dengan *tuning* parameter yang sesuai, kita bisa menangkap kompleksitas data yang ada dengan *Non Linear decision boundary* seperti gambar di atas.

Sigmoid Function adalah salah satu dari banyak *activation function* yang digunakan pada **Machine Learning** dan **ANN**. Kita bisa memilih **activation function** yang sesuai tergantung dengan skenario dan permasalahan yang ada. **Tanh**, **ReLU**, **Leaky ReLU** adalah jenis *activation function* lain yang sering digunakan pada **Machine Learning** dan **ANN**.

Pada pembahasan selanjutnya, **Sigmoid Function** akan kita gunakan sebagai *activation function* pada **Artificial Neural Network(ANN)** yang akan kita bangun dalam pembahasan di buku ini.

Derivative (Turunan) dari Sigmoid Function

Pada saat melatih **Artificial Neural Network** kita butuh mengoptimasi setiap *Learnable parameter* (*weights & biases*) melalui informasi yang kita dapatkan dari **Loss Function** dengan cara mencari *derivative*(menurunkan) setiap variabel yang ada. **Sigmoid Function** adalah salah satu variabel **Total Loss Function** yang perlu kita cari *derivative*(turunannya).

$f(x) = \frac{1}{1+e^{-x}}$

Dengan mekanisme *reciprocal of a fraction* $\frac{1}{x} = x^{-1}$, kita bisa menyederhanakan **Sigmoid Function** menjadi:

$f(x) = \left(1 + e^{-x}\right)^{-1}$ mengubah menjadi bentuk *negative exponent*.

Dengan adanya kurung buka dan kurung tutup, menandakan fungsi diatas adalah *composite function*. Fungsi diatas terdiri dari 2 fungsi, yaitu:

$f(u) = u^{-1}$
$g(x) = 1 + e^{-x}$

Menggunakan *Chain Rule*, $f(g(x)) = f'(g(x))g'(x)$ kita perlu mencari *derivative*(turunan) dari setiap fungsi yang ada sebelum mengaplikasikannya ke dalam *Chain Rule*.

Derivative dari fungsi $f(u) = u^{-1}$:
$= f'(u) = (-1)u^{-1-1}$
$= f'(u) = (-1)u^{-2}$
$= f'(u) = -u^{-2}$

Derivative dari fungsi $g(x) = 1 + e^{-x}$:
$= \frac{\Delta}{\Delta x}(1) = 0$ *derivative* $g(x)$ terhadap *Constant* 1 adalah 0

Sedangkan bagian e^{-x}, kita perlu menggunakan *Chain Rule*. Bisa kita lihat, e^{-x} adalah *composite function* yang terdiri dari

$h(u) = e^{u}$ dan $i(x) = -x$.

Derivative dari fungsi $h(u) = e^{u}$ dengan *Exponential Rule*:

$h'(u) = e^{u}$

Derivative dari fungsi $i(x) = -x$:

$i'(x) = -1$

Jadi *derivative* dari fungsi $g(x) = 1 + e^{-x}$:

$g'(x) = e^{-x} \star (-1)$

$g'(x) = -e^{-x}$

Sejauh ini kita sudah mencari *derivative* dari fungsi $f(u)$ dan fungsi $g(x)$.

$f'(u) = -u^{-2}$

$g'(x) = -e^{-x}$

Selanjutnya kita substitusikan pada *Chain Rule* $f(g(x)) = f'(g(x))g'(x)$.

$s'(x) = -\left(1 + e^{-x}\right)^{-2}\left(-e^{-x}\right)$

$= \frac{-e^{-x}}{-\left(1+e^{-x}\right)^{2}}$ Ubah dalam bentuk *fraction*, *power negative* 2 menjadi *positive* 2.

$= \frac{-1*\left(-e^{-x}\right)}{-1*-\left(1+e^{-x}\right)^{2}}$ Kalikan *negative* -1 pada *numerator* dan *denominator*.

$= \frac{e^{-x}}{\left(1+e^{-x}\right)^{2}}$ Sehingga kita hilangkan *negative sign* pada *numerator* dan *denominator*.

$= \frac{1}{1+e^{-x}} * \frac{e^{-x}}{1+e^{-x}}$ kita *expand* lebih lanjut.

$= \frac{1}{1+e^{-x}} * \frac{e^{-x}+1-1}{1+e^{-x}}$ Menambah dan mengurangi dengan 1.

$= \frac{1}{1+e^{-x}} * \frac{(e^{-x}+1)-1}{1+e^{-x}}$

$= \frac{1}{1+e^{-x}} * \left[\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right]$ Lakukan pembagian pada bagian $\frac{1+e^{-x}}{1+e^{-x}}$ = 1

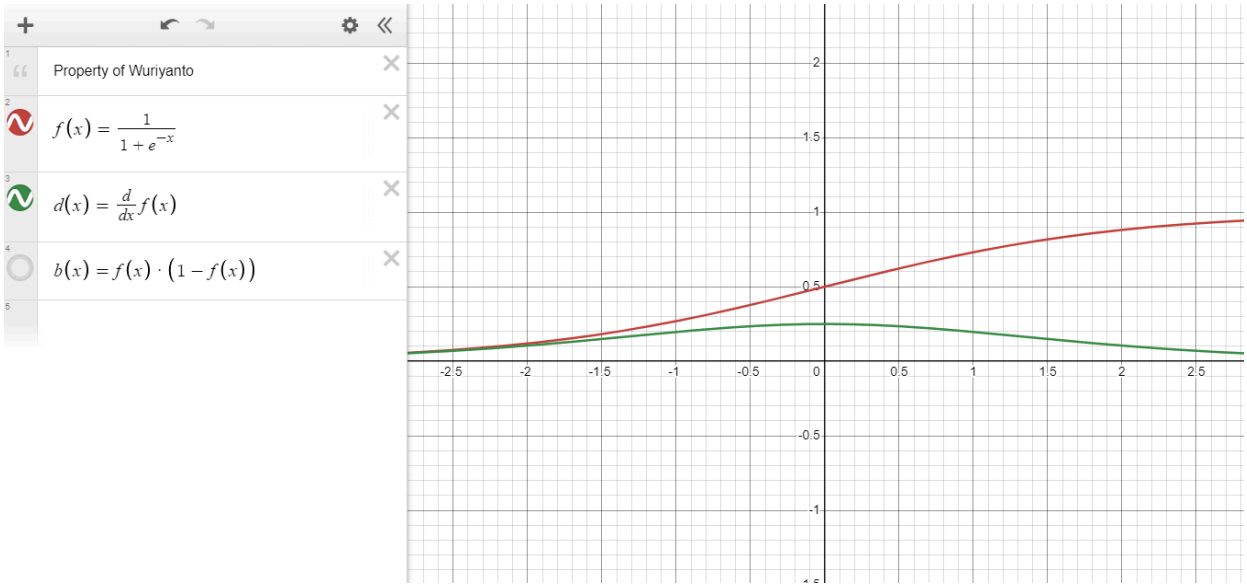
$= \frac{1}{1+e^{-x}} * \left(1 - \frac{1}{1+e^{-x}}\right)$

Sehingga *Derivative* dari **Sigmoid Function**: $s'(x) = \sigma(x)(1 - \sigma(x))$

Implementasi *Derivative* dari **Sigmoid Function** pada Javascript.

```javascript
function derivativeSigmoid(x) {
  return sigmoid(x) * (1 - sigmoid(x));
}
```

Graph **Sigmoid Function** dan *Derivative*-nya



Forward Propagation (Feed Forward)

Secara garis besar kita bisa mempunyai dua pengertian dari **Forward Propagation**. **Forward Propagation** proses training dan **Forward Propagation** proses *inference*. **Forward Propagation** pada saat proses training adalah proses pengiriman data input training ke dalam **Artificial Neural Network** untuk selanjutnya *output* dari **Forward Propagation** dievaluasi secara berulang menggunakan algoritma **Gradient Descent**, yang selanjutnya hasil evaluasi tersebut digunakan untuk mengoptimasi semua parameter (*weights* dan *biases*) yang ada. **Forward Propagation** pada saat proses *inference* adalah proses pengiriman data input aktual kedalam **Artificial Neural Network Model** yang sudah di *training* untuk mendapatkan *output* prediksi.

Contoh Permasalahan: Menentukan siswa lulus atau tidak dari nilai yang diperoleh

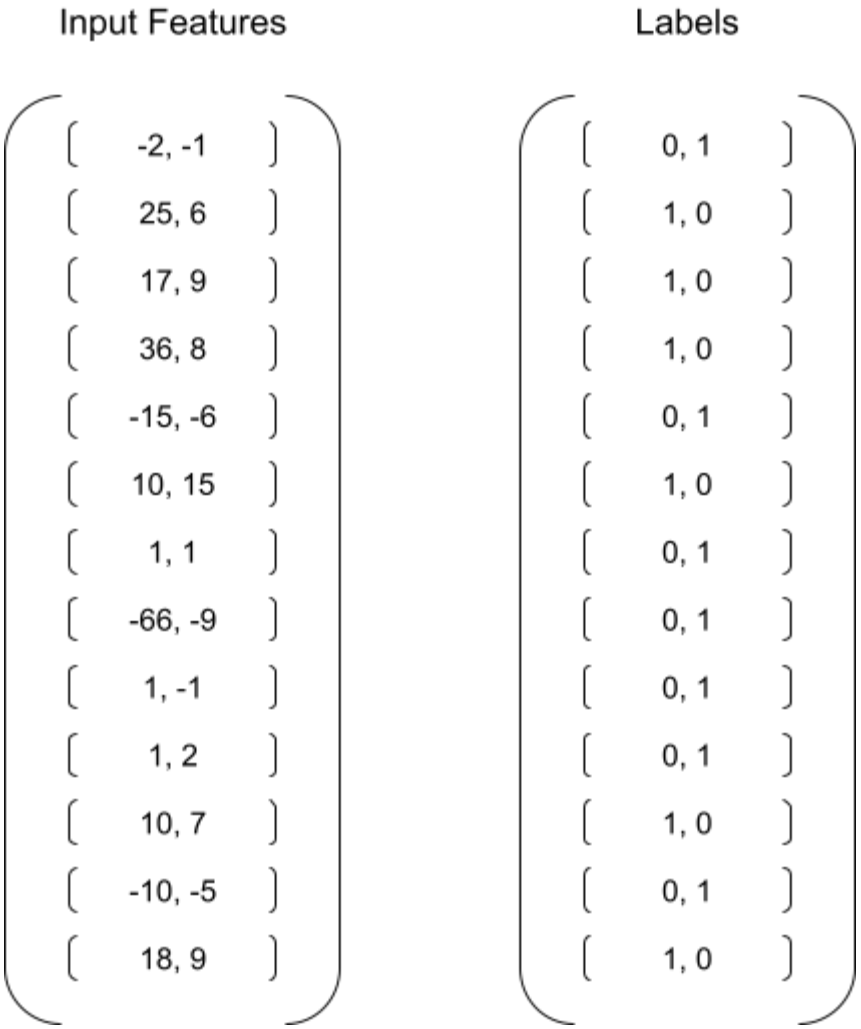
Kita akan mencoba melatih **Artificial Neural Network** dengan arsitektur sederhana menggunakan kumpulan data yang sudah diberikan label untuk menentukan siswa lulus atau tidak dari 2 nilai yang diperoleh.

Dataset untuk kebutuhan data pelatihan:

Nama	Nilai Matematika	Nilai Bahasa Inggris	Lulus	Tidak Lulus
Siswa 1	-2	-1	0	1
Siswa 2	25	6	1	0
Siswa 3	17	9	1	0
Siswa 4	36	8	1	0
Siswa 5	-15	-6	0	1
Siswa 6	10	15	1	0
Siswa 7	1	1	0	1
Siswa 8	-66	-9	0	1
Siswa 9	1	-1	0	1
Siswa 10	1	2	0	1
Siswa 11	10	7	1	0
Siswa 12	-10	-5	0	1
Siswa 13	18	9	1	0

Catatan:
Untuk menyatakan Lulus dan Tidak Lulus kita menggunakan **One Hot Encoding**. Data yang mendapatkan *value* 1 pada kolom Lulus, maka data tersebut masuk dalam label Lulus. Data yang mendapatkan *value* 1 pada kolom Tidak Lulus, maka data tersebut masuk dalam label Tidak Lulus.

Selanjutnya kita akan melakukan *encoding* terhadap diatas dengan menggunakan *Matrix*. *Matrix* pada bahasa pemrograman tertentu bisa di representasikan dengan *Array* Multi Dimensi.



Terdapat 2 data *input*, atau biasa disebut dengan *input features*. Data tersebut adalah Nilai Matematika dan Nilai Bahasa Inggris. Nilai Matematika kita petakan dengan simbol X1 dan Nilai Bahasa Inggris kita petakan dengan simbol X2.

Implementasi *encoding* untuk Data Nilai Matematika dan Bahasa Inggris beserta labelnya dalam Javascript. Data direpresentasikan dengan Matrix, menggunakan Array multi dimensi pada Javascript.

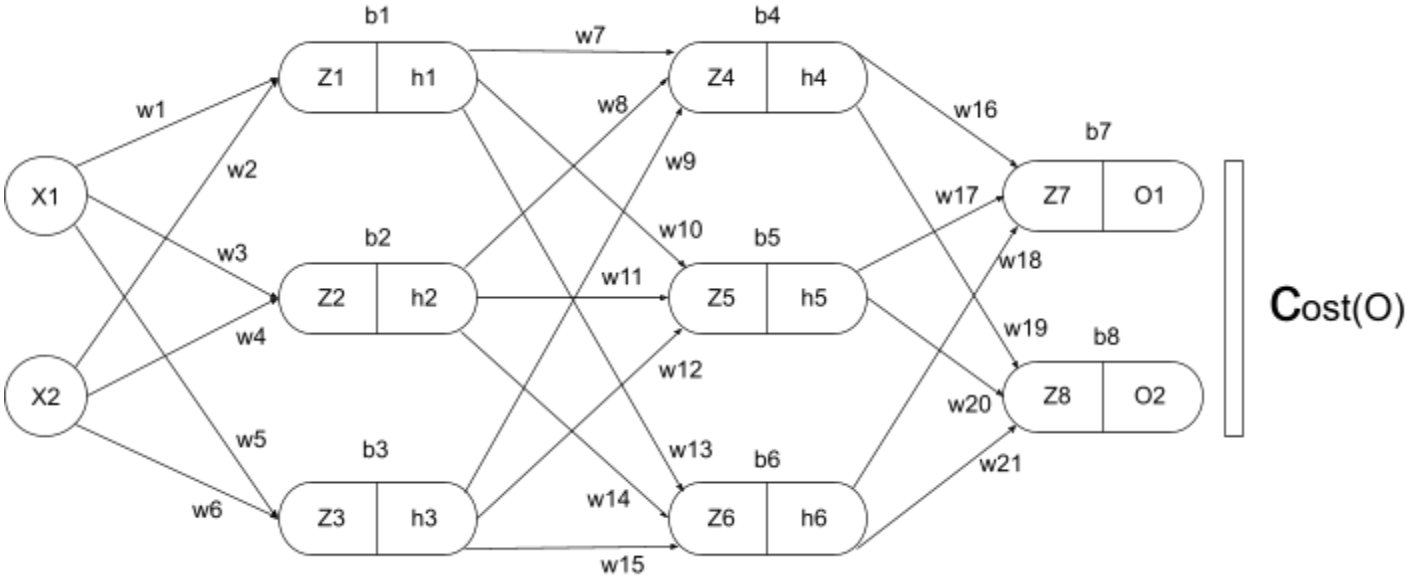
```javascript
const xTrains = [
    [-2, -1], // tidak lulus
    [25, 6], // lulus
    [17, 9], // lulus
    [36, 8], // lulus
    [-15, -6], // tidak lulus
    [10, 15], // lulus
    [1, 1], // tidak lulus
    [-66, -9], // tidak lulus,
    [1, -1], // tidak lulus
    [1, 2], // tidak lulus
    [10, 7], // lulus
    [-10, -5], // tidak lulus,
    [18, 9], // lulus
  ];

  const yTrains = [
    [0,1],
    [1,0],
    [1,0],
    [1,0],
    [0,1],
    [1,0],
    [0,1],
    [0,1],
    [0,1],
    [0,1],
    [1,0],
    [0,1],
    [1,0],
  ];
```

Seperti yang sudah kita bahas sebelumnya. Data label di *encode* menggunakan **One Hot Encoding**.

Arsitektur Network

Arsitektur *Network* yang kita buat untuk menyelesaikan permasalahan diatas terdiri dari 4 layer. *Input Layer*, *Hidden Layer 1*, *Hidden Layer 2*, dan *Output layer*.

Input layer terdapat 2 Neuron, X1 = Nilai Matematika dan X2 = Nilai Bahasa Inggris.
Hidden Layer 1 terdapat 3 *Neuron* dan *Hidden Layer 2* terdapat 3 *Neuron*.
Output Layer terdapat 2 *Neuron*, O1 = Lulus dan O2 = Tidak Lulus.



$X_i = Input$

$w_i = weight$

$b_i = bias$

$O_i = Output$

$Z_i = Linear\ Function \quad Z = \left(\sum_{i=1}^{n} x_i w_i\right) + b$

$h_i = Activation\ Function \quad h = \sigma(Z) \quad Sigmoid = \frac{1}{1+e^{-x}}$

$C_{total} = Cost/Loss\ function \quad C_{total} = \sum_{i=1}^{n} \frac{1}{2}\left(yTrain_i - output_i\right)^2$

$$Z_i = \ Linear\ Function \quad Z = \left(\sum_{i=1}^{n} x_i w_i\right) + b$$

Implementasi Linear Function pada Javascript.

```javascript
function linear(xs, ws, b) {
  if (Array.isArray(xs) && Array.isArray(ws)) {
    let s = 0;
    for (let i = 0; i < xs.length; i++) {
      let x = xs[i];
      let w = ws[i];
      s += w * x;
    }
    return s + b;
  }
  return ws * xs + b;
}

function linearOnObj(xs, ws, b) {
  if (Array.isArray(xs) && Array.isArray(ws)) {
    let s = 0;
    for (let i = 0; i < xs.length; i++) {
      let x = xs[i];
      let w = ws[i];
      s += w * x.h;
    }
    return s + b;
  }
  return ws * xs + b;
}
```

$$Sigmoid = \frac{1}{1+e^{-x}}$$

Implementasi Sigmoid Function pada Javascript:

```javascript
function sigmoid(x) {
  return 1 / (1 + Math.exp(-x));
}
```

$$C_{total} = Cost/Loss\ function \quad C_{total} = \sum_{i=1}^{n} \frac{1}{2}\left(yTrain_i - output_i\right)^2$$

Implementasi Loss/Error Function pada Javascript.

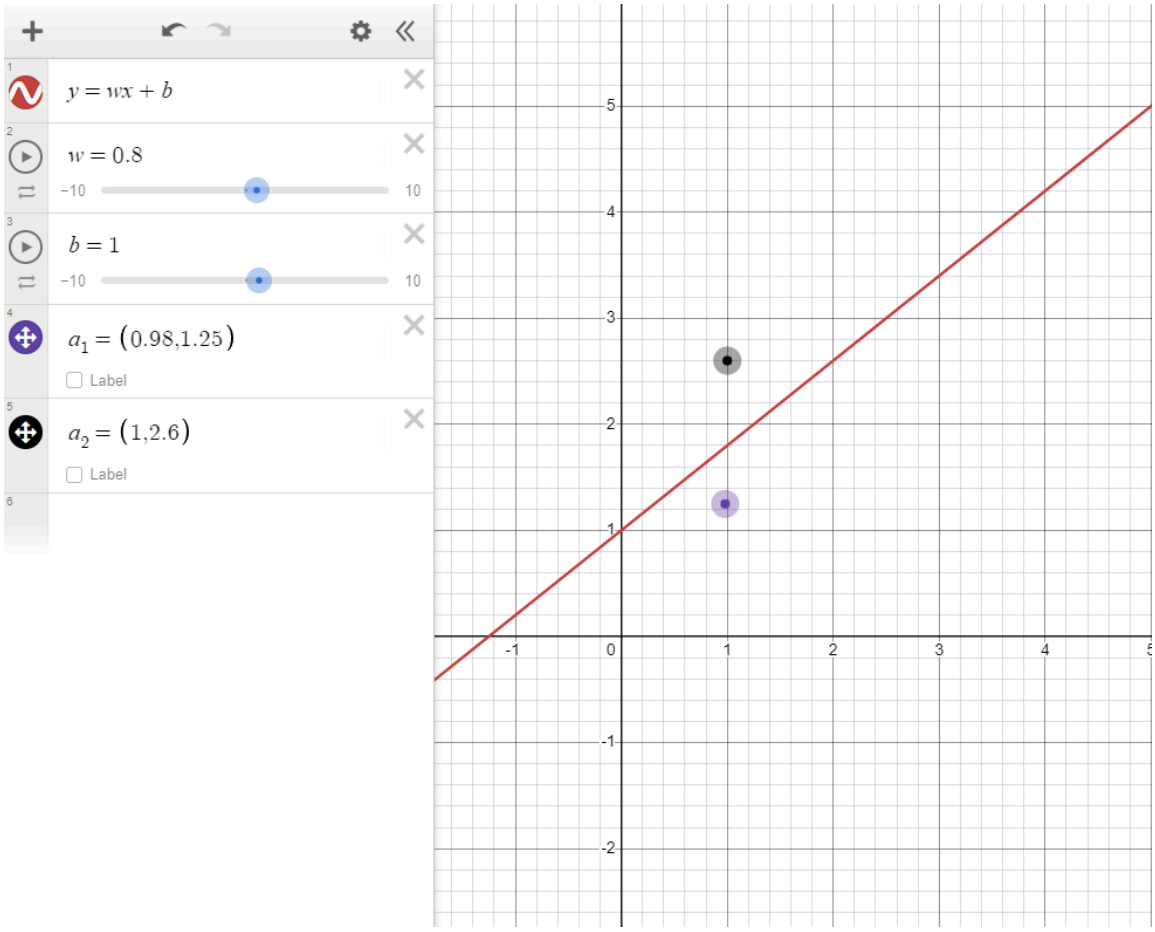
```javascript
function loss(outputs, yTrains) {
  let c = 0;
  for (let i = 0; i < outputs.length; i++) {
    let y = yTrains[i];
    let o = outputs[i];
    c += (1/2) * Math.pow(y - o.h, 2);
  }

  return c;
}
```

Weight dan Bias

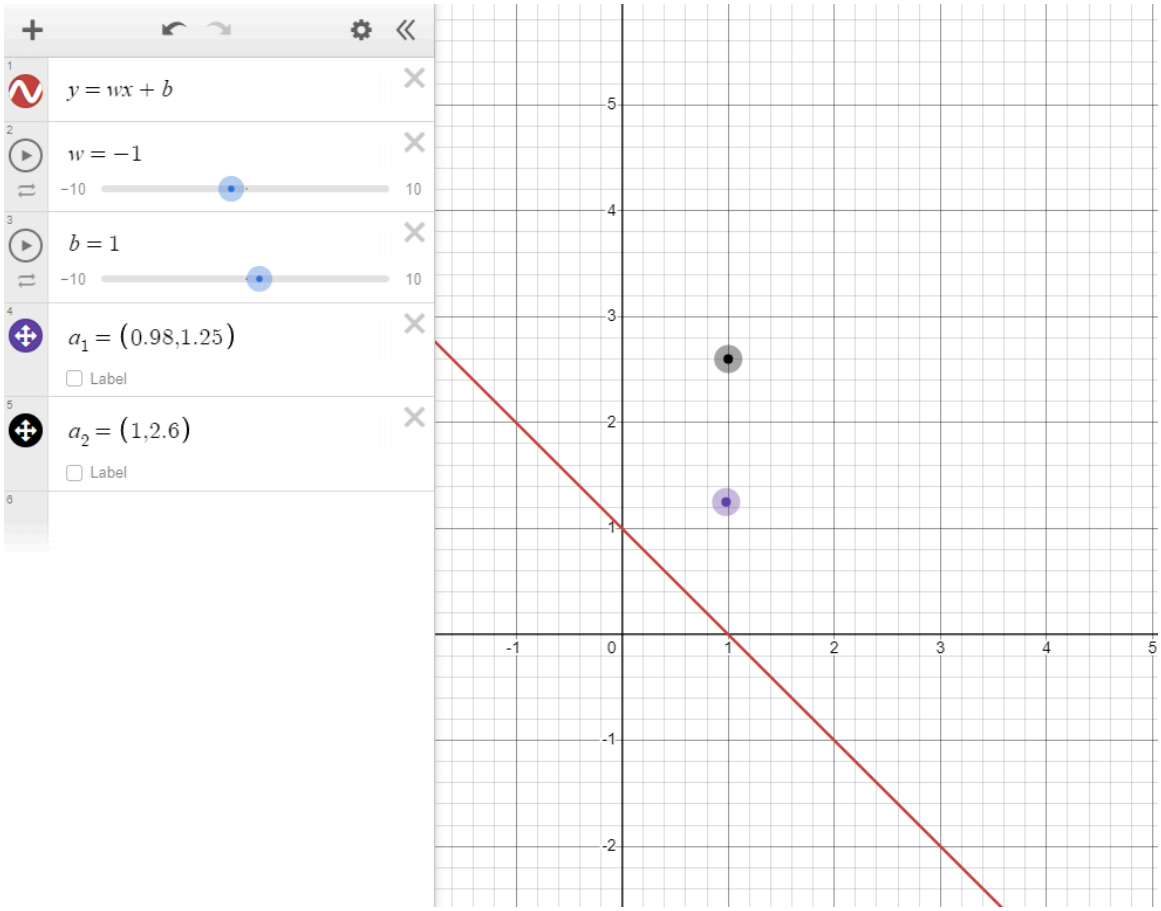
Dua istilah ini sudah sering disebut dalam buku ini. *Weight* dan *Bias* adalah *Learnable Parameter*, yang artinya dua parameter ini bisa dioptimasi seiring berjalannya proses *training*. *Weight* menentukan seberapa kuat koneksi antar *Neuron*. Sedangkan *bias* adalah parameter tambahan yang bisa kita atur untuk menggeser posisi *output* dengan sebuah *value*. **Apa artinya?** Akan kita visualisasikan menggunakan *Linear Function* (*Y-Intercept Form*) yang telah kita pelajari pada bagian **Linear Algebra**.

$$y = wx + b$$

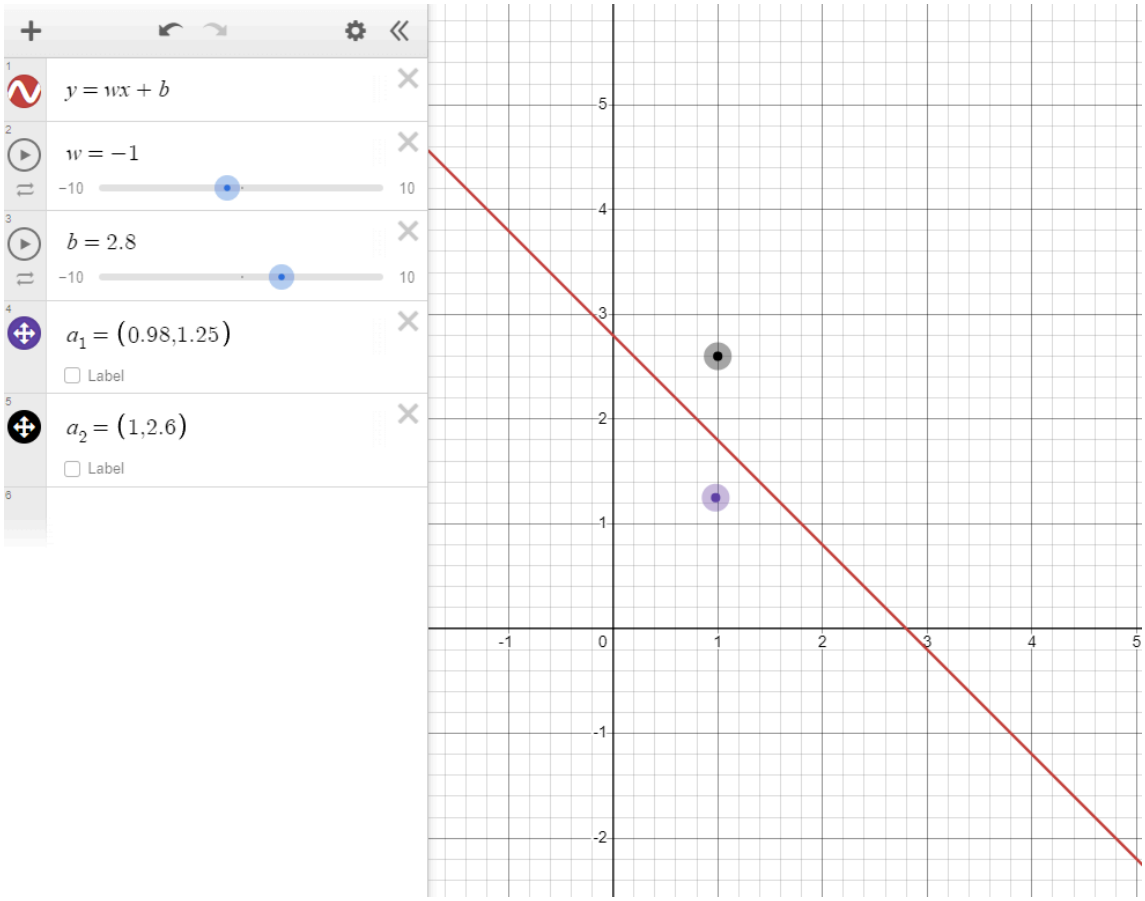
Tuning Weight dan *Bias* 1



Tuning Weight dan *Bias* 2



Tuning Weight dan *Bias* 3



Pada gambar diatas, dengan *Learnable Parameter*(*weight* dan *bias*) kita bisa mengatur *decision boundary* sesuai dengan kebutuhan kita, ke kanan, kekiri, kebawah ataupun keatas. Untungnya, kita tidak akan melakukannya secara manual. Melainkan algoritma **Backpropagation** dan **Gradient Descent** lah yang akan membantu kita mengoptimasi parameter-parameter tersebut.

***Weight* dan *Bias* akan di generate secara *random* pada saat Network pertama kali diinisiasi**. Untuk saat ini semua *Learnable Parameter* di *hardcode*, dengan tujuan lebih intuitif.

JavaScript

```
class NNetwork {
  constructor(epochs, learningRate) {
    this.epochs = epochs;
    this.learningRate = learningRate;
    this.w1 = Math.random();
    this.w2 = Math.random();
    this.w3 = Math.random();
    this.w4 = Math.random();
    this.w5 = Math.random();
    this.w6 = Math.random();
    this.w7 = Math.random();
    this.w8 = Math.random();
    this.w9 = Math.random();
    this.w10 = Math.random();
    this.w11 = Math.random();
    this.w12 = Math.random();
    this.w13 = Math.random();
    this.w14 = Math.random();
    this.w15 = Math.random();
    this.w16 = Math.random();
    this.w17 = Math.random();
    this.w18 = Math.random();
    this.w19 = Math.random();
    this.w20 = Math.random();
    this.w21 = Math.random();
    this.b1 = Math.random();
    this.b2 = Math.random();
    this.b3 = Math.random();
    this.b4 = Math.random();
    this.b5 = Math.random();
    this.b6 = Math.random();
    this.b7 = Math.random();
    this.b8 = Math.random();
  }
```

Activation Function

Artificial Neural Network terdiri dari tumpukan *Node* atau yang biasa kita sebut dengan *Neuron*. *Neuron* yang berada pada posisi selanjutnya menerima *input* dari hasil perkalian dan kalkulasi dari *Neuron* sebelumnya. Setiap input dari *Neuron* sebelumnya adalah hasil dari **Dot Product** dari input sebelumnya dengan *weight*. Kemudian hasil **Dot Product** *input* dan *weight* ditambahkan dengan *bias*.

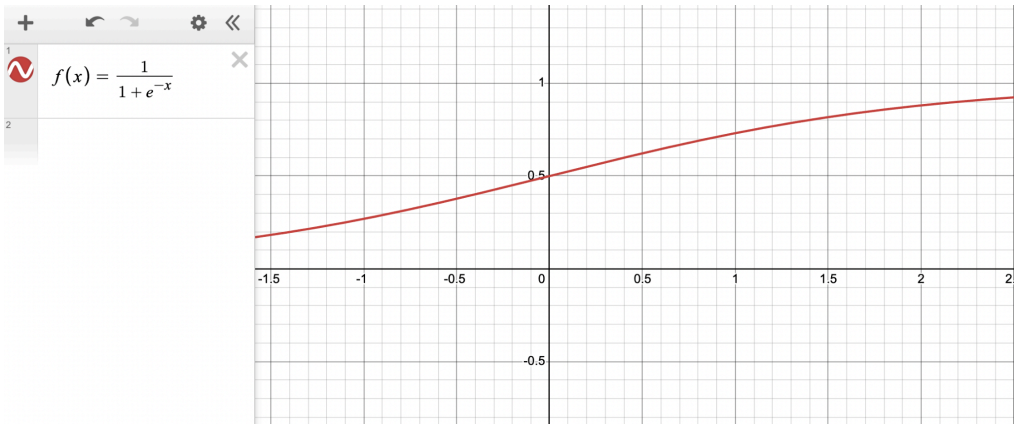
$$Z = \left(\sum_{i=1}^{n} x_i w_i \right) + b$$

Persamaan diatas adalah persamaan *Linear*, atau *Linear Equation*. Seperti yang kita bahas pada bagian sebelumnya. *Linear Equation* hanya bisa belajar atau merepresentasikan relasi *Linear* antara *input* X dan *output* Y. Inilah kondisi dimana kita membutuhkan *Non Linear Activation Function* seperti **Sigmoid Function**.

Setelah operasi **Dot Product** pada *input* dan *weight* dikalkulasi dan kemudian ditambahkan dengan *bias*, kemudian kita tampung pada variabel Z seperti pada persamaan diatas. Selanjutnya Z kita jadikan *input* pada **Activation Function** untuk menghasilkan nilai probabilitas antara $0 \rightarrow 1$. *Non Linear Activation Function* membantu kita merubah kondisi *Linear* menjadi *Non Linear* (*Non-Linearity*). Seperti pada pembahasan sebelumnya, **Activation Function** yang kita pilih adalah **Sigmoid Function**.

Sigmoid Function:

$$f(x) = \frac{1}{1+e^{-x}}$$



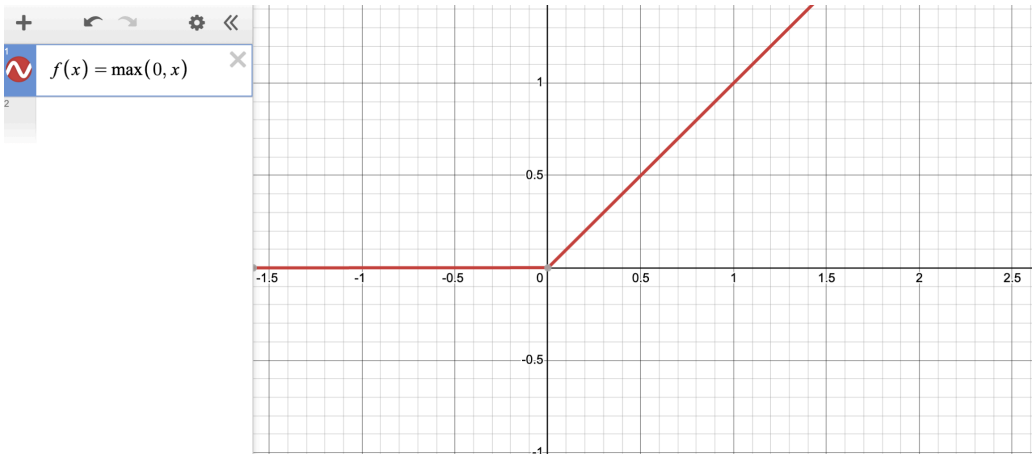
Implementasi Sigmoid Function pada Javascript.

```javascript
function sigmoid(x) {
  return 1 / (1 + Math.exp(-x));
}
```

Activation Function lain yang sering digunakan adalah **ReLU(Rectified Linear Unit)**.

ReLu:

$f(x) = max(0, x)$



Implementasi ReLU Function pada Javascript.

```javascript
function relu(x) {
  return Math.max(0, x);
}
```

Setiap **Activation Function** mempunyai kelebihan dan kekurangannya masing-masing. Misalnya **Sigmoid** di *consider* tidak efisien karena terdapat operasi exponensial. Sedangkan **ReLU** secara komputasi lebih efisien. Hanya saja **ReLU** mempunyai salah satu problem utama yaitu **Dying ReLU Problem**, yaitu kondisi ketika terlalu banyak

Neuron yang mempunyai value dibawah 0. **Pembahasan Activation Function mana yang harus digunakan, saat ini belum masuk dalam pembahasan pada buku ini**.

Operasi Forward Propagation/Feed Forward

Hidden Layer 1

$$\begin{bmatrix} X1 \\ X2 \end{bmatrix} \bullet \begin{bmatrix} W1 \\ W2 \end{bmatrix} + b1 = X1*W1 + X2*W2 + b1 = Z1 = h1(Z1)$$

$$\begin{bmatrix} X1 \\ X2 \end{bmatrix} \bullet \begin{bmatrix} W3 \\ W4 \end{bmatrix} + b2 = X1*W3 + X2*W4 + b2 = Z2 = h2(Z2)$$

$$\begin{bmatrix} X1 \\ X2 \end{bmatrix} \bullet \begin{bmatrix} W5 \\ W6 \end{bmatrix} + b3 = X1*W5 + X2*W6 + b3 = Z3 = h3(Z3)$$

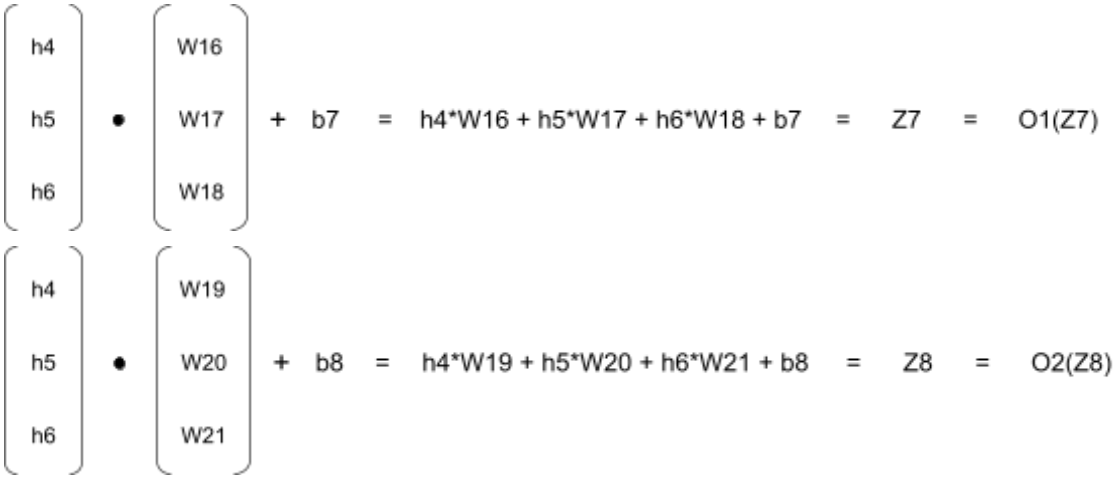
Hidden Layer 2

$$\begin{bmatrix} h1 \\ h2 \\ h3 \end{bmatrix} \bullet \begin{bmatrix} W7 \\ W8 \\ W9 \end{bmatrix} + b4 = h1*W7 + h2*W8 + h3*W9 + b4 = Z4 = h4(Z4)$$

$$\begin{bmatrix} h1 \\ h2 \\ h3 \end{bmatrix} \bullet \begin{bmatrix} W10 \\ W11 \\ W12 \end{bmatrix} + b5 = h1*W10 + h2*W11 + h3*W12 + b5 = Z5 = h5(Z5)$$

$$\begin{bmatrix} h1 \\ h2 \\ h3 \end{bmatrix} \bullet \begin{bmatrix} W13 \\ W14 \\ W15 \end{bmatrix} + b6 = h1*W13 + h2*W14 + h3*W15 + b6 = Z6 = h6(Z6)$$

Output Layer



Dari operasi diatas, kita akan menghasilkan persamaan dibawah ini.

Hidden Layer 1

$Z1 = X1W1 + X2W2 + b1$
$h1 = \sigma(Z1)$
$Z2 = X1W3 + X2W4 + b2$
$h2 = \sigma(Z2)$
$Z3 = X1W5 + X2W6 + b3$
$h3 = \sigma(Z3)$

Hidden Layer 2

$Z4 = h1W7 + h2W8 + h3W9 + b4$
$h4 = \sigma(Z4)$
$Z5 = h1W10 + h2W11 + h3W12 + b5$
$h5 = \sigma(Z5)$
$Z6 = h1W13 + h2W14 + h3W15 + b6$
$h6 = \sigma(Z6)$

Output Layer

$Z7 = h4W16 + h5W17 + h6W18 + b7$
$O1 = \sigma(Z7)$
$Z8 = h4W19 + h5W20 + h6W21 + b8$
$O2 = \sigma(Z8)$

Selanjutnya kita akan inisiasi *Learnable Parameters* (*Weights* dan *Biases*) secara *random* dengan *value* antara $0 \rightarrow 1$.

Total *Learnable Parameters*:

- *Weight* = 21
- *Bias* = 8

Weight:

$W1 = 0.9$	$W8 = 0.5$	$W15 = 0.6$
$W2 = 0.6$	$W9 = 0.4$	$W16 = 0.2$
$W3 = 0.2$	$W10 = 0.3$	$W17 = 0.8$
$W4 = 0.2$	$W11 = 0.7$	$W18 = 0.8$
$W5 = 0.3$	$W12 = 0.7$	$W19 = 0.4$
$W6 = 0.9$	$W13 = 0.1$	$W20 = 0.1$
$W7 = 0.0$	$W14 = 0.2$	$W21 = 0.4$

Bias:

$b1 = 0.9$	$b5 = 0.5$
$b2 = 0.6$	$b6 = 0.4$
$b3 = 0.2$	$b7 = 0.3$
$b4 = 0.2$	$b8 = 0.7$

Kita tidak akan menghitung manual keseluruhan data training yang ada. Kita akan ambil salah satu data training pada dataset kita untuk kita jadikan contoh. Untuk Data Nilai yang akan kita jadikan contoh untuk *Forward Propagation* adalah *row* pertama:

$x = [-2, -1] \quad label = [0, 1]\ (Tidak\ Lulus)$

$X1 = \ -2$
$X2 = \ -1$

Hidden Layer 1

$$Z1 = (-2) * 0.9 + (-1) * 0.6 + 0.9 = -1.5$$
$$h1 = \sigma(-1.5) = 0.18$$
$$Z2 = (-2) * 0.2 + (-1) * 0.2 + 0.6 = 0$$
$$h2 = \sigma(0) = 0.50$$
$$Z3 = (-2) * 0.3 + (-1) * 0.9 + 0.2 = -1.3$$
$$h3 = \sigma(Z3) = 0.21$$

Hidden Layer 2

$$Z4 = 0.18 * 0.0 + 0.50 * 0.5 + 0.21 * 0.4 + 0.2 = 0.534$$
$$h4 = \sigma(Z4) = 0.63$$
$$Z5 = 0.18 * 0.3 + 0.50 * 0.7 + 0.21 * 0.7 + 0.5 = 1.051$$
$$h5 = \sigma(Z5) = 0.74$$
$$Z6 = 0.18 * 0.1 + 0.50 * 0.2 + 0.21 * 0.6 + 0.4 = 0.644$$
$$h6 = \sigma(Z6) = 0.66$$

Output Layer

$$Z7 = 0.63 * 0.2 + 0.74 * 0.8 + 0.66 * 0.8 + 0.3 = 1.546$$
$$O1 = \sigma(Z7) = 0.82$$
$$Z8 = 0.63 * 0.4 + 0.74 * 0.1 + 0.66 * 0.4 + 0.7 = 1.29$$
$$O2 = \sigma(Z8) = 0.78$$

Dengan ekspektasi $label = [0, 1]$ $(Tidak\ Lulus)$, menandakan *output* belum sesuai. Jika kita petakan *output* ke dalam label:
$label\ 0 \rightarrow O1 = 0.82$
$label\ 1 \rightarrow O2 = 0.78$

Output dari label 0 lebih besar dari label 1, $0.82 > 0.78$.

Implementasi **Forward Propagation** pada Javascript.

```javascript
train(xTrains, yTrains) {
    for (let e = 0; e < this.epochs; e++) {
      for (let i = 0; i < yTrains.length; i++) {
        let xTrain = xTrains[i];
        let yTrain = yTrains[i];
        // forward propagation

        // first hidden layer
        let z1 = linear([xTrain[0], xTrain[1]], [this.w1, this.w2], this.b1);
        let h1 = sigmoid(z1);
        let z2 = linear([xTrain[0], xTrain[1]], [this.w3, this.w4], this.b2);
        let h2 = sigmoid(z2);
        let z3 = linear([xTrain[0], xTrain[1]], [this.w5, this.w6], this.b3);
        let h3 = sigmoid(z3);

        // second hidden layer
        let z4 = linear([h1, h2, h3], [this.w7, this.w8, this.w9], this.b4);
        let h4 = sigmoid(z4);
        let z5 = linear([h1, h2, h3], [this.w10, this.w11, this.w12], this.b5);
        let h5 = sigmoid(z5);
        let z6 = linear([h1, h2, h3], [this.w13, this.w14, this.w15], this.b6);
        let h6 = sigmoid(z6);

        // output layer
        let z7 = linear([h4, h5, h6], [this.w16, this.w17, this.w18], this.b7);
        let o1 = sigmoid(z7);
        let z8 = linear([h4, h5, h6], [this.w19, this.w20, this.w21], this.b8);
        let o2 = sigmoid(z8);

        // calculate loss
        let cost = loss([o1, o2], yTrain);
        console.log('epochs: ', e, ' | loss: ', cost);
```

Menghitung Error/ Loss

Setelah kita mendapatkan *output* dari masing-masing label, kita akan menghitung **Total Error/Loss** dari *output* setiap individual training yang kita dapatkan dengan **Loss Function**.

$$C_{total} = \sum_{i=1}^{n} \frac{1}{2}\left(yTrain_i - output_i\right)^2$$

Fungsi **Loss** diatas adalah bentuk untuk individual dari **MSE(Mean Squared Error)**. Jika **MSE** menghitung total error dari keseluruhan *training* data, fungsi **Loss** diatas menghitung *error* dari setiap individual *training* data.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(yTrain_i - output_i\right)^2$$

Kita menggunakan **MSE** ketika kita akan mengevaluasi **Error** dari keseluruhan data *training*.

Dengan ekspektasi label = $[0,\ 1]$ $(Tidak\ Lulus)$, maka:

$O1 = \sigma(Z7) = 0.82$ *Expected output* 0
$O2 = \sigma(Z8) = 0.78$ *Expected output* 1

$C_1 = \frac{1}{2}(0 - 0.82)^2 = 0.34$
$C_2 = \frac{1}{2}(1 - 0.78)^2 = 0.02$

$C_{total} = C1 + C2$
$C_{total} = 0.34 + 0.30 = 0.36$

Value C_{total} selanjutnya akan kita gunakan sebagai acuan untuk mengoptimasi *Learnable Parameter*(*Weights* dan *Biases*) pada *step* **Backpropagation**.

Penggunaan **Loss/Error** function adalah metode untuk mengevaluasi sebaik apa algoritma model yang kita bangun membaca data training yang kita sediakan. Selain **MSE**, masih ada banyak sekali jenis **Loss/Error** *function* yang digunakan pada area

pelatihan Machine Learning. Seperti Cross Entropy, salah satu Loss/Error function yang sering digunakan untuk menyelesaikan masalah klasifikasi.

Derivative (Turunan) dari Loss Function

Pada bagian sebelumnya, kita sudah menghitung **Total Loss Function** dari salah satu contoh *input training* yang ada di data Nilai Siswa. **Individual Loss Function** adalah variabel dari **Total Loss Function**, sama seperti **Sigmoid Function** yang sebelumnya sudah kita cari *Derivative*(Turunan)-nya.

$$C_{total} = \sum_{i=1}^{n} \frac{1}{2}\left(yTrain_i - output_i\right)^2$$

Maka bentuk individual untuk setiap *training* data dari persamaan diatas adalah:
$C = \frac{1}{2}\left(yTrain_i - output_i\right)^2$

Fungsi $C = \frac{1}{2}(yTrain - output)^2$ inilah yang akan kita cari *Derivative* (Turunan)-nya.

$\frac{\Delta C}{\Delta_{output}} = \frac{1}{2} * 2 * (yTrain - output)$ Menggunakan *Power Rule* $x = x^n \rightarrow nx^{n-1}$

Selanjutnya kita mencari *derivative* $\frac{\Delta C}{\Delta_{output}}$ pada bagian yang ada di dalam kurung $(yTrain - output)$. Kita perlakukan $yTrain$ sebagai *constant*. Maka *Derivative* $\frac{\Delta C}{\Delta_{output}}$ pada bagian yang ada di dalam kurung $(yTrain - output)$ adalah (-1). Yaitu *coefficient* dari *negative* $(-output)$.

Sehingga kita mendapatkan *derivative* akhir:

$$\frac{\Delta C}{\Delta_{output}} = \frac{1}{2} * 2 * (yTrain - output) * -1$$

Implementasi *Derivative* dari **Loss/Error Function** pada Javascript.

```javascript
function calculateDerivativeLoss(output, yTrue) {
  return 2 * (1/2) * (yTrue - output) * -1;
}
```

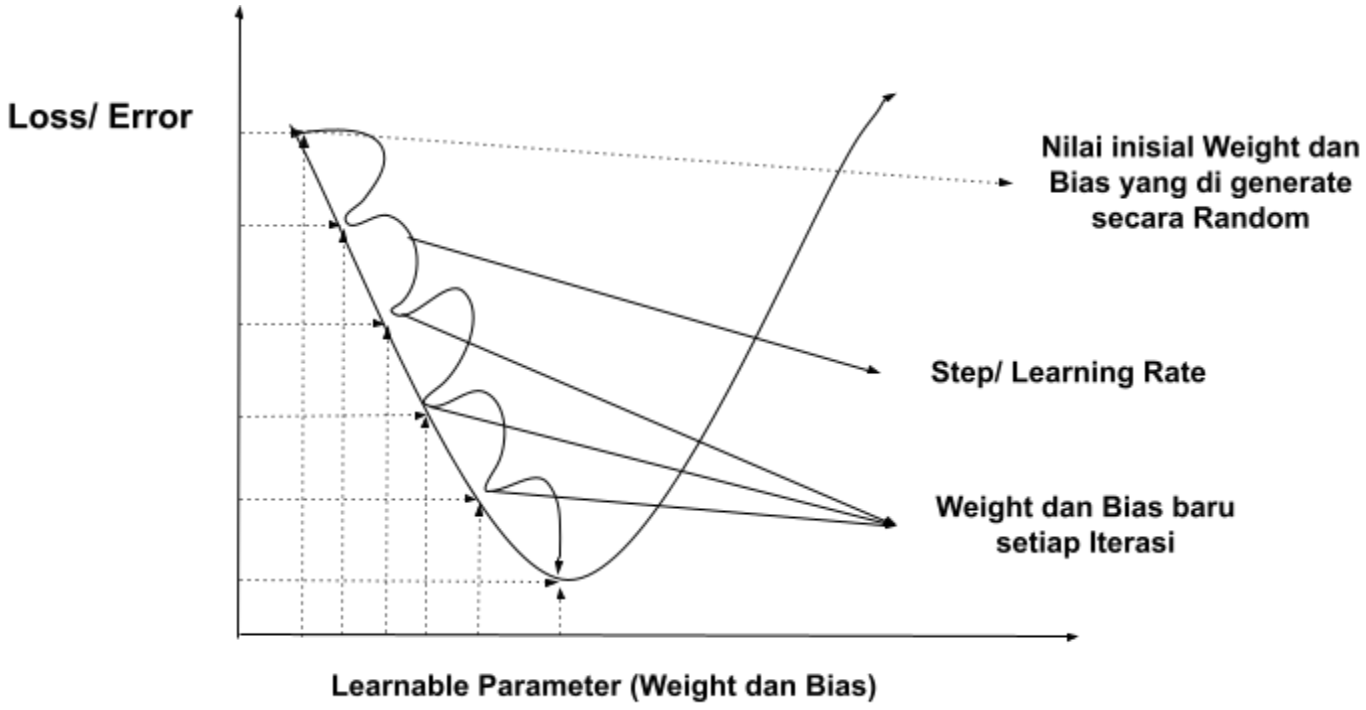
Gradient Descent

Pada bagian **"Menghitung Error/ Loss"**, kita sudah mendapatkan **Total Loss/ Error** dari salah satu *training* data kita.

$C_{total} = 0.34 + 0.30 = 0.36$

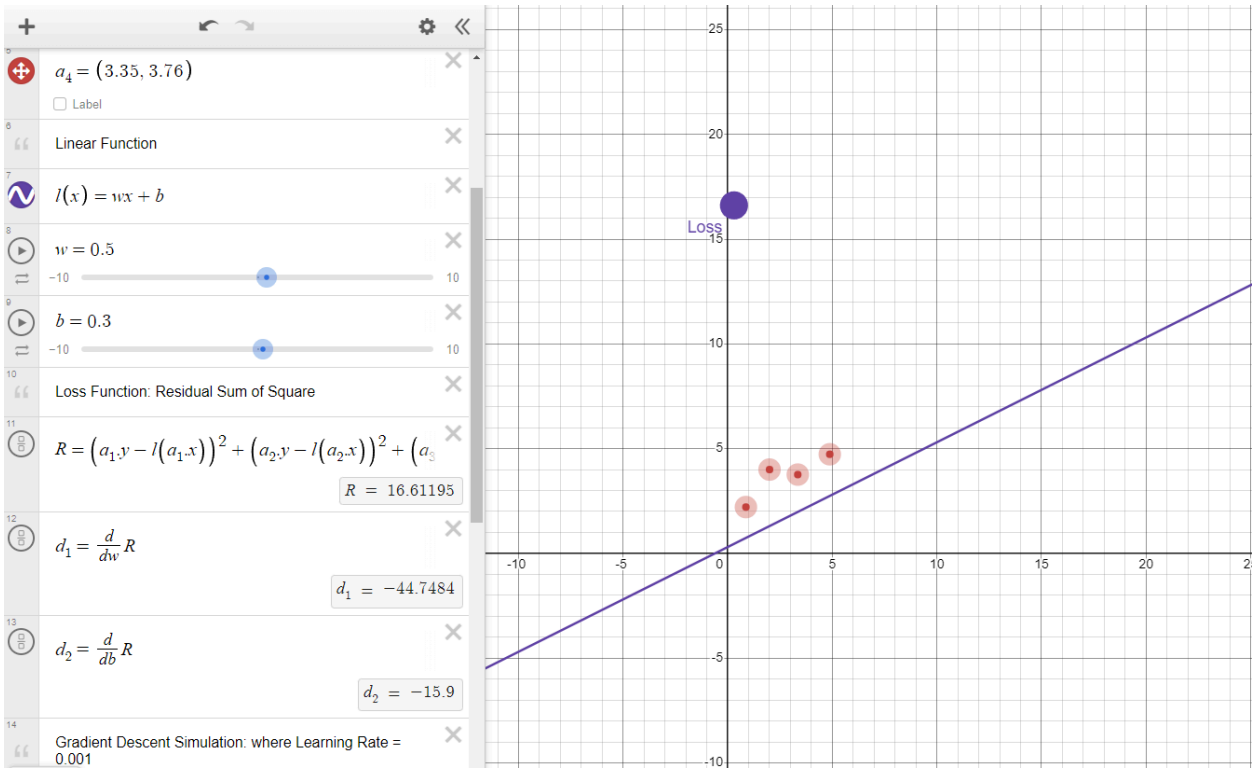
Goal/ tujuan kita melatih **Artificial Neural Network** adalah mengoptimasi *Learnable Parameter* (*Weight* dan *Bias*) dengan cara meminimalkan C_{total} sekecil mungkin.

Dengan algoritma **Gradient Descent** kita bisa meminimalkan Total Loss/ Error sekaligus mengoptimasi *Learnable Parameter* pada **Artificial Neural Network** yang kita bangun. **Gradient Descent** adalah salah satu algoritma optimasi untuk mencari nilai optimal **Loss/ Error** function dengan cara mencari nilai minimumnya.

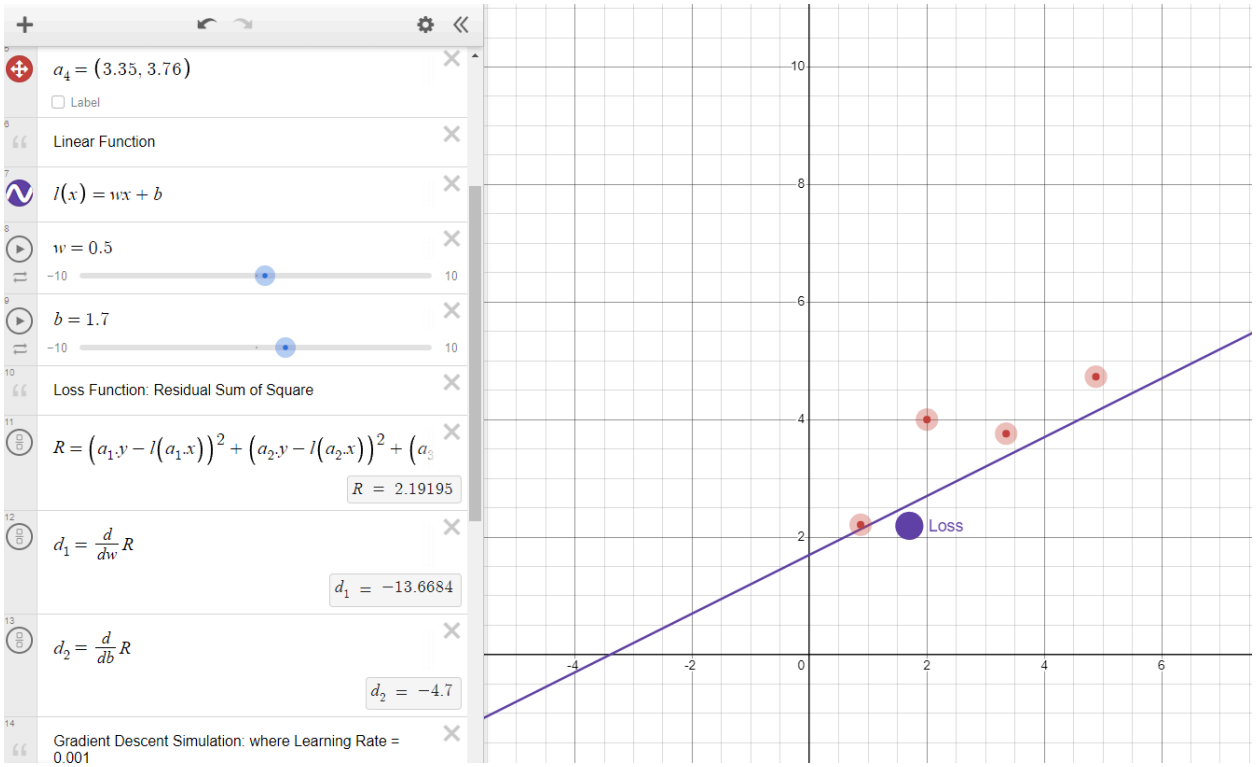


Simulasi Gradient Descent

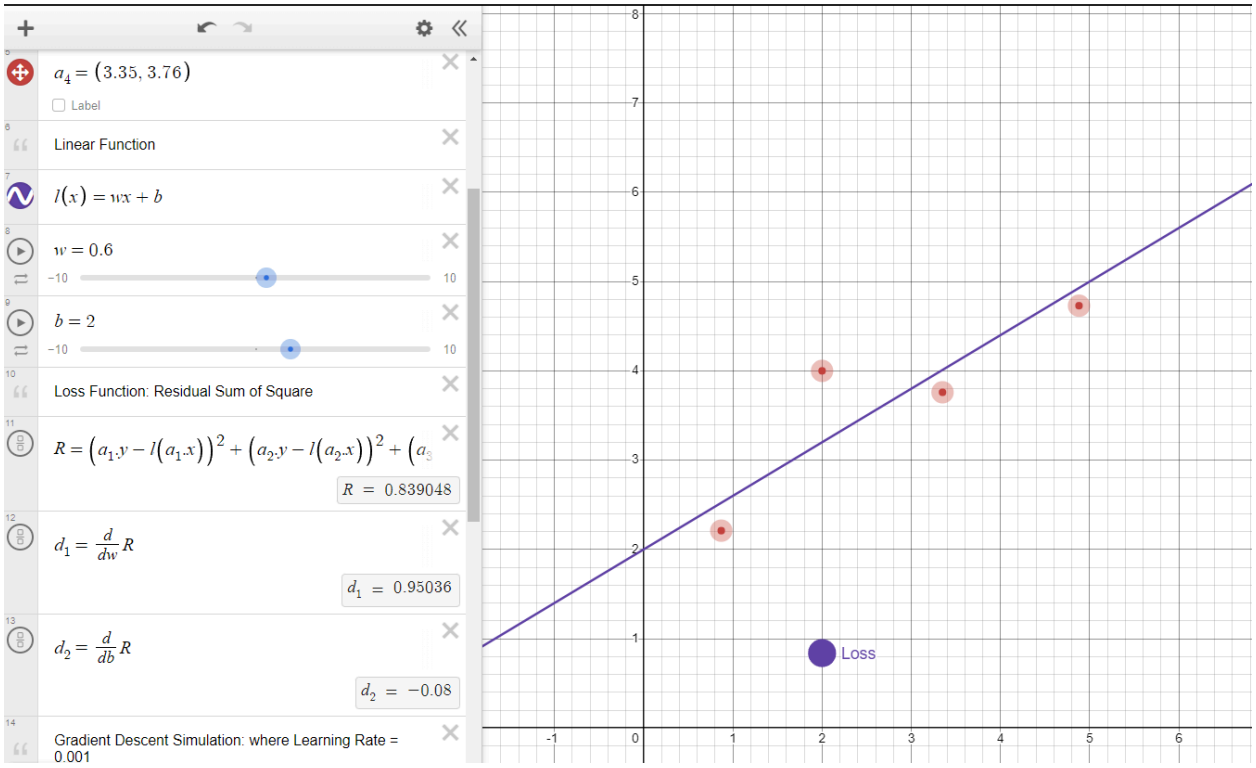
Pada *graph* dibawah ini, kita akan mencoba mensimulasikan **Gradient Descent** pada **Regression Line** sederhana, atau biasa disebut dengan **Linear Regression**. Fungsi **Loss/Error** yang digunakan adalah **RSS (Residual Sum of Square)**. Pada graph pertama berikut ini, adalah **Regression Line** yang belum dioptimasi.



Dengan nilai **Loss/Error** dari **RSS (Residual Sum of Square)** yang mencapai 16.61195, menandakan 2 *Learnable parameters*(1 *weight* dan 1 *bias*) belum optimal. *Graph* kedua di bawah ini adalah hasil dari optimasi selanjutnya. Bisa kita lihat, nilai dari **Loss/Error** semakin mengecil dengan semakin optimalnya **Regression Line**-nya.



Nilai **Loss/Error** 0.839048 adalah nilai minimum yang ditemukan dari proses **Minimum Optimization** yang kita lakukan. Bisa kita lihat pada *graph* ketiga di bawah ini, **Regression Line** yang kita buat sudah *fit* terhadap data *training* yang ada. Menandakan *Learnable Parameter* yang ada (1 *weight* dan 1 *bias*) sudah optimal terhadap data *training* yang ada. Maka alur algoritma **Gradient Descent** bisa kita hentikan.



Gradient Descent formula:

$$w_j = w_j - \eta \frac{\partial C_{total}}{\partial w_j}$$

Dimana:
$w = Learnable\ Parameters(Weight\ dan\ Bias)$
$\eta = step\ atau\ learning\ rate$
$\frac{\partial C_{total}}{\partial w_j} = partial\ derivative\ dari\ C_{total}\ respect\ to\ w_i$

Nilai *step* atau ***learning rate*** biasanya kita *set* dengan *value* yang sangat kecil, seperti $0.01, 0.001$. Karena parameter seperti *learning rate* ini adalah kita yang menentukan, maka parameter seperti ***learning rate*** biasa disebut sebagai **Hyperparameter**. **Hyperparameter** lain adalah **Epoch**. **Epoch** adalah parameter yang menentukan berapa kali **Artificial Neural Network** yang kita bangun melewati keseluruhan data *training*. Atau disebut juga sebagai **learning loop**.

Backpropagation/ Backward Propagation

Setelah step pada bagian sebelumnya kita melakukan **Forward Propagation**. Step dimana kita merambatkan(*propagating*) data input ke *input layer, hidden layer, output layer,* dan akhirnya sampai ke perhitungan dan menghasilkan nilai **Loss/Error**. Pada step **Backpropagation** kita melakukan sebaliknya. Yaitu step dimana kita merambatkan(*propagating*) mundur **Loss/Error** yang sudah diperoleh ke *output layer, hidden layer*, dan akhirnya sampai ke *input layer*. Pada step **Backpropagation**, setiap *Learnable Parameter* akan di *update* dengan dasar nilai *Loss/Erro*r yang didapat dari step **Forward Propagatio**n. Dengan bantuan algoritma **Gradient Descent**, **Backpropagation** akan mengoptimalkan setiap *Learnable parameter* (*weight* dan *bias*) dan meminimalkan nilai **Loss/Error**. Step ini akan dilakukan secara berulang sebanyak **Epoch** (*training loop*) **Hyperparameter** yang kita tentukan.

Jika kita kembali pada arsitektur network yang kita bangun, kita mempunyai total *Learnable parameter*:

- *Weight* = 21
- *Bias* = 8

Dengan kondisi $W1 \rightarrow W6$ dan $b1 \rightarrow b3$ berada pada *layer* 1, $W7 \rightarrow W15$ dan $b4 \rightarrow b6$ berada pada *layer* 2, dan $W16 \rightarrow W21$ dan $b7 \rightarrow b8$ berada pada *layer* 3. Kita akan

ambil contoh *weight* dan *bias* sebagai perwakilan dari setiap *layer*, karena pada dasarnya proses perhitungannya sama pada setiap *layer*, maka kita tidak akan menghitung manual keseluruhan *Learnable parameter* yang ada.

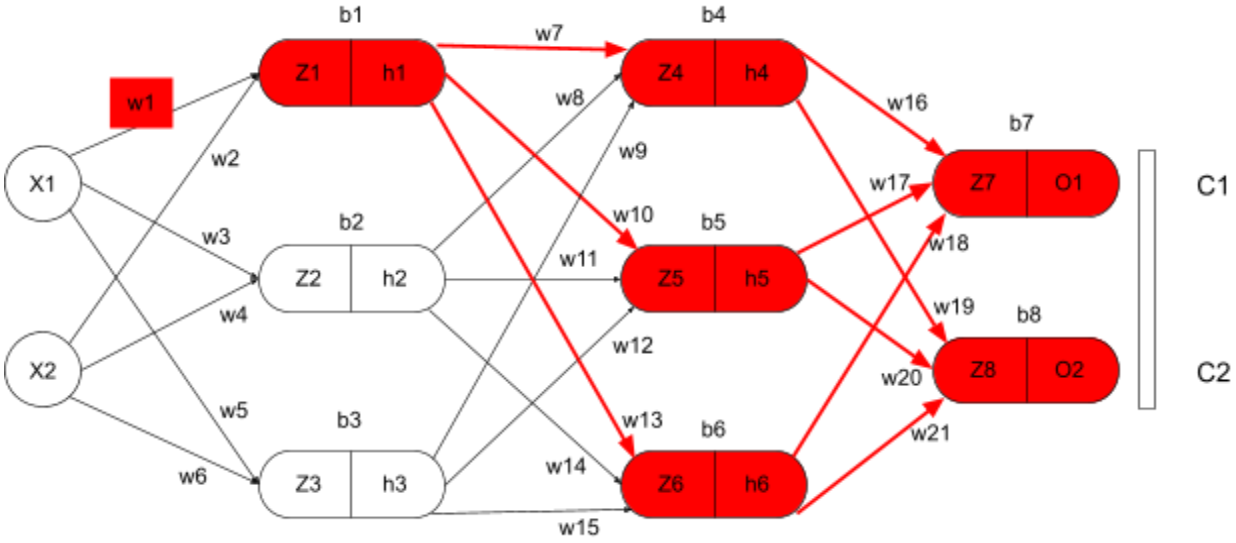
- $W1$ dan $b1$ (perwakilan *layer* 1). Perhitungan $W2 \rightarrow W6$ dan $b2 \rightarrow b3$ sama.
- $W7$ dan $b4$ (perwakilan *layer* 2). Perhitungan $W8 \rightarrow W15$ dan $b5 \rightarrow b6$ sama.
- $W16$ dan $b7$ (perwakilan *layer* 3). Perhitungan $W17 \rightarrow W21$ dan $b8$ sama.

Kita akan mulai menghitung seberapa berpengaruh $W1$ pada pada **Loss/Error** *row* pertama pada data *training* kita.

Setelah pada step sebelumnya kita mendapatkan **Total Loss/Error** dari training data *row* pertama.

$$C_{total} = 0.34 + 0.30 = 0.36$$

Pertama, kita akan mencari keseluruhan jalur(*path*) yang dilewati $W1$ sampai dengan perhitungan C_{total}.



Node dan garis yang berwarna merah adalah keseluruhan komponen yang dilewati oleh $W1$ untuk sampai ke perhitungan C_{total}. Jika diperhatikan, total ada 6 jalur(*path*) dari $W1$. 3 jalur untuk $W1 \rightarrow C_1$ dan 3 jalur untuk $W1 \rightarrow C_2$. Karena pada dasarnya satuan *Loss/Error function* adalah *composite function*, kita perlu menerapkan *Chain Rule* untuk mencari *derivate* dari setiap *function* yang terkomposisi.

$$C_{total} = C_1(O_1(\sigma(Z_i(x,w,b)))) + C_2(O_2(\sigma(Z_i(x,w,b))))$$

Fungsi *Linear* Z adalah *multivariable function*. Pada bagian **Multivariable Calculus dan Partial Derivative** kita sudah mempelajari bagaimana cara mencari *derivative* dari setiap satuan parameter/variabel dari *multivariable function*. Pada bagian ini, kita akan memanfaatkan pengetahuan *Chain Rule* dan *Partial Derivate* untuk mencari sensitivitas setiap parameter dari *Loss/Error function*.

Pertama kita akan mencari setiap formula *derivative* dari semua jalur yang masuk ke C_1 dan C_2 dengan mengikuti jalur(*path*) warna merah dan mundur/ *backward* menggunakan *Chain Rule*.

$$\frac{\partial C_1}{\partial W_1} = \frac{\partial C_1}{\partial O_1} * \frac{\partial O_1}{\partial Z_7} * \frac{\partial Z_7}{\partial h_4} * \frac{\partial h_4}{\partial Z_4} * \frac{\partial Z_4}{\partial h_1} * \frac{\partial h_1}{\partial Z_1} * \frac{\partial Z_1}{\partial W_1} + \frac{\partial C_1}{\partial O_1} * \frac{\partial O_1}{\partial Z_7} * \frac{\partial Z_7}{\partial h_5} * \frac{\partial h_5}{\partial Z_5} * \frac{\partial Z_5}{\partial h_1} * \frac{\partial h_1}{\partial Z_1} * \frac{\partial Z_1}{\partial W_1} + \frac{\partial C_1}{\partial O_1} * \frac{\partial O_1}{\partial Z_7} * \frac{\partial Z_7}{\partial h_6} * \frac{\partial h_6}{\partial Z_6} * \frac{\partial Z_6}{\partial h_1} * \frac{\partial h_1}{\partial Z_1} * \frac{\partial Z_1}{\partial W_1}$$

$$\frac{\partial C_2}{\partial W_1} = \frac{\partial C_2}{\partial O_2} * \frac{\partial O_2}{\partial Z_8} * \frac{\partial Z_8}{\partial h_4} * \frac{\partial h_4}{\partial Z_4} * \frac{\partial Z_4}{\partial h_1} * \frac{\partial h_1}{\partial Z_1} * \frac{\partial Z_1}{\partial W_1} + \frac{\partial C_2}{\partial O_2} * \frac{\partial O_2}{\partial Z_8} * \frac{\partial Z_8}{\partial h_5} * \frac{\partial h_5}{\partial Z_5} * \frac{\partial Z_5}{\partial h_1} * \frac{\partial h_1}{\partial Z_1} * \frac{\partial Z_1}{\partial W_1} + \frac{\partial C_2}{\partial O_2} * \frac{\partial O_2}{\partial Z_8} * \frac{\partial Z_8}{\partial h_6} * \frac{\partial h_6}{\partial Z_6} * \frac{\partial Z_6}{\partial h_1} * \frac{\partial h_1}{\partial Z_1} * \frac{\partial Z_1}{\partial W_1}$$

Untuk menghitung $\frac{\partial C_{total}}{\partial W_1}$, kita jumlahkan $\frac{\partial C_1}{\partial W_1}$ dan $\frac{\partial C_2}{\partial W_1}$.

$$\frac{\partial C_{total}}{\partial W_1} = \frac{\partial C_1}{\partial W_1} + \frac{\partial C_2}{\partial W_1}$$

Kedua, kita menghitung setiap *derivative*(turunan) dari setiap komponen formula diatas.

$\frac{\partial C_1}{\partial O_1} = \frac{1}{2} * 2 * (0 - 0.82) * -1 = 0.82$ (Derivative dari Loss/Error function with respect to $O1$)

$\frac{\partial O_1}{\partial Z_7} = \sigma(1.546)(1 - \sigma(1.546)) = 0.14$ (Derivative dari Sigmoid function $O1$ with respect to $Z7$)

$\frac{\partial Z_7}{\partial h_4} = h4W16 + h5W17 + h6W18 + b7 = 0.2$ (Dengan beberapa derivative rule yang sudah kita pelajari, maka derivative dari $\frac{\partial Z_7}{\partial h_4}$ with respect to $h4$ adalah $W16 = 0.2$. Dalam hal ini coefficient dari $h4$)

$\frac{\partial h_4}{\partial Z_4} = \sigma(0.534)(1 - \sigma(0.534)) = 0.23$ (Derivative dari Sigmoid function $h4$ with respect to $Z4$)

$\frac{\partial Z_4}{\partial h_1} = h1W7 + h2W8 + h3W9 + b4 = 0.0$ (Dengan beberapa derivative rule yang sudah kita pelajari, maka derivative dari $\frac{\partial Z_4}{\partial h_1}$ with respect to $h1$ adalah $W7 = 0.0$. Dalam hal ini coefficient dari $h1$).

$\frac{\partial h_1}{\partial Z_1} = \sigma(-1.5)(1 - \sigma(-1.5)) = 0.15$ (Derivative dari Sigmoid function $h1$ with respect to $Z1$)

$\frac{\partial Z_1}{\partial W_1} = X1W1 + X2W2 + b1 = -2$ (Dengan beberapa derivative rule yang sudah kita pelajari, maka derivative dari $\frac{\partial Z_1}{\partial W_1}$ with respect to $W1$ adalah $X1 = -2$. Dalam hal ini coefficient dari $W1$)

$\frac{\partial Z_7}{\partial h_5} = h4W16 + h5W17 + h6W18 + b7 = 0.8$ (Dengan beberapa derivative rule yang sudah kita pelajari, maka derivative dari $\frac{\partial Z_7}{\partial h_5}$ with respect to $h5$ adalah $W17 = 0.8$. Dalam hal ini coefficient dari $h5$)

$\frac{\partial h_5}{\partial Z_5} = \sigma(1.051)(1 - \sigma(1.051)) = 0.19$ (Derivative dari Sigmoid function $h5$ with respect to $Z5$)

$\frac{\partial Z_5}{\partial h_1} = h1W10 + h2W11 + h3W12 + b5 = 0.3$ (Dengan beberapa derivative rule yang sudah kita pelajari, maka derivative dari $\frac{\partial Z_5}{\partial h_1}$ with respect to $h1$ adalah $W10 = 0.3$. Dalam hal ini coefficient dari $h1$)

$\frac{\partial Z_7}{\partial h_6} = h4W16 + h5W17 + h6W18 + b7 = 0.8$ (Dengan beberapa derivative rule yang sudah kita pelajari, maka derivative dari $\frac{\partial Z_7}{\partial h_6}$ with respect to $h6$ adalah $W18 = 0.8$. Dalam hal ini coefficient dari $h6$)

$\frac{\partial h_6}{\partial Z_6} = \sigma(0.644)(1 - \sigma(0.644)) = 0.23$ (Derivative dari Sigmoid function $h6$ with respect to $Z6$)

$\frac{\partial Z_6}{\partial h_1} = h1W13 + h2W14 + h3W15 + b6 = 0.1$ (Dengan beberapa derivative rule yang sudah kita pelajari, maka derivative dari $\frac{\partial Z_6}{\partial h_1}$ with respect to $h1$ adalah $W13 = 0.1$. Dalam hal ini coefficient dari $h1$)

$\frac{\partial C_2}{\partial O_2} = \frac{1}{2} * 2 * (1 - 0.78) * -1 = -0.22$ (Derivative dari Loss/Error function with respect to $O2$)

$\frac{\partial O_2}{\partial Z_8} = \sigma(1.29)(1 - \sigma(1.29)) = 0.17$ (Derivative dari Sigmoid function $O2$ with respect to $Z8$)

$\frac{\partial Z_8}{\partial h_4} = h4W19 + h5W20 + h6W21 + b8 = 0.4$ (Dengan beberapa derivative rule yang sudah kita pelajari, maka derivative dari $\frac{\partial Z_8}{\partial h_4}$ with respect to $h4$ adalah $W19 = 0.4$. Dalam hal ini coefficient dari $h4$)

$\frac{\partial Z_8}{\partial h_5} = h4W19 + h5W20 + h6W21 + b8 = 0.1$ (Dengan beberapa derivative rule yang sudah kita pelajari, maka derivative dari $\frac{\partial Z_8}{\partial h_5}$ with respect to $h5$ adalah $W20 = 0.1$. Dalam hal ini coefficient dari $h5$)

$\frac{\partial Z_8}{\partial h_6} = h4W19 + h5W20 + h6W21 + b8 = 0.4$ (Dengan beberapa derivative rule yang sudah kita pelajari, maka derivative dari $\frac{\partial Z_8}{\partial h_6}$ with respect to $h6$ adalah $W21 = 0.4$. Dalam hal ini coefficient dari $h6$)

Ketiga, kita sudah mendapatkan *derivative*(turunan) dari setiap komponen formula diatas, selanjutnya kita subtitusikan setiap komponen hasil perhitungan ke dalam formula.

$$\frac{\partial C_1}{\partial W_1} = 0.82 * 0.14 * 0.2 * 0.23 * 0.0 * 0.15 * -2 +$$
$$0.82 * 0.14 * 0.8 * 0.19 * 0.3 * 0.15 * -2 +$$
$$0.82 * 0.14 * 0.8 * 0.23 * 0.1 * 0.15 * -2$$

$$\frac{\partial C_2}{\partial W_1} = -0.22 * 0.17 * 0.4 * 0.23 * 0.0 * 0.15 * -2 +$$
$$-0.22 * 0.17 * 0.1 * 0.19 * 0.3 * 0.15 * -2 +$$
$$-0.22 * 0.17 * 0.4 * 0.23 * 0.1 * 0.15 * -2$$

$$\frac{\partial C_{total}}{\partial W_1} = -0.00220416 + 0.00016717800000000003$$
$$\frac{\partial C_{total}}{\partial W_1} = -0.002036982$$

Keempat, dengan **Gradient Descent**, kita akan update *value* eksisting dari *weight* $W1$ dengan value baru. Katakanlah **Learning Rate** yang kita gunakan adalah **0.01**.

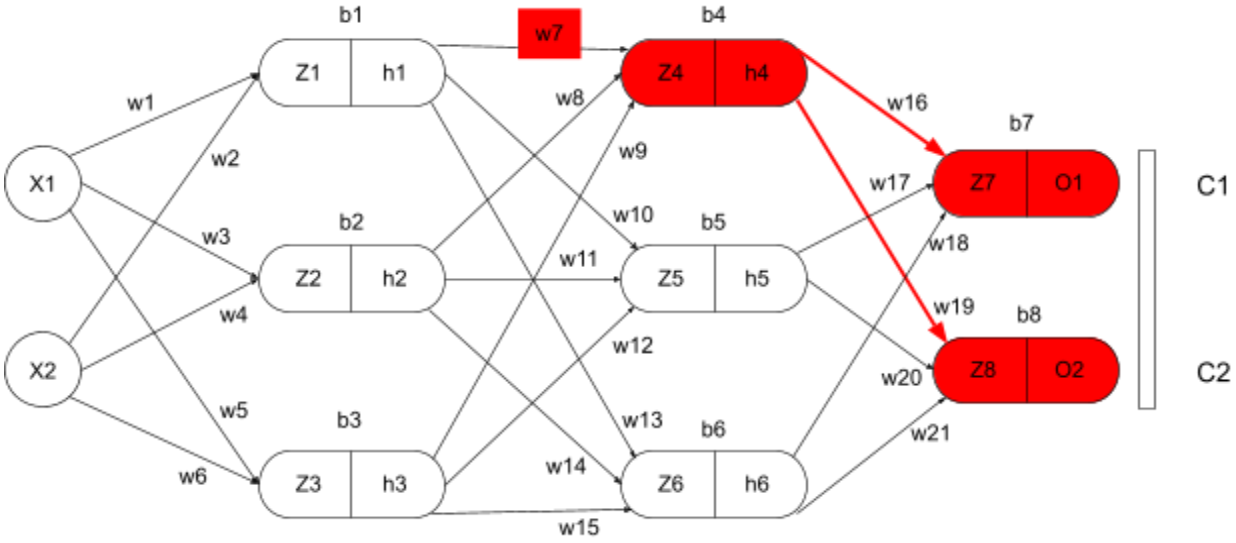
$$W_1 = W_1 - \eta \frac{\partial C_{total}}{\partial w_1}$$
$$W_1 = 0.9 - 0.01 * (-0.002036982)$$
$$W_1 = 0.90002036982$$

Selanjutnya anda bisa mencoba menghitung sendiri sisa weight pada layer 1 $W2 \rightarrow W6$ **dengan step-step yang sama pada** $W1$**.**

Implementasi **Gradient Descent** untuk *weight* $W1$ pada Javascript.

```javascript
let updatedW1 =
          derivativeCostO1 *
            derivativeSigmoid(z7) *
            this.w16 *
            derivativeSigmoid(z4) *
            this.w7 * derivativeSigmoid(z1) * xTrain[0] +

            derivativeCostO1 *
            derivativeSigmoid(z7) *
            this.w17 *
            derivativeSigmoid(z5) *
            this.w10 * derivativeSigmoid(z1) * xTrain[0] +

            derivativeCostO1 *
            derivativeSigmoid(z7) *
            this.w18 *
            derivativeSigmoid(z6) *
            this.w13 * derivativeSigmoid(z1) * xTrain[0] +

            derivativeCostO2 *
            derivativeSigmoid(z8) *
            this.w19 *
            derivativeSigmoid(z4) *
            this.w7 * derivativeSigmoid(z1) * xTrain[0] +

            derivativeCostO2 *
            derivativeSigmoid(z8) *
            this.w20 *
            derivativeSigmoid(z5) *
            this.w10 * derivativeSigmoid(z1) * xTrain[0] +

            derivativeCostO2 *
            derivativeSigmoid(z8) *
            this.w21 *
            derivativeSigmoid(z6) *
            this.w13 * derivativeSigmoid(z1) * xTrain[0];

this.w1 = this.w1 - this.learningRate * updatedW1;
```

Setelah sebelumnya kita menyelesaikan perhitungan **Gradient Descent** untuk $W1$. Selanjutnya kita akan menghitung seberapa berpengaruh $W7$ pada pada **Loss/Error** *row* pertama pada data *training* kita.



Seperti pada $W1$, pertama kita juga akan mencari setiap formula *derivative* dari semua jalur yang masuk ke C_1 dan C_2 dengan mengikuti jalur(*path*) warna merah dan mundur/ *backward* menggunakan *Chain Rule*.

$$\frac{\partial C_1}{\partial W_7} = \frac{\partial C_1}{\partial O_1} * \frac{\partial O_1}{\partial Z_7} * \frac{\partial Z_7}{\partial h_4} * \frac{\partial h_4}{\partial Z_4} * \frac{\partial Z_4}{\partial W_7}$$

$$\frac{\partial C_2}{\partial W_7} = \frac{\partial C_2}{\partial O_2} * \frac{\partial O_2}{\partial Z_8} * \frac{\partial Z_8}{\partial h_4} * \frac{\partial h_4}{\partial Z_4} * \frac{\partial Z_4}{\partial W_7}$$

Untuk menghitung $\frac{\partial C_{total}}{\partial W_7}$, kita jumlahkan $\frac{\partial C_1}{\partial W_7}$ dan $\frac{\partial C_2}{\partial W_7}$.

$$\frac{\partial C_{total}}{\partial W_7} = \frac{\partial C_1}{\partial W_7} + \frac{\partial C_2}{\partial W_7}$$

Kedua, kita menghitung setiap *derivative*(turunan) dari setiap komponen formula diatas. **Beberapa komponen sudah kita temukan pada saat mencari derivative** $W1$ $\frac{\partial C_{total}}{\partial W_1}$**. Sehingga kita hanya perlu mencari sisanya**.

$\frac{\partial Z_4}{\partial W_7} = h1W7 + h2W8 + h3W9 + b4 = 0.18$ (Dengan beberapa derivative rule yang sudah kita pelajari, maka derivative dari $\frac{\partial Z_4}{\partial W_7}$ with respect to $W7$ adalah $h1 = 0.18$. Dalam hal ini coefficient dari $W7$)

Ketiga, kita sudah mendapatkan *derivative*(turunan) dari setiap komponen formula diatas, selanjutnya kita subtitusikan setiap komponen hasil perhitungan ke dalam formula.

$$\frac{\partial C_1}{\partial W_7} = 0.82 * 0.14 * 0.2 * 0.23 * 0.18$$

$$\frac{\partial C_2}{\partial W_7} = -0.22 * 0.17 * 0.4 * 0.23 * 0.18$$

$$\frac{\partial C_{total}}{\partial W_7} = 0.000950544 + (-0.0006193440000000001)$$

$$\frac{\partial C_{total}}{\partial W_7} = 0.0003311999999999999$$

Keempat, dengan **Gradient Descent**, kita akan update *value* eksisting dari *weight* $W7$ dengan *value* baru. Katakanlah **Learning Rate** yang kita gunakan adalah **0.01**.

$$W_7 = W_7 - \eta \frac{\partial C_{total}}{\partial w_7}$$

$$W_7 = 0.0 - 0.01 * 0.0003311999999999999$$

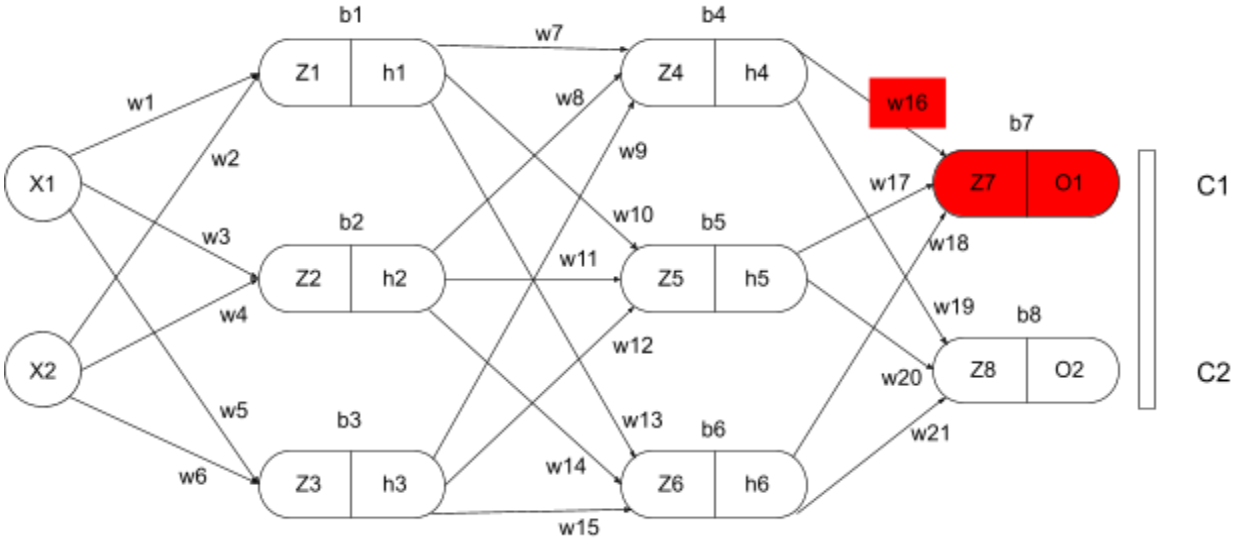
$$W_7 = -0.000003311999999999994$$

Selanjutnya anda bisa mencoba menghitung sendiri sisa weight pada layer 2 $W8 \rightarrow W15$ dengan step-step yang sama pada $W7$.

Implementasi **Gradient Descent** untuk *weight* $W7$ pada Javascript.

```javascript
let updatedW7 =
       derivativeCostO1 *
       derivativeSigmoid(z7) *
       this.w16 *
       derivativeSigmoid(z4) * h1 +
       derivativeCostO2 *
       derivativeSigmoid(z8) *
       this.w19 *
       derivativeSigmoid(z4) * h1;

this.w7 = this.w7 - this.learningRate * updatedW7;
```

Setelah sebelumnya kita menyelesaikan perhitungan **Gradient Descent** untuk $W7$. Selanjutnya kita akan menghitung seberapa berpengaruh $W16$ pada pada **Loss/Error** *row* pertama pada data *training* kita.



Pertama kita juga akan mencari setiap formula *derivative* dari semua jalur yang masuk ke C_1 , sebab weight $W16$ hanya berpengaruh langsung pada C_1 saja. Proses yang dilakukan sama, yaitu dengan mengikuti jalur(*path*) warna merah dan mundur/ *backward* menggunakan *Chain Rule*.

$$\frac{\partial C_1}{\partial W_{16}} = \frac{\partial C_1}{\partial O_1} * \frac{\partial O_1}{\partial Z_7} * \frac{\partial Z_7}{\partial W_{16}}$$

$$\frac{\partial C_{total}}{\partial W_{16}} = \frac{\partial C_1}{\partial W_{16}}$$

Kedua, kita menghitung setiap *derivative*(turunan) dari setiap komponen formula diatas. **Beberapa komponen sudah kita temukan pada saat mencari derivative** $W1$ $\frac{\partial C_{total}}{\partial W_1}$**.**

Sehingga kita hanya perlu mencari sisanya.

$\frac{\partial Z_7}{\partial W_{16}} = h4W16 + h5W17 + h6W18 + b7 = 0.63$ (Dengan beberapa derivative rule yang sudah kita pelajari, maka derivative dari $\frac{\partial Z_7}{\partial W_{16}}$ with respect to $W16$ adalah $h4 = 0.63$. Dalam hal ini coefficient dari $W16$)

Ketiga, kita sudah mendapatkan *derivative*(turunan) dari setiap komponen formula diatas, selanjutnya kita subtitusikan setiap komponen hasil perhitungan ke dalam formula.

$$\frac{\partial C_1}{\partial W_{16}} = 0.82 * 0.14 * 0.63$$

$$\frac{\partial C_{total}}{\partial W_{16}} = 0.072324$$

Keempat, dengan **Gradient Descent**, kita akan update *value* eksisting dari *weight* $W16$ dengan *value* baru. Katakanlah **Learning Rate** yang kita gunakan adalah **0.01**.

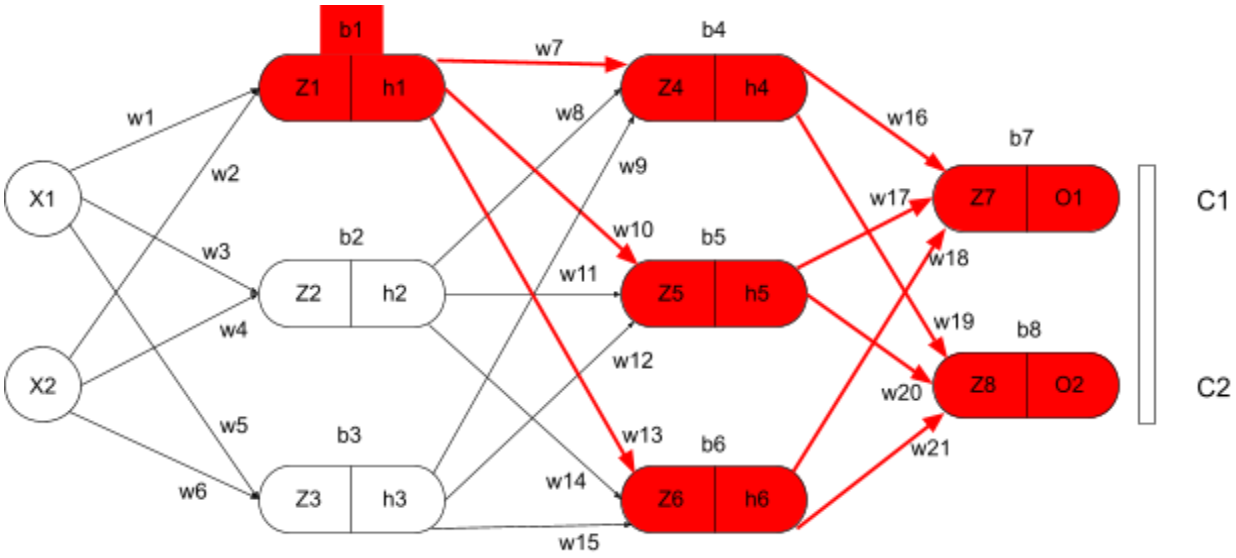
$$W_{16} = W_{16} - \eta \frac{\partial C_{total}}{\partial w_{16}}$$
$$W_{16} = 0.2 - 0.01 * 0.072324$$
$$W_{16} = 0.19927676$$

Selanjutnya anda bisa mencoba menghitung sendiri sisa weight pada layer 3 $W17 \rightarrow W21$ dengan step-step yang sama pada $W16$.

Implementasi **Gradient Descent** untuk *weight* $W16$ pada Javascript.

```javascript
let updatedW16 =
       derivativeCostO1 * derivativeSigmoid(z7) * h4;

this.w16 = this.w16 - this.learningRate * updatedW16;
```

Setelah sebelumnya kita menyelesaikan perhitungan **Gradient Descent** untuk 3 *weight*, yaitu $W1$, $W7$, dan $W16$. Selanjutnya kita akan melakukan perhitungan Gradient Descent terhadap 3 bias pada perwakilan 3 layer yaitu $b1$, $b4$, dan $b7$. Akan kita mulai menghitung seberapa berpengaruh $b1$ pada pada **Loss/Error** *row* pertama pada data *training* kita.



Pertama kita akan mencari setiap formula *derivative* dari semua jalur yang masuk ke C_1 dan C_2 dengan mengikuti jalur(*path*) warna merah dan mundur/ *backward* menggunakan *Chain Rule*.

$$\frac{\partial C_1}{\partial b_1} = \frac{\partial C_1}{\partial O_1} * \frac{\partial O_1}{\partial Z_7} * \frac{\partial Z_7}{\partial h_4} * \frac{\partial h_4}{\partial Z_4} * \frac{\partial Z_4}{\partial h_1} * \frac{\partial h_1}{\partial Z_1} * \frac{\partial Z_1}{\partial b_1} + \frac{\partial C_1}{\partial O_1} * \frac{\partial O_1}{\partial Z_7} * \frac{\partial Z_7}{\partial h_5} * \frac{\partial h_5}{\partial Z_5} * \frac{\partial Z_5}{\partial h_1} * \frac{\partial h_1}{\partial Z_1} * \frac{\partial Z_1}{\partial b_1} + \frac{\partial C_1}{\partial O_1} * \frac{\partial O_1}{\partial Z_7} * \frac{\partial Z_7}{\partial h_6} * \frac{\partial h_6}{\partial Z_6} * \frac{\partial Z_6}{\partial h_1} * \frac{\partial h_1}{\partial Z_1} * \frac{\partial Z_1}{\partial b_1}$$

$$\frac{\partial C_2}{\partial b_1} = \frac{\partial C_2}{\partial O_2} * \frac{\partial O_2}{\partial Z_8} * \frac{\partial Z_8}{\partial h_4} * \frac{\partial h_4}{\partial Z_4} * \frac{\partial Z_4}{\partial h_1} * \frac{\partial h_1}{\partial Z_1} * \frac{\partial Z_1}{\partial b_1} + \frac{\partial C_2}{\partial O_2} * \frac{\partial O_2}{\partial Z_8} * \frac{\partial Z_8}{\partial h_5} * \frac{\partial h_5}{\partial Z_5} * \frac{\partial Z_5}{\partial h_1} * \frac{\partial h_1}{\partial Z_1} * \frac{\partial Z_1}{\partial b_1} + \frac{\partial C_2}{\partial O_2} * \frac{\partial O_2}{\partial Z_8} * \frac{\partial Z_8}{\partial h_6} * \frac{\partial h_6}{\partial Z_6} * \frac{\partial Z_6}{\partial h_1} * \frac{\partial h_1}{\partial Z_1} * \frac{\partial Z_1}{\partial b_1}$$

Untuk menghitung $\frac{\partial C_{total}}{\partial b_1}$, kita jumlahkan $\frac{\partial C_1}{\partial b_1}$ dan $\frac{\partial C_2}{\partial b_1}$.

$$\frac{\partial C_{total}}{\partial b_1} = \frac{\partial C_1}{\partial b_1} + \frac{\partial C_2}{\partial b_1}$$

Kedua, kita menghitung setiap *derivative*(turunan) dari setiap komponen formula diatas. **Beberapa komponen sudah kita temukan pada saat mencari derivative** $W1\ \frac{\partial C_{total}}{\partial W_1}$**. Sehingga kita hanya perlu mencari sisanya**.

$\frac{\partial Z_1}{\partial b_1} = X1W1 + X2W2 + b1 = 1$ (Dengan beberapa derivative rule yang sudah kita pelajari, maka derivative dari $\frac{\partial Z_1}{\partial b_1}$ with respect to $b1$ adalah 1. Dalam hal ini coefficient dari variabel $b1$. Sebab variabel $b1$ juga mempunyai bentuk $b1 * 1$).

Ketiga, kita sudah mendapatkan *derivative*(turunan) dari setiap komponen formula diatas, selanjutnya kita subtitusikan setiap komponen hasil perhitungan ke dalam formula.

$$\frac{\partial C_1}{\partial b_1} = 0.82 * 0.14 * 0.2 * 0.23 * 0.0 * 0.15 * 1 + 0.82 * 0.14 * 0.8 * 0.19 * 0.3 * 0.15 * 1 + 0.82 * 0.14 * 0.8 * 0.23 * 0.1 * 0.15 * 1$$

$$\frac{\partial C_2}{\partial b_1} = -0.22 * 0.17 * 0.4 * 0.23 * 0.0 * 0.15 * 1 - 0.22 * 0.17 * 0.1 * 0.19 * 0.3 * 0.15 * 1 - 0.22 * 0.17 * 0.4 * 0.23 * 0.1 * 0.15 * 1$$

$$\frac{\partial C_{total}}{\partial b_1} = 0.00110208 + (-0.00008358900000000002)$$

$$\frac{\partial C_{total}}{\partial b_1} = 0.001018491$$

Keempat, dengan **Gradient Descent**, kita akan update *value* eksisting dari *weight* $b1$ dengan value baru. Katakanlah **Learning Rate** yang kita gunakan adalah **0.01**.

$$b_1 = b_1 - \eta \frac{\partial C_{total}}{\partial b_1}$$

$$b_1 = 0.9 - 0.01 * 0.001018491$$

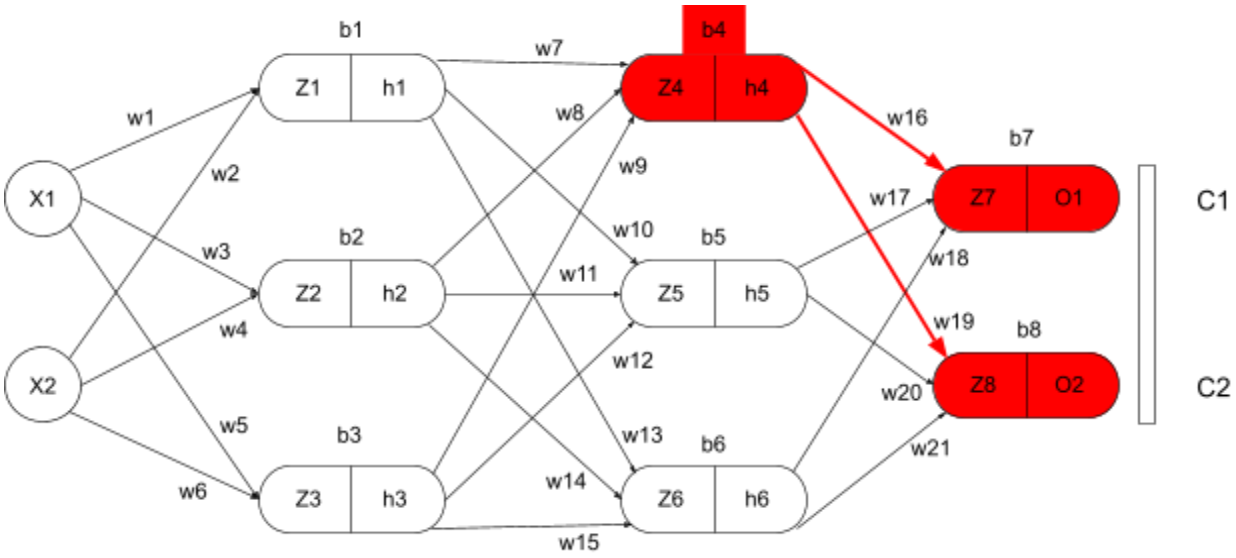
$$b_1 = 0.89998981509$$

Selanjutnya anda bisa mencoba menghitung sendiri sisa bias pada layer 1 $b2 \rightarrow b3$ dengan step-step yang sama pada $b1$.

Implementasi **Gradient Descent** untuk *bias* $b1$ pada Javascript.

```javascript
let updatedB1 =
          derivativeCostO1 *
            derivativeSigmoid(z7) *
            this.w16 *
            derivativeSigmoid(z4) *
            this.w7 * derivativeSigmoid(z1) * 1 +

            derivativeCostO1 *
            derivativeSigmoid(z7) *
            this.w17 *
            derivativeSigmoid(z5) *
            this.w10 * derivativeSigmoid(z1) * 1 +

            derivativeCostO1 *
            derivativeSigmoid(z7) *
            this.w18 *
            derivativeSigmoid(z6) *
            this.w13 * derivativeSigmoid(z1) * 1 +

            derivativeCostO2 *
            derivativeSigmoid(z8) *
            this.w19 *
            derivativeSigmoid(z4) *
            this.w7 * derivativeSigmoid(z1) * 1 +

            derivativeCostO2 *
            derivativeSigmoid(z8) *
            this.w20 *
            derivativeSigmoid(z5) *
            this.w10 * derivativeSigmoid(z1) * 1 +

            derivativeCostO2 *
            derivativeSigmoid(z8) *
            this.w21 *
            derivativeSigmoid(z6) *
            this.w13 * derivativeSigmoid(z1) * 1;
this.b1 = this.b1 - this.learningRate * updatedB1;
```

Setelah sebelumnya kita menyelesaikan perhitungan **Gradient Descent** untuk $b1$. Selanjutnya kita akan menghitung seberapa berpengaruh $b4$ pada pada **Loss/Error** *row* pertama pada data *training* kita.



Seperti pada $b1$, pertama kita juga akan mencari setiap formula *derivative* dari semua jalur yang masuk ke C_1 dan C_2 dengan mengikuti jalur(*path*) warna merah dan mundur/ *backward* menggunakan *Chain Rule*.

$$\frac{\partial C_1}{\partial b_4} = \frac{\partial C_1}{\partial O_1} * \frac{\partial O_1}{\partial Z_7} * \frac{\partial Z_7}{\partial h_4} * \frac{\partial h_4}{\partial Z_4} * \frac{\partial Z_4}{\partial b_4}$$

$$\frac{\partial C_2}{\partial b_4} = \frac{\partial C_2}{\partial O_2} * \frac{\partial O_2}{\partial Z_8} * \frac{\partial Z_8}{\partial h_4} * \frac{\partial h_4}{\partial Z_4} * \frac{\partial Z_4}{\partial b_4}$$

Untuk menghitung $\frac{\partial C_{total}}{\partial b_4}$, kita jumlahkan $\frac{\partial C_1}{\partial b_4}$ dan $\frac{\partial C_2}{\partial b_4}$.

$$\frac{\partial C_{total}}{\partial b_4} = \frac{\partial C_1}{\partial b_4} + \frac{\partial C_2}{\partial b_4}$$

Kedua, kita menghitung setiap *derivative*(turunan) dari setiap komponen formula diatas. **Beberapa komponen sudah kita temukan pada saat mencari derivative** $W1$ $\frac{\partial C_{total}}{\partial W_1}$**. Sehingga kita hanya perlu mencari sisanya**.

$\frac{\partial Z_4}{\partial b_4} = h1W7 + h2W8 + h3W9 + b4 = 1$ (Dengan beberapa derivative rule yang sudah kita pelajari, maka derivative dari $\frac{\partial Z_4}{\partial b_4}$ with respect to $b4$ adalah 1. Dalam hal ini coefficient dari variabel $b4$. Sebab variabel $b4$ juga mempunyai bentuk $b4 * 1$).

Ketiga, kita sudah mendapatkan *derivative*(turunan) dari setiap komponen formula diatas, selanjutnya kita subtitusikan setiap komponen hasil perhitungan ke dalam formula.

$$\frac{\partial C_1}{\partial b_4} = 0.82 * 0.14 * 0.2 * 0.23 * 1$$

$$\frac{\partial C_2}{\partial b_4} = -0.22 * 0.17 * 0.4 * 0.23 * 1$$

$$\frac{\partial C_{total}}{\partial b_4} = 0.0052808 + (-0.0034408000000000004)$$

$$\frac{\partial C_{total}}{\partial b_4} = 0.00184$$

Keempat, dengan **Gradient Descent**, kita akan update *value* eksisting dari *bias* $b4$ dengan *value* baru. Katakanlah **Learning Rate** yang kita gunakan adalah **0.01**.

$$b_4 = b_4 - \eta \frac{\partial C_{total}}{\partial w_4}$$

$$b_4 = 0.2 - 0.01 * 0.00184$$

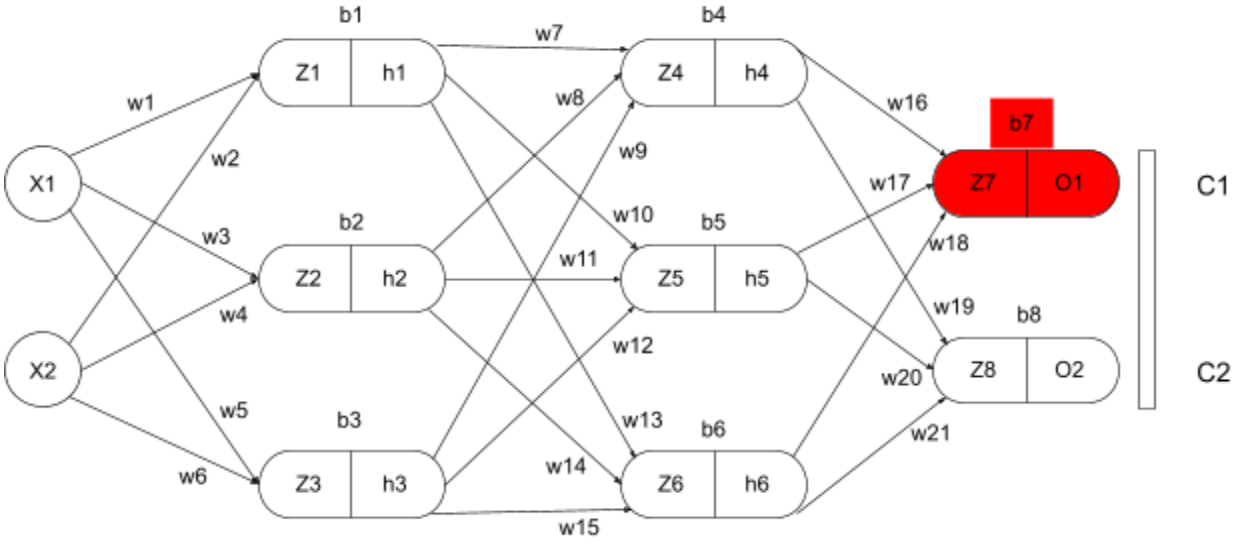
$$b_4 = 0.1999816$$

Selanjutnya anda bisa mencoba menghitung sendiri sisa bias pada layer 2 $b5 \rightarrow b6$ dengan step-step yang sama pada $b4$.

Implementasi **Gradient Descent** untuk *bias* $b4$ pada Javascript.

```javascript
let updatedB4 =
       derivativeCostO1 *
       derivativeSigmoid(z7) *
       this.w16 *
       derivativeSigmoid(z4) * 1 +
       derivativeCostO2 *
       derivativeSigmoid(z8) *
       this.w19 *
       derivativeSigmoid(z4) * 1;

this.b4 = this.b4 - this.learningRate * updatedB4;
```

Setelah sebelumnya kita menyelesaikan perhitungan **Gradient Descent** untuk $b4$. Selanjutnya kita akan menghitung seberapa berpengaruh $b7$ pada pada **Loss/Error** *row* pertama pada data *training* kita.



Pertama kita juga akan mencari setiap formula *derivative* dari semua jalur yang masuk ke C_1 , sebab weight $b7$ hanya berpengaruh langsung pada C_1 saja. Proses yang dilakukan sama, yaitu dengan mengikuti jalur(*path*) warna merah dan mundur/ *backward* menggunakan *Chain Rule*.

$$\frac{\partial C_1}{\partial b_7} = \frac{\partial C_1}{\partial O_1} * \frac{\partial O_1}{\partial Z_7} * \frac{\partial Z_7}{\partial b_7}$$

$$\frac{\partial C_{total}}{\partial b_7} = \frac{\partial C_1}{\partial b_7}$$

Kedua, kita menghitung setiap *derivative*(turunan) dari setiap komponen formula diatas. **Beberapa komponen sudah kita temukan pada saat mencari derivative** $W1$ $\frac{\partial C_{total}}{\partial W_1}$**.**

Sehingga kita hanya perlu mencari sisanya.

$\frac{\partial Z_7}{\partial b_7} = h4W16 + h5W17 + h6W18 + b7 = 1$ (Dengan beberapa derivative rule yang sudah kita pelajari, maka derivative dari $\frac{\partial Z_7}{\partial b_7}$ with respect to $b7$ adalah 1. Dalam hal ini coefficient dari variabel $b7$. Sebab variabel $b7$ juga mempunyai bentuk $b7 * 1$).

Ketiga, kita sudah mendapatkan *derivative*(turunan) dari setiap komponen formula diatas, selanjutnya kita subtitusikan setiap komponen hasil perhitungan ke dalam formula.

$$\frac{\partial C_1}{\partial b_7} = 0.82 * 0.14 * 1$$

$$\frac{\partial C_{total}}{\partial b_7} = 0.1148$$

Keempat, dengan **Gradient Descent**, kita akan update *value* eksisting dari *bias* $b7$ dengan *value* baru. Katakanlah **Learning Rate** yang kita gunakan adalah **0.01**.

$$b_7 = b_7 - \eta \frac{\partial C_{total}}{\partial b_7}$$
$$b_7 = 0.3 - 0.01 * 0.1148$$
$$b_7 = 0.298852$$

Selanjutnya anda bisa mencoba menghitung sendiri sisa weight pada layer 3 $b8$ dengan step-step yang sama pada $b7$.

Implementasi **Gradient Descent** untuk *bias* $b7$ pada Javascript.

```javascript
let updatedB7 = derivativeCostO1 * derivativeSigmoid(z7) * 1;
this.b7 = this.b7 - this.learningRate * updatedB7;
```

Training dan Inference contoh Permasalahan: Menentukan siswa lulus atau tidak dari nilai yang diperoleh.

```javascript
const n = new NNetwork(1000, 0.01);
n.train(xTrains, yTrains);

const labels = ['Lulus', 'Tidak Lulus'];
const r = n.forward([112, 22]);
console.log(r);
console.log(labels[argmax(r)]);
```

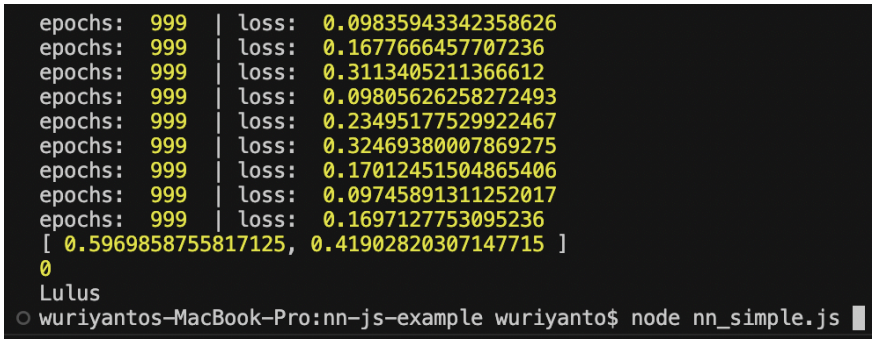
Pada constructor **NNetwork** kita set **Epochs = 1000**, dan **Learning Rate = 0.01**. Anda bisa mencoba bereksperimen dengan **Epoch** dan **Learning Rate** sesuai keinginan.

Selanjutnya pada *method forward*, kita set nilai Matematika dan Bahasa Inggris masing-masing 112 dan 22.

```javascript
const r = n.forward([112, 22]);
```

Anda bisa mencoba sendiri *code* untuk contoh pertama dari ANN yang kita bangun pada repository Github berikut https://github.com/wuriyanto48/nn-js-example/blob/main/nn_simple.js. Jalankan code tersebut.

```unset
$ node nn_simple.js
```



Percobaan kedua dengan nilai yang lebih kecil.

```javascript
const n = new NNetwork(1000, 0.01);
n.train(xTrains, yTrains);

const labels = ['Lulus', 'Tidak Lulus'];
const r = n.forward([-2, -10]);
console.log(r);
console.log(labels[argmax(r)]);
```

Selanjutnya pada *method forward*, kita set nilai Matematika dan Bahasa Inggris masing-masing -2 dan -10.

```javascript
const r = n.forward([-2, -10]);
```

Jalankan code tersebut.

```
$ node nn_simple.js
```

```
epochs:  999  | loss:  0.13152387325830195
epochs:  999  | loss:  0.1912981520231309
epochs:  999  | loss:  0.28677608416453326
epochs:  999  | loss:  0.13103045744000463
epochs:  999  | loss:  0.2531986245922917
epochs:  999  | loss:  0.29300986276701335
epochs:  999  | loss:  0.1938676253455853
epochs:  999  | loss:  0.13076011820958383
epochs:  999  | loss:  0.1934171484165717
[ 0.37886815493849235, 0.6176851301494568 ]
1
Tidak Lulus
wuriyantos-MacBook-Pro:nn-js-example wuriyanto$ node nn_simple.js
```

Hasil sesuai dengan ekspektasi. **Artificial Neural Network (ANN)** yang kita bangun bisa mengklasifikasi data input baru, meskipun data training yang kita berikan hanya berjumlah sedikit secara kuantitas.

Mini project Handwritten Digits Recognition dengan MNIST Datasets

Setelah mempelajari bagaimana kita bisa mengoptimasi *function*, *Learnable parameters* dengan *Derivative* (Turunan), arsitektur **Artificial Neural Network**, proses *training* (**Backpropagation**, **Gradient Descent**) dan mempelajari Linear Algebra untuk mengabstraksi data dengan Matrix dan Vector. Kali ini kita akan membangun mini projek yang lebih menarik dari sebelumnya.

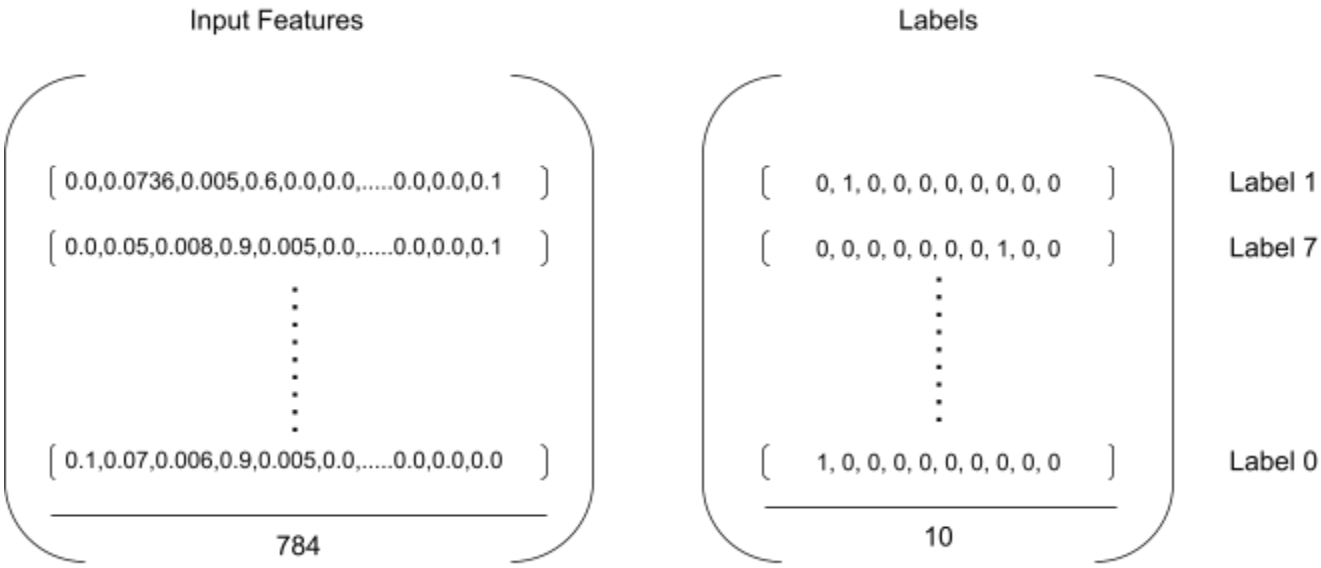
MNIST adalah kumpulan dataset berupa **28x28 *pixel*** gambar angka yang ditulis manual dengan tangan manusia. Dataset ini memungkinkan kita melatih **Artificial Neural Network** yang bisa mengenali tulisan digit/angka yang ditulis oleh manusia.

Dataset-nya sendiri berbentuk kumpulan *Array* 1 dimensi, dengan pasangan *One Hot Encoding labels* dengan *pixel* data dengan panjang 28X28 = 784. *Dataset* ini disimpan dalam data *CSV* yang sebelumnya sudah di *preprocess* oleh penulis.

```
Unset
"0,0,0,0,0,1,0,0,0,0","0.0,0.00736,0.005,1.230138e-05,0.0,0.0,.....0.0,0.0,0.1"
```

Panjang Label = 10
Panjang Data = 784

Data Encoding



Setelah *dataset* dibaca dari file **CSV**, setiap satuan data akan dikonversi menjadi tipe data *float*.

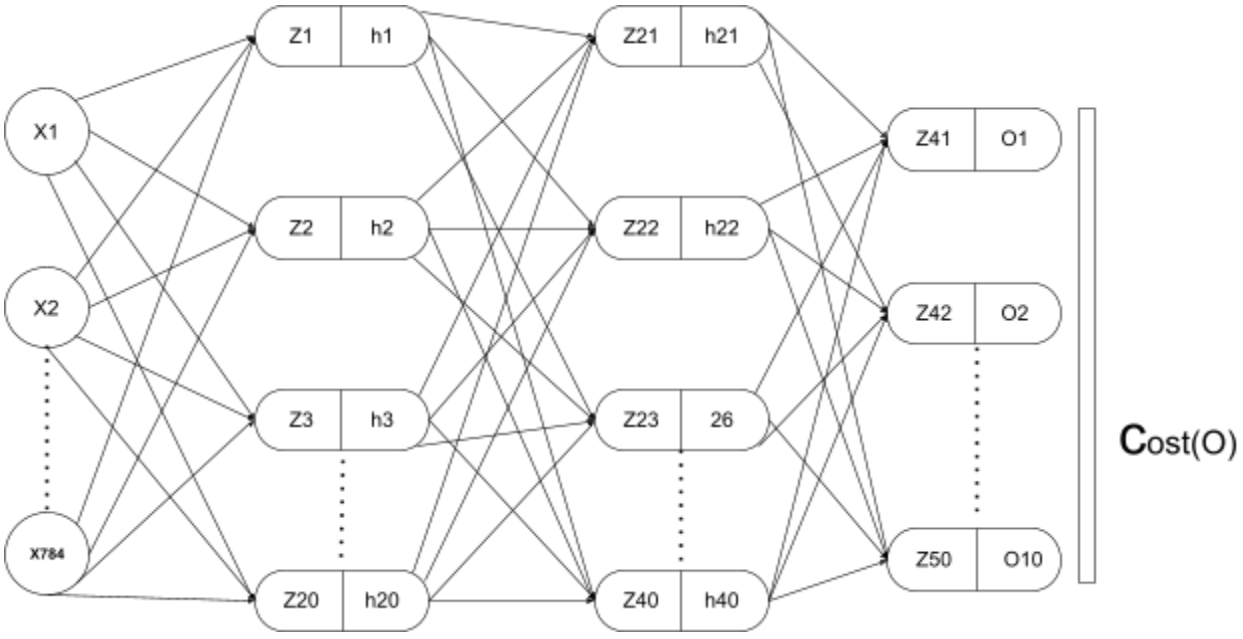
```JavaScript
function main() {

  let labels = [];
  let datas = [];
  fs.createReadStream('mnist.csv')
    .pipe(parse({delimiter: ','}))
    .on('data', (row) => {
      let labelStr = row[0];
      let dataStr = row[1];

      let label = labelStr.split(',').map((v) => parseFloat(v));
      let data = dataStr.split(',').map((v) => parseFloat(v));

      labels.push(label);
      datas.push(data);
```

Arsitektur Network

Arsitektur Network yang kita bangun mirip dengan Arsitektur Network pada contoh sebelumnya, ada 4 *layer*. *Input Layer,* 2 *Hidden Layer*, dan *Output Layer*. Bedanya kali ini kita akan menambahkan jumlah *Node* atau *Neuron*-nya. Jumlah *input Neuron* = 784, sesuai dengan panjang dimensi *dataset* **MNIST** yang sudah disiapkan. Walaupun data yang akan kita proses adalah data gambar, kita tidak menggunakan algoritma varian khusus seperti **CNN(Convolutional Neural Network)**. Yang kita gunakan adalah algoritma original, atau yang biasa disebut dengan **MLP(Multi Layer Perceptron)**.

Network Detail:

- *Input layer* = 784
- *Hidden Layer* 1 = 20
- *Hidden Layer* 2 = 20
- *Output Layer* = 10



Weight dan Bias

Pada contoh sebelumnya, inisiasi *Learnable parameter*(*weight* dan *bias*) kita *hardcode* sejumlah parameter yang dibutuhkan. Kali ini kita akan gunakan *for-loop* supaya lebih *clean*.

JavaScript

```
      .
      .
    this.ws1 = [];
    this.ws2 = [];
    this.ws3 = [];
    this.bs1 = [];
    this.bs2 = [];
    this.bs3 = [];

    for (let i = 0; i < 20; i++) {
      let ws = [];
      for (let j = 0; j < 784; j++) {
        let w = Math.random();
        w = w / Math.pow(784, 0.5);
        ws.push(w);
      }

      this.ws1.push(ws);
    }
      .
      .
      .
      .

    for (let i = 0; i < 10; i++) {
      let ws = [];
      for (let j = 0; j < 20; j++) {
        let w = Math.random();
        w = w / Math.pow(20, 0.5);
        ws.push(w);
      }

      this.ws3.push(ws);
    }
```

Forward Propagation

Semua operasi **Forward Propagation** untuk proses *training* maupun *inference* berada pada *method forward*.

```javascript
forward(x) {
    // first hidden layer
    let zh1 = [];
    for (let i = 0; i < this.ws1.length; i++) {
      let ws = this.ws1[i];
      let b = this.bs1[i];
      let z = linear(x, ws, b);
      let h = sigmoid(z);

      zh1.push({z: z, h: h});
    }

    // second hidden layer
    let zh2 = [];
    for (let i = 0; i < this.ws2.length; i++) {
      let ws = this.ws2[i];
      let b = this.bs2[i];
      let z = linearOnObj(zh1, ws, b);
      let h = sigmoid(z);

      zh2.push({z: z, h: h});
    }

    // output layer
    let zh3 = [];
    for (let i = 0; i < this.ws3.length; i++) {
      let ws = this.ws3[i];
      let b = this.bs3[i];
      let z = linearOnObj(zh2, ws, b);
      let h = sigmoid(z);

      zh3.push({z: z, h: h});
    }

    return [zh1, zh2, zh3];
  }
```

Backpropagation dan Gradient Descent

Pada contoh sebelumnya proses **Gradient Descent** di *hardcode* untuk setiap *Learnable Parameter* (*weight* dan *bias*). Supaya lebih *clean* dan dinamis, kali ini kita akan *refactor code*-nya menggunakan *for-loop*.

JavaScript

```
.
.
.
let derivativeCosts = [];
 for (let i = 0; i < zh3.length; i++) {
   let yt = yTrain[i];
   let h = zh3[i].h;
   let derivativeCost = calculateDerivativeCost(h, yt, yTrain);

   derivativeCosts.push(derivativeCost);
 }

 for (let i = derivativeCosts.length-1; i >= 0; i--) {
   let dc = derivativeCosts[i];
   let ws = this.ws3[i];

   let zh = zh3[i];
   let updatedBias = 0;
   for (let k = ws.length-1; k >= 0; k--) {
     let zh2Temp = zh2[k];
     let updatedWeight = dc * derivativeSigmoid(zh.z) * zh2Temp.h;
     this.ws3[i][k] = this.ws3[i][k] - this.learningRate * updatedWeight;
     if (k == 0) {
       updatedBias = dc * derivativeSigmoid(zh.z) * 1;
     }
   }

   this.bs3[i] = this.bs3[i] - this.learningRate * updatedBias;
 }
.
.
.
```

Penyimpanan Model pada JSON file

Setelah **Artificial Neural Network** selesai melakukan proses training, *Learnable Parameter* (*weight* dan *bias*) yang sudah teroptimasi akan disimpan dalam **JSON** *file*.

```javascript
saveModel(name) {
    if (!name) {
      name = 'model.json';
    }

    const model = {
      layer1: {
        weights: this.ws1,
        biases: this.bs1
      },
      layer2: {
        weights: this.ws2,
        biases: this.bs2
      },
      layer3: {
        weights: this.ws3,
        biases: this.bs3
      }
    };

    const curDir = process.cwd();
    fs.writeFile(path.join(curDir, name), JSON.stringify(model), 'utf8', (err) => {
      if (err) {
        console.log(err);
        return;
      }

      console.log('model saved');
    })
  }
```

File **"model.json"** nantinya akan digunakan untuk proses *inference*. *File* tersebut mempunyai 3 properti utama yaitu layer2, layer2, dan layer3. Ketiga layer tersebut menyimpan semua *Learnable Parameter(weight dan bias)* yang sudah teroptimasi.

Loading Model

Model yang sudah teroptimasi dan tersimpan dalam sebuah file nantinya akan digunakan untuk kebutuhan *inference*. Dalam hal ini digunakan untuk memprediksi *input* data berupa *pixel* data dari tulisan tangan manusia yang masuk ke dalam *Network*-nya. Untuk kebutuhan ini akan kita siapkan dua *method* untuk *serving* dan *loading model*.

Method **loadModel** yang pertama ini untuk kebutuhan *non browser*.

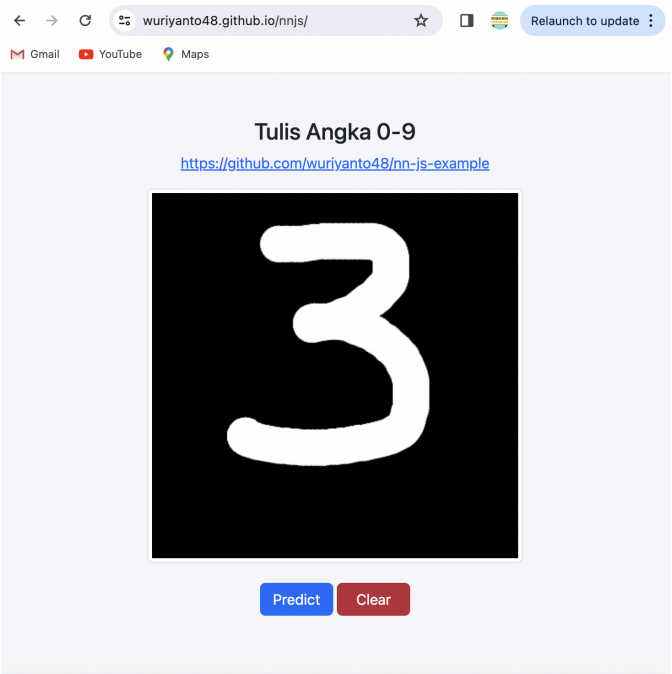
```
JavaScript
  loadModel(name) {
    let model;
    try {
      const data = fs.readFileSync(name);
      model = JSON.parse(data);
    } catch(e) {
      console.log('open model file error ', err);
      return;
    }

    this.ws1 = model.layer1.weights;
    this.bs1 = model.layer1.biases;

    this.ws2 = model.layer2.weights;
    this.bs2 = model.layer2.biases;

    this.ws3 = model.layer3.weights;
    this.bs3 = model.layer3.biases;
  }
}
```

Sedangkan **loadModel** yang kedua ini khusus untuk kebutuhan *serving model* pada *Web browser*.

```javascript
loadModel(name) {
  fetch(name)
    .then((response) => response.json())
    .then((model) => {

      this.ws1 = model.layer1.weights;
      this.bs1 = model.layer1.biases;

      this.ws2 = model.layer2.weights;
      this.bs2 = model.layer2.biases;

      this.ws3 = model.layer3.weights;
      this.bs3 = model.layer3.biases;
    }).catch(e => console.log('open model file error ', e));
}
```

User Input Encoding

Supaya pengalaman pengguna lebih bagus, kita akan menyediakan **HTML5 Canvas** seluas **400x400** *pixel* untuk menerima *input* dari *user*/pengguna.



Data dari **Canvas** dengan luas **400x400** *pixel* tidak bisa langsung kita inputkan ke dalam **ANN Model** kita. Input data tersebut perlu kita preprocess terlebih dahulu sebelum masuk dan dievaluasi oleh **ANN Model** kita.

Kita tambahkan *Canvas virtual* untuk menampung hasil konversi dari **400x400** ke **28x28** *pixel*.

```javascript
function createDrawingCanvas() {
  tempCanvas = document.createElement("canvas");
  tempCanvas.id = "tempCanvas";
  // Set width and height
  tempCanvas.width = 28;
  tempCanvas.height = 28;

  ctxTemp = tempCanvas.getContext("2d");
}
```

Langkah selanjutnya yang dilakukan adalah menangkap *event* dari *user*. Ketika *user* menggerakan *mouse*, menekan *mouse* dan *release* mouse.

```javascript
var drag = false;
.
.
.
function drawCircle(dx, dy) {
  var bounding = canvas.getBoundingClientRect();
  ctx.fillStyle = colorDrawDefault;
  ctx.beginPath();
  ctx.arc(dx-bounding.left, dy-bounding.top, 20, 0, 2 * Math.PI);
  ctx.fill();
}

canvas.addEventListener('mousedown', function(event) {
  drag = true;
});

canvas.addEventListener('mouseup', function(event) {
  drag = false;
});

canvas.addEventListener('mousemove', function(event) {
  var x = event.clientX;
  var y = event.clientY;

  if (drag) {
    drawCircle(x, y);
  }
});
```

Kita siapkan juga untuk kebutuhan menangkap *event* dari *user*/pengguna yang menggunakan *Mobile device*.

```javascript
canvas.addEventListener('touchstart', function(event) {
  if (event.target == canvas) {
    event.preventDefault();
  }

  drag = true;
}, false);

canvas.addEventListener('touchend', function(event) {
  if (event.target == canvas) {
    event.preventDefault();
  }

  drag = false;
}, false);

canvas.addEventListener('touchmove', function(event) {
  if (event.target == canvas) {
    event.preventDefault();
  }

  var touch = event.touches[0];
  var x = touch.clientX;
  var y = touch.clientY;

  if (drag) {
    drawCircle(x, y);
  }
}, false);
```

Langkah selanjutnya adalah menyimpan **Canvas** data dari **Canvas** utama **400x400** ke **Canvas** *virtual* **28x28**.

```javascript
ctxTemp.drawImage(canvas, 0, 0, tempCanvas.width, tempCanvas.height);
var imageData = ctxTemp.getImageData(0, 0, tempCanvas.width, tempCanvas.height);
var pixelData = imageData.data;
```

Variabel **pixelData** sebenarnya berformat **RGBA**. Dengan dimensi *Canvas virtual* yang kita siapkan adalah **28x28** dan mempunyai format **RGBA**, maka total panjang dari **pixelData** adalah **28x28x4 = 3136**. *Input* dari **Artificial Neural Network** yang kita bangun membutuhkan format *Array* satu dimensi dengan panjang **784**. Sehingga **pixelData** perlu kita *preprocess* terlebih dahulu sebelum masuk ke proses *inference*.

Inisiasi variabel **color** dari *instance* **Float32Array** dengan panjang **28x28**.

```
JavaScript
var color = new Float32Array(tempCanvas.width * tempCanvas.height);
```

Selanjutnya kita ambil *value* yang paling besar dari **4** *value* yang ada di **RGBA pixelData**. Sehingga kita hanya mengambil satu *value* dari data **RGBA**. Pada dasarnya potongan *code* di bawah ini akan mengubah **RGBA** menjadi *grayscale*.

```
JavaScript
var j = 0;
for(let i = 0; i < pixelData.length; i = i + 4) {
    color[j] = Math.max(pixelData[i], pixelData[i+1], pixelData[i+2])/255.0;
    j++;
}
```

Selanjutnya variabel **color** yang sudah di *preprocess* masuk ke proses *inference*. Sebelumnya kita konversi terlebih dahulu dari **Float32Array** menjadi *Array*.

```
JavaScript
var layers = n.forward(Array.from(color));
var r = layers[layers.length-1];
console.log(r);
console.log(argmax(r));
```

Keseluruhan *code* dari fungsi **predict**.

```javascript
function predict() {
  ctxTemp.drawImage(canvas, 0, 0, tempCanvas.width, tempCanvas.height);

  var imageData = ctxTemp.getImageData(0, 0, tempCanvas.width, tempCanvas.height);
  var pixelData = imageData.data;

  var color = new Float32Array(tempCanvas.width * tempCanvas.height);

  var j = 0;
  for(let i = 0; i < pixelData.length; i = i + 4) {
    color[j] = Math.max(pixelData[i], pixelData[i+1], pixelData[i+2])/255.0;
    j++;
  }

  var layers = n.forward(Array.from(color));
  var r = layers[layers.length-1];
  console.log(r);
  console.log(argmax(r));

  var labels = ['Nol', 'Satu', 'Dua', 'Tiga', 'Empat', 'Lima', 'Enam', 'Tujuh',
'Delapan', 'Sembilan'];

  document.getElementById('textResult').innerText = labels[argmax(r)];
  document.getElementById('resultAcc').innerText = 'akurasi: ' + (r[argmax(r)].h *
100).toFixed(2) + '%';
}
```

Training

Untuk proses *training* kita perlu menyiapkan beberapa hal.

- Clone keseluruhan Repository Code https://github.com/wuriyanto48/nn-js-example
- Download training data https://drive.google.com/file/d/1t_4XPqk4p0-DBRTQV7BUSjK-kxKOKCte/view?usp=sharing

Setelah semua lengkap, kita langsung bisa menjalankan *file* https://github.com/wuriyanto48/nn-js-example/blob/main/nn_mnist_two_hidden_layer.js.

```
Unset
node nn_mnist_two_hidden_layer.js
```

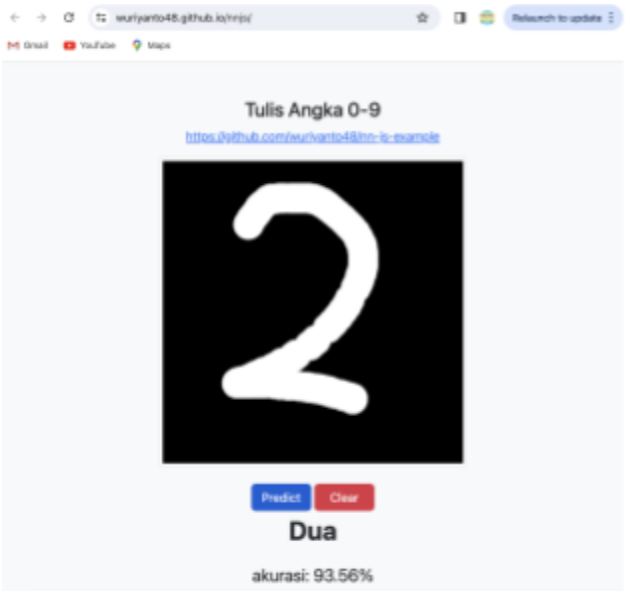
Catatan: Proses *training* membutuhkan waktu yang cukup lama.

Setelah proses *training* selesai, salin *file* **"model.json"** ke dalam folder **public**. Jalankan **HTTP** *server*-nya dan kunjungi alamat http://localhost:3000/ atau anda juga bisa mengunjungi https://wuriyanto48.github.io/nnjs/.

Menjalankan **HTTP** *Server*

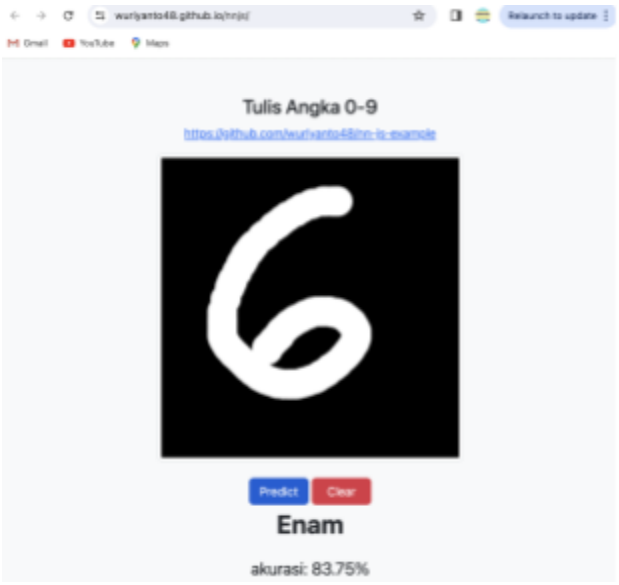
```
Unset
npm start
```

Screenshot hasil percobaan:

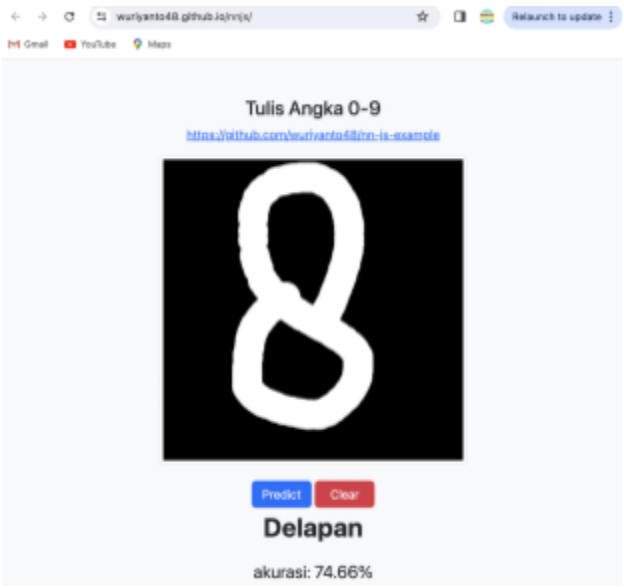



Tulis Angka 0-9
Predict
Clear
Empat
akurasi: 80.30%


Tulis Angka 0-9
Predict
Clear
Lima
akurasi: 84.74%


Tulis Angka 0-9
Predict
Clear
Enam
akurasi: 83.75%

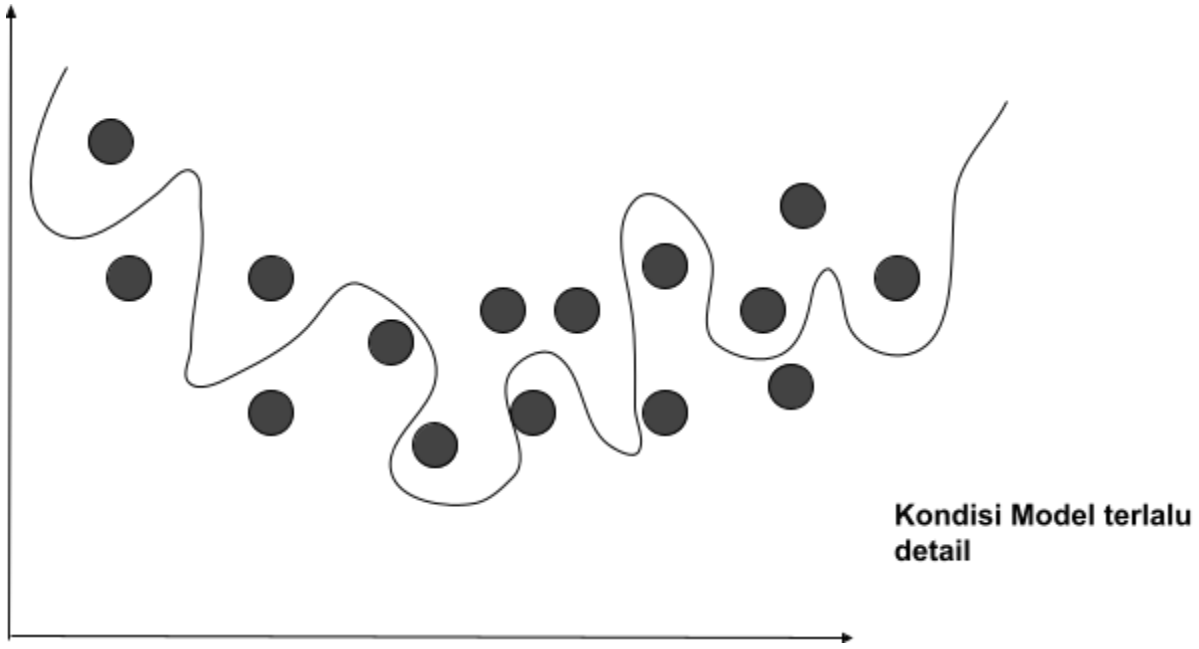

Tulis Angka 0-9
Predict
Clear
Tujuh
akurasi: 91.09%



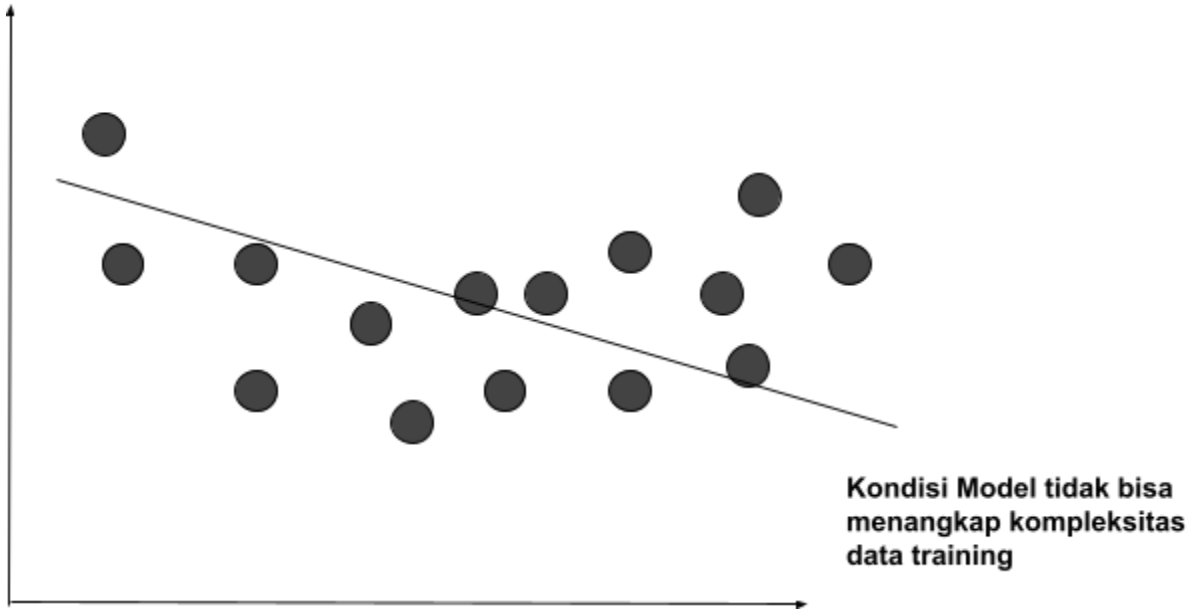
Catatan: model belum bisa menangkap perubahan perspektif dari tulisan angka yang ditulis oleh *user,* misalnya: model belum bisa memprediksi angka 7 yang terbalik . Sebab, dataset yang tersedia belum menangani sampai problem semacam ini. Untuk menambah keakuratan prediksi, kita bisa coba menaikan **Epoch** sedikit demi sedikit.

Overfitting dan Underfitting

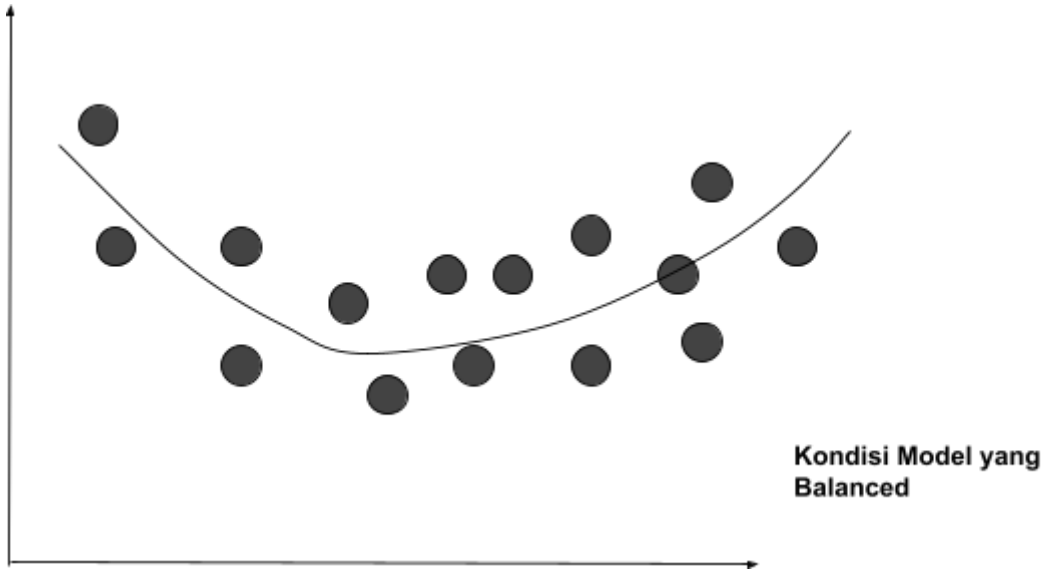
Overfitting adalah kondisi dimana *model* terlalu detail terhadap data *training* yang ada. Sehingga ketika ada data baru (*real* data), *model* tidak bisa memprediksi secara akurat. **Overfitting** terjadi disebabkan oleh beberapa hal, misalnya data *training* terlalu sedikit, **Epoch** terlalu tinggi, dan *model* terlalu kompleks. Ada beberapa hal untuk mengurangi **Overfitting**, misalnya dengan memperbanyak data *training*, memilih *input features* yang relevan, dan mengurangi kompleksitas model(misalnya mengurangi *hidden layer*).



Underfitting adalah kondisi dimana *model* tidak bisa menggeneralisasi data *training*. Sehingga ketika ada data baru (*real* data), *model* tidak bisa memprediksi data secara akurat. **Underfitting** terjadi disebabkan beberapa hal, misalnya *model* terlalu sederhana sehingga tidak bisa menangkap kompleksitas data *training*, **Epoch** terlalu rendah. Ada beberapa hal untuk mencegah **Underfitting**, misalnya dengan memperbanyak *hidden layer*, menambah *Learnable Parameters(weight & bias)*, menaikan **Epoch** (*learning loop*).



Kondisi Model yang ideal dan diharapkan (Balanced)



Kamus Istilah

- **Derivative (Turunan)** salah satu *core* **Calculus** yang berfokus mengamati tingkat perubahan (*Rate of Change*).
- **Derivative of a function** $f(x)$ (Turunan fungsi) adalah tingkat perubahan (*rate of change*) dari fungsi $f(x)$.
- **Linear Algebra** adalah cabang ilmu Matematika yang berfokus pada **Vector**, **Matrix** dan operasinya.
- **Cancel**: mengeliminasi sebuah value dengan cara mengurangi dan membagi dengan *common term*(*term* yang sama). Contoh $x + y = x + 4$ menjadi $y = 4$, *term* x hilang, karena x di sisi kiri mengurangi x di sisi kanan.
- **Expand**: menjabarkan persamaan. Contoh $(x + 4)^2 = (x + 4)(x + 4)$.
- **Inference**: menggunakan **Machine Learning** *model* yang sudah dilatih untuk melakukan *real* prediksi menggunakan *real* data.

Aset

Repository Code: https://github.com/wuriyanto48/nn-js-example

Contoh *Serving Model* Pengenalan *Digit*/ Angka: https://wuriyanto48.github.io/nnjs

Desmos graph:

- Derivative 1 https://www.desmos.com/calculator/61owcjkxun
- Derivative 2 https://www.desmos.com/calculator/2a0l3dhfmv
- Vector https://www.desmos.com/calculator/hlnk3d7l33
- Vector addition https://www.desmos.com/calculator/zmic8tpuyq
- Vector multiplication https://www.desmos.com/calculator/o3lnqtkjpx
- Vector Dot Product https://www.desmos.com/calculator/eojih5kpho
- Vector Euclidean Distance https://www.desmos.com/calculator/qamccdkduh
- Vector Cosine Distance https://www.desmos.com/calculator/oiflzz2mle
- Vector 3D https://www.desmos.com/3d/46e924e5cc
- Neural Net 1 https://www.desmos.com/calculator/pfkogux6jh
- Neural Net 2 https://www.desmos.com/calculator/qgcz86vewi
- Gradient Descent https://www.desmos.com/calculator/zrhnizjgnv
- Derivative Sigmoid Function https://www.desmos.com/calculator/ralkl1q1rg

Penutup

Kesalahan penulisan dan perhitungan akan diperbaiki pada versi selanjutnya.

Terima kasih sudah membaca.

Salam

Wuriyanto