LAPORAN TUGAS BESAR  
IF3270 PEMBELAJARAN MESIN

ARTIFICIAL NEURAL NETWORK  
BAGIAN B:   
IMPLEMENTASI MINI-BATCH GRADIENT DESCENT

****

Oleh:

Eugene Yap Jin Quan 13521074

Michael Utama 13521137

Johann Christian Kandani 13521138

Dewana Gustavus Haraka Otang 13521173

**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2024**

# DAFTAR ISI

[**DAFTAR ISI 2**](#_3u3aj3gc6mfq)

[**I: SPESIFIKASI 3**](#_pp7ipu4efcj6)

[**II: IMPLEMENTASI 5**](#_jyxpea7vd5f0)

[Deskripsi Implementasi 5](#_3acx5qxtwhtl)

[**III: PENGUJIAN 10**](#_duxwm315kzoj)

[Hasil Pengujian Algoritma 10](#_1zhe17p2mouc)

[Pengujian Penyimpanan Model 14](#_cffglnes3kjb)

[Perbandingan Algoritma Dengan Penggunaan Library sklearn dan Keras 15](#_gkkcewi11zcj)

[**IV: PEMBAGIAN TUGAS 22**](#_3zsmy68lzk2b)

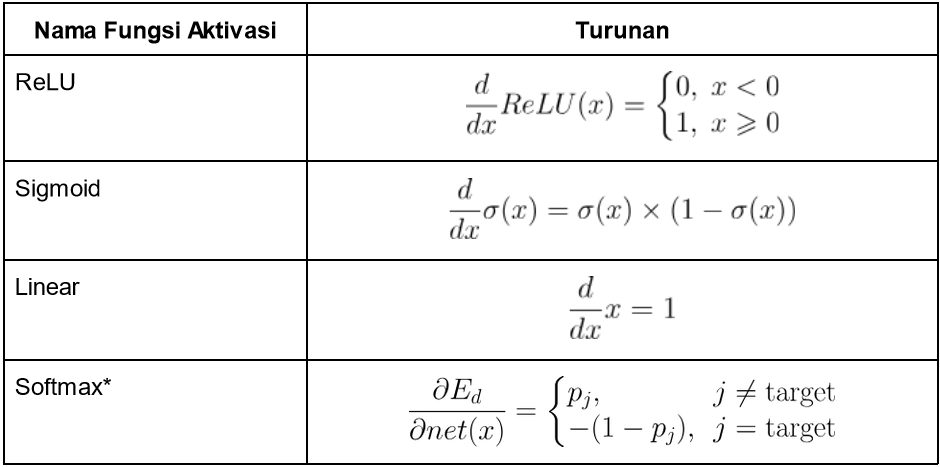
[**REFERENSI 23**](#_fjjdkd6ukp8l)

[**LAMPIRAN 23**](#_g6p8y0hxxcol)

# I: SPESIFIKASI

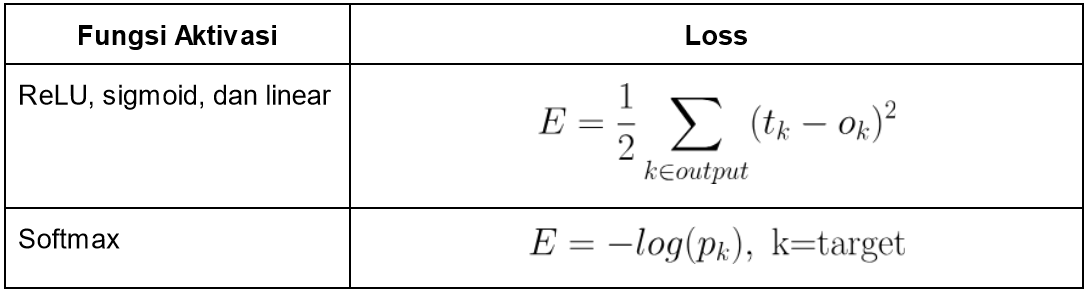
Pada Tugas Besar Bagian B, kami diinstruksikan untuk mengimplementasi algoritma *backpropagation* dari modul FFNN yang telah diimplementasikan pada Tugas Besar Bagian A. Berikut adalah detail spesifikasi implementasi algoritma *backpropagation*.

1. Implementasi algoritma *backpropagation* dapat melakukan *update weight* pada saat *training* menggunakan *mini-batch*, yang diatur melalui parameter *batch\_size*.
2. Implementasi algoritma *backpropagation* menggunakan fungsi aktivasi berupa ReLU, sigmoid, linear, dan *softmax*, dengan fungsi turunan dari masing-masing fungsi aktivasi adalah sebagai berikut.



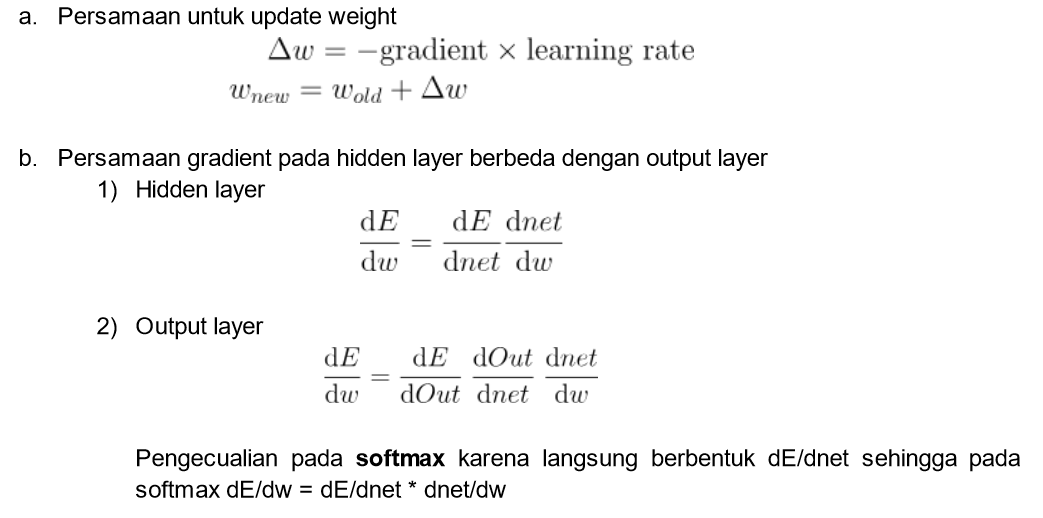
*Turunan fungsi aktivasi*

1. Implementasi algoritma *backpropagation* menghitung *loss* berdasarkan persamaan-persamaan berikut.

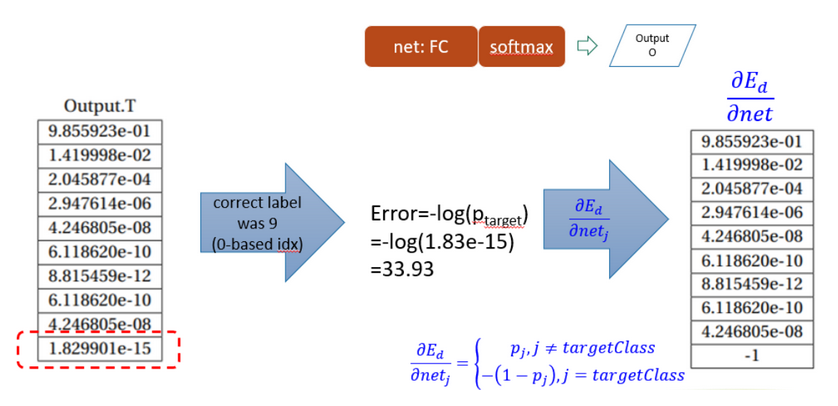


*Perhitungan loss untuk masing-masing fungsi aktivasi*

1. Update weight pada gradient descent dilakukan dengan aturan rantai.



*Perhitungan untuk proses update weight*



1. Kondisi berhenti *training* adalah ketika nilai *error kumulatif <= threshold* atau iterasi maksimum tercapai. Error threshold dan iterasi maksimum merupakan parameter yang dapat diubah.
2. Parameter yang wajib ada pada implementasi algoritma adalah struktur jaringan (jumlah layer, jumlah neuron setiap layer, fungsi aktivasi setiap layer), *learning\_rate*, *error\_threshold*, *max\_iter*, dan *batch\_size*. Parameter lain dapat didefinisikan sesuai kebutuhan.

Batasan dari implementasi adalah satu layer memiliki fungsi aktivasi yang sama; layer yang berbeda dapat memiliki fungsi aktivasi yang berbeda.

# II: IMPLEMENTASI

## Deskripsi Implementasi

Berikut adalah detail model FFNN-*backpropagation* yang telah diimplementasikan.

* Model Neural Network diinisialisasi dengan memasukkan bobot awal dan daftar fungsi aktivasi untuk setiap layer. Model ini memiliki fungsi FFNN untuk melakukan pelatihan dengan menggunakan algoritma *forward propagation/feed-forward*. Model ini melanjutkan implementasi model pada Tugas Besar Bagian A.



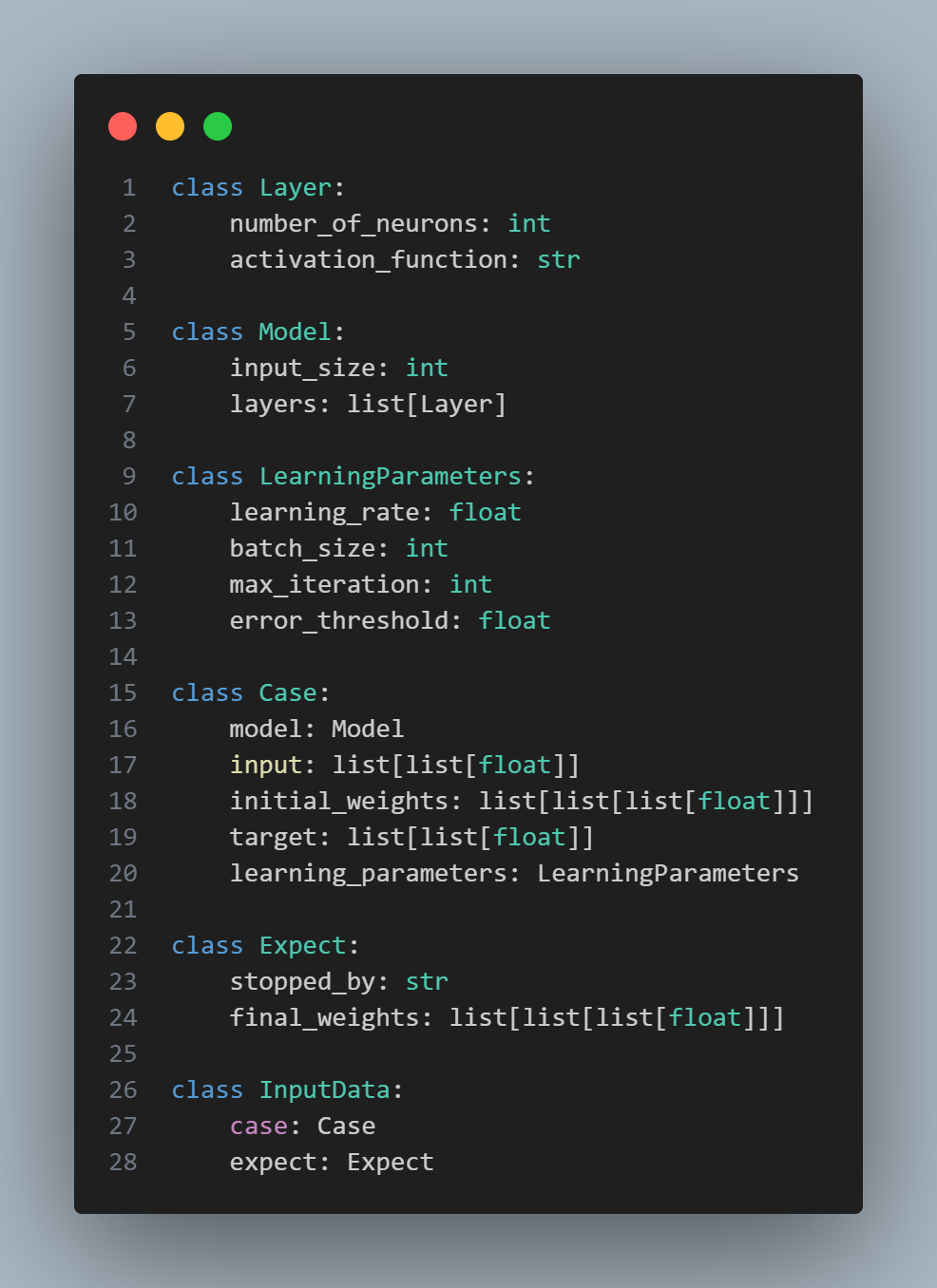
*Implementasi FFNN*

* Implementasi FFNN dilengkapi dengan algoritma *backpropagation* yang dipanggil melalui fungsi *train*. *Backpropagation* digunakan untuk menghitung nilai Δw pada setiap bobot. Proses perhitungan Δw dilakukan secara mundur dari *layer* output menuju *layer* input. Keluaran fungsi ini adalah daftar Δw yang digunakan dalam proses koreksi bobot.
* Fungsi *train* pada model memanggil pelatihan FFNN dan proses *backpropagation* secara iteratif. Pada setiap iterasi dilakukan koreksi menggunakan setiap *batch* baris data. Untuk setiap baris data, model melakukan proses pelatihan menggunakan error prediksi FFNN terhadap atribut baris dan total nilai Δw pada *batch*. Pada akhir batch dilakukan modifikasi terhadap bobot model menggunakan Δw yang tercatat pada *batch* tersebut. Iterasi pelatihan berhenti apabila ditemukan error di bawah *threshold* atau tercapai iterasi maksimum.



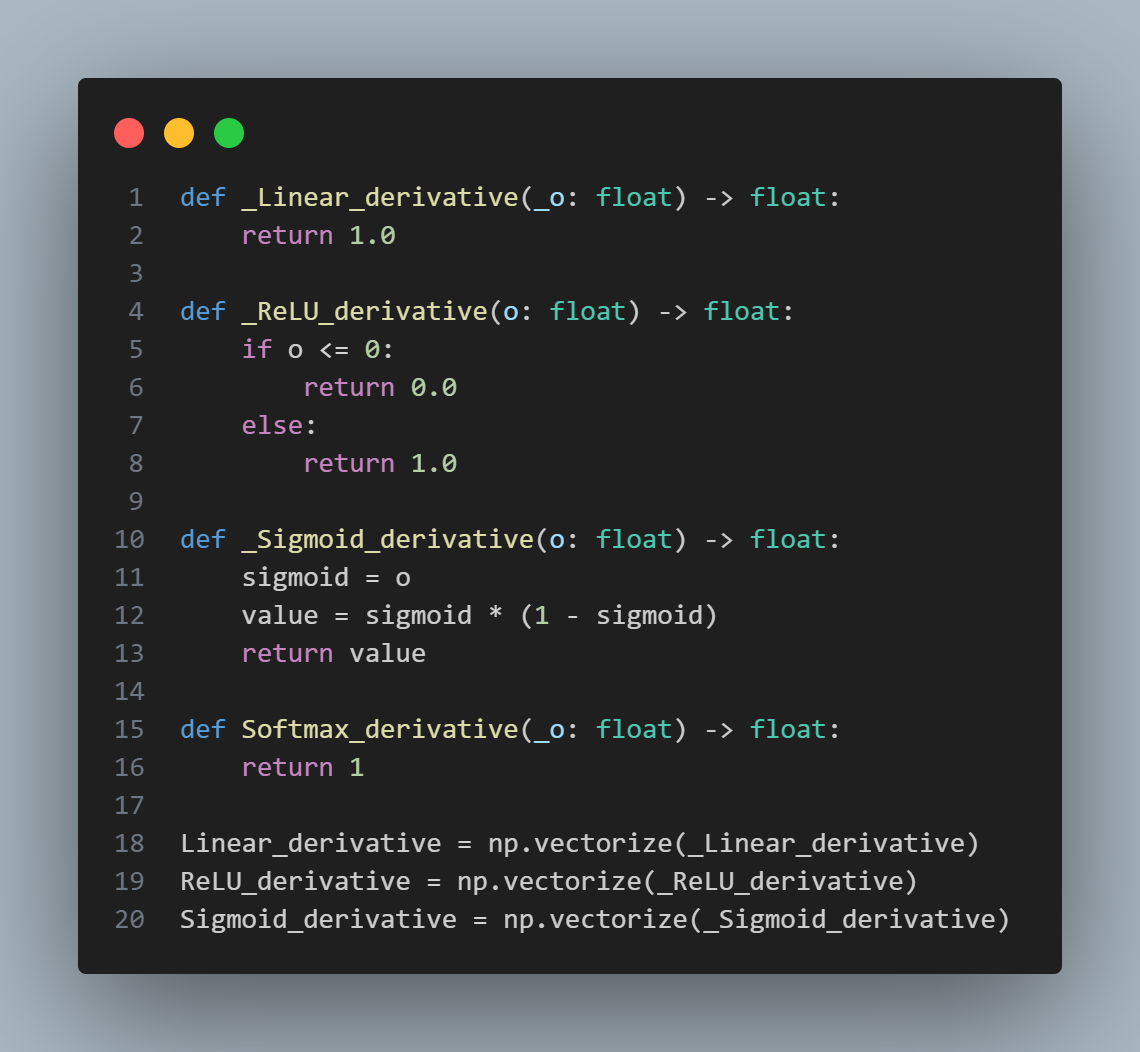
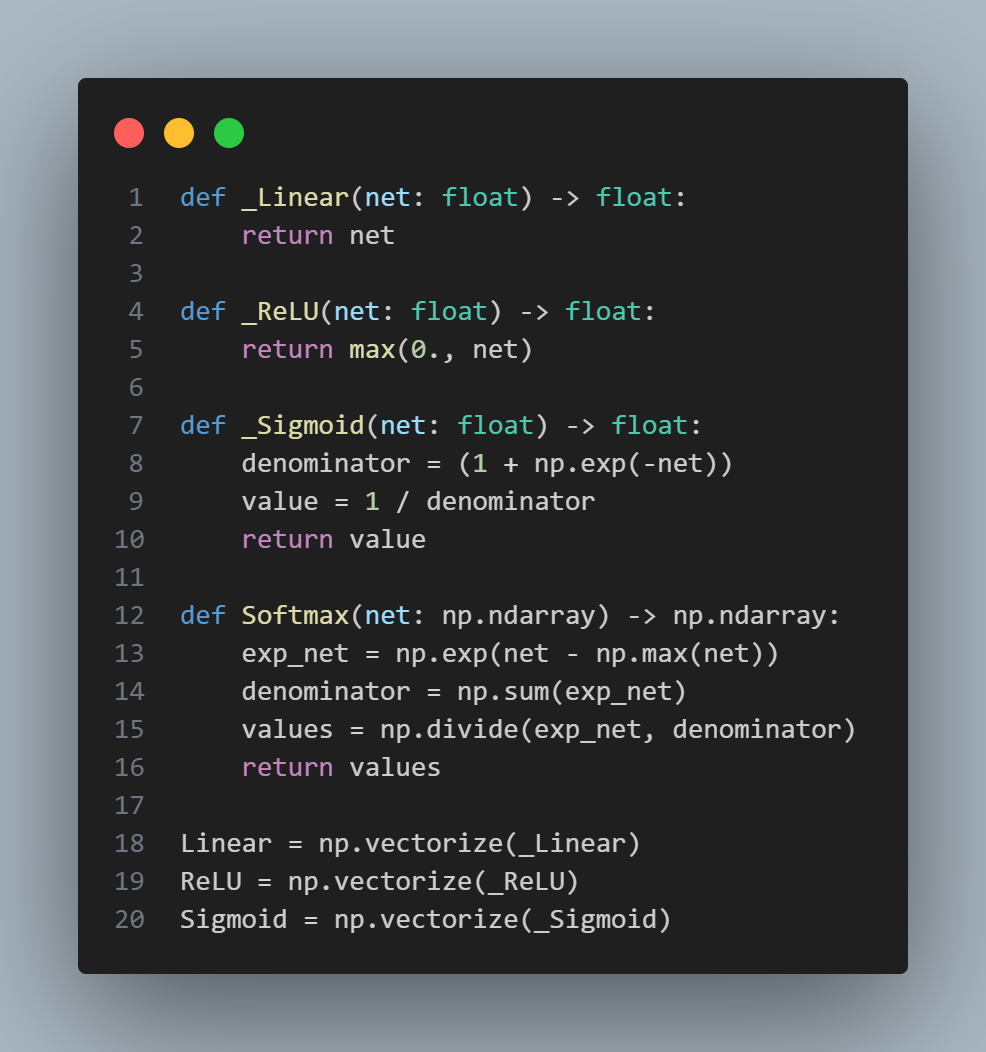
*Implementasi Algoritma Backpropagation*

* Apabila iterasi *train* terhenti akibat *threshold* atau akibat tercapainya jumlah iterasi maksimum, nilai-nilai output pada setiap *layer* pada model dihapus untuk mencegah proses penyimpanan model.
* Fungsi *train* menerima parameter berupa *learning\_rate, batch\_size, max\_iteration,* dan *error\_threshold*.

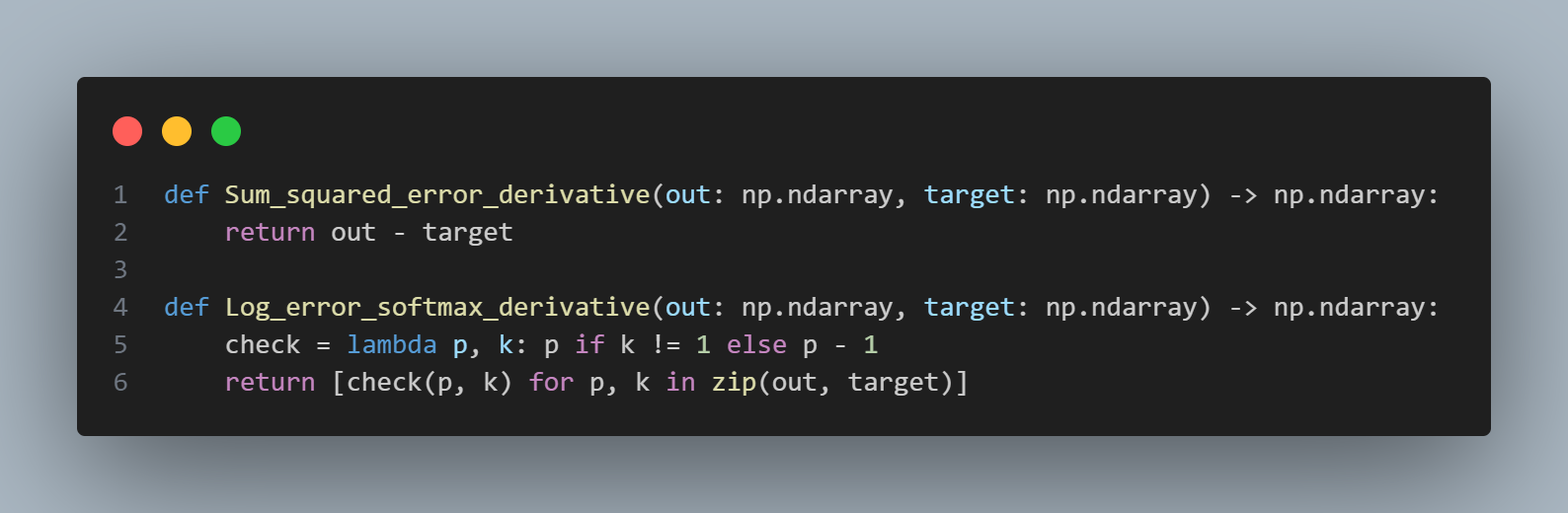
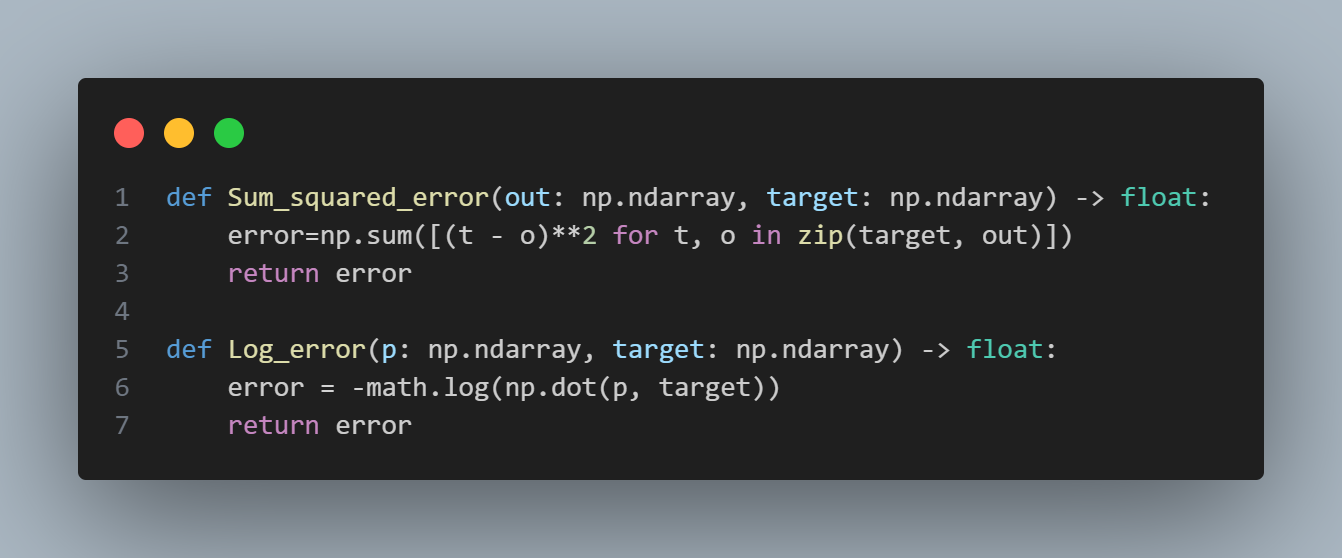


*Tipe data yang digunakan dalam pelatihan model*

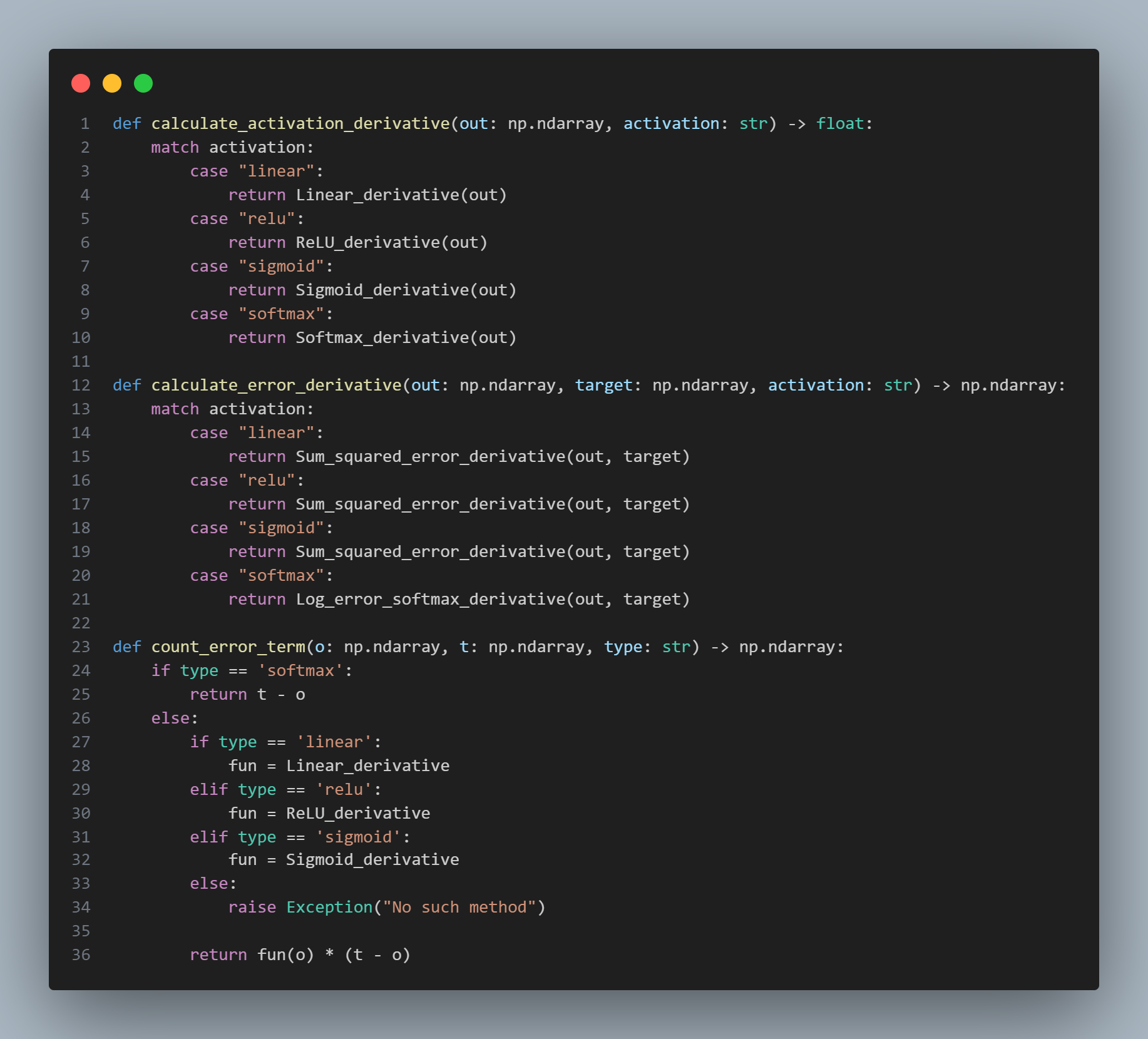
* Model Neural Network yang telah dibuat, baik dilatih menggunakan FFNN atau *backward propagation* dapat disimpan ke dalam sebuah *file*. *File* ini kemudian dapat dibaca kembali sebagai model Neural Network yang terlatih. Penyimpanan dan pembacaan dilakukan menggunakan modul *pickle*.
* Fungsi aktivasi, error, beserta turunan terhadap keduanya diimplementasikan sebagai fungsi yang dapat dipanggil oleh model.



*Implementasi fungsi aktivasi dan turunan fungsi aktivasi*



*Implementasi fungsi error dan turunan fungsi error*



*Fungsi wrapper untuk turunan fungsi aktivasi dan error*

* Contoh pemanggilan proses pelatihan model adalah sebagai berikut.



*Contoh pelatihan model NeuralNetwork*

# III: PENGUJIAN

## Hasil Pengujian Algoritma

Pengujian algoritma *backpropagation* dan pelatihan model dilakukan terhadap delapan kasus uji. Pemanggilan proses pelatihan untuk kedelapan kasus uji tersebut adalah sebagai berikut.



*Pengujian algoritma pelatihan model*

Hasil pengujian kedelapan kasus uji adalah sebagai berikut.

| **testcase: tcB/linear.json**  expected result:  stopped by: max\_iteration  final weights:  [[ 0.22 0.36 0.11]  [ 0.64 0.3 -0.89]  [ 0.28 -0.7 0.37]]  training result:  stopped by: max\_iteration  final weights:  [[ 0.22 0.36 0.11]  [ 0.64 0.3 -0.89]  [ 0.28 -0.7 0.37]] |
| --- |
| **testcase: tcB/linear\_small\_lr.json**  expected result:  stopped by: max\_iteration  final weights:  [[ 0.1008 0.3006 0.1991]  [ 0.402 0.201 -0.7019]  [ 0.101 -0.799 0.4987]]  training result:  stopped by: max\_iteration  final weights:  [[ 0.1012 0.3006 0.1991]  [ 0.4024 0.201 -0.7019]  [ 0.1018 -0.799 0.4987]] |
| **testcase: tcB/linear\_two\_iteration.json**  expected result:  stopped by: max\_iteration  final weights:  [[ 0.166 0.338 0.153]  [ 0.502 0.226 -0.789]  [ 0.214 -0.718 0.427]]  training result:  stopped by: max\_iteration  final weights:  [[ 0.166 0.338 0.153]  [ 0.502 0.226 -0.789]  [ 0.214 -0.718 0.427]] |
| **testcase: tcB/mlp.json**  expected result:  stopped by: max\_iteration  final weights:  [[ 0.08592 0.32276 ]  [-0.33872 0.46172 ]  [ 0.449984 0.440072]]  training result:  stopped by: max\_iteration  final weights:  [[ 0.08592 0.32276 ]  [-0.33872 0.46172 ]  [ 0.449984 0.440072]] |
| **testcase: tcB/relu\_b.json**  expected result:  stopped by: max\_iteration  final weights:  [[-0.211 0.105 0.885 ]  [ 0.3033 0.5285 0.3005]  [-0.489 -0.905 0.291 ]]  training result:  stopped by: max\_iteration  final weights:  [[-0.211 0.105 0.885 ]  [ 0.3033 0.5285 0.3005]  [-0.489 -0.905 0.291 ]] |
| **testcase: tcB/sigmoid.json**  expected result:  stopped by: max\_iteration  final weights:  [[0.2329 0.0601]  [0.1288 0.6484]  [0.8376 0.2315]]  training result:  stopped by: max\_iteration  final weights:  [[0.23291176 0.06015346]  [0.12884088 0.64849474]  [0.837615 0.23158199]] |
| **testcase: tcB/softmax.json**  expected result:  stopped by: max\_iteration  final weights:  [[ 0.12674605 0.9149538 -0.14169985]  [-0.33551647 0.67700488 0.45851159]  [ 0.48314436 -0.85241216 0.2692678 ]  [ 0.3400255 0.57237542 -0.31240092]  [ 0.31397716 0.46349737 0.72252547]  [-0.69652442 0.4789189 0.61760552]  [-0.50884515 -0.36354141 0.57238656]  [ 0.41891295 0.26354517 -0.48245812]  [ 0.90374164 -0.01759501 -0.08614663]]  training result:  stopped by: max\_iteration  final weights:  [[ 0.12674605 0.9149538 -0.14169985]  [-0.33551647 0.67700488 0.45851159]  [ 0.48314436 -0.85241216 0.2692678 ]  [ 0.3400255 0.57237542 -0.31240092]  [ 0.31397716 0.46349737 0.72252547]  [-0.69652442 0.4789189 0.61760552]  [-0.50884515 -0.36354141 0.57238656]  [ 0.41891295 0.26354517 -0.48245812]  [ 0.90374164 -0.01759501 -0.08614663]] |
| **testcase: tcB/softmax\_two\_layer.json**  expected result:  stopped by: error\_threshold  final weights:  [[-0.28730211 -0.28822282 -0.70597451 0.42094471]  [-0.5790794 -1.1836444 -1.34287961 0.69575311]  [-0.41434377 1.51314676 -0.97649086 -1.3043465 ]]  training result:  stopped by: error\_threshold  final weights:  [[-0.28730211 -0.28822282 -0.70597451 0.42094471]  [-0.5790794 -1.1836444 -1.34287961 0.69575311]  [-0.41434377 1.51314676 -0.97649086 -1.3043465 ]] |

Berdasarkan hasil di atas, dapat dilihat bahwa kondisi berhenti pada pengujian dan harapan kasus uji adalah sama untuk setiap kasus uji. Hasil pengujian juga menunjukkan bahwa bobot akhir hasil pelatihan sama dengan nilai harapan pada setiap kasus uji. Akan tetapi, terdapat dua dari delapan kasus uji yang mengalami perbedaan presisi *floating point* pada pencetakan nilai bobot akhir pelatihan, yakni kasus uji *sigmoid.json* dan kasus uji *linear\_small\_lr.json*.

## Pengujian Penyimpanan Model

Penyimpanan dan pembacaan model dari *file* dilakukan menggunakan *library pickle*. Pengujian dilakukan dengan menyimpan model hasil pelatihan menggunakan data *softmax\_two\_layer.json*. Kode pengujian yang digunakan adalah sebagai berikut.



*Penyimpanan dan pembacaan model dari file*

Hasil pengujian yang diperoleh adalah sebagai berikut. Dapat diperhatikan bahwa model yang diperoleh melalui proses penyimpanan dan pembacaan *file* mampu menghasilkan output yang diharapkan pada kasus uji.

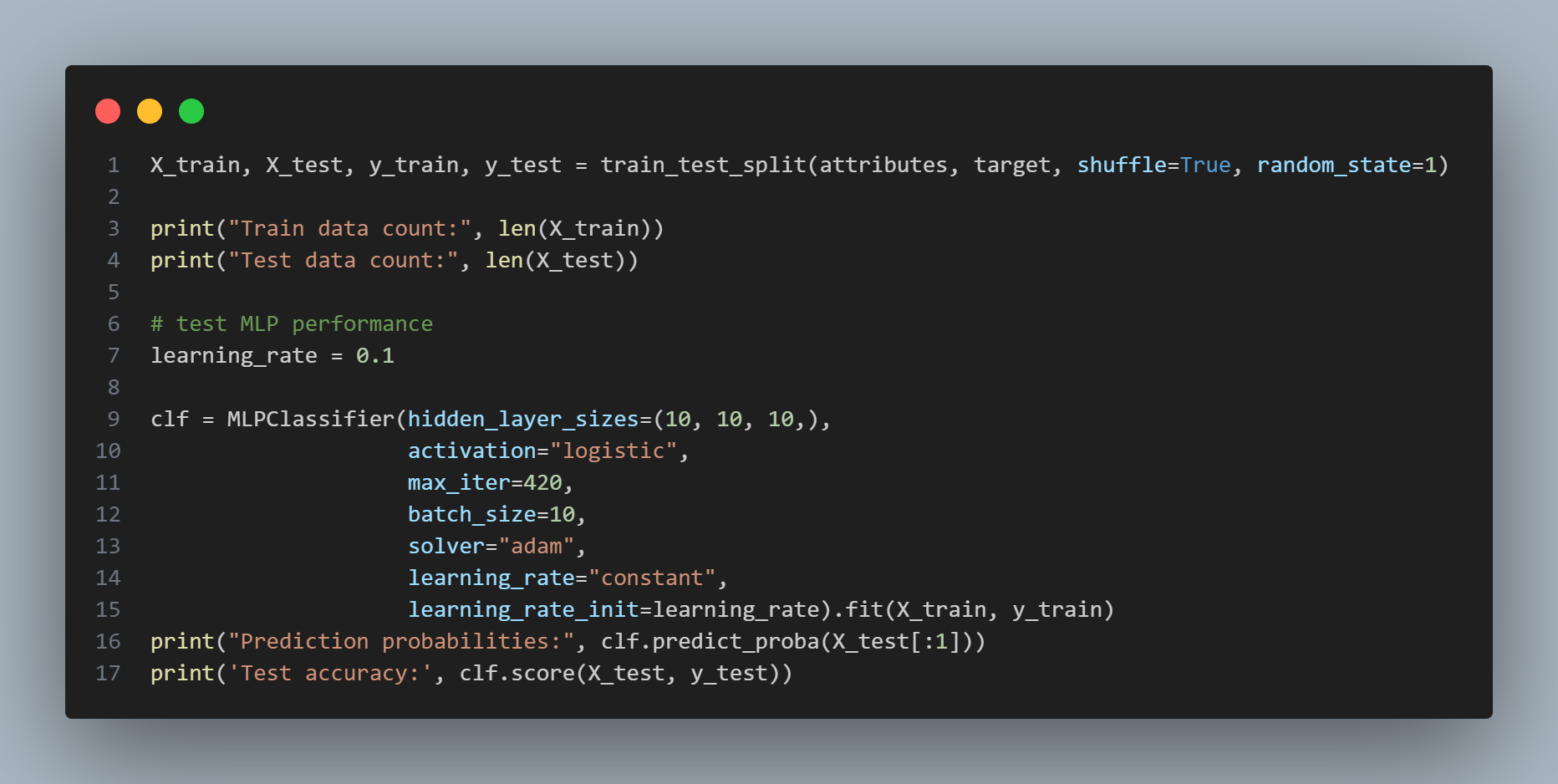
| expected result:  stopped by: error\_threshold  model output:  [0, 1]  [1, 0]  [0, 1]  [1, 0]  [1, 0]  [0, 1]  [1, 0]  [0, 1]  final weights:  [[-0.28730211 -0.28822282 -0.70597451 0.42094471]  [-0.5790794 -1.1836444 -1.34287961 0.69575311]  [-0.41434377 1.51314676 -0.97649086 -1.3043465 ]]  model result:  stopped by: error\_threshold  model output:  [0.0304, 0.9696]  [0.9998, 0.0002]  [0.0048, 0.9952]  [1.0, 0.0]  [0.9938, 0.0062]  [0.0017, 0.9983]  [0.9965, 0.0035]  [0.0304, 0.9696]  final weights:  [[-0.28730211 -0.28822282 -0.70597451 0.42094471]  [-0.5790794 -1.1836444 -1.34287961 0.69575311]  [-0.41434377 1.51314676 -0.97649086 -1.3043465 ]] |
| --- |

## Perbandingan Algoritma Dengan Penggunaan Library sklearn dan Keras

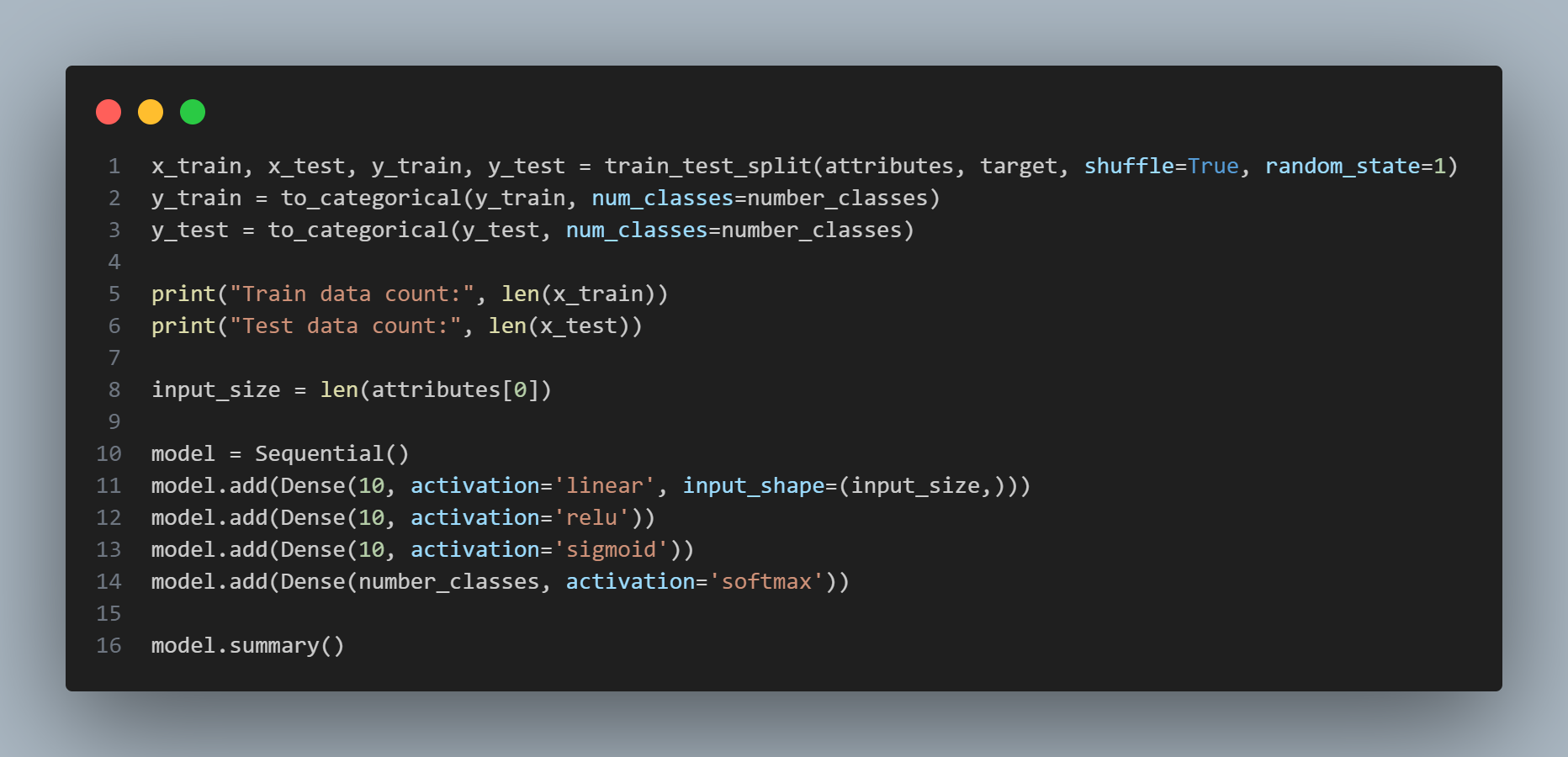
Untuk menguji performa prediksi dari model, kinerja model yang telah diimplementasikan dibandingkan dengan performa model Neural Network dari *library* *sklearn (MLPClassifier)* dan *Keras (Sequential)*. Performa model dari ketiga implementasi diuji menggunakan kasus uji *iris.csv*. Skema pelatihan masing-masing model adalah sebagai berikut.

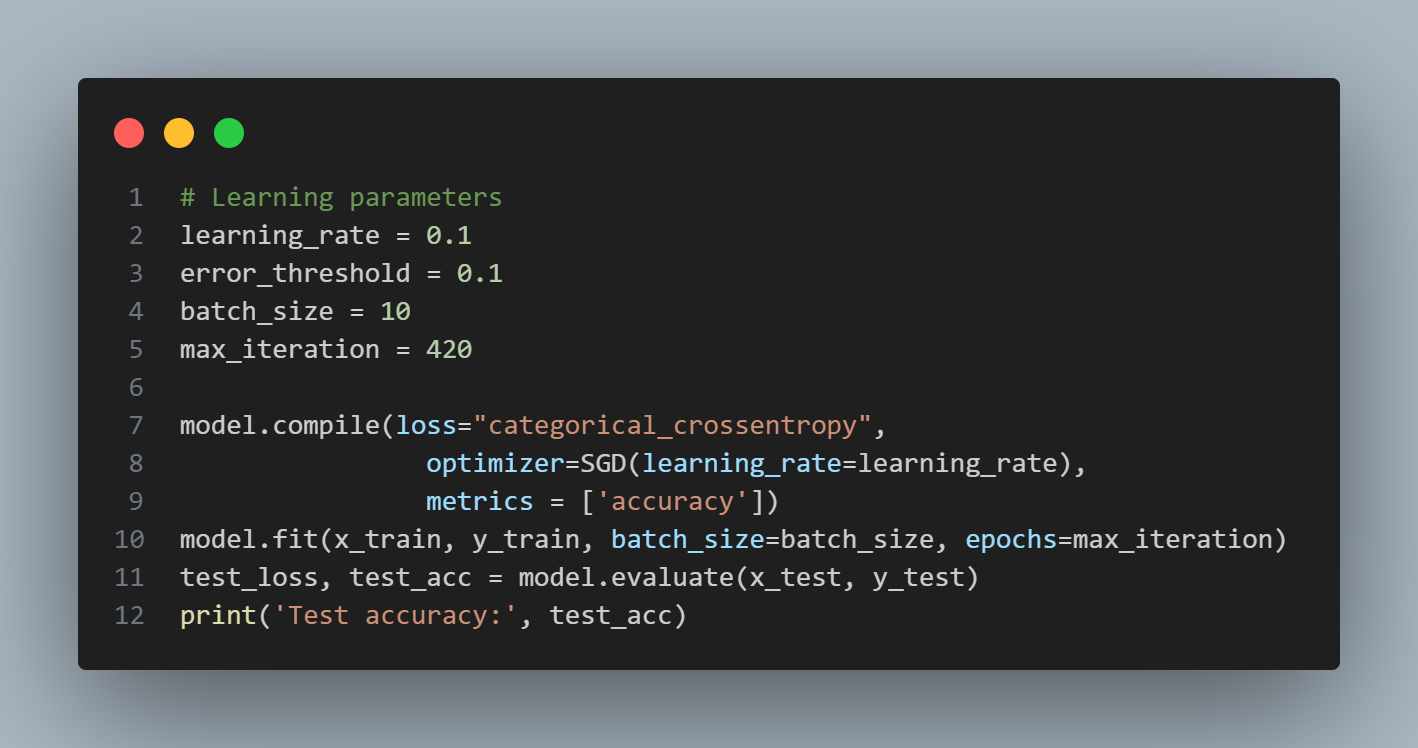
* Pembagian data *train* dan *test* adalah 75% - 25% (diperoleh dari parameter *default*). Pembagian dilakukan dengan menggunakan fungsi *train\_test\_split* dari *sklearn* dengan parameter *train\_test\_split(attributes, target, shuffle=True, random\_state=1)*.
* *learning rate* = 0.1
* *max\_iteration* = 420
* *batch\_size* = 10
* *error\_threshold* = 0.01
* Susunan *hidden layer* adalah 3 *layer* dengan 10 neuron untuk masing-masing *layer*.
* Fungsi aktivasi untuk *hidden layer* dan output secara berturut-turut adalah *linear, relu, sigmoid, softmax*.
* Berikut beberapa pengecualian parameter akibat batasan pada *library sklearn* dan *Keras*
  + Model *sklearn* mengimplementasikan fungsi aktivasi *sigmoid* untuk setiap *hidden layer*.
  + Model *sklearn* dan *Keras* tidak mengimplementasikan *error\_threshold*

Berikut adalah kode pemanggilan proses pelatihan pada masing-masing model.

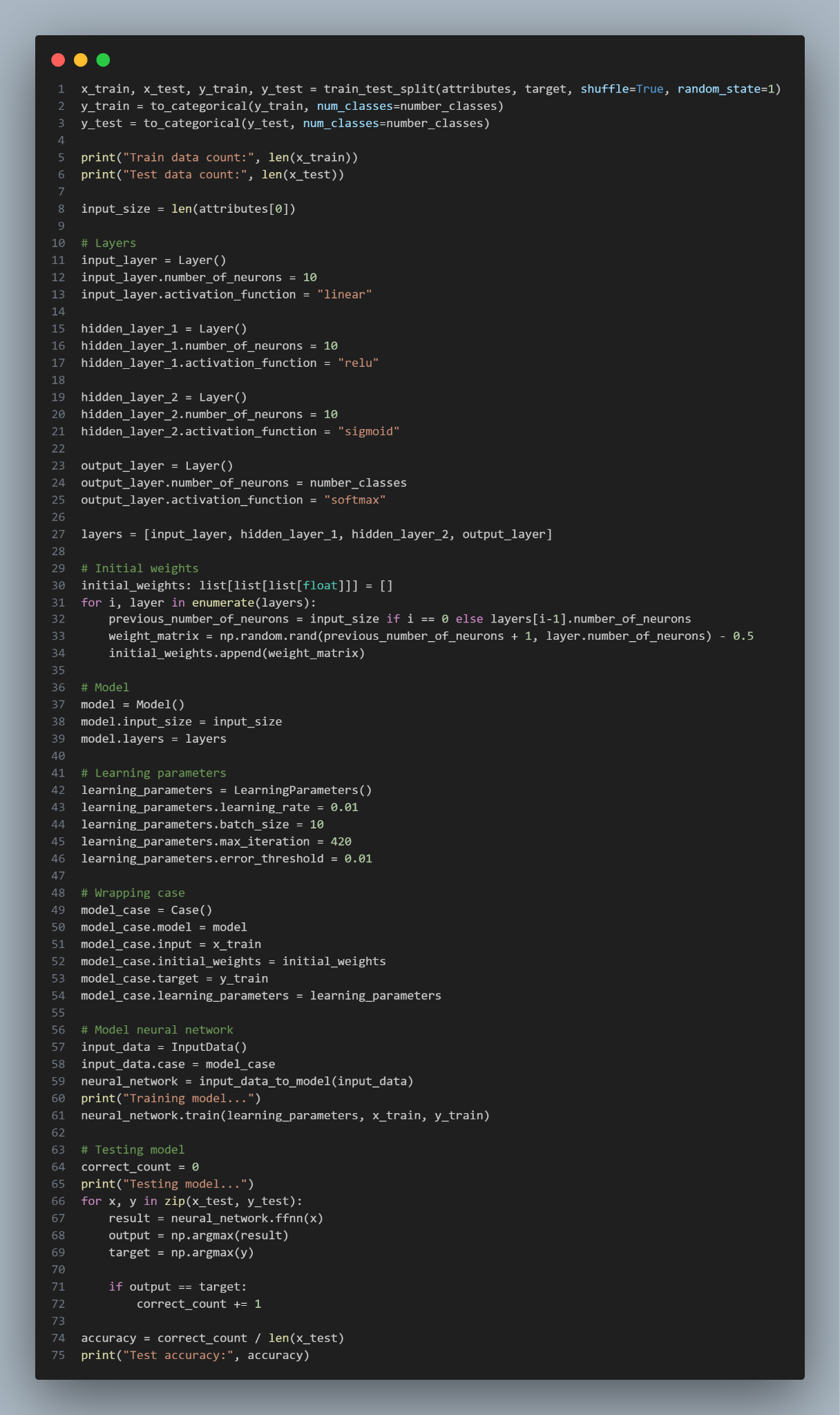


*Kode pngujian performa model MLPClassifier sklearn*





*Kode pengujian performa model Sequential Keras*



*Kode pengujian performa model NeuralNetwork*

Berikut adalah hasil pengujian performa untuk masing-masing implementasi model.

| ***sklearn MLPClassifier*** |
| --- |
| Train data count: 112  Test data count: 38  Prediction probabilities: [[0.99273101 0.0052438 0.00202519]]  **Test accuracy: 0.9473684210526315** |
| ***Keras Sequential*** |
| Train data count: 112  Test data count: 38  Model: "sequential\_1"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  dense\_4 (Dense) (None, 10) 50    dense\_5 (Dense) (None, 10) 110    dense\_6 (Dense) (None, 10) 110    dense\_7 (Dense) (None, 3) 33    =================================================================  Total params: 303 (1.18 KB)  Trainable params: 303 (1.18 KB)  Non-trainable params: 0 (0.00 Byte)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  ===============================================================  Epoch 1/420  12/12 [==============================] - 1s 3ms/step - loss: 1.0790 - accuracy: 0.5089  Epoch 2/420  12/12 [==============================] - 0s 3ms/step - loss: 0.9309 - accuracy: 0.5982  Epoch 3/420  12/12 [==============================] - 0s 6ms/step - loss: 0.7873 - accuracy: 0.6786  Epoch 4/420  12/12 [==============================] - 0s 4ms/step - loss: 0.6743 - accuracy: 0.6964  Epoch 5/420  12/12 [==============================] - 0s 4ms/step - loss: 0.5935 - accuracy: 0.7411  Epoch 6/420  12/12 [==============================] - 0s 4ms/step - loss: 0.5286 - accuracy: 0.6875  Epoch 7/420  12/12 [==============================] - 0s 5ms/step - loss: 0.5065 - accuracy: 0.7500  Epoch 8/420  12/12 [==============================] - 0s 5ms/step - loss: 0.4949 - accuracy: 0.7232  Epoch 9/420  12/12 [==============================] - 0s 4ms/step - loss: 0.4801 - accuracy: 0.7857  Epoch 10/420  12/12 [==============================] - 0s 4ms/step - loss: 0.4795 - accuracy: 0.8214  (400 baris dilewatkan)......  Epoch 410/420  12/12 [==============================] - 0s 13ms/step - loss: 0.1166 - accuracy: 0.9643  Epoch 411/420  12/12 [==============================] - 0s 9ms/step - loss: 0.0958 - accuracy: 0.9732  Epoch 412/420  12/12 [==============================] - 0s 9ms/step - loss: 0.1040 - accuracy: 0.9643  Epoch 413/420  12/12 [==============================] - 0s 6ms/step - loss: 0.0943 - accuracy: 0.9732  Epoch 414/420  12/12 [==============================] - 0s 7ms/step - loss: 0.1658 - accuracy: 0.9286  Epoch 415/420  12/12 [==============================] - 0s 7ms/step - loss: 0.1118 - accuracy: 0.9643  Epoch 416/420  12/12 [==============================] - 0s 6ms/step - loss: 0.0947 - accuracy: 0.9732  Epoch 417/420  12/12 [==============================] - 0s 8ms/step - loss: 0.0875 - accuracy: 0.9732  Epoch 418/420  12/12 [==============================] - 0s 7ms/step - loss: 0.0920 - accuracy: 0.9732  Epoch 419/420  12/12 [==============================] - 0s 9ms/step - loss: 0.1749 - accuracy: 0.9196  Epoch 420/420  12/12 [==============================] - 0s 10ms/step - loss: 0.1335 - accuracy: 0.9464  2/2 [==============================] - 0s 11ms/step - loss: 0.0539 - accuracy: 0.9737  **Test accuracy: 0.9736841917037964** |
| ***Model NeuralNetwork*** |
| Train data count: 112  Test data count: 38  Training model...  Testing model...  **Test accuracy: 1.0** |

Berdasarkan hasil pengujian, performa ketiga model dalam memprediksi data *test* berada di atas nilai akurasi 0.9. Pada instansi pengujian di atas, nilai akurasi untuk model *sklearn, Keras,* dan *NeuralNetwork* secara berturut-turut adalah 0.9473684210526315, 0.9736841917037964, dan 1.0. Nilai akurasi ketiga model menunjukkan bahwa model memiliki kinerja dan akurasi yang sejenis.

Meskipun demikian, nilai akurasi untuk ketiga implementasi model ini tidak bernilai konstan. Pada beberapa pengujian, ditemukan bahwa nilai akurasi ketiga model dapat berubah-ubah ketika pengujian dijalankan berulang kali. Hal ini diduga akibat pemberian nilai bobot acak untuk masing-masing model pada tahap inisialisasi pelatihan model.

# IV: PEMBAGIAN TUGAS

Pembagian tugas pada pengerjaan Tugas Besar IF3270 Bagian B adalah sebagai berikut.

| NIM | Nama | Tugas |
| --- | --- | --- |
| 13521074 | Eugene Yap Jin Quan | sklearn, pengujian, analisis hasil dan perbandingan, input dataset iris |
| 13521137 | Michael Utama | backpropagation, fungsi train, model untuk dataset iris |
| 13521138 | Johann Christian Kandani | backpropagation, fungsi turunan, model untuk dataset iris |
| 13521173 | Dewana Gustavus Haraka Otang | keras, fungsi input json, pengujian dataset iris dengan model, contoh visualisasi dan penyimpanan model |

# REFERENSI

<https://towardsdatascience.com/derivative-of-the-softmax-function-and-the-categorical-cross-entropy-loss-ffceefc081d1>

<https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier>

<https://machinelearningmastery.com/build-multi-layer-perceptron-neural-network-models-keras/>

# LAMPIRAN

Dokumen Spesifikasi: [IF3270 - Spesifikasi Tugas Besar](https://docs.google.com/document/d/1DO4g3y76CX5uN3yAMZ7KcvbKmRnQp2ZpdIpls1Ik4wI/edit?usp=sharing)

Kasus Pengujian: [Test Case Bagian B](https://drive.google.com/drive/folders/11ejn9MB3jrs17cEBwppzfwDPGviukKIx)

Kasus Pengujian (iris): [iris.csv](https://drive.google.com/file/d/1yIG2WVsoEum49iIkCKNoKKBcb2QzJYSq/view?usp=sharing)

# 