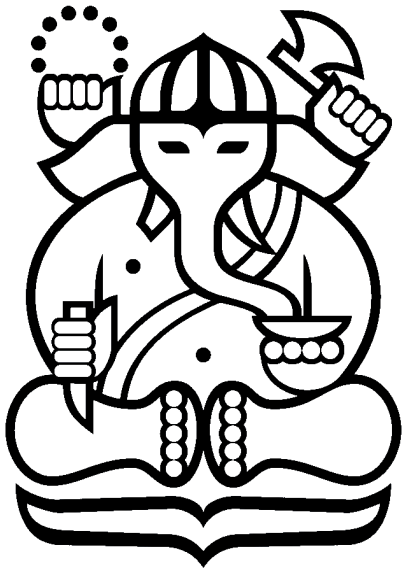
LAPORAN TUGAS BESAR 1

IF3270 PEMBELAJARAN MESIN

Feedforward Neural Network



Disusun Oleh:

13522096 Novelya Putri Ramadhani

13522102 Hayya Zuhailii Kinasih

13522104 Diana Tri Handayani

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

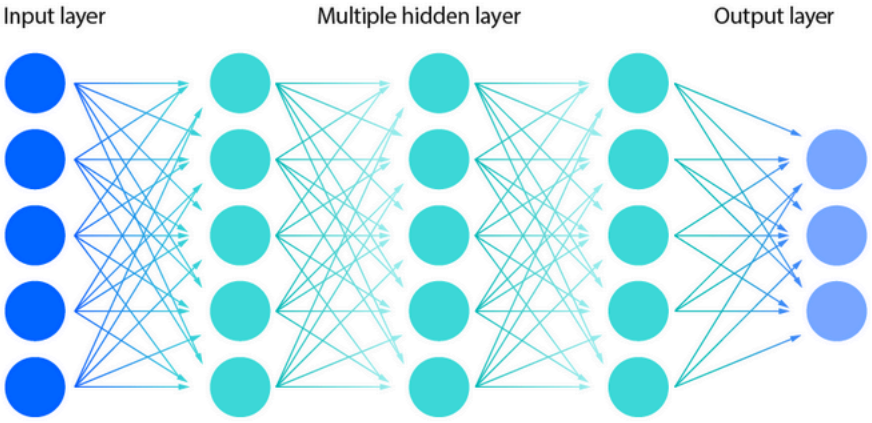
2025

DAFTAR ISI

DAFTAR GAMBAR

BAB I
DESKRIPSI MASALAH

1.1. Deskripsi Persoalan



Gambar 1.1.1 Ilustrasi Graf FFNN
(Sumber: Spesifikasi Tugas Besar 1 Pembelajaran Mesin)

Feedforward Neural Network (FFNN) adalah sebuah implementasi Artificial Neural Network (ANN). Sesuai namanya, algoritma tersebut terinspirasi dari cara kerja jaringan neuron pada otak manusia. ANN terdiri dari tiga komponen utama, yaitu *input layer*, *hidden layer(s)*, dan *output layer*. Setiap jenis *layer* tersebut terdiri atas sejumlah *neuron*, sesuai dengan kebutuhan persoalan. Pada FFNN, setiap *neuron* dalam suatu *layer* terhubung ke semua *neuron* dalam *layer* berikutnya, sehingga terbentuk graf yang terarah dan asiklik. Setiap hubungan tersebut memiliki bobot masing-masing. Informasi mengalir mulai dari *input layer*, melalui beberapa *hidden layers*, dan berakhir di *output layer*.

Model yang dihasilkan FFNN adalah bobot dari setiap hubungan antar-*neuron*. Bobot dan nilai setiap *neuron* digunakan untuk membentuk fungsi *linear*, yang kemudian akan digunakan untuk menemukan nilai *neuron* selanjutnya dengan fungsi aktivasi untuk *layer* tersebut, hingga akhirnya sampai ke *output layer*.

1.2. Spesifikasi Tugas

Tugas ini adalah untuk mengimplementasi model FFNN *from scratch* dengan spesifikasi berikut:

1. FFNN dapat menerima jumlah *neuron* untuk setiap *layer*.
2. FFNN dapat menerima fungsi aktivasi untuk masing-masing *layer*. Fungsi aktivasi yang wajib untuk diimplementasikan adalah Linear, ReLU, Sigmoid, Hyperbolic Tangent, dan Softmax. Terdapat juga ketentuan implementasi dua fungsi aktivasi lain yang bersifat opsional
3. FFNN dapat menerima fungsi *loss* untuk model. Fungsi *loss* yang wajib diimplementasikan adalah MSE, Binary Cross-Entropy, dan Categorical Cross-Entropy.
4. FFNN dapat menerima pilihan untuk inisialisasi bobot. Pilihan itu adalah inisialisasi 0, acak dengan distribusi seragam, dan acak dengan distribusi normal. Terdapat juga pilihan inisialisasi dengan algoritma Xavier dan He/Kaiming yang bersifat opsional.
5. FFNN dapat menyimpan bobot setiap *neuron*.
6. FFNN dapat menyimpan gradien bobot setiap *neuron*.
7. Metode yang menampilkan model dalam bentuk graf yang menyertakan bobot dan gradien bobot setiap *neuron*.
8. Metode yang menampilkan distribusi bobot pada beberapa *layer*.
9. Metode yang menampilkan distribusi gradien bobot pada beberapa *layer*.
10. Metode untuk *save* dan *load* model.
11. FFNN dapat melakukan *forward propagation* dengan *input* berupa *batch*.
12. FFNN dapat melakukan *backward propagation* dengan *input* berupa *batch*.
13. FFNN dapat melakukan *update* bobot menggunakan *gradient descent*.
14. FFNN memiliki parameter ukuran *batch*, *learning rate*, jumlah *epoch*, dan tingkat *verbosity*. *Verbose* 1 menampilkan *progress bar* serta *training loss* dan *validation loss* saat itu.
15. FFNN dapat menerima pilihan regularisasi, antara L1, L2, atau tidak ada. Ketentuan ini opsional.

16. FFNN dapat menerima pilihan normalisasi, antara RMSNorm atau tidak ada. Ketentuan ini opsional.
17. Implementasi FFNN menggunakan *automatic differentiation*. Ketentuan ini opsional.

Selain implementasi, terdapat ketentuan untuk melakukan pengujian terhadap model yang telah dibuat. Pengujian tersebut meliputi membandingkan hasil dari model dengan parameter berbeda serta membandingkan hasil antara model yang dibuat dengan model dari *library.*

BAB II
PEMBAHASAN

2.1. Penjelasan Implementasi

2.1.1. Deskripsi Kelas

1. Kelas FFNN

Kelas ini adalah kelas yang merepresentasikan model FFNN yang dibuat.

Atribut/Metode	Deskripsi
layers	Sebuah array of Layer yang menyimpan informasi tentang *layer* dalam *neural network*.
activations	Sebuah array of function tuple yang menyimpan fungsi aktivasi dan derivasi setiap *layer*.
input_cache	Sebuah atribut yang menyimpan *input* awal dari model.
regularization	Sebuah atribut yang menyimpan tipe regularisasi yang digunakan. Bernilai “none”, “l1”, atau “l2”.
l	Sebuah atribut yang menyimpan nilai lambda yang digunakan untuk regularisasi.
rmsnorm	Sebuah atribut yang menyimpan apakah model menggunakan normalisasi atau tidak.
epsilon	Sebuah atribut yang menyimpan nilai epsilon yang digunakan untuk normalisasi.
gammas	Sebuah array yang menyimpan nilai gamma setiap *layer* untuk normalisasi.
init(self, layer_sizes, activations, regularization, l, rmsnorm, epsilon)	Prosedur untuk menginisialisasi model. Prosedur ini menerima ukuran setiap *layer*, fungsi aktivasi setiap *layer*, jenis regularisasi dan lambdanya, dan normalisasi dan epsilonnya.
forward(self, X)	Fungsi untuk melakukan *forward propagation*. Fungsi ini menerima *input* X dan mengeluarkan *output* hasil *forward*

	propagation.
backward(self, y_true, loss_fn, loss_deriv)	Prosedur untuk melakukan *back propagation*. Prosedur ini menerima *input* y_true (*output* sebenarnya), loss_fn (fungsi loss), dan loss_deriv (derivasi fungsi loss). Prosedur ini mengubah gradien bobot pada setiap *layer*.
update_weights(self, lr)	Prosedur untuk melakukan *weight update*. Prosedur ini menerima *input* lr (learning rate). Prosedur ini mengubah bobot setiap *layer*.
save(self, filename)	Prosedur untuk menyimpan model.
load(self, filename)	Prosedur untuk memuat model.

2. Kelas Layer

   Kelas ini merepresentasikan layer yang membangun FFNN.

Atribut/Metode	Deskripsi
input_size	Sebuah atribut yang menyimpan ukuran *input* kepada *layer*.
output_size	Sebuah atribut yang menyimpan ukuran *output* dari *layer.*
weights	Sebuah atribut yang menyimpan bobot yang ada pada *layer*.
bias	Sebuah atribut yang menyimpan bobot *bias* pada *layer*.
z	Sebuah atribut yang menyimpan hasil perhitungan dari *input* dan bobot.
a	Sebuah atribut yang menyimpan hasil perhitungan dari atribut z dan fungsi aktivasi.
grad_weights	Sebuah atribut yang menyimpan gradien bobot pada *layer.*
grad_bias	Sebuah atribut yang menyimpan gradien bias pada *layer.*
init(self, input_size, output_size,	Prosedur yang menginisialisasi sebuah *layer*. Prosedur ini menerima ukuran *input*, ukuran

init_method, init_params)	*output*, metode inisialisasi bobot, dan parameter lainnya yang dibutuhkan untuk inisialisasi bobot.
forward(self, input)	Fungsi ini menghitung nilai *node* menggunakan *input* dan bobot.

2.1.2. *Forward Propagation*

Forward Propagation adalah proses pengolahan data *input* untuk menghasilkan *output* dengan cara memasukkannya ke dalam model *neural network*. Implementasi *forward propagation* pada model ini adalah sebagai berikut:

```
function FORWARD(X) returns output of neural network
    a ← X
    for each layer in the model do
        z ← layer.FORWARD(X)
        a ← ACTIVATION(z)
        if using RMSNorm then
            a ← RMSNorm(a, gamma, epsilon)
        layer.a ← a
        layer.z ← z
    → a
```

Fungsi ini melakukan iterasi melalui seluruh *layer* dalam model. Pada setiap *layer* dihitung nilainya berdasarkan *input* dan bobot. Kemudian, nilai itu dimasukkan ke dalam fungsi aktivasi. Jika dilakukan normalisasi, maka akan dipanggil fungsi normalisasi. Setelah itu, nilai a dan z di-*update* untuk *layer* tersebut.

2.1.3. *Backward Propagation* dan *Weight Update*

Backward Propagation adalah proses penelusuran *neural network* dari lapisan *output* kembali ke lapisan *input*. Pada proses itu dapat dilakukan *weight update* untuk mengubah bobot agar model dapat menghasilkan hasil yang lebih mendekati *output* sebenarnya.

Implementasi *backward propagation* pada model ini adalah sebagai berikut:

```
procedure BACKWARD(y_true, loss_deriv)
    batch_size ← y_true.shape[1]
    last_idx ← LEN(layers) - 1
    last_layer ← layers[last_idx]
    last_activation_deriv ← activations[last_idx][1]
    dz ← LOSS_DERIV(y_true, last_layer.a) *
LAST_ACTIVATION_DERIV(last_layer.z)
    da ← dz
    last_layer.grad_weights ← np.DOT(da, layers[last_idx].a.T) /
batch_size
    last_layer.grad_bias ← np.SUM(da, axis=1, keepdims=True) /
batch_size

    for i in REVERSED(LEN(layers) - 1) do
        current_layer ← layers[i]
        next_layer ← layers[i + 1]
        activation_deriv ← activations[i][1]
        dz ← np.DOT(next_layer.weights.T, da) *
ACTIVATION_DERIV(last_layer.z)
        da ← dz
        input_activation ← layers[i - 1].a if i > 0 else input_cache
        current_layer.grad_weights ← np.DOT(dz, input_activation.T) /
batch_size
        current_layer.grad_bias ← np.SUM(dz, axis=1, keepdims=True) /
batch_size
```

Prosedur ini menelusuri setiap *layer* mulai dari *layer* terakhir dan menghitung gradien bobot pada setiap *layer*. Perhitungan gradien bobot dilakukan menggunakan nilai-nilai pada *layer* sebelum dan setelahnya.

Sedangkan implementasi *weight update* adalah sebagai berikut:

```
procedure UPDATE_WEIGHTS(lr)
    for each layer in the model do
        if regularization = "l1" then
            layer.weights ← layer.weights -
```

```
lr*(lambda*np.SIGN(layer.weights))
        else if regularization = "l2" then
            layer.weights ← layer.weights - lr*(lambda*2*layer.weights)
        layer.weights ← layer.weights - lr*layer.grad_weights
        layer.bias ← layer.bias - lr*layer.grad_bias
```

Prosedur ini menelusuri setiap *layer* mulai dari awal. Jika model menggunakan regularisasi, maka bobot *layer* akan diubah sesuai dengan algoritma masing-masing regularisasi. Setelah itu, bobot *layer* diubah berdasarkan *learning rate* dan gradien bobot yang telah dihitung pada *back propagation*.

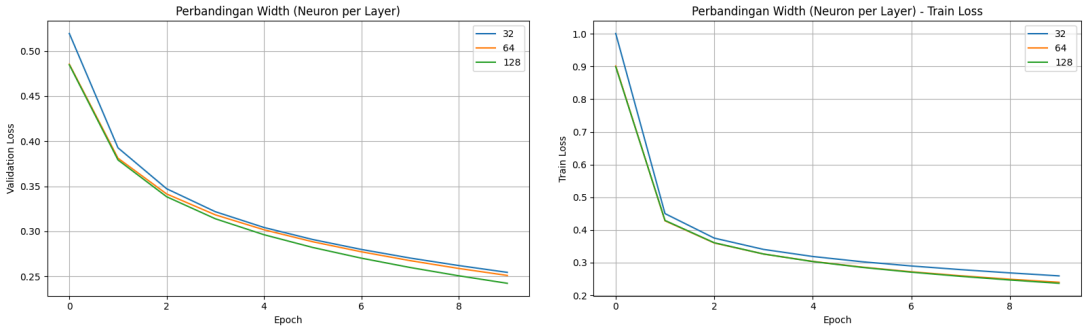
2.2. Hasil Pengujian

2.2.1. Pengaruh *Depth* dan *Width*

Pengaruh *width* dapat dilihat dengan melakukan perubahan parameter banyak neuron per *layer*. Dengan *depth* tetap yaitu 1 hidden layer, banyak epoch 10, batch size 64, *learning rate* 0.01, fungsi aktivasi reLU dan softmax, inisialisasi bobot dengan metode He initialization dan banyak neuron tiap layer;

- *Width* kecil: [784, 32, 10],
- *Width* sedang: [784, 64, 10],
- *Width* besar: [784, 128, 10],

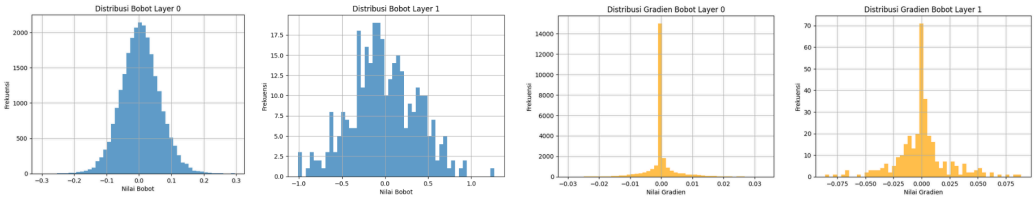
berikut grafik hasil percobaannya:



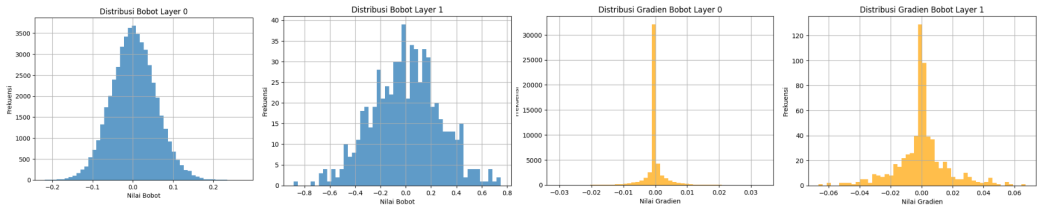
Gambar 2.2.1.1 Grafik *validation loss* dan *train loss* percobaan perbandingan *width*

```
Akurasi FFNN untuk masing-masing width:
Width 32: 0.9269
Width 64: 0.9321
Width 128: 0.9327
```

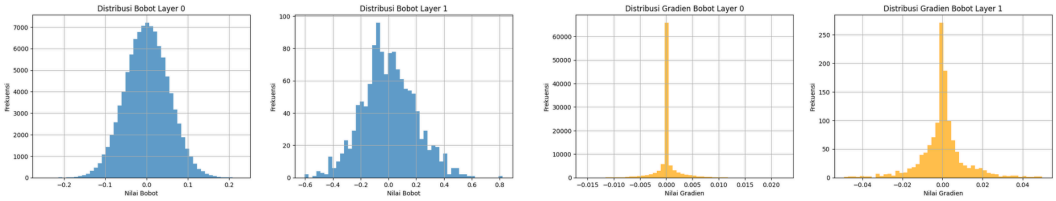
Gambar 2.2.1.2 Akurasi percobaan perbandingan *width* (neuron per layer)



Gambar 2.2.1.3 Distribusi bobot dan gradien bobot percobaan *width* = 32



Gambar 2.2.1.4 Distribusi bobot dan gradien bobot percobaan *width* = 64



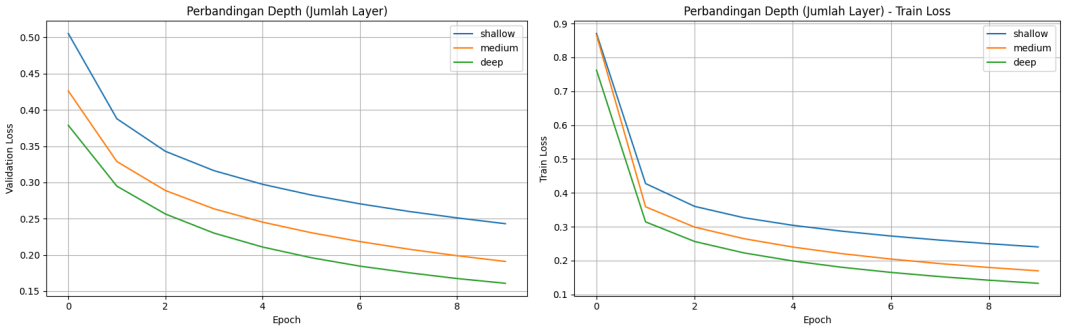
Gambar 2.2.1.5 Distribusi bobot dan gradien bobot percobaan *width* = 128

Berdasarkan grafik yang dihasilkan dapat dilihat bahwa semakin banyak neuron pada *hidden layer* atau *width* semakin lebar maka semakin rendah *validation loss* dan *train loss* yang dihasilkan. Secara penurunan *validation loss* dan *train loss* tidak terlalu berbeda antar perbedaan width-nya. Akurasi yang dihasilkan, semakin banyak neuron pada *hidden layer* maka semakin tinggi tingkat akurasinya. Distribusi bobot dan gradien bobot secara persebaran hampir sama, yang membedakan adalah *range* frekuensi yang dihasilkan.

Pengaruh perbandingan *depth* dapat dilihat dengan perubahan jumlah layer yang dilakukan. Dengan *width* tetap yaitu 64 neuron setiap hidden layer, banyak epoch 10, batch size 64, learning rate 0.01, fungsi aktivasi reLU dan softmax, inisialisasi bobot dengan metode He initialization dan jumlah layer;

- Shallow: 1 hidden layer → [784, 64, 10],
- Medium: 2 hidden layer → [784, 64, 64, 10],
- Deep: 3 hidden layer → [784, 64, 64, 64, 10],

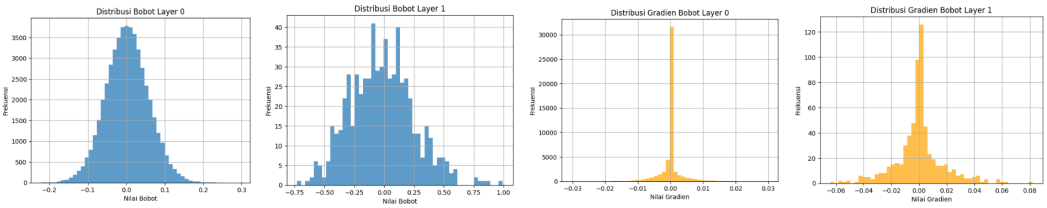
grafik hasil percobaan sebagai berikut:



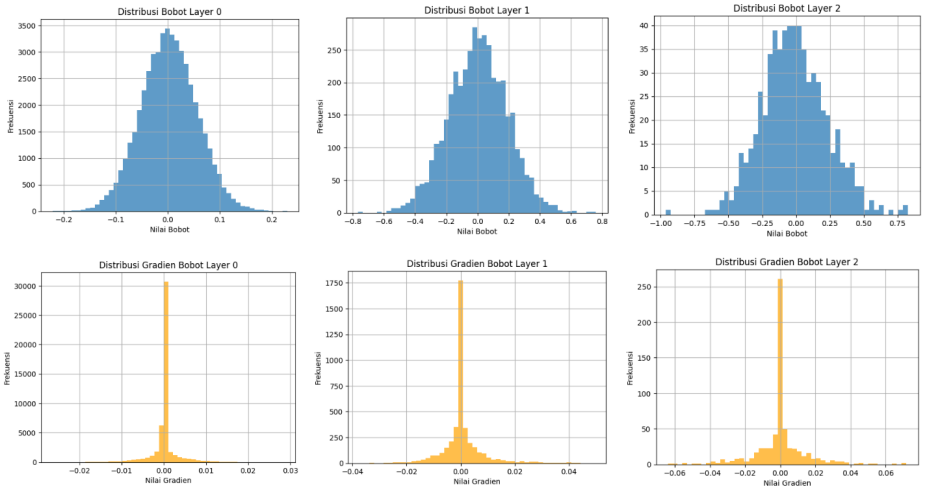
Gambar 2.2.1.6 Grafik *validation loss* dan *train loss* percobaan perbandingan *depth*

```
Akurasi FFNN untuk masing-masing depth:
Shallow: 0.9314
Medium: 0.9479
Deep: 0.9546
```

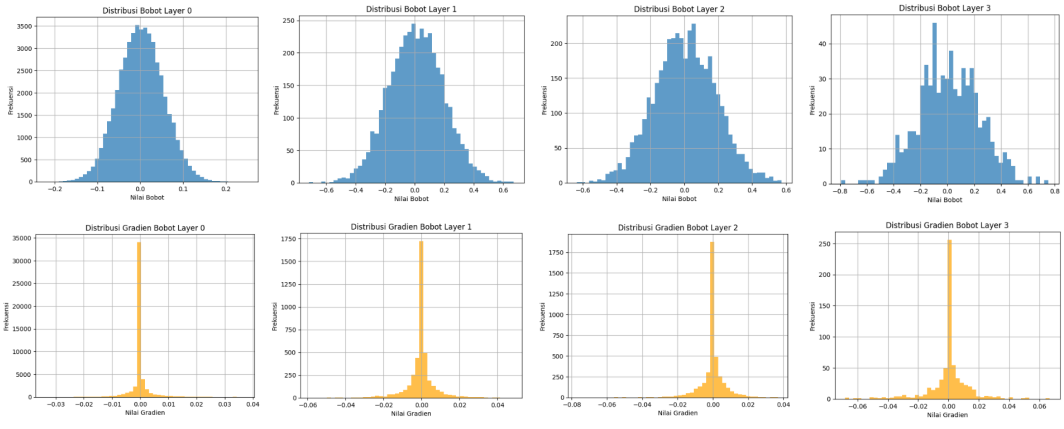
Gambar 2.2.1.7 Akurasi percobaan perbandingan *width* (neuron per layer)



Gambar 2.2.1.8 Distribusi bobot dan gradien bobot percobaan *depth* = 1 *hidden layer*



Gambar 2.2.1.9 Distribusi bobot dan gradien bobot percobaan *depth* = 2 *hidden layer*



Gambar 2.2.1.10 Distribusi bobot dan gradien bobot percobaan *depth* = 3 *hidden layer*

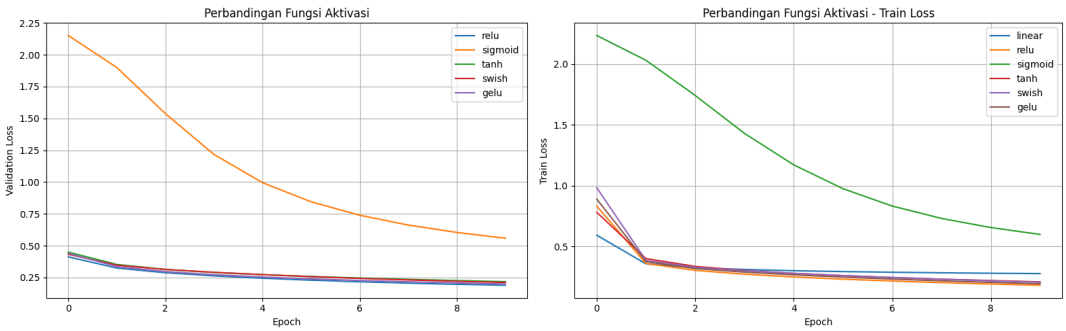
Semakin dalam atau semakin banyak jumlah layer yang digunakan maka semakin rendah *validation loss* dan *train loss* yang didapatkan, baik dari epoch pertama hingga terakhir. Akurasi yang dihasilkan, semakin banyak *hidden layer* maka semakin tinggi tingkat akurasinya. Distribusi bobot dan gradien bobot secara persebaran hampir sama, yang membedakan adalah *range* frekuensi yang dihasilkan.

2.2.2. Pengaruh Fungsi Aktivasi

Terdapat banyak fungsi aktivasi yang dapat digunakan yaitu reLU, sigmoid, tanh, swish, dan geLU. Dengan *depth* tetap yaitu 1 hidden layer, banyak epoch 10, batch size 64, *learning rate* 0.01, fungsi aktivasi reLU dan softmax, inisialisasi bobot dengan metode He initialization dan banyak neuron tiap layer;

- *Width* kecil: [784, 32, 10],
- *Width* sedang: [784, 64, 10],
- *Width* besar: [784, 128, 10],

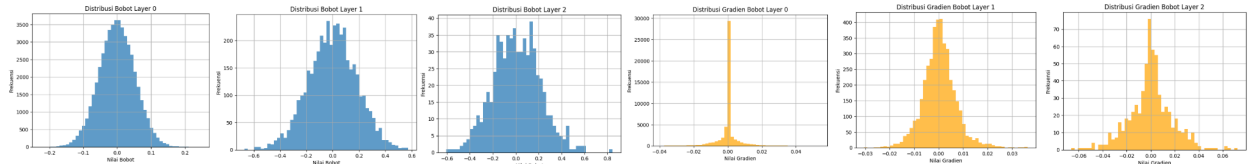
berikut grafik hasil percobaannya:



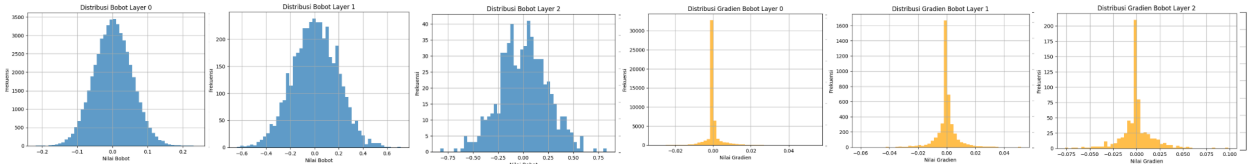
Gambar 2.2.2.1 Grafik *validation loss* dan *train loss* percobaan perbandingan fungsi aktivasi

```
Akurasi FFNN untuk masing-masing fungsi aktivasi:
linear: 0.9171
relu: 0.9464
sigmoid: 0.8589
tanh: 0.9374
swish: 0.9394
gelu: 0.9439
```

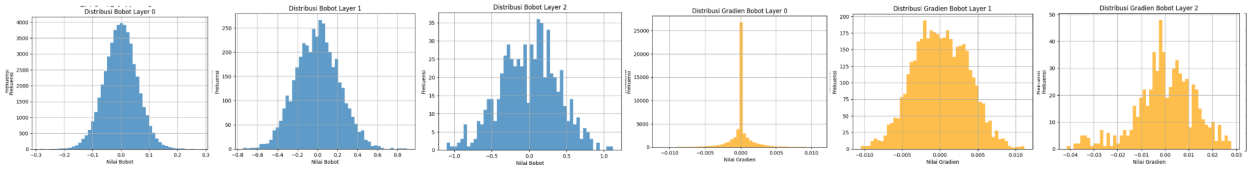
Gambar 2.2.2.2 Akurasi percobaan perbandingan fungsi aktivasi



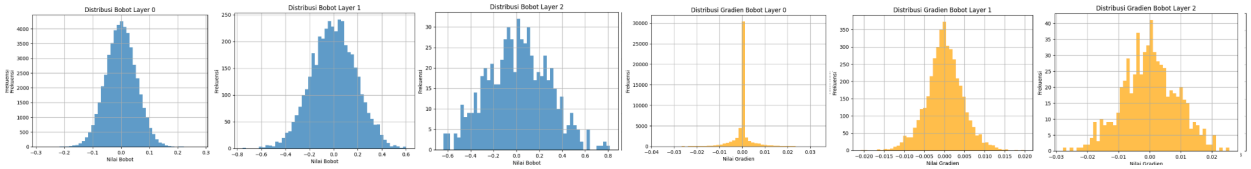
Gambar 2.2.2.3 Distribusi bobot dan gradien bobot percobaan fungsi aktivasi linear



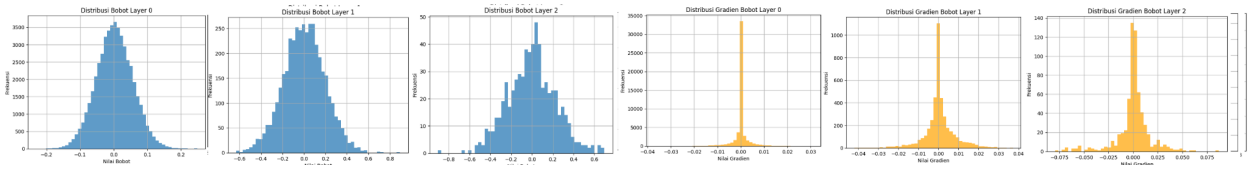
Gambar 2.2.2.4 Distribusi bobot dan gradien bobot percobaan fungsi aktivasi ReLU



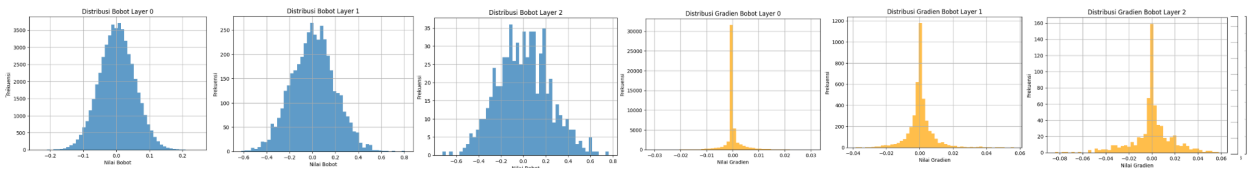
Gambar 2.2.2.5 Distribusi bobot dan gradien bobot percobaan fungsi aktivasi sigmoid



Gambar 2.2.2.6 Distribusi bobot dan gradien bobot percobaan fungsi aktivasi tanh



Gambar 2.2.2.7 Distribusi bobot dan gradien bobot percobaan fungsi aktivasi swish

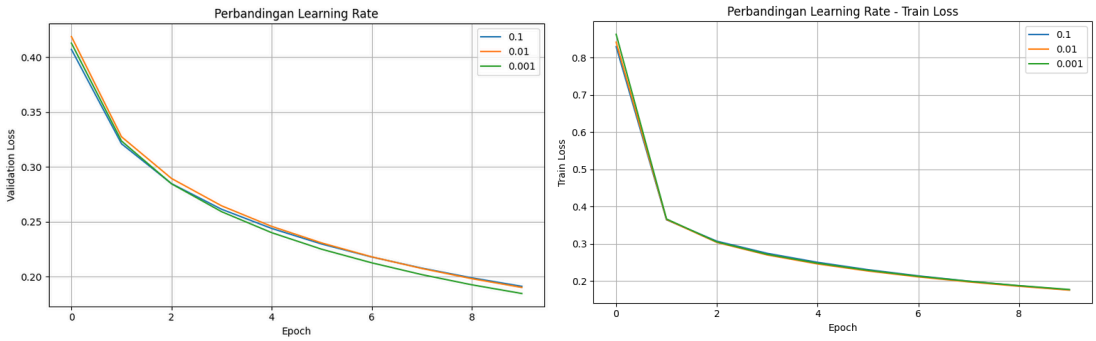


Gambar 2.2.2.8 Distribusi bobot dan gradien bobot percobaan fungsi aktivasi gelu

Semua fungsi aktivasi memberikan hasil yang mirip satu sama lain dengan satu pengecualian, yaitu fungsi aktivasi sigmoid. Hal itu disebabkan oleh metode derivasi sigmoid. Fungsi aktivasi sigmoid memberikan hasil dalam jangkauan 0 sampai 1, sehingga cocok untuk klasifikasi *binary*, namun tidak untuk klasifikasi *multiclass*.

2.2.3. Pengaruh *Learning Rate*

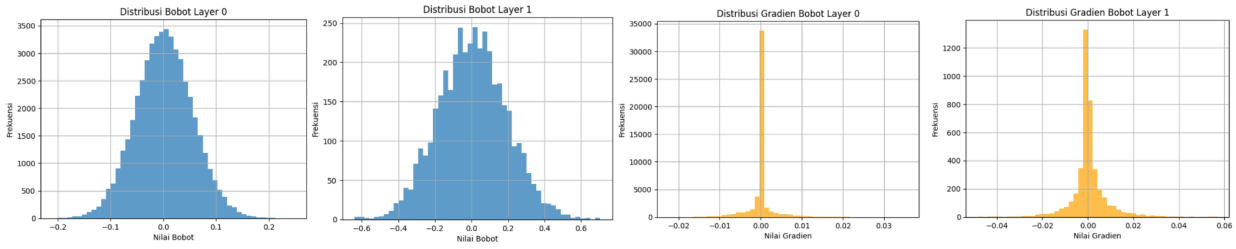
Variasi *learning rate* yang digunakan pada eksperimen ini adalah 0.1, 0.01, dan 0.001. Parameter lain yang digunakan pada model ini adalah dua *hidden layer* dengan *width* 64, inisialisasi bobot He, dan fungsi aktivasi ReLU. Berikut grafik hasil percobaannya:



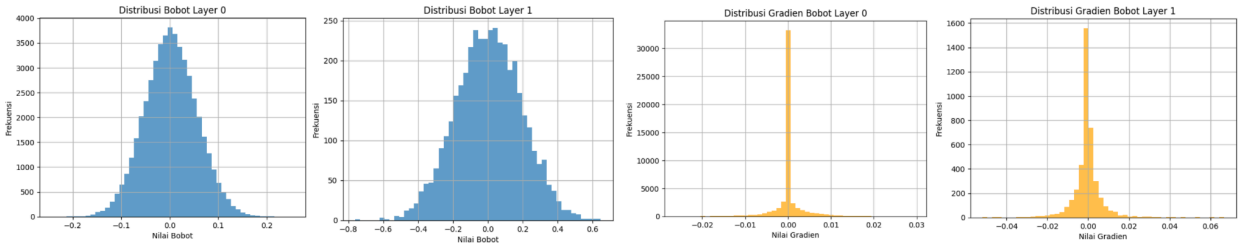
Gambar 2.2.3.1 Grafik *validation loss* dan *train loss* percobaan perbandingan *learning rate*

```
Akurasi FFNN untuk berbagai learning rate:
Learning Rate 0.1: 0.9449
Learning Rate 0.01: 0.9456
Learning Rate 0.001: 0.9466
```

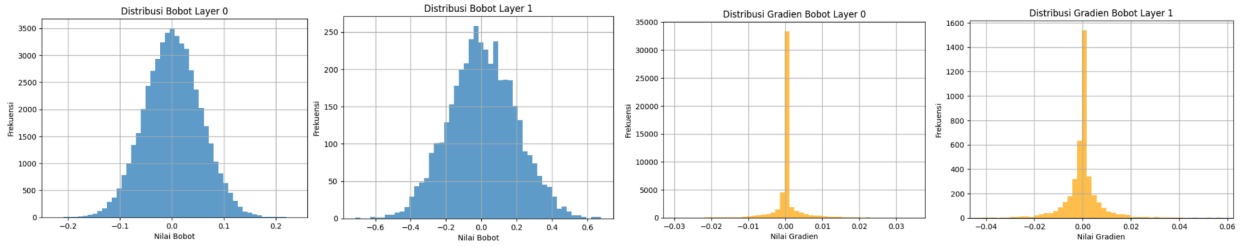
Gambar 2.2.3.2 Akurasi percobaan perbandingan *learning rate*



Gambar 2.2.3.3 Distribusi bobot dan gradien bobot percobaan *learning rate* 0.1



Gambar 2.2.3.4 Distribusi bobot dan gradien bobot percobaan *learning rate* 0.01



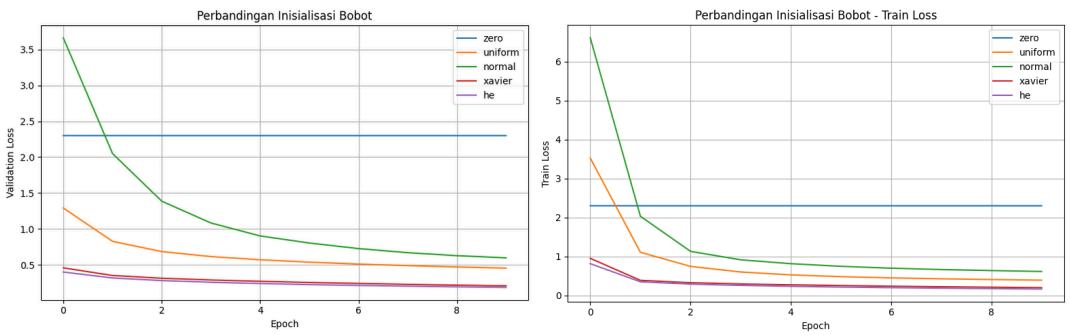
Gambar 2.2.3.5 Distribusi bobot dan gradien bobot percobaan *learning rate* 0.001

Semua *learning rate* menghasilkan *validation loss* dengan laju penurunan yang serupa serta nilai yang tidak jauh berbeda. Akurasinya juga mirip.

Learning rate adalah variabel yang menentukan seberapa cepat model berubah pada setiap *epoch*-nya. Nilai yang terlalu besar bisa membuat model melewati angka konvergennya, namun untuk *dataset* ini, hal itu tidak terjadi. Nilai *learning rate* tidak terlalu berpengaruh terhadap proses pelatihan model.

2.2.4. Pengaruh Inisialisasi Bobot

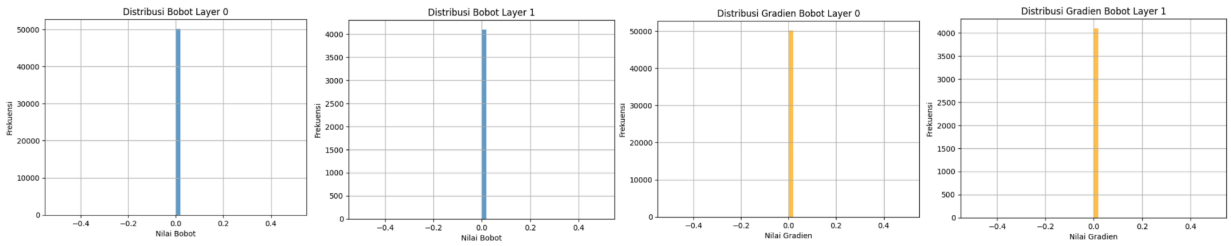
Metode inisialisasi bobot yang akan digunakan adalah zero, uniform, normal, Xavier, dan He. Parameter lain yang digunakan pada model ini adalah dua *hidden layer* dengan *width* 64, *learning rate* 0.01, dan fungsi aktivasi ReLU. Berikut grafik hasil percobaannya:



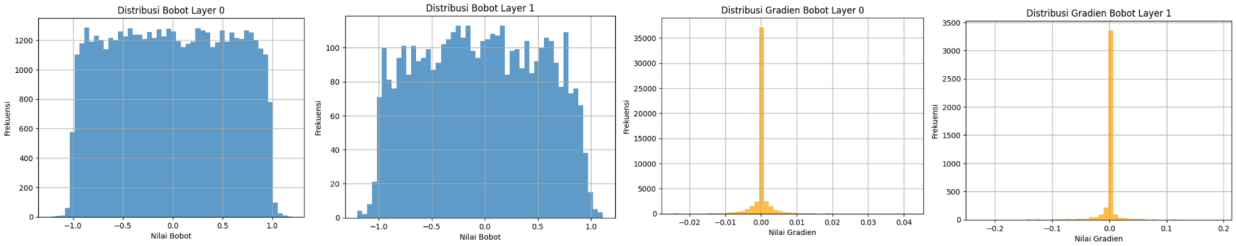
Gambar 2.2.4.1 Grafik percobaan perbandingan inisialisasi bobot

```
Akurasi FFNN untuk berbagai metode inisialisasi:
Init zero: 0.1143
Init uniform: 0.8702
Init normal: 0.8322
Init xavier: 0.9417
Init he: 0.9456
```

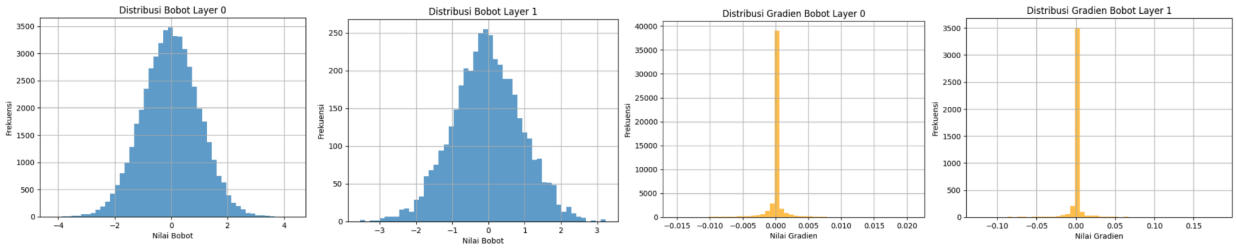
Gambar 2.2.4.2 Akurasi percobaan perbandingan inisialisasi bobot



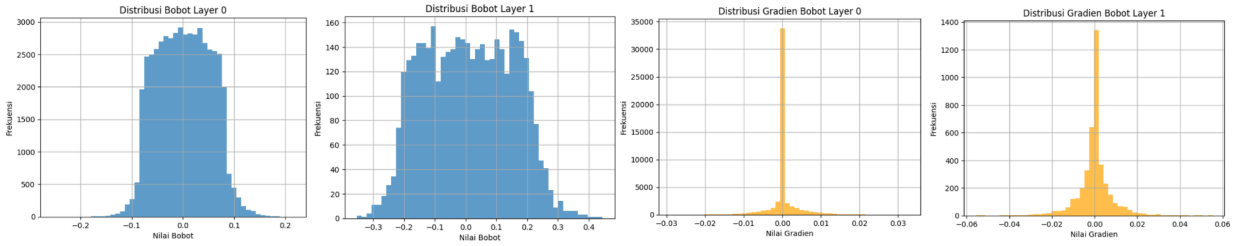
Gambar 2.2.4.3 Distribusi bobot dan gradien bobot percobaan inisialisasi bobot zero



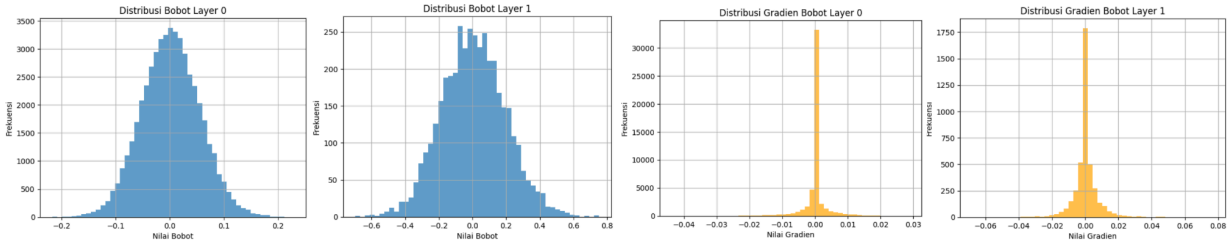
Gambar 2.2.4.4 Distribusi bobot dan gradien bobot percobaan inisialisasi bobot uniform



Gambar 2.2.4.5 Distribusi bobot dan gradien bobot percobaan inisialisasi bobot normal



Gambar 2.2.4.6 Distribusi bobot dan gradien bobot percobaan inisialisasi bobot Xavier



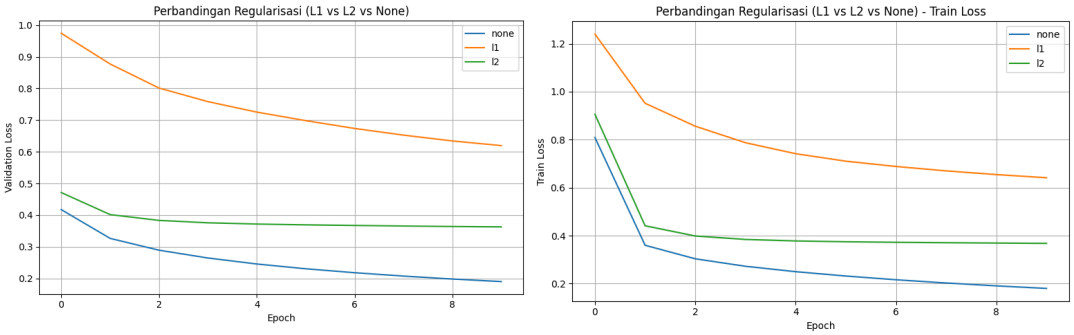
Gambar 2.2.4.7 Distribusi bobot dan gradien bobot percobaan inisialisasi bobot He

Terdapat perbedaan yang cukup signifikan pada hasil antara metode-metode inisialisasi. Metode inisialisasi zero menghasilkan model dengan nilai *validation loss* cukup tinggi dan nilai itu juga tidak turun. Metode inisialisasi uniform dan normal mulai dengan nilai *validation loss* yang juga cukup tinggi, namun menurun secara drastis seiring berlalunya *epoch*. Metode inisialisasi Xavier dan He mulai dengan nilai *validation loss* paling kecil dan membuahkan hasil yang paling kecil juga.

Penyebabnya adalah karena inisialisasi zero mulai dalam keadaan yang cukup jauh dari nilai bobot yang sebenarnya dibutuhkan. Pemilihan bobot secara acak pada metode yang lain membuatnya memberikan hasil yang lebih baik. Pada metode Xavier dan He, walaupun basisnya menggunakan metode normal, mempertimbangkan jumlah *input* dan *output* untuk *layer*, sehingga memberikan nilai yang lebih cocok.

2.2.5. Pengaruh Regularisasi

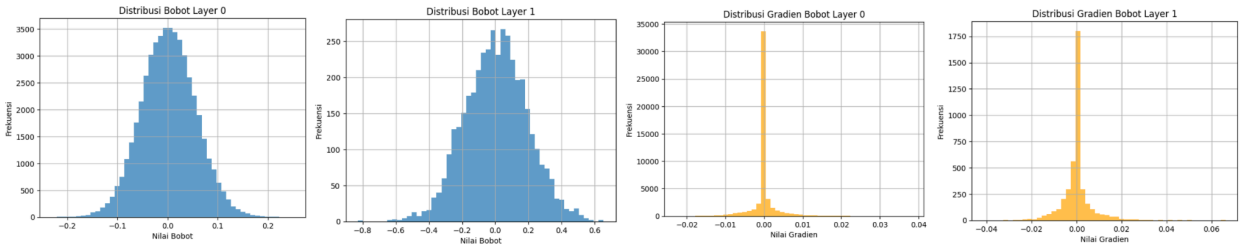
Metode regularisasi yang diimplementasikan adalah L1 dan L2. Parameter lain yang digunakan pada model ini adalah dua *hidden layer* dengan *width* 64, *learning rate* 0.01, inisialisasi bobot He, dan fungsi aktivasi ReLU. Berikut grafik hasil percobaannya:



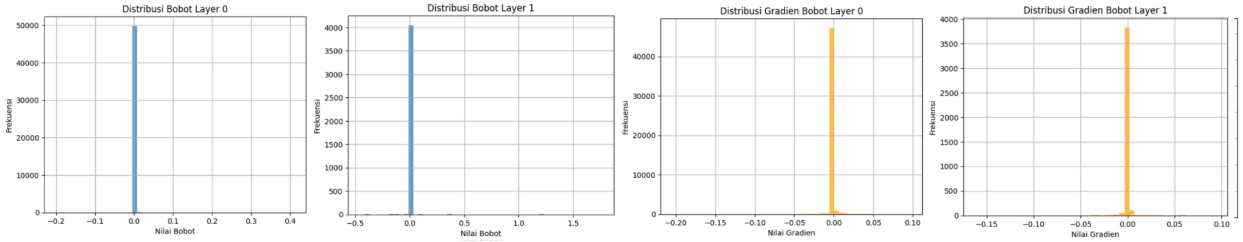
Gambar 2.2.5.1 Grafik percobaan perbandingan regularisasi

```
Akurasi FFNN untuk berbagai regularisasi:
Regularisasi none: 0.9474
Regularisasi l1: 0.8399
Regularisasi l2: 0.9139
```

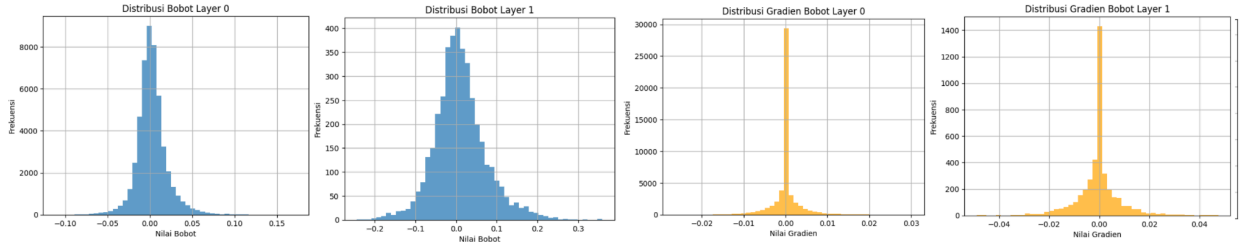
Gambar 2.2.5.2 Akurasi percobaan perbandingan regularisasi



Gambar 2.2.5.3 Distribusi bobot dan gradien bobot percobaan tanpa regularisasi



Gambar 2.2.5.4 Distribusi bobot dan gradien bobot percobaan regularisasi L1



Gambar 2.2.5.5 Distribusi bobot dan gradien bobot percobaan regularisasi L2

Model yang dijalankan tanpa regularisasi memberikan hasil yang terbaik, sedangkan regularisasi L1 memberikan hasil yang terburuk.

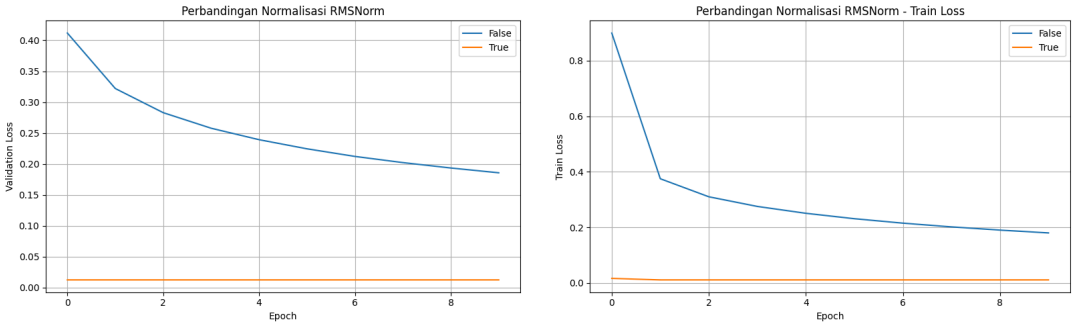
Tujuan dari regularisasi adalah untuk mencegah *overfitting* dan membuat model menjadi lebih sederhana. Penalti dari regularisasi L1 adalah penjumlahan

bobot yang ada. Metode itu dapat mengurangi bobot menjadi nol. Penalti dari regularisasi L2 adalah penjumlah pangkat dua bobot yang ada. Metode itu dapat membuat bobot mendekati nol, namun tidak sama dengan nol.

Regularisasi cocok digunakan jika terjadi *overfitting* pada model. Jika tidak, maka lebih baik untuk dihindari.

2.2.6. Pengaruh Normalisasi RMSNorm

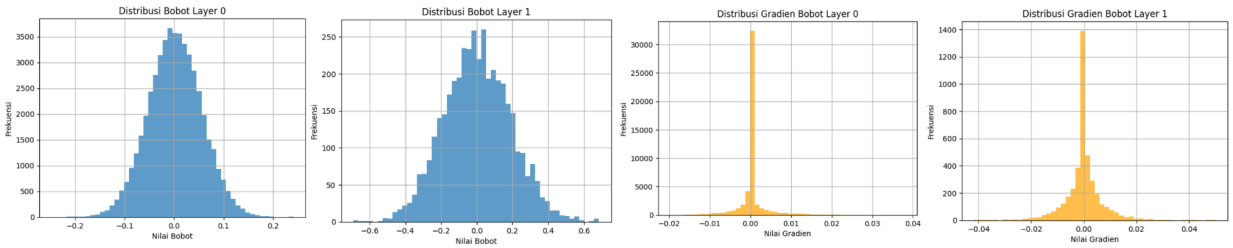
Metode normalisasi yang diimplementasikan adalah RMSNorm. Parameter lain yang digunakan pada model ini adalah dua *hidden layer* dengan *width* 64, *learning rate* 0.01, inisialisasi bobot He, dan fungsi aktivasi ReLU. Berikut grafik hasil percobaannya:



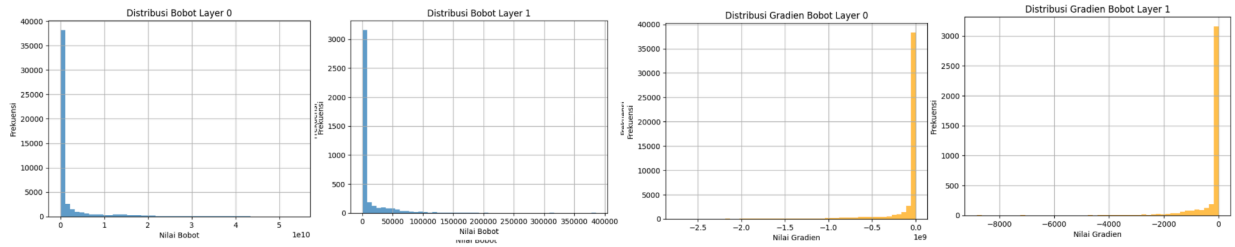
Gambar 2.2.6.1 Grafik percobaan perbandingan normalisasi RMSNorm

```
Akurasi FFNN dengan dan tanpa RMSNorm:
RMSNorm=False: 0.9480
RMSNorm=True: 0.1024
```

Gambar 2.2.6.2 Akurasi percobaan perbandingan normalisasi RMSNorm



Gambar 2.2.6.3 Distribusi bobot dan gradien bobot percobaan tanpa normalisasi



Gambar 2.2.6.4 Distribusi bobot dan gradien bobot percobaan normalisasi RMSNorm

Normalisasi memberikan hasil yang jauh lebih buruk daripada tanpa normalisasi. Tujuan normalisasi adalah untuk menghindari kasus gradien berubah terlalu jauh dan mempercepat proses konvergensi. Pada kasus ini, hal itu tidak terjadi.

2.2.7. Perbandingan dengan *Library Sklearn*

Model FFNN yang telah dibuat dibandingkan dengan MLPClassifier pada sklearn. Parameter model FFNN adalah dua *hidden layer* dengan jumlah *neuron* masing-masing 128 dan 64, fungsi aktivasi ReLU, *learning rate* 0.01, dan metode inisialisasi He. Parameter model MLPClassifier dibuat serupa mungkin dengan model FFNN.

Hasilnya, akurasi MLPClassifier adalah 0.973, sedangkan FFNN adalah 0.967. Nilai akurasi kedua model tidak terlalu beda jauh, walaupun model FFNN memberikan akurasi yang lebih rendah. Hal tersebut terjadi kemungkinan karena perbedaan metode optimasi yaitu pada model FFNN menggunakan *Stochastic Gradient Descent* (SGD), sedangkan pada MLPClassifier biasanya menggunakan *Adaptive Moment Estimation* (Adam) yang dikenal cepat, stabil, dan efektif mengatasi berbagai masalah saat pelatihan model.

BAB III
PENUTUP

3.1. Kesimpulan

FFNN adalah implementasi *neural network* yang menggunakan *forward propagation*, *backward propagation*, dan *weight update* dalam proses pelatihan modelnya. Beberapa parameter yang digunakan dalam pelatihan dapat mempengaruhi kinerja model yang dihasilkan secara signifikan. Parameter tersebut adalah jumlah *hidden layer*, *activation function*, dan inisialisasi bobot. Jumlah *hidden layer* yang lebih banyak, *activation function* selain sigmoid, dan inisialisasi bobot Xavier dan He dapat meningkatkan kinerja model FFNN.

Model FFNN yang telah dibuat belum setara dengan implementasi MLPClassifier dari *library* sklearn, namun hasilnya masih cukup bagus. Implementasi regularisasi dan normalisasi belum sesuai dengan ekspektasi, sehingga ada kemungkinan bahwa teknik itu tidak sesuai dengan *dataset* atau terdapat kekurangan dalam implementasi.

3.2. Saran

Saran untuk implementasi FFNN *from scratch* ini adalah sebagai berikut:

1. Eksplorasi lebih lanjut atas parameter-parameter yang terlibat dalam pelatihan FFNN.
2. Meninjau kembali implementasi fungsi-fungsi yang terlibat dalam pelatihan FFNN agar lebih sesuai dengan model MLPClassifier.
3. Meninjau kembali implementasi regularisasi dan normalisasi agar lebih sesuai dengan ekspektasi tujuan penggunaan metode-metode tersebut.

PEMBAGIAN TUGAS

NIM	Nama	Tugas
13522096	Novelya Putri Ramadhani	○ Desain struktur kelas FFNN dan Layer ○ Implementasi forward, backward propagation ○ Implementasi fungsi loss ○ Laporan
13522102	Hayya Zuhailii Kinasih	○ Implementasi seluruh bonus ○ Pengujian dan validasi output model ○ Eksperimen & analisis pengaruh depth, width, learning rate, dsb. ○ Laporan
13522104	Diana Tri Handayani	○ Buat fungsi training (batch, epoch, verbose) ○ Eksperimen & analisis pengaruh depth, width, learning rate, dsb. ○ Membuat grafik bobot & gradien ○ Laporan

DAFTAR PUSTAKA

[1] Scikit-learn developers. sklearn.neural_network.MLPClassifier. scikit-learn, https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html. Accessed 10 Apr, 2025.

[2] Tim Pengajar IF3270. *Modul FFNN*. Accessed March 27, 2025.