LAPORAN TUGAS BESAR 1

IF3270 Pembelajaran Mesin

Feedforward Neural Network

Disusun oleh:

Ahmad Naufal Ramadan (13522005)

Yusuf Ardian Sandi (13522015)

Rayendra Althaf T. N. (13522107)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2025

BAB I
DESKRIPSI PERSOALAN

Modul Feedforward Neural Network (FFNN) yang diimplementasikan harus memenuhi beberapa ketentuan utama. Pertama, modul harus dapat menerima jumlah neuron pada setiap layer, termasuk input layer dan output layer. Selain itu, setiap layer harus dapat menggunakan berbagai fungsi aktivasi, yaitu Linear, ReLU, Sigmoid, Hyperbolic Tangent (tanh), dan Softmax.

Selain fungsi aktivasi, model juga harus mendukung beberapa fungsi loss, yaitu Mean Squared Error (MSE), Binary Cross-Entropy, dan Categorical Cross-Entropy. Perlu diperhatikan bahwa Binary Cross-Entropy merupakan kasus khusus dari Categorical Cross-Entropy dengan jumlah kelas sebanyak dua, dan logaritma yang digunakan dalam perhitungan berbasis log natural (ln, basis e).

Untuk inisialisasi bobot, modul harus memiliki beberapa metode berikut:

- Zero Initialization, di mana semua bobot dimulai dari nol.
- Random Initialization dengan distribusi uniform, dengan parameter lower bound, upper bound, dan seed untuk reproducibility.
- Random Initialization dengan distribusi normal, dengan parameter mean, variance, dan seed.

Setelah inisialisasi, model yang dibuat harus dapat menyimpan bobot dan bias dari setiap neuron, serta gradien bobot dan bias. Selain itu, model harus memiliki metode untuk:

- Menampilkan struktur jaringan dalam bentuk graf, termasuk bobot dan gradiennya.
- Menampilkan distribusi bobot dari layer tertentu, dengan input berupa daftar indeks layer yang ingin divisualisasikan.
- Menampilkan distribusi gradien bobot dengan cara serupa.
- Menyimpan dan memuat model (save & load).

Dari segi implementasi, model harus mendukung forward propagation, yang dapat menerima input dalam bentuk batch. Selain itu, backward propagation harus diterapkan untuk menghitung gradien dengan menggunakan chain rule, di mana setiap fungsi aktivasi memiliki turunan pertamanya yang harus diperhitungkan. Begitu juga dengan fungsi loss, yang turunannya dihitung terhadap bobot menggunakan aturan rantai (chain rule).

Untuk pembaruan bobot, digunakan metode gradient descent, yang memperbarui bobot berdasarkan gradien yang telah dihitung. Dalam proses pelatihan model, beberapa parameter yang harus diterima meliputi:

- Batch size
- Learning rate
- Jumlah epoch
- Verbose (0 untuk tanpa output, 1 untuk menampilkan progress bar, training loss, dan validation loss)

Sebagai hasil akhir, proses pelatihan harus mengembalikan riwayat pelatihan, yang mencatat training loss dan validation loss pada setiap epoch.

BAB II
PEMBAHASAN

1. Penjelasan Implementasi

1.1. Deskripsi Kelas beserta Deskripsi Atribut dan Methodnya

Tabel 1.1.1 Tabel Deskripsi Kelas FFNN

Atribut / Method	**Deskripsi**
layers: Layer[]	Layer-layer pada FFNN
loss_function: LossFunction	Loss function yang dipakai pada setiap layer kecuali output layer
learning_rate: float	Learning rate saat training
batch_size: int	Jumlah data dalam satu batch saat training
epochs: int	Jumlah epoch saat training
verbose: bool	Opsi menampilkan progres saat training akan ditampilkan Nilai true digunakan untuk menampilkan
random_state: int	Seed untuk randomize
l1: float	Nilai lambda untuk regularisasi L1
l2: float	Nilai lambda untuk regularisasi L2
train_losses: list[float]	Nilai loss pada model terhadap train data sebelum epoch terakhir
val_losses: list[float]	Nilai loss pada model terhadap validation data sebelum epoch terakhir
weights_grad: double[]	Gradient weights pada epoch terakhir
biases_grad: double[]	Gradient bias pada epoch terakhir
__init__(): FFNN	Mengembalikan nilai
show_graph()	Menampilkan visualisasi FFNN
plot_weights()	Menampilkan distribusi weights dan biases untuk layer yang dipilih
plot_gradients()	Menampilkan distribusi gradien weight dan

	gradien bias untuk layer yang dipilih
plot_loss_curve()	Menampilkan tabel loss terhadap epoch
save_model()	Menyimpan model dalam bentuk pickle
load_model()	Memuat model dari file pickle
fit()	Melakukan fit model terhadap data yang diberikan dengan melakukan backward sebanyak epoch dalam bentuk minibatch
calculate_loss()	Menghitung loss dengan tambahan penalty jika parameter l1 dan l2 > 0
forward()	Melakukan inferensi terhadap satu atau lebih nilai x
backward()	Melakukan backward propagation terhadap data yang sudah di forward
predict()	Melakukan forward
set_weights()	Melakukan set nilai ke semua weights pada semua layer
set_biases()	Melakukan set nilai ke semua biases pada semua layer

Tabel 1.1.2 Tabel Deskripsi Kelas Layer

Atribut / Method	**Deskripsi**
input_size: int	Nilai di dalam input neuron
output_size: int	Nilai di dalam in output neuron
activation: ActivationFunction	Fungsi aktivasi yang digunakan pada layer
weight_initializer: WeightInitialization	Metode inisialisasi bobot
weights: double[]	Nilai bobot
biases: double[]	Bobot pada bias
old_weights: double[]	Weight sebelum diupdate
old_biases: double[]	Bias sebelum diupdate
output_value: double[]	Nilai output
Input_value: double[]	Nilai input

z: double[]	Net value pada output layer

Tabel 1.1.3 Tabel Deskripsi Kelas Activation

Atribut / Method	**Deskripsi**
get_activation_from_type(): Activation	Mengembalikan activation function yang sesuai
activate(): double	Mengubah nilai net menjadi nilai yang sudah diaktivasi
derivative(): double	Menurunkan nilai yang diberikan sesuai fungsi aktivasi
get_activation(): ActivationFunction	Mengembalikan tipe fungsi aktivasi

Tabel 1.1.4 Tabel Deskripsi Kelas Loss

Atribut / Method	**Deskripsi**
get_loss_from_type(): Loss	Mengembalikan loss function yang sesuai
calculate(): double	Menghitung nilai loss berdasarkan nilai output (y_pred) dan nilai sesungguhnya (y_true)
derivative(): double	Menurunkan nilai sesuai dengan fungsi loss
get_loss_type(): LossFunction	Mengembalikan tipe fungsi loss

Tabel 1.1.5 Tabel Deskripsi Kelas WeightInitialization

Atribut / Method	**Deskripsi**
get_weight_initializer_from_type(): WeightInitializer	Mengembalikan weight initializer yang sesuai
initialize()	Mengisi weight pada layer sesuai tipe inisialisasi
get_initializer_type(): LossFunction	Mengembalikan tipe inisialisasi weight

1.2. Penjelasan forward propagation

Proses forward propagation adalah proses yang digunakan untuk melakukan inferensi terhadap suatu input. Proses ini akan mengembalikan nilai perkiraan target berdasarkan model dan input yang diberikan. Forward propagation dilakukan dengan memberikan nilai input ke layer pertama dan meneruskannya sampai layer terakhir.

Proses pada setiap layer dimulai dengan melakukan dot operation antara nilai input dan weights yang dibentuk dalam format berikut:
$input_i$: nilai pada node input ke i

$weight_{ij}$: nilai weight untuk input node ke-i yang ditujukan ke output node ke-j

Nilai input diatas dimaksudkan jika input hanya ada satu instance dan input tersebut disimpan dalam satu baris. Jika terdapat lebih dari satu input akan ditambahkan pada baris baru. Data hasil dot tersebut kemudian akan ditambahkan dengan nilai bias dan dioperasikan dengan fungsi aktivasi untuk mendapatkan nilai output. Nilai tersebut kemudian akan dijadikan masukan untuk layer berikutnya. Proses tersebut diulang hingga sampai ke layer terakhir yang memberikan nilai inferensi dalam bentuk nilai output layer tersebut.

1.3. Penjelasan Backward Propagation dan Weight Update

Dalam program kami, proses training dilakukan dengan proses fit. Proses ini akan membagi data input menjadi data training dan validation. Data training kemudian dibagi lagi sehingga terdapat sejumlah subset yang ukuran datanya sama atau lebih kecil dibandingkan ukuran batch pada model. Pada setiap subset tersebut kemudian akan dilakukan proses forward diikuti dengan backward. Backward sendiri dilakukan dengan melakukan langkah-langkah berikut:

1. Menghitung δloss/δactivation pada layer terakhir
   Dilakukan dengan menghitung

   $\delta loss/\delta o \,.\, \delta o/\delta activation$

   Untuk memudahkan nilai diatas akan disebut sebagai delta, setiap output node memiliki nilai delta yang sesuai. Delta nantinya juga akan diubah nilainya seiring berjalannya proses.
2. Melakukan iterasi dari layer terakhir hingga layer pertama dengan melakukan hal berikut
   - Menghitung gradien weight dan gradien bias
     Gradien weight didapatkan dengan mengalikan nilai input node dengan nilai delta, sedangkan gradien bias didapatkan dengan mengalikan nilai delta dengan 1.
     $gradientweight_{ij} = delta_j \,.\, input_i$
     $gradientbias_j = delta_j$
   - Mengurangi weight dengan nilai gradient
     $weight_{ij} -= gradientweight_{ij}$
     Keterangan: $weight_{ij}$ mewakili weight dari node input i ke node output j
   - Mengubah nilai delta menjadi

     $\delta o_i / \delta activation_i \,.\, \sum delta_j \,.\, weight_{ij}$

Untuk i adalah setiap input node dan j adalah setiap output node. Proses ini tidak dilakukan pada layer pertama.

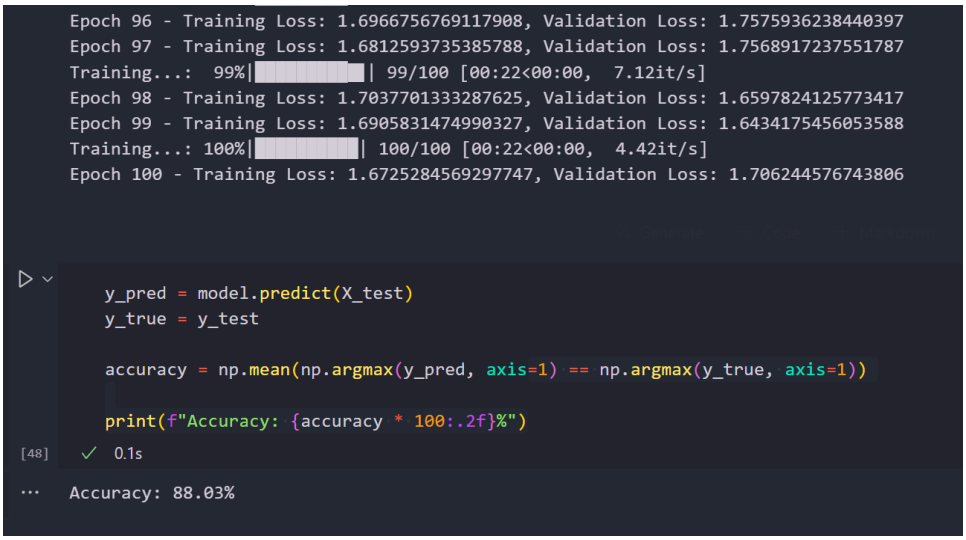
Proses pembagian data dan eksekusi backward akan dilakukan sebanyak nilai epoch pada model. Selama pengulangan tersebut juga akan disimpan nilai training loss dan validation loss. Nilai gradien pada epoch terakhir juga akan disimpan di dalam model.

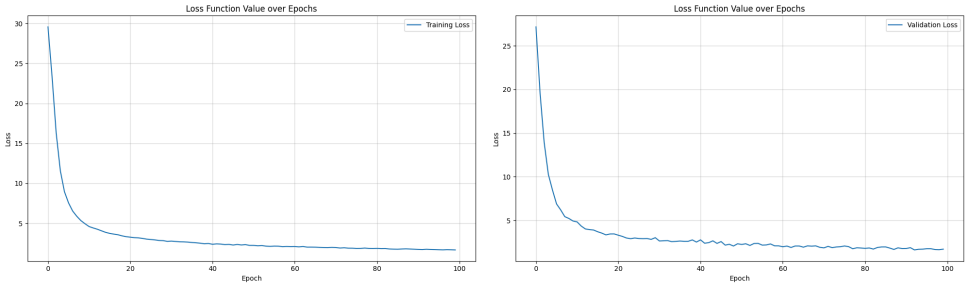
2. Hasil Pengujian

Parameter Default:

- Fungsi Aktivasi: ReLU untuk semua hidden layer, Softmax untuk output layer
- Fungsi Loss: Categorical Cross Entropy
- Inisialisasi Bobot: Xavier
- Learning Rate: 0.0001
- Ukuran Batch: 256
- Epoch: 100
- Verbose: True
- Random State: 42

2.1. Pengaruh depth dan width

Depth	FFNN Implementasi
2	



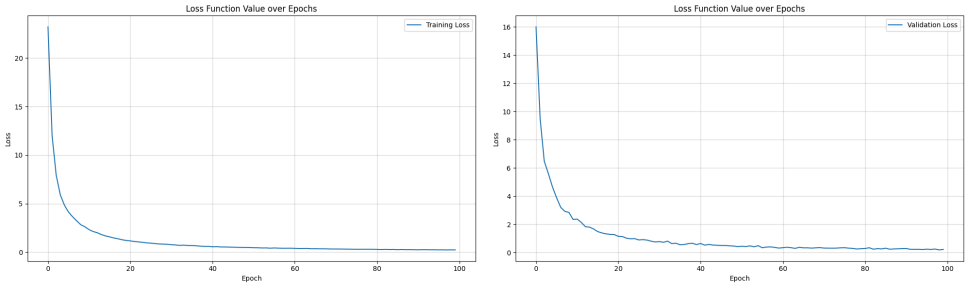
3

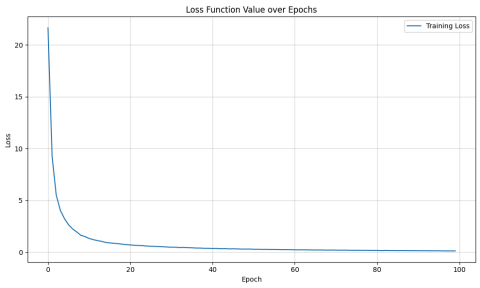
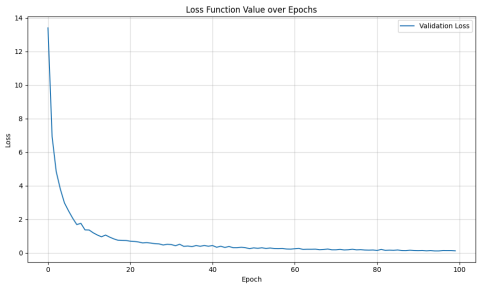
```
Epoch 96 - Training Loss: 0.22786766126742902, Validation Loss: 0.24071917159628156
Training...:  97%|▉| 97/100 [00:24<00:00,  4.18it/s]
Epoch 97 - Training Loss: 0.2309946782558585, Validation Loss: 0.2197282314381602
Training...:  98%|▉| 98/100 [00:24<00:00,  4.08it/s]
Epoch 98 - Training Loss: 0.22141769660888183, Validation Loss: 0.25030385785179715
Training...:  99%|▉| 99/100 [00:25<00:00,  4.16it/s]
Epoch 99 - Training Loss: 0.2290010259272571, Validation Loss: 0.18695369871854464
Training...: 100%|█| 100/100 [00:25<00:00,  3.96it/s]
Epoch 100 - Training Loss: 0.22273040494650606, Validation Loss: 0.22176651843255993
```

```
    y_pred = model.predict(X_test)
    y_true = y_test

    accuracy = np.mean(np.argmax(y_pred, axis=1) == np.argmax(y_true, axis=1))

    print(f"Accuracy: {accuracy * 100:.2f}%")
```

[30] ✓ 0.1s

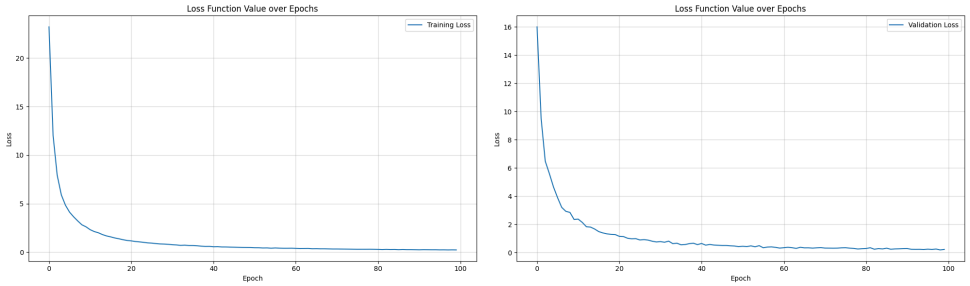
```
Accuracy: 90.35%
```



4	Epoch 96 - Training Loss: 0.1253525056952774, Validation Loss: 0.11036727832303003
	Training...: 97%| | 97/100 [00:25<00:00, 3.83it/s]
	Epoch 97 - Training Loss: 0.11827627561315501, Validation Loss: 0.13304334157986822
	Training...: 98%| | 98/100 [00:25<00:00, 3.83it/s]
	Epoch 98 - Training Loss: 0.11507917523970684, Validation Loss: 0.13065334160190936
	Training...: 99%| | 99/100 [00:26<00:00, 3.88it/s]
	Epoch 99 - Training Loss: 0.11331744495668108, Validation Loss: 0.13142138597842262
	Training...: 100%| | 100/100 [00:26<00:00, 3.79it/s]
	Epoch 100 - Training Loss: 0.11691651317012489, Validation Loss: 0.11700065877369668
	
	Generate Code Markdown
	
	`y_pred = model.predict(X_test)`
	`y_true = y_test`
	
	`accuracy = np.mean(np.argmax(y_pred, axis=1) == np.argmax(y_true, axis=1))`
	
	`print(f"Accuracy: {accuracy * 100:.2f}%")`
	[12] ✓ 0.1s
	… Accuracy: 91.14%
	
	 
	 

Width	FFNN Implementasi

50	Epoch 96 - Training Loss: 0.227867661267429O2, Validation Loss: 0.24071917159628156
	Training...: 97%| | 97/100 [00:24<00:00, 4.18it/s]
	Epoch 97 - Training Loss: 0.2309946782558585, Validation Loss: 0.2197282314381602
	Training...: 98%| | 98/100 [00:24<00:00, 4.08it/s]
	Epoch 98 - Training Loss: 0.22141769660888183, Validation Loss: 0.25030385785179715
	Training...: 99%| | 99/100 [00:25<00:00, 4.16it/s]
	Epoch 99 - Training Loss: 0.2290010259272571, Validation Loss: 0.18695369871854464
	Training...: 100%| | 100/100 [00:25<00:00, 3.96it/s]
	Epoch 100 - Training Loss: 0.22273040494650606, Validation Loss: 0.22176651843255993
	
	▷
	`y_pred = model.predict(X_test)`
	`y_true = y_test`
	
	`accuracy = np.mean(np.argmax(y_pred, axis=1) == np.argmax(y_true, axis=1))`
	
	`print(f"Accuracy: {accuracy * 100:.2f}%")`
	[30] ✓ 0.1s
	… Accuracy: 90.35%



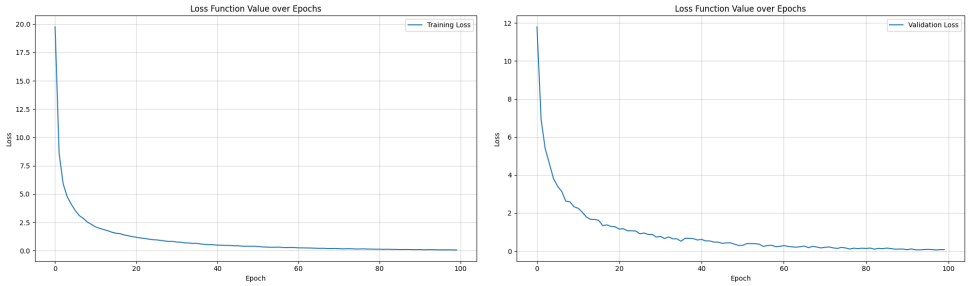
75

```
Epoch 96 - Training Loss: 0.07110751663781692, Validation Loss: 0.0896231195922652
Training...:  97%|▉| 97/100 [00:34<00:00,  3.90it/s]
Epoch 97 - Training Loss: 0.07014152532777598, Validation Loss: 0.07520068627363151
Training...:  98%|▉| 98/100 [00:34<00:00,  3.97it/s]
Epoch 98 - Training Loss: 0.07333276726700944, Validation Loss: 0.05851264625914087
Training...:  99%|▉| 99/100 [00:35<00:00,  3.98it/s]
Epoch 99 - Training Loss: 0.06583234875819856, Validation Loss: 0.07230284842756317
Training...: 100%|▉| 100/100 [00:35<00:00,  2.82it/s]
Epoch 100 - Training Loss: 0.06334134476544917, Validation Loss: 0.0757476801060239
```

```
y_pred = model.predict(X_test)
y_true = y_test

accuracy = np.mean(np.argmax(y_pred, axis=1) == np.argmax(y_true, axis=1))

print(f"Accuracy: {accuracy * 100:.2f}%")
```

[66] ✓ 0.1s

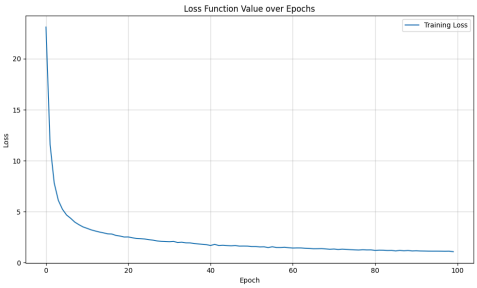
```
Accuracy: 91.42%
```

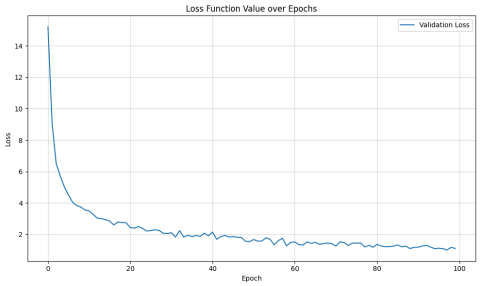


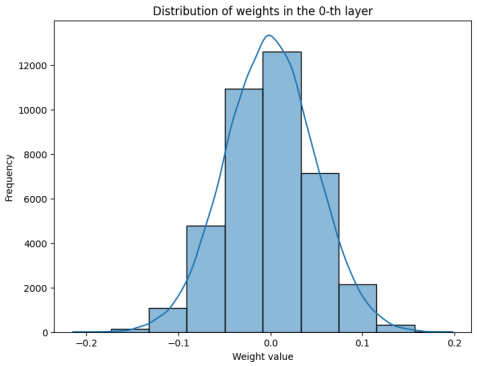
100	

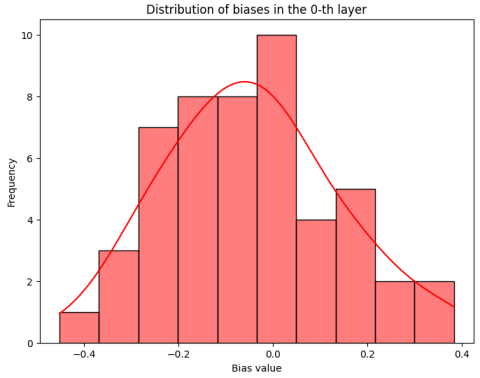
2.2. Pengaruh fungsi aktivasi hidden layer

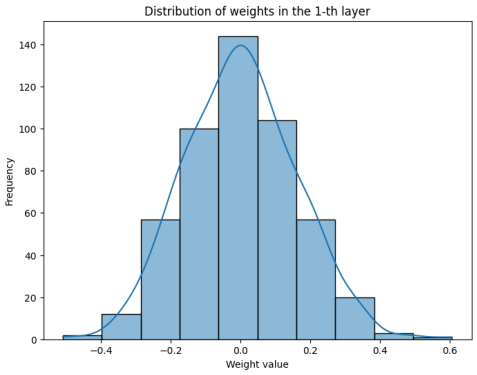
Fungsi Aktivasi	FFNN Implementasi
Linear	

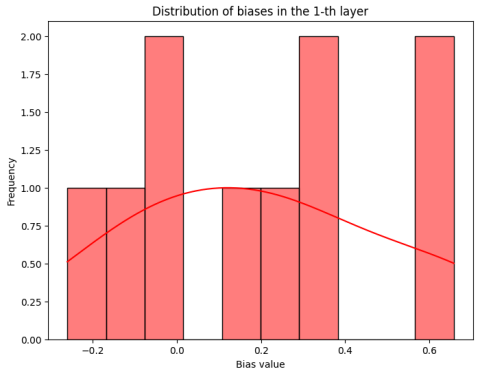

Loss Function Value over Epochs
Training Loss
Loss
Epoch

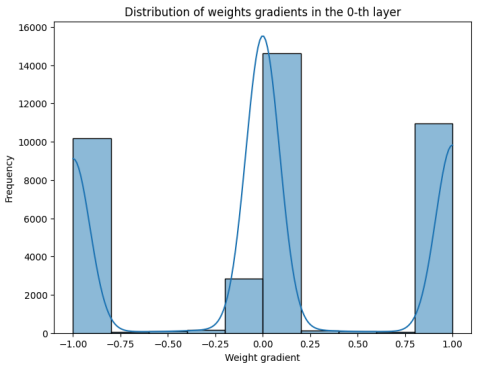

Loss Function Value over Epochs
Validation Loss
Loss
Epoch

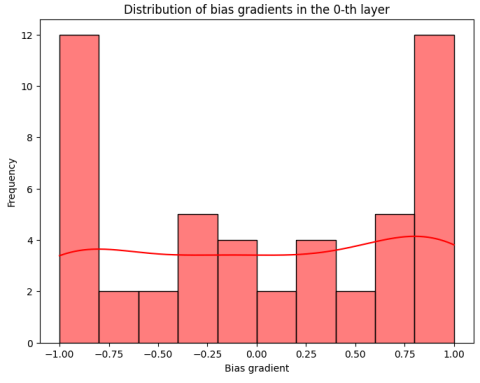

Distribution of weights in the 0-th layer
Frequency
Weight value

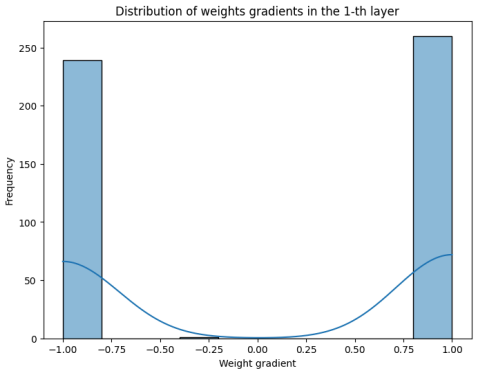

Distribution of biases in the 0-th layer
Frequency
Bias value

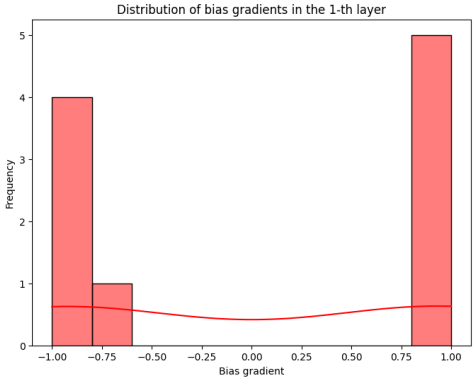

Distribution of weights in the 1-th layer
Frequency
Weight value


Distribution of biases in the 1-th layer
Frequency
Bias value


Distribution of weights gradients in the 0-th layer
Frequency
Weight gradient


Distribution of bias gradients in the 0-th layer
Frequency
Bias gradient





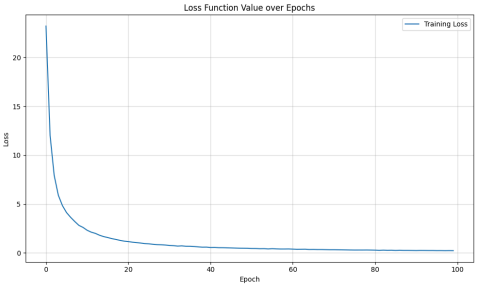
ReLU

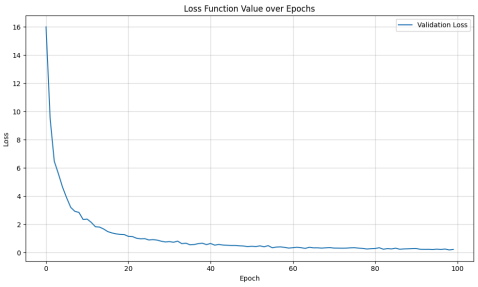
```
Epoch 96 - Training Loss: 0.22786766126742902, Validation Loss: 0.24071917159628156
Training...:  97%|▉| 97/100 [00:24<00:00,  4.18it/s]
Epoch 97 - Training Loss: 0.2309946782558585, Validation Loss: 0.2197282314381602
Training...:  98%|▉| 98/100 [00:24<00:00,  4.08it/s]
Epoch 98 - Training Loss: 0.22141769660888183, Validation Loss: 0.25030385785179715
Training...:  99%|▉| 99/100 [00:25<00:00,  4.16it/s]
Epoch 99 - Training Loss: 0.2290010259272571, Validation Loss: 0.18695369871854464
Training...: 100%|▉| 100/100 [00:25<00:00,  3.96it/s]
Epoch 100 - Training Loss: 0.22273040494650606, Validation Loss: 0.22176651843255993
```

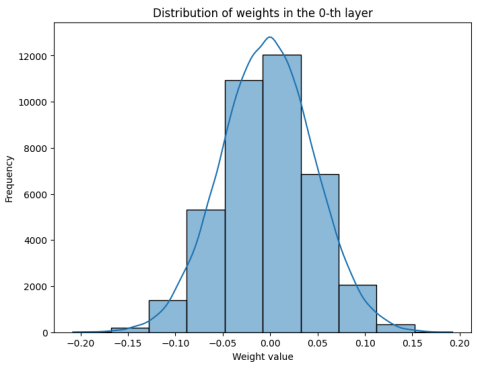
```
y_pred = model.predict(X_test)
y_true = y_test

accuracy = np.mean(np.argmax(y_pred, axis=1) == np.argmax(y_true, axis=1))

print(f"Accuracy: {accuracy * 100:.2f}%")
```

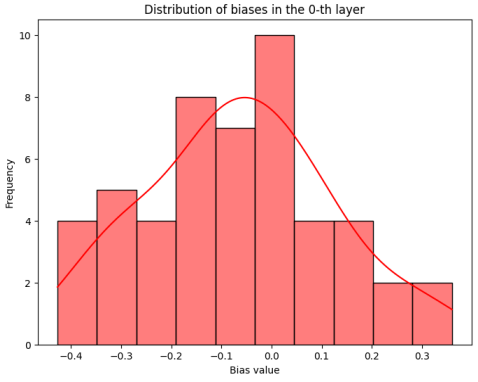
[30] ✓ 0.1s

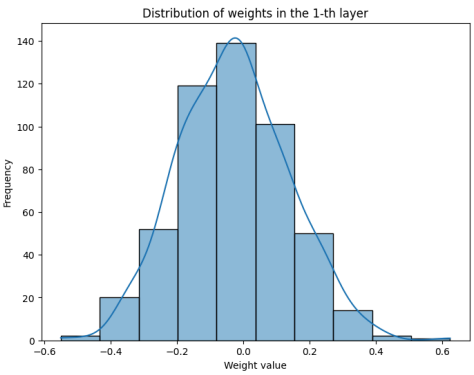
```
Accuracy: 90.35%
```

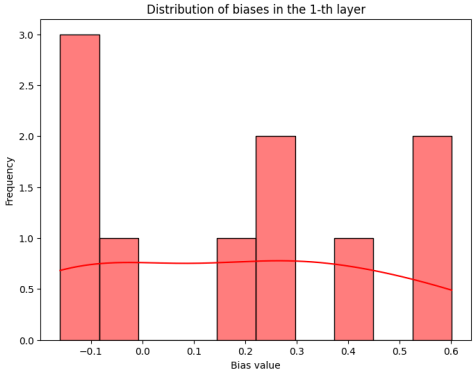


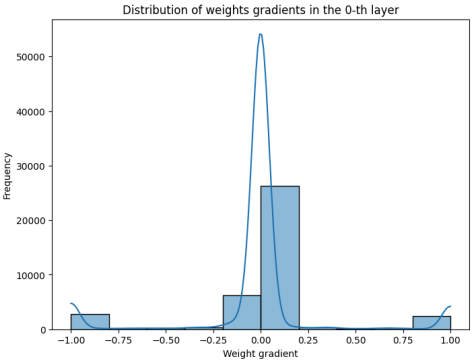


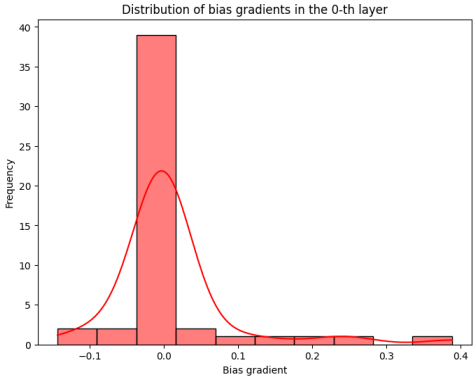


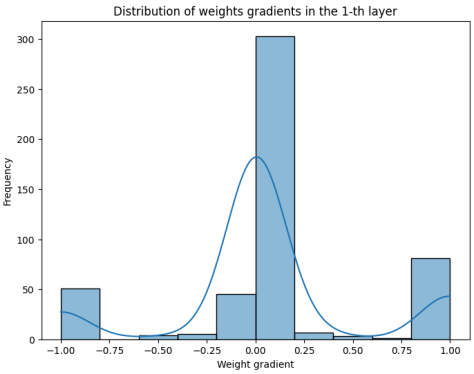


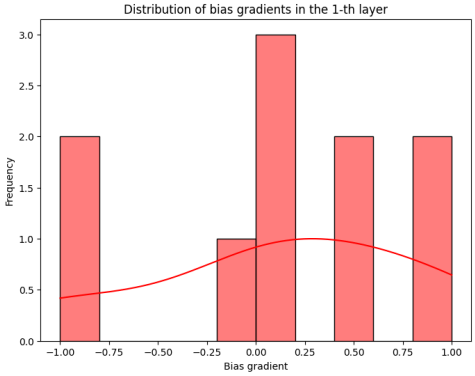

Distribution of weights in the 1-th layer
Frequency
Weight value


Distribution of biases in the 1-th layer
Frequency
Bias value


Distribution of weights gradients in the 0-th layer
Frequency
Weight gradient


Distribution of bias gradients in the 0-th layer
Frequency
Bias gradient


Distribution of weights gradients in the 1-th layer
Frequency
Weight gradient


Distribution of bias gradients in the 1-th layer
Frequency
Bias gradient

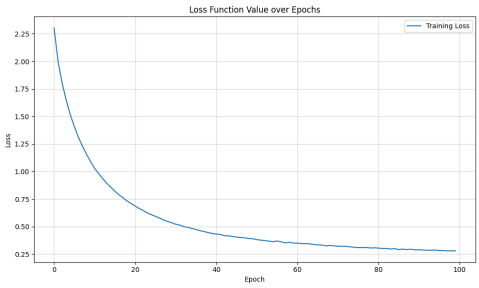
Sigmoid

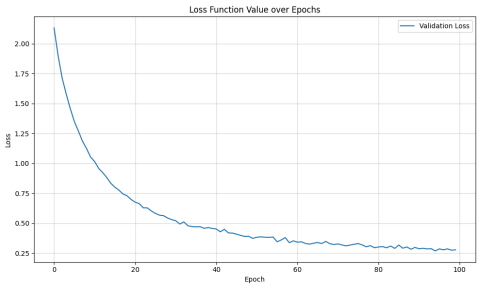
```
Epoch 96 - Training Loss: 0.2818215039414519, Validation Loss: 0.2851066963108756
Training...:  97%|▉| 97/100 [00:43<00:00,  4.14it/s]
Epoch 97 - Training Loss: 0.28253586836600614, Validation Loss: 0.2765387445087387
Training...:  98%|▉| 98/100 [00:43<00:00,  4.11it/s]
Epoch 98 - Training Loss: 0.27873752335893964, Validation Loss: 0.285115103286453
Training...:  99%|▉| 99/100 [00:43<00:00,  4.13it/s]
Epoch 99 - Training Loss: 0.2797760723598396, Validation Loss: 0.273586666597221
Training...: 100%|▉| 100/100 [00:44<00:00,  2.27it/s]
Epoch 100 - Training Loss: 0.27865717702975173, Validation Loss: 0.2771286671381468
```

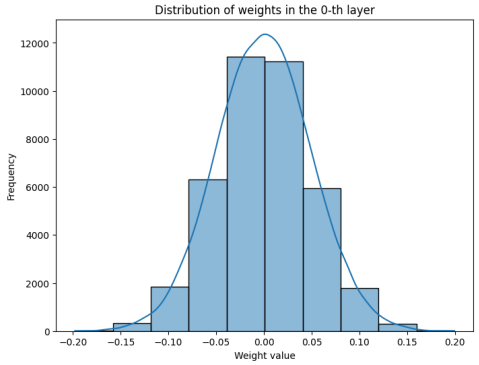
```
y_pred = model.predict(X_test)
y_true = y_test

accuracy = np.mean(np.argmax(y_pred, axis=1) == np.argmax(y_true, axis=1))

print(f"Accuracy: {accuracy * 100:.2f}%")
```

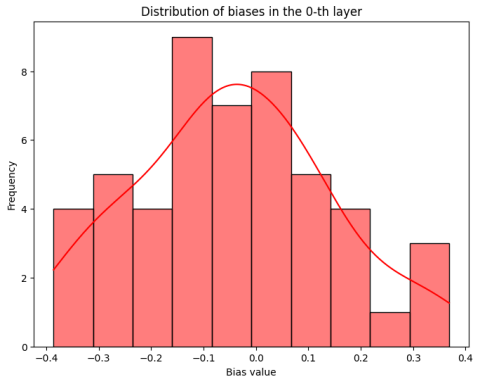
[138] ✓ 0.1s

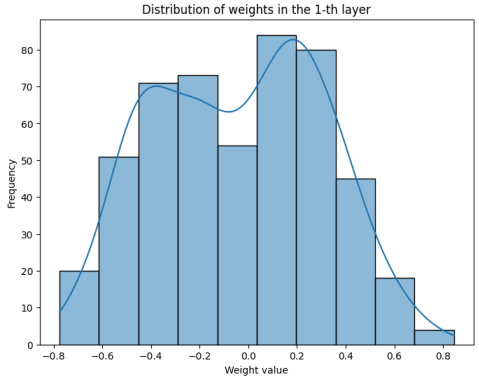
```
Accuracy: 89.27%
```

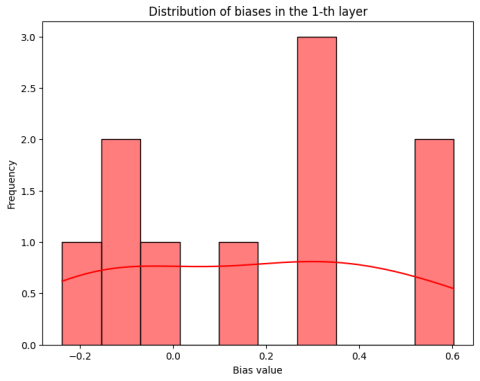


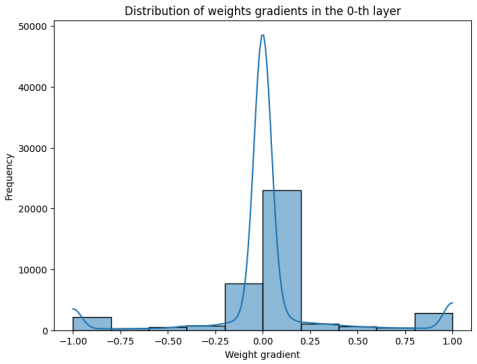


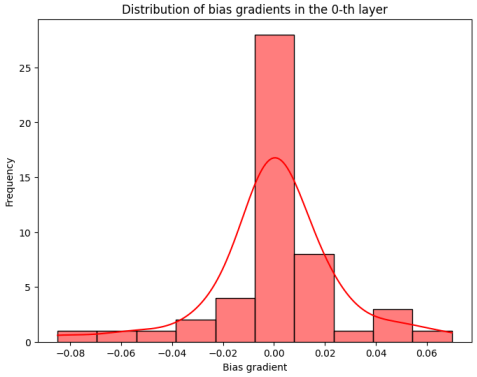


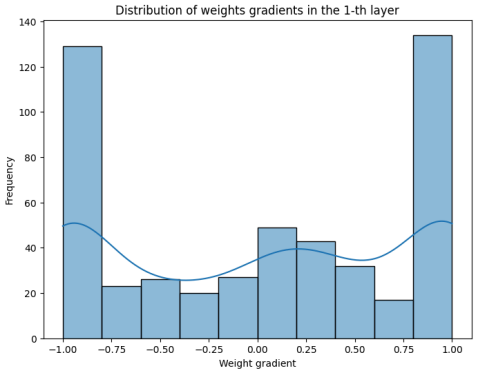


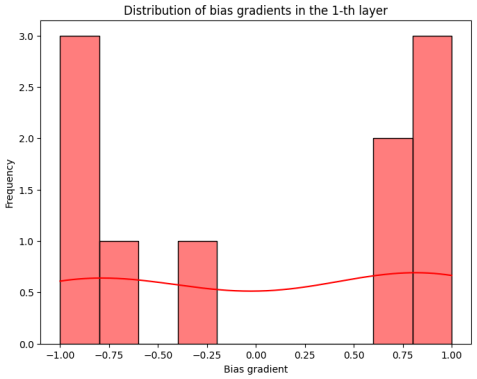












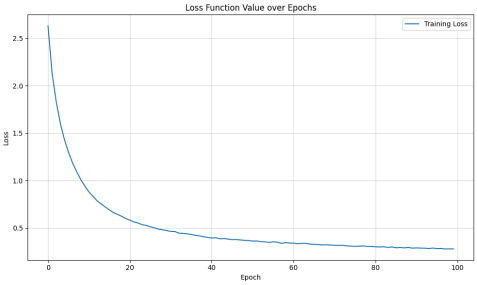
Hyperbolic Tangent (tanh)

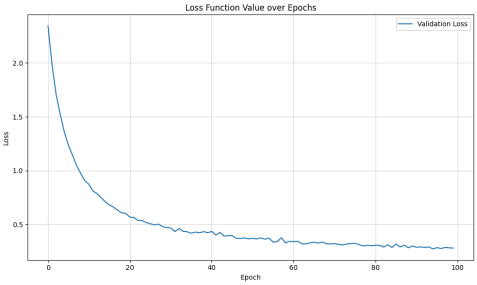
```
Epoch 96 - Training Loss: 0.2835528642402638, Validation Loss: 0.28415901703030444
Training...:  97%|▉| 97/100 [00:34<00:01,  1.91it/s]
Epoch 97 - Training Loss: 0.2848042000350585, Validation Loss: 0.275469301470724
Training...:  98%|▉| 98/100 [00:34<00:01,  1.88it/s]
Epoch 98 - Training Loss: 0.2800895191575974, Validation Loss: 0.28581459587429076
Training...:  99%|▉| 99/100 [00:35<00:00,  1.96it/s]
Epoch 99 - Training Loss: 0.2806603591134273, Validation Loss: 0.2818126417995616
Training...: 100%|▉| 100/100 [00:35<00:00,  2.81it/s]
Epoch 100 - Training Loss: 0.2803302362664339, Validation Loss: 0.27975375356878596
```

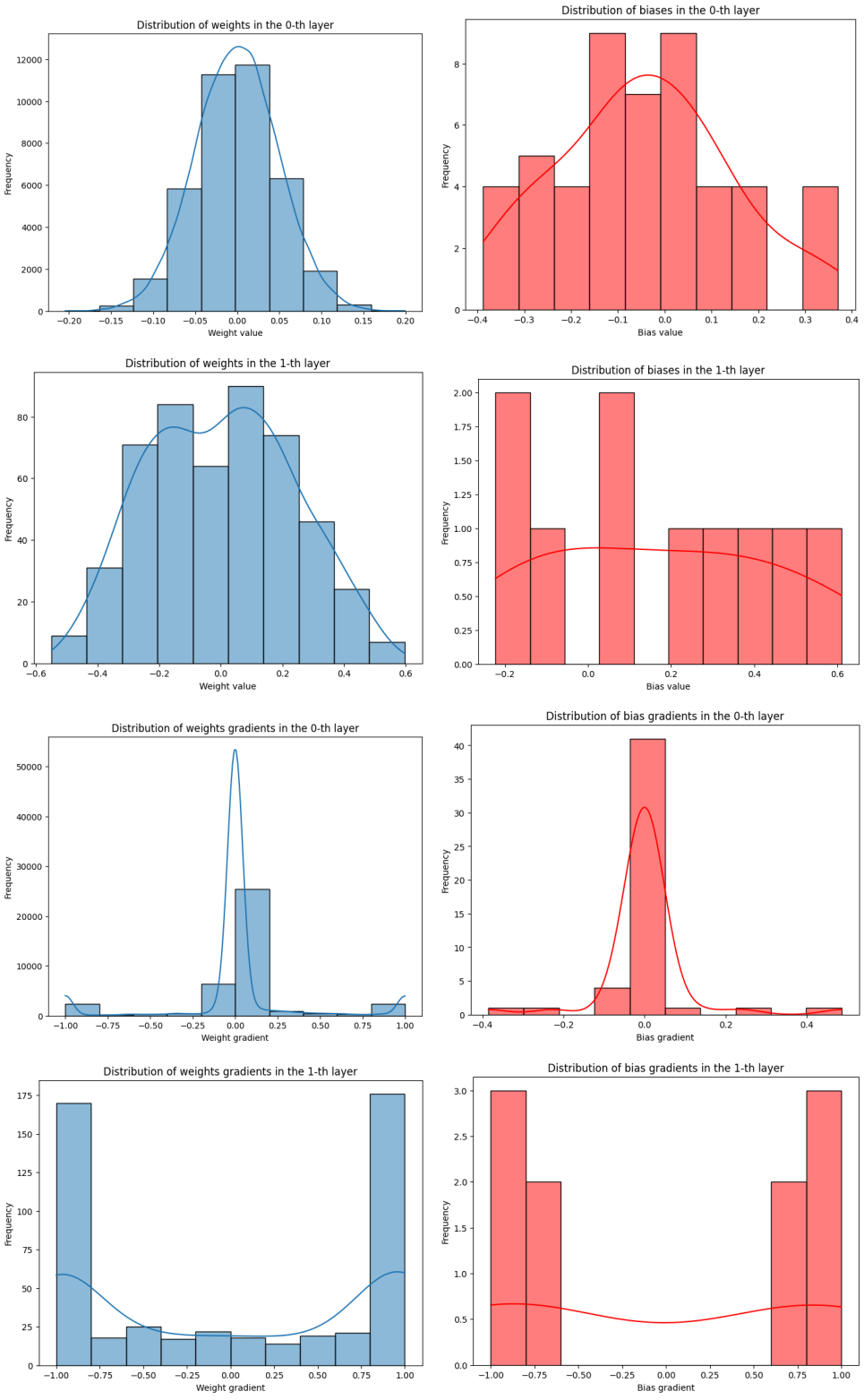
```
y_pred = model.predict(X_test)
y_true = y_test

accuracy = np.mean(np.argmax(y_pred, axis=1) == np.argmax(y_true, axis=1))

print(f"Accuracy: {accuracy * 100:.2f}%")
```

[156] ✓ 0.5s

```
Accuracy: 89.07%
```







2.3. Pengaruh learning rate

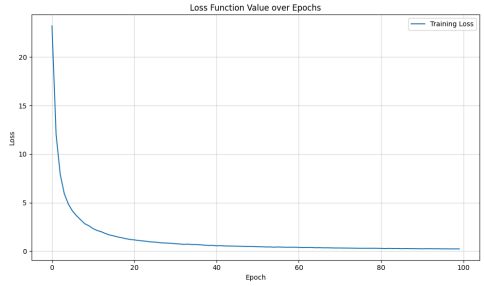
Learning Rate	FFNN Implementasi
0.0001	

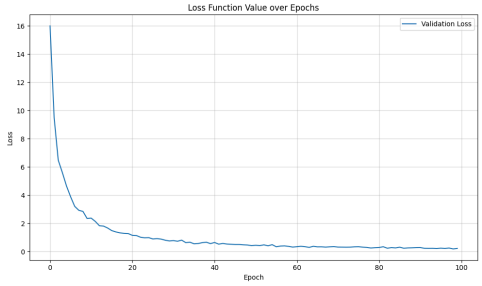
```
Epoch 96 - Training Loss: 0.22786766126742902, Validation Loss: 0.24071917159628156
Training...:  97%|▉| 97/100 [00:24<00:00,  4.18it/s]
Epoch 97 - Training Loss: 0.2309946782558585, Validation Loss: 0.2197282314381602
Training...:  98%|▉| 98/100 [00:24<00:00,  4.08it/s]
Epoch 98 - Training Loss: 0.22141769660888183, Validation Loss: 0.25030385785179715
Training...:  99%|▉| 99/100 [00:25<00:00,  4.16it/s]
Epoch 99 - Training Loss: 0.2290010259272571, Validation Loss: 0.18695369871854464
Training...: 100%|▉| 100/100 [00:25<00:00,  3.96it/s]
Epoch 100 - Training Loss: 0.22273040494650606, Validation Loss: 0.22176651843255993
```

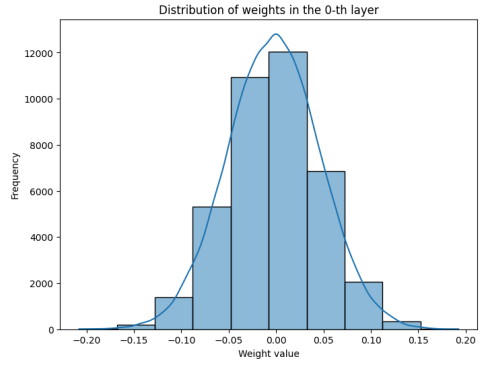
```
y_pred = model.predict(X_test)
y_true = y_test

accuracy = np.mean(np.argmax(y_pred, axis=1) == np.argmax(y_true, axis=1))

print(f"Accuracy: {accuracy * 100:.2f}%")
```

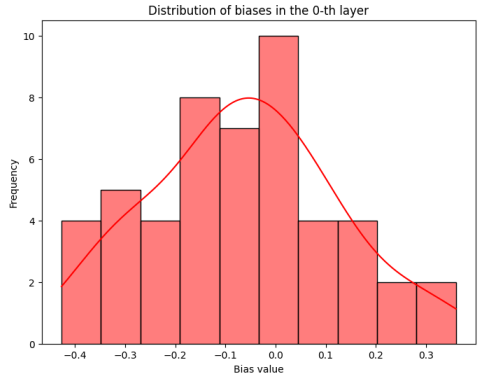
[30] ✓ 0.1s

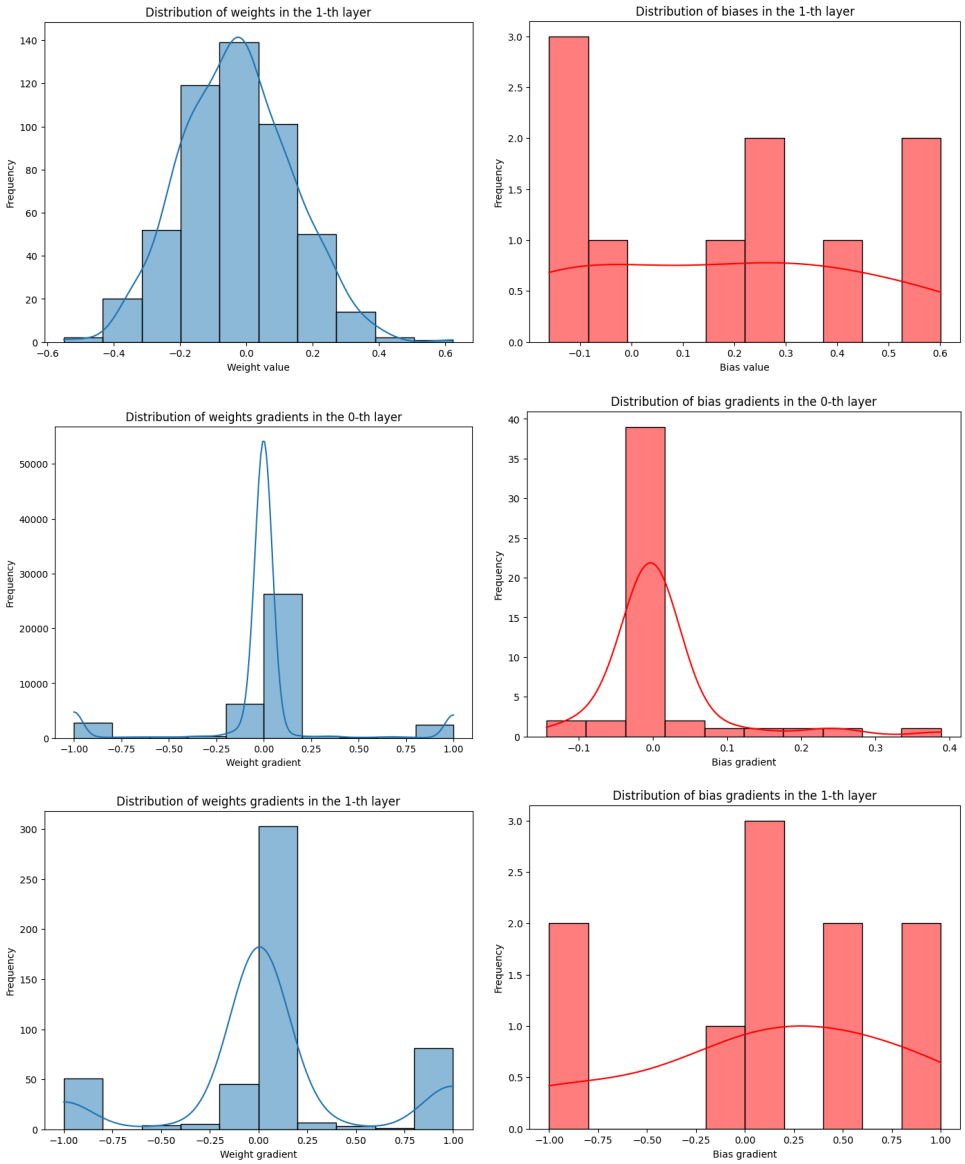
```
Accuracy: 90.35%
```










Distribution of weights in the 1-th layer
Frequency
Weight value
Distribution of biases in the 1-th layer
Frequency
Bias value
Distribution of weights gradients in the 0-th layer
Frequency
Weight gradient
Distribution of bias gradients in the 0-th layer
Frequency
Bias gradient
Distribution of weights gradients in the 1-th layer
Frequency
Weight gradient
Distribution of bias gradients in the 1-th layer
Frequency
Bias gradient

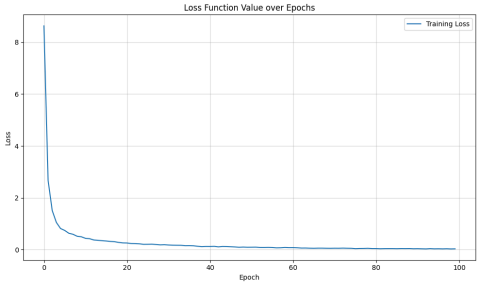
0.001

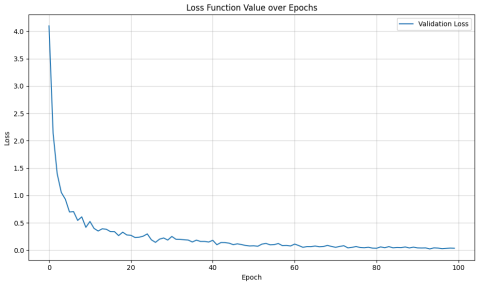
```
Epoch 96 - Training Loss: 0.037666823851267205, Validation Loss: 0.03878444518592223
Training...:  97%|▉| 97/100 [00:26<00:00,  4.57it/s]
Epoch 97 - Training Loss: 0.033828487938683614, Validation Loss: 0.027659404562870537
Training...:  98%|▉| 98/100 [00:26<00:00,  4.51it/s]
Epoch 98 - Training Loss: 0.037911819771662754, Validation Loss: 0.03329518333526633
Training...: 100%|▉| 100/100 [00:27<00:00,  4.50it/s]
Epoch 99 - Training Loss: 0.031359990315428486, Validation Loss: 0.03844855018050317
Epoch 100 - Training Loss: 0.034609571753523756, Validation Loss: 0.03577343239917139
Training...: 100%|▉| 100/100 [00:27<00:00,  3.69it/s]
```

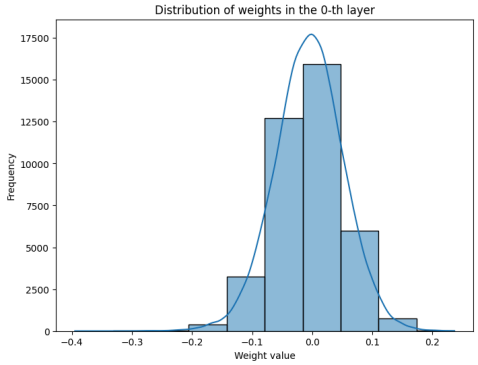
```
y_pred = model.predict(X_test)
y_true = y_test

accuracy = np.mean(np.argmax(y_pred, axis=1) == np.argmax(y_true, axis=1))

print(f"Accuracy: {accuracy * 100:.2f}%")
```

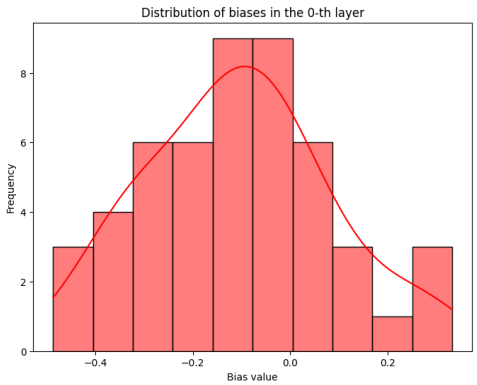
[174] ✓ 0.1s

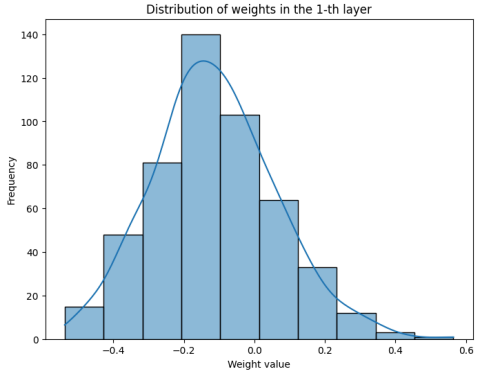
```
Accuracy: 93.41%
```

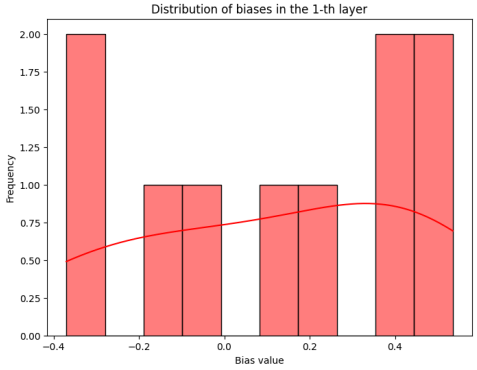


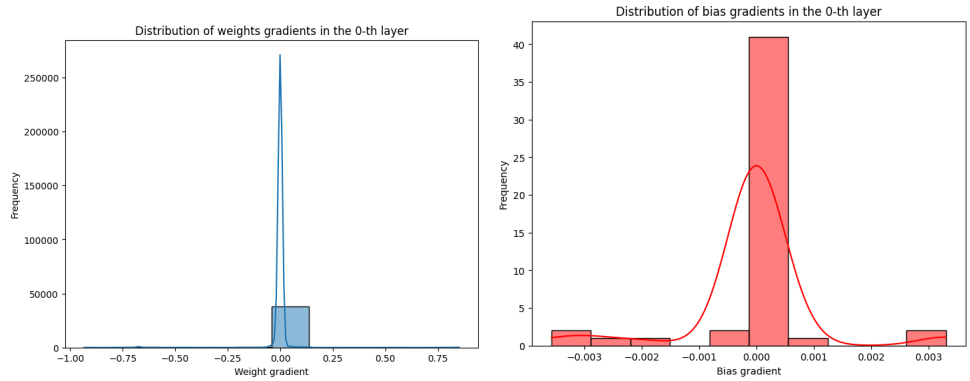


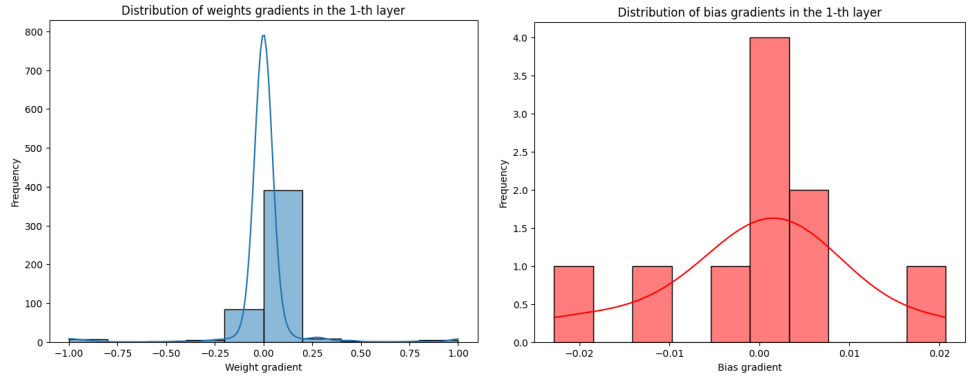










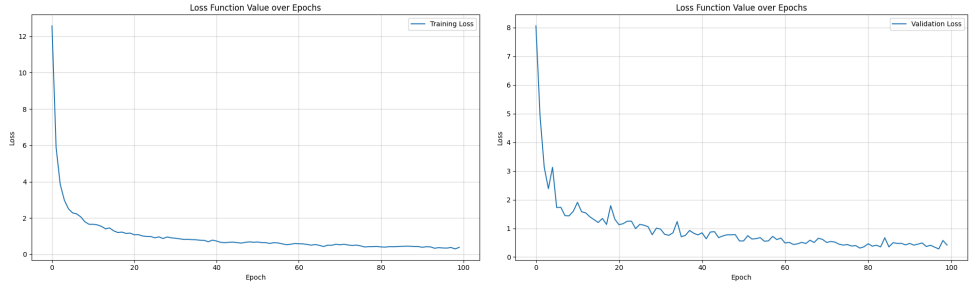


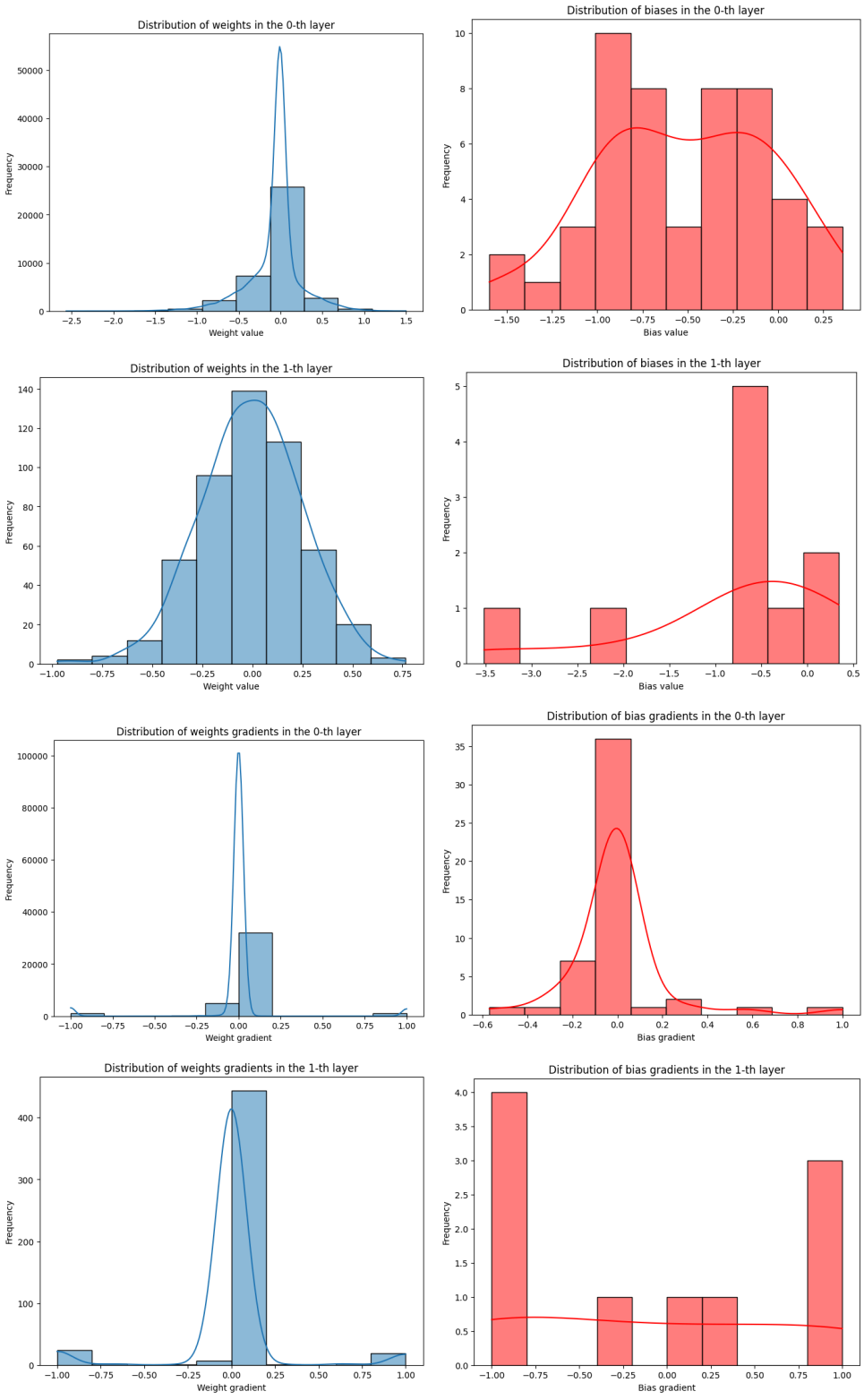
0.01

```
Epoch 96 - Training Loss: 0.34834145106659337, Validation Loss: 0.4114446304600896
Training...:  97%|▉| 97/100 [00:38<00:00,  3.93it/s]
Epoch 97 - Training Loss: 0.3429216705227836, Validation Loss: 0.3443308671334091
Training...:  98%|▉| 98/100 [00:38<00:00,  3.83it/s]
Epoch 98 - Training Loss: 0.3740485513210244, Validation Loss: 0.2827820851133835
Training...:  99%|▉| 99/100 [00:38<00:00,  3.81it/s]
Epoch 99 - Training Loss: 0.2959980752182042, Validation Loss: 0.5782272613257007
Training...: 100%|▉| 100/100 [00:38<00:00,  2.57it/s]
Epoch 100 - Training Loss: 0.3851885753704157, Validation Loss: 0.4186893593418138
```

```
y_pred = model.predict(X_test)
y_true = y_test

accuracy = np.mean(np.argmax(y_pred, axis=1) == np.argmax(y_true, axis=1))

print(f"Accuracy: {accuracy * 100:.2f}%")
```

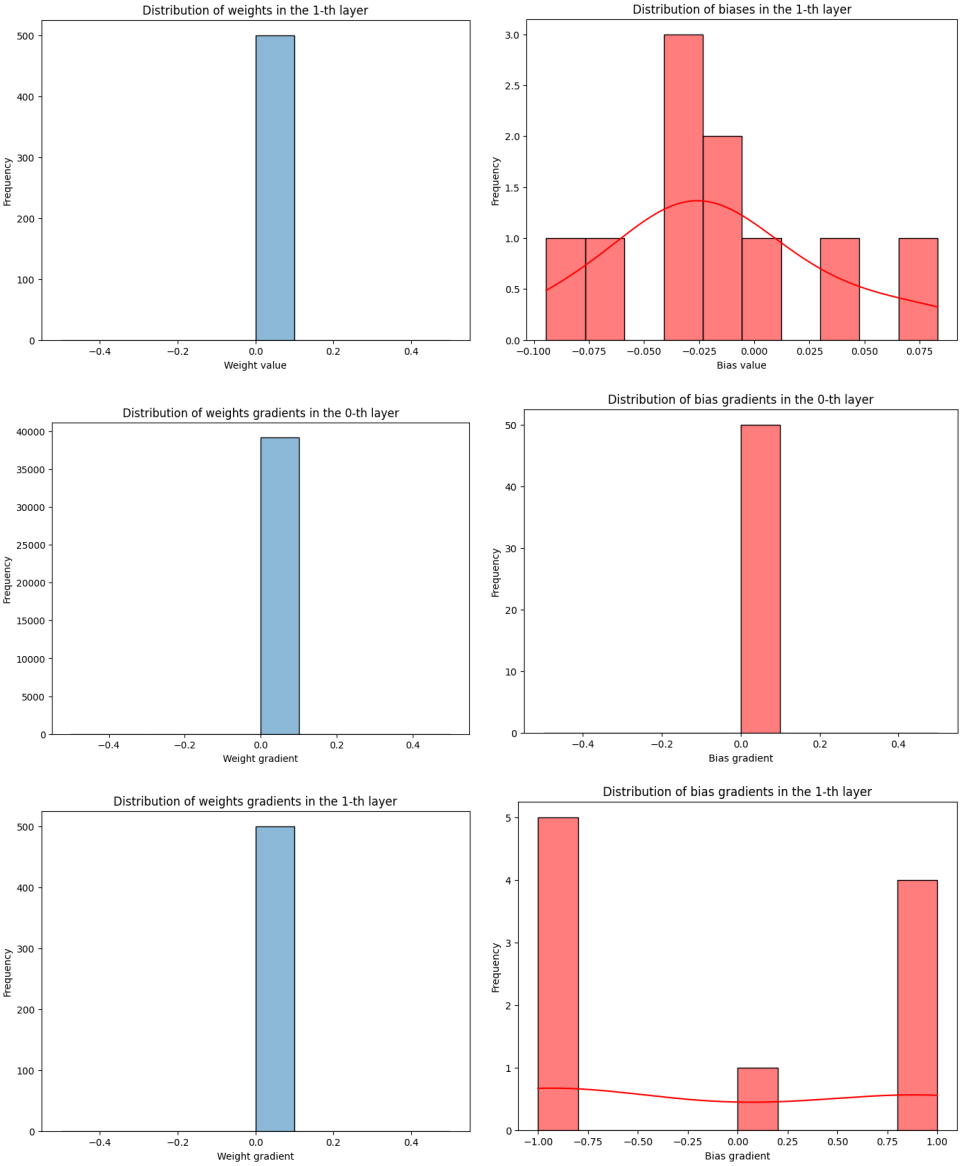
```
[192]  ✓ 0.1s
...  Accuracy: 92.21%
```




Distribution of weights in the 0-th layer
Frequency
Weight value
Distribution of biases in the 0-th layer
Frequency
Bias value
Distribution of weights in the 1-th layer
Frequency
Weight value
Distribution of biases in the 1-th layer
Frequency
Bias value
Distribution of weights gradients in the 0-th layer
Frequency
Weight gradient
Distribution of bias gradients in the 0-th layer
Frequency
Bias gradient
Distribution of weights gradients in the 1-th layer
Frequency
Weight gradient
Distribution of bias gradients in the 1-th layer
Frequency
Bias gradient

2.4. Pengaruh inisialisasi bobot

Inisialisasi Bobot	FFNN Implementasi
Zero	


Distribution of weights in the 1-th layer
Frequency
Weight value
Distribution of biases in the 1-th layer
Frequency
Bias value
Distribution of weights gradients in the 0-th layer
Frequency
Weight gradient
Distribution of bias gradients in the 0-th layer
Frequency
Bias gradient
Distribution of weights gradients in the 1-th layer
Frequency
Weight gradient
Distribution of bias gradients in the 1-th layer
Frequency
Bias gradient

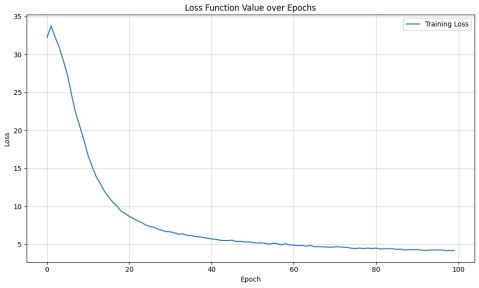
Uniform

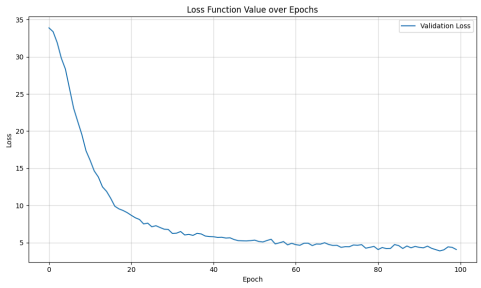
```
Epoch 96 - Training Loss: 4.250347620464657, Validation Loss: 3.877100358884709
Training...:  97%|▉| 97/100 [00:27<00:00,  3.38it/s]
Epoch 97 - Training Loss: 4.230998584844614, Validation Loss: 4.0278632297164165
Training...:  98%|▉| 98/100 [00:27<00:00,  3.38it/s]
Epoch 98 - Training Loss: 4.143600181455897, Validation Loss: 4.4237062502271565
Training...:  99%|▉| 99/100 [00:27<00:00,  3.34it/s]
Epoch 99 - Training Loss: 4.153030486973047, Validation Loss: 4.356146782375881
Training...: 100%|▉| 100/100 [00:27<00:00,  3.58it/s]
Epoch 100 - Training Loss: 4.162840177780078, Validation Loss: 4.06789831722803
```

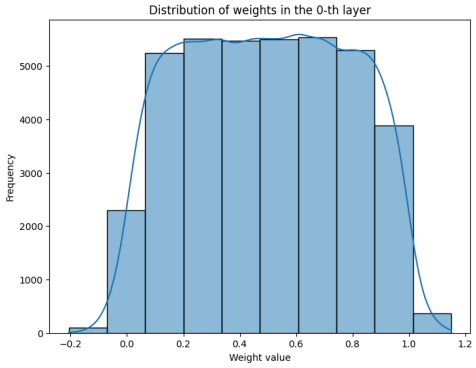
```
y_pred = model.predict(X_test)
y_true = y_test

accuracy = np.mean(np.argmax(y_pred, axis=1) == np.argmax(y_true, axis=1))

print(f"Accuracy: {accuracy * 100:.2f}%")
```

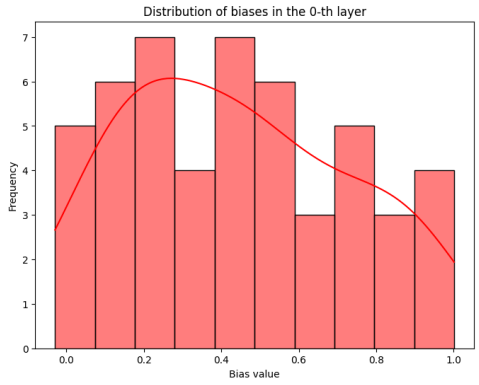
[228] ✓ 0.1s

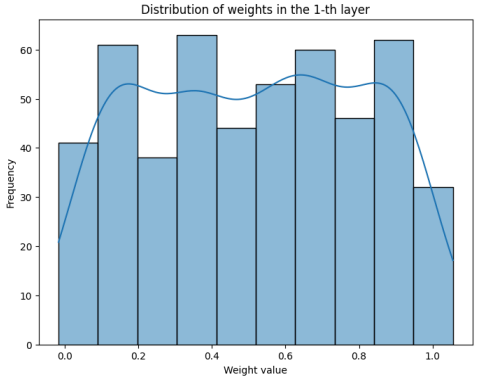
```
Accuracy: 86.05%
```

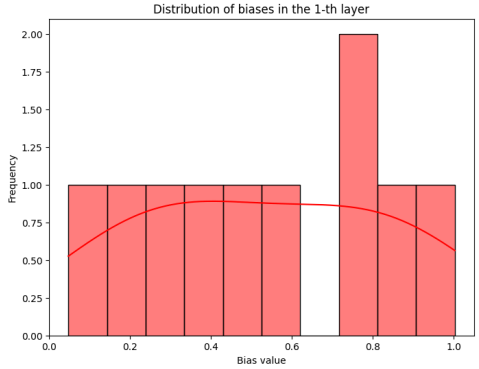


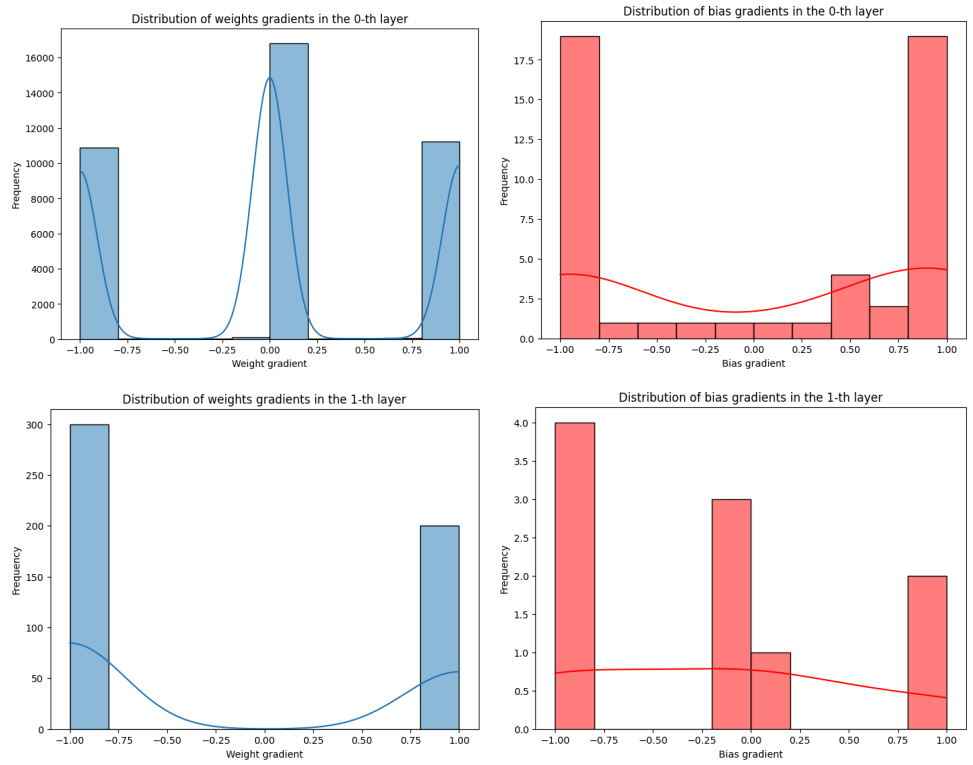












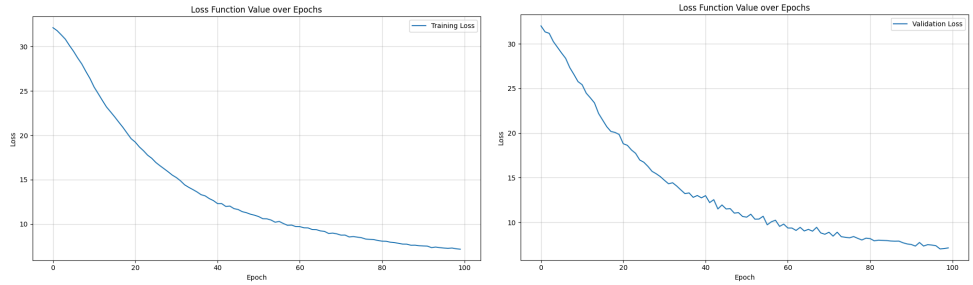
Normal

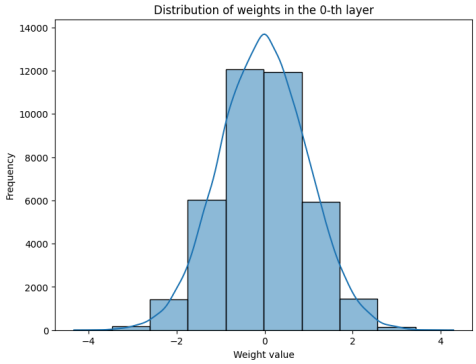
```
Epoch 96 - Training Loss: 7.295784052555163, Validation Loss: 7.459996819901453
Training...:  97%|          | 97/100 [00:21<00:00,  4.45it/s]
Epoch 97 - Training Loss: 7.2644106468770095, Validation Loss: 7.38160400067541
Training...:  98%|          | 98/100 [00:21<00:00,  4.46it/s]
Epoch 98 - Training Loss: 7.297002427343978, Validation Loss: 7.024275294845938
Training...:  99%|          | 99/100 [00:21<00:00,  4.48it/s]
Epoch 99 - Training Loss: 7.214239262313788, Validation Loss: 7.0644605334272494
Training...: 100%|          | 100/100 [00:21<00:00,  4.55it/s]
Epoch 100 - Training Loss: 7.166176222613777, Validation Loss: 7.140244603930162
```

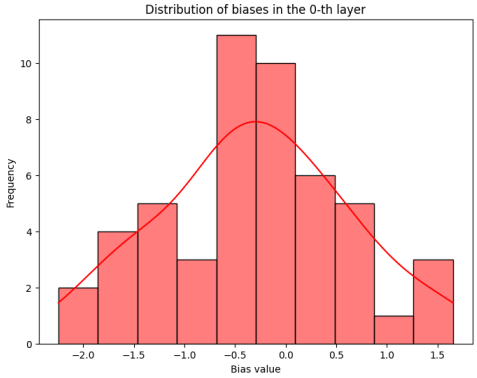
```
y_pred = model.predict(X_test)
y_true = y_test

accuracy = np.mean(np.argmax(y_pred, axis=1) == np.argmax(y_true, axis=1))

print(f"Accuracy: {accuracy * 100:.2f}%")
```

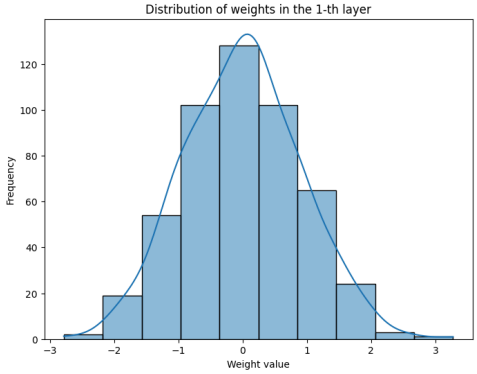
[246] ✓ 0.1s

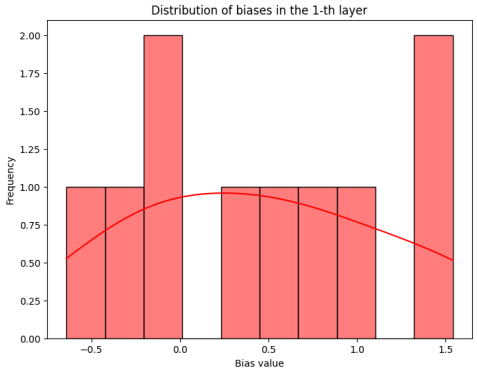
```
Accuracy: 77.90%
```

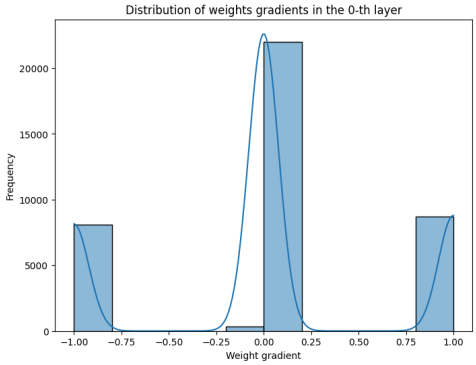


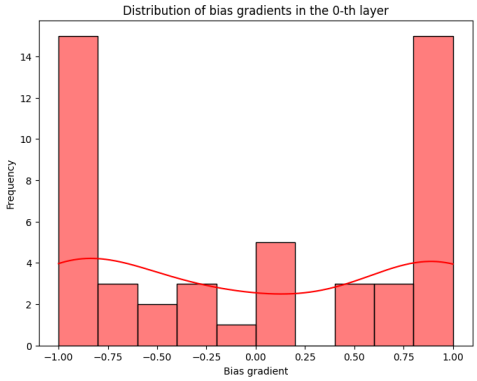

Distribution of weights in the 0-th layer
Frequency
Weight value

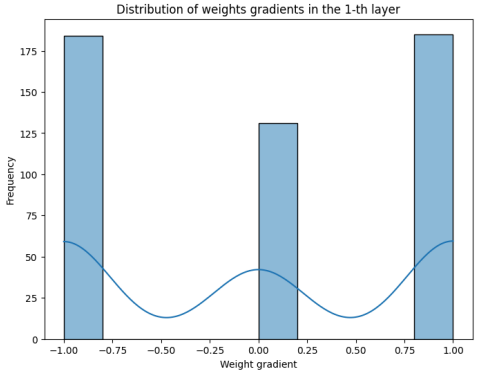

Distribution of biases in the 0-th layer
Frequency
Bias value

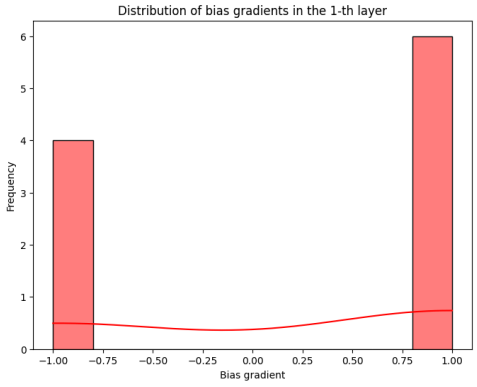

Distribution of weights in the 1-th layer
Frequency
Weight value


Distribution of biases in the 1-th layer
Frequency
Bias value


Distribution of weights gradients in the 0-th layer
Frequency
Weight gradient


Distribution of bias gradients in the 0-th layer
Frequency
Bias gradient


Distribution of weights gradients in the 1-th layer
Frequency
Weight gradient


Distribution of bias gradients in the 1-th layer
Frequency
Bias gradient

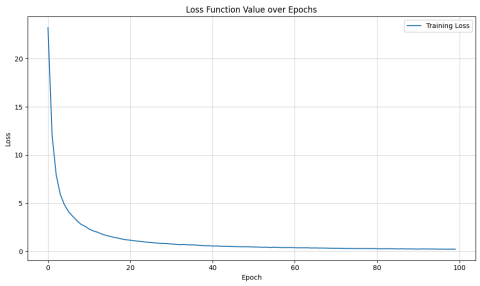
Xavier

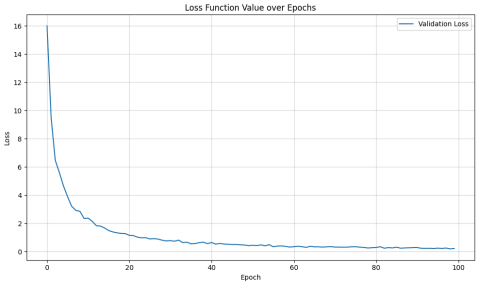
```
Epoch 96 - Training Loss: 0.22786766126742902, Validation Loss: 0.24071917159628156
Training...:  97%|▉| 97/100 [00:24<00:00,  4.18it/s]
Epoch 97 - Training Loss: 0.2309946782558585, Validation Loss: 0.2197282314381602
Training...:  98%|▉| 98/100 [00:24<00:00,  4.08it/s]
Epoch 98 - Training Loss: 0.22141769660888183, Validation Loss: 0.25030385785179715
Training...:  99%|▉| 99/100 [00:25<00:00,  4.16it/s]
Epoch 99 - Training Loss: 0.2290010259272571, Validation Loss: 0.18695369871854464
Training...: 100%|▉| 100/100 [00:25<00:00,  3.96it/s]
Epoch 100 - Training Loss: 0.22273040494650606, Validation Loss: 0.22176651843255993
```

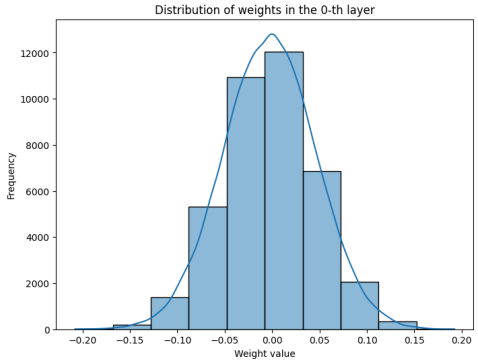
```
y_pred = model.predict(X_test)
y_true = y_test

accuracy = np.mean(np.argmax(y_pred, axis=1) == np.argmax(y_true, axis=1))

print(f"Accuracy: {accuracy * 100:.2f}%")
```

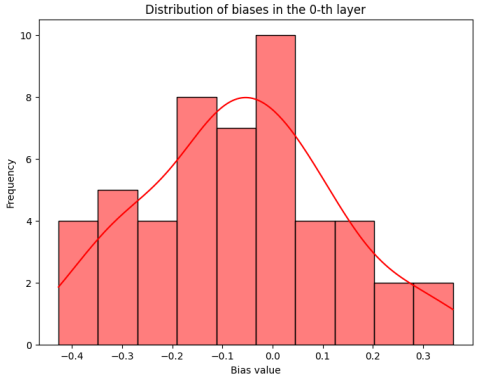
[30] ✓ 0.1s

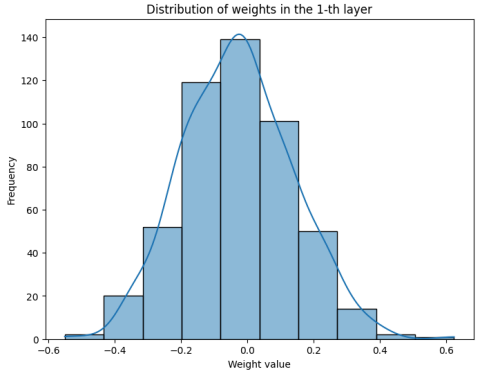
```
Accuracy: 90.35%
```

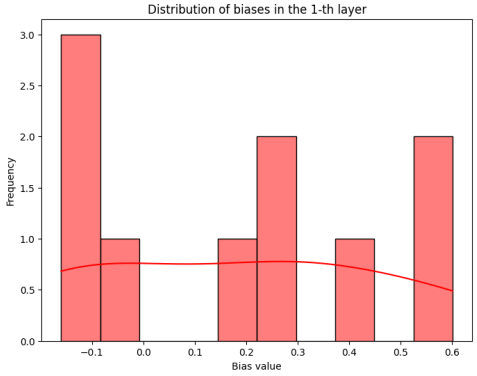


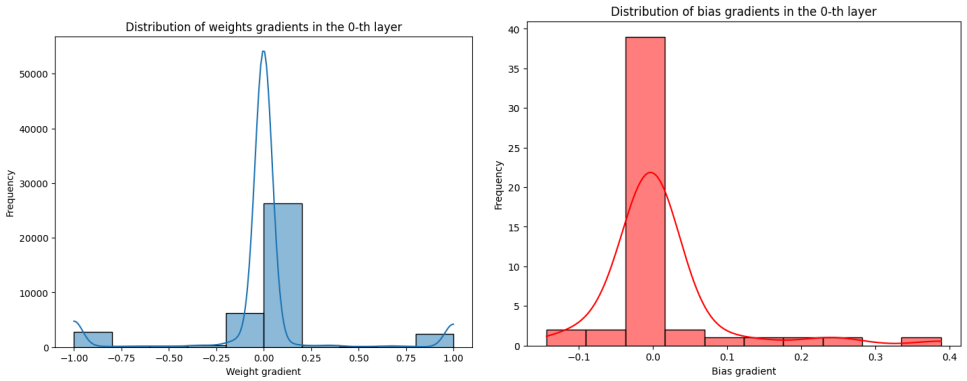


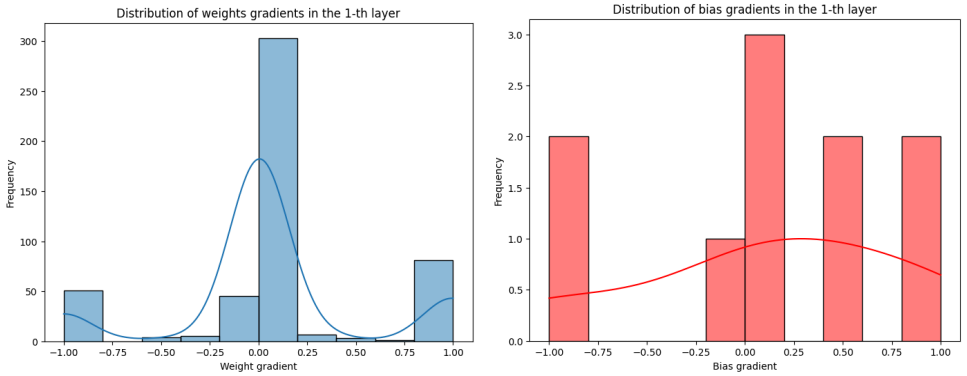












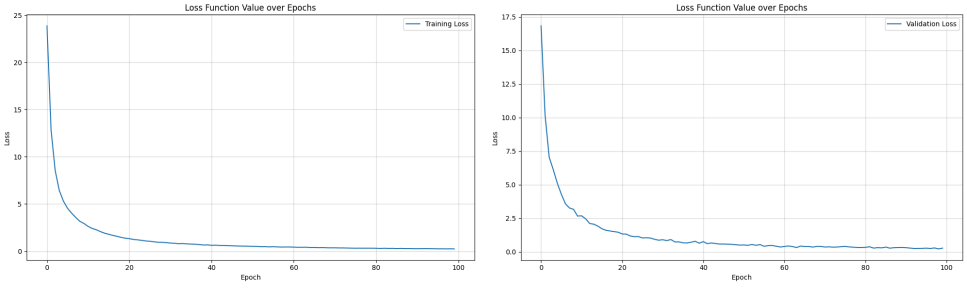
He

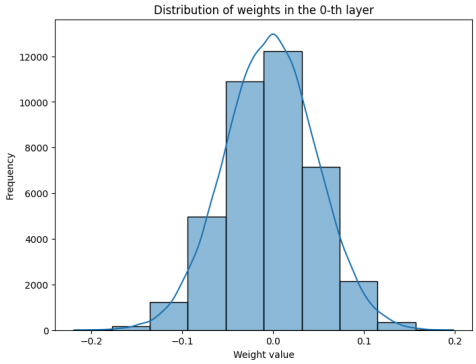
```
Epoch 96 - Training Loss: 0.2523206574092719, Validation Loss: 0.26487632501298936
Training...:  97%|▉| 97/100 [00:20<00:00,  4.78it/s]
Epoch 97 - Training Loss: 0.2576595902623791, Validation Loss: 0.23771634174915554
Training...:  98%|▉| 98/100 [00:20<00:00,  4.77it/s]
Epoch 98 - Training Loss: 0.24546618169058407, Validation Loss: 0.28261623681440773
Training...:  99%|▉| 99/100 [00:20<00:00,  4.72it/s]
Epoch 99 - Training Loss: 0.2589525275081902, Validation Loss: 0.20768056461493267
Epoch 100 - Training Loss: 0.24333605523552457, Validation Loss: 0.26713429349004025
Training...: 100%|▉| 100/100 [00:21<00:00,  4.73it/s]
```

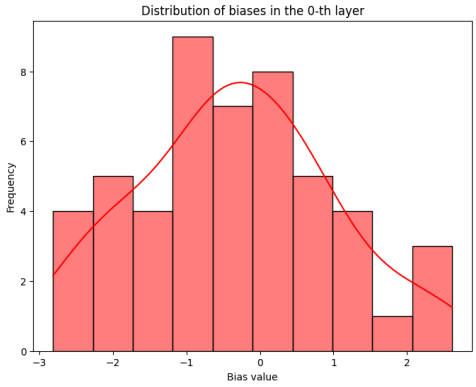
```
y_pred = model.predict(X_test)
y_true = y_test

accuracy = np.mean(np.argmax(y_pred, axis=1) == np.argmax(y_true, axis=1))

print(f"Accuracy: {accuracy * 100:.2f}%")
```

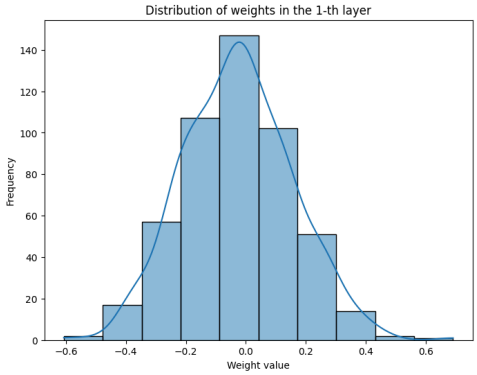
[264] ✓ 0.1s

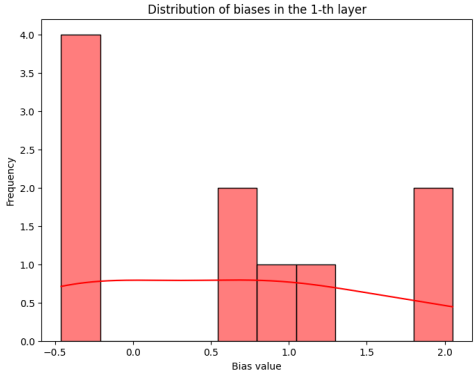
```
Accuracy: 90.11%
```

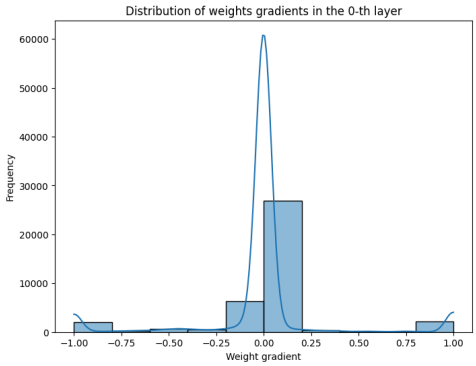


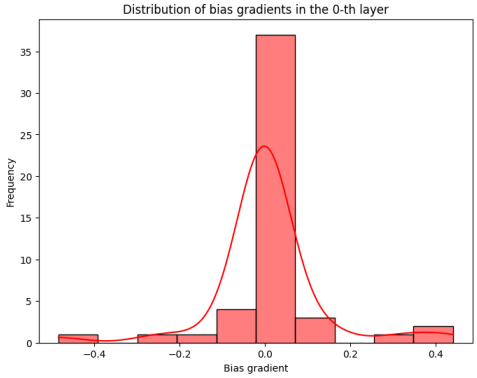

Distribution of weights in the 0-th layer
Frequency
Weight value

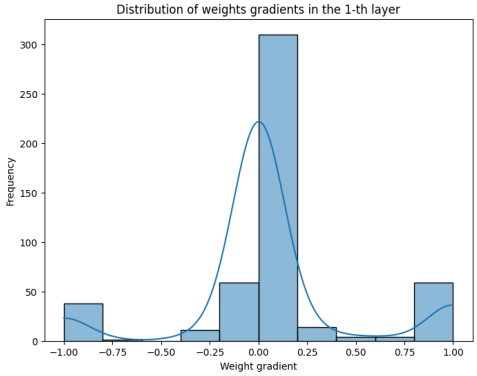

Distribution of biases in the 0-th layer
Frequency
Bias value

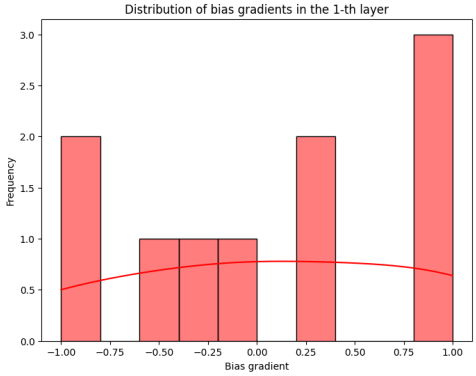

Distribution of weights in the 1-th layer
Frequency
Weight value


Distribution of biases in the 1-th layer
Frequency
Bias value


Distribution of weights gradients in the 0-th layer
Frequency
Weight gradient


Distribution of bias gradients in the 0-th layer
Frequency
Bias gradient


Distribution of weights gradients in the 1-th layer
Frequency
Weight gradient


Distribution of bias gradients in the 1-th layer
Frequency
Bias gradient

2.5. Pengaruh regularisasi

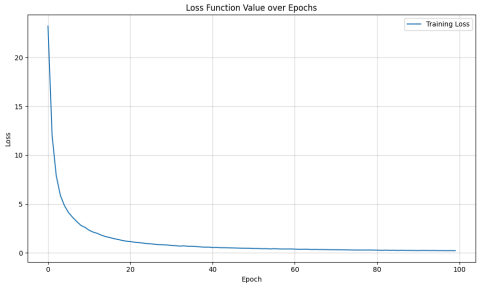
Parameter	FFNN Implementasi
Tanpa Regularisasi	

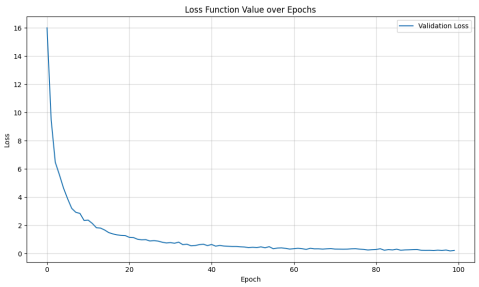
```
Epoch 96 - Training Loss: 0.22786766126742902, Validation Loss: 0.24071917159628156
Training...:  97%|▉| 97/100 [00:24<00:00,  4.18it/s]
Epoch 97 - Training Loss: 0.2309946782558585, Validation Loss: 0.2197282314381602
Training...:  98%|▉| 98/100 [00:24<00:00,  4.08it/s]
Epoch 98 - Training Loss: 0.22141769660888183, Validation Loss: 0.25030385785179715
Training...:  99%|▉| 99/100 [00:25<00:00,  4.16it/s]
Epoch 99 - Training Loss: 0.2290010259272571, Validation Loss: 0.18695369871854464
Training...: 100%|▉| 100/100 [00:25<00:00,  3.96it/s]
Epoch 100 - Training Loss: 0.22273040494650606, Validation Loss: 0.22176651843255993
```

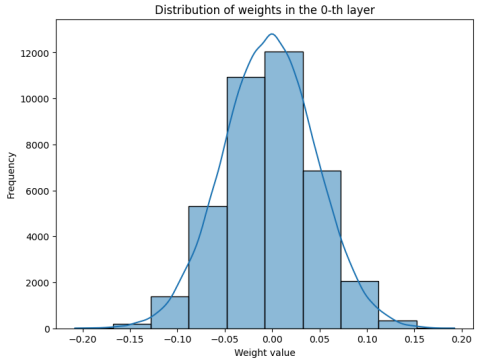
```
y_pred = model.predict(X_test)
y_true = y_test

accuracy = np.mean(np.argmax(y_pred, axis=1) == np.argmax(y_true, axis=1))

print(f"Accuracy: {accuracy * 100:.2f}%")
```

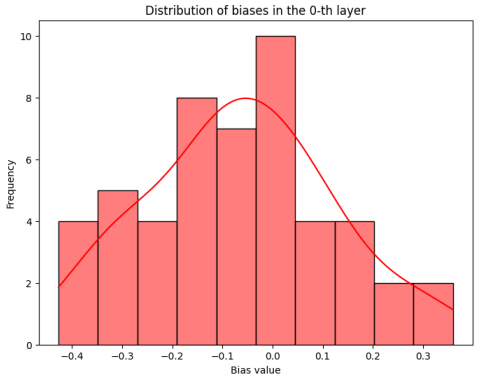
[30] ✓ 0.1s

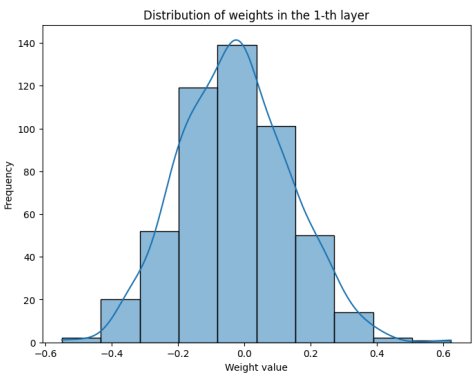
```
Accuracy: 90.35%
```

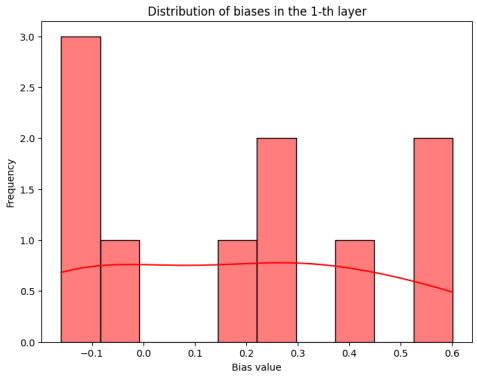


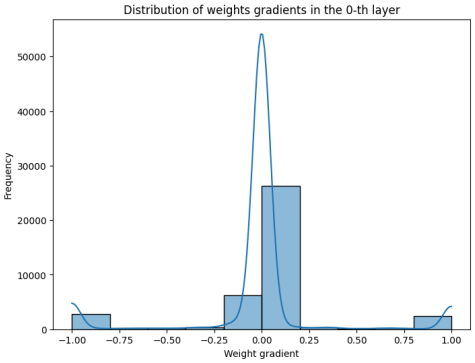


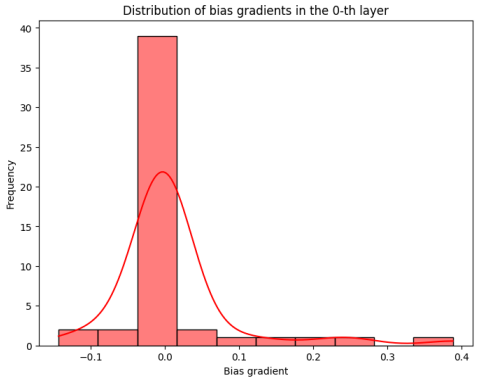


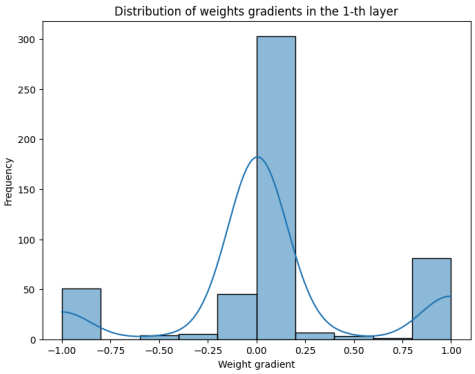


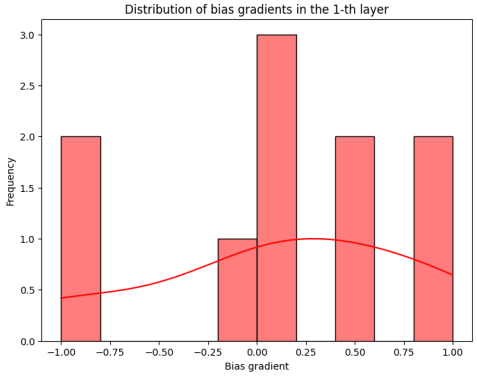

Distribution of weights in the 1-th layer
Frequency
Weight value


Distribution of biases in the 1-th layer
Frequency
Bias value


Distribution of weights gradients in the 0-th layer
Frequency
Weight gradient


Distribution of bias gradients in the 0-th layer
Frequency
Bias gradient


Distribution of weights gradients in the 1-th layer
Frequency
Weight gradient


Distribution of bias gradients in the 1-th layer
Frequency
Bias gradient

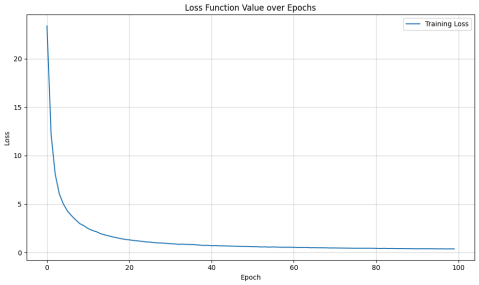
L1 = 0.0001

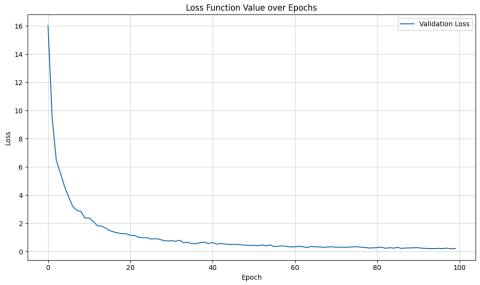
```
Epoch 96 - Training Loss: 0.3807594588760687, Validation Loss: 0.2272605101743105
Training...:  97%|▉| 97/100 [00:23<00:00,  4.20it/s]
Epoch 97 - Training Loss: 0.3834949039065537, Validation Loss: 0.21420913958232676
Training...:  98%|▉| 98/100 [00:23<00:00,  4.21it/s]
Epoch 98 - Training Loss: 0.37344038351280767, Validation Loss: 0.2456640493140506
Training...:  99%|▉| 99/100 [00:23<00:00,  4.28it/s]
Epoch 99 - Training Loss: 0.38057662100689993, Validation Loss: 0.1888151044104507
Training...: 100%|▉| 100/100 [00:23<00:00,  4.20it/s]
Epoch 100 - Training Loss: 0.3729218225921984, Validation Loss: 0.22149774180930315
```

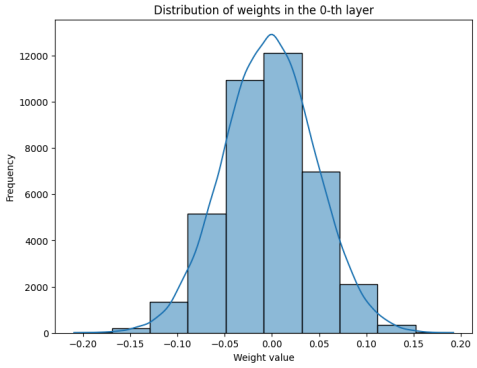
```
y_pred = model.predict(X_test)
y_true = y_test

accuracy = np.mean(np.argmax(y_pred, axis=1) == np.argmax(y_true, axis=1))

print(f"Accuracy: {accuracy * 100:.2f}%")
```

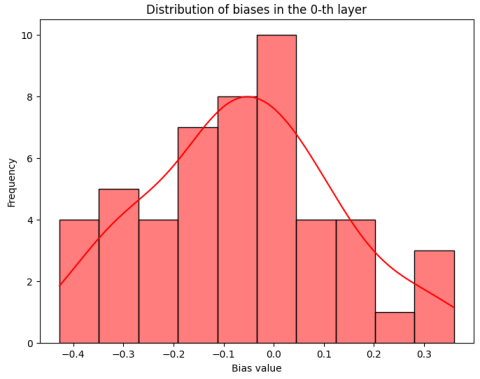
[320] ✓ 0.1s

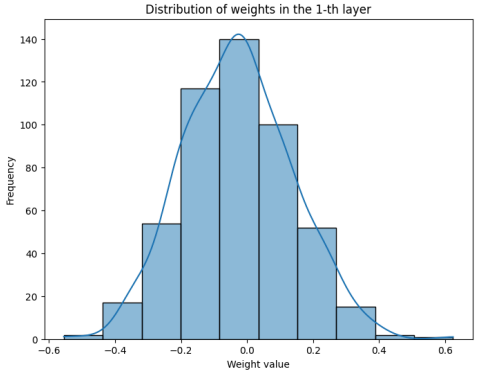
```
Accuracy: 90.53%
```

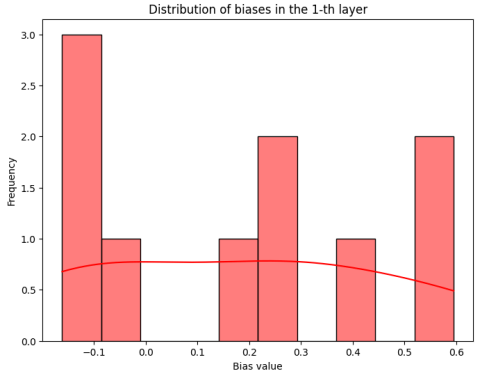


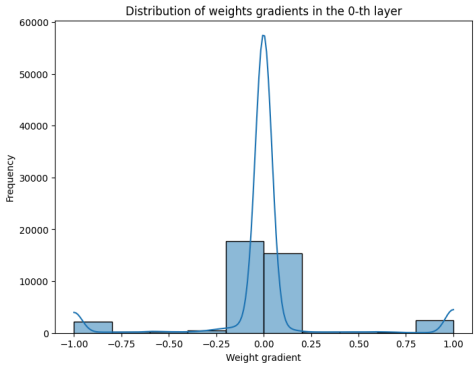


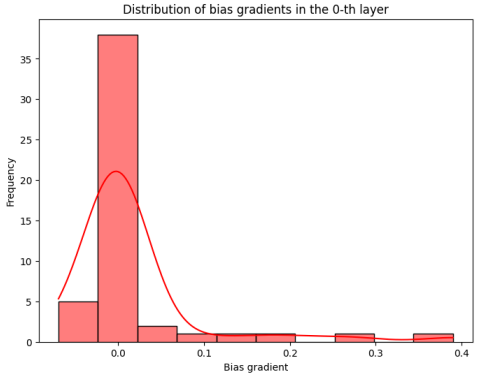


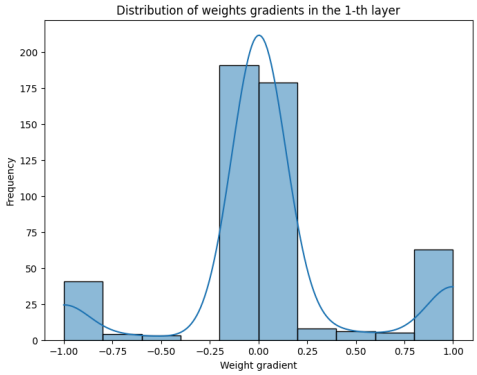


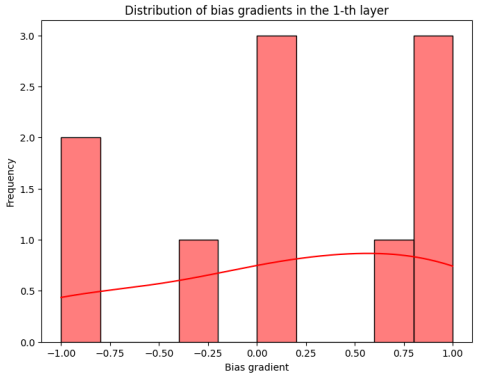












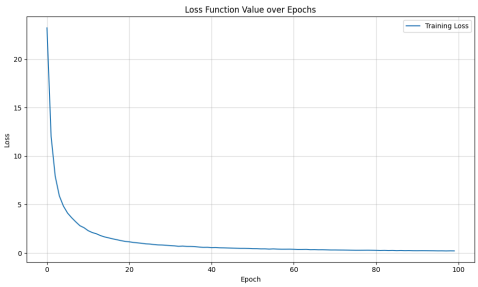
L2 = 0.0001

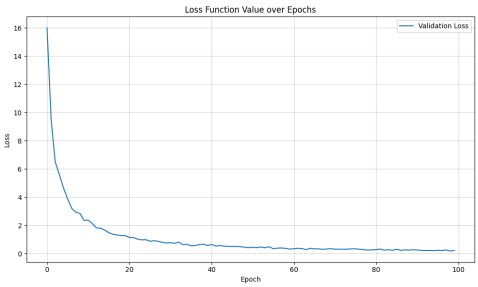
```
Epoch 96 - Training Loss: 0.22326438783775623, Validation Loss: 0.23204972045691571
Training...:  97%|          | 97/100 [00:21<00:00,  4.39it/s]
Epoch 97 - Training Loss: 0.22648566532352526, Validation Loss: 0.21747441445278853
Training...:  98%|          | 98/100 [00:21<00:00,  4.37it/s]
Epoch 98 - Training Loss: 0.21598822553843533, Validation Loss: 0.2504403633146141
Training...:  99%|          | 99/100 [00:22<00:00,  4.40it/s]
Epoch 99 - Training Loss: 0.22546142370327585, Validation Loss: 0.18858308374867075
Training...: 100%|          | 100/100 [00:22<00:00,  4.49it/s]
Epoch 100 - Training Loss: 0.2204032912980592, Validation Loss: 0.21358190551013798
```

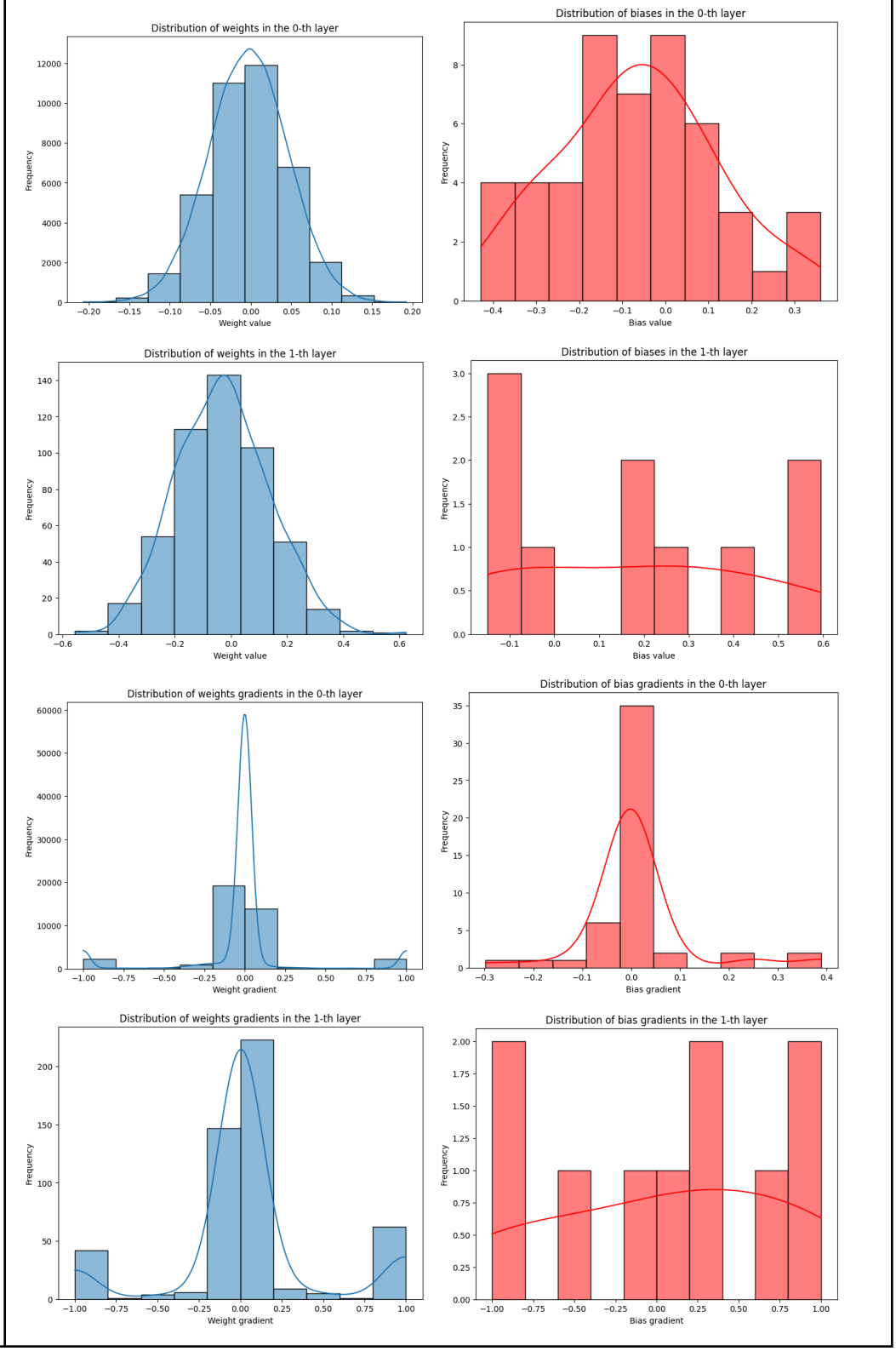
```
y_pred = model.predict(X_test)
y_true = y_test

accuracy = np.mean(np.argmax(y_pred, axis=1) == np.argmax(y_true, axis=1))

print(f"Accuracy: {accuracy * 100:.2f}%")
```

[340] ✓ 0.1s

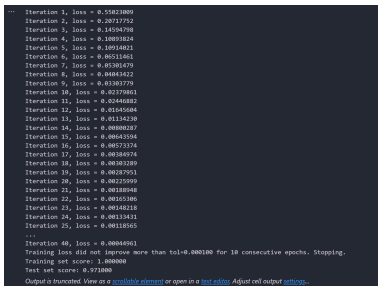
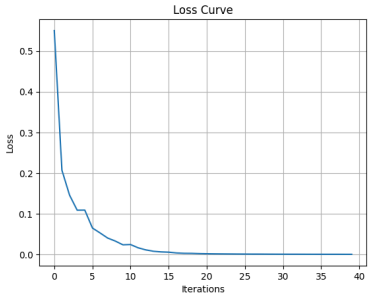
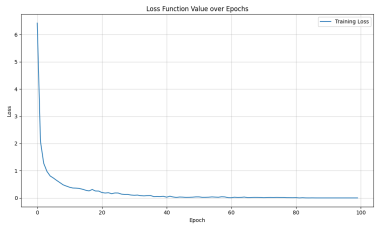
```
Accuracy: 90.50%
```







2.6. Perbandingan dengan library sklearn

Hyperparameter	Scikit Learn	FFNN Implementasi
Epoch = 100 Depth = 4 Width = [784, 256, 128, 10] Fungsi Aktivasi = [ReLU, ReLU, Softmax] Loss Function = Categorical Cross Entropy Learning Rate = 0.001 Batch Size = 256 Inisialisasi Bobot = [Xavier, Xavier, Xavier]	 	 

BAB III

KESIMPULAN DAN SARAN

Kesimpulan

Berdasarkan hasil eksperimen, fungsi aktivasi ReLU (Rectified Linear Unit) terbukti lebih efektif untuk digunakan pada hidden layer. ReLU tidak hanya mempercepat proses pelatihan, tetapi juga membantu mengurangi masalah vanishing gradient yang sering terjadi pada fungsi aktivasi lainnya seperti Sigmoid atau Tanh.

Untuk tugas klasifikasi multi-class, seperti yang dilakukan pada dataset MNIST, penggunaan fungsi aktivasi Softmax pada output layer sangat disarankan. Softmax memberikan probabilitas yang lebih baik untuk setiap kelas, sehingga memudahkan dalam pengambilan keputusan akhir.

Fluktuasi pada fungsi loss selama proses pelatihan dapat diminimalkan dengan melakukan penyesuaian pada hyperparameter, seperti learning rate dan batch size. Melakukan eksperimen untuk menemukan kombinasi parameter yang optimal sangat penting untuk mencapai konvergensi yang stabil.

Tugas besar ini juga menunjukkan bahwa kedalaman (depth) dan lebar (width) dari jaringan berpengaruh signifikan terhadap performa model. Penambahan layer dan neuron dapat meningkatkan kemampuan model dalam menangkap pola, namun juga berisiko menyebabkan overfitting jika tidak diimbangi dengan teknik regularisasi yang tepat.

Selain itu, hyperparameter regularisasi L1 dan L2 juga perlu dituning lebih lanjut, khususnya nilai lambda-nya, untuk menemukan keseimbangan yang optimal antara kompleksitas model dan kemampuan generalisasi.

Hasil evaluasi menunjukkan bahwa model FFNN yang diimplementasikan mampu memberikan akurasi yang memuaskan pada dataset yang digunakan. Namun, penting untuk terus melakukan evaluasi dan perbaikan model agar dapat beradaptasi dengan data baru dan kompleksitas yang meningkat.

Saran

- Melakukan optimasi hyperparameter secara sistematis menggunakan teknik seperti Grid Search atau Random Search untuk menemukan kombinasi parameter yang paling efektif.
- Disarankan untuk melakukan tuning lebih lanjut pada hyperparameter regularisasi L1 dan L2, khususnya nilai lambda, untuk meningkatkan performa model dan mencegah overfitting secara lebih efektif.

BAB IV

REFERENSI

Osajima, J. (n.d.). *Forwardprop*. Diambil dari https://www.jasonosajima.com/forwardprop

Osajima, J. (n.d.). *Backprop*. Diambil dari https://www.jasonosajima.com/backprop

NumPy. (n.d.). *NumPy v2.2 Manual*. Diambil dari https://numpy.org/doc/2.2/

Scikit-learn. (n.d.). *sklearn.neural_network.MLPClassifier*. Diambil dari https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

OpenStax. (n.d.). *The Chain Rule for Multivariable Functions*. Diambil dari https://math.libretexts.org/Bookshelves/Calculus/Calculus_(OpenStax)/14%3A_Differentiation_of_Functions_of_Several_Variables/14.05%3A_The_Chain_Rule_for_Multivariable_Functions

Bendersky, E. (2016). *The Softmax Function and Its Derivative*. Diambil dari https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/

Orr, D. (2021). *Autodiff from Scratch*. Diambil dari https://douglasorr.github.io/2021-11-autodiff/article.html

Grosse, R. (2022). *CSC2541: Neural Net Training Dynamics - Tutorial 1 - Backpropagation & Automatic Differentiation*. Diambil dari https://www.cs.toronto.edu/~rgrosse/courses/csc2541_2022/tutorials/tut01.pdf

Khelli07. (n.d.). *Feedforward Neural Network (FNN) Implementation from Scratch using Python*. Diambil dari https://khelli07.medium.com/feedforward-neural-network-fnn-implementation-from-scratch-using-python-467f51ecca3d

Turing.com. (n.d.). *How to Choose an Activation Function for Deep Learning*. Diambil dari https://www.turing.com/kb/how-to-choose-an-activation-function-for-deep-learning

LAMPIRAN

NIM	Pembagian Tugas
13522005	Inisialisasi struktur proyek, implementasi fungsi aktivasi dan fungsi loss beserta turunannya, membantu implementasi forward dan backward, implementasi L1 dan L2 regularization
13522015	Implementasi visualisasi neural network dengan graph, membantu implementasi forward pada FFNN class, implementasi grafik penurunan loss terhadap epoch
13522107	Implementasi visualisasi distribusi weights dan loss, membantu implementasi backward, penjelasan implementasi dalam laporan