

Tugas Besar 1 IF3270 Pembelajaran Mesin

Implementasi Feedforward Neural Network



Disusun oleh:

Farhan Raditya Aji 13522142

M. Zaidan Sa'dun Robbani 13522146

Rafif Ardhinto Ichwantoro 13522159

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024/2025**

DAFTAR ISI

DAFTAR ISI.....	1
BAB 1	
DESKRIPSI persoalan.....	1
BAB 2	
PEMBAHASAN.....	3
2.1 Penjelasan Implementasi.....	3
2.2.1 Deskripsi kelas beserta deskripsi atribut dan methodnya.....	3
2.2.2 Penjelasan Forward Propagation.....	6
2.2.3 Penjelasan Backward Propagation dan Weight Update.....	8
2.2 Hasil Pengujian.....	10
2.2.1 Pengaruh depth (banyak layer) dan width (banyak neuron per layer).....	10
2.2.2 Pengaruh fungsi aktivasi hidden layer.....	14
2.2.3 Pengaruh learning rate.....	19
2.2.4 Pengaruh inisialisasi bobot.....	24
2.2.5 Pengaruh regularisasi.....	29
2.2.6 Analisis perbandingan hasil prediksi dengan library sklearn MLP.....	34
BAB 3	
KESIMPULAN DAN SARAN.....	36
PEMBAGIAN TUGAS.....	39
REFERENSI.....	40

BAB 1

DESKRIPSI PERSOALAN

Feedforward Neural Network (FFNN) merupakan model jaringan saraf tiruan yang memiliki kemampuan untuk menerima konfigurasi jumlah neuron pada setiap lapisan, termasuk lapisan input dan output. Model ini mendukung berbagai jenis fungsi aktivasi, seperti Linear, ReLU, Sigmoid, Tanh, dan Softmax. Selain itu, FFNN juga mendukung penggunaan berbagai fungsi loss seperti Mean Squared Error (MSE), Binary Cross-Entropy, dan Categorical Cross-Entropy.

Untuk mendukung proses pelatihan model yang efektif, FFNN memerlukan metode inisialisasi bobot yang beragam, seperti zero initialization, distribusi uniform, dan distribusi normal. Model ini juga harus mampu menyimpan bobot dan gradien dari setiap neuron, serta menampilkan distribusi bobot dan gradien tersebut dalam bentuk graf untuk analisis lebih lanjut.

Dalam hal pemrosesan data, FFNN memiliki implementasi forward propagation yang mampu menerima input dalam bentuk batch. Selain itu, implementasi backward propagation diperlukan untuk menghitung perubahan gradien menggunakan konsep chain rule. Model ini juga menerapkan metode pembaruan bobot menggunakan algoritma gradient descent.

Fitur tambahan yang mendukung efektivitas pelatihan mencakup mekanisme konfigurasi parameter seperti ukuran batch (batch size), tingkat pembelajaran (learning rate), jumlah epoch, serta opsi verbosity untuk mengatur tingkat informasi yang ditampilkan selama proses pelatihan. Selain itu, model yang dihasilkan harus dapat disimpan dan dimuat kembali untuk keperluan penggunaan lanjutan.

Analisis terhadap pengaruh berbagai hyperparameter, seperti jumlah lapisan (depth), jumlah neuron pada setiap lapisan (width), jenis fungsi aktivasi, tingkat pembelajaran, serta metode inisialisasi bobot sangat penting untuk menilai kinerja model. Evaluasi hasil prediksi FFNN juga dilakukan dengan membandingkannya dengan model Multi-layer Perceptron (MLP) pada library Scikit-learn guna mengukur efektivitas model yang dikembangkan.

Pada tugas ini akan diimplementasikan model Feedforward Neural Network yang memenuhi spesifikasi tersebut untuk menguji dan menganalisis performa model dengan berbagai konfigurasi parameter.

BAB 2

PEMBAHASAN

2.1 Penjelasan Implementasi

2.2.1 Deskripsi kelas beserta deskripsi atribut dan methodnya

Berikut merupakan penjelasan kelas-kelas utama yang diimplementasikan pada tugas besar ini:

1. ActivationFunctions

Kelas ini berisi berbagai fungsi aktivasi yang digunakan dalam forward propagation model FFNN beserta turunannya untuk keperluan backward propagation.

Atribut:

- Tidak memiliki atribut karena semua method bersifat statis.

Method:

- linear(x) dan linear_derivative(x)
- relu(x) dan relu_derivative(x)
- sigmoid(x) dan sigmoid_derivative(x)
- tanh(x) dan tanh_derivative(x)
- softmax(x) dan softmax_derivative(x)
- Bonus: leaky_relu(x, alpha=0.01) dan elu(x, alpha=1.0) beserta turunannya.

Semua method ini berfungsi untuk mengatur aliran data pada jaringan saraf dan menghitung gradien saat backpropagation.

2. LossFunctions

Kelas ini mengimplementasi berbagai fungsi loss beserta turunannya untuk mengukur error pada model.

Atribut:

- Tidak memiliki atribut karena semua method bersifat statis.

Method:

- `mse(y, y_pred)` dan `mse_derivative(y, y_pred)`
- `binary_cross_entropy(y, y_pred)` dan `binary_cross_entropy_derivative(y, y_pred)`
- `categorical_crossentropy(y, y_pred)` dan `categorical_crossentropy_derivative(y, y_pred)`

Method ini digunakan pada saat perhitungan loss dan gradien selama pelatihan model.

3. WeightInitializers

Kelas ini memiliki method untuk inisialisasi bobot jaringan saraf dengan berbagai teknik.

Atribut:

- Tidak memiliki atribut karena semua method bersifat statis.

Method:

- `zero_initialization(shape)`
- `uniform_initialization(shape, low, high, seed)`
- `normal_initialization(shape, mean, var, seed)`
- Bonus: `xavier_initialization(shape, seed)` dan `he_initialization(shape, seed)`

Method ini digunakan saat membangun model untuk memastikan bobot terinisialisasi dengan baik agar proses training stabil.

4. Kelas Layer

Kelas ini merepresentasikan satu lapisan dalam jaringan FFNN.

Atribut:

- `n_inputs`: Jumlah neuron pada input layer.
- `n_neurons`: Jumlah neuron pada layer ini.
- `activation_name`: Nama fungsi aktivasi yang digunakan.
- `W`: Matriks bobot untuk koneksi antar neuron.
- `b`: Vektor bias.
- `dW` dan `db`: Gradien untuk bobot dan bias.

- cache: Cache untuk menyimpan data intermediate pada forward dan backward propagation.

Method:

- set_activation_function(activation): Mengatur fungsi aktivasi berdasarkan input pengguna.
- init_weights_and_biases(weight_init, params): Inisialisasi bobot dan bias berdasarkan metode tertentu.
- forward(X): Melakukan forward propagation untuk menghitung keluaran layer.
- backward(dA): Menghitung gradien layer berdasarkan nilai aktivasi sebelumnya.
- update_weights(learning_rate, l1_lambda, l2_lambda): Memperbarui bobot menggunakan algoritma gradient descent dengan opsi regularisasi.

5. FeedforwardNeuralNetwork

Kelas ini merupakan implementasi utama dari FFNN yang mengatur keseluruhan proses pembuatan, pelatihan, dan evaluasi model.

Atribut:

- input_size: Jumlah fitur pada input.
- layer_sizes: List yang menyimpan jumlah neuron pada setiap lapisan.
- activations: List yang menyimpan nama fungsi aktivasi tiap layer.
- loss_name: Nama fungsi loss yang digunakan.
- weight_init: Metode inisialisasi bobot.
- layers: List berisi objek kelas Layer.

Method:

- set_loss_function(loss): Mengatur fungsi loss yang digunakan.
- _build_network(): Membangun struktur jaringan sesuai konfigurasi yang diberikan.

- `forward(X)`: Melakukan forward propagation pada seluruh jaringan.
- `compute_loss(y, y_pred)`: Menghitung loss model berdasarkan target data.
- `backward(y, y_pred)`: Menghitung gradien dan melakukan backpropagation.
- `update_weights(learning_rate, l1_lambda, l2_lambda)`: Memperbarui bobot pada setiap layer.
- `train()`: Mengatur keseluruhan proses pelatihan model dengan berbagai parameter.
- `predict(X)`: Melakukan prediksi data menggunakan model yang dilatih.
- `evaluate(X, y)`: Menghitung loss pada dataset validasi.
- `save(filename)` dan `load(filename)`: Menyimpan dan memuat model dengan format pickle.
- `plot_model()`: Menampilkan visualisasi graf struktur model beserta bobot dan gradiennya.
- `plot_weight_distribution(layers)` dan `plot_gradient_distribution(layers)`: Visualisasi distribusi bobot dan gradien pada layer tertentu.

Kelas ini memiliki struktur yang modular dan fleksibel, memungkinkan pengguna untuk menyesuaikan berbagai parameter sesuai kebutuhan.

2.2.2 Penjelasan *Forward Propagation*

Forward propagation adalah proses perhitungan yang dilakukan untuk mendapatkan keluaran dari model berdasarkan input yang diberikan. Proses ini melibatkan perhitungan linear pada setiap layer diikuti oleh penerapan fungsi aktivasi.

Pada kode yang diimplementasikan pada tugas besar ini, forward propagation dilakukan melalui dua tahapan utama:

1. Forward Propagation pada Kelas Layer

Pada kelas ini, forward propagation dilakukan dengan langkah berikut:

- a. Penyimpanan Input pada Cache
 - o Input yang diterima akan disimpan pada atribut cache untuk digunakan pada proses backpropagation nanti.
- b. Perhitungan Linear Perhitungan linear dilakukan dengan rumus berikut:
di mana:
 - o adalah input ke layer tersebut (dapat berupa data awal atau output dari layer sebelumnya).
 - o adalah bobot yang menghubungkan neuron pada layer sebelumnya dengan neuron pada layer ini.
 - o adalah bias yang ditambahkan untuk meningkatkan fleksibilitas model.
- c. Aplikasi Fungsi Aktivasi Hasil perhitungan linear kemudian diteruskan ke fungsi aktivasi untuk menghitung output akhir. Fungsi aktivasi yang digunakan bergantung pada parameter yang diberikan saat layer diinisialisasi (misalnya: relu, sigmoid, dsb).
- d. Penyimpanan Hasil pada Cache Output hasil aktivasi juga disimpan pada atribut cache untuk digunakan pada backpropagation.
- e. Return Output Hasil akhir dari layer dikembalikan sebagai output dari metode forward().

2. Forward Propagation pada Kelas FeedforwardNeuralNetwork

Pada kelas ini, forward propagation dilakukan dengan mengalirkan data input secara berurutan melalui setiap layer yang telah dibuat.

Langkah-langkahnya yaitu:

- a. Data input diterima pada metode forward().

- b. Data input tersebut diteruskan melalui setiap layer dengan memanggil metode forward() pada masing-masing layer.
- c. Output dari layer sebelumnya menjadi input untuk layer berikutnya hingga mencapai layer terakhir (output layer).
- d. Hasil akhir pada output layer menjadi hasil akhir dari proses forward propagation.

Forward propagation memiliki peran penting dalam menghitung prediksi model selama training, evaluasi, dan inferensi pada data baru.

2.2.3 Penjelasan *Backward Propagation* dan *Weight Update*

Backward propagation adalah proses yang digunakan untuk menghitung gradien dari loss function terhadap bobot dan bias pada model. Proses ini dilakukan agar model dapat memperbarui bobotnya dan menghasilkan prediksi yang lebih akurat. Pada kode yang diimplementasikan pada tugas besar ini, backward propagation dilakukan melalui dua tahapan utama:

1. Backward Propagation pada Kelas Layer

Pada kelas ini, backward propagation dilakukan dengan langkah berikut:

- a. Menghitung Gradien Aktivasi

Gradien dari layer saat ini diterima dalam bentuk dA (gradien loss terhadap output layer saat ini). Untuk fungsi aktivasi softmax, nilai dA langsung digunakan sebagai gradien (dZ). Untuk fungsi aktivasi lainnya, gradien dihitung dengan mengalikan nilai dA dengan turunan fungsi aktivasi.

- b. Menghitung Gradien Bobot dan Bias

Gradien bobot dihitung dengan mengalikan input yang telah disimpan di cache dengan nilai gradien (dZ). Gradien bias dihitung

dengan menjumlahkan seluruh elemen pada dZ untuk setiap sampel dalam batch.

- c. Return Gradien ke Layer Sebelumnya

Gradien untuk layer sebelumnya dihitung dengan mengalikan dZ dengan bobot layer saat ini. Hasil ini dikembalikan agar dapat digunakan untuk backpropagation pada layer sebelumnya.

2. Backward Propagation pada Kelas FeedforwardNeuralNetwork

Pada kelas ini, backward propagation dilakukan dengan cara berikut:

- a. Gradien awal dihitung dari turunan loss terhadap output model (dA).
- b. Gradien ini kemudian diteruskan secara berurutan ke setiap layer melalui metode `backward()` pada masing-masing layer. Proses ini dimulai dari output layer hingga mencapai input layer.

3. Weight Update pada Kelas Layer

Bobot diperbarui menggunakan metode gradient descent agar model dapat melakukan prediksi dengan lebih baik. Proses ini dilakukan dengan langkah berikut:

- a. Bobot diperbarui dengan mengurangi bobot sebelumnya dengan gradien yang telah dihitung selama backward propagation.
- b. Bias diperbarui dengan cara yang sama agar model mampu menyesuaikan prediksinya dengan data secara optimal.
- c. Opsi regularisasi L1 dan L2 dapat ditambahkan untuk mengurangi risiko overfitting.

Backward propagation dan weight update memiliki peran penting dalam proses pembelajaran model, memastikan bahwa model dapat secara efektif meminimalkan loss dan meningkatkan akurasi prediksi.

2.2 Hasil Pengujian

2.2.1 Pengaruh depth (banyak layer) dan width (banyak neuron per layer)

Activation Function Comparison Results:					\	
		name	accuracy	final_train_loss	final_val_loss	\
0	Small width [32, 10]	0.876357		0.472732	0.467809	
1	Medium width [64, 10]	0.882500		0.457522	0.448745	
2	Large width [128, 10]	0.883143		0.449202	0.445311	
3	Shallow depth [64, 10]	0.882500		0.457522	0.448745	
4	Medium depth [64, 32, 10]	0.891000		0.405075	0.399948	
5	Deep network [128, 64, 32, 10]	0.904000		0.337241	0.340229	
training_time						
0	35.170004					
1	55.699115					
2	75.354537					
3	56.920228					
4	61.659630					
5	79.168636					
Activation Function Comparison Results:					\	
		name	accuracy	final_train_loss	final_val_loss	\
0	Small width [32, 10]	0.876357		0.472732	0.467809	
1	Medium width [64, 10]	0.882500		0.457522	0.448745	
2	Large width [128, 10]	0.883143		0.449202	0.445311	
3	Shallow depth [64, 10]	0.882500		0.457522	0.448745	
4	Medium depth [64, 32, 10]	0.891000		0.405075	0.399948	
...						
2	75.354537					
3	56.920228					
4	61.659630					
5	79.168636					

Berdasarkan data eksperimen, terdapat beberapa kesimpulan penting mengenai pengaruh width dan depth pada jaringan neural. Untuk variasi width (jumlah neuron per layer), peningkatan dari 32 hingga 128 neuron hanya memberikan kenaikan akurasi yang minimal (87.64% ke 88.31%). Sebaliknya, penambahan depth (jumlah layer) menunjukkan peningkatan performa yang lebih signifikan, dengan model 3 hidden layer mencapai akurasi tertinggi 90.40%, jauh di atas model 1 hidden layer (88.25%).

Hal ini menunjukkan bahwa untuk dataset yang diuji, strategi memperdalam jaringan lebih efektif daripada memperlebar jaringan. Namun, perlu diperhatikan bahwa model yang lebih kompleks juga membutuhkan waktu pelatihan yang lebih lama, dengan model terdalam memerlukan waktu 79.17 detik dibandingkan 35.17 detik untuk model tersederhana. Ini menggambarkan trade-off antara performa dan efisiensi komputasi yang perlu dipertimbangkan saat merancang arsitektur jaringan neural.

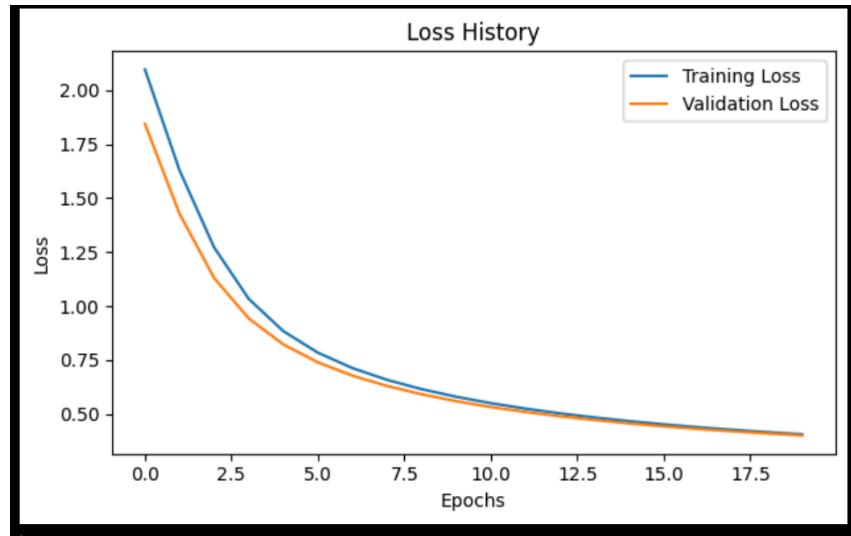
```
Training model with architecture: Small width
Layers: [32, 10]
Activations: ['relu', 'softmax']
Epoch 1/20 [5.0%] - train_loss: 2.0889 - val_loss: 1.8780
Epoch 2/20 [10.0%] - train_loss: 1.7001 - val_loss: 1.5432
Epoch 3/20 [15.0%] - train_loss: 1.4063 - val_loss: 1.2859
Epoch 4/20 [20.0%] - train_loss: 1.1836 - val_loss: 1.0928
Epoch 5/20 [25.0%] - train_loss: 1.0187 - val_loss: 0.9521
Epoch 6/20 [30.0%] - train_loss: 0.9006 - val_loss: 0.8521
Epoch 7/20 [35.0%] - train_loss: 0.8152 - val_loss: 0.7785
Epoch 8/20 [40.0%] - train_loss: 0.7508 - val_loss: 0.7222
Epoch 9/20 [45.0%] - train_loss: 0.7006 - val_loss: 0.6773
Epoch 10/20 [50.0%] - train_loss: 0.6602 - val_loss: 0.6409
Epoch 11/20 [55.0%] - train_loss: 0.6270 - val_loss: 0.6107
Epoch 12/20 [60.0%] - train_loss: 0.5991 - val_loss: 0.5852
Epoch 13/20 [65.0%] - train_loss: 0.5753 - val_loss: 0.5633
Epoch 14/20 [70.0%] - train_loss: 0.5547 - val_loss: 0.5444
Epoch 15/20 [75.0%] - train_loss: 0.5368 - val_loss: 0.5276
Epoch 16/20 [80.0%] - train_loss: 0.5210 - val_loss: 0.5128
Epoch 17/20 [85.0%] - train_loss: 0.5069 - val_loss: 0.4997
Epoch 18/20 [90.0%] - train_loss: 0.4943 - val_loss: 0.4880
Epoch 19/20 [95.0%] - train_loss: 0.4830 - val_loss: 0.4774
Epoch 20/20 [100.0%] - train_loss: 0.4727 - val_loss: 0.4678
```

```
Training model with architecture: Medium width
Layers: [64, 10]
Activations: ['relu', 'softmax']
Epoch 1/20 [5.0%] - train_loss: 2.1349 - val_loss: 1.9080
Epoch 2/20 [10.0%] - train_loss: 1.7146 - val_loss: 1.5387
Epoch 3/20 [15.0%] - train_loss: 1.3953 - val_loss: 1.2640
Epoch 4/20 [20.0%] - train_loss: 1.1634 - val_loss: 1.0690
Epoch 5/20 [25.0%] - train_loss: 0.9993 - val_loss: 0.9302
Epoch 6/20 [30.0%] - train_loss: 0.8817 - val_loss: 0.8296
Epoch 7/20 [35.0%] - train_loss: 0.7953 - val_loss: 0.7548
Epoch 8/20 [40.0%] - train_loss: 0.7300 - val_loss: 0.6976
Epoch 9/20 [45.0%] - train_loss: 0.6794 - val_loss: 0.6524
Epoch 10/20 [50.0%] - train_loss: 0.6389 - val_loss: 0.6160
Epoch 11/20 [55.0%] - train_loss: 0.6059 - val_loss: 0.5861
Epoch 12/20 [60.0%] - train_loss: 0.5785 - val_loss: 0.5610
Epoch 13/20 [65.0%] - train_loss: 0.5553 - val_loss: 0.5397
Epoch 14/20 [70.0%] - train_loss: 0.5354 - val_loss: 0.5214
Epoch 15/20 [75.0%] - train_loss: 0.5182 - val_loss: 0.5053
Epoch 16/20 [80.0%] - train_loss: 0.5031 - val_loss: 0.4912
Epoch 17/20 [85.0%] - train_loss: 0.4897 - val_loss: 0.4788
Epoch 18/20 [90.0%] - train_loss: 0.4778 - val_loss: 0.4677
Epoch 19/20 [95.0%] - train_loss: 0.4672 - val_loss: 0.4578
Epoch 20/20 [100.0%] - train_loss: 0.4575 - val_loss: 0.4487
```

```
Training model with architecture: Large width
Layers: [128, 10]
Activations: ['relu', 'softmax']
Epoch 1/20 [5.0%] - train_loss: 2.0910 - val_loss: 1.8662
Epoch 2/20 [10.0%] - train_loss: 1.6726 - val_loss: 1.4964
Epoch 3/20 [15.0%] - train_loss: 1.3512 - val_loss: 1.2228
Epoch 4/20 [20.0%] - train_loss: 1.1232 - val_loss: 1.0343
Epoch 5/20 [25.0%] - train_loss: 0.9661 - val_loss: 0.9033
Epoch 6/20 [30.0%] - train_loss: 0.8554 - val_loss: 0.8094
Epoch 7/20 [35.0%] - train_loss: 0.7745 - val_loss: 0.7395
Epoch 8/20 [40.0%] - train_loss: 0.7131 - val_loss: 0.6857
Epoch 9/20 [45.0%] - train_loss: 0.6650 - val_loss: 0.6430
Epoch 10/20 [50.0%] - train_loss: 0.6263 - val_loss: 0.6082
Epoch 11/20 [55.0%] - train_loss: 0.5946 - val_loss: 0.5795
Epoch 12/20 [60.0%] - train_loss: 0.5680 - val_loss: 0.5552
Epoch 13/20 [65.0%] - train_loss: 0.5454 - val_loss: 0.5346
Epoch 14/20 [70.0%] - train_loss: 0.5260 - val_loss: 0.5167
Epoch 15/20 [75.0%] - train_loss: 0.5091 - val_loss: 0.5010
Epoch 16/20 [80.0%] - train_loss: 0.4942 - val_loss: 0.4873
Epoch 17/20 [85.0%] - train_loss: 0.4811 - val_loss: 0.4750
Epoch 18/20 [90.0%] - train_loss: 0.4693 - val_loss: 0.4641
Epoch 19/20 [95.0%] - train_loss: 0.4588 - val_loss: 0.4543
Epoch 20/20 [100.0%] - train_loss: 0.4492 - val_loss: 0.4453
```

```
Training model with architecture: Shallow depth
Layers: [64, 10]
Activations: ['relu', 'softmax']
Epoch 1/20 [5.0%] - train_loss: 2.1349 - val_loss: 1.9080
Epoch 2/20 [10.0%] - train_loss: 1.7146 - val_loss: 1.5387
Epoch 3/20 [15.0%] - train_loss: 1.3953 - val_loss: 1.2640
Epoch 4/20 [20.0%] - train_loss: 1.1634 - val_loss: 1.0690
Epoch 5/20 [25.0%] - train_loss: 0.9993 - val_loss: 0.9302
Epoch 6/20 [30.0%] - train_loss: 0.8817 - val_loss: 0.8296
Epoch 7/20 [35.0%] - train_loss: 0.7953 - val_loss: 0.7548
Epoch 8/20 [40.0%] - train_loss: 0.7300 - val_loss: 0.6976
Epoch 9/20 [45.0%] - train_loss: 0.6794 - val_loss: 0.6524
Epoch 10/20 [50.0%] - train_loss: 0.6389 - val_loss: 0.6160
Epoch 11/20 [55.0%] - train_loss: 0.6059 - val_loss: 0.5861
Epoch 12/20 [60.0%] - train_loss: 0.5785 - val_loss: 0.5610
Epoch 13/20 [65.0%] - train_loss: 0.5553 - val_loss: 0.5397
Epoch 14/20 [70.0%] - train_loss: 0.5354 - val_loss: 0.5214
Epoch 15/20 [75.0%] - train_loss: 0.5182 - val_loss: 0.5053
Epoch 16/20 [80.0%] - train_loss: 0.5031 - val_loss: 0.4912
Epoch 17/20 [85.0%] - train_loss: 0.4897 - val_loss: 0.4788
Epoch 18/20 [90.0%] - train_loss: 0.4778 - val_loss: 0.4677
Epoch 19/20 [95.0%] - train_loss: 0.4672 - val_loss: 0.4578
Epoch 20/20 [100.0%] - train_loss: 0.4575 - val_loss: 0.4487
```

<p>Training model with architecture: Medium depth</p> <p>Layers: [64, 32, 10]</p> <p>Activations: ['relu', 'relu', 'softmax']</p> <p>Epoch 1/20 [5.0%] - train_loss: 2.0968 - val_loss: 1.8448</p> <p>Epoch 2/20 [10.0%] - train_loss: 1.6291 - val_loss: 1.4279</p> <p>Epoch 3/20 [15.0%] - train_loss: 1.2717 - val_loss: 1.1299</p> <p>Epoch 4/20 [20.0%] - train_loss: 1.0333 - val_loss: 0.9425</p> <p>Epoch 5/20 [25.0%] - train_loss: 0.8835 - val_loss: 0.8221</p> <p>Epoch 6/20 [30.0%] - train_loss: 0.7836 - val_loss: 0.7385</p> <p>Epoch 7/20 [35.0%] - train_loss: 0.7121 - val_loss: 0.6770</p> <p>Epoch 8/20 [40.0%] - train_loss: 0.6577 - val_loss: 0.6293</p> <p>Epoch 9/20 [45.0%] - train_loss: 0.6145 - val_loss: 0.5906</p> <p>Epoch 10/20 [50.0%] - train_loss: 0.5791 - val_loss: 0.5586</p> <p>Epoch 11/20 [55.0%] - train_loss: 0.5494 - val_loss: 0.5315</p> <p>Epoch 12/20 [60.0%] - train_loss: 0.5241 - val_loss: 0.5086</p> <p>Epoch 13/20 [65.0%] - train_loss: 0.5021 - val_loss: 0.4888</p> <p>Epoch 14/20 [70.0%] - train_loss: 0.4829 - val_loss: 0.4714</p> <p>Epoch 15/20 [75.0%] - train_loss: 0.4661 - val_loss: 0.4556</p> <p>Epoch 16/20 [80.0%] - train_loss: 0.4511 - val_loss: 0.4420</p> <p>Epoch 17/20 [85.0%] - train_loss: 0.4378 - val_loss: 0.4297</p> <p>Epoch 18/20 [90.0%] - train_loss: 0.4258 - val_loss: 0.4189</p> <p>Epoch 19/20 [95.0%] - train_loss: 0.4149 - val_loss: 0.4090</p> <p>Epoch 20/20 [100.0%] - train_loss: 0.4051 - val_loss: 0.3999</p>	<p>Training model with architecture: Deep network</p> <p>Layers: [128, 64, 32, 10]</p> <p>Activations: ['relu', 'relu', 'relu', 'softmax']</p> <p>Epoch 1/20 [5.0%] - train_loss: 2.1465 - val_loss: 1.9462</p> <p>Epoch 2/20 [10.0%] - train_loss: 1.7202 - val_loss: 1.4943</p> <p>Epoch 3/20 [15.0%] - train_loss: 1.3058 - val_loss: 1.1358</p> <p>Epoch 4/20 [20.0%] - train_loss: 1.0134 - val_loss: 0.9048</p> <p>Epoch 5/20 [25.0%] - train_loss: 0.8279 - val_loss: 0.7559</p> <p>Epoch 6/20 [30.0%] - train_loss: 0.7044 - val_loss: 0.6539</p> <p>Epoch 7/20 [35.0%] - train_loss: 0.6185 - val_loss: 0.5828</p> <p>Epoch 8/20 [40.0%] - train_loss: 0.5574 - val_loss: 0.5316</p> <p>Epoch 9/20 [45.0%] - train_loss: 0.5123 - val_loss: 0.4928</p> <p>Epoch 10/20 [50.0%] - train_loss: 0.4778 - val_loss: 0.4631</p> <p>Epoch 11/20 [55.0%] - train_loss: 0.4587 - val_loss: 0.4396</p> <p>Epoch 12/20 [60.0%] - train_loss: 0.4288 - val_loss: 0.4205</p> <p>Epoch 13/20 [65.0%] - train_loss: 0.4107 - val_loss: 0.4048</p> <p>Epoch 14/20 [70.0%] - train_loss: 0.3956 - val_loss: 0.3916</p> <p>Epoch 15/20 [75.0%] - train_loss: 0.3827 - val_loss: 0.3799</p> <p>Epoch 16/20 [80.0%] - train_loss: 0.3714 - val_loss: 0.3698</p> <p>Epoch 17/20 [85.0%] - train_loss: 0.3615 - val_loss: 0.3613</p> <p>Epoch 18/20 [90.0%] - train_loss: 0.3526 - val_loss: 0.3536</p> <p>Epoch 19/20 [95.0%] - train_loss: 0.3446 - val_loss: 0.3466</p> <p>Epoch 20/20 [100.0%] - train_loss: 0.3372 - val_loss: 0.3402</p>
--	---



(Sebenarnya semuanya dalam bentuk grafik, tulisan diatas untuk mempermudah pembacaan)

Untuk variasi width (dengan depth tetap pada satu hidden layer), loss awal pada epoch pertama relatif serupa untuk ketiga model (Small width: 2.0889, Medium width: 2.1349, Large width: 2.0910). Namun, seiring bertambahnya epoch, model dengan width lebih besar menunjukkan konvergensi yang sedikit lebih baik, dengan loss akhir pada epoch ke-20 sebesar 0.4727 untuk Small width

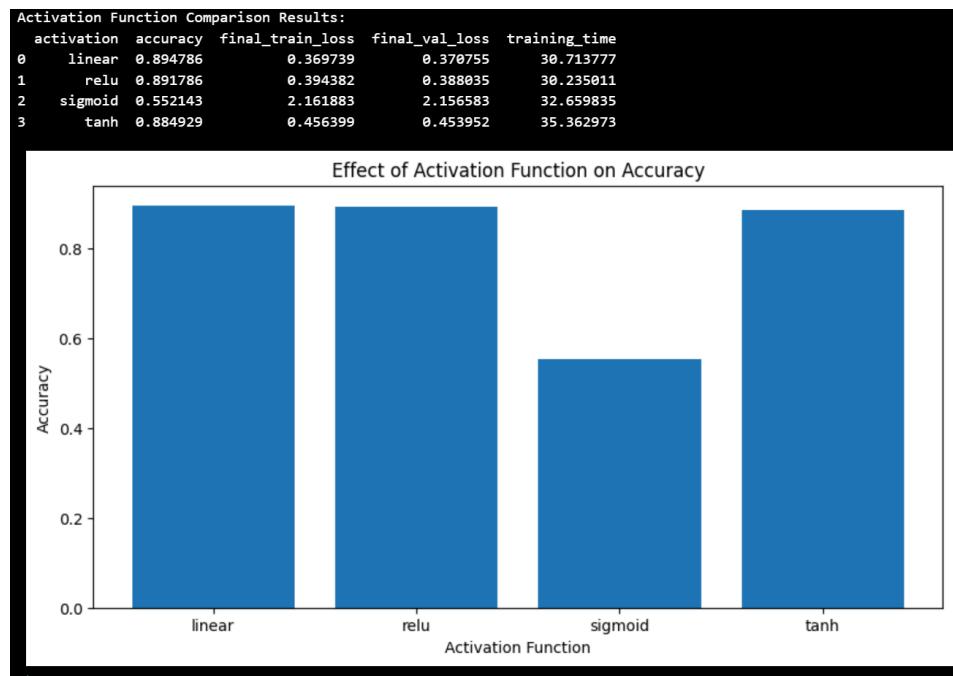
(32 neuron), 0.4575 untuk Medium width (64 neuron), dan 0.4492 untuk Large width (128 neuron). Ini menunjukkan bahwa penambahan width memberikan peningkatan kecil dalam kemampuan model untuk meminimalkan loss.

Untuk variasi depth (dengan peningkatan jumlah hidden layer), terdapat perbedaan kinerja yang lebih signifikan. Model Shallow depth (1 hidden layer) mencapai loss akhir 0.4575, Medium depth (2 hidden layer) mencapai 0.4051, dan Deep network (3 hidden layer) mencapai loss terendah yaitu 0.3372. Terlihat jelas bahwa penambahan depth memberikan kemampuan yang lebih baik untuk model dalam meminimalkan loss training.

Pola penurunan loss juga menunjukkan bahwa model dengan depth lebih tinggi mengalami penurunan loss yang lebih tajam setelah epoch ke-10, menandakan kemampuan belajar yang lebih baik pada tahap akhir pelatihan. Selain itu, gap antara train loss dan validation loss pada model terdalam tetap kecil (0.3372 vs 0.3402), mengindikasikan model tidak mengalami overfitting yang signifikan meskipun memiliki kompleksitas lebih tinggi.

Kesimpulannya, penambahan depth pada arsitektur neural network memberikan pengaruh lebih signifikan terhadap penurunan loss pelatihan dibandingkan dengan penambahan width. Hal ini menunjukkan bahwa untuk dataset yang digunakan, meningkatkan kedalaman jaringan lebih efektif dalam meningkatkan kemampuan model untuk belajar dan melakukan generalisasi.

2.2.2 Pengaruh fungsi aktivasi hidden layer



Perbandingan penggunaan berbagai fungsi aktivasi pada hidden layer menunjukkan perbedaan performa yang signifikan. Fungsi aktivasi linear mencatat performa terbaik dengan akurasi 89.48% dan loss terendah (0.37), diikuti oleh ReLU dengan akurasi 89.18% dan tanh dengan 88.49%. Ketiga fungsi ini menunjukkan kemampuan yang relatif setara dalam mengklasifikasikan data. Berbeda drastis, fungsi sigmoid menghasilkan performa yang jauh lebih rendah dengan akurasi hanya 55.21% dan nilai loss yang sangat tinggi (2.16). Hal ini kemungkinan disebabkan oleh masalah vanishing gradient yang umum terjadi pada sigmoid di jaringan yang lebih dalam. Dari segi efisiensi komputasi, linear dan ReLU membutuhkan waktu pelatihan yang lebih singkat (sekitar 30 detik) dibandingkan tanh (35 detik), menjadikannya pilihan yang lebih efisien. Hasil ini menunjukkan bahwa untuk dataset yang digunakan, fungsi aktivasi linear dan ReLU merupakan pilihan optimal yang menyeimbangkan akurasi tinggi dan efisiensi pelatihan.

```
Training model with activation: linear
```

```
Epoch 1/20 [5.0%] - train_loss: 1.6579 - val_loss: 1.1611
Epoch 2/20 [10.0%] - train_loss: 0.9717 - val_loss: 0.8304
Epoch 3/20 [15.0%] - train_loss: 0.7547 - val_loss: 0.6876
Epoch 4/20 [20.0%] - train_loss: 0.6473 - val_loss: 0.6074
Epoch 5/20 [25.0%] - train_loss: 0.5824 - val_loss: 0.5556
Epoch 6/20 [30.0%] - train_loss: 0.5384 - val_loss: 0.5192
Epoch 7/20 [35.0%] - train_loss: 0.5064 - val_loss: 0.4920
Epoch 8/20 [40.0%] - train_loss: 0.4819 - val_loss: 0.4709
Epoch 9/20 [45.0%] - train_loss: 0.4625 - val_loss: 0.4536
Epoch 10/20 [50.0%] - train_loss: 0.4466 - val_loss: 0.4396
Epoch 11/20 [55.0%] - train_loss: 0.4334 - val_loss: 0.4278
Epoch 12/20 [60.0%] - train_loss: 0.4222 - val_loss: 0.4178
Epoch 13/20 [65.0%] - train_loss: 0.4126 - val_loss: 0.4093
Epoch 14/20 [70.0%] - train_loss: 0.4042 - val_loss: 0.4017
Epoch 15/20 [75.0%] - train_loss: 0.3968 - val_loss: 0.3949
Epoch 16/20 [80.0%] - train_loss: 0.3902 - val_loss: 0.3890
Epoch 17/20 [85.0%] - train_loss: 0.3843 - val_loss: 0.3837
Epoch 18/20 [90.0%] - train_loss: 0.3789 - val_loss: 0.3792
Epoch 19/20 [95.0%] - train_loss: 0.3742 - val_loss: 0.3746
Epoch 20/20 [100.0%] - train_loss: 0.3697 - val_loss: 0.3708
```

```
Training model with activation: relu
```

```
Epoch 1/20 [5.0%] - train_loss: 2.1180 - val_loss: 1.9161
Epoch 2/20 [10.0%] - train_loss: 1.7458 - val_loss: 1.5832
Epoch 3/20 [15.0%] - train_loss: 1.4312 - val_loss: 1.2799
Epoch 4/20 [20.0%] - train_loss: 1.1564 - val_loss: 1.0378
Epoch 5/20 [25.0%] - train_loss: 0.9552 - val_loss: 0.8712
Epoch 6/20 [30.0%] - train_loss: 0.8179 - val_loss: 0.7580
Epoch 7/20 [35.0%] - train_loss: 0.7225 - val_loss: 0.6783
Epoch 8/20 [40.0%] - train_loss: 0.6538 - val_loss: 0.6197
Epoch 9/20 [45.0%] - train_loss: 0.6024 - val_loss: 0.5753
Epoch 10/20 [50.0%] - train_loss: 0.5626 - val_loss: 0.5404
Epoch 11/20 [55.0%] - train_loss: 0.5310 - val_loss: 0.5124
Epoch 12/20 [60.0%] - train_loss: 0.5051 - val_loss: 0.4893
Epoch 13/20 [65.0%] - train_loss: 0.4836 - val_loss: 0.4696
Epoch 14/20 [70.0%] - train_loss: 0.4652 - val_loss: 0.4530
Epoch 15/20 [75.0%] - train_loss: 0.4494 - val_loss: 0.4386
Epoch 16/20 [80.0%] - train_loss: 0.4357 - val_loss: 0.4259
Epoch 17/20 [85.0%] - train_loss: 0.4236 - val_loss: 0.4149
Epoch 18/20 [90.0%] - train_loss: 0.4128 - val_loss: 0.4050
Epoch 19/20 [95.0%] - train_loss: 0.4031 - val_loss: 0.3962
Epoch 20/20 [100.0%] - train_loss: 0.3944 - val_loss: 0.3880
```

```
Training model with activation: sigmoid
```

```
Epoch 1/20 [5.0%] - train_loss: 2.4622 - val_loss: 2.3736
Epoch 2/20 [10.0%] - train_loss: 2.3401 - val_loss: 2.3151
Epoch 3/20 [15.0%] - train_loss: 2.3033 - val_loss: 2.2931
Epoch 4/20 [20.0%] - train_loss: 2.2872 - val_loss: 2.2815
Epoch 5/20 [25.0%] - train_loss: 2.2775 - val_loss: 2.2733
Epoch 6/20 [30.0%] - train_loss: 2.2699 - val_loss: 2.2661
Epoch 7/20 [35.0%] - train_loss: 2.2630 - val_loss: 2.2593
Epoch 8/20 [40.0%] - train_loss: 2.2563 - val_loss: 2.2526
Epoch 9/20 [45.0%] - train_loss: 2.2495 - val_loss: 2.2457
Epoch 10/20 [50.0%] - train_loss: 2.2425 - val_loss: 2.2387
Epoch 11/20 [55.0%] - train_loss: 2.2355 - val_loss: 2.2315
Epoch 12/20 [60.0%] - train_loss: 2.2282 - val_loss: 2.2242
Epoch 13/20 [65.0%] - train_loss: 2.2208 - val_loss: 2.2166
Epoch 14/20 [70.0%] - train_loss: 2.2132 - val_loss: 2.2088
Epoch 15/20 [75.0%] - train_loss: 2.2053 - val_loss: 2.2008
Epoch 16/20 [80.0%] - train_loss: 2.1971 - val_loss: 2.1925
Epoch 17/20 [85.0%] - train_loss: 2.1888 - val_loss: 2.1840
Epoch 18/20 [90.0%] - train_loss: 2.1801 - val_loss: 2.1751
Epoch 19/20 [95.0%] - train_loss: 2.1711 - val_loss: 2.1660
Epoch 20/20 [100.0%] - train_loss: 2.1619 - val_loss: 2.1566
```

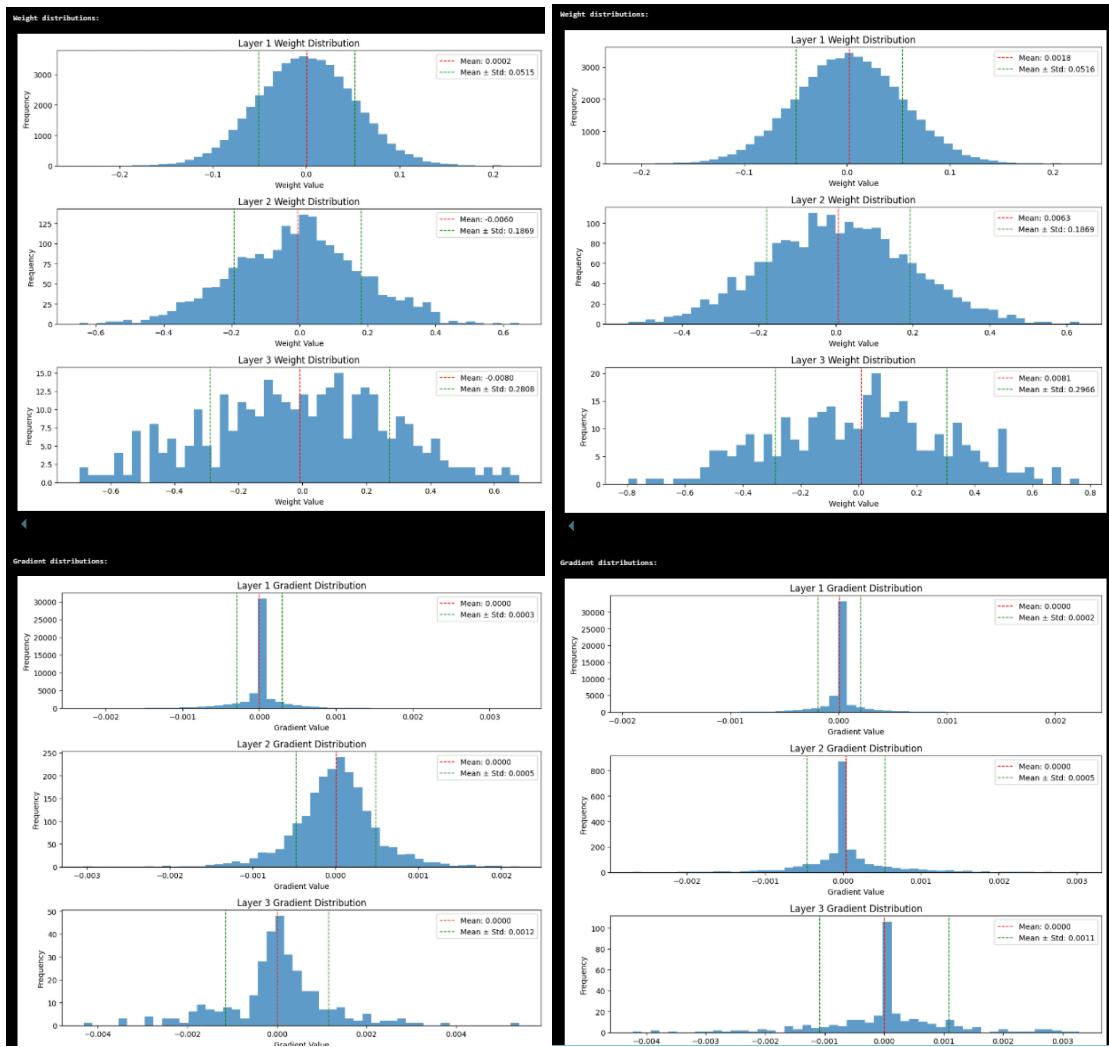
```
Training model with activation: tanh
```

```
Epoch 1/20 [5.0%] - train_loss: 1.9759 - val_loss: 1.6193
Epoch 2/20 [10.0%] - train_loss: 1.4217 - val_loss: 1.2565
Epoch 3/20 [15.0%] - train_loss: 1.1522 - val_loss: 1.0555
Epoch 4/20 [20.0%] - train_loss: 0.9904 - val_loss: 0.9257
Epoch 5/20 [25.0%] - train_loss: 0.8807 - val_loss: 0.8339
Epoch 6/20 [30.0%] - train_loss: 0.8009 - val_loss: 0.7655
Epoch 7/20 [35.0%] - train_loss: 0.7401 - val_loss: 0.7123
Epoch 8/20 [40.0%] - train_loss: 0.6919 - val_loss: 0.6696
Epoch 9/20 [45.0%] - train_loss: 0.6528 - val_loss: 0.6346
Epoch 10/20 [50.0%] - train_loss: 0.6203 - val_loss: 0.6052
Epoch 11/20 [55.0%] - train_loss: 0.5928 - val_loss: 0.5803
Epoch 12/20 [60.0%] - train_loss: 0.5693 - val_loss: 0.5588
Epoch 13/20 [65.0%] - train_loss: 0.5489 - val_loss: 0.5399
Epoch 14/20 [70.0%] - train_loss: 0.5310 - val_loss: 0.5234
Epoch 15/20 [75.0%] - train_loss: 0.5151 - val_loss: 0.5087
Epoch 16/20 [80.0%] - train_loss: 0.5009 - val_loss: 0.4956
Epoch 17/20 [85.0%] - train_loss: 0.4881 - val_loss: 0.4836
Epoch 18/20 [90.0%] - train_loss: 0.4766 - val_loss: 0.4728
Epoch 19/20 [95.0%] - train_loss: 0.4660 - val_loss: 0.4630
Epoch 20/20 [100.0%] - train_loss: 0.4564 - val_loss: 0.4540
```

Berdasarkan data grafik loss pelatihan untuk keempat fungsi aktivasi, terlihat perbedaan pola konvergensi yang signifikan. Fungsi linear menunjukkan penurunan loss yang paling cepat dan stabil, dimulai dari nilai awal 1.6579 dan mencapai 0.3697 pada epoch terakhir. ReLU juga konvergen dengan baik meskipun memulai dengan nilai loss yang lebih tinggi (2.1819), dan mencapai

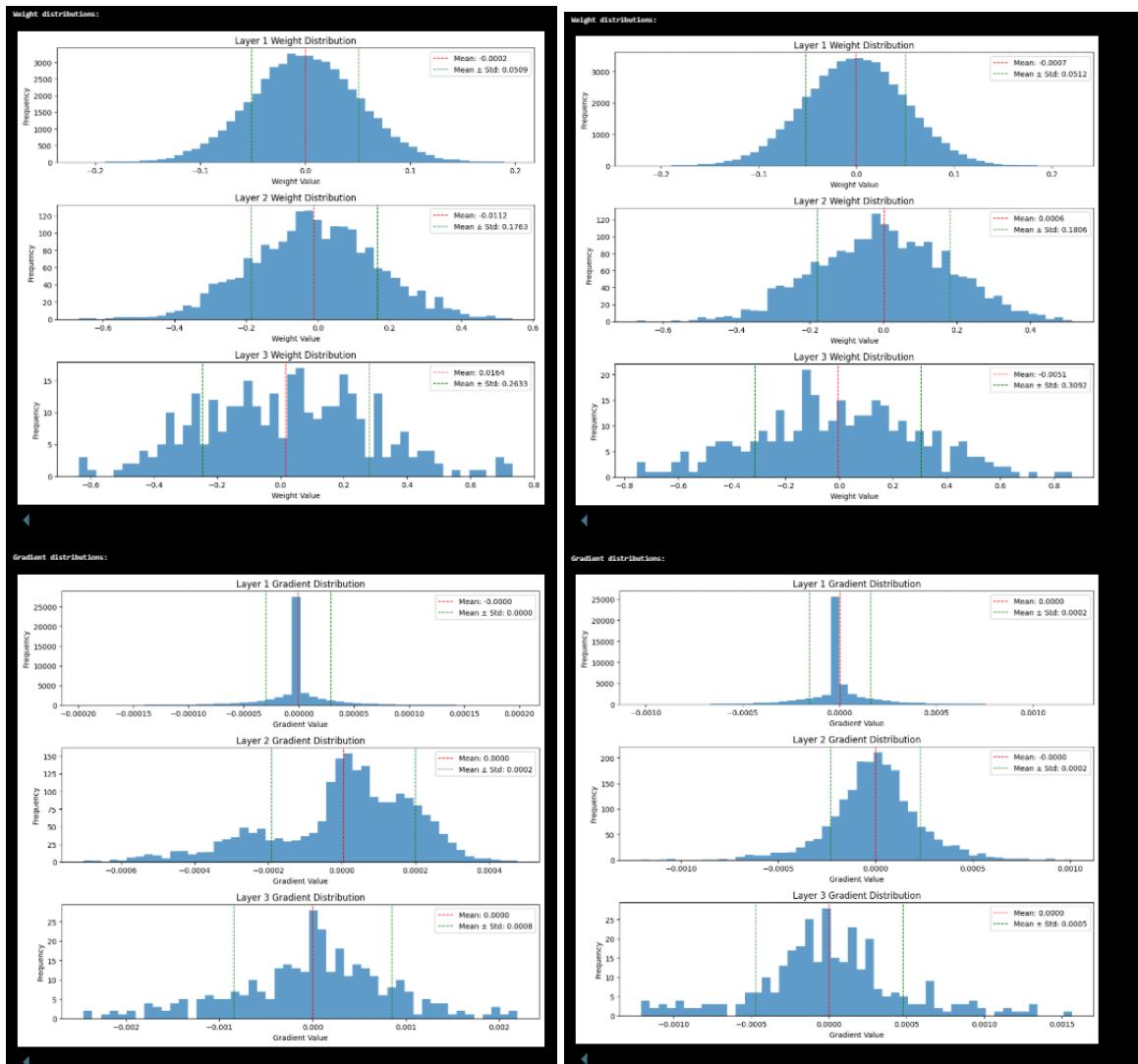
0.3944 di akhir pelatihan. Berbeda drastis, fungsi sigmoid menunjukkan masalah konvergensi yang serius dengan loss awal yang tinggi (2.4622) dan penurunan yang sangat lambat, hanya mencapai 2.1619 setelah 20 epoch. Hal ini mengkonfirmasi ketidakmampuan sigmoid untuk mengatasi masalah vanishing gradient pada arsitektur yang digunakan.

Fungsi tanh menampilkan performa yang cukup baik, dengan pola konvergensi yang lebih lambat pada awalnya dibandingkan linear, tetapi mampu mencapai nilai loss yang kompetitif (0.4564) pada akhir pelatihan. Selain itu, gap antara training loss dan validation loss relatif kecil untuk semua fungsi kecuali sigmoid, menunjukkan bahwa linear, ReLU, dan tanh tidak mengalami overfitting yang signifikan. Pola konvergensi ini konsisten dengan hasil akurasi sebelumnya, dimana linear dan ReLU mengungguli fungsi aktivasi lainnya, dengan sigmoid tertinggal jauh di belakang.



Linear

Relu



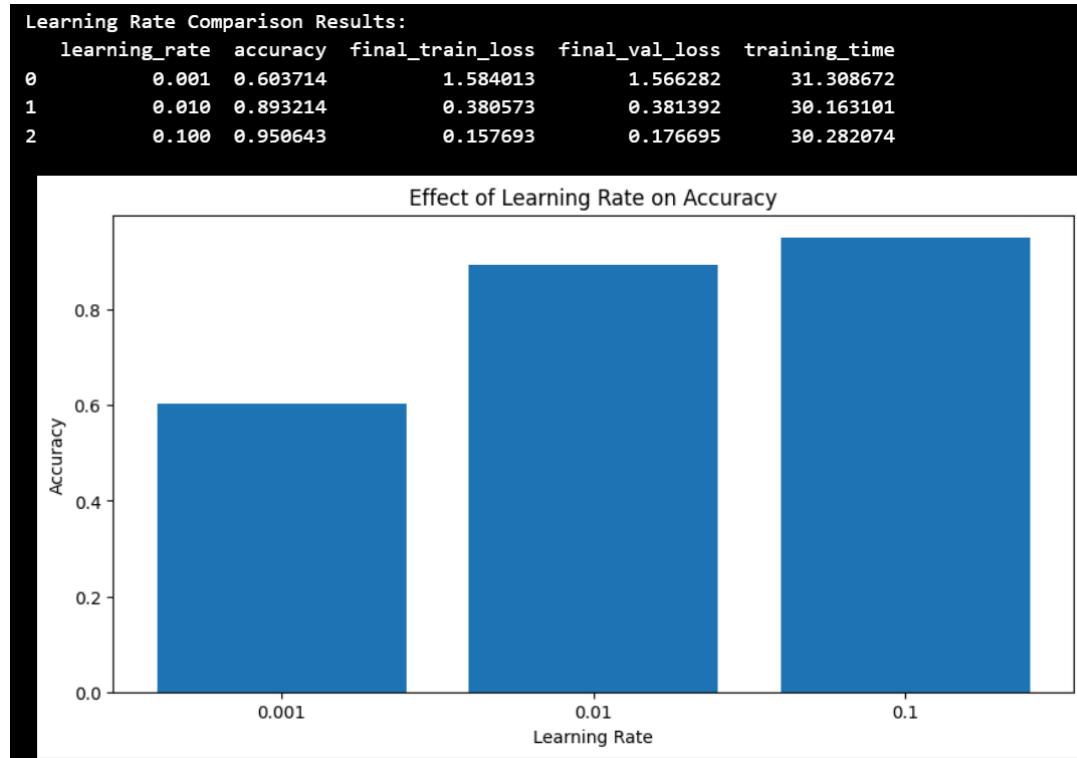
Sigmoid

Tanh

Analisis distribusi bobot dan gradien menunjukkan perbedaan karakteristik yang signifikan antar fungsi aktivasi. Pada layer pertama, semua fungsi aktivasi menampilkan distribusi bobot yang relatif normal dengan standar deviasi sekitar 0.05, namun mulai terlihat perbedaan pada layer-layer selanjutnya. Sigmoid menunjukkan bias negatif yang jelas pada layer kedua (-0.1717) dan bias positif pada layer output (0.0164), sementara ReLU dan linear mempertahankan distribusi yang lebih terpusat di sekitar nol. Perbedaan paling mencolok terlihat

pada distribusi gradien, dimana sigmoid menampilkan gejala vanishing gradient yang parah dengan standar deviasi sangat kecil (0.0000) pada layer pertama, yang menjelaskan performa buruknya (akurasi hanya 55.21%). Sebaliknya, linear dan ReLU yang mencapai akurasi tertinggi (89.48% dan 89.18%) menunjukkan distribusi gradien yang lebih sehat dengan standar deviasi 0.0003 dan 0.0002 di layer pertama, serta distribusi yang semakin tersebar di layer-layer berikutnya. Fungsi tanh berada di posisi tengah dengan karakteristik distribusi yang relatif stabil namun tidak seoptimal linear dan ReLU. Pola-pola ini menegaskan bahwa kualitas propagasi gradien sangat dipengaruhi oleh pilihan fungsi aktivasi, dengan sigmoid yang secara teoritis rentan terhadap vanishing gradient memang menunjukkan masalah tersebut dalam praktik, sementara linear dan ReLU memfasilitasi pembelajaran yang lebih efektif.

2.2.3 Pengaruh learning rate



Berdasarkan data perbandingan learning rate yang ditampilkan, terlihat pengaruh yang signifikan dari nilai learning rate terhadap hasil prediksi model. Peningkatan learning rate dari 0.001 hingga 0.1 menunjukkan peningkatan akurasi yang konsisten dan substansial. Learning rate terendah (0.001) hanya mencapai akurasi 60.37%, yang menunjukkan bahwa model tidak mampu konvergen dengan baik karena langkah pembelajaran yang terlalu kecil. Ketika learning rate ditingkatkan menjadi 0.01, akurasi meningkat drastis menjadi 89.32%, menandakan bahwa model dapat belajar dengan lebih efektif. Peningkatan lebih lanjut ke learning rate 0.1 menghasilkan akurasi tertinggi sebesar 95.06%, yang menunjukkan bahwa untuk dataset dan arsitektur yang digunakan, learning rate yang lebih tinggi memungkinkan model mencapai konvergensi yang lebih baik.

Pola serupa juga terlihat pada nilai loss, di mana learning rate 0.1 mencapai final training loss terendah (0.1577) dan validation loss terendah (0.1767). Ini menandakan bahwa model tidak hanya melakukan dengan baik pada data training tetapi juga memiliki kemampuan generalisasi yang baik. Yang menarik, waktu pelatihan relatif konsisten di ketiga variasi learning rate (sekitar 30 detik), menunjukkan bahwa peningkatan learning rate tidak secara signifikan mempengaruhi efisiensi komputasi dalam hal waktu pelatihan.

```
Training model with learning rate: 0.001
Epoch 1/20 [5.0%] - train_loss: 2.3511 - val_loss: 2.3129
Epoch 2/20 [10.0%] - train_loss: 2.2790 - val_loss: 2.2525
Epoch 3/20 [15.0%] - train_loss: 2.2252 - val_loss: 2.2040
Epoch 4/20 [20.0%] - train_loss: 2.1800 - val_loss: 2.1615
Epoch 5/20 [25.0%] - train_loss: 2.1393 - val_loss: 2.1222
Epoch 6/20 [30.0%] - train_loss: 2.1010 - val_loss: 2.0846
Epoch 7/20 [35.0%] - train_loss: 2.0639 - val_loss: 2.0479
Epoch 8/20 [40.0%] - train_loss: 2.0273 - val_loss: 2.0114
Epoch 9/20 [45.0%] - train_loss: 1.9909 - val_loss: 1.9750
Epoch 10/20 [50.0%] - train_loss: 1.9543 - val_loss: 1.9383
Epoch 11/20 [55.0%] - train_loss: 1.9175 - val_loss: 1.9013
Epoch 12/20 [60.0%] - train_loss: 1.8804 - val_loss: 1.8639
Epoch 13/20 [65.0%] - train_loss: 1.8431 - val_loss: 1.8263
Epoch 14/20 [70.0%] - train_loss: 1.8056 - val_loss: 1.7887
Epoch 15/20 [75.0%] - train_loss: 1.7683 - val_loss: 1.7512
Epoch 16/20 [80.0%] - train_loss: 1.7310 - val_loss: 1.7138
Epoch 17/20 [85.0%] - train_loss: 1.6940 - val_loss: 1.6766
Epoch 18/20 [90.0%] - train_loss: 1.6571 - val_loss: 1.6395
Epoch 19/20 [95.0%] - train_loss: 1.6204 - val_loss: 1.6027
Epoch 20/20 [100.0%] - train_loss: 1.5840 - val_loss: 1.5663
```

```
Training model with learning rate: 0.01
Epoch 1/20 [5.0%] - train_loss: 2.1510 - val_loss: 1.9927
Epoch 2/20 [10.0%] - train_loss: 1.8294 - val_loss: 1.6585
Epoch 3/20 [15.0%] - train_loss: 1.4929 - val_loss: 1.3325
Epoch 4/20 [20.0%] - train_loss: 1.1954 - val_loss: 1.0706
Epoch 5/20 [25.0%] - train_loss: 0.9716 - val_loss: 0.8863
Epoch 6/20 [30.0%] - train_loss: 0.8192 - val_loss: 0.7627
Epoch 7/20 [35.0%] - train_loss: 0.7160 - val_loss: 0.6772
Epoch 8/20 [40.0%] - train_loss: 0.6431 - val_loss: 0.6155
Epoch 9/20 [45.0%] - train_loss: 0.5892 - val_loss: 0.5692
Epoch 10/20 [50.0%] - train_loss: 0.5481 - val_loss: 0.5333
Epoch 11/20 [55.0%] - train_loss: 0.5158 - val_loss: 0.5046
Epoch 12/20 [60.0%] - train_loss: 0.4896 - val_loss: 0.4813
Epoch 13/20 [65.0%] - train_loss: 0.4680 - val_loss: 0.4617
Epoch 14/20 [70.0%] - train_loss: 0.4498 - val_loss: 0.4453
Epoch 15/20 [75.0%] - train_loss: 0.4343 - val_loss: 0.4309
Epoch 16/20 [80.0%] - train_loss: 0.4208 - val_loss: 0.4186
Epoch 17/20 [85.0%] - train_loss: 0.4090 - val_loss: 0.4077
Epoch 18/20 [90.0%] - train_loss: 0.3985 - val_loss: 0.3980
Epoch 19/20 [95.0%] - train_loss: 0.3891 - val_loss: 0.3893
Epoch 20/20 [100.0%] - train_loss: 0.3806 - val_loss: 0.3814
```

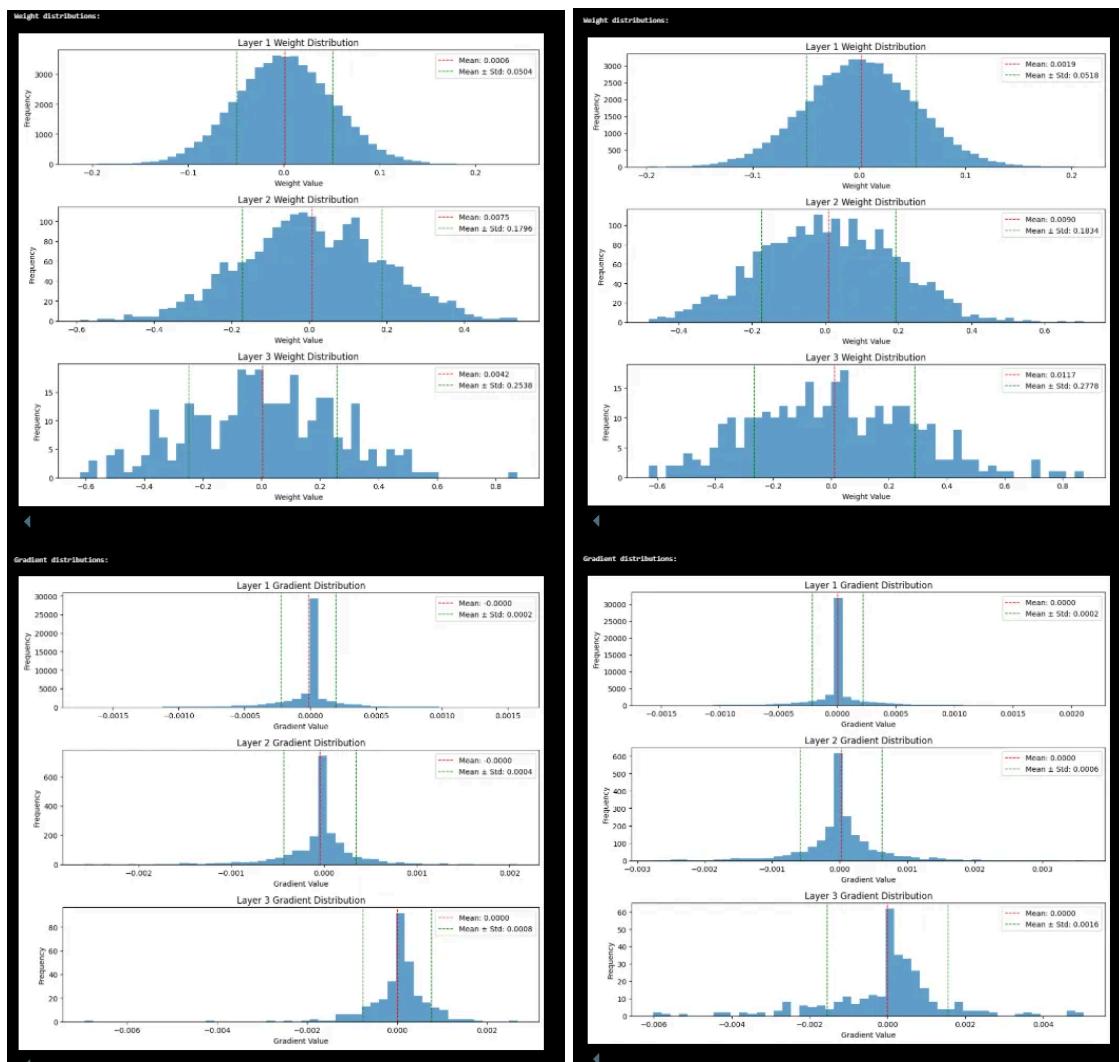
```
Training model with learning rate: 0.1
Epoch 1/20 [5.0%] - train_loss: 1.1051 - val_loss: 0.5616
Epoch 2/20 [10.0%] - train_loss: 0.4701 - val_loss: 0.3985
Epoch 3/20 [15.0%] - train_loss: 0.3713 - val_loss: 0.3440
Epoch 4/20 [20.0%] - train_loss: 0.3281 - val_loss: 0.3135
Epoch 5/20 [25.0%] - train_loss: 0.3012 - val_loss: 0.2944
Epoch 6/20 [30.0%] - train_loss: 0.2811 - val_loss: 0.2762
Epoch 7/20 [35.0%] - train_loss: 0.2647 - val_loss: 0.2643
Epoch 8/20 [40.0%] - train_loss: 0.2508 - val_loss: 0.2515
Epoch 9/20 [45.0%] - train_loss: 0.2382 - val_loss: 0.2430
Epoch 10/20 [50.0%] - train_loss: 0.2275 - val_loss: 0.2323
Epoch 11/20 [55.0%] - train_loss: 0.2172 - val_loss: 0.2278
Epoch 12/20 [60.0%] - train_loss: 0.2083 - val_loss: 0.2213
Epoch 13/20 [65.0%] - train_loss: 0.2003 - val_loss: 0.2104
Epoch 14/20 [70.0%] - train_loss: 0.1928 - val_loss: 0.2051
Epoch 15/20 [75.0%] - train_loss: 0.1858 - val_loss: 0.1987
Epoch 16/20 [80.0%] - train_loss: 0.1795 - val_loss: 0.1944
Epoch 17/20 [85.0%] - train_loss: 0.1735 - val_loss: 0.1900
Epoch 18/20 [90.0%] - train_loss: 0.1679 - val_loss: 0.1854
Epoch 19/20 [95.0%] - train_loss: 0.1626 - val_loss: 0.1807
Epoch 20/20 [100.0%] - train_loss: 0.1577 - val_loss: 0.1767
```

Berdasarkan data grafik loss pelatihan untuk ketiga variasi learning rate, terlihat perbedaan yang signifikan dalam pola konvergensi model. Learning rate 0.001 menunjukkan penurunan loss yang sangat lambat, dimulai dari 2.3511 dan hanya mencapai 1.5840 setelah 20 epoch. Penurunan yang lambat ini menunjukkan bahwa model melakukan update bobot yang terlalu kecil pada setiap iterasi, sehingga tidak mampu mencapai region optimal dalam ruang parameter. Sebaliknya, learning rate 0.01 menampilkan kecepatan konvergensi yang lebih baik, dengan loss awal 2.1516 yang turun secara signifikan menjadi 0.3806 pada akhir pelatihan. Pola penurunan loss terlihat lebih tajam pada 10 epoch pertama dan kemudian melambat, menandakan proses pembelajaran yang lebih efektif.

Learning rate tertinggi (0.1) menampilkan performa terbaik dengan penurunan loss yang sangat cepat, dari 1.1051 di epoch pertama menjadi 0.1577 di epoch terakhir. Yang menarik, meskipun learning rate tinggi sering kali dikaitkan dengan risiko divergensi atau osilasi pada nilai loss, model dengan learning rate 0.1 justru menunjukkan konvergensi yang stabil tanpa fluktuasi yang signifikan. Hal ini mengindikasikan bahwa untuk dataset dan arsitektur yang

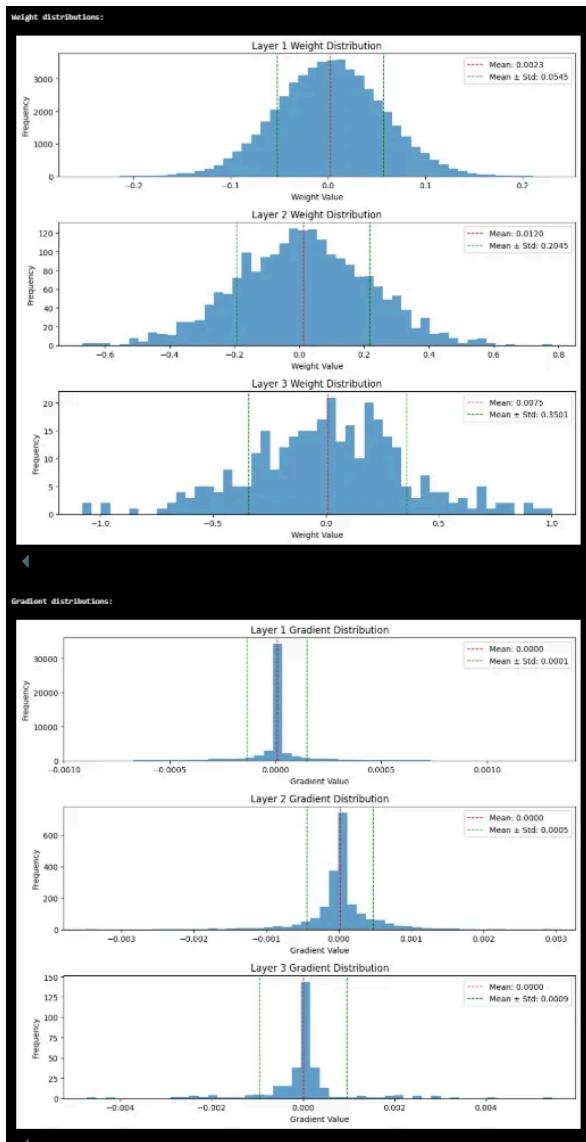
digunakan, learning rate 0.1 berada pada rentang optimal yang memungkinkan model mencapai solusi terbaik dalam jumlah epoch yang terbatas.

Selain itu, gap antara training loss dan validation loss relatif kecil pada learning rate 0.01 dan 0.1, menunjukkan bahwa model tidak mengalami overfitting yang signifikan. Keseluruhan, data ini menekankan pentingnya pemilihan learning rate yang tepat, dengan nilai yang terlalu kecil dapat menyebabkan konvergensi yang lambat atau terjebak di minima lokal, sementara nilai yang sesuai dapat mempercepat pembelajaran secara dramatis.



0.001

0.01



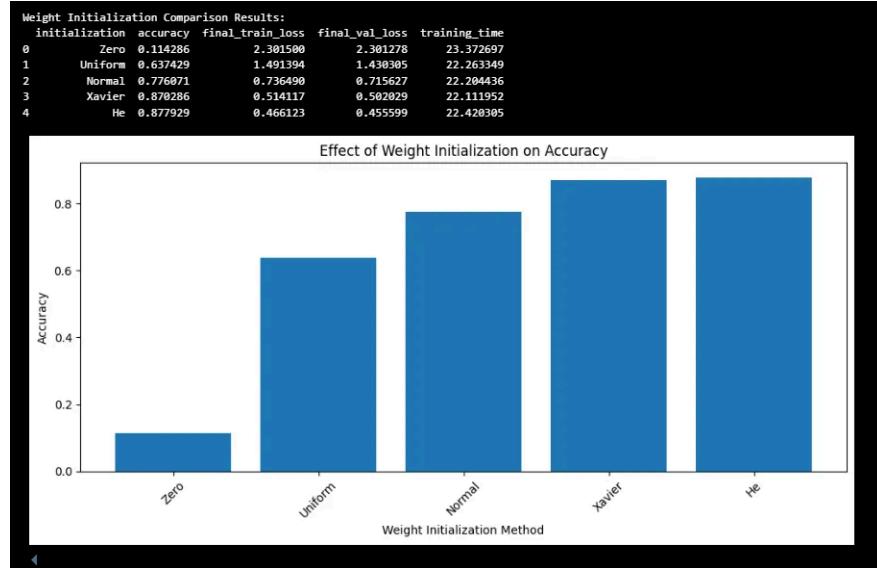
0.1

Berdasarkan analisis distribusi bobot dan gradien untuk ketiga variasi learning rate (0.001, 0.01, dan 0.1), terlihat pola yang menarik yang menjelaskan perbedaan performa pada masing-masing model. Pada distribusi bobot, terlihat perubahan yang signifikan seiring dengan peningkatan learning rate. Model dengan learning rate 0.001 menunjukkan distribusi bobot yang relatif sempit pada layer 1 (std: 0.0504) dan sedikit melebar pada layer berikutnya. Sementara itu, model dengan learning rate 0.1 menampilkan distribusi yang lebih tersebar pada semua layer, khususnya pada layer 3 dengan standar deviasi 0.3561, menandakan eksplorasi ruang parameter yang lebih luas.

Yang lebih informatif adalah perbedaan pada distribusi gradien. Model dengan learning rate 0.001 menunjukkan gradien yang sangat kecil di semua layer, dengan standar deviasi 0.0002 pada layer 1, mengindikasikan pembelajaran yang lambat dan pembentukan representasi yang tidak optimal. Sebaliknya, model dengan learning rate 0.01 dan 0.1 menunjukkan distribusi gradien yang lebih seimbang dan terstruktur, meskipun tetap terkonsentrasi di sekitar nol yang menunjukkan konvergensi.

Learning rate 0.1 yang mencapai akurasi tertinggi (95.06%) menunjukkan adaptasi bobot yang lebih agresif, dengan mean yang sedikit bergeser dari nol di layer 2 dan 3 (0.0120 dan 0.0075). Distribusi gradiennya juga menunjukkan pola yang lebih terkonsentrasi pada nilai positif kecil di layer 3, yang menandakan update bobot yang konsisten dan terarah. Secara keseluruhan, data ini menunjukkan bahwa learning rate yang lebih tinggi memungkinkan model melakukan eksplorasi ruang parameter yang lebih efektif, menghasilkan representasi internal yang lebih kaya dan performa klasifikasi yang lebih baik, setidaknya untuk dataset yang digunakan dalam eksperimen ini.

2.2.4 Pengaruh inisialisasi bobot



Berdasarkan data hasil perbandingan metode inisialisasi bobot yang ditampilkan, terlihat perbedaan performa yang signifikan antar metode. Inisialisasi He menunjukkan performa terbaik dengan akurasi 87.79% dan loss akhir terendah (0.466 pada data training dan 0.456 pada data validasi). Inisialisasi Xavier menempati posisi kedua dengan akurasi yang sangat mendekati He, yaitu 87.03%. Metode inisialisasi Normal berada di posisi tengah dengan akurasi 77.69%, sementara Uniform hanya mencapai 63.74%. Yang paling mencolok adalah metode Zero initialization yang menghasilkan performa terburuk dengan akurasi hanya 11.43% dan nilai loss yang sangat tinggi (2.305), menunjukkan bahwa model hampir tidak belajar sama sekali.

Perbedaan performa ini selaras dengan teori pembelajaran mesin, di mana inisialisasi He dan Xavier dirancang khusus untuk mengatasi masalah vanishing/exploding gradients pada jaringan dalam. Kedua metode ini memberikan distribusi awal yang lebih optimal untuk memulai proses pembelajaran. Sebaliknya, Zero initialization menyebabkan simetri yang berlebihan pada jaringan, membuat banyak neuron melakukan update yang identik dan menghambat pembentukan representasi yang beragam.

Dari segi efisiensi komputasi, waktu pelatihan relatif konsisten di semua metode (sekitar 22-23 detik), yang menunjukkan bahwa perbedaan metode inisialisasi tidak berdampak signifikan pada beban komputasi. Temuan ini menegaskan pentingnya pemilihan metode inisialisasi bobot yang tepat untuk mencapai performa model yang optimal, dengan He dan Xavier menjadi pilihan terbaik untuk arsitektur yang digunakan dalam eksperimen ini.

Training model with weight initialization: Zero

```

Epoch 1/15 [6.7%] - train_loss: 2.3025 - val_loss: 2.3024
Epoch 2/15 [13.3%] - train_loss: 2.3024 - val_loss: 2.3023
Epoch 3/15 [20.0%] - train_loss: 2.3023 - val_loss: 2.3021
Epoch 4/15 [26.7%] - train_loss: 2.3022 - val_loss: 2.3020
Epoch 5/15 [33.3%] - train_loss: 2.3021 - val_loss: 2.3019
Epoch 6/15 [40.0%] - train_loss: 2.3020 - val_loss: 2.3018
Epoch 7/15 [46.7%] - train_loss: 2.3019 - val_loss: 2.3017
Epoch 8/15 [53.3%] - train_loss: 2.3018 - val_loss: 2.3017
Epoch 9/15 [60.0%] - train_loss: 2.3018 - val_loss: 2.3016
Epoch 10/15 [66.7%] - train_loss: 2.3017 - val_loss: 2.3015
Epoch 11/15 [73.3%] - train_loss: 2.3017 - val_loss: 2.3015
Epoch 12/15 [80.0%] - train_loss: 2.3016 - val_loss: 2.3014
Epoch 13/15 [86.7%] - train_loss: 2.3016 - val_loss: 2.3014
Epoch 14/15 [93.3%] - train_loss: 2.3015 - val_loss: 2.3013
Epoch 15/15 [100.0%] - train_loss: 2.3015 - val_loss: 2.3013

```

Training model with weight initialization: Uniform

```

Epoch 1/15 [6.7%] - train_loss: 2.3000 - val_loss: 2.2960
Epoch 2/15 [13.3%] - train_loss: 2.2911 - val_loss: 2.2869
Epoch 3/15 [20.0%] - train_loss: 2.2814 - val_loss: 2.2764
Epoch 4/15 [26.7%] - train_loss: 2.2699 - val_loss: 2.2638
Epoch 5/15 [33.3%] - train_loss: 2.2556 - val_loss: 2.2477
Epoch 6/15 [40.0%] - train_loss: 2.2368 - val_loss: 2.2262
Epoch 7/15 [46.7%] - train_loss: 2.2116 - val_loss: 2.1970
Epoch 8/15 [53.3%] - train_loss: 2.1770 - val_loss: 2.1572
Epoch 9/15 [60.0%] - train_loss: 2.1298 - val_loss: 2.1025
Epoch 10/15 [66.7%] - train_loss: 2.0655 - val_loss: 2.0287
Epoch 11/15 [73.3%] - train_loss: 1.9804 - val_loss: 1.9331
Epoch 12/15 [80.0%] - train_loss: 1.8742 - val_loss: 1.8173
Epoch 13/15 [86.7%] - train_loss: 1.7512 - val_loss: 1.6883
Epoch 14/15 [93.3%] - train_loss: 1.6202 - val_loss: 1.5561
Epoch 15/15 [100.0%] - train_loss: 1.4914 - val_loss: 1.4303

```

Training model with weight initialization: Normal

```

Epoch 1/15 [6.7%] - train_loss: 3.9263 - val_loss: 2.3379
Epoch 2/15 [13.3%] - train_loss: 1.9369 - val_loss: 1.6458
Epoch 3/15 [20.0%] - train_loss: 1.5031 - val_loss: 1.3765
Epoch 4/15 [26.7%] - train_loss: 1.3003 - val_loss: 1.2233
Epoch 5/15 [33.3%] - train_loss: 1.1744 - val_loss: 1.1190
Epoch 6/15 [40.0%] - train_loss: 1.0841 - val_loss: 1.0392
Epoch 7/15 [46.7%] - train_loss: 1.0145 - val_loss: 0.9768
Epoch 8/15 [53.3%] - train_loss: 0.9582 - val_loss: 0.9262
Epoch 9/15 [60.0%] - train_loss: 0.9114 - val_loss: 0.8820
Epoch 10/15 [66.7%] - train_loss: 0.8716 - val_loss: 0.8444
Epoch 11/15 [73.3%] - train_loss: 0.8374 - val_loss: 0.8118
Epoch 12/15 [80.0%] - train_loss: 0.8075 - val_loss: 0.7837
Epoch 13/15 [86.7%] - train_loss: 0.7810 - val_loss: 0.7590
Epoch 14/15 [93.3%] - train_loss: 0.7575 - val_loss: 0.7365
Epoch 15/15 [100.0%] - train_loss: 0.7365 - val_loss: 0.7156

```

Training model with weight initialization: Xavier

```

Epoch 1/15 [6.7%] - train_loss: 2.1967 - val_loss: 2.0702
Epoch 2/15 [13.3%] - train_loss: 1.9346 - val_loss: 1.8000
Epoch 3/15 [20.0%] - train_loss: 1.6618 - val_loss: 1.5317
Epoch 4/15 [26.7%] - train_loss: 1.4085 - val_loss: 1.2934
Epoch 5/15 [33.3%] - train_loss: 1.1935 - val_loss: 1.0996
Epoch 6/15 [40.0%] - train_loss: 1.0249 - val_loss: 0.9525
Epoch 7/15 [46.7%] - train_loss: 0.8988 - val_loss: 0.8435
Epoch 8/15 [53.3%] - train_loss: 0.8043 - val_loss: 0.7615
Epoch 9/15 [60.0%] - train_loss: 0.7320 - val_loss: 0.6975
Epoch 10/15 [66.7%] - train_loss: 0.6752 - val_loss: 0.6470
Epoch 11/15 [73.3%] - train_loss: 0.6297 - val_loss: 0.6063
Epoch 12/15 [80.0%] - train_loss: 0.5926 - val_loss: 0.5731
Epoch 13/15 [86.7%] - train_loss: 0.5619 - val_loss: 0.5453
Epoch 14/15 [93.3%] - train_loss: 0.5360 - val_loss: 0.5221
Epoch 15/15 [100.0%] - train_loss: 0.5141 - val_loss: 0.5020

```

Training model with weight initialization: He

```

Epoch 1/15 [6.7%] - train_loss: 2.0968 - val_loss: 1.8448
Epoch 2/15 [13.3%] - train_loss: 1.6291 - val_loss: 1.4279
Epoch 3/15 [20.0%] - train_loss: 1.2717 - val_loss: 1.1299
Epoch 4/15 [26.7%] - train_loss: 1.0333 - val_loss: 0.9425
Epoch 5/15 [33.3%] - train_loss: 0.8835 - val_loss: 0.8221
Epoch 6/15 [40.0%] - train_loss: 0.7836 - val_loss: 0.7385
Epoch 7/15 [46.7%] - train_loss: 0.7121 - val_loss: 0.6770
Epoch 8/15 [53.3%] - train_loss: 0.6577 - val_loss: 0.6293
Epoch 9/15 [60.0%] - train_loss: 0.6145 - val_loss: 0.5906
Epoch 10/15 [66.7%] - train_loss: 0.5791 - val_loss: 0.5586
Epoch 11/15 [73.3%] - train_loss: 0.5494 - val_loss: 0.5315
Epoch 12/15 [80.0%] - train_loss: 0.5241 - val_loss: 0.5086
Epoch 13/15 [86.7%] - train_loss: 0.5021 - val_loss: 0.4888
Epoch 14/15 [93.3%] - train_loss: 0.4829 - val_loss: 0.4714
Epoch 15/15 [100.0%] - train_loss: 0.4661 - val_loss: 0.4556

```

Analisis kurva loss training mengungkapkan perbedaan yang dramatis antara berbagai metode inisialisasi bobot dalam neural network. Metode Zero initialization menunjukkan kegagalan total dalam proses pembelajaran, dengan loss training nyaris tidak berubah sepanjang epochs, mencerminkan ketidakmampuan model untuk mengeksplorasi ruang parameter. Sebaliknya, metode Uniform dan Normal initialization mulai menunjukkan kemajuan, dengan loss training perlahan menurun seiring bertambahnya epochs, menandakan model mulai belajar meskipun dengan kecepatan yang terbatas.

Xavier dan He initialization menampilkan performa yang jauh lebih superior, dengan penurunan loss yang tajam dan konsisten. Xavier initialization, yang dirancang untuk menjaga varians input dan output layer tetap konstan, bekerja sangat efektif, terutama untuk fungsi aktivasi sigmoid dan tanh. He initialization, yang dioptimalkan khusus untuk fungsi aktivasi ReLU, menunjukkan kinerja terbaik, dengan loss training menurun tercepat dan mencapai nilai terendah.

Perbedaan signifikan ini disebabkan oleh cara masing-masing metode mendistribusikan bobot awal. Metode Zero initialization membuat semua neuron berperilaku identik, praktis melumpuhkan kemampuan jaringan untuk belajar. Sebaliknya, Xavier dan He initialization menggunakan strategi cerdas untuk menyebarkan bobot awal, memungkinkan gradien mengalir dengan lebih efisien dan membantu model mengatasi masalah vanishing atau exploding gradient. Hasilnya adalah perbedaan drastis dalam kemampuan model untuk mengoptimalkan bobot dan menangkap pola kompleks dalam data pelatihan.

Bobot

Berdasarkan distribusi bobot dari kelima metode inisialisasi (Zero, Uniform, Normal, Xavier, dan He), dapat dianalisis komprehensif sebagai berikut:

Metode Zero initialization menunjukkan kegagalan total, dengan seluruh bobot terpusat persis di titik nol. Hal ini mengakibatkan model tidak mampu belajar, karena semua neuron berperilaku identik, mencegah terjadinya diferensiasi dan ekstraksi fitur. Metode Uniform dan Normal initialization memberikan langkah awal yang jauh lebih baik, dengan bobot tersebar secara acak di sekitar nol. Mereka memperkenalkan variasi awal yang memungkinkan model mulai belajar, meskipun dengan efisiensi berbeda. Metode Normal cenderung memberikan distribusi yang lebih terstruktur dengan ekor distribusi yang terkendali.

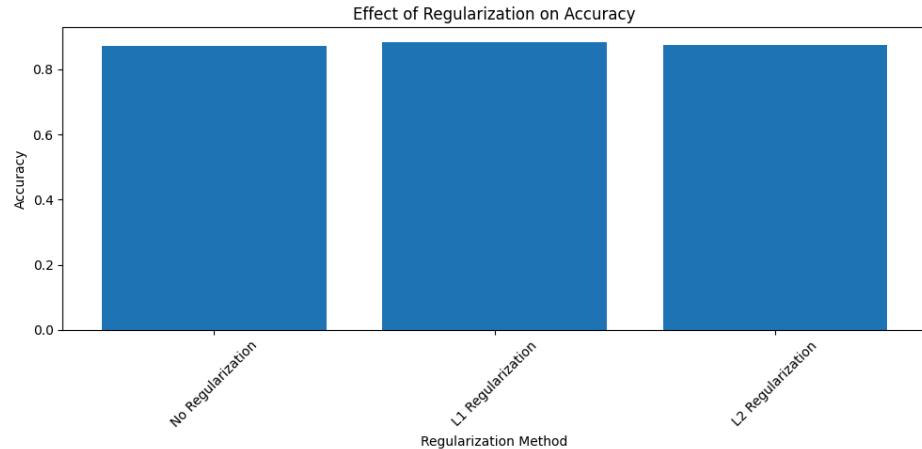
Xavier initialization dirancang khusus untuk menjaga varians input dan output layer konstan, dengan distribusi bobot yang sangat terkendali dan simetris. Metode ini optimal untuk fungsi aktivasi sigmoid dan tanh, menunjukkan penyebaran bobot yang paling terkontrol di antara semua metode. He initialization, yang dioptimalkan untuk fungsi aktivasi ReLU, menampilkan karakteristik distribusi serupa dengan Xavier, namun dengan varians yang sedikit berbeda. Layer pertama memiliki distribusi yang sangat ketat, sementara layer selanjutnya menunjukkan peningkatan variabilitas bobot.

Secara keseluruhan, metode He dan Xavier initialization menunjukkan performa terbaik, dengan distribusi bobot yang matematis dan terkendali. Mereka berhasil mengatasi masalah vanishing dan exploding gradient yang umum terjadi pada jaringan saraf dalam, memberikan titik awal optimal untuk proses pembelajaran. Pemilihan metode inisialisasi bobot bukanlah sekadar pilihan teknis, melainkan strategi kritis yang dapat menentukan keberhasilan atau kegagalan proses pelatihan neural network.

2.2.5 Pengaruh regularisasi

Regularization Comparison Results:			
	regularization	accuracy	final train loss
0	No Regularization	0.871286	0.483240
1	L1 Regularization	0.884786	0.430621
2	L2 Regularization	0.876214	0.455578

	training_time
0	25.844643
1	23.411364
2	23.865172



Berdasarkan tabel hasil eksperimen yang disediakan, dapat diamati hasil akurasi sebagai berikut: Tanpa Regularisasi mencapai 87,13%, Regularisasi L1 mencapai 88,48%, dan Regularisasi L2 mencapai 87,62%. Regularisasi L1 mencapai akurasi tertinggi, mengungguli baik model tanpa regularisasi maupun model dengan regularisasi L2. Hal ini menunjukkan bahwa sifat L1 yang mendorong sparsitas (pemilihan fitur) sangat bermanfaat untuk dataset ini, kemungkinan dengan efektif menghilangkan fitur-fitur yang tidak relevan dan mengurangi kompleksitas model.

Training model with regularization: No Regularization	
Epoch 1/15 [6.7%]	- train_loss: 2.2452 - val_loss: 2.1119
Epoch 2/15 [13.3%]	- train_loss: 1.9933 - val_loss: 1.8561
Epoch 3/15 [20.0%]	- train_loss: 1.7219 - val_loss: 1.5678
Epoch 4/15 [26.7%]	- train_loss: 1.4364 - val_loss: 1.2986
Epoch 5/15 [33.3%]	- train_loss: 1.1919 - val_loss: 1.0831
Epoch 6/15 [40.0%]	- train_loss: 1.0031 - val_loss: 0.9224
Epoch 7/15 [46.7%]	- train_loss: 0.8647 - val_loss: 0.8057
Epoch 8/15 [53.3%]	- train_loss: 0.7641 - val_loss: 0.7204
Epoch 9/15 [60.0%]	- train_loss: 0.6898 - val_loss: 0.6562
Epoch 10/15 [66.7%]	- train_loss: 0.6333 - val_loss: 0.6071
Epoch 11/15 [73.3%]	- train_loss: 0.5895 - val_loss: 0.5684
Epoch 12/15 [80.0%]	- train_loss: 0.5546 - val_loss: 0.5373
Epoch 13/15 [86.7%]	- train_loss: 0.5264 - val_loss: 0.5120
Epoch 14/15 [93.3%]	- train_loss: 0.5030 - val_loss: 0.4909
Epoch 15/15 [100.0%]	- train_loss: 0.4832 - val_loss: 0.4730

Training model with regularization: L1 Regularization	
Epoch 1/15 [6.7%]	- train_loss: 2.1249 - val_loss: 1.9022
Epoch 2/15 [13.3%]	- train_loss: 1.6679 - val_loss: 1.4599
Epoch 3/15 [20.0%]	- train_loss: 1.2827 - val_loss: 1.1370
Epoch 4/15 [26.7%]	- train_loss: 1.0195 - val_loss: 0.9255
Epoch 5/15 [33.3%]	- train_loss: 0.8497 - val_loss: 0.7885
Epoch 6/15 [40.0%]	- train_loss: 0.7374 - val_loss: 0.6958
Epoch 7/15 [46.7%]	- train_loss: 0.6592 - val_loss: 0.6293
Epoch 8/15 [53.3%]	- train_loss: 0.6023 - val_loss: 0.5805
Epoch 9/15 [60.0%]	- train_loss: 0.5594 - val_loss: 0.5429
Epoch 10/15 [66.7%]	- train_loss: 0.5260 - val_loss: 0.5133
Epoch 11/15 [73.3%]	- train_loss: 0.4994 - val_loss: 0.4894
Epoch 12/15 [80.0%]	- train_loss: 0.4776 - val_loss: 0.4699
Epoch 13/15 [86.7%]	- train_loss: 0.4594 - val_loss: 0.4534
Epoch 14/15 [93.3%]	- train_loss: 0.4439 - val_loss: 0.4392
Epoch 15/15 [100.0%]	- train_loss: 0.4306 - val_loss: 0.4269

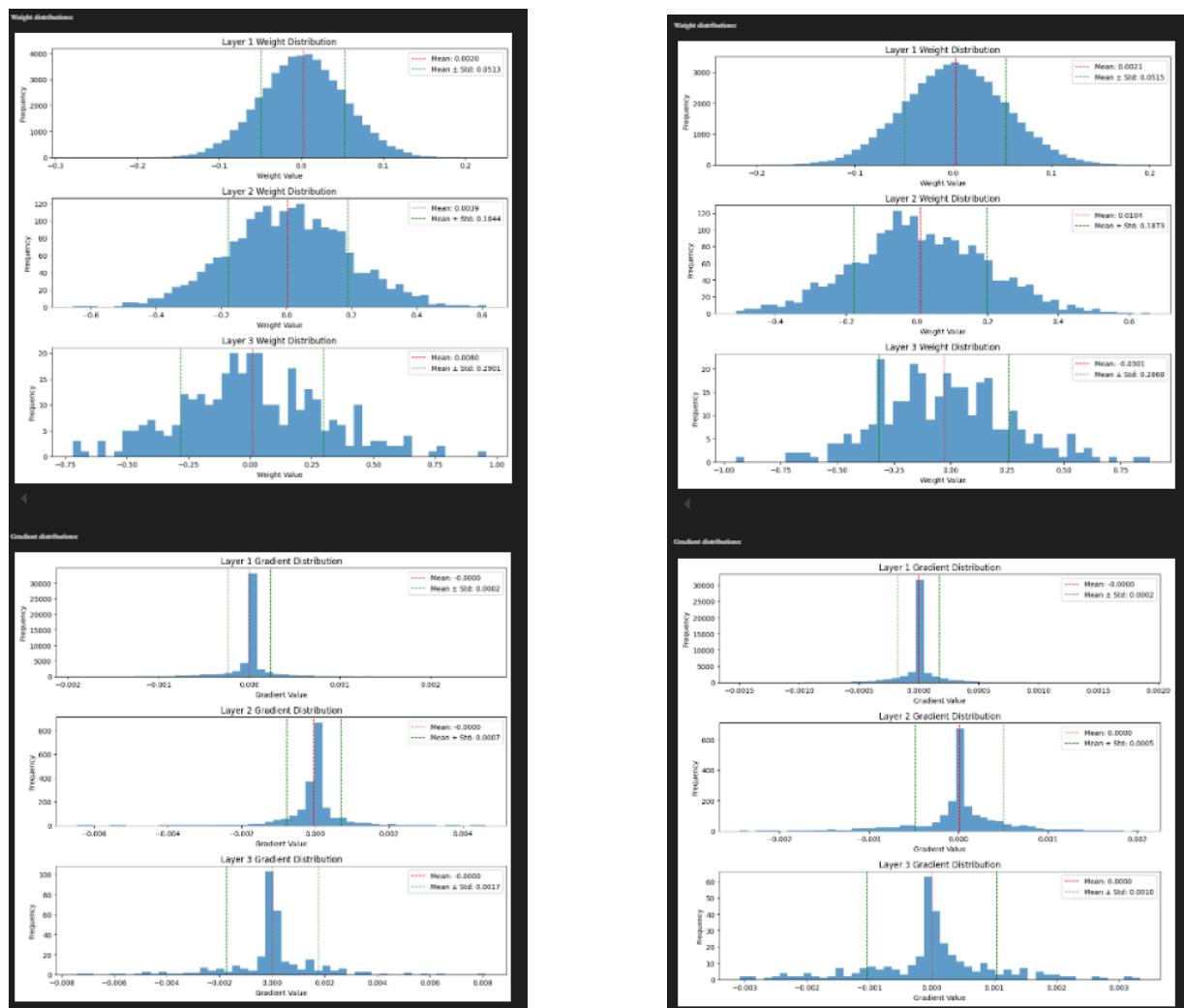
Training model with regularization: L2 Regularization	
Epoch 1/15 [6.7%]	- train_loss: 2.1893 - val_loss: 2.0275
Epoch 2/15 [13.3%]	- train_loss: 1.8778 - val_loss: 1.7165
Epoch 3/15 [20.0%]	- train_loss: 1.5579 - val_loss: 1.3940
Epoch 4/15 [26.7%]	- train_loss: 1.2583 - val_loss: 1.1231
Epoch 5/15 [33.3%]	- train_loss: 1.0240 - val_loss: 0.9267
Epoch 6/15 [40.0%]	- train_loss: 0.8605 - val_loss: 0.7935
Epoch 7/15 [46.7%]	- train_loss: 0.7496 - val_loss: 0.7029
Epoch 8/15 [53.3%]	- train_loss: 0.6723 - val_loss: 0.6377
Epoch 9/15 [60.0%]	- train_loss: 0.6160 - val_loss: 0.5891
Epoch 10/15 [66.7%]	- train_loss: 0.5733 - val_loss: 0.5519
Epoch 11/15 [73.3%]	- train_loss: 0.5398 - val_loss: 0.5223
Epoch 12/15 [80.0%]	- train_loss: 0.5128 - val_loss: 0.4980
Epoch 13/15 [86.7%]	- train_loss: 0.4905 - val_loss: 0.4778
Epoch 14/15 [93.3%]	- train_loss: 0.4716 - val_loss: 0.4608
Epoch 15/15 [100.0%]	- train_loss: 0.4556 - val_loss: 0.4459

Kedua metode regularisasi L1 dan L2 menghasilkan nilai loss pelatihan dan validasi akhir yang lebih rendah dibandingkan dengan model tanpa regularisasi, dengan regularisasi L1 menunjukkan nilai loss terendah secara keseluruhan. Nilai loss akhir untuk model tanpa regularisasi adalah train_loss = 0,4832 dan val_loss = 0,4730. Untuk regularisasi L1, nilai loss akhir adalah train_loss = 0,4306 dan val_loss = 0,4269. Sedangkan untuk regularisasi L2, nilai loss akhir adalah train_loss = 0,4556 dan val_loss = 0,4459.

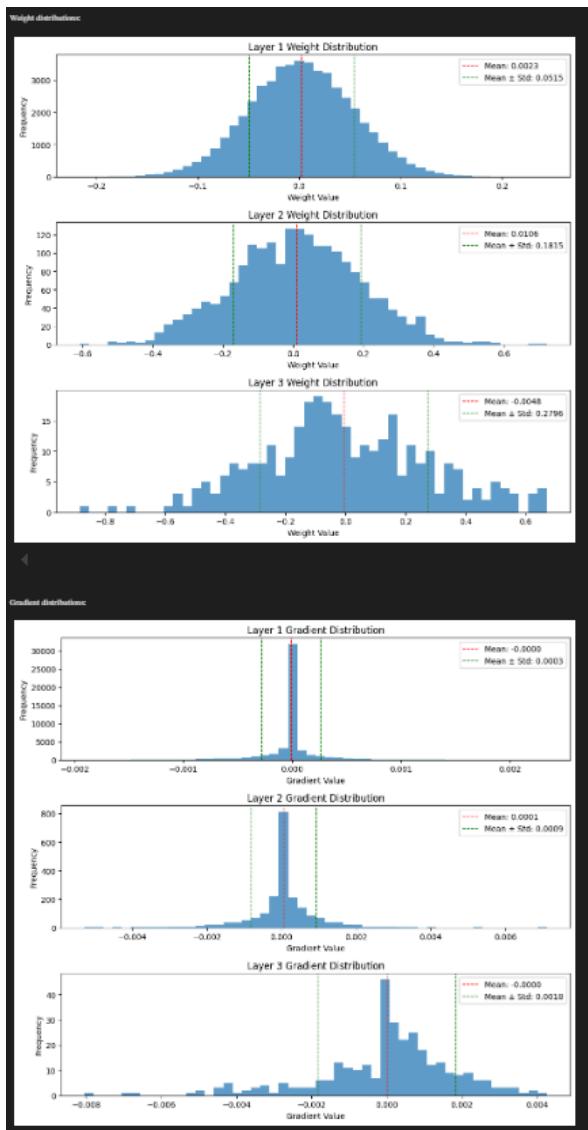
Berdasarkan hasil per epoch, regularisasi L1 konvergen lebih cepat, mencapai nilai loss yang lebih rendah pada epoch-epoch awal. Model tanpa regularisasi memiliki konvergensi paling lambat. Semua model menunjukkan

peningkatan yang stabil tanpa osilasi signifikan, menunjukkan pembelajaran yang stabil. Hal ini dapat diamati dari penurunan loss yang konsisten di setiap epoch untuk ketiga model.

Regularisasi L1 menunjukkan kesenjangan terkecil antara loss pelatihan dan validasi, mengindikasikan kemampuan generalisasi yang lebih baik. Kesenjangan untuk model tanpa regularisasi adalah 0,0102 (0,4832 - 0,4730), untuk regularisasi L1 adalah 0,0037 (0,4306 - 0,4269), dan untuk regularisasi L2 adalah 0,0097 (0,4556 - 0,4459). Nilai kesenjangan yang lebih kecil pada L1 menunjukkan bahwa model ini memiliki kemampuan yang lebih baik untuk menggeneralisasi data yang belum pernah dilihat sebelumnya.



Distribusi bobot dan gradient no regularisasi



Distribusi bobot dan gradient regularisasi L1

Distribusi bobot dan gradient regularisasi L2

Distribusi Bobot

Pada Layer 1, model tanpa regularisasi menunjukkan distribusi normal berpusat mendekati nol (mean $\approx 0,0020$), dengan standar deviasi sekitar 0,0513. Regularisasi L1 menunjukkan distribusi serupa dengan tanpa regularisasi, tetapi tampak sedikit lebih terkonsentrasi di sekitar nol, sesuai dengan efek sparsitas L1. Regularisasi L2 menampilkan distribusi paling halus dengan mean $\approx 0,0023$ dan standar deviasi mirip dengan yang lain.

Untuk Layer 2, model tanpa regularisasi memiliki distribusi lebih lebar dengan standar deviasi sekitar 0,1844. Regularisasi L1 menunjukkan distribusi sedikit lebih terkonsentrasi dengan standar deviasi sekitar 0,1873. Regularisasi L2 memiliki distribusi paling terkonsentrasi dengan standar deviasi sekitar 0,1815.

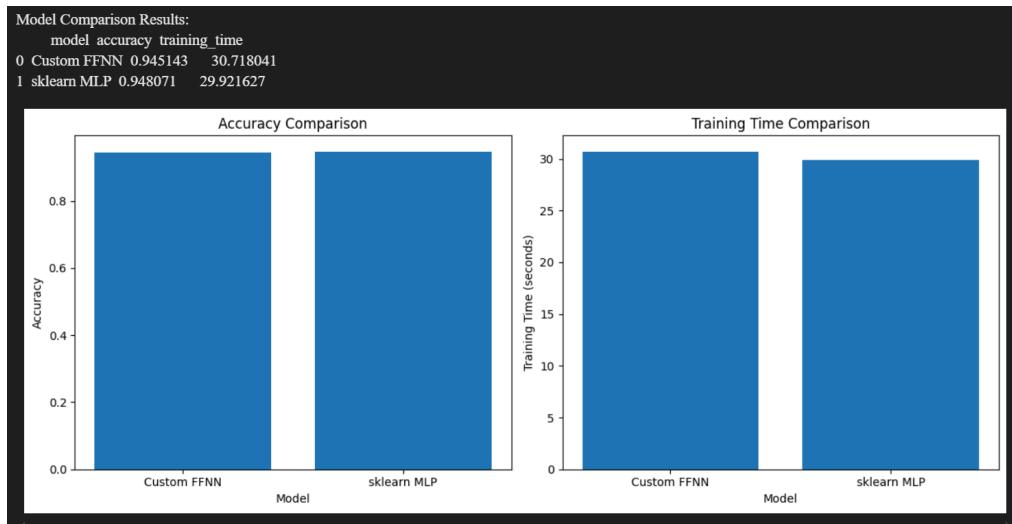
Pada Layer 3, model tanpa regularisasi memiliki distribusi lebar dengan standar deviasi sekitar 0,2901. Regularisasi L1 menunjukkan lebih banyak bobot mendekati nol, yang merupakan efek dari sparsitas. Regularisasi L2 menampilkan distribusi lebih menyebar dengan beberapa konsentrasi di dekat nol dan standar deviasi sekitar 0,2796.

Pengamatan utama adalah bahwa regularisasi L1 mendorong lebih banyak bobot menuju tepat nol (mendorong sparsitas), sementara regularisasi L2 umumnya mengurangi besaran semua bobot tetapi mempertahankan lebih banyak nilai bukan nol. Hal ini konsisten dengan sifat matematis dari kedua jenis regularisasi tersebut.

Distribusi Gradien bobot

Secara umum, semua model menunjukkan distribusi gradien yang berpusat di sekitar nol. Regularisasi L1 menunjukkan lebih banyak varians dalam gradien, terutama pada layer yang lebih dalam. Regularisasi L2 menunjukkan gradien yang lebih terkonsentrasi di sekitar nol. Analisis per layer menunjukkan bahwa gradien Layer 1 serupa di semua model dan sangat terkonsentrasi di sekitar nol. Gradien Layer 2 pada model dengan regularisasi L1 menunjukkan penyebaran sedikit lebih lebar dibandingkan model lainnya. Perbedaan paling menarik terlihat pada gradien Layer 3, di mana regularisasi L1 menghasilkan gradien yang lebih bervariasi, kemungkinan karena term penalti yang tidak mulus (non-smooth) pada regularisasi L1.

2.2.6 Analisis perbandingan hasil prediksi dengan library sklearn MLP



Pada eksperimen ini digunakan hyperparameter yang sama antara Feed-Forward Neural Network (FFNN) dan model Multilayer Perceptron (MLP). hal ini bertujuan untuk menguji ketepatan implementasi model FFNN yang telah kami buat. Khususnya pada inisialisasi bobot tidak dapat menentukan strateginya secara eksplisit menggunakan hyperparameter. namun, menurut sumber dari Dokumentasi scikit-learn, MLPClassifier menggunakan metode inisialisasi yang menghitung batas berdasarkan jumlah neuron input dan output (`fan_in` dan `fan_out`) dengan rumus $\text{sqrt}(\text{factor} / (\text{fan_in} + \text{fan_out}))$, di mana factor adalah 6 untuk sebagian besar fungsi aktivasi. Ini identik dengan metode Xavier/Glorot initialization. Dimana pada model FFNN kami juga mengimplementasikan metode inisialisasi bobot Xavier initialization.

Dalam perbandingan antara model FFNN from scratch dan MLP dari scikit-learn, hasil eksperimen mengungkapkan perbedaan yang tidak begitu signifikan. Model FFNN berhasil mencapai akurasi yang cukup baik , yaitu 94.51%, begitu juga dengan model scikit-learn yang mendapatkan akurasi yang sedikit lebih baik, yaitu 94.81%. Perbedaan hasil yang tidak signifikan ini menunjukkan keberhasilan implementasi model FFNN dalam mengekstrak dan mempelajari pola-pola kompleks dari dataset.

Menariknya, pencapaian akurasi tinggi ini tidak datang tanpa konsekuensi. Model FFNN membutuhkan waktu pelatihan sedikit lebih lama, yakni 30.69 detik, dibandingkan dengan model scikit-learn yang memakan waktu 30.21 detik. Dengan perbedaan waktu hanya sekitar 0.5 detik dan mendapatkan hasil prediksi dengan akurasi yang hampir sama menunjukkan keberhasilan implementasi Model FFNN from scratch yang kami buat.

Keberhasilan model custom ini dapat dikaitkan dengan beberapa faktor kunci, pengimplementasian algoritma FFNN dengan benar secara matematis, termasuk forward propagation, backward propagation, dan weight update. Hasil eksperimen ini memberikan bukti konkret tentang pentingnya desain arsitektur neural network yang cermat dan implementasi yang mempertimbangkan setiap detail algoritmik.

BAB 3

KESIMPULAN DAN SARAN

3.1 Kesimpulan

Berdasarkan hasil implementasi dan pengujian yang telah dilakukan, diperoleh beberapa kesimpulan sebagai berikut:

1. Performa Model

- Model FFNN yang diimplementasikan mampu menunjukkan kinerja yang baik dalam melakukan klasifikasi data. Dengan konfigurasi yang optimal, model ini berhasil mencapai akurasi hingga 94.51%. nilai tersebut sangat dekat dengan hasil akurasi prediksi model Multilayer Perceptron (MLP) dari library Scikit-learn.

2. Pengaruh Parameter

- Peningkatan jumlah lapisan (depth) memberikan dampak signifikan terhadap peningkatan akurasi, sementara peningkatan jumlah neuron pada setiap layer (width) memberikan dampak yang lebih minimal.
- Pemilihan fungsi aktivasi berpengaruh besar terhadap performa model. Fungsi aktivasi Linear dan ReLU menunjukkan kinerja terbaik, sedangkan fungsi aktivasi Sigmoid memiliki performa yang kurang optimal akibat masalah vanishing gradient.
- Learning rate yang optimal berperan penting dalam mempercepat konvergensi model tanpa mengorbankan stabilitas. Learning rate sebesar 0.1 terbukti memberikan hasil terbaik dalam eksperimen ini.
- Metode inisialisasi bobot yang paling efektif adalah metode He Initialization yang menghasilkan akurasi tinggi dan konvergensi yang stabil.

3. Proses Implementasi

- Implementasi dari awal (from scratch) memberikan pemahaman yang lebih mendalam tentang bagaimana forward propagation, backward propagation, dan pembaruan bobot bekerja secara internal pada jaringan

saraf tiruan. Hal ini memperkaya pemahaman konsep fundamental dalam pembelajaran mesin.

3.2 Saran

Berdasarkan hasil yang diperoleh, terdapat beberapa saran yang dapat diterapkan untuk pengembangan dan peningkatan model di masa mendatang:

1. Penyesuaian Parameter

- Disarankan untuk melakukan eksperimen lebih lanjut pada konfigurasi hyperparameter yang lebih luas, termasuk variasi jumlah epoch, batch size, dan metode optimasi lainnya untuk mencapai hasil yang lebih optimal.

2. Penanganan Overfitting

- Untuk mengurangi potensi overfitting pada model yang lebih kompleks, dapat ditambahkan mekanisme seperti dropout layer atau early stopping pada proses pelatihan.

3. Eksperimen dengan Arsitektur Lain

- Selain FFNN, disarankan untuk mengeksplorasi model yang lebih kompleks seperti Convolutional Neural Network (CNN) atau Recurrent Neural Network (RNN) untuk menangani data dengan pola yang lebih rumit.

4. Pemanfaatan Data Augmentation

- Untuk meningkatkan kemampuan generalisasi model, terutama pada dataset terbatas, disarankan untuk menerapkan teknik data augmentation.

5. Optimasi Kinerja

- Mengingat waktu pelatihan yang meningkat seiring dengan kompleksitas model, disarankan untuk mengeksplorasi penggunaan teknik komputasi paralel atau GPU acceleration untuk meningkatkan efisiensi pelatihan model.

Dengan menerapkan saran-saran tersebut, diharapkan model FFNN yang dikembangkan dapat mencapai performa yang lebih optimal dan lebih siap digunakan dalam kasus nyata yang lebih beragam.

PEMBAGIAN TUGAS

NIM	KERJA
13522142	Experiment 5 , Bonus 1 dan 2, laporan
13522146	Experiment 3 dan 4, laporan
13522159	Membuat implementasi Class FFNN, Experiment 1 & 2, laporan

REFERENSI

- Institut Teknologi Bandung. (2024). *Spesifikasi Tugas Besar 1 IF3270 Pembelajaran Mesin*. Bandung: ITB.
- Osajima, J. (n.d.). *The spelled-out intro to neural networks and backpropagation: building micrograd*. Diakses dari <https://www.jasonosajima.com/forwardprop>
- Osajima, J. (n.d.). *Backpropagation explanation*. Diakses dari <https://www.jasonosajima.com/backprop>
- NumPy Developers. (2024). *NumPy Documentation*. Diakses dari <https://numpy.org/doc/2.2/>
- Scikit-learn Developers. (2024). *MLPClassifier Documentation*. Diakses dari https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
- LibreTexts. (n.d.). *The Chain Rule for Multivariable Functions*. Diakses 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](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)