LAPORAN TUGAS BESAR 1

IF3270 PEMBELAJARAN MESIN

FEEDFORWARD NEURAL NETWORK

Kelompok 16
Anggota Kelompok :

Muhammad Zakkiy	10122074
Ghaisan Zaki Pratama	10122078

Semester II Tahun 2024/2025

INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2025

Daftar Isi

A. Deskripsi Persoalan

Dalam tugas besar ini, permasalahan yang dibahas ialah bagaimana cara mengimplementasikan Feedforward Neural Network (FFNN) *from scratch.* Tugas ini mengaplikasikan konsep-konsep fundamental jaringan saraf, seperti:

- Forward Propagation

  Proses perhitungan output dari input melalui serangkaian lapisan jaringan dengan fungsi aktivasi yang beragam.
- Backward Propagation

  Proses perhitungan gradien untuk setiap bobot dengan chain rule agar dapat menentukan bagaimana bobot harus diperbarui.
- Gradient Descent

  Metode optimasi untuk memperbarui bobot jaringan berdasarkan gradien yang dihitung, dengan tujuan meminimalkan loss function.

Permasalahan ini mengharuskan implementasi berbagai komponen penting seperti berbagai fungsi aktivasi (Linear, ReLU, Sigmoid, tanh, dan Softmax), berbagai fungsi loss (Mean Squared Error, Binary Cross-Entropy, dan Categorical Cross-Entropy), serta metode inisialisasi bobot (zero, random uniform, dan random normal). Perlu dilakukan analisis eksperimental terhadap pengaruh hyperparameter seperti depth (banyak layer) dan width (banyak neuron per layer), berbagai fungsi aktivasi, variasi learning rate, dan metode inisialisasi terhadap performa model. Selain itu, analisis perbandingan hasil prediksi dengan library sklearn MLP.

FFNN yang diimplementasikan dapat menerima jumlah neuron dari tiap layer (termasuk input layer dan output layer). FFNN yang diimplementasikan dapat menerima fungsi aktivasi dari tiap layer. FFNN yang diimplementasikan dapat menerima fungsi loss dari model tersebut. Inisialisasi bobot tiap neuron (termasuk bias) harus dapat diimplementasikan dengan berbagai metode.

Instance model yang diinisialisasikan harus bisa menyimpan bobot tiap neuron (termasuk bias). Instance model yang diinisialisasikan harus bisa menyimpan gradien bobot tiap neuron (termasuk bias). Instance model memiliki method untuk menampilkan model berupa struktur jaringan beserta bobot dan gradien bobot tiap neuron dalam bentuk graf. Instance model memiliki method untuk menampilkan distribusi bobot dari tiap layer.

Instance model memiliki method untuk menampilkan distribusi gradien bobot dari tiap layer. Instance model memiliki method untuk save dan load.

Model dapat menerima input berupa batch. Model dapat menangani perhitungan perubahan gradien untuk input data batch. Model dapat menggunakan chain rule untuk menghitung gradien tiap bobot terhadap loss function. Pelatihan model harus dapat menerima parameter batch size, learning rate, jumlah epoch, dan verbose (verbose 0 berarti tidak menampilkan apa-apa selama pelatihan dan verbose 1 berarti hanya menampilkan progress bar beserta dengan kondisi training loss dan validation loss saat itu. Proses pelatihan mengembalikan history dari proses pelatihan yang berisi training loss dan validation loss tiap epoch.

Berikut merupakan berbagai fungsi dan persamaan matematis yang digunakan.

- Pilihan fungsi aktivasi yang harus diimplementasikan adalah sebagai berikut:

Nama Fungsi Aktivasi	Definisi Fungsi
Linear	$Linear(x) = x$
ReLU	$ReLU(x) = \max(0, x)$
Sigmoid	$\sigma(x) = \frac{1}{1 + e^{-x}}$
Hyperbolic Tangent (tanh)	$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$
Softmax	Untuk vector $\vec{x} = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$, $softmax(\vec{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$

- Pilihan loss function yang harus diimplementasikan adalah sebagai berikut:

Nama Fungsi	Definisi Fungsi

Loss	
MSE	$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$
Binary Cross-Entropy	$$\mathcal{L}_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i))$$ y_i = Actual binary label (0 or 1) $\hat{y}_i$ = Predicted value of y_i n = Batch size
Categorical Cross-Entropy	$$\mathcal{L}_{CCE} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{C}(y_{ij} \log \hat{y}_{ij})$$ y_{ij} = Actual value of instance i for class j $\hat{y}_{ij}$ = Predicted value of y_{ij} C = Number of classes n = Batch size

- Berikut merupakan **turunan pertama** untuk setiap fungsi aktivasi:

Nama Fungsi Loss	Definisi Fungsi
MSE	$$\frac{\partial \mathcal{L}_{MSE}}{\partial W} = -\frac{2}{n}\sum_{i=1}^{n}\frac{\partial \mathcal{L}_{MSE}}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial W} = -\frac{2}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)\frac{\partial \hat{y}_i}{\partial W}$$
Binary Cross-Entropy	$$\frac{\partial \mathcal{L}_{BCE}}{\partial W} = -\frac{1}{n}\sum_{i=1}^{n}\frac{\partial \mathcal{L}_{BCE}}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial W} = -\frac{1}{n}\sum_{i=1}^{n}\frac{\hat{y}_i - y_i}{\hat{y}_i(1 - \hat{y}_i)}\frac{\partial \hat{y}_i}{\partial W}$$

Categorical Cross-Entropy	$\frac{\partial \mathcal{L}_{CCE}}{\partial W} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{C}\frac{\partial \mathcal{L}_{CCE}}{\partial \hat{y}_{ij}}\frac{\partial \hat{y}_{ij}}{\partial W} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{C}\frac{y_{ij}}{\hat{y}_{ij}}\frac{\partial \hat{y}_{ij}}{\partial W}$

- Model memiliki implementasi **weight update** dengan menggunakan **gradient descent** untuk memperbarui bobot berdasarkan gradien yang telah dihitung, berikut persamaannya:

$$W_{new} = W_{old} - \alpha\left(\frac{\partial \mathcal{L}}{\partial W_{old}}\right)$$

α = Learning rate

B. Pembahasan

1. Penjelasan Implementasi

Implementasi neural network dibuat dengan menggunakan paradigma pemrograman berorientasi objek (OOP). Kelas utama bernama `NeuralNetwork`, yang bertugas menginisialisasi parameter jaringan (bobot, bias), menyimpan fungsi aktivasi beserta turunannya, serta mengimplementasikan algoritma forward dan backward propagation untuk melatih model. Selain itu, kelas ini juga menyediakan method untuk memvisualisasikan performa training (plot loss), struktur jaringan (display graph), plot distribusi bobot, dan plot distribusi gradien bobot..

a. Deskripsi kelas beserta deskripsi atribut dan methodnya

Kelas `NeuralNetwork` merupakan implementasi dari *Artificial Neural Network* yaitu *Feedforwad Neural Network* yang dibangun *from scratch* menggunakan bahasa Python. Beberapa atribut dari kelas ini ialah:

- `self.weights`

  List yang berisi matriks bobot untuk setiap koneksi antar layer. Bentuknya ialah matriks dengan dimensi (n_i,n_{i+1}) untuk koneksi dari layer i ke i+1, dengan n_i adalah banyak neuron di layer i dan n_{i+1} adalah banyak neuron di layer i + 1.

- `self.biases`

List yang berisi vektor bias untuk setiap koneksi antar layer. Bentuknya ialah array dengan dimensi $(1,n_{i+1})$ dengan n_{i+1} adalah banyak neuron di layer $i + 1$.

- `self.weight_gradients`

  List yang berisi matriks gradien bobot untuk setiap koneksi antar layer. Bentuknya ialah matriks dengan dimensi (n_i,n_{i+1}) untuk koneksi dari layer i ke i+1, dengan n_i adalah banyak neuron di layer i dan n_{i+1} adalah banyak neuron di layer $i + 1$.

- `self.activations` dan `self.activation_derivatives`

  List yang menyimpan fungsi aktivasi dan turunannya untuk tiap koneksi antar layer. Diinisiasi menggunakan `getattr(self, act)` sehingga berdasarkan nama fungsi aktivasinya, secara dinamis mengambil method dalam kelas. Jumlah elemennya sama dengan jumlah hidden layer + 1.

- `self.loss_function` dan `self.loss_derivative`

  List yang menyimpan fungsi loss dan turunannya untuk mengukur error pada model. Diambil menggunakan `getattr(self, loss_function)` dan `getattr(self, loss_function + "_derivative")`.

- `self.history`

  Dictionary untuk menyimpan riwayat nilai training loss dan validation loss. Akan digunakan untuk plotting loss dan evaluasi performa training.

Beberapa method dari kelas ini ialah:

- `__init__(...)`

  Menginisialisasi model dengan parameter, fungsi aktivasi, fungsi loss, dan metode inisialisasi bobot. Membangun list layer sizes dengan menggabungkan `input_size`, `hidden_layers`, dan `output_size`. Inisialisasi fungsi aktivasi dan turunannya menggunakan getattr sehingga model fleksibel dalam mengganti fungsi aktivasi. Inisialisasi bobot dan bias untuk tiap koneksi antar layer menggunakan method `initialize_weights`. Inisialisasi `self.weight_gradients` sebagai list array nol yang memiliki bentuk sama dengan tiap bobot. Menyiapkan `self.history` untuk mencatat loss selama training.

- `forward(self, x)`

Melakukan forward propagation, yaitu menghitung output jaringan dari input `x` melalui tiap layer. Inisialisasi list activations dengan input awal. Untuk setiap koneksi antar layer, hitung $z = x.W + b$ kemudian terapkan fungsi aktivasi $a = f(z)$. Update `x` menjadi `a` dan simpan ke list `activations`. Mengembalikan list activations yang berisi output tiap layer.

- `backward(self, activations, y, learning_rate)`

  Menghitung gradien loss terhadap parameter dan mengupdate parameter menggunakan gradient descent. Hitung turunan loss terhadap output layer kemudian simpan di list deltas. Untuk setiap layer dari belakang ke depan, hitung $\delta^{(i)} = (\delta^{(i+1)}.W^{(i+1)T})f'^{(i)}.(z^{(i)})$ dan simpan ke list `deltas`. Untuk setiap koneksi, hitung gradien bobot $\frac{\partial L}{\partial W^{(i)}} = \frac{1}{N}a^{(i)T}\delta^{(i)}$ dengan N ialah jumlah sampel dalam batch, simpan ke `self.weight_gradients`. Lakukan update bobot dan bias.

- `train(self, X, y, batch_size, learning_rate, epochs, verbose, X_val, y_val)`

  Melatih model menggunakan mini-batch gradient descent. Acak data di setiap epoch. Bagi data menjadi batch, untuk setiap batch lakukan forward propagation untuk menghitung output, kemudian hitung loss (rata-rata per batch) dan akumulasi ke total loss, lalu panggil backward untuk menghitung gradien dan update parameter. Simpan rata-rata loss training per epoch ke `self.history["train_loss"]`. Jika data validasi tersedia, hitung dan simpan loss validasi. Jika verbose aktif, cetak status training setiap beberapa epoch.

- `predict(self, X)`

  Menghasilkan prediksi dari input X menggunakan forward propagation. Lakukan forward propagation dan kembalikan output dari layer terakhir.

- `plot_loss(self, title)`

  Memvisualisasikan grafik loss selama training. Plot nilai train loss dan validasi loss dari self.history.

- `initialize_weights(self, shape, method, **kwargs)`

Menghasilkan bobot yang diinisialisasi sesuai dengan metode yang dipilih. Zero menghasilkan array nol dengan bentuk shape. Random Uniform menghasilkan bobot dari distribusi uniform antara nilai lower dan upper (default nya [-1,1]). Random Normal menghasilkan bobot dari distribusi normal dengan mean dan variance tertentu (default nya mean 0 dan variance 1). He menghasilkan bobot dari distribusi normal dengan variance $\sqrt{\frac{2}{fan_{inn}}}$. Xavier menghasilkan bobot dari distribusi normal dengan variansi $\sqrt{\frac{1}{fan_{inn}}}$. Parameter tambahan kwargs digunakan untuk mengkustomisasi inisialisasi.

- `mse dan mse_derivative`

  Mean Squared Error menghitung rata-rata kuadrat perbedaan antara output dan target.

- `Binary_cross_entropy dan binary_cross_entropy_derivative`

  Loss ini digunakan untuk klasifikasi biner dan mengukur seberapa jauh probabilitas prediksi dan target. Turunannya ditambahkan epsilon untuk mencegah pembagian dengan nol.

- `categorical_cross_entropy dan categorical_cross_entropy_derivative`

  Loss ini digunakan untuk klasifikasi multi-kelas. Fungsi loss menghitung rata-rata loss per sampel, sedangkan turunannya digunakan dalam backpropagation.

- `linear dan linear_derivative`

  Fungsi aktivasi dengan fungsi linear yaitu input secara langsung.

- `relu dan relu_derivative`

  Fungsi aktivasi dengan fungsi relu yaitu nilai maksimum antara 0 dan input.

- `sigmoid dan sigmoid_derivative`

  Fungsi aktivasi dengan fungsi sigmoid yaitu $\frac{1}{1+\exp(-net)}$.

- `hyperbolic_tangent dan hyperbolic_tangent_derivative`

  Fungsi aktivasi dengan fungsi $tanh(net)$.

- `softmax dan softmax_derivative`

Mengubah output menjadi distribusi probabilitas, diimplementasikan secara stabil dengan mengurangkan nilai maksimum per sampel (menggunakan `axis=1, keepdims=True`). Menghitung matriks Jacobian untuk softmax.

- `plot_weight_distribution(self, layer)` dan `plot_gradient_distribution(self, layer)`

  Memvisualisasikan distribusi bobot dan gradien bobot untuk layer tertentu dengan parameter layers adalah list indeks layer yang ingin diplot histogramnya.
- `display_graph(self)`

  Menampilkan graf struktur jaringan beserta nilai bobot, gradien, dan bias.

b. Penjelasan forward propagation

Forward propagation dalam neural network adalah proses melewatkan input data melalui layer-layer jaringan untuk menghasilkan prediksi dari input data. Forward propagation pada algoritma ini dalam kode ini bertanggung jawab untuk melakukan propagasi maju (forward propagation) pada jaringan saraf tiruan. Algoritma yang digunakan pada tugas besar ini, menerima input berupa batch Proses forward propagation melakukan iterasi melalui setiap pasangan bobot dan bias dalam jaringan.

Pada setiap iterasi, nilai input dikalikan dengan bobot menggunakan operasi perkalian matriks, kemudian hasil dari perkalian tersebut ditambahkan dengan bias. Hasil ini kemudian dimasukkan ke dalam fungsi aktivasi yang menghasilkan nilai output teraktivasi dari layer tersebut.Proses ini berlanjut hingga semua lapisan jaringan telah diproses.

c. Penjelasan backward propagation dan weight update

Dalam machine learning, backpropagation adalah metode estimasi gradien yang umum digunakan untuk melatih jaringan saraf guna menghitung pembaruan parameter dari model machine learning. Backpropagation adalah penerapan aturan rantai yang efisien pada jaringan saraf. Algoritma backpropagation bertujuan untuk memperbarui bobot dan bias jaringan saraf agar meminimalkan fungsi kerugian selama proses pelatihan. Proses dimulai dengan perhitungan error term pada lapisan output. Error term ini menunjukkan seberapa besar kesalahan prediksi terhadap nilai target.

Selanjutnya, error term dihitung untuk setiap lapisan tersembunyi dengan menggunakan aturan rantai dari backpropagation. Proses ini dilakukan dengan cara mengalikan error term dari lapisan berikutnya dengan bobot yang terhubung ke lapisan saat ini. Lalu, hasil yang diperoleh dikalikan dengan turunan fungsi aktivasi pada lapisan tersebut. Proses ini berjalan mundur dari lapisan output ke lapisan tersembunyi pertama agar setiap lapisan dapat menyesuaikan bobotnya sesuai dengan kontribusi kesalahannya terhadap prediksi akhir.

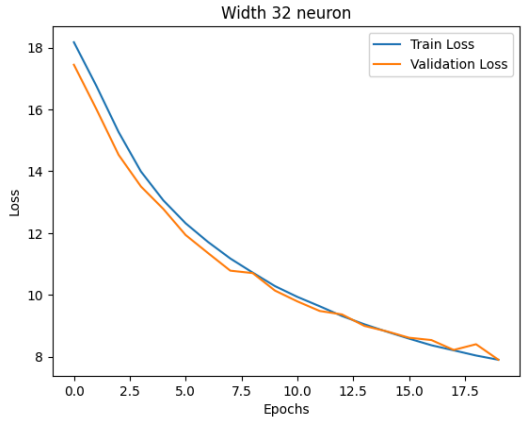
Setelah semua error term dihitung, langkah selanjutnya adalah memperbarui bobot dan bias menggunakan aturan penurunan gradien. Bobot diperbarui dengan mengurangkan nilai gradien bobot yang diperoleh dari hasil perkalian antara aktivasi lapisan sebelumnya dengan error term yang sudah dihitung. Nilai gradien bobot dinormalisasi berdasarkan ukuran batch. Bias diperbarui dengan cara mengurangkan nilai rata-rata dari error term pada lapisan tersebut. Pembaruan ini dilakukan untuk setiap lapisan jaringan, sehingga model secara bertahap menyesuaikan parameter-parameternya agar menghasilkan prediksi yang lebih akurat. Proses ini diulangi selama jumlah epoch yang telah ditentukan hingga model mencapai konvergensi atau hasil yang diinginkan.

2. Hasil Pengujian

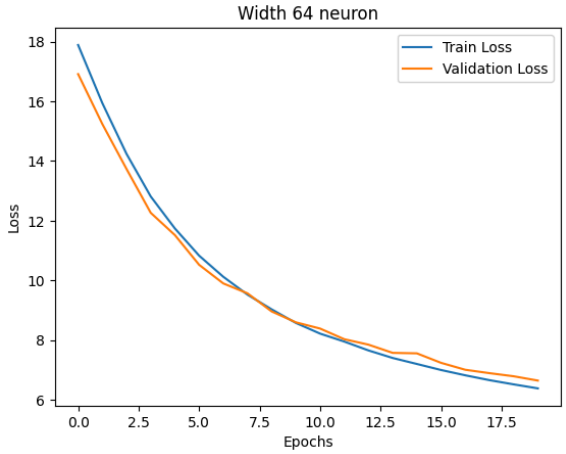
Pengujian dilakukan dengan menggunakan dataset berikut : mnist_784

a. Pengaruh depth dan width

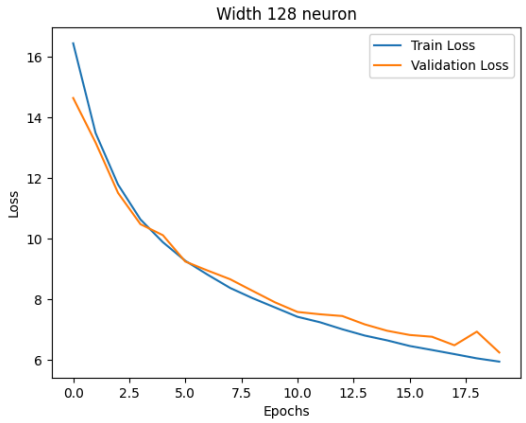
Variasi width diterapkan dengan menetapkan depth sebanyak 2 hidden layer dan width sebanyak 32, 64, dan 128 neuron. Sedangkan variasi depth diterapkan dengan menetapkan width sebanyak 64 neuron dan depth sebanyak 1, 3, dan 5 hidden layer Berbagai hasil plot loss dapat dilihat pada gambar di bawah



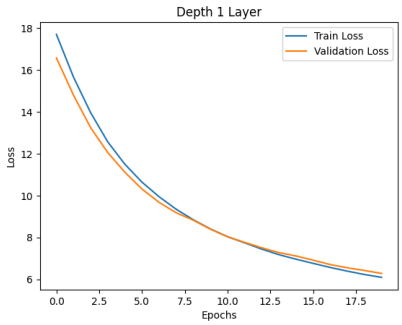
Plot Loss Width 32 Neuron



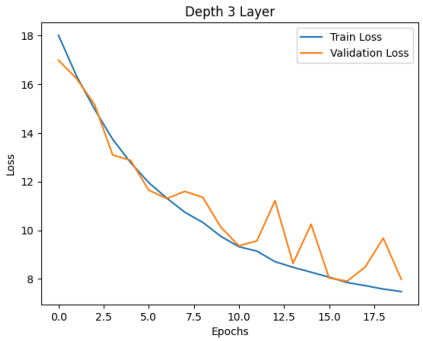
Plot Loss Width 64 Neuron



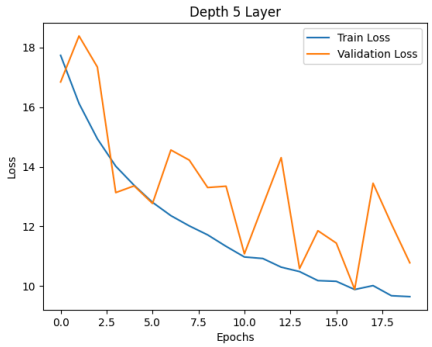
Plot Loss Width 128 Neuron



Plot Loss Depth 1 Hidden Layer



Plot Loss Depth 3 Hidden Layer



Plot Loss Depth 5 Hidden Layer

Selain plot loss, model dari berbagai variasi tersebut diterapkan untuk memprediksi data dengan hasil akurasi sebagai berikut:

Depth dan Width	Akurasi
Width 32 neuron	0.7473
Width 64 neuron	0.8

Width 128 neuron	0.8079
Depth 1 layer	0.8278
Depth 3 layer	0.7124
Depth 5 layer	0.5418

Dari hasil pengujian tersebut, dapat disimpulkan bahwa untuk perubahan width dari plot loss tidak tampak perbedaan signifikan sedangkan untuk perubahan depth terdapat perbedaan signifikan. Selain itu, dari hasil akurasi prediksi didapat bahwa semakin banyak neuron yang digunakan (width semakin besar) maka akurasinya semakin baik dan semakin sedikit layer yang digunakan (depth semakin kecil) maka akurasinya semakin baik.

b. Pengaruh fungsi aktivasi

Pengaruh fungsi aktivasi akan diuji dengan menggunakan model yang memiliki satu hidden layer. Fungsi aktivasi hidden layer tersebut adalah fungsi aktivasi yang akan diuji. Fungsi Softmax sebagai fungsi aktivasi output layer dan fungsi loss yang digunakan adalah Categorical Cross-Entropy. Kemudian, parameter pelatihan yang digunakan adalah sebagai berikut.

Batch size : 8

Learning rate : 0.01

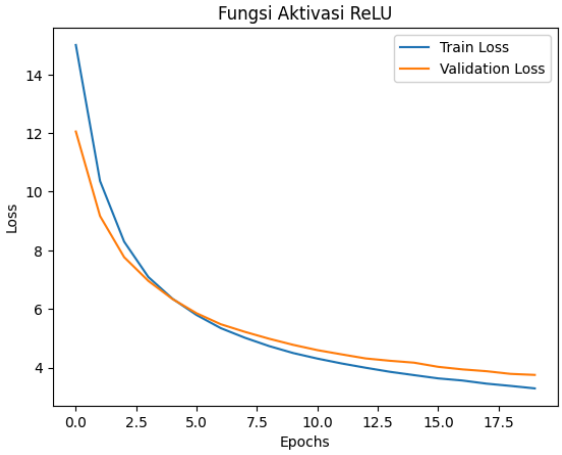
Epoch : 20

Verbose : 1

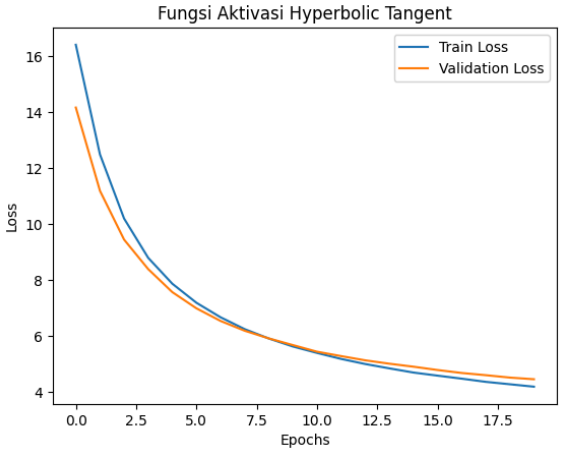
Berikut merupakan hasil yang didapatkan.

Fungsi Aktivasi	**Validation Loss Akhir Pelatihan**
ReLU	3.747689388086874
Hyperbolic Tangent(tanh)	4.457603233791455
Sigmoid	5.2430470859555225
Linear	3.879180060995114

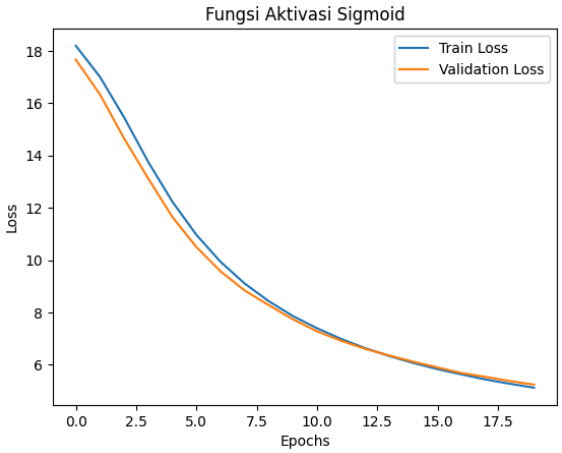
Berikut merupakan fungsi loss dari masing masing fungsi aktivasi.



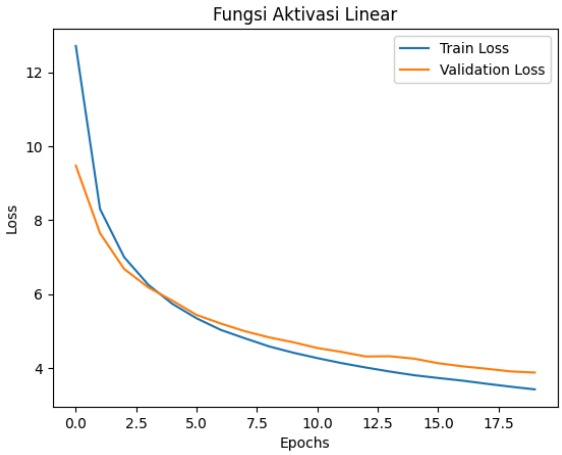
Plot Grafik Loss ReLU



Plot Grafik Loss Hyperbolic Tangent



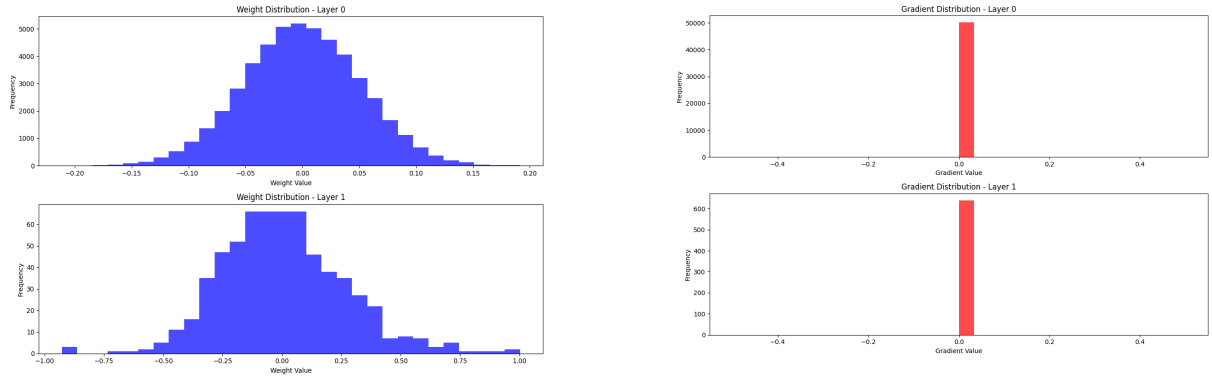
Plot Grafik Loss Sigmoid



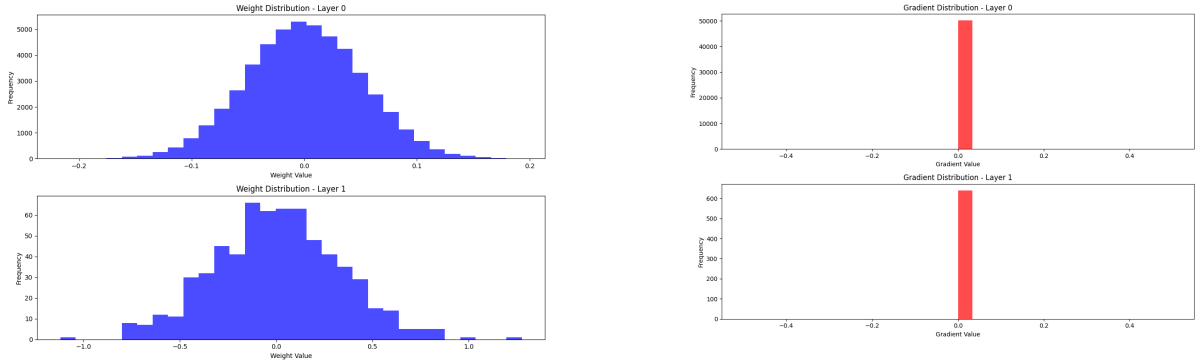
Plot Grafik Loss Linear

Lalu, berikut merupakan distribusi bobot dan gradien bobot dari beberapa/semua layer pada model.

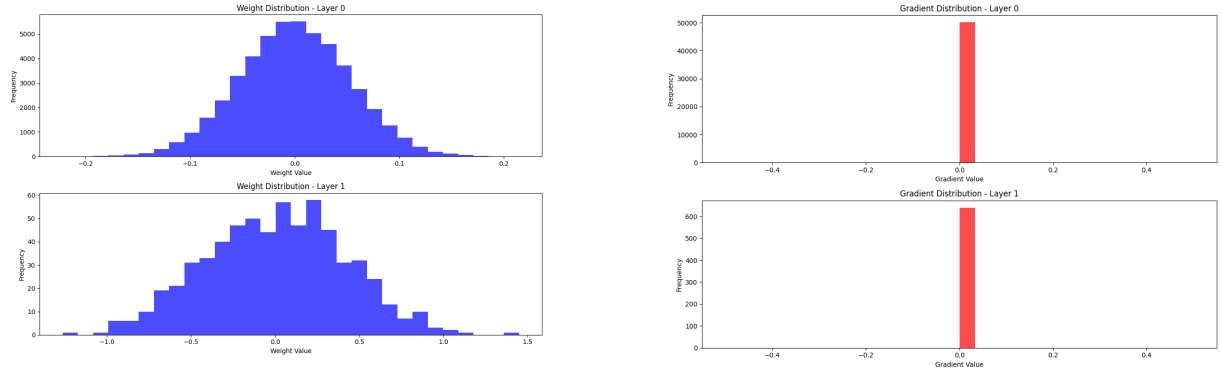
Distribusi Bobot dan Gradien Bobot Fungsi Aktivasi ReLU



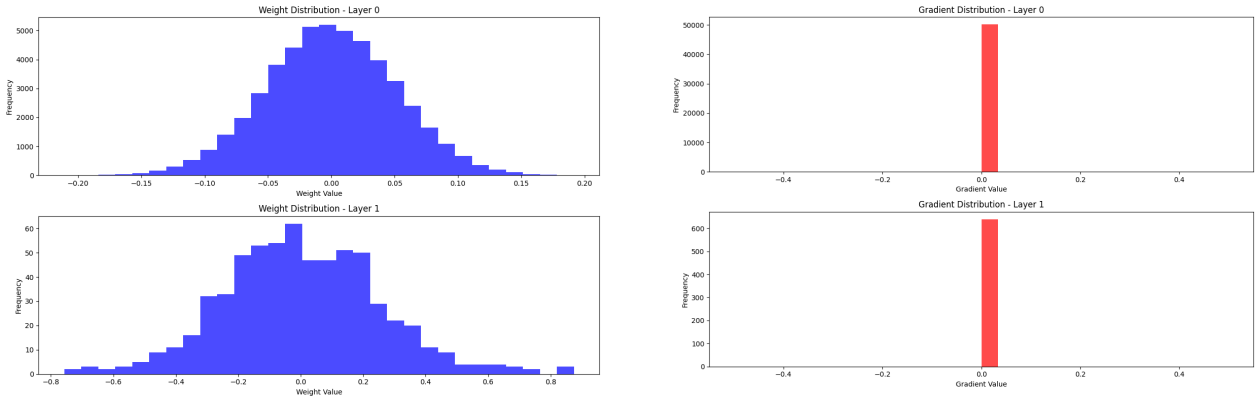
Distribusi Bobot dan Gradien Bobot Fungsi Aktivasi Tangent



Distribusi Bobot dan Gradien Bobot Fungsi Aktivasi Sigmoid



Distribusi Bobot dan Gradien Bobot Fungsi Aktivasi Linear



Terlihat bahwa fungsi aktivasi ReLU dan fungsi aktivasi Linear memiliki nilai validation loss yang kecil jika dibandingkan dengan fungsi aktivasi Sigmoid dan Hyperbolic Tangent. Selain itu, dari grafik fungsi loss, fungsi aktivasi ReLU dan fungsi aktivasi Linear mencapai kekonvergenan lebih cepat jika dibandingkan dengan fungsi Sigmoid dan Hyperbolic Tangent. Dengan demikian, untuk dataset mnsit_78, fungsi aktivasi ReLU dan Linear lebih efektif daripada Sigmoid dan Hyperbolic Tangent. Dengan demikian, pemilihan fungsi aktivasi baik dalam model sangat penting. Pemilihan fungsi aktivasi yang baik dapat memberikan kinerja yang baik pada model yang dibuat. Lalu, tidak ada perbedaan signifikan dari distribusi bobot dan gradien bobot dari semua layer pada model.

c. Pengaruh learning rate

Pengaruh fungsi learning rate akan diuji dengan menggunakan model yang memiliki satu hidden layer. Fungsi aktivasi hidden layer tersebut adalah fungsi aktivasi ReLU. Lalu, Fungsi Softmax sebagai fungsi aktivasi output layer dan fungsi loss yang digunakan adalah Categorical Cross-Entropy. Kemudian, parameter pelatihan yang digunakan adalah sebagai berikut.

Batch size : 8

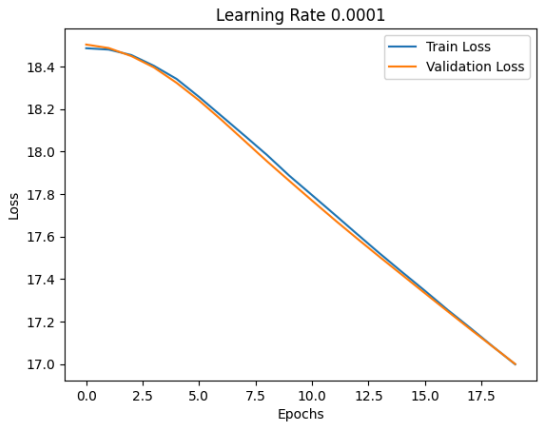
Epoch : 20

Verbose : 1

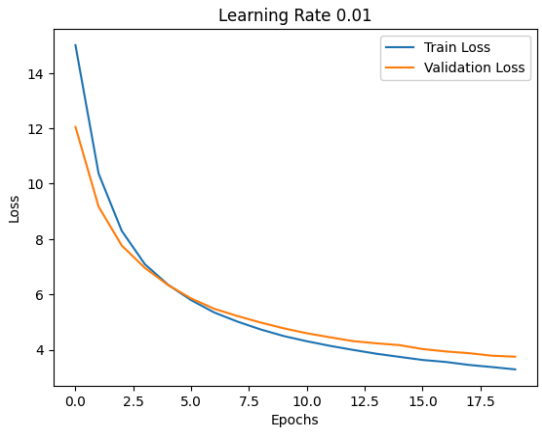
Nilai learning rate yang diuji adalah 0.0001, 0.01, 1. Berikut merupakan hasil yang didapatkan.

Learning Rate	Validation Loss Akhir Pelatihan
0.0001	17.000639716227017
0.01	3.747689388086874
1	4.538941418861973

Kemudian, berikut merupakan grafik loss pelatihannya.



Plot Grafik Loss Learning Rate 0.0001

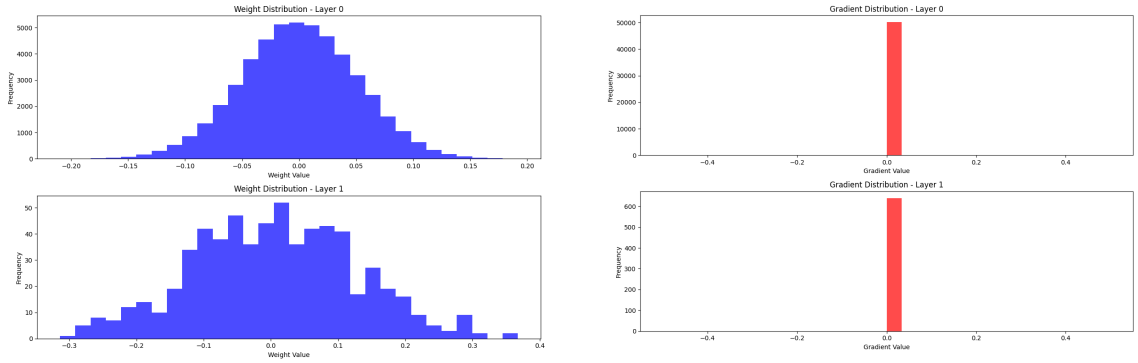


Plot Grafik Loss Learning Rate 0.01

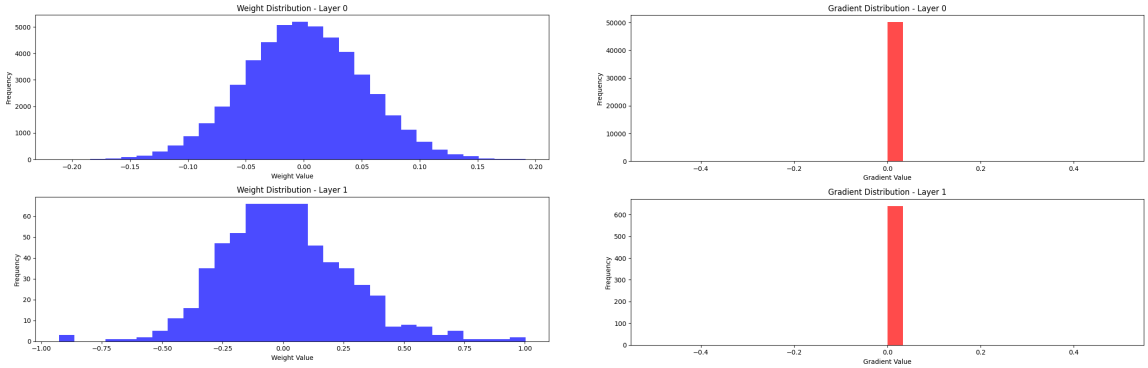
Plot Grafik Loss Learning Rate 1

Kemudian, berikut merupakan distribusi bobot dan gradien bobot dari semua layer pada model.

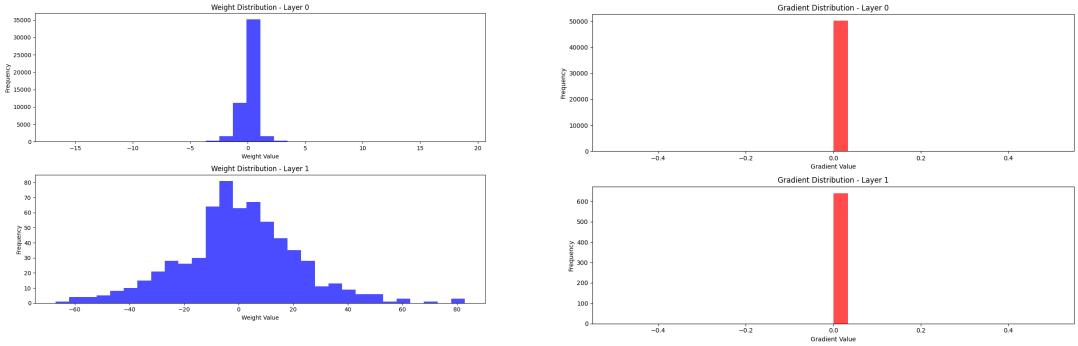
Distribusi Bobot dan Gradien Bobot Learning Rate 0.0001



Distribusi Bobot dan Gradien Bobot Learning Rate 0.01



Distribusi Bobot dan Gradien Bobot Learning Rate 1

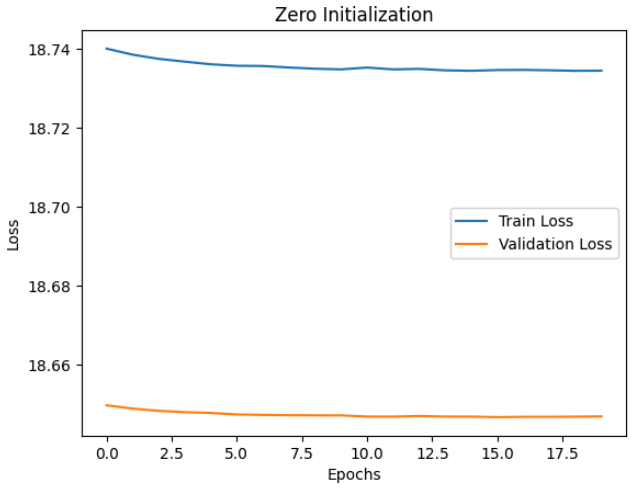


Dari hasil yang didapatkan, model yang memiliki learning rate 0.0001 memiliki nilai validation loss paling tinggi dibandingkan dengan model lainnya. Hal ini dikarenakan, learning rate menentukan seberapa cepat suatu bobot berubah dalam satu pembelajaran. Karena model memiliki learning rate sangat kecil, maka pembelajarannya akan sangat lambat.

Kemudian, model yang memiliki learning rate 1 memiliki nilai validation loss yang cukup kecil. Namun, grafik loss pada model ini tidak stabil. Hal ini terjadi karena, model memiliki learning rate yang besar. Nilai learning rate yang besar mengakibatkan pembaruan bobot pada saat pelatihan menjadi besar. Akibatnya, nilai dari bobot model menjadi tidak stabil. Hal tersebut mengakibatkan model memiliki grafik fungsi loss yang tidak stabil. Lalu, tidak ada perbedaan signifikan dari distribusi bobot dan gradien bobot dari semua layer pada model.

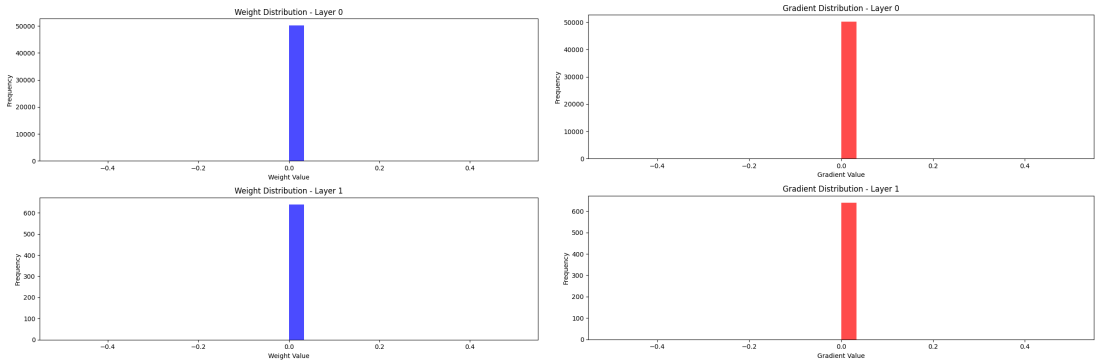
d. Pengaruh inisialisasi bobot

Variasi inisialisasi bobot diterapkan dengan menggunakan Zero, Random Uniform default, Random Uniform custom, Random Normal default, Random Normal custom, He, dan Xavier. Berikut merupakan plot loss, distribusi bobot, dan distribusi gradien bobot untuk setiap variasi.

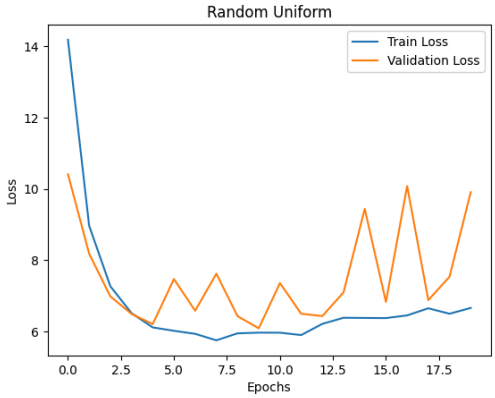


Plot Loss Zero Initialization

Plot Weight Distribution Loss Zero Initialization

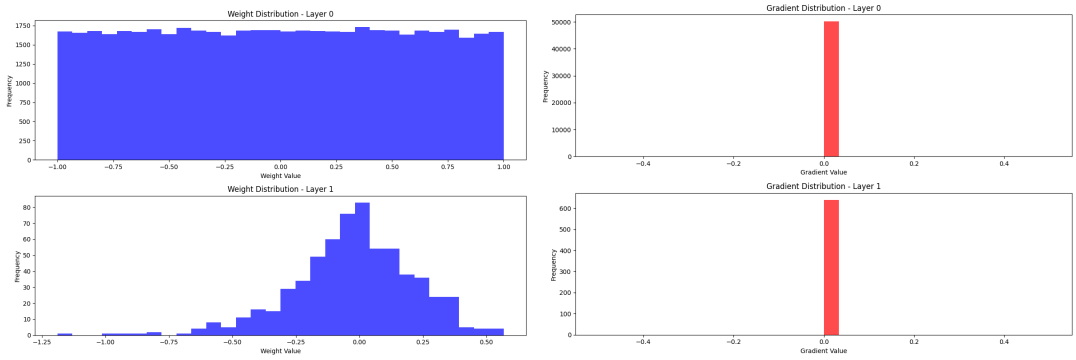


Plot Gradient Distribution Loss Zero Initialization

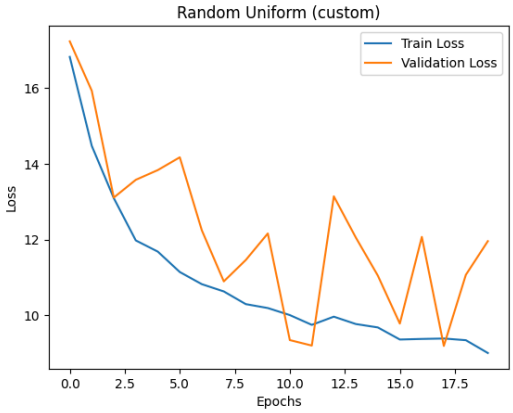


Plot Loss Random Uniform Initialization (default)

Plot Weight Distribution Loss Random Uniform Initialization (default)

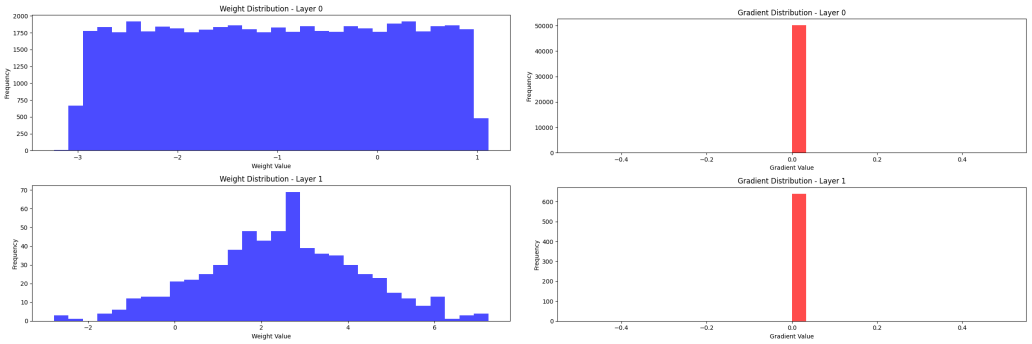


Plot Gradient Distribution Loss Random Uniform Initialization (default)

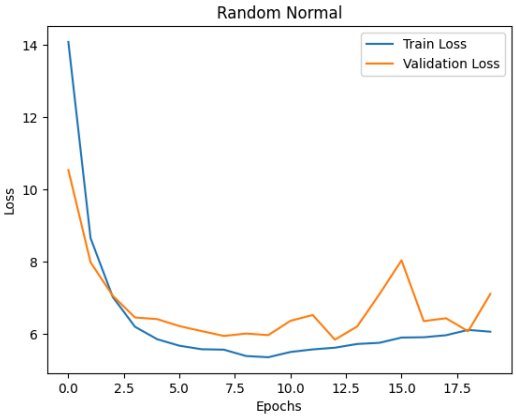


Plot Loss Random Uniform Initialization (custom)

Plot Weight Distribution Loss Random Uniform Initialization (custom)

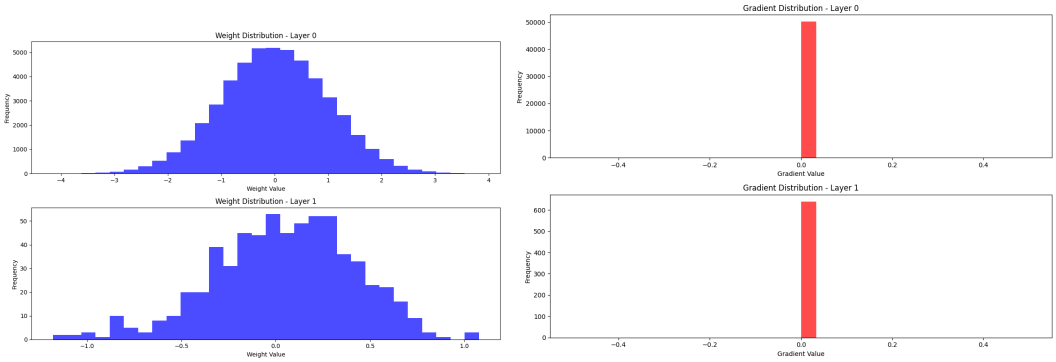


Plot Gradient Distribution Loss Random Uniform Initialization (custom)

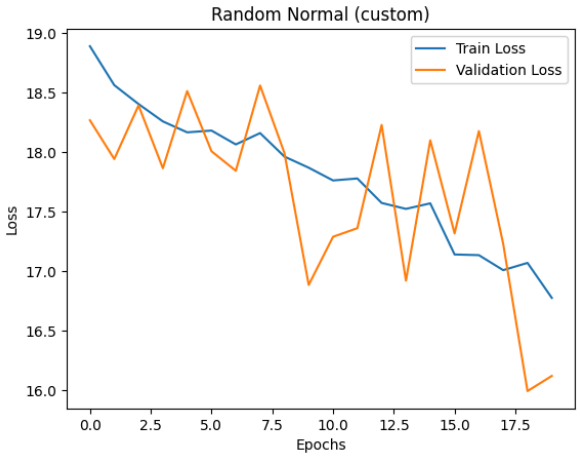


Plot Loss Random Normal Initialization (default)

Plot Weight Distribution Loss Random Normal Initialization (default)

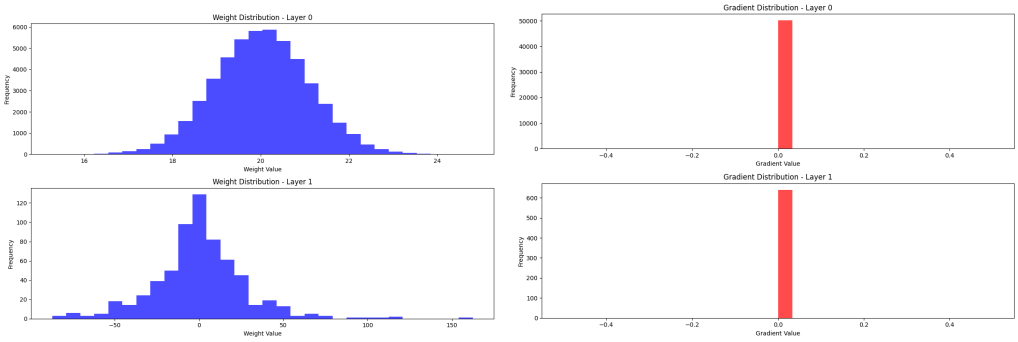


Plot Gradient Distribution Loss Random Normal Initialization (default)

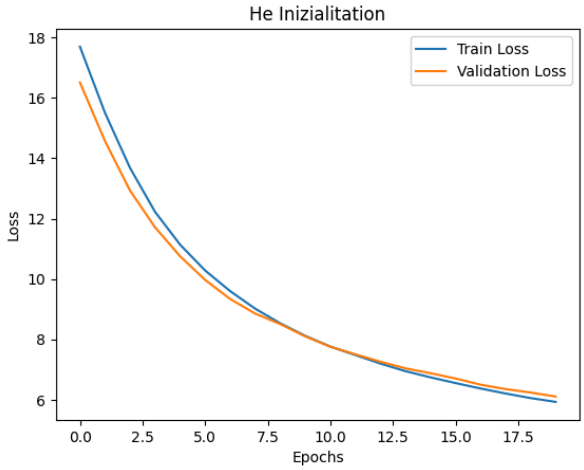


Plot Loss Random Normal Initialization (custom)

Plot Weight Distribution Loss Random Normal Initialization (custom)

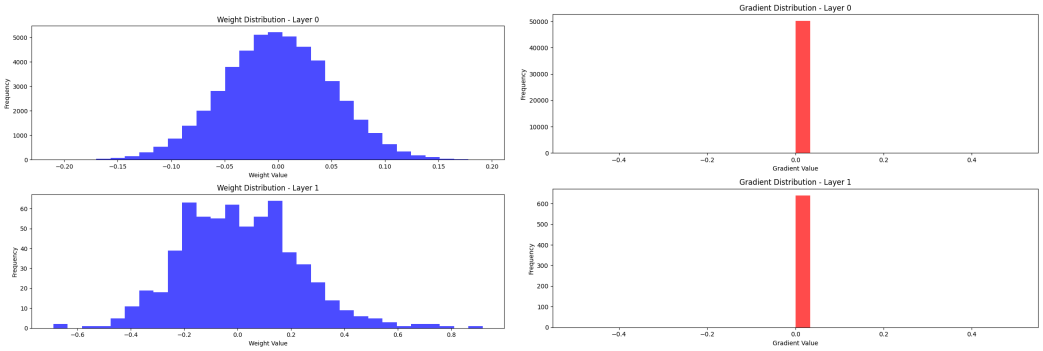


Plot Gradient Distribution Loss Random Normal Initialization (custom)

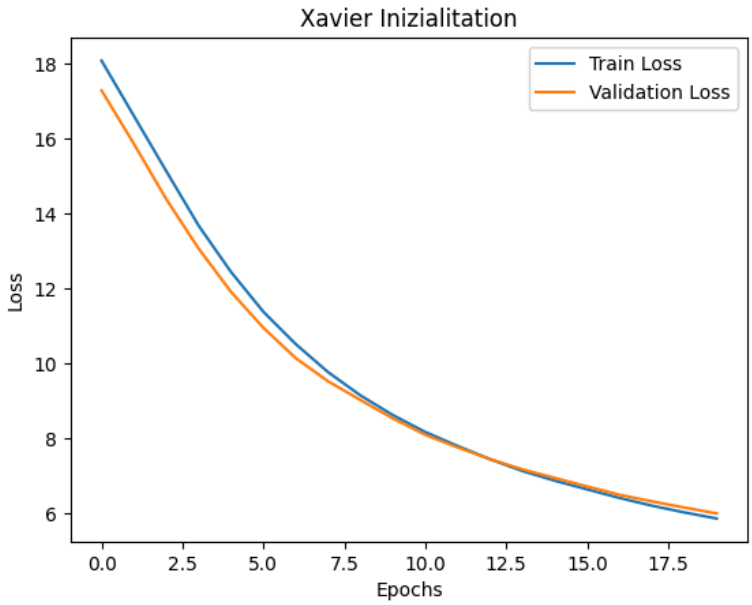


Plot Loss He Initialization

Plot Weight Distribution Loss He Initialization

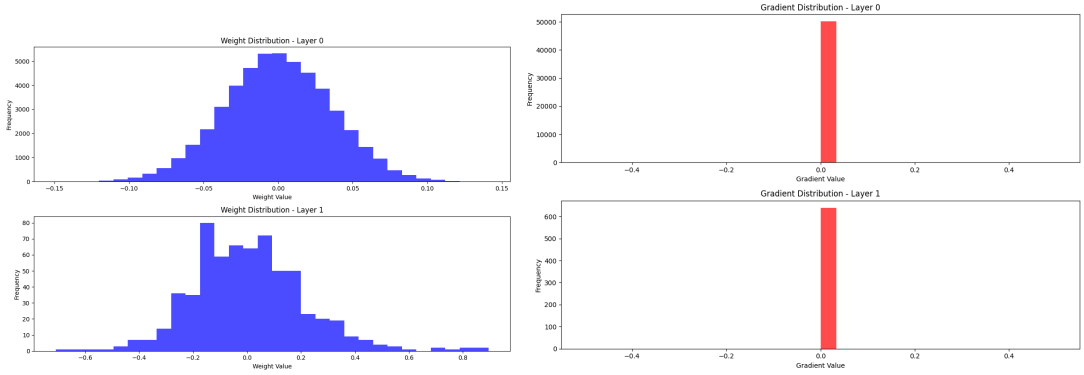


Plot Gradient Distribution Loss He Initialization



Plot Loss Xavier Initialization

Plot Weight Distribution Loss Xavier Initialization



Plot Gradient Distribution Loss Xavier Initialization

Selain plot loss, model dari berbagai variasi tersebut diterapkan untuk memprediksi data dengan hasil akurasi sebagai berikut:

Inisialisasi Bobot	Akurasi
Zero	0.1031
Random Uniform default	0.5294
Random Uniform custom	0.4446

Random Normal default	0.6619
Random Normal custom	0.2693
He	0.8281
Xavier	0.8425

Dari hasil pengujian tersebut, dapat disimpulkan bahwa dari plot loss, distribusi bobot, dan distribusi gradien bobot tampak perbedaan signifikan dalam berbagai metode inisialisasi bobot. Selain itu, dari hasil akurasi prediksi didapat bahwa inisialisasi menggunakan metode He dan Xavier memberikan hasil yang sangat baik. Sedangkan inisialisasi dengan Zero memberikan hasil yang sangat buruk. Hal tersebut terjadi karena He dan Xavier merupakan metode inisialisasi bobot yang dirancang untuk menjaga kestabilan aktivasi dan gradien selama training dan mengatur variasi bobot agar output tiap layer tidak terlalu kecil (vanishing) atau terlalu besar (exploding).

e. Perbandingan dengan library sklearn

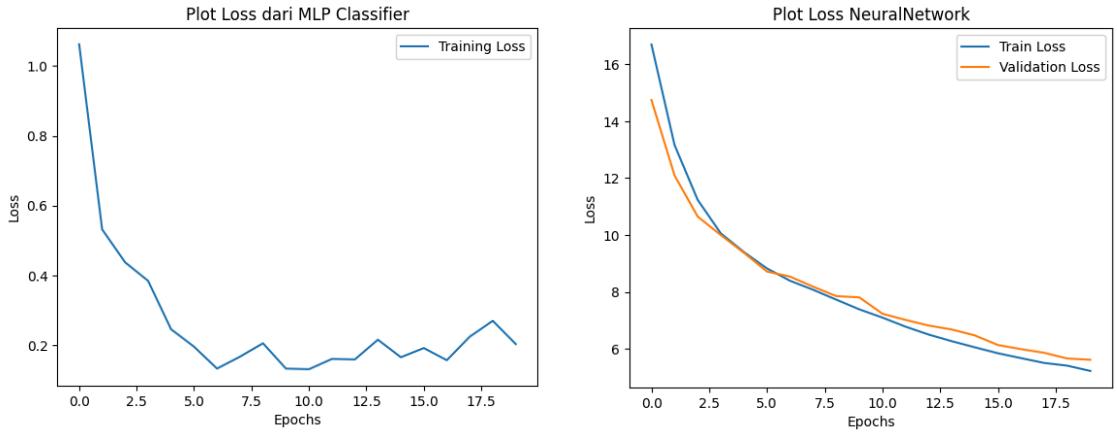
Model akan dibandingkan dengan model MLPClassifier dari sklearn. Hyperparameter yang digunakan adalah sebagai berikut.

- Learning rate : 0.001
- Epochs : 20
- Batch size : 1

Masing-masing model akan memiliki dua hidden layer dengan jumlah neuron pada hidden layer pertama adalah 64 dan jumlah neuron pada hidden layer kedua adalah 32. Berikut merupakan hasil yang didapatkan.

Model	**Akurasi**
NeuralNetwork	0.8987
MLPClassifier	0.814

Kemudian, berikut merupakan grafik loss dari kedua model.



Dari hasil yang didapatkan, model MLPClassifier memiliki akurasi yang lebih baik dibandingkan model yang kami buat. Hal ini menunjukkan bahwa model MLPClassifier lebih baik dalam melakukan proses pelatihan jika dibandingkan dengan model yang kami buat. Namun, pada grafik MLPClassifier, terlihat bahwa training loss mengalami fluktuasi setelah beberapa epoch. Hal ini bisa mengindikasikan bahwa model mengalami kesulitan dalam konvergensi. Sedangkan, pada model yang kami buat, train loss dan validation loss menurun dengan pola yang lebih stabil. Hal tersebut menandakan model lebih stabil dalam belajar dan memiliki generalization yang lebih baik.

C. Kesimpulan dan Saran

1. Kesimpulan

- Fungsi aktivasi dapat mempengaruhi kinerja model. Pemilihan fungsi aktivasi yang baik dapat membuat proses pelatihan menjadi lebih cepat menuju kekonvergenan dan membuat model bisa memprediksi dengan lebih baik.
- Nilai learning rate sangat mempengaruhi proses pelatihan model. Nilai learning rate yang terlalu kecil akan membuat model lebih lama dalam melakukan proses pelatihan. Nilai learning rate yang terlalu besar akan membuat pembaruan bobot menjadi tidak stabil.

- Variasi width dan depth mempengaruhi mode. Semakin banyak neuron (semakin tinggi width) maka proses pembelajaran semakin baik. Semakin banyak layer (semakin tinggi depth) maka proses pembelajaran akan cenderung tidak baik jika pemilihan hyperparameter lainnya belum sesuai (fungsi aktivasi dan inisialisasi yang belum sesuai membuat faktor banyak hidden layer tidak menambah kualitas model tetapi cenderung menurunkan jika tidak cocok).
- Pengaruh metode inisialisasi model cukup signifikan. Zero, Random Uniform, dan Random Normal memberikan model yang tidak terlalu baik. Terutama Zero, karena inisialisasi bobot dengan 0 semua membuat proses pembelajaran kurang efektif dan kesannya semua neuron sama saja pengaruhnya. He dan Xavier merupakan metode yang cukup baik untuk membuat model yang kinerjanya bagus.
- Model yang dibuat sudah cukup baik. Namun, model masih cukup lambat menuju kekonvergenan jika dibandingkan dengan model MLPClassifier pada sklearn.

2. Saran

- Optimisasi hyperparameter dengan melakukan pencarian secara sistematis (menggunakan grid search atau random search) untuk menentukan nilai learning rate, width, dan depth yang optimal.
- Pemilihan fungsi aktivasi yang cocok dengan dataset karena dapat mempercepat konvergensi dan meningkatkan akurasi.
- Sebaiknya menggunakan kombinasi metode inisialisasi He dan Xavier untuk memaksimalkan performa.
- Penerapan regularisasi dan normalisasi akan mengurangi overfitting terutama jika cenderung overfit pada data training.

D. Pembagian Tugas/Kerja per Anggota Kelompok

Nama	NIM	Pembagian Kerja
Muhammad Zakkiy	10122074	Mengerjakan syntax bagian forward propagation, backward propagation, train, predict, plot_loss,

		initialize_weights, load dataset mnist_784 dan menggunakan method fetch_openml. Melakukan pengujian untuk variasi fungsi aktivasi, learning rate, dan perbandingan dengan library sklearn MLP. Mengerjakan laporan bagian penjelasan implementasi, penjelasan forward propagation, penjelasan backward propagation dan weight update, pengaruh fungsi aktivasi, pengaruh learning rate, perbandingan dengan library sklearn, dan kesimpulan dan saran. Melakukan fiksasi syntax. Membuat repository github.
Ghaisan Zaki Pratama	10122078	Mengerjakan syntax bagian fungsi aktivasi, turunan fungsi aktivasi, loss function, turunan loss function, plot bobot, plot distribusi bobot, representasi graf. Melakukan pengujian untuk variasi depth dan width dan inisialisasi bobot. Mengerjakan laporan bagian deskripsi persoalan, penjelasan implementasi, deskripsi kelas beserta atribut dan method, pengaruh depth dan width, dan pengaruh inisialisasi bobot. Membuat template laporan.

E. Referensi

GeeksforGeeks. (n.d.). *Kaiming Initialization in Deep Learning*. Diakses pada 28 Maret 2025, dari https://www.geeksforgeeks.org/kaiming-initialization-in-deep-learning/

GeeksforGeeks. (n.d.). *Xavier Initialization*. Diakses pada 28 Maret 2025, dari https://www.geeksforgeeks.org/xavier-initialization/

Kashyap, P. (n.d.). *Mastering weight initialization in neural networks: A beginner's guide*. Medium. Diakses pada 28 Maret 2025, dari https://medium.com/@piyushkashyap045/mastering-weight-initialization-in-neural-networks-a-beginners-guide-6066403140e9