**TUGAS BESAR 1**

**IF3270 Pembelajaran Mesin**

Feedforward Neural Network



Disusun oleh Kelompok 37

| Benjamin Sihombing | 13522054 |
| --- | --- |
| M. Atpur Rafif | 13522086 |
| Suthasoma M. Munthe | 13522098 |

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2025**

# **Dekripsi Persoalan**

*Feed forward neural network* merupakan *artificial* *neural network* yang hanya bisa prosesnya (*forward propagation*) hanya bergerak satu arah dari *layer* terawal ke *layer* yang terakhir dan tidak memungkinkan adanya *looping.* Setiap *layer* terdiri dari 1 atau lebih *neuron*. Di dalam suatu *neuron* akan dilakukan operasi perhitungan kombinasi linear dan fungsi aktivasi. Model yang bagus adalah model yang hasil prediksinya mirip bahkan sama dengan nilai sesungguhnya. Untuk mengevaluasi model digunakan loss function. Ada 3 jenis *loss function* yang digunakan, yaitu *mean square error*, *binary* *cross entropy*, *categorical cross* *entropy*. Selain itu, *loss function* juga digunakan untuk *backpropagation*. *Backpropagation* digunakan untuk mengubah bobot dengan harapan model dengan bobot terbaru memiliki performa yang lebih baik.

Pada tugas ini akan dibuat model ANN dari *scratch*. Hal-hal yang perlu diimplementasikan dari pada tugas ini:

1. Fungsi aktivasi
   1. Linear
   2. ReLU
   3. Sigmoid
   4. Tanh
   5. Softmax
2. *Loss function*
   1. *Mean cross error*
   2. *Binary cross entropy*
   3. *Categorical cross entropy*
3. Inisialisasi bobot
   1. *Zero*
   2. *Uniform*
   3. *Normal*
4. *Forward propagation*
5. *Backward propagation*
6. Model harus bisa disimpan dan dimuat
7. Data dari model FFNN seperti bobot, gradien bobot, beserta distribusinya bisa ditampilkan baik.
8. Arsitektur/Struktur model FFNN harus bisa ditampilkan dalam bentuk graf.

# 

# 

# **Pembahasan**

## Implementasi

### Deskripsi Kelas

#### Fungsi aktivasi

Ada 5 jenis fungsi aktivasi, yaitu linear, ReLU, sigmoid, tanh, dan softmax. Fungsi aktivasi diimplementasikan pada file Activation.py. ActivationFunction merupakan kelas *abstract* dan LinearActivationFunction, ReluActivationFunction, SigmoidActivationFunction, TanhActivationFunction, SoftmaxActivationFunction merupakan *inheritance* dari ActivationFunction.

#### Inisialisasi bobot

Ada 3 jenis inisialisasi bobot, yaitu *zero*, *uniform*, dannormal. Fungsi aktivasi diimplementasikan pada file Initialization.py. InitializationFunction merupakan kelas *abstract* dan ZeroInitialization, UniformInitialization, NormalInitialization merupakan *inheritance* dari InitializationFunction.

#### *Loss function*

Ada 3 jenis *loss function*, yaitu *mean square error*, *binary cross entropy*, dan *categorical cross entropy*. Fungsi aktivasi diimplementasikan pada file Loss.py. ErrorFunction merupakan kelas *abstract* dan MeanSquaredError, BinaryCrossEntropy, CategoricalCrossEntropy merupakan *inheritance* dari ErrorFunction.

#### *Layer*

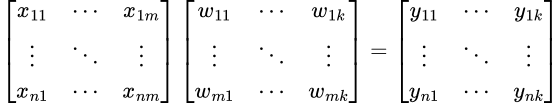
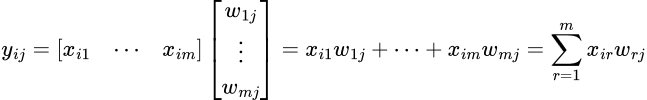
*Layer* adalah sebuah kelas yang merepresentasikan layer pada FFNN. Di dalam kelas ini terdapat atribut activation, initializer, weight, bias, weight\_gradient, bias\_gradient, input, dan linear\_combinations. Kelas ini juga memiliki metode run, update\_gradient, dan update\_weight. Metode *run* digunakan untuk menghitung kombinasi linear (net) dan hasil aktivasi net. Metode lainnya berfungsi sesuai namanya.

#### FFNN

FFNN adalah sebuah kelas yang merepresentasikan model FFNN. Di dalam kelas ini terdapat atribut activations, loss, weight\_initializer, layers, dan layers\_sizes. Kelas ini juga memiliki metode forward, backward, update\_weight, fit, predict, save\_model, plot\_weight, plot\_gradient\_weight, show\_graph, load\_model dan update\_weight. Metode plot\_weight digunakan untuk menampilkan distribusi weight dari suatu atau beberapa *layer*. Metode plot\_gradient\_weight digunakan untuk menampilkan distribusi gradient weight dari suatu atau beberapa *layer*. Metode show\_graph digunakan untuk memvisualisasi arsitektur FFNN dalam bentuk graf.

### *Forward propagation*

Setiap layer pada network direpresentasikan menggunakan matrix. Definisi perkalian matrix sendiri merupakan kombinasi linear antara baris dengan kolom. Singkatnya,

dengan,

Perhatikan bahwa kita bisa mendeskripsikan variabel di atas sebagai berikut:

1. : Input *neuron* untuk *instance* ke-, serta dimensi ke-
2. : Bobot dari neuron ke-, menuju neuron ke- di layer selanjutnya
3. : Output *neuron* untuk *instance* ke-, serta dimensi ke-

Setelah mendapatkan kombinasi linear yang dibutuhkan oleh *neuron*, untuk mendapatkan *output*, kita harus memasukan matrix kedalam fungsi aktivasi. Fungsi ini bergantung pada setiap *layer*. Jika fungsi aktivasi memiliki bentuk sederhana seperti ℝ → ℝ, mengaplikasikan fungsi pada setiap elemen matriks sudah cukup. Namun hal ini tidak bisa dilakukan dan harus ditangani secara khusus untuk fungsi yang lebih rumit, contohnya fungsi aktivasi *softmax*.

Selain itu, jangan lupa untuk menambahkan bias sebelum memasukan input ke dalam sebuah layer. Misalkan kita menggunakan sigmoid untuk seluruh fungsi aktivasi. Algoritma yang diimplementasikan sebagai berikut:

### *Backward propagation*

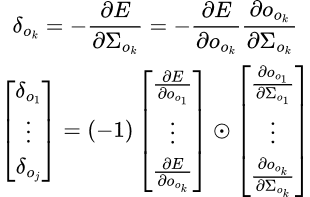
Penurunan mengenai persamaaan penyesuaian bobot berdasarkan gradien telah diajarkan di kelas, namun tidak dijelaskan penggunaan matrix untuk melakukan perhitungan. Berikut merupakan pemanfaatan matrix dalam forward dan backward propagation:



Misalkan kita memiliki network dengan bentuk seperti diatas. Fokus terhadap output dan satu hidden layer sebelum output. Error term () dimiliki oleh neuron, atau bisa dipandang sebagai kontribusi error total dari sebuah neuron. Berdasarkan definisinya, nilainya adalah:

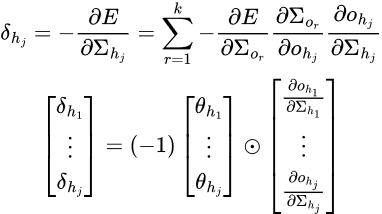
Terdapat dua kasus untuk menghitung error term, pertama pada output layer, dan kedua pada hidden layer.

1. Output Layer

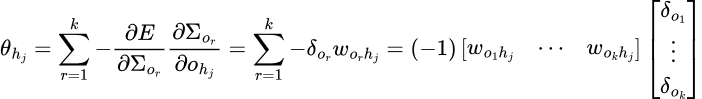


Penggunaan matrix untuk menghitung error term pada output layer merupakan hal yang straightforward. Kita hanya perlu menyusun error term menjadi sebuah kolom. Simbol merupakan perkalian Hadamard, yaitu perkalian matrix dengan melakukan perkalian pada setiap elemen. Kita tidak bisa menyederhanakannya lebih lanjut, karena kedua suku matrix diatas bergantung terhadap fungsi error dan fungsi aktivasi output layer.

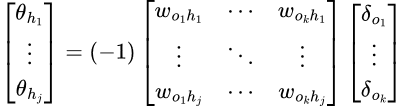
1. Hidden Layer



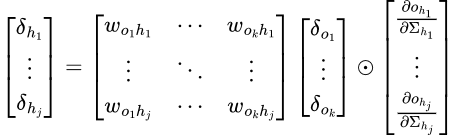
Pada hidden layer, fungsi error sudah terlalu complicated. Sehingga untuk mempermudah, kita dapat meminjam error term yang sudah dihitung pada layer didepannya. Pada persamaan diatas, untuk mempermudah penulisan, kita membuat variabel baru, dengan definisi:



Kita bisa menyusunnya menjadi kolom, sehingga didapat:



Suku matrix pertama didapat dari error term output layer. Sedangkan suku matrix kedua merupakan matrix bobot pada output layer. Perhatikan bahwa pada matrix bobot, bobot dari bias tidak dituliskan. Hal ini karena bias sendiri bukan merupakan neuron, dan tidak memiliki fungsi aktivasi, sehingga tidak memiliki error term. Substitusi dapat dilakukan, menghasilkan:

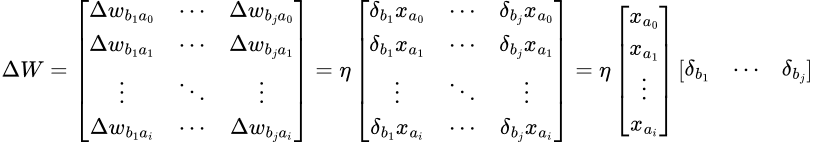


Setelah mendapatkan error term setiap neuron, sekarang saatnya mencari penyesuaian bobot. Hal ini dapat dilakukan dengan melakukan ekspansi dari perkalian dua vektor. Perkalian dua vektor (biasanya disebut perkalian dot) menghasilkan sebuah nilai, yaitu skalar. Namun jika dimensi tetap, namun urutan perkalian dibalik, dapat menghasilkan matrix.

Tahap ini merupakan tahap yang lebih general, dapat digunakan untuk output atau hidden layer. Setiap layernya, hal yang dibutuhkan adalah error term (sudah didapatkan sebelumnya) dalam bentuk vektor, dan vektor input yang digunakan pada layer tersebut, termasuk bias. Misalkan kita ingin menghitung penyesuaian bobot pada layer , yang memiliki input berupa output dari layer (layer sebelumnya) ditambah nilai bias. Persamaan untuk menghitung penyesuaian sebuah bobot adalah sebagai berikut:



Sebenarnya di kelas diajarkan untuk menggunakan dua parameter, yaitu neuron asal dan neuron tujuan. Namun perhatikan bahwa tidak seperti bobot, asalkan neuron asal sama, tidak peduli neuron tujuan, nilai mereka semua adalah sama. Selanjutnya, ekspansi dapat dilakukan dengan:

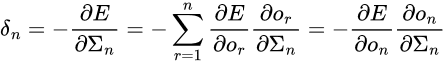


Hal terakhir yang perlu dilakukan adalah menyesuaikan bobot dengan menjumlahkan matrix bobot dan matrix penyesuaian bobot ini.

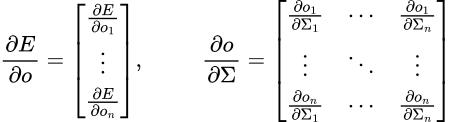
## 

### Analisis Softmax

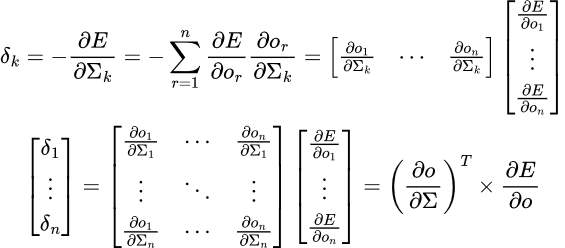
Pada bagian sebelumnya dijelaskan bahwa fungsi *softmax* merupakan fungsi yang relatif kompleks dibandingkan dengan fungsi lainnya. Hal ini karena domain dan range dari fungsi ini adalah vektor, bukan bilangan real. Sehingga terdapat ketergantungan antara elemen vektor. Pada *output layer*, fungsi aktivasi sebelumnya dapat melakukan penyederhanaan sebagai berikut:



Hal ini dikarenakan, sesama kombinasi linear pada sebuah layer tidak akan mempengaruhi nilai output neuron pada layer tersebut. Namun hal ini tidak benar untuk fungsi *softmax*, karena berdasarkan definisinya sendiri, setiap elemen output bergantung terhadap seluruh elemen input. Oleh karena itu, kita harus melakukan manipulasi yang lebih rumit dibandingkan dengan yang biasanya. Seperti yang dijelaskan pada referensi yang diberikan, fungsi ini memiliki turunan yang dinamakan Matrix Jacobian. Selain itu, juga terdapat penurunan *error* yang berupa skalar terhadap vektor output neuron. Keduanya dituliskan sebagai berikut:

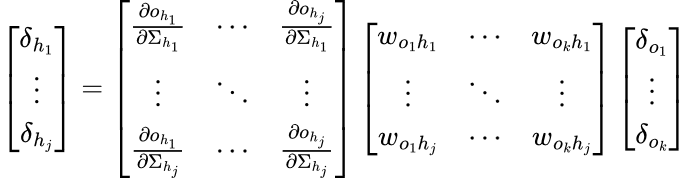


Lalu, dengan melakukan manipulasi dan penyusunan seperti tahap sebelumnya, didapat:

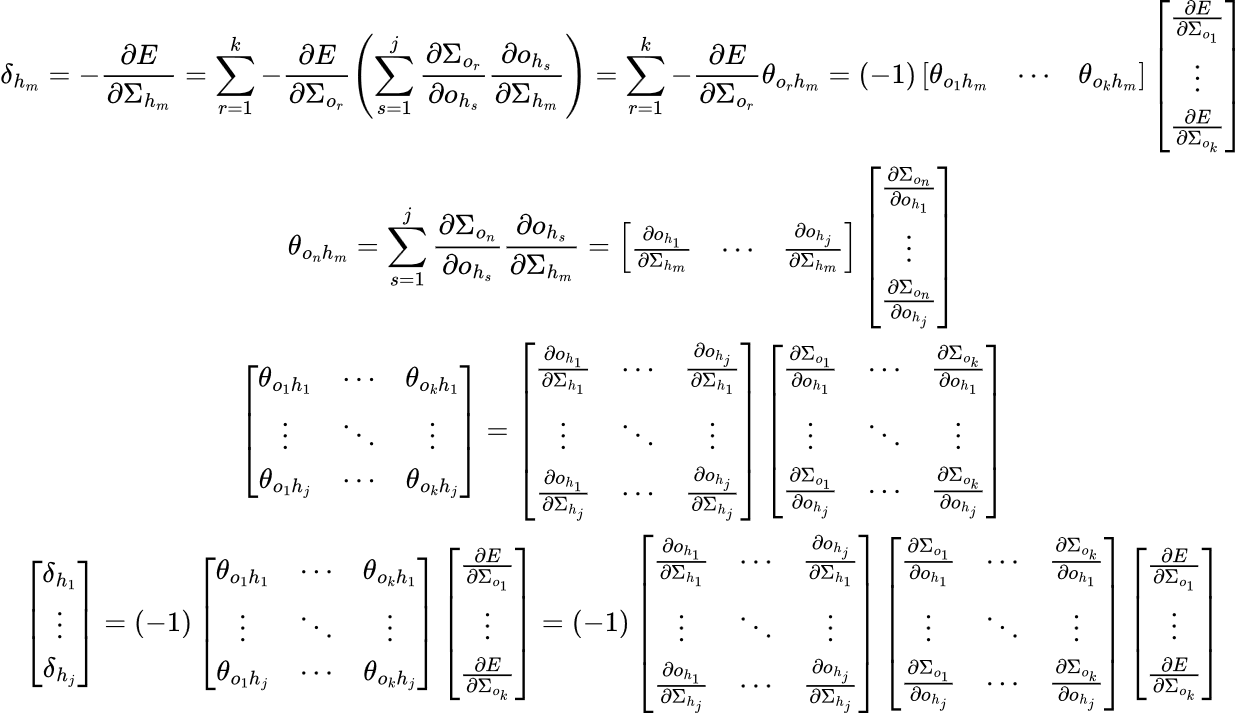


Bentuk ini jauh lebih general dibandingkan bentuk sebelumnya (pada bagian *output layer*), karena dengan membuat nol seluruh elemen matrix jacobian, kecuali diagonal. Maka didapatkan bentuk yang sama seperti sebelumnya, yaitu perkalian Hadamard.

Menggunakan inspeksi, terlihat bahwa bentuk *backpropagation* pada hidden layer di bagian sebelumnya dapat digeneralisasi menjadi bentuk sebagai berikut:



Bentuk diatas, secara lebih detail didapatkan dari:



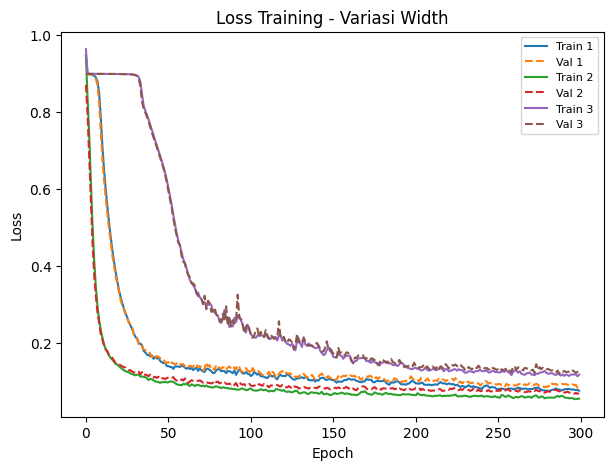
Suku kedua diubah menjadi matrix bobot, serta suku ketiga diubah menjadi error term layer setelahnya. Sama seperti sebelumnya, apabila pada suku pertama, seluruh elemen selain diagonal dijadikan nol. Maka didapatkan perkalian Hadamard.

## Hasil Pengujian

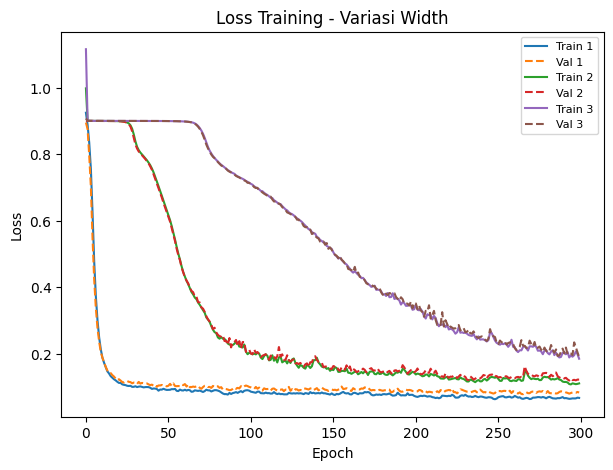
### **Pengaruh *depth* dan *width***

| **Konfigurasi** | |
| --- | --- |
| Activation | Sigmoid (all) |
| Epoch | 300 |
| Learning rate | 0.1 |
| Batch size | 50 |
| Error | MSE |
| Weight initialization | Normal (seed: 73) |
| Width | 128 (untuk uji depth) |
| Depth | 4 (untuk uji width) |
| Model\_depth\_1 | [784, 128, 128, 128, 10] |
| Model\_depth\_2 | [784, 128, 128, 10] |
| Model\_depth\_3 | [784, 128, 128, 128, 128, 10] |
| Model\_width\_1 | [784, 256, 256, 256, 10] |
| Model\_width\_2 | [784, 64, 64, 64, 10] |
| Model\_width\_3 | [784, 32, 32, 32, 10] |

| **Hasil** | |
| --- | --- |
| Model\_depth\_1 | 0.9494 |
| Model\_depth\_2 | 0.9616 |
| Model\_depth\_3 | 0.9224 |
| Model\_width\_1 | 0.9550 |
| Model\_width\_2 | 0.9317 |
| Model\_width\_3 | 0.8981 |



Gambar 2.a.2 Grafik loss pelatihan 3 model variasi depth

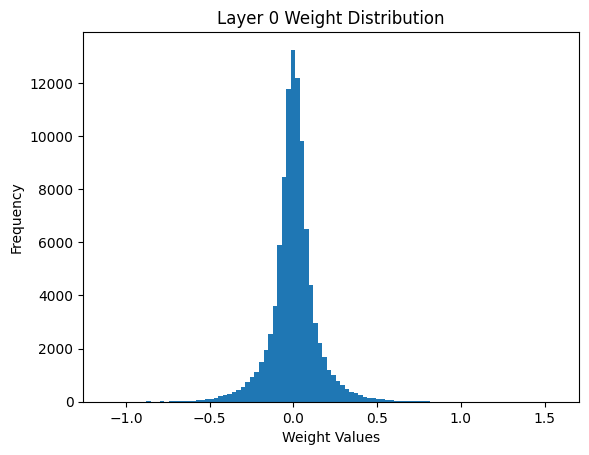


Gambar 2.a.2 Grafik loss pelatihan 3 model variasi width

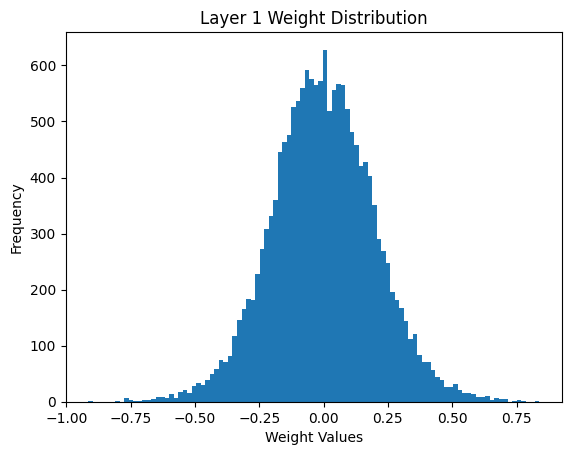
### **Pengaruh fungsi aktivasi**

| **Konfigurasi** | |
| --- | --- |
| Epoch | 100 |
| Learning rate | 0.1 |
| Batch size | 50 |
| Error | MSE |
| Weight initialization | Normal (seed: 73) |
| Layer size (depth dan width) | [784, 128, 128, 64, 10] |

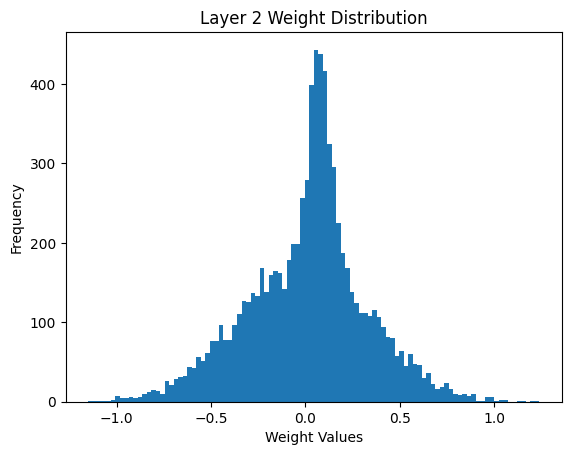
| **Hasil** | |
| --- | --- |
| Sigmoid | 0.9156 |
| ReLU | 0.0979 |
| Linear | Error (overflow) |
| Tanh | 0.8740 |
| SeLU | Error (overflow) |
| Leaky | Error (overflow) |



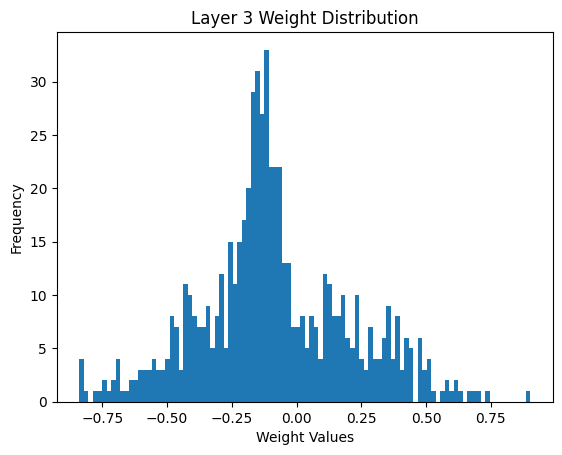
Gambar 2.b.1 Grafik distribusi weight pada layer 0 model sigmoid



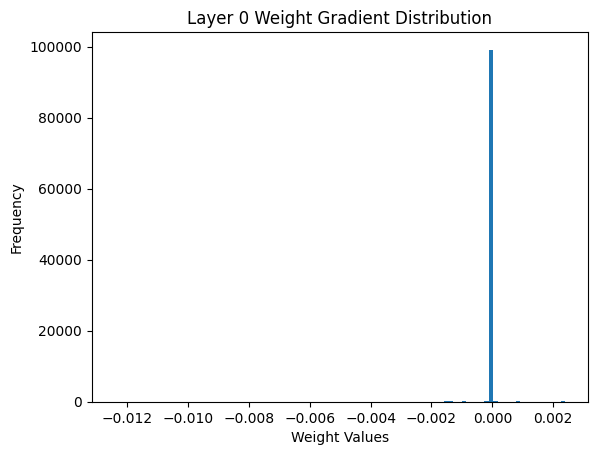
Gambar 2.b.2 Grafik distribusi weight pada layer 1 model sigmoid



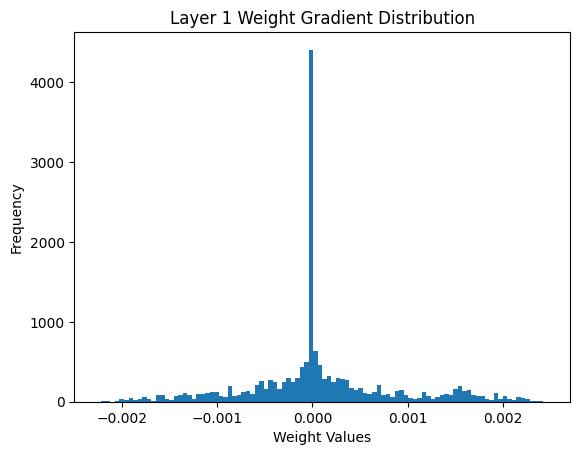
Gambar 2.b.3 Grafik distribusi weight pada layer 2 model sigmoid



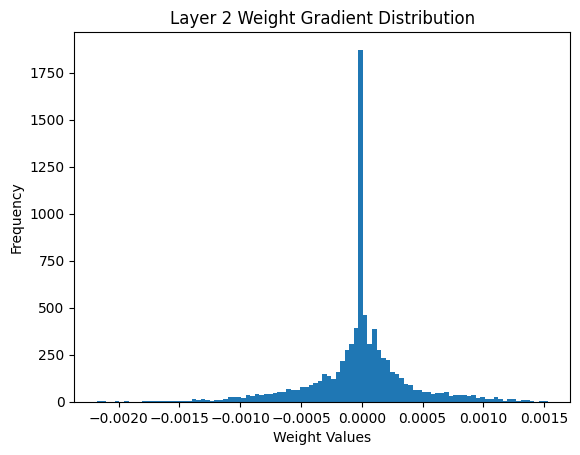
Gambar 2.b.4 Grafik distribusi weight pada layer 3 model sigmoid



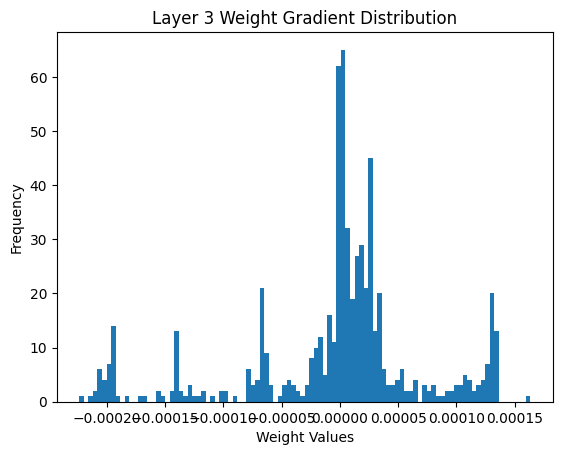
Gambar 2.b.5 Grafik distribusi weight gradient pada layer 0 model sigmoid



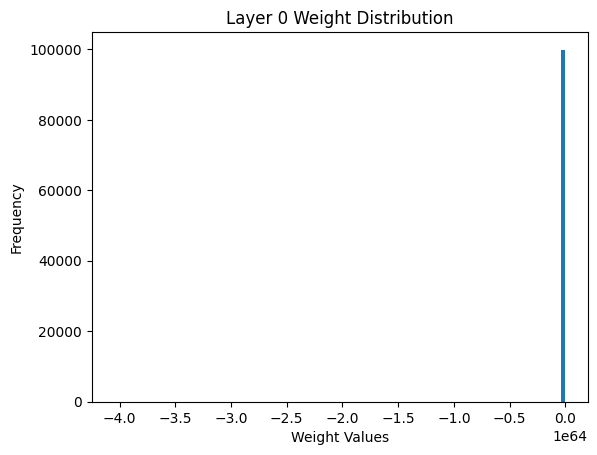
Gambar 2.b.6 Grafik distribusi weight gradient pada layer 1 model sigmoid



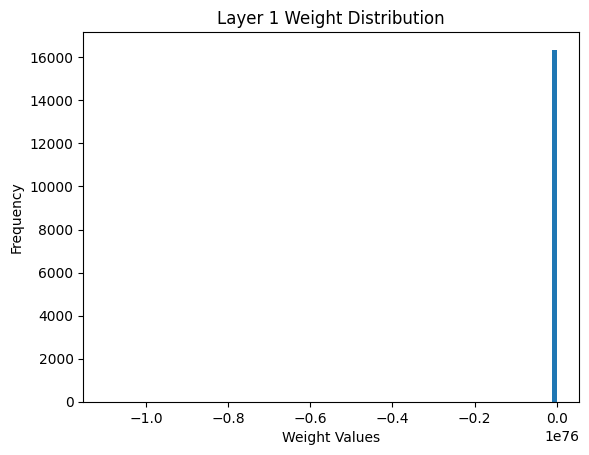
Gambar 2.b.7 Grafik distribusi weight gradient pada layer 2 model sigmoid



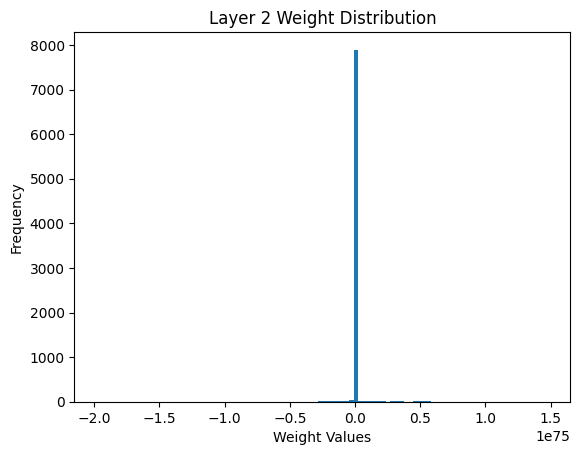
Gambar 2.b.8 Grafik distribusi weight gradient pada layer 3 model sigmoid



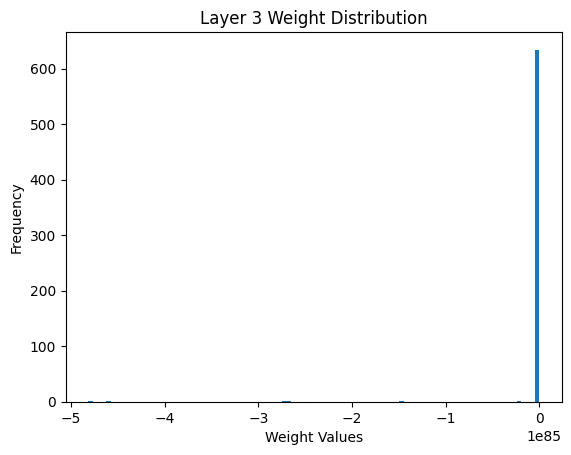
Gambar 2.b.9 Grafik distribusi weight pada layer 0 model relu



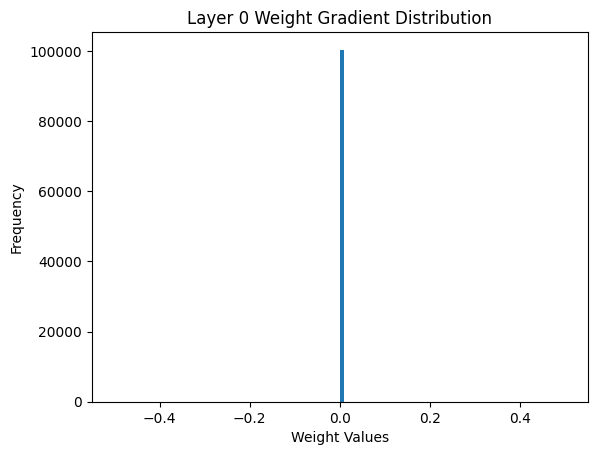
Gambar 2.b.10 Grafik distribusi weight pada layer 1 model relu



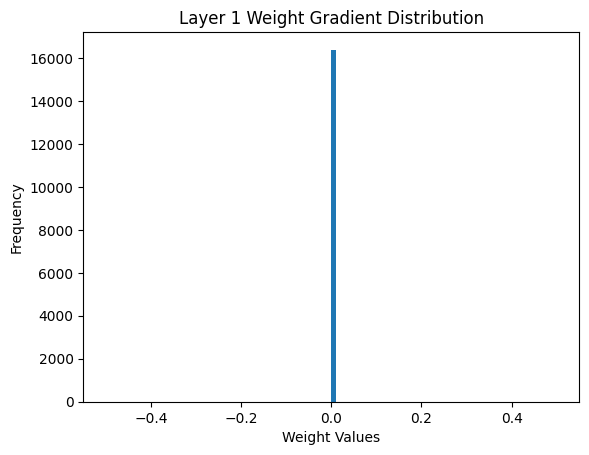
Gambar 2.b.11 Grafik distribusi weight pada layer 2 model relu



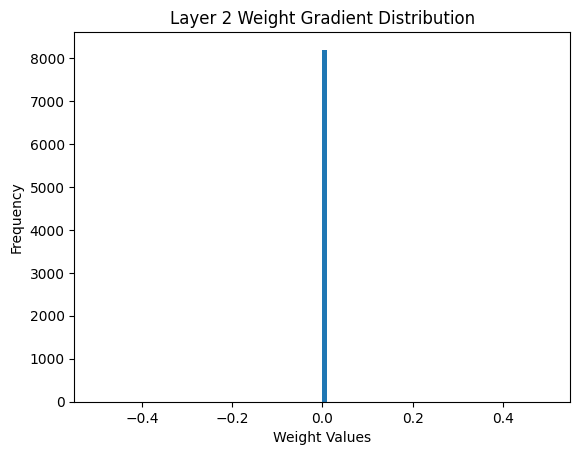
Gambar 2.b.12 Grafik distribusi weight pada layer 3 model relu



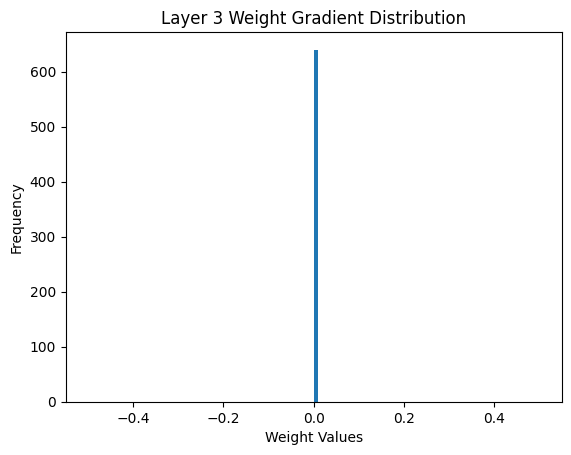
Gambar 2.b.13 Grafik distribusi weight gradient pada layer 0 model relu



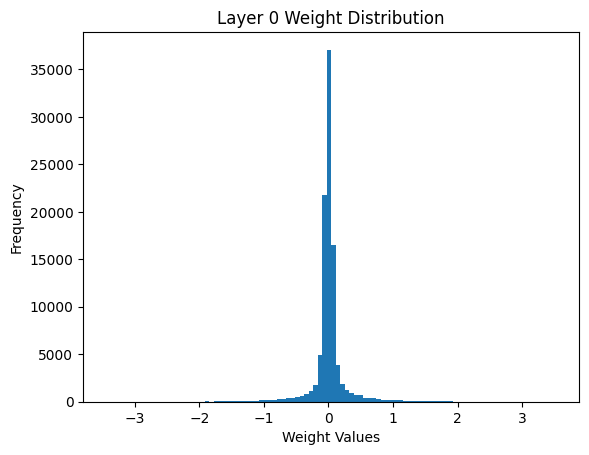
Gambar 2.b.14 Grafik distribusi weight gradient pada layer 1 model relu



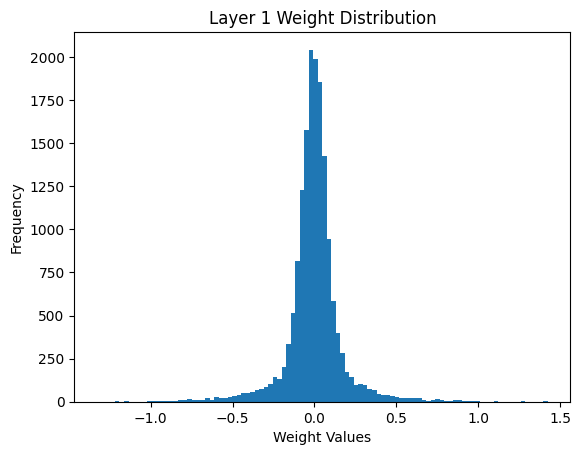
Gambar 2.b.15 Grafik distribusi weight gradient pada layer 2 model relu



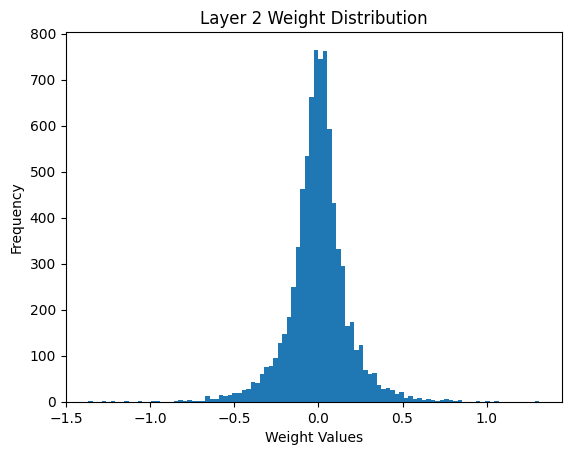
Gambar 2.b.16 Grafik distribusi weight gradient pada layer 3 model relu



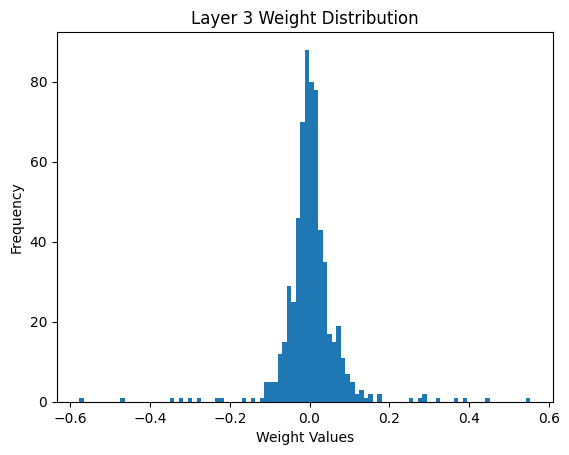
Gambar 2.b.17 Grafik distribusi weight pada layer 0 model tanh



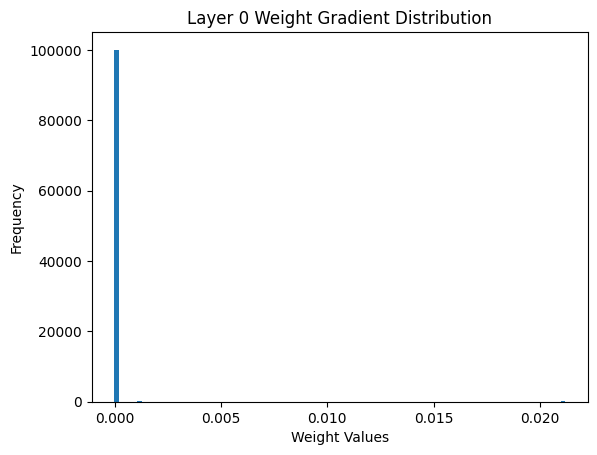
Gambar 2.b.18 Grafik distribusi weight pada layer 1 model tanh



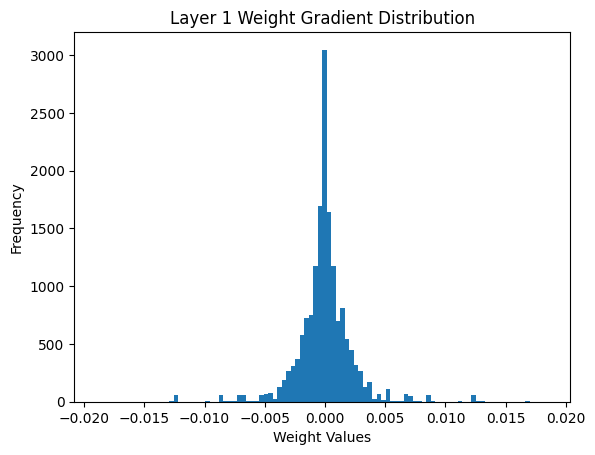
Gambar 2.b.19 Grafik distribusi weight pada layer 2 model tanh



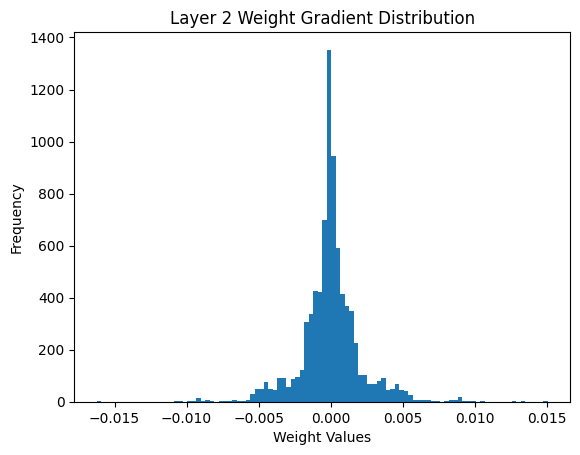
Gambar 2.b.20 Grafik distribusi weight pada layer 3 model tanh



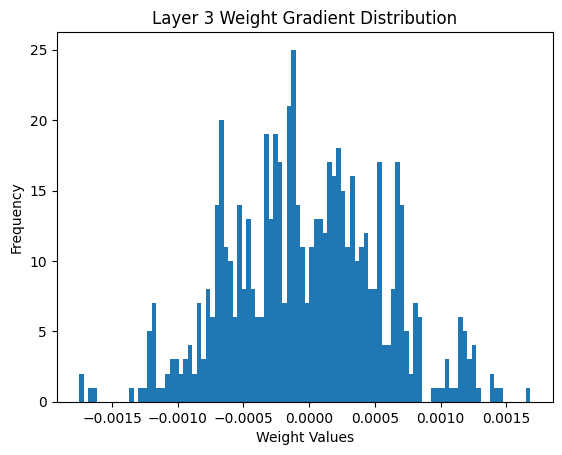
Gambar 2.b.21 Grafik distribusi weight gradien pada layer 0 model tanh



Gambar 2.b.22 Grafik distribusi weight gradien pada layer 2 model tanh



Gambar 2.b.23 Grafik distribusi weight gradien pada layer 2 model tanh



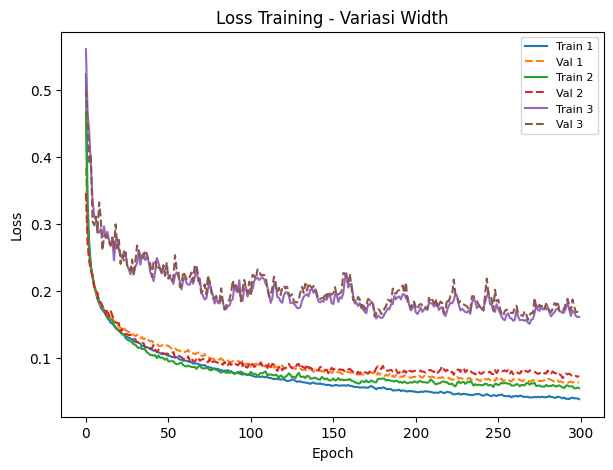
Gambar 2.b.24 Grafik distribusi weight gradien pada layer 0 model tanh

### **Pengaruh *learning rate***Konfigurasi yang digunakan pada pengujian ini adalah Mean Squared Error, dengan jumlah epoch sebanyak 300. Bobot diinisialisasi menggunakan Normal Initializer. Terdapat 2 hidden layer (masing-masing 128 *neurons*) dan 1 output layer (10 *neurons*). Fungsi aktivasi pada *hidden layer* adalah fungsi sigmoid dan fungsi pada *output layer* adalah fungsi Hyperbolic Tangent. Variasi nilai *learning rate* yang digunakan adalah 0.05, 0.1, dan 0.5.

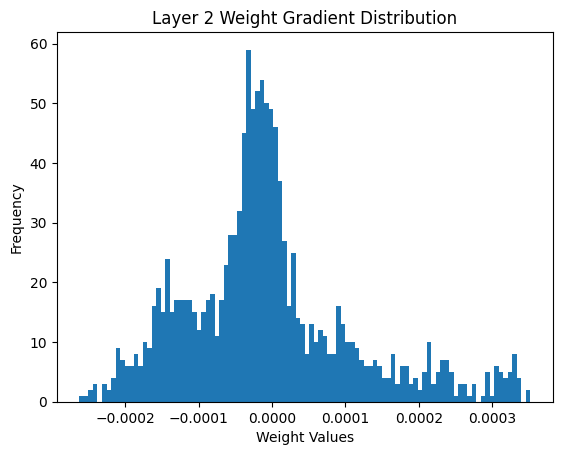
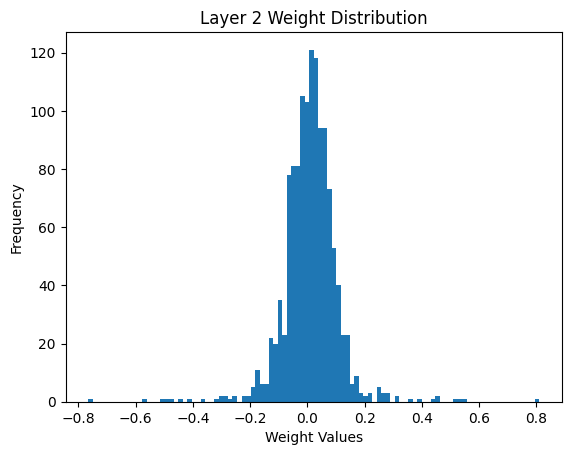
Hasil pengujian ketiga model menunjukkan nilai akurasi yang berbeda. Nilai akurasi pada model dengan learning rate yang lebih kecil lebih baik. Berikut ini adalah nilai learning rate ketiga model.

| Learning rate | Nilai akurasi |
| --- | --- |
| 0.05 | 0.9679 |
| 0.1 | 0.9626 |
| 0.5 | 0.8991 |

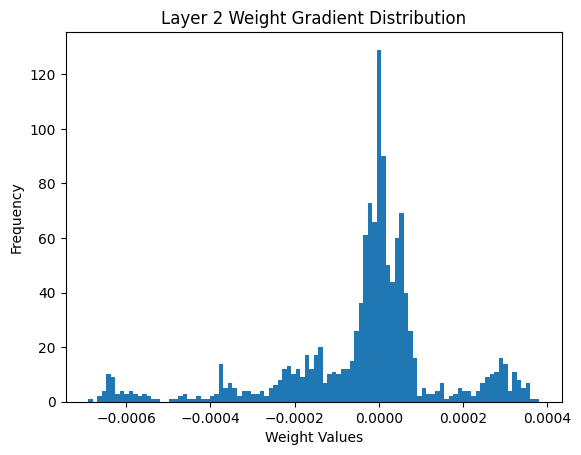
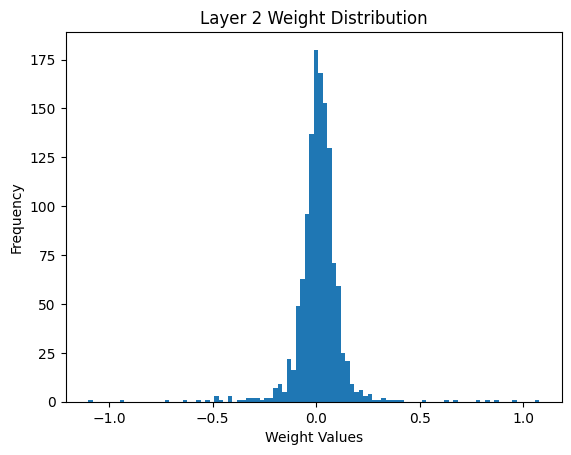
Grafik loss yang dihasilkan saat pelatihan dapat dilihat pada gambar di bawah ini. Grafik loss model dengan learning rate yang lebih kecil, cenderung lebih kecil pada epoch di atas 5. Sementara model dengan epoch yang paling besar (0.5) menunjukkan grafik loss yang jauh lebih besar.



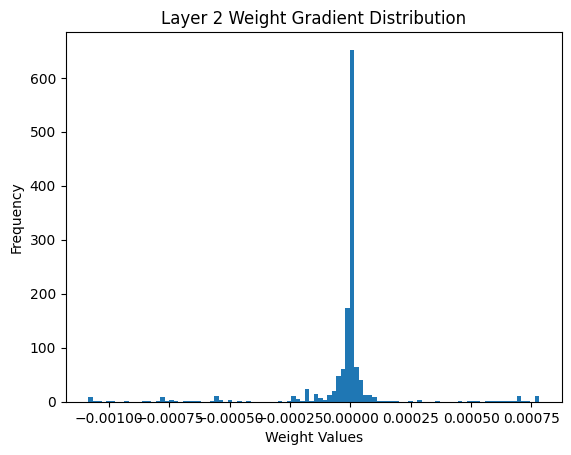
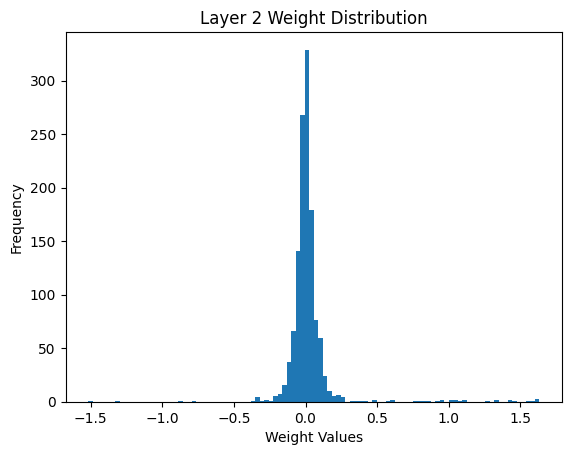
Gambar 2.c.1 Grafik loss pelatihan 3 model variasi learning rate



Gambar 2.c.2 Distribusi bobot dan gradien bobot *output layer* model 1 (0.05)



Gambar 2.c.3 Distribusi bobot dan gradien bobot *output layer* model 2 (0.1)

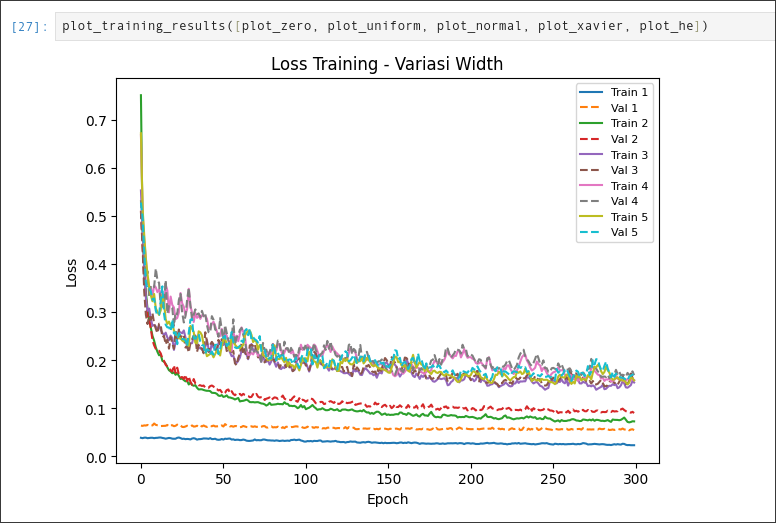


Gambar 2.c.4 Distribusi bobot dan gradien bobot *output layer* model 3 (0.5)

### 

### **Pengaruh inisialisasi bobot**

Konfigurasi yang digunakan adalah Mean Squared Error, Epoch = 300, dan Batch = 50. Neuron pada setiap layer adalah 784, 128, 128, 10. Pada plot dibawah ini, merupakan hasil loss traning yang didapatkan

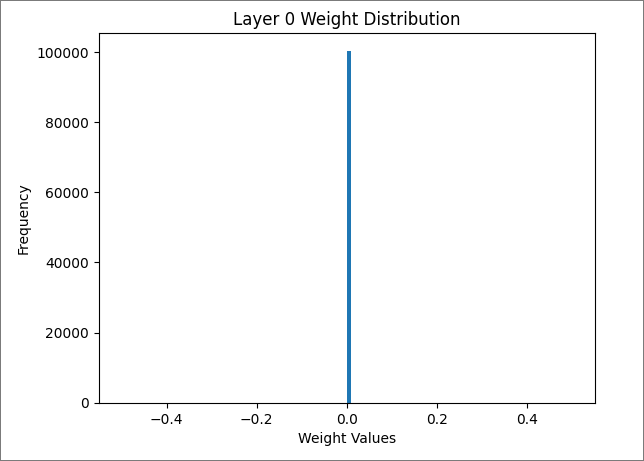


Gambar 2.d.1 Grafik training loss terhadap epoch

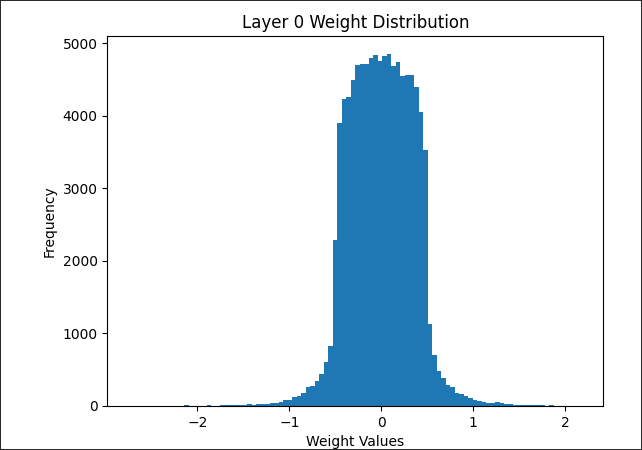
Sedangkan, untuk akurasi, berikut merupakan nilai masing-masing:

* Zero : 0.0979
* Uniform : 0.9534
* Normal : 0.9050
* Xavier : 0.9011
* He : 0.9073

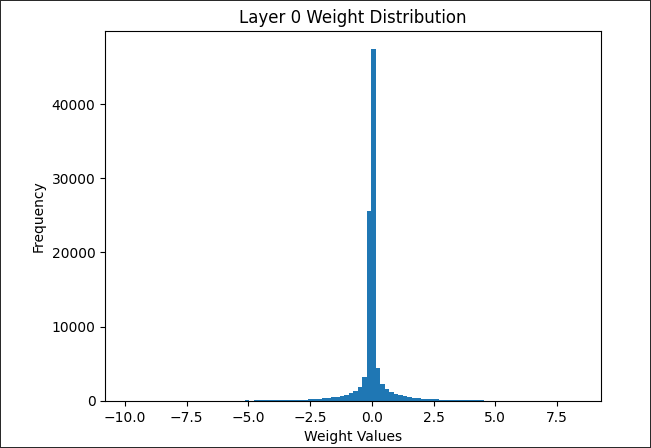
Kemudian, untuk distribusi dari bobot adalah sebagai berikut:



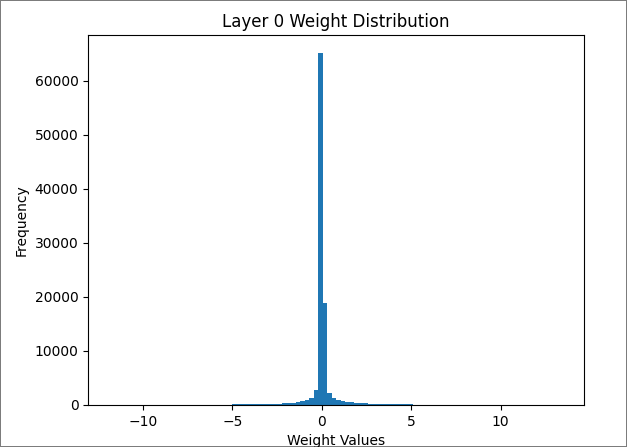
Gambar 2.d.2 Grafik distribusi bobot menggunakan initializer zero



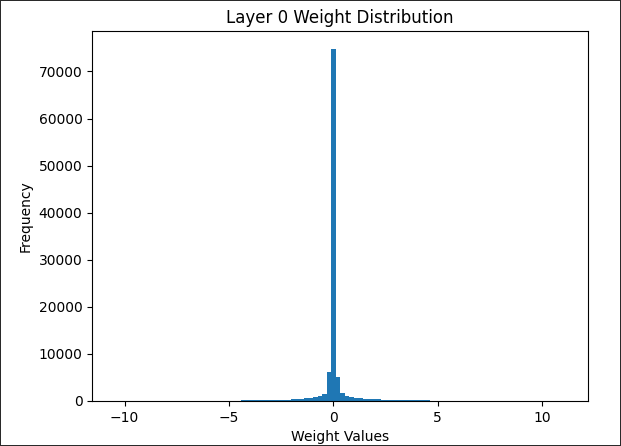
Gambar 2.d.3 Grafik distribusi bobot menggunakan initializer uniform



Gambar 2.d.4 Grafik distribusi bobot menggunakan initializer normal



Gambar 2.d.5 Grafik distribusi bobot menggunakan initializer xavier



Gambar 2.d.6 Grafik distribusi bobot menggunakan initializer he

### **Perbandingan dengan *library* sklearn** Berikut merupakan perbandingan dengan library sklearn, dengan konfigurasi Epoch = 10, Activation = Sigmoid, Learning Rate = 0.05, Batch Size = 50. Neuron pada setiap layer adalah 784, 128, 128, 10. Keduanya sama-sama menggunakan log-loss (berdasarkan dokumentasi sklearn), namun memiliki nilai loss awal yang berbeda.

### 

Selain itu, akurasi yang didapatkan sebagai berikut:

* FFNN: 0.1187
* Sklearn: 0.3551

# 

# **Kesimpulan dan Saran**

## Kesimpulan

Pada pengujian *depth* dan *width*, hasil menunjukkan semakin besar *depth*, akurasi model akan semakin menurun. Selain itu, semakin besar *width*, akurasi model akan semakin meningkat.

Pada pengujian aktivasi, ada beberapa fungsi yang berhasil dan ada beberapa yang gagal. Fungsi yang berhasil adalah fungsi sigmoid dan tanh. Sedangkan, relu mengalami error atau anomali dan fungsi lainnya mengalami error overflow. Dari hasil sigmoid dan tanh, terlihat bahwa untuk distribusi bobot sigmoid lebih tersebar (terdistribusi) dan nilai frekuensinya lebih sedikit dibandingkan dengan tanh. Untuk distribusi gradien bobot, tanh lebih tersebar (terdistribusi) dan nilai frekuensinya lebih kecil dibandingkan dengan sigmoid.

Pada pengujian variasi *learning rate*, model dengan *learning rate* yang lebih kecil menunjukkan tingkat akurasi yang paling baik. Model dengan *learning rate* yang lebih besar kurang handal dalam memprediksi karena adanya osilasi dan sulit untuk mencapai minimum loss. Model dengan learning rate (0.1) cukup baik tetapi masih sedikit tidak stabil dibandingkan yang lebih kecil.

Pada variasi weight initializer, berdasarkan konfigurasi yang digunakan, jika menggunakan zero terlihat memiliki nilai inisialisasi, maka terlihat memiliki nilai loss yang rendah. Namun hal ini tidak disebabkan oleh akurasi yang tinggi. Malahan, akurasi menggunakan initializer ini adalah yang paling rendah. Kemungkinan penyebabnya adalah penggunaan one hot encoding yang banyak menghasilkan nilai nol. Salah satu cara untuk mengatasi masalah ini adalah menggunakan normalizer, seperti softmax pada output layer.

Selain itu, untuk tiga initializer lain selain uniform, memiliki nilai akurasi yang relatif dekat. Hanya terdapat perbedaan pada uniform yang memiliki nilai akurasi yang lebih tinggi. Terlihat bahwa nilai loss menggunakan initializer uniform memiliki bentuk yang stabil. Hal ini mungkin dikarenakan initializer ini memberikan bobot secara random tanpa membuat asumsi terhadap jenis permasalahan yang diberikan. Berbeda dengan normal misalnya, initializer ini beranggapan bahwa terdapat sebuah titik pusat dari distribusi bobot.

Pada perbandingan dengan sklearn, waktu yang dibutuhkan oleh sklearn lebih lama, mungkin karena kompleksitas implementasi yang lebih rumit. Namun hal ini membuat hasil akurasi menggunakan sklearn menjadi lebih baik dibandingkan dengan hasil implementasi kita. Kemungkinan lain adalah terdapat hyperparameter lain yang tidak bisa diset pada model sklearn, seperti weight initializer.

## Saran

Penulis menyarankan untuk memperhatikan nilai desimal dengan baik. Beberapa fungsi aktivasi yang menggunakan desimal yang rumit sehingga merusak komputasi. Hal ini disebabkan adanya overflow akibat nilai desimal tersebut.

Penulis menyarankan untuk menggunakan learning rate yang sesuai dengan kasus yang ingin diselesaikan. Learning rate yang lebih kecil dapat menghindari osilasi sehingga dapat mencapai minimum loss dengan lebih stabil. Namun, model dengan *learning rate* yang lebih kecil membutuhkan jumlah epoch yang lebih banyak. Nilai *learning rate* yang lebih besar cenderung mengakibatkan osilasi sehingga model sulit mencapai minimum loss.

# 

# **Pembagian Tugas**

| Nama | NIM | Tugas |
| --- | --- | --- |
| Benjamin Sihombing | 13522054 | Dokumen bagian deskripsi kelas, pengujian depth dan width, dan pengujian aktivasi.  Program bagian kelas aktivasi, fungsi plotting distribusi, dan fungsi visualisasi graf. |
| M. Atpur Rafif | 13522086 | Dokumen bagian forward, backward, analisis softmax, pengujian inisialisasi bobot, dan perbandingan dengan sklearn.  Program bagian fungsi forward, fungsi backward, dan kelas layer. |
| Suthasoma M. Munthe | 13522098 | Pengujian dan dokumen bagian variasi *learning rate*, *save* dan *load, plot training* dan *validation loss*,kelas *loss function*, kelas FFNN, dan kelas *initializer*. |

# 

# **Referensi**

* [The spelled-out intro to neural networks and backpropagation: building micrograd](https://www.youtube.com/watch?v=VMj-3S1tku0)
* <https://www.jasonosajima.com/forwardprop>
* <https://www.jasonosajima.com/backprop>
* <https://numpy.org/doc/2.2/>
* <https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html>