# TUGAS BESAR 1
# IF3270 PEMBELAJARAN MESIN

Feedforward Neural Network


Disusun oleh:

| | |
|---|---|
| Dewantoro Triatmojo | 13522011 |
| Moh Fairuz Alauddin Yahya | 13522057 |
| Rayhan Fadhlan Azka | 13522095 |

**Program Studi Teknik Informatika**
**Sekolah Teknik Elektro dan Informatika**
**Institut Teknologi Bandung**

**2024**

# DESKRIPSI PERSOALAN

Feed Forward Neural Network (FFNN) adalah salah satu jenis model di pembelajaran mesin yang terdiri dari lapisan-lapisan neuron yang tersusun secara berurutan, di mana informasi mengalir dari input layer ke hidden layer, dan akhirnya ke output layer, dimana setiap neuron di suatu layer terhubung ke seluruh neuron di layer berikutnya. Di masing-masing neuron terdapat weight yang dapat diubah dengan melakukan training dengan algoritma backpropagation yang meminimalkan kesalahan prediksi dengan memperbarui weight berdasarkan perbedaan antara keluaran yang diharapkan dan keluaran yang dihasilkan oleh jaringan.

Pada tugas besar kali ini kami mengimplementasikan FFNN from scratch, yaitu hanya menggunakan library untuk perhitungan. Model FFNN yang kami buat memiliki ketentuan sebagai berikut :

- FFNN dapat menerima jumlah neuron dari tiap layer.
- FFNN dapat menerima fungsi aktivasi dari tiap layer, meliputi :
    - Linear
    - ReLU
    - Sigmoid
    - Hyperbolic Tangent (tanh)
    - Softmax
    - Leaky ReLU
    - ELU
- FFNN yang diimplementasikan dapat menerima fungsi loss dari model tersebut, meliputi:
    - Mean Squared Error
    - Binary Cross-Entropy
    - Categorical Cross-Entropy
- Terdapat metode untuk inisialisasi bobot neuron, meliputi :
    - Zero initialization
    - Random dengan distribusi uniform.
    - Random dengan distribusi normal.
- Model dapat menerima parameter berikut:
    - Batch size
    - Learning rate
    - Jumlah epoch
    - Verbose

# PEMBAHASAN

## 1. Penjelasan Implementasi

### A. Deskripsi Kelas

#### 1. Kelas Activation

Kelas ini merupakan *abstract class* yang menjadi base untuk seluruh fungsi aktivasi

- Method:
  - forward(x):
    - *Abstract method* yang mengimplementasikan fungsi aktivasi, dimana parameter x merepresentasikan input dari fungsi aktivasi, yaitu hasil komputasi $\Sigma w.x$ dari *layer* sebelumnya.
  - backward(x, grad_output):
    - *Abstract method* yang mengimplementasikan perhitungan gradien secara *backward*. Method ini menghitung nilai $\partial L/\partial x$
    - x: Input asli dari fungsi aktivasi
    - grad_output: Gradient *layer* berikutnya $\partial L/\partial y$ atau $\partial L/\partial f(x)$
- Subclass
  a. Linear
     Method:
     - Forward: $f(x) = x$
     - Backward: $\partial f(x)/\partial x = 1$
  b. ReLU (Rectified Linear Unit)
     Method:
     - Forward: $f(x) = max(0, x)$
     - Backward: $\partial f(x)/\partial x = 1\ if\ x > 0,\ else\ 0$
  c. Sigmoid (Tangent Hiperbolik)
     Method:
     - Forward: $f(x) = 1/(1 + e^{-x})$
     - Backward: $\partial f(x)/\partial x = f(x)\cdot(1 - f(x))$
     - Pada nilai x digunakan clipping untuk menghindari overflow dengan boundary [-500, 500]
  d. Tanh
     Method:
     - Forward: $f(x) = tanh(x)$
     - Backward: $\partial f(x)/\partial x = 1 - tanh^2(x)$

e. Softmax
Method:

- Forward: $f(x) = e^{x_i - max(x)} / \sum_j e^{x_j - max(x)}$
- Backward: $\partial f(x)/\partial x = fi(\delta_{ij} - f_j)$
- Pada kode ini digunakan evaluasi pada nilai x dengan *substract* terhadap nilai maximumnya untuk menghindari overflow

f. LeakyReLU
Method:

- Forward: $f(x) = max(\alpha x, x)$
- Backward: $\partial f(x)/\partial x = 1\ if\ x > 0,\ else\ \alpha$

g. ELU (Exponential Linear Unit - Bonus)
Method:

- Forward: $f(x) = x\ if\ x > 0\ else\ \alpha(e^x - 1)$
- Backward: $\partial f(x)/\partial x = 1\ if\ x > 0\ else\ \alpha e^x$

## 2. Kelas Loss

Kelas ini merupakan *abstract class* yang menjadi *base* untuk semua fungsi loss

- Methods:
  - forward(y_pred, y_true):
    - *Abstract method* untuk menghitung nilai loss antara prediksi model (y_pred) dan target sebenarnya (y_true).
  - backward(y_pred, y_true):
    - *Abstract method* untuk menghitung gradien loss terhadap prediksi $\partial L/\partial y_{pred}$ yang digunakan untuk memulai proses *backpropagation*.
- Subclasses
  - MSE (Mean Squared Error)
    Method:
    - Forward: $L(y_{true}, y_{pred}) = 1/n. \sum_{i=1}^{n} (y_{true(i)} - y_{pred(i)})^2$
    - Backward: $\partial L/\partial y_{pred(i)} = 2/n\,.(y_{pred(i)} - y_{true(i)})^2$
  - BinaryCrossEntropy
    Method
    - Forward:

$$L(y_{true}, y_{pred}) = -1/n. \sum_{i=1}^{n} (y_{true(i)} log(y_{pred(i)}) - (1 - y_{true(i)}) log(1 - y_{pred(i)}))$$

- Backward: $\partial L/\partial y_{pred(i)} = -1/n\left[\frac{y_{true(i)}}{y_{pred(i)}} - \frac{1-y_{true(i)}}{1-y_{pred(i)}}\right]$
- Pada bagian ini digunakan clipping untuk menghindari division by zero pada y_pred dengan *boundary* [epsilon, 1 - epsilon]. Dengan nilai epsilon dapat diatur, tetapi by default adalah $1.10^{-8}$

- CategoricalCrossEntropy
  Method
  - Forward: $L(y_{true}, y_{pred}) = -1/n. \sum_{i=1}^{n}\sum_{j=1}^{C} y_{true\ ij} log(y_{pred\ ij})$
  - Backward: $\partial L/\partial y_{pred(i)} = -1/n\left[\frac{y_{true(ij)}}{y_{pred(ij)}}\right]$
  - Pada bagian ini digunakan clipping untuk menghindari division by zero pada y_pred dengan *boundary* [epsilon, 1 - epsilon]. Dengan nilai epsilon dapat diatur, tetapi by default adalah $1.10^{-8}$

## 3. Kelas Initializer

Kelas ini adalah *abstract class* yang menjadi *base* untuk semua *method* inisialisasi *weight.*

Methods:

- initialize(shape)
  - *Abstract method* untuk membuat dan menginisialisasi array *weight* dengan dimensi tertentu sesuai *shape*.

- Subclasses:
  a. ZeroInitializer
     Method initialize melakukan assignment $w = 0$
  b. UniformInitializer
     Method initialize melakukan assignment $w \sim U(low, high)$
  c. NormalInitializer
     Method initialize melakukan assignment $w \sim N(\mu, \sigma^2)$ . Dengan μ adalah *mean* dan σ adalah *std*
  d. XavierInitializer (Bonus)
     Method initialize melakukan assignment $w \sim U(-\sqrt{6/n_{in} + n_{out}}, \sqrt{6/n_{in} + n_{out}})$. Dengan nin adalah jumlah neuron input dan nout adalah jumlah neuron output.
  e. HeInitializer (Bonus)

## 4. Kelas Regularizer

Kelas Regularizer adalah *abstract class* untuk semua regularisasi yang digunakan untuk mencegah overfitting dalam model.

- Methods:
  - loss(weights):
    - *Abstract method* untuk menghitung regularisasi terhadap fungsi loss total ( $R(w)$)
  - gradient(weights):
    - *Abstract method* untuk menghitung regularisasi terhadap gradien weight ( $\partial R/\partial w$)
- Subclasses:
  a. NoRegularizer
     Method:
     - loss: $R(w) = 0$
     - gradient: $\partial R/\partial w = 0$
  b. L1Regularizer (Bonus)
     Method:
     - loss: $R(w) = \lambda \sum_i \left| w_i \right|$
     - gradient: $\partial R/\partial w = \lambda \, sign(w)$
  c. L2Regularizer (Bonus)
     Method:
     - loss: $R(w) = \lambda \sum_i w_i^2$
     - gradient: $\partial R/\partial w = 2\lambda \, w$

## 5. Kelas Normalization

Kelas Normalization adalah a*bstract class untuk* teknik normalisasi untuk *stabilization* dan mempercepat *training*.

- Methods:
  - forward(x):
    - *Abstract method* yang mengimplementasikan normalisasi pada input.
  - backward(x, grad_output):
    - *Abstract method* yang menghitung gradien balik melalui normalisasi.
- Subclasses:
  a. NoNormalization
     Method:
     - Forward: $f(x) = x$

- Backward: $\partial f(x)/\partial x = 1$

b. RMSNorm (Root Mean Square Normalization - Bonus)
Method:

- Forward: $f(x) = \frac{x}{RMS}$
- Backward: $\partial f(x)/\partial x = 1/RMS - x\Sigma x/d\, RMS^3$
- dimana d adalah dimensi input dan ϵ adalah parameter agar tidak division by zero dan RMS = $\sqrt{1/d \sum_{i=1}^{d} x_i^2 + \epsilon}$

## 6. Kelas Layer

Kelas ini merepresentasikan *layer* perceptron yang *fully connected* dimana tiap neuron menerima input dari semua neuron pada *layer* sebelumnya

- Parameter:
  - input_size: Jumlah neuron pada layer sebelumnya (dimensi input)
  - output_size: Jumlah neuron pada layer ini (dimensi output)
  - activation: Fungsi aktivasi yang akan digunakan (default: Linear)
  - weight_initializer: Metode inisialisasi untuk weight (default: HeInitializer)
  - bias_initializer: Metode inisialisasi untuk bias (default: ZeroInitializer)
  - regularizer: Metode regularisasi untuk weight (default: NoRegularizer)
  - normalization: Metode normalisasi (default: NoNormalization)
- Atribut:
  - weights: Matriks bobot berukuran (input_size, output_size)
  - biases: Vektor bias berukuran (output_size)
  - weights_grad: Gradien untuk matriks bobot
  - biases_grad: Gradien untuk vektor bias
  - input: Menyimpan input untuk digunakan saat backpropagation
  - output_before_activation: Output sebelum penerapan fungsi aktivasi
  - output: Output akhir setelah fungsi aktivasi
  - weight_momentum dan bias_momentum: *Velocity decay* untuk learning rate
- Method:
  - Forward
    Melakukan proses forward propagation berdasarkan step berikut
    - Linear transformation
      Melakukan komputasi terhadap input $Z = wx + b$, dimana $Z$ adalah output linear, x adalah input berukuran (batch_size, input_size), w adalah matriks *weight* berukuran (input_size, output_size), dan b adalah vektor biars berukuran output size

- Normalisasi

  Menormalisasi hasil linear transformation dengan $Z' = norm(Z)$
- Aktivasi

  Menjalankan fungsi aktivasi terhadap $Z'$ dengan $A = f(Z')$ dimana nilai aturan fungsi f bergantung pada activation parameter

- Backward

  Melakukan proses backward propagation dengan menggunakan *chain rule* dalam beberapa step berikut
  - Gradien Activation
    - Mengkalkulasi nilai gradien loss terhadap Z' dengan $\partial L/\partial Z' = \partial L/\partial A \,.\, \partial A/\partial Z'$
    - Pada kode kita $\partial L/\partial A$ direpresentasikan oleh gradient output dari *layer* berikutnya
  - Gradien Normalisasi
    - Normalisasi menerapkan chain rule untuk menghitung $\partial L/\partial Z = \partial L/\partial Z' \,.\, \partial Z'/\partial Z$
    - Pada kode kita $\partial L/\partial Z$ direpresentasikan oleh grad_before_norm
  - Gradien Weight untuk update nilai weight
    - Untuk menghitung $\partial L/\partial w$. Perlu tahu bahwa Z = x·w + b, maka $\partial Z/\partial w = x^T$ (transpose dari x)
    - Sehingga dengan menggunakan chain rule: $\partial Z/\partial w = x^T \,.\, \partial L/\partial Z$
    - Lalu dibagi dengan batch_size sebagai implementasi dari mini batch gradient descent
    - Dan ditambahkan gradien dari regularisasi $\frac{\partial R(W)}{\partial W}$, jika ada
  - Gradien Bias untuk update nilai bias
    - Untuk menghitung $\partial L/\partial b$. Perlu tahu bahwa Z = x·w + b, maka $\partial L/\partial b = 1$
    - Sehingga dengan chain rule: $\partial L/\partial b = \partial L/\partial Z \cdot \partial Z/\partial b = \partial L/\partial Z$
  - Gradien Input untuk grad_output untuk *layer sebelumnya*
    - Untuk menghitung $\partial L/\partial x$. Perlu tahu bahwa Z = x·w + b, maka $\partial L/\partial b = w^T$
    - Sehingga dengan menggunakan chain rule: $\partial L/\partial x = \partial L/\partial Z \cdot \partial Z/\partial x = \partial L/\partial Z \cdot w^T$
- Weight Update

  Melakukan update weight dilakukan dalam beberapa step berikut
  - Penggunaan momentum (optional)

Terinspirasi dari scikit menggunakan adam untuk optimizernya dengan define velocity decay berdasarkan parameter momentum nya β, dengan formula sebagai berikut

$$v_t = \beta v_{t-1} + (1 - \beta).\nabla w_t$$

- Penggunaan RMSProp (Root Mean Square Propagation) u
  Untuk optimizer nya juga dengan define decay_rate γ, dengan formula sebagai berikut

$$v_t = \gamma \cdot v_{t-1} + (1 - \gamma) \cdot (\nabla w_t)^2$$

- Update Final:
  Tinggal update weight $w_{t+1} = w_t - \eta \cdot \Delta w$

## 7. Kelas FFNN

Kelas FFNN adalah implementasi utama model Feed Forward Neural Network

- Parameter
  - layer_sizes: List ukuran setiap layer (termasuk input dan output)
  - activations: List fungsi aktivasi untuk setiap layer
  - loss: Fungsi loss untuk model (default: MSE)
- Atribut
  - layer_sizes: Menyimpan ukuran setiap layer
  - activations: Menyimpan fungsi aktivasi untuk setiap layer
  - loss_function: Fungsi loss yang digunakan
  - layers: List dari objek Layer (implementasi standar)
- Method:
  - forward(x):
    - Implementasi dengan iterasi tiap layer berurutan dan melakukan aktivasi di setiap layer
  - backward(x):
    - Hitung loss lakukan gradient backward dari loss dan backward dari layer secara terbalik
  - Fit
    Proses pada function ini dilakukan dalam beberapa tahap sebagai berikut
    - Iterasi pada epoch yang ditentukan, di tiap epoch akan dilakukan shuffling, process minibatch, forward, backward, update weight, dan training loss calculation
    - Jika early stopping condition tercapai maka stopping

## B. Penjelasan Forward Propagation

Telah dijelaskan secara detail pada bagian Kelas Layer method forward

## C. Penjelasan Backward Propagation dan Update weights

Telah dijelaskan secara detail pada bagian Kelas Layer method backward dan update weight

# 2. Hasil Pengujian

Seluruh pengujian yang kami lakukan menggunakan Feed Forward Neural Network dengan full dataset mnist_784, dengan pembagian train_test_split : train 0.8 dan test 0.2 .

### a. Pengaruh Depth dan Width

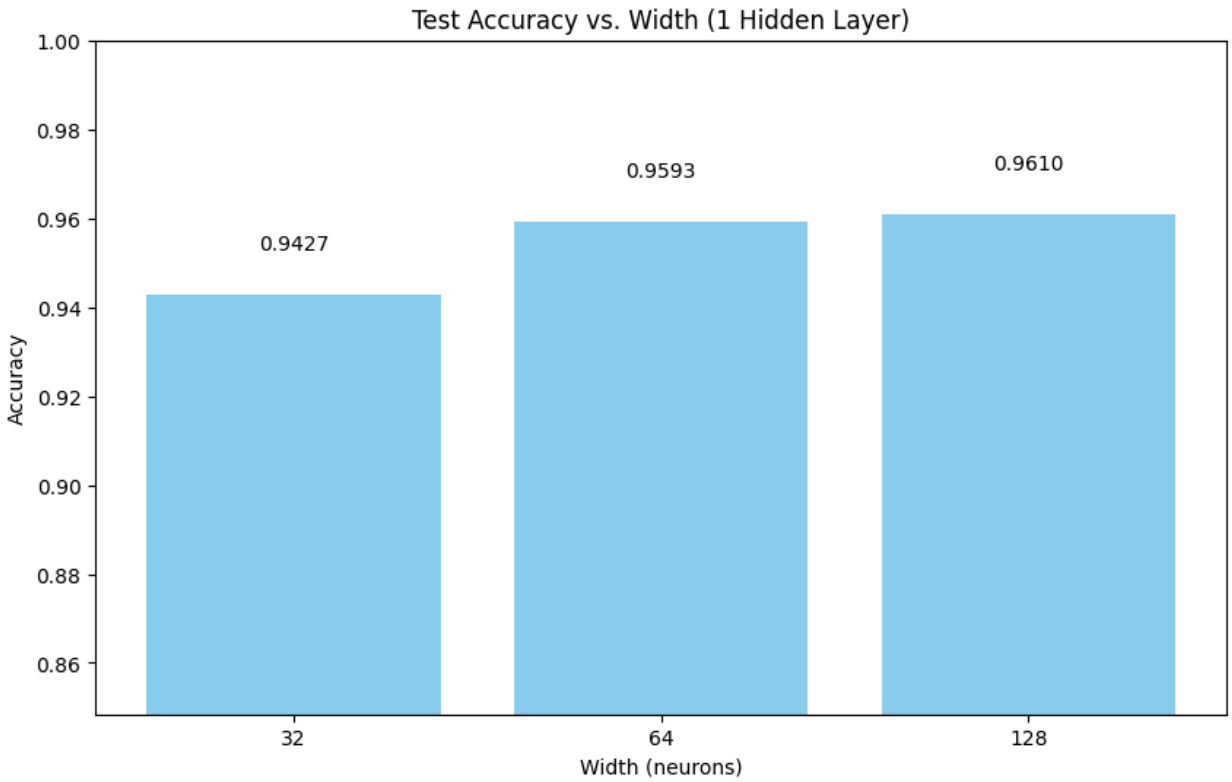
Kami melakukan uji coba dengan depth dan width yang berbeda dan beberapa parameter lain sama untuk masing masing depth dan width, dengan rincian parameter tersebut sebagai berikut :

- epochs = 10
- batch_size = 32
- learning_rate = 0.01

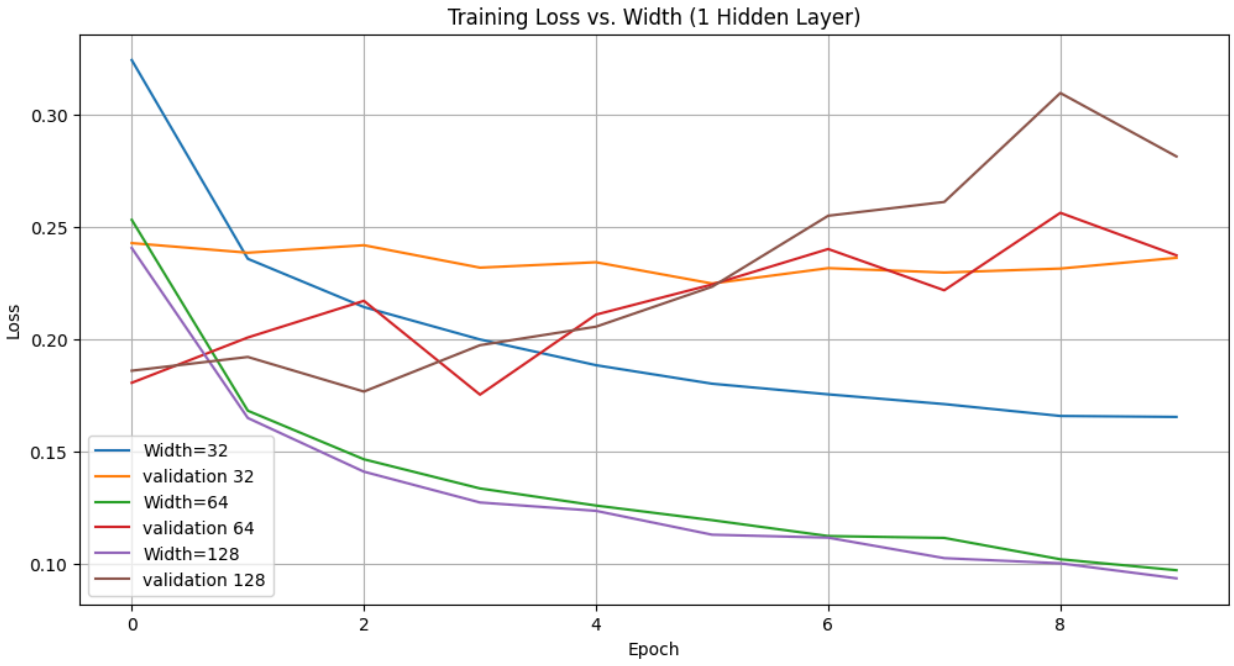
Percobaan dengan width yang berbeda dengan konfigurasi sebagai berikut :

- 1 input layer, 1 hidden layer, 1 output layer.
- Fungsi aktivasi hidden layer ReLU dan Fungsi aktivasi output layer Softmax
- Fungsi Loss dengan Categorical Cross-Entropy

Lalu kami gunakan variasi width pada hidden layer sebesar 32, 64, dan 128. Lalu kami mendapatkan hasil akurasi sebagai berikut.




Dan training perubahan training loss sebagai berikut :



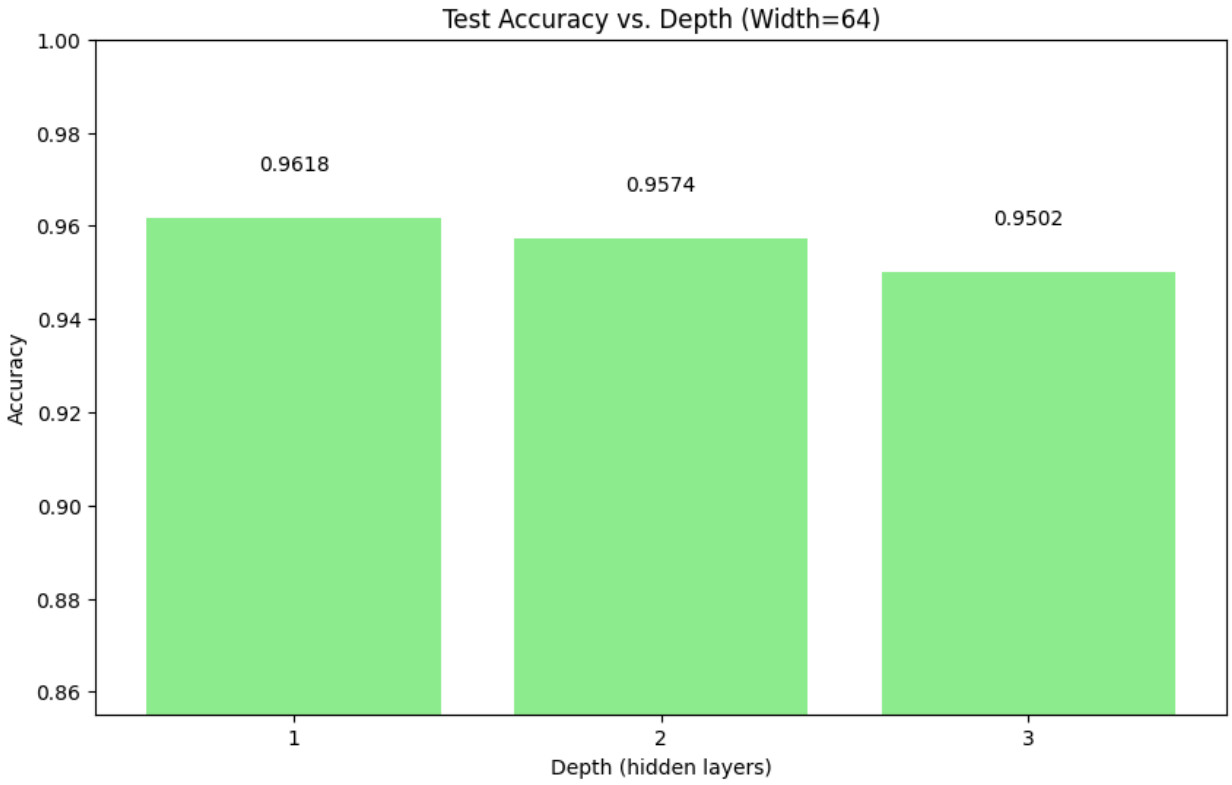

Dapat dilihat bahwa akurasi menjadi lebih baik seiring meningkatnya lebar dari tiap layer. Hal ini karena jika semakin banyak neuron yang terdapat di suatu layer, maka model

tersebut dapat menangkap lebih banyak fitur dari data. Selain itu, training loss yang didapat juga lebih kecil jika lebar dari layer meningkat. Hal ini karena jika lebih banyak neuron pada suatu layer, maka setiap neuron tersebut dapat dapat menyesuaikan bobotnya sehingga meminimalisir loss.

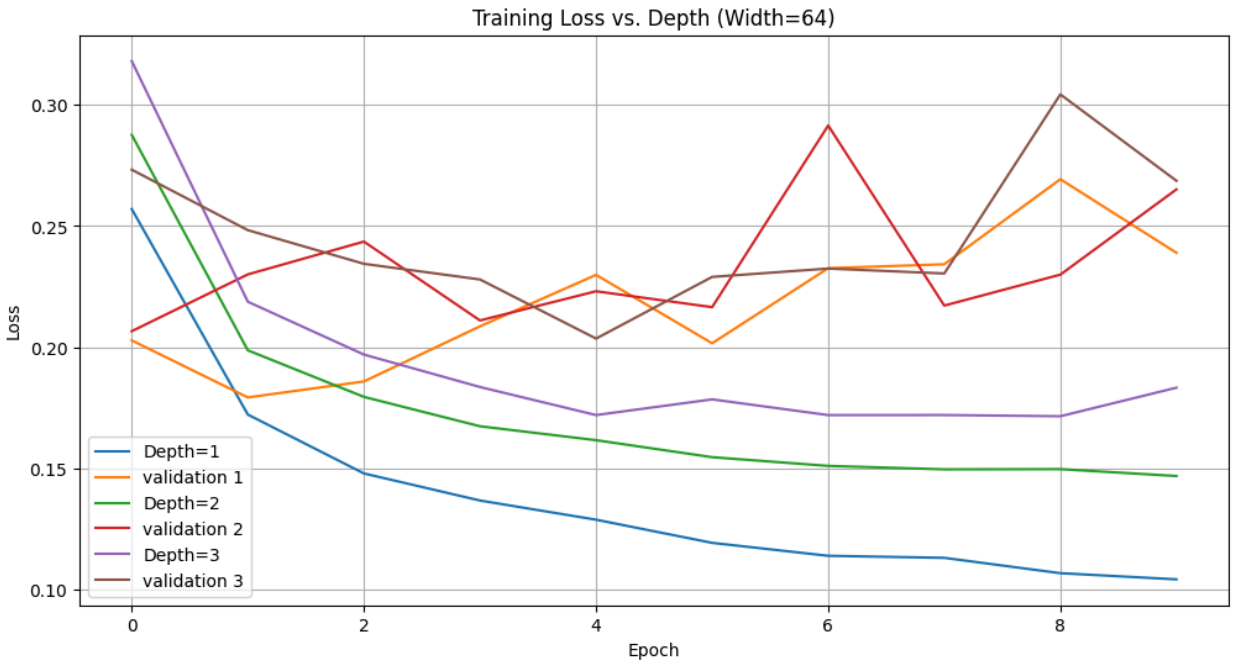
Lalu kami juga melakukan percobaan dengan variasi depth, yaitu dengan konfigurasi sebagai berikut :

- Untuk tiap hidden layer memiliki lebar 64
- Tiap hidden layer memiliki fungsi aktivasi ReLU
- Output layer memiliki fungsi aktivasi Softmax

Kami melakukan percobaan dengan variasi kedalaman 1, 2, dan 3 (1 hidden layer, 2 hidden layer, 3 hidden layer). Dari percobaan tersebut didapat akurasi :



Dan training loss :




Akurasi tertinggi didapatkan pada model dengan depth 1. Hal ini dapat disebabkan karena dataset MNIST merupakan dataset yang simpel, sehingga tidak membutuhkan arsitektur yang terlalu dalam untuk mengekstrak fitur-fiturnya.

Berdasarkan grafik tersebut, justru depth 1 menghasilkan training loss lebih rendah dan akurasi tertinggi, hal ini dapat terjadi karena model dengan kedalaman yang lebih tinggi bisa saja lebih sulit untuk dioptimasi dimana semakin banyak depth berarti semakin banyak parameter dan semakin sulit untung di *tune,* sehingga proses training menjadi kurang stabil. Hal ini juga dapat terjadi karena persamaan parameter seperti learning rate dan epochs di konfigurasi depth yang berbeda, seharusnya semakin dalam suatu jaringan, learning rate yang digunakan harus di *tune* menjadi lebih kecil untuk mencapai konvergen.

## b. Pengaruh Fungsi Aktivasi

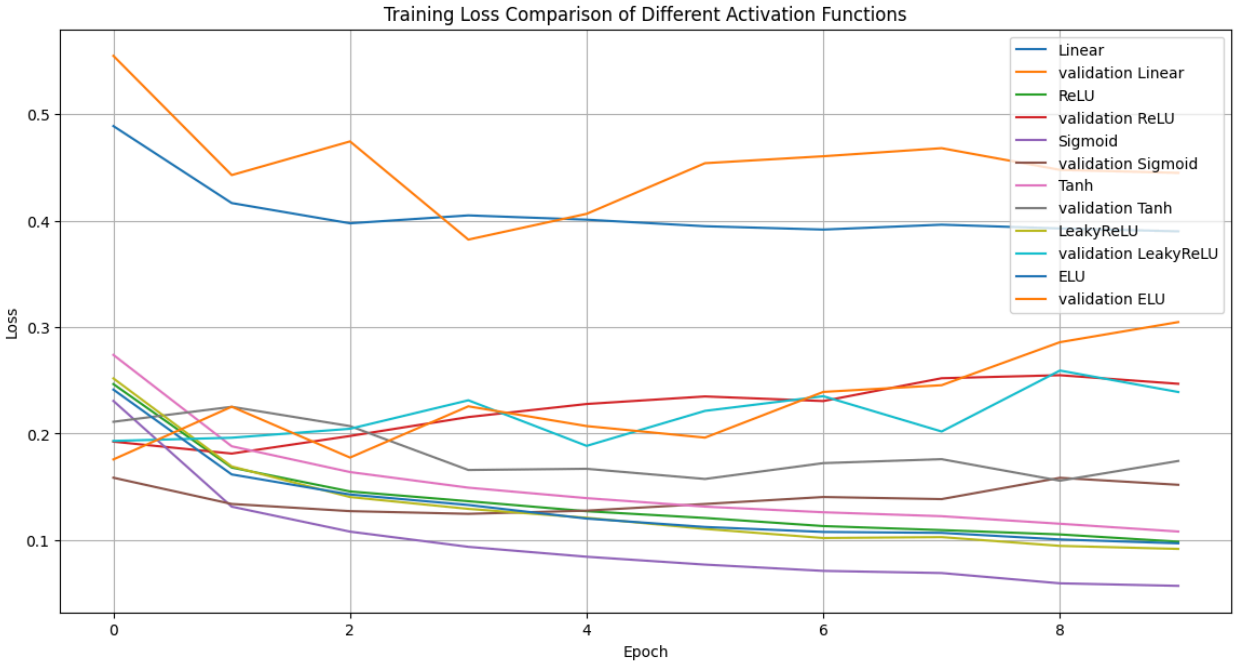
Percobaan ini menggunakan konfigurasi :

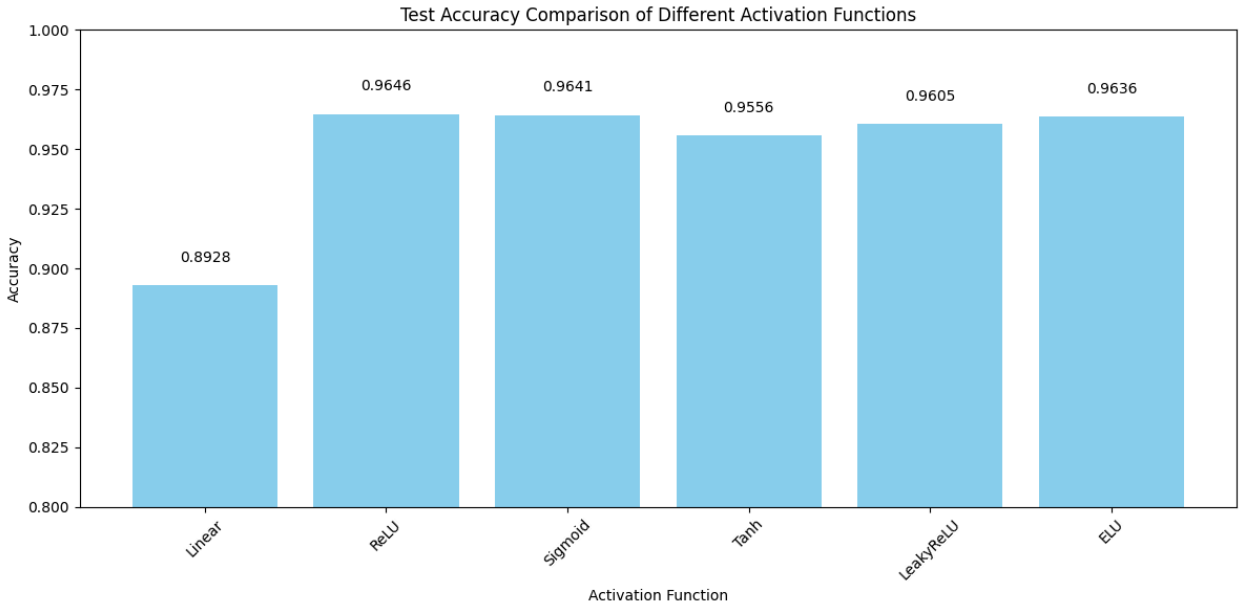
- hidden_size = 64
- epochs = 10
- batch_size = 32
- learning_rate = 0.01
- 1 hidden layer
- Loss Categorical Cross-Entropy

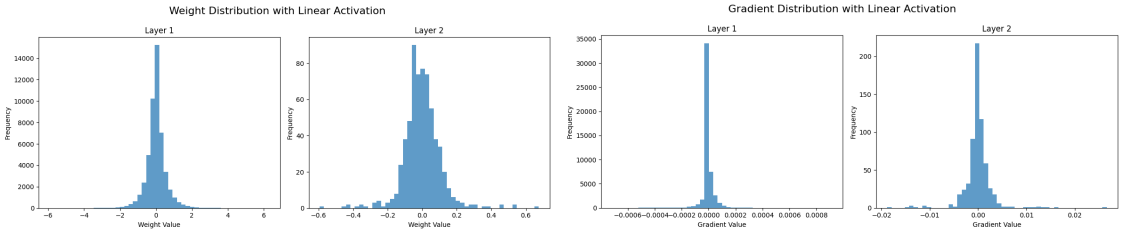
Dan variasi fungsi aktivasi :

- Linear
- ReLU
- Sigmoid
- Tanh
- LeakyReLU(alpha=0.01)
- ELU(alpha=1.0)

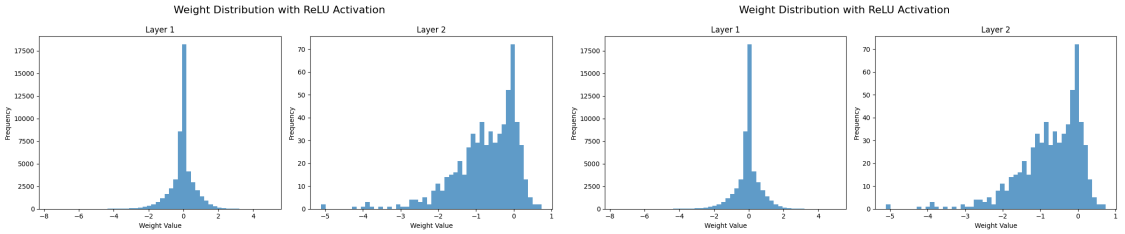
Variasi fungsi aktivasi tersebut diterapkan pada 1 hidden layer, dan output layer menggunakan fungsi aktivasi softmax. Dari hasil percobaan tersebut didapat :



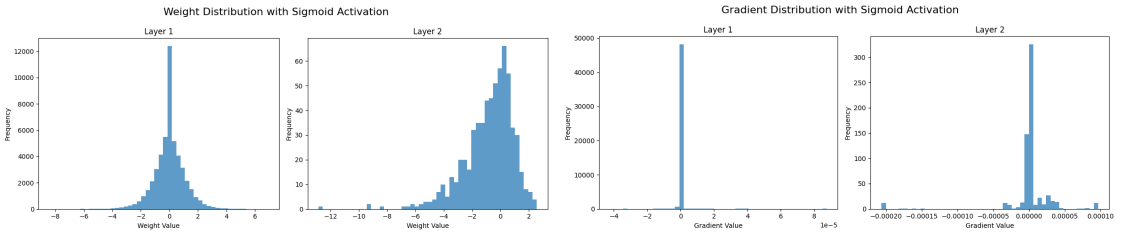

Test Accuracy Comparison of Different Activation Functions
Accuracy
Activation Function
Linear
ReLU
Sigmoid
Tanh
LeakyReLU
ELU
0.8928
0.9646
0.9641
0.9556
0.9605
0.9636


Weight Distribution with Linear Activation
Layer 1
Layer 2
Frequency
Weight Value

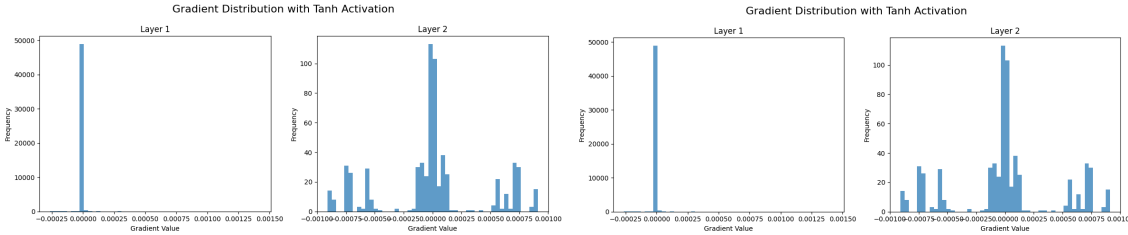

Gradient Distribution with Linear Activation
Layer 1
Layer 2
Frequency
Gradient Value


Weight Distribution with ReLU Activation
Layer 1
Layer 2
Frequency
Weight Value

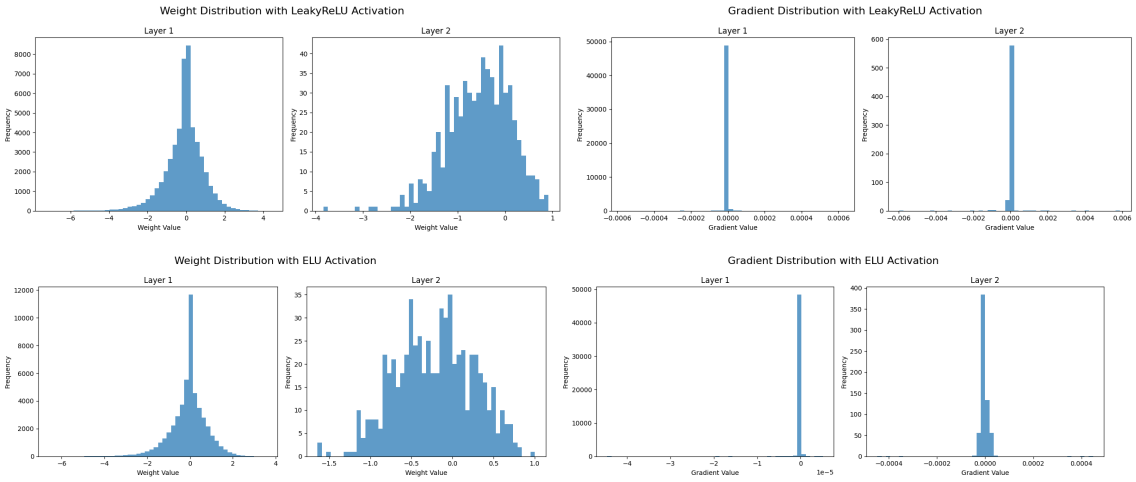

Weight Distribution with ReLU Activation
Layer 1
Layer 2
Frequency
Weight Value


Weight Distribution with Sigmoid Activation
Layer 1
Layer 2
Frequency
Weight Value


Gradient Distribution with Sigmoid Activation
Layer 1
Layer 2
Frequency
Gradient Value


Gradient Distribution with Tanh Activation
Layer 1
Layer 2
Frequency
Gradient Value


Gradient Distribution with Tanh Activation
Layer 1
Layer 2
Frequency
Gradient Value




Dari hasil tersebut terlihat bahwa fungsi aktivasi linear terlihat memiliki akurasi dan training loss yang sangat buruk, dan fungsi aktivasi sigmoid memiliki training loss terbaik dan ELU memiliki akurasi terbaik. Fungsi linear memiliki hasil yang sangat buruk karena tidak adanya *non-linearity*, sehingga model hanya berfungsi seperti kombinasi linear dari inputnya, dimana *non-linearity* justru adalah hal terpenting dari sebuah fungsi aktivasi. Tanpa adanya *non-linearity*, model tidak bisa menangkap pola dalam data, yang menyebabkan akurasi rendah dan training loss tinggi. Selain fungsi aktivasi linear, fungsi aktivasi lainnya memiliki hasil yang mirip, dan juga tiap bobot dan gradien terdistribusi mendekati nol.

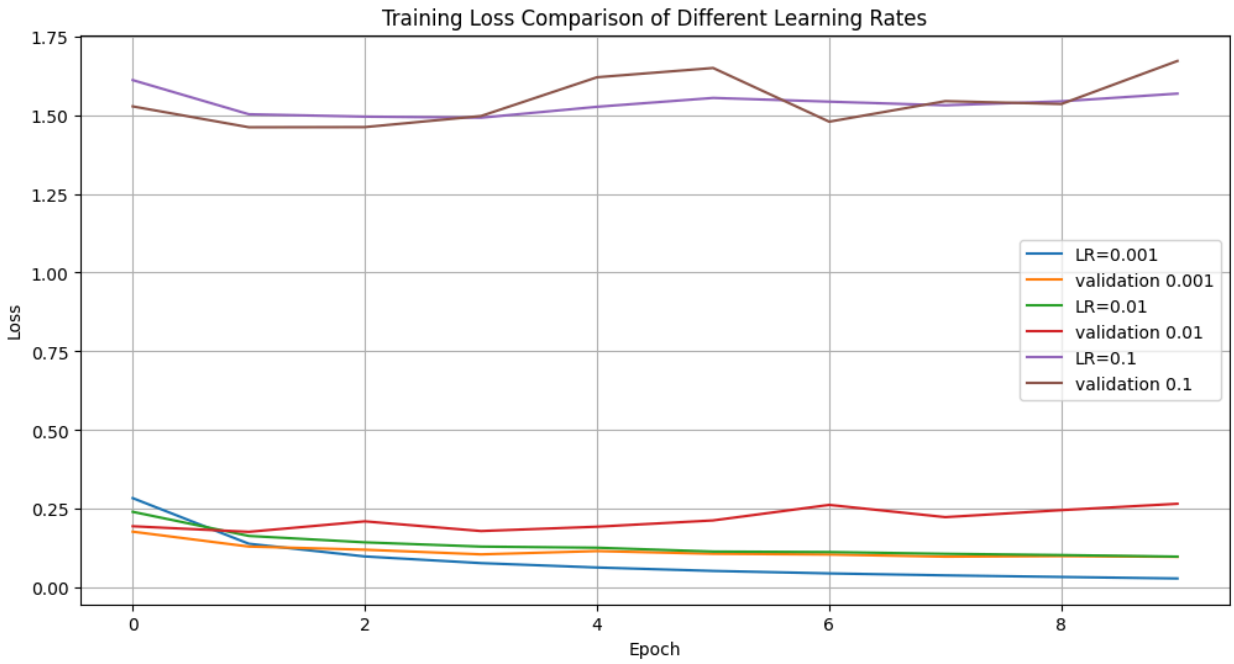
## c. Pengaruh Learning Rate

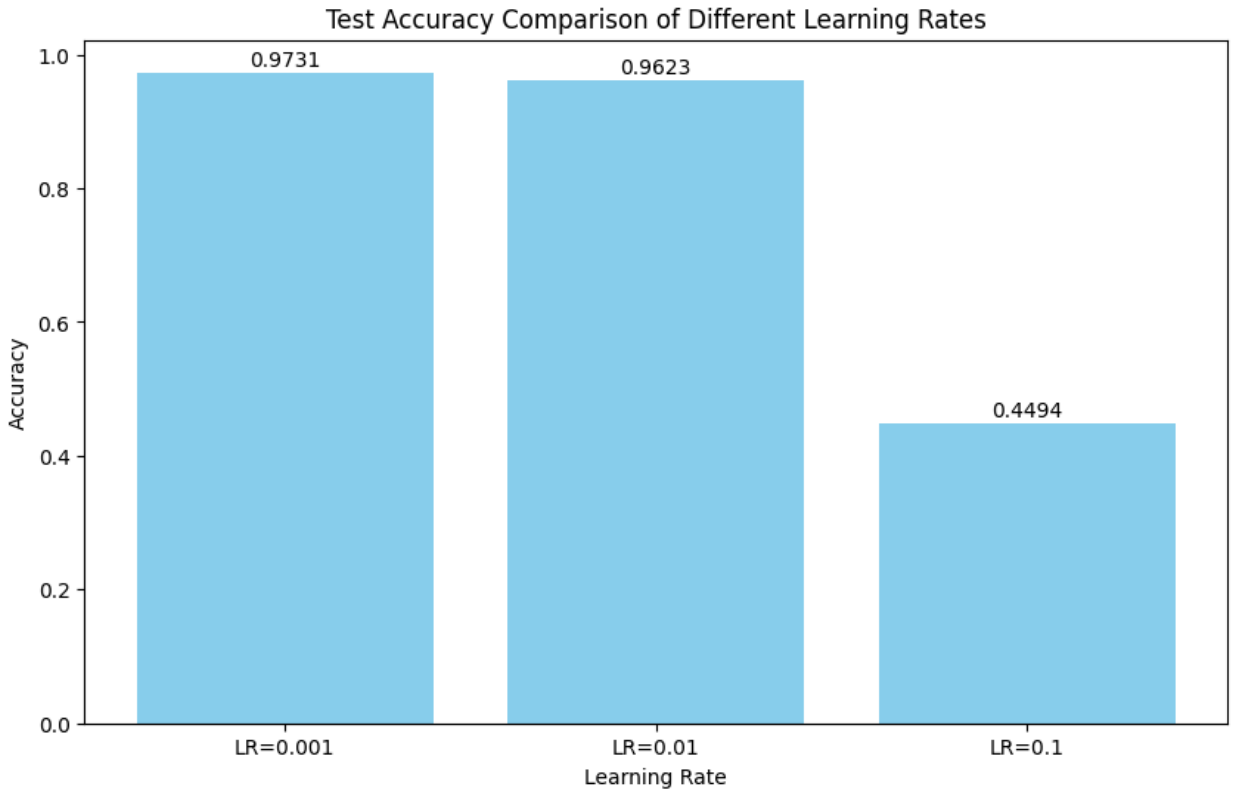
Percobaan ini menggunakan konfigurasi :

- hidden_size = 64
- epochs = 10
- batch_size = 32
- 1 hidden layer
- Loss Categorical Cross-Entropy
- Fungsi aktivasi Relu dan Softmax

Dengan variasi learning rate 0.001, 0.01, dan 0.1.

Didapat hasil pengujian sebagai berikut :




Kurangnya akurasi dan besarnya training loss pada learning rate 0.1 karena pada model tersebut bobot diperbarui dengan langkah yang terlalu besar, menyebabkan model melompat-lompat di sekitar solusi optimal tanpa benar-benar menemukannya, hal ini menyebabkan training loss tetap tinggi dan kegagalan *learning* pada model.

Learning rate 0.001 menghasilkan hasil paling optimal diantara ketiganya karena kecilnya learning rate, model update bobot secara lebih bertahap dan model lebih mudah mencapai konvergen.

## d. Pengaruh Inisialisasi Bobot

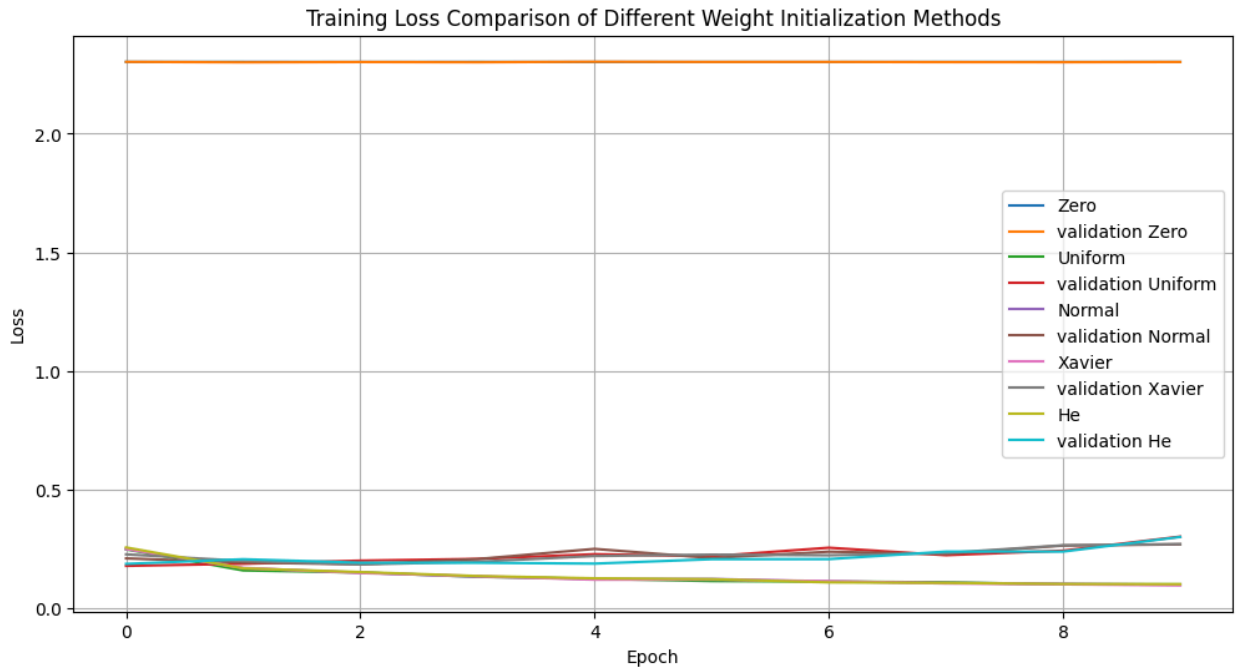
Percobaan ini menggunakan konfigurasi :

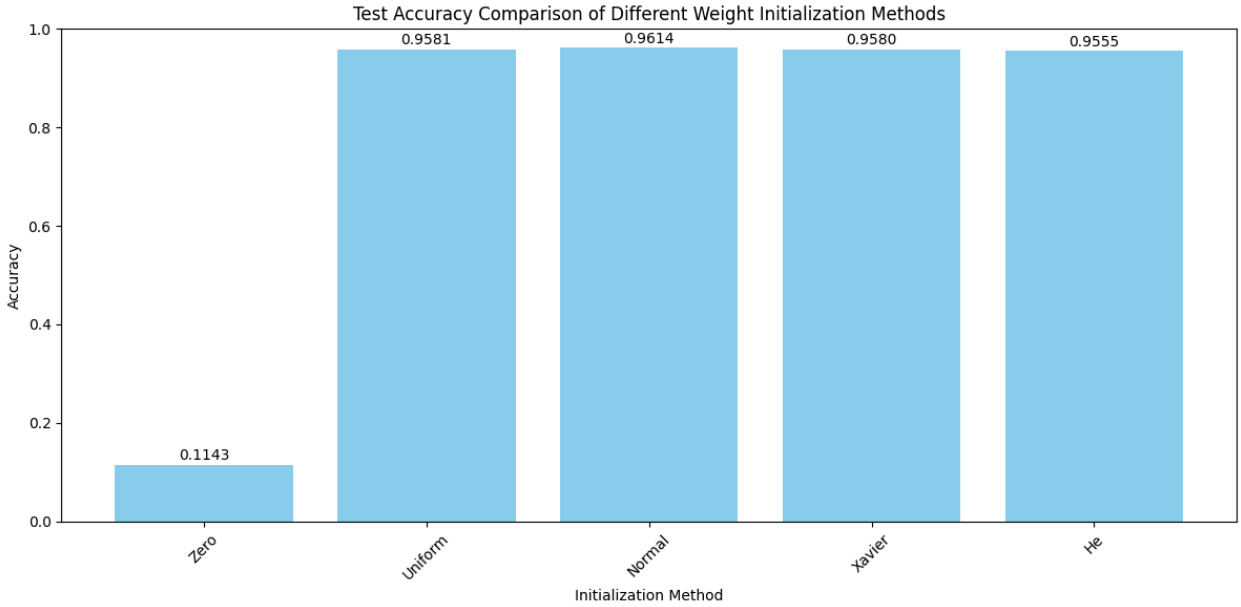
- hidden_size = 64
- epochs = 10
- batch_size = 32
- 1 hidden layer
- Loss Categorical Cross-Entropy
- Fungsi aktivasi Relu dan Softmax
- Learning rate 0.01

Dengan variasi inisialisasi bobot neuron :

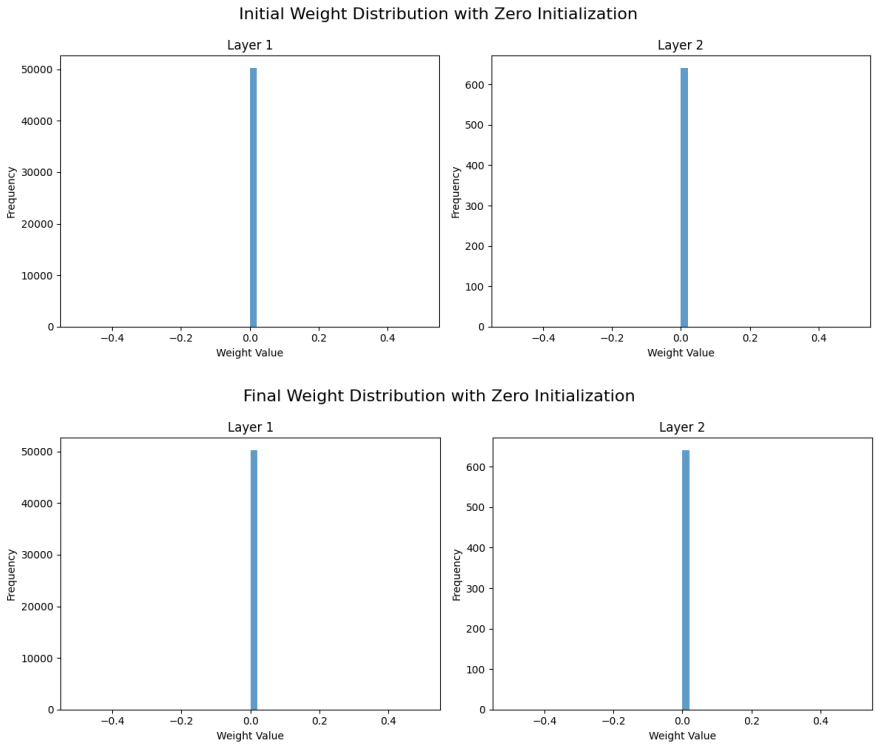
- Zero Initializer
- Uniform Initializer(low=-0.1, high=0.1, seed=42)
- Normal Initializer(mean=0.0, std=0.1, seed=42)
- Xavier Initializer(seed=42)
- He Initializer(seed=42)

Didapat hasil percobaan :




Dapat dilihat bahwa hampir semua inisialisasi memiliki hasil yang sama selain zero initialization yang hasilnya sangat buruk sendiri. Zero initialization memiliki training loss yang sangat buruk dan akurasi yang sangat rendah. Hal ini terjadi karena jika semua bobot diinisialisasi dengan nol, semua neuron dalam satu layer akan memiliki perhitungan yang sama selama backpropagation, sehingga mereka akan mendapatkan gradien yang sama. Akibatnya, semua neuron tetap simetris dan tidak belajar fitur yang berbeda, menyebabkan jaringan gagal melakukan klasifikasi dengan baik. Hal ini dibuktikan dengan distribusi bobot zero initialization yang seluruhnya bernilai 0.



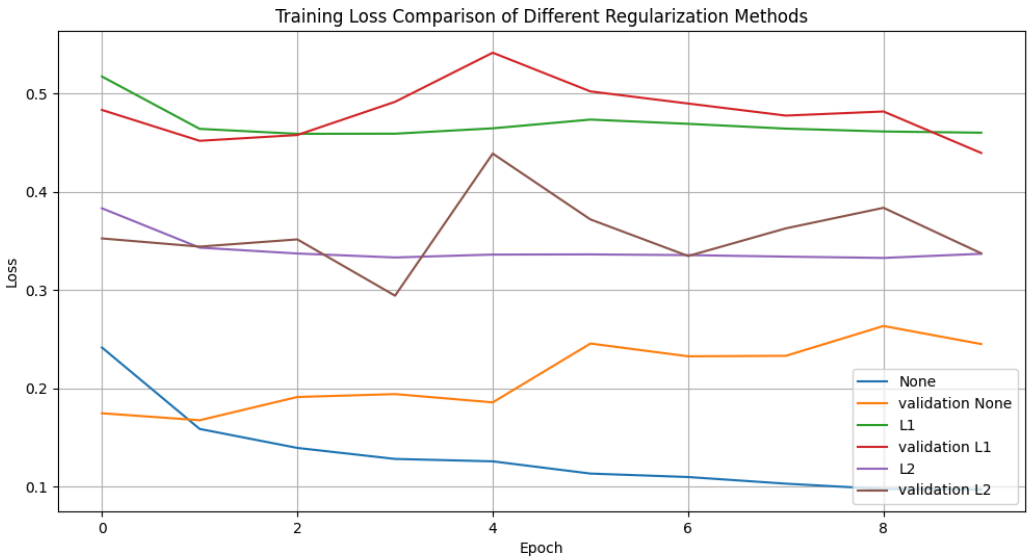

## e. Pengaruh Regularisasi

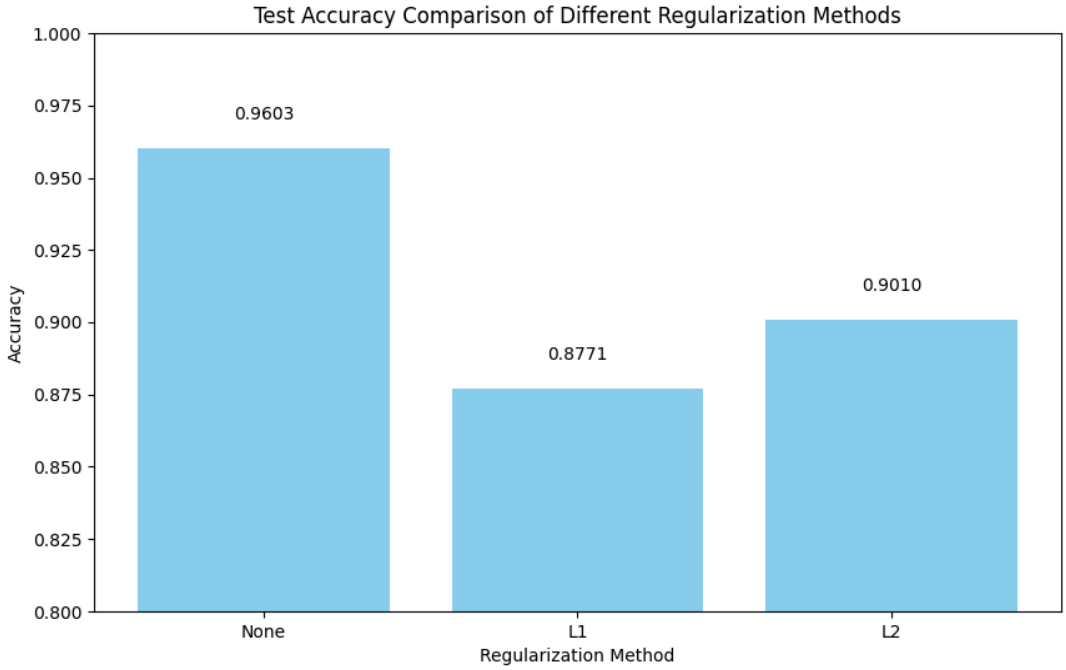
Percobaan ini menggunakan konfigurasi :

- hidden_size = 64
- epochs = 10
- batch_size = 32
- 1 hidden layer
- Loss Categorical Cross-Entropy
- Fungsi aktivasi Relu dan Softmax
- Learning rate 0.01

Variasi pada percobaan ini adalah perbedaan pada regularisasi, yaitu penggunaan No Regularizer, L1 Regularizer, L2 Regularizer.

Dari percobaan kami, didapat hasil sebagai berikut :




Dari eksperimen tersebut, tanpa regularisasi menghasilkan akurasi tertinggi dan training loss terendah. Regularisasi L1 dan L2 memiliki training loss yang lebih tinggi, karena mereka menambahkan penalti terhadap bobot besar sehingga model tidak bisa sepenuhnya menyesuaikan bobot dengan data training. L1 memiliki training loss tertinggi, karena cenderung membuat bobot ke nol, menyebabkan lebih banyak fitur diabaikan. Regularisasi berguna untuk mencegah overfitting, namun jika regularisasi terlalu kuat (seperti L1), malah justru membuat model underfit.

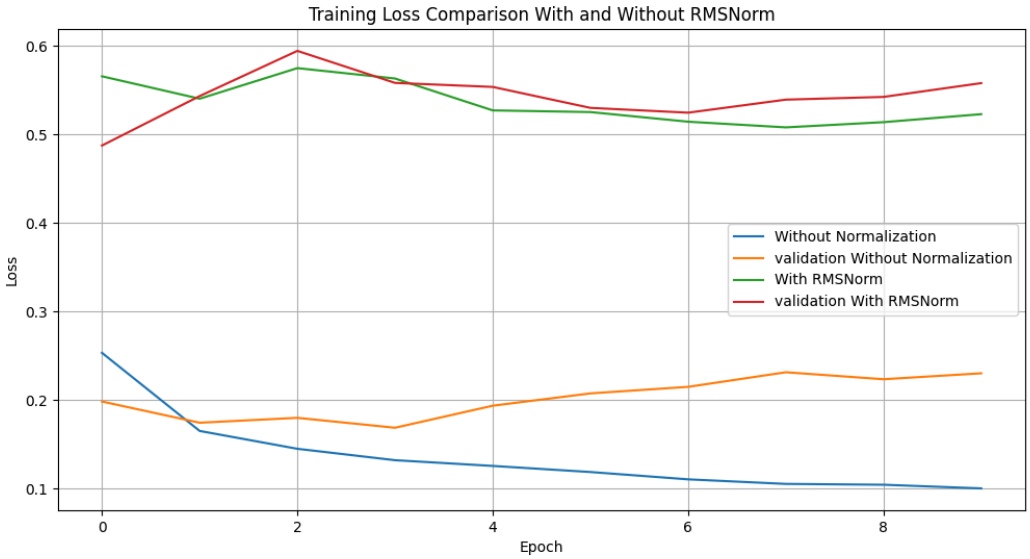
## f. Pengaruh Normalisasi RMSNorm

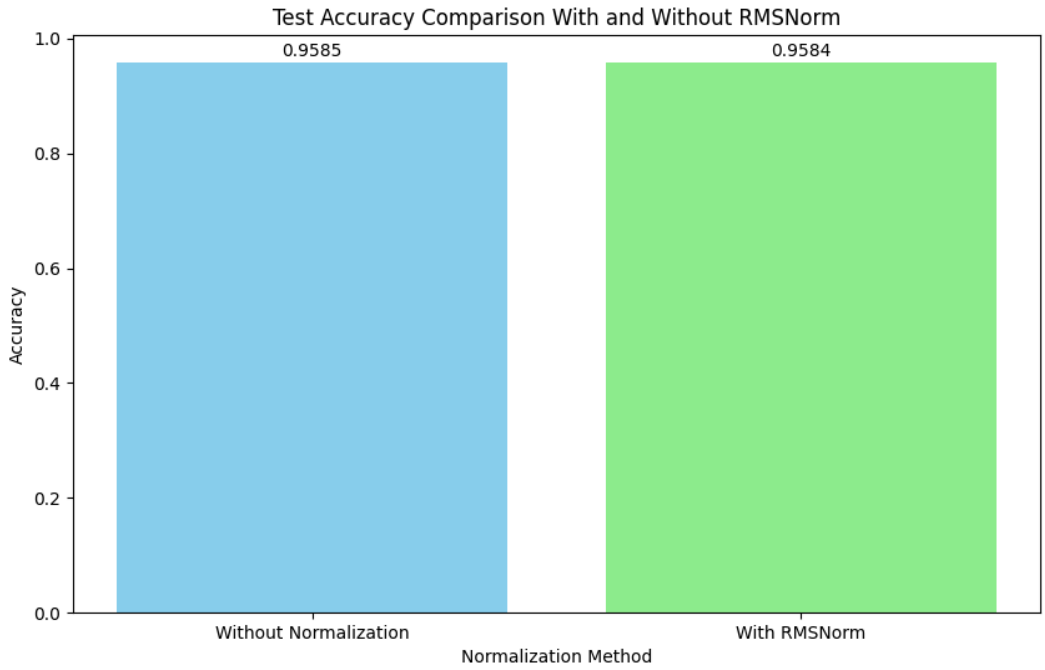
Percobaan ini menggunakan konfigurasi :

- hidden_size = 64
- epochs = 10
- batch_size = 32
- 1 hidden layer
- Loss Categorical Cross-Entropy
- Fungsi aktivasi Relu dan Softmax
- Learning rate 0.01

Percobaan ini menggunakan variasi penggunaan RMSNorm dan tidak.

Dari percobaan tersebut didapat hasil :

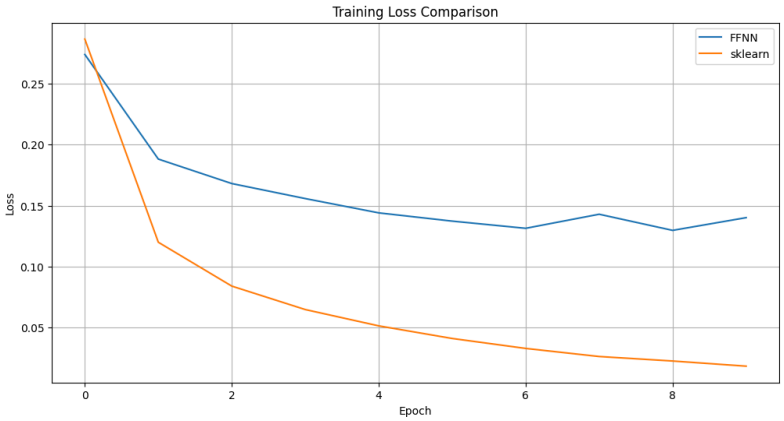


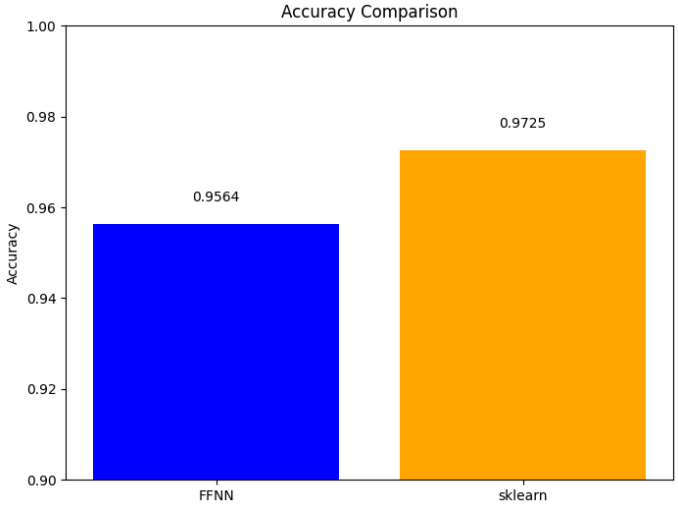


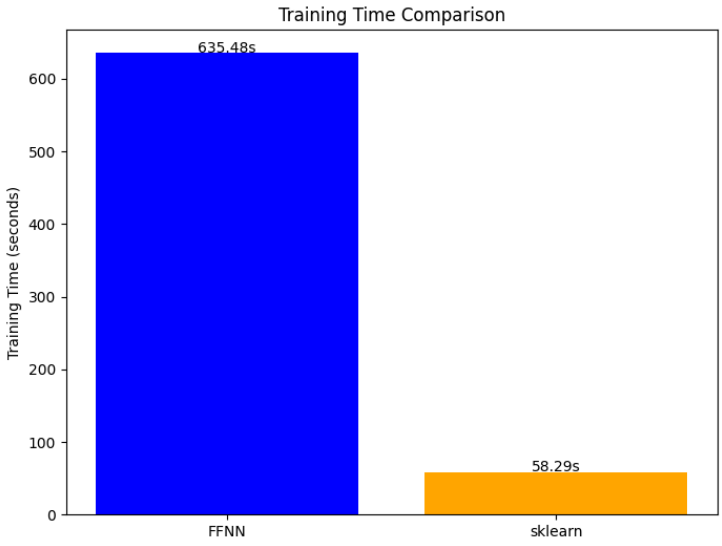

Grafik menunjukkan perbandingan antara model dengan dan tanpa RMSNorm (Root Mean Square Normalization). Dari hasil training loss, model tanpa normalisasi memiliki loss yang mirip dibandingkan dengan RMSNorm, tapi dalam beberapa kasus percobaan RMS malah mendapatkan hasil yang lebih buruk. Hal ini dikarenakan cara kerja RMSNorm yang mengubah distribusi aktivasi dengan faktor pembagi dengan akar kuadrat rata-rata, yang dapat menyebabkan perubahan drastis pada representasi fitur gradien, sehingga ketika tidak adanya penyesuaian learning rate yang tepat, algoritma bisa saja lebih buruk dalam memprediksi minimumnya.Terus pada dataset tertentu di mana fitur memiliki skala yang sudah relatif seimbang, menambahkan normalisasi justru dapat menghilangkan makna ataau representasi informasi penting. Lalu normalisasi juga bekerja dengan mengasumsikan sebuah distribusi data mengikuti *rule* nya, jadi bisa jadi datanya tidak sesuai asumsi.

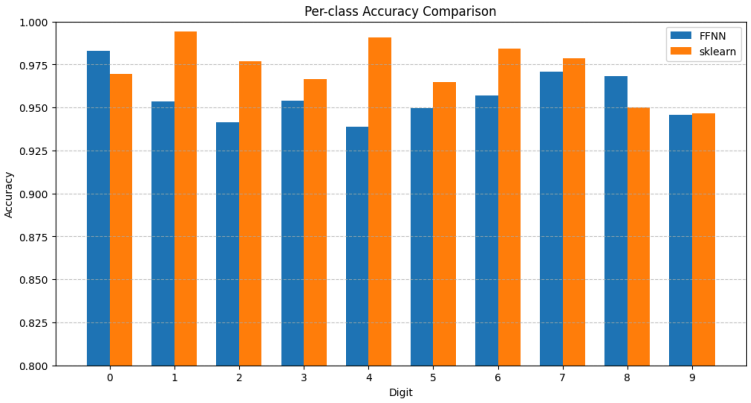
## g. Perbandingan dengan Library sklearn

Kami melakukan perbandingan model FFNN *from scratch* dengan model MLPClassifier dengan parameter yang sama. Dari hasil training didapat hasil sebagai berikut :


Training Loss Comparison
FFNN
sklearn
Loss
Epoch
0.25
0.20
0.15
0.10
0.05
0
2
4
6
8


Accuracy Comparison
Accuracy
1.00
0.98
0.96
0.94
0.92
0.90
0.9564
0.9725
FFNN
sklearn




Dari hasil yang didapat, performa FFNN from scratch sedikit berbeda dengan model dari sklearn dari segi akurasi dan learning loss. Hal ini karena model dari sklearn didalamnya terdapat banyak optimasi sehingga lebih cepat mencapai konvergen dan menghasilkan akurasi yang lebih baik. Seperti contohnya, MLPClassifier secara default menggunakan optimasi adam, secara default juga MLPClassifier menerapkan regularisasi dan tentunya terdapat optimasi-optimasi lainnya. Selain itu, model sklearn juga lebih cepat secara waktu karena terdapat optimasi-optimasi dibalik layar dibanding hanya menggunakan perhitungan dengan numpy.

# KESIMPULAN DAN SARAN

## Kesimpulan

Implementasi model FFNN from scratch menghasilkan hasil yang baik dalam mengklasifikasi dataset MNIST, meskipun performanya masih sedikit di bawah model MLPClassifier dari scikit-learn. Hal ini terlihat dari perbedaan akurasi, di mana model scikit-learn mencapai 97.25%, sedangkan FFNN from scratch mencapai 95.64%. Selain itu, model scikit-learn mengalami penurunan loss yang lebih cepat, menunjukkan optimasi yang lebih stabil dan efisien. Perbedaan ini dapat disebabkan oleh beberapa faktor seperti pemilihan algoritma optimasi dan penanganan angka numerik (clipping dll) yang diterapkan dalam model scikit-learn.

## Saran

Pengerjaan tugas besar ini bersamaan dengan beberapa tugas-tugas dari mata kuliah lain, sehingga sangat dianjurkan untuk mencicil tugas-tugas yang ada daripada *speed running* di beberapa hari terakhir

Pembagian Tugas Kelompok

| NIM | Task |
|---|---|
| 13522011 | All tests notebook implementation |
| 13522057 | All function implementations |
| 13522095 | All function implementations |

Referensi


- https://medium.com/analytics-vidhya/activation-function-c762b22fd4da
- https://www.geeksforgeeks.org/xavier-initialization/
- https://www.deeplearningbook.org/contents/numerical.html
- https://www.geeksforgeeks.org/adam-optimizer/
- https://www.geeksforgeeks.org/gradient-descent-with-rmsprop-from-scratch/
- https://dl.acm.org/doi/pdf/10.5555/3454287.3455397
- https://www.geeksforgeeks.org/regularization-in-machine-learning/
-