LAPORAN TUGAS BESAR 1

IF3270 Pembelajaran Mesin

Feedforward Neural Network

Disusun oleh:

Farhan Nafis Rayhan (13522037)

Raden Francisco Trianto B. (13522091)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2025

DAFTAR ISI

BAB I
DESKRIPSI PERSOALAN

Artificial Neural Network (ANN) adalah model komputasi yang terinspirasi oleh jaringan saraf biologis dalam otak manusia. ANN terdiri dari sejumlah unit pemrosesan sederhana yang disebut neuron, yang saling terhubung dan bekerja secara paralel untuk memproses informasi. Model ini mampu mengenali pola dan hubungan dalam data, sehingga banyak digunakan dalam berbagai aplikasi seperti klasifikasi, regresi, dan pemrosesan sinyal.

Feedforward Neural Network (FFNN) merupakan salah satu jenis ANN di mana informasi mengalir secara searah dari lapisan input melalui satu atau lebih lapisan tersembunyi hingga mencapai lapisan output, tanpa adanya umpan balik atau siklus. Setiap neuron dalam FFNN terhubung secara penuh (fully connected) dengan neuron di lapisan berikutnya, memungkinkan jaringan untuk mempelajari representasi data yang kompleks. Struktur ini membuat FFNN efektif dalam menangani berbagai tugas pembelajaran mesin seperti pengenalan pola dan prediksi.

Dalam tugas besar ini, kami diberikan persoalan yaitu membuat sebuah model FFNN dalam bahasa python dari scratch. Model kami harus memiliki ketentuan sesuai spesifikasi sebagai berikut:

- Input berupa batch.
- Menerima jumlah neuron dari setiap layer.
- Menerima fungsi aktivasi setiap layer yang terdiri dari: Linear, ReLu, Sigmoid, tanh, dan softmax.
- Menerima fungsi loss yang terdiri dari: MSE, Binary Cross-Entropy, Categorical Cross-Entropy.
- Memiliki 3 mekanisme inisialisasi bobot: Zero Initialization, Random dengan distribusi uniform, Random dengan distribusi normal.
- Model menyimpan bobot, bias, dan gradien setiap neuron.
- Model memiliki fitur save dan load.

- Model dapat ditampilkan dalam bentuk graf yang menunjukan struktur jaringan, bobot serta gradien bobot.
- Model dapat menampilkan distribusi bobot dari setiap layer.
- Model dapat menampilkan distribusi gradien dari setiap layer.
- Model dapat melakukan forward dan backward propagation
- Pelatihan Model harus dapat menerima parameter berikut:
    - Batch size
    - Learning rate
    - Jumlah epoch
    - Verbose
        - Verbose 0 berarti tidak menampilkan apa-apa selama pelatihan
        - 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.

Untuk ketentuan lebih lengkap, dapat dilihat pada tautan berikut:

Spesifikasi Tugas Besar 1 IF3270 Pembelajaran Mesin

BAB II

PEMBAHASAN

A. Penjelasan Implementasi

a. Hasil Implementasi

i. **Kelas Value** (src/model/value.py)

Class Value	
Representasi nilai skalar pada ekspresi matematika. Digunakan untuk implementasi Automatic Gradient (Autograd) dengan cara menyimpan fungsi chain rule dalam attribut.	
Attributes	
data	Nilai asli skalar dalam tipe data primitif (integer, float).
grad	Nilai gradien dari operasi matematika.
_backward	Fungsi untuk menghitung gradien menggunakan chain rule. Digunakan untuk automatic gradient.
_prev	Value-Value yang digunakan untuk mendapatkan Value saat ini. Dapat dianggap sebagai Child dari Value saat ini.
_op	Operasi yang dilakukan untuk mendapatkan nilai Value saat ini.
label	Label dari Value, misal bias atau weight.
Methods	
__init__	Konstruktor Value. Menerima data, children, operation, dan label dari Value.
Operasi matematika dasar (tambah, kurang, kali, bagi, pangkat, negatif)	Sama seperti operasi matematika dasar pada tipe data primitif.
backward	Melakukan backward propagation dengan menggunakan chain rule. Fungsi secara

	rekursif menghitung gradien dari semua childnya sehingga otomatis mendapatkan gradien dari semua operasi atau fungsi.

ii. **Kelas Matrix** (src/model/matrix.py)

Class Matrix	
Representasi matriks dua dimensi dengan setiap elemen merupakan objek Value. Digunakan untuk melakukan operasi matrix dan batch input pada model.	
Attributes	
data	Data asli dari matriks berupa list of list of Value.
rows	Jumlah baris pada Matrix.
cols	Jumlah kolom pada Matrix.
Methods	
__init__	Konstruktor yang memastikan dan mengkonversi semua data menjadi object Value. Menerima data yaitu data asli dalam bentuk list of list.
transpose	Melakukan Transpose Matrix.
dot	Melakukan perkalian dot product. Digunakan saat menghitung net/sigma pada setiap layer. Menerima other yaitu Matrix lain yang akan dikalikan.
add_scalar	Menambahkan nilai skalar pada matrix. Digunakan saat menambahkan bias. Menerima nilai skalar yang akan ditambahkan.

iii. **Kelas Abstract Module** (src/model/nn.py)

Class Module (Abstract)
Kelas abstract sebagai kelas dasar untuk semua komponen kelas dari Neural Network (MLP, Layer, Neuron).
Attributes
Tidak memiliki attribute karena kelas abstrak.

Methods	
zero_grad	Mengubah nilai gradien semua Value pada kelas menjadi 0. Digunakan untuk mereset nilai gradien sebelum perhitungan Automatic Gradient (Autograd).
parameters	Mengembalikan semua variabel parameter yang digunakan pada kelas.

iv. **Kelas Neuron** (src/model/nn.py)

Class Neuron	
Representasi sebuah Neuron pada Neural Network.	
Attributes	
w	List nilai weight/bobot untuk setiap Neuron tujuan.
b	Nilai bias.
active_func	Fungsi aktivasi pada Neuron.
label	Label dari Neuron.
Methods	
__init__	Konstruktor Neuron yang menginisialisasi weight dan bias. Menerima jumlah input, mode inisialisasi bobot dan bias, fungsi aktivasi, label, dan seed untuk bobot.
__call__	Melakukan forward pass pada Neuron. Menghitung $Z = wT \cdot x + b$ dan menggunakan fungsi aktivasi untuk menghasilkan output pada neuron.
_validate_param	Fungsi untuk mengecek parameter untuk inisialisasi bobot seperti upper bound dan lower bound.
parameters	Mengembalikan semua variabel parameter yang digunakan pada Neuron yaitu semua bobot dan bias.

v. **Kelas Layer** (src/model/nn.py)

Class Layer
Representasi sebuah Layer pada Neural Network.

Attributes	
neurons	List Neuron yang ada pada layer.
label	Label dari Layer.
Methods	
__init__	Konstruktor Layer yang sekaligus membuat Neuron-Neuron pada layer tersebut.
__call__	Melakukan forward pass pada Layer. Menghitung $Z = wT . x + b$ dengan menggunakan operasi matriks.
parameters	Mengembalikan semua variabel parameter yang digunakan pada Layer.

vi. **Kelas MLP** (src/model/nn.py)

Class MLP	
Representasi sebuah Multi Layered Perceptron atau fully connected neural network.	
Attributes	
layers	List Layer pada MLP.
Methods	
__init__	Konstruktor MLP yang sekaligus membuat Layer-Layer pada MLP..
__call__	Melakukan forward pass pada Layer dengan memanggil fungsi milik layer.
parameters	Mengembalikan semua variabel parameter yang digunakan pada MLP.

vii. **Kelas FFNN** (src/model/ffnn.py)

Class FFNN
Model Feed Forward Neural Network dari scratch.
Attributes

X	Input batch.
y	Target.
loss	Fungsi loss
NN	Object MLP yang menyimpan struktur Neural Network.
Methods	
__init__	Konstruktor FFNN yang membuat MLP.
forward	Melakukan forward propagation dari FFNN..
backpropagation	Melakukan backward propagation dari FFNN.
Training	Melakukan pelatihan model.
zero_grad	Menginisialisasi semua gradien pada FFNN menjadi 0. Digunakan sebelum backward propagation untuk automatic gradient.
parameters	Mengembalikan semua variabel parameter yang digunakan pada MLP.

viii. **Kelas Converter** (src/utils/converter.py)

Class Converter	
Kelas Statik yang menyediakan fungsi untuk konversi dari tipe dasar menjadi Value.	
Attributes	
Tidak ada karena kelas statik.	
Methods	
to_Value	Mengkonversi tipe dasar integer atau float menjadi Value
to_Values	Mengkonversi list dari tipe dasar integer atau float menjadi list of Value

ix. **Kelas ActiveFunction** (src/func/activations.py)

Class ActiveFunction

Kelas Statik yang menyediakan fungsi-fungsi aktivasi.	
Attributes	
Tidak ada karena kelas statik.	
Methods	
linier	Fungsi aktivasi linear.
relu	Fungsi aktivasi relu.
sigmoid	Fungsi aktivasi sigmoid.
tanh	Fungsi aktivasi Hyperbolic Tangent.
softmax	Fungsi aktivasi softmax.
exp	Fungsi aktivasi exponensial (exp).
log	Fungsi aktivasi natural log.

x. **Kelas LossFunction** (src/func/loss.py)

Class LossFunction	
Kelas Statik yang menyediakan fungsi-fungsi loss.	
Attributes	
Tidak ada karena kelas statik.	
Methods	
mse	Fungsi loss means square error.
bce	Fungsi loss binary cross-entropy.
cce	Fungsi loss categorical cross-entropy

xi. **Kelas RegularizationFunctions** (src/func/regularization.py)

Class RegularizationFunctions
Kelas Statik yang menyediakan fungsi-fungsi regularisasi.
Attributes

Tidak ada karena kelas statik.	
Methods	
none	Fungsi apabila tidak menggunakan regularisasi.
l1	Fungsi regularization L1 (Lasso).
l2	Fungsi regularization L2 (Ridge).

b. Forward Propagation

Pada implementasi kami. Forward Propagation dimulai dari FFNN yang memanggil forward untuk MLP dengan input berupa X. MLP kemudian memanggil forward untuk setiap Layer dengan melanjutkan output dari masing-masing layer. Masing-masing Layer menghitung $Z = wT \,.\, x + b$ dengan cara memanggil forward dari masing-masing Neuron pada Layer tersebut. Hasil dari Neuron digabungkan menjadi matrix dan dikembalikan oleh Layer ke MLP. MLP menyimpan hasil layer dan memanggil forward pada layer selanjutnya hingga tidak ada layer lagi. Hasil terakhir adalah hasil dari output layer dan menjadi hasil dari forward propagation.

Berikut adalah potongan kode yang berhubungan dalam forward propagation:

```
Class Neuron:
def __call__(self, X: Matrix) -> Matrix:
        """Forward pass for a batch (Matrix)."""
        assert isinstance(X, Matrix), "Layer input must be a Matrix."

        # Z = wT . x + b
        weighted_sum = X.dot(Matrix([self.w]).transpose())

        activated_output = self.active_func(weighted_sum.add_scalar(self.b))

        return activated_output
```

```
Class Layer:
def __call__(self, X: Matrix):
        """Call the forward pass of each neuron"""

        assert isinstance(X, Matrix), "Layer input must be a Matrix."

        out_data = [
            [neuron(Matrix([row])).data[0][0] for neuron in self.neurons]
            for row in X.data
        ]

        out = Matrix(out_data)
```

```
        return out
```

```
Class MLP:
def __call__(self, x: Matrix):
        """Forward pass of the network."""
        for layer in self.layers:
            x = layer(x)
        return x
```

```
Class FFNN:
   def forward(self, batch_X : np.ndarray, batch_y : np.ndarray) -> Value:
        """ Forward Pass of the neural network using loss function. """
        return self.loss(Matrix(batch_y).transpose(),
self.NN(Matrix(batch_X)))
```

c. Backward Propagation, weight update, dan Autograd

Pada implementasi kami, backward propagation menggunakan automatic gradient (Autograd). Proses backward propagation dimulai dari inisialisasi gradient menjadi 0. Hal tersebut bertujuan untuk mereset nilai gradient sehingga tidak terpengaruh dengan gradient pada backward pass sebelumnya.

Setelah mereset nilai gradient, kita bisa memulai Automatic Gradient (Autograd). Autograd sendiri bekerja dengan cara menyimpan semua ekspresi matematika, operasinya dan nilai-nilai asal yang menghasilkannya. Hal ini dicapai dengan membuat class baru sebagai representasi nilai scalar/dasar dari ekspresi matematika yaitu class Value.

```
class Value:
    """ Class to represent a value in a mathematical expression. Used for
automatic differentiation.

        Attributes:
            data (_type_): Numerical value of node.
            grad (float): Gradient of the value.
            _backward (function): Function to compute the gradient. This will
contain the chain rule for derivatives.
            _prev (set): Children nodes.
            _op (str): Operation performed to get this value.
            label (str): Label for the node. Used to visualize the graph.
    """
```

Dengan mengetahui semua ekspresi matematika, operasi, dan nilai-nilai asalnya, secara langsung kita bisa menyimpan turunan chain rule dari semua operasi matematika termasuk loss dan activation function. Perlu diperhatikan

bahwa pada implementasi, setiap fungsi aktivasi atau loss atau operasi lainnya sudah menyimpan fungsi backward. Fungsi ini menyimpan fungsi chain rule untuk mendapatkan gradien dari fungsi atau operasi tersebut.

Berikut contoh fungsi exponensial yang menyimpan fungsi chain rule pada _backward:

```
def exp(X: Matrix) -> Matrix:
        """ Exponential activation function. exp(X) = e^X """
        out_data = []
        for row in X.data:
            out_data.append([Value(math.exp(val.data), (val,), "exp") for val
in row])

        def _backward():
            for i, row in enumerate(X.data):
                for j, val in enumerate(row):
                    val.grad += out_data[i][j].data * out_data[i][j].grad

        for row in out_data:
            for val in row:
                val._backward = _backward

        return Matrix(out_data)
```

Secara rekursif dari ekspresi matematika pada fungsi loss, kita dapat menghitung gradien untuk semua operasi matematika yang terjadi pada fase forward propagation. Sehingga kita dapat menemukan gradien untuk fungsi-fungsi aktivasi pada masing-masing Neuron yang menggunakan parameter berupa bias serta weight.

Setelah mendapatkan nilai gradien kita dapat menggunakan fungsi $w_i = w_i + \eta\,(\frac{\partial L}{\partial w_i})$ dengan L adalah loss function, w_i adalah bobot i, η adalah learning rate, dan $(\frac{\partial L}{\partial w_i})$ adalah gradient yang dihitung dengan automatic gradient (autograd).

Fungsi tersebut digunakan untuk mengoptimalkan nilai bobot sehingga bobot semakin besar jika gradien nilainya besar dan sebaliknya. Sehingga backward propagation berhasil menghasilkan bobot yang terbaru.

B. Hasil Pengujian

Hasil Pengujian dapat dilihat pada notebook testing.ipynb.

a. Pengaruh Depth dan Width

Testing dilakukan pada file testing_depth_width.ipynb.

Depth adalah banyak hidden layer pada Neural Network. Width adalah jumlah neuron pada masing-masing layer. Depth dan Width sangat mempengaruhi hasil dari neural network.

Berikut adalah hasil testing model dengan menggunakan data MNIST 784 dari OpenML. Pada testing ini semua layer menggunakan sigmoid dan output layer menggunakan relu. Loss function yang digunakan adalah means square error atau MSE.

Tabel Hasil Testing Pengaruh Depth

Hasil Testing Pengaruh Depth					
Jumlah Data	**Depth**	**Width**	**Epoch**	**Resulting Training Loss**	**Resulting Validation Loss**
1000	2	10	5	62.73248968509594	60.008666898982376
	3	10		62.732467920566926	60.00864709782901
	4	10		62.732467891372316	60.008647071266864

Tabel Hasil Training Loss dengan Depth berbeda

Hasil Testing Pengaruh Depth			
Epoch	**Depth**		
	2	**3**	**4**
1	74.93193990485591	74.93190883071136	74.93190878902904

2	71.67568316223442	71.67565465108737	71.67565461284316
3	68.5610863941962	68.56106028289224	68.56106024786729
4	65.58198679889037	65.58196293353575	65.58196290152316
5	62.73248968509594	62.732467920566926	62.732467891372316

Tabel Hasil Validation Loss dengan Depth berbeda

Hasil Testing Pengaruh Depth			
Epoch	**Depth**		
	2	**3**	**4**
1	71.67604055950052	71.67601204821523	71.67601200997053
2	68.56179332820858	68.56176721663381	68.56176718160798
3	65.58303558201584	65.58301171626543	65.58301168425386
4	62.73387279878433	62.73385103373885	62.733851004545606
5	60.008666898982376	60.00864709782901	60.008647071266864

Tabel Hasil Training Loss dengan Width berbeda

Hasil Testing Pengaruh Depth			
Epoch	**Width**		
	5	**12**	**20**
1	6.710846788541321	68.8175754633515	79.71934376451574
2	6.552709019553877	65.28979078827028	73.17028716269677
3	6.398343785183401	61.943066133853065	67.15979877958542
4	6.247661087844074	58.76810877559925	61.643588919147554
5	6.100573076932449	55.75610292880026	56.58101009279244

Tabel Hasil Validation Loss dengan Width berbeda

Hasil Testing Pengaruh Depth			
Epoch	**Width**		
	5	**12**	**20**
1	6.552904019553916	65.2902132882681	73.17096966269735
2	6.398731445183423	61.94390014885298	67.16113511458447
3	6.2482390959240615	58.76934360620927	61.645551628076
4	6.101339148915396	55.75772815381423	56.58357286794678
5	5.957945876728356	52.900690739247054	51.93789513740076

Tabel Hasil Testing Pengaruh Width

Hasil Testing Pengaruh Width					
Jumlah Data	**Depth**	**Width**	**Epoch**	**Resulting Training Loss**	**Resulting Validation Loss**
1000	1	5	5	6.100573076932449	5.957945876728356
	1	12		55.75610292880026	52.900690739247054
	1	20		56.58101009279244	51.93789513740076

1) Pengaruh Depth

   Semakin besar depth dalam sebuah neural network, semakin kompleks fungsi yang dapat dipelajari oleh model. Hal ini dapat dilihat pada nilai hasil fungsi loss yang berbeda pada percobaan dengan tiga besar depth yang digunakan yaitu 2, 3, dan 4. Nilai loss berbeda, namun masih saling mendekati. Hal ini dikarenakan data yang digunakan masih sedikit yaitu 1000 data saja. Tidak hanya itu, jumlah epoch yang relatif sedikit yaitu 5 epoch juga mempengaruhi besar dari nilai loss.

2) Pengaruh Width

Jumlah neuron dalam suatu layer, atau width, berpengaruh terhadap kapasitas pembelajaran model. Semakin banyak neuron, semakin banyak parameter (bobot) yang digunakan untuk menyimpan informasi dan mengenali pola dari data. Dengan width yang lebih besar, model dapat menangkap pola yang lebih kompleks.

Namun, jika jumlah neuron terlalu banyak dibandingkan dengan jumlah data yang tersedia, model berisiko mengalami overfitting. Overfitting terjadi ketika model terlalu menyesuaikan diri dengan data pelatihan sehingga sulit untuk menggeneralisasi ke data baru. Oleh karena itu, jumlah neuron yang optimal perlu dipilih agar model tetap efektif.

Pada eksperimen, model dengan width = 5 memiliki training loss dan validation loss lebih rendah dibandingkan model dengan width = 12 atau 20. Kemungkinan, model dengan width lebih besar memiliki terlalu banyak parameter dibandingkan jumlah data, menyebabkan overfitting.

Sebaliknya, model dengan width = 5 memiliki kapasitas yang cukup untuk menangkap pola utama dalam data tanpa menjadi terlalu kompleks. Hal ini membuatnya belajar lebih efisien dan memiliki generalisasi yang lebih baik. Ini menunjukkan bahwa lebih banyak neuron tidak selalu lebih baik.

b. Pengaruh Fungsi Aktivasi

Activation function menentukan bagaimana sebuah neuron memproses input dan menghasilkan output, memperkenalkan non-linearitas yang memungkinkan jaringan mempelajari hubungan kompleks dalam data. Jika fungsi aktivasi yang dipilih tidak sesuai, model dapat mengalami masalah seperti vanishing gradients (gradien terlalu kecil sehingga pembelajaran terhambat) atau exploding gradients (gradien terlalu besar menyebabkan ketidakstabilan). Sebaliknya, pemilihan yang tepat mempercepat konvergensi dan meningkatkan akurasi model.

- Linear: Tidak memperkenalkan non-linearitas, cocok untuk masalah regresi sederhana, tetapi tidak efektif untuk jaringan yang dalam karena tidak dapat memodelkan hubungan kompleks.
- ReLU (Rectified Linear Unit): Efisien secara komputasi dan mengurangi vanishing gradient, tetapi dapat menyebabkan dying ReLU (neuron mati jika output selalu nol).
- Sigmoid: Berguna untuk klasifikasi biner (output antara 0 dan 1), tetapi lambat dalam pembelajaran karena gradien yang kecil di daerah saturasi.
- Tanh (Hyperbolic Tangent): Menghasilkan output dalam rentang [-1, 1], cocok untuk data terpusat, tetapi rentan terhadap vanishing gradient pada nilai ekstrem.
- Softmax: Digunakan di lapisan output untuk klasifikasi multi-kelas, mengubah logits menjadi probabilitas yang terdistribusi dengan baik.

Berikut adalah hasil pengujian model dengan berbagai fungsi aktivasi menggunakan dataset MNIST 784 dari OpenML.

Tabel Hasil Testing Pengaruh Fungsi Aktivasi

Pengaruh Inisialisasi Bobot				
Jumlah Data	**Fungsi Aktivasi**	**Epoch**	**Resulting Training Loss**	**Resulting Validation Loss**
1000	Linear	4	-	-
	ReLu		0.1	0.1
	Sigmoid		69.70503703574086	65.59148219644256
	Tanh		-	-

Data menunjukkan bahwa training dengan fungsi Linear atau Tanh sebagai fungsi aktivasi layer pertama tidak dapat dilakukan. Bagi fungsi linear, masalahnya terletak pada fakta bahwa hasil perhitungan mulai epoch ke-2 sudah

sangat besar, sampai melebihi ukuran float Python. Penyebabnya adalah fungsi linear yang tidak menurunkan nilai hasil sehingga sangat rentan terjadinya overflow setelah beberapa kali *feedforward*. Untuk kasus Tanh, nilai input yang cukup besar menjadi masalah bagi fungsi aktivasi Tanh. Fakta bahwa fungsi ini menggunakan perpangkatan dari bilangan *e* menjadi batasan mengingat Python hanya dapat mencapai e^{1000} sebelum melewati limit tipe data float.

Untuk kasus ReLu, fungsi ini sering menghadapi masalah Dying ReLu. Penyebabnya adalah fakta bahwa ReLu akan selalu menyebabkan nilai negatif menjadi 0. Apabila banyak perceptron yang menghasilkan nilai negatif, lalu dibuat 0 oleh ReLu, gradiennya akan turut menjadi 0. Akibatnya, banyak perceptron yang berhenti belajar atau "mati", mengurangi kapasitas model dan membuatnya terjebak di solusi suboptimal.

Berbeda dengan fungsi Sigmoid yang menunjukkan hasil cukup baik. Data menunjukkan bahwa hanya dalam 4 epoch, fungsi sigmoid berhasil menurunkan loss sebesar 14 selama training. Penurunan ini cukup besar, sehingga dapat disimpulkan bahwa Sigmoid merupakan fungsi aktivasi terbaik bagi *hidden layer* pertama.

c. Pengaruh Learning Rate

Learning rate menentukan seberapa besar langkah perubahan bobot model selama pelatihan. Jika terlalu tinggi, model dapat melewati solusi optimal dan gagal berkonvergensi, menyebabkan osilasi atau ketidakstabilan. Sebaliknya, jika terlalu rendah, pelatihan menjadi lambat dan berisiko terjebak dalam local minima. Pemilihan learning rate yang tepat sangat penting untuk memastikan model belajar secara efisien tanpa kehilangan akurasi atau stabilitas.

Berikut adalah hasil testing model dengan menggunakan data MNIST 784 dari OpenML.

Tabel Hasil Testing Pengaruh Learning Rate

Hasil Testing Pengaruh Depth				
Jumlah Data	**Learning Rate**	**Epoch**	**Resulting Training Loss**	**Resulting Validation Loss**
1000	0.1	5	1.2937111514275614	1.0701408885016703
	0.01		2.6794073249420394	2.6285154066382987
	0.001		2.8744304343276745	2.8689294617777863

Tabel Hasil Training Loss Dengan Learning Rate Berbeda

Hasil Testing Pengaruh Depth			
Epoch	**Learning Rate**		
	0.1	**0.01**	**0.001**
1	2.896870707589629	2.896870707589629	2.896870707589629
2	2.362046031174357	2.8408548638912294	2.8912437894592276
3	1.9288378992625854	2.7859537338232467	2.8856281195213382
4	1.5779391957586455	2.732145134617462	2.880023675290873
5	1.2937111514275614	2.6794073249420394	2.8744304343276745

Tabel Hasil Validation Loss dengan Width berbeda

Hasil Testing Pengaruh Depth			
Epoch	**Learning Rate**		
	0.1	**0.01**	**0.001**
1	2.3636710287427123	2.8410173636481075	2.891260039434877
2	1.931925394885616	2.786277108341785	2.885660603222839
3	1.5823429398497313	2.7326277751524337	2.8800723764843195
4	1.2992995193368868	2.680047638835742	2.8744304343276745
5	1.0701408885016703	2.6285154066382987	2.8689294617777863

Dari data, learning rate 0.1 menunjukkan penurunan loss paling cepat, menandakan konvergensi yang baik tetapi berisiko overshooting jika terlalu banyak epoch. Learning rate 0.01 lebih stabil tetapi lebih lambat dalam menurunkan loss, sedangkan 0.001 hampir stagnan, menunjukkan bahwa model belajar terlalu lambat. Ini menegaskan bahwa pemilihan learning rate yang tepat sangat penting untuk keseimbangan antara kecepatan dan stabilitas pelatihan.

d. Pengaruh Inisiasi Bobot

Inisialisasi bobot menentukan nilai awal parameter model sebelum pelatihan dimulai. Jika bobot diinisialisasi dengan nilai yang terlalu besar atau terlalu kecil, model dapat mengalami masalah seperti vanishing gradients atau exploding gradients, yang menghambat kemampuan jaringan untuk belajar secara efektif. Sebaliknya, inisialisasi yang tepat membantu menjaga distribusi aktivasi yang stabil antar lapisan, memungkinkan gradien mengalir dengan baik selama backpropagation. Pemilihan metode inisialisasi yang sesuai sangat penting untuk memastikan model FFNN dapat berkonvergensi dengan cepat dan mencapai performa optimal tanpa terjebak pada masalah optimisasi awal.

Berikut adalah hasil pengujian model FFNN dengan berbagai metode inisialisasi bobot menggunakan dataset MNIST 784 dari OpenML.

Tabel Hasil Testing Pengaruh Inisialisasi Bobot

Pengaruh Inisialisasi Bobot				
Jumlah Data	**Inisialisasi Bobot**	**Epoch**	**Resulting Training Loss**	**Resulting Validation Loss**
1000	Normal	5	2.6794073249420394	2.6285154066382987
	Uniform		7.345891785678532	7.331470346294391
	Zero		0.1	0.1

Tabel Hasil Training Loss Dengan Inisialisasi Bobot berbeda

Hasil Training Loss			
Epoch	**Learning Rate**		
	Zero	**Uniform**	**Normal**
1	0.1	7.404192201200848	2.896870707589629
2	0.1	7.389573370801301	2.8408548638912294
3	0.1	7.3749837302589905	2.7859537338232467
4	0.1	7.360423221277931	2.732145134617462
5	0.1	7.345891785678532	2.6794073249420394

Tabel Hasil Validation Loss Dengan Inisialisasi Bobot berbeda

Hasil Validation Loss			
Epoch	**Learning Rate**		
	Zero	**Uniform**	**Normal**
1	0.1	7.389589599368569	2.8410173636481075
2	0.1	7.375016171183404	2.786277108341785
3	0.1	7.360471858365303	2.7326277751524337
4	0.1	7.34595660275146	2.680047638835742
5	0.1	7.331470346294391	2.6285154066382987

Data diatas menunjukkan bahwa inisialisasi zero mengakibatkan model terjebak pada local minima dengan loss 0.1 baik untuk training maupun validation. Model kesulitan untuk mengeksplorasi arah berbeda pada gradien loss karena simetri bobot mengakibatkan setiap perceptron bekerja dengan cara yang sama persis. Inisialisasi Uniform menghasilkan loss yang cukup baik namun masih tidak sebaik inisialisasi Normal. Pada kasus ini, inisialisasi normal mulai dengan bobot yang membuat loss sudah cukup rendah. Terlebih, inisialisasi ini menurunkan loss lebih cepat dibandingkan yang lainya. Oleh karena itu, dapat

disimpulkan bahwa Inisialisasi Normal akan membuat model konvergen lebih cepat untuk dataset ini.

Tabel Hasil Testing Pengaruh Regularisasi

Pengaruh Regularisasi				
Jumlah Data	**Regularisasi**	**Epoch**	**Resulting Training Loss**	**Resulting Validation Loss**
1000	None	5	2.6794073249420394	2.6285154066382987
	L1		290.8360780067369	290.78679541854245
	L2		-	-

e. Pengaruh Regularisasi

Regularisasi merupakan teknik yang digunakan untuk mencegah overfitting dengan membatasi kompleksitas model selama pelatihan. Tanpa regularisasi, model cenderung menghafal data pelatihan alih-alih mempelajari pola umum, sehingga performanya buruk pada data baru. Terlalu banyak regularisasi dapat menyebabkan underfitting, di mana model menjadi terlalu sederhana dan gagal menangkap hubungan penting dalam data. Pemilihan teknik regularisasi yang tepat sangat penting untuk menyeimbangkan bias dan varian model. Umumnya, regularisasi yang digunakan adalah L1 dan L2

L1 Regularization (Lasso) memiliki konsep menambahkan penalti berupa jumlah absolut nilai bobot (L1-norm) ke fungsi loss. Dampaknya fungsi ini menyebabkan sparsity (bobot bernilai nol), sehingga berguna untuk feature selection. Fungsi ini cocok apabila hanya ada sedikit fitur yang relevan pada dataset.

Di lain pihak, L2 Regularization (Ridge) Menambahkan penalti berupa kuadrat nilai bobot (L2-norm) ke fungsi loss. Efek dari fungsi ini adalah membatasi besar bobot tanpa menghilangkannya sepenuhnya. Regularisasi ini lebih stabil secara numerik dibanding L1 dan hasilnya akan lebih *smooth*.

f. Perbandingan dengan library sklearn

Kami membandingkan model kami yang dibuat dari scratch dengan library sklearn yang populer. Hasil training 1000 data 5 epoch atau 200 steps adalah sebagai berikut:

	Model Kami	Sklearn
Training Loss	0.6593121603631645	2.11312311
Validation Loss	0.6591439046145068	3.25010116

Dengan membandingkan dengan library sklearn, model kami memang masih kurang baik. Hal tersebut karena model kami yang kurang efisien dalam implementasi. Model kami juga memerlukan waktu yang jauh lebih lama daripada library sklearn sebab implementasi autograd yang memberatkan dan membuat lambat proses training dan prediction.

Tidak hanya itu hasil prediksi juga berbeda dengan sklearn memiliki kurva pembelajaran yang jauh lebih baik dari pada model kami. Hal terjadi karena implementasi model ffnn pada sklearn yang lebih baik.

BAB III
KESIMPULAN DAN SARAN

A. Kesimpulan

Dengan ini, kami berhasil membuat sebuah model Feed Forward Neural Network (FFNN) dari scratch dengan menggunakan bahasa Python. Model kami dapat menerima input berupa data, batch size, learning rate, jumlah epoch, dan verbose serta parameter-parameter seperti jumlah neuron per layer, fungsi aktivasi, fungsi loss, dan cara inisialisasi bobot dan bias. Model FFNN berhasil melakukan forward propagation dan backward propagation, menampilkan graf dari jaringan Neural Network dan dapat melakukan save dan load dari model tersebut. Model kami juga mengimplementasikan automatic gradient yang digunakan saat terjadinya backward propagation. Model juga dapat mengembalikan sejarah dari proses pelatihan yang berisi training loss dan validation loss pada setiap epoch yang dijalani.

Pada tugas ini, kami juga berhasil melihat pengaruh besar depth dan width dari neural network, fungsi aktivasi yang digunakan, besar learning rate, dan metode inisialisasi bobot pada model kami. Kami juga membandingkan hasil implementasi terhadap library sklearn MLP yang paling sering digunakan pada pembuat NN.

B. Saran

Berikut adalah saran untuk kelompok kami pada tugas-tugas selanjutnya:

- Melakukan pembagian tugas pada awal pengerjaan
- Pemisahan tugas sehingga adil dan terstruktur
- Memperbanyak diskusi selama pengerjaan tugas
- Memperdalam pengetahuan mengenai FFNN sehingga memiliki pengertian dan konsep yang sama
- Memahami konsep bonus autograd sebelum memulai implementasi tugas
- Mereview konsep diferensial yang digunakan pada chain rule

Pembagian Tugas

NIM	Nama	Tugas
13522037	Farhan Nafis Rayhan	● Backward Propagation ● Automatic Gradient ● Weight Initialization and seed ● Verbose and progress bar ● Activation Functions ● Loss Functions ● Training ● Regularization ● Report
13522091	Raden Francisco Trianto B.	● Batch Input ● Activation Functions ● Forward Propagation ● Automatic Gradient ● Neural Network (class Value, Neuron, Layer, dkk) ● Visualisasi model dan plot ● Testing ● Report

Referensi

1. Osajima, J. (n.d.). *Forward Propagation Visualization*. Diakses dari https://www.jasonosajima.com/forwardprop

2. Karpathy, A. (2020). *Micrograd: A tiny scalar-valued autograd engine and a neural net library on top of it*. GitHub Repository. Diakses dari https://github.com/karpathy/micrograd

3. Karpathy, A. (2020). *The spelled-out intro to neural networks and backpropagation: building micrograd*. YouTube. Diakses dari https://www.youtube.com/watch?v=VMj-3S1tku0

4. GeeksforGeeks. (n.d.). Feedforward Neural Network. Diakses dari https://www.geeksforgeeks.org/feedforward-neural-network/

5. Tim Pengajar IF 3270. Artificial Neural Network. Diakses sebagai bahan materi perkuliahan.