Laporan Tugas Besar IF3270 Pembelajaran Mesin
Bagian B: Implementasi Mini-batch Gradient Descent

Disusun oleh:

Jevant Jedidia Augustine	13520133
Vincent Christian Siregar	13520136
David Karel Halomoan	13520154
Willy Wilsen	13520160

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2023

Bab I
Implementasi

Program

Pada program model Mini-batch Gradient Descent blok 1, dilakukan import library yang dibutuhkan. Selain itu, dilakukan juga pengambilan dataset iris yang kemudian dipisah menjadi data dan target.

Pada program model Mini-batch Gradient Descent blok 2, terdapat fungsi-fungsi aktivasi yang diperlukan yaitu linear, reLU, sigmoid, dan softmax yang diperlukan untuk melakukan FFNN terhadap input. Setiap fungsi aktivasi ini akan menerima parameter berupa nilai net dan mengembalikan output yang dihitung berdasarkan fungsi aktivasi tersebut. Pada blok ini juga didefinisikan fungsi derive dan deriveSoftmax untuk memberikan hasil turunan berdasarkan jenis aktivasi yang diterima. Selain itu, diberikan fungsi loss untuk menghitung error dari model FFNN.

Pada program model Mini-batch Gradient Descent blok 3, disediakan kelas ANN dengan berbagai method sebagai berikut.

Method	Fungsi
readInput(self, activation, layer, neuron_each_layer, inp, weight, target, learning_rate, batch_size, max_iter, error_threshold)	Membaca seluruh atribut yang diperlukan dari input pengguna
readFile(self, file)	Membaca seluruh atribut yang diperlukan dari sebuah file
generateWeight(self)	Inisialisasi bobot awal secara random
forwardProp(self, inp)	Melakukan FFNN terhadap input berdasarkan batch size yang diberikan
gradient(self, index, level, j, k, is_leaf)	Menghitung nilai gradien dari bobot yang diberikan
backwardProp(self, start, end, count)	Melakukan Backward Propagation terhadap input berdasarkan batch size dan learning rate yang diberikan
mini_batch(self)	Melakukan Mini Batch Gradient Descent terhadap seluruh input berdasarkan batch size, max iteration, dan error threshold yang diberikan

saveModel(self, file_name)	Melakukan save terhadap model yang sudah dibangun

Program model Mini-batch Gradient Descent blok 4 merupakan blok yang berisi program pertama untuk melakukan Mini-batch Gradient Descent. Pada blok ini, diberikan variabel file yang berisi nama file model yang akan dibaca. Program kemudian melakukan Mini-batch Gradient Descent terhadap file tersebut dan menyimpan modelnya dalam sebuah folder output.

Pada program model Mini-batch Gradient Descent blok 5, dilakukan pembangunan model Mini-batch Gradient Descent dari dataset iris yang telah dipisah. Kemudian, model ini disimpan dalam sebuah file untuk dilakukan uji coba dengan model FFNN.

Pada program model Mini-batch Gradient Descent blok 6, dilakukan pengujian hasil model Mini-batch Gradient Descent dari dataset iris dengan menggunakan file model yang sudah disimpan. Pengujian ini dilakukan dengan menggunakan fungsi model FFNN yang sudah diimplementasikan pada bagian sebelumnya.

Pada program model Mini-batch Gradient Descent blok 7, dilakukan visualisasi terhadap model dataset iris yang telah dibuat.

Teks Input

Penjelasan berikut dikutip dari README yang diberikan asisten:

```
Ada dua bagian: `case` dan `expect`.
Isi dari `case`:
* `model.input_size` (sudah jelas)
* `model.layers`, terdiri dari daftar objek beratribut berikut:
    * `number_of_neurons` (sudah jelas)
        * `activation_function` (nilai valid: `linear`, `relu`, `sigmoid`, `softmax`)
* `input`:
    * Dimensi 1 menandakan vektor ke-i. Ukurannya _arbitrary_.
        * Dimensi 2 menandakan isi suatu vektor. Ukurannya sesuai dengan `model.input_size`
* `weights`:
    * Dimensi 1 bersesuaian dengan _layer_ di `layers`.
    * Dimensi 2 berukuran banyak neuron pada lapisan sebelumnya + 1.
        * Khusus untuk _layer_ pertama: `model.input_size` + 1
        * Baris pertama adalah bias.
    * Dimensi 3 berukuran banyak _neuron_ pada lapisan yang bersesuaian.
```

Isi dari `expect`:

* `output`:
    * Ukuran dimensi 1 sesuai dengan ukuran dimensi 1 dari `input`.
    * Ukuran dimensi 2 sesuai dengan banyak _*neuron*_ pada lapisan terakhir.
* `max_sse`, nilai _*sum of squares error*_ yang dapat ditoleransi.

Bab II
Hasil Eksekusi

Test Case linear.json

Model:

```
{
    "case": {
        "model": {
            "input_size": 2,
            "layers": [
                {
                    "number_of_neurons": 3,
                    "activation_function": "linear"
                }
            ]
        },
        "input": [
            [3.0, 1.0],
            [1.0, 2.0]
        ],
        "initial_weights": [
            [
              [0.1, 0.3, 0.2],
              [0.4, 0.2, -0.7],
              [0.1, -0.8, 0.5]
            ]
        ],
        "target": [
            [ 2.0,  0.3, -1.9],
            [ 1.3, -0.7,  0.1]
        ],
        "learning_parameters": {
            "learning_rate": 0.1,
            "batch_size": 2,
            "max_iteration": 1,
            "error_threshold": 0.0
        }
```

```
    },
    "expect": {
        "stopped_by": "max_iteration",
        "final_weights": [
            [
                [ 0.22,  0.36, 0.11],
                [ 0.64,  0.3, -0.89],
                [ 0.28, -0.7,  0.37]
            ]
        ]
    }
}
```

Hasil:

```
{
    "case": {
        "model": {
            "input_size": 2,
            "layers": [
                {
                    "number_of_neurons": 3,
                    "activation_function": "linear"
                }
            ]
        },
        "input": [
            [
                3.0,
                1.0
            ],
            [
                1.0,
                2.0
            ]
        ],
        "weights": [
            [
```

```
                [
                    0.21999999999999997,
                    0.36,
                    0.10999999999999999
                ],
                [
                    0.6399999999999999,
                    0.3,
                    -0.89
                ],
                [
                    0.28,
                    -0.7,
                    0.37
                ]
            ]
        ]
    },
    "expect": {
        "output": [
            [
                1.4000000000000004,
                0.10000000000000009,
                -1.3999999999999997
            ],
            [
                0.7,
                -1.1,
                0.5
            ]
        ],
        "max_sse": 0.0
    }
}
```


Test Case linear_small_lr.json

Model:

```
{
    "case": {
        "model": {
            "input_size": 2,
            "layers": [
                {
                    "number_of_neurons": 3,
                    "activation_function": "linear"
                }
            ]
        },
        "input": [
            [3.0, 1.0],
            [1.0, 2.0]
        ],
        "initial_weights": [
            [
              [0.1, 0.3, 0.2],
              [0.4, 0.2, -0.7],
              [0.1, -0.8, 0.5]
            ]
        ],
        "target": [
            [ 2.0,  0.3, -1.9],
            [ 1.3, -0.7,  0.1]
        ],
        "learning_parameters": {
            "learning_rate": 0.001,
            "batch_size": 2,
            "max_iteration": 1,
            "error_threshold": 0.0
        }
    },
    "expect": {
        "stopped_by": "max_iteration",
        "final_weights": [
            [
                [ 0.1008,  0.3006, 0.1991],
                [ 0.402,  0.201, -0.7019],
```

```
                [ 0.101, -0.799,  0.4987]
            ]
        ]
    }
}
```

Hasil:


```
{
    "case": {
        "model": {
            "input_size": 2,
            "layers": [
                {
                    "number_of_neurons": 3,
                    "activation_function": "linear"
                }
            ]
        },
        "input": [
            [
                3.0,
                1.0
            ],
            [
                1.0,
                2.0
            ]
        ],
        "weights": [
            [
                [
                    0.10120000000000001,
                    0.3006,
                    0.1991
                ],
                [
                    0.40240000000000004,
```

```
                    0.201,
                    -0.7019
                ],
                [
                    0.1018,
                    -0.799,
                    0.4987
                ]
            ]
        ]
    },
    "expect": {
        "output": [
            [
                2.0,
                0.3,
                -1.9
            ],
            [
                1.3,
                -0.7,
                0.1
            ]
        ],
        "max_sse": 0.0
    }
}
```

Test Case linear_two_iteration.json

Model:

```
{
    "case": {
        "model": {
            "input_size": 2,
            "layers": [
                {
                    "number_of_neurons": 3,
```

```
                    "activation_function": "linear"
                }
            ]
        },
        "input": [
            [3.0, 1.0],
            [1.0, 2.0]
        ],
        "initial_weights": [
            [
              [0.1, 0.3, 0.2],
              [0.4, 0.2, -0.7],
              [0.1, -0.8, 0.5]
            ]
        ],
        "target": [
            [ 2.0,  0.3, -1.9],
            [ 1.3, -0.7,  0.1]
        ],
        "learning_parameters": {
            "learning_rate": 0.1,
            "batch_size": 2,
            "max_iteration": 2,
            "error_threshold": 0.0
        }
    },
    "expect": {
        "stopped_by": "max_iteration",
        "final_weights": [
            [
              [ 0.166,  0.338, 0.153],
              [ 0.502,  0.226, -0.789],
              [ 0.214, -0.718,  0.427]
            ]
        ]
    }
}
```

Hasil:

```
{
    "case": {
        "model": {
            "input_size": 2,
            "layers": [
                {
                    "number_of_neurons": 3,
                    "activation_function": "linear"
                }
            ]
        },
        "input": [
            [
                3.0,
                1.0
            ],
            [
                1.0,
                2.0
            ]
        ],
        "weights": [
            [
                [
                    0.16599999999999998,
                    0.338,
                    0.153
                ],
                [
                    0.5019999999999999,
                    0.22600000000000003,
                    -0.7889999999999999
                ],
                [
                    0.21400000000000005,
                    -0.718,
                    0.427
                ]
```

```
            ]
        ]
    },
    "expect": {
        "output": [
            [
                2.0,
                0.3,
                -1.9
            ],
            [
                1.3,
                -0.7,
                0.1
            ]
        ],
        "max_sse": 0.0
    }
}
```

Test Case sigmoid.json

Model:

```
{
    "case": {
        "model": {
            "input_size": 2,
            "layers": [
                {
                    "number_of_neurons": 2,
                    "activation_function": "sigmoid"
                }
            ]
        },
        "input": [
            [0.0, 0.0],
            [0.0, 0.1]
        ],
```

```
        "initial_weights": [
            [
              [0.1, 0.2],
          [0.4, 0.5],
          [0.9, 0.1]
            ]
        ],
        "target": [
            [0.1, 1.0],
            [1.0, 0.0]
        ],
        "learning_parameters": {
            "learning_rate": 0.1,
            "batch_size": 2,
            "max_iteration": 1,
            "error_threshold": 0.1
        }
    },
    "expect": {
        "stopped_by": "max_iteration",
      "final_weights":[
         [
            [0.7125, -1.1742],
            [2.3583, -0.2848],
            [0.1112, -1.1396]
         ]
      ]
    }
}
```

Hasil:

```
{
    "case": {
        "model": {
            "input_size": 2,
            "layers": [
                {
```

```
                    "number_of_neurons": 2,
                    "activation_function": "sigmoid"
                }
            ]
        },
        "input": [
            [
                0.0,
                0.0
            ],
            [
                0.0,
                0.1
            ]
        ],
        "weights": [
            [
                [
                    0.10061658075420078,
                    0.1974857748026805
                ],
                [
                    0.4,
                    0.5
                ],
                [
                    0.901121454344799,
                    0.09863434201890192
                ]
            ]
        ]
    },
    "expect": {
        "output": [
            [
                0.1,
                1.0
            ],
            [
```

```
                1.0,
                0.0
            ]
        ],
        "max_sse": 0.1
    }
}
```

Test Case sigmoid_mini_batch_GD.JSON

Model:

```
{
    "case": {
        "model": {
            "input_size": 2,
            "layers": [
                {
                    "number_of_neurons": 2,
                    "activation_function": "sigmoid"
                }
            ]
        },
        "input": [
            [0.0, 0.0],
            [0.0, 0.1],
        [1.0, 0.0],
        [1.0, 1.0]
        ],
        "initial_weights": [
            [
              [0.15, 0.25],
          [0.2, 0.3],
          [0.35, 0.35]
            ]

        ],
        "target": [
            [0.1, 1.0],
```

```
            [1.0, 0.0],
        [1.0, 1.0],
        [0.0, 0.0]
        ],
        "learning_parameters": {
            "learning_rate": 0.1,
            "batch_size": 2,
            "max_iteration": 10000,
            "error_threshold": 0.2
        }
    },
    "expect": {
        "stopped_by": "error_threshold"
    }
}
```

Hasil:

```
{
    "case": {
        "model": {
            "input_size": 2,
            "layers": [
                {
                    "number_of_neurons": 2,
                    "activation_function": "sigmoid"
                }
            ]
        },
        "input": [
            [
                0.0,
                0.0
            ],
            [
                0.0,
                0.1
            ],
```

```
            [
                1.0,
                0.0
            ],
            [
                1.0,
                1.0
            ]
        ],
        "weights": [
            [
                [
                    0.3971925369083511,
                    0.4044854813273678
                ],
                [
                    1.162907647618478,
                    2.477929132462899
                ],
                [
                    -3.1315764933116856,
                    -7.978430009400752
                ]
            ]
        ]
    },
    "expect": {
        "output": [
            [
                0.1,
                1.0
            ],
            [
                1.0,
                0.0
            ],
            [
                1.0,
                1.0
```

```
            ],
            [
                0.0,
                0.0
            ]
        ],
        "max_sse": 0.2
    }
}
```

Test Case sigmoid_stochastic_GD.JSON

Model:

```
{
    "case": {
        "model": {
            "input_size": 2,
            "layers": [
                {
                    "number_of_neurons": 2,
                    "activation_function": "sigmoid"
                }
            ]
        },
        "input": [
            [0.0, 0.0],
            [0.0, 0.1],
        [1.0, 0.0],
        [1.0, 1.0]
        ],
        "initial_weights": [
            [
              [0.15, 0.25],
          [0.2, 0.3],
          [0.35, 0.35]
            ]
        ],
        "target": [
```

```
            [0.1, 1.0],
            [1.0, 0.0],
        [1.0, 1.0],
        [0.0, 0.0]
        ],
        "learning_parameters": {
            "learning_rate": 0.1,
            "batch_size": 1,
            "max_iteration": 10000,
            "error_threshold": 0.1
        }
    },
    "expect": {
        "stopped_by": "error_threshold"
    }
}
```

Hasil:

```
{
    "case": {
        "model": {
            "input_size": 2,
            "layers": [
                {
                    "number_of_neurons": 2,
                    "activation_function": "sigmoid"
                }
            ]
        },
        "input": [
            [
                0.0,
                0.0
            ],
            [
                0.0,
                0.1
```

```
            ],
            [
                1.0,
                0.0
            ],
            [
                1.0,
                1.0
            ]
        ],
        "weights": [
            [
                [
                    0.05751529943716102,
                    1.3824252205846155
                ],
                [
                    1.632549139403422,
                    3.1649986451680863
                ],
                [
                    -2.889335832142161,
                    -25.56506608619053
                ]
            ]
        ]
    },
    "expect": {
        "output": [
            [
                0.1,
                1.0
            ],
            [
                1.0,
                0.0
            ],
            [
                1.0,
```

```
                1.0
            ],
            [
                0.0,
                0.0
            ]
        ],
        "max_sse": 0.1
    }
}
```

Test Case relu.json

Model:

```
{
    "case": {
        "model": {
            "input_size": 2,
            "layers": [
                {
                    "number_of_neurons": 3,
                    "activation_function": "relu"
                }
            ]
        },
        "input": [
            [-1.0, 0.5],
            [0.5, -1.0]
        ],
        "initial_weights": [
            [
                [0.1, 0.2, 0.3],
                [0.4, -0.5, 0.6],
                [0.7, 0.8, -0.9]
            ]
        ],
        "target": [
            [0.1, 1.0, 0.1],
```

```
            [0.1, 0.1, 1.0]
        ],
        "learning_parameters": {
            "learning_rate": 0.1,
            "batch_size": 2,
            "max_iteration": 1,
            "error_threshold": 0.0
        }
    },
    "expect": {
        "stopped_by": "max_iteration",
        "final_weights": [
            [
                [0.105, 0.19, 0.25],
                [0.395, -0.49, 0.575],
                [0.7025, 0.795, -0.85]
            ]
        ]
    }
}
```

Hasil:

```
{
    "case": {
        "model": {
            "input_size": 2,
            "layers": [
                {
                    "number_of_neurons": 3,
                    "activation_function": "relu"
                }
            ]
        },
        "input": [
            [
                -1.0,
                0.5
```

```
            ],
            [
                0.5,
                -1.0
            ]
        ],
        "weights": [
            [
                [
                    0.10500000000000001,
                    0.19,
                    0.25
                ],
                [
                    0.395,
                    -0.49,
                    0.575
                ],
                [
                    0.7025,
                    0.795,
                    -0.85
                ]
            ]
        ]
    },
    "expect": {
        "output": [
            [
                0.1,
                1.0,
                0.1
            ],
            [
                0.1,
                0.1,
                1.0
            ]
        ],
```

```
        "max_sse": 0.0
    }
}
```

Test Case softmax.json

Model:

```
{
    "case": {
        "model": {
            "input_size": 2,
            "layers": [
                {
                    "number_of_neurons": 2,
                    "activation_function": "softmax"
                }
            ]
        },
        "input": [
            [-1.0, 0.5],
            [0.5, -1.0]
        ],
        "initial_weights": [
            [
                [0.1, 0.2],
                [0.4, -0.5],
                [0.7, 0.8]
            ]
        ],
        "target": [
            [1.0, 0],
            [0, 1.0]
        ],
        "learning_parameters": {
            "learning_rate": 0.1,
            "batch_size": 2,
            "max_iteration": 1,
            "error_threshold": 0.0
```

```
        }
    },
    "expect": {
        "stopped_by": "max_iteration",
        "final_weights": [
            [
                [0.11301357, 0.18698643],
                [0.29539055, -0.39539055],
                [0.79810267, 0.70189733]
            ]
        ]
    }
}
```

Hasil:

```
{
    "case": {
        "model": {
            "input_size": 2,
            "layers": [
                {
                    "number_of_neurons": 2,
                    "activation_function": "softmax"
                }
            ]
        },
        "input": [
            [
                -1.0,
                0.5
            ],
            [
                0.5,
                -1.0
            ]
        ],
        "weights": [
```

```
            [
                [
                    0.11301356652329321,
                    0.1869864334767068
                ],
                [
                    0.2953905483843235,
                    -0.3953905483843235
                ],
                [
                    0.7981026683540299,
                    0.7018973316459701
                ]
            ]
        ]
    },
    "expect": {
        "output": [
            [
                1.0,
                0
            ],
            [
                0,
                1.0
            ]
        ],
        "max_sse": 0.0
    }
}
```

Test Case softmax_error_only.json

Model:

```
{
    "case": {
        "model": {
            "input_size": 1,
```

```
            "layers": [
                {
                    "number_of_neurons": 2,
                    "activation_function": "softmax"
                }
            ]
        },
        "input": [
            [-1.0]
        ],
        "initial_weights": [
            [
                [0.4, 0.7],
                [0.2, -0.9]
            ]
        ],
        "target": [
            [0.0, 1.0]
        ],
        "learning_parameters": {
            "learning_rate": 0.1,
            "batch_size": 1,
            "max_iteration": 1,
            "error_threshold": 0.225
        }
    },
    "expect": {
        "stopped_by": "error_threshold",
        "final_weights": [
            [
                [0.4, 0.7],
                [0.2, -0.9]
            ]
        ]
    }
}
```

Hasil:

```
{
    "case": {
        "model": {
            "input_size": 1,
            "layers": [
                {
                    "number_of_neurons": 2,
                    "activation_function": "softmax"
                }
            ]
        },
        "input": [
            [
                -1.0
            ]
        ],
        "weights": [
            [
                [
                    0.4,
                    0.7
                ],
                [
                    0.2,
                    -0.9
                ]
            ]
        ]
    },
    "expect": {
        "output": [
            [
                0.0,
                1.0
            ]
        ],
        "max_sse": 0.225
    }
}
```

Test Case mlp.json

Model:


```
{
    "case": {
        "model": {
            "input_size": 2,
            "layers": [
                {
                    "number_of_neurons": 2,
                    "activation_function": "linear"
                },
                {
                    "number_of_neurons": 2,
                    "activation_function": "relu"
                }
            ]
        },
        "input": [
            [-1.0, 0.5],
            [0.5, -1.0]
        ],
        "initial_weights": [
            [
                [0.1, 0.2],
                [0.4, -0.5],
                [0.7, 0.8]
            ],
            [
                [0.1, 0.2],
                [0.4, -0.5],
                [0.7, 0.8]
            ]
        ],
        "target": [
            [0.1, 1.0],
            [1.0, 0.1]
        ],
```

```
        "learning_parameters": {
            "learning_rate": 0.1,
            "batch_size": 2,
            "max_iteration": 1,
            "error_threshold": 0.0
        }
    },
    "expect": {
        "stopped_by": "max_iteration",
        "final_weights": [
            [
                [0.07115, 0.1403],
                [0.42885, -0.4403],
                [0.685575, 0.77015]
            ],
            [
                [0.121, 0.2045],
                [0.35605, -0.504275],
                [0.5281, 0.78545]
            ]
        ]
    }
}
```

Hasil:

```
{
    "case": {
        "model": {
            "input_size": 2,
            "layers": [
                {
                    "number_of_neurons": 2,
                    "activation_function": "linear"
                },
                {
                    "number_of_neurons": 2,
                    "activation_function": "relu"
```

```
                }
            ]
        },
        "input": [
            [
                -1.0,
                0.5
            ],
            [
                0.5,
                -1.0
            ]
        ],
        "weights": [
            [
                [
                    0.07115,
                    0.1403
                ],
                [
                    0.42885,
                    -0.44029999999999997
                ],
                [
                    0.6855749999999999,
                    0.77015
                ]
            ],
            [
                [
                    0.02099999999999999,
                    0.1945
                ],
                [
                    0.39605,
                    -0.500275
                ],
                [
                    0.6131,
```

```
                    0.79395
                ]
            ]
        ]
    },
    "expect": {
        "output": [
            [
                0.1,
                1.0
            ],
            [
                1.0,
                0.1
            ]
        ],
        "max_sse": 0.0
    }
}
```

Test Case sse_only.json

Model:

```
{
    "case": {
        "model": {
            "input_size": 1,
            "layers": [
                {
                    "number_of_neurons": 2,
                    "activation_function": "linear"
                }
            ]
        },
        "input": [
            [0.0]
        ],
        "initial_weights": [
```

```
            [
                [0.5, 0.5],
                [0.0, 0.0]
            ]
        ],
        "target": [
            [0.0, 1.0]
        ],
        "learning_parameters": {
            "learning_rate": 0.01,
            "batch_size": 1,
            "max_iteration": 10000,
            "error_threshold": 0.25
        }
    },
    "expect": {
        "stopped_by": "error_threshold",
        "final_weights": [
            [
                [0.5, 0.5],
                [0.0, 0.0]
            ]
        ]
    }
}
```

Hasil:

```
{
    "case": {
        "model": {
            "input_size": 1,
            "layers": [
                {
                    "number_of_neurons": 2,
                    "activation_function": "linear"
                }
            ]
```

```
        },
        "input": [
            [
                0.0
            ]
        ],
        "weights": [
            [
                [
                    0.5,
                    0.5
                ],
                [
                    0.0,
                    0.0
                ]
            ]
        ]
    },
    "expect": {
        "output": [
            [
                0.0,
                1.0
            ]
        ],
        "max_sse": 0.25
    }
}
```

Perbandingan dengan MLP sklearn

```
MLP Sklearn
Accuracy: 0.6666666666666666
F1: 0.5555555555555555
Confusion Matrix:
Class ANN
Accuracy: 0.7666666666666667
F1 Score: 0.7340930674264008
```

Nilai yang didapat untuk sklearn bergantung dengan parameter random_state sedangkan untuk class ANN yang sudah dibuat di tubes B bergantung dengan shuffling pada urutan data (yang berdampak terhadap pembelajaran) serta inisialisasi bobot awal. Kedua perbandingan juga menggunakan parameter masukan yang sama (nilai ini bisa dibuat lebih baik dengan meningkatkan max iteration).

Bab III
Pembagian Tugas

NIM	Nama	Tugas
13520133	Jevant Jedidia Augustine	Laporan, Integrasi dengan FFNN, Perbandingan dengan Sklearn, Fungsi Turunan Softmax, Training Dataset Iris
13520136	Vincent Christian Siregar	Backward Propagation, Perhitungan Gradient, Mini Batch, Training Dataset Iris
13520154	David Karel Halomoan	Laporan, Save Model, Fungsi Turunan, Fungsi Loss
13520160	Willy Wilsen	Laporan, Read Input, Backward Propagation, Perhitungan Gradient