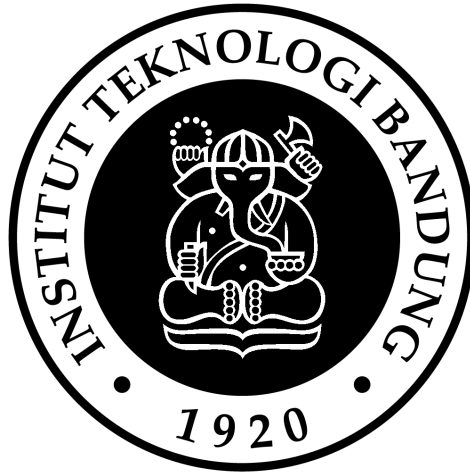


LAPORAN TUGAS BESAR B

“Implementasi Mini-batch Gradient Descent”

Dibuat untuk Memenuhi Tugas Mata Kuliah IF3270 Pembelajaran Mesin



Disusun Oleh:

| | |
|----------|--------------------------|
| 13520130 | Nelsen Putra |
| 13521117 | Maggie Zeta RS |
| 13521133 | Cetta Reswara Parahita |
| 13521161 | Ferindya Aulia Berlianty |

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024

DAFTAR ISI

| | |
|-----------------------------------------------------------|-----------|
| DAFTAR ISI | 2 |
| DASAR TEORI | 3 |
| A. Mini-batch Gradient Descent | 3 |
| B. Backpropagation | 4 |
| C. Mini-batch Gradient Descent dengan Backpropagation | 4 |
| PENJELASAN IMPLEMENTASI | 5 |
| HASIL PENGUJIAN | 16 |
| PERBANDINGAN DENGAN PENGGUNAAN LIBRARY MLP SKLEARN | 21 |
| A. Library MLP Sklearn | 21 |
| B. Uji Kasus Data Iris | 22 |
| C. Analisis Perbandingan | 24 |
| LAMPIRAN | 27 |
| A. Repositori | 27 |
| B. Pembagian Kerja | 27 |
| C. Link Google Colab: Penggunaan Library MLP Sklearn | 28 |
| DAFTAR REFERENSI | 29 |

DASAR TEORI

A. *Mini-batch Gradient Descent*

Gradient Descent (turunan gradien) adalah metode optimasi yang esensial dalam pembelajaran mesin, yang digunakan untuk meminimalkan fungsi biaya. Proses ini melibatkan iterasi pergerakan dalam arah gradien negatif, yang ditentukan oleh turunan dari fungsi biaya tersebut. Dalam konteks jaringan saraf tiruan, turunan gradien digunakan untuk menemukan parameter (bobot) yang optimal yang meminimalkan fungsi kerugian.

Mini-batch Gradient Descent adalah variasi dari algoritma *gradient descent* standar yang memperbarui parameter model berdasarkan subset dari data daripada seluruh dataset sekaligus. Pendekatan ini menciptakan keseimbangan antara kelebihan dari *stochastic gradient descent*, yang menggunakan satu sampel per pembaruan, dan *batch gradient descent*, yang menggunakan seluruh dataset untuk menghitung pembaruan.

Proses *Mini-batch Gradient Descent*:

1. Data pelatihan dibagi menjadi beberapa *mini-batch* yang masing-masing terdiri dari sejumlah sampel terbatas.
2. Untuk setiap *mini-batch*, gradient fungsi kerugian dihitung hanya berdasarkan sampel dalam *mini-batch* tersebut.
3. Bobot jaringan di-*update* berdasarkan *gradient* yang dihitung, dengan tujuan mengurangi nilai kerugian.

Keuntungan *Mini-batch Gradient Descent*:

1. Dengan menggunakan subset data (*mini-batch*), ia mengurangi varians pembaruan parameter, yang dapat mengarah pada konvergensi yang lebih stabil dibandingkan dengan *stochastic gradient descent*.
2. *Mini-batch* memanfaatkan arsitektur komputasi yang sangat paralel di lingkungan komputasi modern, memungkinkan perhitungan secara paralel.
3. *Mini-batch gradient descent* cenderung konvergen lebih cepat daripada *stochastic gradient descent* ketika menghadapi dataset yang besar.

B. *Backpropagation*

Backpropagation adalah algoritma yang digunakan untuk melatih jaringan saraf tiruan, di mana *gradient* dari fungsi kerugian dihitung untuk setiap bobot dalam jaringan dengan tujuan mengoptimalkan bobot tersebut agar fungsi kerugian seminimal mungkin. Beberapa komponen utama pada *Backpropagation*, yaitu *Chain Rule* dan Fungsi Aktivasi. *Chain Rule* digunakan untuk secara efisien menghitung *gradient* melalui lapisan-lapisan jaringan. Sedangkan, fungsi aktivasi merupakan fungsi non-linear yang diterapkan pada setiap neuron, contoh umum termasuk sigmoid dan ReLU. Gradien dari fungsi ini sangat penting selama fase *backward pass*. *Backpropagation* memungkinkan jaringan saraf tiruan untuk belajar dari kesalahan (*error*) dan melakukan penyesuaian pada bobotnya, sehingga output jaringan semakin mendekati output yang diharapkan.

Proses *Backpropagation* :

1. Menghitung output jaringan berdasarkan input dan bobot saat ini (*forward pass*).
2. Menghitung *gradient* fungsi kerugian terhadap setiap bobot dengan mengikuti arah terbalik dari jaringan (*backward pass*).
3. Bobot diperbarui dengan menggunakan *gradient* yang telah dihitung, biasanya dengan metode *Gradient Descent*.

C. *Mini-batch Gradient Descent* dengan *Backpropagation*

Dengan menggunakan kedua teknik ini dalam satu *framework* melalui implementasi *Mini-batch Gradient Descent* yang menggunakan *backpropagation* memberikan pendekatan yang kuat dalam meminimalkan fungsi kerugian dan meningkatkan kemampuan prediksi dari model ANN. Implementasi ini harus mempertimbangkan faktor-faktor seperti ukuran *batch*, tingkat pembelajaran (*learning rate*), dan *threshold error* untuk menentukan kapan iterasi harus dihentikan agar efektif dan efisien dalam pelatihan jaringan saraf tiruan.

PENJELASAN IMPLEMENTASI

Dalam mengimplementasikan *backpropagation* pada algoritma FFNN, dibuat struktur repositori sebagai berikut:

```
|— README.md
|— doc
|   |— TubesA_13520130.pdf
|   |— TubesB_13520130.pdf
|— main.ipynb
|— model
|— src
|   |— activation.py
|   |— ffnn.py
|   |— gradient_descent.py
|   |— gradient_equation.py
|   |— hidden_layer.py
|   |— input_json.py
|   |— layer.py
|   |— loss_function.py
|   |— node.py
|— temporary.json
|— test
    |— b_linear.json
    |— b_linear_small_lr.json
    |— b_linear_two_iteration.json
    |— b_mlp.json
    |— b_relu.json
    |— b_sigmoid.json
    |— b_softmax.json
    |— b_softmax_two_layer.json
    |— linear.json
    |— multilayer.json
    |— multilayer_softmax.json
    |— relu.json
```

```

├── relu_1.json
├── sigmoid.json
└── softmax.json

```

Folder 'src' berisi file-file kode dalam bahasa python yang masing-masing file berisi kelas-kelas modul sesuai dengan namanya untuk menangani struktur kelas yang diperlukan berikut fungsi-fungsi yang sering digunakan dalam mengakses tiap struktur kelas yang ada. Folder 'test' berisi test case dalam bahasa json sejumlah 8 dengan struktur:

```

{
  "case": {
    "model": {
      "input_size": input data,
      "layers": [ array of
        {
          "number_of_neurons": total neuron,
          "activation_function": fungsi aktivasi
        }
      ]
    },
    "input": [array of array input dengan input size yang sudah ditetapkan],
    "weights": [array of weights array for all layers],
    "target": [array of output target for each input],
    "learning_parameters": [array of
      - learning_rate
      - batch_size
      - max_iteration
      - error_threshold
    ]
  },
  "expect": {
    "stopped_by": condition where the learning stop (max_iteration or error_threshold),
    "final_weights": final weight after all iterations
  }
}

```

Dalam bagian backpropagation ini beberapa perubahan yang dilakukan adalah **update** pada kelas FFNN dan juga pembuatan **algoritma gradient descent** dan **loss function**. Dibuat juga fungsi **create_json()** dan **update_json()** pada file input_json.py untuk membuat file json terbaru yang didapatkan dari hasil training dengan backpropagation dan mengupdate weight setiap sebuah iterasi epoch diselesaikan.

Pada kelas FFNN, penambahan yang dilakukan adalah seperti cuplikan dari kode di ffnn.py berikut:

```
class FFNN:
    '''Kelas untuk membuat model ffnn'''
    ...
    def __init__(self, case):
        self.input_size = case.get("model").get("input_size")
        self.nlayers = len(case.get("model").get("layers"))
        self.layers = HiddenLayer(case.get("model").get("layers"), case.get("weights"))
        self.noutput = self.layers.n_output
        self.weight = case.get("weights")
        self.output = []
```

Pada FFNN ditambahkan tiga atribut yakni (1) noutput berupa jumlah output yang akan dihasilkan oleh model, (2) weight yakni array weight yang disimpan dari *case* yang diberikan, dan (3) output yang berisi output dari setiap layernya Selanjutnya diimplementasikan beberapa fungsi tambahan untuk melakukan proses predicting dan training dalam FFNN sebagai berikut:

```
def forward_propagation_ffnn(input_file_name):
    checked_file_name = file_name(input_file_name)
    case, expect = open_json(checked_file_name)
    model = FFNN(case)
    input = case.get("input")
    output = model.predict(input)
    output_val = model.output_value(output)
    #sse = FFNN.f_sse(output_val, expect.get("output"))
    return model, input, output, output_val, case, expect
```

Pertama, fungsi **forward_propagation_ffnn()**. Fungsi ini adalah fungsi yang sebelumnya telah diimplementasikan pada Tugas Besar A di .ipynb dan dibuat lebih modular agar dapat dimanfaatkan dalam tiap epoch yang akan dilakukan saat backpropagation.

```

def backpropagation(learning_rate, target, batch, weight):
    updated_weights = np.zeros_like(batch[0].weight)
    delta_all = []
    activation = batch[0].layers.layers[-1].activation_function_name

    for i in range(batch[0].nlayers):
        back_index = batch[0].nlayers - 1 - i

        if i == 0 : # output layer
            output = []
            input = []
            weight_dim = []

            for i in range(len(batch)):
                output.append(batch[i].layers.layers[back_index].value)
                if (batch[0].nlayers > 1 and i != batch[0].nlayers - 1):
                    input.append([1] + batch[i].layers.layers[back_index-1].value)
                else:
                    input.append(batch[i].layers.layers[back_index-1].value)
                weight_dim.append(weight[-1])
            delta, update = gd_o(target, output, input, weight_dim, learning_rate, activation)
            to_update_weights = update[0]
            for i in range(1, len(update)):
                to_update_weights += update[i]
            updated_weights[back_index] = to_update_weights
            delta_all.append(delta)

        else: #hidden layer
            output = []
            input = []
            weight_dim = []
            weight_dim_next = []

            for j in range(len(batch)):
                if (batch[0].nlayers > 1): #and i != batch[0].nlayers-1:
                    input.append([1] + batch[j].layers.layers[back_index-1].value)
                    output.append([1] + batch[j].layers.layers[back_index].value)
                else:
                    input.append(batch[j].layers.layers[back_index-1].value)
                    output.append(batch[j].layers.layers[back_index].value) # Next layer's output
                weight_dim.append(weight[back_index])
                weight_dim_next.append(weight[back_index + 1])

            # Calculate delta and weight update for the hidden layer
            delta, update = gd_h(output, input, delta_all[-1], weight_dim, weight_dim_next, learning_rate, activation)
            to_update_weights = update[0]
            for i in range(1, len(update)):
                to_update_weights += update[i]
            #print(type(to_update_weights))
            updated_weights[back_index] = to_update_weights
            delta_all.append(delta)

    return updated_weights

```

Selanjutnya fungsi **backpropagation()** akan melakukan mini batch backpropagation dengan masukan berupa learning rate, target, batch, dan weight. Batch merupakan model-model yang telah dirandomisasi melalui fungsi **create_and_randomize_batch()** sedangkan weight adalah array weight yang sebelumnya telah disimpan di model. Fungsi backpropagation akan

memproses dan menyimpan gradient descent yang didefinisikan pada file `gradient_descent.py` kemudian mengembalikan delta weight yang akan diproses pada fungsi **train()**.

```
def create_and_randomize_batch(total_input, batch_size):
    batches_num=[]
    total_input_array = np.arange(total_input)
    np.random.shuffle(total_input_array)

    total_of_batches=total_input // batch_size
    for i in range(total_of_batches):
        batch_indices = total_input_array[i * batch_size: (i + 1) * batch_size]
        batches_num.append(batch_indices)
    if total_input % batch_size != 0:
        batch_indices = total_input_array[total_of_batches * batch_size:]
        batches_num.append(batch_indices)

    return batches_num
```

Berikut adalah fungsi **create_and_randomize_batch()** yang akan melakukan randomisasi dengan `np.random.shuffle()` untuk semua indeks input dan membaginya sesuai dengan batch size yang didefinisikan pada input `.json`.

Backpropagation akan dimanfaatkan dalam proses training pada fungsi **train()**. Fungsi ini akan melakukan iterasi sesuai dengan jumlah iterasi maksimal yang dijelaskan pada file `.json`. Pada tiap iterasi, langkah-langkah yang dilakukan adalah:

- (1) Melakukan forward propagation untuk mendapatkan output model
- (2) Melakukan pengecekan apakah error/ loss yang didapatkan dari model yang telah ditrain telah berada di bawah *error threshold*. Apabila telah memenuhi, maka model akan berhenti melakukan iterasi epoch dan mengembalikan hasil akhir weight yang telah diupdate
- (3) Apabila belum, maka akan dilakukan **mini-batch backpropagation** dengan melakukan pembagian batch, kemudian memanggil fungsi **backpropagation()** sesuai dengan batch yang sedang dicek
- (4) Setiap batch ditrain, weight akan di *update* kemudian setelah semua batch selesai maka akan dilanjutkan pada iterasi selanjutnya.
- (5) Apabila maximal iteration telah dicapai namun error threshold belum terpenuhi, maka program akan diterminasi dan mengembalikan final weight yang dapat telah didapatkan.

Fungsi **train()** yang diimplementasikan dapat dilihat pada kode berikut:

```

def train(input_file_name):
    # variables
    model, input, output, output_val, case, expect = FFNN.forward_propagation_ffnn(input_file_name)
    weight = case.get("weights")
    target = case.get("target")
    learning_parameters = case.get("learning_parameters")
    learning_rate = learning_parameters.get("learning_rate")
    batch_size = learning_parameters.get("batch_size")
    max_iteration = learning_parameters.get("max_iteration")
    error_threshold = learning_parameters.get("error_threshold")
    stopped_by = "max_iteration"
    error = error_threshold+1

    temporary_file_name = 'temporary.json'
    create_json(case, expect, temporary_file_name)

    for i in range(max_iteration):
        if i != 0 :
            # model, input, output, output_val, case = FFNN.forward_propagation_ffnn(input_file_name)
            # weight = case.get("weights")
        # else :
            model, input, output, output_val, case, expect = FFNN.forward_propagation_ffnn(temporary_file_name)
            weight = case.get("weights")

        mse = 0
        if model.layers.layers[-1].activation_function_name == "softmax":
            for j in range(len(output_val)):
                error = 0
                for k in range(len(output_val[j])):
                    error += cross_entropy(case.get("target")[j][k], output_val[j][k])
                mse += error
        else:
            for j in range(len(output_val)):
                error = 0
                for k in range(len(output_val[j])):
                    error += sse(case.get("target")[j][k], output_val[j][k])
                mse += error
        mse /= len(output_val)

        if (error <= error_threshold):
            stopped_by == "error_threshold"
            break
        else:
            # do backpropagation mini-batch
            batches = FFNN.create_and_randomize_batch(len(input), batch_size)
            weight_updates= np.zeros_like(weight)

            for batch in batches:
                #prosses backpropagation sampe update per batch
                batch_weights_updates = np.zeros_like(weight)
                batch_ffnn = []
                for i in range(len(batch)):
                    batch_ffnn.append(output[i])
                batch_weights_updates = FFNN.backpropagation(learning_rate, target, batch_ffnn, weight)

```

```

        weight_updates += batch_weights_updates

    #update weights
    #print(weight_updates)
    new_weight = weight + weight_updates

    #print(new_weight)

    #overwrite temporary files for next batch
    update_weight_final = []

    for i in range(len(new_weight)):
        update_weight_final.append(new_weight[i].tolist())

    update_json(temporary_file_name, update_weight_final)

case, expect = open_json(temporary_file_name)
final_weights = case.get("weights")

return stopped_by, final_weights

```

Untuk memastikan backpropagation dapat dijalankan dengan maksimal, maka kami melakukan penurunan fungsi-fungsi aktivasi dan loss function berikut dalam **loss_function.py** dan **activation.py**.

a. Turunan fungsi aktivasi:

| Nama Fungsi Aktivasi | Turunan |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| ReLU | $\frac{d}{dx}ReLU(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$ |
| Sigmoid | $\frac{d}{dx}\sigma(x) = \sigma(x) \times (1 - \sigma(x))$ |
| Linear | $\frac{d}{dx}x = 1$ |
| Softmax* | $\frac{\partial E_d}{\partial net(x)} = \begin{cases} p_j, & j \neq \text{target} \\ -(1 - p_j), & j = \text{target} \end{cases}$ |

Implementasi pada **activation.py**

```
import numpy as np

## ----- Activation Functions -----
def linear(net, derivative=False):
    if derivative:
        return 1
    else:
        return net

def relu(net, derivative=False):
    if derivative:
        return 1 if net > 0 else 0
    else:
        return max(0, net)

def sigmoid(net, derivative=False):
    fsigmoid = (1 / (1 + np.exp(-net)))
    if derivative:
        return net * (1 - net)
    else:
        return fsigmoid

def softmax(net, numOfNet=1, derivative=False, target=False):
    if derivative:
        return net * (1 - net)
    else:
        sigma = 0
        for i in range(len(numOfNet)):
            sigma += np.exp(numOfNet[i])
        fsoftmax = np.exp(net) / sigma
        return fsoftmax
```

b. Fungsi loss:

| Fungsi Aktivasi | Loss |
|---------------------------|------------------------------------------------------------|
| ReLU, sigmoid, dan linear | $E = \frac{1}{2} \sum_{k \in \text{output}} (t_k - o_k)^2$ |
| Softmax | $E = -\log(p_k), \text{ k=target}$ |

Implementasi pada `loss_function.py`

```
import numpy as np

# For ReLU, sigmoid, and linear
def sse(target, output, derivative=False):
    if (derivative):
        return (output - target)
    else:
        return 0.5 * ((target - output) ** 2)

# For softmax
def cross_entropy(target, output, derivative=False):
    if (derivative):
        return -(target / output) + ((1 - target) / (1 - output))
    else:
        return -np.sum([target * np.log(output)])
```

Fungsi-fungsi di atas diimplementasikan pada `gradient_descent.py` sebagai fungsi `gd_o()` untuk menghitung gradient descent dari layer output dengan menerapkan fungsi yang diturunkan dari persamaan berikut:

$$\frac{dE}{dw} = \frac{dE}{dOut} \frac{dOut}{dnet} \frac{dnet}{dw}$$

Menjadi persamaan yang lebih sederhana sebagai berikut:

$$\begin{aligned} \frac{dSSR}{dw_x} &= \frac{\sum (target - predict)^2}{dw_x} \\ &= \frac{\sum (target - predict)^2}{d predict} \cdot \frac{d predict}{dw_x} \\ &= \sum -2 (target - predict) \cdot \frac{d predict}{dw_x} \\ &= \sum -2 (target - predict) \cdot \frac{df(\sum x)}{d \sum x} \\ &= \sum -2 (target - predict) \cdot \frac{df(\sum x)}{d \sum x} \cdot \frac{d \sum x}{dw_x} \\ &= \sum -2 (target - predict) \cdot \frac{df(x)}{d \sum x} \cdot \text{input (output hidden n...)} \end{aligned}$$

Berikut adalah hasil implementasi dari gradient descent untuk output layer:

```
import numpy as np
from activation import linear, relu, sigmoid, softmax
from loss_function import sse, cross_entropy

def gd_o (target, output, input, weight, learning_rate, activation):

    activation_function_list = {
        "linear" : linear,
        "relu" : relu,
        "sigmoid" : sigmoid,
        "softmax" : softmax,
        "None" : None
    }
    activation_function = activation_function_list[activation]

    dssr_dw = np.zeros_like(output)
    for i in range(len(output)):
        for j in range(len(output[i])):
            if (activation == "softmax"):
                dssr_dw[i][j] += -1 * cross_entropy(target[i][j], output[i][j], True) * activation_function(output[i][j], derivative=True)
            else:
                dssr_dw[i][j] += -1 * sse(target[i][j], output[i][j], True) * activation_function(output[i][j], derivative=True)

    delta_w = np.zeros_like(weight)
    for k in range(len(input)):
        for i in range(len(weight[k])):
            for j in range(len(weight[k][i])):
                if (activation == "softmax"):
                    delta_w[k][i][j] += -learning_rate * activation_function(output[k][j], derivative=True) * input[k][i]
                else:
                    delta_w[k][i][j] += -learning_rate * sse(target[k][j], output[k][j], True) * activation_function(output[k][j], derivative=True) * input[k][i]

    return dssr_dw, delta_w
```

Selanjutnya diimplementasikan pula untuk gradient descent pada hidden layer dengan melakukan penurunan fungsi berikut:

$$\frac{dE}{dw} = \frac{dE}{dnet} \frac{dnet}{dw}$$

Menjadi persamaan yang lebih sederhana sebagai berikut:

$$\begin{aligned} \frac{de}{dw} &= \frac{de}{dnet} \cdot \frac{dnet}{dw} \\ &= \underbrace{\frac{de}{dnet}}_{\text{fungsi}} \cdot \underbrace{\frac{df(\text{net})}{d \text{sum}}}_{\text{fungsi}} \cdot \underbrace{\frac{d \text{sum}}{dw}}_{\text{input}} \end{aligned}$$

Berikut adalah hasil implementasi dari gradient descent untuk hidden layer:

```
def gd_h(output, input, gd_h1, weight, weight_next, learning_rate, activation):
    activation_function_list = {
        "linear" : linear,
        "relu" : relu,
        "sigmoid" : sigmoid,
        "softmax" : softmax,
        "None" : None
    }
    activation_function = activation_function_list[activation]

    de_dw = np.zeros_like(output)
    for i in range(len(output)):
        for j in range(len(output[i])):
            de_dw[i][j] += np.dot(gd_h1[i] * activation_function(output[i][j], derivative=True), weight_next[i][j])

    delta_w = np.zeros_like(weight)
    #if (activation != "softmax"):
    for k in range(len(input)):
        for i in range(len(weight[k])): #input
            for j in range(len(weight[k][i])): #output
                delta_w[k][i][j] += learning_rate * np.dot(gd_h1[i] * activation_function(output[i][j], derivative=True), weight_next[i][j]) * input[k][i]

    return de_dw, delta_w
```

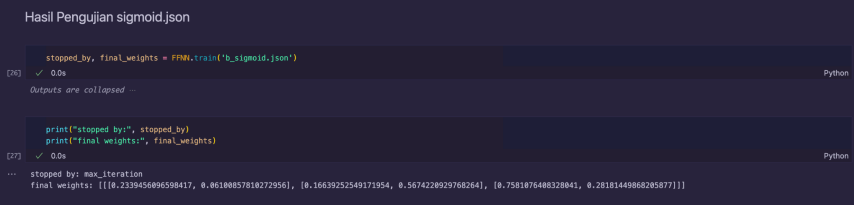
Selanjutnya modul-modul ini diimpor pada **main_B.ipynb** dan dilakukan uji-uji yang diperlukan untuk mendapatkan hasil pengujian yang diperlukan.

HASIL PENGUJIAN

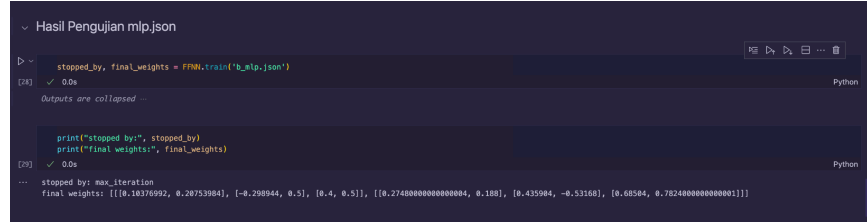
Berikut ini merupakan hasil pengujian dari setiap test case yang menampilkan nilai ekspektasi, SSE, serta hasil *Neural Network Diagram* dari masing-masing test case.

| | |
|----------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Test case 1: linear.json | <div data-bbox="565 457 1419 640"><p>Hasil Pengujian linear.json</p><pre>stopped_by, final_weights = FFW.train('b_linear.json') [0] ✓ 0.0s Python print("stopped by:", stopped_by) print("final weights:", final_weights) [0] ✓ 0.0s Python stopped by: max_iteration final weights: [[0.22599999999999998, 0.356, 0.10199999999999995], [0.34, 0.158, -0.6609999999999999], [0.0460000000000000076, -0.808, 0.55]]</pre></div> <p>Dari hasil pengujian yang telah dilakukan, dapat dilihat bahwa untuk test case linear.json memiliki final weights: [[[0.22599999999999998, 0.356, 0.10199999999999995], [0.34, 0.158, -0.6609999999999999], [0.0460000000000000076, -0.808, 0.55]]]. Nilai final weights yang mendekati 0 tersebut menandakan bahwa model lebih stabil dan model tidak terlalu bergantung pada variabilitas tinggi dalam data input sehingga dapat meningkatkan stabilitas dan kemampuan generalisasi model.</p> |
| Test case 2: linear_small_lr.json | <div data-bbox="565 1136 1419 1318"><p>Hasil Pengujian linear_small_lr.json</p><pre>stopped_by, final_weights = FFW.train('b_linear_small_lr.json') ✓ 0.0s Python print("stopped by:", stopped_by) print("final weights:", final_weights) ✓ 0.0s Python stopped by: max_iteration final weights: [[0.10126, 0.30056, 0.19902], [0.39940000000000003, 0.19958, -0.69961], [0.09946, -0.80008, 0.5005]]</pre></div> <p>Dari hasil pengujian yang telah dilakukan, dapat dilihat bahwa untuk test case linear_small_lr.json memiliki final weights: [[[0.10126, 0.30056, 0.19902], [0.39940000000000003, 0.19958, -0.69961], [0.09946, -0.80008, 0.5005]]]. Nilai final weights yang mendekati 0 tersebut menandakan bahwa model lebih stabil dan model tidak terlalu bergantung pada variabilitas tinggi dalam data input sehingga dapat meningkatkan stabilitas dan kemampuan generalisasi model.</p> |

| | |
|------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Test case 3:</p> <p>linear_two_iteration.json</p> <p>n</p> | <div data-bbox="565 210 1421 394"> <p>Hasil Pengujian linear_two_iteration.json</p> <pre> stopped_by, final_weights = FFM.train('b_linear_two_iteration.json') ✓ 0.0s Python print("stopped by:", stopped_by) print("final weights:", final_weights) ✓ 0.0s Python stopped by: max_iteration final weights: [[[0.35971719999999996, 0.4183692, -0.0005326000000001052], [0.2708092, 0.11431119999999997, -0.6136535999999999], [-0.013502599999999906, -0.8232536, 0.6018758000000000]]]</pre> </div> <p>Dari hasil pengujian yang telah dilakukan, dapat dilihat bahwa untuk test case linear_two_iteration.json memiliki final weights: [[[0.35971719999999996, 0.4183692, -0.0005326000000001052], [0.2708092, 0.11431119999999997, -0.6136535999999999], [-0.013502599999999906, -0.8232536, 0.6018758000000000]]]. Nilai final weights yang mendekati 0 tersebut menandakan bahwa model lebih stabil dan model tidak terlalu bergantung pada variabilitas tinggi dalam data input sehingga dapat meningkatkan stabilitas dan kemampuan generalisasi model.</p> |
| <p>Test case 4: relu.json</p> | <div data-bbox="565 949 1421 1176"> <p>Hasil Pengujian relu.json</p> <pre> stopped_by, final_weights = FFM.train('b_relu.json') ✓ 0.0s Python /Users/cettarewara/Documents/GitHub/Tubes4_13520138/src/ffn.py:155: RuntimeWarning: divide by zero encountered in divide mse /= output_val print("stopped by:", stopped_by) print("final weights:", final_weights) ✓ 0.0s Python stopped by: max_iteration final weights: [[[-0.202310000000000002, 0.180050000000000002, 1.01365], [0.28845, 0.40025, 0.56825], [-0.50385, -1.03325, 0.18075000000000002]]]</pre> </div> <p>Dari hasil pengujian yang telah dilakukan, dapat dilihat bahwa untuk test case relu.json memiliki final weights: [[[-0.202310000000000002, 0.180050000000000002, 1.01365], [0.28845, 0.40025, 0.56825], [-0.50385, -1.03325, 0.18075000000000002]]]. Final weights dalam sebuah model neural network yang menunjukkan variasi yang besar, dengan beberapa mendekati 0 dan yang lainnya mendekati 1 atau lebih, dapat disebabkan oleh beberapa faktor yang berkaitan dengan karakteristik data, desain model, dan proses pelatihan.</p> |

| | |
|-----------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Test case 5:</p> <p>softmax_two_layer.json</p> |  <p>Dari hasil pengujian yang telah dilakukan, dapat dilihat bahwa untuk test case softmax_two_layer.json memiliki final weights: [[[0.1, -0.1, 0.1, -0.1], [-0.1, 0.1, -0.1, 0.1], [0.1, 0.1, -0.1, 0.1]], [[-38.673122477054434, -38.89312247705442], [-0.12, 0.1], [-22.961907873847373, -23.18190787384738], [-0.12, 0.1], [-0.09637936743116556, -0.11637936743116566]]]. Final weights dalam sebuah model neural network yang menunjukkan variasi yang besar, dengan beberapa mendekati 0 dan yang lainnya mendekati 1 atau lebih, dapat disebabkan oleh beberapa faktor yang berkaitan dengan karakteristik data, desain model, dan proses pelatihan.</p> |
| <p>Test case 6:</p> <p>sigmoid.json</p> |  <p>Dari hasil pengujian yang telah dilakukan, dapat dilihat bahwa untuk test case sigmoid.json memiliki final weights: [[[0.2339456096598417, 0.06100857810272956], [0.16639252549171954, 0.5674220929768264], [0.7581076408328041, 0.28181449868205877]]]. Nilai final weights yang mendekati 0 tersebut menandakan bahwa model lebih stabil dan model tidak terlalu bergantung pada variabilitas tinggi dalam data input sehingga dapat meningkatkan stabilitas dan kemampuan generalisasi model.</p> |

Test case 7: mlp.json



```
Hasil Pengujian mlp.json

In [28]: stopped_by, final_weights = FNNI.train('0_mlp.json')
Out[28]: 0.0s

Outputs are collapsed ...

In [29]: print("stopped by:", stopped_by)
print("final weights:", final_weights)
Out[29]: 0.0s

... stopped by: max_iteration
final weights: [[[0.10376992, 0.20753984], [-0.298944, 0.5]], [[0.27480000000000004, 0.188], [0.435904, -0.53168], [0.68504, 0.7824000000000001]]]
```

Dari hasil pengujian yang telah dilakukan, dapat dilihat bahwa untuk test case mlp.json memiliki final weights: [[[0.10376992, 0.20753984], [-0.298944, 0.5]], [[0.27480000000000004, 0.188], [0.435904, -0.53168], [0.68504, 0.7824000000000001]]]. Nilai final weights yang mendekati 0 tersebut menandakan bahwa model lebih stabil dan model tidak terlalu bergantung pada variabilitas tinggi dalam data input sehingga dapat meningkatkan stabilitas dan kemampuan generalisasi model.

PERBANDINGAN DENGAN PENGGUNAAN LIBRARY MLP SKLEARN

A. Library MLP Sklearn

Library MLP (*Multilayer Perceptron*) di *scikit-learn* (sklearn) adalah bagian dari modul `'neural_network'` yang menyediakan implementasi dari Jaringan Saraf Tiruan (Artificial Neural Network) dengan beberapa lapisan tersembunyi. MLP adalah salah satu jenis arsitektur jaringan saraf yang paling umum digunakan, yang terdiri atas beberapa lapisan node yang terhubung secara penuh (*fully-connected*) satu sama lain. Berikut ini adalah beberapa hal penting terkait dengan library MLP di sklearn:

1. Implementasi: Sklearn menyediakan implementasi yang sederhana dan mudah digunakan dari MLP. Ini memungkinkan pengguna untuk membuat, melatih, dan menggunakan model jaringan saraf dengan sedikit kode.
2. Konfigurasi Lapisan: Dalam MLP sklearn, pengguna dapat menentukan jumlah lapisan dan jumlah node dalam setiap lapisan. Ini memungkinkan untuk membuat berbagai arsitektur jaringan saraf dengan ukuran dan kompleksitas yang berbeda.
3. Fungsi Aktivasi: Pengguna dapat memilih fungsi aktivasi untuk setiap lapisan dalam jaringan saraf. Sklearn mendukung beberapa fungsi aktivasi yang umum digunakan seperti ReLU, sigmoid, dan tangen hiperbolik (tanh).
4. Pelatihan/*Training*: Library MLP sklearn menggunakan algoritma pembelajaran yang disebut *backpropagation* untuk melatih model. Ini berarti bahwa model akan disesuaikan secara iteratif menggunakan data pelatihan untuk mengurangi kesalahan prediksi.
5. Regularisasi: Sklearn menyediakan opsi untuk menerapkan regularisasi pada model MLP. Regularisasi digunakan untuk mengurangi *overfitting* dengan menambahkan hukuman pada parameter model.
6. *Cross-Validation*: Untuk mengevaluasi kinerja model, sklearn menyediakan alat untuk validasi silang (*cross-validation*). Ini memungkinkan pengguna untuk mendapatkan perkiraan yang lebih baik tentang kinerja model pada data yang tidak terlihat.
7. Optimisasi: Sklearn menyertakan beberapa pengoptimalan dan parameter yang dapat diatur untuk meningkatkan kinerja dan kecepatan konvergensi model.

Penggunaan library MLP di sklearn memungkinkan para praktisi untuk dengan mudah menerapkan dan mengeksplorasi jaringan saraf tiruan tanpa perlu menulis kode yang rumit atau mengimplementasikan dari awal. Hal ini membuatnya menjadi alat yang berguna untuk pemodelan dan analisis data yang melibatkan struktur dan pola yang kompleks. Baiklah, berikut adalah langkah-langkah umum untuk menggunakan library MLP dari scikit-learn dengan mini-batch gradient descent:

1. Import library: Import `'MLPClassifier'` dari `'sklearn.neural_network'`.
2. Persiapkan data: Persiapkan data dalam format yang sesuai, termasuk fitur dan labelnya.
3. Inisialisasi model: Inisialisasi model `'MLPClassifier'` dengan parameter yang sesuai, termasuk jumlah layer, jumlah neuron per layer, fungsi aktivasi, dan lainnya.
4. Training model: Panggil metode `'fit'` pada model dengan data pelatihan. Di sini, akan ditentukan juga parameter seperti jumlah epoch, ukuran mini-batch, dan lainnya.
5. Evaluasi model: Setelah pelatihan selesai, evaluasi kinerja model pada data pengujian.

B. Uji Kasus Dataset Iris

Berikut ini merupakan perbandingan hasil uji kasus menggunakan [dataset Iris](#) dengan menggunakan program buatan dan library MLP sklearn.

Library MLP

Sklearn

```
import pandas as pd

file_path = '/content/iris.csv'
iris_data = pd.read_csv(file_path)

iris_data.head()
```

| | Id | SepalLengthCm | SepalWidthCm | PetalLengthCm | PetalWidthCm | Species |
|---|----|---------------|--------------|---------------|--------------|-------------|
| 0 | 1 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 1 | 2 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 2 | 3 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 3 | 4 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 4 | 5 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |

```

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

iris_data.drop('Id', axis=1, inplace=True)

X = iris_data.drop('Species', axis=1)
y = iris_data['Species']

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.3, random_state=42)

X_train.shape, X_test.shape, y_train.shape, y_test.shape

((105, 4), (45, 4), (105,), (45,))

from sklearn.neural_network import MLPClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
import numpy as np

# Load dataset Iris
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize the model
mlp = MLPClassifier(hidden_layer_sizes=(100,), activation='relu', random_state=42, max_iter=1000)

# Train model with training data
mlp.fit(X_train, y_train)

# Predict on test data
y_pred = mlp.predict(X_test)


# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Generate classification report
classification_report_str = classification_report(y_test, y_pred)

# Calculate SSE (Sum of Squared Errors)
sse = np.sum((y_test - y_pred) ** 2)

print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report_str)
print("SSE (Sum of Squared Errors):", sse)

```

| | |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| |  Accuracy: 1.0 Classification Report: <pre> precision recall f1-score support 0 1.00 1.00 1.00 19 1 1.00 1.00 1.00 13 2 1.00 1.00 1.00 13 accuracy 1.00 45 macro avg 1.00 1.00 1.00 45 weighted avg 1.00 1.00 1.00 45 SSE (Sum of Squared Errors): 0 </pre> |
| Keterangan | <p>Hasil uji kasus menggunakan dataset Iris dengan program buatan dan library MLP sklearn memberikan hasil yang relatif sama dengan SSE bernilai 0. Jika terdapat perbedaan di antara keduanya, jumlahnya sangat sedikit dan mungkin disebabkan karena adanya perbedaan penurunan rumus yang disesuaikan pada saat pembuatan program.</p> |

C. Analisis Perbandingan

Dari berbagai hasil kasus uji di atas dapat dilakukan analisis perbandingan antara pengujian kasus menggunakan program *Mini-batch Gradient Descent* buatan dengan penggunaan library MLP sklearn. Di samping itu terdapat pula beberapa aspek lain yang dapat dipertimbangkan selain melihat perbandingan hasil di antara keduanya. Analisis perbandingan tersebut dapat dilihat sebagai berikut.

1. Kontrol dan Fleksibilitas

Dalam program buatan, kita sebagai pembuat program memiliki kendali penuh atas setiap langkah dalam proses, mulai dari inisialisasi model hingga penyesuaian *learning rate*. Selain itu, dari segi fleksibilitas, dapat pula menyesuaikan setiap aspek dari algoritma, seperti pemilihan fungsi aktivasi, jumlah *hidden layer*, dan penanganan *overfitting*. Dengan menggunakan library MLP sklearn, kita akan menggunakan implementasi yang sudah ada dan tidak memiliki kendali penuh atas setiap langkah dalam proses, karena sebagian besar

langkah sudah diatur dalam library. Terkait dengan fleksibilitas, pengujian menggunakan library tetap bisa menyesuaikan beberapa parameter seperti jumlah *hidden layer* dan *neuron per layer*, namun tentunya kustomisasi lebih terbatas dibandingkan dengan program buatan.

2. Implementasi

Dalam program buatan, kita harus mengimplementasikan algoritma *mini-batch gradient descent* secara manual, termasuk komputasi gradien, perubahan parameter model, dan pengelolaan batch data. Tentunya, segala penurunan rumus dan implementasi *backpropagation* ke dalam algoritma perlu dilakukan dengan matang dan memerlukan berbagai *trial and error* untuk memperbaiki hasil uji program dan mengurangi gap dengan hasil sebenarnya. Dengan menggunakan MLP sklearn, implementasi *mini-batch gradient descent* sudah disediakan dan dioptimalkan secara efisien sehingga kita hanya perlu mengaplikasikan kasus uji dengan bantuan library dari *scikit-learn* tersebut.

3. Kinerja dan Akurasi

Kinerja program *Mini-batch Gradient Descent* buatan dapat bervariasi tergantung pada kualitas implementasi dan optimisasi yang dilakukan oleh pembuat program. Begitu pula dengan akurasi perhitungan yang dihasilkan akan bergantung dengan kualitas implementasi. Harapannya program buatan tersebut dapat memberikan hasil yang akurat dengan waktu sesingkat mungkin. Sedangkan implementasi MLP sklearn umumnya sudah dioptimalkan untuk kinerja yang baik dan memiliki dukungan untuk operasi yang cepat melalui penggunaan pustaka numerik seperti NumPy dan Cython.

4. Waktu Eksekusi

Meskipun tidak dilakukan penghitungan waktu eksekusi pada penyelesaian persoalan *Mini-batch Descent Gradient* dengan kedua cara, dapat diasumsikan bahwa program buatan memiliki ketergantungan yang erat dengan kualitas implementasi program yang dilakukan oleh pembuat program sehingga efisiensi dan waktu eksekusi mungkin lebih cepat atau justru menjadi lebih lambat daripada menggunakan library yang tersedia. Penggunaan library MLP sklearn telah memberikan kemudahan bagi penggunanya untuk menghasilkan hasil yang

akurat dan efisien, namun pembuatan program secara mandiri tentunya memiliki objektif pembelajaran atau pemahaman terhadap algoritma *backpropagation* itu sendiri serta pencapaian waktu eksekusi program yang diharapkan lebih efisien daripada menggunakan library.

5. Kemudahan Penggunaan dan Kemampuan Pemecahan Masalah

Implementasi menggunakan library MLP sklearn memang lebih mudah karena seluruh prosesnya sudah tersedia pada library dan hanya perlu mengaplikasikannya ke dalam kasus uji, tetapi tidak selalu memberikan wawasan yang mendalam tentang proses yang sebenarnya terjadi di dalamnya. Di samping itu, membuat program *Mini-batch Gradient Descent* buatan tentu memerlukan pemahaman yang kuat tentang matematika di balik *backpropagation*, termasuk mengetahui rumus dan langkah penyelesaiannya, serta memerlukan waktu dan perhatian ekstra, tetapi dapat membantu memahami inti dari algoritma *backpropagation* pada *Mini-batch Gradient Descent* dengan lebih baik.

6. Skalabilitas

Program buatan lebih mudah untuk disesuaikan dengan jaringan yang lebih besar atau lebih kompleks sehingga disebut lebih fleksibel meskipun penggunaan library MLP sklearn juga dapat menyesuaikan beberapa parameter dan kompleksitas jaringan secara terbatas.

Dalam praktiknya, biasanya digunakan kombinasi dari kedua metode tersebut. Program buatan umumnya digunakan untuk pelatihan dan pengujian model dengan tujuan menciptakan model yang dapat menguji model secara efisien, sementara penggunaan library MLP sklearn dapat digunakan untuk memeriksa hasil, memvalidasi pemahaman, atau melakukan analisis teoritis seperti yang telah dilakukan pada tugas besar ini.

LAMPIRAN

A. Repositori

https://github.com/ferindya/TubesA_13520130.git

B. Pembagian Kerja

| NIM | Nama Anggota | Peran |
|----------|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 13520130 | Nelsen Putra | <ul style="list-style-type: none">• Melakukan uji kasus• Implementasi uji kasus model dengan menggunakan library MLP sklearn• Membuat analisis perbandingan hasil uji kasus program <i>Mini-batch Gradient Descent</i> dengan penggunaan library MLP sklearn |
| 13521117 | Maggie Zeta RS | <ul style="list-style-type: none">• Melakukan uji kasus• Implementasi uji kasus model dengan menggunakan library MLP sklearn• Membuat dasar teori |
| 13521133 | Cetta Reswara Parahita | <ul style="list-style-type: none">• Melakukan implementasi penambahan instrumen kelas FFNN• Melakukan implementasi train• Melakukan implementasi backpropagation• Optimasi kelas gradient descent, |

| | | |
|----------|-----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | | <ul style="list-style-type: none"> • Debugging activation function, dan • Membuat main_b.ipynb |
| 13521161 | Ferindya Aulia Berlianty | <ul style="list-style-type: none"> • Melakukan implementasi pembuatan kelas <i>loss function</i> • Melakukan implementasi pembuatan kelas gradient equation • Implementasi uji kasus model dengan menggunakan library MLP sklearn |

C. Link Google Colab: Penggunaan Library MLP Sklearn

https://colab.research.google.com/drive/1dJCCDDOYAg2bxHkElkF0ATV4KgYSE4zh?oid=118002787027657853116&usp=drive_link

DAFTAR REFERENSI

https://cdn-edunex.itb.ac.id/38024-Machine-Learning-Parallel-Class/69139-Minggu-06/36288-Backpropagation/1645000861740_IF3270_Materi05_Seg04_ANN-Backpropagation.pdf

https://cdn-edunex.itb.ac.id/38024-Machine-Learning-Parallel-Class/69139-Minggu-06/36286-Gradient-Descent/1644999883956_IF3270_Materi05_Seg02_ANN-GradientDescent.pdf

<https://www.youtube.com/@statquest>