Modul Praktikum 2

DEEP LEARNING

PROGRAM STUDI SAINS DATA FAKULTAS SAINS

INSTITUT TEKNOLOGI SUMATERA

2024

Backpropagation Neural Network

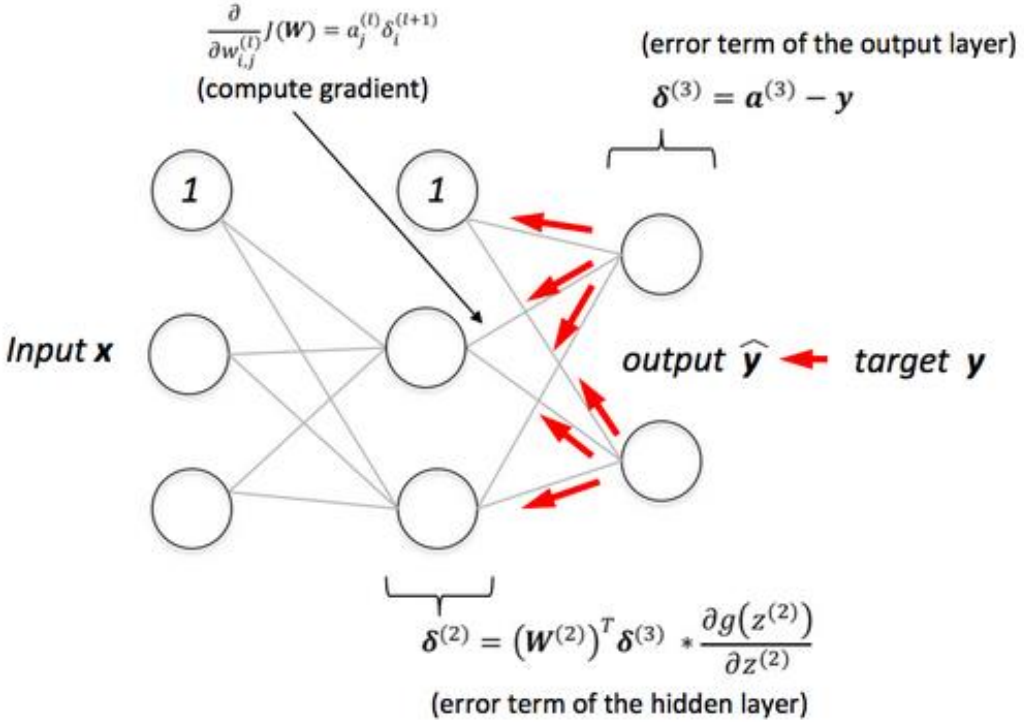
1. Tujuan

- ✓ Mahasiswa mampu memahami konsep metode *Backpropagation Neural Network*.
- ✓ Mahasiswa mampu menerapkan perhitungan matematika *Backpropagation Neural Network* dan menerapkan pemrograman hitungannya menggunakan Python.

2. Dasar Teori

a) *Backpropagation Neural Network*

Backpropagation Neural Network digunakan untuk meningkatkan akurasi *Neural Network*. Backpropagation berarti *"backward propagation of errors"*, *error* tersebut artinya kesalahan disebarkan ke arah sebaliknya untuk mencapai kinerja yang lebih baik. Konsep inti Backpropagation yaitu melakukan backpropagation atau menyebarkan kesalahan dari *neuorun output* layer ke *hiden layer* untuk mengupdate bobot guna memastikan tingkat kesalahan yang lebih rendah. Hal tersebut dianggap sebagai penyempurnaan bobot n*eural network* dalam setiap iterasi.



Gambar 1 Cara kerja B*ackpropagation neural network*

Berikut adalah tahapan umum untuk algoritma backpropagation pada *input layer*, *hiden layer*, dan *output layer* menggunakan fungsi aktivasi sigmoid:

1. Langkah 1 : Inisialisasi bobot (ambil bobot awal dengan nilai random yang cukup kecil)
2. Langkah 2 : ketika kondisi berhenti salah, lakukan langkah 3-9.

Feedforward:

3. Langkah 3 : Tiap-tiap neuron menerima input $x_{i\ ,}\ i = \ 1, 2, 3, \ldots n$ dan meneruskan input tersebut ke semua neuron pada layer yang ada diatasnya *(hidden layer)*
4. Langkah 4 : Tiap-tiap neuron pada hiden layer $z_{j\ ,}\ j = \ 1, 2, 3, \ldots m$ menjumlahkan input terbobot :
    i. Hitung fungsi aktivasi untuk menghitung output,
    ii. Lalu kirimkan output kesemua neuron pada *neuron output*.
5. Langkah 5 : Tiap-tiap *neuron output* menjumlahkan input terbobot. gunakan fungsi aktivasi untuk menghitung outputnya, dan kirimkan output tersebut ke semua neuron di layer atasnya (*neuron output*).

Backpropagation:

6. Langkah 6 : Tiap-tiap *neuron output* menerima target, hitung informasi error-nya:
    i. kemudian hitung koreksi bobot (yang nantinya akan digunakan untuk memperbaiki nilai W)
    ii. hitung koreksi bias (yang nantinya akan digunakan untuk memperbaiki nilai b)
    iii. kirimkan i dan ii ke neuron yang ada pada layer bawahnya (*hidden layer*)
7. Langkah 7 : Tiap-tiap *neuron hiden layer* menjumlahkan delta inputnya (delta **delta (δ)** merujuk pada error term atau istilah kesalahan yang digunakan untuk memperbarui bobot pada inputnya).
    i. Kalikan nilai tersebut dengan turunan dari fungsi aktivasinya untuk menghitung informasi error
    ii. Kemudian hitung koreksi bobot
    iii. Hitung juga koreksi bias
8. Langkah 8 : Tiap-tiap *neuron output* memperbaiki bias dan bobotnya
9. Langkah 9 : berhenti Setelah training selesai dilakukan.

b). Menghitung error

Untuk menghitung error untuk setiap neuron output gunakan *square error function* dan lakukan penjumlahan total untuk mendapatkan error nya:

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Misalnya, target outputnya adalah 0,01 tetapi outputnya neuronnya adalah 0,75136507, oleh karena itu kesalahannya adalah:

$$E_{output\ 1} = \frac{1}{2}(target\ _{o1} - output_{o1})^2 = \frac{1}{2}(0.01 - 0{,}75136507)^2 = 0.274811082$$

Mengulangi proses tersebut untuk mendapatkan nilai error output, misalnya diperolah :

$$E_{output\ 2} = \ 0.023560026$$

Kesalahan total untuk *neural network* adalah jumlah dari error-error berikut:

$$E_{total} = \ E_{output1} - \ E_{output2} = \ 0.274811082 - \ 0.023560026 = 0.298371109$$

c). Learning rate,

Learning rate merupakan salah satu parameter *training* untuk menghitung nilai koreksi bobot pada waktu proses *training*. Nilai *learning rate* ini berada pada *range* nol (0) sampai (1). Semakin besar nilai *learning rate*, maka proses *training* akan berjalan semakin cepat. Semakin besar *learning rate*, maka ketelitian jaringan akan semakin berkurang, tetapi berlaku sebaliknya, apabila *learning rate*-nya semakin kecil, maka ketelitian jaringan akan semakin besar atau bertambah dengan konsekuensi proses *training* akan memakan waktu yang semakin lama. **Catatan** ada istilah *Epochs*, istilah tersebut untuk mewakili jumlah iterasi yang harus dilakukan pada set data. *Epochs* menandakan satu siklus algoritma *deep learning* belajar dari keseluruhan *training dataset*. Satu *epochs* berarti menandakan sebuah algoritma *deep learning* telah belajar dari *training dataset* secara keseluruhan.

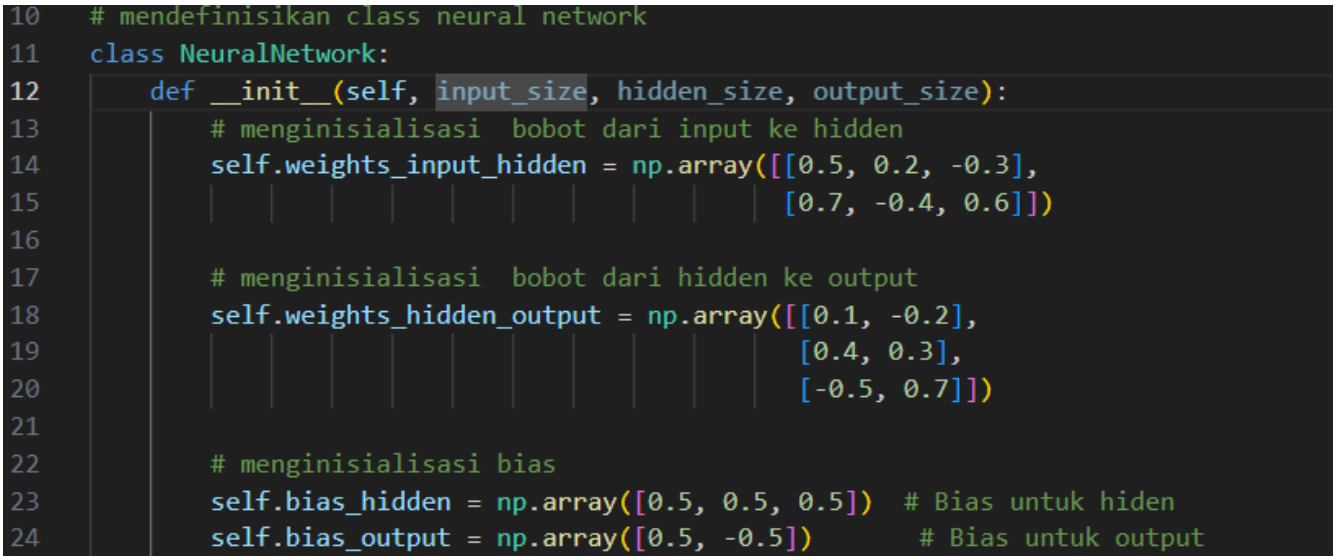
3. Kegiatan Praktikum Backpropagation Neural Network

Pada praktikum ini akan dilakukan pengujian pada suatu backpropagation neural network menggunakan Python. Data yang digunakan adalah sebagai berikut:
$x1 = 0,\ x2 = 5,$
$w\ dari\ input\ layer\ ke\ hidden\ layer\ adalah$ $[0.5, 0.2, -0.3], [0.7, -0.4, 0.6]$
$w\ dari\ input\ layer\ ke\ hidden\ layer\ adalah$ [0.1, -0.2], [0.4, 0.3], [0.5, 0.7]
$bias\ hidden\ layer\ adalah$ $[0.5, 0.5, 0.5]$
$bias\ output\ layer\ adalah [0.5, -0.5]$

4. Langkah-langkah Praktikum

Pada praktikum ini, kode program Python akan dijalankan menggunakan layanan Google Colab yang dapat diakses di https://colab.research.google.com/. Langkah pertama yang dilakukan setelah membuka layanan Google Colab adalah mengimpor pustaka yang akan digunakan. Pada praktikum ini pustaka yang digunakan adalah numpy.

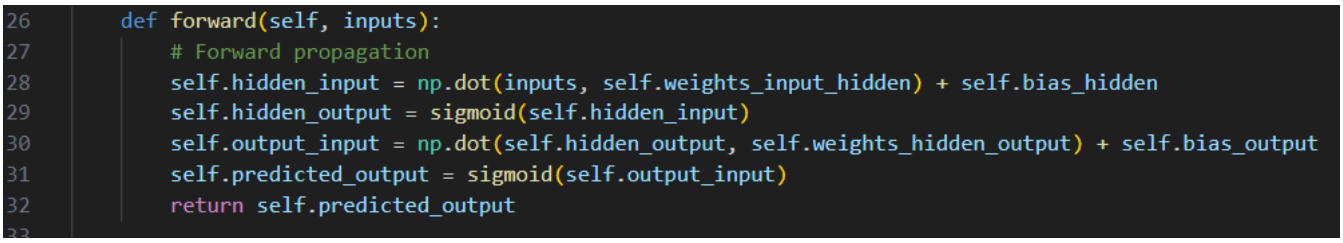
1. Inisiasi nilai **W** dan **Bias**. Pada gambar 2 dapat dilihat bahwa nilai **W** dan **Bias** dinisiasi di dalam `class NeuralNetwork`. Pada gambar nilai untuk W dari input ke hidden adalah matrix 2x3, dan matrix untuk W dari hidden ke output yaitu 3 x 2. Pada gambar juga dapat dilihat nilai bias untuk *hiden layer* dan *output layer* sudah diinisiasi.



```
10  # mendefinisikan class neural network
11  class NeuralNetwork:
12      def __init__(self, input_size, hidden_size, output_size):
13          # menginisialisasi  bobot dari input ke hidden
14          self.weights_input_hidden = np.array([[0.5, 0.2, -0.3],
15                                                [0.7, -0.4, 0.6]])
16
17          # menginisialisasi  bobot dari hidden ke output
18          self.weights_hidden_output = np.array([[0.1, -0.2],
19                                                 [0.4, 0.3],
20                                                 [-0.5, 0.7]])
21
22          # menginisialisasi bias
23          self.bias_hidden = np.array([0.5, 0.5, 0.5])  # Bias untuk hiden
24          self.bias_output = np.array([0.5, -0.5])       # Bias untuk output
```

Gambar 2

2. Setelah melakukan inisiasi selanjutnya adalah membuat sebuh fungsi **forward** (fungsi maju untuk mendapatkan nilai prediksi nya). Potongan kode dapat dilihat pada gambar 3. Fungsi forward masih berada dalam `class NeuralNetwork`.



```
26      def forward(self, inputs):
27          # Forward propagation
28          self.hidden_input = np.dot(inputs, self.weights_input_hidden) + self.bias_hidden
29          self.hidden_output = sigmoid(self.hidden_input)
30          self.output_input = np.dot(self.hidden_output, self.weights_hidden_output) + self.bias_output
31          self.predicted_output = sigmoid(self.output_input)
32          return self.predicted_output
```

Gambar 3

3. Hasil dari fungsi forward yaitu nilai prediksi akan digunakan di dalam fungsi backward. Fungsi backwar masih berada di dalam `class NeuralNetwork.` Potongan kode untuk fungsi backward dapat dilihat pada gambar 4. Pada gambar tiga dapat dilihat ada proses perhitungan error (dapat dilihat pada baris ke 36 dan 37). Perhitungan error tersebut masih terhadap output saja. Pada gambar 4, dapat dilihat juga ada proses mengupdate nilai dari W, nilai W diupdate berdasarkan proses yang terjadi (line 39 dan 40 pada gambar).

```
    def backward(self, inputs, target, learning_rate):
        # Backpropagation
        error = target - self.predicted_output
        delta_output = error * sigmoid_derivative(self.predicted_output)

        error_hidden = delta_output.dot(self.weights_hidden_output.T)
        delta_hidden = error_hidden * sigmoid_derivative(self.hidden_output)

        # Update weights
        self.weights_hidden_output += np.outer(self.hidden_output, delta_output) * learning_rate
        self.weights_input_hidden += np.outer(inputs, delta_hidden) * learning_rate
```

Gambar 4

4. Masih di class yang sama yaitu `class NeuralNetwork,` terdapat suatu fungsi yang diberi nama `train.` Fungs `train` mencakup perhitungan output (forward propagation, update bobot (backward propagation) dan melihat total error yang terjadi. Potongan kode dapat dilihat pada gambar 5.

```
    def train(self, training_data, targets, epochs, learning_rate):
        for epoch in range(epochs):
            total_error = 0  # Initialize total error for the epoch
            for i in range(len(training_data)):
                inputs = training_data[i]
                target = targets[i]
                self.forward(inputs)
                self.backward(inputs, target, learning_rate)
                total_error += np.mean(np.square(target - self.predicted_output))  # Squared Error

            # Print error setiap 100 iterasi aja
            if epoch % 100 == 0:
                print(f"Epoch {epoch}, Average Error: {total_error / len(training_data)}")
```

Gambar 5.

5. Jangan lupa untuk menambahkan sebuah fungsi untuk memanggil fungsi forward pada input, di dalam `class NeuralNetwork`. Seperti pada gambar 6.

```
    def predict(self, inputs):
        return self.forward(inputs)
```

Gambar 6.

6. Terakhir panggil `class NeuralNetwork`. Potongan kode dapa dilihat pada gambar 7. Pada gambar terdapat `input_size` dengan nilai 2, artinya terdapat 2 input yaitu x1 dan x2.

```
63  # Input atau nilai x nya.
64  training_data = np.array([[0, 5]])
65  targets = np.array([0])
66
67  input_size = 2
68  hidden_size = 3
69  output_size = 1
70  learning_rate = 0.1
71  epochs = 100
72
73  nn = NeuralNetwork(input_size, hidden_size, output_size)
74  nn.train(training_data, targets, epochs, learning_rate)
75
76  # Test network
77  for inputs in training_data:
78      prediction = nn.predict(inputs)
79      print(f"Input: {inputs}, Predicted Output: {prediction}")
```

Gambar 7

7. Tambahan: tambahakan fungsi aktivasi, yang digunakan. Fungsi aktivasi terletak diluar `class NeuralNetwork`. Dapat dilihat pada gambar 8.

```
3  #fungsi aktivasi yang akan dipanggil
4  def sigmoid(x):
5      return 1 / (1 + np.exp(-x))
6
7  def sigmoid_derivative(x):
8      return x * (1 - x)
```

Gambar 8.