Modul Praktikum 3

DEEP LEARNING

PROGRAM STUDI SAINS DATA FAKULTAS SAINS

INSTITUT TEKNOLOGI SUMATERA

2024

Hyperparameter & Optimizer

1. Tujuan

- ✓ Mahasiswa mampu memahami konsep penggunaan Hyperparameter & Optimizer
- ✓ Mahasiswa mampu menerapkan perhitungan matematika *Hyperparameter & Optimizer* dan menerapkan pemrograman hitungannya menggunakan Python.

2. Dasar Teori

a) *Hyperparameter*

Hyperparameter memiliki peran penting dalam menentukan performa model. Penyesuaian nilai hyperparameter (hyperparameter tuning) dapat meningkatkan akurasi model, mengurangi overfitting, dan mempercepat konvergensi pelatihan.

Contoh Hyperparameter pada Beberapa Model:

1. Pada regresi polinomial, hyperparameter yang perlu ditentukan adalah derajat (degree) dari polinomial.
2. Dalam Random Forest, hyperparameter seperti jumlah pohon (number of trees) dan kedalaman maksimum pohon (maximum tree depth) sangat mempengaruhi hasil.
3. Pada Decision Tree, kedalaman maksimum dan kriteria pembelahan (splitting criteria) adalah hyperparameter penting.

Hyperparameter merupakan sesuatu yang dapat kita ubah. Proses ini juga dinamakan **Hyperparameter Tuning.** Hyperparameter pada Neural Network yang harus ditentukan sebelum proses pelatihan dimulai:

1. Jumlah Neuron dalam Hidden Layer: Jumlah neuron pada layer tersembunyi mempengaruhi kapasitas model dalam menangkap pola dari data. Terlalu banyak neuron dapat menyebabkan overfitting, sedangkan terlalu sedikit dapat menyebabkan underfitting.
2. Jumlah Hidden Layer: Jumlah layer tersembunyi yang digunakan dalam jaringan. Semakin banyak layer, semakin kompleks jaringan tersebut.
3. Fungsi Aktivasi: Fungsi ini memberikan non-linearitas pada jaringan. Contoh fungsi aktivasi adalah ReLU, Sigmoid, dan Tanh.
4. Inisialisasi Bobot: Pemilihan metode inisialisasi bobot awal mempengaruhi kecepatan dan stabilitas konvergensi.
5. *Learning rate* merupakan salah satu parameter untuk menghitung nilai koreksi bobot pada waktu proses *training*. Nilai *learning rate* ini berada pada *range* nol (0) sampai (1).

b) *Epoch, Batch Size, Iterasi*

Terdapat beberapa konsep penting dalam cara model belajar dari data:

1. Epoch: Satu epoch adalah ketika seluruh dataset telah melalui model sekali. Misalnya, jika dataset berisi 1000 sampel dan pelatihan dilakukan dengan 10 epoch, maka seluruh dataset digunakan 10 kali selama pelatihan.
2. Batch Size: Merupakan jumlah sampel data yang digunakan dalam satu iterasi pelatihan untuk memperbarui bobot. Dengan menggunakan batch size, model dapat memperbarui bobot secara bertahap daripada setelah seluruh dataset selesai diproses. Misalnya, jika dataset memiliki 1000 sampel dan batch size adalah 100, maka setiap epoch akan dibagi menjadi 10 batch.
3. Iterasi: Setiap iterasi mencakup satu kali pemrosesan batch data dan pembaruan bobot. Jumlah iterasi dalam satu epoch sama dengan jumlah batch per epoch. Jika dilakukan pelatihan dengan 5 epoch dan batch size 100, maka total iterasi adalah 50.

c) ***Optimizer***

Optimizer adalah algoritma atau metode yang digunakan untuk memperbarui bobot model guna meminimalkan fungsi loss. Berikut adalah beberapa optimizer yang sering digunakan dalam deep learning:

1. Gradient Descent (GD): Menggunakan seluruh dataset untuk memperbarui bobot dalam setiap iterasi. GD memerlukan waktu komputasi yang tinggi terutama pada dataset besar.
2. Stochastic Gradient Descent (SGD): Menggunakan satu sampel data secara acak pada setiap iterasi. SGD lebih cepat namun memiliki ketidakstabilan dalam proses pembelajaran.
3. AdaGrad: Menyesuaikan learning rate untuk setiap parameter berdasarkan seberapa sering parameter tersebut diperbarui. Cocok untuk data dengan fitur jarang (sparse data).
4. RMSProp: Varian dari AdaGrad yang menambahkan faktor peluruhan untuk mencegah learning rate terlalu kecil. RMSProp stabil pada masalah dengan data seimbang.
5. Momentum: Menambahkan momentum pada pembaruan bobot untuk mempercepat konvergensi. Momentum mengurangi osilasi pada pembaruan bobot.
6. Adam (Adaptive Moment Estimation) adalah salah satu optimizer yang paling populer dalam deep learning. Adam menggabungkan kelebihan dari momentum dan RMSProp, dengan mengestimasi rata-rata dan varians dari gradien.

3. Kegiatan Praktikum

Pada praktikum ini akan dilakukan pengujian pada suatu backpropagation neural network menggunakan Python. Data yang digunakan link:

https://raw.githubusercontent.com/mirohmi/Heart_Disease_Diagnose/refs/heads/master/heart_diseases.csv

Fungsi Aktivasi yang digunakan yaitu : Fungsi Aktivasi Sigmoid.

Algoritma optimizer yang diimplementasikan yaitu : SGD, Momentum, RMSProp, Adam.

Jumlah sampel data yang digunakan dalam satu iterasi adalah 10.

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, pandas, json, time.

```
1  import json
2  import numpy as np
3  import pandas as pd
4  import time
```

Gambar 1

1. Buat fungsi Aktivasi yang digunakan. Pada praktikum ini, fungsi aktivasi yang digunakan adalah Sigmoid.

```
6  # Fungsi aktivasi
7  def sigmoid(x):
8      return 1 / (1 + np.exp(-x))
9
10 def sigmoid_derivative(x):
11     return x * (1 - x)
```

Gambar 2

2. Optimasi yang digunakan adalah SGD, Momentum, RMSprop, ADAM. Pada gambar dapat dilihat fungsi dari setiap optimasi.

```
13 # Fungsi update SGD
14 def sgd_update(weight, gradient, learning_rate):
15     return weight + learning_rate * gradient
16
17 # Fungsi update Momentum
18 def momentum_update(weight, gradient, learning_rate, velocity, momentum):
19     velocity = momentum * velocity + learning_rate * gradient
20     return weight + velocity, velocity
21
22 # Fungsi update RMSProp
23 def rmsprop_update(weight, gradient, v, learning_rate, beta2, epsilon=1e-8):
24     v = beta2 * v + (1 - beta2) * (gradient ** 2)
25     weight -= learning_rate * gradient / (np.sqrt(v) + epsilon)
26     return weight, v
27
28 # Fungsi update Adam
29 def adam_update(weight, gradient, m, v, t, learning_rate, beta1, beta2, epsilon=1e-8):
30     m = beta1 * m + (1 - beta1) * gradient
31     v = beta2 * v + (1 - beta2) * (gradient ** 2)
32     m_hat = m / (1 - beta1 ** t)
33     v_hat = v / (1 - beta2 ** t)
34     weight += learning_rate * m_hat / (np.sqrt(v_hat) + epsilon)
35     return weight, m, v
```

Gambar 3

3. Buat sebuah class neural network, di dalam class tersebut inisialisasi bobot dan setup variabel-variabel terkait untuk beberapa optimizer dan inisialisasi.

```
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size, optimizer='sgd'):
        # Inisialisasi bobot dan bias
        self.weights_input_hidden = np.random.randn(input_size, hidden_size)
        self.weights_hidden_output = np.random.randn(hidden_size, output_size)
        self.bias_hidden = np.random.randn(hidden_size)
        self.bias_output = np.random.randn(output_size)

        # Setup variabel terkait optimizer
        self.optimizer = optimizer
        self.velocity_ih = np.zeros_like(self.weights_input_hidden)
        self.velocity_ho = np.zeros_like(self.weights_hidden_output)
        self.m_ih = np.zeros_like(self.weights_input_hidden)
        self.v_ih = np.zeros_like(self.weights_input_hidden)
        self.m_ho = np.zeros_like(self.weights_hidden_output)
        self.v_ho = np.zeros_like(self.weights_hidden_output)
        self.beta1, self.beta2 = 0.9, 0.999
        self.t = 1
```

Gambar 4

4. Buat sebuah fungsi forward.

```
    def forward(self, inputs):
        # Propagasi maju untuk batch
        self.hidden_input = np.dot(inputs, self.weights_input_hidden) + self.bias_hidden
        self.hidden_output = sigmoid(self.hidden_input)
        self.output_input = np.dot(self.hidden_output, self.weights_hidden_output) + self.bias_output
        self.predicted_output = sigmoid(self.output_input)
        return self.predicted_output

```

Gambar 5

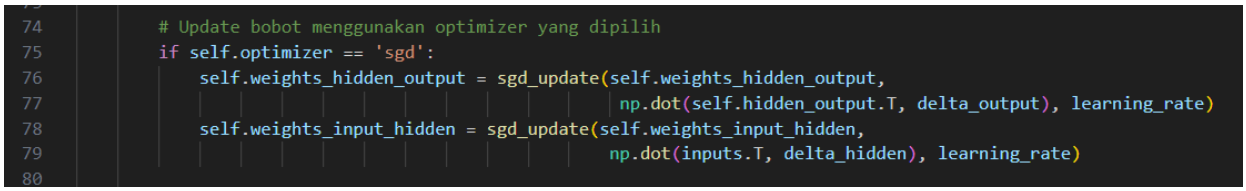
5. Buat sebuah fungs backward untuk setiap batch

```

    def backward(self, inputs, targets, learning_rate):
        # Backpropagation untuk batch
        error = targets - 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)
```

Gambar 6

6. Update bobot menggunakan optimiser. Buat kondisi untuk menentukan optimasi apa yang digunakan. Gambar 8, merupakan kondisi yang digunakan untuk SGD.

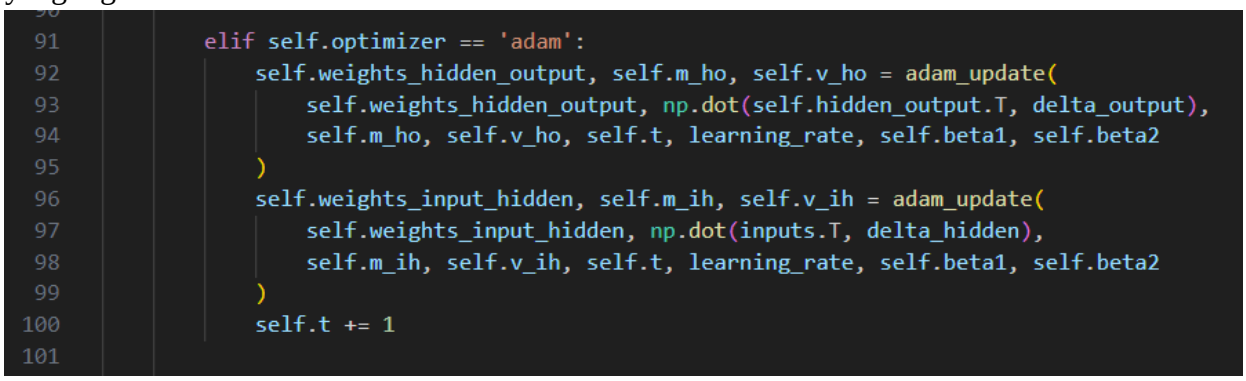


```
        # Update bobot menggunakan optimizer yang dipilih
        if self.optimizer == 'sgd':
            self.weights_hidden_output = sgd_update(self.weights_hidden_output,
                                                    np.dot(self.hidden_output.T, delta_output), learning_rate)
            self.weights_input_hidden = sgd_update(self.weights_input_hidden,
                                                   np.dot(inputs.T, delta_hidden), learning_rate)

```

Gambar 8

7. Buat kondisi untuk menentukan optimasi apa yang digunakan. Gambar 9, merupakan kondisi yang digunakan untuk ADAM.

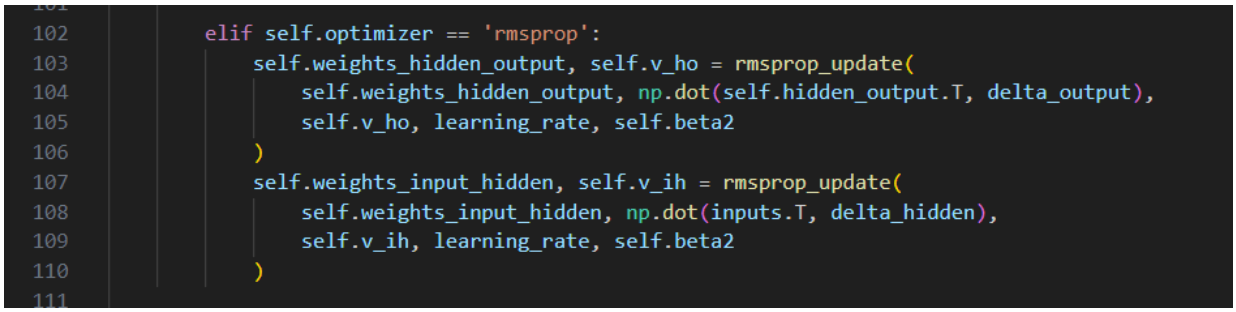


```
        elif self.optimizer == 'adam':
            self.weights_hidden_output, self.m_ho, self.v_ho = adam_update(
                self.weights_hidden_output, np.dot(self.hidden_output.T, delta_output),
                self.m_ho, self.v_ho, self.t, learning_rate, self.beta1, self.beta2
            )
            self.weights_input_hidden, self.m_ih, self.v_ih = adam_update(
                self.weights_input_hidden, np.dot(inputs.T, delta_hidden),
                self.m_ih, self.v_ih, self.t, learning_rate, self.beta1, self.beta2
            )
            self.t += 1

```

Gambar 9

8. Buat kondisi untuk menentukan optimasi apa yang digunakan. Gambar 10, merupakan kondisi yang digunakan untuk RMSprop.

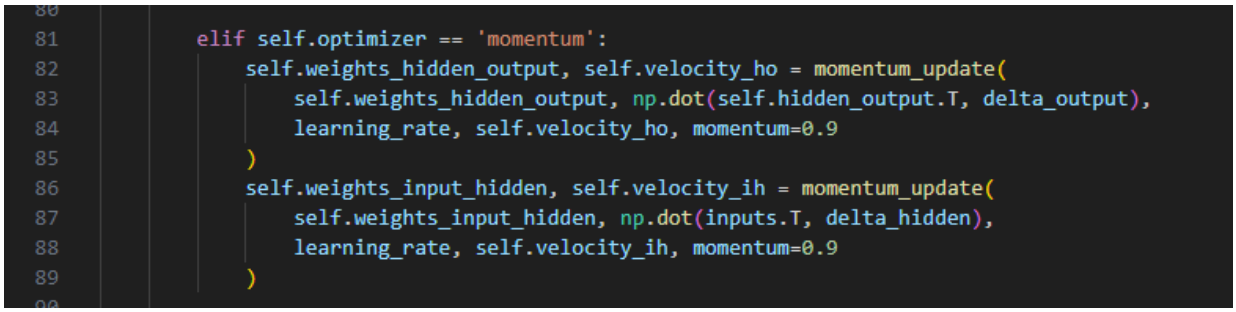


```
        elif self.optimizer == 'rmsprop':
            self.weights_hidden_output, self.v_ho = rmsprop_update(
                self.weights_hidden_output, np.dot(self.hidden_output.T, delta_output),
                self.v_ho, learning_rate, self.beta2
            )
            self.weights_input_hidden, self.v_ih = rmsprop_update(
                self.weights_input_hidden, np.dot(inputs.T, delta_hidden),
                self.v_ih, learning_rate, self.beta2
            )

```

Gambar 10

9. Buat kondisi untuk menentukan optimasi apa yang digunakan. Gambar 11, merupakan kondisi yang digunakan untuk Momentum.



```
        elif self.optimizer == 'momentum':
            self.weights_hidden_output, self.velocity_ho = momentum_update(
                self.weights_hidden_output, np.dot(self.hidden_output.T, delta_output),
                learning_rate, self.velocity_ho, momentum=0.9
            )
            self.weights_input_hidden, self.velocity_ih = momentum_update(
                self.weights_input_hidden, np.dot(inputs.T, delta_hidden),
                learning_rate, self.velocity_ih, momentum=0.9
            )

```

Gamabr 11

10. Buat sebuah fungsi untuk melakukan iterasi sejumlah epoch yang ditentukan, di mana setiap batch data melalui forward pass, backward pass, dan pembaruan bobot. **catatan**: Error history menyimpan nilai error untuk setiap epoch. Setiap 10 epoch, error akan ditampilkan di konsol.

```
112     def train(self, training_data, targets, epochs, learning_rate, batch_size):
113         error_history = []  # Untuk menyimpan error di setiap epoch
114         for epoch in range(epochs):
115             total_error = 0
116             for i in range(0, len(training_data), batch_size):
117                 # Buat batch
118                 batch_inputs = training_data[i:i + batch_size]
119                 batch_targets = targets[i:i + batch_size].reshape(-1, 1)
120 
121                 # Forward pass
122                 self.forward(batch_inputs)
123 
124                 # Backward pass
125                 self.backward(batch_inputs, batch_targets, learning_rate)
126 
127                 # Hitung error untuk batch
128                 total_error += np.mean(np.square(batch_targets - self.predicted_output))
129 
130             error_history.append(total_error / (len(training_data) / batch_size))
131 
132             # Print error setiap 10 epoch
133             if epoch % 10 == 0:
134                 print(f"Epoch {epoch}, Error: {total_error / (len(training_data) / batch_size)}")
135         return error_history
136 
137     def predict(self, inputs):
138         return self.forward(inputs)
```

Gambar 12

11. Gambar 13, merupakan source code untuk meload dataset atau inputan yang digunakan.

```
141 data = pd.read_excel("dataset_cleaned.xlsx")
142 training_data = data.iloc[:, :-1].values
143 targets = data.iloc[:, -1].values
144
```

Gabar 13

12. Gambar 14 merupakan source code untuk menentukan hyperparameter yang digunakan.

```
145 # Parameter jaringan saraf
146 input_size = training_data.shape[1]
147 hidden_size = 4  # Jumlah node di hidden layer
148 output_size = 1  # Jumlah output
149 learning_rate = 0.01
150 epochs = 1000
151 batch_size = 10
```

Gambar 15

13. Terakhir silahkan jalankan program dengan menentukan optimasi yang digunakan

```
154  nn = NeuralNetwork(input_size, hidden_size, output_size, optimizer='adam')
155  nn.train(training_data, targets, epochs, learning_rate, batch_size)
156
157  # Test network
158  for inputs in training_data:
159      prediction = nn.predict(inputs.reshape(1, -1))  # Ubah input ke bentuk batch
160      print(f"Input: {inputs}, Predicted Output: {prediction}")
161
```

Gambar 16

Optional :

14. Buat sebuah fungsi untuk melatih model neural network dengan optimizer yang ditentukan.
    Catatan : tambahkan time.time untuk mengukur waktu yang dihabiskan selama pelatihan.

```
153  # Fungsi untuk melatih dengan optimizer tertentu
154  def train_with_optimizer(optimizer, input_size, hidden_size, output_size,
155                           training_data, targets, epochs, learning_rate, batch_size):
156      nn = NeuralNetwork(input_size, hidden_size, output_size, optimizer)
157      start_time = time.time()
158      error_history = nn.train(training_data, targets, epochs, learning_rate, batch_size)
159      training_time = time.time() - start_time
160      return error_history, training_time
```

15. Buat sebuah kondisi untuk menyimpan hasil error dan waktu training

```
162  # Latih model dengan berbagai optimizer
163  optimizers = ['sgd', 'momentum', 'adam', 'rmsprop']
164  results = {}
165
166  for opt in optimizers:
167      print(f"Training with {opt} optimizer...")
168      error_history, training_time = train_with_optimizer(
169          opt, input_size, hidden_size, output_size,
170          training_data, targets, epochs, learning_rate, batch_size
171      )
172      results[opt] = {
173          'error_history': error_history,
174          'training_time': training_time
175      }
```

16. Silahkan simpan hasil kedalam file JSON

```
177  # Simpan hasil ke file JSON
178  with open('optimizer_results_single_line.json', 'w') as f:
179      for optimizer, data in results.items():
180          json_line = json.dumps({optimizer: data}, separators=(',', ':'))
181          f.write(json_line + '\n')
182
183  print("Results saved to optimizer_results_single_line.json")
```