Modul Praktikum 4

DEEP LEARNING

PROGRAM STUDI SAINS DATA FAKULTAS SAINS

INSTITUT TEKNOLOGI SUMATERA

2024

Regularisasi

1. Tujuan

- ✓ Mahasiswa mampu memahami konsep Regularisasi pada *Deep Learning*.
- ✓ Mahasiswa mampu mengimplementasikan metode-metode Regularisasi pada Deep Learning

2. Dasar Teori

a) Overfitting dan Regularisasi

Saat melakukan suatu pemodelan, sering terjadi dimana akurasi pelatihan model tinggi tetapi akurasi saat pengujian atau validasi rendah yang disebut sebagai kondisi *overfitting.* Overfitting terjadi saat model bekerja dengan baik saat proses pelatihan sedangkan saat pengujian atau melakukan prediksi menggunakan data baru model tidak bekerja dengan baik atau memiliki akurasi yang rendah.

Masalah *overfitting* merupakan salah satu masalah pada Deep Learning ataupun *Machine Learning* yang perlu diselesaikan. Biasanya untuk mengatasi *overfitting* dilakukan validasi silang dan pelatihan dengan lebih banyak data, namun teknik ini tidak selalu dapat dilakukan dan tidak berlaku jika kumpulan data terlalu besar. **Regularisasi** adalah serangkaian metode untuk mengurangi *overfitting* dalam model pembelajaran mesin dengan menambahkan penalti ke fungsi *loss*, sehingga model tidak terlalu mementingkan fitur individu atau koefisien. Beberapa teknik regularisasi yang sering dilakukan adalah Lasso atau L1, Ridge atau L2, serta teknik Dropout.

b) Teknik Regularisasi

1) Regularisasi L1 atau Lasso

Regularisasi Lasso (L1) atau *Least Absolute Shrinkage and Selection Operator* adalah regularisasi dengan menambahkan penalti sebesar nilai absolut dari bobot pada fungsi loss atau error sehingga mendorong ketersebaran dengan mengarahkan beberapa koefisien ke angka nol, melakukan pemilihan fitur secara efektif dengan menghilangkan fitur yang kurang penting. Kemampuan pemilihan fitur ini menjadikan regularisasi L1 sangat berguna ketika menangani kumpulan data dengan sejumlah besar fitur, karena

membantu menyederhanakan model dengan berfokus pada fitur yang paling relevan. Penyederhanaan model yang dihasilkan mengurangi akan *overfitting*. Berikut ini adalah bentuk persamaannya.

$$L(w) = L_0(w) + \lambda \sum_i |w_i|$$

Dengan L_0 adalah fungsi loss atau error, dan λ adalah *hyperparameter* yang mengendalikan seberapa besar penalti yang dikenakan pada bobot.

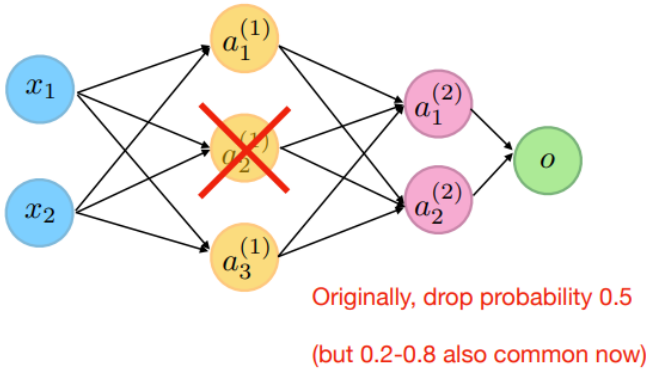
2) Regularisasi L2 atau Ridge

Regularisasi L2 atau Ridge adalah teknik regularisasi yang menambahkan penalti sebesar nilai kuadrat dari bobot pada fungsi loss atau error. Berbeda dengan Lasso, pada Ridge, jumlah koefisien yang dikuadratkan, bukan nilai absolut dari koefisien. Selain itu, Ridge tidak melakukan pemilihan fitur. Meskipun istilah penalti dari Lasso dapat menghilangkan fitur dari model dengan mengecilkan nilai koefisien menjadi nol, Ridge hanya menyusutkan bobot fitur ke arah nol tetapi tidak pernah menjadi nol. Berikut adalah bentuk persamaan dari teknik Rideg.

$$L(w) = L_0(w) + \lambda \sum_i (w_i)^2$$

3) Dropout

Dropout merupakan salah satu teknik untuk mencegah *overfitting* dengan cara menghapus (menonaktifkan) sejumlah neuron secara acak selama pelatihan. Dropout dapat digunakan pada sebagian besar jenis lapisan, seperti *dense fully connected layers, convolutional layers*, dan *recurrent layers* begitu juga layer pada LSTM. *Dropout* dapat diterapkan pada setiap atau semua lapisan tersembunyi dalam jaringan tetapi tidak pada lapisan keluaran. Berikut ini adalah contoh gambar dari dropout pada *neuron hidden layer.*



3. Kegiatan Praktikum Regularisasi

Pada praktikum ini, data yang digunakan merupakan data transfusi darah yang dapat dilihat pada tautan: https://raw.githubusercontent.com/mirohmi/Heart_Disease_Diagnose/refs/heads/master/transfusion_data.csv

Fungsi aktivasi yang digunakan adalah sigmoid, dan metode optimizer yang digunakan adalah adam.

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 dan pandas untuk praktikum L1 dan L2 sedangkan pada praktikum Dropout pustaka yang digunakan adalah tensorflow.

I. Lasso dan Ridge

a) Import pustaka yang akan digunakan.

```
import numpy as np
import pandas as pd
```

b) Buat fungsi aktivasi. Pada praktikum ini fungsi aktivasi yang digunakan adalah sigmoid

```
# Fungsi aktivasi Sigmoid
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)
```

c) Buat fungsi optimizer menggunakan adam.

```python
# Fungsi update Adam optimizer
def adam_update(weight, gradient, m, v, t, learning_rate, beta1, beta2, epsilon=1e-8):
    m = beta1 * m + (1 - beta1) * gradient
    v = beta2 * v + (1 - beta2) * (gradient ** 2)
    m_hat = m / (1 - beta1 ** t)
    v_hat = v / (1 - beta2 ** t)
    weight += learning_rate * m_hat / (np.sqrt(v_hat) + epsilon)
    return weight, m, v
```

d) Buat class NeuralNetwork yang berisi inisialiasi bobot, bias, optimizer dan regularisasi pada model neural network yang akan dibuat.

```python
# Neural network class
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size, l1_lambda=0.0, l2_lambda=0.0):
        # 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)

        # Regularisasi
        self.l1_lambda = l1_lambda
        self.l2_lambda = l2_lambda

        # Adam optimizer parameters
        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
```

e) Buat fungsi forward yang berada pada class NeuralNetwork

```python
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
```

f) Buat fungsi untuk menghitung errornya

```python
def compute_loss(self, targets):
    # Hitung MSE loss
    error = targets - self.predicted_output
    mse_loss = np.mean(np.square(error))

    # regularisasi L1 dan L2
    l1_penalty = self.l1_lambda * (np.sum(np.abs(self.weights_input_hidden)) + np.sum(np.abs(self.weights_hidden_output)))
    l2_penalty = self.l2_lambda * (np.sum(np.square(self.weights_input_hidden)) + np.sum(np.square(self.weights_hidden_output)))

    total_loss = mse_loss + l1_penalty + l2_penalty
    return total_loss
```

g) Buat fungsi backward yang di dalamnya terdapat proses update bobot

```
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)

    # Update bobot menggunakan Adam optimizer
    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
```

h) Buat sebuah fungsi untuk melakukan iterasi sejumlah epoch yang ditentukan, di mana setiap batch data melalui forward pass, backward pass, dan pembaruan bobot.

```
def train(self, training_data, targets, epochs, learning_rate, batch_size):
    error_history = []  # Untuk menyimpan error di setiap epoch
    for epoch in range(epochs):
        total_error = 0
        for i in range(0, len(training_data), batch_size):
            # Buat batch
            batch_inputs = training_data[i:i + batch_size]
            batch_targets = targets[i:i + batch_size].reshape(-1, 1)

            # Forward pass
            self.forward(batch_inputs)

            # Backward pass
            self.backward(batch_inputs, batch_targets, learning_rate)

            # Hitung error dan tambahkan regularisasi
            total_error += self.compute_loss(batch_targets)

        error_history.append(total_error / (len(training_data) / batch_size))
```

```
        # Print error setiap 10 epoch
        if epoch % 10 == 0:
            print(f"Epoch {epoch}, Error: {total_error / (len(training_data) / batch_size)}")

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

i) Impor dataset yang akan digunakan. Gunakan dataset yang ada pada tautan sebelumnya

```
# Load dataset
data = pd.read_csv('transfusion_data.csv')
training_data = data.iloc[:, :-1].values
targets = data.iloc[:, -1].values
```

j) Tentukan hyperparameter yang akan digunakan.

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

k) Panggil NeuralNetwork untuk melakukan inisialisasi dan training pada data, kemudian jalankan program. Amati nilai error yang dihasilkan oleh model.

```
# Inisialisasi dan latih jaringan saraf dengan regularisasi L1 dan L2
nn = NeuralNetwork(input_size, hidden_size, output_size,
                   l1_lambda=0.01, l2_lambda=0.01)
nn.train(training_data, targets, epochs, learning_rate, batch_size)
```

II. Dropout

Pada praktikum regularisasi Dropout, pustaka yang digunakan adalah sklearn untuk membagi data serta keras tensorflow yang menyediakan regularisasi Dropout. Berikut adalah langkah-langkah melakukan regularisasi Dropout menggunakan Tensorflow.

a. Import pustaka yang akan digunakan

```
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
```

b. Import data transfusi darah yang dapat diunduh pada tautan sebelumnya.

```
data = pd.read_csv('transfusion_data.csv')
```

c. Lihat kolom pada data untuk menentukan fitur dan target

```
data.head(3)
```

	Recency (months)	Frequency (times)	Monetary (c.c. blood)	Time (months)	whether he/she donated blood in March 2007
0	2	50	12500	98	1
1	0	13	3250	28	1
2	1	16	4000	35	1

d. Buat variabel X untuk fitur dan variabel y untuk target, kemudian bagi data menjadi data latih dan data uji menggunakan sklearn

```
X = data.iloc[:,:4]
y = data.iloc[:,4]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state=50)
print("X_train shape: ", X_train.shape)
print("y_train shape: ", y_train.shape)
print("X_test shape: ", X_test.shape)
print("y_test shape: ", y_test.shape)
```

e. Buat model menggunakan tensorflow seperti pada gambar berikut.

```
model = Sequential()
model.add(Dense(4, input_shape=(4,), activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(4, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
```

f. Compile model yang telah dibuat dengan menggunakan optimizer **adam**, fungsi loss adalah **mse** dan penghitungan kinerja model menggunakan **accuracy** seperti pada gambar berikut.

```
model.compile(optimizer='adam', loss='mse',
              metrics=['accuracy'])
```

g. Latih model menggunakan data latih yang sudah dibuat kemudian amati hasil erornya.

```
model.fit(x=X_train, y=y_train, epochs = 250)
```

h. Lakukan pengujian terhadap data uji dan bandingkan nilai eror serta akurasinya antara saat pelatihan dan saat pengujian.

```
model.evaluate(X_test, y_test, verbose=0)
```

```
[0.2513369023799896, 0.7486631274223328]
```