Laporan Tugas Besar II

Feed Forward Neural Network

IF4071 Pembelajaran Mesin

Oleh:

- Muhammad Hilmi Asyrofi - 13515083
- Muhammad Rafid Amrullah - 13515125

1.a. Implementasi Classifier from Scratch

Dibuat Neural Network untuk melakukan klasifikasi data weather. Neural Network merupakan fully connected layer yang memiliki jumlah hidden layer maksimal 10. Jumlah node dalam setiap hidden layer dapat bervariasi. Bagian backpropagation diimplementasikan seperti contoh algoritma pada buku Tom Mitchell hal. 98. Neural network menggunakan fungsi aktivasi sigmoid untuk semua hidden layer maupun output layer. Node output untuk klasifikasi berjumlah 1.

Program memberikan pilihan untuk menggunakan momentum atau tidak. Program mengimplementasikan mini-batch stochastic gradient descent. Prorgram Stokasti Gradien Descent diimplementasikan jenis incremental (batch-size=1) dan jenis batch (batch-size=jumlah data).

Fungsi loss yang digunakan pada program yang diimplementasikan kali ini adalah MSE, yaitu:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

Program

In [1]:

```
"""
~~~~~~~~~~
Sebuah kelas yang mengimplementasikan SGD untuk
sebuah feed forward neural network.
"""

#### Libraries
# Standard library
import random

# Third-party libraries
import numpy as np
import time

class Network(object):

    def __init__(self, sizes):
        """List ``sizes`` berisi jumlah neuron sesuai dengan
        urutan layer. Sebagai contoh, jika dimasukkan list [2, 3, 1]
        maka akan digenerate 3 layer network dengan layer input berisi
        2 neuron, hiddel layer berisi 3 neuron dan layer output 1 neuron.
        Bias dan weight diinisialisasi secara random, menggunakan
        distribusi Gaussian dengan mean 0, dan variance 1."""
        self.num_layers = len(sizes)
        self.sizes = sizes
        self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
        self.weights = [np.random.randn(y, x) for x, y in zip(sizes[:-1], sizes[
1:])]
        self.print_status = False

    def feedforward(self, a):
        """Melakukan feed forward dengan input ``a``"""
        activation = a
        for b, w in zip(self.biases, self.weights):
            z = np.dot(w, activation)+b.transpose()[0]
            activation = sigmoid(z)
        return activation

    def SGD(self, training_data, epochs, mini_batch_size, learning_rate,
            momentum=0,test_data=None, print_status=False):
        """Melatih NN dengan mini-batch SGD."""
        start_time = time.time()
        self.print_status = print_status
        if test_data: n_test = len(test_data)
        n = len(training_data)
        for j in range(epochs):
            mini_batches = [
                training_data[k:k+mini_batch_size]
                for k in range(0, n, mini_batch_size)]
            prev_weights = None
            prev_biases = None
            first = True
            for mini_batch in mini_batches:
                if first :
                    prev_weights = self.weights
                    prev_biases = self.biases
                prev_weights, prev_biases = self.update_mini_batch(mini_batch, l
earning_rate, momentum, prev_weights, prev_biases)
    #             if self.print_status :
```

```
#             if test_data:
#                 print("Epoch " +  str(j+1))
#                 print("\tAccuracy: " + str(100*self.evaluate(test_data)/n_tes
t) + "%")
#             else:
#                 print("Epoch " + str(j+1) + " complete")
        elapsed_time = time.time() - start_time
        if test_data:
            print("Accuracy: " + str(100*self.evaluate(test_data)/n_test) + "%")
            print("Execution time: " + str(elapsed_time) + "s")


    def update_mini_batch(self, mini_batch, learning_rate, momentum, prev_weight
s, prev_biases):
        """Update bobot dari network dengan mengaplikasikan
        backrop ke sebuah single mini batch."""
        nabla_b = [np.zeros(b.shape) for b in self.biases]
        nabla_w = [np.zeros(w.shape) for w in self.weights]
        for x, y in mini_batch:
            delta_nabla_b, delta_nabla_w = self.backprop(x, y)
            nabla_b = [nb + dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
            nabla_w = [nw + dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
        temp_weights = self.weights
        self.weights = [w + momentum * pw +(learning_rate/len(mini_batch)) * nw
                        for w, nw, pw in zip(self.weights, nabla_w, prev_weights
)]
        temp_biases = self.biases
        self.biases = [b + momentum *pb +  (learning_rate/len(mini_batch)) * nb
                       for b, nb, pb in zip(self.biases, nabla_b, prev_biases)]
        return (temp_weights, temp_biases)

    def backprop(self, x, y):
        """Mengembalikan nilai tuple ``(nabla_b, nabla_w)`` yang
        merepresentasikan gradien untuk cost function C_x.  ``nabla_b`` dan
        ``nabla_w`` adalah layer lists dari numpy arrays."""
        nabla_b = [np.zeros(b.shape) for b in self.biases]
        nabla_w = [np.zeros(w.shape) for w in self.weights]

        if self.print_status :
            print("FORWARD")
        # feedforward
        activation = x
        activations = [x] # list untuk menyimpan semua activation, layer by laye
r
        zs = [] # list untuk menyimpan net function, layer by layer
        for b, w in zip(self.biases, self.weights):
            z = np.dot(w, activation)+b.transpose()[0]
            zs.append(z)
            activation = sigmoid(z)
            activations.append(activation)

        if self.print_status :
            print("BACKWARD")
        # backward pass
        delta = self.cost_derivative(activations[-1], y) * sigmoid_prime(zs[-1])
        nabla_b[-1] = delta
        nabla_w[-1] = multiply(delta, activations[-2])

        if self.print_status :
            print("delta")
            print(delta)
```

```
            print("activation")
            print(activations[-2])
            print("nabla_w[-1]")
            print(nabla_w[-1])

        for l in range(2, self.num_layers):
            z = zs[-l]
            sp = sigmoid_prime(z)
            delta = np.dot(self.weights[-l+1].transpose(), delta) * sp
            nabla_b[-l] = delta
            nabla_w[-l] = multiply(delta, activations[-l-1].transpose())
            if self.print_status :
                print("delta")
                print(delta)
                print("activation")
                print(activations[-l-1])
                print("nabla_w[-l]")
                print(nabla_w[-l])
        return (nabla_b, nabla_w)

    def evaluate(self, test_data):
        """Mengembalikan jumlah nilai prediksi benar
        dari test data."""
        predicted_class = None
        count = 0
        for (x, y) in test_data :
            if 2*self.feedforward(x) >= 1 :
                predicted_class = 1
            else :
                predicted_class = 0

            if predicted_class == y :
                count += 1
        return count

    def predict(self, test_data):
        """Mengembalikan prediksi terhadap data."""
        predicted_class = None
        count = 0
        predicted = []
        for x in test_data :
            if 2*self.feedforward(x) >= 1 :
                predicted_class = 1
            else :
                predicted_class = 0
            predicted.append(predicted_class)
        return predicted

    def cost_derivative(self, output_activations, y):
        """derivatif parsial dari cost function"""
        return np.squeeze(y-output_activations)

#### Miscellaneous functions
def sigmoid(z):
    """Fungsi sigmoid."""
    return 1.0/(1.0+np.exp(-z))

def sigmoid_prime(z):
    """Turunan fungsi sigmoid."""
    return sigmoid(z)*(1-sigmoid(z))
```

```
def multiply(A, B):
    result = []
    for i in range(len(A)) :
        row = []
        for j in range(len(B)):
            row.append(A[i]*B[j])
        result.append(row)

    return row
```

Pengujian Program dengan Dataset tertentu

In [2]:

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import numpy as np

from sklearn import datasets, metrics
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()

X_train, X_test, y_train, y_test = train_test_split(iris.data[0:100], iris.targe
t[0:100],
    test_size=0.2, random_state=42)
train_data = [(x, y) for x, y in zip(X_train, y_train)]
test_data = [(x, y) for x, y in zip(X_test, y_test)]

neural_network = Network([4,5,10,1])
neural_network.SGD(train_data, 50, 5, 0.1, momentum=0.0001, test_data=test_data)
# neural_network.SGD(train_data, 4000, 5, 0.1, test_data, print_status=True)

# check model load with test data
# score = metrics.accuracy_score(y_test, neural_network.predict(X_test))
# print("Accuracy: " + str(score*100.0) + "%")
```

```
Accuracy: 40.0%
Execution time: 0.4452085494995117s
```

1.b. Implementasi Classifier dengan Keras

In [3]:

```
# Melakukan import library yang dibutuhkan dan import dataset weather
import numpy as np
import pandas as pd
from keras import models
from keras import layers
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

iris = pd.read_csv('dataset/weather.csv')
```

```
Using TensorFlow backend.
```

In [4]:

```
# Melakukan encoding data non-numerik menjadi numerik menggunakan LabelEncoder
# Kemuadian melakukan hold-out dengan split 10%

dt = iris.iloc[:, np.r_[0:1, 3:4]].apply(LabelEncoder().fit_transform)
dt = dt.assign(temp=iris.iloc[:, 1:2])
dt = dt.assign(humidity=iris.iloc[:, 2:3])
cols = dt.columns.tolist()
cols = cols[0:1] + cols[2:] + cols[1:2]
dt = dt[cols]
data_train = np.array(dt)

iris_label = iris.iloc[:,-1]
species_encoder = LabelEncoder().fit(iris_label)
label_target = species_encoder.transform(iris_label)

X_train, X_test, y_train, y_test = train_test_split(data_train, label_target, te
st_size=0.1, random_state=42)
```

In [5]:

```
# Membuat neural network dengan input layer yang tersusun dari 4 node,
# 2 hidden layer yang masing-masing jumlah nodenya 2 dan 10,
# dan layer output dengan jumlah node 1

network = models.Sequential()

network.add(layers.Dense(units=4, activation='sigmoid', input_shape=(4,)))

network.add(layers.Dense(units=2, activation='sigmoid'))

network.add(layers.Dense(units=10, activation='sigmoid'))

network.add(layers.Dense(units=1, activation='sigmoid'))

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

In [6]:

```
# Melakukan pembelajaran pada neural network yang telah dibuat.
# Dari hasil pembelajaran, akurasi test selalu 100%, dapat dilihat dari val_acc
 yang selalu bernilai 1 (100%)

training = network.fit(X_train,
                      y_train,
                      epochs=50,
                      verbose=1,
                      batch_size=5,
                      validation_data=(X_test, y_test))
```

```
Train on 12 samples, validate on 2 samples
Epoch 1/50
12/12 [==============================] - 1s 84ms/step - loss: 0.3433
- acc: 0.4167 - val_loss: 0.5341 - val_acc: 0.0000e+00
Epoch 2/50
12/12 [==============================] - 0s 353us/step - loss: 0.341
0 - acc: 0.4167 - val_loss: 0.5279 - val_acc: 0.0000e+00
Epoch 3/50
12/12 [==============================] - 0s 904us/step - loss: 0.337
9 - acc: 0.4167 - val_loss: 0.5218 - val_acc: 0.0000e+00
Epoch 4/50
12/12 [==============================] - 0s 425us/step - loss: 0.335
6 - acc: 0.4167 - val_loss: 0.5155 - val_acc: 0.0000e+00
Epoch 5/50
12/12 [==============================] - 0s 377us/step - loss: 0.332
8 - acc: 0.4167 - val_loss: 0.5094 - val_acc: 0.0000e+00
Epoch 6/50
12/12 [==============================] - 0s 1ms/step - loss: 0.3307
- acc: 0.4167 - val_loss: 0.5033 - val_acc: 0.0000e+00
Epoch 7/50
12/12 [==============================] - 0s 882us/step - loss: 0.328
0 - acc: 0.4167 - val_loss: 0.4974 - val_acc: 0.0000e+00
Epoch 8/50
12/12 [==============================] - 0s 1ms/step - loss: 0.3253
- acc: 0.4167 - val_loss: 0.4917 - val_acc: 0.0000e+00
Epoch 9/50
12/12 [==============================] - 0s 944us/step - loss: 0.323
3 - acc: 0.4167 - val_loss: 0.4856 - val_acc: 0.0000e+00
Epoch 10/50
12/12 [==============================] - 0s 1ms/step - loss: 0.3207
- acc: 0.4167 - val_loss: 0.4798 - val_acc: 0.0000e+00
Epoch 11/50
12/12 [==============================] - 0s 945us/step - loss: 0.318
7 - acc: 0.4167 - val_loss: 0.4741 - val_acc: 0.0000e+00
Epoch 12/50
12/12 [==============================] - 0s 903us/step - loss: 0.316
9 - acc: 0.4167 - val_loss: 0.4684 - val_acc: 0.0000e+00
Epoch 13/50
12/12 [==============================] - 0s 644us/step - loss: 0.314
0 - acc: 0.4167 - val_loss: 0.4631 - val_acc: 0.0000e+00
Epoch 14/50
12/12 [==============================] - 0s 890us/step - loss: 0.311
7 - acc: 0.4167 - val_loss: 0.4575 - val_acc: 0.0000e+00
Epoch 15/50
12/12 [==============================] - 0s 1ms/step - loss: 0.3098
- acc: 0.4167 - val_loss: 0.4518 - val_acc: 0.0000e+00
Epoch 16/50
12/12 [==============================] - 0s 877us/step - loss: 0.307
4 - acc: 0.4167 - val_loss: 0.4463 - val_acc: 0.0000e+00
Epoch 17/50
12/12 [==============================] - 0s 829us/step - loss: 0.306
0 - acc: 0.4167 - val_loss: 0.4405 - val_acc: 0.0000e+00
Epoch 18/50
12/12 [==============================] - 0s 976us/step - loss: 0.303
1 - acc: 0.4167 - val_loss: 0.4354 - val_acc: 0.0000e+00
Epoch 19/50
12/12 [==============================] - 0s 1ms/step - loss: 0.3013
- acc: 0.4167 - val_loss: 0.4299 - val_acc: 0.0000e+00
Epoch 20/50
12/12 [==============================] - 0s 581us/step - loss: 0.299
9 - acc: 0.4167 - val_loss: 0.4243 - val_acc: 0.0000e+00
```

```
Epoch 21/50
12/12 [==============================] - 0s 901us/step - loss: 0.298
0 - acc: 0.4167 - val_loss: 0.4196 - val_acc: 0.0000e+00
Epoch 22/50
12/12 [==============================] - 0s 1ms/step - loss: 0.2961
- acc: 0.4167 - val_loss: 0.4152 - val_acc: 0.0000e+00
Epoch 23/50
12/12 [==============================] - 0s 1ms/step - loss: 0.2946
- acc: 0.4167 - val_loss: 0.4113 - val_acc: 0.0000e+00
Epoch 24/50
12/12 [==============================] - 0s 682us/step - loss: 0.293
6 - acc: 0.4167 - val_loss: 0.4073 - val_acc: 0.0000e+00
Epoch 25/50
12/12 [==============================] - 0s 963us/step - loss: 0.291
8 - acc: 0.4167 - val_loss: 0.4034 - val_acc: 0.0000e+00
Epoch 26/50
12/12 [==============================] - 0s 700us/step - loss: 0.290
5 - acc: 0.4167 - val_loss: 0.3994 - val_acc: 0.0000e+00
Epoch 27/50
12/12 [==============================] - 0s 633us/step - loss: 0.288
9 - acc: 0.4167 - val_loss: 0.3952 - val_acc: 0.0000e+00
Epoch 28/50
12/12 [==============================] - 0s 1ms/step - loss: 0.2874
- acc: 0.4167 - val_loss: 0.3911 - val_acc: 0.0000e+00
Epoch 29/50
12/12 [==============================] - 0s 803us/step - loss: 0.286
3 - acc: 0.4167 - val_loss: 0.3865 - val_acc: 0.0000e+00
Epoch 30/50
12/12 [==============================] - 0s 822us/step - loss: 0.284
8 - acc: 0.4167 - val_loss: 0.3822 - val_acc: 0.0000e+00
Epoch 31/50
12/12 [==============================] - 0s 645us/step - loss: 0.283
4 - acc: 0.4167 - val_loss: 0.3780 - val_acc: 0.0000e+00
Epoch 32/50
12/12 [==============================] - 0s 718us/step - loss: 0.282
0 - acc: 0.4167 - val_loss: 0.3740 - val_acc: 0.0000e+00
Epoch 33/50
12/12 [==============================] - 0s 1ms/step - loss: 0.2806
- acc: 0.4167 - val_loss: 0.3702 - val_acc: 0.0000e+00
Epoch 34/50
12/12 [==============================] - 0s 923us/step - loss: 0.279
9 - acc: 0.4167 - val_loss: 0.3663 - val_acc: 0.0000e+00
Epoch 35/50
12/12 [==============================] - 0s 710us/step - loss: 0.278
4 - acc: 0.4167 - val_loss: 0.3629 - val_acc: 0.0000e+00
Epoch 36/50
12/12 [==============================] - 0s 679us/step - loss: 0.277
0 - acc: 0.4167 - val_loss: 0.3593 - val_acc: 0.0000e+00
Epoch 37/50
12/12 [==============================] - 0s 812us/step - loss: 0.276
2 - acc: 0.4167 - val_loss: 0.3552 - val_acc: 0.0000e+00
Epoch 38/50
12/12 [==============================] - 0s 638us/step - loss: 0.275
0 - acc: 0.4167 - val_loss: 0.3513 - val_acc: 0.0000e+00
Epoch 39/50
12/12 [==============================] - 0s 567us/step - loss: 0.273
8 - acc: 0.4167 - val_loss: 0.3476 - val_acc: 0.0000e+00
Epoch 40/50
12/12 [==============================] - 0s 825us/step - loss: 0.273
1 - acc: 0.4167 - val_loss: 0.3440 - val_acc: 0.0000e+00
Epoch 41/50
```

```
12/12 [==============================] - 0s 571us/step - loss: 0.271
7 - acc: 0.4167 - val_loss: 0.3411 - val_acc: 0.0000e+00
Epoch 42/50
12/12 [==============================] - 0s 1ms/step - loss: 0.2707
- acc: 0.4167 - val_loss: 0.3382 - val_acc: 0.0000e+00
Epoch 43/50
12/12 [==============================] - 0s 737us/step - loss: 0.269
7 - acc: 0.4167 - val_loss: 0.3347 - val_acc: 0.0000e+00
Epoch 44/50
12/12 [==============================] - 0s 824us/step - loss: 0.268
9 - acc: 0.4167 - val_loss: 0.3308 - val_acc: 0.0000e+00
Epoch 45/50
12/12 [==============================] - 0s 1ms/step - loss: 0.2679
- acc: 0.4167 - val_loss: 0.3267 - val_acc: 0.0000e+00
Epoch 46/50
12/12 [==============================] - 0s 708us/step - loss: 0.266
4 - acc: 0.4167 - val_loss: 0.3230 - val_acc: 0.0000e+00
Epoch 47/50
12/12 [==============================] - 0s 631us/step - loss: 0.265
8 - acc: 0.4167 - val_loss: 0.3190 - val_acc: 0.0000e+00
Epoch 48/50
12/12 [==============================] - 0s 950us/step - loss: 0.264
8 - acc: 0.4167 - val_loss: 0.3153 - val_acc: 0.0000e+00
Epoch 49/50
12/12 [==============================] - 0s 940us/step - loss: 0.263
9 - acc: 0.4167 - val_loss: 0.3119 - val_acc: 0.0000e+00
Epoch 50/50
12/12 [==============================] - 0s 872us/step - loss: 0.263
0 - acc: 0.4167 - val_loss: 0.3088 - val_acc: 0.0000e+00
```

Perbandingan Kinerja Classifier A dan B

Load Dataset

In [7]:

```
# Melakukan encoding data non-numerik menjadi numerik menggunakan LabelEncoder
# Kemuadian melakukan hold-out dengan split 10%

dt = iris.iloc[:, np.r_[0:1, 3:4]].apply(LabelEncoder().fit_transform)
dt = dt.assign(temp=iris.iloc[:, 1:2])
dt = dt.assign(humidity=iris.iloc[:, 2:3])
cols = dt.columns.tolist()
cols = cols[0:1] + cols[2:] + cols[1:2]
dt = dt[cols]
data_train = np.array(dt)

iris_label = iris.iloc[:,-1]
species_encoder = LabelEncoder().fit(iris_label)
label_target = species_encoder.transform(iris_label)

X_train, X_test, y_train, y_test = train_test_split(data_train, label_target, te
st_size=0.1, random_state=42)
```

Kinerja Classifier 1.a.

Batch-size=1

In [8]:

```
neural_network = Network([4,5,10,1])

start_time = time.time()
neural_network.SGD(train_data, 50, 1, 0.1, momentum=0.00001)
elapsed_time = time.time() - start_time

accuracy = metrics.accuracy_score(y_test, neural_network.predict(X_test))
print("Accuracy: " + str(accuracy*100.0) + "%")
print("Execution time: " + str(elapsed_time) + "s")
```

```
Accuracy: 100.0%
Execution time: 0.594524621963501s
```

Batch-size=n

In [9]:

```
neural_network = Network([4,5,10,1])

start_time = time.time()
neural_network.SGD(train_data, len(train_data), 1, 0.1, momentum=0.00001)
elapsed_time = time.time() - start_time

accuracy = metrics.accuracy_score(y_test, neural_network.predict(X_test))
print("Accuracy: " + str(accuracy*100.0) + "%")
print("Execution time: " + str(elapsed_time) + "s")
```

```
Accuracy: 100.0%
Execution time: 0.9152624607086182s
```

Kinerja Classifier 1.b.

Batch-size=1

In [10]:

```
start_time = time.time()
network.summary()
training = network.fit(X_train,
                       y_train,
                       epochs=50,
                       verbose=0,
                       batch_size=1)

scores = network.evaluate(X_test, y_test)
elapsed_time = time.time() - start_time
print("Accuracy: " + str(scores[1]*100.0) + "%")
print("Execution time: " + str(elapsed_time) + "s")
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 4)                 20        
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 10        
_________________________________________________________________
dense_3 (Dense)              (None, 10)                30        
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 11        
=================================================================
Total params: 71
Trainable params: 71
Non-trainable params: 0
_________________________________________________________________
2/2 [==============================] - 0s 124us/step
Accuracy: 100.0%
Execution time: 0.3622128963470459s
```

Batch-size=n

In [11]:

```
start_time = time.time()
network.summary()
training = network.fit(X_train,
                       y_train,
                       epochs=50,
                       verbose=0,
                       batch_size=len(X_train))

scores = network.evaluate(X_test, y_test)
elapsed_time = time.time() - start_time
print("Accuracy: " + str(scores[1]*100.0) + "%")
print("Execution time: " + str(elapsed_time) + "s")
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 4)                 20        
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 10        
_________________________________________________________________
dense_3 (Dense)              (None, 10)                30        
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 11        
=================================================================
Total params: 71
Trainable params: 71
Non-trainable params: 0
_________________________________________________________________
2/2 [==============================] - 0s 127us/step
Accuracy: 100.0%
Execution time: 0.044872760772705078s
```

Analisis Perbandingan Kinerja

Berdasarkan beberapa hasil di atas diperoleh tabel sebagai berikut.

Classifier	Batch-size	Accuracy	Execution time
Classifier 1.a.	1	100%	0.5934426784515381s
Classifier 1.a.	Jumlah data	100%	0.937950849533081s
Classifier 1.b.	1	100%	0.3534877300262451s
Classifier 1.b.	Jumlah data	100%	0.0456850528717041s

Semua jenis Classifier dengan variasi batch-sizenya telah dilakukan pengujian kinerja dan menunjukkan hasil yang bagus, yaitu memiliki akurasi 100%. Berdasarkan tabel tersebut, dapat diperoleh informasi bahwa waktu ekseksusi yang dibutuhkan classifier 1.b cenderung lebih cepat dibandingkan dengan waktu yang diperlukan oleh Classifier 1.a. Hal ini diduga karena Classifier 1.b. merupakan library yang dikembangkan oleh para pakar sehingga telah dilakukan optimasi. Disamping itu, waktu yang dibutuhkan pada Classifier 1.b dengan batch-size=n cenderung lebih cepat karena diduga jumlah peng-update-an yang dilakukan jauh lebih sedikit dibandingkan dengan batch-size=1.

Pembagian Tugas

- 13515083 Muhammad Hilmi Asyrofi - Implementasi 1.a. Classifier from Scratch dan Analisis Perbandingan Kinerja
- 13515125 Muhammad Rafid Amrullah - Implementasi 1.b. Classifier dengan Keras

Dokumentasi pengerjaan tugas besar ini dapat dilihat pada repositori github berikut: https://github.com/mhilmiasyrofi/feed-forward-neural-network (https://github.com/mhilmiasyrofi/feed-forward-neural-network)