TUGAS BESAR
Bagian B: Implementasi Mini-batch Gradient Descent

IF3270 Pembelajaran Mesin

Disusun Oleh:

Gede Prasidha Bhawarnawa	13520004
Moh. Aghna Maysan Abyan	13521076
Yanuar Sano Nur Rasyid	13521110
Muhammad Naufal Nalendra	13521152

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

2023/2024

IMPLEMENTASI

Berikut adalah penjelasan implementasi dari program yang telah dibuat.

❖ MLPClassifier Model

```
iris = load_iris()

learning_data = np.array(iris.data)
learning_target = np.array(iris.target)

classifier = MLPClassifier(
    hidden_layer_sizes=(5,3,3),
    activation='relu',
    max_iter=1000,
    learning_rate='constant',
    learning_rate_init=0.001,
    random_state=1
    )

classifier.fit(learning_data, learning_target)
y_pred = classifier.predict(learning_data)

print("SKLearn MLPClassifier with 5,3,3,2 hidden layers and relu activation
      function")
print(f"Prediction:\n{y_pred}")
print()
print(f"Target:\n{learning_target}")
print()

weight_data = classifier.coefs_
bias_data = classifier.intercepts_

print("layer count: {}".format(classifier.n_layers_))
print("iteration count: {}".format(classifier.n_iter_))
print("output count: {}".format(classifier.n_outputs_))

print("Weight Input: \n", weight_data[0])
print("Bias Input: \n", bias_data[0])
print("Weight Output: \n", weight_data[2])
print("Bias Output: \n", bias_data[2])

error = countSSError(y_pred, learning_target)
print(f"Error: {error / len(learning_data) * 100} %")
```

➢ Penjelasan

Kode di atas adalah contoh penggunaan MLPClassifier dari modul sklearn.neural_network untuk melakukan klasifikasi menggunakan *Artificial Neural Network* (Multilayer Perceptron) pada

dataset Iris

Berikut penjelasan untuk setiap bagian kode:
 	
1. Import Library dan Dataset:
  a. load_iris() dari sklearn.datasets digunakan untuk memuat dataset Iris.
  b. MLPClassifier dari sklearn.neural_network digunakan untuk membuat model MLP.
  c. np.array() dari numpy digunakan untuk mengonversi data dan target ke dalam array numpy.
2. Pelatihan Model:
  a. MLPClassifier diinisialisasi dengan parameter:
    i. hidden_layer_sizes=(5, 3, 3)
    ii. activation='relu':
    iii. max_iter=1000:
    iv. learning_rate='constant':
    v. learning_rate_init=0.001:
    vi. random_state=1:
  b. Model kemudian dipelajari (fit()) dengan menggunakan data latih dan target.
3. Prediksi:
  a. Model melakukan prediksi (predict()) terhadap data latih yang sama.
4. Menampilkan Hasil:
  a. Hasil prediksi dan target kemudian ditampilkan beserta informasi seperti jumlah lapisan, jumlah iterasi, dan jumlah output.
  b. Error dari model dihitung dan ditampilkan.

- **Tensorflow Model**

```
import json
import tensorflow as tf

# Define the loss function
def loss_function(y_true, y_pred, activation_function):
    squared_error = tf.square(y_true - y_pred)
    loss = tf.reduce_sum(squared_error) / tf.cast(tf.shape(y_true)[0], tf.float32)
    return 0.5 * loss

# Load the JSON data
with open('testcase/softmax_two_layer.json') as f:
    data = json.load(f)

# Define the model architecture
class TensorModel(tf.keras.Model):
    def __init__(self, input_size, layers):
        super(TensorModel, self).__init__()
        self.dense_layers = []
        for i, layer in enumerate(layers):
```

```
            activation_function = layer["activation_function"]
            num_neurons = layer["number_of_neurons"]
            self.dense_layers.append(tf.keras.layers.Dense(num_neurons,
                                 activation=activation_function, use_bias=False,
                                 name=f'dense_{i}'))
        self(tf.keras.Input(shape=(input_size,)))

    def call(self, inputs):
        x = inputs
        for layer in self.dense_layers:
            # Concatenate a column vector of ones to the input data
            x_with_bias = tf.concat([tf.ones((tf.shape(x)[0], 1)), x], axis=1)
            x = layer(x_with_bias)
        return x

# Parse JSON data
input_size = data["case"]["model"]["input_size"]
layers = data["case"]["model"]["layers"]

# Extract input data from JSON and add bias
json_input = data["case"]["input"]
input_with_bias = json_input
input_data = tf.constant(input_with_bias, dtype=tf.float32)

# Extract target data from JSON and reshape it
json_target = data["case"]["target"]
target_data = tf.constant(json_target, dtype=tf.float32)
height, width = target_data.shape
target_data = tf.reshape(target_data, [1, height, width])

# Extract other parameters from JSON
initial_weights = [tf.constant(weights, dtype=tf.float32) for weights in
                  data["case"]["initial_weights"]]
learning_rate = data["case"]["learning_parameters"]["learning_rate"]
batch_size = data["case"]["learning_parameters"]["batch_size"]
max_iteration = data["case"]["learning_parameters"]["max_iteration"]
error_threshold = data["case"]["learning_parameters"]["error_threshold"]

# Initialize the model
model = TensorModel(input_size, layers)

# Set the initial weights for the dense layers
for i, weights in enumerate(initial_weights):
    model.dense_layers[i].set_weights([weights])

optimizer = tf.keras.optimizers.SGD(learning_rate=learning_rate)

# Training
for iteration in range(max_iteration):
    total_loss = 0
    for i in range(0, input_data.shape[0], batch_size):
        x_batch = input_data[i:i+batch_size]
        y_batch = target_data[i:i+batch_size]
```

```
        with tf.GradientTape() as tape:
            predictions = model(x_batch, training=True)
            # activation_function = layers[i]["activation_function"]
            loss = loss_function(y_batch, predictions, activation_function)

        gradients = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(gradients, model.trainable_variables))

        total_loss += loss

    avg_loss = total_loss / (input_data.shape[0] / batch_size)

    # Check if the average loss is below the error threshold
    if avg_loss < error_threshold:
        print(f"Error threshold reached. Stopping training at iteration {iteration +
              1}.")
        break

# Print the final weights and check
final_weights = [layer.get_weights()[0] for layer in model.dense_layers]
expected_final_weights = data["expect"]["final_weights"]

print("Final Weights:")
for i, weights in enumerate(final_weights):
    print(f"Layer {i+1} weights:")
    print(weights)
print("Stopped by:", "max_iteration" if iteration == max_iteration - 1 else
      "error_threshold")
```

➢ **Penjelasan**

Kode di atas melakukan pelatihan neural network metode backpropagation menggunakan *library* tensorflow dengan arsitektur dari file JSON *test case* yang diberikan
Berikut penjelasan untuk setiap bagian kode: 1. Import *Library*: Mengimport modul json dan tensorflow 2. Mendefinisikan *loss function*: Fungsi *loss function* didefinisikan untuk menghitung loss model 3. Muat Data JSON: Data JSON dimuat dari directory yang ditentukan 4. Mendefinisikan Arsitektur Model: Kelas TensorModel didefinisikan untuk model neural network dengan backpropagation 5. Parse Data JSON: Data JSON di-*parse* untuk mendapatkan data yang dibutuhkan seperti ukuran input, lapisan-lapisan model, data input dan target, serta parameter pembelajaran lainnya. 6. Inisialisasi Model: Model dibuat menggunakan kelas TensorModel dengan bobot awal yang sudah ditentukan oleh data JSON

7. Inisialisasi Optimizer: Optimizer SGD (Stochastic Gradient Descent) diinisialisasi dengan *learning rate* yang diberikan.
8. Pelatihan Model: Model dilatih dengan menggunakan *loop* hingga jumlah iterasi maksimum atau *error threshold* dilampaui
9. Cetak Bobot Akhir: Menunjukkan bobot akhir model setelah diperbarui. Hasil ini menjadi pembantu untuk menentukan apakah pelatihan model sudah benar

- **Class ActivationFunctions**

```
class ActivationFunctions:
    @staticmethod
    def linear(x, derivative=False):
        if derivative:
            return 1
        return x

    @staticmethod
    def relu(x, derivative=False):
        if derivative:
            return np.where(x <= 0, 0, 1)
        return np.maximum(0, x)

    @staticmethod
    def sigmoid(x, derivative=False):
        if derivative:
            return x * (1 - x)
        return 1 / (1 + np.exp(-x))

    @staticmethod
    def softmax(x, derivative=False):
        if derivative:
            return x * (1 - x)
        return np.exp(x) / np.sum(np.exp(x), axis=0, keepdims=True)
```

- **Penjelasan**

Kode di atas memiliki semua metode statik untuk tiap fungsi aktivasi yang digunakan pada *neural networks*.

Berikut penjelasan untuk setiap bagian kode:

1. linear: Implementasi fungsi aktivasi linear.
2. relu: Implementasi fungsi aktivasi ReLU.
3. sigmoid: Implementasi fungsi aktivasi sigmoid.
4. softmax: Implementasi fungsi aktivasi softmax.

❖ Class NN

```
class NN:
    def __init__(self, model_config, initial_weights):
        self.input_size, self.layers = parse_model(model_config)
        self.weights = np.array(initial_weights)
        self.bias = 1
        self.learning_rate = None
        self.batch_size = None
        self.max_iteration = None
        self.error_threshold = None

    def forward_propagation(self, x):
        input_data = np.array(x)

        for layer, weight in zip(self.layers, self.weights):
            input_data = np.append(self.bias, input_data)
            activation_function = getattr(ActivationFunctions,
                                          layer['activation_function'])
            output = activation_function(np.dot(input_data, weight))

            input_data = output

        return output

    def train(self, input_data, target, learning_rate, batch_size, max_iteration,
              error_threshold):
        self.learning_rate = learning_rate
        self.batch_size = batch_size
        self.max_iteration = max_iteration
        self.error_threshold = error_threshold

        for epoch in range(max_iteration):
            total_error = 0.0

            for i in range(0, len(input_data), batch_size):
                batch_input = input_data[i:i+batch_size]
                batch_target = target[i:i+batch_size]

                batch_error = 0.0

                deltas = [None] * len(self.layers)

                for x, t in zip(batch_input, batch_target):
                    delta = 0
                    output = self.forward_propagation(x)

                    # Compute error
                    error = t - output
                    batch_error += np.sum(error ** 2 / 2)

                    # Backpropagation
                    delta = - error
```

```
            for idx, layer in enumerate(self.layers[::-1]):
                activation = getattr(ActivationFunctions,
                             layer['activation_function'])
                error = -1 * np.log(output) if activation == "softmax" else
                        error
                delta_output = delta * activation(output, derivative=True)
                dw = np.append(self.bias, x)
                delta_output = learning_rate * np.outer(delta_output, dw)

                if idx > 0 and deltas[-idx] is not None:
                    delta_output = np.dot(self.weights[-idx][ 1:].T,
                                   deltas[-idx])

                if deltas[-idx-1] is None:
                    deltas[-idx-1] = delta_output
                else:
                    deltas[-idx-1] += delta_output


        total_error += batch_error

        # Update weights
        for idx, delta in enumerate(deltas):
            self.weights[idx] -= delta.T

    avg_error = total_error / len(input_data)


    print(f"Epoch {epoch + 1}/{max_iteration}, Average Error: {avg_error}")

    if avg_error < error_threshold:
        print("Training stopped: Error threshold reached.")
        break

print("Training complete.")
```

➢ **Penjelasan**

Kode di atas melakukan seluruh proses terkait *feed forward neural network* (FFNN)
Berikut penjelasan untuk setiap bagian kode: 1. __init__ : Inisialisasi semua atribut yang dibutuhkan. 2. forward_propagation: Melakukan *propagation* pada input data melalui *neural network*, seperti proses pada layer, memasukan nilai bias, menerapkan fungsi aktivasi, serta meng-*update* input. 3. train: Melakukan pelatihan *neural network* dengan input data yang ada serta nilai targetnya, seperti melakukan iterasi epoch, pemrosesan batch, perhitungan error,

backpropagation, dan meng-*update* bobot.

HASIL PENGUJIAN

Hasil pengujian melalui perhitungan manual pada Google Spreadsheets dapat dilihat pada *link* berikut : https://docs.google.com/spreadsheets/d/1BokxcCFVVMqee4Me-yi_3ygKHWefYMaPos2YZVXc-sQ/edit?usp=sharing

linear.json

30	Output O1 (Activation)	Output O2 (Activation)	Output O3 (Activation)			TARGET O11	TARGET O12	TARGET O13					
31	0.7	-1.1	0.5			1.3	-0.7	0.1					
32													
33	Error o1	Error 02	Error o3	Error Total									
34	0.18	0.08	0.08	0.34									
35													
36	dE/dOut1	dE/dOut2	dE/dOut3		dOut1/dNet1	dOut2/dNet2	dOut2/dNet3		dNet1/dW01	dNet1/dW02	dNet1/dW03		
37	-0.6	-0.4	0.4		1	1	1		1	1	2		
38													
39	dE/dW01	dE/dW02	dE/dW03						dNet2/dW11	dNet2/dW12	dNet2/dW13		
40	-0.6	-0.6	-1.2						1	1	2		
41	dE/dW11	dE/dW12	dE/dW13										
42	-0.4	-0.4	-0.8						dNet3/dW21	dNet3/dW22	dNet3/dW23		
43	dE/dW21	dE/dW22	dE/dW23						1	1	2		
44	0.4	0.4	0.8										
45						dW01+	dW02+	dW03+		W01+	W02+	W03+	FINAL RESULT
46						-0.06	-0.06	-0.12		0.22	0.64	0.28	
47						dW11+	dW12+	dW13+		W11+	W12+	W13+	
48						-0.04	-0.04	-0.08		0.36	0.3	-0.7	
49						dW21+	dW22+	dW23+		W21+	W22+	W23+	
50						0.04	0.04	0.08		0.11	-0.89	0.37	
51													
52					total	dW01+	dW02+	dW03+					
53						-0.12	-0.24	-0.18					
54						dW11+	dW12+	dW13+					
55						-0.06	-0.1	-0.1					
56						dW21+	dW22+	dW23+					
57						0.09	0.19	0.13					

```
Epoch 1/1, Average Error: 0.3325
Training complete.
[[[ 0.22  0.36  0.11]
  [ 0.64  0.3  -0.89]
  [ 0.28 -0.7   0.37]]]
```

linear_small_lr.json

30	Output O1 (Activation)	Output O2 (Activation)	Output O3 (Activation)			TARGET O11	TARGET O12	TARGET O13					
31	0.7	-1.1	0.5			1.3	-0.7	0.1					
32													
33	Error o1	Error 02	Error o3	Error Total									
34	0.18	0.08	0.08	0.34									
35													
36	dE/dOut1	dE/dOut2	dE/dOut3		dOut1/dNet1	dOut2/dNet2	dOut2/dNet3		dNet1/dW01	dNet1/dW02	dNet1/dW03		
37	-0.6	-0.4	0.4		1	1	1		1	1	2		
38													
39	dE/dW01	dE/dW02	dE/dW03						dNet2/dW11	dNet2/dW12	dNet2/dW13		
40	-0.6	-0.6	-1.2		dW01+	dWO2+	dWO3+		1	1	2		
41	dE/dW11	dE/dW12	dE/dW13		-0.0006	-0.0006	-0.0012						
42	-0.4	-0.4	-0.8		dW11+	dW12+	dW13+		dNet3/dW21	dNet3/dW22	dNet3/dW23		
43	dE/dW21	dE/dW22	dE/dW23		-0.0004	-0.0004	-0.0008		1	1	2		
44	0.4	0.4	0.8		dW21+	dW22+	dW23+						
45					0.0004	0.0004	0.0008			W01+	WO2+	WO3+	
46					total					0.1012	0.4024	0.1018	
47					dW01+	dWO2+	dWO3+			W11+	W12+	W13+	FINAL RESULT
48					-0.0012	-0.0024	-0.0018			0.3006	0.201	-0.799	
49					dW11+	dW12+	dW13+			W21+	W22+	W23+	
50					-0.0006	-0.001	-0.001			0.1991	-0.7019	0.4987	
51					dW21+	dW22+	dW23+						
52					0.0009	0.0019	0.0013						

```
Epoch 1/1, Average Error: 0.3325
Training complete.
[[[ 0.1012  0.3006  0.1991]
  [ 0.4024  0.201  -0.7019]
  [ 0.1018 -0.799   0.4987]]]
```

linear_two_iteration.json

87	Output O1 (Activation)	Output O2 (Activation)	Output O3 (Activation)			TARGET O11	TARGET O12	TARGET O13				
88	1.42	-0.74	-0.04			1.3	-0.7	0.1				
89												
90	Error o1	Error 02	Error o3	Error Total								
91	0.0072	0.0008	0.0098	0.0178								
92												
93	dE/dOut1	dE/dOut2	dE/dOut3		dOut1/dNet1	dOut2/dNet2	dOut2/dNet3		dNet1/dW01	dNet1/dW02	dNet1/dW03	
94	0.12	-0.04	-0.14		1	1	1		1	1	2	
95												
96	dE/dW01	dE/dW02	dE/dW03		dW01+	dWO2+	dWO3+		dNet2/dW11	dNet2/dW12	dNet2/dW13	
97	0.12	0.12	0.24		0.012	0.012	0.024		1	1	2	
98	dE/dW11	dE/dW12	dE/dW13		dW11+	dW12+	dW13+					
99	-0.04	-0.04	-0.08		-0.004	-0.004	-0.008		dNet3/dW21	dNet3/dW22	dNet3/dW23	
100	dE/dW21	dE/dW22	dE/dW23		dW21+	dW22+	dW23+		1	1	2	
101	-0.14	-0.14	-0.28		-0.014	-0.014	-0.028					
102					total				W01+	WO2+	WO3+	
103					dW01+	dWO2+	dWO3+		0.166	0.502	0.214	
104					0.054	0.138	0.066		W11+	W12+	W13+	FINAL RESULT
105					dW11+	dW12+	dW13+		0.338	0.226	-0.718	
106					0.022	0.074	0.018		W21+	W22+	W23+	
107					dW21+	dW22+	dW23+		0.153	-0.789	0.427	
108					-0.043	-0.101	-0.057					

```
Epoch 1/2, Average Error: 0.3325
Epoch 2/2, Average Error: 0.09092500000000002
Training complete.
[[[ 0.166  0.338  0.153]
  [ 0.502  0.226 -0.789]
  [ 0.214 -0.718  0.427]]]
```

relu_b.json

	A	B	C	D	E	F	G	H	I	J	K	L	M
30	Output O1 (Activation)	Output O2 (Activation)	Output O3 (Activation)			TARGET O11	TARGET O12	TARGET O13					
31	0.21	1.05	0.35			0.1	0.1	1					
32													
33	Error o1	Error 02	Error o3	Error Total									
34	0.00605	0.45125	0.21125	0.66855									
35													
36	dE/dOut1	dE/dOut2	dE/dOut3		dOut1/dNet1	dOut2/dNet2	dOut2/dNet3		dNet1/dW01	dNet1/dW02	dNet1/dW03		
37	0.11	0.95	-0.65		1	1	1		1	-0.3	-1		
38													
39	dE/dW01	dE/dW02	dE/dW03						dNet2/dW11	dNet2/dW12	dNet2/dW13		
40	0.11	-0.033	-0.11		dW01+	dWO2+	dWO3+		1	-0.3	-1		
41	dE/dW11	dE/dW12	dE/dW13		0.011	-0.0033	-0.011						
42	0.95	-0.285	-0.95		dW11+	dW12+	dW13+		dNet3/dW21	dNet3/dW22	dNet3/dW23		
43	dE/dW21	dE/dW22	dE/dW23		0.095	-0.0285	-0.095		1	-0.3	-1		
44	-0.65	0.195	0.65		dW21+	dW22+	dW23+						
45					-0.065	0.0195	0.065			W01+	WO2+	WO3+	FINAL RESULT
46										-0.211	0.3033	-0.489	
47				total	dW01+	dWO2+	dWO3+			W11+	W12+	W13+	
48					0.011	-0.0033	-0.011			0.105	0.5285	-0.905	
49					dW11+	dW12+	dW13+			W21+	W22+	W23+	
50					0.095	-0.0285	-0.095			0.885	0.3005	0.291	
51					dW21+	dW22+	dW23+						
52					0.115	0.1995	0.209						

```
Epoch 1/1, Average Error: 1.3967749999999999
Training complete.
[[[-0.211   0.105   0.885 ]
  [ 0.3033  0.5285  0.3005]
  [-0.489  -0.905   0.291 ]]]
```

sigmoid.json

	A	B	C	D	E	F	G	H	I	J	K
465											
466	Epoch	Bias	X1	X2		W01	W02	W03		Learning Rate	
467	10	1	0	0.5		0.2392995654	0.1358761819	0.8337736008		0.1	
468						W11	W12	W13			
469	Sum O1	Sum O2				0.0639296315	0.6436190709	0.2383457449			
470	0.6561863658	0.1831025039									
471											
472											
473	Output O1 (Activation)	Output O2 (Activation)				TARGET OUT1	TARGET OUT2				
474	0.6584031888	0.5456481618				1	0				
475											
476	Error o1	Error 02		Error Total							
477	0.0583441907	0.1488659583		0.207210149							
478											
479	dE/dOut1	dE/dOut2			dOut1/dNet1	dOut2/dNet2			dNet1/dW01	dNet1/dW02	dNet1/dW03
480	-0.3415968112	0.5456481618			0.2249084298	0.2479162453			1	0	0.5
481											
482	dE/dW01	dE/dW02	dE/dW03						dNet2/dW11	dNet2/dW12	dNet2/dW13
483	-0.07682800241	0	-0.03841400121						1	0	0.5
484	dE/dW11	dE/dW12	dE/dW13								
485	0.1352750435	0	0.06763752177								
486					FINAL	W01	W02	W03	RESULT		
487						0.2329117629	0.1288408806	0.8376150009			
488						W11	W12	W13			
489						0.0601534562	0.6484947354	0.2315819927			

```
Epoch 1/10, Average Error: 0.23640507030096655
Epoch 2/10, Average Error: 0.23539036744879444
Epoch 3/10, Average Error: 0.23438378905993162
Epoch 4/10, Average Error: 0.2333853294417322
Epoch 5/10, Average Error: 0.2323949777800034
Epoch 6/10, Average Error: 0.2314127183172226
Epoch 7/10, Average Error: 0.23043853053433763
Epoch 8/10, Average Error: 0.22947238933549008
Epoch 9/10, Average Error: 0.22851426523501334
Epoch 10/10, Average Error: 0.2275641245460821
Training complete.
[[[0.23291176 0.06015346]
  [0.12884088 0.64849474]
  [0.837615   0.23158199]]]
```

mlp.json

57	Output O1 (Activation)	Output O2 (Activation)				TARGET O11	TARGET O12				
58	0	0.12				0.1	1				
59											
60	Error o1	Error o2		Error Total							
61	0.005	0.3872		0.3922							
62											
63	dE/dOut1	dE/dOut2			dOut1/dNet1	dOut2/dNet2			dNet1/dW21	dNet1/dW22	dNet1/dW23
64	-0.1	-0.88			0	1			1	-0.36	-0.2
65											
66	dE/dW21	dE/dW22	dE/dW23		W01	W02	W03		dNet2/dW31	dNet2/dW32	dNet2/dW33
67	0	0	0		0.08592	-0.33872	0.449984		1	-0.36	-0.2
68	dE/dW31	dE/dW32	dE/dW33		W11	W12	W13				
69	-0.88	0.3168	0.176		0.32276	0.46172	0.440072				
70											
71	dE/dH1	dE/dH2							dNet1/dW01	dNet1/dW02	dNet1/dW03
72	0.44	-0.704							1	0.2	-1
73											
74	dE/dW01	dE/dW02	dE/dW03						dNet2/dW11	dNet2/dW12	dNet2/dW13
75	0.44	0.088	-0.44						1	0.2	-1
76	dE/dW11	dE/dW12	dE/dW13		W21	W22	W23				
77	-0.704	-0.1408	0.704		0.2748	0.435904	0.68504				
78					W31	W32	W33				
79					0.188	-0.53168	0.7824				

```
Epoch 1/1, Average Error: 0.338476
Training complete.
[[[ 0.22144   0.1956  ]
  [-0.34752   0.58888 ]
  [ 0.350368  0.41464 ]]

 [[ 0.2748    0.188   ]
  [ 0.3252   -0.4824  ]
  [ 0.71496   0.712   ]]]
```

softmax.json

```
Epoch 1/10, Average Error: 0.6626829218214885
Epoch 2/10, Average Error: 0.6497532419843801
Epoch 3/10, Average Error: 0.6363579868724346
Epoch 4/10, Average Error: 0.622602255574003
Epoch 5/10, Average Error: 0.6086185574566528
Epoch 6/10, Average Error: 0.5945610927498114
Epoch 7/10, Average Error: 0.5805961840120056
Epoch 8/10, Average Error: 0.5668899957351127
Epoch 9/10, Average Error: 0.5535956018123217
Epoch 10/10, Average Error: 0.5408418171596572
Training complete.
[[[ 0.0755546   0.9088089  -0.08597513]
  [-0.18370219  0.75633116  0.2249669 ]
  [ 0.31604839 -0.75296816  0.33318705]
  [ 0.40542907  0.58987544 -0.39566149]
  [ 0.46957724  0.49230158  0.5346057 ]
  [-0.6407751   0.43246096  0.60751439]
  [-0.70024868 -0.32084212  0.71881792]
  [ 0.74063868  0.22981958 -0.77298742]
  [ 0.85639344 -0.06579594  0.00836027]]]
```

```json
"expect": {
    "stopped_by": "max_iteration",
    "final_weights": [
        [
            [ 0.12674605,  0.9149538,  -0.14169985],
            [-0.33551647,  0.67700488,  0.45851159],
            [ 0.48314436, -0.85241216,  0.2692678 ],
            [ 0.3400255,   0.57237542, -0.31240092],
            [ 0.31397716,  0.46349737,  0.72252547],
            [-0.69652442,  0.4789189,   0.61760552],
            [-0.50884515, -0.36354141,  0.57238656],
            [ 0.41891295,  0.26354517, -0.48245812],
            [ 0.90374164, -0.01759501, -0.08614663]
        ]
    ]
}
```

PERBANDINGAN DENGAN PENGGUNAAN LIBRARY TENSORFLOW

1. Linear

```
Final Weights:
Layer 1 weights:
[[ 0.21999998  0.36        0.11000001]
 [ 0.64        0.3        -0.89      ]
 [ 0.28       -0.7         0.37      ]]
Stopped by: max_iteration
```

2. Linear Small LR

```
Final Weights:
Layer 1 weights:
[[ 0.1012      0.30060002  0.1991    ]
 [ 0.40240002  0.201      -0.7019    ]
 [ 0.1018     -0.799       0.4987    ]]
Stopped by: max_iteration
```

3. Linear Two Iteration

```
Final Weights:
Layer 1 weights:
[[ 0.166       0.338       0.15300001]
 [ 0.50200003  0.22600001 -0.789     ]
 [ 0.21400002 -0.718       0.42700002]]
Stopped by: max_iteration
```

4. ReLU B

```
Final Weights:
Layer 1 weights:
[[-0.211       0.10500001  0.885     ]
 [ 0.30330002  0.5285      0.30049998]
 [-0.489      -0.90500003  0.291     ]]
Stopped by: max_iteration
```

5. Sigmoid

```
Final Weights:
Layer 1 weights:
[[0.23291178 0.06015345]
 [0.12884086 0.6484948 ]
 [0.837615   0.23158205]]
Stopped by: max_iteration
```

6. MLP

```
Final Weights:
Layer 1 weights:
[[ 0.08592001  0.32276   ]
 [-0.33872002  0.46172   ]
 [ 0.449984    0.440072  ]]
Layer 2 weights:
[[ 0.2748    0.188   ]
 [ 0.435904 -0.53168 ]
 [ 0.68504   0.7824  ]]
Stopped by: max_iteration
```

7. Softmax

```
Final Weights:
Layer 1 weights:
[[ 0.08524656  0.91434073 -0.09958734]
 [-0.16459174  0.76558226  0.1990096 ]
 [ 0.34101456 -0.73986727  0.2988528 ]
 [ 0.4088521   0.5913957  -0.4002476 ]
 [ 0.49454123  0.505306    0.5001527 ]
 [-0.6362934   0.4352783   0.6010152 ]
 [-0.68642676 -0.31319353  0.69962037]
 [ 0.7592804   0.23958053 -0.79886097]
 [ 0.8613458  -0.06242719  0.00108119]]
Stopped by: max_iteration
```

8. Softmax Two Layer

```
Final Weights:
Layer 1 weights:
[[ 0.1        -0.12026902  0.1        -0.10067384]
 [-0.1         0.01912663 -0.1         0.09731138]
 [ 0.1         0.04000372 -0.1        -0.10199457]]
Layer 2 weights:
[[ 4.8625828e-03  1.5137411e-02]
 [-1.2000000e-01  1.0000000e-01]
 [ 7.8998387e-02 -5.8998372e-02]
 [-1.2000000e-01  1.0000000e-01]
 [ 1.9898925e-02  1.0107547e-04]]
Stopped by: max_iteration
```

Dari percobaan diatas, semua perhitungan dari program yang telah kami buat mendapatkan hasil yang sama dengan nilai target/*expected*, kecuali pada program softmax dan mlp. Hal ini terjadi karena masih ada kesalahan perhitungan pada mlp pada bagian *error hidden layer* yang masih belum sempurna, sedangkan pada softmax terdapat kesalahan pada perhitungan turunan fungsi aktivasi.

PEMBAGIAN TUGAS

NIM	Nama	Pembagian Tugas
13520004	Gede Prasidha Bhawarnawa	Backpropagation
13521076	Moh. Aghna Maysan Abyan	Laporan, perhitungan manual
13521110	Yanuar Sano Nur Rasyid	Backpropagation, perhitungan manual
13521152	Muhammad Naufal Nalendra	Tensorflow, laporan