TubesA_13519103

March 5, 2022

1 Tugas Besar IF3270 - Pembelajaran Mesin Bag. A

Anggota Kelompok:

1. 13519103 - Bryan Rinaldo
2. 13519135 - Naufal Alexander Suryasumirat
3. 13519141 - Naufal Yahya Kurnianto
4. 13519153 - Maximillian Lukman

[1]:
```
import math
import numpy as np
```

[2]:
```
# Activation functions
## Linear
linear = lambda x: x
linear = np.vectorize(linear)
## Sigmoid
sigmoid = lambda x: 1 / (1 + math.exp(-x))
sigmoid = np.vectorize(sigmoid)
## ReLU
relu = lambda x: max(0, x)
relu = np.vectorize(relu)
## Softmax
softmax = lambda x: np.exp(x) / np.exp(x).sum()
## Dict
activation_functions = {
    'linear': linear,
    'sigmoid': sigmoid,
    'relu': relu,
    'softmax': softmax
}
```

[3]:
```
class Layer:
    # n_neuron: number of neuron, weights: weight matrix, activation:␣
 ↪activation function
    def __init__(self, n_neuron: int, weights: np.array, activation: str) ->␣
 ↪None:
        self.n_neuron = n_neuron
```

```
        self.weights = weights
        self.activation = activation
        self.act_function = activation_functions[activation]

    def calculate(self, in_matrix: np.array) -> np.array:
        return self.act_function(np.dot(self.weights.transpose(), in_matrix))

    def get_structure(self) -> tuple((int, np.array, np.array, str, np.array)):
        # n_neuron: int, weight matrix: np.array, bias weight matrix: np.array,␣
↪activatio: string
        n_neuron = self.n_neuron
        weight_neuron = self.weights[:-1,]
        weight_bias = self.weights[-1:,].flatten()
        activation = self.activation
        combinedWeight = self.weights
        return (n_neuron, weight_neuron, weight_bias, activation,␣
↪combinedWeight)
```

[4]:
```
class FFNN:
    def __init__(self,  hidden_layers: list, input_layer = None, threshold = 0.
↪5) -> None:
        self.hidden_layers = hidden_layers
        self.output_layer = hidden_layers[-1]
        self.input_layer = input_layer
        self.threshold = threshold

    def feed_forward(self) -> (np.array or None):
        if (isinstance(self.input_layer, type(None))): return None
        if len(self.input_layer.shape) == 1: return self.forward(self.
↪input_layer)
        else:
            outputs = []
            for data in self.input_layer: outputs.append(self.forward(data))
            if (self.output_layer.activation == 'softmax'): return outputs
            return np.array(outputs).flatten()

    def forward(self, input) -> (np.array or None):
        output = input
        for i in range(0, len(self.hidden_layers)):
            output = self.hidden_layers[i].calculate(np.append(output, 1))
        if (self.output_layer.activation == 'softmax'): return output
        return int(output > self.threshold)

    def attach_hidden_layer(self, hidden_layer: Layer) -> None:
        self.hidden_layers.append(hidden_layer)

    def predict(self, input_layer: np.array) -> list: # input_layer without bias
```

```
        self.input_layer = input_layer
        return self.feed_forward()

    def get_structure(self) -> tuple((np.array, list)):
        return (self.input_layer, [layer.get_structure() for layer in self.
↪hidden_layers])
```

[5]:
```
# Fungsi membaca input file
def _input(filename: str, with_input = False) -> tuple((FFNN, np.array, np.
↪array)):
  f = open(filename, "r")
  f = f.readlines()
  f = [line.strip() for line in f]

  nLayer = int(f[0])
  f = f[1:]
  n_layer_neurons = []
  struct_model = {}

  for i in range(nLayer-1):
    struct_model[i] = {}
    n_layer_neurons.append(int(f[0]))
    struct_model[i]["b"] = [float(b) for b in f[1].split()]
    struct_model[i]["w"] = [[float(w) for w in weights.split()] for weights in␣
↪f[2:(2 + int(f[0]))]]
    struct_model[i]["f"] = f[2 + int(f[0])]
    f = f[2 + int(f[0]) + 1:]

  n_layer_neurons.append(int(f[0]))

  if (with_input):
    n_input = int(f[1])
    f = f[2:]
    input_data = []
    for i in range(n_input):
      input = [int(x) for x in (f[i].split())]
      input_data.append(input)

    f = f[n_input:]
    validation_data = []
    for i in range(n_input):
      result = [int(y) for y in (f[i].split())]
      validation_data.append(result)

  model_layers = []
  for i in range (nLayer-1):
    weight = struct_model[i]["w"]
```

```
    weight.append(struct_model[i]["b"])
    layer = Layer(n_layer_neurons[i+1], np.array(weight), struct_model[i]["f"].
↪lower())
    model_layers.append(layer)

  if (with_input):
    return FFNN(model_layers, np.array(input_data)), input_data, validation_data
  else:
    return FFNN(model_layers)
```

[6]:
```
# Memperlihatkan koefisien dan struktur dari model
def showModel(model: FFNN): #masukan berupa FFNN
  initLayers = model.get_structure()

  countInput = len(initLayers[0])
  countLayer = len(initLayers[1])
  if(initLayers[0].ndim == 1):
    countInput = 1
  else:
    countInput = len(initLayers[0])
  for i in range(0, countInput):
    if(countInput == 1):
      inputLayer = initLayers[0]
    else:
      inputLayer = initLayers[0][i]
    output = inputLayer
    print("==============Data %d==============\n"%(i+1))
    print("-------------Input------------")
    print("Input Layer: ", inputLayer)
    print("------------------------------")
    for j in range(0, countLayer):
      weight = initLayers[1][j][1]
      bias = initLayers[1][j][2]
      activation = initLayers[1][j][3]
      combinedArr = initLayers[1][j][4]

      if(j==(countLayer-1)):
        print("------ Output Layer ------" )
        print("Input: ", output)
        print("Weight: " , weight)
        print("Bias: " , bias)
        print('\n')
        output = activation_functions[activation](np.dot(combinedArr.
↪transpose(), np.append(output, 1)))
        print("------------------------------")
        print(inputLayer)
        print("Predicted Result: ", model.forward(inputLayer))
```

```
      else:
        print("--- Hidden Layer %d ---" %(j+1))
        print("Input: ", output)
        print("H%d Weight: " %(j+1), weight)
        print("H%d Bias: " %(j+1), bias)
        print('\n')
        output = activation_functions[activation](np.dot(combinedArr.
↪transpose(), np.append(output, 1)))
        print("------------------------------")
```

[7]:
```
# Fungsi menghitung akurasi dari model
def calculate_accuracy(model: FFNN, input_set, validation_set: list, is_softmax␣
↪= False):
  # returns range from 1..100 (percentage)
  predicted_set = model.predict(np.array(input_set))
  if (not isinstance(predicted_set, (list, np.ndarray))): return␣
↪int(predicted_set == validation_set[0]) * 100
  if (len(predicted_set) != len(validation_set)): return None
  num_correct = 0
  for i in range(len(predicted_set)):
    if is_softmax:
      if (np.argmax(predicted_set[i]) == validation_set[i]): num_correct += 1
    else:
      if predicted_set[i] == validation_set[i]: num_correct += 1
  return num_correct / len(validation_set) * 100
```

[8]:
```
ffnn_sigmoid, input, validation = _input("./model/model_sigmoid_input.txt",␣
↪True)
ffnn_relu_linear = _input("./model/model_relu_linear.txt")
ffnn_relu = _input("./model/model_relu.txt")
ffnn_sigmax = _input("./model/model_sigmax.txt")

# XOR Dataset
# Bentuk dari input XOR Dataset

# input = np.array([
#     [0, 0],
#     [0, 1],
#     [1, 0],
#     [1, 1]
# ])

# Bentuk dari validation set
# input = [
#     0,
#     1,
#     1,
```

```
#     0
# ]

input_vector = np.array(input) # Konversi input menjadi np.array
```

[9]:
```
# Batch prediction
print("Hasil prediksi batch input")
print("Hasil prediksi model 1 (sigmoid sigmoid):", ffnn_sigmoid.feed_forward())
print("Hasil prediksi model 2 (relu linear):", ffnn_relu_linear.
 ↪predict(input_vector))
print("Hasil prediksi model 3 (relu relu):", ffnn_relu.predict(input_vector))
print("Hasil prediksi model 4 (sigmoid softmax):", ffnn_sigmax.
 ↪predict(input_vector))
```

```
Hasil prediksi batch input
Hasil prediksi model 1 (sigmoid sigmoid): [0 1 1 0]
Hasil prediksi model 2 (relu linear): [0 1 1 0]
Hasil prediksi model 3 (relu relu): [0 1 1 0]
Hasil prediksi model 4 (sigmoid softmax): [array([0.88066832, 0.11933168]),
array([1.39289858e-11, 1.00000000e+00]), array([1.39289858e-11,
1.00000000e+00]), array([9.99997736e-01, 2.26422698e-06])]
```

[10]:
```
# Showing model 1
showModel(ffnn_sigmoid)
```

```
==============Data 1==============

-------------Input------------
Input Layer:  [0 0]
------------------------------
--- Hidden Layer 1 ---
Input:  [0 0]
H1 Weight:  [[ 20. -20.]
 [ 20. -20.]]
H1 Bias:  [-10.  30.]


------------------------------
------ Output Layer ------
Input:  [4.53978687e-05 1.00000000e+00]
Weight:  [[20.]
 [20.]]
Bias:  [-30.]


------------------------------
[0 0]
Predicted Result:  0
```

```
==============Data 2==============

-------------Input------------
Input Layer:  [0 1]
------------------------------
--- Hidden Layer 1 ---
Input:  [0 1]
H1 Weight:  [[ 20. -20.]
 [ 20. -20.]]
H1 Bias:  [-10.  30.]


------------------------------
------ Output Layer ------
Input:  [0.9999546 0.9999546]
Weight:  [[20.]
 [20.]]
Bias:  [-30.]


------------------------------
[0 1]
Predicted Result:  1
==============Data 3==============

-------------Input------------
Input Layer:  [1 0]
------------------------------
--- Hidden Layer 1 ---
Input:  [1 0]
H1 Weight:  [[ 20. -20.]
 [ 20. -20.]]
H1 Bias:  [-10.  30.]


------------------------------
------ Output Layer ------
Input:  [0.9999546 0.9999546]
Weight:  [[20.]
 [20.]]
Bias:  [-30.]


------------------------------
[1 0]
Predicted Result:  1
==============Data 4==============
```

```
-------------Input------------
Input Layer:  [1 1]
------------------------------
--- Hidden Layer 1 ---
Input:  [1 1]
H1 Weight:  [[ 20. -20.]
 [ 20. -20.]]
H1 Bias:  [-10.  30.]


------------------------------
------ Output Layer ------
Input:  [1.00000000e+00 4.53978687e-05]
Weight:  [[20.]
 [20.]]
Bias:  [-30.]


------------------------------
[1 1]
Predicted Result:  0
```

[11]:
```
# Showing model 2
showModel(ffnn_relu_linear)
```

```
==============Data 1==============

-------------Input------------
Input Layer:  [0 0]
------------------------------
--- Hidden Layer 1 ---
Input:  [0 0]
H1 Weight:  [[1. 1.]
 [1. 1.]]
H1 Bias:  [ 0. -1.]


------------------------------
------ Output Layer ------
Input:  [0 0]
Weight:  [[ 1.]
 [-2.]]
Bias:  [0.]


------------------------------
[0 0]
Predicted Result:  0
```

```
==============Data 2==============

-------------Input------------
Input Layer:  [0 1]
------------------------------
--- Hidden Layer 1 ---
Input:  [0 1]
H1 Weight:  [[1. 1.]
 [1. 1.]]
H1 Bias:  [ 0. -1.]


------------------------------
------ Output Layer ------
Input:  [1. 0.]
Weight:  [[ 1.]
 [-2.]]
Bias:  [0.]


------------------------------
[0 1]
Predicted Result:  1
==============Data 3==============

-------------Input------------
Input Layer:  [1 0]
------------------------------
--- Hidden Layer 1 ---
Input:  [1 0]
H1 Weight:  [[1. 1.]
 [1. 1.]]
H1 Bias:  [ 0. -1.]


------------------------------
------ Output Layer ------
Input:  [1. 0.]
Weight:  [[ 1.]
 [-2.]]
Bias:  [0.]


------------------------------
[1 0]
Predicted Result:  1
==============Data 4==============
```

```
-------------Input------------
Input Layer:  [1 1]
------------------------------
--- Hidden Layer 1 ---
Input:  [1 1]
H1 Weight:  [[1. 1.]
 [1. 1.]]
H1 Bias:  [ 0. -1.]


------------------------------
------ Output Layer ------
Input:  [2. 1.]
Weight:  [[ 1.]
 [-2.]]
Bias:  [0.]


------------------------------
[1 1]
Predicted Result:  0
```

[12]:
```
# Showing model 3
showModel(ffnn_relu)
```

```
==============Data 1==============

-------------Input------------
Input Layer:  [0 0]
------------------------------
--- Hidden Layer 1 ---
Input:  [0 0]
H1 Weight:  [[1. 1.]
 [1. 1.]]
H1 Bias:  [ 0. -1.]


------------------------------
------ Output Layer ------
Input:  [0 0]
Weight:  [[ 1.]
 [-2.]]
Bias:  [0.]


------------------------------
[0 0]
Predicted Result:  0
```

```
==============Data 2==============

-------------Input------------
Input Layer:  [0 1]
------------------------------
--- Hidden Layer 1 ---
Input:  [0 1]
H1 Weight:  [[1. 1.]
 [1. 1.]]
H1 Bias:  [ 0. -1.]


------------------------------
------ Output Layer ------
Input:  [1. 0.]
Weight:  [[ 1.]
 [-2.]]
Bias:  [0.]


------------------------------
[0 1]
Predicted Result:  1
==============Data 3==============

-------------Input------------
Input Layer:  [1 0]
------------------------------
--- Hidden Layer 1 ---
Input:  [1 0]
H1 Weight:  [[1. 1.]
 [1. 1.]]
H1 Bias:  [ 0. -1.]


------------------------------
------ Output Layer ------
Input:  [1. 0.]
Weight:  [[ 1.]
 [-2.]]
Bias:  [0.]


------------------------------
[1 0]
Predicted Result:  1
==============Data 4==============
```

```
-------------Input------------
Input Layer:  [1 1]
------------------------------
--- Hidden Layer 1 ---
Input:  [1 1]
H1 Weight:  [[1. 1.]
 [1. 1.]]
H1 Bias:  [ 0. -1.]


------------------------------
------ Output Layer ------
Input:  [2. 1.]
Weight:  [[ 1.]
 [-2.]]
Bias:  [0.]


------------------------------
[1 1]
Predicted Result:  0
```

[13]:
```
# Showing model 4
showModel(ffnn_sigmax)
```

```
==============Data 1==============

-------------Input------------
Input Layer:  [0 0]
------------------------------
--- Hidden Layer 1 ---
Input:  [0 0]
H1 Weight:  [[ 20. -20.]
 [ 20. -20.]]
H1 Bias:  [-10.  30.]


------------------------------
------ Output Layer ------
Input:  [4.53978687e-05 1.00000000e+00]
Weight:  [[-10.  17.]
 [-20.  18.]]
Bias:  [ 30. -10.]


------------------------------
[0 0]
Predicted Result:  [0.88066832 0.11933168]
```

```
==============Data 2==============

-------------Input------------
Input Layer:  [0 1]
------------------------------
--- Hidden Layer 1 ---
Input:  [0 1]
H1 Weight:  [[ 20. -20.]
 [ 20. -20.]]
H1 Bias:  [-10.  30.]


------------------------------
------ Output Layer ------
Input:  [0.9999546 0.9999546]
Weight:  [[-10.  17.]
 [-20.  18.]]
Bias:  [ 30. -10.]


------------------------------
[0 1]
Predicted Result:  [1.39289858e-11 1.00000000e+00]
==============Data 3==============

-------------Input------------
Input Layer:  [1 0]
------------------------------
--- Hidden Layer 1 ---
Input:  [1 0]
H1 Weight:  [[ 20. -20.]
 [ 20. -20.]]
H1 Bias:  [-10.  30.]


------------------------------
------ Output Layer ------
Input:  [0.9999546 0.9999546]
Weight:  [[-10.  17.]
 [-20.  18.]]
Bias:  [ 30. -10.]


------------------------------
[1 0]
Predicted Result:  [1.39289858e-11 1.00000000e+00]
==============Data 4==============
```

```
-------------Input------------
Input Layer:  [1 1]
------------------------------
--- Hidden Layer 1 ---
Input:  [1 1]
H1 Weight:  [[ 20. -20.]
 [ 20. -20.]]
H1 Bias:  [-10.  30.]


------------------------------
------ Output Layer ------
Input:  [1.00000000e+00 4.53978687e-05]
Weight:  [[-10.  17.]
 [-20.  18.]]
Bias:  [ 30. -10.]


------------------------------
[1 1]
Predicted Result:  [9.99997736e-01 2.26422698e-06]
```

[14]:
```
# Perhitungan akurasi model
print(input)
print("Akurasi model 1: ", calculate_accuracy(ffnn_sigmoid, input, validation),␣
 ↪"%")
print("Akurasi model 2: ", calculate_accuracy(ffnn_relu_linear, input,␣
 ↪validation), "%")
print("Akurasi model 3: ", calculate_accuracy(ffnn_relu, input, validation),␣
 ↪"%")
print("Akurasi model 4: ", calculate_accuracy(ffnn_sigmax, input, validation,␣
 ↪True), "%")
```

```
[[0, 0], [0, 1], [1, 0], [1, 1]]
Akurasi model 1:  100.0 %
Akurasi model 2:  100.0 %
Akurasi model 3:  100.0 %
Akurasi model 4:  100.0 %
```

[15]:
```
input1 = np.array([1,1])
input2 = np.array([1,0])
input3 = np.array([0,0])
input4 = np.array([0,1])

print("Hasil Prediksi Single")
print("Hasil Prediksi Single model 1: ", ffnn_sigmoid.predict(input1))
print("Hasil Prediksi Single model 2: ", ffnn_relu_linear.predict(input2))
print("Hasil Prediksi Single model 3: ", ffnn_relu.predict(input3))
```

```
print("Hasil Prediksi Single model 4: ", ffnn_sigmax.predict(input4))
```

```
Hasil Prediksi Single
Hasil Prediksi Single model 1:  0
Hasil Prediksi Single model 2:  1
Hasil Prediksi Single model 3:  0
Hasil Prediksi Single model 4:  [1.39289858e-11 1.00000000e+00]
```