Neural Network

Implementasi Oleh Kelompok

In [37]:
```
import math
import numpy as np

def sigmoid(x):
  return 1 / (1 + math.exp(-x))
```

In [38]:
```
import random

class NeuralNet:

    # Constructor
    def __init__(self, nodes_n_in_hidden_layers, learning_rate, momentum):
        nodes_n_in_hidden_layers.append(1) # satu node buat output
        self.nodes_n_in_hidden_layers = nodes_n_in_hidden_layers
        self.inputs = [] #data input
        self.learning_rate = learning_rate
        self.momentum = momentum
        self.outputs = [] # output dari setiap node pada satu iterasi
        self.weights = [] # weight dari setiap edge pada satu iterasi
        self.biases = [] # bias dari setiap node pada satu iterasi
        self.weight_biases = []
        self.local_gradients = [] # local gradient dari setiap node pada satu iterasi
        self.delta_weights = [] # delta weight dari setiap edge pada satu iterasi
        self.delta_biases = [] # delta bias dari setiap node pada satu iterasi
        self.layer_nodes = [] # node-node pada layer-layer
        self.v = [] # v pada setiap node
        self.targets = [] # target dari data input
        self.test_outputs = [] # hasil predict data test
        self.test_labels = [] # label hasil predict data test

    # Feed Forward
    def feed_forward(self, datum_idx):
        for i in range (1, len(self.layer_nodes)):
            for j in range (0, len(self.layer_nodes[i])):
                current_node = self.layer_nodes[i][j]
                weights = []
                weights.append(self.biases[current_node])
                inputs = []
                inputs.append(1)
                for k in range(0, len(self.layer_nodes[i-1])):
                    if (self.weights[self.layer_nodes[i-1][k]][current_node] != None):
                            weights.append(self.weights[self.layer_nodes[i-1][k]][current_n
ode])

                    if (i==1):
                        inputs.append(self.inputs[datum_idx][self.layer_nodes[i-1][k]])
                    else:
                        inputs.append(self.outputs[self.layer_nodes[i-1][k]])
                v = np.dot(inputs, weights)
                self.v[current_node] = v
                self.outputs[current_node] = sigmoid(v)

    # Back Propagation
    def back_propagation(self, datum_idx):
        for i in range(len(self.layer_nodes)-1, 0, -1):
            for j in range(len(self.layer_nodes[i])-1, -1, -1):
                current_node = self.layer_nodes[i][j]
                if (i == len(self.layer_nodes)-1):
                    v = self.v[current_node]
                    sig_v = sigmoid(v)
                    self.local_gradients[current_node] = self.local_gradients[current_node]
 + (sig_v * (1 - sig_v) * (self.targets[datum_idx] - self.outputs[current_node]))
                else:
                    v = self.v[current_node]
                    sig_v = sigmoid(v)
                    weight_delta = 1
                    for k in range(0, len(self.layer_nodes[i+1])):
                        weight_delta = weight_delta * self.local_gradients[self.layer_nodes
[i+1][k]] * self.weights[current_node][self.layer_nodes[i+1][k]]
                    self.local_gradients[current_node] = self.local_gradients[current_node]
 + (sig_v * (1 - sig_v) * weight_delta)

    # Update Weight
    def update_weight(self):
        for i in range(0, len(self.layer_nodes)-1):
            for j in range(0, len(self.layer_nodes[i])):
                current_node = self.layer_nodes[i][j]
```

```
                for k in range(0, len(self.layer_nodes[i+1])):
                    current_next_node = self.layer_nodes[i+1][k]
                    self.delta_weights[current_node][current_next_node] = self.momentum * s
elf.delta_weights[current_node][current_next_node] + self.learning_rate * self.local_gradie
nts[current_next_node] * self.outputs[current_next_node]
                    self.weights[current_node][current_next_node] += self.delta_weights[cur
rent_node][current_next_node]
        for i in range(1, len(self.biases)):
            new_bias = self.biases[i] + self.momentum  * self.delta_biases[i] + self.learni
ng_rate * self.local_gradients[i]
            self.delta_biases[i] = new_bias - self.biases[i]
            self.biases[i] = new_bias

    # Fit
    def fit(self, X, Y, batch_size, max_iter, threshold): # data = array of arrays
        #data[0] ke n merupakan label
        #nodes_n_in_hidden_layers[0] merupakan jumlah input
        self.nodes_n_in_hidden_layers.insert(0, len(X[0]))

        self.targets = Y
        self.inputs = X

        n_nodes = 0
        init_weight = 1 # Weights diinisalisasi 0

        # Inisialisasi output, bias, local gradient, v, dan delta bias di setiap node pada
 layer
        for i in range(0, len(self.nodes_n_in_hidden_layers)):
            l_nodes = []
            for j in range(0, self.nodes_n_in_hidden_layers[i]):
                self.outputs.append(0)
                self.v.append(0)
                self.biases.append(0) #asumsi x bias = 1
                self.local_gradients.append(0)
                self.delta_biases.append(0)
                l_nodes.append(n_nodes)
                n_nodes += 1
            self.layer_nodes.append(l_nodes)

        for i in range(0, n_nodes):
            self.weights.append([])
            self.delta_weights.append([])
            for j in range(0, n_nodes):
                self.weights[i].append(None)
                self.delta_weights[i].append(None)

        current_node = 0
        for i in range(0, len(self.nodes_n_in_hidden_layers)-1):
            if (i < len(self.nodes_n_in_hidden_layers)-1):
                next_layer_first_node = current_node + self.nodes_n_in_hidden_layers[i]
                for j in range(0, self.nodes_n_in_hidden_layers[i]):
                    for k in range(0, self.nodes_n_in_hidden_layers[i+1]):
                        self.weights[current_node][k+next_layer_first_node] = random.unifor
m(-0.5,0.5)
                        self.delta_weights[current_node][k+next_layer_first_node] = 0
                    current_node += 1

        n_batch = math.ceil(len(Y)/batch_size)

        n_iter = 0
        error = 100
        while (n_iter < max_iter and error > threshold):
            datum_idx = 0
            for i in range(0, n_batch):
                # Mengembalikan local_gradient menjadi 0
                for i in range(0, n_nodes):
                    self.local_gradients[i] = 0

                j = 0
                accum_error = 0
                num_datum = 0
                while (j < batch_size):
```

```
                    if (datum_idx < len(Y)):
                        self.feed_forward(datum_idx)
                        accum_error = accum_error + (0.5 * ((self.targets[datum_idx] - self
.outputs[-1])**2))
                        num_datum = num_datum + 1
                        self.back_propagation(datum_idx)
                        datum_idx += 1
                        j += 1
                    else:
                        j = batch_size + 1
                error = accum_error / num_datum
                if (error < threshold):
                    break
                self.update_weight()

            n_iter += 1

    # Predict
    def predict(self, data_test):
        feed_forward_result = []
        self.inputs = data_test

        for i in range (0, len(data_test)):
            self.feed_forward(i)
            self.test_outputs.append(self.outputs[-1])

        return self.test_outputs

    def predict_label(self):
        for i in range (0, len(self.test_outputs)):
            if self.test_outputs[i] >= 0.5:
                self.test_labels.append(1)
            else:
                self.test_labels.append(0)
        return self.test_labels
```

Penjelasan Model Neural Network

Model Neural Network pada program kami diimplementasikan dengan mendefinisikan variabel-variabel (inputs, outputs, jumlah node pada satu layer, dll) dalam bentuk matrix dan list. Pada saat penciptaan suatu obyek Neural Network, diperlukan parameter jumlah node di setiap layer, learning rate, dan momentum. Setelah itu, obyek Neural Network yang terbentuk akan menginisiasi beberapa variabel seperti weights, biases, delta weights, delta biases, local gradients, dan beberapa variabel lainnya.

Pada kelas Neural Network, terdapat sejumlah fungsi, yaitu:

1. Feed forward
2. Back propagation
3. Update weight
4. Fit
5. Predict
6. Predict label

1. Feed Forward

Pada fungsi ini, program akan melakukan pembelajaran terhadap setiap node dengan memasukkan seluruh weight (termasuk weight bias) dan seluruh nilai (termasuk bias) yang menuju ke node tersebut ke dalam list weight dan list input. Kedua list tersebut kemudian di kalikan secara dot matrix yang kemudian hasil tersebut akan dimasukkan ke dalam fungsi sigmoid untuk mendapatkan satu output dari node tersebut. Hasil output tersebut kemudian dijadikan input untuk menghitung output pada node selanjutnya.

2. Back Propagation

Pada fungsi back propagation, dilakukan perhitungan untuk memperbarui nilai local gradient setiap node.

3. Update Weight

Pada fungsi update weight, dilakukan perhitungan untuk mendapatkan delta weight dan weight baru. Adapun delta_weight = momentum x delta_weight sebelumnya + learning_rate x local_gradient x output. Perhitungan ini dilakukan bukan hanya pada hubungan antarsimpul, tetapi juga pada bias tiap simpul.

4. Fit

Pada fungsi ini program melakukan pembelajaran terhadap data input. Pembelajaran dilakukan dengan mengkombinasikan ketiga fungsi sebelumnya yaitu feed forward, back propagation, dan update weight yang dieksekusi secara berurutan yang dilakukan sebanyak max_iter atau dilakukan hingga error yang dihasilkan <= threshold.

Selain itu pada fungsi ini diimplementasikan juga mini-batch stochastic gradient descent. Implementasi dilakukan dengan cara menerima input berupa batch_size. Program kemudian akan melakukan update weight setelah dilakukan feed forward dan back propagation sebanyak n_batch kali, di mana n_batch adalah jumlah datum dibagi dengan batch_size. Jika nilainya tidak bulat, diambil pembulatan ke atas.

5. Predict

Pada fungsi ini program melakukan prediksi terhadap data test. Prediksi dilakukan dengan cara memanfaatkan model hasil fit dan menjalankan fungsi feed forward untuk setiap data test.

6. Predict Label

Fungsi ini hanya mengembalikan label hasil prediksi.

Implementasi dengan Keras

In [44]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers

class KerasNeuralNet:

    # Construction
    def __init__(self, nodes_n_in_hidden_layers, learning_rate, momentum):
        nodes_n_in_hidden_layers.append(1) # satu node buat output
        self.learning_rate = learning_rate
        self.momentum = momentum
        self.model = Sequential()
        self.model.add(Dense(output_dim=nodes_n_in_hidden_layers[0], input_dim=4, activatio
n="sigmoid"))
        for i in range(1, len(nodes_n_in_hidden_layers)):
            self.model.add(Dense(output_dim=nodes_n_in_hidden_layers[i], input_dim=nodes_n_
in_hidden_layers[i-1], activation='sigmoid'))
        sgd = optimizers.SGD(lr=learning_rate, momentum=momentum, nesterov=False)
        self.model.compile(loss='mean_squared_error', optimizer=sgd, metrics=['accuracy'])

    def fit(self, X, Y, batch_size, max_iter):
        self.model.fit(X, Y, batch_size=batch_size, epochs=max_iter)

    def predict(self, X):
        return self.model.predict(X)
```

Penjelasan Implementasi dengan Keras

Pada kelas KerasNeuralNet, terdapat 3 fungsi yaitu:

1. **init**
2. fit
3. predict

1. init

Untuk membuat Neural Network dengan Keras, masukan yang dibutuhkan adalah jumlah simpul pada setiap layer, learning_rate, dan momentum. Pada inisialisasi, Neural Network akan membuat model dengan fungsi Sequential yang sudah tersedia dengan Keras. Layer pertama akan memiliki input_dim berdasarkan jumlah atribut data, sedangkan layer-layer setelahnya akan memiliki input_dim berdasarkan jumlah simpul pada layer sebelumnya. Output_dim = jumlah node pada layer setelahnya. Optimizer yang digunakan adalah Stochastic Gradient Descent dengan lr = learning_rate, momentum=momentum, dan nesterov=False. Nilai nesterov diberikan False agar lebih mirip dengan implementasi kelompok, di mana implementasi kelompok tidak menggunakan nesterov momentum. Dengan alasan yang sama, loss yang digunakan adalah mean_squared_error.

2. fit

Fungsi memanggil fungsi fit yang sudah tersedia untuk objek Sequential pada Keras.

3. predict

Fungsi memanggil fungsi predict yang sudah tersedia untuk objek Sequential pada Keras.

Membaca Data Tennis

In [40]:
```
import pandas as pd
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler

# Load data
dataframe = pd.read_csv('tennis.csv')

# Transform outlook, temperature, humidity, and windy to numerical values
le = preprocessing.LabelEncoder()
encoded = dataframe.apply(le.fit_transform)
dataset = encoded.values

# X and Y values
X = dataset[:,0:4]
Y = dataset[:,4]

# Rescale min and max for X
scaler = MinMaxScaler(feature_range=(0, 1))
rescaledX = scaler.fit_transform(X)
```

```
/home/suzaneringoringo/anaconda3/lib/python3.6/site-packages/sklearn/utils/validation.py:47
5: DataConversionWarning: Data with input dtype int64 was converted to float64 by MinMaxSca
ler.
  warnings.warn(msg, DataConversionWarning)
```

Fitting dengan Hasil Implementasi Kelompok

In [41]:
```
nn = NeuralNet([6], 0.25, 0.0001)
nn.fit(rescaledX, Y, 1, 5, 0.01)
nn.predict(rescaledX)
print(nn.test_outputs)
print(nn.predict_label())
```

```
[0.6371934873427952, 0.6336026730531541, 0.6539361621846131, 0.6508008415326048, 0.64504601
98663713, 0.6413939157872407, 0.6496933577094223, 0.6424431892105261, 0.6366263075766724,
0.6552220214899362, 0.6432353410188569, 0.6552559471539289, 0.6583535244641116, 0.647059737
1112335]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

Fitting dengan Keras

In [45]:
```
knn = KerasNeuralNet([6], 0.25, 0.0001)

# Train model
knn.fit(rescaledX, Y, 1, 5)

# Predict
labels = knn.predict(rescaledX)
print(labels)
for i in range(0, len(labels)):
    if (labels[i] >= 0.5):
        labels[i] = 1
    else:
        labels[i] = 0
print(labels)
```

```
/home/suzaneringoringo/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:13: User
Warning: Update your `Dense` call to the Keras 2 API: `Dense(input_dim=4, activation="sigmo
id", units=6)`
  del sys.path[0]
/home/suzaneringoringo/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:15: User
Warning: Update your `Dense` call to the Keras 2 API: `Dense(input_dim=6, activation="sigmo
id", units=1)`
  from ipykernel import kernelapp as app

Epoch 1/5
14/14 [==============================] - 0s 16ms/step - loss: 0.2565 - acc: 0.5714
Epoch 2/5
14/14 [==============================] - 0s 930us/step - loss: 0.2412 - acc: 0.6429
Epoch 3/5
14/14 [==============================] - 0s 911us/step - loss: 0.2368 - acc: 0.6429
Epoch 4/5
14/14 [==============================] - 0s 970us/step - loss: 0.2324 - acc: 0.6429
Epoch 5/5
14/14 [==============================] - 0s 922us/step - loss: 0.2331 - acc: 0.6429
[[ 0.63403666]
 [ 0.59249121]
 [ 0.68072295]
 [ 0.66368133]
 [ 0.66530591]
 [ 0.62578082]
 [ 0.64969957]
 [ 0.64091301]
 [ 0.64231777]
 [ 0.67312676]
 [ 0.61770022]
 [ 0.64808822]
 [ 0.69283575]
 [ 0.62445682]]
[[ 1.]
 [ 1.]
 [ 1.]
 [ 1.]
 [ 1.]
 [ 1.]
 [ 1.]
 [ 1.]
 [ 1.]
 [ 1.]
 [ 1.]
 [ 1.]
 [ 1.]
 [ 1.]]
```

Perbandingan Hasil Implementasi Kelompok dengan Keras

Meskipun label yang dihasilkan sama persis, nilai sebenarnya dari feed_forward berbeda. Hal ini dapat terjadi karena nilai weights pada saat inisialisasi adalah random sehingga kemungkinan besar berbeda.

Pembagian Tugas

1. Helena Suzane Graciella (13515032)
    - Update Weight
    - Implementasi Keras
    - Fit
    - Laporan
2. Lathifah Nurrahmah (13515046)
    - Back Propagation
    - Fit
    - Predict
    - Laporan
3. Aya Aurora Rimbamorani (13515098)
    - Feed Forward
    - Fit
    - Predict & Predict Label
    - Laporan