Laporan Tugas Besar II
IF4071 Pembelajaran Mesin

FEED FORWARD NEURAL NETWORK

oleh

13515021	Dewita Sonya T.
13515057	Erick Wijaya
13515063	Kezia Suhendra

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2018

Daftar Isi

Source Code

Seluruh kode program *classifier* dapat diakses pada repositori github melalui pranala https://github.com/tugas-itb-erick/feed-neural-network. Berikut adalah potongan kode program untuk setiap algoritma beserta penjelasannya.

1. Classifier A - Feed Forward Neural Network

File: *main.ipynb*

```
class NNClassifier:
    def __init__(self, n_nodes=[], lrate=0.05, momentum=0, batch_size=1, max_iter=100):
        # Checking parameter input
        if (len(n_nodes) > self.__MAX_HIDDEN):
            raise ValueError('Number of hidden layers cannot be greater than
{}'.format(self.__MAX_HIDDEN))

        if (not all(x > 0 for x in n_nodes)):
            raise ValueError('Number of nodes in a layer cannot be nonpositive')

        if (batch_size <= 0):
            raise ValueError('Batch size cannot be nonpositive')

        # Setting parameter
        self.n_nodes = n_nodes
        self.n_hiddens = len(n_nodes)
        self.lrate = lrate
        self.momentum = momentum
        self.batch_size = batch_size
        self.max_iter = max_iter
        self.weights = []
        self.prev_weights = []

    @property
    def __MAX_HIDDEN(self):
        return 10

    def __stochastic_gradient_descend(self, data, target):
        for x, y in zip(data, target):
            x = np.append(x, 1.)
            values_layers = self.__feed_forward(x)
            errors_layers = self.__backward_prop(y, values_layers)
            values_layers.insert(0, x)

            # Update weight
            new_weights = []
            for ilayer, (weights_per_layer, prev_weights_per_layer) in enumerate(zip(self.weights,
```

```
self.prev_weights)):
                new_weights_per_layer = []
                for inode, (weight_all, prev_weight_all) in enumerate(zip(weights_per_layer,
prev_weights_per_layer)):
                    new_weight_all = []
                    for iweight, (weight, prev_weight) in enumerate(zip(weight_all,
prev_weight_all)):
                        new_weight_all.append(self.__calculate_weight(weight, prev_weight,
                        values_layers[ilayer][inode], errors_layers[ilayer][iweight]))
                    new_weights_per_layer.append(new_weight_all)
                new_weights.append(np.array(new_weights_per_layer))
            for i in range(len(new_weights)):
                for j in range(len(new_weights[i])):
                    for k in range(len(new_weights[i][j])):
                        self.prev_weights[i][j][k] = new_weights[i][j][k] - self.weights[i][j][k]
            self.weights = new_weights

    def __feed_forward(self, x):
        outputs = [x]
        for weight in self.weights:
            outputs.append(self.__sigmoid(outputs[-1] @ weight))
        del outputs[0]
        return outputs

    def __sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def __backward_prop(self, target, values_layers):
        n_hiddens_out_layers = len(values_layers)
        errors_layers = [None] * n_hiddens_out_layers
        for i in range(n_hiddens_out_layers-1, 0-1, -1):
            errors = []
            if i < n_hiddens_out_layers-1: # (hidden layer)
                for inode, output in enumerate(values_layers[i]):
                    errors.append(self.__hidden_error(output, inode, i, errors_layers))
            else: # i == n_hiddens_out_layers-1 (output layer)
                for output in values_layers[i]:
                    errors.append(self.__output_error(output, target))
            errors_layers[i] = np.array(errors)
        return errors_layers

    def __output_error(self, output, target):
        return output * (1 - output) * (target - output)

    def __hidden_error(self, output, inode, index_layer, errors_layers):
        index_delta = index_layer + 1
        index_weight = index_layer + 1
        sigma = 0
        for i in range(0, len(self.weights[index_weight][inode])):
            sigma += self.weights[index_weight][inode][i] * errors_layers[index_delta][i]
        return output * (1 - output) * sigma

    def __calculate_weight(self, weight, prev_weight, err, val):
        return weight + self.momentum * prev_weight + self.lrate * err * val

    def fit(self, data, target):
```

```
        self.__initialize_weights(data)

        for _ in range(self.max_iter):
            # Random shuffle data and target simultaneously
            p = np.random.permutation(data.shape[0])
            data, target = data[p], target[p]

            # Do gradient descent per batch
            for i in range(0, data.shape[0], self.batch_size):
                index = list(range(i, i+self.batch_size))
                self.__stochastic_gradient_descend(data[index], target[index])

        return self

    def __initialize_weights(self, data):
        # Initialize weights with random numbers
        n_features = data.shape[1]
        if (self.n_hiddens > 0):
            self.weights = [np.random.randn(n_features + 1, self.n_nodes[0])]
            for i in range(1, self.n_hiddens):
                self.weights.append(np.random.randn(self.n_nodes[i-1], self.n_nodes[i]))
            self.weights.append(np.random.randn(self.n_nodes[self.n_hiddens - 1], 1))
        else:
            self.weights = [np.random.randn(n_features + 1, 1)]

        # Assume first prev_weights be zeroes
        self.prev_weights = deepcopy(self.weights)

    def predict(self, data):
        result = [self.__feed_forward(np.append(d, 1.))[-1][0] for d in data]
        return [1 if r >= 0.5 else 0 for r in result]
```

Kode program diatas adalah implementasi *classifier* Feed Forward Neural Network dengan algoritma *gradient descent.* Terdapat fungsi `fit` yang dapat digunakan untuk melakukan *fitting* terhadap data *training,* dan fungsi *predict* yang dapat digunakan untuk melakukan prediksi terhadap data *testing.* Fungsi `fit` menerima parameter berupa data yang akan di-*train* dan *label* dari data yang akan digunakan. Fungsi ini menggunakan *gradient descent* untuk melakukan *update* pada berat di setiap *layer*, *update* dilakukan sebanyak jumlah iterasi maksimal. Fungsi `predict` digunakan untuk memprediksi kelas dari data *training* yang menjadi masukan. Fungsi ini memasukkan data masukan ke dalam jaringan dan mengeluarkan hasilnya.

2. Classifier B - Keras

File: *main.ipynb*

```
from tensorflow.keras import optimizers
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.models import Sequential

class NNKeras():
    def __init__(self, nnodes_per_hidden_layer=[100], lrate=0.05, momentum=0, batch_size=1):
        self.nnodes_per_hidden_layer = nnodes_per_hidden_layer
        self.lrate = lrate
        self.momentum = momentum
        self.batch_size = batch_size

    def fit(self, data, labels, epochs=1):
        """data: ndarray"""
        n_rows = len(data)
        n_attr = len(data[n_rows-1])
        self.model = Sequential()
        # First Hidden Layer
        self.model.add(Dense(units=self.nnodes_per_hidden_layer[0], activation='sigmoid',
input_dim=n_attr))
        # 2nd .. Last Hidden Layer
        for i in range(1, len(self.nnodes_per_hidden_layer)):
            self.model.add(Dense(units=self.nnodes_per_hidden_layer[i], activation='sigmoid'))
        # Output Layer
        self.model.add(Dense(units=1, activation='sigmoid'))

        sgd = optimizers.SGD(lr=self.lrate, momentum=self.momentum)
        self.model.compile(optimizer=sgd, loss='mean_squared_error')
        self.model.fit(data, labels, batch_size=self.batch_size, epochs=epochs, verbose=0)
        return self

    def predict(self, sample):
        return self.model.predict_classes(sample, batch_size=self.batch_size)
```

Kode diatas adalah implementasi dari Keras untuk melakukan *training* data dengan menggunakan fungsi `fit` dan memprediksi kelas dari data baru dengan menggunakan fungsi `predict`. Fungsi `fit` menerima parameter berupa data yang akan di-*train*, *label* dari data yang akan digunakan serta banyaknya *epoch* yang akan dilakukan. Pada fungsi ini, digunakan `models.Sequential` untuk mengkonfigurasi model. *Optimizers* yang digunakan adalah *stochastic gradient descent* dengan parameter berupa *learning rate* dan momentum. Dipilih SGD sebagai *optimizers* agar menyesuaikan dengan *optimizers* yang digunakan pada *classifier* A. Fungsi `predict` digunakan untuk memprediksi kelas dari data yang baru di-*input*.

Sedangkan *dense* digunakan untuk merepresentasikan setiap layer yang ada seperti *hidden layer* dan *output layer*. *Dense* yang terakhir akan dijadikan sebagai *output layer* dan sisanya sebagai *hidden layer*. *Units* yang ada pada *dense* merepresentasikan jumlah *node* yang ada pada setiap *layer*. Fungsi aktivasi yang digunakan adalah *sigmoid*. *Loss* melambangkan nilai dari *error* yang ada pada setiap *epoch*, semakin kecil nilai tersebut maka bobot dari data akan semakin konvergen.

3. Hold-out validation

File: *main.ipynb*

```
# Evaluate with validation data
model_nn = NNClassifier(n_nodes=[20,10], lrate=0.1, momentum=0.1, batch_size=1, max_iter=450).fit(
    train_df.values, train_labels)

print("Test with train data:")
print("  Predicted", model_nn.predict(train_df.values))
print("  Expected", train_labels)
print("Test with validation data:")
print("  Predicted", model_nn.predict(validation_df.values))
print("  Expected", validation_labels)

# Evaluate with validation data
model_keras = NNKeras(nnodes_per_hidden_layer=[20,10], lrate=0.1, momentum=0.1, batch_size=1).fit(
    train_df.values, train_labels, epochs=450)
print("Test with train data:")
print("  Predicted", model_keras.predict(train_df.values).tolist())
print("  Expected", train_labels)
print("Test with validation data:")
print("  Predicted", model_keras.predict(validation_df.values).tolist())
print("  Expected", validation_labels)
```

Kode program diatas adalah *training* yang dilakukan dengan menggunakan algoritma kedua classifier. Untuk melakukan hold-out, validasi dilakukan pada data validasi terlebih dulu. Setelah beberapa percobaan yang dilakukan kelompok kami, diperoleh bahwa jumlah *hidden layer* sebanyak 2 (20 unit dan 10 unit) dengan *learning rate* dan *momentum* bernilai 0.1 dan dengan *epoch* yang besar menghasilkan model yang paling baik.

Hasil Eksekusi

Hasik eksekusi program selengkapnya dapat dilihat pada repositori github melalui pranala https://github.com/tugas-itb-erick/feed-neural-network.

1. Tahap Preprocessing (Encoding & Split dataset)

Train data (80%):

	_overcast	_rainy	_sunny	_windy	z_temperature	z_humidity
8	0	0	1	0	-0.721886	-1.174731
6	1	0	0	1	-1.511449	-1.679217
3	0	1	0	0	-0.563974	1.448595
13	0	1	0	1	-0.406061	0.944109
0	0	0	1	0	1.804715	0.338726
9	0	1	0	0	0.225589	-0.165760
5	0	1	0	1	-1.353537	-1.174731
7	0	0	1	0	-0.248148	1.347697
4	0	1	0	0	-0.879799	-0.165760
1	0	0	1	1	1.015152	0.843212
11	1	0	0	1	-0.248148	0.843212

Validation data (10%):

	_overcast	_rainy	_sunny	_windy	z_temperature	z_humidity
2	1	0	0	0	1.48889	0.439623

Test data (10%):

	_overcast	_rainy	_sunny	_windy	z_temperature	z_humidity
12	1	0	0	0	1.173065	-0.670245
10	0	0	1	1	0.225589	-1.174731

2. Classifier A - Feed Forward Neural Network

Validasi Hold-out dengan data validasi:

```
Test with train data:
  Predicted [[1], [1], [1], [0], [0], [1], [0], [0], [1], [0], [1]]
  Expected [1 1 1 0 0 1 0 0 1 0 1]
Test with validation data:
  Predicted [[1]]
  Expected [1]
```

Evaluasi akhir dengan data uji:

```
Test with train+validation data:
  Predicted [[1], [1], [1], [1], [1], [1], [0], [1], [0], [0], [1], [1]]
  Expected [1 1 1 1 1 1 0 1 0 0 1 1]
Test with test data:
  Predicted [[1], [1]]
  Expected [1 1]
```

3. Classifier B - Keras

Validasi Hold-out dengan data validasi:

```
Test with train data:
  Predicted [[1], [1], [1], [0], [0], [1], [0], [0], [1], [0], [1]]
  Expected [1 1 1 0 0 1 0 0 1 0 1]
Test with validation data:
  Predicted [[1]]
  Expected [1]
```

Evaluasi akhir dengan data uji:

```
Test with train+validation data:
  Predicted [[1], [1], [1], [1], [1], [1], [0], [1], [0], [0], [1], [1]]
  Expected [1 1 1 1 1 1 0 1 0 0 1 1]
Test with test data:
  Predicted [[1], [1]]
  Expected [1 1]
```

Perbandingan Classifier

Classifier A dan *classifier* B menghasilkan hasil prediksi untuk data baru pada kelas yang sama. Hasil validasi *hold-out* dan evaluasi akhir dapat dilihat pada bagian hasil eksekusi (eksperimen). Hal ini dikarenakan, kedua *classifier* menggunakan *optimizers* yang sama yaitu *stochastic gradient descent* dan pada dasarnya kedua *classifier* akan selalu menghasilkan prediksi yang sama. Namun, untuk *classifier* B akan lebih mudah untuk digunakan karena hanya memanggil *library* keras milik Tensorflow. Dari segi waktu yang dibutuhkan saat melakukan eksekusi, *classifier* A masih kurang optimal karena tidak melakukan optimasi dan banyak terdapat operasi yang cukup boros, seperti *copy list* dan sebagainya.

Pembagian Tugas

Dewita Sonya T **13515021**	**Erick Wijaya** **13515057**	**Kezia Suhendra** **13515063**
Classifier A tahap Feed Forward, Debugging	Classifier A tahap Backward Propagation, Preprocessing	Classifier B (Keras)