Laporan Tugas Besar B
IF3270 Pembelajaran Mesin

Implementasi Forward Propagation untuk Feed Forward Neural Network

Disusun oleh:

K01 13520010 - Ken Kalang Al Qalyubi

K01 13520036 - I Gede Arya Raditya Parameswara

K02 13520061 - Gibran Darmawan

K03 13520119 - Marchotridyo

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha No. 10, Bandung 40132

2023

Penjelasan Implementasi

Seluruh implementasi mengenai tugas besar bagian B ini kami letakkan semua di *file* `TubesB_13520036.ipynb`. Dalam *file* ini, terdapat beberapa kelas yang bekerja sama untuk menuntaskan masalah yang diberikan, di antaranya:

1. Kelas FileUtility: Kelas ini bertugas untuk melakukan import data .json (digunakan saat me-*load* model atau *test case* dari asisten.
2. Kelas Layer: Kelas ini merupakan representasi dari suatu *layer* di ANN. Suatu *layer* didefinisikan oleh kumpulan neuronnya, tipenya (*input, hidden, atau output*), serta fungsi aktivasinya (linear, ReLU, sigmoid, atau softmax).
3. Kelas Neuron: Kelas ini merupakan representasi dari unit ANN yaitu neuron. Suatu neuron didefinisikan oleh *layer* tempat ia berada, kumpulan beban yang menuju kepadanya (beban yang menyusun netnya), net dari unit tersebut, *output* dari unit tersebut, dan *error term* dari unit tersebut.
4. Kelas ANNGraph: Kelas ini merupakan intisari dari implementasi kami. Kelas ini memodelkan suatu ANN yang memiliki beberapa layer. Fungsi-fungsi untuk melakukan proses pelatihan melalui *backpropagation* dan melakukan prediksi diimplementasikan pada kelas ini.

Secara umum, *flow* cara kerja program dijelaskan pada poin-poin berikut:

1. Dibuat ANNGraph baru berdasarkan *file* yang di-*load*, contoh melalui *snippet* `graph = ANNGraph("config/linear_small_lr.json")` .
    a. Proses yang terjadi di sini adalah ia akan membaca konten *file* .json tersebut dengan bantuan FileUtility lalu menginstantiasi *layer-layer* dan neuron-neuron sesuai dengan data untuk dimasukkan ke dalam ANNGraph.
2. Dilakukan pelatihan melalui fungsi `__train` .
    a. Pelatihan mengikuti algoritma *backpropagation* umum seperti berikut:
        i. Selama batas iterasi belum tercapai (yaitu *cumulative error* <= *error threshold* atau maksimum iterasi tercapai), dalam setiap iterasi, proses semua input, di mana pada setiap input terjadi:
            1. Untuk pasangan *input* dan target <xi, ti>, lakukan dahulu *forward propagation* untuk mengisi net dan output dari setiap unit ANN.
                a. Ini diimplementasikan di fungsi `__forward_propagation .`
            2. Ketika semua unit sudah selesai, kalkulasi *error* di *output layer*.
            3. *Error* tersebut akan disimpan sebagai *error term* di setiap unit. *Error term* dari unit di *layer* terakhir akan digunakan oleh unit-unit di *layer-layer* sebelumnya untuk menghitung *error term*-nya sendiri.
                a. Ini disebut sebagai bagian *backward propagation*.
                b. Kalkulasi *error term* menggunakan turunan dengan tujuan menemukan gradien dengan nilai terkecil (vektor gradien yang paling menuju arah minimum lokal/global).

    c. Ini diimplementasikan di fungsi `__backward_propagation`.
b. Visualisasi ANNGraph dapat dimunculkan menggunakan fungsi `draw_ann_graph()`.
c. Setelah pelatihan dilakukan, bobot baru dapat ditunjukkan menggunakan fungsi `show_weights()`.
d. Model dapat melakukan prediksi MLP menggunakan fungsi `predict_mlp(input_vector)`, contoh `predict_mlp([1, 2, 3, 4])`.
e. Model dapat disimpan menggunakan fungsi `save(filename)`. Untuk me-*load* model, cukup instantiasi lagi ANNGraph menggunakan *filename* yang diinginkan.

Hasil Eksekusi

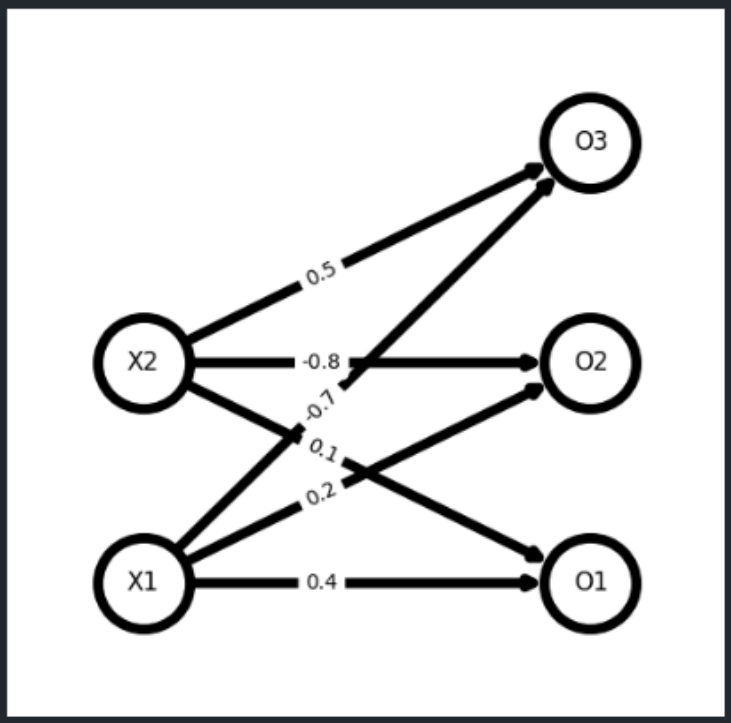
Perbandingan terhadap *test case* yang diberikan oleh asisten

linear_small_lr.json

```
graph = ANNGraph("config/linear_small_lr.json")
graph.draw_ann_graph()
graph.solve()
graph.show_weights()
```

✓ 0.1s

```
Bobot bias untuk O1 = 0.1
Bobot bias untuk O2 = 0.3
Bobot bias untuk O3 = 0.2
```



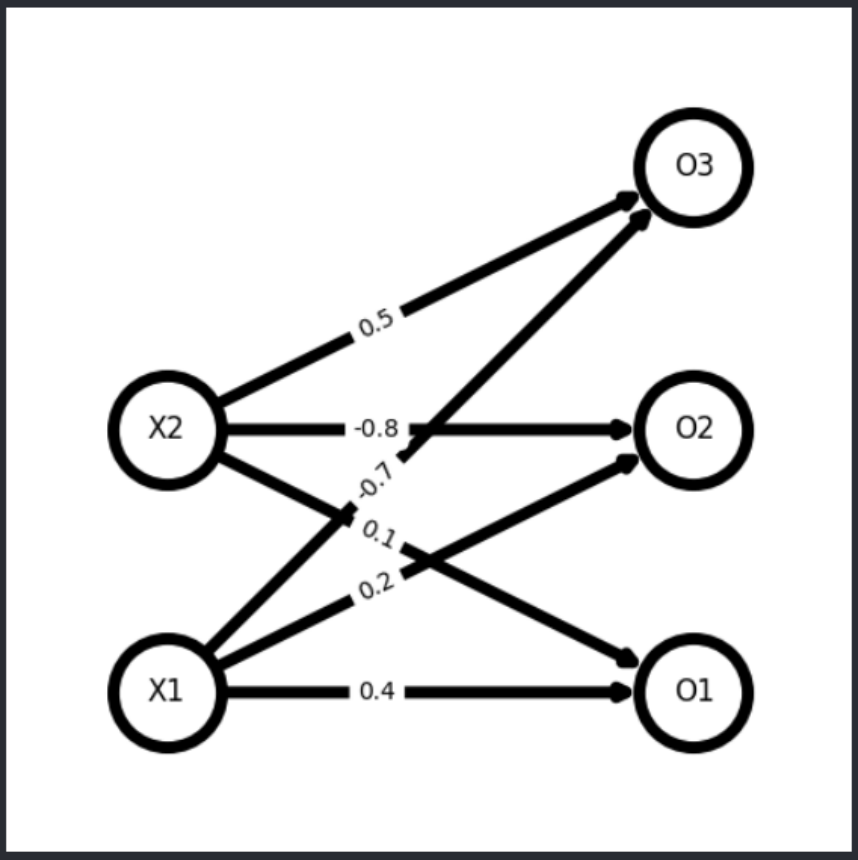

```
Epoch: 1 || Error: 0.665
Max iteration reached!

Trained weights:
[[[0.10120000000000001 0.3006 0.1991]
  [0.40240000000000004 0.201 -0.7018999999999999]
  [0.10180000000000002 -0.799 0.4987]]]

SSE compared to final_weights: 9.600000000000327e-07
```

linear_two_iteration.json

```
graph = ANNGraph("config/linear_two_iteration.json")
graph.draw_ann_graph()
graph.solve()
graph.show_weights()
```

✓ 0.1s

```
Bobot bias untuk O1 = 0.1
Bobot bias untuk O2 = 0.3
Bobot bias untuk O3 = 0.2
```



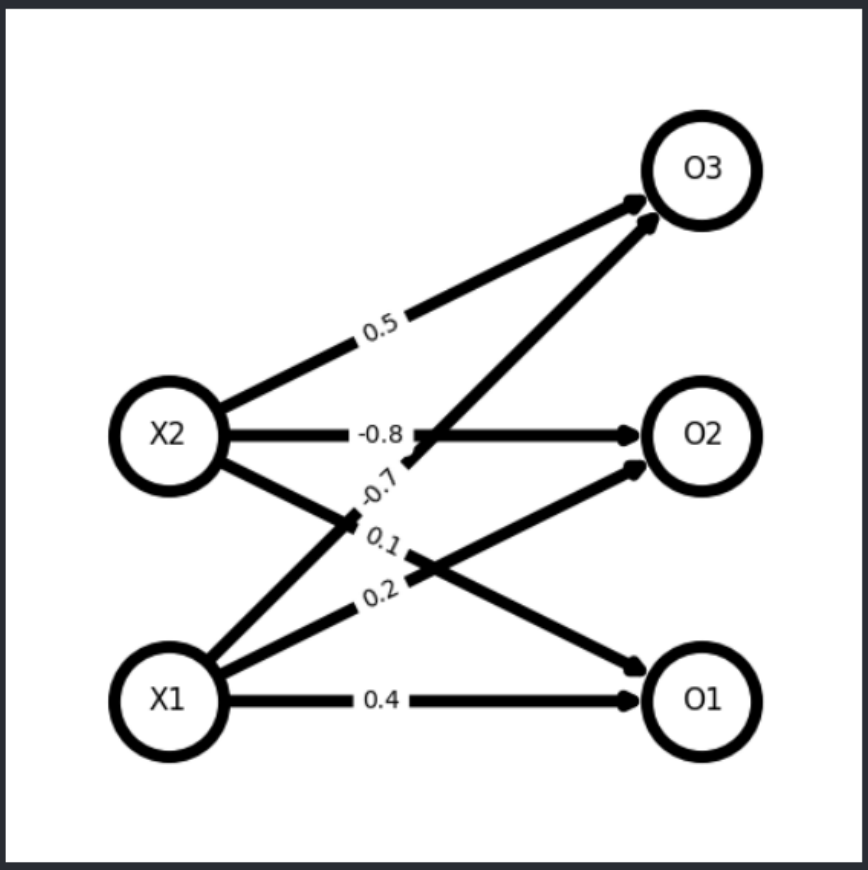

```
Epoch: 1 || Error: 0.665
Epoch: 2 || Error: 0.1818500000000002
Max iteration reached!

Trained weights:
[[[0.166 0.33799999999999997 0.15300000000000008]
  [0.502 0.22599999999999995 -0.7889999999999999]
  [0.21400000000000008 -0.718 0.4270000000000005]]]

SSE compared to final_weights: 3.543711097672514e-32
```

linear.json

```
graph = ANNGraph("config/linear.json")
graph.draw_ann_graph()
graph.solve()
graph.show_weights()
```

0.1s

```
Bobot bias untuk O1 = 0.1
Bobot bias untuk O2 = 0.3
Bobot bias untuk O3 = 0.2
```



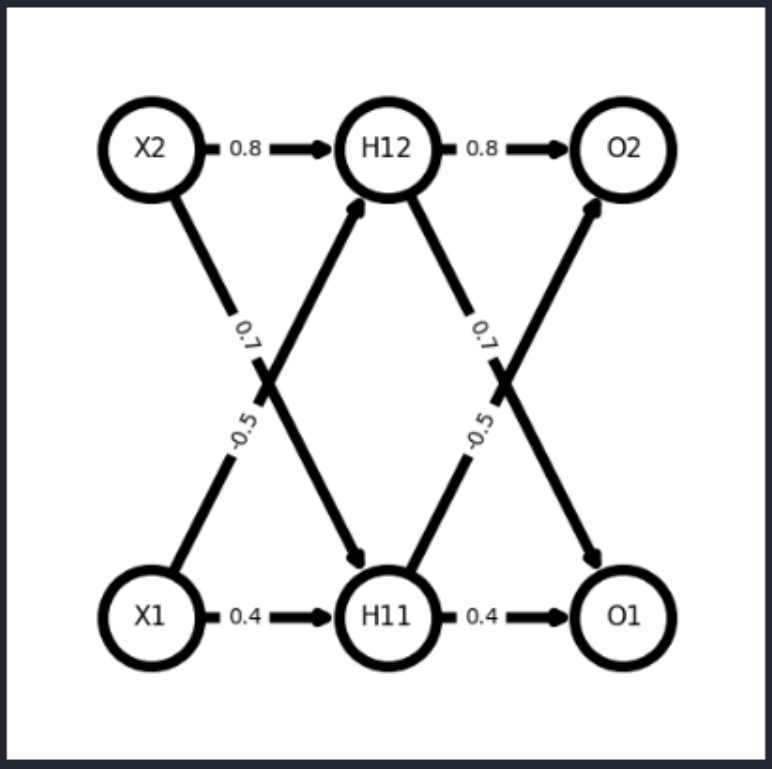

```
Epoch: 1 || Error: 0.665
Max iteration reached!

Trained weights:
[[[0.21999999999999997 0.36 0.10999999999999999]
  [0.64 0.30000000000000004 -0.8900000000000001]
  [0.28 -0.7 0.36999999999999994]]]

SSE compared to final_weights: 1.9451892438311082e-32
```

mlp.json

```
graph = ANNGraph("config/mlp.json")
graph.draw_ann_graph()
graph.solve()
graph.show_weights()
```

✓ 0.1s

```
Bobot bias untuk H11 = 0.1
Bobot bias untuk H12 = 0.2
Bobot bias untuk O1 = 0.1
Bobot bias untuk O2 = 0.2
```



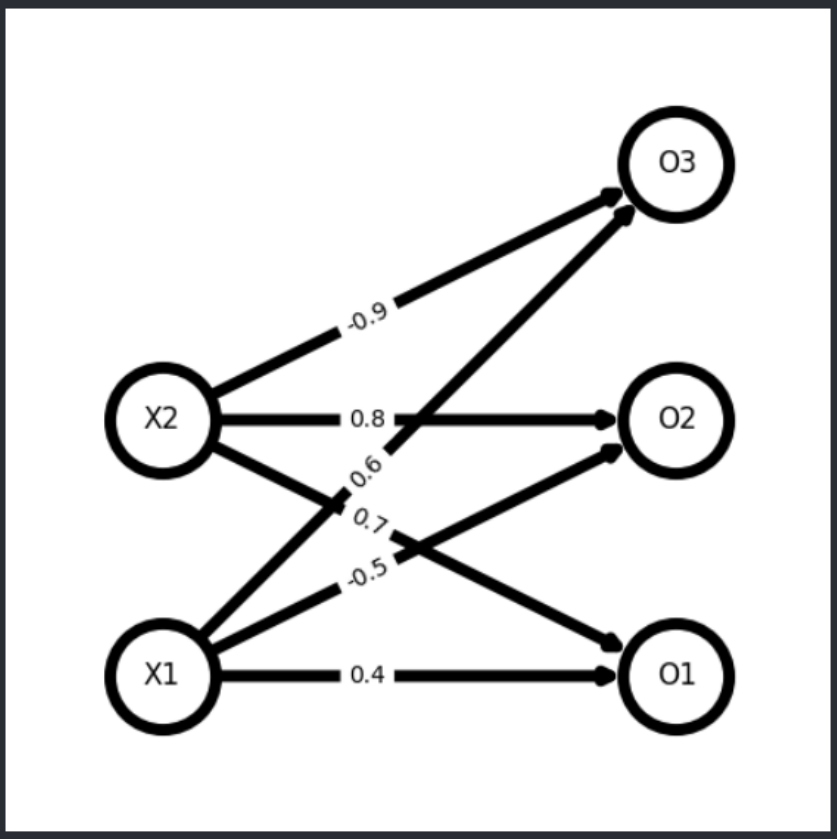

```
Epoch: 1 || Error: 0.8185625000000001
Max iteration reached!

Trained weights:
[[[0.07115 0.1403]
  [0.42885 -0.44029999999999997]
  [0.6855749999999999 0.77015]]

 [[0.02099999999999999 0.1945]
  [0.39605 -0.500275]
  [0.6131 0.79395]]]

SSE compared to final_weights: 0.01901325
```

relu.json

```
graph = ANNGraph("config/relu.json")
graph.draw_ann_graph()
graph.solve()
graph.show_weights()
```

✓ 0.1s

```
Bobot bias untuk O1 = 0.1
Bobot bias untuk O2 = 0.2
Bobot bias untuk O3 = 0.3
```



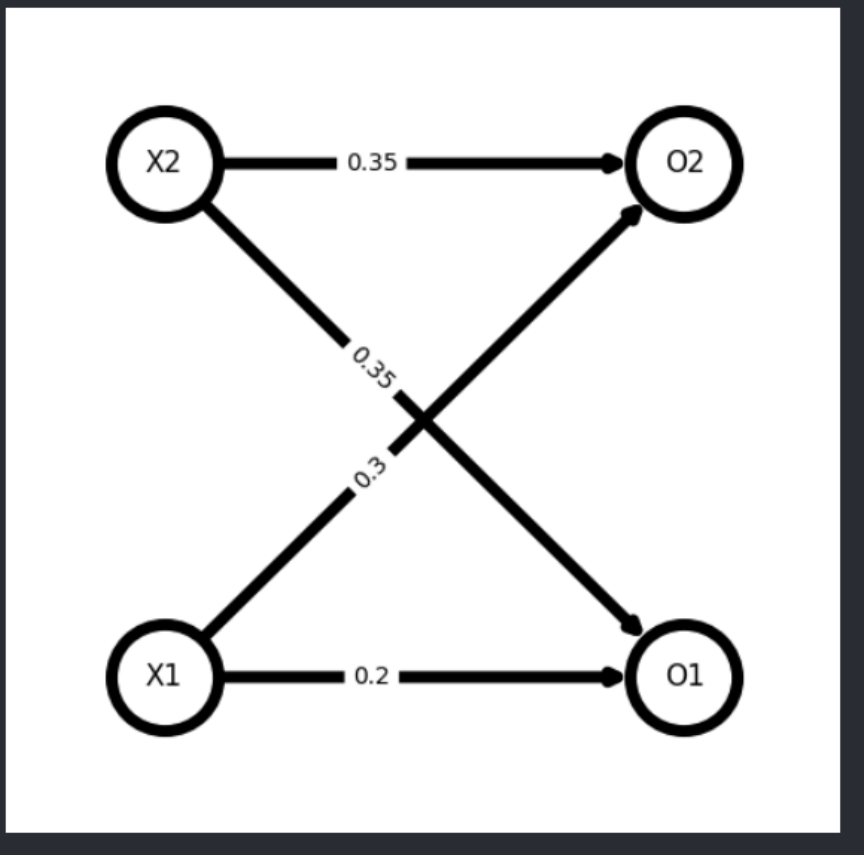

```
Epoch: 1 || Error: 0.14625000000000002
Max iteration reached!

Trained weights:
[[[0.10500000000000001 0.19 0.25]
  [0.395 -0.49 0.575]
  [0.7025 0.795 -0.85]]]

SSE compared to final_weights: 1.925929944387236e-34
```

sigmoid_mini_batch_GD.json

```
graph = ANNGraph("config/sigmoid_mini_batch_GD.json")
graph.draw_ann_graph()
graph.solve()
graph.show_weights()
```

✓ 0.9s

```
Bobot bias untuk O1 = 0.15
Bobot bias untuk O2 = 0.25
```



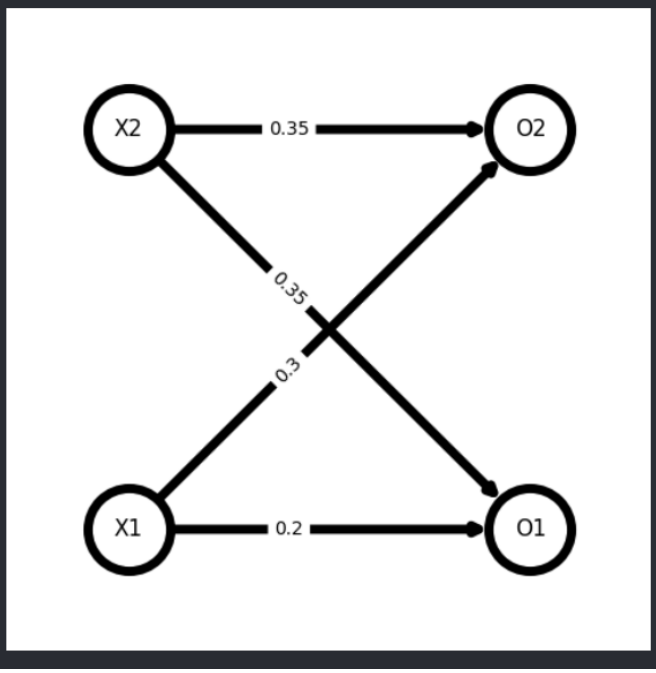

```
 9998    Epoch: 9998 || Error: 0.38104477234246903
 9999    Epoch: 9999 || Error: 0.38103878041654066
10000    Epoch: 10000 || Error: 0.38103278894838283
10001    Max iteration reached!
10002
10003    Trained weights:
10004    [[[0.45660671737000963 0.6519113877939242]
10005      [1.5606349591530178 3.2007363438706724]
10006      [-4.035800724602569 -13.032836766232941]]
```

sigmoid_stochastic_GD.json

```
graph = ANNGraph("config/sigmoid_stochastic_GD.json")
graph.draw_ann_graph()
graph.solve()
graph.show_weights()
```

✓ 0.8s

```
Bobot bias untuk O1 = 0.15
Bobot bias untuk O2 = 0.25
```



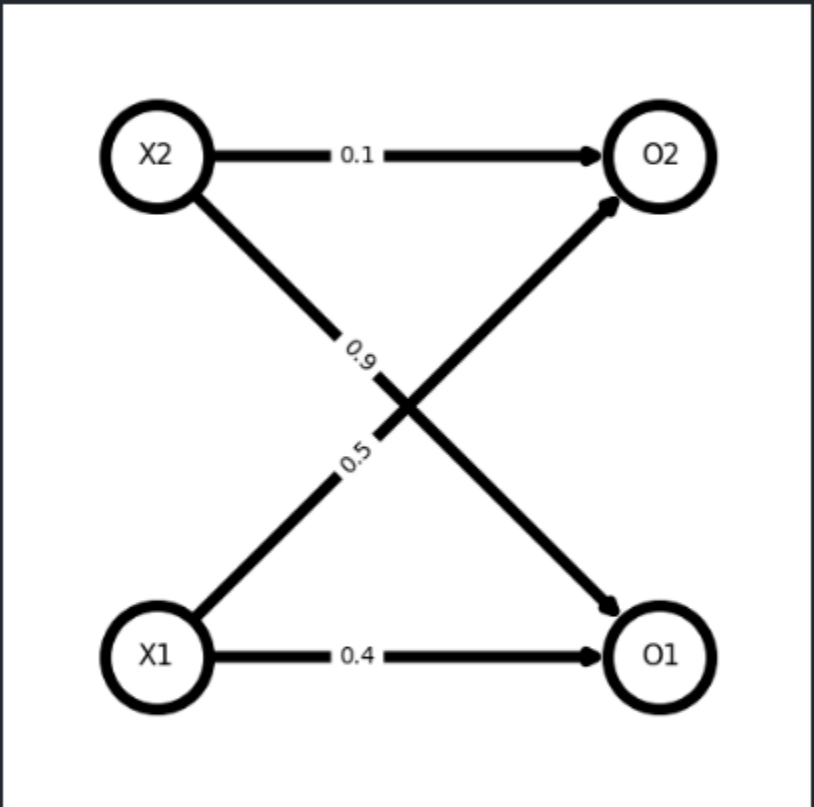

```
 9998    Epoch: 9998 || Error: 0.38104477234246903
 9999    Epoch: 9999 || Error: 0.38103878041654066
10000    Epoch: 10000 || Error: 0.38103278894838283
10001    Max iteration reached!
10002
10003    Trained weights:
10004    [[[0.45660671737000963 0.6519113877939242]
10005      [1.5606349591530178 3.2007363438706724]
10006      [-4.035800724602569 -13.03283676623294]]]
```

sigmoid.json

```
graph = ANNGraph("config/sigmoid.json")
graph.draw_ann_graph()
graph.solve()
graph.show_weights()
```

✓ 0.1s

```
Bobot bias untuk O1 = 0.1
Bobot bias untuk O2 = 0.2
```



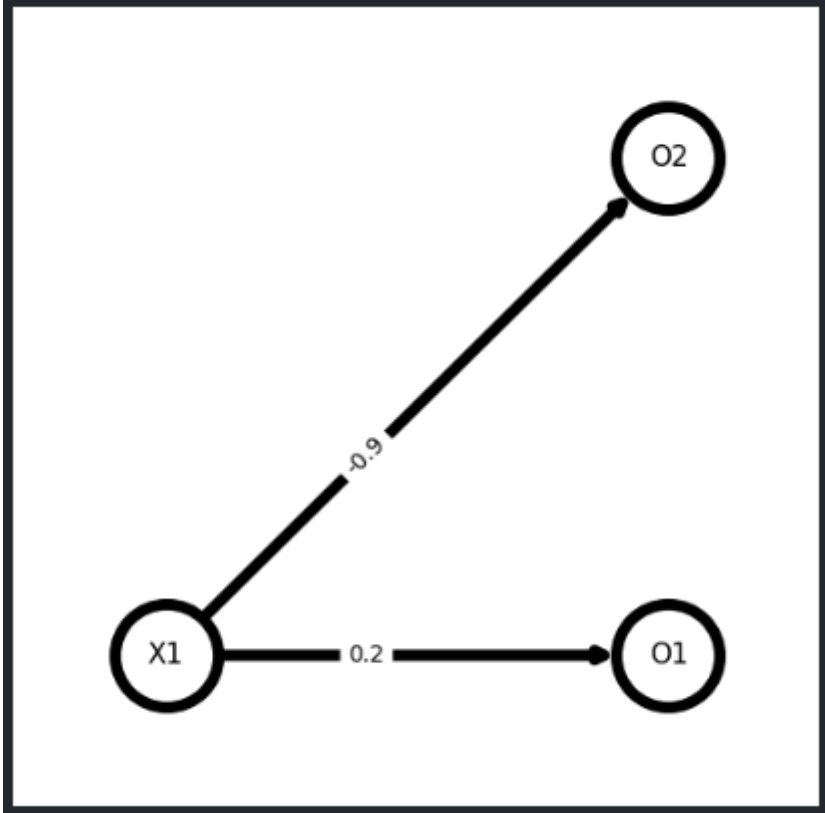

```
Epoch: 1 || Error: 0.4465929462986903
Max iteration reached!

Trained weights:
[[[0.10061658075420078 0.1974857748026805]
  [0.4 0.5]
  [0.901121454344799 0.09863434201890192]]]

SSE compared to final_weights: 8.863973303333145
```

softmax_error_only.json

```
graph = ANNGraph("config/softmax_error_only.json")
graph.draw_ann_graph()
graph.solve()
graph.show_weights()
```

✓ 0.1s

```
Bobot bias untuk O1 = 0.4
Bobot bias untuk O2 = 0.7
```



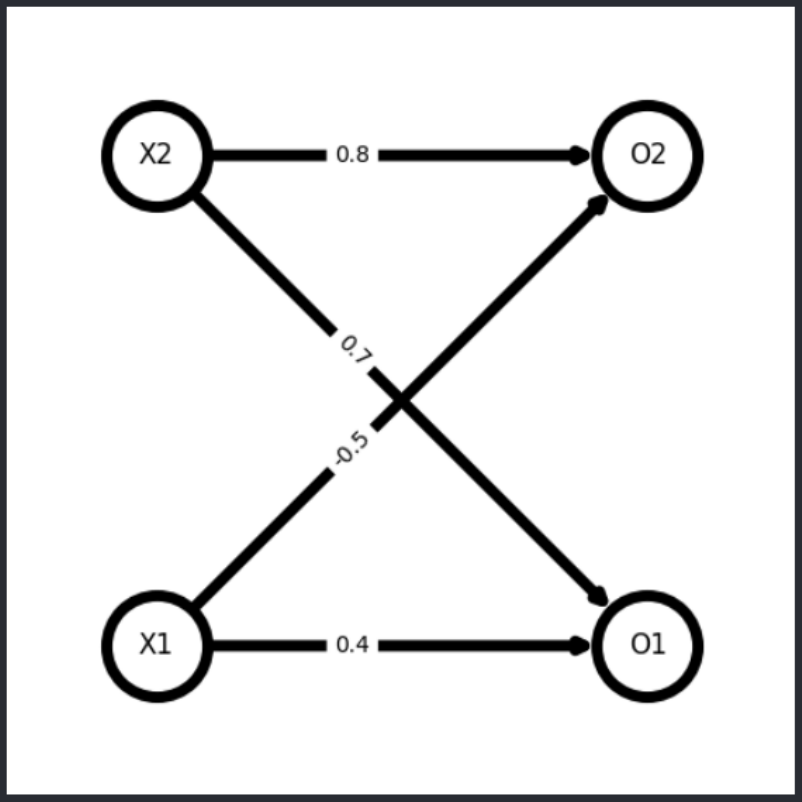

```
Epoch: 1 || Error: 0.22041740991845085
Error ⩽ error threshold!

Trained weights:
[[[0.3802183888558582 0.7197816111441417]
  [0.21978161114414185 -0.9197816111441418]]]

SSE compared to final_weights: 0.0015652485578321424
```

softmax.json

```
graph = ANNGraph("config/softmax.json")
graph.draw_ann_graph()
graph.solve()
graph.show_weights()
```

✓ 0.1s

```
Bobot bias untuk O1 = 0.1
Bobot bias untuk O2 = 0.2
```



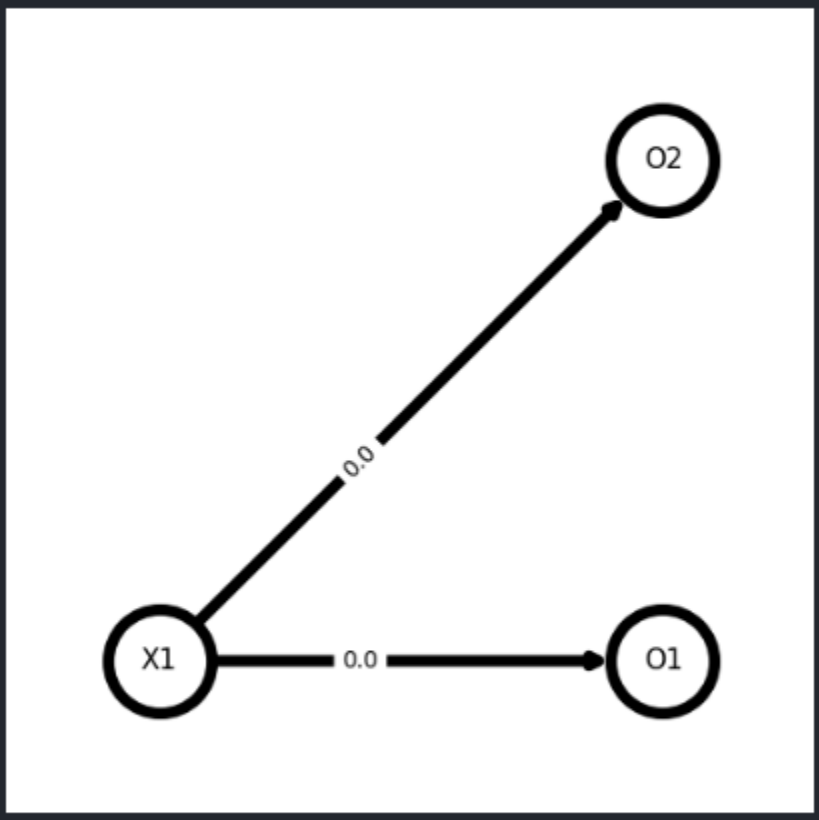

```
Epoch: 1 || Error: 2.2933074256150983
Max iteration reached!

Trained weights:
[[[0.11301356652329322 0.18698643347670682]
  [0.29539054838432355 -0.3953905483843235]
  [0.7981026683540299 0.7018973316459701]]]

SSE compared to final_weights: 3.4814236410918116e-17
```

sse_only.json

```
graph = ANNGraph("config/sse_only.json")
graph.draw_ann_graph()
graph.solve()
graph.show_weights()
```

✓ 0.1s

```
Bobot bias untuk O1 = 0.5
Bobot bias untuk O2 = 0.5
```




```
Epoch: 1 || Error: 0.25
Error ⩽ error threshold!

Trained weights:
[[[0.495 0.505]
  [0.0 0.0]]]

SSE compared to final_weights: 5.000000000000009e-05
```

Pelatihan terhadap dataset iris

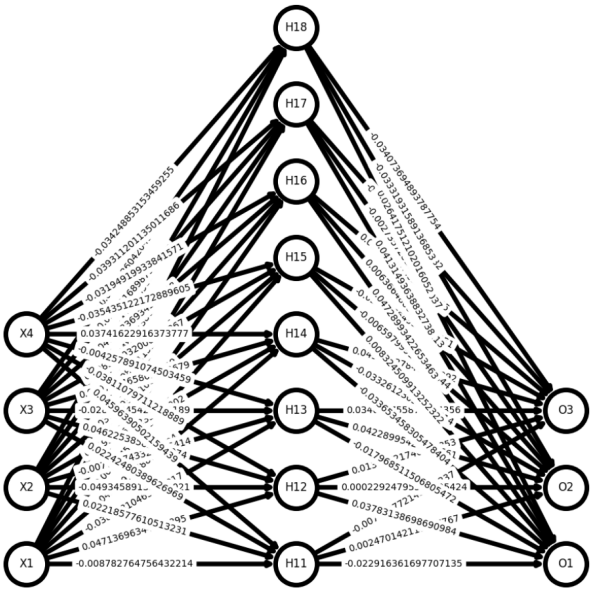
Dataset iris kami ubah dulu ke bentuk `.json` dengan format yang sama dengan *test case* dari asisten pada fungsi `generate_iris_to_train()` .

Struktur ANN yang kami gunakan:

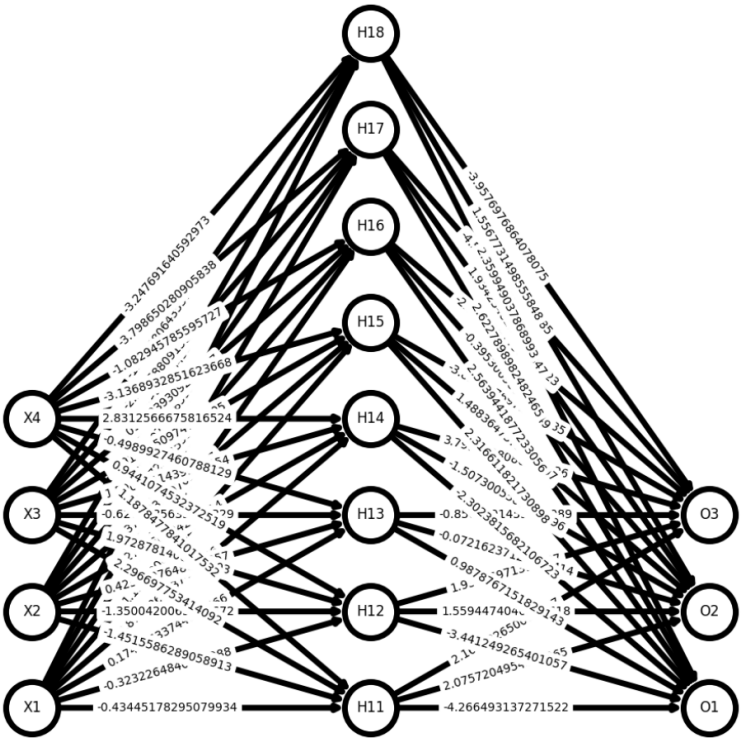
1. Terdapat *input layer* dengan 4 buah neuron. Hal ini sesuai dengan 4 *input* yang ada di *dataset* iris.
2. Terdapat *output layer* dengan 3 buah neuron. Fungsi aktivasinya adalah *softmax* karena kami menginginkan membuat ANN yang kami bertindak sebagai *multiclass classifier*. Kami menggunakan *one-hot encoding* untuk mengonversi ketiga target yang dimiliki oleh dataset iris: `[1, 0, 0]` untuk setosa, `[0, 1, 0]` untuk versicolor, dan `[0, 0, 1]` untuk virginica.
3. Terdapat satu buah *hidden layer* dengan 8 buah neuron. Fungsi aktivasi yang kami gunakan adalah fungsi sigmoid.
4. *Learning parameter* yang kami gunakan:
    a. *Learning rate*: 5 x 10^{-3}
    b. *Batch size*: 2
    c. *Max iteration*: 2000
    d. *Error threshold*: 0.1

Awalnya, seluruh weight kami inisialisasi secara *random* sebagaimana yang disebutkan pada algoritma buku Tom Mitchell, yaitu di interval [-0.05, 0.05].

Berikut adalah tampilan model sebelum dilatih:



dan berikut adalah tampilan model setelah dilatih (*weights*-nya berbeda):



untuk weights lengkapnya bisa dilihat di hasil perbandingan dengan sklearn pada bagian selanjutnya.

Perbandingan dengan MLP sklearn

Model yang kami latih terhadap *dataset iris* menggunakan dibandingkan dengan model MLP sklearn yang dilatih terhadap *dataset* yang sama. Hasil pelatihan sebelumnya kami simpan dalam `trained/iris.json` lalu kami *load* dalam variabel `iris_graph` .

Untuk MLP classifier dari sklearn, kami menggunakan variabel bernama `mlp` .

Buat MLPClassifier pada dataset iris

```
from sklearn.neural_network import MLPClassifier

iris = datasets.load_iris()

X = iris.data
y = iris.target

mlp = MLPClassifier(hidden_layer_sizes=(8,), activation='logistic', solver='adam', max_iter=20000, learning_rate_init=5e-3)

mlp.fit(X, y)
```

Model sengaja kami buat memiliki struktur ANN yang sama agar bisa dibandingkan *weights* hasil pelatihannya juga. Berikut adalah *weights* dari model yang kami latih:

```
=== Trained model without sklearn ===


Trained weights:
[array([[-0.26961330416389895, -0.2616319528720129, 0.22095467146265482,
         -2.071667133299912, 2.2331815914138344, 0.528983924009489,
         2.722170136348069, 2.3313057286432195],
        [-0.43445178295079934, -0.3232264840763388, 0.17444633744808466,
         -0.8789369037428446, 1.1532800486223658, 0.40926377830641403,
         1.4378763821886906, 1.2050881402424252],
        [-1.4515586289058913, -1.3500420063618672, 0.4254497648091857,
         -1.276760184281314, 1.3727336507014047, 0.8604725075991804,
         1.5631078368592324, 1.3884565686452621],
        [2.296697753414092, 1.9728781406524103, -0.6275485633465329,
         1.5904111439996254, -1.9111350974708905, -1.0923239309222117,
         -2.270688091512734, -1.9700806435374743],
        [1.1878477841017532, 0.9441074532372519, -0.4989927460788129,
         2.8312566675816524, -3.1368932851623668, -1.082945785595727,
         -3.798650280905838, -3.247691640592973]], dtype=object)
 array([[0.12080130062108951, -0.7290272752453892, 0.7022066050338848],
        [-4.266493137271522, 2.0757204954854265, 2.163232650060837],
        [-3.441249265401057, 1.5594474046301618, 1.93286971312187],
        [0.9878767151829143, -0.07216237124522414, -0.8567591453591089],
        [-2.3023815682106723, -1.5073005507323096, 3.7918018086026226],
        [2.316611821730898, 1.4883647375654097, -3.8303476979068485],
        [2.5639441877233056, -0.395306986069549, -2.1057961755300023],
        [2.6227898982482465, 1.9342348216983847, -4.5500208831724285],
        [2.359949037868993, 1.5567731498555848, -3.9576976864078075]],
       dtype=object)                                                    ]
```

dan berikut adalah *weights* dari model yang dilatih menggunakan sklearn.

```
=== Trained model using sklearn ===

[[ 0.71404769  0.88189926  0.31037945  0.11469911  0.93227654  0.16759878
   0.21829645 -1.33322973]
 [ 0.58252856  1.2770512  -1.39841174 -1.28957514  0.79950029  0.93262692
  -1.62465901 -1.31914035]
 [ 0.89940934 -1.60646545  0.77072427  1.08426916 -1.48086977 -1.22370044
   1.21881876  2.2737151 ]
 [-0.06017165 -2.6148003   1.38235149  1.09794305 -1.58318292 -1.17280788
   1.67702502  2.15821883]]
[[ 1.19664762 -0.59337455 -0.29088539]
 [ 1.7568838   2.30843923 -2.29453791]
 [-1.44281096  1.55996889  1.86481706]
 [-1.63712923  1.45975375  1.72134197]
 [ 1.3756749   1.11063461 -1.72729666]
 [ 1.21995355 -1.08486123 -1.50952357]
 [-1.6332828   1.63372432  0.91605542]
 [-1.23729873 -2.27178918  2.28190122]]
```

Kami melakukan uji coba untuk memprediksi dua *input*, yaitu `[5.1, 3.5, 1.4, 0.2]` dan `[6.2, 2.8, 4.8, 1.8]` untuk kedua model. Berikut adalah prediksi dari model yang kami buat (model yang kami buat disimpan pada variabel `iris_graph`):

```python
to_predict = [5.1, 3.5, 1.4, 0.2]
print(f"\n>>> Percobaan untuk memprediksi {to_predict}")
predicted = iris_graph.predict_mlp(to_predict)

if predicted == 0:
    print("Setosa")
elif predicted == 1:
    print("Versicolor")
else:
    print("Virginica")

to_predict = [6.2, 2.8, 4.8, 1.8]
print(f"\n>>> Percobaan untuk memprediksi {to_predict}")
predicted = iris_graph.predict_mlp(to_predict)

if predicted == 0:
    print("Setosa")
elif predicted == 1:
    print("Versicolor")
else:
    print("Virginica")
```

✓ 0.2s

```
>>> Percobaan untuk memprediksi [5.1, 3.5, 1.4, 0.2]
Setosa

>>> Percobaan untuk memprediksi [6.2, 2.8, 4.8, 1.8]
Virginica
```

dan berikut adalah prediksi yang kami buat menggunakan model dari MLPClassifier dari sklearn:

```python
to_predict = [5.1, 3.5, 1.4, 0.2]
print(f"\n>>> Percobaan untuk memprediksi {to_predict}")
predicted = mlp.predict([to_predict])

if predicted == 0:
    print("Setosa")
elif predicted == 1:
    print("Versicolor")
else:
    print("Virginica")

to_predict = [6.2, 2.8, 4.8, 1.8]
print(f"\n>>> Percobaan untuk memprediksi {to_predict}")
predicted = mlp.predict([to_predict])

if predicted == 0:
    print("Setosa")
elif predicted == 1:
    print("Versicolor")
else:
    print("Virginica")
```

✓ 0.2s

```
>>> Percobaan untuk memprediksi [5.1, 3.5, 1.4, 0.2]
Setosa

>>> Percobaan untuk memprediksi [6.2, 2.8, 4.8, 1.8]
Virginica
```

dapat dilihat bahwa walaupun kedua model memiliki *weight* yang berbeda hasilnya, kedua model, baik yang dibuat kami maupun yang dibuat oleh sklearn.MLPClassifier, memiliki klasifikasi yang sama untuk uji coba yang kami berikan.

Pembagian Tugas

NIM	Nama	Bagian Tugas
13520010	Ken Kalang Al Qayubi	Inisialisasi repository, *set up* dari tugas A. Mencoba membuat fungsi *backpropagation* bersama I Gede.
13520036	I Gede Arya Raditya Parameswara	Inisialisasi repository, *set up* dari tugas A. Mencoba membuat fungsi *backpropagation* bersama Ken Kalang.
13520061	Gibran Darmawan	Inisialisasi dokumen, melakukan pengujian terhadap *test case* yang diberikan asisten.
13520119	Marchotridyo	Melanjutkan pekerjaan Ken Kalang dan I Gede: membenarkan fungsi *backpropagation*, mengatur fungsi *load* agar sesuai dengan *test case* asisten, melakukan pelatihan terhadap *dataset* iris, melakukan pengujian terhadap *test case* asisten, mengerjakan dokumen.

Daftar Pustaka

Slide perkuliahan backpropagation

Mitchell, Tom. 1997. *Machine Learning*. Diunduh dari pranala berikut: https://www.cin.ufpe.br/~cavmj/Machine - Learning - Tom Mitchell.pdf

Spesifikasi Tugas Besar IF3270