# LAPORAN TUGAS BESAR 1C IF3270 PEMBELAJARAN MESIN

# IMPLEMENTASI *MINI-BATCH GRADIENT DESCENT*


Disusun Oleh:

| | |
|---|---|
| Kevin Angelo | 13517086 |
| Juro Sutantra | 13517113 |
| Nurul Utami Amaliah W. | 13517132 |
| Tasya Lailinissa Diandraputri | 13517141 |


**INSTITUT TEKNOLOGI BANDUNG**

**SEMESTER II TAHUN 2019/2020**

## 1. Penjelasan Implementasi

Artificial Neural Network yang dikembangkan menggunakan fungsi aktivasi sigmoid, serta data iris yang telah di-*batch*. Ukuran *batch* yang dipakai ditentukan pada program, sehingga *update weight* tidak perlu menunggu satu *epoch*. Struktur data yang digunakan untuk menampung *weight* antara tiap *node* pada tiap *layer* adalah *dictionary*, dimana *key* (a, b, c) dan *value* v, dengan a = indeks *layer* pada neural network, b = indeks *node* pada layer a, c = indeks *node* pada layer a+1, dan v = *weight* dari hubungan antar *node* tersebut.

*Backpropagation* dilakukan pada setiap *batch*. Proses pertama pada *backpropagation* adalah *feed forward*. *Feed forward* dilakukan dengan memproses *record* pada *batch*. Output setiap *node* disimpan pada matriks. Net pada setiap node, dihitung menggunakan rumus berikut.

$$net = \sum_{i=0}^{n} w_i x_i$$

Lalu, hasil net tersebut dimasukkan ke fungsi aktivasi yang merupakan fungsi sigmoid dengan rumus sebagai berikut.

$$o = \sigma(net) = \frac{1}{1 + e^{-net}}$$

Proses tersebut dilakukan pada setiap *hidden layer* sampai mencapai *output layer*. Matriks yang berisi output dari setiap node akan diproses pada *backward phase*.

Pada proses *backward phase*, dilakukan juga dekomposisi menjadi beberapa tahap yaitu pembuatan matriks delta, pengubahan delta *weight* untuk *hidden layer*, pengubahan delta *weight* untuk *input layer*, dan tahap terakhir adalah pengubahan delta *weight bias*.

Pada tahap pembuatan matriks delta akan dilakukan penghitungan delta pada setiap *node hidden layer* dan *output layer*. Hasil delta akan dicatat

dengan format *layer* sebagai baris dan *node* sebagai kolom pada matriks yang merupakan hasil tahap ini. Penghitungan delta menggunakan rumus sebagai berikut,

- Untuk setiap output node *k*.

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

- Untuk setiap hidden node *h*.

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k$$

Tahap pengubahan delta *weight* pada *hidden layer* merupakan sebuah pengulangan biasa yang mengiterasi pada setiap *hidden layer* yang dimiliki. Pada setiap pengulangan akan dilakukan pencarian *edge* milik semua *node* pada *layer* terkait dan melakukan penghitungan delta *weight*-nya dengan rumus sebagai berikut.

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

Setelah hasil perhitungan didapatkan, hasilnya akan ditambahkan pada delta *weight* yang sebelumnya agar delta *weight* terkumpul untuk setiap *record* dan siap untuk diproses pada tahap *update weight.*

Tahap pengubahan delta *weight* pada *input layer* sebenarnya hampir sama dengan tahap sebelumnya, hanya saja penghitungannya menggunakan rumus sebagai berikut.

$$\Delta w_{ji} = \eta \, \delta_j \, x_{ji}$$

Tahap terakhir pada *backward phase* adalah pengubahan delta *weight* untuk bias. Khusus untuk bias, iterasi dilakukan sesuai dengan jumlah bias yang ada dan prosesnya sama dengan proses pengubahan delta *weight* pada

*input layer*, hanya saja $x_{ji}$ nya diubah menjadi 1 karena nilai bias selalu sama yaitu satu. Setelah *backward phase* selesai, *update weight* dilakukan jika error belum 0. Jika error sudah mencapai 0 atau jumlah iterasi sudah mencapai maksimal iterasi, proses *training* berhenti dan graf ditampilkan.

## 2. Hasil Eksekusi

1.) Test Case 1

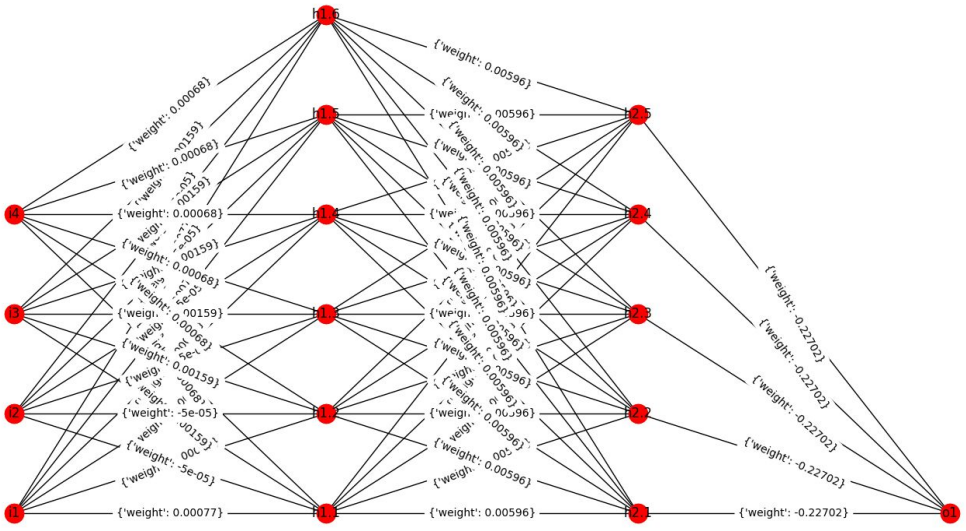
Maksimal iterasi: 100

*Learning rate*: 0,1

Jumlah *hidden layer*: 2

Jumlah *node* tiap *layer*: 6, 5

```
Tasyas-MacBook-Air:gradientDescent Tasya$ python myMLP.py
Jumlah node tiap layer, dipisahkan dengan space: 6 5
```




2.) Test Case 2

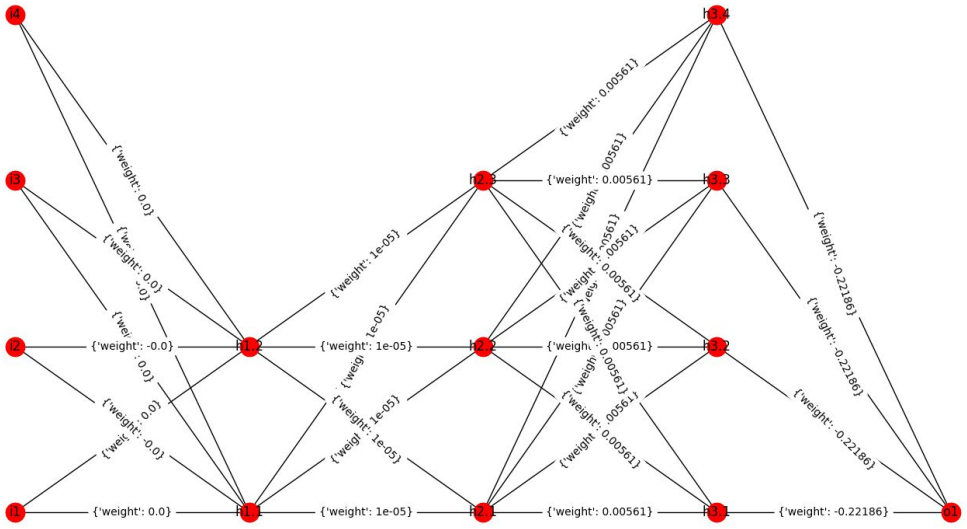
Maksimal iterasi: 100

*Learning rate*: 0,1

Jumlah *hidden layer*: 3

Jumlah *node* tiap *layer*: 2, 3, 4

```
Tasyas-MacBook-Air:gradientDescent Tasya$ python myMLP.py
Jumlah node tiap layer, dipisahkan dengan space: 2 3 4
```



3.) Test Case 3

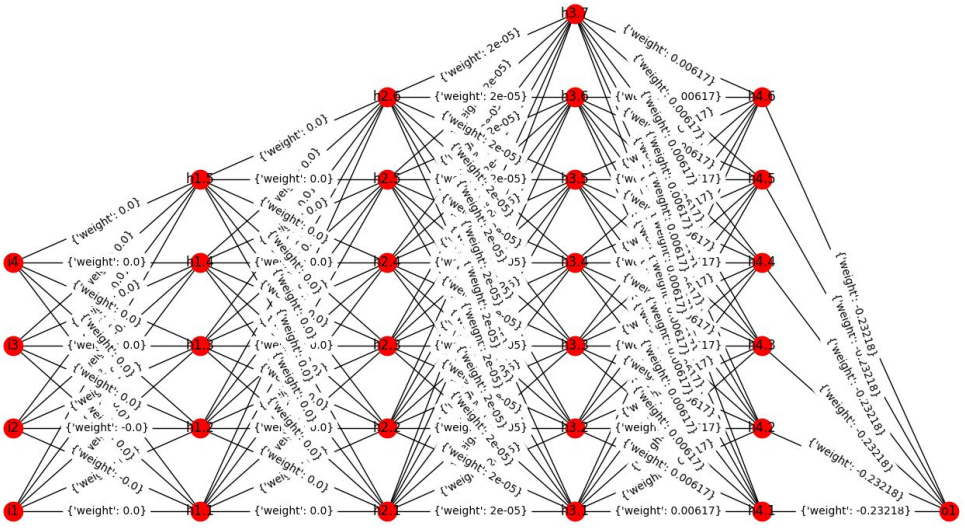
Maksimal iterasi: 100

*Learning rate*: 0,1

Jumlah *hidden layer*: 4

Jumlah *node* tiap *layer*: 5, 6, 7, 6

```
Tasyas-MacBook-Air:gradientDescent Tasya$ python myMLP.py
Jumlah node tiap layer, dipisahkan dengan space: 5 6 7 6
```



## 3. Perbandingan dengan Hasil MLP Sklearn

MLP sklearn menggunakan *stochastic gradient descent* yang meng-*update* bobot per *record*. Fungsi aktivasi yang digunakan di MLP sklearn Agar dapat dibandingkan dengan myMLP, jumlah *record* per *batch* diatur menjadi satu *record* per *batch*. Fungsi aktivasi yang digunakan juga sama yaitu fungsi sigmoid.

Perbedaan MLP sklearn dengan myMLP adalah jumlah node pada output layer. Pada output layer MLP sklearn, terdapat tiga node, sesuai dengan jumlah kemungkinan target, sedangkan pada output layer myMLP, hanya terdapat satu node. Cara klasifikasi MLP sklearn adalah memilih *output node* dengan nilai terbesar.

Dengan test case 3, error yang dihasilkan pada akhir training myMLP sebesar 50.035458512834595. Pada akhir training MLP sklearn, error yang dihasilkan adalah sebesar 1,1042522000055357.

## 4. Pembagian Tugas

| NIM | Tugas |
|---|---|
| 13517086 | Main program, batch breakdown, sklMLP, weight data structure |
| 13517113 | Backward phase |
| 13517132 | Error, update weight, print model |
| 13517141 | Feed forward |