Penerapan Greedy dalam Pembuatan ANN

Stefanus Stanley Yoga Setiawan/13518122
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13518122@std.stei.itb.ac.id

***Abstract*—*Artificial Neural Network* atau biasa disingkat sebagai ANN adalah salah satu metode *Deep Learning* yang biasa digunakan Data Scientist untuk melakukan klasifikasi dan analisis data.**

Keywords*—ANN, *Deep Learning*, *Artificial Intelligence*, Klasifikasi, *Machine Learning

I. Pendahuluan

Salah satu bidang yang mempelajari tentang kecerdasan buatan adalah *Machine Learning*. Dalam *Machine Learning* sendiri terdapat sebuah bidang lagi yang berfokus terhadap algoritma yang memiliki struktur dan fungsi seperti otak manusia, bidang ini adalah *Deep Learning*. Beberapa ahli menerapkan *Deep Learning* ini dengan tujuan untuk membuat *Learning Algorithm* yang lebih bagus dan lebih mudah untuk dipakai dan menciptakan revolusi yang menunjang terhadap *Artificial Intelligence*. Kenapa *Deep Learning* dapat menciptakan revolusi dalam bidang AI? Hal ini bisa kita bandingkan dengan Algoritma Pembelajaran biasa di mana peningkatan performa *Deep Learning* akan terus meningkat seiring dengan banyaknya data yang ada, hal ini berbeda dengan Algoritma Pembelajaran biasa, di mana performanya akan meningkat, tetapi akan berhenti pada suatu titik di mana data tersebut sudah dibilang cukup.

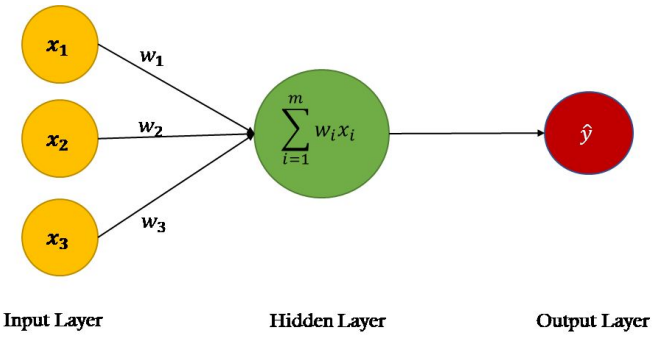
Salah satu bagian dari *Deep Learning* yang paling dasar adalah *Artificial Neural Network* atau biasa disingkat ANN. ANN terdiri dari *input layer*, beberapa *hidden layer,* dan akhirnya *output layer*. Tiap *layer* memiliki beberapa *nodes* yang tiap-tiap *nodes* nya memiliki *weights* nya sendiri-sendiri. Cara perhitungan *weights* tiap nodes dapat menggunakan beberapa metode.

II. Dasar Teori

A. *Artifical Neural Network (ANN)*

ANN adalah salah satu *Supervised Learning Learning Algorithm* di mana kita menyediakan input berupa data dan label output data tersebut untuk dipelajari oleh model. ANN melakukan proses klasifikasi dengan memberikan prediksi acak dahulu, kemudian prediksi ini akan dibandingkan dengan output sebenarnya. Fungsi untuk mencari perbedaan hasil prediksi dengan hasil aslinya disebut sebagai *Cost Function*. Secara harfiah, semakin kecil *Cost* dari suatu model, maka semakin bagus model tersebut. ANN sendiri dibagi menjadi dua fase, yaitu *Feedforward* dan *Backpropagation*.

Pada *Feedforward Phase,* prediksi dilakukan dengan melakukan *dot product weight* dengan *value* masukan kemudian ditambahkan dengan bias.



Gambar 1 *Feed Forward Phase*

(Sumber:https://towardsdatascience.com/single-neuron-training-3fc7f84d67d)

Setelah berhasil menghitung hasil *dot product,* kita masukkan hasilnya ke dalam *activation function* yang kita gunakan. Ada banyak sekali *activation function* yang ada, pada kasus klasifikasi sering digunakan salah satu dari *sigmoid* atau *softmax*. *Sigmoid* biasa digunakan untuk klasifikasi biner (*True* atau *False*), sedangkan *softmax* digunakan untuk klasifikasi dengan lebih banyak label.

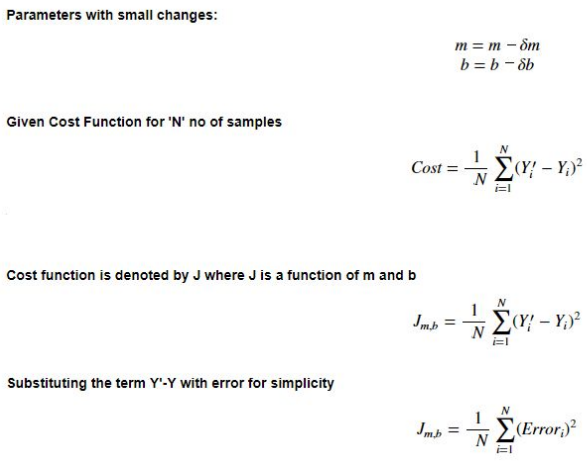
Pada *Backpropagation Phase,* ada 2 langkah yang harus dilakukan, yaitu menghitung *cost* dan meminimalisir *cost*. Ada beberapa *cost function* yang dapat digunakan, antara lain MSE *(Mean Squared Error)*, *Quadratic Cost, Cross-entropy Cost, Exponential Cost,* dll. Setelah menghitung *cost,* kita melakukan algoritma untuk meminimalisir *cost* nya dengan *gradient descent.*

B. *Gradient Descent*

Gradient Descent adalah salah satu algoritma untuk mencari titik terendah/tertinggi dari suatu kurva (nilai maksimum/minimum). Algoritma ini bisa dibayangkan dengan suatu titik pada kurva kemudian ingin berpindah ke titik minimum kurva. Langkah yang mungkin adalah berpindah ke atas atau ke bawah, setelah menentukan langkahnya, langkah tersebut bisa besar atau kecil. Secara singkat ada dua hal

yang harus diketahui untuk mencapai titik minimum, yaitu arah mana dan berapa besar langkah yang harus dilakukan.

Semisal kita mempunyai persamaan $y = mx + b$ *m dan b adalah parameternya. Saat proses training*, akan terdapat perubahan yang kecil *(error)*, kemudian parameter m dan b akan berubah menjadi $m = m - \partial m$ dan $b = b - \partial b$. Tujuannya adalah mencari m dan b yang memiliki *error* paling minimum.



Gambar 2 *Gradient Descent*

(Sumber:https://towardsdatascience.com/understanding-the-mathematics-behind-gradient-descent-dde5dc9be06e)

Jika dilakukan perhitungan hanya dengan menggunakan rumus tersebut, akan didapatkan iterasi yang cukup panjang untuk mendapatkan nilai minimum suatu kurva. Maka dari itu, ada suatu pengali yang bernama *Learning Rate*. Semakin tinggi *Learning Rate*, semakin besar langkah yang akan dilakukan, tetapi semakin rendah juga tingkat akurasinya. Maka dari itu, dibutuhkanlah *Learning Rate* yang tepat untuk mengurangi waktu *training* suatu model. Secara singkat, *Gradient Descent* dapat dirumuskan menjadi

$$W_x = W_x - l(\frac{\delta Error}{\delta W_x})$$

W_x = *Weight*

l = *Learning Rate*

C. *Greedy*

Algoritma *Greedy* adalah algoritma yang yang membangun solusi potong demi potong. Hal ini membuat solusi yang diambil adalah optimum lokal, sehingga bisa jadi tidaklah merupakan optimum global Seperti contoh dalam persoalan *knapsack,* dengan strategi *Greedy* bisa dilakukan dengan mengambil barang yang memiliki harga dibagi berat paling tinggi terlebih dahulu, atau bisa juga mengambil barang yang memiliki nilai jual lebih dahulu, atau dengan strategi lainnya.

Contoh lain dalam algoritma *Greedy* adalah seperti membuat strategi pada suatu permainan. Semisal ada sebuah permainan poker, kita bisa melakukan strategi melakukan fold jika tidak ada pasangan sama sekali dengan kartunya dan melakukan call saat ada kartu berpasangan di tangan.

D. *Backtracking*

Backtracking dapat dipandang sebagai salah satu dari

III. PEMBAHASAN

A. *Scoring* dan *Activation*

Pada bagian ini, akan dibuat beberapa *scoring* dan *activation function* yang akan digunakan.

MSE, Sigmoid, ReLU, Accuracy Score

```
def relu(x, derivative=False):
  if derivative:
    x[x > 0] = 1
    x[x <= 0] = 0
  return np.maximum(0, x)

def sigmoid(x, derivative=False):
  if derivative:
    return x * (1 - x)
  return 1 / (1 + np.exp(-x))

def mse(y_pred, y, derivative=False):
  if derivative:
    return y_pred * (y_pred - y)
  return 0.5 * (y_pred - y) ** 2

def accuracy_score(y_pred, y_test):
  count = 0
  total = len(y_pred)
  for i in range(total):
    if(y_pred[i] != y_test[i]):
      count += 1
  return 1 - count/total
```

Method di atas akan digunakan sebagai *activation function* dan *scoring* yang akan digunakan dalam makalah ini.

B. *Hidden Layer*

Pada bagian ini, kita membuat *class* yang berisikan tentang tingkah dari tiap *layer*. ANN yang kita buat akan memiliki *activation function* sendiri tiap *layer*nya dan dapat melakukan perhitungan untuk dilakukan dalam *Feedforward Phase*.

HiddenLayer

```
class HiddenLayer:
  def __init__(self, dimension,
activation_function, prev_res=None):
    self.weights = np.random.uniform(-1,
1, dimension)
    self.out = None
    self.prev_res = prev_res
    self.bias = np.random.uniform(-1,1)
    self.activation_function =
activation_function

  def calculate_out(self, prev_res):
    self.prev_res = prev_res
    self.out =
self.activation_function(self.prev_res.do
t(self.weights) + self.bias)
```

Layer yang kita buat memiliki atribut *weights* yang berisi *weight* dari tiap *node* yang ada. *Weight* awal diinisialisasi secara acak dari -1 sampai 1 Atribut *out* diisi *None* terlebih dahulu karena akan diisi setelahnya saat *Feedforward Phase*. Atribut *prev_res* yang merupakan hasil sebelumnya dari layer sebelumnya yang akan diisi nanti saat *Feedforward Phase*. Kemudian ada atribut *bias* yang akan diinisasi secara acak terlebih dahulu dari -1 sampai 1. Terakhir adalah atribut *activation_function* yang merupakan *activation function* yang akan dipakai di *layer* tersebut.

Layer memiliki konstruktor yang berparameter *dimension* yang berarti berapa banyak *node* yang akan dibuat, *activation_function* yang merupakan *activation function* apa yang akan digunakan, dan *prev_res* yang merupakan hasil sebelumnya yang memiliki *default value None.*

Pada *method calculate_cost*, dilakukan perhitungan untuk melakukan *Feed Forward*. Hal ini dilakukan dengan cara menyimpan *prev_res* dengan hasil sebelumnya, kemudian melakukan *dot product prev_res* dengan *weight* dari tiap node, yang kemudian akan ditambahkan dengan *bias*. Hasil dari perhitungan ini kemudian dihitung lagi di dalam *activation function* yang dimiliki *Layer* tersebut. Hasil dari perhitungan tersebut akan disimpan ke dalam atribut *out* yang merupakan hasil dari *feedforward layer* tersebut.

C. *ANNClassifier*

Pada bagian ini, kita membuat *class* yang berisi proses-proses dalam *ANN* itu sendiri.

ANNClassifier

```
class ANNClassfier:
  def __init__(self, in_val, out,
learning_rate, metrics=accuracy_score,
verbose=1):
    self.in_val = in_val
    self.layers = []
    self.out = out
    self.learning_rate = learning_rate
    self.metrics = metrics
    self.verbose = verbose
    self.cost = []
    self.accuracy = []
```

Pada *Class ANNClassifier,* terdapat atribut *in_val* yang merupakan *input value* atau data yang akan diprediksi. Atribut selanjutnya adalah *layers* yang akan berisi *HiddenLayer* yang akan ditambahkan nantinya. Atribut lain adalah *out* yang berisi label hasil dari *in_val*. Atribut selanjutnya adalah *learning_rate*, yang merupakan *learning rate* dalam algoritma *gradient descent* yang akan dipakai pada proses *backpropagation.* Atribut lainnya adalah *metrics* yang mengacu pada *scoring* yang akan digunakan untuk melihat kualitas model. Atribut selanjutnya adalah *verbose* yang menandakan berapa setiap berapa kali iterasi akan dicetak ke dalam layar. Untuk dua atribut terakhir, yaitu *cost* dan *accuracy* adalah *history* dari *cost* dan *accuracy* dari setiap iterasinya.

D. *add*

Method ini berada dalam *class ANNClassifier. Method* ini digunakan untuk menambahkan *HiddenLayer* ke dalam *ANNClassifier*.

add

```
def add(self, nodes,
activation_function=sigmoid):
  if len(self.layers) == 0:
      weights_shape =
(self.in_val.shape[1], nodes)
      prev_res = self.in_val
      layer = HiddenLayer(weights_shape,
activation_function, prev_res)
  else:
```

```
      weights_shape =
(self.layers[-1].weights.shape[1], nodes)
      layer = HiddenLayer(weights_shape,
activation_function, self.layers[-1].out)
  self.layers.append(layer)
```

Method ini memiliki *parameter nodes* dan *activation_function* yang *default value*nya adalah *sigmoid*. *Nodes* merepresentasikan berapa banyak *nodes* yang dimiliki oleh *HiddenLayer* ini. *Parameter activaion_function* adalah *activation function* yang akan digunakan di *HiddenLayer* ini.

E. Feedforward

Method ini berada dalam *class ANNClassifier* yang berfungsi untuk melakukan *Feedforward Phase* yang akan dilakukan pada *train* data nantinya.

Feedforward

```
def feed_forward(self):
  prev_res = self.layers[0].prev_res
  for layer in self.layers:
    layer.calculate_out(prev_res)
    prev_res = layer.out
```

Pada *Method* ini akan dilakukan perhitungan dengan menggunakan method *calculate_out* yang berada pada *class HiddenLayer*. *Feedforward* dilakukan dengan memasukkan terlebih dahulu *layer* awal *(self.layers[0].prev_res)* pada *prev_res* yang kemudian dalam iterasi akan berlanjut sampai *layer* terakhir yang ada dalam *ANNClassifier*.

F. Backpropagation

Method ini berada dalam class ANNClassifier yang berfungsi untuk melakukan *Backpropagation Phase* yang akan dilkaukan pada train data nantinya.

Backpropagation

```
def back_propagation(self):
  prev_layer = last_layer =
self.layers[-1]
  d = mse (last_layer.out, self.out,
True) *
last_layer.activation_function(last_layer
.out, True)
  last_layer.weights -=
self.learning_rate *
last_layer.prev_res.T.dot(d)
  last_layer.bias -= self.learning_rate *
np.mean(d)
  for layer in np.flip(self.layers,
axis=0)[1:]:
    d = d.dot(prev_layer.weights.T) *
last_layer.activation_function(last_layer
.out, True)
    layer.weights -= self.learning_rate *
layer.prev_res.T.dot(d)
    layer.bias -= self.learning_rate *
np.mean(d)
    prev_layer = layer
```

Pada *Method* ini akan dilakukan perhitungan *Backpropagation*. Perhitungan ini dimulai dengan melakukan *assign layer* terakhir *(self.layers[-1])* ke dalam *variable prev_layer* dan *last_layer*. Setelah itu, kita mulai dengan menggunakan algoritma *gradient descent*, yaitu dengan menggunakan turunan *MSE,* dikalikan dengan turunan *activation function* yang digunakan. Setelah itu kurangilah *weight* node tersebut dengan *layer* terakhir yang di *transpose* dan dilakukan *dot product* dengan perhitungan sebelumnya dan dikalikan dengan *learning rate* yang digunakan. Langkah selanjutnya adalah meng*update bias* dengan mengurangi dengan *learning rate* dikalikan dengan rata-rata dari perhitungan *MSE* dan *activation function* tadi. Lakukan *update prev_layer* menjadi *layer* yang baru saja diiterasi. Kemudian lakukanlah hal tersebut kembali sampai dengan layer tersebut habis (dalam *code* digunakan iterasi menggunakan *for*).

G. Train

Method ini berada dalam *class ANNClassifier* yang berfungsi untuk melakukan *training*.

Train

```
def train(self, epochs):
  self.epochs = epochs
  self.layers = np.array(self.layers)
  for i in range(epochs):
      self.feed_forward()
      self.back_propagation()
      self.cost.append(
      np.mean(mse(self.layers[-1].out,
```

```
self.out)))
      self.accuracy.append(
accuracy_score( self.out, np.round(
self.predict(self.in_val))))
      if((i+1) % self.verbose == 0):
        print('epoch {}/{}'.format(i+1,
epochs), end='')
        print(' cost =
{}'.format(np.mean(mse(self.layers[-1].ou
t, self.out))))
```

Pada *Method* ini akan dilakukan *training*. Hal ini menyangkut dengan penggabungan dari beberapa *method* sebelumnya, yaitu *Feedforward* dan *Backpropagation*. Hasil perhitungan tersebut akan disimpan ke dalam *history* untuk dibuat sebuah *plot*.

Method ini memiliki parameter *epochs* yang merupakan berapa banyak iterasi yang akan dilakukan untuk melakukan *feedforward* dan *backpropagation*.

H. *Predict*

Method ini berada pada *class ANNClassifier*. *Method* ini digunakan untuk melakukan prediksi pada data yang ingin diprediksi.

Predict

```
def predict(self, in_val):
  prev_res = np.array(in_val)
  for layer in self.layers:
      layer.calculate_out(prev_res)
      prev_res = layer.out
  return prev_res
```

Prediksi memiliki konsep yang sama dengan *feedforward,* yaitu dengan melakukan iterasi dari *layer* awal hingga *layer* akhir dan melakukan perhitungan menggunakan *activation function* yang ada. Hasilnya adalah perhitungan terakhir pada *layer* terakhir.

I. *Plot*

Method ini digunakan untuk melihat visualisasi *history*

Plotting

```
def plot_cost(self):
  plt.plot(range(self.epochs), self.cost,
color = 'r')
  plt.ylabel('cost')
  plt.xlabel('epochs')
  plt.title('Cost')
  plt.show()
  return

def plot_accuracy(self):
  plt.plot(range(self.epochs),
self.accuracy, color = 'r')
  plt.ylabel('accuracy')
  plt.xlabel('epochs')
  plt.title('Accuracy')
  plt.show()
  return
```

J. *Testing*

Pada *testing*, digunakan *dataset breast cancer* yang dapat *diimport* langsung dari *python*.

```
from sklearn.datasets import
load_breast_cancer
breast_cancer_dataset =
load_breast_cancer()
x = breast_cancer_dataset['data']
y = breast_cancer_dataset['target']
y = y.reshape(-1,1)
```

Setelah melakukan *load dataset*, kita lakukan *scalling* data dengan menggunakan *Standard Scaler*.

```
from sklearn.preprocessing import
StandardScaler
scale = StandardScaler()
x = scale.fit_transform(x)
```

Untuk meningkatkan tingkat keaslian test, kita lakukan *splitting* data terlebih dahulu dengan perbandingan 4:1 untuk *train* dan *test* datanya.

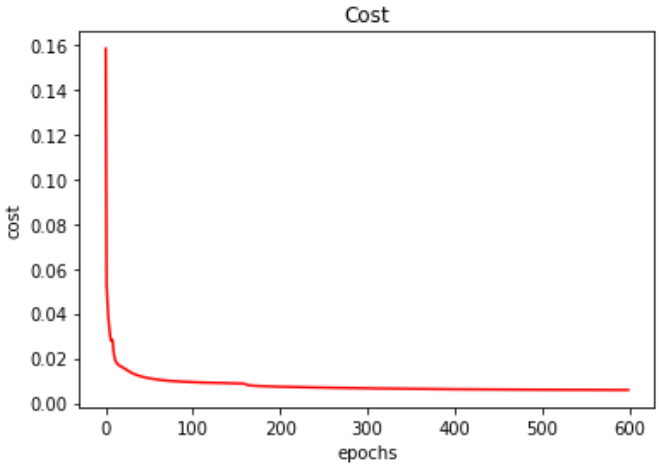
```
from sklearn.model_selection import
train_test_split
x_train, x_test, y_train, y_test =
train_test_split(x, y, random_state=0,
test_size=0.2)
```

Sekarang kita bentuk model *ANN* kita sekaligus melakukan testing.

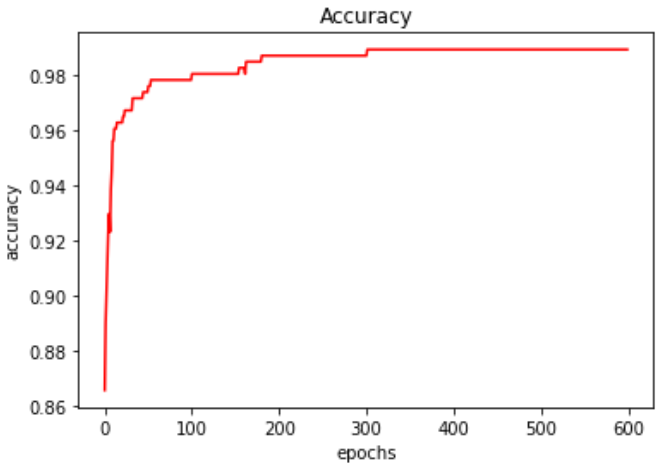
```
ann = ANNClassfier(x_train, y_train,
learning_rate=0.1, verbose=100)
ann.add(4, activation_function=relu)
ann.add(1, activation_function=sigmoid)
ann.train(epochs=600)
```

```
epoch 100/600 cost = 0.00935379100515441
epoch 200/600 cost = 0.007330621980625289
epoch 300/600 cost = 0.006616493847535088
epoch 400/600 cost = 0.0061976735034369726
epoch 500/600 cost = 0.005990463342669781
epoch 600/600 cost = 0.005871010890050755
```

Berikut adalah hasil *plot* dari *history cost* dan *accuracy* nya.



Grafik 1 *Cost History*



Grafik 2 *Accuracy History*

Setelah melakukan *train* untuk model *ANN*, kita dapat melakukan prediksi dengan *weight* yang ada di tiap *Node layer*.

```
y_pred = np.round(ann.predict(x_test))
print("Accuracy = " +
str(accuracy_score(y_pred, y_test)))
print(confusion_matrix(y_pred, y_test))
```

```
Accuracy = 0.956140350877193
[[45  3]
 [ 2 64]]
```

Didapatkan akurasi sebesar 95.6%. Hal ini bisa dilihat bahwa hasil ANN dengan *node* sebanyak 4 dan hanya 1 *layer* cukup untuk membuat prediksi yang bisa dibilang cukup baik dalam data test ini.

IV. KESIMPULAN

Setelah melakukan *testing* dengan menggunakan data *breast cancer* yang didapatkan dari *python,* kita berhasil mendapatkan *plot graph* untuk *cost* dan akurasi. Dilihat *cost* secara periodik menurun, hal inilah yang diharapkan saat melakukan *training* dengan menggunakan model *ANN*. Dampak dari *cost* yang semakin kecil adalah meningkatnya tingkat akurasi. Bisa dilihat pada *plot graph* akurasi, akurasi meningkat secara drastis pada iterasi awal, sedangkan pada suatu titik, akurasi tidak dapat meningkat lagi.

Terdapat banyak cara untuk meningkatkan akurasi, beberapa di antaranya adalah melakukan analisis data terlebih dahulu, membuang/mengganti data kotor, menggabungkan beberapa kolom, membuang beberapa kolom, menganalisis korelasi tiap kolom, menambahkan data, atau melakukan *parameter tuning* dalam modelnya. *Parameter Tuning* pada

ANN dapat dilakukan dengan merubah modelnya atau dengan merubah *activation function* yang digunakan.

Penerapan algoritma *greedy* bisa dilihat pada *gradient descent.* Dilakukan persamaan kalkulus yang menggunakan turunan untuk mendapatkan nilai minimum dari suatu kurva.

ink Video YouTube

Sebagai pendukung makalah, dibuatlah video yang akan diupload di platform YouTube. Link video tersebut dapat dilihat di https://youtu.be/HnktmwFkurY.

Link Github

Source code dapat dilihat di link berikut https://github.com/stanleyyoga123/ANN.

Ucapan Terima Kasih

Penulis berterima kasih kepada Tuhan Yang Maha Esa yang telah berkat-Nya sehingga penulis dapat menyelesaikan makalah ini. Penulis juga berterima kasih kepada Ibu Dr. Nur Ulfa Maulidevi, S.T., M.Sc selaku dosen K2 karena telah membimbing saya dan memberikan ilmu kepada saya karena jika tanpa dia, makalah ini tidak akan terselesaikan sebagaimana mestinya. Penulis juga berterima kasih kepada teman penulis yang atas dukungannya, penulis dapat mengerjakan, dan menyelesaikan makalah ini.

References

1. Chauhan, Nagesh Singh. "*Build and Artificial Neural Network (ANN) from scratch: Part-1*".https://towardsdatascience.com/build-an-artificial-neural-network-ann-from-scratch-part-1-a21988497962, diakses pada Jumat, 1 Mei 2020 pukul 21.05 WIB.
2. Brownie, Jason. "*Gradient Descent For Machine Learning*". https://machinelearningmastery.com/gradient-descent-for-machine-learning/, diakses pada Jumat, 1 Mei 2020 pukul 21.34 WIB.
3. Munir, Rinaldi. "Algoritma Greedy (2020)". http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Algoritma-Greedy-(2020).pdf, diakses pada Jumat, 1 Mei 2020 pukul 22.01 WIB.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 2 Mei 2020

Stefanus Stanley Yoga Setiawan / 13518122