**Jurnal JTIK (Jurnal Teknologi Informasi dan Komunikasi)**

journal homepage: http://journal.lembagakita.org/index.php/jtik

# Pengembangan *Stochastic Gradient Descent* dengan Penambahan Variabel Tetap

Adimas Tristan Nagara Hartono [1*], Hindriyanto Dwi Purnomo [2]

[1*,2] *Program Studi Teknik Informatika, Fakultas Teknologi Informasi, Universitas Kristen Satya Wacana, Kota Salatiga, Provinsi Jawa Tengah, Indonesia.*

a r t i c l e i n f o

*Article history:*
Received 19 November 2022
Received in revised form
23 February 2023
Accepted 1 May 2023
Available *online* July 2023

DOI:
https://doi.org/10.35870/jtik.v7i3.840

*Keywords:*
Stochastic Gradient Descent (SGD); Modification; Performance.

*Kata Kunci:*
Stochastic Gradient Descent (SGD); Modifikasi; Performa.

a b s t r a c t

Stochastic Gradient Descent (SGD) is one of the commonly used optimizers in deep learning. Therefore, in this work, we modify stochastic gradient descent (SGD) by adding a fixed variable. We will then look at the differences between standard stochastic gradient descent (SGD) and stochastic gradient descent (SGD) with additional variables. The phases performed in this study were: (1) optimization analysis, (2) fix design, (3) fix implementation, (4) fix test, (5) reporting. The results of this study aim to show the additional impact of fixed variables on the performance of stochastic gradient descent (SGD).

a b s t r a k

Stochastic Gradient Descent (SGD) adalah salah satu dari optimizer yang sering digunakan dalam deep learning, maka dari itu dalam penelitian ini akan melakukan sebuah modifikasi terhadap Stochastic Gradient Descent (SGD) dengan menambahkan sebuah variabel tetap. Dan akan melihat perbedaan antara Stochastic Gradient Descent (SGD) standar dan Stochastic Gradient Descent (SGD) dengan tambahan variabel. Tahapan yang dilakukan dalam penelitian ini yaitu: (1) Analisis Optimizer, (2) Perancangan Modifikasi, (3) Implementasi Modifikasi, (4) Pengujian Modifikasi, (5) Penulisan Laporan. Hasil penelitian adalah menunjukan dampak dari penambahan variabel tetap terhadap performa Stochastic Gradient Descent (SGD).

*Corresponding Author. Email: 672018063@student.uksw.edu [1*].

# 1. Latar Belakang

Manusia selalu berinovasi agar dapat memudahkan kehidupan, sebagai contoh *Artificial Intelligence* (*AI*) *yang* adalah simulasi kecerdasan manusia yang dimodelkan di dalam komputer, agar komputer itu dapat berpikir seperti layaknya manusia [1]. Salah satu bagian dari ilmu komputer yang mempelajari bagaimana membuat mesin (komputer) dapat melakukan pekerjaan seperti dan sebaik yang dilakukan oleh manusia bahkan bisa lebih baik daripada yang dilakukan manusia. Dalam pembuatan *Artificial Intelligence* dibutuhkan *Machine Learning* yang adalah cabang dari *artificial intelligence (AI)* dan *computer science*. Yang berfokus pada penggunaan data dan algoritma untuk mencontoh proses manusia belajar, dan secara bertahap meningkatkan akurasinya [2], proses belajar *artificial intelligence* tidaklah sederhana, untuk *Artificial Intelligence* dapat membedakan antara pensil dan buku, dibutuhkan proses yang memakan waktu. Proses belajar tersebut dibutuhkan data yang banyak dan waktu yang relatif lama, tetapi waktu yang dibutuhkan dalam pembuatan *Artificial Intelligence* bergantung pada banyaknya data dan juga kemampuan dari komputer tersebut. Semakin banyak data berarti akan semakin optimal juga kinerja dari *Artificial Intelligence* tetapi dengan banyaknya data akan memperlama proses belajar. Tetapi masalah ini dapat diatasi dengan *Video Graphics Array* (VGA) dan *Central Processing Unit* (CPU*)* komputer yang berspesifikasi tinggi. Dengan berkembangnya teknologi, cara untuk mempercepat proses belajar ditemukan, yaitu *Optimizer Deep Learning* yang adalah komponen penting yang dapat meningkatkan performa secara signifikan [3].

*Deep Learning* dimulai pada tahun 2006 yaitu setelah Geoffrey Hinton mempublikasikan *paper* yang memperkenalkan salah satu varian *neural network* yang disebut *deep belief network*. Yang merupakan cikal bakal optimizer yang memakai konsep gradien, seperti; *Gradient Descent, Stochastic Gradient Descent, Mini-Batch Gradient Descent, Momentum,* dan ADAM. Dalam artikel ini akan menggali lebih dalam tentang SGD *optimizer*, SGD adalah singkatan dari *Stochastic Gradient Descent* sesuai namanya SGD adalah *optimizer* yang algoritmanya berdasarkan pada gradien. SGD juga dikenal sebagai algoritma yang cepat dalam melakukan optimalisasi, dikarenakan SGD tidak menghitung semua data yang ada melainkan mengambil salah satu data dan menghitung gradien data tersebut, lalu nilai dari gradien tersebut akan di *update* dengan menggunakan *learning rate*, proses *update* nilai ini akan berlangsung sampai nilai mencapai titik terendah. SGD termasuk dalam *optimizer* yang cepat tetapi memiliki fluktuasi yang cukup banyak, dikarenakan SGD menghitung penafsiran gradien dan bukan nilai asli gradien tersebut. Dikarenakan *optimizer* yang beragam dan memiliki algoritma yang berbeda-beda, dari sini munculah ide untuk menambahkan variabel tetap yang dapat mengubah performa dari SGD tersebut dan diharapkan dapat mempercepat proses yang seharusnya memakan banyak waktu menjadi lebih cepat.

*Deep Learning* adalah sub bidang dari *machine learning* dimana abstraksi tingkat tinggi dari data dipelajari [13]. Proses belajar ini adalah proses yang berulang ulang yang melibatkan penyebaran *weight* dan bias jaringan yang bekerja dari *input* ke *output* dan sebaliknya. Perbedaan antara *machine learning* dan *deep learning, machine learning* adalah sistem kecerdasan buatan yang dapat belajar sendiri berdasarkan algoritma. Sistem ini akan semakin pintar seiring berjalannya waktu tanpa campur tangan manusia. *Deep learning* adalah machine learning yang diaplikasikan dengan dataset berukuran besar.

Proses belajar ini memiliki banyak sekali variasi dan tidak mungkin satu *optimizer* dapat mencangkup semua, maka dari itu banyak ragam optimizer diciptakan yang memiliki keunikannya masing-masing, berikut adalah beberapa contoh dari *optimizer* dan penjelasan singkatnya. DNNs (*Deep Neural Network*) terstruktur di dalam *layer-layer*, dan *layer-layer* tersebut disusun dalam bentuk *chain structure,* dengan setiap *layer* menjadi fungsi untuk *layer* sebelumnya. Dengan cara ini *output* keseluruhan dari *Neural network* didapatkan dengan menghitung output dari *layer* berturut turut yang berasal dari *input layer* [14].

F*eed forward neural network,* adalah *optimizer* yang terinspirasi dari otak manusia, dalam *feed forward neural network* setiap unit dalam *layer* saling terhubung dengan *layer* sebelumnya [15]. *Radial basis function neural network* merupakan keluarga dari *Multi Layer perceptron neural network*(MLPNN) yang mampu mengestimasi sebuah fungsi *non linier* [16].

*Multi Layer Perceptron neural network* (MLP) merupakan kelas dari *feed forward neural network* yang mempelajari hubungan data *linear* dan *non linear*. Keunggulan dari MLP adalah dalam penentuan bobot yang lebih baik daripada metode lain, dan MLP dapat digunakan tanpa pengetahuan sebelum nya dan algoritma nya tergolong mudah untuk diimplementasikan [17].

*Convolutional Neural Network* (CNN) adalah tipe *neural network* yang memiliki keuntungan karena memiliki akurasi dalam klasifikasi yang sangat tinggi. CNN membagi pekerjaannya menjadi beberapa layer yang dimana memiliki 3 *layer* utama, yang dinamai *Convolutional layer, Pooling layer,* dan *Full connection layer*. *Convolutional layer* berfungsi untuk mengekstrak data yang akan digunakan untuk *training*, *Pooling layer* berfungsi untuk membuat filter berdasarkan aturan yang ditetapkan, dan *Full connection layer* sebenarnya adalah *Multi layer perceptron* (MLP) [18].

Lalu untuk mengetahui seberapa bagus hasil *training* dapat dilihat dari hasil akhir bila semakin banyak gambar yang tidak sesuai dengan labelnya maka kinerja dari *optimizer* belum maksimal, dan bisa juga dengan menampilkan *confidence level* untuk melihat seberapa tinggi mesin yakin terhadap klasifikasinya, dan bila dirasa *confidence level* masih rendah, dapat ditambahkan data dengan catatan, ini akan menambahkan waktu dalam proses mesin belajar, tetapi akurasi dan *confidence level* mesin tersebut akan meningkat.

*Recurrent Neural Network* (RNN) adalah jenis *artificial intelligence* (AI) yang membutuhkan banyak layer untuk memproses data inputan. Proses berjalannya adalah *input* diproses dan dipanggil secara berulang-ulang, *Input* yang dipakai biasanya adalah data sekuensial, suatu data termasuk dalam data sekuensial apabila mempunyai suatu urutan waktu atau suatu data didapat secara berurutan. Data sekuensial pasti memiliki hubungan erat satu sama lain, proses perulangan yang dilakukan dalam arsitektur *recurrent neural network* (RNN) dapat menyimpan informasi dari masa lalu, sehingga secara otomatis informasi tersebut disimpan dalam jaringan dan digunakan untuk *training* selanjutnya [19].

*Modular neural network* adalah sistem yang mencoba untuk memanfaatkan modularitas masalah. *Modular neural network* memecah masalah menjadi beberapa bagian, setiap bagian diberikan modul untuk solusi komputasi, tiap-tiap modul menyelesaikan masalah bagian mereka, dan modul akan mengembalikan hasil mereka kepada *central integrator*. Lalu *central integrator* menyatukan hasil dari tiap modul dan memberikan hasil akhir menjadi sebuah *output* [20].

Adam (*adaptive moment estimation.*) adalah ekstensi dari *stochastic gradient descent* yang digunakan secara luas dalam *deep learning* di bidang kecerdasan buatan agar komputer dapat melihat dan memproses bahasa. Lalu apa yang membuat adam ini spesial dibandingkan *stochastic gradient descent*?. Bila *stochastic gradient descent* selalu menggunakan *alpha* yang konstan dari awal hingga akhir, Adam pada awalnya menggunakan *alpha* yang sudah ditentukan tetapi seiring berjalannya waktu *alpha* tersebut akan beradaptasi menyesuaikan parameter yang ada, dengan menggunakan beta1 dan beta2 yang akan mengubah alpha yang telah ditentukan. Berikut adalah algoritma yang digunakan Adam untuk mengubah nilai *alpha*.

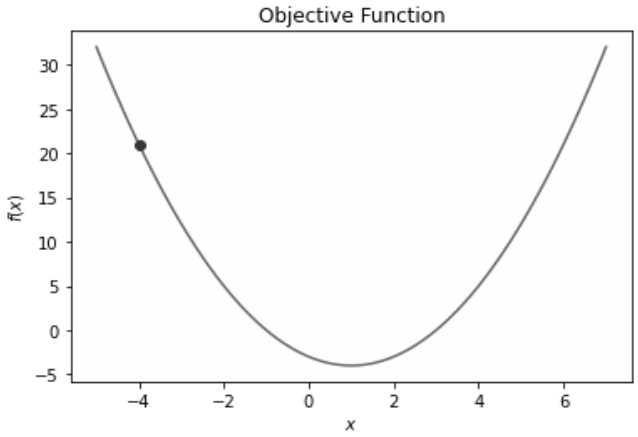
$$alpha\ (t) = \frac{alpha.sqrt(1-beta2(t))}{(1-beta1(t))} \quad (1)$$

Adagrad (*Adaptive Gradient Descent*) *Deep Learning Optimizer* berbeda dari *gradient descent algorithm*, yang menggunakan learning rate atau alpha yang konstan, Adagrad mengubah learning ratenya setiap perulangan atau disebut juga *adaptive learning rate*, perubahan learning rate ini berdasarkan dari perbedaan parameter selama training, semakin tinggi perubahan di parameternya maka akan semakin kecil perubahan learning ratenya, berikut algoritma dari Adagrad.

$$\eta_t = \frac{\eta}{sqrt(at+\epsilon)} \quad (2)$$

Dimana $\alpha$ akan berubah seiring berjalanya *training*, $\eta$ yang memiliki nilai konstan dan $\varepsilon$ adalah nilai positif kecil untuk menghindari pembagi menjadi 0. RMSProp (*Root Mean Squared Propagations*) termasuk ekstensi dari *gradient descent optimization algorithm*. RMSProp ini didesain untuk mempercepat proses *optimization* dan mengurangi jumlah evaluasi fungsi yang dibutuhkan untuk mencapai *local optima*. Dalam RMSProp *alpha* atau *learning rate*nya juga tidak konstan melainkan beradaptasi seperti Adam dan Adagrad. RMSProp dikenal dapat meningkatkan kemampuan sebuah *optimizer algorithm* dengan baik.

*Gradient Descent* seperti namanya *optimizer* ini menggunakan rumus *gradien* untuk mencari *local minimum*.

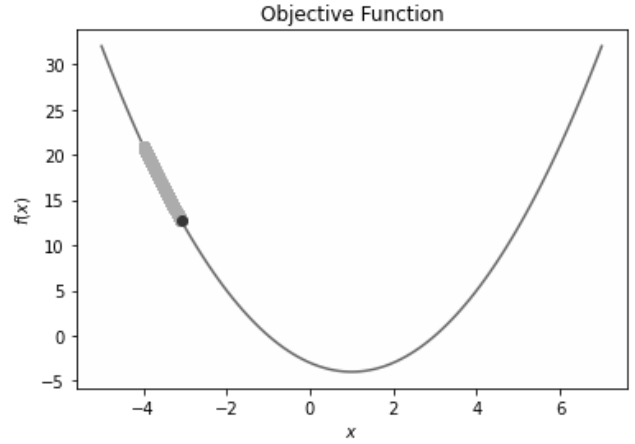



Gambar 1. Gradient descent graph

Gambar 1 adalah grafik *gradien* dari *gradient descent*, dan untuk rumus yang digunakan adalah:

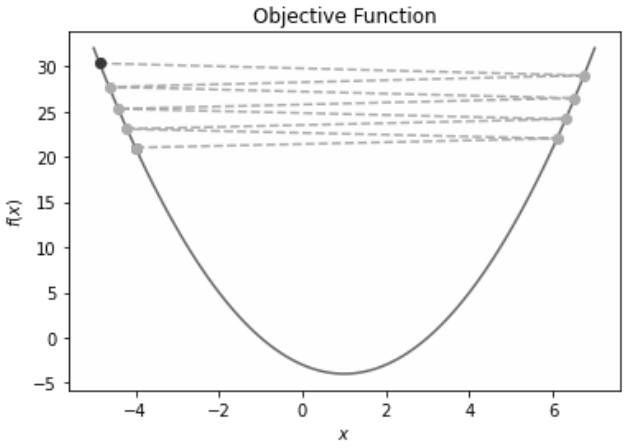
$$\chi\kappa + 1 = \chi\kappa + \alpha\kappa\,\rho\kappa \quad (3)$$

Dalam rumus tersebut $\chi\kappa$ adalah titik awal dimana gradient akan dimulai, $\alpha\kappa$ adalah *learning rate* yang akan menentukan seberapa jauh titik pada gradien akan berpindah semakin tinggi $\alpha\kappa$ maka semakin jauh juga langkah yang diambil dan $\rho\kappa$ adalah arah kemana gradient ini akan berjalan.




Gambar 2. Gradient descent graph with small alpha

Gambar 2 adalah contoh penggunaan *alpha* ($\alpha$) yang rendah yaitu $1x10^{-4}$ bila menggunakan *alpha* yang serendah ini akan memakan banyak waktu dan dibutuhkan banyak langkah untuk mencapai *local minimum*.




Gambar 3. Gradient descent with high alpha

Gambar 3 adalah contoh bila menggunakan *alpha* ($\alpha$) yang tinggi yaitu 1.0, dalam *deep learning* alpha 1 sudah tergolong tinggi dikarenakan *deep learning* bertujuan untuk mencari angka loss sekecil mungkin hingga angka mendekati 0, biasanya *alpha* yang digunakan adalah sekitar 0.1 dan dalam *gradient descent* alpha 0.1 dapat menghasilkan grafik yang baik.

Begitu banyak jenis dari *optimizer deep learning* dan tentu saja *optimizer* tidak luput dengan namanya kelebihan dan kekurangan. Dalam jurnal yang berjudul analisa bermacam optimizer pada convolutional neural network untuk deteksi pemakaian masker pengemudi kendaraan, dituliskan semakin banyak *epoch* atau perulangan dalam *training* akan menambah akurasi untuk *testing* dan *training*, di dalam jurnal ini juga membandingkan tiap-tiap *optimizer* yang ada dan mendapatkan hasil sebagai berikut [21]

Tabel 1. Hasil akurasi dan loss dari 3 tipe optimiser

| | Accuracy | | Loss | |
|---|---|---|---|---|
| | Training | Testing | Training | Testing |
| SGD | 0,6064 | 0,7577 | 0,6764 | 0,4945 |
| RMSprop | 0,97 | 0,9577 | 0,0791 | 0,1062 |
| ADAM | 0,97 | 0,9654 | 0,0856 | 0,0851 |

Dari hasil yang didapatkan sesuai yang tercantum di tabel 1, menunjukan bahwa SGD *optimizer* mendapatkan nilai *loss* tertinggi dan nilai akurasi terendah saat *testing* dan *training*, RMSprop dan ADAM memiliki nilai yang hampir mirip dalam akurasi tetapi nilai *loss* RMSprop kurang baik saat

*testing*, sedangkan ADAM mendapatkan hasil *loss* stabil. Dalam proses *training neural network* tanpa disadari akan menggunakan yang dinamakan *deep learning optimizer* proses ini disebut optimalisasi. *Optimizer* bisa juga bekerja sebagai pengubah nilai *weight* dan juga *learning rate,* mengubah *weight* dan *learning rate* dibutuhkan karena saat melakukan *training*, pada saat perulangan awal, *loss* yang didapatkan pasti tinggi bila tidak mengubah nilai *weight* dan *learning rate* nilai loss itu akan turun relatif stabil. Berbeda bila menggunakan *optimizer*, *optimizer* berperan mengubah *weight* dan *learning rate* maka nilai *loss* akan mendekati nilai *minimum* lebih cepat.

Dari jurnal milik Rizki Tri Prasetio dan Endang Ripandi mereka melakukan klarifikasi jenis populasi tumbuhan di hutan di Ibaraki, Jepang menggunakan algoritma SVM (*Support Vector Machine*) dan MLP (*Multi Layer Perceptron*) dan mendapatkan hasil akurasi sebesar 85.9%, dan menurutnya akurasi ini masih bisa ditingkatkan akurasinya dengan menggunakan optimasi fitur *Optimize Selection* pada algoritma *deep learning* dan akurasi pun meningkat secara signifikan menjadi 94,46% [12].

Dalam artikel milik Sandeep Giri, dia mendapatkan ide untuk membuat sebuah *custom optimizer,* di dalam artikelnya Sandeep Giri menjelaskan langkah-langkah membuat *optimizer* dari awal meng*import* hingga *optimizer* itu dapat berjalan. Pada saat dia mencoba *optimizer*nya, nilai *loss* yang didapatkan berkurang seiring waktu *training* berlangsung, hingga pada akhirnya nilai *loss* mencapai angka 0.5312 yang awalnya 3.7333, tetapi Sandeep Giri berkata kalau dibanginakn *optimizer* miliknya dengan *gradient descent* ataupun varian lainnya, akan terlihat bahwa *optimizer* buatan Sandeep Giri bukanlah sebuah kemajuan ataupun perkembangan [2]. *Loss rate* sendiri berfungsi untuk menghitung *error*, yang menjadi representasi dari perbedaan hasil dengan hasil yang diharapkan [4], dan semakin kecil *loss rate* yang didapatkan maka semakin akurat mesin dalam melakukan prediksi ataupun penilaian.

Untuk melakukan sebuah modifikasi *optimizer* dibutuhkan pemahaman tentang *optimizer* yang ingin di modifikasi. Mulai dari cara kerja *optimizer* tersebut, proses perubahan nilai yang terjadi, dan algoritma dari *optimizer* tersebut. Dalam artikel ini akan mencoba untuk memodifikasi SGD *optimizer*. SGD *optimizer* menggunakan konsep *gradien* yang adalah nilai kemiringan suatu garis yang dibandingkan antara x dan y. nilai x dan y akan berubah dalam setiap perulangan dan, titik temu dari nilai x dan y tersebut akan semakin mendekati angka 0,0 (nol koma nol). dan untuk algoritma SGD adalah sebagai berikut:

$$\theta = \theta - \eta \cdot \nabla\theta J(\theta; x^{(i)}; y^{(i)}) \quad (4)$$

Dari algoritma (4) diimplementasikan kedalam python menjadi:

$$x1 - eta\_t * g1, x2 - eta\_t * g2 \quad (5)$$

dimana x1 dan x2 adalah representasi dari x dan y, yang akan dipilih secara acak, eta_t adalah perkalian dari *estimator* dan *learning rate*, eta_t berfungsi untuk merubah nilai x1 dan x2 dan untuk eta_t biasanya menggunakan angka < 0, dan g1 dan g2 adalah *objective function*. O*bjective function* digunakan sebagai fungsi acuan dalam melakukan optimalisasi, tetapi fungsi tersebut perlu untuk dicari nilai turunannya terlebih dahulu sebelum digunakan, berikut adalah contoh fungsi yang digunakan sebagai *objective function*.

$$x1^{2} + 2 * x2^{2} \quad (6)$$

lalu fungsi (6) diturunkan menjadi

$$2 * x1, 4 * x2 \quad (7)$$

Untuk melakukan modifikasi dapat dilakukan dengan menambahkan (+), (-), (*), ataupun (/) dalam rumus pythonnya dan itu akan mengubah performa *optimizer* tersebut. Sebagai contoh

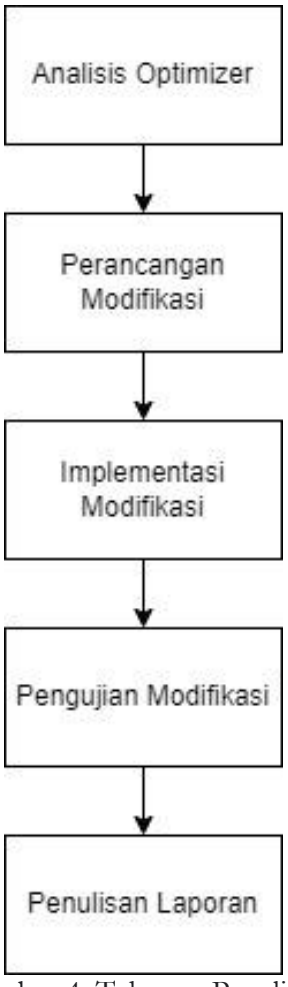
$$x1 - eta\_t * g1 + 1, x2 - eta\_t * g2 + 1 \quad (8)$$

Pada rumus (8) ditambahkan dengan +1 pada sisi x1 dan x2, untuk memodifikasi rumus perlu dilakukan di sisi x1 dan x2 karena kedua sisi tersebut memiliki nilai yang berbeda dan perhitungan masing-masing, maka dari itu perlu ditambahkan di kedua sisi agar nilai tidak berat sebelah.

## 2. Metode Penelitian

Tahapan yang dilakukan dalam penelitian ini yaitu: (1)

Analisis *Optimizer*, (2) Perancangan Modifikasi, (3) Implementasi Modifikasi, (4) Pengujian Modifikasi, (5) Penulisan Laporan.




Gambar 4. Tahapan Penelitian

Berdasarkan Gambar 4 dapat bahwa tahap pertama penelitian adalah analisis *optimizer* adalah proses untuk mengetahui bagaimana algoritma dari *optimizer* tersebut dan bagaimana kerjanya. Tahap kedua dan ketiga adalah perancangan dan implementasi modifikasi, y ang berarti melakukan *trial* terhadap banyak variabel terhadap SGD dan mengimplementasikannya. Tahap keempat adalah pengujian yaitu proses pengetesan terhadap modifikasi yang sudah diimplementasikan dan membandingkan performanya dengan SGD standar. Tahap terakhir adalah penulisan laporan dari hasil yang telah diuji.

## 3. Hasil dan Pembahasan

*Proses Modifikasi*

Algoritma *Stochastic Gradien*t *Descent* (SGD) yang diterapkan di dalam python ditulis sebagai berikut rumus (9).

$$(x1 - eta\_t * g1, x2 - eta\_t * g2, 0, 0) \quad (9)$$

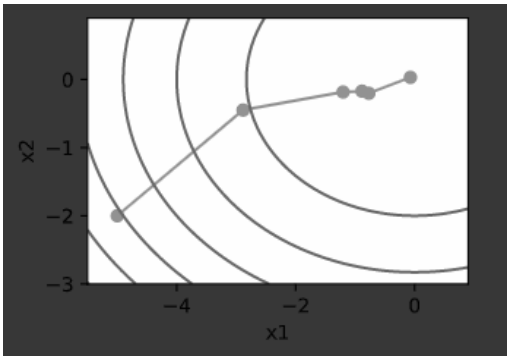
Dari rumus (9) ditambahkan sebuah variabel tetap, dan setelah melakukan percobaan akhirnya ditemukanlah variabel yang dapat mempercepat kinerja SGD, dan algoritma SGD *standard* itu ditambahkan *2 dan menjadi rumus (10).

$$(x1 - eta\_t * g1 * 2, x2 - eta\_t * g2 * 2, 0, 0) \quad (10)$$

Dan dari rumus (9) berubah menjadi rumus (10) dengan penambahan (*2) disisi x1 dan x2. Penambahan *2 didasari oleh percobaan yang telah dilakukan, dengan mencoba menambahkan penjumlahan, pengurangan, perkalian, dan pembagian dasar, dan hasil yang didapatkan *2 menunjukan hasil yang cukup memuaskan dibandingkan operasi matematika yang telah dicoba.

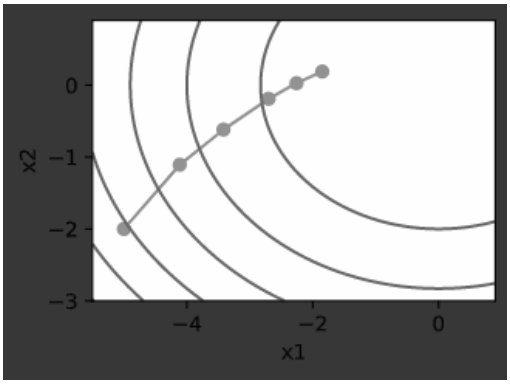
*Pengujian*

Setelah algoritma SGD ditambahkan variabel dan selanjutnya adalah pengujian, dan pengujiannya adalah sebagai berikut kedua algoritma dijalankan dengan inputan *steps* atau perulangan dengan jumlah yang sama dan disini digunakan *steps* dengan angka yang tergolong rendah. Grafik yang dihasilkan SGD modifikasi seperti berikut (Gambar 5).




Gambar 5. Grafik hasil SGD modifikasi

Dengan menambahkan x2 maka langkah yang diambil oleh SGD untuk tiap perulangannya akan lebih besar dan akan mempercepat untuk mencapai *local optima*. Dan berikut adalah Grafik yang dihasilkan dari algoritma SGD *standard* dengan *steps* atau perulangan yang sama.




Gambar 6. Grafik hasil SGD standard

Dari kedua grafik tersebut dapat disimpulkan bahwa dengan menambahkan sebuah variabel tetap dapat membuat sebuah algoritma SGD berubah dan menjadi lebih cepat dalam mencari *local optima*.

*Proses Benchmark*

Tabel 2. Hasil Benchmark dari SGD dan SGD modifikasi

| | $X1^2 + 2 * x2^2$ Function | | | Sphere Function | | |
|---|---|---|---|---|---|---|
| | Best | Mean | Standard Deviation | Best | Mean | Standard Deviation |
| SGD Modification | 0.016634 | 0.075048 | 0.843444 | 0.038852 | 0.11673 | 0.752873 |
| Standard SGD | 0.004158 | 0.01537 | 0.493374 | 0.027946 | 0.006237 | 0.580105 |

Bila hanya menguji berdasarkan grafik, dirasa tidaklah adil maka dari itu dilakukanlah proses benchmark. tabel 2 menunjukan hasil dari benchmark antara SGD *standard* dan SGD modifikasi, dapat dilihat bila *mean* dari SGD standar pada *sphere function* menunjukan hasil yang lebih baik yaitu 0.006237 sedangkan SGD modifikasi menunjukan hasil 0.11673. Bila hanya menjalankan dengan perulangan yang rendah SGD modifikasi dapat menemukan titik minimum lebih cepat tetapi pada saat dibandingkan dengan perulangan yang lebih banyak SGD modifikasi menghasilkan *loss* yang lebih tinggi dan ini berarti SGD *standard* masih lebih baik dibandingkan dengan SGD modifikasi. Dan dapat disimpulkan bahwa modifikasi yang dilakukan bukanlah sebuah perkembangan dan masih diperlukan percobaan yang lebih mendalam agar dapat menemukan sebuah rumus yang dapat membuat performa SGD menjadi berkembang.

## 4. Kesimpulan

Berdasarkan hasil penelitian dan pembahasan maka dapat disimpulkan variabel tetap dapat mengubah performa dari SGD, dan bila diuji dengan perulangan yang sedikit SGD modifikasi menunjukan hasil yang baik. Tetapi dalam pengetesan yang umum digunakan menunjukan hasil bahwa SGD standar masih lebih baik daripada SGD modifikasi.

## 5. Daftar Pustaka

[1] Dahria, M., 2008. Kecerdasan Buatan (Artificial Intelligence). *Jurnal Saintikom*, *5*(2), pp.185-197.

[2] Giri, S., 2020. Writing Custom Optimizer in TensorFlow Keras API, https://cloudxlab.com/blog/writing-custom-optimizer-in-tensorflow-and-keras/ [Accessed at 28 Oktober 2021].

[3] Wright, L. and Demeure, N., 2021. Ranger21: a synergistic deep learning optimizer. *arXiv preprint arXiv:2106.13731*.

[4] Bircanoğlu, C., 2017. *A comparison of loss functions in deep embedding* (Master's thesis, Fen Bilimleri Enstitüsü).

[5] Nurfita, R.D. and Gunawan Ariyanto, S.T., 2018. *Implementasi Deep Learning Berbasis Tensorflow Untuk Pengenalan Sidik Jari* (Doctoral dissertation, Universitas Muhammadiyah Surakarta).

[6] Machine Learning what it is and why it matters, https://www.sas.com/en_in/insights/analytics/machine-learning.html [Accessed at 2 November 2021].

[7] Albers Uzila, 2021, Complete step by step gradient descent algorithm from scratch, https://towardsdatascience.com/complete-step-by-step-gradient-descent-algorithm-from-scratch-acba013e8420 [Accessed at 25 November 2021]

[8] Jason Brownlee, 2021, Adam optimization from the scratch, https://machinelearningmastery.com/adam-optimization-from-scratch/ [Accessed at 25 November 2021]

[9] Ayush Gupta, 2021, Comprehensive Guide on Deep Learning Optimizers, https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/ [Accessed at 25 November 2021]

[10] Jason Brownlee, 2021, Gradient descent with rmsprop from scratch, https://machinelearningmastery.com/gradient-descent-with-rmsprop-from-scratch/ [Accessed at 25 November 2021]

[11] Roan Gylberth, 2018, An introduction to adagrad, https://medium.com/konvergen/an-introduction-to-adagrad-f130ae871827 [Accessed at 25 November 2021].

[12] Prasetio, R.T. and Ripandi, E., 2019. Optimasi Klasifikasi jenis hutan menggunakan deep learning berbasis optimize selection. *Jurnal Informatika*, *6*(1), pp.100-106. DOI: https://doi.org/10.31294/ji.v6i1.5176

[13] Bengio, Y. and LeCun, Y., 2007. Scaling learning algorithms towards AI. *Large-scale kernel machines*, *34*(5), pp.1-41. DOI: https://doi.org/10.1038/nature14539.

[14] Nwankpa, C.E., 2020. Advances in optimisation algorithms and techniques for deep learning. *Advances in Science, Technology and Engineering Systems Journal*, *5*(5), pp.563-577. DOI: https://doi.org/10.25046/aj050570.

[15] Sazli, M.H., 2006. A brief review of feed-forward neural networks. *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering*, *50*(01). DOI: https://doi.org/10.1501/commua1-2_0000000026.

[16] Rinanto, N., Wahyudi, M.T. and Khumaidi, A., 2018. Radial basis function neural network sebagai pengklasifikasi citra cacat pengelasan. *Rekayasa*, *11*(2), pp.118-131. DOI: https://doi.org/10.21107/rekayasa.v11i2.4418 .

[17] Wibawa, A.P., Lestari, W., Utama, A.B.P., Saputra, I.T. and Izdihar, Z.N., 2020. Multilayer Perceptron untuk Prediksi Sessions pada Sebuah Website Journal Elektronik. *Indonesian Journal of Data and Science*, *1*(3), pp.57-67. DOI: https://doi.org/10.33096/ijodas.v1i3.15.

[18] Ibrahim, H.S., Jondri, J. and Wisesty, U.N., 2018. Analisis Deep Learning Untuk Mengenali Qrs Kompleks Pada Sinyal Ecg Dengan Metode Cnn. *eProceedings of Engineering*, *5*(2).

[19] Faishol, M.A., Endroyono, E. and Irfansyah, A.N., 2020. Predict Urban Air Pollution in Surabaya Using Recurrent Neural Network–Long Short Term Memory. *JUTI J. Ilm. Teknol. Inf*, *18*(2), p.102-114. DOI: http://dx.doi.org/10.12962/j24068535.v18i2.a988.

[20] Kala, R., Vazirani, H., Shukla, A. and Tiwari, R., 2010. Medical Diagnosis using Incremental Evolution of Neural Network. *Journal of Hybrid Computing Research*, *3*(1), pp.9-17.

[21] Wikarta, A., Pramono, A.S. and Ariatedja, J.B., 2020, December. Analisa Bermacam Optimizer Pada Convolutional Neural Network Untuk Deteksi Pemakaian Masker Pengemudi Kendaraan. In *Seminar Nasional Informatika (SEMNASIF)* (Vol. 1, No. 1, pp. 69-72).

[22] IBM Cloud Education, 2020, Machine Learning, https://www.ibm.com/cloud/learn/machine-learning#:~:text=Machine%20learning%20is%20a%20branch,learn%2C%20gradually%20improving%20its%20accuracy. [Accessed at 5 Juli 2022].