Schedae Informaticae Vol. 27 (2018): 19–30
doi: 10.4467/20838476SI.18.002.10407

Traffic Signal Settings Optimization Using Gradient Descent

Marcin Możejko[1], Maciej Brzeski[12], Łukasz Madry[13],
Łukasz Skowronek[1], Paweł Gora[13]
[1] TensorCell
[2] Faculty of Mathematics and Computer Science, Jagiellonian University
[3] Faculty of Mathematics, Informatics and Mechanics, University of Warsaw
e-mail: *mmozejko1988@gmail.com, maciej.brzeski@doctoral.uj.edu.pl, lukaszmadry95@gmail.com, lukasz.m.skowronek@gmail.com, p.gora@mimuw.edu.pl*

Abstract. We investigate performance of a gradient descent optimization (GR) applied to the traffic signal setting problem and compare it to genetic algorithms. We used neural networks as metamodels evaluating quality of signal settings and discovered that both optimization methods produce similar results, e.g., in both cases the accuracy of neural networks close to local optima depends on an activation function (e.g., TANH activation makes optimization process converge to different minima than ReLU activation).

Keywords: traffic optimization, metamodels, activation functions, genetic algorithm, gradient descent.

1. Introduction

The aim of this research is to investigate the gradient descent optimization method applied to the traffic signal setting problem defined in [1]. We investigate a special case of the traffic signal setting problem in which the optimized function is the total time of waiting on a red signal, while a setting of traffic signals is represented as a vector of 21 offsets corresponding to 21 crossroads with traffic signals in Warsaw, in the Stara Ochota district.

We present the gradient descent optimization approach which applies a backpropagation to modify inputs of the metamodel (built using neural networks) in order to minimize the network's output. We compare results of our experiments with results achieved using genetic algorithms (GA). One of interesting observations is that applying a hiperbolic tangent (TANH) as an activation function leads to better performance of neural networks comparing to ReLU in terms of accuracy and symmetry of error close to local optima. Another observation is that the GR is able to find similar optima (in terms of a value of the fitness function) as GA and they can be achieved faster which makes the method very promising for the future applications in solving the traffic signal setting problem and similar combinatorial optimization problems with surrogate models.

The rest of the paper is organized as follows: Section 2. presents the state of the art of the research on the traffic signal setting problem and some other approaches to traffic optimization. It also discusses some similar works related to applications of gradient optimization on inputs of neural networks (e.g., producing adversarial examples). Section 3. describes our approach to solving the traffic signal setting problem and setup of our experiments: available datasets, the process of training metamodels, settings of the gradient optimization method and genetic algorithms to which we compare. Section 4. presents and discusses results of our experiments. Section 5. concludes the paper and outlines plans for the future research.

2. Related work

Vehicular traffic in cities is a complex process, so it is not easy to manage it optimally. There are many possible traffic management approaches, but in our work we focus on the traffic signal control, for which there also already exist many research works, scientific approaches and real-world implementations - the good overview can be found, e.g., in [2] and [3]. Existing approaches have at least a few drawbacks:

- lack of evaluating impact of introduced changes on traffic,
- limited scalability and coordination on larger areas,
- limited adaptability in case of significant changes (e.g., car accidents),
- adaptation based only on small changes in signal settings.

Our long-term goal is to build a traffic management system solving the aforementioned issues of existing systems. One of the core elements of our method is solving the traffic signal setting problem - it is a combinatorial optimization problem which was formally introduced in [1] (however, some earlier works presented its simplified versions which in cases of relatively simple mathematical models were proven to be NP-hard [4]). The goal is to find the optimal settings of traffic signals located in some vertices of a graph representing the road network. The signal setting can be represented as a vector of signal offsets, durations of red and green signal phases or, in general, any strategy making decisions regarding the state of traffic signals. Formally, it can be considered as a function which takes as an argument time and traffic conditions (e.g., positions and speeds of cars) and returns a decision regarding states of each signal. The optimized function may be, e.g., the total time of travel or the total time of waiting on a red signal, and its values can be computed using traffic simulations.

The standard approaches to solve these combinatorial optimization problems are metaheuristics [5], e.g., evolutionary algorithms. Past approaches using genetic algorithms and evaluating quality of signal settings using traffic simulations proved to be successful in finding suboptimal solutions to that problem, but they required significant computational power and may be time-consuming [1]. The most compute-demanding component are traffic simulations, so some recent papers aimed to replace it using metamodels (or surrogate models), e.g., to train machine learning algorithms (e.g., neural networks or LightGBM [6]) to approximate outcomes of traffic simulations very fast and with a good accuracy (that approach is related to surrogate modelling [7] and counterfactual learning [8]). Thanks to that, it was possible to achieve a great speed up of the traffic optimization task (assuming that the pretrained metamodel is available) by replacing traffic simulations using accurate metamodels as evaluators of the quality of traffic signal settings, [9], [10].

However, the method had still some drawbacks, e.g., it was discovered that genetic algorithms converge to local optima close to which the accuracy of metamodels suddenly decreases, which makes the further optimization process more difficult [10]. It was also discovered that this phenomenon strongly depends on the type of a metamodel and in case of neural networks also on the type of an activation function (e.g., applying hiperbolic tangent may give better results than ReLU and may help mitigate that effect [10]). In this paper, we investigate this phenomenon in case of the gradient descent optimization (GR). Nowadays, GR is a standard procedure for training neural networks, i.e., finding weights of connections minimizing the loss function. However, the application of GR on the space of inputs to neural networks is much less common, such approach is used mostly for finding adversarial examples, i.e., after training the neural network weights are set to be constant, but the inputs to the network are modified using GO in order to maximize the loss function [11].

For the particular problem studied in this paper, the choice of neuron activation functions turns out to have a huge influence on approximation accuracy near the model minima. ReLU activation functions usually result in models that underestimate times of waiting when they are predicted to be small. This bias is much less visible when using TANH activation function, which could probably be attributed to the fact that, as opposed to ReLU, TANH does not grow very large in the yet unexplored regions of our input space.

The superiority of TANH over ReLU in our problem is not something very typical in machine learning literature. ReLUs have been popularized by [12] and well advocated by [13], using arguments such as sparsity of the learned representations and non-saturation during training. Also in the more recent years researchers have confirmed superiority of ReLU in a systematic study concerning a typical image recognition task [14]. In short, often ReLUs are the default choice of activation function, but our study shows that one should sometimes take them with a grain of salt and explore other options as well.

3. Setup of experiments

One of the goals of our research was to compare the gradient descent optimization (GR) approach with genetic algorithms (GA) approach described in [10]. Therefore, we decided to use the same dataset and the same models of neural networks as in case of that previous research.

Our dataset was built based on 105336 simulation runs using Traffic Simulation Framework (TSF) [15] with randomly selected traffic signal settings as an input. Each traffic signal setting is represented as a vector of 21 elements, each corresponding to a traffic signal offset (being an integer value from the set $\{0,1,. . . , 119\}$) for one of 21 selected crossroads in a specified region (Stara Ochota district in Warsaw). For each setting, TSF computed the total time of waiting on a red signal in the selected area. For the purpose of experiments (i.e., training metamodels), we divided that set into a training set (consisting of the first 85336 elements) and a test set (the remaining 20000 elements). Our dataset was also described in details in [10] and is publicly available [16].

We tested multiple fully-connected network architectures with and without residual connections [17], regularized using $l2$-weight regularization and batch normalization layers [18]. Our models were trained using RMSProp optimizer [19], for 200 epochs with the batch size of 128 and a mean squared error as a loss function. In order to normalize the input data and capture its periodic properties we transformed each coordinate x of the input data to a pair $\left(\cos\frac{2\pi x}{120}, \sin\frac{2\pi x}{120}\right)$, what changed the input shape from $(21,)$ (21 offsets of traffic signals) to $(42,)$. We trained 216 models of neural networks with different values of regularization rates, number of units and layers and activations (ReLU [20] and TANH [21]) and chose 8 models (from the set of the best 30 models with the lowest relative error rates) which had the greatest variability in terms of model metaparameters. All models were trained using Keras [22] package with TensorFlow [23] backend.

After training metamodels, we ran optimization experiments using GA. We developed a dedicated Python library and tested 504 configurations of hyperparameters of GA [24]. For each configuration, we ran experiments for each of our 8 metamodels, 5 times per each model (each run started from a random initial population to test different initial settings), giving 2520 runs for each model. For the purpose of further analysis, we selected 20 best GA runs for each model, so in total we had 160 trajectories of GA.

We had observed earlier that evaluating genotypes using TSF takes about 30 seconds, which is too long, but in case of evaluations using neural networks the time of inference was much lower (less than 0.1 second), so we were able to conduct such compute-intensive experiments.

Later, we ran similar traffic optimization experiments using GR. We tried a few optimizers with properly tuned learning rate: stochastic gradient descent (SGD, with $lr = 0.1$), Nesterov[25] ($lr = 0.1$ and $\beta = 0.9$), RMSProp ($lr = 1.0$) and ADAM[26] ($lr = 0.5$). For each method and each of 8 metamodels we ran 1000 gradient descent experiments (each with 100 iterations) and chose 20 best runs (with the lowest predicted values). This resulted in 160 trajectories per GR method. Although setting high learning rate for both ADAM and RMSProp is believed to harm the training process [26], it turned out that in our case lower values of a learning rate parameter slowed down the training process and made it stucking in local minima, whereas higher - made the optimization process highly unstable. In case of RMSProp - the proposed value of a learning rate helped solve these problems. Unfortunately, even careful fine-tuning of optimizer hyper-parameters did not work for ADAM.

As an explicit example, the update rules for parameter estimates θ_t in case of RMSProp are given by the following formulas:

$$v_t = \gamma v_{t-1} + (1-\gamma)\, g_t^2 \tag{1}$$

$$\theta_t = \theta_{t-1} + \frac{\eta}{\sqrt{v_t + \epsilon}} g_t \tag{2}$$

where g_t is a gradient calculated at time t and g_t^2 is g_t with all coordinates raised (component-wise) to power 2. The typical values for γ, η and ϵ are 0.9, 0.001 and 10^{-8}, respectively.

4. Results of experiments

In order to compare results of different optimization runs (4 based on GO and 1 on GA) we decided to analyze found minima: their simulation values, predicted (by metamodels) values and errors of each model. In order to perform analysis we chose last 20 iterations of optimization algorithm trajectories. Analysis of values showed that before this phase optimization algorithms had already converged, so these points represent minima to which a given optimization process converged.

4.1. Analysis of minima

During the analysis of the minima to which different optimization runs converged, we found out that for a given metamodel gradient descent and genetic algorithms tend to converge to the roughly the same regions of input space, what is visualized on Figure 1 using PCA dimensionality reduction.

4.1.1. Method for comparing optimization algorithms

In order to compare results obtained by different optimization methods we introduced the following measure of discrepancy between their results:

$$\Delta(A \mid B) = \mathbb{E}_{X \in res(A)} \min_{Y \in res(B)} \delta(X, Y), \tag{3}$$

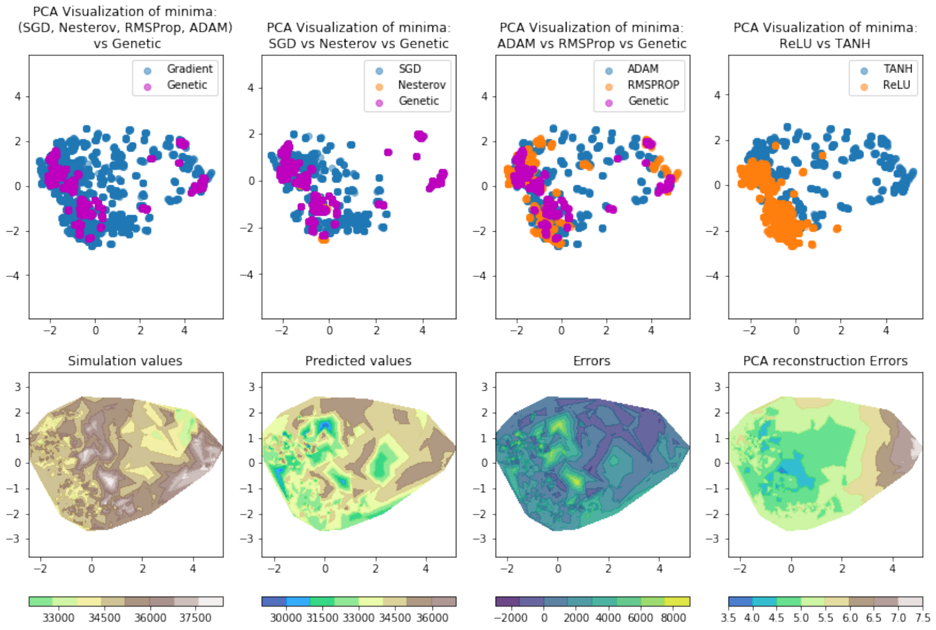

Figure 1. PCA visualization of points obtained in the last 20 iterations for different types of optimization algorithms. Most of the points concentrate in roughly the same input region. In regions on the left of the image, the error is smaller, whereas minima on the right have much steeper error / simulation structure. The PCA reconstruction error suggests that these trajectories might be outliers.

where A and B are different optimization methods, $res(M)$ is a set of point collections of last 20 iterations from each of 160 trajectories obtained by method M, and δ is an Euclidean distance between 2 sets of points given by:

$$\delta(X, Y) = \min_{x \in X, y \in Y} ||x - y||_2. \tag{4}$$

So, the Δ-discrepancy between method A and B is given by a mean distance of trajectories obtained by method A from trajectories obtained by method B. Note that this discrepancy measure is not symmetric, so we computed it for all possible 20 tuples of methods we used.

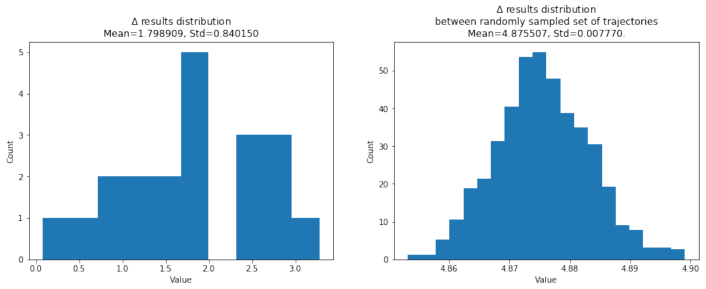

Figure 2. Distribution of Δ-discrepancy between all 20 possible tuples of different optimization methods we used (Genetic Algorithm, SGD, Nesterov, RMSProp and ADAM) and between randomly sampled sets of trajectories (we sampled $N = 1000$ pairs).

The results of Δ-discrepancy applied to all pairs of methods varied between 0.08 and 3.28 with the mean value equal to 1.8 and 0.84 standard deviation (see Figure 2). In order to confirm that such results could not be obtained by chance we sampled $N = 1000$ random pairs of sets of trajectories, which matched the shape of our data and computed Δ-discrepancy measures between them. The results varied between 4.85 and 4.9, with mean equal to 4.88 and 0.001 standard deviation. All statistical tests rejected hypothesis that such distribution of Δ-discrepancy might be obtained by chance. Thus, we conclude that all methods (both gradient based and genetic algorithm) produce results from roughly the same regions of input space.

4.1.2. Analysis of different regions close to minima

Based on Figure 1 we conclude that there are three main regions to which optimization algorithms converge. Two on the left have rather small errors. They have also small PCA-reconstruction error what suggests that they lie close in the space of signal settings. The region on the right seems to have much steeper error / simulation value structure. A greater reconstruction error suggests that this region lies far away from two regions on the center-right and is underrepresented in the trajectory space.

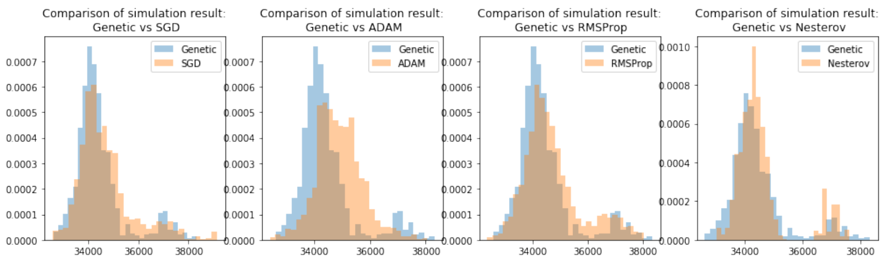

Figure 3. Distribution of simulation results across different methods.

4.2. Analysis of simulation values

Metric	Genetic	SGD	ADAM	RMSProp	Nesterov
Min	32620	32565	32406	32383	33026
Max	38320	39143	38062	38015	37602
Mean	34439	34682	34891	34731	34598
Median	34216	34420	34831	34490	34354
Std	1008	1095	817	1060	986

Table 1. Comparison of simulation results across different optimization methods.

As optimization processes make trajectories to settle in roughly the same regions - ground truth outcomes of simulations, results of predictions and errors should be similar. This is confirmed by analysis of simulations. Table 1 shows actual results and Figure 3 compares histograms of different methods.

Multiple statistics of distributions of results of different methods show that all distributions are quite similar. Analysis of histograms shows that most of methods produce a bi-modal distribution (which is explained in section 4.4.). The only distribution which differs significantly is a distribution of ADAM simulation values. This is caused by optimization process instability explained in section 3.

4.3. Analysis of errors

In case of errors of approximations (accuracy of metamodels) we noticed that Nesterov algorithm seems to be the most accurate. Moreover, in minima reached by that algorithm errors are only positive. The second best algorithm with respect to errors is Genetic Algorithm, which also suffers the least from extreme positive errors. Other gradient methods (RMSProp, ADAM and SGD) have similar mean, standard deviation for both error and absolute (ABS) error values.

Metric	Genetic	SGD	ADAM	RMSProp	Nesterov
Min	-1573	-1688	-731	-1549	-1826
Max	3326	4041	3657	3834	3814
Mean	1112	1363	1442	1432	905
Median	1206	1388	1435	1600	1010
Std	1033	1066	934	1018	1079
Min ABS	20	1	1	4	1
Max ABS	3326	4041	3657	3834	3814
Mean ABS	1294	1470	1460	1508	1171
Median ABS	1239	1393	1435	1600	1115
Std ABS	792	912	905	899	782

Table 2. Comparison of errors (differences between predictions of models and values from simulations) across different optimization methods. ABS means *absolute error*.

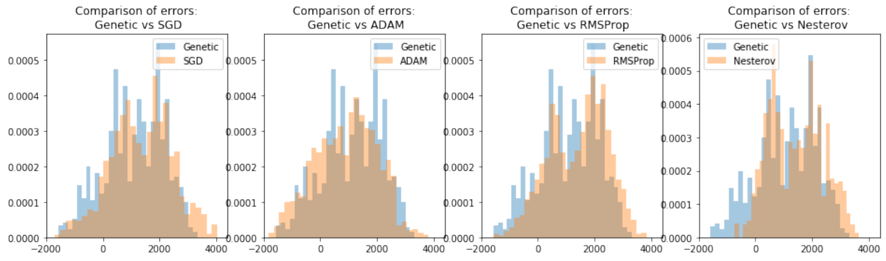

Figure 4. Distribution of errors across different methods.

4.4. Comparison of activation functions

The most important factor which made the optimization trajectories different was the activation function. The experiments which used models with the same activation (ReLU or TANH) were similar (even if a different optimization method was used) whereas different activations usually resulted in completely different minima (Figure 1). Only models based on TANH activation explored the less frequent region (see Section 4.1.2. and Figure 1) in the input space. This explains differences in errors and simulation values distributions, which seem to be bi-modal for TANH-based models. Although the TANH-based models have lower mean and median absolute errors - the possibility of reaching poor local minima is greater (what is confirmed by higher maximal value and standard deviation of simulation values).

Metric	ReLU Sim.	TANH Sim.	ReLU Error	TANH Error	ReLU ABS Error	TANH ABS Error
Min Value	32620	32383	-246	-1826	7	1
Max Value	36787	38320	3834	3198	3834	3198
Mean Value	34464	34883	1872	603	1872	866
Median	34416	34526	1887	524	1887	660
Std	546	1237	683	946	682	713

Table 3. Comparison of simulation results across different optimization methods.

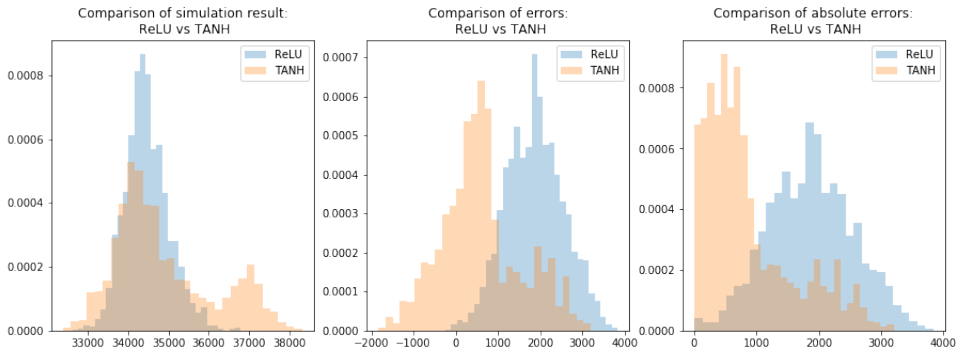

Figure 5. Distribution of simulation results across different methods.

5. Conclusions and future work

We investigated performance of a gradient descent optimization applied to the traffic signal setting problem and compared it to genetic algorithms. We discovered that both optimization methods produce similar results and showed that the accuracy of neural networks and reached minima depend on an activation function. Gradient optimization gives good results a few times faster than GA which makes it a natural candidate for the optimization procedure in our future experiments.

As a future work we plan to test other activation functions to see how they influence performance of optimization algorithms and investigate the influence of activation function on minima reached on multiple datasets.

6. References

[1] P. Gora and P. Pardel. Application of genetic algorithms and high-performance computing to the traffic signal setting problem. *24th International Workshop, CS&P 2015, Vol. 1", ISBN: 978-83-7996-181-8*, pages 146–157, 2015.

[2] Federal Highway Administ. *TRAFFIC SIGNAL TIMING MANUAL.* 2008.

[3] H. Prothmann. *Organic Traffic Control.* KIT Scientific Publishing, 2011.

[4] C. B. Yang and Y. J. Yeh. The model and properties of the traffic light problem. *Proc. of International Conference on Algorithms*, pages 19–26, 1996.

[5] S. Luke. *Essentials of Metaheuristics.* lulu.com, first edition, 2009. Available at http://cs.gmu.edu/~sean/books/metaheuristics/.

[6] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu. LightGBM: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems 30*, 2017.

[7] Y. Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. pages 61–70, 2011.

[8] D.F. Johansson, U. Shalit, and D. Sontag. Learning representations for counterfactual inference. *33rd International Conference on Machine Learning*, 2016.

[9] P. Gora and M. Bardoński. Training neural networks to approximate traffic simulation outcomes. *5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems, IEEE*, pages 889–894, 2017.

[10] P. Gora, M. Brzeski, M. Możejko, A. Klemenko, and A. Kochański. Investigating performance of neural networks and gradient boosting models approximating microscopic traffic simulations in traffic optimization tasks. *NIPS Workshop on Machine Learning for Intelligent Transportation Systems*, 2018.

[11] U. Jang, W. Xi, and S. Jha. Objective metrics and gradient descent algorithms for adversarial examples in machine learning. *Proceedings of the 33rd Annual Computer Security Applications Conference*, pages 262–277, 2017.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[13] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudk, editors, *AISTATS*, volume 15 of *JMLR Proceedings*, pages 315–323. JMLR.org, 2011.

[14] Systematic evaluation of convolution neural network advances on the imagenet. *Comput. Vis. Image Underst.*, 161(C):11–19, August 2017.

[15] P. Gora. Traffic Simulation Framework - a cellular automaton based tool for simulating and investigating real city traffic. *Recent Advances in Intelligent Information Systems, ISBN: 978-83-60434-59-8*, pages 641–653, 2009.

[16] Dataset used in experiments. `https://goo.gl/ytPRQg`, 2018.

[17] K. He, X. Zhang, and S. Ren. Deep residual learning for image recognition. 2015.

[18] S. loffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015.

[19] S. Ruder. An overview of gradient descent optimization algorithms. 2016.

[20] R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature. 405, pp. 947-951*, 2016.

[21] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks.

[22] F. et al. Chollet. Keras.

[23] M. et al Abadi. Tensorflow: Large-scale machine learning on heterogeneous systems.

[24] Settings of genetic algorithms. `https://goo.gl/G3YomX`, 2018.

[25] Y. E. Nesterov. A method for solving the convex programming problem with convergence rate $o(1/k^2)$. *Dokl. Akad. Nauk SSSR*, 269:543 – 547, 1983.

[26] J. Ba and D. P. Kingma. Adam: A method for stochastic optimization. *Proc. of ICLR*, 2015.