Neuroevolution for Transportation Applications

Toan V. Tran
fpy148@mocs.utc.edu
Center for Urban Informatics and Progress
University of Tennessee at Chattanooga
Chattanooga, TN, USA

Mina Sartipi
Mina-Sartipi@utc.edu
Center for Urban Informatics and Progress
University of Tennessee at Chattanooga
Chattanooga, TN, USA

ABSTRACT

This paper investigates training neural networks for transportation applications by Neuroevolution (NE), which is a competitive alternative to Gradient Descent (GD). We conduct experiments for two typical transportation tasks: traffic prediction (supervised learning) and traffic signal control (reinforcement learning). For traffic prediction, our proposed NE framework outperforms GD in most cases. To investigate the reason NE performs better than GD, we apply a visualization technique to the loss function. The visualization shows non-convex surfaces with many local minimums that cause challenges to GD. Finally, we show that NE can also train reinforcement learning agents for traffic signal control.

CCS CONCEPTS

• **Applied computing** → **Transportation**; • **Computing methodologies** → **Neural networks**; • **Mathematics of computing** → **Mathematical optimization**.

KEYWORDS

neuroevolution, training neural networks, transportation, traffic prediction, traffic signal control

ACM Reference Format:
Toan V. Tran and Mina Sartipi. 2022. Neuroevolution for Transportation Applications. In *Proceedings of The 11th International Workshop on Urban Computing (UrbComp '22).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Many transportation applications have benefited from the success of neural networks and deep learning. In general, there are three approaches: supervised learning, unsupervised learning, and reinforcement learning [15]: supervised learning solves prediction tasks, unsupervised learning detects anomaly and represent features, and reinforcement learning is the state-of-the-art method used in control tasks such as traffic signal control and automated driving. Due to the high demand of deep learning in transportation, any small improvement in neural-network fundamentals, such as the training process and network architecture, can bring a huge impact to this field.

UrbComp '22, August 15th, 2022, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Despite its importance, not many studies investigate the fundamental research for transportation related problems. For example, neural networks in transportation problems are typically trained by Gradient Descent (GD) and Back-propagation algorithms. However, GD has challenges such as gradient vanishing and exploding [17]. Another problem of GD is related to local minimums, which frequently appear in the loss surfaces of neural networks [19]. There are several ways to handle these issues, such as changing activation functions to ReLU, adjusting network architectures with skipping connections to avoid vanishing, and momentum for optimizers to escape local minimums. However, these solutions cannot fully address the issues since the root cause of the problems, i.e., the gradient descent strategy, remains unchanged.

This paper investigates an alternative to GD – Neuroevolution (NE), also known as Deep Neuroevolution, to train neural networks for transportation applications. Note that NE is sometimes known as a Neural Architecture Search (NAS) [30] that automatically generates network architectures. The scope of this paper is to perform NE/evolutionary algorithms for the training process. The main contributions of our work are as follows:

- To the best of our knowledge, this is the first paper that investigates Neuroevolution for transportation problems. We also conduct experiments for two common tasks: Supervised learning (traffic prediction) and reinforcement learning (traffic signal control).
- For traffic prediction, our proposed NE framework outperforms the traditional approach using GD. Moreover, we also demonstrate that NE is able to train neural networks for the traffic signal control task.
- We conduct extensive experiments to compare different NE algorithms and figure out the effects of hyperparameters. Further, we apply the state-of-the-art visualization technique to understand more about why NE outperforms GD and vice versa.

The rest of this paper is organized as follows: Section 2 and 3 present related works and the background of training neural networks, respectively. Section 4 describes the details of the Neuroevolution framework and algorithms proposed in this paper. Experiments and results are summarized in 5. Finally, Section 6 concludes the paper and gives an outlook on future work.

2 RELATED WORKS

2.1 Neural networks in transportation

As mentioned above, there are three main categories in machine learning: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning is widely applied to three common tasks: traffic flow, demand prediction, and accident prediction

[15]. To exploit the temporal dependency in these tasks, [6] utilized LSTM and GRU, while [16] proposed a method with Meta-RNN. Another approach takes advantage of the spatial dependency by using CNN [11, 28]. The state-of-the-art prediction method exploits both temporal and spatial features with RNN-based Graph Convolutional Networks (GCN) [5]. Unsupervised learning is mainly applied for feature extraction using AutoEncoder networks [13]. Finally, Reinforcement Learning (RL) is the state-of-the-art method for transportation control tasks such as traffic signal control [24] and automated driving in the scenario of connected vehicles [26]. Furthermore, all neural networks applied in this field are trained by gradient descent.

2.2 Neuroevolution

Neuroevolution (NE) is related to two problems: designing neural network architectures and training the networks. Designing the network architectures through NE, known as NAS, has achieved many breakthroughs (e.g., reinforcement-learning-enabled NAS [30] defeated humans in designing a computer-vision neural network architecture on CIFAR-10). Moreover, NAS has helped to create a compact but powerful architecture for edge devices [8]. In the context of transportation, [11] and [1] performed NAS to automatically search the suitable neural architectures for traffic flow and ridership prediction, respectively.

Neuroevolution has also demonstrated its effectiveness for the training task. In [22], the authors have shown that Genetic Algorithm is a competitive alternative for training reinforcement-learning agents. Moreover, [20] proposed a highly-scalable evolutionary strategy that rivaled the performance of standard RL (i.e., GD-based RL). The matrix-free evolution strategy proposed in [9] outperforms GD on the FashionMNIST dataset – a classification task. This is especially true with a library named EvoJax, which provides parallel running across GPUs/TPUs to significantly reduce the training time of NE approaches [23].

3 BACKGROUND

3.1 Modelling the training task

Training process, represented in Eq. (1), finds the values of trainable parameters W that minimize the loss function on the whole training data:

$$W = \underset{W}{\operatorname{argmin}} \left(\mathcal{L}(\text{net}(W, X), y) \right), \tag{1}$$

where $\mathcal{L}$ is the loss function, net is the forward of the neural network, X is the training data, y is the training label.

3.2 Training by Gradient Descent (GD)

Gradient Descent is an optimization algorithm used to find minimums of a given function. When training by GD, at every iteration, each trainable parameter takes a step in the opposite direction of its gradient. Eq. (2) illustrates an iteration of this training process.

$$w = w - \eta \cdot g\left(\frac{\partial \mathcal{L}(\text{net}(W, X_{batch}), y_{batch})}{\partial w} \right), \tag{2}$$

where $\frac{\partial \mathcal{L}}{\partial w}$ is gradient and calculated by backpropagation; w denotes trainable parameters; η is the learning rate; and g is the optimizer's policy. For example, Stochastic Gradient Descent (SGD) uses policy g that is exactly equal to the value of the gradient. There are also some improvement methods using momentum, such as Adam and RMSprop [19]. In addition, some studies apply recurrent neural networks to play a role as policy g [2, 25].

3.3 Training by Neuroevolution (NE)

Instead of iterating an update strategy for a unique model like GD, Neuroevolution conducts an evolutionary process for a population of models. The following section describes this training in detail.

4 METHODS

4.1 Neuroevolution Framework

Figure 1 presents the flow chart of the training process using neuroevolution. The first step is Initialization which initializes a population of models. After that, the evolutionary process is conducted for the whole population. More specifically, this process iterates a loop of 4 steps: Evaluation, Selection, Recombination, and Mutation. Evaluation is to calculate value of the objective function obtained by the current weights. For supervised-learning tasks, the objective function is common loss functions such as maximum likelihood or mean square error. Meanwhile, reinforcement-learning tasks evaluate the current weights/biases by the cumulative reward obtained after running an episode in the environment. The next step is Selection, which selects the best models (i.e., models that achieve the highest values of the objective function). These models are recombined to form the next generation of the population. Sequentially, the new generation is randomly mutated. This looping procedure is iterated until the objective is fully optimized. Moreover, various evolutionary algorithms have different ways to conduct these steps, which are explained in the next section.

4.2 Neuroevolution Algorithms

- ***GA*** – Genetic Algorithm [14]

  GA is inspired by the natural evolution and mimics biological operators such as selection, mutation, and crossover.
- ***OpenES*** – OpenAI Evolution Strategies [20]

  OpenES is presented in Algorithm 1. For each iteration, the population is mutated by adding a random noise into the current weight values and then updated based on the average fitness value.

Algorithm 1 OpenES algorithm

Input: learning rate η, noise standard deviation σ, initial weight W_0, population size P
for t in 1 to #iters **do**
  Sample $\epsilon_1, ..., \epsilon_P \sim \mathcal{N}(0, I)$,
  $f_i \leftarrow \text{Evaluate}(W_t + \sigma\epsilon_i)$ for $i = 1, ..., P$ ▷ Evaluation
  $W_{t+1} \leftarrow W_t + \eta \frac{1}{\sigma \cdot P} \sum_{j=1}^{P} f_j \epsilon_j$ ▷ Update
end for

- ***ARS*** – Augmented Random Search

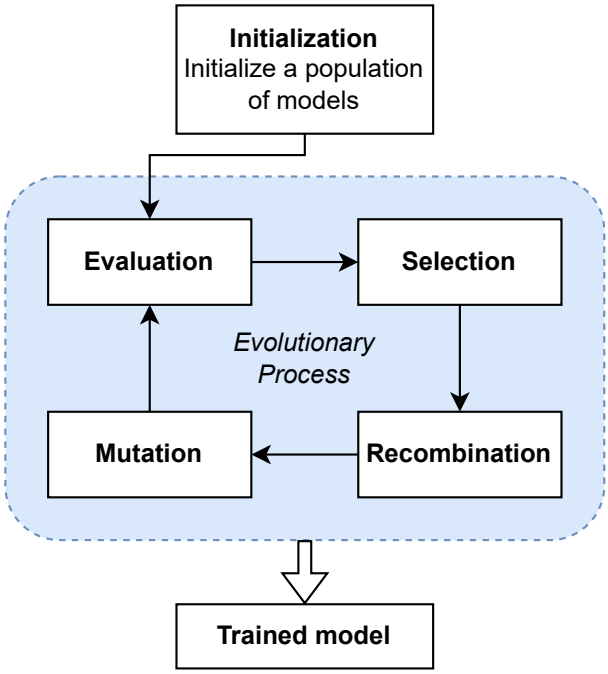

Figure 1: Flow chart of training neural network by NE

ARS is presented in Algorithm 2 and is considered as an improvement of OpenES. First, it considers both directions of samples instead of only one like OpenES. Second, ARS conducts Selection, while OpenES only calculates the average of all solutions. Because the goal is to maximize the collected fitness values, removing some bad solutions can be a judicious strategy. [12]

Algorithm 2 ARS algorithm

Input: learning rate η, noise standard deviation σ, initial weight W_0, population size P, number of top-performing solutions to use b
for t in 1 to #iters **do**
  Sample $\epsilon_1, ..., \epsilon_P$ with i.i.d. standard normal entries
  $f_i^{\pm} \leftarrow \text{Evaluate}(W_t \pm \sigma\epsilon_i)$ for $i = 1, ..., P$ ▷ Evaluation
  Sort solutions by $\max(f_i^+, f_i^-)$
  Select b top solutions $\epsilon_{1..b}$ ▷ Selection
  $W_{t+1} \leftarrow W_t + \eta \frac{1}{\sigma \cdot b} \sum_{j=1}^{b} \epsilon_j \left[f_j^+ - f_j^-\right]$ ▷ Update
end for

• ***CMA-ES***– Covariance Matrix Adaptation Evolution Strategy [7]
While OpenES and ARS sample from a static distribution, CMA-ES changes the distribution parameters during searching. Moreover, CMA-ES is introduced as a stochastic method for non-linear, non-convex, black-box optimization problems in continuous domains. Algorithm 3 simplifies the operation of CMA-ES. The key idea of CMA-ES is the maximum-likelihood principle. At every generation, the distribution is updated to maximize the likelihood of previously successful candidates.

Algorithm 3 CMA-ES algorithm

Input: population size P
Initialize: distribution parameters: $C = \mathbb{I}$, μ, σ
for t in 1 to #iters **do**
  for i in 1 to P **do**
    $W^i \sim \mathcal{N}(\mu, \sigma^2 C)$, ▷ Multivariate normal distribution
    $f_i \leftarrow \text{Evaluate}(W_i)$
  end for
  $W^{\{1...P\}} \leftarrow W^{s(1)...s(P)}$ ▷ Sort solutions based on f_i
  $\mu \leftarrow$ Update_mean $(W_1, ...W_P)$
  $C \leftarrow$ Update covariance matrix
  $\sigma \leftarrow$ Update standard deviation
end for

• ***PGPE*** – Policy Gradients with Parameter-based Exploration [21]
Algorithm 4 illustrates PGPE. More specifically, PGPE estimates a likelihood gradient by sampling directly in the parameter space. The parameters of the distribution are updated during searching, which is similar to CMA-ES.

Algorithm 4 PGPE algorithm

Input: population size P
Initialize: distribution parameters: μ, σ
for t in 1 to #iters **do**
  for i in 1 to P **do**
    Sample $W^i \sim \mathcal{N}(\mu, \sigma^2 \mathbb{I})$
    $f_i \leftarrow \text{Evaluate}(W^i)$
  end for
  $\boldsymbol{T} = [t_{ij}]_{ij}$ with $t_{ij} := \left(W_i^j - \mu_i\right)$
  $\boldsymbol{S} = [s_{ij}]_{ij}$ with $s_{ij} := \frac{t_{ij}^2 - \sigma_i^2}{\sigma_i}$
  $\boldsymbol{r} = [(f_1 - b), ..., (f_P - b)]^{\mathrm{T}}$
  Update $\mu = \mu + \eta \boldsymbol{T}\boldsymbol{r}$
  Update $\sigma = \sigma + \eta \boldsymbol{S}\boldsymbol{r}$
  Update baseline b accordingly
end for

5 EXPERIMENTS AND RESULTS

General setting

Computing and Software: Our experiments are conducted by a machine with the Intel Xeon Gold 6140 CPU and four Nvidia Tesla 32GB V100s. The gradient-descent approach is developed by Keras and the neuroevolution is implemented by EvoJax [23], which is a Jax-based [3] hardware-accelerated toolkit for parallel running across multiple TPU/GPUs. Our source code is publicly accessible at *https://github.com/toan-tv/NE-for-Transportation*.

5.1 Supervised learning – Traffic Prediction

Dataset: In our experiments, we use the PeMS dataset which has been collected from 11,160 detectors in major metropolitan areas of California [4]. Due to the heavy computing demand, we randomly chose 50 sensors and picked data over the course of four weeks for our traffic prediction experiments. We use data from the first

three weeks for training and data from the last week for testing. In addition, we aggregate the original 30-second data into 5-minute flows by calculating average values.

Baseline: For a fair comparison, we implement the same neural network architecture proposed in [6], for both the neuroevolution and gradient-based approaches. The network consists of a LSTM layer with 32 hidden units and a fully connected layer with 32 units. In total, the model has 6,081 trainable parameters. The activation function is ReLU and the optimizer is RMSprop with the default learning rate of 0.001. The training and testing data are the same, the only difference being the training process, i.e., gradient descent and neuroevolution. Since patterns in time-series data of various sensors are different, we train a separate model for each sensor. Each model is trained with 500 epochs and the best model is saved during training.

• **Algorithm effects**. In this experiment, we implement various evolutionary algorithms for training LSTM networks to measure the influence of algorithms. To fairly compare, we use identical hyperparameters.

Figure 2 presents the average MSE over iterations. The baseline is the best model found by GD. In general, all NE methods except GA outperform GD on the training loss. Among NE algorithms, ARS and PGPE converge fastest after 40K iterations and also reach the lowest training loss. Table 1 enumerates the number of time series where the methods achieve the best performance. PGPE is the best method, with 18 time series getting the lowest testing MSE. Meanwhile, GD outperforms NE methods on only 1 time series (i.e., 2% of the total dataset). Although the average training loss of ARS is better than CMA-ES, CMA-ES obtains the best testing MSE for 16 time series, compared to 8 time series by ARS. That means ARS is not good at generalization and gets overfitted results. Figure 3 shows detailed MSEs on the testing dataset of the selected time series. Moreover, Figures 4 illustrates the testing loss curves of Sensors 2 and 11. While CMA-ES achives the best result for Sensor 2, PGPE and ARS outperform other methods in case of Sensor 11. Furthermore, Table 2 presents the average training time for each timeseries of all methods. Generally, GD's training time is shorter than all NE's.

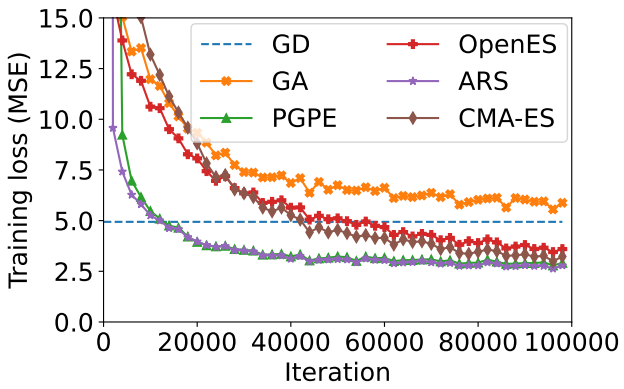

Figure 2: Average MSE of 50 time series during training

Method	GD	GA	**PGPE**	OpenES	ARS	CMA-ES
#TimeSeries	1	5	**18**	2	8	16

Table 1: Number of time series that methods achieve the best performance on the testing data

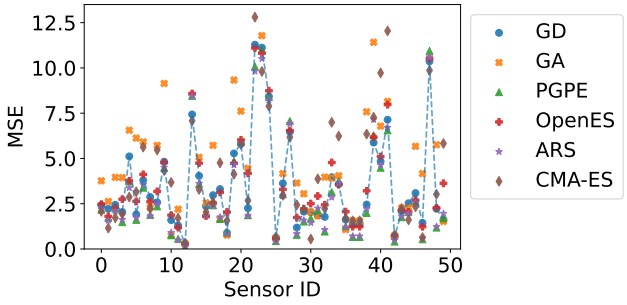

Figure 3: MSE on the testing data of 50 selected time series

Method	Total training time (s)	One-iteration training time (ms)
GD	**89.24**	**1.89**
GA	332.08	3.32
PGPE	228.47	2.28
OpenES	272.75	2.73
ARS	265.16	2.65
CMA-ES	244.76	2.45

Table 2: Average training time for each time series

• **Population size effects**. In this experiment, we train models with different settings for the population size. All other hyperparameters are identical. We only investigate PGPE, which obtains the best results among those NE algorithms.

Figure 5 presents the performance of PGPE while varying the population size from 64 to 2048. All settings with different population sizes outperform GD. Surprisingly, the population setting of 512 is better than the bigger populations of 1024 and 2048. The best setting, i.e., 512, reduces the training by around 50%. Moreover, our population sizes are considered quite small for this search space of $\mathbb{R}^{6081}$. For example, a population size of 200 is used for the search space of $\mathbb{R}^3$ in [18]. Normally, a larger population provides a better global ability to avoid local minimums. However, a huge population is challenging to the current computing capacity.

• **Learning rate effects**. In this experiment, we vary the learning rate while keeping other hyperparameters identical.

Figure 6 illustrates the PGPE's performance in different settings of the learning rate. Generally, a lower learning rate provides better results for both the loss value and convergence time. Furthermore, the learning rate of 0.1 obtains an unstable result and can not outperform GD. Meanwhile, the learning rates of 0.01 and 0.001 have better performance than the baseline. Notably, the setting with the learning rate of 0.001 outperforms the baseline by around 35% on the training loss.

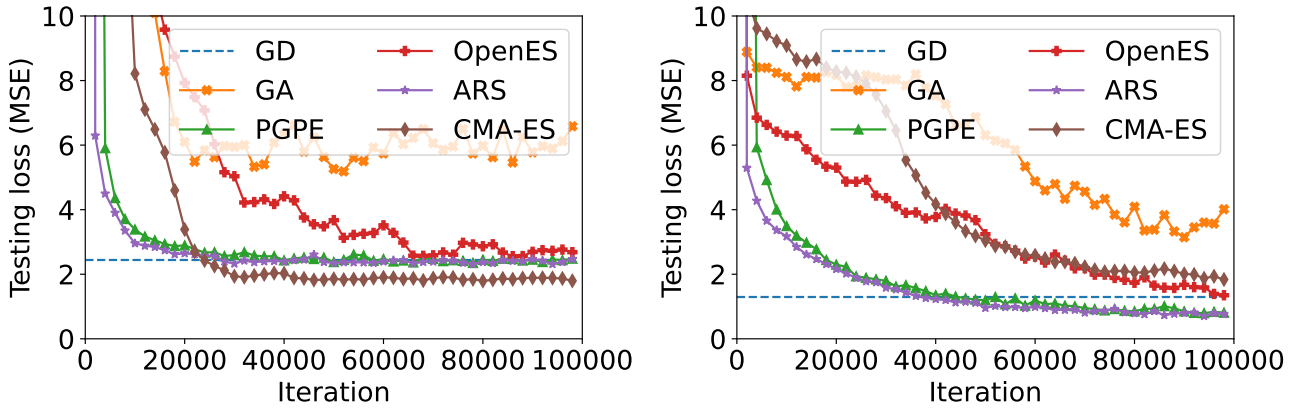

Figure 4: Testing loss curves of Sensor 2 and 11 over iterations

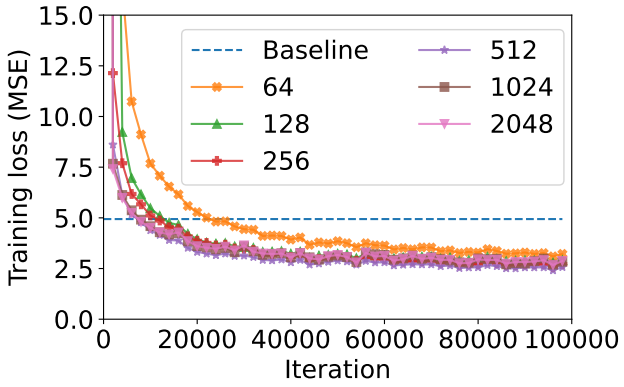

Figure 5: PGPE's performance with various population sizes

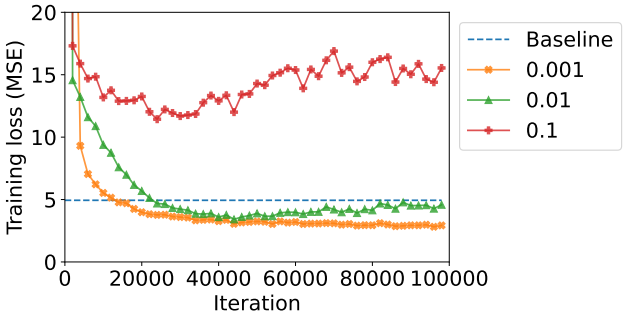

Figure 6: PGPE's performance with various learning rates

• **An interesting visualization**. We use filter-wise normalization, proposed in [10], to visualize loss surfaces of the LSTM networks for traffic prediction. Note that existing state-of-the-art neural-network-surface visualizations can only express a part of the characteristics of the comprehensive surface because the loss function is extremely complicated with thousand variables/trainable parameters. Although a comprehensive visualization for all parameters is still a gap in research, existing visualizations can help us make comparisons across disparate network architectures and training data. In the context of this paper, we focus on comparing loss surfaces of the same network architecture but for different time series. Figures 7 and 8 present the loss surfaces of the time series of Sensors 04 and 39, respectively. As shown in Figure 3, NE outperforms GD on Sensor 04. On the other hand, GD is better than NE for Sensor 39. Moreover, Figures 9 and 10 present the contour lines when the loss value is less than 50. Generally, the loss function of Sensor 04 is non-convex with many local minimums, while the one of Sensor 39 is quite convex. Because GD can not guarantee a global-optimal convergence [19], such a loss function with multiple local minimums is challenging for GD. In contrast, NE is flexible to escape local minimums. Moreover, in the convex case, i.e., Sensor 39, GD is better than NE because GD can easily reach the global minimum.

Our visualization shows that despite the same problem, i.e., traffic prediction, various time series have different patterns. Therefore, loss surfaces for these time series are not similar. Furthermore, most surfaces are non-convex, where NE outperforms GD.

5.2 Reinforcement learning – Signal Control

Jax-based Simulation: Because of the current limitation of the library, i.e., EvoJax, all environments have to be written on Jax and executed on GPUs/TPUs instead of CPUs. Therefore, existing transportation simulators have not been able to play with EvoJax yet. To deal with this problem, we implement a new Jax-based simulation.

Initially, any traffic simulation requires a key component – a car-following model. This model determines the speed of vehicles based on their leaders. In our simulation, we implement the collision-avoiding car-following model [27]. The speed of a vehicle can be calculated as Eq. (3).

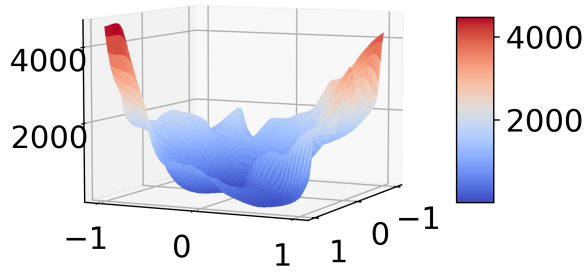

Figure 7: The loss surface of Sensor 04

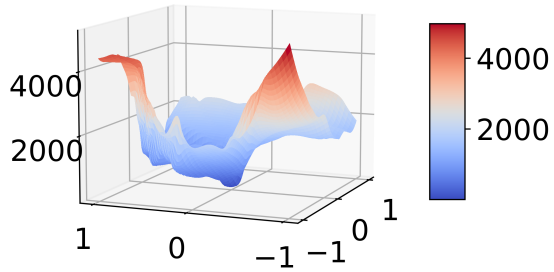

Figure 8: The loss surface of Sensor 39

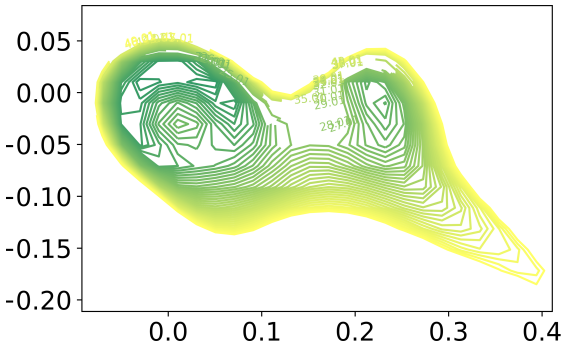

Figure 9: The contour-line visualization for the loss surface of Sensor 04, in which NE outperforms GD

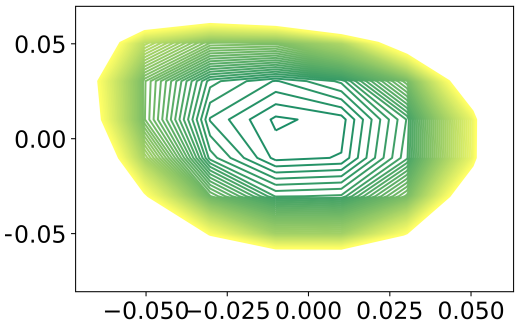

Figure 10: The contour-line visualization for the loss surface of Sensor 39, in which GD is better than NE

$$
\begin{aligned}
c &= \frac{v \cdot \Delta t}{2} \frac{v_L^2}{2 \cdot d_L} - gap \\
a &= \frac{1}{2 \cdot d} \\
b &= \frac{\Delta t}{2} \\
s &= \frac{-b + \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a},
\end{aligned} \tag{3}
$$

where v and v_L are speeds of the vehicle and its leader, respectively; Δt is the interval duration; gap is the distance between these two vehicles; and d and d_L is the maximum deceleration of the vehicle and its leader. In addition to the car following model, vehicles have to follow speed constraints including vehicle-type maximum speed and road speed limit. Moreover, vehicles are affected by traffic lights and intersections. To simplify these effects, we assume:

- Vehicles accelerate and decelerate at their maximum capacity
- When traffic lights switch to yellow and red signals, vehicles will stop if it is possible depending on their current speed, distance, and maximum deceleration. Otherwise, they will pass with normal behaviors.

To validate the functionality of our simulation, we conduct the same experiment of an isolated intersection in both SUMO (which is a well-known microscopic traffic simulation) and our simulation. We set up identical scenarios, including traffic demand and light setting. Generally, the difference between the two simulations is slight. Table 3 presents the average travel time of vehicles in SUMO and our simulation.

Vehicles/Hour	100	200	300	400
SUMO	38.94	35.40	35.67	35.85
Jax-based SIM	40.34	35.26	35.45	35.98
Difference	3.60%	0.40%	0.62%	0.36%

Table 3: Average travel time in SUMO and our simulation in different scenarios

Simulation setting: We implement an isolated intersection with a synthetic traffic demand of 400 vehicles per hour. The simulation executes a 15-minute scenario. The traffic light has the standard 4-phase setting.

Reinforcement learning design: We implement the RL agent that is proposed in [29]. There are three main components of the RL agent: State, Action, and Reward. More specifically, State is a vector of numbers of vehicles on approaching lanes. Meanwhile, Action is the phase index that will be executed in the next interval. The last component, i.e., Reward, is the negative of the total queue length of approaching lanes. For the neural network architecture, we implement a fully-connected one with two hidden layers, 32 units each. In total, we have 1408 trainable parameters.

Result: Figure 11 illustrates the result of training the RL agent by PGPE. In general, the agent is better over time and the cumulative reward is converged. That means NE is able to train RL agents for traffic signal control. Moreover, the curve presents a local minimum that PGPE successfully escapes at around Iteration 150. Surprisingly, this RL task converges much faster than the traffic prediction task. The reason could be the number of parameters. Although the number of trainable parameters of the traffic prediction model is only 4.25 times the traffic signal control's, the search spaces are exponentially different (i.e., $\mathbb{R}^{6081}$ vs. $\mathbb{R}^{1408}$). Due to the exponential growth of search spaces when the number of trainable parameters increases, NE may require the huge number of iterations.

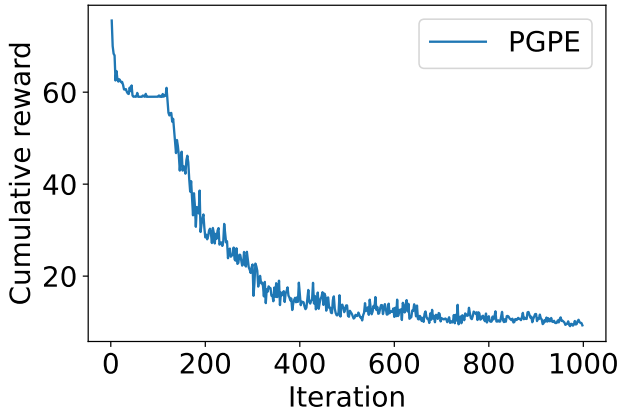

Figure 11: The performance of Neuroevolution for training a RL agent for traffic signal control

6 CONCLUSION

We have presented the proposed method for training neural networks by Neuroevolution. The proposed framework outperforms the gradient-descent-based learning method by up to 50% for the traffic prediction task. We also provided the visualizations of the loss surfaces so that we could understand why NE is better than GD in some cases and vice versa. Finally, we also showed that NE is possible for training reinforcement learning agents for traffic signal control. Although the NE framework is highly scalable, our settings of the population size are still relatively small for the vast search spaces of training neural networks. Therefore, in the future, when we have enough computing resources, we will investigate more about the functionality of larger populations. Furthermore, recent advances of graph neural networks are also an exciting topic. Training graph neural networks by NE is still a gap in research that we want to investigate.

7 ACKNOWLEDGMENTS

This material is based upon work partially supported by the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy (EERE) under the Award Number DE-EE0009208. This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

REFERENCES

[1] Ayman Afiya, Martinez Juan, Pugliese Philip, Dubey Abhishek, and Laszka Aron. 2022. Neural Architecture and Feature Search for Predicting the Ridership of Public Transportation Routes. In *Proceedings of the 8th IEEE International Conference on Smart Computing (SMARTCOMP' 22)*.

[2] Marcin Andrychowicz, Misha Denil, Sergio Gómez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. 2016. Learning to learn by gradient descent by gradient descent. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS' 16)*.

[3] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. *JAX: composable transformations of Python+NumPy programs.* http://github.com/google/jax

[4] Tom Choe, Alexander Skabardonis, and Pravin Varaiya. 2002. Freeway Performance Measurement System: Operational Analysis Tool. *Transportation Research Record* 1811, 1 (2002), 67–75.

[5] Xiaomin Fang, Jizhou Huang, Fan Wang, Lingke Zeng, Haijin Liang, and Haifeng Wang. 2020. ConSTGAT: Contextual Spatial-Temporal Graph Attention Network for Travel Time Estimation at Baidu Maps. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD' 20)*.

[6] Rui Fu, Zuo Zhang, and Li Li. 2016. Using LSTM and GRU neural network methods for traffic flow prediction. In *Proceedings of 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*. https://doi.org/10.1109/YAC.2016.7804912

[7] Nikolaus Hansen. 2006. The CMA Evolution Strategy: A Comparing Review. In *Towards a New Evolutionary Computation.*

[8] Andrew G. Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. 2019. Searching for MobileNetV3. *Proceedings of IEEE/CVF International Conference on Computer Vision (ICCV' 19)*.

[9] Dariusz Jagodziński, Łukasz Neumann, and Paweł Zawistowski. 2021. Deep Neuroevolution: Training Neural Networks Using a Matrix-Free Evolution Strategy. In *Neural Information Processing*, Teddy Mantoro, Minho Lee, Media Anugerah Ayu, Kok Wai Wong, and Achmad Nizar Hidayanto (Eds.). Springer International Publishing, Cham, 524–536.

[10] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. 2018. Visualizing the Loss Landscape of Neural Nets. In *Proceedings of Neural Information Processing Systems (NeurIPS' 18)*.

[11] Ting Li, Junbo Zhang, Kainan Bao, Yuxuan Liang, Yexin Li, and Yu Zheng. 2020. AutoST: Efficient Neural Architecture Search for Spatio-Temporal Prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '20)*. https://doi.org/10.1145/3394486.3403122

[12] Horia Mania, Aurelia Guy, and Benjamin Recht. 2018. Simple random search of static linear policies is competitive for reinforcement learning. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS' 18)*.

[13] Christos Markos and James J.Q. Yu. 2020. Unsupervised Deep Learning for GPS-Based Transportation Mode Identification. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC' 20)*. 1–6. https://doi.org/10.1109/ITSC45102.2020.9294673

[14] Tom V. Mathew. [n.d.]. Genetic Algorithm.

[15] Hoang Nguyen, Minh Kieu, Tao Wen, and Chen Cai. 2018. Deep learning methods in transportation domain: A review. *IET Intelligent Transport Systems* 12 (07 2018). https://doi.org/10.1049/iet-its.2018.0064

[16] Zheyi Pan, Yuxuan Liang, Weifeng Wang, Yong Yu, Yu Zheng, and Junbo Zhang. 2019. Urban Traffic Prediction from Spatio-Temporal Data Using Deep Meta Learning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '19)*.

[17] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the Difficulty of Training Recurrent Neural Networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning (ICML' 13)*.

[18] Olympia Roeva, Stefka Fidanova, and Marcin Paprzycki. 2013. Influence of the population size on the genetic algorithm performance in case of cultivation process modelling. In *2013 Federated Conference on Computer Science and Information Systems*. 371–376.

[19] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. https://doi.org/10.48550/ARXIV.1609.04747

[20] Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. 2017. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *ArXiv* abs/1703.03864 (2017).

[21] Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. 2008. Policy Gradients with Parameter-Based Exploration for Control. In *Proceedings of the 18th International Conference on Artificial Neural Networks (ICANN)*. https://doi.org/10.1007/978-3-540-87536-9_40

[22] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. 2017. Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning. *CoRR* abs/1712.06567 (2017). arXiv:1712.06567 http://arxiv.org/abs/1712.06567

[23] Yujin Tang, Yingtao Tian, and David Ha. 2022. EvoJAX: Hardware-Accelerated Neuroevolution. In *Proceedings of 2022 Genetic and Evolutionary Computation Conference (GECCO' 22)*.

[24] Hua Wei, Guanjie Zheng, Vikash Gayah, and Zhenhui Li. 2021. Recent Advances in Reinforcement Learning for Traffic Signal Control: A Survey of Models and Evaluation. *SIGKDD Explorer Newsletter* 22, 2 (2021). https://doi.org/10.1145/3447556.3447565

[25] Olga Wichrowska, Niru Maheswaranathan, Matthew W. Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein. 2017. Learned Optimizers That Scale and Generalize. In *Proceedings of the 34th International Conference on Machine Learning (ICML' 17)*.

[26] Zhongxia Yan, Abdul Rahman Kreidieh, Eugene Vinitsky, Alexandre M. Bayen, and Cathy Wu. 2022. Unified Automatic Control of Vehicular Systems With Reinforcement Learning. *IEEE Transactions on Automation Science and Engineering* (2022). https://doi.org/10.1109/TASE.2022.3168621

[27] Huichu Zhang, Siyuan Feng, Chang Liu, Yaoyao Ding, Yichen Zhu, Zihan Zhou, Weinan Zhang, Yong Yu, Haiming Jin, and Zhenhui Jessie Li. 2019. CityFlow: A Multi-Agent Reinforcement Learning Environment for Large Scale City Traffic Scenario. In *Proceedings of The World Wide Web Conference (WWW' 19)*.

[28] Junbo Zhang, Yu Zheng, Junkai Sun, and Dekang Qi. 2020. Flow Prediction in Spatio-Temporal Networks Based on Multitask Deep Learning. *IEEE Transactions on Knowledge and Data Engineering* 32, 3 (2020), 468–478. https://doi.org/10.1109/TKDE.2019.2891537

[29] Guanjie Zheng, Xinshi Zang, Nan Xu, Hua Wei, Zhengyao Yu, Vikash Gayah, Kai Xu, and Zhenhui Li. 2019. Diagnosing Reinforcement Learning for Traffic Signal Control. https://doi.org/10.48550/ARXIV.1905.04716

[30] Barret Zoph and Quoc V. Le. 2017. Neural Architecture Search with Reinforcement Learning. In *Proceedings of International Conference on Learning Representations (ICLR' 17)*.