2018 IEEE Intelligent Vehicles Symposium (IV)
Changshu, Suzhou, China, June 26-30, 2018

Training RNN simulated vehicle controllers using the SVD and evolutionary algorithms

Daniel K. McNeill
Dipartimento di Ingegneria Elettrica e dell'Informazione (DEI)
Politecnico di Bari
Bari, Italy
danmcne@gmail.com

***Abstract*—We describe an approach to creating a controller for The Open Car Racing Simulator (TORCS), based on The Simulated Car Racing Championship (SCRC) client, using unsupervised evolutionary learning for recurrent neural networks. Our method of training the recurrent neural network controllers relies on combining the components of the singular value decomposition of two different neural network connection matrices.**

***Index Terms*—Neural networks, Evolutionary computation, Autonomous vehicles**

I. Introduction

A. Motivation

It is well-known that recurrent neural networks are useful for learning sequences or analyzing time-series data, for this reason we have chosen to use recurrent networks as the controllers for simulated vehicles. It is also well-known that recurrent neural networks are difficult to train, especially via gradient methods, i.e. back-propagation through time, due to the exploding/vanishing gradient problem — we avoid this problem with an evolutionary training algorithm making use of the singular value decomposition.

B. Related Work

Developing self-driving control systems with realistic actions can be a difficult task — and one which has received much study both in simulation and in real autonomous vehicles. First, we focus on those works which have used the TORCS/SCRC [16], [29] system to have the most direct comparison to our control mechanism.

The Simulated Car Racing Championship (SCRC) has been held in 2007, 2009, 2010, 2011, 2012, 2013, and 2015 [16], [17].

Butz & Lönneker [6] have developed an effective TORCS/SCRC controller dubbed COBOSTAR. Butz, Linhardt & Lönneker [5] have subsequently developed this controller further.

Cardamone, Loiacono & Lanzi [7], [8] have applied on-line neuroevolution to the problem of developing a controller for TORCS/SCRC.

Muñoz, Gutierrez & Sanchis [19] have developed a "human-like" TORCS controller for the SCRC, as well as a controller created by imitation of hand-coded controllers and human play [18] and considered a multi-objective optimization of their controller via evolutionary algorithms [20].

Yee & Teo [30] have developed evolutionary spiking neural networks as controllers for the SCRC/TORCS.

Preuss, Quadflieg & Rudolph [23] have considered TORCS/SCRC sensor noise removal and multi-objective track selection for adapting the driving style of a controller.

Botta, Gautieri, Loiacono & Lanzi [4] have considered the problem of evolving the optimal racing line for a track in the TORCS simulator.

Athanasiadis, Galanopoulos & Tefas [2] have developed a progressive neural network and a training methodology for it for TORCS.

Quadflieg, Rudolph & Preuss [25] have examined the difficulty of optimizing parameters for a controller to perform well on many tracks — and the trade-offs inherent to that attempt.

In [21], Musoles considers using only visual input from TORCS to train a convolutional neural network controller — his work indicating that sensors in addition to the visual may be useful, acceleration being particularly singled out.

The list of papers regarding control of real-world autonomous vehicles is too numerous to list in its entirety. We note a few which have considered neural network or similar controllers for some aspect of the vehicle. Pérez, Milanés & Onieva [22] have considered cascade architectures applied to lateral control of real autonomous vehicles. Deep neural networks have been used by Li, Mei, Prokhorov & Tao [15] for structural prediction and lane detection in traffic scenes.

Concerning evolutionary algorithms applied to recurrent neural networks, we can point to Blanco, Delgado & Pegalajar [3], while the more recent paper of Salimans, Ho, Chen & Sutskever [26] considers evolution strategies as a scalable alternative to reinforcement learning.

The singular value decomposition has been used in the training of neural networks by Abid, Fnaiech & Najim [1] and by Psichogios & Ungar [24] — specifically, in relation to pruning extraneous hidden neurons in feed-forward networks.

Teoh, Tan & Xiang [28] and Santos, Barreto & Medeiros [27] have applied the singular value decomposition to estimating the number of hidden neurons in a feed-forward network.

Huynh & Won [12] have used the singular value decomposition for training single hidden layer feed-forward networks, and a recent paper of Fontenla-Romero, Pérez-Sánchez & Guijarro-Berdiñas [11] develops a non-iterative method for training single hidden layer feed-forward neural networks based on the singular value decomposition.

Kanjilal, Dey & Banerjee [14] have developed reduced size feed-forward neural networks using the singular value decomposition and subset selection.

Jia [13] has considered training feed-forward neural networks with the constraint that the singular values of the connection matrix be bounded near 1, i.e. the connection matrix is nearly orthogonal, and achieved state-of-the-art results on various benchmarks.

Cox [9] has considered parameter compression in recurrent neural networks using the singular value decomposition.

C. *TORCS, SCRC and SnakeOil*

Our current work focuses on using a Python implementation of a recurrent neural network as a controller for a car as in the Simulated Car Racing Championship (SCRC) [16] — which is built atop The Open Racing Car Simulator (TORCS) [29]. We use SnakeOil [10] as a client for SCRC to maintain focus on developing a controller for the car and to allow the use of Python.

TORCS/SCRC has a number of features that lend it to researching self-driving vehicles and video game AIs. First, TORCS is a quite realistic simulator with a sophisticated physics engine taking into account many aspects of driving such as collisions and traction under various conditions of wheel spin and acceleration. TORCS also provides many different tracks, though we restrict ourselves to only one here.

Fig. 1. Screenshot of TORCS

The SCRC modification of TORCS provides a collection of simple sensors for observing the local environment at small time steps (with the option that the sensors are noisy), a standardized racing car and a real-time client/server modularity — allowing one to develop the car controller without learning a great deal about the functionality of TORCS as well as forcing one to use only the given local sensor information and to develop a controller capable of operating in real-time.

We note that though the sensors provided by SCRC are few, have relatively low precision (particularly with regard to the number of "range to edge of track" measurements) and are at a relatively high level of abstraction (again, "range to edge of track" being a good example), controllers have been developed which are considerably better than the author (an experienced driver but novice gamer). We note also that the abstraction "range to edge of track" could be built into a module coming before a network such as ours in the control sequence.

SnakeOil simply implements the client in Python — allowing us to focus exclusively on developing the controllers in our language of choice.

II. Experiment

A. *Design of neural networks*

Our neural networks are single hidden layer fully recurrent neural networks, of varying numbers of neurons, of which 21 are inputs:

1) Damage,
2-10) Distance to edge of track at angles ± 1, ± 4, ± 12, ± 33, and ± 88 degrees from forward,
12) Race position,
13) X (Longitudinal) Acceleration,
14) Y (Transverse) Acceleration,
15) Z Acceleration,
16) Distance to opponent to back-left,
17) Distance to opponent to front-left,
18) Distance to opponent to front,
19) Distance to opponent to front-right,
20) Distance to opponent to back-right,
21) Kinetic energy,

three are outputs:

1) Steering,
2) Acceleration,
3) Braking,

and the balance (between 3 and 16 neurons) are hidden. We note that the track distance ranges were affected by i.i.d. normal noises with standard deviation equal to 10% of the sensor's range, while the "raw" SCRC distance to opponent sensors are affected by 2% i.i.d. i.i.d. normal noise — our opponent sensors are averages of up to 10 "raw" sensors.

Given the nature of the problem, driving a car, and the sensors provided, including distances to the edge of the track and opponents, as described above, we initialize several connections directly between input and output to random values with what we consider to be the correct sign. For example, if a car is detected to the left, we expect to have some inclination to steer to the right.

We then fully connect the inputs to the hidden neurons, the hidden neurons to the output neurons and the hidden neurons to other hidden neurons — with uniformly random weights in $[-1, 1]$. We forbid self-connection at this stage.

Our networks have something like the topology in Fig. 2 at this stage — except that we have 21 inputs, three outputs and varying numbers of hidden neurons. That is, we have full connections from the input to the hidden layer, full connections from the hidden layer to the output layer, full connections between the hidden neurons, and some chosen direct connections from the input layer to the output layer.

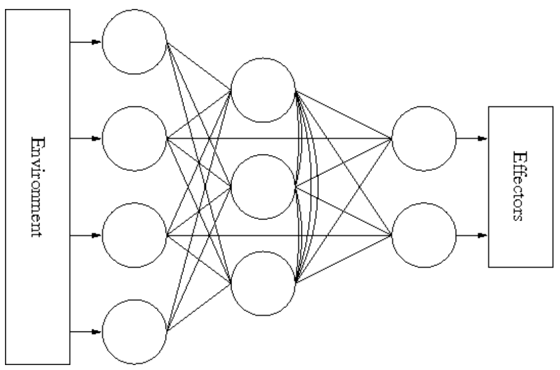

Fig. 2. Scheme of neural network (not depicting numbers of neurons)

1) The Singular Value Decomposition: Every complex (real) matrix M has a decomposition into a product of three matrices

$$M = U\Sigma V^*$$

where U and V are unitary (orthogonal) matrices, V^* is the conjugate transpose (transpose, in the case of real M) of V, and Σ is a diagonal matrix with non-increasing non-negative entries on the main diagonal.

The diagonal entries of Σ are called the singular values of M, while the columns of U and V are called the left-singular vectors and right-singular vectors of M, respectively.

The singular value decomposition (SVD) may be thought of as a generalisation of the eigenvalue decomposition. In fact, the left-singular vectors of M (the columns of U) are the set of orthonormal eigenvectors of MM^*, while the right-singular vectors (the columns of V) are the orthonormal eigenvectors of M^*M. Finally, the non-zero singular values of M are the square roots of the non-zero eigenvalues of both M^*M and MM^*.

2) The SVD in our construction: Given the connection matrix M — that is, the matrix of connection weights from neuron n_i to neuron n_j, where the ij^{th} entry of M is the weight of the connection from n_i to n_j — we next perform the singular value decomposition:

$$M = U\Sigma V^*,$$

where U and V are orthogonal matrices and Σ is the diagonal matrix of singular values in descending order. We maintain U and V as they are in the decomposition, but at this point condition the final connection matrix by creating an entirely new Σ_1 with values near 1 — giving

$$M_1 = U\Sigma_1 V^*.$$

This has the effect of preserving the "structure" of the neural network, but normalizing the weights so that no particular response to input is much stronger than any other. We note that our rule against self-connection is often broken here, but with a connection having negative weight. Notice also that at this stage we may even introduce connections into input neurons from other neurons (which are ignored in our implementation) and connections from output neurons to other neurons.

Our activation function is $\tanh$ — this was chosen because most inputs and outputs are bounded in $[0, 1]$, except steering which is bounded in $[-1, 1]$.

B. Design of SVD evolutionary algorithm

When we consider the singular value decomposition of a neural network connection matrix M:

$$M = U\Sigma V^*,$$

we note that the matrix has been decomposed into essentially two parts, one which is "structural" — that is, it determines the direction of the next neural state vector, and the other which is "weighty" — determining only the magnitude of each neural network state. More specifically, U and V^* taken together determine the "structure" of the network — any network state will be mapped to a network state vector in the same direction by both M and UV^*, only their magnitudes will differ. In fact, UV^*, being an orthonormal matrix, will not change the magnitude of a given state. On the other hand, the diagonal matrix Σ determines only the magnitude of the next state.

This allows us to divide neural network learning into two types: "structural" — that is, what qualitative responses to input are correct, and "weight" — that is, what (relative) level/importance is each of those responses to be? Here we develop an evolutionary algorithm to capitalize on the differences of these two types of learning.

Given "father" and "mother" networks from the previous generation, a new neural network connection matrix for the next generation is obtained by combining their singular value decompositions. If

$$M = U_m \Sigma_m V_m^*$$

is the singular value decomposition of the "mother" matrix,

$$F = U_f \Sigma_f V_f^*$$

the decomposition of the "father" matrix, P and Q orthonormal perturbations of the identity matrix, and S a perturbation of a diagonal matrix with linearly decreasing entries from $1+\varepsilon$ to $1-\varepsilon$ (where ε decreases with later generations), then our offspring matrix has the form:

$$O = (U_m P)((\Sigma_m \Sigma_f)^{0.5} S)(V_f Q)^*, \tag{1}$$

where we interpret $(\Sigma_m \Sigma_f)^{0.5}$ to be the element-wise geometric mean of Σ_m and Σ_f. In fact, Equation 1 is nearly the singular value decomposition of O as $U_m P$ and $V_f Q$ are orthonormal and $((\Sigma_m \Sigma_f)^{0.5} S)$ is a diagonal matrix with positive entries — though they may no longer be in strictly decreasing order. Our perturbation S has the effect of generally increasing larger singular values and decreasing smaller singular values, but may not always do so since the perturbation may leave it with non-decreasing entries.

So, up to perturbation, the "structure" of the offspring network is determined partly by the "father" and partly by the "mother" — specifically, the left singular vectors are determined by the "mother" and the right singular vectors by the "father" — while the singular values are an average of the two, with some perturbation.

C. *Outline of experiment and implementation*

Our current implementation in Python consists of a multi-stage process.

The first is an initialization of a population of 1000 recurrent neural networks, with a set number of input and output and different numbers of hidden neurons — as described above.

Our evolutionary training method involves racing randomly chosen cohorts of five cars around the chosen track and recording their fitness — their distance traveled within the allotted time. We select up to about 30 of the neural networks which have attained the highest fitness as the parents for the next generation — excluding those networks which failed to pass the first turn.

We then "cross" each of the neural networks with each of the others (and with itself) as described above to produce the next generation of networks. We continue this process of testing and "breeding" for several generations — until the highest fitness of a generation seems to converge.

D. *Results*

Current results are promising, with the best networks after few generations performing better than the author (an experienced driver, but novice gamer) after training — and the best times being near or at the same level as reported by other authors (as in [2], [7], [8], [18], [21]) though direct comparison is impossible in the space provided given the diversity of tracks used by other authors, combined with the fact that controllers optimized for one particular set of tracks may perform sub-optimally on a different set.

We note that in preliminary testing it was relatively easy to find controllers which are able to complete a lap on all tested tracks — in contrast to [18]. (No doubt this is partly due to our "pre-programming" of the neural networks.) However, these controllers are far from optimal for any particular track. This suggests that one might be able to use the U and V^* for a controller able to complete all tracks and then optimize the network for each individual track, $\mathcal{T}$, by saving $\Sigma_{\mathcal{T}}$ (perhaps not in descending order, depending on which 'circuits,' that is inputs and responses, are more important for a given track) for each one.

We do notice some problems: First, if the car exits the roadway completely, it is very unlikely to return — this is partly due to the sensors we have equipped the car with not providing useful information off-track. We note that most entries in the SCRC maintained a special set of routines for bringing a car back onto the track, we have left this problem aside.

Second, contact with other vehicles (and walls) is fairly common, even with the best networks — this is surely due to the fitness function (the distance traveled) not punishing damage enough, and not punishing contact at all. In fact, with some runs of the experiment we have seen the evolution of hyper-aggressive controllers — pushing the other cars off the track, or causing more damage than one receives, can be evolutionarily advantageous under the conditions we have set up.

Finally, it is common for one car to become stuck behind a stalled car, especially from the starting position, this is simply because the stuck car doesn't have enough room to maneuver, and doesn't have access to a reverse gear. Ramming frequently occurs in this situation (and the behavior persists), again because distance traveled is more heavily weighted than damage. However, controllers don't usually get stuck or ram when there ample opportunity to avoid a stalled or slow car — when lapping the car, for example.

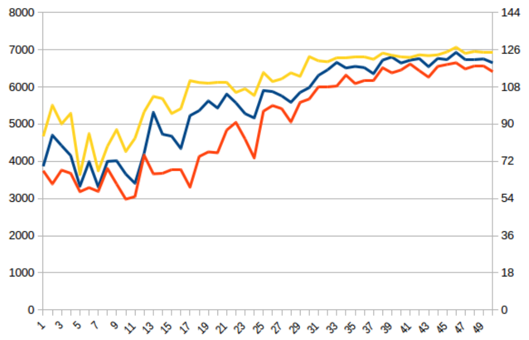

Fig. 3. Q1-Q2-Q3 of best fitness for each generation

Fig. 3 shows the median and interquartile fitness (distance traveled in 200 seconds) of the fittest controller of each generation over several runs on the track Wheel 1. We can see that the controllers start to converge to a top fitness between 6500 and 7000 meters — corresponding to an average speed of between 117 and 126 kph on this track. We note that the chosen track, Wheel 1, is about 4300 meters long — meaning the controllers are easily able to complete the track.

III. Conclusion

We have presented a novel implementation of an evolutionary algorithm, using the singular value decomposition of a connection matrix for a neural network and applied this algorithm to a recurrent neural network controller for a simulated race car in TORCS/SCRC.

Though the inputs from the simulated car to the network are few, with low angular resolution, and noisy, our evolved networks were able to perform better than the author (an experienced driver but novice gamer), were usually able to complete all tracks contained in TORCS/SCRC and were close to being competitive on time with previous TORCS/SCRC controllers on the Wheel 1 track.

We note that the network's control of the simulated cars might have been improved even further by using more inputs (at the expense of longer evolution time) and perhaps by placing final acceleration, steering and braking control under PID control — allowing the neural network to give only "target" values.

REFERENCES

[1] Sabeur Abid, Farhat Fnaiech, and Mohamed Najim. A new neural network pruning method based on the singular value decomposition and the weight initialisation. In *Signal Processing Conference, 2002 11th European*, pages 1–4. IEEE, 2002.

[2] Christos Athanasiadis, Damianos Galanopoulos, and Anastasios Tefas. Progressive neural network training for the open racing car simulator. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 116–123. IEEE, 2012.

[3] Armando Blanco, Miguel Delgado, and MC Pegalajar. A genetic algorithm to obtain the optimal recurrent neural network. *International Journal of Approximate Reasoning*, 23(1):67–83, 2000.

[4] Matteo Botta, Vincenzo Gautieri, Daniele Loiacono, and Pier Luca Lanzi. Evolving the optimal racing line in a high-end racing game. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 108–115. IEEE, 2012.

[5] Martin V Butz, Matthias J Linhardt, and Thies D Lonneker. Effective racing on partially observable tracks: Indirectly coupling anticipatory egocentric sensors with motor commands. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(1):31–42, 2011.

[6] Martin V Butz and Thies D Lonneker. Optimized sensory-motor couplings plus strategy extensions for the torcs car racing challenge. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 317–324. IEEE, 2009.

[7] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. On-line neuroevolution applied to the open racing car simulator. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 2622–2629. IEEE, 2009.

[8] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Learning to drive in the open racing car simulator using online neuroevolution. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(3):176–190, 2010.

[9] Jonathan A Cox. Parameter compression of recurrent neural networks and degradation of short-term memory. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 867–872. IEEE, 2017.

[10] Chris X Edwards. SnakeOil. http://xed.ch/project/snakeoil/index.html, 2016.

[11] Oscar Fontenla-Romero, Beatriz Pérez-Sánchez, and Bertha Guijarro-Berdiñas. LANN-SVD: a non-iterative SVD-based learning algorithm for one-layer neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2017.

[12] Hieu Trung Huynh and Yonggwan Won. Training single hidden layer feedforward neural networks by singular value decomposition. In *Computer Sciences and Convergence Information Technology, 2009. ICCIT'09. Fourth International Conference on*, pages 1300–1304. IEEE, 2009.

[13] Kui Jia. Improving training of deep neural networks via singular value bounding. *CoRR*, abs/1611.06013, 2016.

[14] PP Kanjilal, PK Dey, and DN Banerjee. Reduced-size neural networks through singular value decomposition and subset selection. *Electronics Letters*, 29(17):1516–1518, 1993.

[15] Jun Li, Xue Mei, Danil Prokhorov, and Dacheng Tao. Deep neural network for structural prediction and lane detection in traffic scene. *IEEE transactions on neural networks and learning systems*, 28(3):690–703, 2017.

[16] Daniele Loiacono, Luigi Cardamone, and Pier Luca Lanzi. Simulated car racing championship: Competition software manual. *CoRR*, abs/1304.1672, 2013.

[17] Daniele Loiacono, Pier Luca Lanzi, Julian Togelius, Enrique Onieva, David A Pelta, Martin V Butz, Thies D Lonneker, Luigi Cardamone, Diego Perez, Yago Sáez, et al. The 2009 simulated car racing championship. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(2):131–147, 2010.

[18] Jorge Muñoz, German Gutierrez, and Araceli Sanchis. Controller for TORCS created by imitation. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 271–278. IEEE, 2009.

[19] Jorge Muñoz, German Gutierrez, and Araceli Sanchis. A human-like TORCS controller for the simulated car racing championship. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 473–480. IEEE, 2010.

[20] Jorge Muñoz, German Gutierrez, and Araceli Sanchis. Multi-objective evolution for car setup optimization. In *Computational Intelligence (UKCI), 2010 UK Workshop on*, pages 1–5. IEEE, 2010.

[21] Carlos Fernandez Musoles. Towards neuroimaging real-time driving using convolutional neural networks. In *2016 8th Computer Science and Electronic Engineering (CEEC)*, pages 130–135, Sept 2016.

[22] Joshué Pérez, Vicente Milanés, and Enrique Onieva. Cascade architecture for lateral control in autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 12(1):73–82, 2011.

[23] Mike Preuss, Jan Quadflieg, and Günter Rudolph. TORCS sensor noise removal and multi-objective track selection for driving style adaptation. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 337–344. IEEE, 2011.

[24] Dimitris C Psichogios and Lyle H Ungar. SVD-NET: An algorithm that automatically selects network structure. *IEEE Transactions on Neural Networks*, 5(3):513–515, 1994.

[25] Jan Quadflieg, Günter Rudolph, and Mike Preuss. How costly is a good compromise: Multi-objective torcs controller parameter optimization. In *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*, pages 454–460. IEEE, 2015.

[26] Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[27] Jose Daniel A Santos, Guilherme A Barreto, and Claudio MS Medeiros. Estimating the number of hidden neurons of the mlp using singular value decomposition and principal components analysis: a novel approach. In *Neural Networks (SBRN), 2010 Eleventh Brazilian Symposium on*, pages 19–24. IEEE, 2010.

[28] Eu Jin Teoh, Kay Chen Tan, and Cheng Xiang. Estimating the number of hidden neurons in a feedforward network using the singular value decomposition. *IEEE Transactions on Neural Networks*, 17(6):1623–1629, 2006.

[29] Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. TORCS, The Open Racing Car Simulator. http://www.torcs.org, 2014.

[30] Elias Yee and Jason Teo. Evolutionary spiking neural networks as racing car controllers. In *Hybrid Intelligent Systems (HIS), 2011 11th International Conference on*, pages 411–416. IEEE, 2011.