Proceedings of the 2022 IEEE
International Conference on Robotics and Biomimetics
December 5-9, 2022, Xishuangbanna, China

Learning to Solve Nonlinear Optimization Problem with Deep Reinforcement Learning

Yue Gao*[1], Qiyue Yang[2], Huajian Wu[2] and Ming Sun[2]

***Abstract*— Nonlinear least-squares problems (NLS) are popular in engineering and scientific fields. Traditional optimization methods such as Newton's method and Gaussian-Newton method (GN) suffer from the sensibility to initial values and the high computational complexity. In this paper, we propose LS-DDPG, a robust optimization method utilizing deep reinforcement learning algorithms to solve nonlinear least-squares problems. The experiment results on synthetic data demonstrate that the proposed method outperforms Newton's method in terms of computation cost, convergence speed and initial values sensibility. In addition, LS-DDPG is utilized on model predictive control (MPC) problems for trajectory planning and tracking tasks in self-driving with longer prediction horizon and higher accuracy than baseline methods.**

I. Introduction

Nonlinear least-squares problem (NLS) is widely involved in various application scenarios, for instance, data fitting, image processing, optimal control and self-driving. It is usually considered as the problem which has the form of sum of squared errors

$$\min_{x} \quad f(x), \quad f(x) = \frac{1}{2}||r(x)||^2 = \frac{1}{2}\sum_{j} r_j^2(x) \qquad (1)$$

where r_j is the j-th residual term. Nonlinear least-squares problems can be solved either by specialized algorithms formulated according to specific structure of objective function, or by unconstrained optimization methods such as Newton's method and Gradient Descent [1]. Compared with Gradient Descent, Newton's method has quadratic convergence, but it is not suitable for solving large-scale problems due to the increased computational complexity in calculating Hessian matrix. In order to tackle this deficiency, Gauss-Newton method (GN) [2] and Levenberg-Marquardt method (LM) [3] use gradient vectors instead of Hessian matrix to compute the updates.

Another problem of Newton-type methods is that the optimization results are influenced by initial values. If the initial value is improperly assigned, Newton-type methods may not converge. To overcome these difficulties, Fletcher *et al.* [4] proposed a filter method for solving Nonlinear Programming problems. Shang *et al.* proposed a new filter QP-free method [5]. Each iteration of this method can be viewed as a perturbation of a Newton or Quasi-Newton iteration on both the primal and dual variables for the solution of the KKT conditions. Other works attempted to capitalize on the powerful fitting capabilities of deep neural networks to solve optimization problems, but mostly use first-order gradient information only. For instance, Clark *et al.* used a neural network to solve nonlinear least squares for monocular stereo [6], which is the first to use second-order approximations of the objective to learn optimization updates.

In this work, Deep Reinforcement Learning (DRL) is utilized to learn optimization updates. Reinforcement learning is to learn an optimal strategy that maximizes cumulative rewards by exploring an unknown environment. Basic reinforcement learning algorithms such as Q-learning [7] and Sarsa can only solve problems with low-dimensional, discrete state space and action space. To apply RL in more complex real-world scenarios with continuous state, deep Q network (DQN) [8] is proposed to estimate the state-action-value function using deep neural network. However, DQN is limited to handling discrete and low-dimensional action spaces. Another method that combines actor-critic method [9] with DQN, Deep Deterministic Policy Gradient (DDPG) [10] comes out and is proved to be effective for dealing with continuous action space. It has achieved great success in a variety of challenging physical control problems such as cartpole swing-up problem, urban traffic signal control and dynamical motor control [11].

The proposed optimization method is applied to the model predictive control (MPC) problems in trajectory planning and tracking tasks of self-driving. MPC intends to predict the future control law of the controlled system by solving a nonlinear optimization problem [12], which generally can be expressed as a quadratic programming (QP) problem. Some popular approaches to solve QP problems include interior point method, trust region reflective method [13] and Goldfarb and Idnani active set method [14]. The speed of solving optimization problem determines the real-time performance of MPC. Faster solution speed allows longer prediction horizon, thus leading to better control performance. In this paper, we transform a standard MPC problem into a proper format using Sigmoid function and solve it using LS-DDPG solver.

The contributions of this paper are summarized below:

- A robust method named LS-DDPG is proposed to incorporate deep reinforcement learning into approximate-

This work is supported by the National Key Research and Development Program of China (Grant No. 2021YFF0307900), National Natural Science Foundation of China (Grant No. 61903247), and Shanghai Municipal Science and Technology Major Project (Grant No. 2021SHZDZX0102).

[1]Yue Gao is with MoE Key Lab of Artificial Intelligence and AI Institute, Shanghai Jiao Tong University, Shanghai, P.R. China. Email: yuegao@sjtu.edu.cn

[2]Qiyue Yang, Huajian Wu and Ming Sun are with Department of Automation, Shanghai Jiao Tong University, Shanghai, P.R. China. Email: {yangqiyue, summeryuki, mingsun}@sjtu.edu.cn

*Corresponding author.

978-1-6654-8109-0/22/$31.00 © 2022 IEEE

Hessian-based optimization approaches for nonlinear least-squares problems.

- LS-DDPG outperforms Newton's method in terms of computation speed, convergence steps and initial values sensibility.
- The model predictive control problem for trajectory planning and tracking tasks is formulated and solved through LS-DDPG to verify the effectiveness of LS-DDPG over self-driving problems.

II. Preliminaries

A. Unconstrained Optimization Solvers

Newton's method is one of the most common and widely used method in solving unconstrained optimization problems. Given an estimate of x^*, x_k at each step k, Newton's method compute the update Δx_k to find a better solution. At step $k+1$, $x_{k+1} = x_k + \Delta x_k$. We can derive from Taylor expansion that,

$$\Delta x_k = -H(x_k)^{-1}\nabla f(x_k) \tag{2}$$

$H(x_k)$ is Hessian matrix, which can be difficult to compute in high dimensional problems. Gauss-Newton (GN) and Levenberg-Marquadt (LM) method seek to approximate the Hessian matrix using $J^{\mathrm{T}}J$ to avoid this deficiency. The update of GN at step k is

$$\Delta x_k = -(J_k^{\mathrm{T}}J_k)^{-1}J_k^{\mathrm{T}}r_k \tag{3}$$

where $r_k = r(x_k)$, $J_k = \frac{\mathrm{d}r}{\mathrm{d}x_k}$. LM extends GN by adding a positive correction scalar to the update term

$$\Delta x_k = -(J_k^{\mathrm{T}}J_k + \lambda diag(J_k^{\mathrm{T}}J_k))^{-1}J_k^{\mathrm{T}}r_k \tag{4}$$

which avoids the possible noninvertible problems caused by $J_k^{\mathrm{T}}J_k$ and makes the algorithm more robust.

In our method, the update is computed by Jacobian J_k using LS-DDPG rather than Hessian matrix or its approximation as in Newton's method or GN. The proposed approach is proved to have faster computing speed, fewer convergence steps and better robustness over different initial points than Newton's method in solving nonlinear equations.

B. DDPG

Reinforcement learning is an algorithm that the agent interacts with a complex, uncertain environment and tries to learn a policy to maximize the cumulative reward. At time t, the agent interacts with the environment, of which state is s_t, through action a_t and then receives a reward r_t from the environment. The environment responds to this action and moves to a new state s_{t+1}. After a series of interactions between agent and environment, the expected return from current state to a future state is defined as the sum of discounted rewards $R_t = \sum_{i=t}\gamma^{i-t}r(s_i, a_i)$, where $\gamma \in (0, 1)$ is the discount rate. Then RL tries to learn a policy that maximizes the expected return.

Our method is based on Deep Deterministic Actor-Critic Policy Gradient (DDPG). It consists of two networks: the Actor network and the Critic network. The Actor network receives the current state s_t and outputs the actions a_t, and the Critic network estimates the value function of the policy. DDPG uses deterministic strategy μ to select action $a_t = \mu(s_t|\theta^\mu)$, where θ^μ is the parameter of Actor network. So the objective function can be defined as

$$J(\theta^\mu) = E_{\theta^\mu}[r + \gamma r_2 + \gamma^2 r_3 + \ldots] \tag{5}$$

Meanwhile, it uses the Critic network to fit the Q function $Q_t = Q(s_t, a_t|\theta^Q)$. θ^Q is the parameter of the Critic network. The Q function here represents the expected value of the reward for the selected action under the deterministic strategy μ. So we can get

$$Q^\mu(s_t, a_t) = E[r(s_t, a_t) + \gamma Q^\mu(s_{t+1}, \mu(s_{t+1}))] \tag{6}$$

Because the state space and action space are continuous, the objective function can be formulated as

$$J_\beta(\mu) = \int_{\mathcal{S}} \rho^\beta(s)Q^\mu(s, \mu(s))\, ds = E_{s\sim\rho^\beta}[Q^\mu(s, \mu(s))] \tag{7}$$

where β is the behavior policy. So the gradient of the Actor network is

$$\begin{aligned}\frac{\partial J(\theta^\mu)}{\partial \theta^\mu} &= E_s[\frac{\partial Q(s, a|\theta^Q)}{\partial \theta^\mu}] \\ &= E_s[\frac{\partial Q(s, a|\theta^Q)}{\partial a}\frac{\partial \pi(s|\theta^\mu)}{\partial \theta^\mu}]\end{aligned} \tag{8}$$

$$\begin{aligned}\nabla_\theta J_\beta(\mu_\theta) &= \int_{\mathcal{S}} \rho^\beta(s)\nabla_\theta \mu_\theta(s)Q^\mu(s, a)|_{a=\mu_\theta}\, ds \\ &= E_{s\sim\rho^\beta}[\nabla_\theta \mu_\theta(s)Q^\mu(s, a)|_{a=\mu_\theta}]\end{aligned} \tag{9}$$

The gradient of the Critic network is

$$\begin{aligned}\frac{\partial L(\theta^Q)}{\partial \theta^Q} &= E_{s,a,r,s'\sim D}[(Q_t - Q(s, a|\theta^Q)\frac{\partial Q(s, a|\theta^Q)}{\partial \theta^Q})] \\ Q_t &= r + \gamma Q'(s', \pi(s'|\theta^{\mu'})|\theta^{Q'})\end{aligned} \tag{10}$$

The loss function is Mean-squared-error. Then gradient descent can be used to update two networks parameters based on (8)-(10).

C. Model Predictive Control

In this paper, we focus on MPC in self-driving with unconstrained predicted trajectory η. The objective function can be defined as

$$\begin{aligned}J = &\sum_{i=1}^{N_p} \|\eta(t+i \mid t) - \eta_{ref}(t+i \mid t)\|_Q^2 \\ &+ \sum_{i=1}^{N_c-1} \|\Delta u(t+i \mid t)\|_R^2\end{aligned} \tag{11}$$

where Q and R are weight matrices and η_{ref} is the reference trajectory. The objective function in (11) is designed to narrow the difference between reference trajectory and predicted trajectory and at the same time ensure that the control variables change steadily.

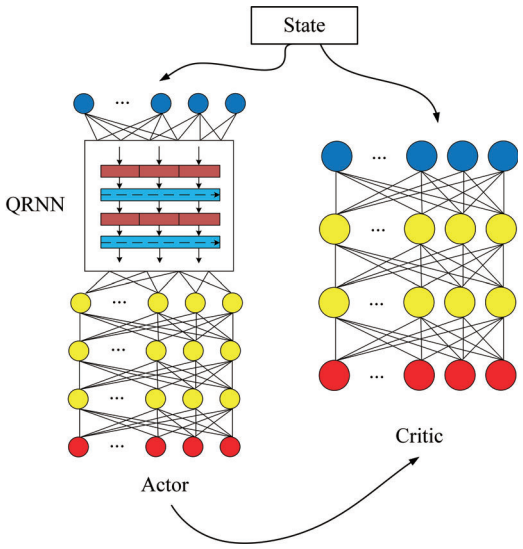


Fig. 1: Architecture of Actor and Critic network in LS-DDPG. The Actor consists of a Q-RNN following by a fully-connected network, and the critic network is a four-layer fully-connected network.

III. LS-DDPG

A. The Structure of Actor and Critic

The implementation of Actor and Critic network is shown in Fig. 1. The Actor network receives the current state s_t as input and outputs action a_t. It starts with a Quasi-Recurrent Neural Networks (Q-RNN), which benefits from both CNNs and RNNs by combining them together [15]. The first layer of Q-RNN is a convolution layer for extracting features, and the second layer is a pooling layer. Q-RNN is followed by a four-layer fully-connected network with Rectified Linear Units (ReLU) activation. The output layer has linear activation.

The Critic network receives the current state s_t, target state s^* and action a_t generated by the actor, and outputs the corresponding state action value $Q(s_t, a_t)$. It consists of a four-layer fully-connected network with ReLU activation, and the output layer consists of 1 unit with linear activation.

B. State, Action and Reward

1) State: The state of our model is associated with Jacobian and residual at step k, (J_k, r_k). Transformation $\Omega(J_k, r_k)$ is applied to the Jacobian and residual term to define an appropriate state

$$\Omega(J_k, r_k) = [J_k^{\mathrm{T}} J_k, J_k^{\mathrm{T}} r_k] \tag{12}$$

In LS-DDPG, $J_k^{\mathrm{T}} J_k$ instead of $(J_k^{\mathrm{T}} J_k)^{-1}$ is computed as this reduces computational complexity. LS-DDPG models of 1, 2, 4, and 5 variables are trained. If the optimization problem has more than 5 variables, we optimize specific variables by temporarily fixing other variables.

2) Action: Our model uses update Δx_k as action. At step k, the estimate of x^* is x_k and current state s_k is $[J_k^{\mathrm{T}} J_k, J_k^{\mathrm{T}} r_k]$. LS-DDPG outputs an action $a_k = \Delta x_k$ according to the state. The estimate of x^* will be updated by

$$x_{k+1} = x_k + a_k \tag{13}$$

Then the new state s_{k+1} at step $k+1$ is

$$\Omega(J_{k+1}, r_{k+1}) = [J_{k+1}^{\mathrm{T}} J_{k+1}, J_{k+1}^{\mathrm{T}} r_{k+1}] \tag{14}$$

3) Reward: The reward of our model is defined as

$$reward = -1 - \sum_j \|r_j\| \tag{15}$$

The fixed reward term -1 is designed to minimize the number of steps required for convergence. Each step, the agent will receive a negative reward -1 until the residual has met certain termination conditions. The second term $\sum_j \|r_j\|$ is the residual of each step and it can be relatively large in the first few steps. By minimizing this residual term, the objective function will learn to descend rapidly in few steps.

C. Solving MPC utilizing LS-DDPG

Assume that η is unconstrained. In this case (11) can be transformed into a proper unconstrained optimization problem that fits for LS-DDPG. It is formulated as

$$\begin{aligned} \min_u \quad & J = \sum_{i=1}^{N_p} \|\eta(t+i \mid t) - \eta_{ref}(t+i \mid t)\|_Q^2 \\ & + \sum_{i=1}^{N_c-1} \|\Delta u(t+i \mid t)\|_R^2 \\ \text{s.t.} \quad & \Delta u_{min} \le \Delta u(t+i \mid t) \le \Delta u_{max} \\ & i = 0, 1, \ldots, N_c - 1 \end{aligned} \tag{16}$$

It is a quadratic programming problem with the form of

$$\begin{aligned} \min_U \quad & \frac{1}{2} U(t)^{\mathrm{T}} H(t) U(t) + G(t)^{\mathrm{T}} U(t) + P(t) \\ \text{s.t.} \quad & \Delta u_{min} \le \Delta u(t+i \mid t) \le \Delta u_{max} \\ & i = 0, 1, \ldots, N_c - 1 \end{aligned} \tag{17}$$

$$\begin{aligned} & \Delta u(t+i \mid t) = \frac{\Delta u_{max} - \Delta u_{min}}{1 + e^{-\Delta \hat{u}(t+i|t)}} + \Delta u_{min} \\ & i = 0, 1, \ldots, N_c - 1 \end{aligned} \tag{18}$$

We use the Sigmoid function in (18) to transform it into the following unconstrained optimization problem:

$$\min_{\hat{U}} \quad \frac{1}{2} \hat{U}(t)^{\mathrm{T}} H(t) \hat{U}(t) + G(t)^{\mathrm{T}} \hat{U}(t) + P(t) \tag{19}$$

Compared with (17), such transformation increases the nonlinearity of the objective function but eliminates constraints. It will not cause much inconvenience since LS-DDPG is good at dealing with nonlinear optimization problems.

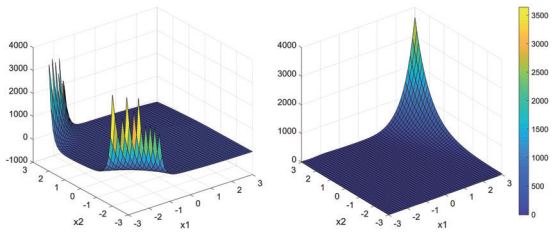


Fig. 2: Graph of E_1. The left subfigure shows the distribution of first equation of E_1 and the right one displays the second equation.

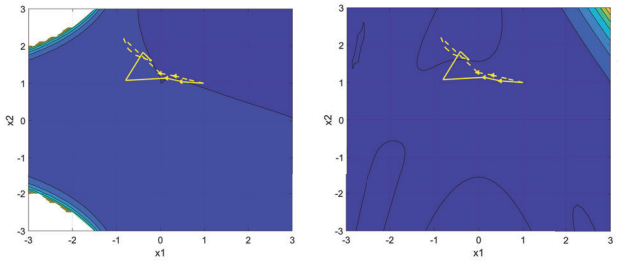


Fig. 3: Iterative paths of applying LS-DDPG and Newton's method to E_1. Solid and dotted lines represent LS-DDPG and Newton's method, respectively. Left subfigure shows the first equation of E_1 and right subfigure shows the second equation.

IV. Experiments

A. Training

LS-DDPG network is trained using a basic function pool and backpropagate the loss. The function pool consists of many basic functions such as $sin(x)$, $cos(x)$, x^a, e^x and $ln(x)$. At each episode, a new nonlinear equation problem is formulated based on the basic function pool and LS-DDPG is to solve the problem before reaching max steps. The reward discount is $\gamma = 0.9$. The variable representing action randomness for exploration decays from 2.0 to 0.05 and the learning rates for Actor and Critic network are both 10^{-4}.

The training process of the network can be accelerated by changing the action at step k into

$$a_k = \alpha_k \ (0 \leq \alpha_k \leq \alpha_{max}) \tag{20}$$

In this case, the state is still $[J_k^{\mathrm{T}} J_k, J_k^{\mathrm{T}} r_k]$. At step $k+1$, the estimate of x^* is

$$x_{k+1} = x_k + a_k((J_k^{\mathrm{T}} J_k)^{-1} J_k^{\mathrm{T}} r_k) \tag{21}$$

This will significantly speed up the training process, but it may be unable to handle the case where $J_k^{\mathrm{T}} J_k$ is noninvertible, which can be solved by setting a large negative reward at invertible points.

B. Synthetic Data Experiments

We evaluate the performance of proposed method on a number of nonlinear equation problems. Experiments are conducted on the following four sets of nonlinear equations:

$$E_1: \begin{cases} x_1^2 e^{-\frac{x_1 x_2^2}{2}} - (x_1 + x_2 - 1)^3 = 0 \\ x_2^2 cos(x_1^2 + x_2) + x_1^2 e^{x_1 + x_2} = 0 \end{cases} \tag{22}$$

$$E_2: \begin{cases} 3x_1 - cos(x_2 x_3) - \frac{1}{2} = 0 \\ x_1^2 - 81(x_2 + 1)^2 + sin(x_3) + 1.06 = 0 \\ e^{-x_1 x_2} + 20x_3 + \frac{(10\pi - 3)}{3} = 0 \end{cases} \tag{23}$$

$$E_3: \begin{cases} x_1^3 + e^{x_1} + 2x_2 + x_3 + 1 = 0 \\ -x_1 + x_2 + x_2^3 + 2e^{x_2} - 3 = 0 \\ -2x_2 + x_3 + e^{x_3} + 1 = 0 \end{cases} \tag{24}$$

$$E_4: \begin{cases} 4x_1^3 - x_2 + x_3 - x_1 x_4 = 0 \\ -x_1 + 3x_2 - 2x_3 - x_2 x_4 = 0 \\ x_1 - 2x_2 + 3x_3 - x_3 x_4 = 0 \\ x_1^2 + x_2^2 + x_3^2 - 1 = 0 \end{cases} \tag{25}$$

These nonlinear equation problems have the form

$$\begin{aligned} f_1(x_1, x_2, \ldots) &= 0 \\ f_2(x_1, x_2, \ldots) &= 0 \\ &\cdots \\ f_i(x_1, x_2, \ldots) &= 0 \end{aligned} \tag{26}$$

They can be equivalently transformed into a nonlinear least-squares problem in (27).

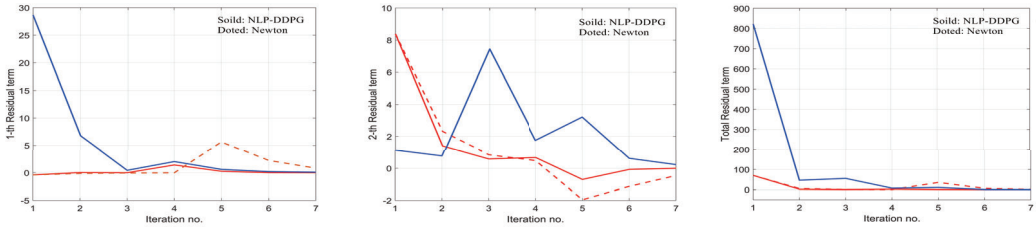
$$\min_{x=[x_1, x_2, \ldots]} \begin{aligned} r_1^2 &= \|f_1(x_1, x_2, \ldots)\|^2 \\ r_2^2 &= \|f_2(x_1, x_2, \ldots)\|^2 \\ &\cdots \\ r_i^2 &= \|f_i(x_1, x_2, \ldots)\|^2 \end{aligned} \tag{27}$$

During these experiments, problems E_1 to E_4 are solved using LS-DDPG and Newton's method respectively with different initial points. The termination criterion is $||\phi|| \leq 10^{-3}$. Then we compare the performance of the two algorithms in terms of convergence speed and robustness. Take problem E_1 as an example. The distribution of its objective function is displayed in Fig. 2 and the iterative paths of Newton's method and LS-DDPG are shown in Fig.3. The proposed method requires less iterations to converge and finds a more efficient path than Newton's method does.

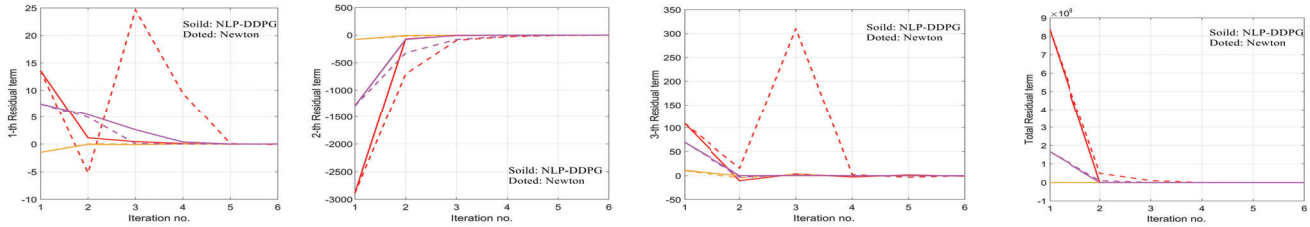
Fig. 4 shows the experimental results on functions E_1 to E_4. Doted lines and solid lines are used to differentiate between the results of Newton's method and LS-DDPG. The proposed method clearly outperforms Newton's method in terms of convergence speed and stability. We also use lines with different colors to display the results of choosing various initial values. In those cases where Newton's method fails to converge, the learned approach still performs well, proving its robustness over various selections of initial points.

C. MPC for Self-driving Vehicles

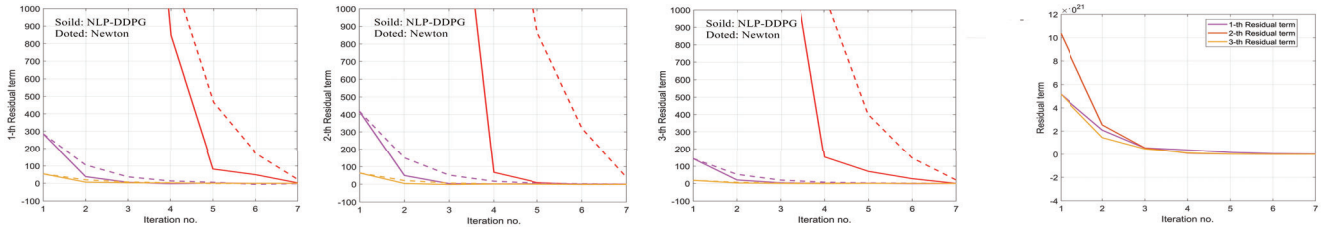
In this section, we apply the proposed method to self-driving vehicles with low-speed dynamics model. The performance of MPC and our method are evaluated using CarSim



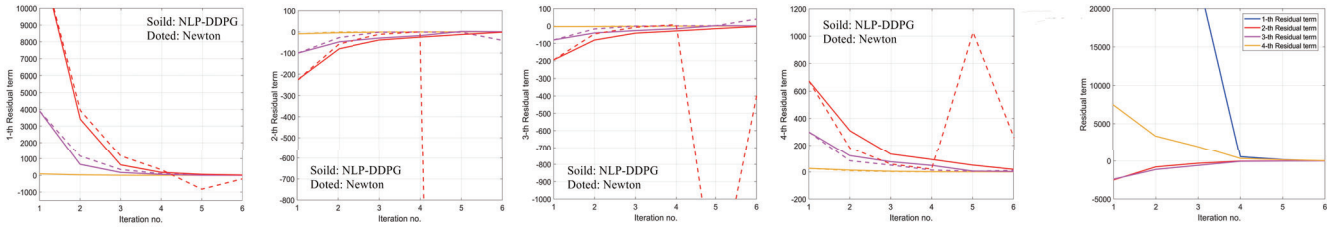
(a) Results of solving E_1 from different initial points. The solution is $(-0.2099, 1.5960)^{\mathrm{T}}$. Red lines and blue lines represent the results with a initial value of $(1, 1)^{\mathrm{T}}$ and $(-1, -1)^{\mathrm{T}}$, respectively. When starting from $(-1, -1)^{\mathrm{T}}$, Newton's method is unable to converge while our method can find the solution in few iterations.



(b) Results of solving E_2 from initial points $(0, 0, 0)^{\mathrm{T}}$,$(3, 3, 3)^{\mathrm{T}}$ and $(5, 5, 5)^{\mathrm{T}}$, the corresponding marking lines are yellow, pink and red respectively. The solution is $(0.4597, -0.9038, -0.9454)^{\mathrm{T}}$,$(0.4404, -1.0948, -0.5546)^{\mathrm{T}}$. Newton's method exists oscillations but our method does not.



(c) E_3 solved with initial values of $(3, 3, 3)^{\mathrm{T}}$, $(5, 5, 5)^{\mathrm{T}}$ and $(10, 10, 10)^{\mathrm{T}}$, the corresponding marking lines are yellow, pink and red respectively. The solution is $(-0.7678, 0.0753, -1.1622)^{\mathrm{T}}$. Rightmost figure shows the iteration results starting from $(50, 50, 50)^{\mathrm{T}}$, in which case Newton's method fails to converge.



(d) Results of solving E_4 from initial points $(3, 3, 3, 3)^{\mathrm{T}}$, $(10, 10, 10, 10)^{\mathrm{T}}$, $(15, 15, 15, 15)^{\mathrm{T}}$, the corresponding marking lines are yellow, pink and red. The solution is $(0.8165, 0.4082, -0.4082, 3.0000)^{\mathrm{T}}$. The rightmost image shows the iteration results of LS-DDPG starting from $(50, 50, 50, 50)^{\mathrm{T}}$, in which case Newton's method is unable to converge.

Fig. 4: Results of solving E_1 to E_4. For all the pictures above, the solid lines represent results using LS-DDPG while the doted lines represent results of Newton's method. Our method clearly outperforms Newton's method in terms of speed of convergence and robustness, for Newton's method may cause oscillations or non-convergence while ours remains stable.

software.

The goal of this experiment is to track the trajectory of self-driving vehicles while driving at low speed. Its dynamics model is

$$\begin{bmatrix} \dot{x}_r \\ \dot{y}_r \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} cos\varphi \\ sin\varphi \\ tan\delta_f / l \end{bmatrix} v_r \tag{28}$$

x_r and y_r are the coordinates of rear axle center and φ is the course angle of vehicles. δ_f is the steering angle and l is wheelbase. v_r is the speed of rear axle center. Linearize the model to obtain the linear time-varying system as follows:

$$\xi(k+1) = A(k)\xi(k) + B(k)u(k) \tag{29}$$

where

$$\xi = \begin{bmatrix} x - x_r \\ y - y_r \\ \varphi - \varphi_r \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 0 & -v_r sin\varphi_r T \\ 0 & 1 & -v_r cos\varphi_r T \\ 0 & 0 & 1 \end{bmatrix} \tag{30}$$

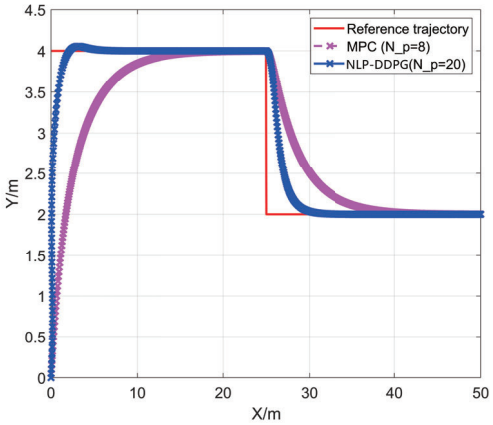


Fig. 5: Trajectory tracking of low-speed self-driving model. The maximum prediction horizon of MPC solved by active-set in 0.01 s is 8 and ours is 20.

TABLE I: COMPARED WITH MPC

	Ours	MPC
Solver	LS-DDPG	Goldfarb/Idnani dual algorithm
Max Prediction Horizon in $0.01s$	20	8
Average Time with 20 Prediction Horizon	$0.008s$	$0.03s$
Programming Language	Python	Python

$$B = \begin{bmatrix} cos\varphi_r T & 0 \\ sin\varphi_r T & 0 \\ \frac{tan\delta_{f,r} T}{l} & \frac{v_r T}{l cos^2(\delta_{f,r})} \end{bmatrix}$$

T is the sampling time. The optimization problem without constraint on η can be formulated as (16). We use Sigmoid function in (18) to transform it into an unconstrained optimization problem in (19) and then evaluate the performance of state-of-art methods and ours.

The results are displayed in Fig. 5 and TABLE I. Within $0.01s$, the maximum prediction horizon of MPC solved by Goldfarb/Idnani dual algorithm is 8 and ours is 20. Longer prediction horizon means better performance. As shown in Fig. 5, the predicted trajectory of proposed approach fits the reference trajectory better than that of Newton's method. In addition, the average time spent to solve the problem with 20 prediction horizon using Goldfarb/Idnani dual algorithm is $0.03s$, compared with $0.008s$ using LS-DDPG.

V. Conclusion

In this paper, we propose LS-DDPG, a DDPG-based optimization method that builds upon Hessian approximation architecture. We firstly evaluated the performance of proposed method on synthetic equation problems. Compared with Newton's method, LS-DDPG consistently achieves faster computation speed, fewer convergence steps and better robustness over different initial values. Furthermore, a model predictive control problem for trajectory planning and tracking tasks is formulated and solved through LS-DDPG, demonstrating the effectiveness of the proposed method. In future works, LS-DDPG will be extended to explore the ability of the proposed method to handle larger scale nonlinear least-squares problems.

References

[1] H. Mohammad, M. Y. Waziri, and S. A. Santos, "A brief survey of methods for solving nonlinear least-squares problems," *Numerical Algebra, Control & Optimization*, vol. 9, no. 1, p. 1, 2019.

[2] H. O. Hartley, "The modified gauss-newton method for the fitting of non-linear regression functions by least squares," *Technometrics*, vol. 3, no. 2, pp. 269–280, 1961.

[3] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly of applied mathematics*, vol. 2, no. 2, pp. 164–168, 1944.

[4] R. Fletcher, S. Leyffer, and P. L. Toint, "On the global convergence of a filter–sqp algorithm," *SIAM Journal on Optimization*, vol. 13, no. 1, pp. 44–59, 2002.

[5] Y. Shang, Z.-F. Jin, and D. Pu, "A new filter qp-free method for the nonlinear inequality constrained optimization problem," *Journal of Inequalities and Applications*, vol. 2018, no. 1, pp. 1–14, 2018.

[6] R. Clark, M. Bloesch, J. Czarnowski, S. Leutenegger, and A. J. Davison, "Learning to solve nonlinear least squares for monocular stereo," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 284–299.

[7] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[9] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in neural information processing systems*, vol. 12, 1999.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[11] H. Shi, Y. Sun, and J. Li, "Dynamical motor control learned with deep deterministic policy gradient," *Computational intelligence and neuroscience*, vol. 2018, 2018.

[12] C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—a survey," *Automatica*, vol. 25, no. 3, pp. 335–348, 1989.

[13] A. R. Conn, N. I. Gould, and P. L. Toint, *Trust region methods*. SIAM, 2000.

[14] D. Goldfarb and A. Idnani, "A numerically stable dual method for solving strictly convex quadratic programs," *Mathematical programming*, vol. 27, no. 1, pp. 1–33, 1983.

[15] J. Bradbury, S. Merity, C. Xiong, and R. Socher, "Quasi-recurrent neural networks," *arXiv preprint arXiv:1611.01576*, 2016.