Optimal Convergence Rate in FFNNs using HJB Equation[2]

Problem Statement

To use control theoretic approach(HJB equation) for both batch and instantaneous updates of weights in feed-forward neural networks. How to use HJB(Hamilton-Jacobi-Bellman) equation to generate an optimal weight update law?

Mathematical Formulation

The output $\mathbf{y}_p \in \mathbf{R}^{N_o}$ for a given input pattern $\mathbf{x}_p \in \mathbf{R}^{N_i}$ can be written as,

$$\mathbf{y}_p = f(\mathbf{w}, \mathbf{x}_p)$$

Question Is the availability of closed form of f always ensured? What happens when f is not available in closed form?

Here $\mathbf{w} \in R^{N_w}$ is the vector of weight parameters involved in the FFNN to be trained, with total N_x weight parameters. The derivative of y w.r.t time t is

$$\begin{aligned}\frac{\partial y_p}{\partial t} &= \frac{\partial f(w, x_p)}{\partial w}\frac{\partial w}{\partial t} \\ &= J_p \frac{\partial w}{\partial t}\end{aligned}$$

where $J_p = \frac{\partial f(w,x_p)}{\partial w}$ is the Jacobian matrix. The desired output is given by $y_p^d = f(w, x_p)$ and its derivative w.r.t time is

$$\begin{aligned}\frac{\partial y_p^d}{\partial t} &= J_p \frac{\partial w}{\partial t} \\ &= 0\end{aligned}$$

The estimation error is $e_p = y_p^d - y_p$ and its derivative is given by

$$\begin{aligned}\frac{\partial e_p}{\partial t} &= \frac{\partial y_p^d}{\partial t} - \frac{\partial y_p}{\partial t} \\ &= -J_p \frac{\partial w}{\partial t}\end{aligned}$$

Question Can we use a different error function?

Mathematical Formulation

Define $\frac{\partial w}{\partial t} = u$. In batch mode, all the patterns are learnt simultaneously. Hence,

$$\frac{\partial e}{\partial t} = -Ju$$

The cost function is deined over time interval $(t, T]$ as

$$V(e(t)) = \int_t^T L(e(\tau), u(\tau))d\tau$$

where $L(e, u) = \frac{1}{2}(e^\top e + u\top Ru)$. We have to find optimal update law $u(t)$, hence we get the HJB equation

$$\min_u \{\frac{dV*}{de}\frac{\partial e(t)}{\partial t} + L(e(t), u(t))\} = 0$$

Question Could we use a different loss function?

ML Approach Used in the Paper

WEIGHT UPDATE LAWS OF VARIOUS ALGORITHMS

Algorithm	Update law, $\dot{\mathbf{w}} = \mathbf{u}$
BP	$\mathbf{u} = \eta\mathbf{J}^\intercal\mathbf{e}$
LF	$\mathbf{u} = \mu\frac{\|\|\mathbf{e}\|\|^2}{\|\|\mathbf{J}^\intercal\mathbf{e}\|\|^2}\mathbf{J}^\intercal\mathbf{e}$
LM	$\mathbf{u} = [\mathbf{J}^\intercal\mathbf{J} + \mu\mathbf{I}]^{-1}\mathbf{J}^\intercal\mathbf{e}$
HJB	$\mathbf{u} = \frac{1}{r}\mathbf{J}^\intercal\mathbf{C}\mathbf{e}$
single output online HJB	$\mathbf{u} = \frac{1}{(r\mathbf{J}\mathbf{J}^\intercal)^{1/2}}\mathbf{J}^\intercal e$

- **Global Minimization**
  The paper uses Lyapunov function to impose further condition on the solution. However, it turns out that these conditions compel the error $e(t)$ to reduce at each step. This behaves like a greedy search scheme, ending up in the local minimum. On the contrary, the convergence to global optimum may sometimes require $e(t)$ to increase also.

Suggestion We can use exploration/exploitation trade-off idea of Reinforcement Learning here. If we encourage exploration in the initial period of training slowly shift the importance towards exploitation we can have better chances of convergence to global minimum.

HJB-Equation-Based Optimal Learning Scheme for NNs With Applications in Brain–Computer Interface[4]

ML Approach Used in the Paper

WEIGHT UPDATE LAWS OF VARIOUS ALGORITHMS

Algorithm	Update law, $\dot{\hat{\mathbf{w}}} = \mathbf{u}$
BP	$\mathbf{u} = \eta \mathbf{J}^\intercal \mathbf{e}$
LF	$\mathbf{u} = \mu \frac{\|\|\mathbf{e}\|\|^2}{\|\|\mathbf{J}^\intercal \mathbf{e}\|\|^2} \mathbf{J}^\intercal \mathbf{e}$
LM	$\mathbf{u} = [\mathbf{J}^\intercal \mathbf{J} + \mu \mathbf{I}]^{-1} \mathbf{J}^\intercal \mathbf{e}$
EVDHJB	$\mathbf{u} = \mathbf{R}^{-1} \mathbf{J}^\intercal \mathbf{C} \mathbf{e}$
HJB with time optimality	$\mathbf{u} = \frac{\sqrt{2\mathbf{P}(\mathbf{e})}}{\|\|\mathbf{J}^\intercal \mathbf{e}\|\|} \mathbf{R}^{-1/2} \mathbf{J}^\intercal \mathbf{e}$

- The HJB formulation from previous paper is extended by deriving a closed from solution without using eigenvalue decomposition(EVD). This significantly reduces the number of computations, which is otherwise needed for EVD in each iteration.
- Another advantage of this algorithm is that it is easily integrable with state-of-the-art adaptation schemes like adding a momentum term, AdaGrad, etc..
- The update rule:

$$u^*(t) = \frac{\sqrt{e^\top(t)e(t)}}{|J^\top e(t)|} R^{-1/2} J^\top e(t)$$

- In the general case, $e^\top Q e$ can be replaced by any other cost function say $P(e(t))$, then we get

$$u^*(t) = \frac{\sqrt{2P(e(t))}}{|J^\top e(t)|} R^{-1/2} J^\top e(t)$$

Comparision between HJB-Equation-Based Optimal Learning Scheme and Optimal Convergence Rate in FFNNs using HJB Equation[2][4]

Main Similarities

- Both uses HJB equation to derive update rules which are faster than present standard Gradient Descent Back Propagation.

  Question Is the equation in paper-2 not under-determined?

- Both use Lyapunov function to define the stability of system. The function is used to put the constraint that input u^* must

Main Differences

- Update rule in paper-1 uses EVD whereas it is not use in case of paper-2.
- We can use state-of-the-art adaptation schemes like adding a momentum term, AdaGrad, etc.. in paper-2 but such a thing is not mentioned in paper-1.

Comparison between HJB-Equation-Based Optimal Learning Scheme and Learning to Learn by Gradient Descent by Gradient Descent[2][1]

Main Similarities

- Both uses explicit form of the loss function to calculate the update rule.

Main Differences

- Any loss can be used in paper-2 but a specific loss function is used in paper-1.
- Paper-1 uses HJB equation to derive the update rule. Whereas paper-2 uses LSTM(RNN) for the update rule(details in report 1).
- Paper-1 is less extensive compared to paper-2 in terms of computation complexity.

Comparison between HJB-Equation-Based Optimal Learning Scheme and Learning to Learn without Gradient Descent by Gradient Descent[2][3]

Main Similarities

- We can use exploration/exploitation trade-off idea of RL.

Main Differences

- Paper-1 uses explicit form of the loss function whereas paper-2 uses implicit form of loss function to derive update rule.
- Paper-1 is less extensive compared to paper-2 in terms of computation complexity.

Learning Algorithms for RL[5]

Policy Iteration

1. Initialization
   - $V(s) \in \mathbf{R}$ and $\pi(s) \in A(s)$ arbitrarily for all $s \in S$
2. Policy Evaluation
   Loop
   - $\Delta \longleftarrow 0$
   - Loop for each $s \in S$:
     - $v \longleftarrow V(s)$
     - $V(s) \longleftarrow \sum_{s' \in S} P_{\pi(s)}(s, s') \times V(s')$
     - $\Delta \longleftarrow \max(\Delta, \|v - V(s)\|)$
   - until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)
3. Policy Improvement
   - $policy - stable \longleftarrow true$
   - For each $s \in S$:
     - $old - action \longleftarrow \pi(s)$
     - $\pi(s) \longleftarrow argmax_a \sum_{s' \in S} P_{\pi(s)}(s, s')\big[R_{\pi(s)}(s, s') + \gamma V(s')\big]$
   - If $policy - stable$, then stop and return $V \sim v_*$ and $\pi \sim \pi_*$; else go to 2

Value Iteration

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation

1. Initialize $V(s)$, for all $s \in S$, arbitrarily except that $V(terminal) = 0$
2. Policy Evaluation and Improvement
   Loop
   - $\Delta \longleftarrow 0$
   - Loop for each $s \in S$:
     - $v \longleftarrow V(s)$
     - $V(s) \longleftarrow \max_{\pi(s)} \sum_{s' \in S} P_{\pi(s)}(s, s')\big[R_{\pi(s)}(s, s') + \gamma V(s')\big]$
     - $\Delta \longleftarrow \max(\Delta, \|v - V(s)\|)$
   - until $\Delta < \theta$

   Output a deterministic policy, $\pi \sim \pi_*$, such that

$$\pi(s) = argmax_a \sum_{s' \in S} P_{\pi(s)}(s, s')\big[R_{\pi(s)}(s, s') + \gamma V(s')\big]$$

Value iteration effectively combines, in each of its sweeps, one sweep of policy evaluation and one sweep of policy improvement.

References

[1] Marcin Andrychowicz et al. "Learning to learn by gradient descent by gradient descent". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee et al. Curran Associates, Inc., 2016, pp. 3981–3989. URL: http://papers.nips.cc/paper/6461-learning-to-learn-by-gradient-descent-by-gradient-descent.pdf.

[2] Vipul Arora, Laxmidhar Behera, and Ajay Pratap Yadav. "Optimal Convergence Rate in Feed Forward Neural Networks using HJB Equation". In: *arXiv preprint arXiv:1504.07278* (2015).

[3] Yutian Chen et al. "Learning to learn without gradient descent by gradient descent". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 748–756.

[4] Tharun Kumar Reddy, Vipul Arora, and Laxmidhar Behera. "HJB-Equation-Based Optimal Learning Scheme for Neural Networks With Applications in Brain–Computer Interface". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 99 (2018), pp. 1–12.

[5] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge, 1998.