DeepSolar: Optimizing Rotational Speed for Solar Heating System with Deep Reinforcement Learning

Tianchi Huang*
Dept. of Computer Science and Technology, Tsinghua University
htc19@mails.tsinghua.edu.cn

ABSTRACT

The solar heating and cooling systems are among the best solutions for the current energy and environment issues. In this work, we aim to leverage deep learning technologies for enhancing solar heating systems. In detail, we first investigate the overall performance of predicting future energy with different learning-based methods. Next, we propose DeepSolar, aiming to dynamically adjust the rotational speed during the entire process via deep reinforcement learning method. Using trace-driven evaluation, we indicate the superiority of DeepSolar over existing state-of-the-art approaches.

CCS CONCEPTS

• **Information systems** → *Multimedia streaming*; • **Computing methodologies** → *Neural networks*;

KEYWORDS

Solar Heating System, Deep Learning, Deep Reinforcement Learning.

ACM Reference Format:
Tianchi Huang. 2020. DeepSolar: Optimizing Rotational Speed for Solar Heating System with Deep Reinforcement Learning. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, Article 4, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Solar heating systems capture thermal energy from the sun and use it to heat water for your home. In this work, we consider solar space heaters, a system which uses the energy of the sun to heat your home. Such systems typically require more collectors, as well as bigger storage units to get the job done. The thermal energy is harnessed at the solar collectors and used to heat air, which is then circulated to disperse heat.

An interesting question for the solar heating system is to estimate the storage energy that the system collected for each time. It's quite challenging since the value is often determined by the solar radiation, the rotational speed of the blower, or even, the *natural noise* which affected by the wind and the climate. Recent work have already generalized an outstanding formulation with an auto-regressive manner. However, we observe that prior work model the rotational speed of the blower as the fixed rule, while such schemes fail to achieve the optimal performance. For example, To that end, it's trivial to dynamically adjust the current speed w.r.t the income, such as the past storage energy and the past speed.

In this study, we ask if deep learning can help taming the complexity of controlling the rational speed for solar heating systems. Motivated by this quest, we first analysis the overall performance of state-of-the-art supervised learning methods meets solar heating systems, in which the neural network (NN) method is composed of the vanilla fully connected [5], Conv-1D [11], Long short-term memory (LSTM) [4], Gated Recurrent Unit (GRU) [3], as well as Echo State Network (ESN) [6]. We train and test the proposed schemes under both noise and no-noise scenario.

Next, we model the optimization process as a Markov Decision Process (MDP), and utilize a neural network (NN) to determine the proper rotational speed for each resolution *autoregressively*. Specifically, DeepLadder employs Dual-PPO [14], the state-of-the-art DRL method, to learn the policy by interacting with the environments without any presumptions, attempting to balance a variety of goals such as maximizing overall solar energy, reducing rotational speed as well as avoiding the frequent change of the speed. Therefore, such *many-goal* technique enables DeepSolar to optimize the rotational speed w.r.t. maximizing the reward function.

Finally, we summarize the contributions as follow:

- We have done the Question 1 (sampling 2000 samples for training and 400 for testing) in §2.1.
- We have finished Question 2 (adopting 1D-CNN layers to train the dataset) in §2.2.
- For Question 3 (training the NN with the noise dataset), we have solved it in §2.3.
- We have discussed the performance of ESN in §2.2 and §2.3 (Question 4). Meanwhile, we also compare another NN architectures, such as fully connected, LSTM, as well as GRU.
- We have open-sourced the code in https://github.com/godka/DeepSolar.
- In addition, we propose a novel deep reinforcement learning-based approach which dynamically adjust the rotational speed of the blower for solar heating system (§3).

2 PRELIMINARIES

In this section, we attempt to tap the potential for various typical learning-based schemes. We choose several representational learning algorithms from both academic and industry, including

(1) Fully-connected neural networks (FC): uses a single layer with 32 neurons to predict future video quality;

* Corresponding Author.

Conference'17, July 2017, Washington, DC, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

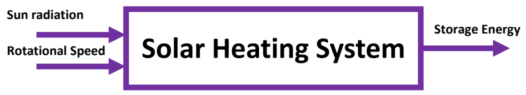

Figure 1: Solar Heating System Overview

Table 1: Comparing performance (MSE) of different regression algorithms. Results are collected without white noise.

Method	MSE	Improvement(%)
FC(Baseline)	0.00643	-
LSTM	0.00756	-17.54
GRU	0.00537	16.51
1D-CNN	**0.00153**	**76.21**
ESN	0.00443	31.13

(2) Recurrent Neural Network (RNN): uses a conventional single-layer recurrent neural network with 32 hidden units;
(3) Gated Recurrent Unit (GRU): employs a recurrent neural network with 32 hidden units.
(4) Conv-1D (1D-CNN): leverages an 1D-CNN for extracting features. We set the filter number as 128, the filter size as 4.
(5) ESN: utilizes several echo state network (ESN) for representing the NN. We set the reservoir as 50.

2.1 System Modeling

Figure 1 shows the dynamic model of the solar heating system. The solar heating system works as follows: the solar radiation radiates on the windsurfing board and heats the air in it, and then the warm air is sent into the heat storage cylinder by the blower to heat the house. The *nonlinear autoregressive moving average model* (NARMA) of the solar heating system is defined as Eq. 1,

$$\begin{aligned} y_t = & (1-d_1)y_{t-1} + (1-d_3)\frac{y_{t-1}u_{t-1}}{u_{t-2}} \\ & + (d3-1)(1+d_1)\frac{y_{t-2}u_{t-1}}{u_{t-2}} \\ & + d_0 d_2 u_{t-1} I_{t-2} - d_0 u_{t-1} y_{t-1} \\ & + d_0(1+d_1)u_{t-1}y_{t-2} \end{aligned} \tag{1}$$

where $d_0 = 0.3$, $d_1 = 0.6$, $d_2 = 2.0$, $d_3 = 1.3$, I_t represents the degree of solar radiation, u_t reflects the default rotational speed for the blower. In this work, we refer I_t as Eq. 2, u_t as Eq 3.

$$I_t = -\frac{t-700^2}{700^2} + 1 \tag{2}$$

$$u_t = \begin{cases} 0.12 & 0 \le t \le 349 \\ 0.25 & 349 < t \le 528 \\ 0.5 & 528 < t \le 872 \\ 0.25 & 872 < t \le 1042 \\ 0.12 & 1042 < t \le 1400 \end{cases} \tag{3}$$

Table 2: Comparing performance (MSE) of different regression algorithms. Results are collected with white noise.

Method	MSE	Improvement(%)
FC(Baseline)	0.005787	-
LSTM	0.006781	-17.18
GRU	0.005511	4.78
1D-CNN	**0.003612**	**37.58**
ESN	0.003999	30.90

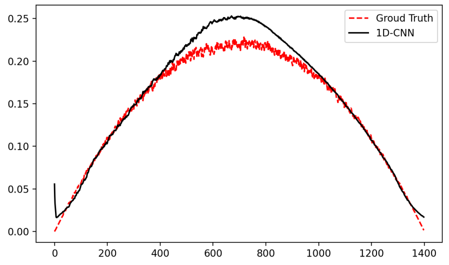

Figure 2: 1D-CNN vs. Ground Truth

2.2 Learning without Noise

We first compare the performance of the assigned learning-based algorithms over the solar heating system. In detail, we generate 2,000 samples for training, and 400 samples for testing. Moreover, each algorithm will be trained about 50 epochs. We set the learning rate as 1.0×10^{-4}, and employ Adam optimizer [7] to optimize the NN. Table 1 reports the results for each method, in which the performance are evaluated with mean square error (MSE). We can see that 1D-CNN achieves the best performance among all candidates, with the improvements on average MSE of 76.21% compared to the baseline scheme. Moreover, we find that ESN also performs well with lower training overhead. Surprisingly, the modern NN model, i.e., LSTM and GRU fails to yield a reliable result in our task. It makes sense since too many inputs may interfere with the final performance of the model.

2.3 Learning with Noise

Next, we aim to learning the model with the noise dataset. Specifically, the dataset will be generated by adding white noise, ranging from 0.95 − 1.05. Results are reported in Table 2. We can find the same conclusion: 1D-CNN works best, while LSTM and GRU performs not well. One interesting observation is that the 1D-CNN network is sensitive to the noise distribution, which heavily degrades nearly 30% compared with learning from the tabular-rasa data. On the contrary, we can see that ESN also works well, or even betters the previous no-noise network. What's more, we plot the curve of 1D-CNN model and ground truth in Figure 2. Here we apply an fully connected layer (sized 64, 64, and 32) behind the 1D-CNN layer.

2.4 Generalization

In summary, exploring the aforementioned experimental results, we observe that deep learning technologies have the abilities to handle the solar heating system, as such schemes can precisely predict the future energy. Thus, we attempt to start the next challenge: considering that the default fixed speed or rules can hardly tackle all considered scenarios (for example, the different sun radiation between the sunny day and the rainy day), is it possible to leverage deep reinforcement learning method for generalizing a policy to optimize the rotational speed? In this paper, we propose DeepSolar, a novel NN-based system that can control the rotational speed for solar heating system.

3 DEEPSOLAR SYSTEM MECHANISM

DeepSolar leverages an NN-based decision model for determining the proper rotational speed for each step. Note that DeepSolar is a zero-shot learning approach, that means, we can train the model once in the offline stage. Meanwhile, the model will be inferenced without tuning on the online stage. In this section, we start by introducing DeepSolar's decision core, including its state, action, reward, NN representation. Then, we show a detailed training methodology. Finally, we report the evaluation setup and the experimental results.

3.1 NN Overview

We first explain the details of the NN, including states, actions, reward definition, as well as the NN architecture.

State. We mainly categorize the DeepSolar's input into three sequences, that is, past speed u_{t-2}, u_{t-1}, past collected energy y_{t-2}, y_{t-1}, and the current step t.

Action. The action space of DeepSolar is a 10-dim-discrete vector, representing the speed of the blower u_t for the next time step t. Here we bound the speed in the range of $[0.14, 0.5]$.

Reward. We list the instant reward function r_t in Eq. 4. Here we consider three factors: i) we aim to maximize the energy y_t while ii) minimizing the rotational speed u_t (too much speed may waste the overall energy) and iii) preserving the smoothness $|u_t - u_{t-1}|$ (in practice, we don't hope the speed change continuously and frequently.) α and β is the hyper-parameter which controls the importance of each metric. In this work, we set $\alpha = 1$ and $\beta = 1$. It's notable that we can use any hyper-parameter to represent the reward function.

$$r_t = y_t - \alpha u_t - \beta |u_t - u_{t-1}| \tag{4}$$

NN Architecture. For the network state representation, we adopt use three 128-dim fully connected layers to extract the features. The outputs of the DeepSolar are the policy network and the value network, in which the activation function of the policy network is *softmax* and we set *linear* function for value network.

3.2 Policy Optimization.

Our key idea is to improve the policy via *elevating* the probabilities of the high-reward-samples from the collected trajectories and *diminishing* the possibilities of the failure sample from the worse trajectories. In other words, the improved policy π at state s_t is required to pick the action a_t which produced the best accumulated reward R_t, i.e., $a_t = argmax_a E[R_t(s_t, a)]$. In this work, We use Dual-clip Proxy Policy Optimization (Dual-PPO) [14], a state-of-the-art on-policy DRL method, as the basic reinforcement learning algorithm. The gradient of policy network are computed as Eq. 7, where $\hat{A}_t$ is the advantage function (Eq. 9), $p_t(\theta)$ denotes the probability ratio between the policy π_θ and the old policy $\pi_{\theta_{\text{old}}}$ (Eq. 5), ϵ and c are hyper-parameters that control how to clip the gradient. We set $\epsilon = 0.2$, $c = 3$ as consistent with the original paper [14]. Briefly, the Dual-PPO algorithm adopts dual-clip method to restrict the step size of the policy iteration and update the NN by minimizing the following *clipped surrogate objective*. If $\hat{A}_t > 0$, Dual-PPO will work equal to the original PPO algorithm [12]. Otherwise, Dual-PPO will clip the ratio $p_t(\theta)$ with a lower bound of the value $\hat{A}_t$.

$$\pi_{\theta_{\text{old}}} = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{5}$$

As listed Eq. 10, the value network V_{θ_p} is updated via minimizing the mean square error (MSE) between R_t and the estimated value $V_{\theta_p}(s)$. At the same time, we also add entropy of the policy $H(s_t;\theta)$ into the loss function to encourage exploration feedback. Here β is the entropy weight. To sum up, we summarize the loss function $\mathcal{L}^{DeepSolar}$ in Eq. 11.

$$\mathcal{L}^{PPO} = \min\left(p_t(\theta)\hat{A}_t, \text{clip}\big(p_t(\theta), 1-\epsilon, 1+\epsilon\big)\hat{A}_t\right). \tag{6}$$

$$\mathcal{L}^{Policy} = \begin{cases} \hat{\mathbb{E}}_t[\max(\mathcal{L}^{PPO}, c\hat{A}_t)] & \hat{A}_t \le 0 \\ \hat{\mathbb{E}}_t[\mathcal{L}^{PPO}] & \hat{A}_t > 0 \end{cases} \tag{7}$$

$$R_t = r_t + \gamma R_{t+1}. \tag{8}$$

$$\hat{A}_t = r_t + \gamma R_{t+1} - V_{\theta_p}(s_t). \tag{9}$$

$$\mathcal{L}^{Value} = \frac{1}{2}\hat{\mathbb{E}}_t\left[V_{\theta_p}(s_t) - R_t\right]^2. \tag{10}$$

$$\nabla\mathcal{L}^{DeepSolar} = -\nabla_\theta\mathcal{L}^{Policy} + \nabla_{\theta_p}\mathcal{L}^{Value} + \nabla_\theta \beta H(s_t;\theta). \tag{11}$$

3.3 Adaptive Entropy Weight

Notice that the training results of recent on-policy reinforcement learning methods are sensitive to the entropy weight β [8]. E.g., if the value of the weight is too small (e.g. default settings in most OpenAI gym [2] environments is 0.01.), the overall training will be converged quickly but fail to achieve optimal performance. On the contrary, if the weight is too big (e.g., in practice, the starting weight of Pensieve [9] is 5.), the value network will fail to estimate the accurate baselines due to the high variance of collected trajectories. As a result, the learned policy eventually lacks stability compared with the optimal strategy. To alleviate this issue, we propose a novel training trick called adaptive entropy weight decay to dynamically adjust the entropy weight β_t during the training process. In detail, unlike typical reinforcement learning tasks that train and test the model on the same dataset, we split the collected dataset into two datasets, i.e., a training set and a validation set. DeepSolar is trained over the training set and validated on the validation set every T steps. In the beginning, the entropy weight is initialized at β_0. Then the weight will be decreased if the overall performance over the validation set has no longer been improved for K times. At step t, the entropy weight is updated as $\beta_t = \eta\beta_{t-1}$, in which η means the decay rate. In this work, we set $\beta_0 = 1$, $\eta = 0.8$.

3.4 Parallel Training

During the training process, we observe that the training progress is inefficient while using a single process. Inspired by the multi-agent training method [10], we modify DeepSolar's training in the single agent as training in multi-agents. Multi-agents training consists of two parts, a central agent and a group of forwarding propagation agents. The forward propagation agents only decide with both policy and critic via state inputs and neural network model received by the central agent for each step; then it sends the n-dim vector containing $\{s, a, r\}$ to the central agent. The central agent uses the actor-critic algorithm to compute gradient and then updates its neural network model. Finally, the central agent pushes the updated network parameters to each forward propagation agent. Note that this can happen asynchronously among all agents, for instance, there is no locking between agents. By default, DeepSolar employs 16 forward propagation agents and one central agent.

3.5 Training Methodology

We now describe DeepSolar's training methodology. See alg. 1 for more details.

Algorithm 1 DeepSolar Overall Training Procedure

```
Require: Training model $\theta$.
 1: procedure DeepSolar Training
 2:     Get initial state $s_t$.
 3:     repeat
 4:         Picks $a_t$ according to policy $\pi(s_t; \theta)$.
 5:         Produces the next state $s_{t+1}$ according to $s_t$ and $a_t$.
 6:         Receives the instant reward $r_t$ w.r.t Eq. 4.
 7:         if done then            ▷ End of the construction process.
 8:             Estimates the accumulative reward $R$ using Eq. 8.
 9:             Computes the advantage $A$ using Eq. 9.
10:             Update $\theta$ with the collected batch using Eq. 11.
11:             Get initial state $s_t$.
12:         end if
13:         $t \leftarrow t + 1$
14:     until Converged
15: end procedure
```

4 EVALUATION

4.1 Implementation

We use TFlearn [13] to implement the NN and leverage TensorFlow [1] to construct DeepSolar. DeepSolar consists of two NNs, policy network and value network. Policy network takes a n-dims vector with *softmax* active function as the output. The value network outputs a value with *linear* function. Moreover, we use the same set of hyper-parameters for training the NN: learning rate $\alpha = 10^{-4}$, and the entropy weight decay $\zeta = 0.8$. In addition, we adopt Adam optimizer [7] with default settings.

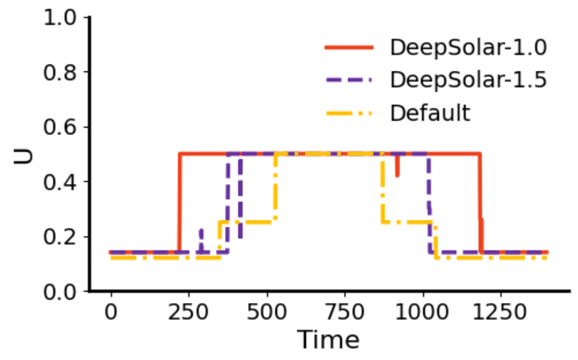

Figure 3: Comparing the Rotational Speed of DeepSolar and default settings in the solar heating system.

Table 3: Detailed List.

Method	y	u	$y-u$	$\lvert u_t - u_{t-1}\rvert$
DeepSolar-1.0	0.545	0.388	0.157	0.0008
DeepSolar-1.5	0.440	0.306	0.139	0.0010
Default	0.34	0.2457	0.097	0.0005

4.2 Experimental Results

The results of DeepSolar is shown in Figure 3. DeepSolar performs different from the previous algorithm. Unlike the default scheme, DeepSolar attempt to directly achieve the highest speed in the very early stage (almost step 250), and in the meanwhile, reducing the speed to the lowest value in the very late stage. Table 3 illustrates that DeepSolar can effectively improve the overall energy, with the improvements of 58.82% compared with the default scheme. It's notable that DeepSolar consumes more cost in terms of the rotational speed compared to the original approach, while such extra cost is worth it since it increases higher efficiency of 61.86% compared with the baseline method. In particular, we evaluate DeepSolar with different reward parameter α. Results demonstrate that the deep reinforcement learning method can generalize the outstanding strategy for any form of reward functions.

5 CONCLUSION

In this work, we investigate how to use deep learning method in the typical solar heating systems. We show that recent deep learning methods can precisely predict the future storage energy. What's more, we argue that the rotational speed of the blower is not always standing for the optima. Thus, we propose DeepSolar, a deep reinforcement learning-based speed control system. Trace-driven experiments reveal that DeepSolar can achieve state-of-the-art (SOTA) performances compared with previously proposed baselines.

ACKNOWLEDGEMENT

Special thank to our teacher, Prof. Zhidong Deng, for his great help and kind suggestions. It's really a nice journey, that is, fully finished the course on the cloud. Hope to meet everyone in the real-world and, let's combat the COVID-19 together!

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A System for Large-Scale Machine Learning.. In *OSDI*, Vol. 16. 265–283.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[3] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[4] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 1999. Learning to forget: Continual prediction with LSTM. (1999).

[5] Robert Hecht-Nielsen. 1992. Theory of the backpropagation neural network. In *Neural networks for perception*. Elsevier, 65–93.

[6] Herbert Jaeger. 2007. Echo state network. *scholarpedia* 2, 9 (2007), 2330.

[7] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[8] Junjie Li, Sotetsu Koyamada, Qiwei Ye, Guoqing Liu, Chao Wang, Ruihan Yang, Li Zhao, Tao Qin, Tie-Yan Liu, and Hsiao-Wuen Hon. 2020. Suphx: Mastering Mahjong with Deep Reinforcement Learning. *arXiv preprint arXiv:2003.13590* (2020).

[9] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *SIGCOMM 2017*. ACM, 197–210.

[10] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*. 1928–1937.

[11] Mostafa Naghizadeh and Mauricio D Sacchi. 2009. Multidimensional convolution via a 1D convolution algorithm. *The Leading Edge* 28, 11 (2009), 1336–1337.

[12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[13] Yuan Tang. 2016. TF. Learn: TensorFlow's high-level module for distributed machine learning. *arXiv preprint arXiv:1612.04251* (2016).

[14] Deheng Ye, Zhao Liu, Mingfei Sun, Bei Shi, Peilin Zhao, Hao Wu, Hongsheng Yu, Shaojie Yang, Xipeng Wu, Qingwei Guo, et al. 2019. Mastering Complex Control in MOBA Games with Deep Reinforcement Learning. *arXiv preprint arXiv:1912.09729* (2019).