Neural Network Solutions to Witsenhausen problem

Jiaojiao Fan
GTID:903565753
Email: jiaojiaofan@gatech.edu

***Abstract*—In this report, several neural networks with different structures are implemented to solve Witsenhausen problem. Other improving strategies include optimizers, initializations and forced function fixing. Finally, the result are compared with former people and a better result is obtained. Also, the shortcoming of the neural network also shows in this project. The neural network may be stuck into a near local minima.**

1. Introduction

In this report, we proposed several solutions to the well-known and still unsolved Witsenhausen counterexample. [1] There have been some meaningful tries to detect the global minima of the min problem, such as Lee [2]and M. Barglietto [3] Some of their manipulations are also refered in this project. Other than that, thanks to the development of the neural networks, many other meaningful attempts are also taken such as input convex neural network (ICNN) structure [4] Different results would be listed to show the effect.

2. The Witsenhausen Counterexample

The Witsenhausen counterexample has been outstanding for more than 50 years. It is formulated by Hans Witsenhausen in 1968.[1] It is a counterexample to a natural conjecture that in a system with linear dynamics, Gaussian disturbance, and quadratic cost,affine control laws are optimal to minimize the cost. However, Witsenhausen courterexample, shown in figure below, has nonlinear control laws

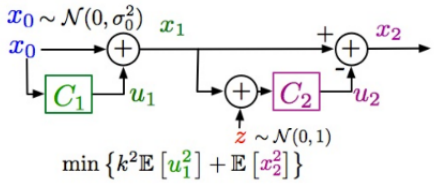


Figure 1: Witsenhausen couterexample

that outperform all linear laws.

$$f(x) = \gamma_1(x) + x \qquad g(x) = \gamma_2(x) \tag{1}$$

As a result, $f(x_0) = x_1$ and $g(x_1 + z) = u_2$. Out goal is to minimize the quadratic cost:$k^2\,\mathbb{E}[U_1^2] + \mathbb{E}[X_2^2]$, which can also be written as

$$\begin{aligned}\min_{f,g} J^C(f,g) :=& k^2\,\mathbb{E}[f(X_0) - X_0]^2 + \\ & \mathbb{E}[(f(X_0) - g(f(X_0) + N))^2]\end{aligned} \tag{2}$$

In equation (2), there is a parameter k^2, which in fact determines the cost gap between the linear controller and nonlinear controller[3]. If k^2 is smaller, the gap is bigger. For better comparison with the results got by previous researchers, k^2 is set as 0.04 in this report.

However, there is not only one way to represent costs.[5] We can view Witsenhausen problem from the optimal transport theory. In this way, we could rewrite the cost in another equivalent way:

$$\begin{aligned}&\min_{Q} J^W(f,g) \\ :=& k^2 W_2(P,Q)^2 + mmse(Q,\sigma^2) \\ :=& k^2 W_2(P,Q)^2 + \min_{g} \mathop{\mathbb{E}}_{\substack{X_1\sim Q \\ N\sim\mathcal{N}(0,1)}} [(g(X_1+N) - X_1)^2]\end{aligned} \tag{3}$$

which is the same with

$$\min_{Q,g} = k^2 W_2(P,Q)^2 + \mathop{\mathbb{E}}_{\substack{X_1\sim Q \\ N\sim\mathcal{N}(0,1)}} [(g(X_1+N) - X_1)^2] \tag{4}$$

where W_2 means Wasserstein-2 distance. Compared to the equation (3), equation (2) could be called as classic Witsenhausen cost.

In addition, it is already known that $f(x)$ must have some strict property to be optimal [5] :

- *Any optimal controller f is a strictly increasing unbounded piecewise real analytic function with a real analytic inverse*

This means f has to be smooth enough. But interestingly, the neural network(NN) optimized result is exactly opposite from this property. The sharper the f becomes (opposite to smooth), the smaller the cost is.

3. Basic Neural Network Setup

The whold process could be generally separated into 4 parts:
1) Initialization setup for the f net and g net
2) Train the NNs using Gaussian distribution data. In this

report, all data keeps the consistency: $x_0 \sim \mathcal{N}(0, \sigma^2)$, where $\sigma = 5$, and $N \sim \mathcal{N}(0, 1)$.
3) Fix f net and continue to train g net.

3.1. Neural Network Architecture

Basically, two NNs are taken to represent f and g seperately using Pytorch structure. [1] For f net and g net, all layers are linear layers and activated by CELU[6] function, i.e.

$$CELU(x) = max(0, x) + min(0, \alpha * (exp(x/\alpha) - 1)) \quad (5)$$

is used since it makes the activation function continuously differentiable and improves the performance in initialization setup process. It is worth noting that CELU is convex and monotone increasing activation function. For f net structure, we tried ICNN as f net's integral function:F net, i.e. f NN works as the derivative function of the function represented by F NN. Why can we do this? Because f is proved to be monotone, so F must be convex. And ICNN could has the ability to represent all convex functions. That's why we choose ICNN to represent F here. In this way, x_1 in Fig.1 is got from taking back propogation of the F NN from the output to the input. The fully ICNN structure is as below and we stricly used the same structure.

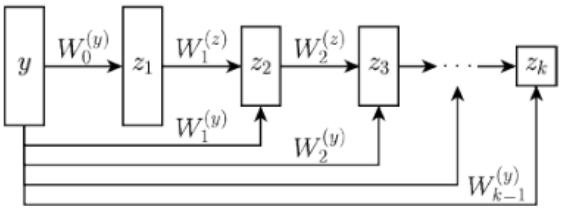


Figure 2: fully ICNN structure

The function f is convex provided that all $W_{1:k-1}^{(z)}$ are non-negative and all activation functions are convex and non-decreasing.[4] To be noticed, sometimes we cancel the non-negative weight restriction but the monotone property of f NN doesn't change. In the meantime, we could use this as a way to improve the loss decreasing.

We also have tried ResNet[7] since it performs much better than ordinary linear layer NNs.

On the other hand, the g net is treated as normal linear layer NN if f is ICNN structure and ResNet if f is ReNet. The experiment proves that f net is much more important than g net for loss decreasing.

3.2. Optimizer

For updating parameters of two NNs, we first use Stochastic Gradient Descent (SGD) and then used ADAM. In comparison, SGD performs much slower and becomes not stable while entering plateau of loss decreasing. ADAM increases the stability and speed a lot. So we focus on the better optimizer compared to ADAM later. There were a lot of variation of ADAM during the past several years. We mainly care whether the optimizer could lead the NN to the global optimizer. Some outstanding optimizers came out like AdaBound[8], RAdam[9] and Yogi[10]. [2] We tried to use RAdam and Yogi but found Yogi didn't work well in this problem picture at all. So we finally proposed to use RAdam for main part of the project. But we will still list the result of RMSprop and Adam as the comparison.

3.3. Initialization

The f net and g net both have initialization. According to the previous researchers' work, we choose the 7-step-stair as basic two NN initialization. This is refering to the best result got from past researchers and the step parameters are from Yu-Chi Ho's group[9] as below:

$$f(x) = \begin{cases} 0 & 0 \le x < 3.25 \\ 6.5 & 3.25 \le x < 9.90 \\ 13.2 & 9.90 \le x < 16.65 \\ 19.9 & 16.65 \le x. \end{cases} \quad (6)$$

According to symmetry, the $x \ge 0$ part of $f(x)$ could be obtained. We also tried segmented stair, which parameters are also obtained from Yu-Chi Ho's group[2]:

$$f(x) = \begin{cases} 0.00 & 0.00 \le x < 0.65 \\ 0.05 & 0.65 \le x < 1.95 \\ 0.10 & 1.95 \le x < 3.25 \\ 6.40 & 3.25 \le x < 4.58 \\ 6.45 & 4.58 \le x < 5.91 \\ 6.50 & 5.91 \le x < 7.24 \\ 6.55 & 7.24 \le x < 8.57 \\ 6.60 & 8.57 \le x < 9.90 \\ 13.10 & 9.90 \le x < 11.25 \\ 13.15 & 11.25 \le x < 12.60 \\ 13.20 & 12.60 \le x < 13.95 \\ 13.25 & 13.95 \le x < 15.30 \\ 13.20 & 15.30 \le x < 16.65 \\ 19.90 & 16.65 \le x. \end{cases} \quad (7)$$

To distinguish, we call (6) as the 7-stair-init and (7) as the segmented-7-stair-init.

We mark f_0 as the result from initialization process for f and the same for g_0. And we mark the integral function of f_0 as F_0. For example, if the f_0 is (6), then its corresponding F_0 looks like below:

It is worth mentioning that while doing initialization training for ICNN f net,we are not training f_0 directly but training F_0. So the loss criteria in the initialization training

1. The code of this project could be found at: https://github.com/sbyebss/Witsenhausen

2. The pytorch optimizer source code could be found here: https://github.com/jettify/pytorch-optimizer

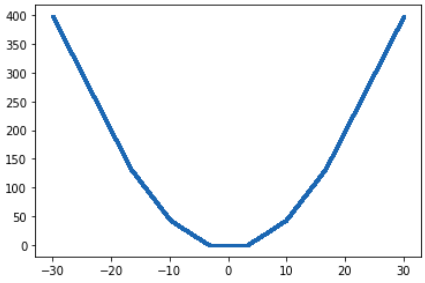


Figure 3: The corresponding F_0 when f_0 is taken as Equation (6)

may be two ways. Firstly, if we get the loss by comparing F NN's derivative with the equation (6) or (7) directly, the trained f would be very bad which only shapes 2-step as Fig.4:

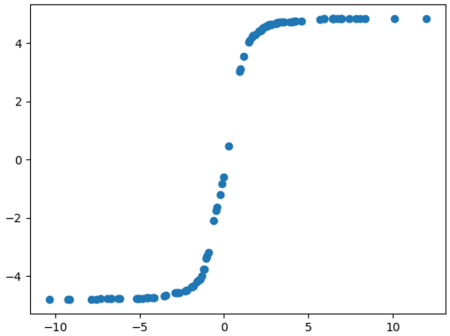


Figure 4: f_0 from initialization process with criteria between F_0 derivative and the 7-stairs-init directly

Secondly, if we treat the loss criteria as comparison between F with the piecewise linear function F_0, the f_0 would becomes much better as Fig.5 and Fig.6:

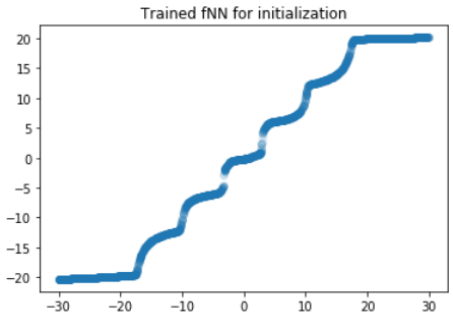


Figure 5: 7-step-stair f_0, f_0 is from initialization process with criteria between F_0 and ideal integral function

3.4. Computational Graph

From the classic view of Witsenhausen problem(2), the loss is directly taking MSE loss between the compared batch. So the computational graph is shown as below:

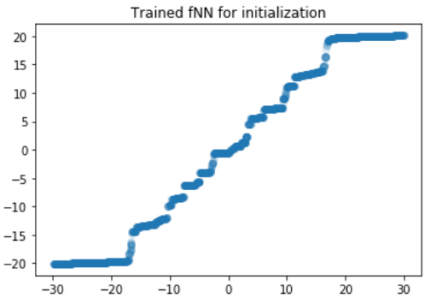


Figure 6: 7-step-stair f_0, f_0 is from initialization process with criteria between F_0 and ideal integral function

However, if choosing the optimal transport expression (4),

3.5. Test Data Set

In the process of training, we choose to get the test set loss for every 1000 epoches. Therefore, a good test data size need to be chosen. The variance of the empirical test loss is expected to be less than 0.005^2 to be compared successfully with previous results. If take the test data size as 5×10^5, the empirical variance of the test loss is 8.65791389375e-7. Finally, we chose the test data size as 1×10^5

4. J^C Experiment Result

If there is no more specification, the following results are corresponding to the classic cost expression (2). The detailed computation graph could be seen in 3.4 section.

4.1. ICNN structure

In this subsection, we present the results of setting f NN as ICNN.

1) **No initialization**: The f net could only go to two-step and total loss J is around 0.36. So does g net. This proves that the final NN rely on the initialization a lot. And the optimizer couldn't lead the f net to very complex steps shape without initialization.

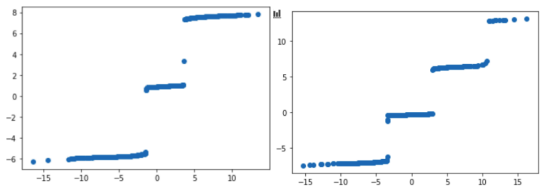
2) **7-step-stair Initialization**:

For this part, the f net has 6 layers and the g net has only 3 layers.

TABLE 1: $J^C(f, g)$ Comparison When f is with 7-step-stair Init and ICNN Structure

Optimizer	J^C
Adadelta	0.286
RMSprop	0.250
RAdam	0.220
Adam	0.216

It can be seen that RAdam and Adam perform best and RAdam is more stable and faster than Adam. Because while



(a) 3-step f (b) 4-step f

Figure 7: Different result caused by Adam instability

we use Adam, the f may update from 5-step to 3-step but 5-step performs better than 3-step. This proves that Adam may lead f deviate the (local) minima to a worse NN. The worse thing is that while we use Adam, we couldn't reproduce the same result. With the same initializations and NN structure setup, sometimes Adam leads f to be 3-step finally but sometimes 4-step finally as below.

As a result, afterwards if there is no more specification, we always use RAdam as the optimizer. This is very foundamental result for the experiments afterwards.

3) **Segmented-7-stair-init**: In this part, we began trying not to use ICNN weights clamping and the optimizer is always RAdam.

TABLE 2: $J^C(f, g)$ Comparison When f is with segmented-7-stair-init and ICNN Structure

f structure	g structure	ICNN Clamping	J^C	Remark
7 layers 300n/l	3 layer 100n/l	Yes	0.2050	
	3 layers 100n/l	No	0.1865	
	3 layers 300n/l	No	0.1758	fix f, then update g: $J^C \to 0.1754$
	6 layers 300n/l	No	0.1739	fix f, then update g: $J^C \to 0.1746$
fixed segmented 7 stair function	6 layers 300n/l	No	0.1670	fix g, then update f: $J^C \to 0.1782$

In the Table 2, n/l means the number of units each layer. From which we could tell the f structure matters a lot. If f is not shaped well, sometimes continuing updating f with f fixed would only make the cost worse.

The f and g shape of the best result: J^C=0.167 is shown below:

In the remark, we continue to update only one NN while fixing another NN which was saved from the lowest cost point. The final f result for the last array is shown below:

We could tell from the figure that the f is stuck in 5-step-stair, which is local minima compared to better result got from 7-step-stair. In fact, this is also closely related to the x_0 distribution. Because x_0 has very little data which locates bigger than 15 or smaller than -15. So it's hard for optimizer to detect a good shape in those areas.

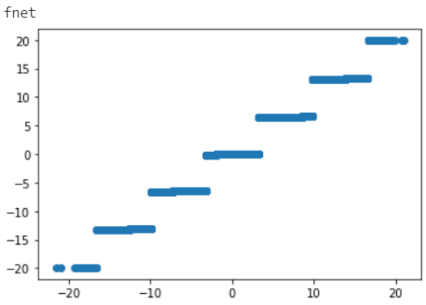


Figure 8: Fixed f as a step function to obtain best result:0.167

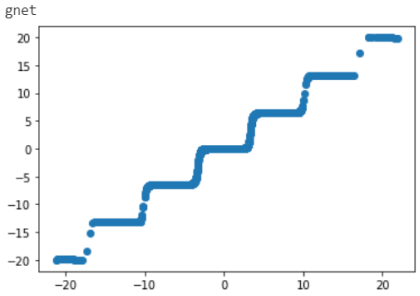


Figure 9: g solution while taking f fixed as a step function

4.2. ResNet structure

This didn't behave better than ICNN structure. While taking f as 6 layers and 300n/l, g as 6 layers and 300n/l, $J^C \to 0.2153$ as below:

5. J^W Experiment Result

In this section, we don't have a well-organized initializations for f as there is no previous paper reference. But the result is very closed to the linear controller. This proves that the Wasserstein distance formula still needs some great initializations to improve the solution, which could be saved for the future research.

1) **No initialization**:

The same as the J^C, the cost goes to 0.961

1) **Train h firstly**: If we first train h with the hope that it could get a better initialization, then the cost still goes to 0.961.

6. Discussion

From the experiment result we could see the best result of this project comes when f is fixed as a segmented 7 step piecewise-constant function. This is opposite to the property in the paper[5], which asks the f has to be smooth enough.

Information for code

Different train process are edited in train_*specificName*.py file.

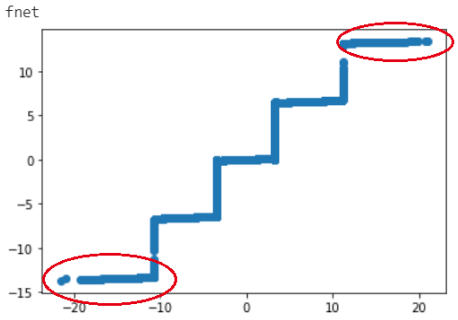


Figure 10: Final f solution of updating f with g fixed

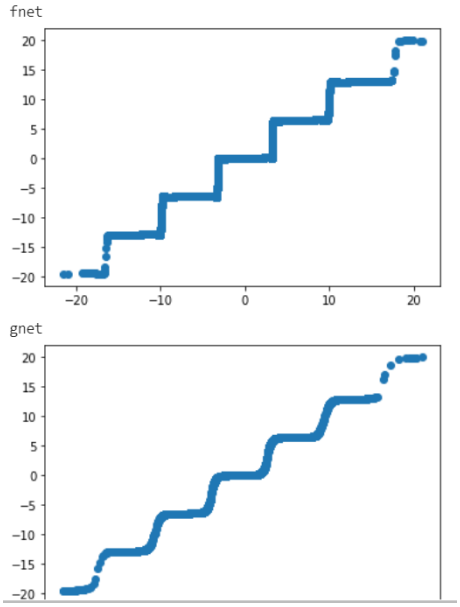


Figure 11: Final f solution of updating f with g fixed

The *modules* folder saves the network structure and ICNN weight clamping function.

The *runs* folder record the tensorboard loss file. Importing those files into tensorboard, we can compare the loss decreasing trend in one plot.

The *data* folder saves the main body training data as well as initialization training data. Specifically, the generate_data.ipynb file stores the process and result of generating all kinds of data needed in this project.

The *test* folder saves some test files which verified some important rudimentary ideas.

The *model* folder saves the different NN parameters for initialization or some trained solution NNs.

Acknowledgment

Thanks to my advisor: Dr. Yongxin Chen and the other PhD studens: Rahul Singh and Qinsheng Zhang for their kind help. They give me a lot of ideas about how to improve the performance during the process of project.

References

[1] H. S. Witsenhausen, "A counterexample in stochastic optimum control," *SIAM Journal on Control*, vol. 6, no. 1, pp. 131–147, 1968.

[2] J. T. Lee, E. Lau, and Y.-C. Ho, "The witsenhausen counterexample: A hierarchical search approach for nonconvex optimization problems," *IEEE Transactions on Automatic Control*, vol. 46, no. 3, pp. 382–397, 2001.

[3] M. Baglietto, T. Parisini, and R. Zoppoli, "Numerical solutions to the witsenhausen counterexample by approximating networks," *IEEE Transactions on Automatic Control*, vol. 46, no. 9, pp. 1471–1477, 2001.

[4] B. Amos, L. Xu, and J. Z. Kolter, "Input convex neural networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 146–155.

[5] Y. Wu and S. Verdú, "Witsenhausen's counterexample: A view from optimal transport theory," in *2011 50th IEEE Conference on Decision and Control and European Control Conference*. IEEE, 2011, pp. 5732–5737.

[6] J. T. Barron, "Continuously differentiable exponential linear units," *arXiv preprint arXiv:1704.07483*, 2017.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[8] L. Luo, Y. Xiong, Y. Liu, and X. Sun, "Adaptive gradient methods with dynamic bound of learning rate," *arXiv preprint arXiv:1902.09843*, 2019.

[9] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," *arXiv preprint arXiv:1908.03265*, 2019.

[10] M. Zaheer, S. Reddi, D. Sachan, S. Kale, and S. Kumar, "Adaptive methods for nonconvex optimization," in *Advances in neural information processing systems*, 2018, pp. 9793–9803.