Cruz, Hayes, Silva, Santos (eds.)

Deducing Decision by Error Propagation

Brown Wang
brownwang0426@gmail.com
Science and Technology Law Institute, Institute for Information Industry
Taipei, Taiwan, R.O.C

ABSTRACT

We introduce a very simple and general way for deep neural network to perform deductive reasoning based on error backpropagation. Our idea is to view state and action as the input data, and reward as the output data. Then by trained deep neural network and a sequence of known state and unknown action, we update the unknown action by error backpropagation with given optimal reward as target. Then the agent adopts the updated action. This approach can be applied to various types of digital environment. Moreover, in contrast to recent approaches, it does not require Bellman function or other hand-written functions. We provide experimental results in this paper.

KEYWORDS

deep learning, neural networks, error backpropagation, deductive reasoning

ACM Reference Format:
Brown Wang. 2022. Deducing Decision by Error Propagation. In *Proceedings of Proc. of the Adaptive and Learning Agents Workshop (ALA 2022) (ALA '22).* ACM, New York, NY, USA, 8 pages.

1 INTRODUCTION

Humans are renowned for their inductive and deductive reasoning. Deep learning successes in modeling inductive reasoning by using error backpropagation in deep neural network. A deep neural network is trained as an universal function to model $f : x \longrightarrow y$ where x is the input data and y is the output data. However, whether there exists an universal inverse function capable of modeling $f^{-1} : y \longrightarrow x$ remains questionable. Such universal inverse function might be crucial in modeling human deductive reasoning ability.

In this paper, we propose that, for a deep-learning-trained deep neural network f where $f : x \longrightarrow y$, $f^{-1} : y \longrightarrow x$ and f^{-1} can be represented by:

$$\hat{x} \longleftarrow \hat{x} - \beta \frac{\partial}{\partial \hat{x}} E\Big(y, f(\hat{x})\Big)$$

where $\hat{x}$ is an estimated input data, $f(\hat{x})$ is the neural network output by forward feeding, y is a given output data, $E(\cdot)$ is the error function, and β is the deducing rate controlling the update rate of $\hat{x}$. The purpose is to let $\hat{x}$ approximate x upon a given y through error backpropagation and find $\arg\min_{\hat{x}} E\big(y, f(\hat{x})\big)$.

The main advantage of the proposed method is that, based on trained deep neural network f where $f : a \longrightarrow r$ and given optimal reward $\acute{r}$, we can derive the optimal action $\hat{a}$ of an agent simply by using error backpropagation and $f^{-1} : \acute{r} \longrightarrow \hat{a}$. This distinguishes our method from models using Bellman function or other hand-written functions. Moreover, our experimental results show that our method can be applied to various types of digital environment requiring little modification in our proposed model.

However, the above method implies that $\hat{x}$ is been optimized upon an error surface created by a single neural network so there inevitably exists local minima problem. To solve local minima problem as stated in Rumelhart et al. [14], we propose to train multiple neural networks and randomly select a neural network for $\hat{x}$ to be optimized upon. We name this technique Multi-Weight-Matrices Stochastic Gradient Descent (MWM-SGD) to differentiate it from the traditional SGD. We now give formal technical description as below.

2 TECHNICAL DESCRIPTION

The technique comprises two phases – the learning phase and the deducing phase. The learning phase is completely deep learning (in Rumelhart et al. [14]).

2.1 The learning phase

Consider a fully connected deep neural network with $h + 1$ layers. The dimension of each layer is denoted by d_h. The weight matrix connecting the h^{th} and the next layer is denoted by $W_h \in \mathbb{R}^{d_{h+1} \times d_h}$. The weight matrices are denoted by $W \triangleq \big[W_0, W_1, \ldots, W_{h-1}\big]$. The activation function in the h^{th} layer is denoted by σ_h.

The input data matrix is denoted by:

$$X \triangleq \big[x^{(0)}, x^{(1)}, \ldots, x^{(N)}\big] \in \mathbb{R}^{d_0 \times d_{N+1}}$$

The corresponding output data matrix is denoted by:

$$Y \triangleq \big[y^{(0)}, y^{(1)}, \ldots, y^{(N)}\big] \in \mathbb{R}^{d_h \times d_{N+1}}$$

By SGD and error backpropagation, in each iteration, a pair of input data $X^{(N)} \in \mathbb{R}^{d_0}$ and output data $y^{(N)} \in \mathbb{R}^{d_h}$ is randomly selected where $x^{(N)} \sim X$ and $y^{(N)} \sim Y$[1], and the set of weight matrices W is updated by:

$$\begin{aligned} f(W, x^{(N)}) &\triangleq \sigma_h\Big(W_{h-1} \cdots \sigma_1(W_0 x^{(N)})\Big) \\ W &\longleftarrow W - \alpha \frac{\partial}{\partial W} E\Big(y^{(N)}, f(W, x^{(N)})\Big) \end{aligned} \quad (1\text{-}1)$$

where α is the learning rate, and $E(\cdot)$ is the error function.

After training, for any input data $x^{(N)} \in \mathbb{R}^{d_0}$ where $x^{(N)} \sim X$ and its corresponding output data $y^{(N)} \in \mathbb{R}^{d_h}$ where $y^{(N)} \sim Y$, it widely known that:

$$y^{(N)} \approx f(W, X^{(N)}) \quad (1\text{-}2)$$

and that:

$$y^{(N)} \approx f(W, X^{(\hat{N})}) \quad (1\text{-}3)$$

where $x^{(\hat{N})} \approx x^{(N)}$.

[1] We use "$A \sim B$" to note that A is randomly selected from B where $A \in B$, and "$A \approx B$" to note that A is approximately or close to B in a less mathematically formal term.

2.2 The deducing phase

Consider that, in the learning phase, there are $M + 1$ samples of input data matrix X, its corresponding output data matrix Y and its corresponding trained weight matrices W such that[2]:

$$\mathbb{X} \triangleq \left\{X^{(0)}, X^{(1)}, \ldots, X^{(M)}\right\}$$
$$\mathbb{W} \triangleq \left\{W^{(0)}, W^{(1)}, \ldots, W^{(M)}\right\}$$
$$\mathbb{Y} \triangleq \left\{Y^{(0)}, Y^{(1)}, \ldots, Y^{(M)}\right\}$$

Suppose there exists a common and unknown input data $x \in X^{(M)}$ for each M such that there exists its corresponding output data $y^{(M)} \in Y^{(M)}$ for each M and $\mathbb{y} \triangleq \left\{y^{(0)}, y^{(1)}, \ldots, y^{(M)}\right\}$.

To find out such common and unknown input data x, we first initialize an estimated input data $\hat{x}$. Then, by MWM-SGD[3] and error backpropagation, in each iteration, a pair of corresponding output data $y^{(M)}$ and its corresponding weight matrices $W^{(M)}$ is randomly selected where $y^{(M)} \sim \mathbb{y}$ and $W^{(M)} \sim \mathbb{W}$, and $\hat{x}$ is updated by:

$$\hat{x} \leftarrow \hat{x} - \beta \frac{\partial}{\partial \hat{x}} E\Big(y^{(M)}, f(W^{(M)}, \hat{x})\Big) \tag{2-1}$$

where β is the deducing rate, and $E(\cdot)$ is the error function.

After training, for any corresponding output data $y^{(M)}$ where $y^{(M)} \sim \mathbb{y}$ and its corresponding weight matrices $W^{(M)}$ where $W^{(M)} \sim \mathbb{W}$, we propose that:

$$y^{(M)} \approx f(W^{(M)}, \hat{x}) \tag{2-2}$$

We also propose that even if statement (2-1) is replaced by:

$$\hat{x} \leftarrow \hat{x} - \beta \frac{\partial}{\partial \hat{x}} E\Big(y^{(\hat{M})}, f(W^{(M)}, \hat{x})\Big)$$

where $y^{(\hat{M})} \approx y^{(M)}$, it still holds that:

$$y^{(\hat{M})} \approx f(W^{(M)}, \hat{x}) \tag{2-3}$$

We apply our proposed model to a few applications as below.

3 APPLICATIONS

Consider a digital (twin) environment provides state $s \in \mathbb{R}^{d_0 - d_k}$, action $a \in \mathbb{R}^{d_k}$ and its corresponding reward $r \in \mathbb{R}^{d_h}$ such that:

$$X \triangleq \left[(s,a)^{(0)}, (s,a)^{(1)}, \ldots, (s,a)^{(N)}\right]$$
$$Y \triangleq \left[r^{(0)}, r^{(1)}, \ldots, r^{(N)}\right]$$

where (s,a) is an ordered pair.

In the learning phase, by SGD and error backpropagation, in each iteration, a pair of $(s,a)^{(N)} \in \mathbb{R}^{d_0}$ and its corresponding $r^{(N)} \in \mathbb{R}^{d_h}$ is randomly selected where $(s,a)^{(N)} \sim X$ and $r^{(N)} \sim Y$, and the set of weight matrices W is updated by:

$$W \leftarrow W - \alpha \frac{\partial}{\partial W} E\Big(r^{(N)}, f\big(W, (s,a)^{(N)}\big)\Big) \tag{i}$$

In the deducing phase, consider that, in the learning phase, there are $M + 1$ samples of input data matrix X, its corresponding output data matrix Y and its correspondingly trained weight matrices W such that:

$$\mathbb{X} \triangleq \left\{X^{(0)}, X^{(1)}, \ldots, X^{(M)}\right\}$$
$$\mathbb{W} \triangleq \left\{W^{(0)}, W^{(1)}, \ldots, W^{(M)}\right\}$$
$$\mathbb{Y} \triangleq \left\{Y^{(0)}, Y^{(1)}, \ldots, Y^{(M)}\right\}$$

Suppose, given a common and known state s, there exist a common and unknown action a and an ordered pair $(s,a) \in X^{(M)}$ for each M such that there exists its corresponding optimal reward[4] $r^{(M)} \in Y^{(M)}$ for each M and $\mathbb{y} \triangleq \left\{r^{(0)}, r^{(1)}, \ldots, r^{(M)}\right\}$.

To find out such common and unknown action a, we first initialize an estimated action $\hat{a}$. Then, by MWM-SGD and error backpropagation, in each iteration, a pair of corresponding optimal reward $r^{(M)}$ and its corresponding weight matrices $W^{(M)}$ is randomly selected where $r^{(M)} \sim \mathbb{y}$ and $W^{(M)} \sim \mathbb{W}$, and $\hat{a}$ is updated by:

$$\hat{a} \leftarrow \hat{a} - \beta \frac{\partial}{\partial \hat{a}} E\Big(r^{(M)}, f\big(W^{(M)}, (s, \hat{a})\big)\Big) \tag{ii}$$

After training, given the known state s, for any corresponding optimal reward $r^{(M)}$ where $r^{(M)} \sim \mathbb{y}$ and its corresponding weight matrices $W^{(M)}$ where $W^{(M)} \sim \mathbb{W}$, it holds that:

$$r^{(M)} \approx f\big(W^{(M)}, (s, \hat{a})\big)$$

Thus, for the given known state s, the unknown action to achieve the optimal rewards for the present epoch, $\hat{a}$, is found. The agent then takes action $\hat{a}$, moves on to the next state or epoch and repeats the deducing phase til the next optimal action is found.

The above method can be viewed as having trained deep neural nets serve as overall environmental perception experiences. And by these experiences and given or desired optimal rewards, the neural nets figure out an optimal action to achieve the optimal rewards based on the present state. In this sense, not only datum but also environmental perceptions are compressed into deep neural networks as experiences (in the learning phase), allowing the the neural networks as well as the experiences to be transferred or stacked as integrated objects (in the deducing phase).

3.1 Classic Control

The source code can be downloaded at:
https://github.com/DeepDeducing/Classic_control.

We apply the above method to classic control problem in open-ai-gym. We take cartpole problem as a template. The same rationale can be generalized to other environments.

A cartpole environment is a traditional simulated physical environment where a pole needs to be constantly staying upright in the middle of the map. The cart under the pole can take two actions, namely moving right or left. The cart has to decide which direction it will move for the present epoch in order to let the pole remain upright in the middle of the map.

In the learning phase, consider s_v represents angle velocity of the pole, s_p represents position of the pole from the middle of the map, $a \triangleq a_0, \ldots, a_t$ represents a series of actions of the pole (tilting left

[2] Please note that each $X^{(M)}$ or $Y^{(M)}$ does not need to identical and each $W^{(M)}$ can be trained in parallel. See reasons afterwards.

[3] We name this MWM-SGD (Multi-Weight-Matrices Stochastic Gradient Descent) since a set of trained weight matrices is randomly selected in each iteration for SGD. It helps $\hat{x}$ escape local minima caused by initial weight variance. Also, MWM-SGD helps updated $\hat{x}$ and its neural outputs coincides with multiple given outputs as stated in statement (2-2). We can also use dropout (Srivastava et al. [18]) to boost MWM-SGD.

[4] We solve sparse reward problem by: (1) Randomizing initial state in every iteration in the learning phase. (2) Parallelly training multiple neural nets and their respective sets of weight matrices in the learning phase to enhance the probability of accessing to a sequence of optimal reward. The neural nets which learned a sequence of optimal rewards will supplement those did not. It lessens the supposition we set here and afterwards.

or right), $r_v \in \mathbb{R}^{d_h}$ is inversely proportional to the angle velocity of the pole, and $r_p \in \mathbb{R}^{d_h}$ is inversely proportional to the position of the pole (since the pole has to remain upright in the middle, higher angle velocity or higher deviation of the pole from the middle of the map results in lower reward) such that[5]:

$$X_v \triangleq \left[(s_v, s_p, a)^{(0)}, \dots, (s_v, s_p, a)^{(N)}\right]$$
$$Y_v \triangleq \left[r_v^{(0)}, \dots, r_v^{(N)}\right]$$

and that:

$$X_p \triangleq \left[(s_v, s_p, a)^{(0)}, \dots, (s_v, s_p, a)^{(N)}\right]$$
$$Y_p \triangleq \left[r_p^{(0)}, \dots, r_p^{(N)}\right]$$

Then in the deducing phase, consider:

$$\mathbb{X} \triangleq \left\{X_v^{(0)}, \dots, X_v^{(K)}, X_p^{(0)}, \dots, X_p^{(M-K)}\right\}$$
$$\mathbb{W} \triangleq \left\{W_v^{(0)}, \dots, W_v^{(K)}, W_p^{(0)}, \dots, W_p^{(M-K)}\right\}$$
$$\mathbb{Y} \triangleq \left\{Y_v^{(0)}, \dots, Y_v^{(K)}, Y_p^{(0)}, \dots, Y_p^{(M-K)}\right\}$$

Suppose, given a known state s_v and s_p, there exist unknown actions $a \triangleq a_0, \dots, a_t$ and an ordered pair $(s_v, s_p, a) \in X_v^{(K)}, X_p^{(M-K)}$ for each $K, M-K$ such that there exists its corresponding optimal reward $r_v^{(K)} \in Y_v^{(K)}, r_p^{(M-K)} \in Y_p^{(M-K)}$ for each $K, M-K$ and $\mathbb{y} \triangleq \left\{r_v^{(0)}, \dots, r_v^{(K)}, r_p^{(0)}, \dots, r_p^{(M-K)}\right\}$.

Then following statement (ii), to find out such unknown actions a, we first initialize estimated actions $\hat{a} \triangleq \hat{a}_0, \dots, \hat{a}_t$[6]. Then, by MWM-SGD and error backpropagation, in each iteration, a pair of corresponding optimal reward $r^{(M)}$ and its corresponding weight matrices $W^{(M)}$ is randomly selected where $r^{(M)} \sim \mathbb{y}$ and $W^{(M)} \sim \mathbb{W}$, and $\hat{a}$ is updated by:

$$\hat{a} \leftarrow \hat{a} - \beta \frac{\partial}{\partial \hat{a}} E\left(r^{(M)}, f\left(W^{(M)}, (s_v, s_p, \hat{a}_0, \dots, \hat{a}_t)\right)\right)$$

After training, given known state s_v and s_p, for any corresponding optimal reward $r^{(M)}$ where $r^{(M)} \sim \mathbb{y}$ and its corresponding weight matrices $W^{(M)}$ where $W^{(M)} \sim \mathbb{W}$, it holds that:

$$r^{(M)} \approx f\left(W^{(M)}, (s, \hat{a})\right)$$

Thus, for the given known state s_v and s_p, the unknown action to achieve the optimal rewards for the present epoch, $\hat{a}_0$, is found. The agent then takes action $\hat{a}_0$[7], moves on to the next state or epoch and repeats the deducing phase til the next optimal action is found.

We apply our proposed model to the other classical control environments. Under mountain-car environment, the reward is shaped to be inversely proportional to the distance between the car and the mountain hill on the right. Under pendulum environment, the reward is shaped to be inversely proportional to the cos of the theta. Under acrobot environment, there are two kinds of reward (like cart-pole). The first kind of reward is inversely proportional to the cos of the theta 1. The second kind of reward is inversely proportional to the cos of the theta 2. MWM-SGD then ensures that the updated actions and its neural output coincide with these two different kinds of optimal rewards as in statement (2-2). Lastly, We measure the quality of the deducing process of our proposed model by using the original reward in each environment and show the record in table 1.

K/M-K	Average	Std Dev
1/1	215.47	116.45
2/2	288.06	135.15
3/3	576.74	289.64
4/4	520.46	239.40
5/5	851.43	205.01

(a) Cartpole. K/M-K refers to the size of sets of weight matrices involved in MWM-SGD. Reward average and standard deviation are calculated among 100 trials. The maximum epoch for each trial is 999.

M	Average	Std Dev
1	-131.57	28.70
2	-109.26	11.65
3	-104.40	11.41
4	-104.86	11.37
5	-104.53	12.71

(b) MountainCar. M refers to the size of sets of weight matrices involved in MWM-SGD. Reward average and standard deviation are calculated among 100 trials.

M	Average	Std Dev
1	-4.6129	1.4883
2	-5.0134	1.2755
3	-3.3252	1.4072
4	-1.0197	1.5654
5	-1.1678	2.0337

(c) Pendulum. M refers to the size of sets of weight matrices involved in MWM-SGD. Reward average and standard deviation are calculated among 100 trials. The maximum epoch for each trial is 999. Please note that separate reward is the averaged-return in each trial.

K/M-K	Average	Std Dev
1/1	-199.75	18.04
2/2	-136.50	11.41
3/3	-134.01	27.13
4/4	-121.55	42.50
5/5	-117.66	15.82

(d) Acrobot. K/M-K refers to the size of sets of weight matrices involved in MWM-SGD. Reward average and standard deviation are calculated among 100 trials.

Table 1: Performance comparison under different sizes of sets of weight matrices

3.2 Tic Tac Toe

The source code can be downloaded at:
https://github.com/DeepDeducing/TicTacToe.

We further show our method can be applied to adversarial planning in strategic board game environment.

In the learning phase, consider s represents the state of the game board, $a \triangleq a_0, \dots, a_t$ represents a series of actions of players A, $b \triangleq b_0, \dots, b_t$ represents a series of actions of players B, and r_a, r_b represents corresponding reward of player A and B such that:

$$X \triangleq \left[(s, a_0, b_0, \dots, a_t, b_t)^{(0)}, \dots, (s, a_0, b_0, \dots, a_t, b_t)^{(N)}\right]$$
$$Y \triangleq \left[(r_a, r_b)^{(0)}, \dots, (r_a, r_b)^{(N)}\right]$$

Then in the deducing phase, consider:

$$\mathbb{X} \triangleq \left\{X^{(0)}, X^{(1)}, \dots, X^{(M)}\right\}$$
$$\mathbb{W} \triangleq \left\{W^{(0)}, W^{(1)}, \dots, W^{(M)}\right\}$$
$$\mathbb{Y} \triangleq \left\{Y^{(0)}, Y^{(1)}, \dots, Y^{(M)}\right\}$$

Suppose that $(\acute{r}_a, 0)$ represents optimal reward for player A, $(0, \acute{r}_b)$ represents optimal reward for player B, and that known state s, unknown actions and corresponding optimal rewards $(\acute{r}_a, 0)$, $(0, \acute{r}_b)$ for player A, B exist in each $X^{(M)}$ and $Y^{(M)}$.

Then given the known state s, to find out the unknown optimal actions for player A, we first initialize estimated actions $\hat{a}$ for player A where $\hat{a} \triangleq \hat{a}_0, \dots, \hat{a}_t$, and estimated actions $\hat{b}$ for player B where

[5] s_v, s_p and a_t can be one-hotted. r_v and r_p are vectors with activated neurons where the density of the activated neurons is proportional to reward in real value.
[6] Each $\hat{a}_t$ is initialized close to $\mathbf{0}$. And $\hat{a}$ need not to have the same length as a.
[7] This step is usually done by taking $\arg\max(\hat{a}_0)$.

$\hat{b} \triangleq \hat{b}_0, \ldots, \hat{b}_t$. Then by MWM-SGD and error backpropagation, in each iteration, $\hat{a}$ and $\hat{b}$ are updated by:

$$\hat{a} \leftarrow \hat{a} - \beta \frac{\partial}{\partial \hat{a}} E\Big((\acute{r}_a, 0), f\big(W^{(M)}, (s, \hat{a}_0, \hat{b}_0, \ldots, \hat{a}_t, \hat{b}_t)\big) \Big)$$
$$\hat{b} \leftarrow \hat{b} - \beta \frac{\partial}{\partial \hat{b}} E\Big((0, \acute{r}_b), f\big(W^{(M)}, (s, \hat{a}_0, \hat{b}_0, \ldots, \hat{a}_t, \hat{b}_t)\big) \Big)$$

where $W^{(M)} \sim \mathbb{W}$.

This amounts to minimax:

$$\min_{\hat{a}} \max_{\hat{b}} E\Big((\acute{r}_a, 0), f\big(W^{(M)}, (s, \hat{a}_0, \hat{b}_0, \ldots, \hat{a}_t, \hat{b}_t)\big) \Big)$$
$$\min_{\hat{b}} \max_{\hat{a}} E\Big((0, \acute{r}_b), f\big(W^{(M)}, (s, \hat{a}_0, \hat{b}_0, \ldots, \hat{a}_t, \hat{b}_t)\big) \Big)$$

After training, given the known state s, the unknown optimal action for player A for the present epoch, $\hat{a}_0$, is found. The agent (here is the player A) then takes action $\hat{a}_0$ and moves on to the next state or epoch. Then player B takes over and repeats deducing phase.

However, the result is hard to quantify. We show only part of the result as in figure 1.

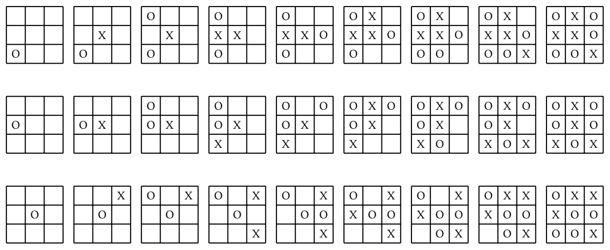

Figure 1: States of the play by our proposed model

3.3 Sudoku

The source code can be downloaded at:
https://github.com/DeepDeducing/Sudoku.

We show our method can also be applied to spatial planning. We use Sudoku as an example and try to let our proposed model solve completely blank Sudoku. In the learning phase, consider $a_0, \ldots, a_8$ represents a series of random 9 algebras[8], and r represents reward, where $r = \mathbf{1}$ if $a_0, \ldots, a_8$ are completely different and $r = \mathbf{0}$ if otherwise, such that:

$$X \triangleq \big[(a_0, \ldots, a_8)^{(0)}, \ldots, (a_0, \ldots, a_8)^{(N)} \big]$$
$$Y \triangleq \big[r^{(0)}, r^{(1)}, \ldots, r^{(N)} \big]$$

Then in the deducing phase, consider:

$$\mathbb{X} \triangleq \big\{ X^{(0)}, X^{(1)}, \ldots, X^{(M)} \big\}$$
$$\mathbb{W} \triangleq \big\{ W^{(0)}, W^{(1)}, \ldots, W^{(M)} \big\}$$
$$\mathbb{Y} \triangleq \big\{ Y^{(0)}, Y^{(1)}, \ldots, Y^{(M)} \big\}$$

Suppose, $\acute{r}$ represents the optimal reward, $t^{(i,:)}$ represents the present state[9] in row i, $t^{(:,j)}$ represents the present state in column j, and $g^{(k)}$ represents the present state in grid k.

[8] Each a is one-hotted.
[9] Visible algebra is one-hotted while missing algebra is initialized close to $\mathbf{0}$.

Then by MWM-SGD and error backpropagation, in each iteration, a $W^{(M)}$ is randomly selected from $\mathbb{W}$, and the missing algebras in each row, column and grid are updated by (while the visible algebras are not updated):

$$t^{(i,:)}_{\text{update}} = -\beta \frac{\partial}{\partial t^{(i,:)}} E\Big(\acute{r}, f\big(W^{(M)}, t^{(i,:)}\big) \Big)$$
$$t^{(:,j)}_{\text{update}} = -\beta \frac{\partial}{\partial t^{(:,j)}} E\Big(\acute{r}, f\big(W^{(M)}, t^{(:,j)}\big) \Big)$$
$$g^{(k)}_{\text{update}} = -\beta \frac{\partial}{\partial g^{(k)}} E\Big(\acute{r}, f\big(W^{(M)}, g^{(k)}\big) \Big)$$

and[10]:

$$t^{(i,:)} \leftarrow t^{(i,:)} + t^{(i,:)}_{\text{update}}$$
$$t^{(:,j)} \leftarrow t^{(:,j)} + t^{(:,j)}_{\text{update}}$$
$$g^{(k)} \leftarrow g^{(k)} + g^{(k)}_{\text{update}}$$

After training, whichever missing algebra holds the highest value will become the new visible algebra and the whole deducing phase starts over again til the next visible algebra is found.

We present part of our result as below:

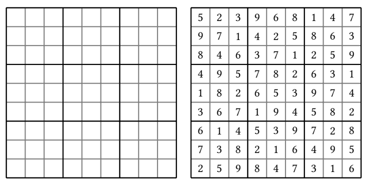

Figure 2: Blank 9x9 Sudoku table being solved by our proposed model

We compare our result under different sizes of sets of weight matrices (denoted **M**) with relative works solving 9x9 Sudoku as below:

Model used (paper)	Sudoku size	Givens	Accuracy
This paper (**M** = 20)*	9x9	0	97.3%
This paper (**M** = 20)*	9x9	17	97.1%
RecRelaN (Palm et al. [11])	9x9	17	96.6%
Loopy BP, modified (Khan et al. [6])	9x9	17	92.5%
Loopy BP, random (Bauke [1])	9x9	17	61.7%
Loopy BP, parallel (Bauke [1])	9x9	17	53.2%
This paper (**M** = 1)*	9x9	17	51.7%
Deeply Learned Messages (Lin et al. [7])	9x9	17	00.0%
RelaN, node (Santoro et al. [15])	9x9	17	00.0%
RelaN, graph (Santoro et al. [15])	9x9	17	00.0%
Deep Convolutional Network (Park [12])	9x9	24	70.0%

Table 2: Comparison of related papers according to Palm et al. [11]

The result further shows that there are no overlapping Sudoku tables generated by our proposed model[11].

[10] In each iteration, all missing algebras are simultaneously, rather than sequentially, updated. Each missing algebra also simultaneously receives three gradients from row, column and grid.
[11] The test was run on 10000 blank 9x9 Sudoku tables.

3.4 Black Jack

The source code can be downloaded at:
https://github.com/DeepDeducing/BlackJack.

We further demonstrate our proposed model can be applied to environment with undetermined reward. We take black jack problem for example. We use the same model in classic control except we now apply particle swarm optimization (as in Kennedy [5]) into our model to address the issue of undetermined reward. The unknown actions of the player (our model), $\hat{a}$, compose a single particle. Several particles are initialized and updated by gradient provided by error backpropagation and MWM-SGD, following particle swarm optimization rules[12]. We show the result of our proposed model as player under 10000 trials.

M	Player wins	Draw	Dealer wins
2	4016	883	5101
4	4039	856	5105
6	4131	938	4931

Table 3: Performance comparison in black jack

3.5 Class Saliency Visualisation

The source code can be downloaded at:
https://github.com/DeepDeducing/Class_Saliency_Visualisation.

Last but not least, since deducing action based on corresponding optimal rewards is equivalent to retrieving or optimizing input data based on given output data, thus our method can be viewed as a kind of model inversion or XAI (explainable A.I.) tool to see which part of input data is more emphasized by trained deep neural network and receives more gradient during error backpropagation.

Thus we further demonstrate our proposed model can be applied to class saliency visualisation and provide explainibility to deep neural network. We also demonstrate that MWM-SGD improves overall saliency in visualisation. The dataset we use is the traditional MNIST hand-written digits.

In the deducing phase, consider multiple MNIST-trained deep neural networks and their sets of weight matrices such that:

$$\mathbb{W} \triangleq \left\{W^{(0)}, W^{(1)}, \ldots, W^{(M)}\right\}$$

Suppose, $\acute{c}$ represents the given classification. To find out which part of input data is more emphasized by trained deep neural network and receives more gradient during error backpropagation, we first initialize estimated image $\hat{I}$. Then, by MWM-SGD and error backpropagation, in each iteration, a set of weight matrices $W^{(M)}$ is randomly selected where $W^{(M)} \sim \mathbb{W}$, and $\hat{I}$ is updated by:

$$\hat{I} \leftarrow \hat{I} - \beta \frac{\partial}{\partial \hat{I}} E\Big(\acute{c}, f\big(W^{(M)}, \hat{I}\big)\Big)$$

After training, the trained $\hat{I}$ can be viewed as a saliency visualisation image. We show the degree of saliency of the trained $\hat{I}$ for each hand-written digit and corresponding classification $\acute{c}$ under different sizes of sets of weight matrices (denoted M) as below:

[12] With 100 particles, inertia coefficient $w = 0.9$, cognitive coefficient $c_1 = \beta$, social coefficient $c_2 = 0$, and $r_1 = 1$ constantly. The idea is to let each particle freely explore the decision space without social interrelation (cognition-only). After training, the particle holding the highest value will be chosen as the updated $\hat{a}$.

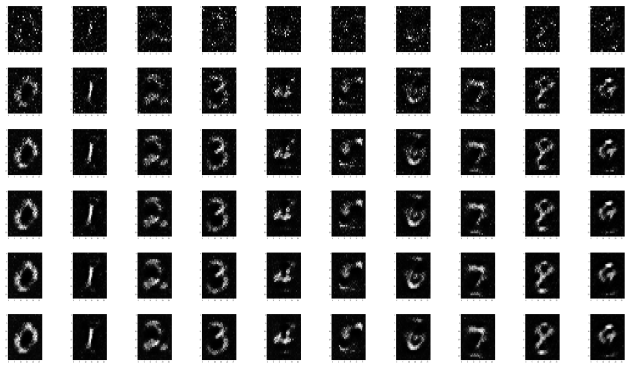

Figure 3: Saliency comparison under different size of sets of weight matrices with respect to M = 1, M = 5, M = 10, M = 15, M = 20, and M = 25

The result shows that MWM-SGD improves overall saliency.

4 DISCUSSION

For the sake of discussion, we name our proposed model “deep deducing” with respect to deep learning.

4.1 Difference with other works

4.1.1 Deep Q Learning. The basic idea behind deep Q learning is to view a deep neural network as a policy function π:

$$a = \pi(s)$$

where s, a are the state and action of the agent respectively.

To obtain such ideal policy function, deep Q learning uses state-value function (Bellman function or other hand-written function) to give a value to each action and train the deep neural network according to these values. After training, for any given state s, the trained deep neural network can derive an action a simply by forward-feeding based on $a = \pi(s)$. Mnih et al. [8] further demonstrated that deep Q learning can be applied to Atari game successfully. Nonetheless, despite its success in games, it is questionable whether Bellman function or hand-written equation exists in human brain biologically. Upon this view, deep Q learning seldom explicitly explains the interplay of state, action and reward inside human brain neural network.

On the other side, deep deducing has deep neural network learn the state input, action input and reward output "stochastically" from the digital (twin) environment by using deep learning. Thus the deep neural network itself factually replaces Bellman functionality or any other hand-written equation.

Then by these trained neural networks, deep deducing derives the action from the input layer. Namely, the action of an agent is "generated" from the input layer by using error backpropagation. Since the brain has many pathways from later layers back to earlier ones (as in Hinton [4]), it is also biologically possible that the brain has pathways from later layers back to the first one (input layer) to convey the information required for generating or updating action. Thus, if not all, to a certain extend, deep deducing explains the interplay of state, action and reward in human brain neural network. Upon this view, deep deducing is different from deep Q learning and its reinforcement learning method.

4.1.2 *Generative Adversarial Neural Net.*

Generative Adversarial Neural Net is probably the most interesting phenomenon in deep learning area in the last few decades. Goodfellow et al. [3] shows that by combining discriminator and generator and training their respective weight matrices according to a two-player minimax game with opposite target values, a generator can generate patterns that mimic the original data.

Upon this view, deep deducing can be said to have a digital (twin) environment serve as as a generator to reinforce the deep neural networks in the learning phase. Then in the deducing phase, the trained deep neural networks serves as discriminator.

However, from the above applications, we know that even for a very thin layer of input data and discriminator, by using error backpropagation, it can still "generate" patterns of actions without any generator. Furthermore, in Tic Tac Toe application, we show that splitting input data and training them adversarially under the same discriminator also according to a two-player minimax game with opposite target values, it can still perform generative adversarial deduction.

The question then arises how a discriminator "generates" patterns of actions with only data rather than generator.

We think the answer lies in the nature of the weight matrices in deep neural network. Mordvintsev et al. [9, 10] and Szegedy et al. [20] demonstrated that error backpropagtion toward randomized input data generated trippy image without generator, which technique is commonly referred to as "deep dream". The technique behind deep dream demonstrated that a deep neural network itself is enough for generating patterns. More explicitly, deep dream demonstrated that trained weight matrices in deep neural network gradually form an error surface where an input data cannot only be placed to test if it reaches global minima, but can also be optimized upon this error surface to converge to global minima by using error backpropagation, and the optimized input data will gradually form a pattern mimicking the original input data. Deep deducing further refines this process to shed light on the potential deductive reasoning ability in deep neural network.

Upon this view, GAN can be said to have used a generator to "perturbate" the error surface of deep neural network to adversarially train the generator while deep deducing (in Tic Tac Toe) can be said to have used a part of input data to "perturbate" this error surface to adversarially train the other part of input data. Taking only data into consideration, deep deducing might be more similar to deep dream than GAN.

4.1.3 *Other gradient methods used in decision-making.*

Since our proposed method utilizes gradient provided by error backpropagation in deep neural networks, our proposed model can be viewed as allowing estimated actions to be optimized upon an error surface. Therefore, the comparison between our proposed model and other optimization techniques is discussed here.

Gradient method has already been widely used in robotics. This is usually done by designing an expert-designed cost function of robot motion trajectory where the cost is the robot motion trajectory length or distance from certain obstacles (such as in Nathan et al. [13]).

However, designing expert-level cost function requires much human expert interference. And since deep neural network can form an cost function by its innate universal approximation feature, it inevitably becomes a question whether expert-designed cost functions is needed.

Aside from expert function, there are works using gradient provided by error backpropagation in trained deep neural networks to optimize actions. For example, a work by Yize et al. [2] utilizes gradient provided by error backpropagation in a single recurrent neural network to optimize or tune a time sequence of parameters in a building to cut down overall energy consumption. Also, Miguel et al. [19] utilizes gradient provided by error backpropagation to optimize wireless network. These works are usually referred to as "differentiable planning".

However, the above methods utilize gradient provided by a only single neural network or its variant, which inevitably faces local minima problems. Even with momentum added to gradient descent, it can only overcome some but not all local minima problems [2]. Even if we initialize different initial values for estimated action to try to avoid poor local minima [19], there is no guarantee that a single initial value will converge to desirable minima.

This is in fact due to the non-convex nature of error surface provided by a single deep neural net or its variant. Our proposed model overcomes these problems by allowing the estimated action to be optimized or performing gradient descent stochastically among several trained neural nets. Our experiment shows that MWM-SGD provides more stable results as in table 1 to 3. Also, MWM-SGD helps an input data's neural outputs coincide with multiple given outputs as in statement (2-2).

4.1.4 *Other gradient methods used to provide explainibility.*

Using gradients provided by error backpropagation in deep neural network to optimize input data to provide data saliency is no longer news in XAI (Explainable A.I.) territory. For example, Simonyan et al. [17] demonstrated that, for a trained ConvNet, performing optimization with respect to the input image while the learned weights are fixed during this optimization generates saliency visualization. Also, DeepLIFT [16] was introduced to be applied to the top of deep neural network prediction, enhancing gradient method by multiplying the gradient with the input signal. The above methods are referred to as "gradients explanation techniques" or "backpropagation-based approaches".

However, the above methods are done in a single deep neural network and its variant. Since optimizing input image by gradient provided by error backpropagation in a single deep neural is equivalent to optimizing input image on a single error surface created by a single deep neural network, the above methods inevitably face local minima problem.

To solve this problem, rather than a single deep neural network, we train multiple neural networks and randomly select a neural network for input image to be optimized upon. We show that this method (MWM-SGD) can easily help input image to escape local minima during the optimization process and generate salient visualization as shown in figure 3 comparable to the above methods.

4.2 Pros and Cons

We discuss the pros and cons regarding deep deducing. The pros and cons are two sides of a coin.

The obvious pro is that deep deducing is biologically plausible. Since the brain has many pathways from later layers back to earlier ones (as in Hinton [4]), it is possible that the brain also has pathways from later layers back to the first one (input layer) to convey the information required for deducing action. Also, by stacking sets of trained neural networks, deep deducing helps deduced actions escape local minima and provides a simple and integrated optimal action to satisfy different optimal reward requirements, without using any Bellman function or hand-written equation.

The obvious cons are that: (1) First, deep deducing heavily relies on parrellism where multiple neural nets are parallelly trained in digital (twin) environment to resolve sparse reward problem. Thus, a refined digital (twin) environment is needed. (2) Second, deep deducing obviously requires heavy parallel computational power and memory resource which limits its application for the present time.

Deep deducing differs from other kinds of deep reinforcement learning in that deep deducing directly learns from the digital (twin) environment stochastically (since deep deducing is completely deep learning in the learning phase), then, based on these learned cases, it makes deduction or automated reasoning according to the present state. A reinforcement learning method could be applied to deep deducing. However, it could be computationally time expensive for the present time.

4.3 Future Broader Impact

With the advance of digital (twin) environment and metaverse, the training of intelligent A.I. would largely be done in digital (twin) environment rather than real physical environment. In this scenario, reinforcement learning and stochastic learning would both be options. This paper's model can serve as an stochastic learning method to compress environment experiences into deep neural networks and allow A.I. to make decision by these neural experiences. Furthermore, with the advance of quantum parallelism, the problem of lack of parallel computational power might be solved, furthering the impact of this paper's model.

5 CONCLUSION

In this paper, we introduce a general approach for deep neural network to perform deductive reasoning based on error backpropagation, which can be used to deduce action upon a given state-reward sequence. Our method first learns a neural network that predicts a sequence of rewards from a sequence of states and actions. Then, given a state-reward sequence, appropriate actions can be optimized through backpropagation. Thanks to deep learning, it is possible to obtain an off-the-shelf error surface for actions to be optimized upon. We show that this approach can be applied to various types of digital environment with little modification. Moreover, in contrast to recent approaches, it does not require Bellman function or other hand-written functions.

ACKNOWLEDGMENTS

Thanks to Science and Technology Law Institute, Institute for Information Industry located in Taipei, Taiwan. The author is a registered attorney with highest degree of master in law conferred by National Taiwan University.

REFERENCES

[1] Heiko Bauke. Passing messages to lonely numbers. *Computing in Science & Engineering*, 10(2), 2008.

[2] Yize Chen, Yuanyuan Shi, and Baosen Zhang. Modeling and optimization of complex building energy systems with deep neural networks. *51st Asilomar Conference on Signals, Systems, and Computers*, 2017.

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, 2014.

[4] G.E. Hinton. How neural networks learn from experience. *Scientific American*, pages 145–151, 1992.

[5] James Kennedy and Russell C. Eberhart. Particle swarm optimization. *Proceedings of ICNN - International Conference on Neural Networks*, 1995.

[6] Sheehan Khan, Shahab Jabbari, Shahin Jabbari, and Majid Ghanbarinejad. Solving sudoku using probabilistic graphical models. 2014.

[7] Guosheng Lin, Chunhua Shen, Ian Reid, and Anton van den Hengel. Deeply learning the messages in message passing inference. *Advances in Neural Information Processing Systems*, pages 361–369, 2015.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *NIPS Deep Learning Workshop*, 2013.

[9] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks. In *Archived from the original on 2015-07-03*. Google Research, 2015.

[10] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Deepdream - a code example for visualizing neural networks. In *Archived from the original on 2015-07-08*. Google Research, 2015.

[11] Rasmus Berg Palm, Ulrich Paquet, and Ole Winther. Recurrent relational networks. *32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*, 2018.

[12] Kyubyong Park. Can neural networks crack sudoku? 2016. URL https://github.com/Kyubyong/sudoku.

[13] Nathan Ratliff, Matt Zucker, J. Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. *IEEE International Conference on Robotics and Automation*, 2009.

[14] David E. Rumelhart, Geoffrey E. Hinton, and James L. McClelland. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. MIT Press, 1987.

[15] Adam Santoro, David Raposo, David GT Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. *arXiv preprint arXiv:1706.01427*, 2017.

[16] A. Shrikumar, P. Greenside, and A. Kundaje. Learning important features through propagating activation differences. *Proceedings of the 34th International Conference on Machine Learning*, 2017.

[17] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv:1312.6034*, 2014.

[18] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[19] Miguel Suau, Alexandros Agapitos, David Lynch, Derek Farrell, Mingqi Zhou, and Aleksandar Milenovic. Offline contextual bandits for wireless network optimization. *arXiv:2111.08587v1*, 2021.

[20] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.