On Reinforcement Learning for Full-Length Game of StarCraft*

Zhen-Jia Pang, Ruo-Ze Liu, Zhou-Yu Meng, Yi Zhang, Yang Yu, Tong Lu
National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
pangzj@lamda.nju.edu.cn, liuruoze@163.com, misskanagi@gmail.com, zhangyi@smail.nju.edu.cn,
yuy@nju.edu.cn, lutong@nju.edu.cn

AAAI 2019

Contents

Introduction

- StarCraft Π is a ideal environment for reinforcement learning research due to complexity of states, diversity of actions, and long time horizon.

- Most existing reinforcement learning algorithms are still difficult to be used in such large-scale reinforcement learning problems.

- A hierarchical architecture is proposed to make intractable large-scale reinforcement learning problems easier to handle, and an effective training algorithm tailed to the architecture is also investigated.

Background

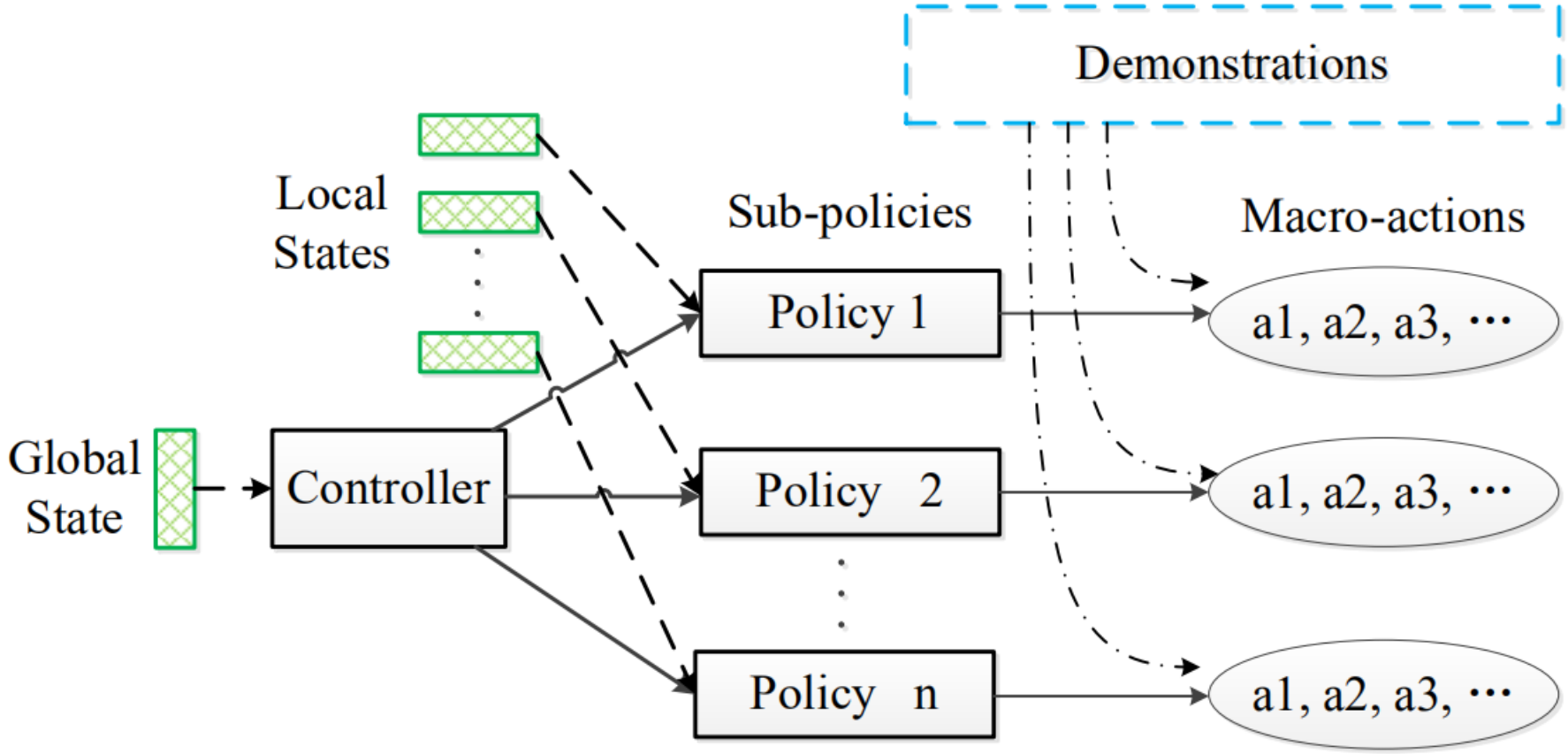
- Reinforcement Learning

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_\pi [\sum_{t=0}^{T} \gamma^t R(s_t, a_t)]$$

- Hierarchical Reinforcement Learning
  - Decomposing a complex problem into several sub-problems and solving each sub-question in turn.

- Curriculum Learning
  - Training the agent from the simplest opponent to harder ones.

Method



Overall Architecture

- Hierarchical Architecture
- Generation of Macro-actions
- Training Algorithm


3DMGAME

Method - Hierarchical Architecture

- The controller chooses a sub-policy in every K time units

$$a_{t_c}^c = \Pi(s_{t_c}^c), \quad s_{t_c}^c \in S_c$$

  - Replay buffer D_C $\quad (s_{t_c}^c, a_{t_c}^c, r_{t_c}^c, s_{t_c+1}^c)$

- The chosen sub-policy makes a decision (picks a macro-action) in every time unit

$$a_{t_i}^i = \pi_i(s_{t_i}^i)$$

  - Replay buffer D_i $\quad (s_{t_i}^i, a_{t_i}^i, r_{t_i}^i, s_{t_i+1}^i)$

$$r_{t_c}^c = r_{t_i}^i + r_{t_i+1}^i + \ldots + r_{t_i+K-1}^i$$

Method - Hierarchical Architecture

- Such a hierarchical structure can split the original huge state space into a plurality of subspaces corresponding to different policy networks.
- The hierarchical structure can also split the tremendous action space.
- The hierarchical architecture makes the design of the reward function easier. Different sub-policies may have different targets, thus they can learn more quickly by their own suitable reward functions.

Method – Generation of Macro-actions

- It always needs to do a sequence of raw actions to achieve one simple purpose.
- Thus we instead generate a macro-action space A^{η} which is obtained through data mining from trajectories of experts.
- The original action space A is replaced by the macro-action space A^{η}.

Method – The generation process of macro-actions

- Firstly, we collect some expert trajectories which are sequence of operations $a \in A$ from game replays.
- Secondly, we use a prefix-span (Yan, Han, and Afshar 2003) algorithm to mine the relationship of the each operation and combine the related operations to be a sequence of actions a^{seq} of which max length is C and constructed a set A^{seq} which is defined as $A^{seq} = \{ a^{seq} = (a_0, a_1, a_2, ...a_i) \mid a_i \in A \text{ and } i \leq C, \}$
- Thirdly, we sort this set by $\text{frequency}(a^{seq})$.
- Fourthly, we remove duplicated and meaningless ones, remaining the top K ones. Meaningless refers to the sequences like continuous selection or camera's movement.
- Finally, the reduced set is marked as newly generated macro-action space A^{η}.

Method – Training Algorithm (PPO)

$$L_t(\theta) = \hat{\mathbb{E}}_t[L_t^{\text{clip}}(\theta) + c_1 L_t^{\text{vf}}(\theta) + c_2 S[\pi_\theta](s_t)] \quad (9)$$

where c_1, c_2 are the coefficients we need to tune, and S denotes an entropy bonus. $L_t^{\text{clip}}(\theta)$ is defined as follows:

$$L_t^{\text{clip}}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t] \quad (10)$$

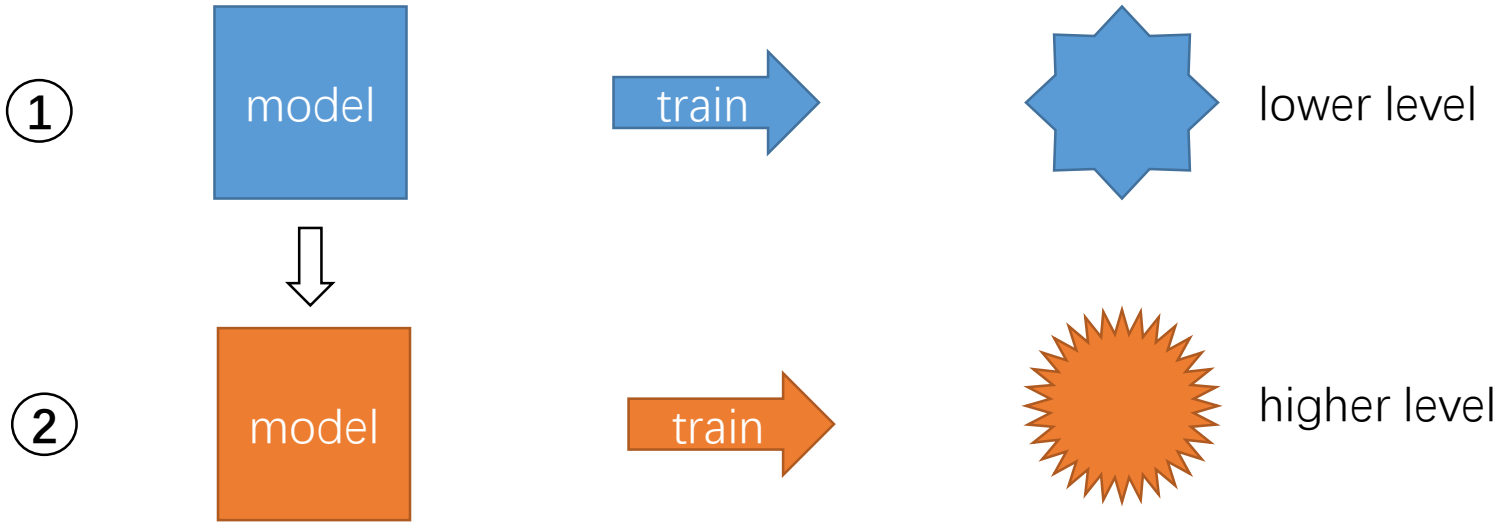
$$L_t^{\text{vf}}(\theta) = \hat{\mathbb{E}}_t[(r(s_t, a_t) + \hat{V}_t(s_t) - \hat{V}_t(s_{t+1}))^2] \quad (11)$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, $\hat{A}_t$ is computed by a truncated version of generalized advantage estimation.

Method – Reward Design & Curriculum Learning

- **Reward Design:** It's hard to learn a effective policy using Win/Loss reward. Thus we have designed reward functions for the sub-policies which combine dense reward (Blizzard score) and sparse reward.

- **Curriculum Learning:** Training on higher difficulty levels gives less positive feedback, making it difficult for agent to learn from scratch.

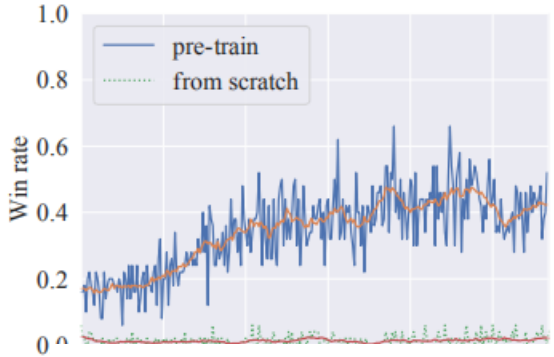


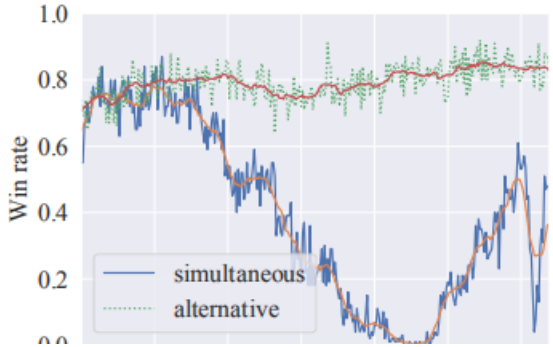
Experiments – three battle policy

- **Combat Rule:** It is a simple battle policy, and there is only one action in combat rule model: attacking a sequence of fixed positions.

- **Combat network:** Three actions and a position vector can be obtained from combat network. The position of attack and movement are specified by the position vector. These actions are:
  - All attack a certain position
  - All retreat to a certain position
  - Do nothing

- **Mixture model:** a combination of combat rule and combat network.





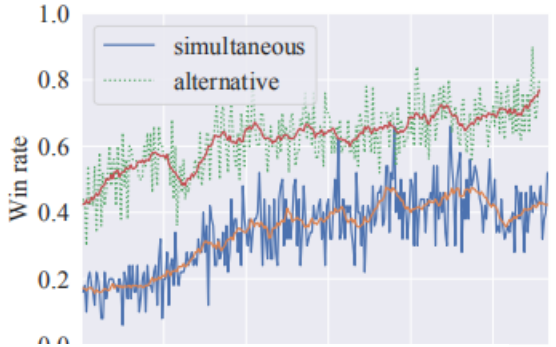
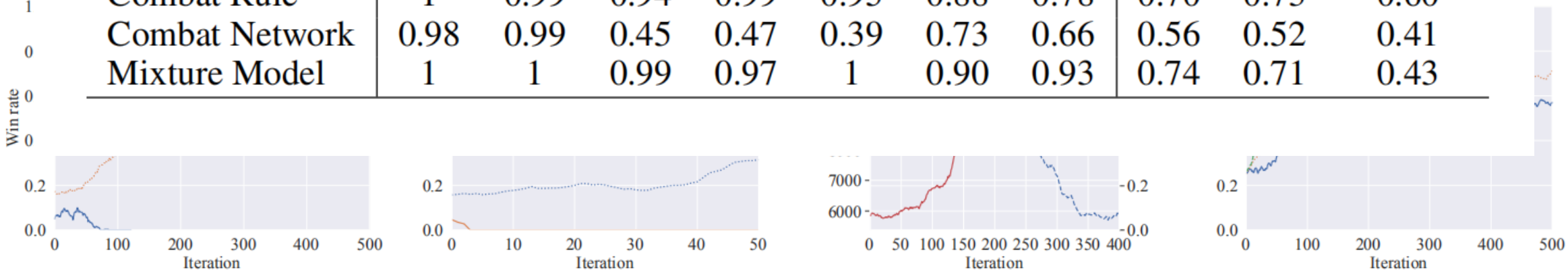


Table 1: Evaluation Results.

Opponent's Type	Non-cheating (Training)							Cheating (No-training)		
Difficulty Level	1	2	3	4	5	6	7	8	9	10
Combat Rule	1	0.99	0.94	0.99	0.95	0.88	0.78	0.70	0.73	0.60
Combat Network	0.98	0.99	0.45	0.47	0.39	0.73	0.66	0.56	0.52	0.41
Mixture Model	1	1	0.99	0.97	1	0.90	0.93	0.74	0.71	0.43



(a) Hierarchy vs single (b) Win/Loss vs handcrafted (c) Score and win rate (d) Impact of episode num

Figure 4: Comparison of settings.

Experiments – some results

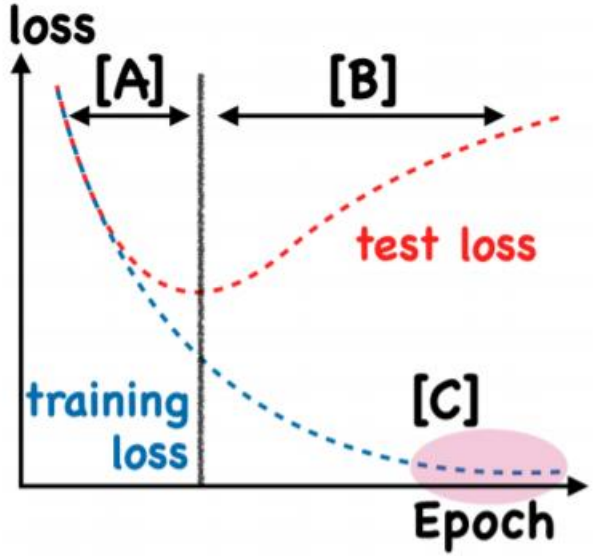
- While replacing a sub-policy, the parameters of the controller network and other sub-policies are retained, only the parameters of the newly replaced sub-policy are updated, which can accelerate learning. (Fig. 3(b))
- When all the networks of the hierarchical architecture update at the same time, they are sometimes unstable. We can use an alternate updating strategy to address this issue. (Fig. 3(c) & Fig. 3(d))
- Curriculum learning (Fig. 3(a)) and module training (Fig. 3(b)) are effective.
- Hierarchical architecture is effective.
- Both Win/Loss reward and handcrafted reward can achieve good results on low difficulty level, but fail on high difficulty level.
- Blizzard score and winning rate are sometimes not in a proportional relationship.
- ……

Do We Need Zero Training Loss After Achieving Zero Training Error?

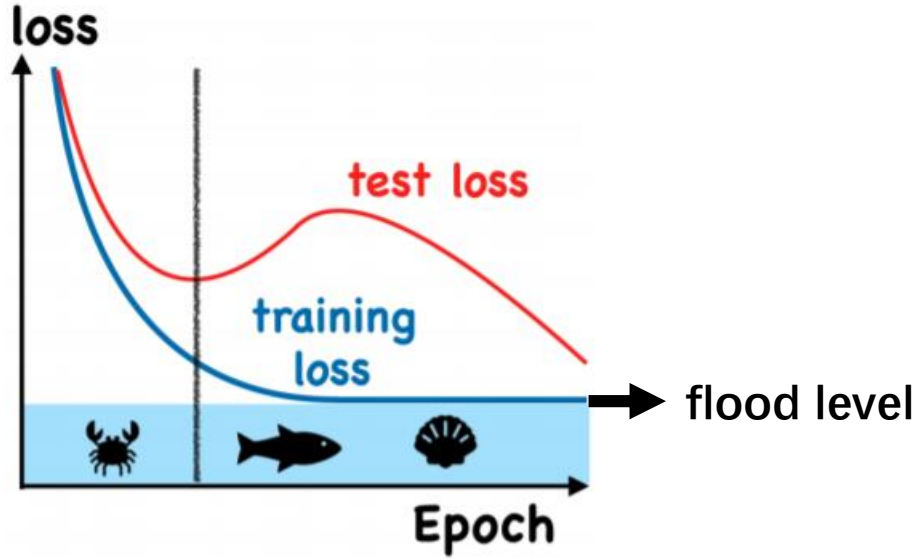
Takashi Ishida [1 2] **Ikko Yamane** [1] **Tomoya Sakai** [3] **Gang Niu** [2] **Masashi Sugiyama** [2 1]

ICML 2020

flooding

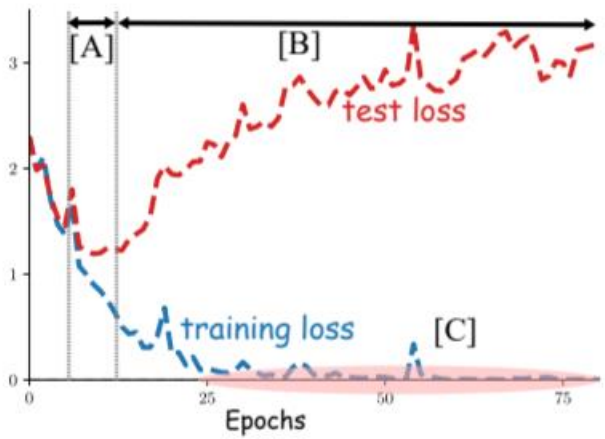


(a) w/o Flooding

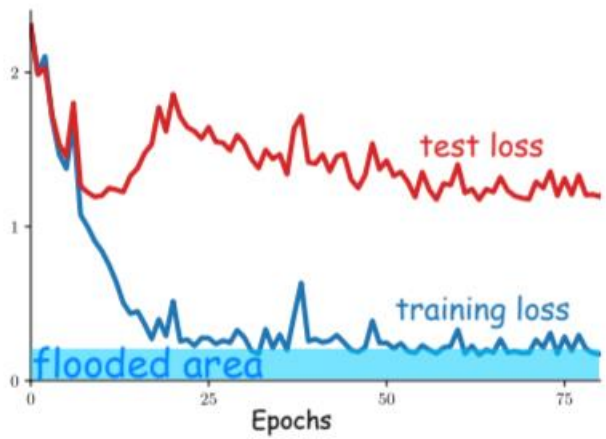


(b) w/ Flooding

Our approach makes the loss float around the flood level by doing mini-batched gradient descent as usual but gradient ascent if the training loss is below the flood level.



(c) C10 w/o Flooding



(d) C10 w/ Flooding

With flooding, the model will continue to “random walk” with the same non-zero training loss, and we expect it to drift into an area with a flat loss landscape that leads to better generalization.

flooding

This modification can be incorporated into existing machine learning code easily: Add one line of code for Eq. (1) after evaluating the original objective function $J(\boldsymbol{\theta})$. A minimal working example with a mini-batch in PyTorch (Paszke et al., 2019) is demonstrated below to show the additional one line of code:

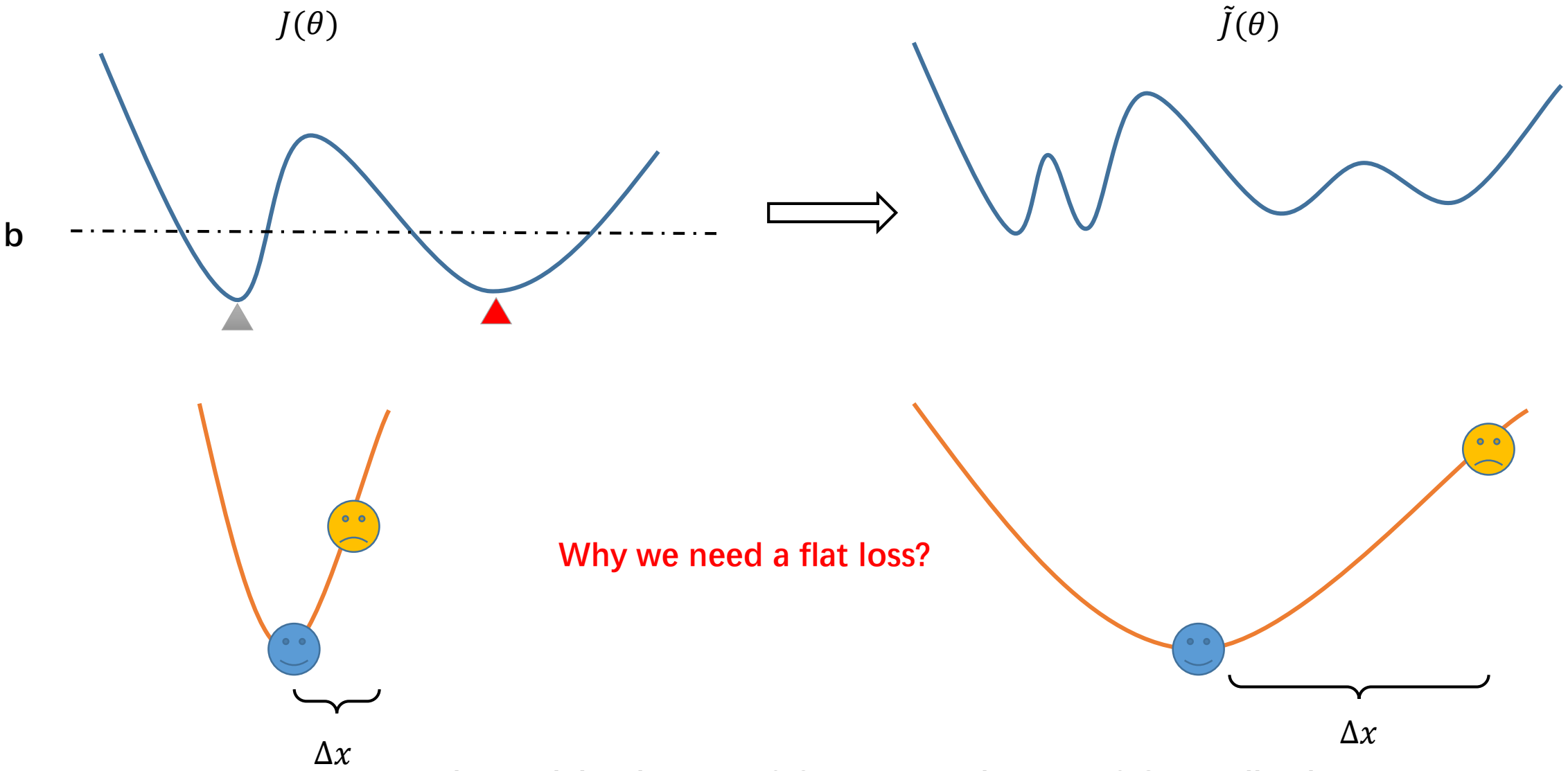
```
outputs = model(inputs)
loss = criterion(outputs, labels)
flood = (loss-b).abs()+b  # This is it!
optimizer.zero_grad()
flood.backward()
optimizer.step()
```

It may be hard to set the flood level without expert knowledge on the domain or task. We can circumvent this situation

[1] For the details of these experiments, see Appendix D.

$$\widetilde{J}(\boldsymbol{\theta}) = |J(\boldsymbol{\theta}) - b| + b,$$

Some Thinking



Why we need a flat loss?

Adversarial Robustness? Common Robustness? Generalization

Experiments

Table 2. Experimental results for the synthetic data. Sub-table (A) shows the results without early stopping. Sub-table (B) shows the results with early stopping. The better method is shown in **bold** in each of sub-tables (A) and (B). "BR" stands for the Bayes risk. For Two Gaussians, the distance between the positive and negative distributions is larger for larger m.

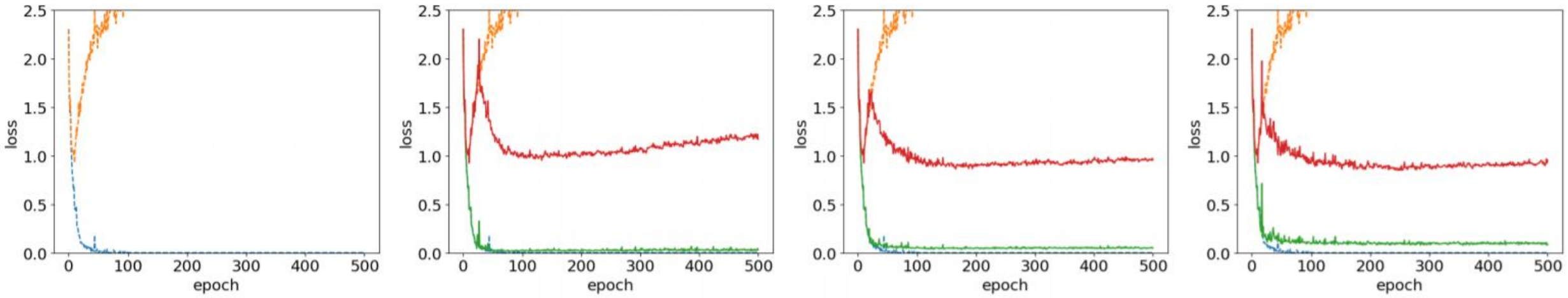
		(A) Without Early Stopping			(B) With Early Stopping		
Data	Setting	Without Flooding	With Flooding	Chosen b	Without Flooding	With Flooding	Chosen b
Two Gaussians	m: 1.0, BR: 0.14	87.96%	**92.25%**	0.28	91.63%	**92.25%**	0.27
Two Gaussians	m: 0.8, BR: 0.24	82.00%	**87.31%**	0.33	86.57%	**87.29%**	0.35
Sinusoid	Label Noise: 0.01	93.84%	**94.46%**	0.01	**92.54%**	**92.54%**	0.00
Sinusoid	Label Noise: 0.05	91.12%	**95.44%**	0.10	93.26%	**94.60%**	0.01
Sinusoid	Label Noise: 0.10	86.57%	**96.02%**	0.17	**96.70%**	**96.70%**	0.00
Spiral	Label Noise: 0.01	**98.96%**	97.85%	0.01	98.60%	**98.88%**	0.01
Spiral	Label Noise: 0.05	93.87%	**96.24%**	0.04	**96.58%**	95.62%	0.14
Spiral	Label Noise: 0.10	89.70%	**92.96%**	0.16	89.70%	**92.96%**	0.16

Experiments

Table 3. Benchmark datasets. Reporting accuracy for all combinations of weight decay (✓, ×), early stopping (✓, ×) and flooding (✓, ×). W, E, and F stand for weight decay, early stopping, and flooding, respectively. "—" means that flood level of zero was optimal. "N/A" means that we skipped the experiments because zero weight decay was optimal in the case without flooding. The best and equivalent are shown in **bold** by comparing "F:✓" and "F:×" for two columns with the same W and E, e.g., the 1st and 5th columns out of the 8 columns. The best combination is in **red**.

Dataset	W:	×	×	✓	✓	×	×	✓	✓
	E:	×	✓	×	✓	×	✓	×	✓
	F:	×	×	×	×	✓	✓	✓	✓
MNIST		98.32%	98.30%	**98.51%**	98.42%	**98.46%**	**98.53%**	98.50%	**98.48%**
Fashion		89.34%	89.36%	N/A	N/A	—	—	N/A	N/A
Kuzushiji		91.63%	91.62%	91.63%	91.71%	**92.40%**	**92.12%**	**92.11%**	**91.97%**
CIFAR-10		**73.59%**	73.36%	73.65%	73.57%	73.06%	**73.44%**	—	**74.41%**
CIFAR-100		42.16%	42.33%	42.67%	42.45%	**42.50%**	**42.36%**	—	—
SVHN		92.38%	92.41%	93.20%	92.99%	**92.78%**	**92.79%**	—	**93.42%**

Experiments



(a) CIFAR10 (0.00) (b) CIFAR10 (0.03) (c) CIFAR10 (0.07) (d) CIFAR10 (0.20)

Figure 2. Learning curves of training and test loss for training/validation proportion of 0.8. For each row, the first figure is the learning curves without flooding. The 2nd, 3rd, and 4th figures show the results with different flood levels. Blue: training loss w/o flooding. Orange: test loss w/o flooding. Green: training loss w/ flooding. Red: test loss w/ flooding. See Fig. 6 in Appendix for other datasets.

THANKS