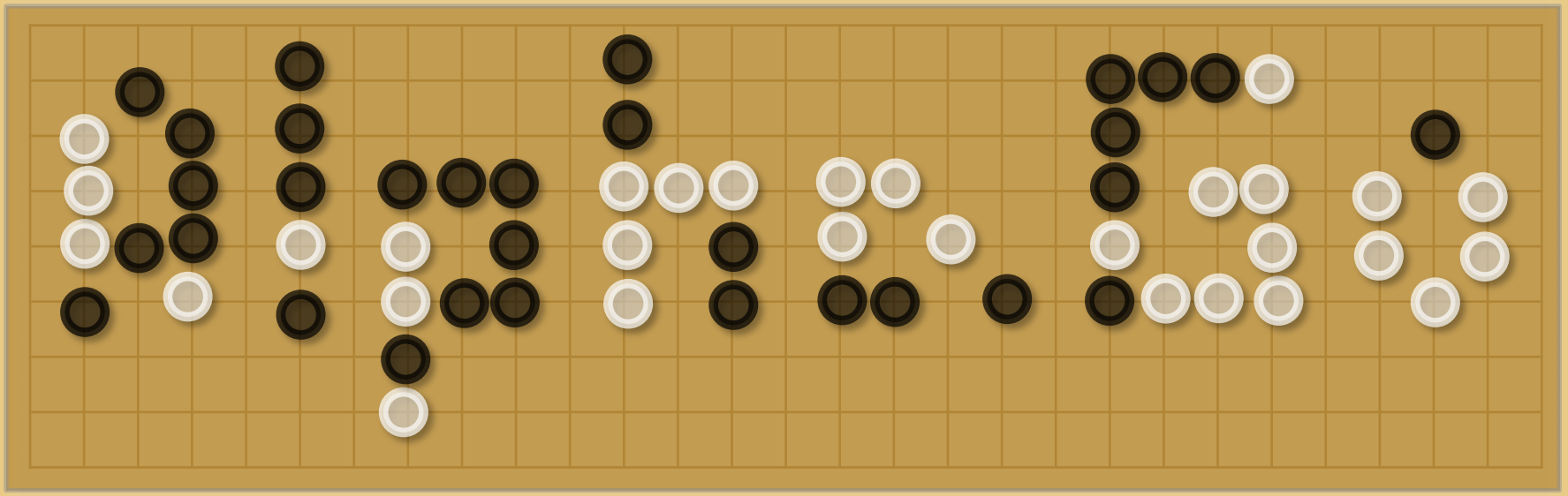


AlphaGo in Depth

By Mark Chang

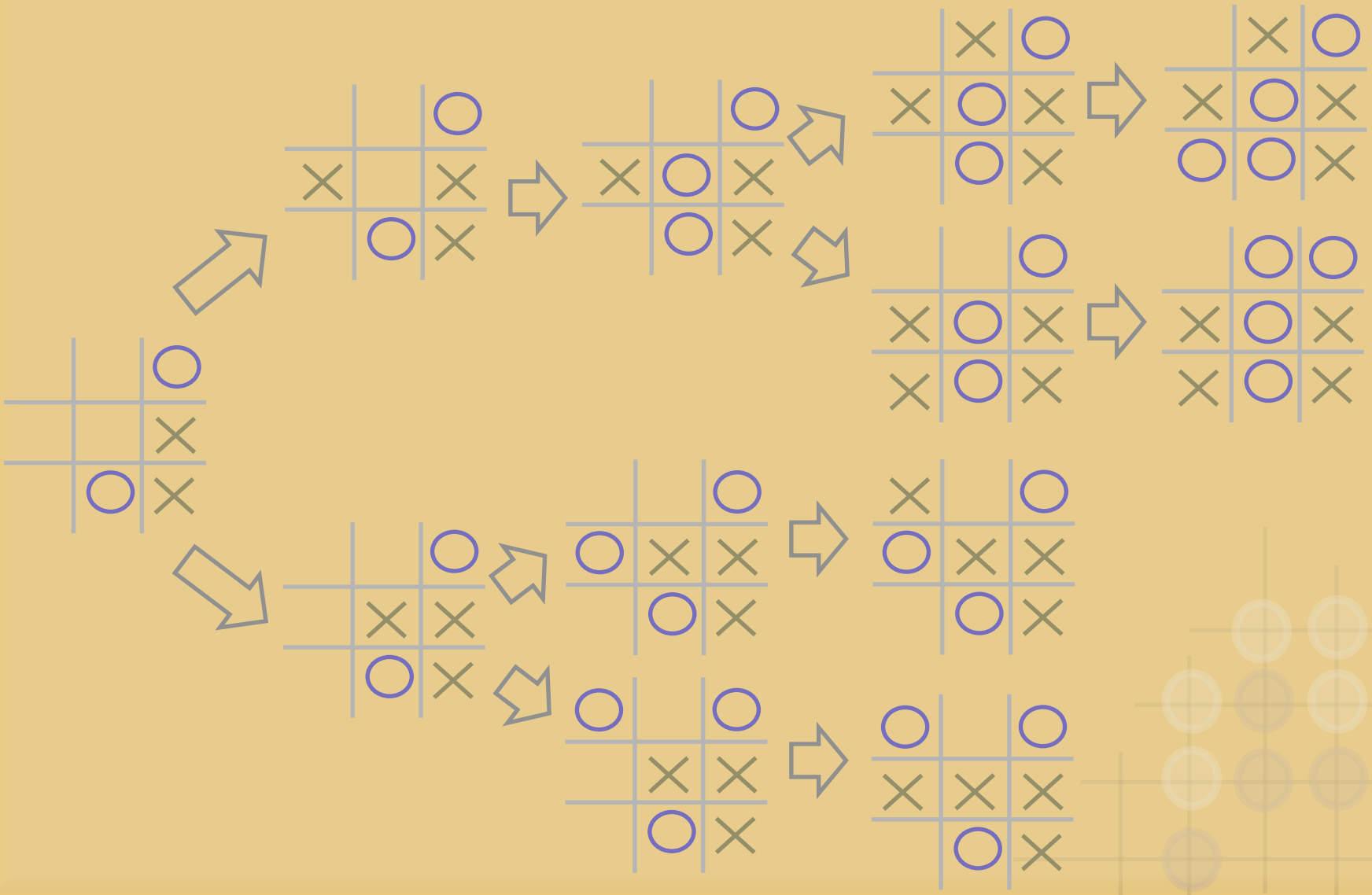
Overview

- AI in Game Playing
- Machine Learning and Deep Learning
- Reinforcement Learning
- AlphaGo's Methods

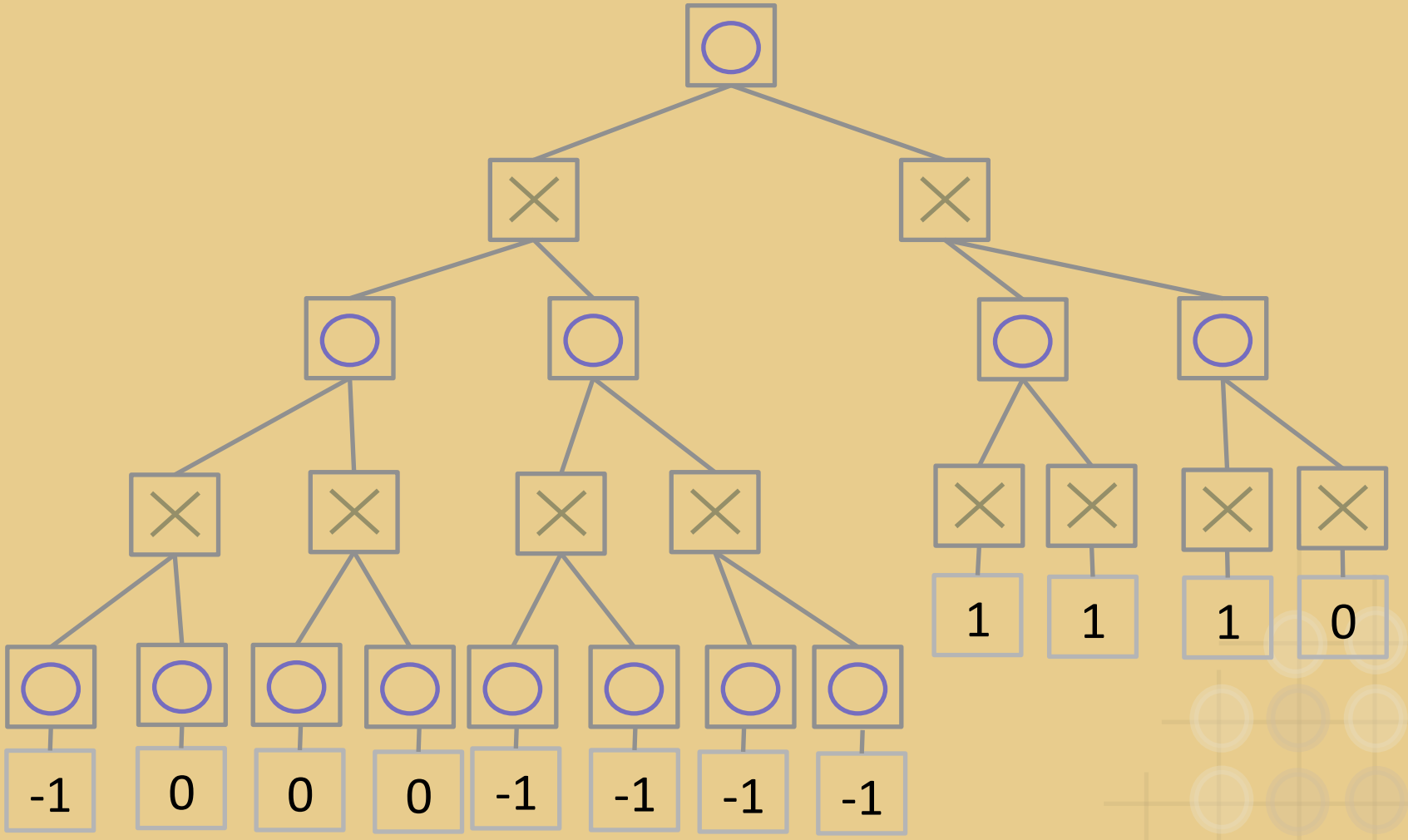
AI in Game Playing

- AI in Game Playing
  - Adversarial Search – MiniMax
  - Monte Carlo Tree Search
  - Multi-Armed Bandit Problem

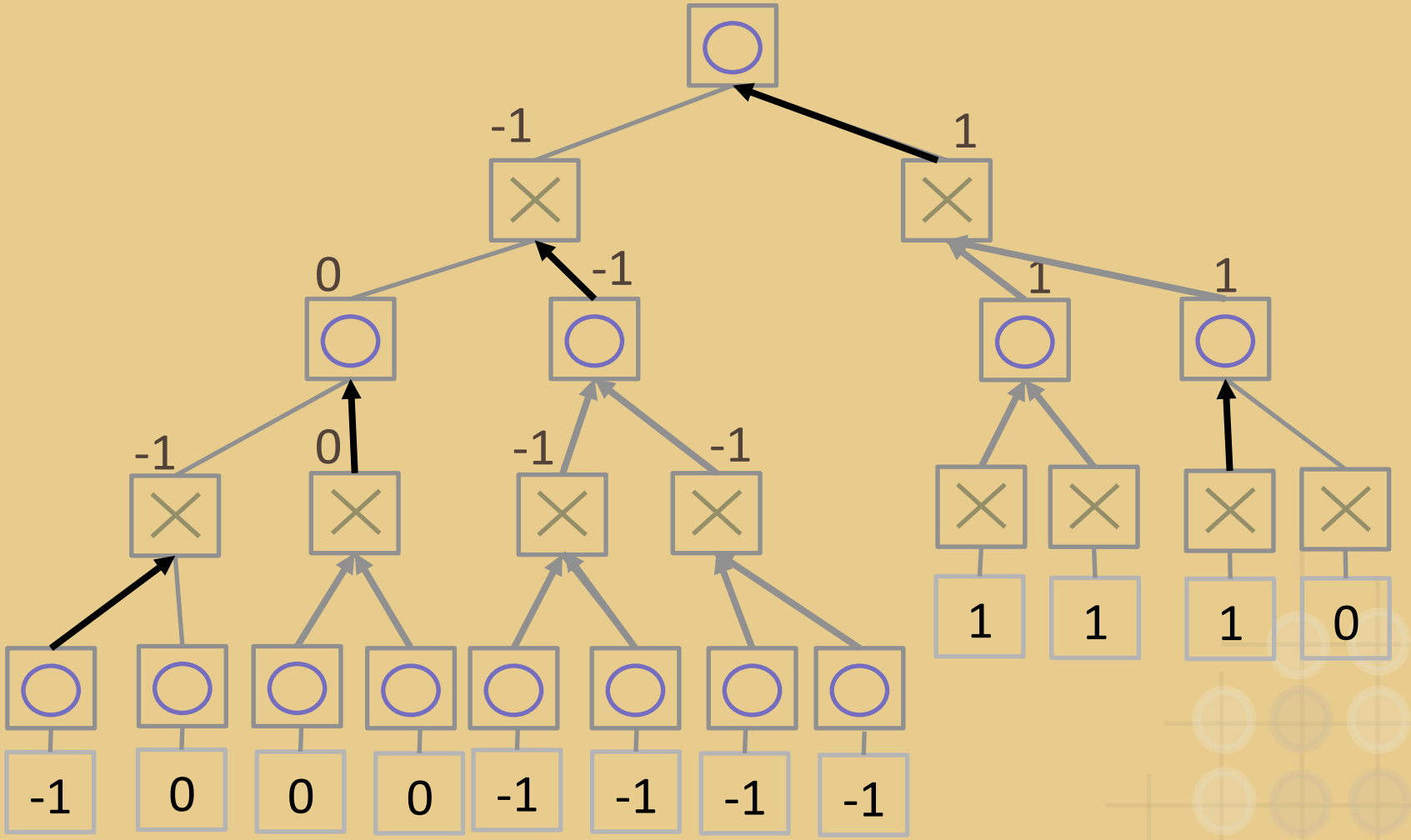
Game Playing



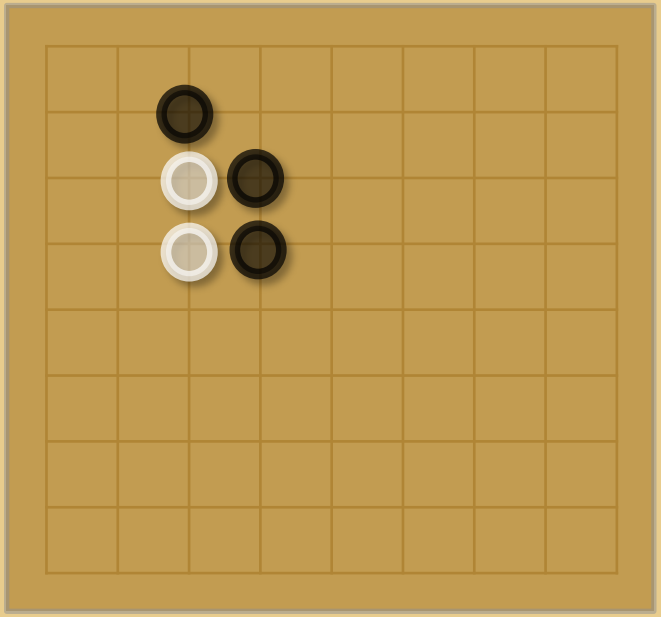
Adversarial Search – MiniMax



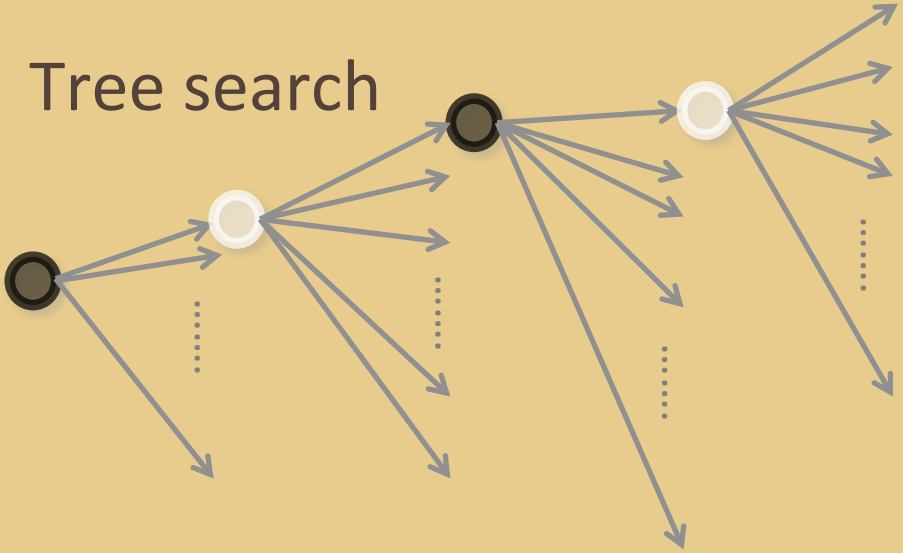
Adversarial Search – MiniMax



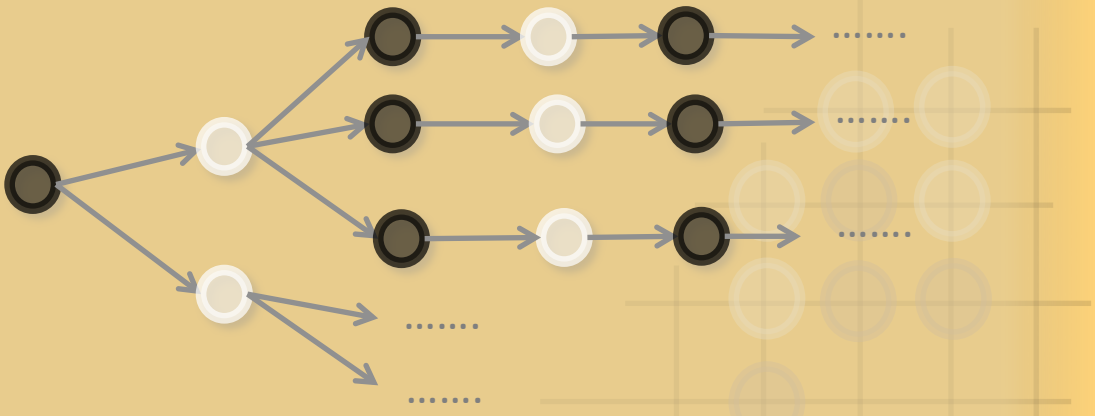
Monte Carlo Tree Search



Tree search

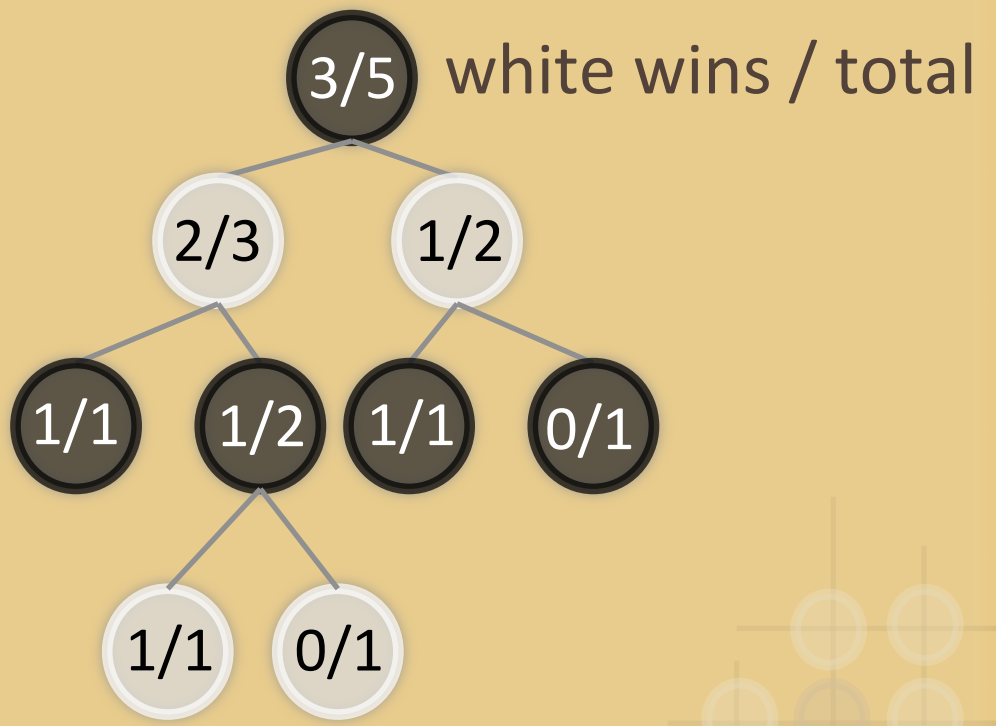


Monte Carlo search

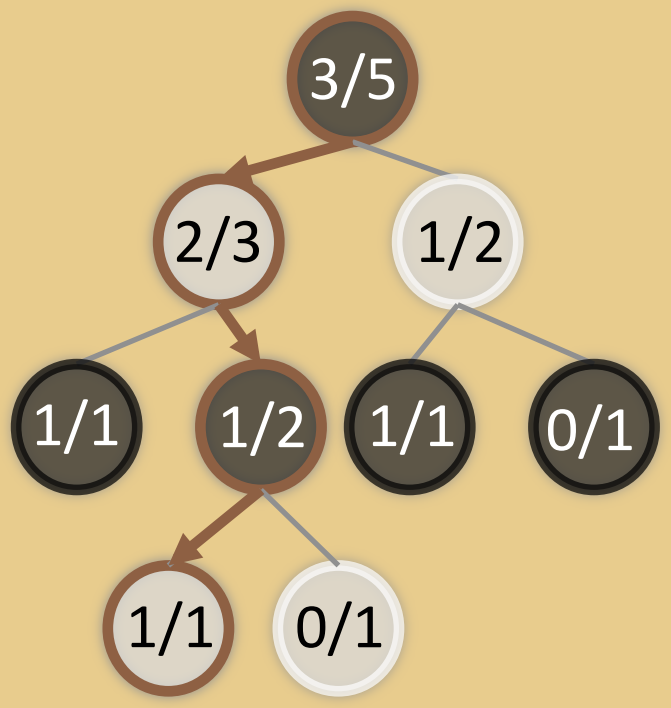


Monte Carlo Tree Search

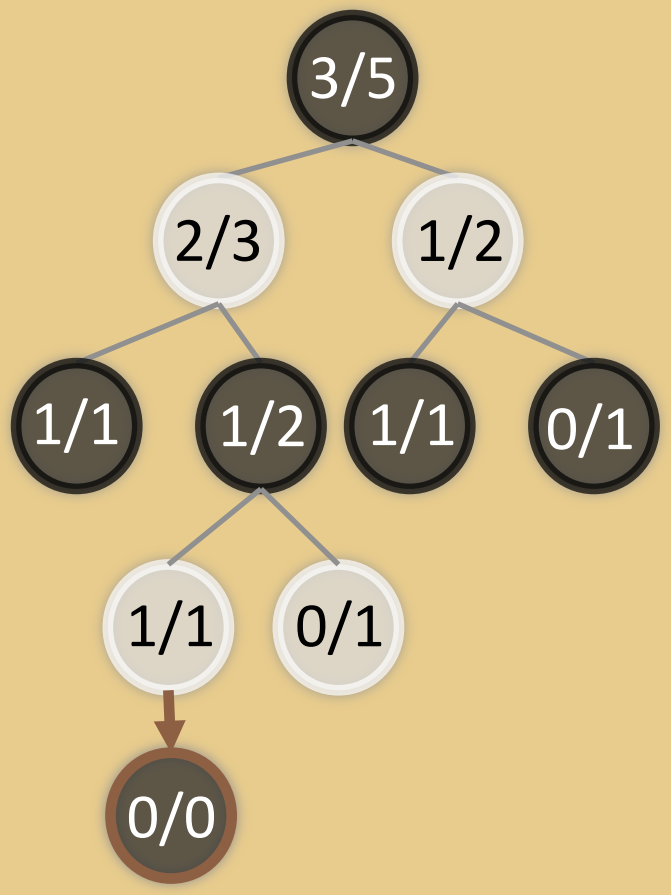
- Tree Search + Monte Carlo Method
  - Selection
  - Expansion
  - Simulation
  - Back-Propagation



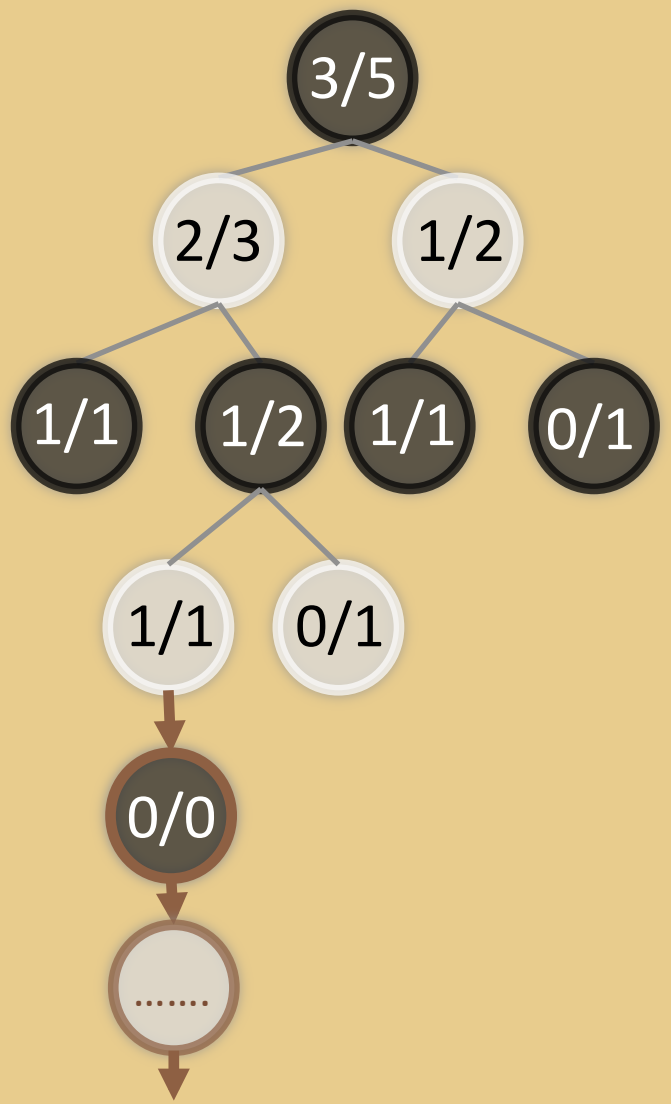
Selection



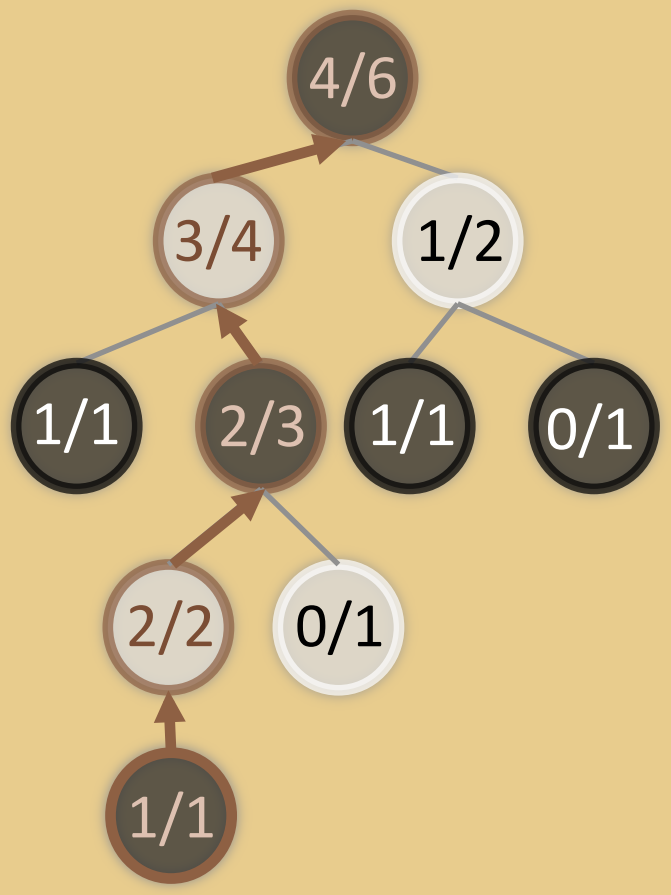
Expansion



Simulation

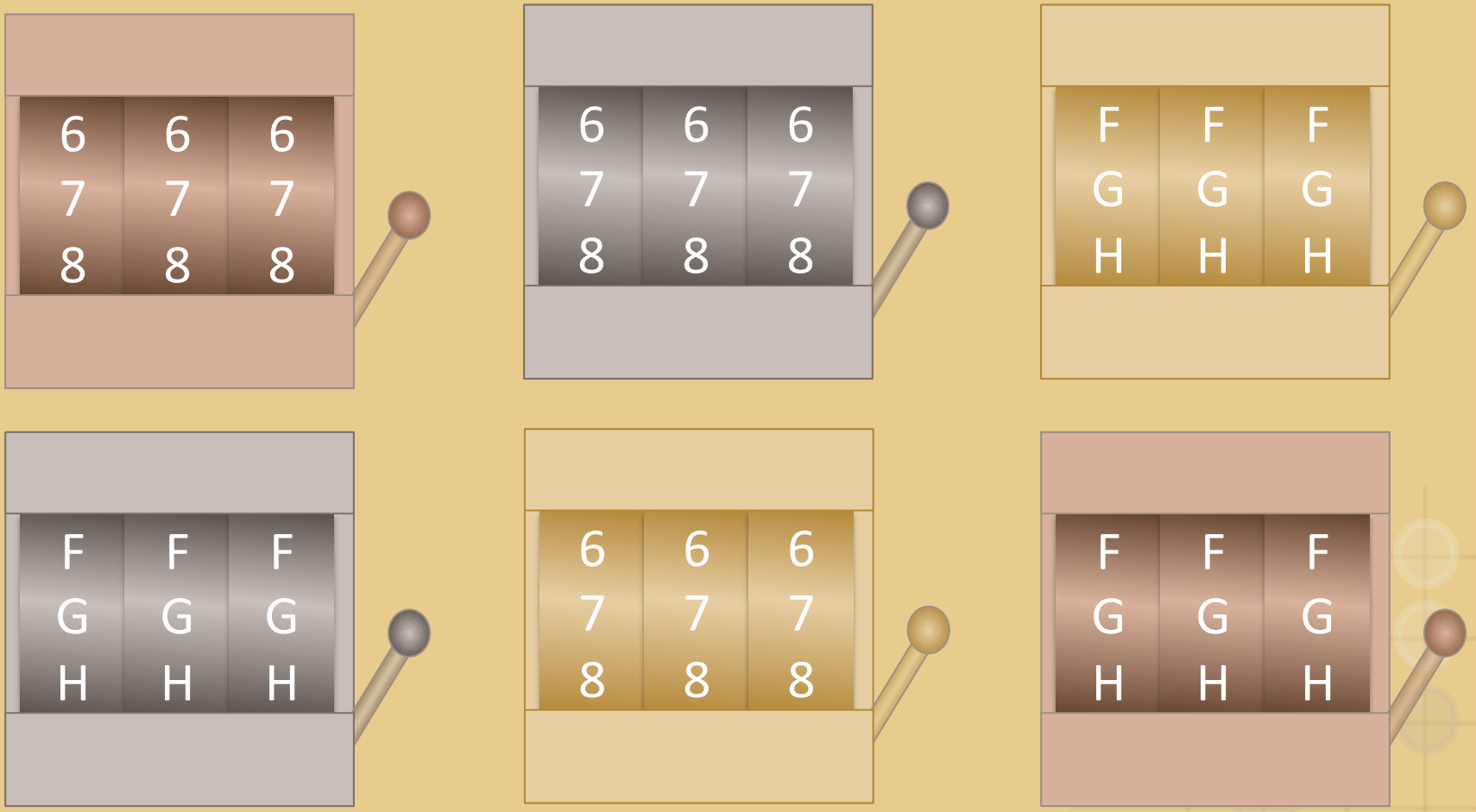


Back-Propagation

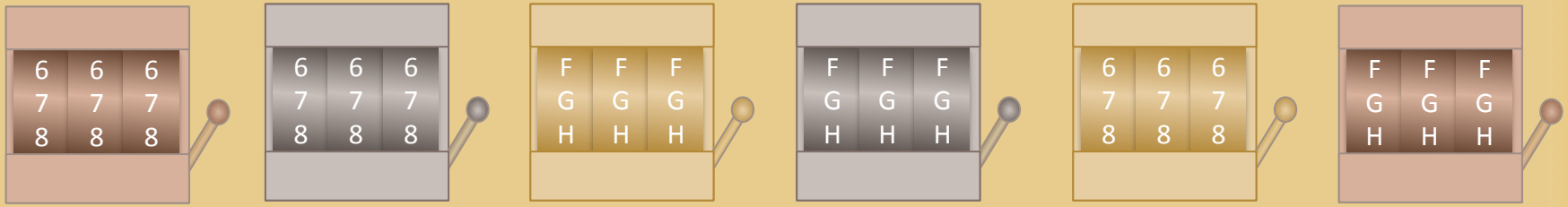


Multi-Armed Bandit Problem

- Exploration v.s. Exploitation



UCB1 algorithm



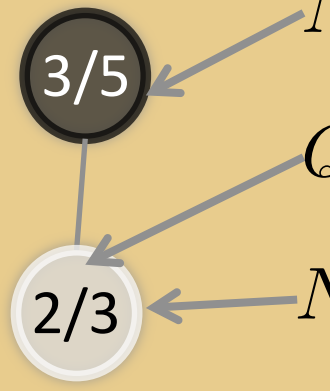
$$\underset{i}{\operatorname{argmax}}(\bar{x_i} + \sqrt{\frac{2\log n}{n_i}})$$

- $\bar{x_i}$: The mean payout for machine i
- n_i : The number of plays of machine i
- n : The total number of plays

UCB1 algorithm

$$R(s,a) = Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}}$$

$$a^* = \underset{a}{\operatorname{argmax}} R(s,a)$$

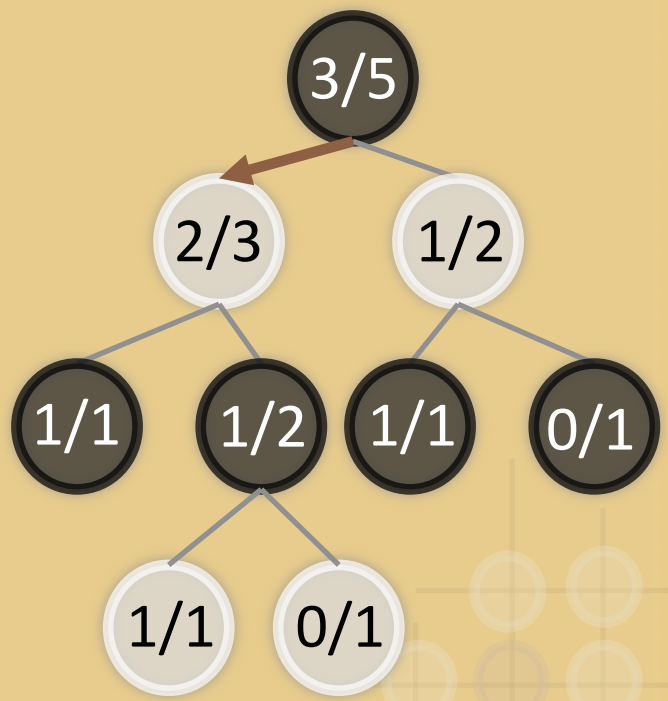


$N(s) = 5$

$Q(s,a) = 2/3$

$N(s,a) = 3$

$c = constant$



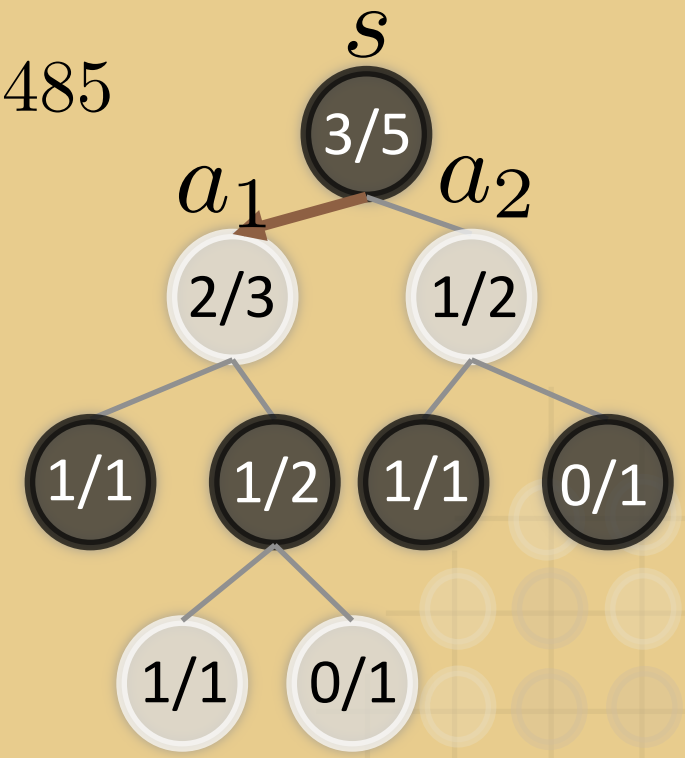
UCB1 algorithm

$$R(s, a_1) = 2/3 + 0.5\sqrt{\frac{\log 5}{3}} = 1.3029$$

$$R(s, a_2) = 1/2 + 0.5\sqrt{\frac{\log 5}{2}} = 0.9485$$

$$R(s, a_1) > R(s, a_2)$$

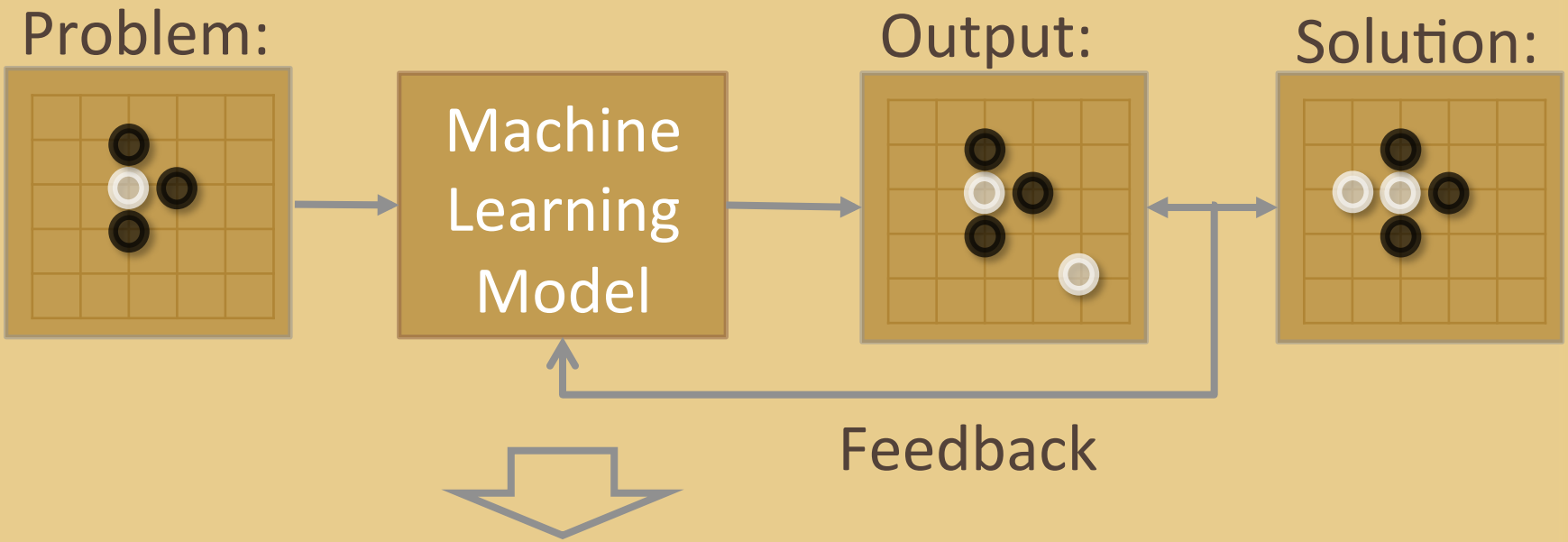
$$a^* = \underset{a}{\operatorname{argmax}} R(s, a) = a_1$$



Machine Learning and Deep Learning

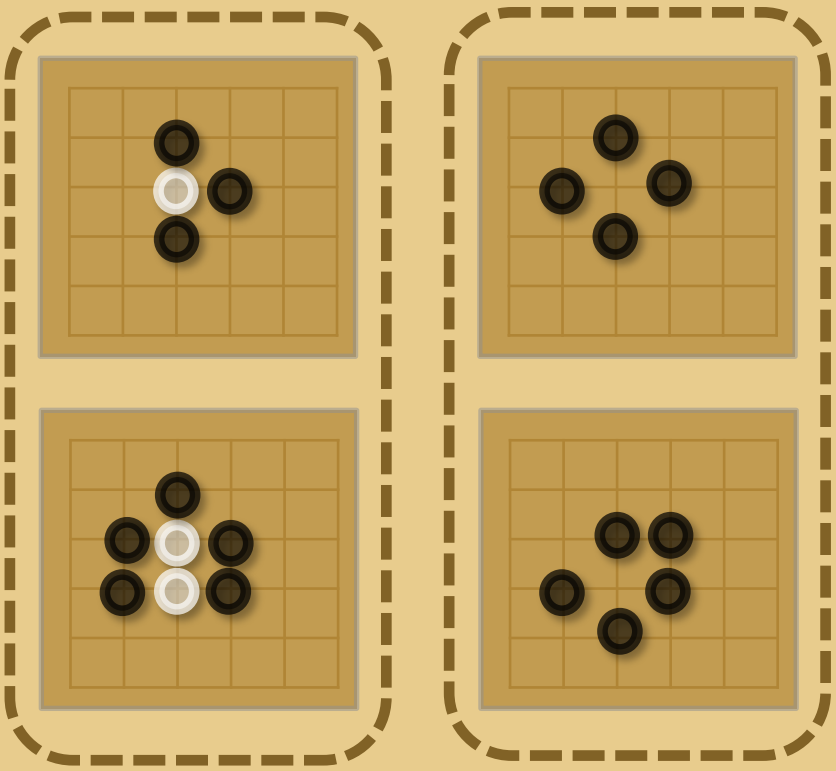
- Machine Learning and Deep Learning :
  - Supervised Machine Learning
  - Neural Networks
  - Convolutional Neural Networks
  - Training Neural Networks

Supervised Machine Learning



Supervised Machine Learning

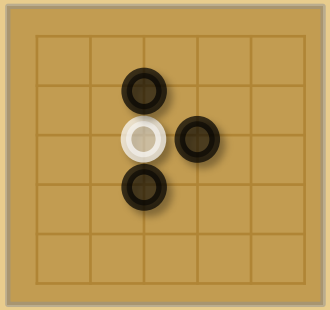
Classification



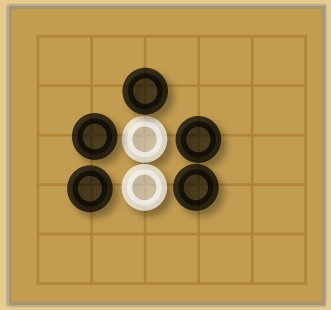
Class A

Class B

Regression

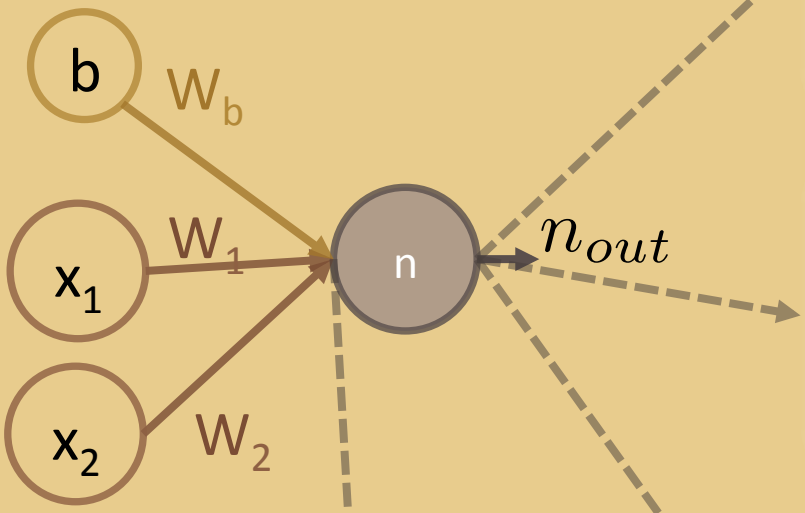


Score=1



Score=2

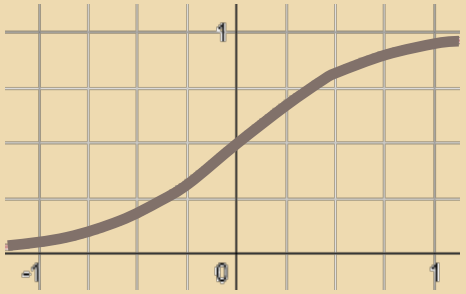
Neural Networks



$$n_{in} = w_1 x_1 + w_2 x_2 + w_b$$

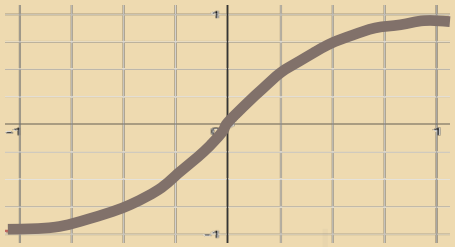
Sigmoid:

$$\frac{1}{1 + e^{-n_{in}}}$$



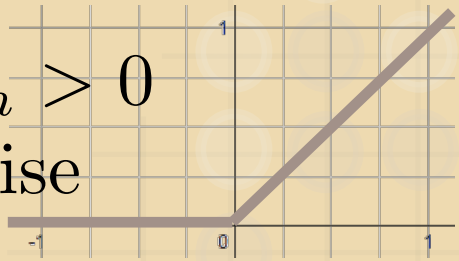
tanh:

$$\frac{1 - e^{-2n_{in}}}{1 + e^{-2n_{in}}}$$

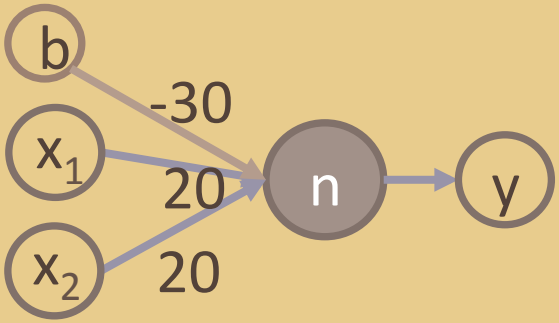


ReLU:

$$\begin{cases} n_{in} \text{ if } n_{in} > 0 \\ 0 \text{ otherwise} \end{cases}$$

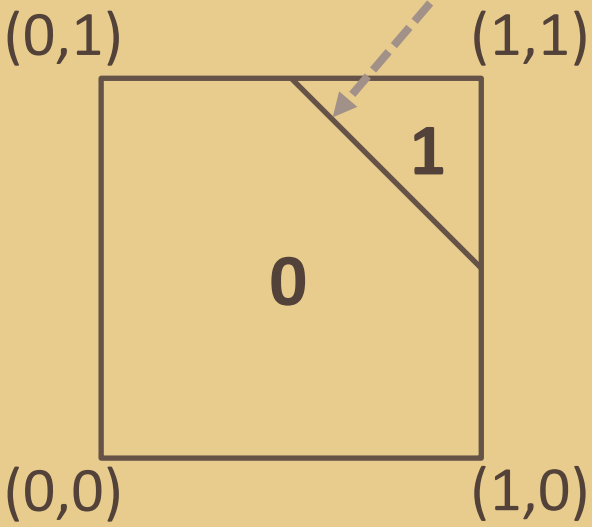


Neural Networks



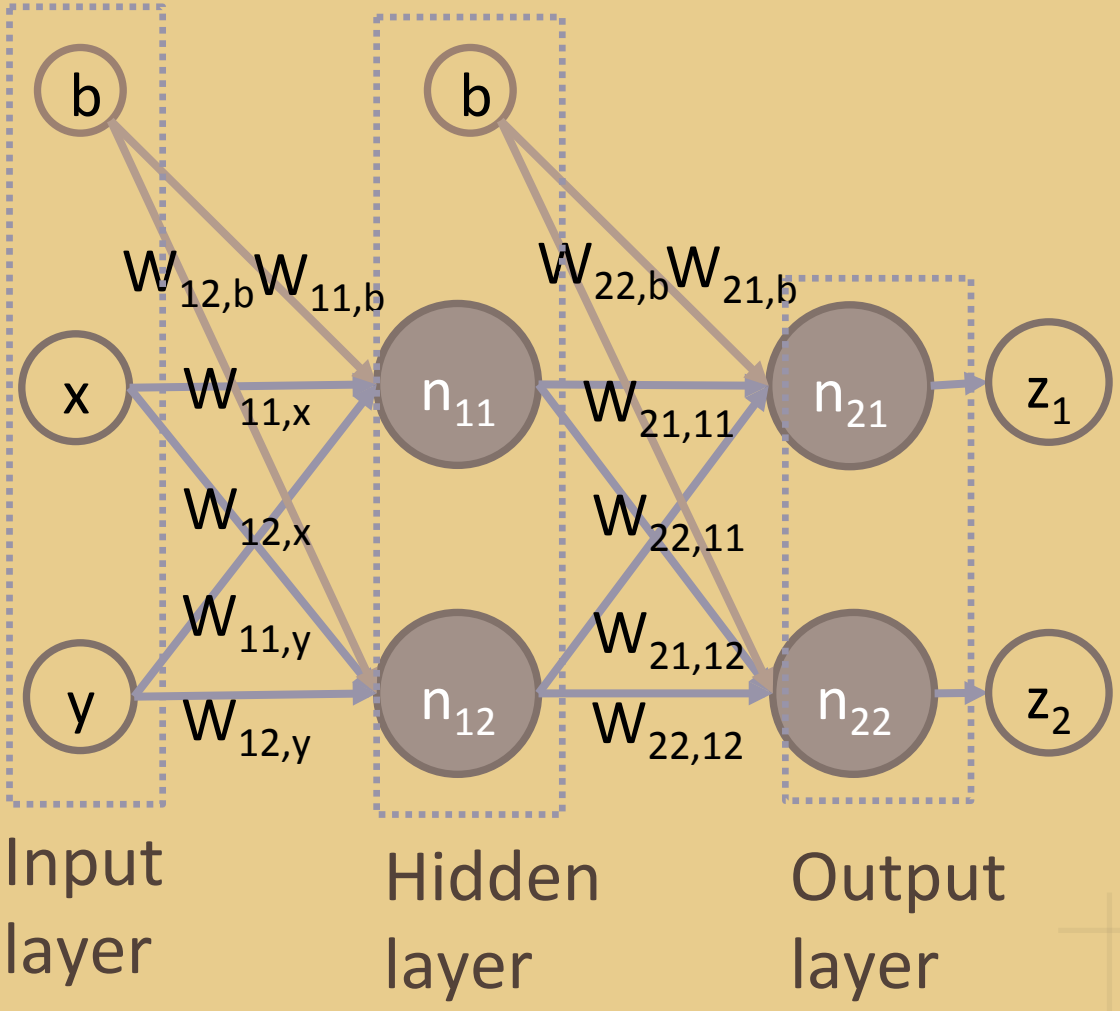
$$y = \frac{1}{1 + e^{-(20x_1 + 20x_2 - 30)}}$$

$$20x_1 + 20x_2 - 30 = 0$$



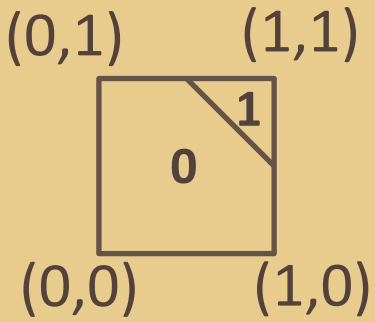
x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

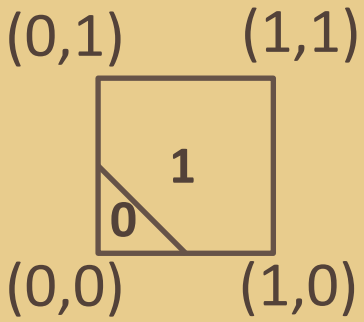
Neural Networks

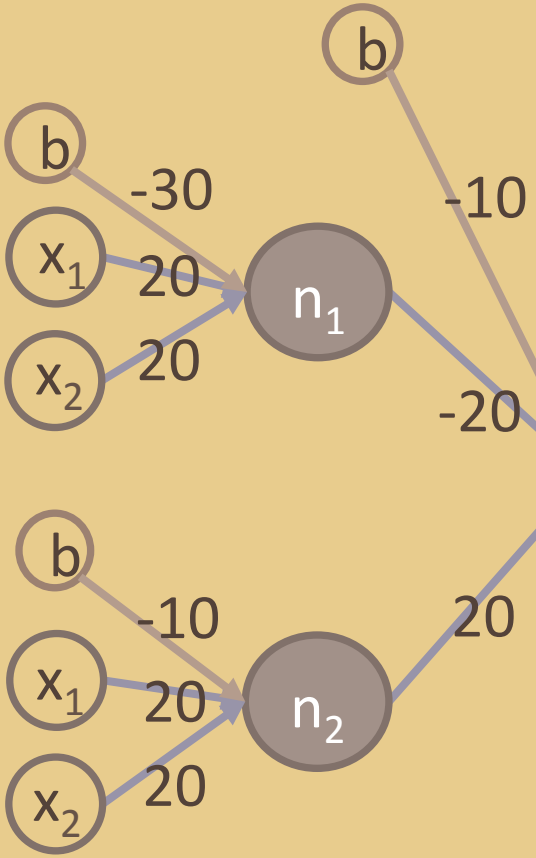


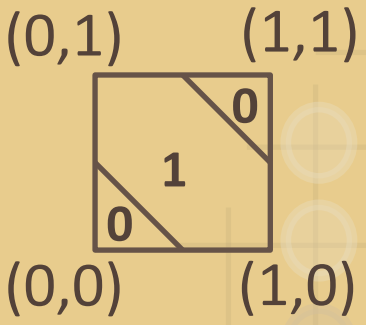
Neural Networks

x_1	x_2	n_1	n_2	y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0



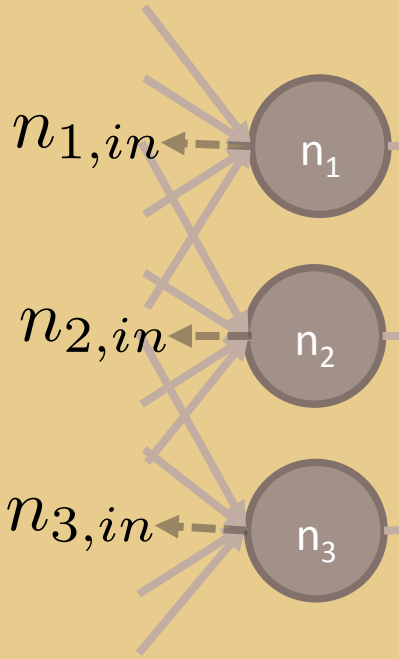






Multi-Class Classification

- SoftMax



$$n_{1,out} = \frac{e^{n_{1,in}}}{e^{n_{1,in}} + e^{n_{2,in}} + e^{n_{3,in}}}$$

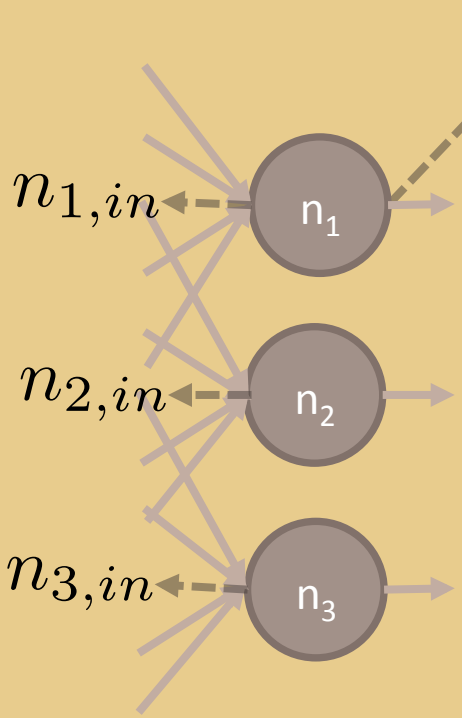
$$n_{2,out} = \frac{e^{n_{2,in}}}{e^{n_{1,in}} + e^{n_{2,in}} + e^{n_{3,in}}}$$

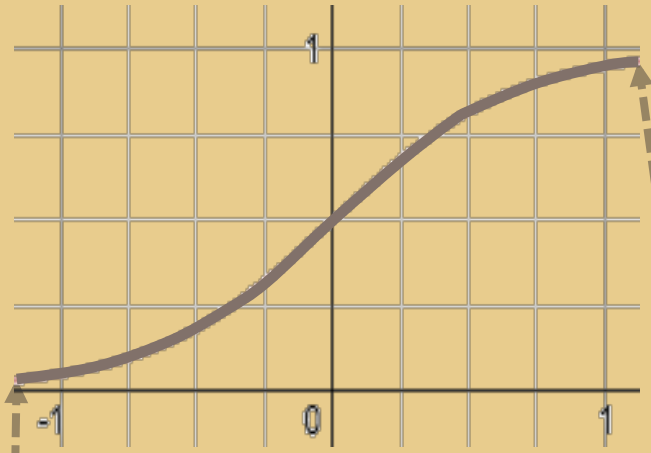
$$n_{3,out} = \frac{e^{n_{3,in}}}{e^{n_{1,in}} + e^{n_{2,in}} + e^{n_{3,in}}}$$

Multi-Class Classification

- SoftMax

$$n_{1,out} = \frac{e^{n_{1,in}}}{e^{n_{1,in}} + e^{n_{2,in}} + e^{n_{3,in}}}$$



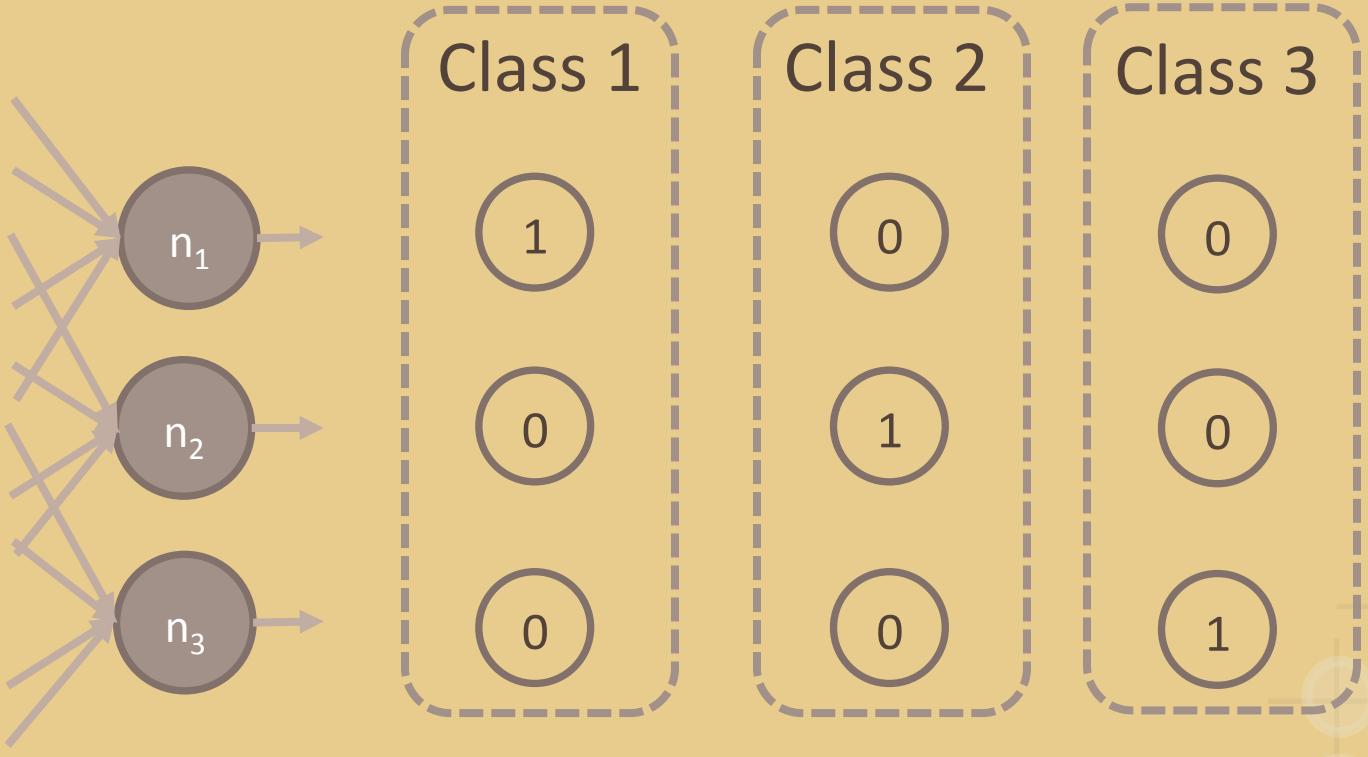


$n_{1,in} < n_{2,in}$ *or* $n_{1,in} < n_{3,in}$

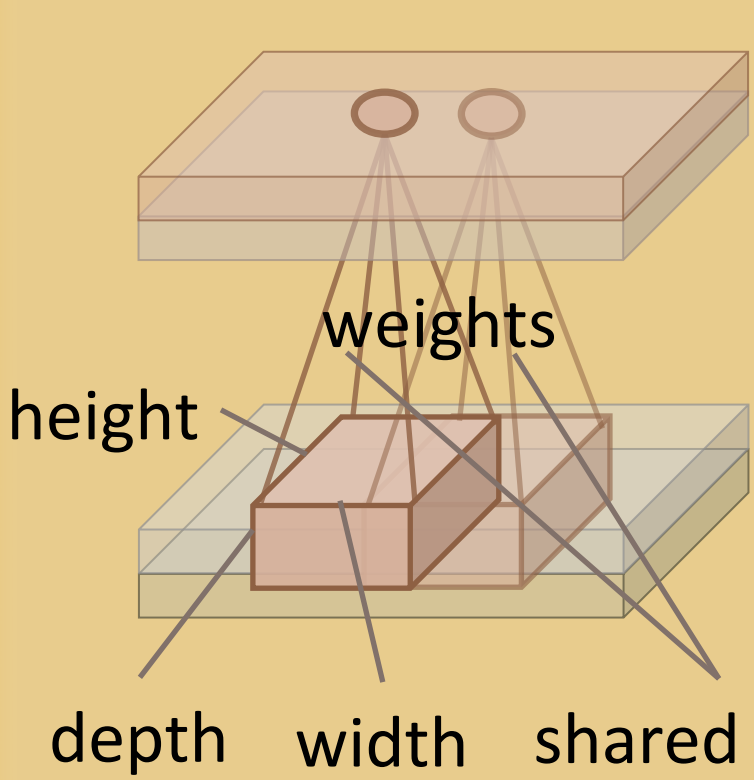
$n_{1,in} > n_{2,in}$ *and* $n_{1,in} > n_{3,in}$

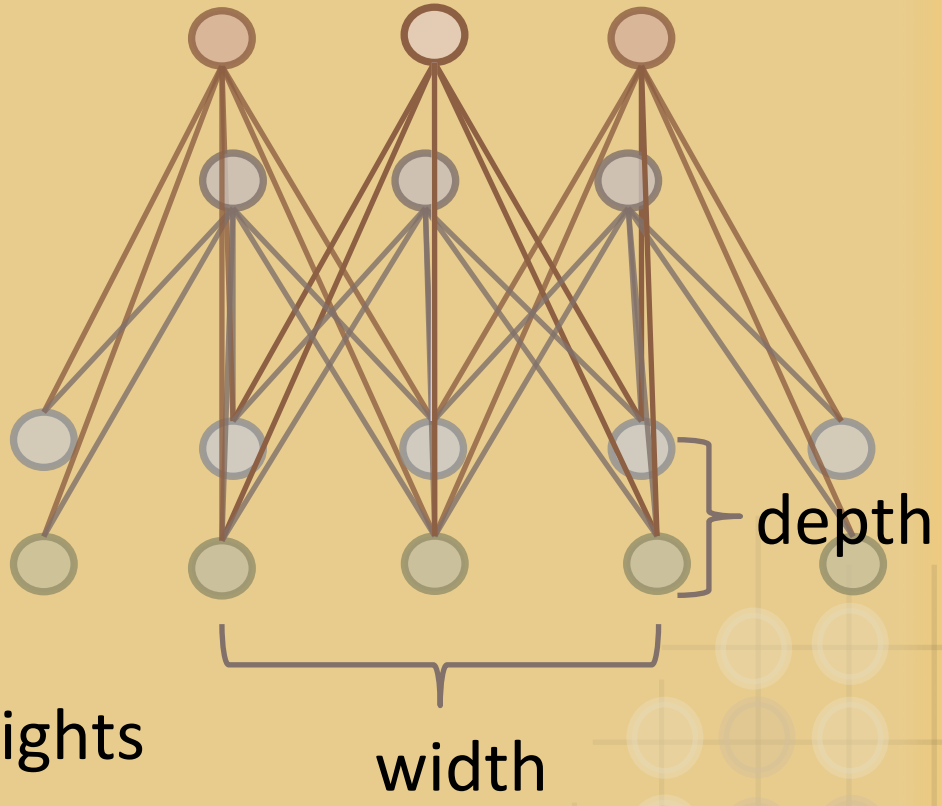
Multi-Class Classification

- One-Hot Encoding:

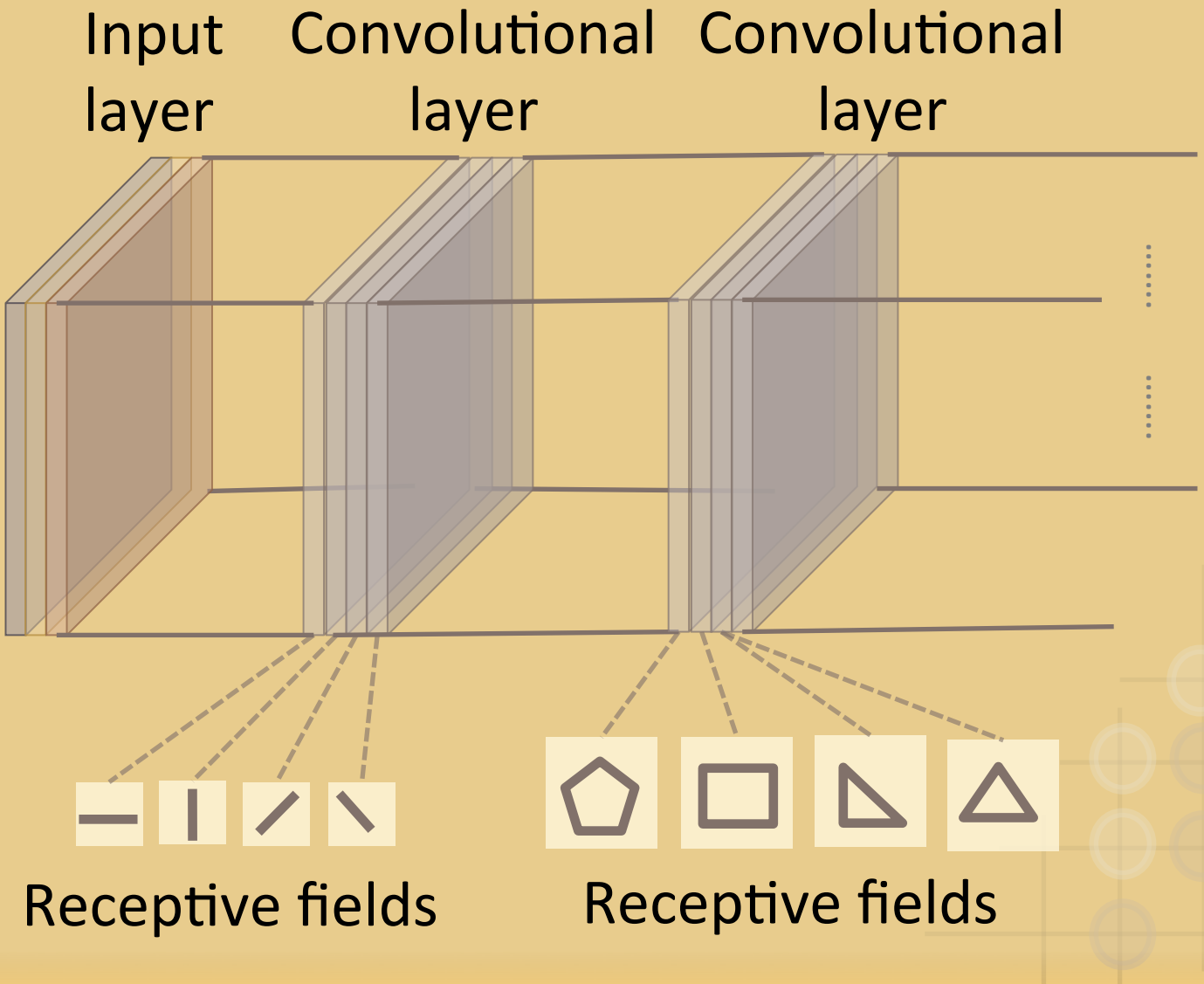


Convolutional Neural Neworks

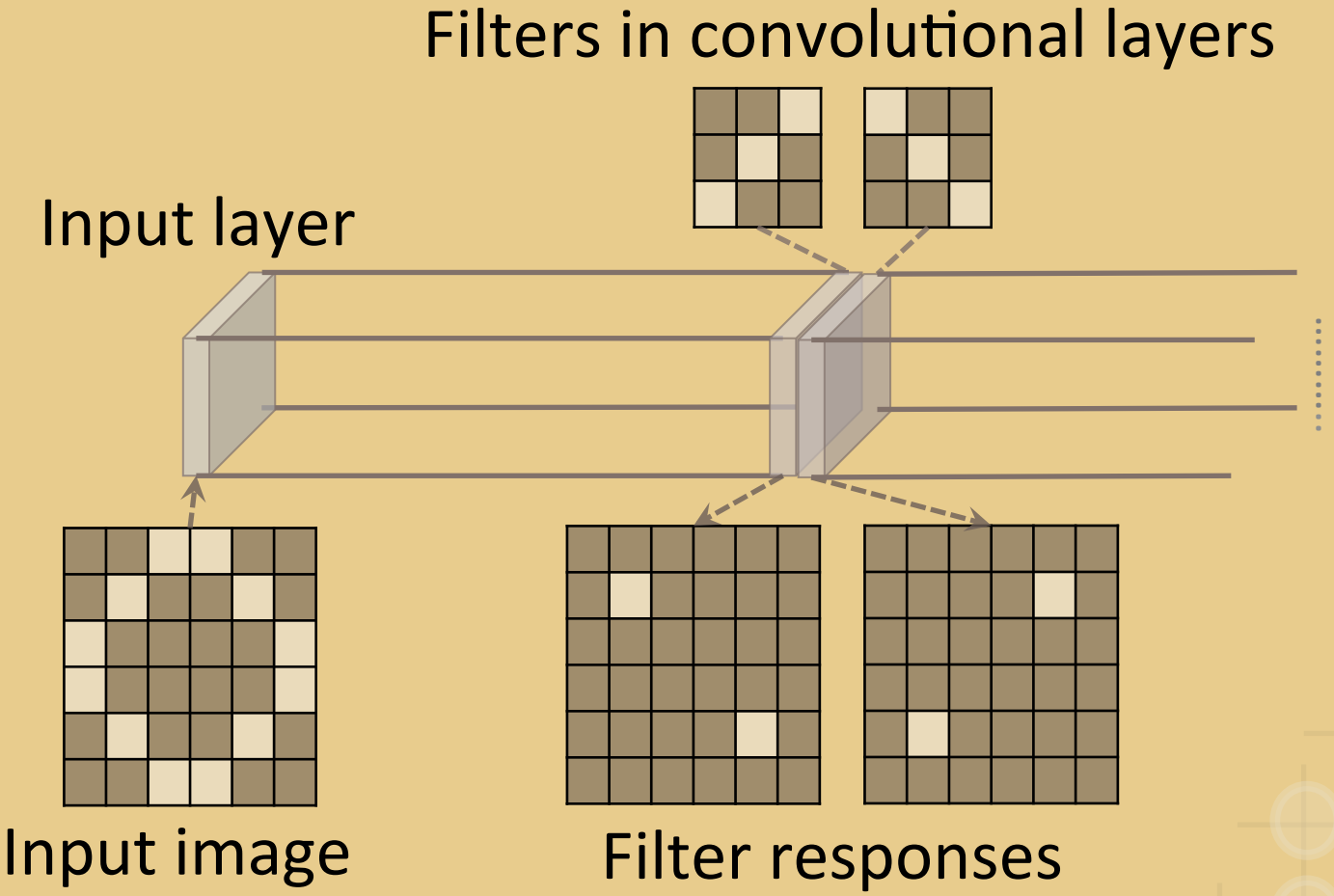




Convolutional Neural Neworks

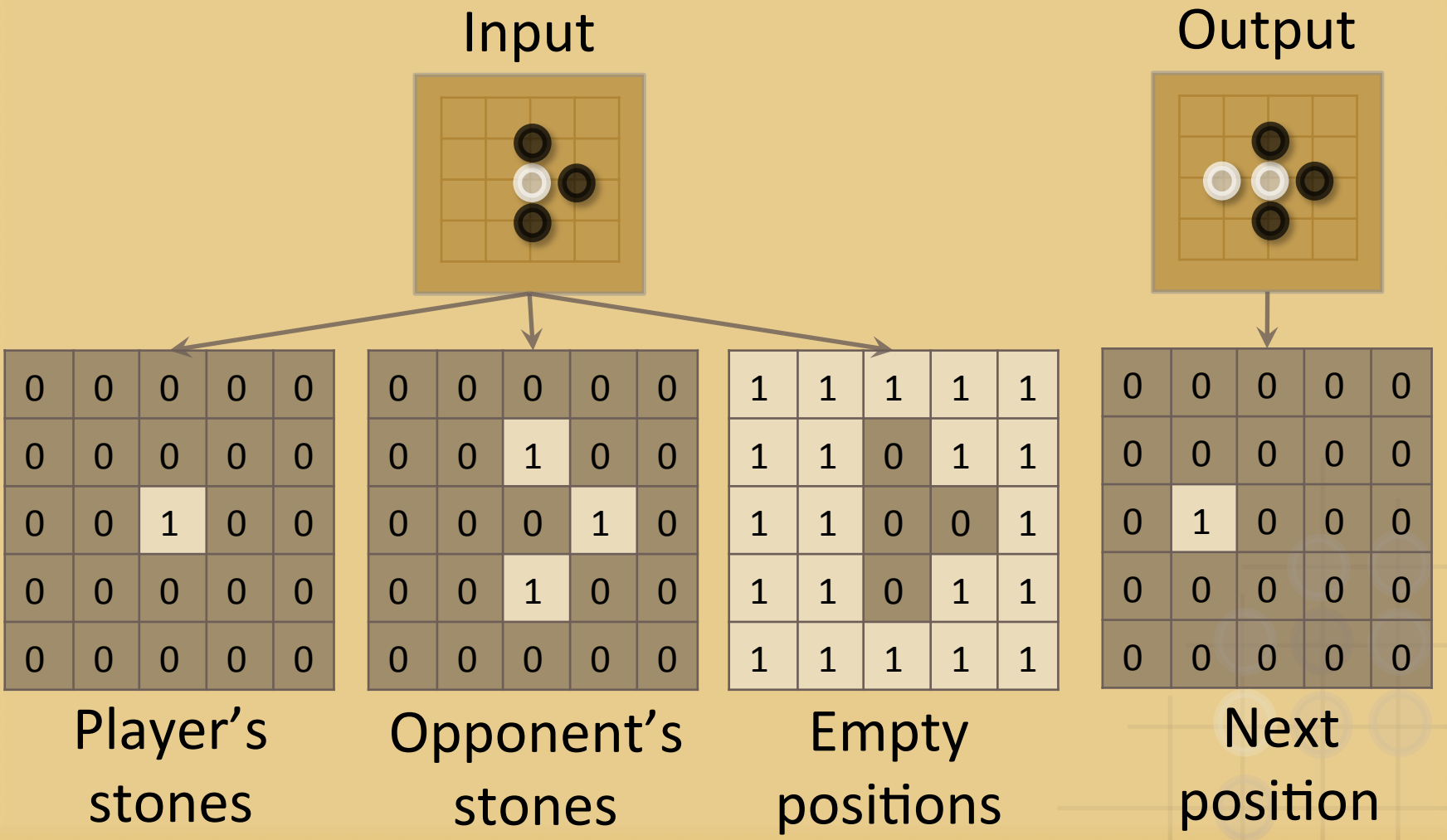


Convolutional Neural Neworks

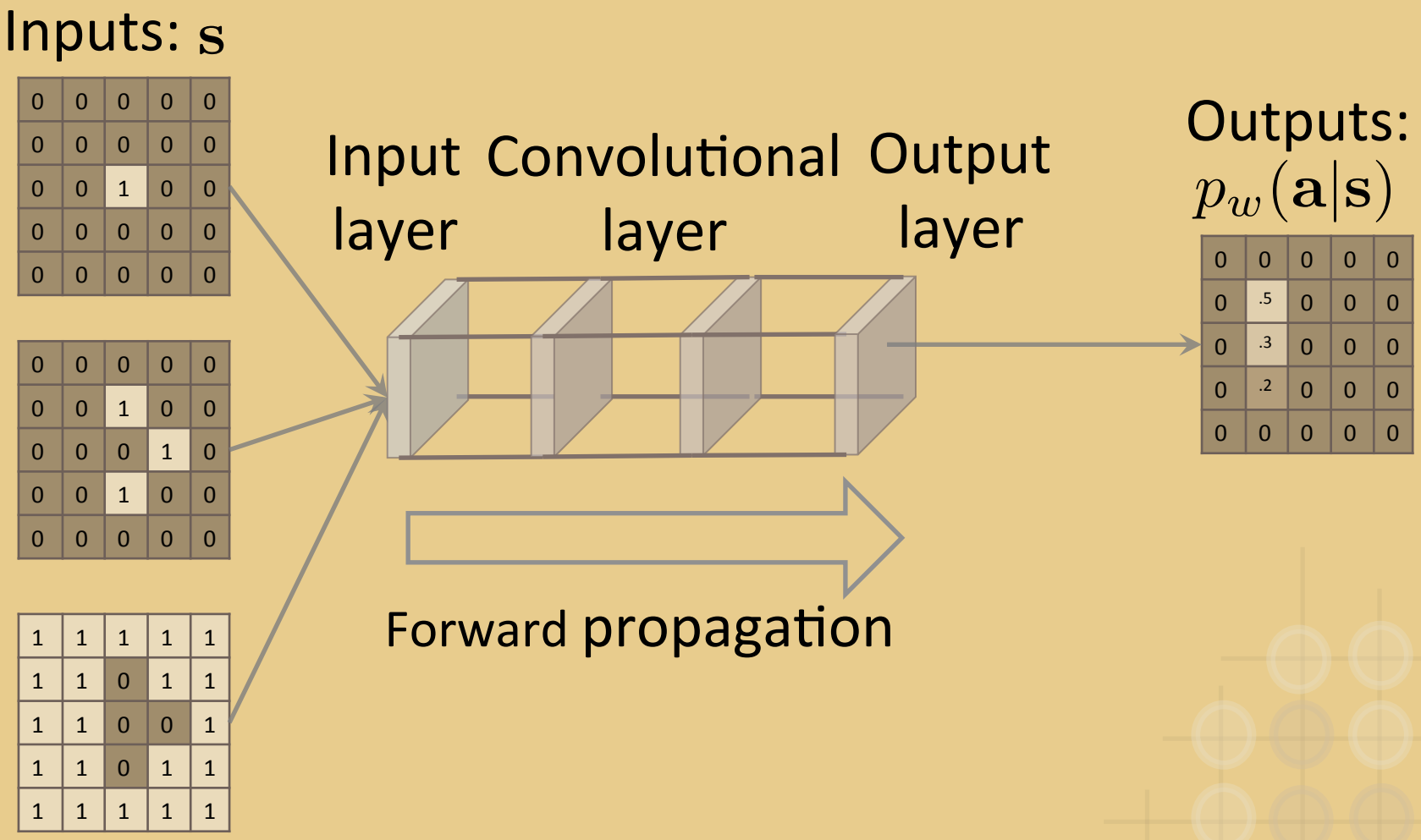


Training Neural Networks

- One-Hot Encoding:

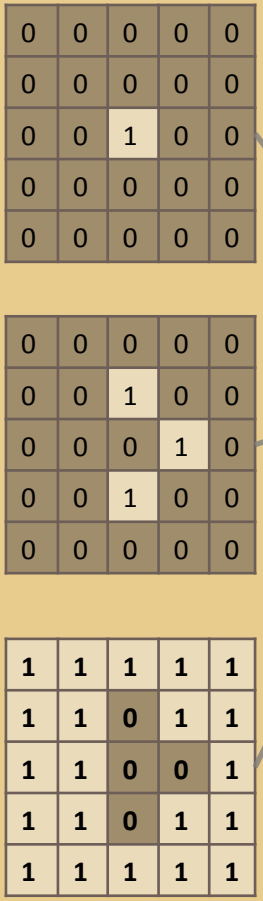


Training Neural Networks



Training Neural Networks

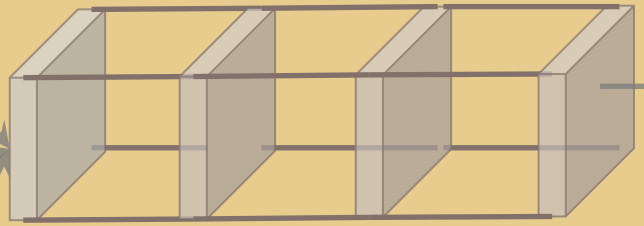
Inputs: $\mathbf{s}$



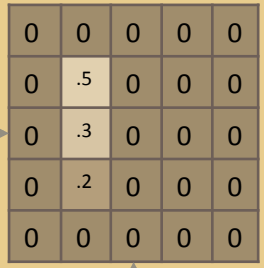
Input layer

Convolutional layer

Output layer



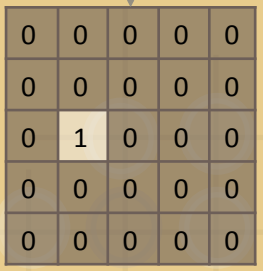
Outputs: $p_w(\mathbf{a}|\mathbf{s})$



Cost function: $-\log(p_w(a|\mathbf{s}))$

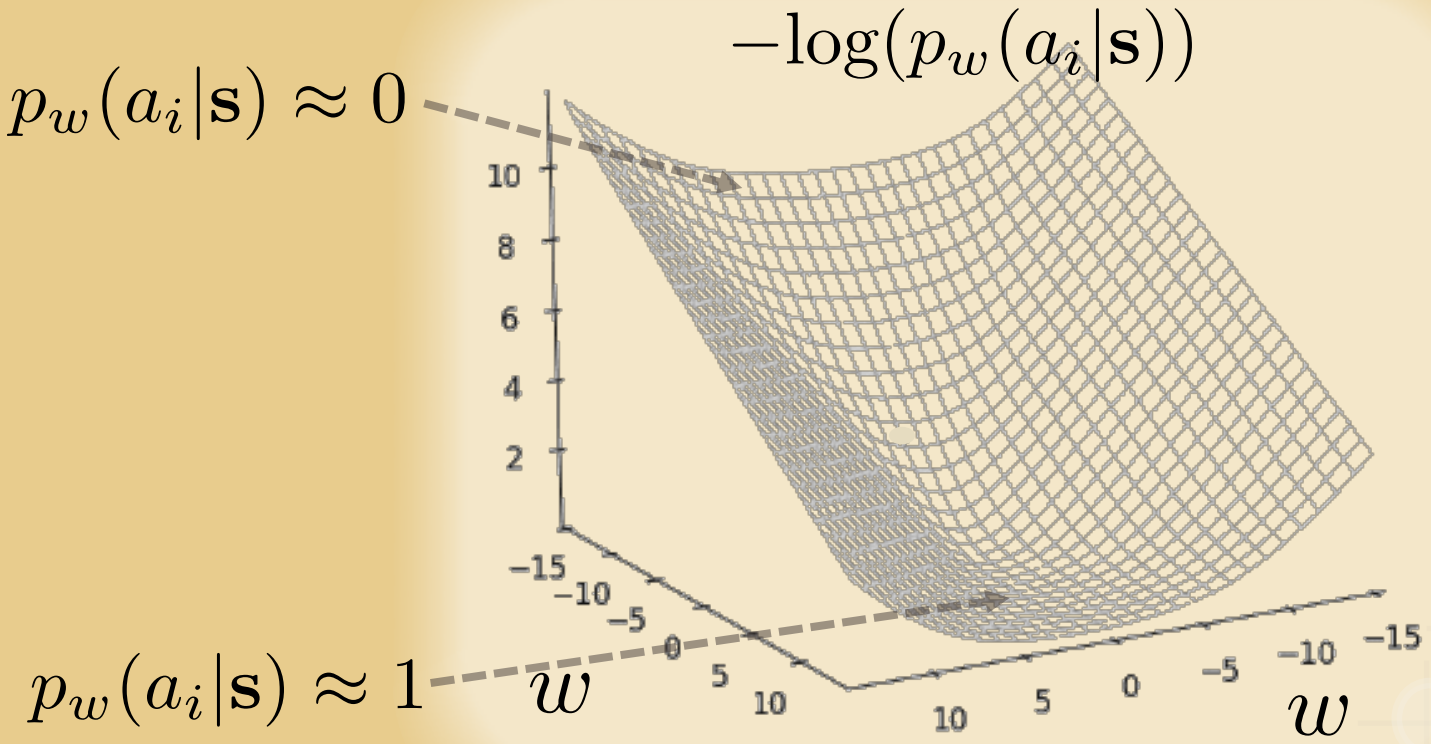
Backward propagation

$$w = w - \eta \frac{\partial - \log(p_w(a_i|\mathbf{s}))}{\partial w}$$

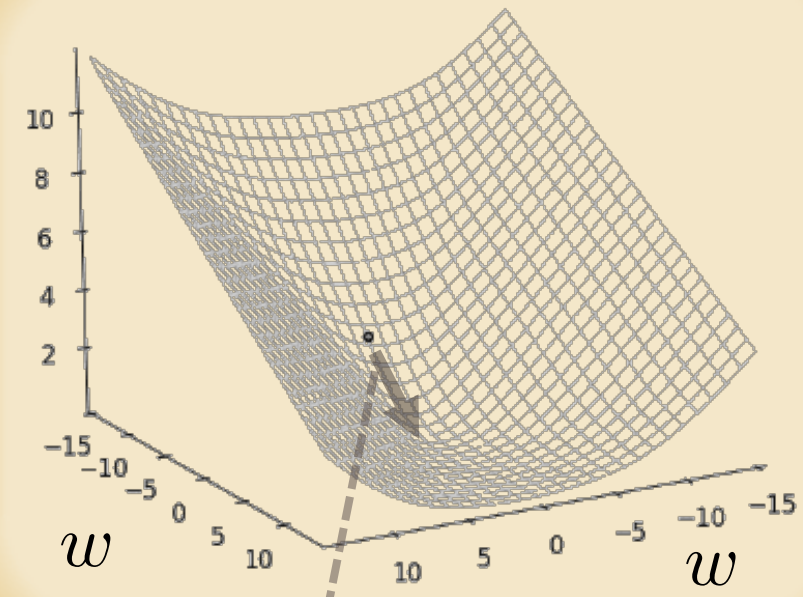


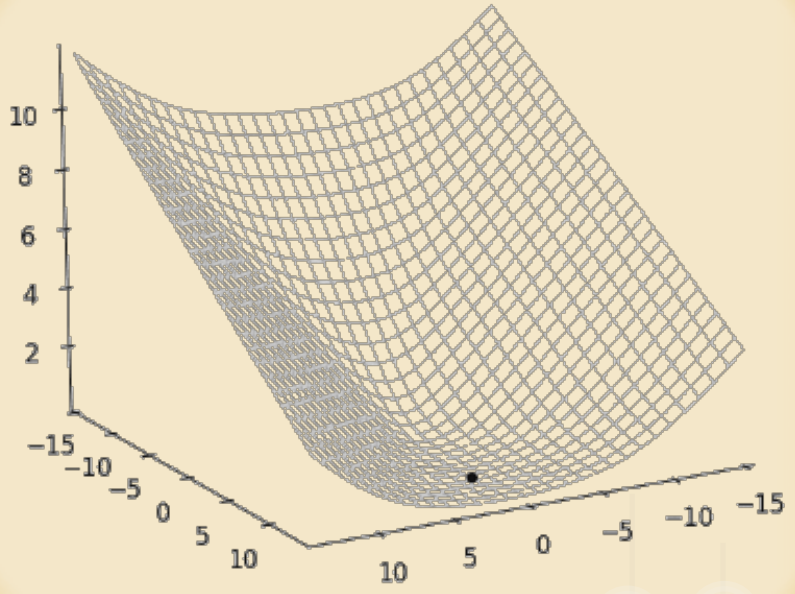
Golden: a_i

Cost Function



Gradient Descent

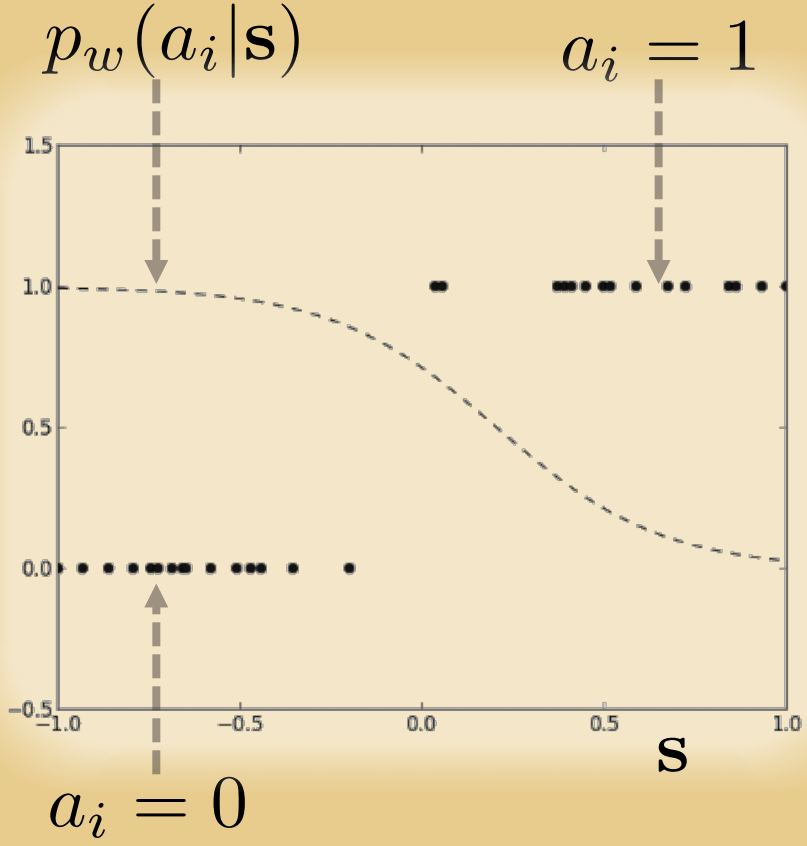


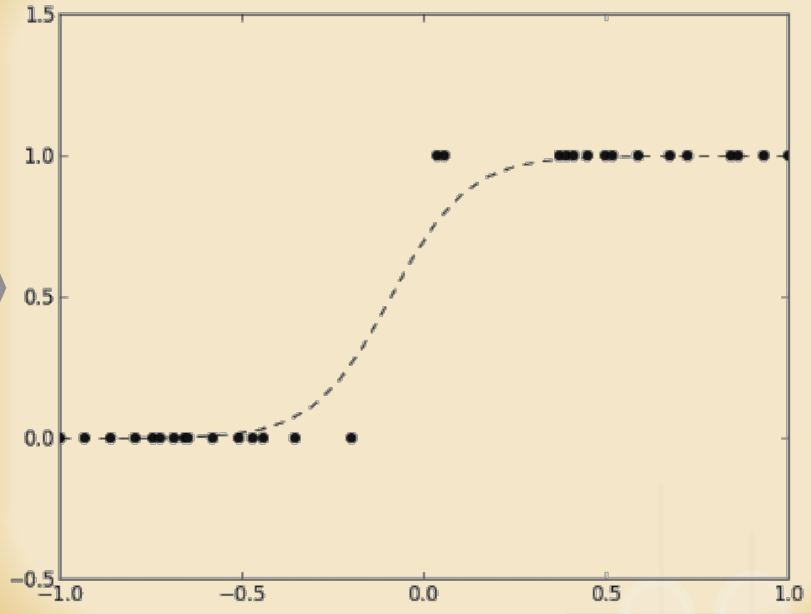


$$-\frac{\partial - \log(p_w(a_i|\mathbf{s}))}{\partial w} \qquad w = w - \eta \frac{\partial - \log(p_w(a_i|\mathbf{s}))}{\partial w}$$

: Learning Rate

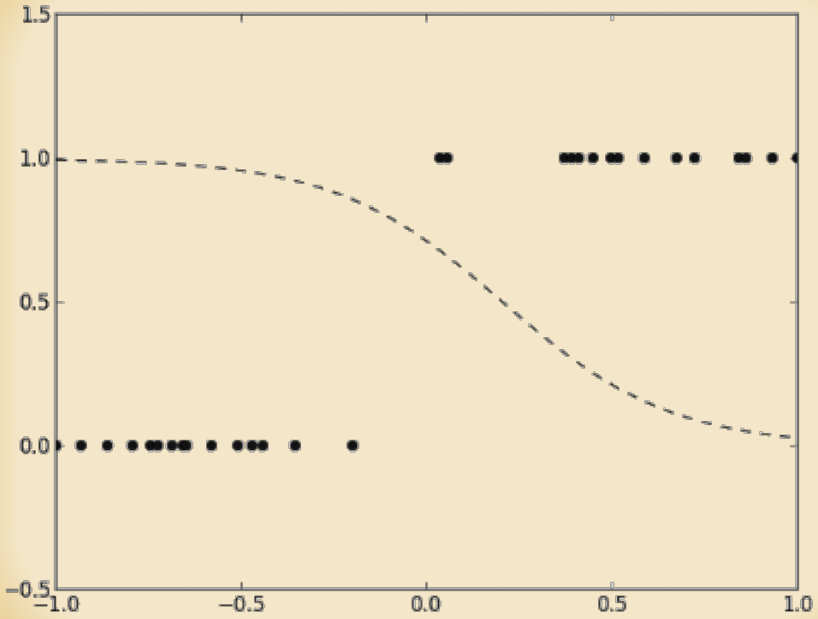
Gradient Descent

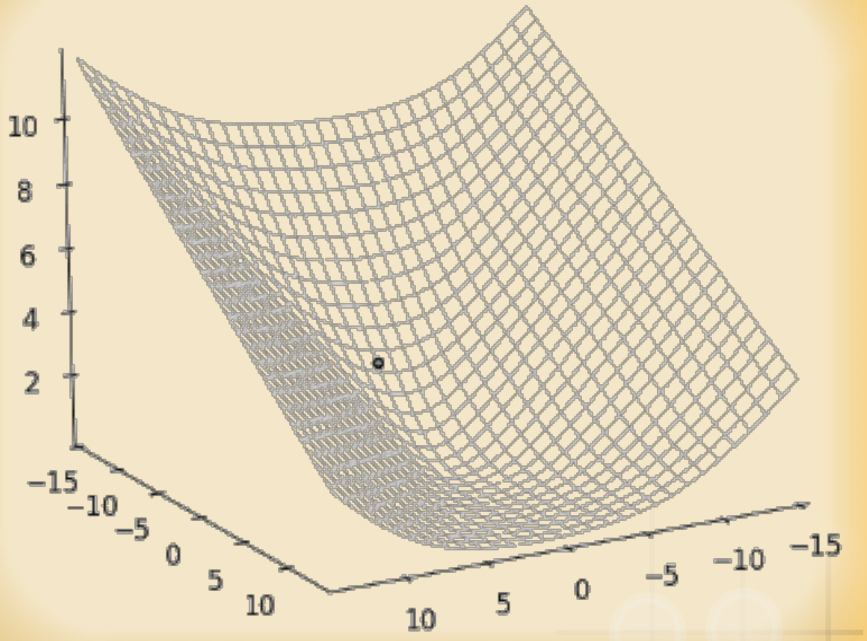




$$w \leftarrow w - \eta \frac{\partial \log(p_w(a_i|\mathbf{s}))}{\partial w}$$

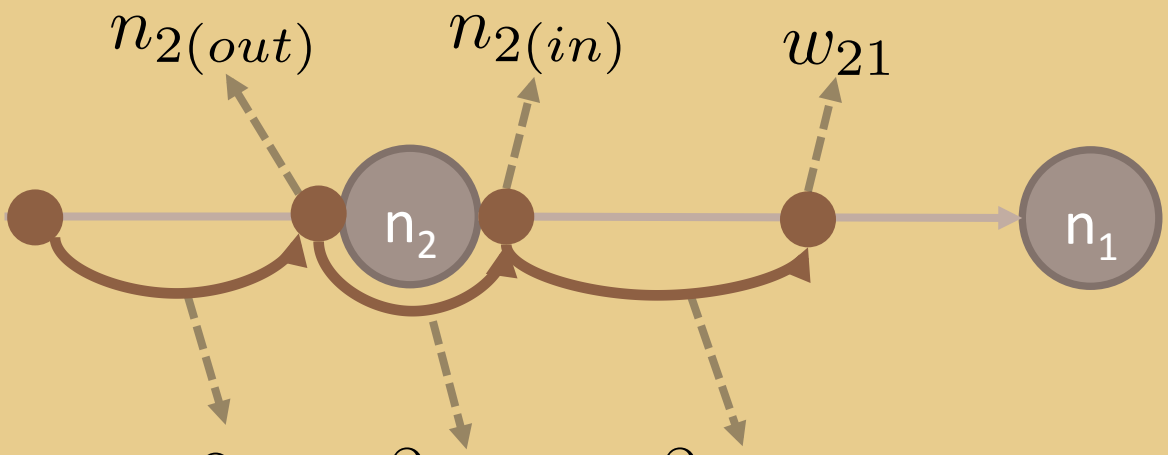
Gradient Descent





Backward Propagation

Cost function: J



$$\frac{\partial J}{\partial n_{2(out)}} = \frac{\partial J}{\partial n_{2(out)}} \frac{\partial n_{2(out)}}{\partial n_{2(in)}} \frac{\partial n_{2(in)}}{\partial w_{21}}$$

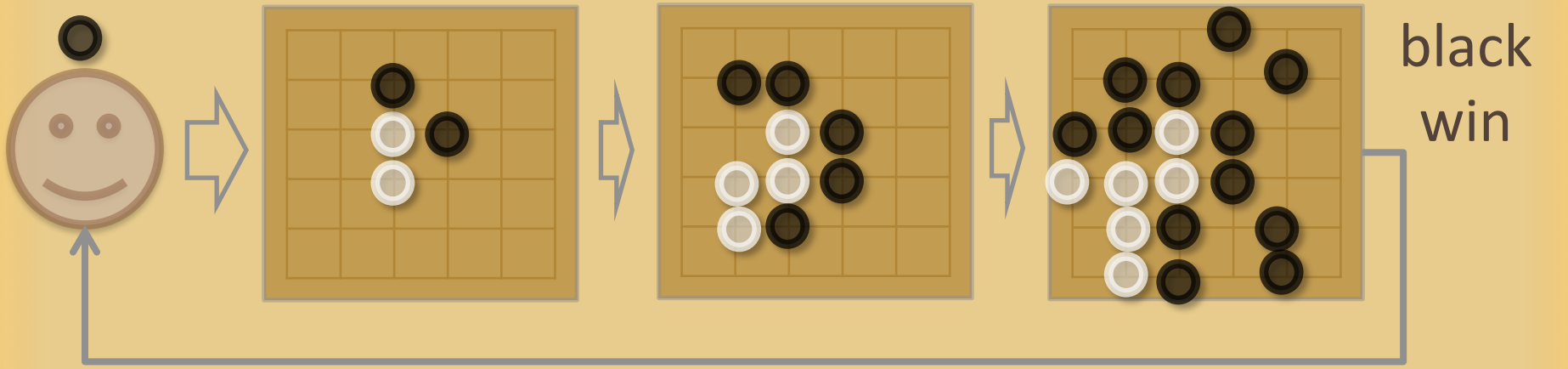
$$w_{21} \leftarrow w_{21} - \eta \frac{\partial J}{\partial w_{21}}$$

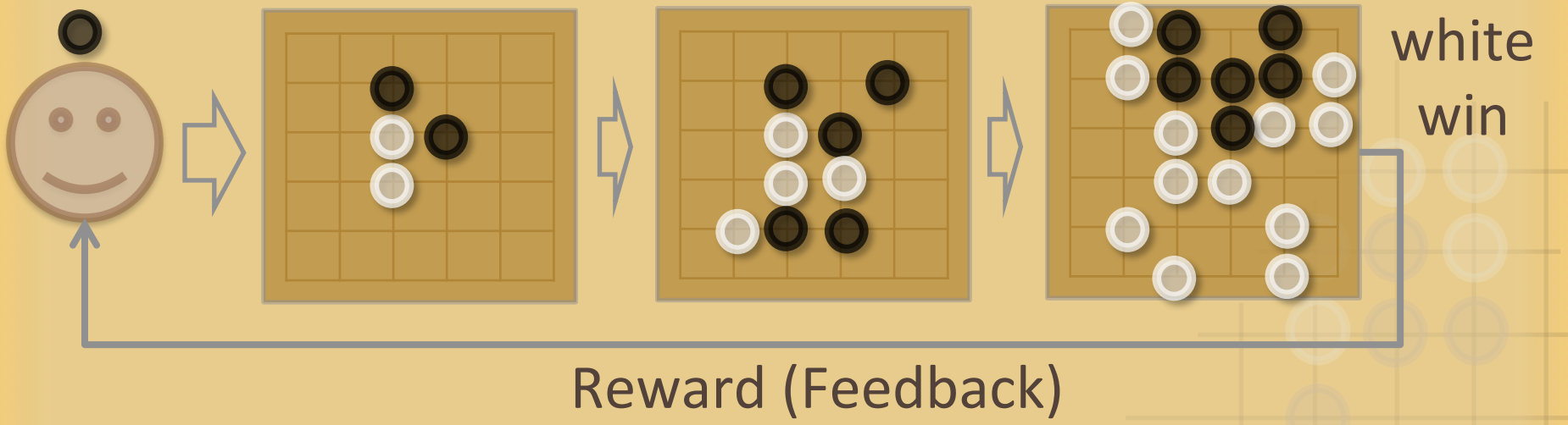
$$w_{21} \leftarrow w_{21} - \eta \frac{\partial J}{\partial n_{2(out)}} \frac{\partial n_{2(out)}}{\partial n_{2(in)}} \frac{\partial n_{2(in)}}{\partial w_{21}}$$

Reinforcement Learning

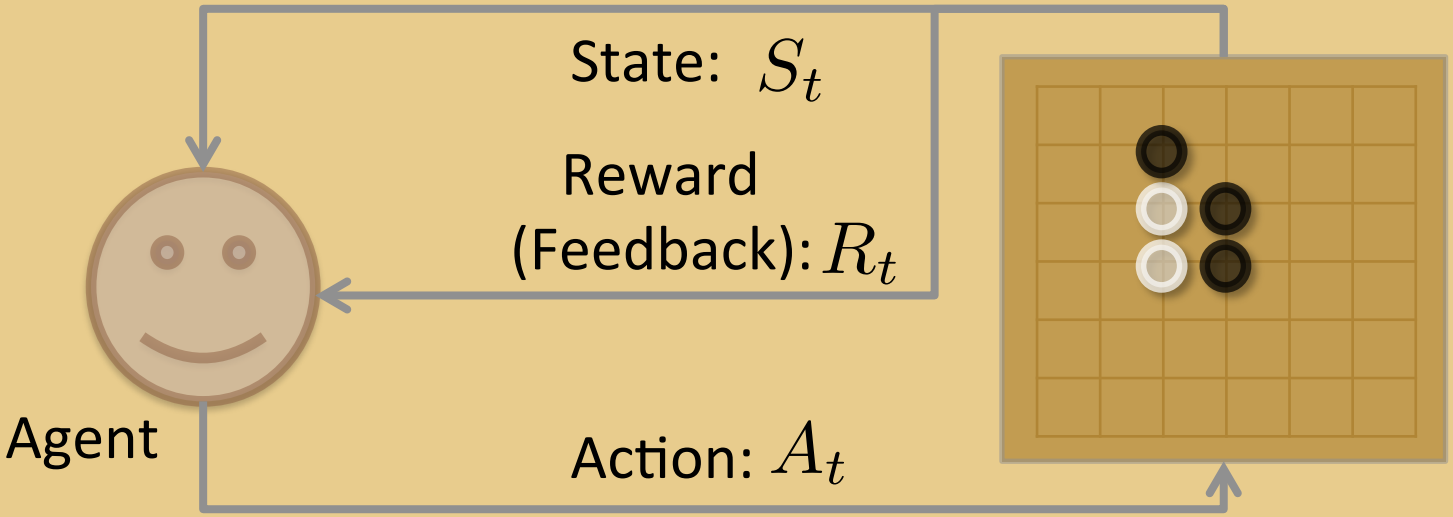
- Reinforcement Learning :
  - Policy & Value
  - Policy Gradient Method

Reinforcement Learning





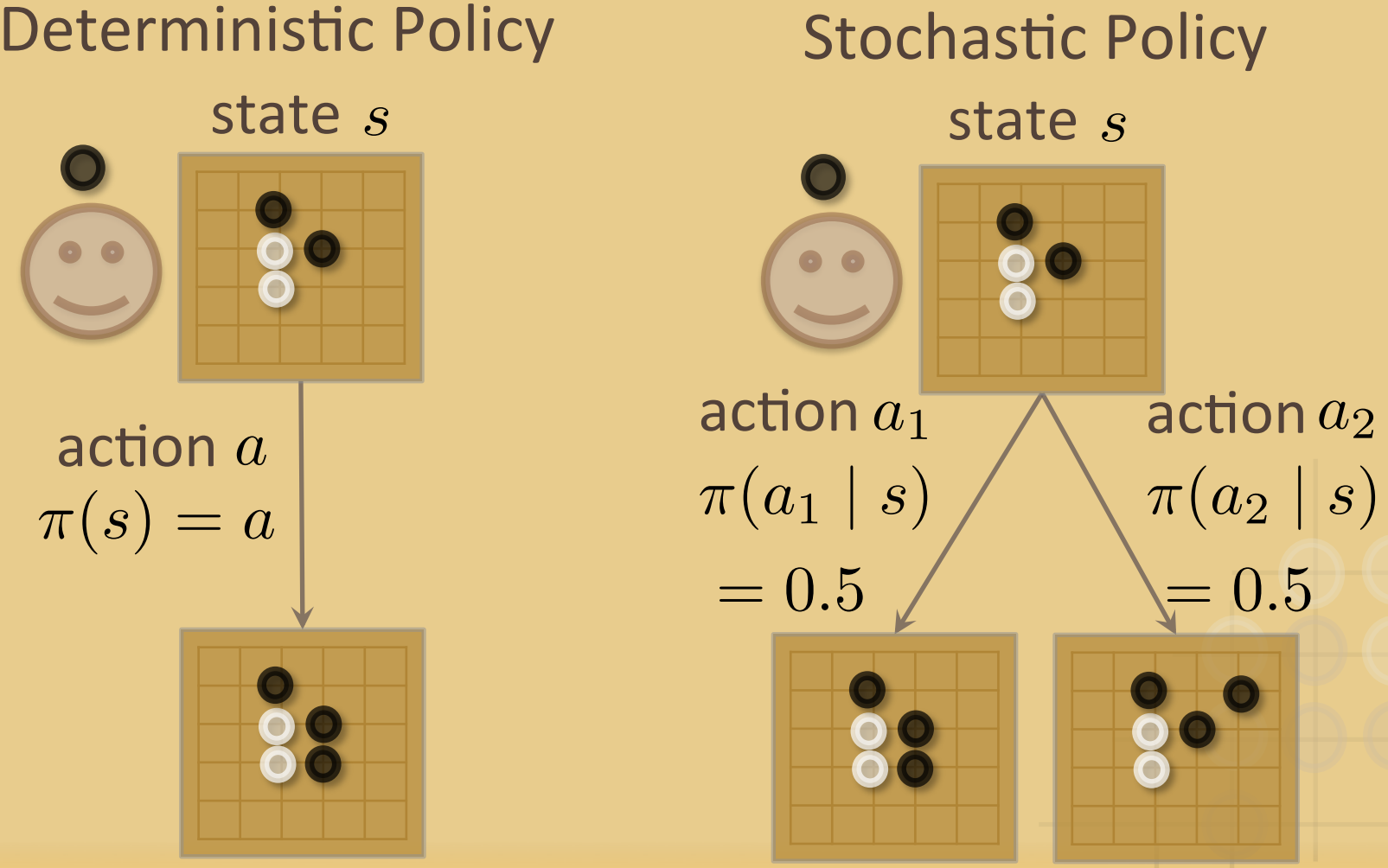
Reinforcement Learning



- Feedback is delayed.
- No supervisor, only a reward signal.
- Rules of the game are unknown.
- Agent's actions affect the subsequent state

Policy

- The behavior of an agent

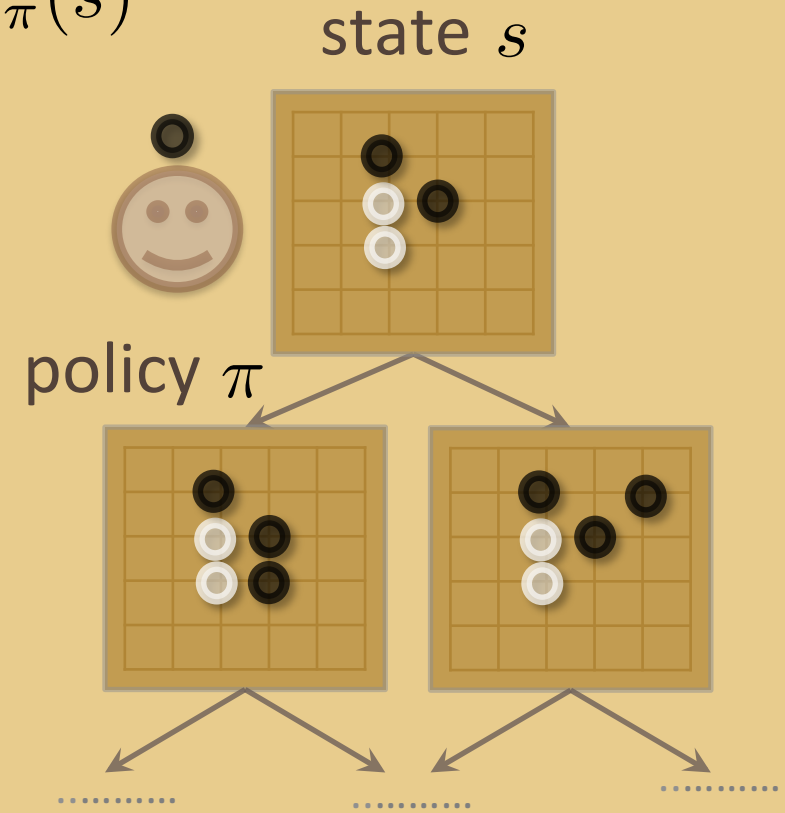


Value

- The expected long-term reward

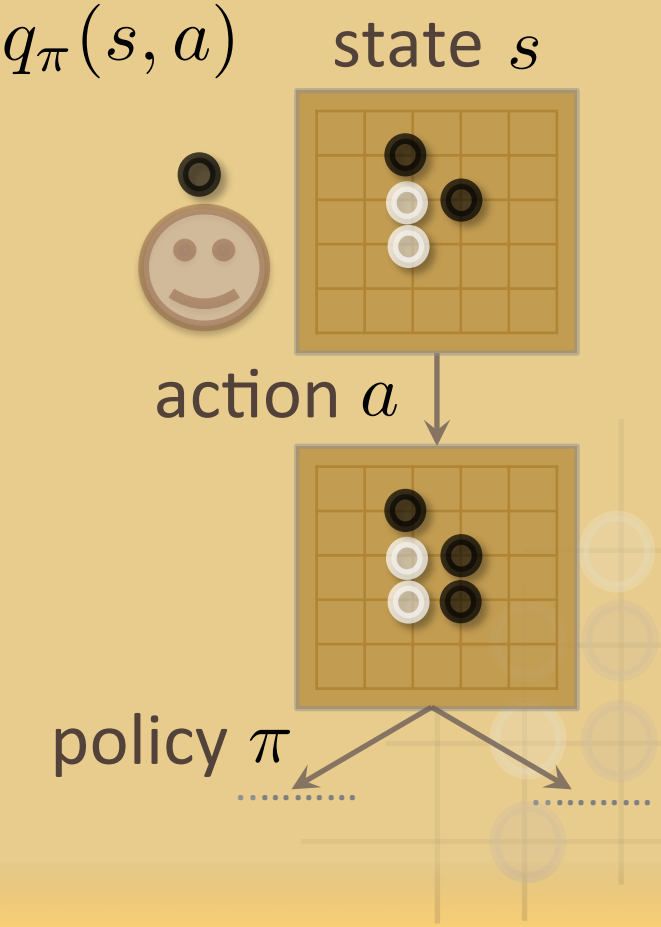
State-value Function

$v_\pi(s)$



Action-value Function

$q_\pi(s, a)$



Policy Gradient Method

- REINFOCE
  - the REward Increment = Nonnegative Factor Offset ReinforCEment

$$w \leftarrow w + \alpha(r - b)\frac{\partial \pi(a|s)}{\partial w}$$

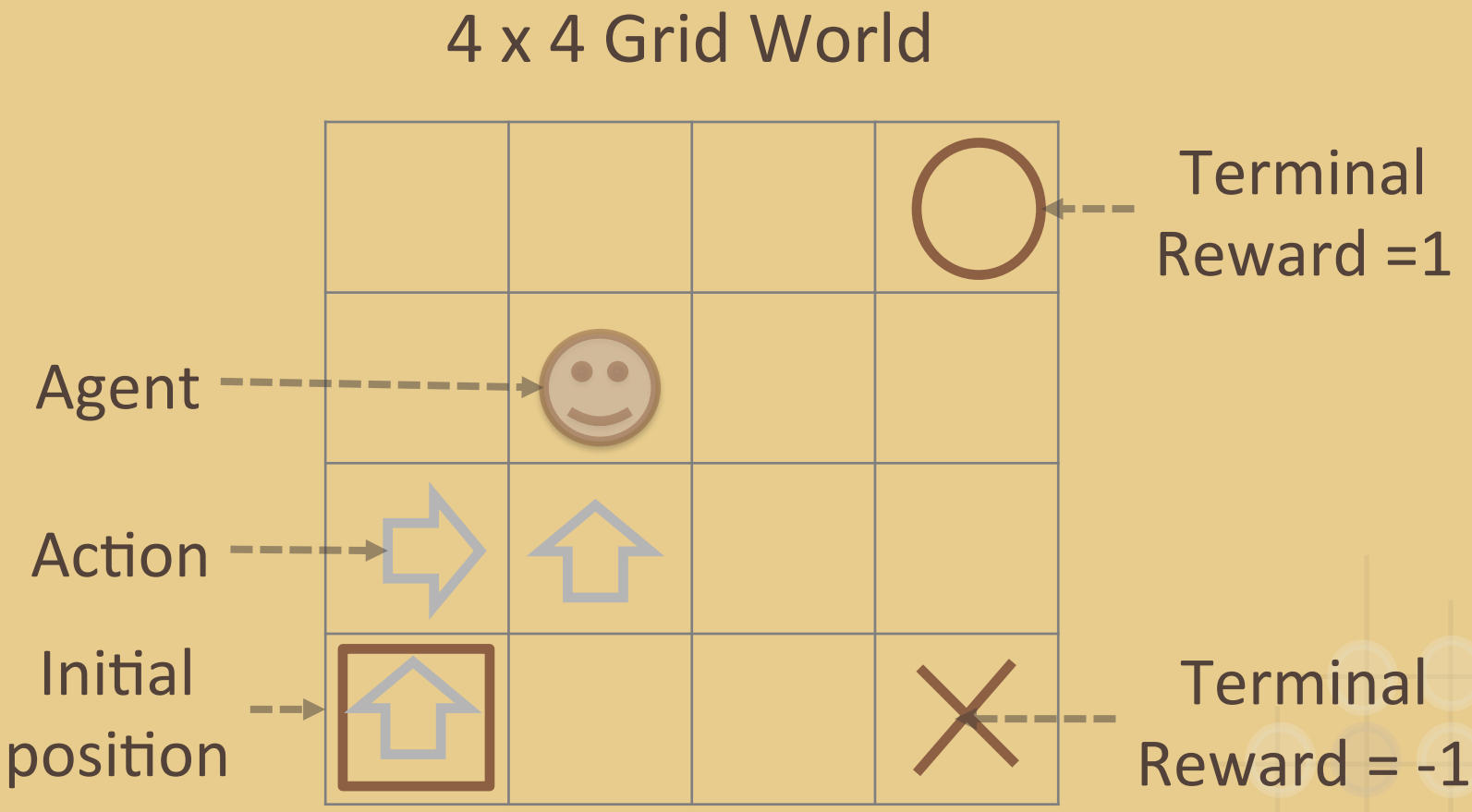
weights in policy function

learning rate

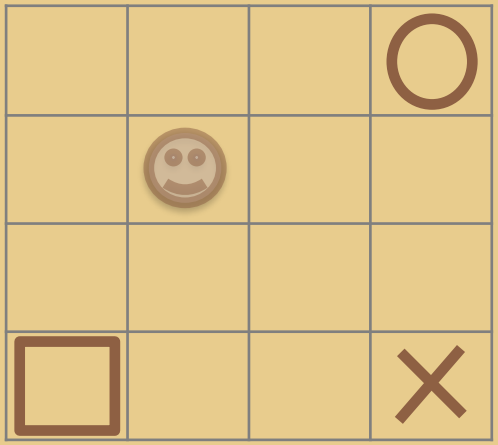
reward

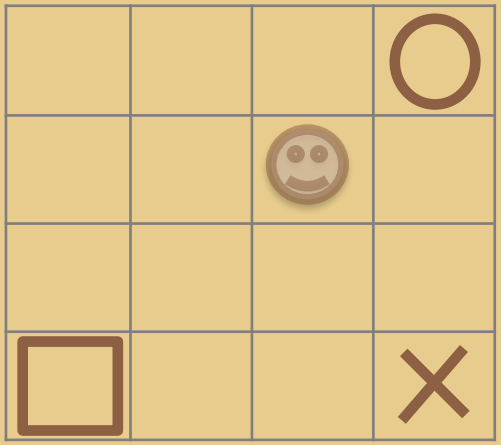
baseline (usually = 0)

Grid World Example



Policy Networks





Execute action

One-hot encoding

Sampling

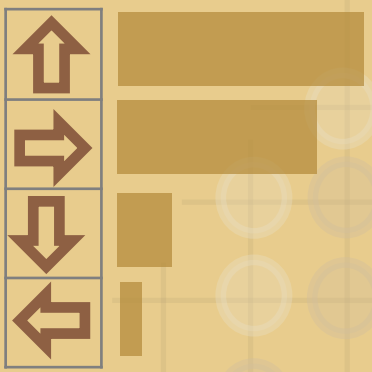
Probabilities of actions

0	0	0	0
0	1	0	0
0	0	0	0
0	0	0	0

s

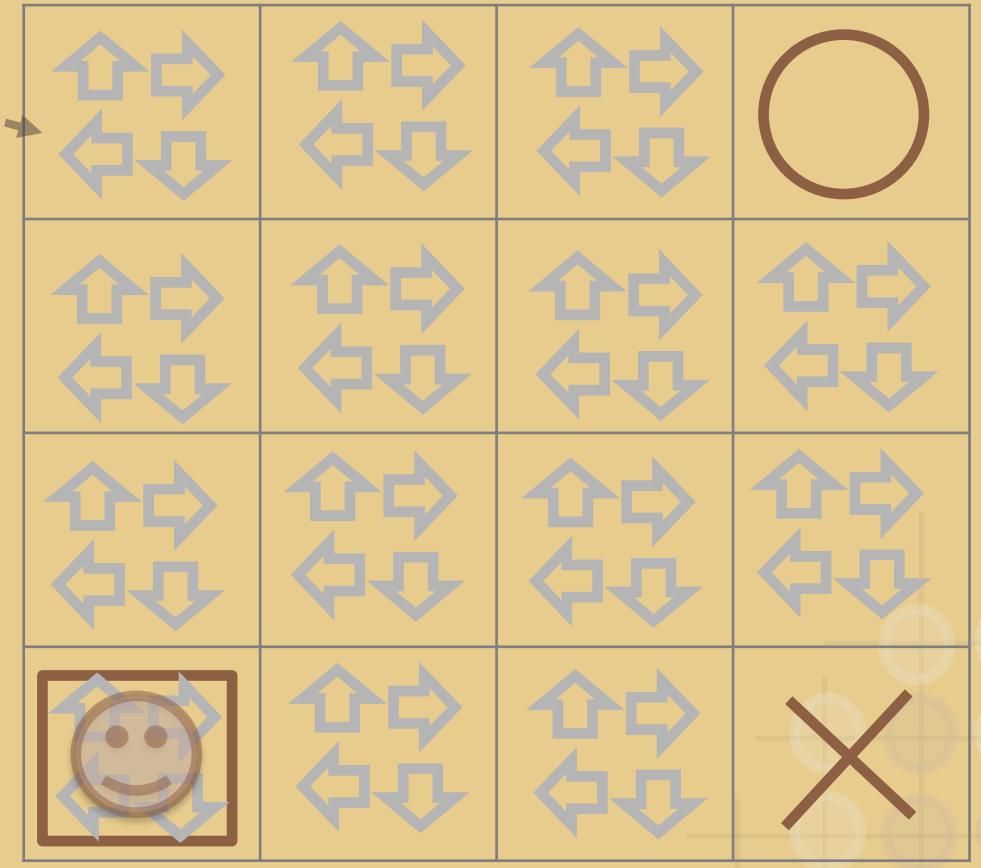
Neural Networks

$\pi(a|s)$



Initialization

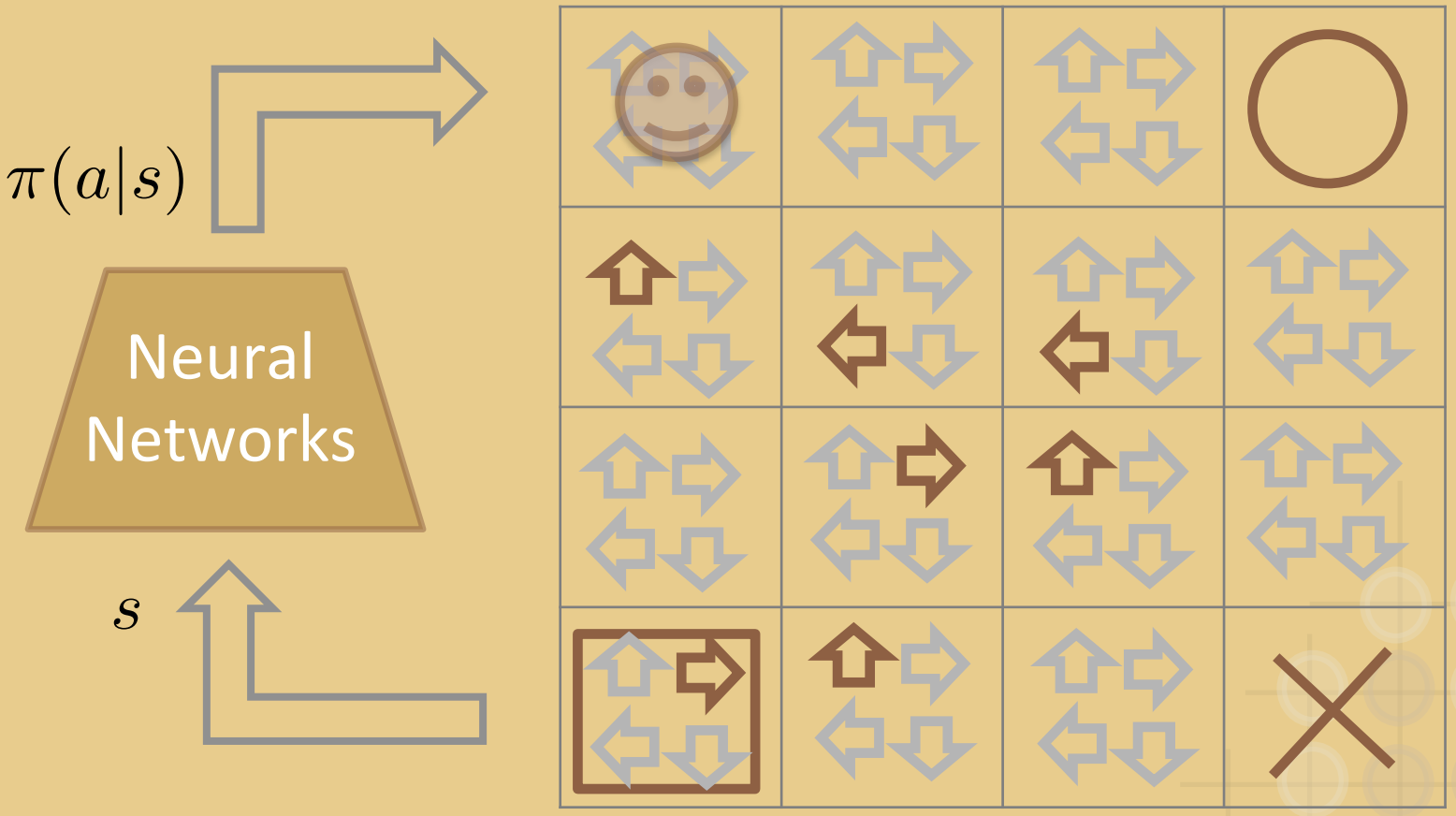
$\pi(a|s)$ for every s



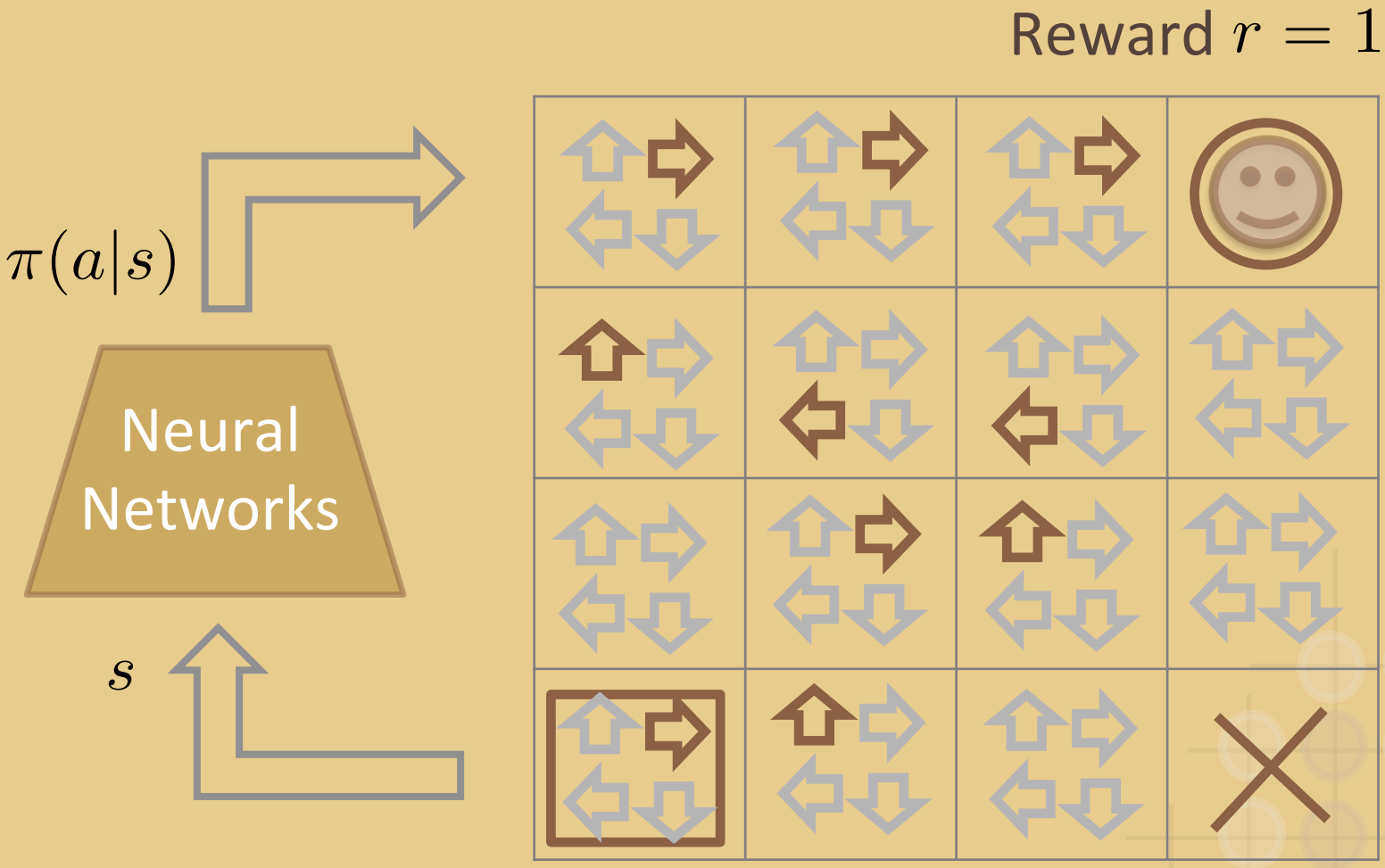
Neural Networks

Initialized with random weights

Forward Propagation



Forward Propagation



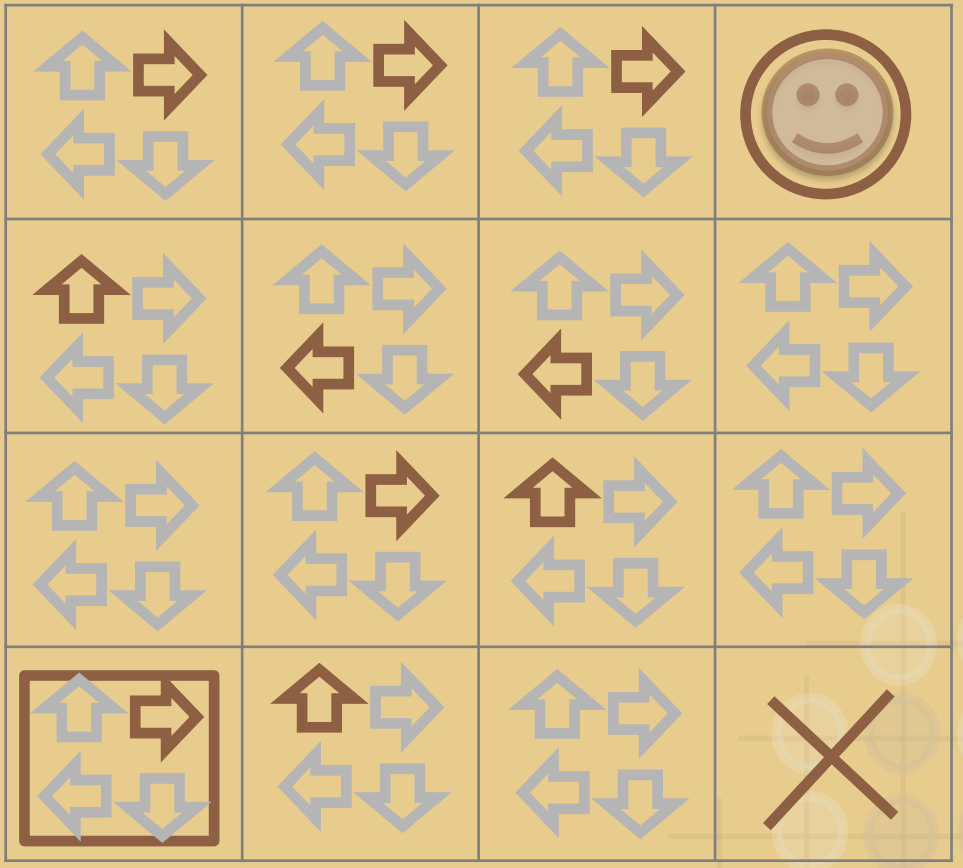
Backward Propagation

Reward $r = 1$

$$r \frac{\partial \pi(a|s)}{\partial w}$$

Neural Networks

$$w \leftarrow w + (r - b) \frac{\partial \pi(a|s)}{\partial w}$$



Backward Propagation

Reward $r = 1$

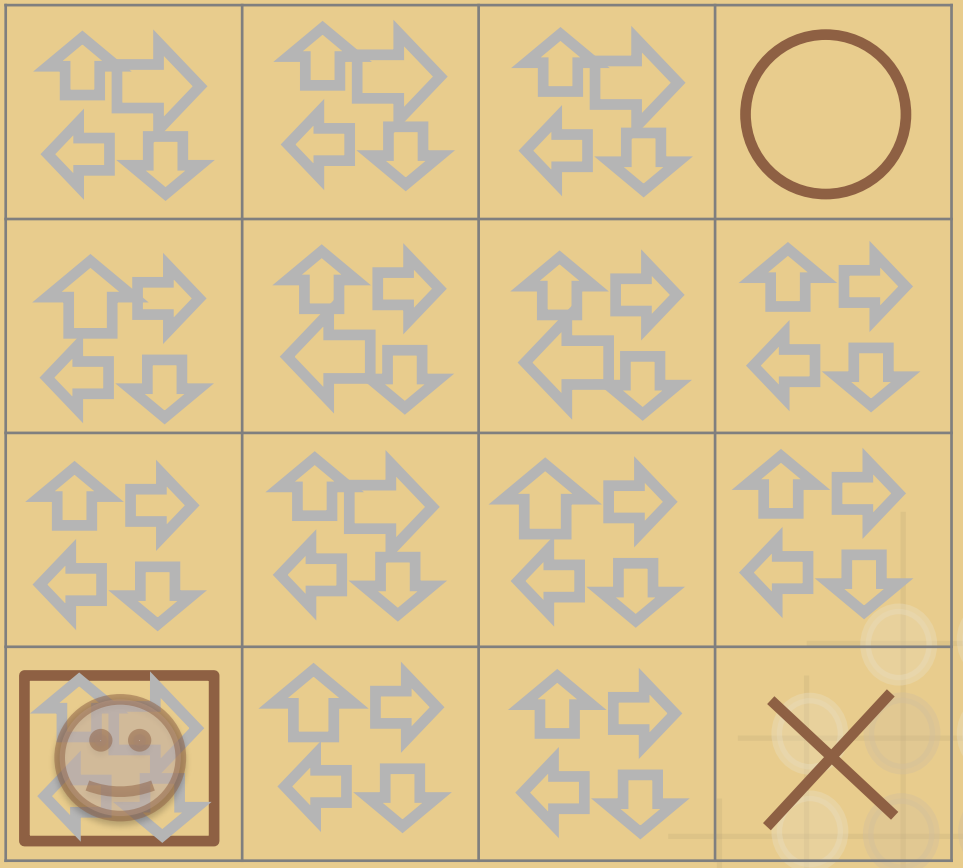
$$r \frac{\partial \pi(a|s)}{\partial w}$$

Neural Networks

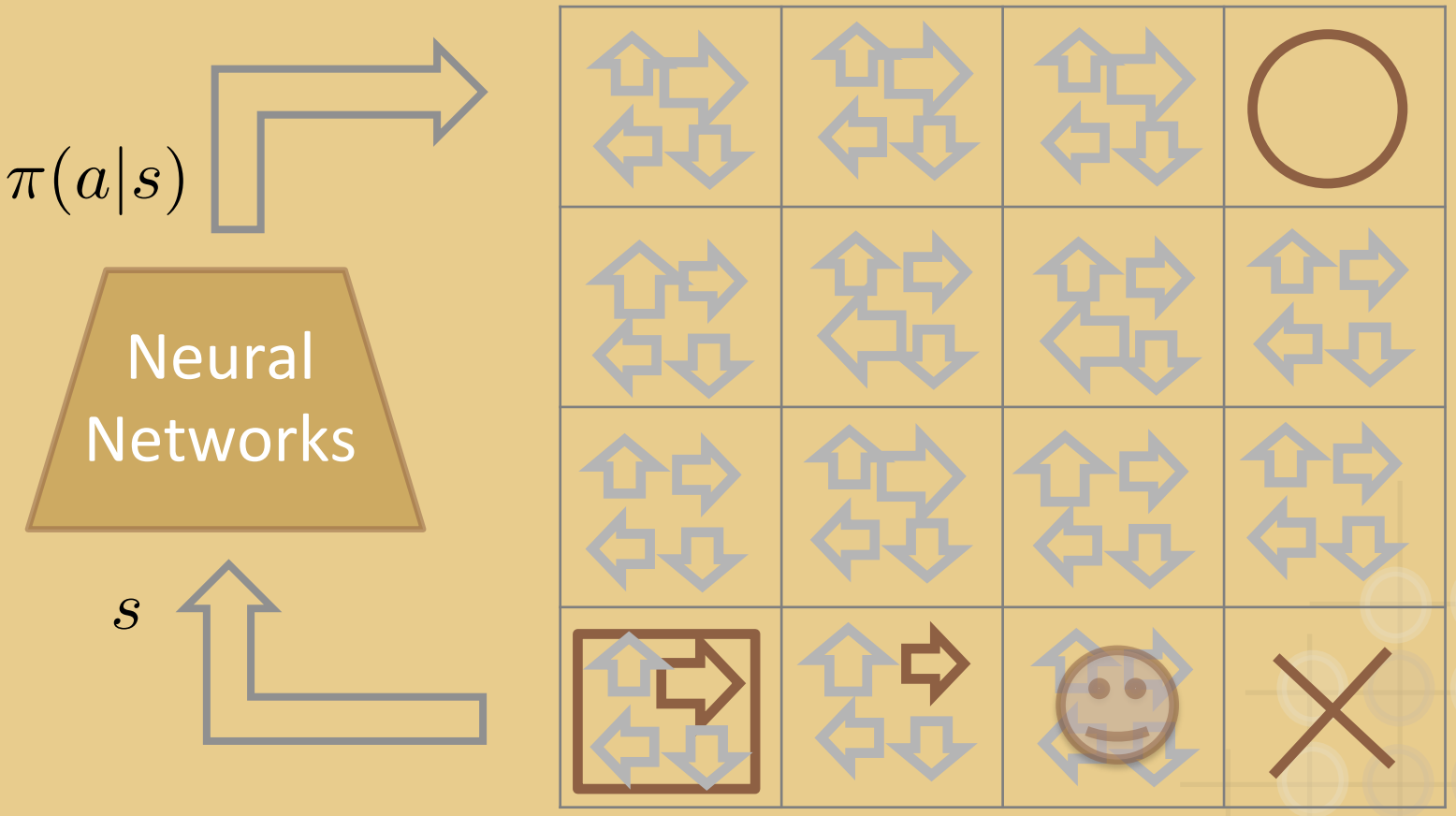
$$w \leftarrow w + (r - b) \frac{\partial \pi(a|s)}{\partial w}$$

Next Iteration

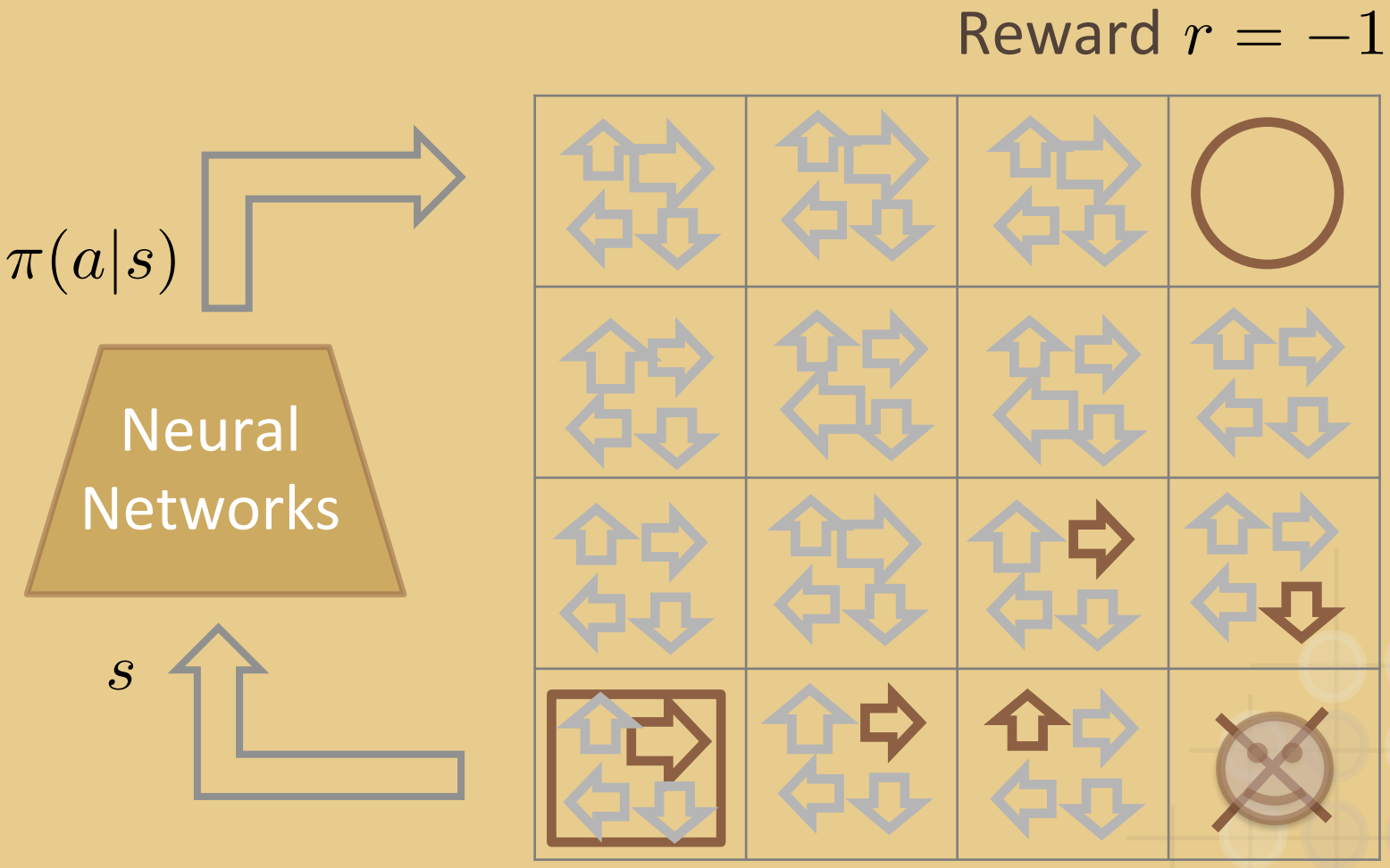
Neural Networks



Forward Propagation



Forward Propagation



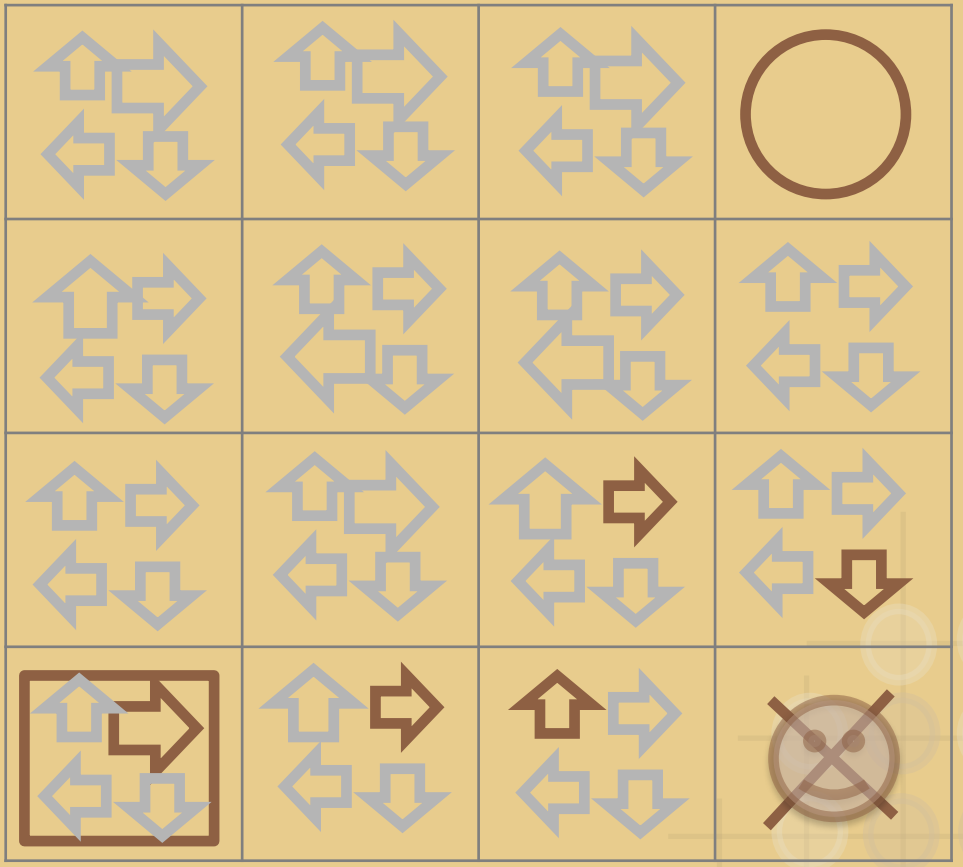
Backward Propagation

$$r \frac{\partial \pi(a|s)}{\partial w}$$

Neural Networks

$$w \leftarrow w + (r - b) \frac{\partial \pi(a|s)}{\partial w}$$

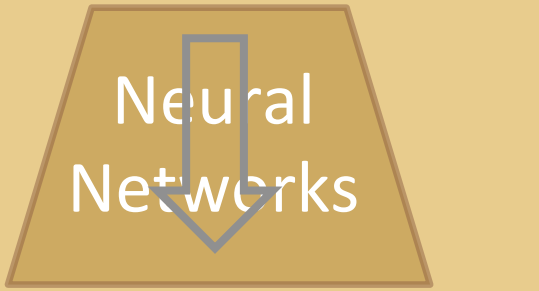
Reward $r = -1$



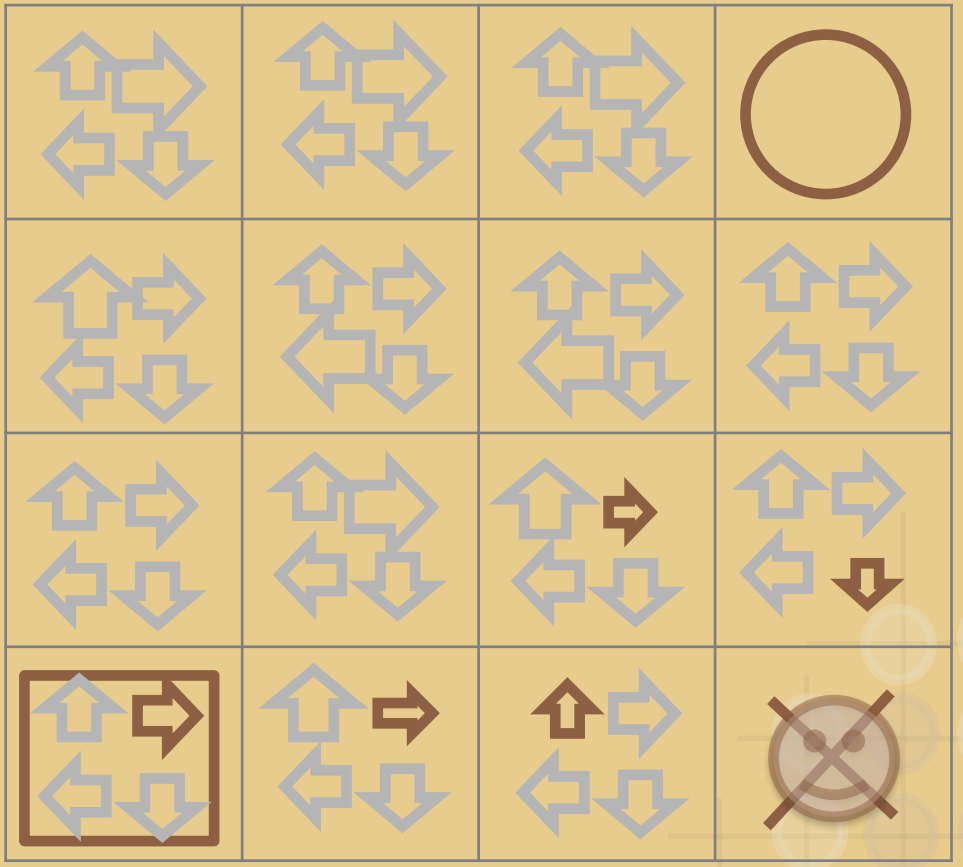
Backward Propagation

Reward $r = -1$

$$r \frac{\partial \pi(a|s)}{\partial w}$$

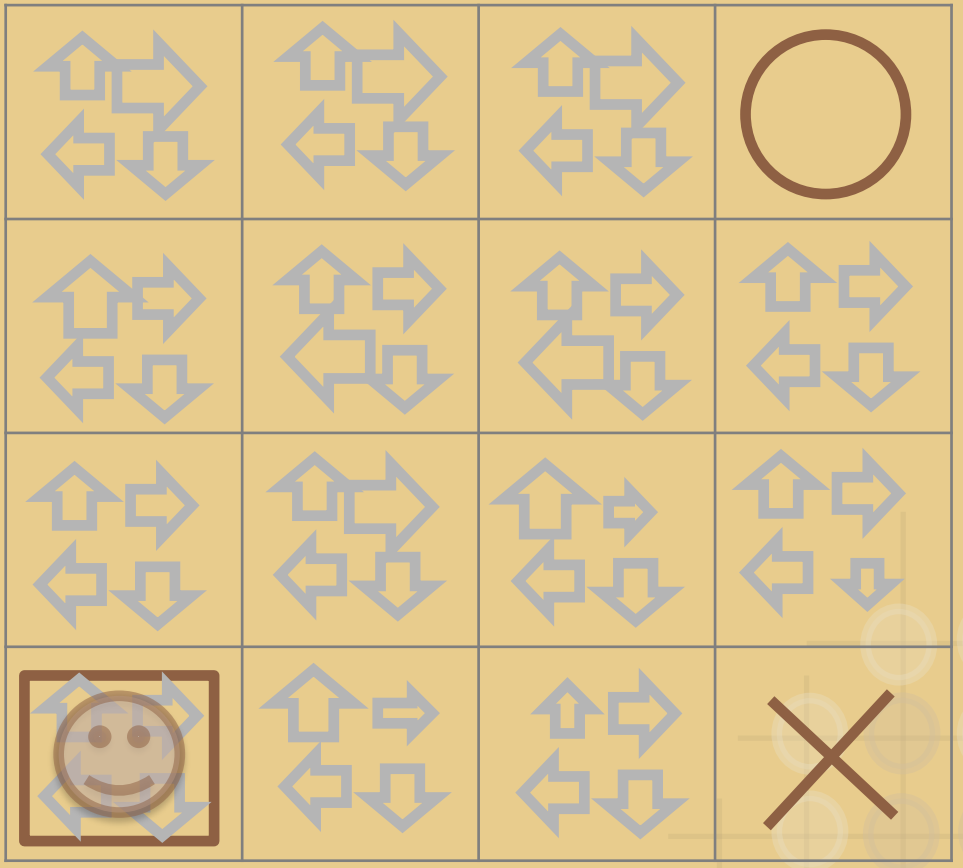


$$w \leftarrow w + (r - b) \frac{\partial \pi(a|s)}{\partial w}$$

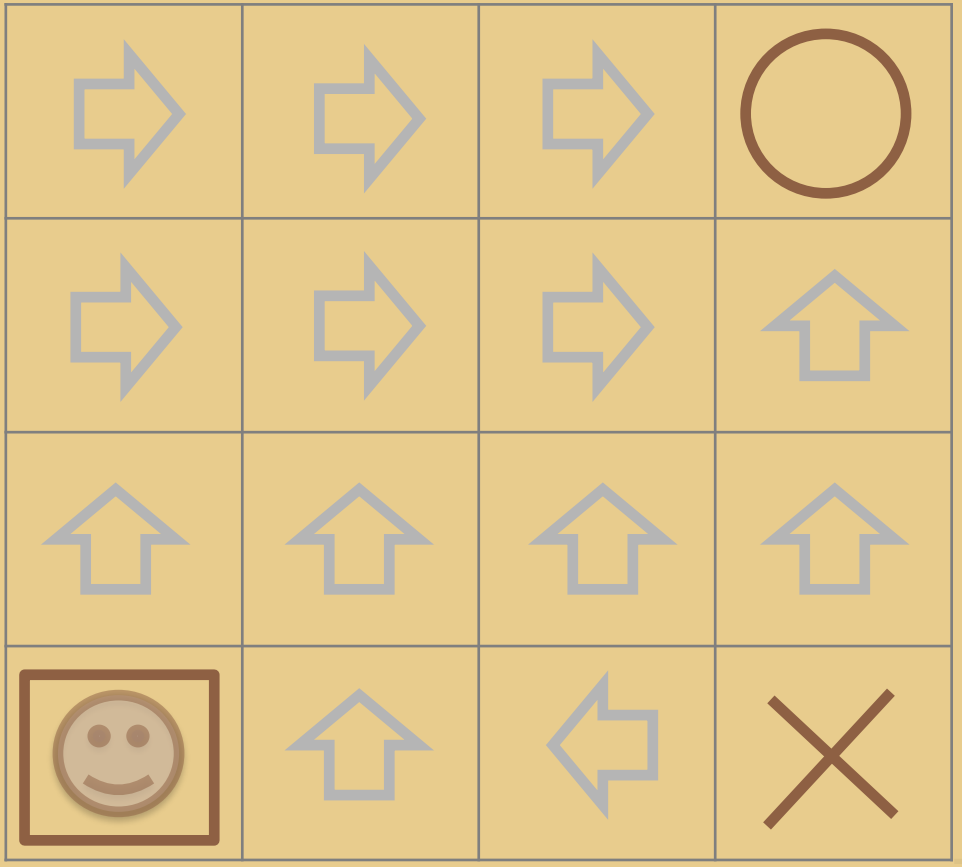


Next Iteration …

Neural Networks



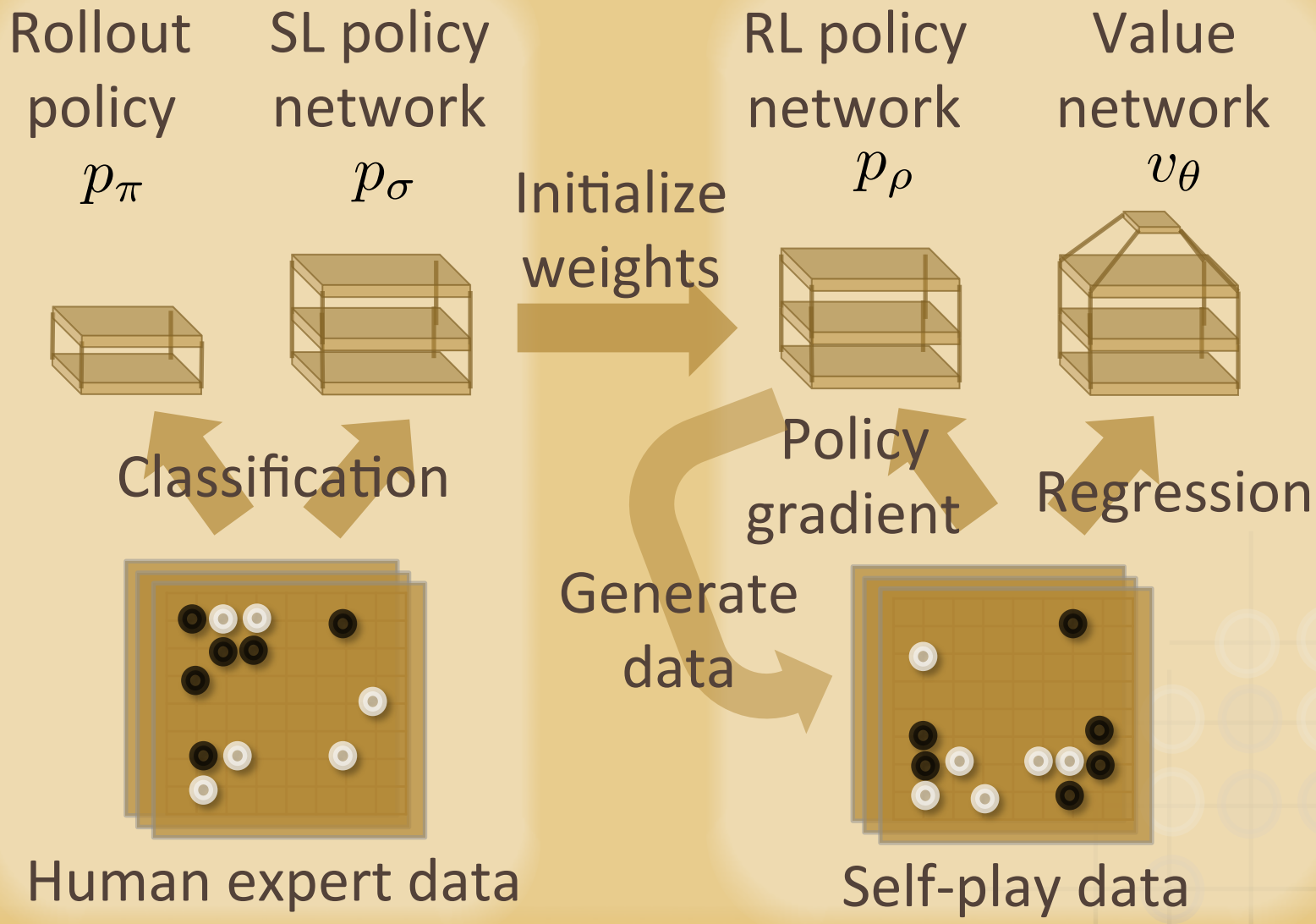
After Several Iterations …



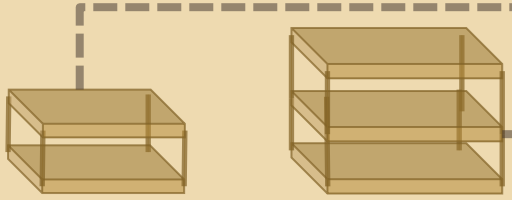
AlphaGo's Methods

- Training:
  - Supervised learning : Classification
  - Reinforcement learning
  - Supervised learning : Regression
- Searching:
  - Searching with policy and value networks
  - Distributed search

Training


Rollout policy p_π
SL policy network p_σ
Initialize weights
RL policy network p_ρ
Value network v_θ
Classification
Policy gradient
Regression
Generate data
Human expert data
Self-play data

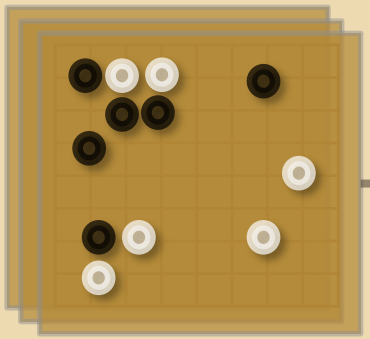
Supervised Learning : Classification



Rollout policy p_π

SL policy network p_σ

Classification



Human expert data

13-layers convolutional neural network

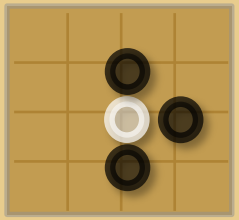
linear-softmax network (faster but less accurate)

50 GPUs, 3 weeks
Accuracy : 57.0%

KGS dataset
160,000 games
29.4 million positions

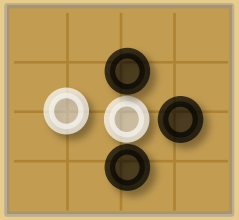
Input/Output Data

Input



Stone color, Liberty, Turns since, Capture size, Self-atari size, Ladder capture, Ladder escape, Sensibleness.
Total: 48 planes

Output



Next position

0	0	0
1	0	0
0	0	0

Stone color:
3 planes
player, opponent, empty

0	0	0
0	1	0
0	0	0

0	1	0
0	0	1
0	1	0

1	0	1
1	0	0
1	0	1

Liberty:
8 planes
1~8 liberties

0	0	0
0	1	0
0	0	0

0	0	0
0	0	0
0	0	0

0	1	0
0	0	1
0	1	0

0	0	0
0	0	0
0	0	0

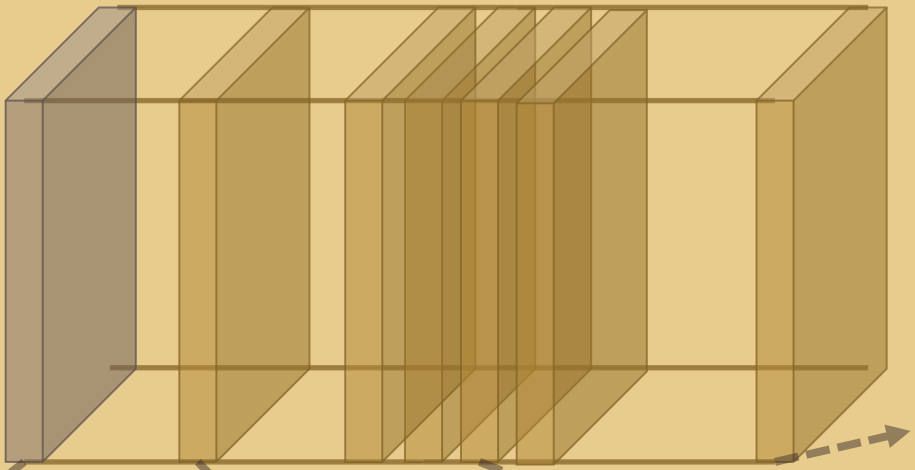
0	0	0
0	0	0
0	0	0

0	0	0
0	0	0
0	0	0

0	0	0
0	0	0
0	0	0

0	0	0
0	0	0
0	0	0

SL Policy Network



Input
Size: 19x19
48 planes

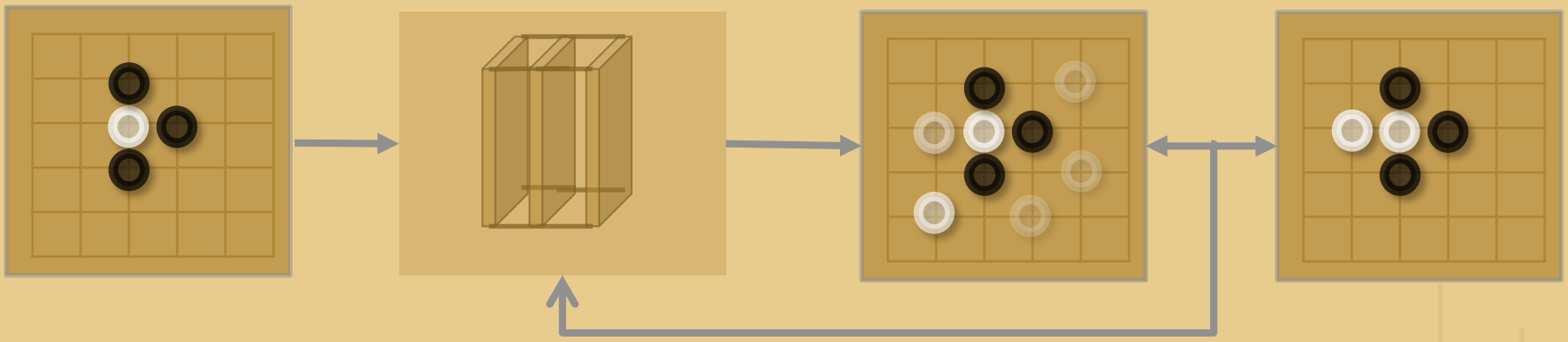
First layer
Kernel size: 5x5
k filters

2^{nd} to 12^{th} layers
Convolutional,
Kernel size: 3x3
k filters

13^{th} layer
Kernel size: 1x1
1 filters
Softmax

k = 192

Supervised Learning : Classification



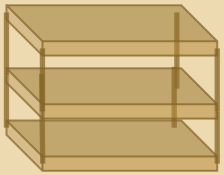
Backward propagation

$$\sigma \leftarrow \sigma - \eta \frac{\partial - \log p_\sigma(a|s)}{\partial \sigma}$$

Learning rate

Reinforcement Learning

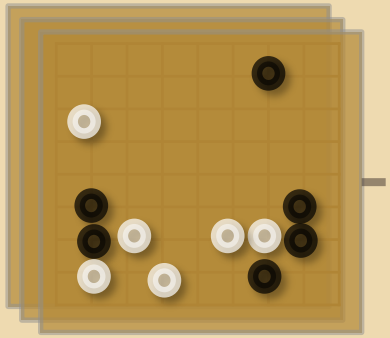
RL policy network p_ρ



Weights initialized by SL policy network

Policy gradient

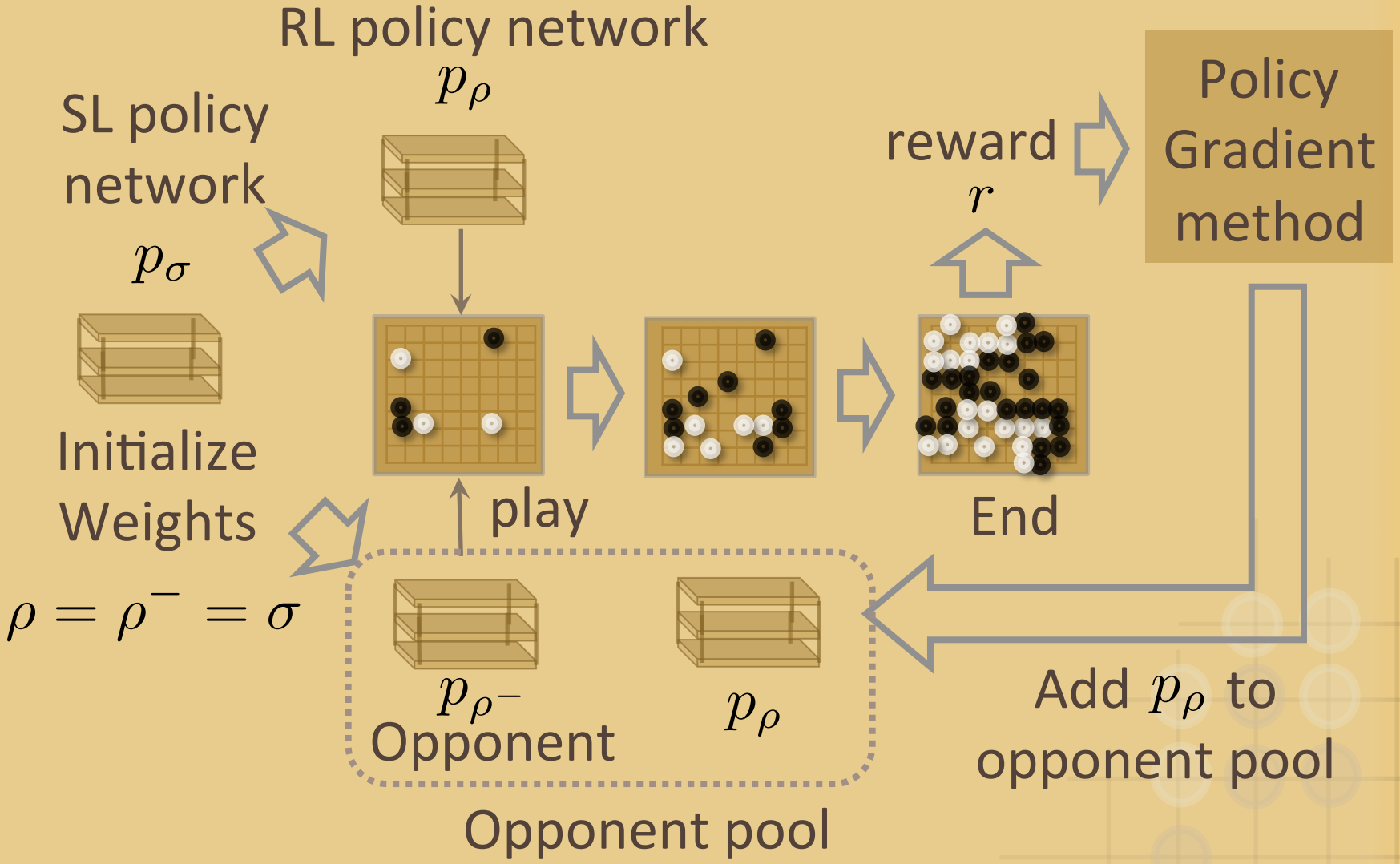
50 GPUs, 1 day
won 80% SL network



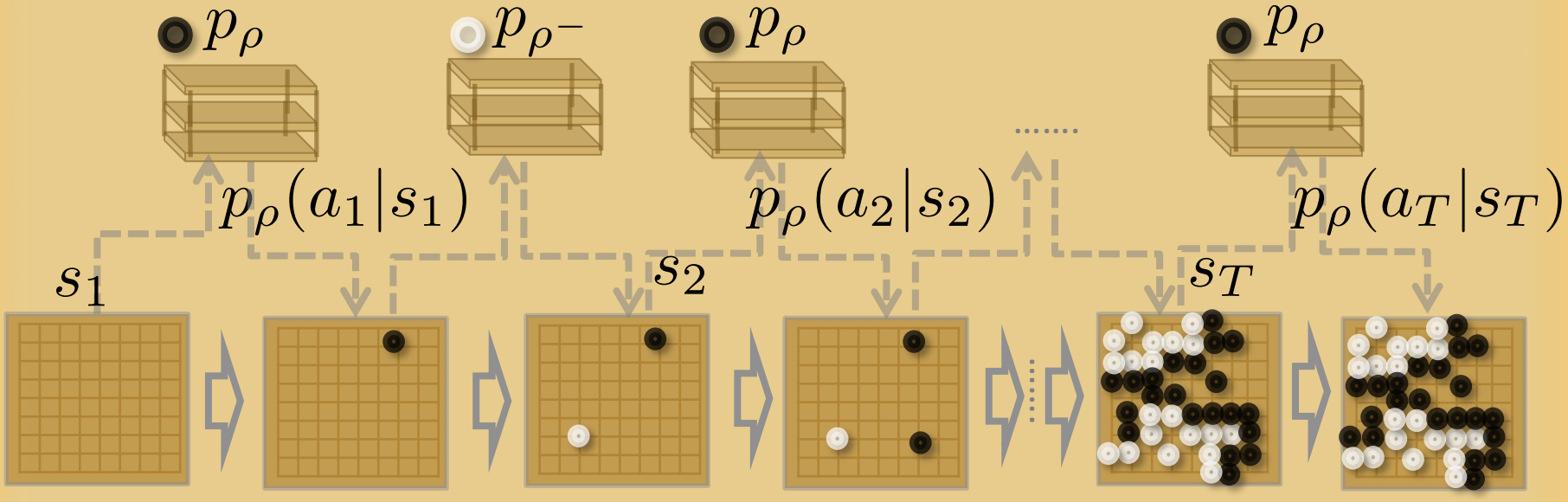
10,000 x 128 games

Self-play data

Reinforcement Learning



Policy Gradient Method



Backward propagation

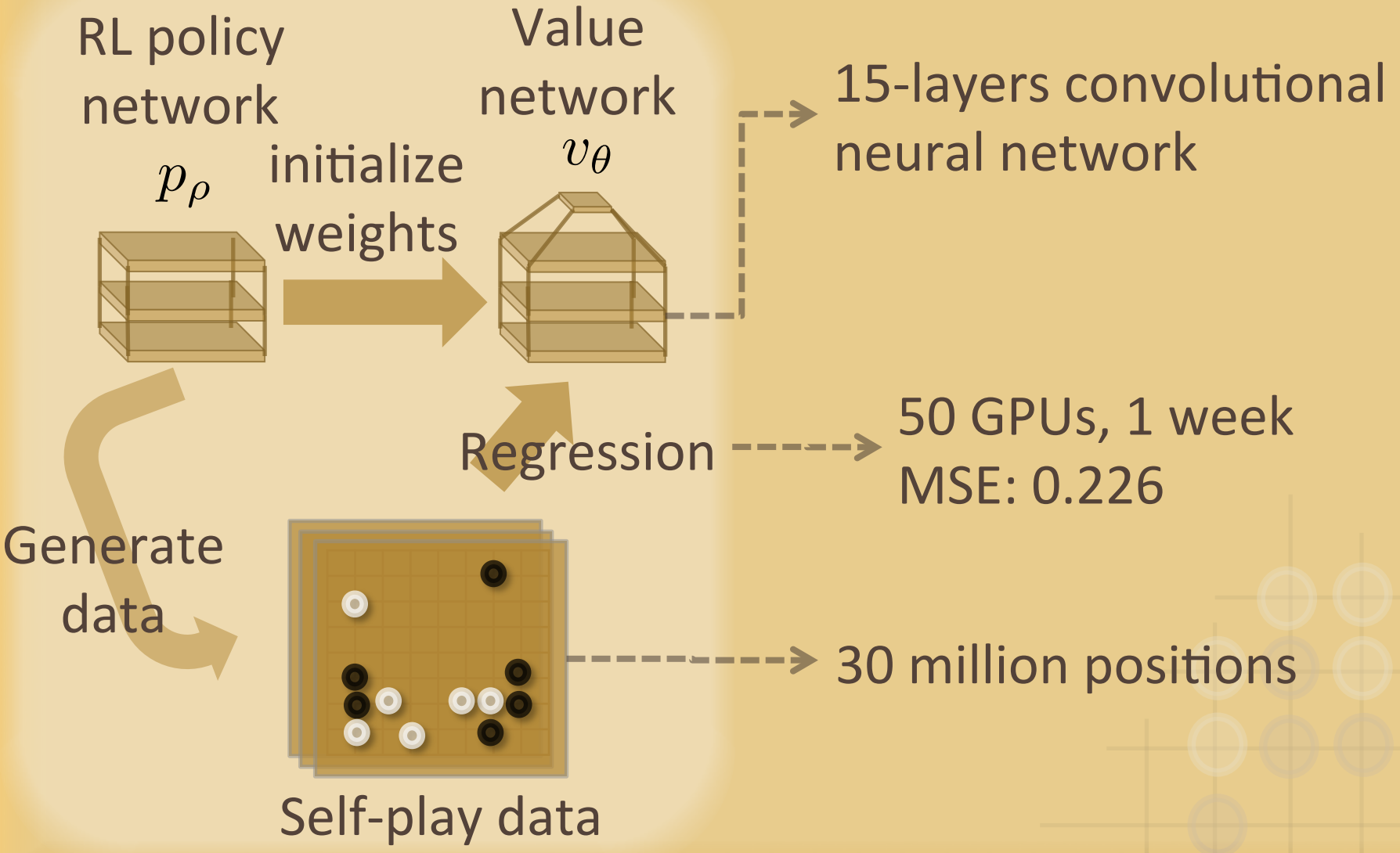
Reward $r(s_T)$

$$\rho \to \rho + \alpha \sum_{i=1}^{T} \frac{\partial p_\rho(a_i|s_i)}{\partial \rho} (r(s_T) - b(s_t))$$

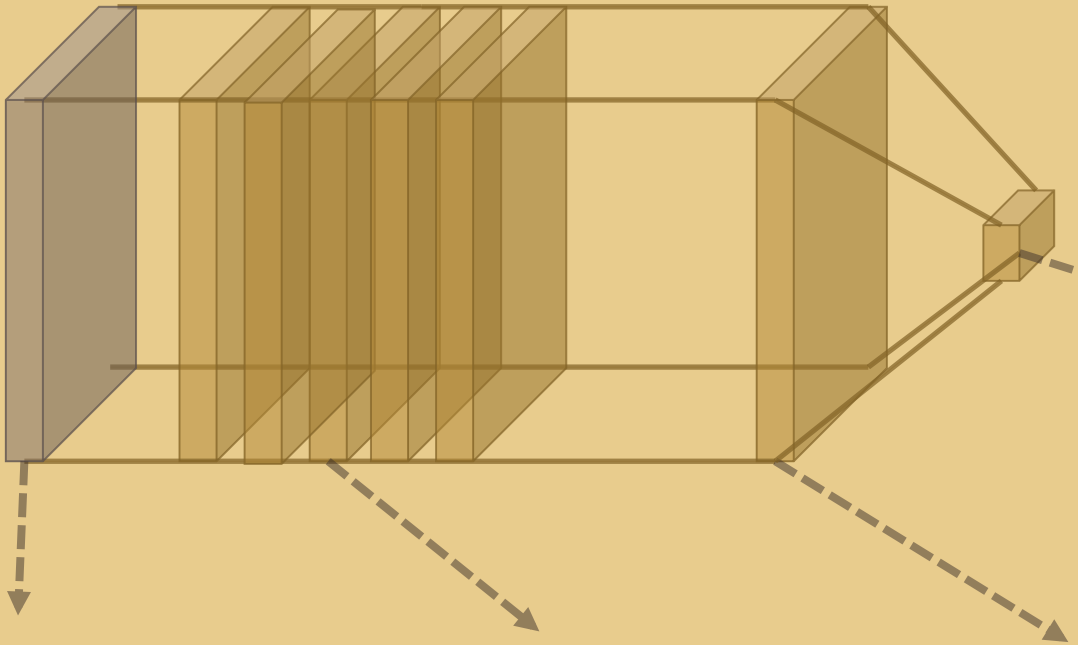
Learning rate

Baseline

Supervised learning : Regression



Value Network



Input
Size: 19x19
48 planes
+1 unit
(current color)

1^{st} ~ 13^{th}
The same as
Policy networks

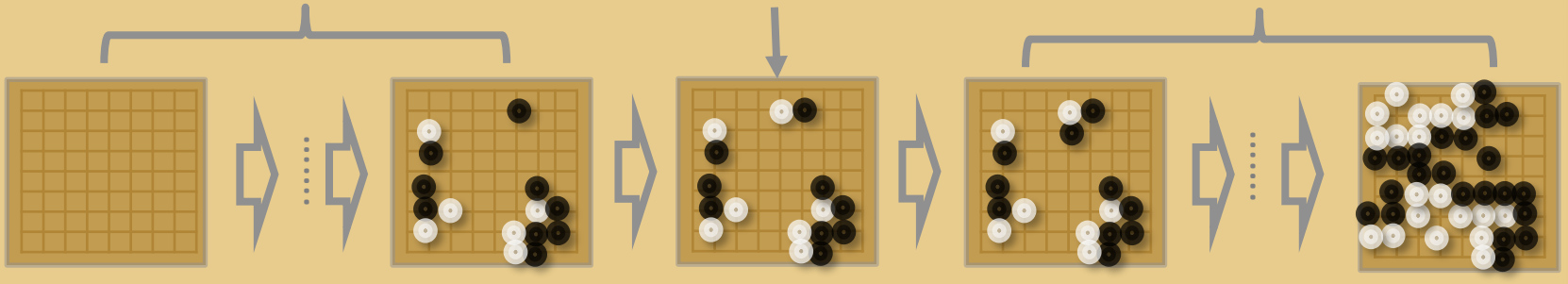
14^{th} layer
Fully-connected
256 ReLU unit

15^{th} layer
Fully-connected
1 tanh unit

Input/Output Data

1. Randomly sample an integer U in 1~450

2. Played by SL policy network p_σ | Random action | Played by RL policy network p_ρ

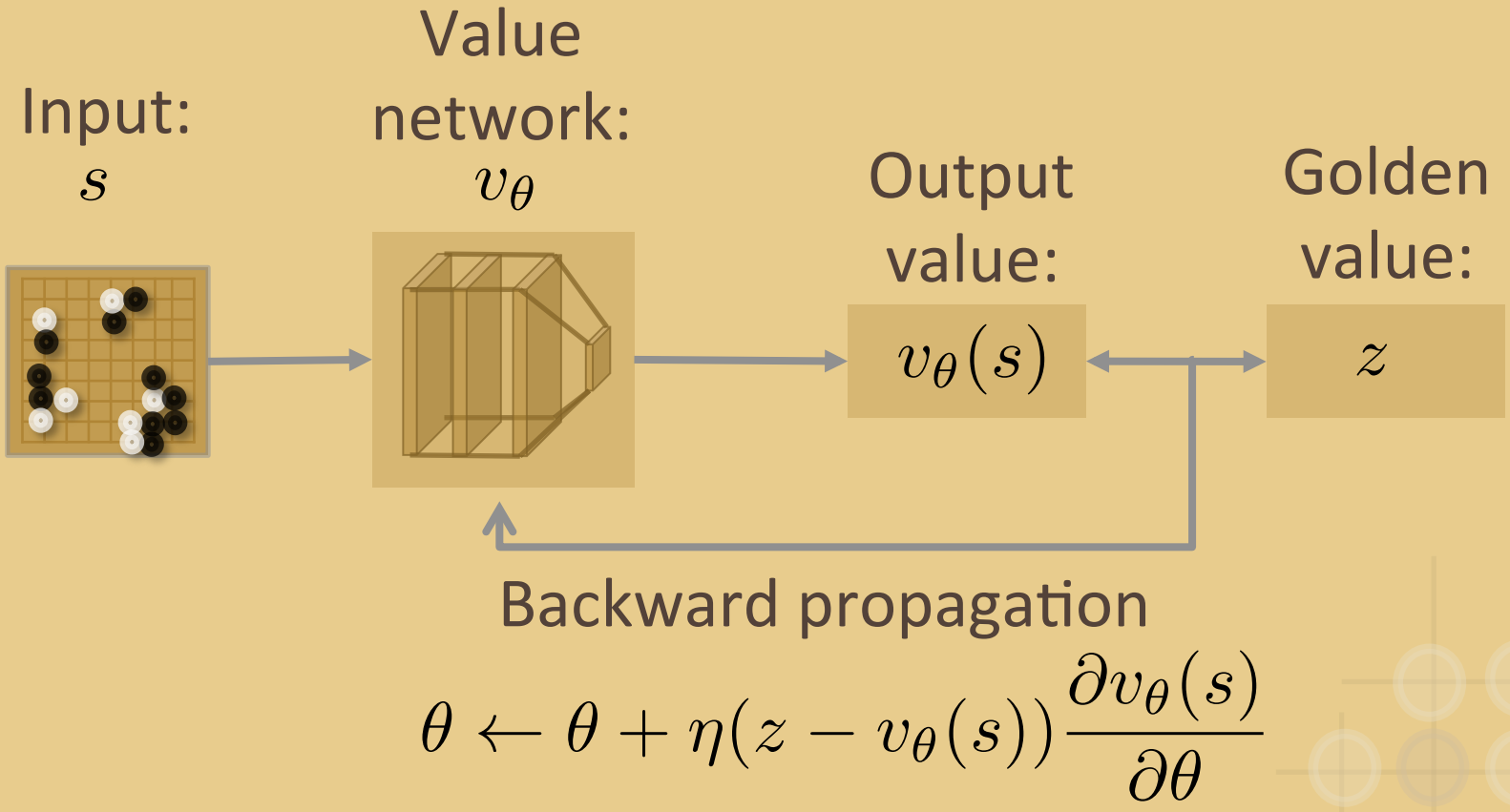


state: s_{U+1} reward: r

value: z_{U+1}

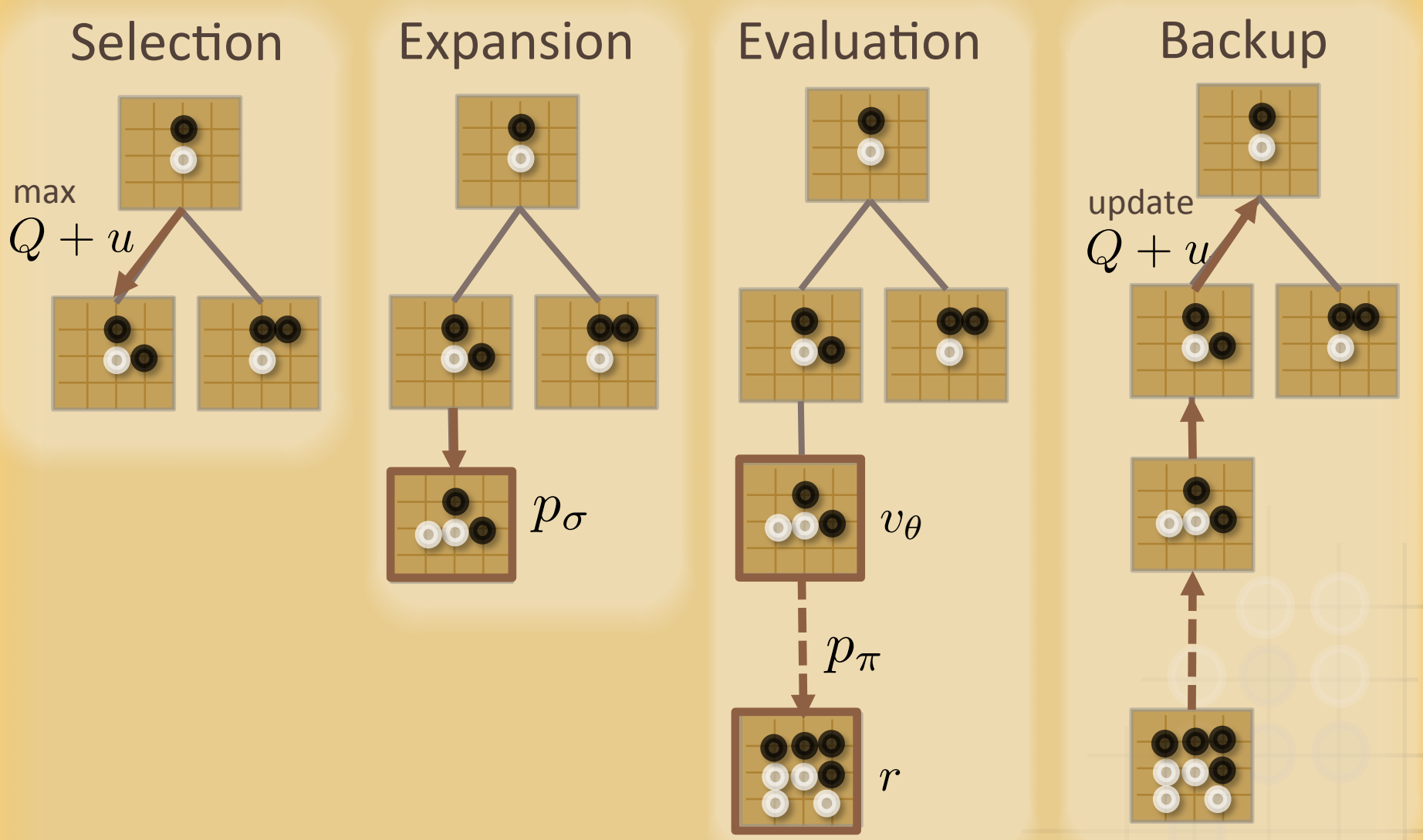
3. Generate Training Example: (s_{U+1}, z_{U+1})

Supervised learning : Regression



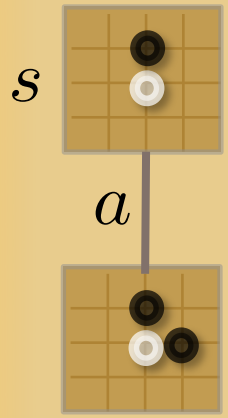
$$\theta \leftarrow \theta + \eta(z - v_\theta(s)) \frac{\partial v_\theta(s)}{\partial \theta}$$

Searching


Selection
max
$Q + u$
Expansion
p_σ
Evaluation
v_θ
p_π
r
Backup
update
$Q + u$

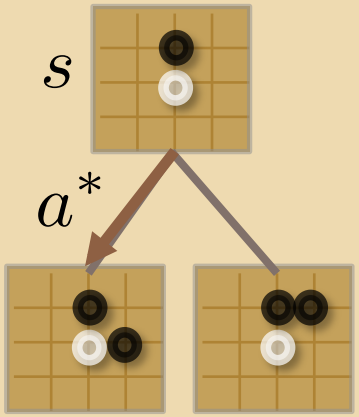
Searching

- Each edge stores a set of statistics



- $Q(s,a)$: combined mean action value
- $P(s,a)$: prior probability evaluated by $p_\sigma(a|s)$
- $W_r(s,a)$: estimated action value by $p_\pi(a|s)$
- $W_v(s,a)$: estimated action value by $v_\theta(s)$
- $N_r(s,a)$: counts of evaluations by $p_\pi(a|s)$
- $N_v(s,a)$: counts of evaluations by $v_\theta(s)$

Selection



Choose action

$$a^* = \underset{a}{\operatorname{argmax}}(Q(s,a) + u(s,a))$$

Exploitation

Exploration

PUCT Algorithm:

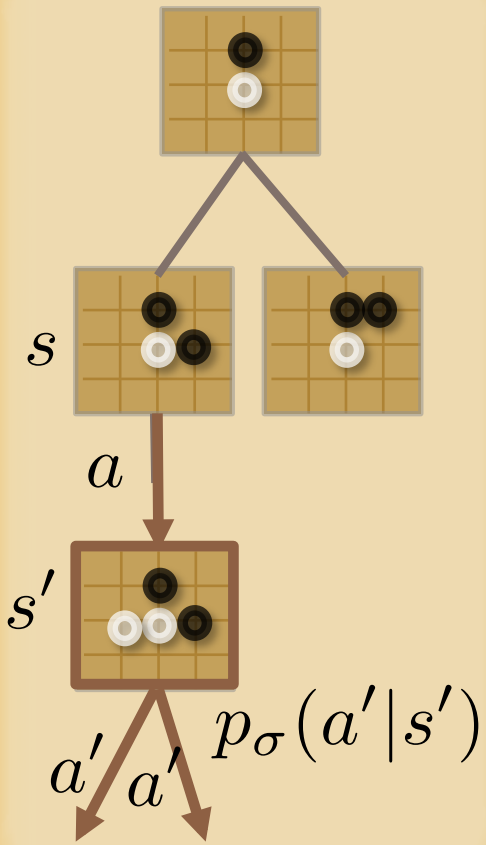
$$u(s,a) = cP(s,a)\frac{\sqrt{\sum_b N_r(s,b)}}{1 + N_r(s,a)}$$

Visit counts of parent node s

Visit counts of edge (s,a)

Level of exploration

Expansion



1 If visit count exceed a threshold : $N_r(s', a') > n_{thr}$,

Insert the node for the successor state s'.

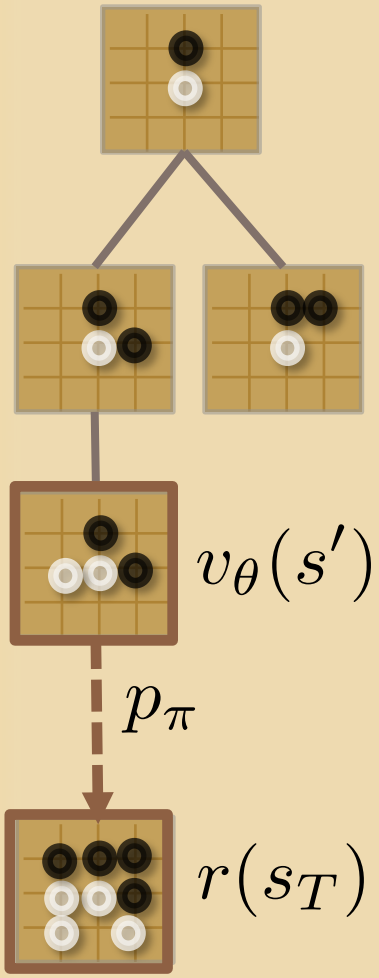
2 For every possible a' , initialize the statistics:

$$N_v(s', a') = N_r(s', a') = 0$$

$$W_r(s', a') = W_v(s', a') = 0$$

$$P(s', a') = p_\sigma(a'|s')$$

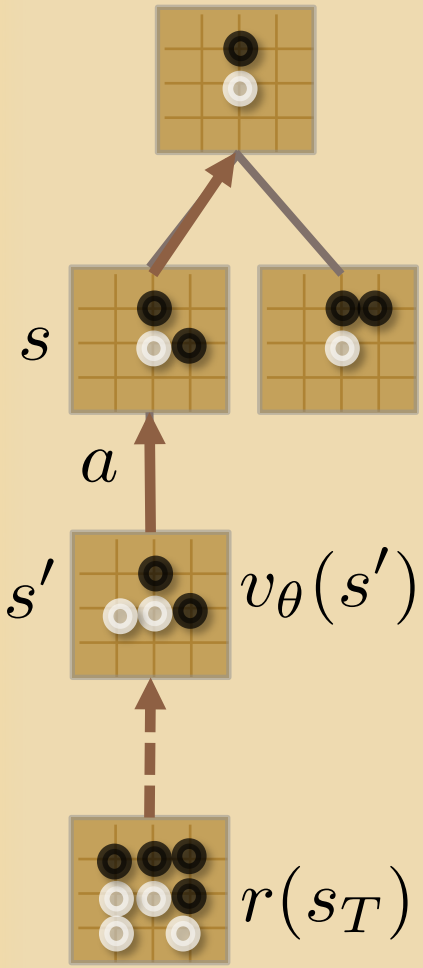
Evaluation



1. Evaluate $v_\theta(s')$ by value network v_θ .

2. Simulate the action by rollout policy network p_π .
   When reaching terminal s_T , calculate the reward $r(s_T)$.

Backup



Update the statistics of every visited edge (s, a) :

$$N_r(s,a) \leftarrow N_r(s,a) + 1$$

$$W_r(s,a) \leftarrow W_r(s,a) + r(s_T)$$

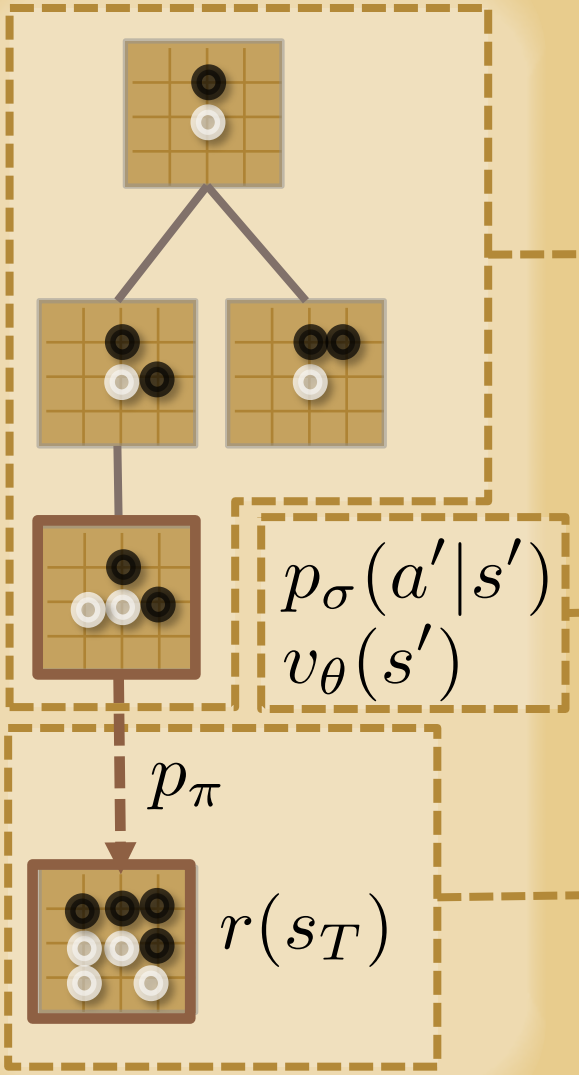
$$N_v(s,a) \leftarrow N_v(s,a) + 1$$

$$W_v(s,a) \leftarrow W_v(s,a) + v_\theta(s')$$

$$Q(s,a) = (1-\lambda)\frac{W_v(s,a)}{N_v(s,a)} + \lambda\frac{W_r(s,a)}{N_r(s,a)}$$

Interpolation constant

Distributed Search



Main search tree
Master CPU

Policy & value networks
176 GPUs

Rollout policy networks
1,202 CPUs

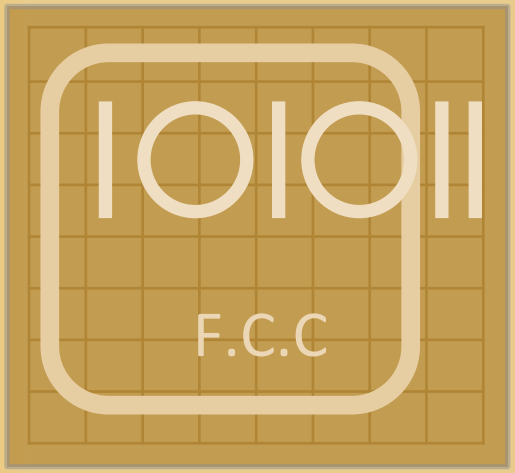
Reference

- Mastering the game of Go with deep neural networks and tree search
  - http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html

Further Reading

- Monte Carlo Tree Search
  - https://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/
- Neural Networks Backward Propagation
  - http://cpmarkchang.logdown.com/posts/277349-neural-network-backward-propagation
- Convolutional Neural Networks
  - http://cs231n.github.io/convolutional-networks/
- Policy Gradient Method: REINFORCE
  - https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a-html/node37.html

About the Speaker



Mark Chang

- Email: ckmarkoh at gmail dot com
- Blog: http://cpmarkchang.logdown.com
- Github: https://github.com/ckmarkoh
- Facebook: https://www.facebook.com/ckmarkoh.chang
- Slideshare: http://www.slideshare.net/ckmarkohchang
- Linkedin: https://www.linkedin.com/pub/mark-chang/85/25b/847