DSC291: Machine Learning with Few Labels

Reinforcement learning

Zhiting Hu

Lecture 17, February 17, 2023

UC San Diego
HALICIOĞLU DATA SCIENCE INSTITUTE

Recap: Solving for the optimal policy: Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s',a') | s,a \right]$$

Recap: Solving for the optimal policy: Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

Forward Pass

Loss function: $L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[(y_i - Q(s,a;\theta_i))^2 \right]$

where $y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$

Recap: Solving for the optimal policy: Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') | s,a\right]$$

Forward Pass

Loss function: $L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right]$

where $y_i = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) | s,a\right]$

Backward Pass

Gradient update (with respect to Q-function parameters θ):

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i))\nabla_{\theta_i} Q(s,a;\theta_i)\right]$$

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Recap: Q-network Architecture

$Q(s, a; \theta)$: neural network with weights θ

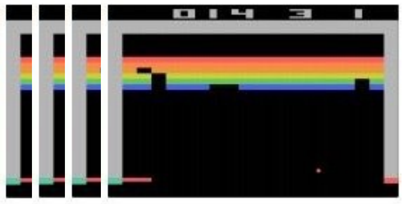
A single feedforward pass to compute Q-values for all actions from the current state => efficient!

FC-4 (Q-values)

FC-256

32 4x4 conv, stride 2

16 8x8 conv, stride 4



Last FC layer has 4-d output (if 4 actions), corresponding to $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$, $Q(s_t,a_4)$

Current state s_t: 84x84x4 stack of last 4 frames

(after RGB->grayscale conversion, downsampling, and cropping)

Recap: Training the Q-network: Experience Replay

Learning from batches of consecutive samples is problematic:

- Samples are correlated => inefficient learning
- Current Q-network parameters determines next training samples (e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size) => can lead to bad feedback loops

Address these problems using **experience replay**

- Continually update a **replay memory** table of transitions (s_t, a_t, r_t, s_{t+1}) as game (experience) episodes are played
- Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples

Recap: Training the Q-network: Experience Replay

Learning from batches of consecutive samples is problematic:

- Samples are correlated => inefficient learning
- Current Q-network parameters determines next training samples (e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size) => can lead to bad feedback loops

Address these problems using **experience replay**

- Continually update a **replay memory** table of transitions (s_t, a_t, r_t, s_{t+1}) as game (experience) episodes are played
- Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples

Each transition can also contribute to multiple weight updates
=> greater data efficiency

Recap: Deep Q-Learning with Experience Replay

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Other concepts

- Q-learning:
  - Value-based
    - learns Q-value function
  - Off-policy
    - E.g., replay memory

- Policy Gradient:
  - Policy-based
    - Learns policy itself
  - On-policy

Policy Gradients

What is a problem with Q-learning?
The Q-function can be very complicated!

Example: a robot grasping an object has a very high-dimensional state => hard to learn exact value of every (state, action) pair

Policy Gradients

What is a problem with Q-learning?
The Q-function can be very complicated!

Example: a robot grasping an object has a very high-dimensional state => hard to learn exact value of every (state, action) pair

But the policy can be much simpler: just close your hand
Can we learn a policy directly, e.g. finding the best policy from a collection of policies?

Policy Gradients

Formally, let's define a class of parametrized policies: $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value:

$$J(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi_\theta\right]$$

Policy Gradients

Formally, let's define a class of parametrized policies: $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value:

$$J(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi_\theta\right]$$

We want to find the optimal policy $\theta^* = \arg\max_\theta J(\theta)$

How can we do this?

Policy Gradients

Formally, let's define a class of parametrized policies: $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value:

$$J(\theta) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | \pi_\theta\right]$$

We want to find the optimal policy $\theta^* = \arg\max_\theta J(\theta)$

How can we do this?

Gradient ascent on policy parameters!

REINFORCE algorithm

Mathematically, we can write:

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)} \left[r(\tau) \right]$$
$$= \int_{\tau} r(\tau) p(\tau;\theta) \mathrm{d}\tau$$

Where $r(\tau)$ is the reward of a trajectory $\tau = (s_0, a_0, r_0, s_1, \ldots)$

REINFORCE algorithm

Expected reward:

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)} [r(\tau)]$$
$$= \int_{\tau} r(\tau) p(\tau;\theta) \mathrm{d}\tau$$

REINFORCE algorithm

Expected reward:

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right]$$
$$= \int_{\tau} r(\tau) p(\tau;\theta) \mathrm{d}\tau$$

Now let’s differentiate this:

$$\nabla_\theta J(\theta) = \int_{\tau} r(\tau) \nabla_\theta p(\tau;\theta) \mathrm{d}\tau$$

REINFORCE algorithm

Expected reward:
$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right]$$
$$= \int_\tau r(\tau) p(\tau;\theta) \mathrm{d}\tau$$

Now let's differentiate this:
$$\nabla_\theta J(\theta) = \int_\tau r(\tau) \nabla_\theta p(\tau;\theta) \mathrm{d}\tau$$

Intractable! Gradient of an expectation is problematic when p depends on θ

REINFORCE algorithm

Expected reward:
$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right]$$
$$= \int_{\tau} r(\tau)p(\tau;\theta)\mathrm{d}\tau$$

Now let's differentiate this:
$$\nabla_\theta J(\theta) = \int_{\tau} r(\tau)\nabla_\theta p(\tau;\theta)\mathrm{d}\tau$$

Intractable! Gradient of an expectation is problematic when p depends on θ

However, we can use a nice trick:
$$\nabla_\theta p(\tau;\theta) = p(\tau;\theta)\frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta)\nabla_\theta \log p(\tau;\theta)$$

REINFORCE algorithm

Expected reward:
$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right] = \int_\tau r(\tau)p(\tau;\theta)\mathrm{d}\tau$$

Now let's differentiate this:
$$\nabla_\theta J(\theta) = \int_\tau r(\tau)\nabla_\theta p(\tau;\theta)\mathrm{d}\tau$$

Intractable! Gradient of an expectation is problematic when p depends on θ

However, we can use a nice trick:
$$\nabla_\theta p(\tau;\theta) = p(\tau;\theta)\frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta)\nabla_\theta \log p(\tau;\theta)$$

If we inject this back:

$$\nabla_\theta J(\theta) = \int_\tau \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right)p(\tau;\theta)\mathrm{d}\tau = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right]$$

Can estimate with Monte Carlo sampling

REINFORCE algorithm

Can we compute those quantities without knowing the transition probabilities?

We have: $p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$

REINFORCE algorithm

Can we compute those quantities without knowing the transition probabilities?

We have: $p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$

Thus: $\log p(\tau;\theta) = \sum_{t\geq 0} \log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)$

REINFORCE algorithm

Can we compute those quantities without knowing the transition probabilities?

We have: $p(\tau;\theta) = \prod_{t \geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$

Thus: $\log p(\tau;\theta) = \sum_{t \geq 0} \log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)$

And when differentiating: $\nabla_\theta \log p(\tau;\theta) = \sum_{t \geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t)$

Doesn't depend on transition probabilities!

REINFORCE algorithm

$$\nabla_\theta J(\theta) = \int_\tau \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right) p(\tau;\theta)\mathrm{d}\tau$$
$$= \mathbb{E}_{\tau\sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right]$$

Can we compute those quantities without knowing the transition probabilities?

We have: $p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$

Thus: $\log p(\tau;\theta) = \sum_{t\geq 0} \log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)$

And when differentiating: $\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t)$

Doesn't depend on transition probabilities!

Therefore when sampling a trajectory τ, we can estimate J(θ) with

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} r(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)$$

Intuition

Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Interpretation:

- If $r(\tau)$ is high, push up the probabilities of the actions seen
- If $r(\tau)$ is low, push down the probabilities of the actions seen

Intuition

Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Interpretation:

- If r(τ) is high, push up the probabilities of the actions seen
- If r(τ) is low, push down the probabilities of the actions seen

Might seem simplistic to say that if a trajectory is good then all its actions were good. But in expectation, it averages out!

Intuition

Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Interpretation:

- If r(τ) is high, push up the probabilities of the actions seen
- If r(τ) is low, push down the probabilities of the actions seen

Might seem simplistic to say that if a trajectory is good then all its actions were good. But in expectation, it averages out!

However, this also suffers from high variance because credit assignment is really hard. Can we help the estimator?

Variance reduction

Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Variance reduction

Gradient estimator: $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$

First idea: Push up probabilities of an action seen, only by the cumulative future reward from that state

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Variance reduction

Gradient estimator: $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$

First idea: Push up probabilities of an action seen, only by the cumulative future reward from that state

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Second idea: Use discount factor γ to ignore delayed effects

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Variance reduction: Baseline

Problem: The raw value of a trajectory isn't necessarily meaningful. For example, if rewards are all positive, you keep pushing up probabilities of actions.

What is important then? Whether a reward is better or worse than what you expect to get

Idea: Introduce a baseline function dependent on the state.
Concretely, estimator is now:

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

How to choose the baseline?

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

A simple baseline: constant moving average of rewards experienced so far from all trajectories

How to choose the baseline?

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} \left(\sum_{t'\geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

A simple baseline: constant moving average of rewards experienced so far from all trajectories

Variance reduction techniques seen so far are typically used in “Vanilla REINFORCE”

How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Q: What does this remind you of?

How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Q: What does this remind you of?

A: Q-function and value function!

How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Q: What does this remind you of?

A: Q-function and value function!

Intuitively, we are happy with an action a_t in a state s_t if $Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$ is large. On the contrary, we are unhappy with an action if it's small.

How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Q: What does this remind you of?

A: Q-function and value function!

Intuitively, we are happy with an action a_t in a state s_t if $Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$ is large. On the contrary, we are unhappy with an action if it's small.

Using this, we get the estimator: $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} (Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t)$

Actor-Critic Algorithm

Problem: we don’t know Q and V. Can we learn them?

Yes, using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-function).

- The actor decides which action to take, and the critic tells the actor how good its action was and how it should adjust
- Also alleviates the task of the critic as it only has to learn the values of (state, action) pairs generated by the policy
- Can also incorporate Q-learning tricks e.g. experience replay
- **Remark:** we can define by the **advantage function** how much an action was better than expected

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

Actor-Critic Algorithm

Initialize policy parameters 8, critic parameters ø

For iteration=1, 2 … **do**

Sample m trajectories under the current policy

$\Delta\theta \leftarrow 0$

For i=1, …, m **do**

For t=1, ... , T **do**

$$A_t = \sum_{t' \geq t} \gamma^{t'-t} r_t^i - V_\phi(s_t^i)$$

$$\Delta\theta \leftarrow \Delta\theta + A_t \nabla_\theta \log(a_t^i | s_t^i)$$

$$\Delta\phi \leftarrow \sum_i \sum_t \nabla_\phi ||A_t^i||^2$$

$$\theta \leftarrow \alpha\Delta\theta$$

$$\phi \leftarrow \beta\Delta\phi$$

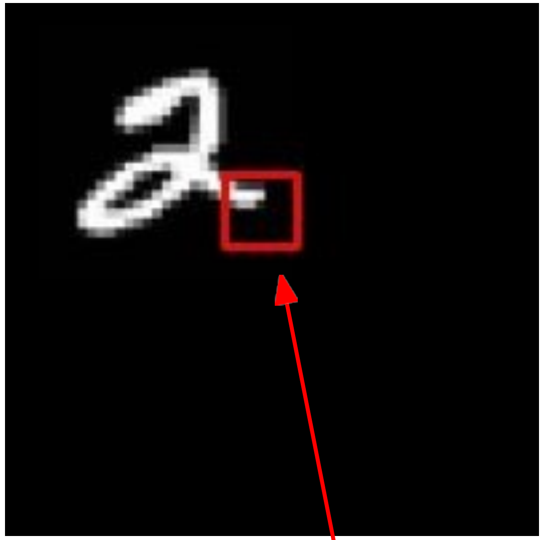
End for

REINFORCE in action: Recurrent Attention Model (RAM)

Objective: Image Classification

Take a sequence of “glimpses” selectively focusing on regions of the image, to predict class
- Inspiration from human perception and eye movements
- Saves computational resources => scalability
- Able to ignore clutter / irrelevant parts of image

State: Glimpses seen so far
Action: (x,y) coordinates (center of glimpse) of where to look next in image
Reward: 1 at the final timestep if image correctly classified, 0 otherwise



glimpse

[Mnih et al. 2014]

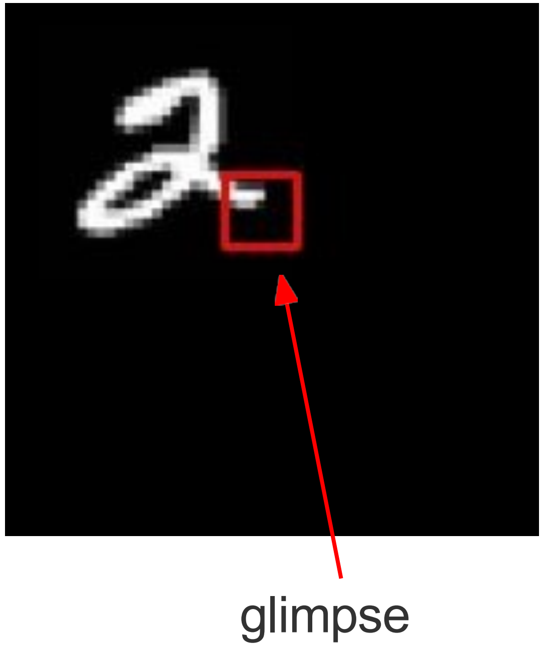
REINFORCE in action: Recurrent Attention Model (RAM)

Objective: Image Classification

Take a sequence of “glimpses” selectively focusing on regions of the image, to predict class

- Inspiration from human perception and eye movements
- Saves computational resources => scalability
- Able to ignore clutter / irrelevant parts of image

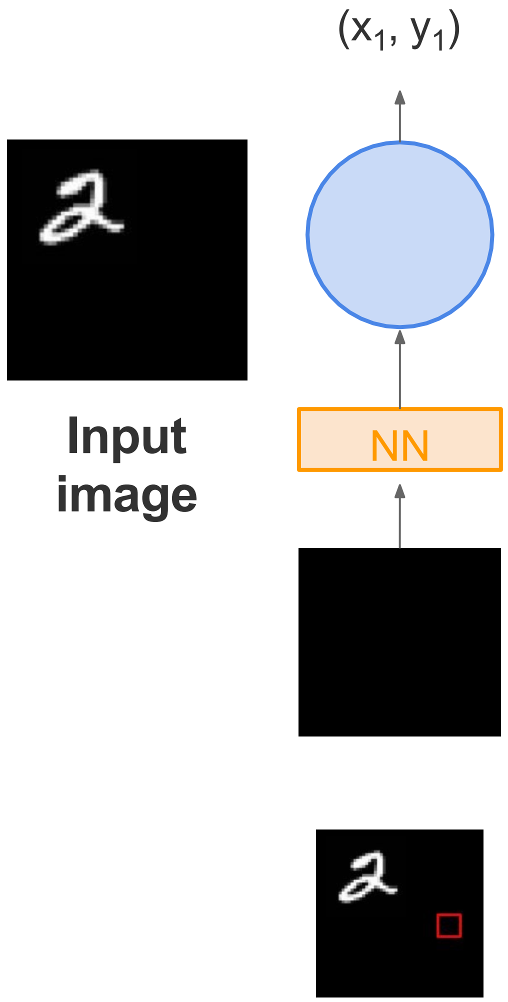
State: Glimpses seen so far
Action: (x,y) coordinates (center of glimpse) of where to look next in image
Reward: 1 at the final timestep if image correctly classified, 0 otherwise



Glimpsing is a non-differentiable operation => learn policy for how to take glimpse actions using REINFORCE
Given state of glimpses seen so far, use RNN to model the state and output next action

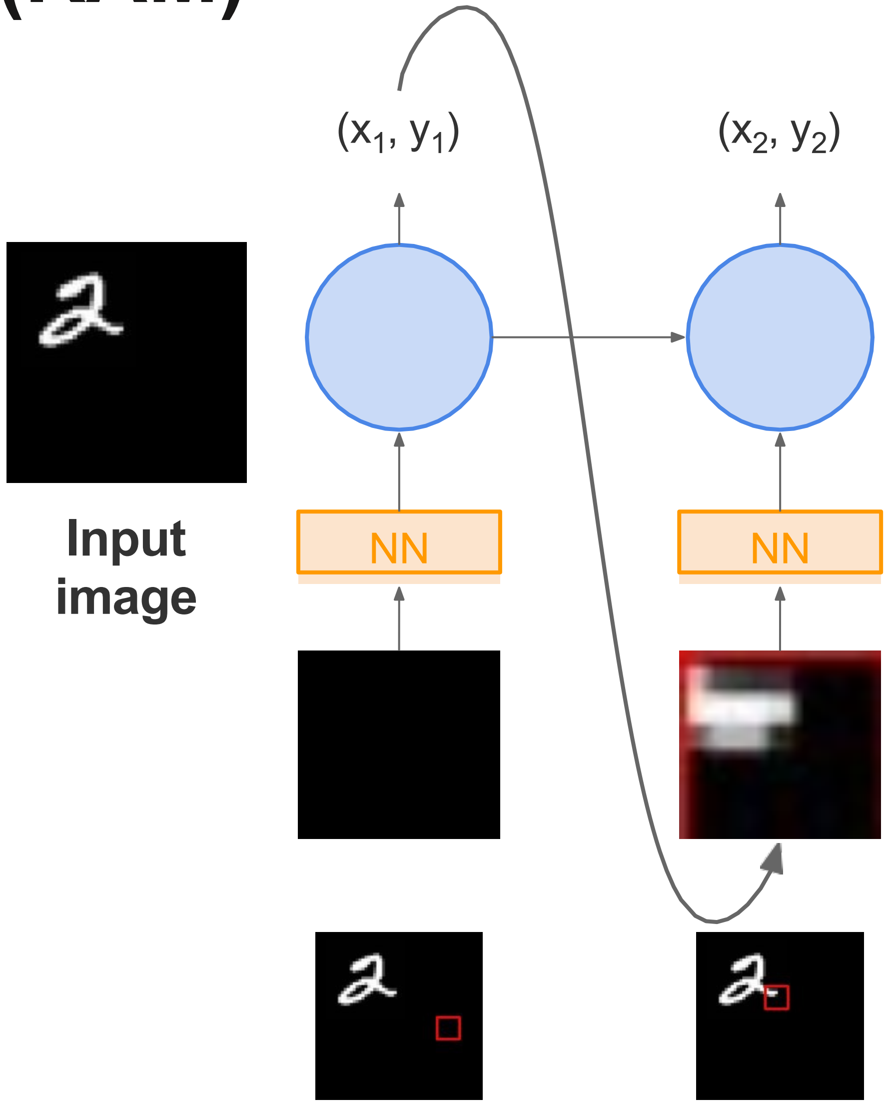
[Mnih et al. 2014]

REINFORCE in action: Recurrent Attention Model (RAM)



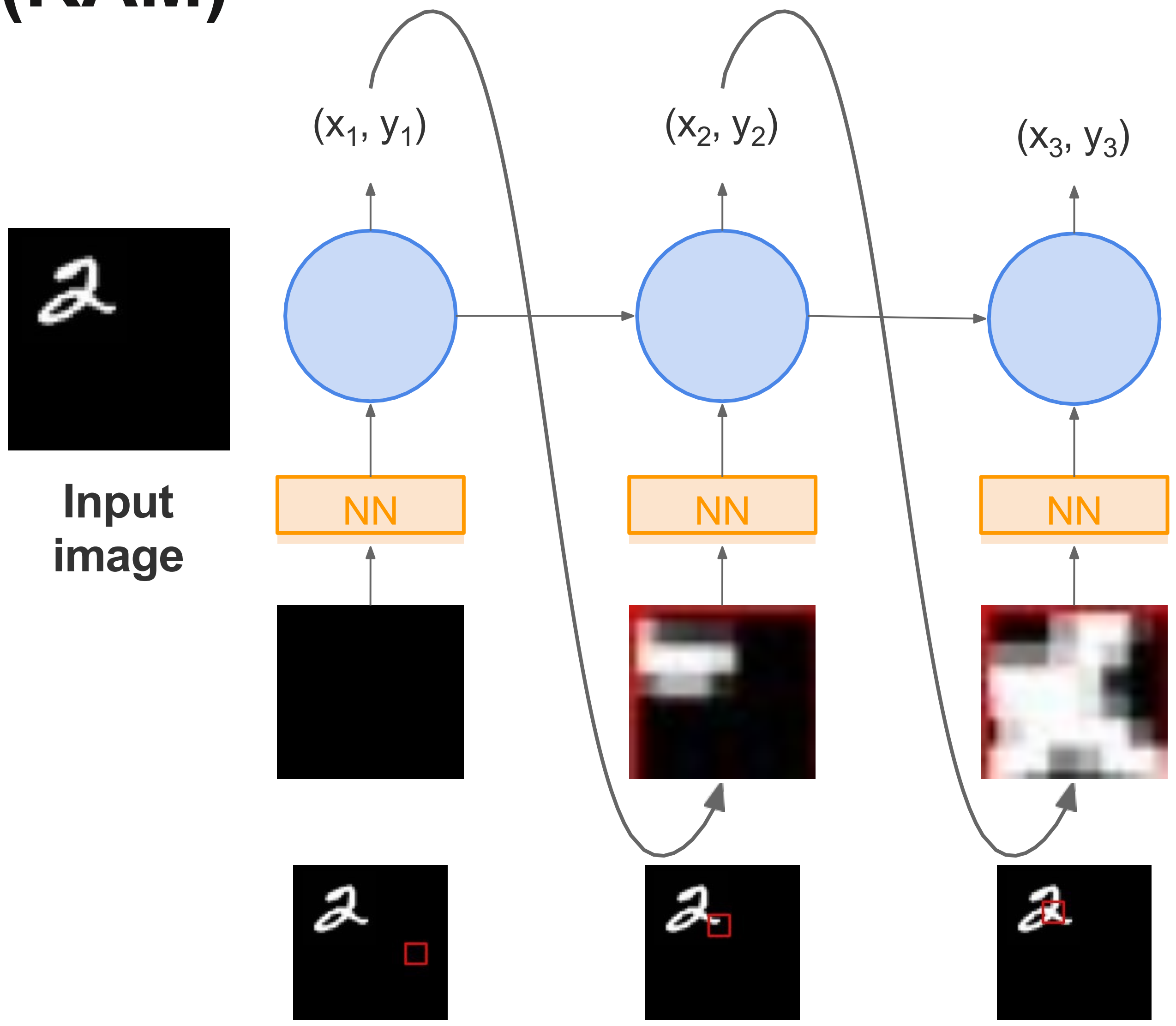
[Mnih et al. 2014]

REINFORCE in action: Recurrent Attention Model (RAM)



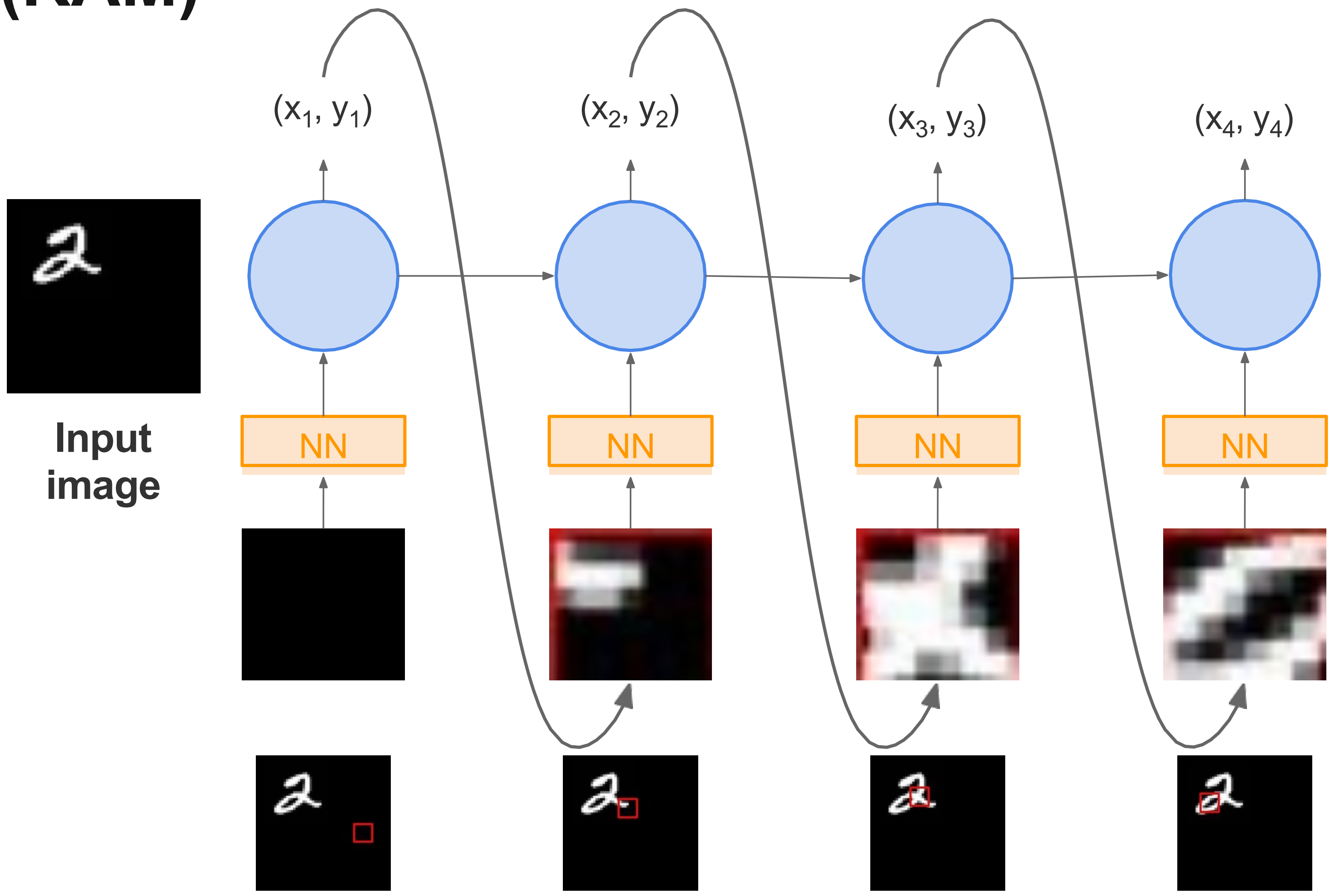
[Mnih et al. 2014]

REINFORCE in action: Recurrent Attention Model (RAM)



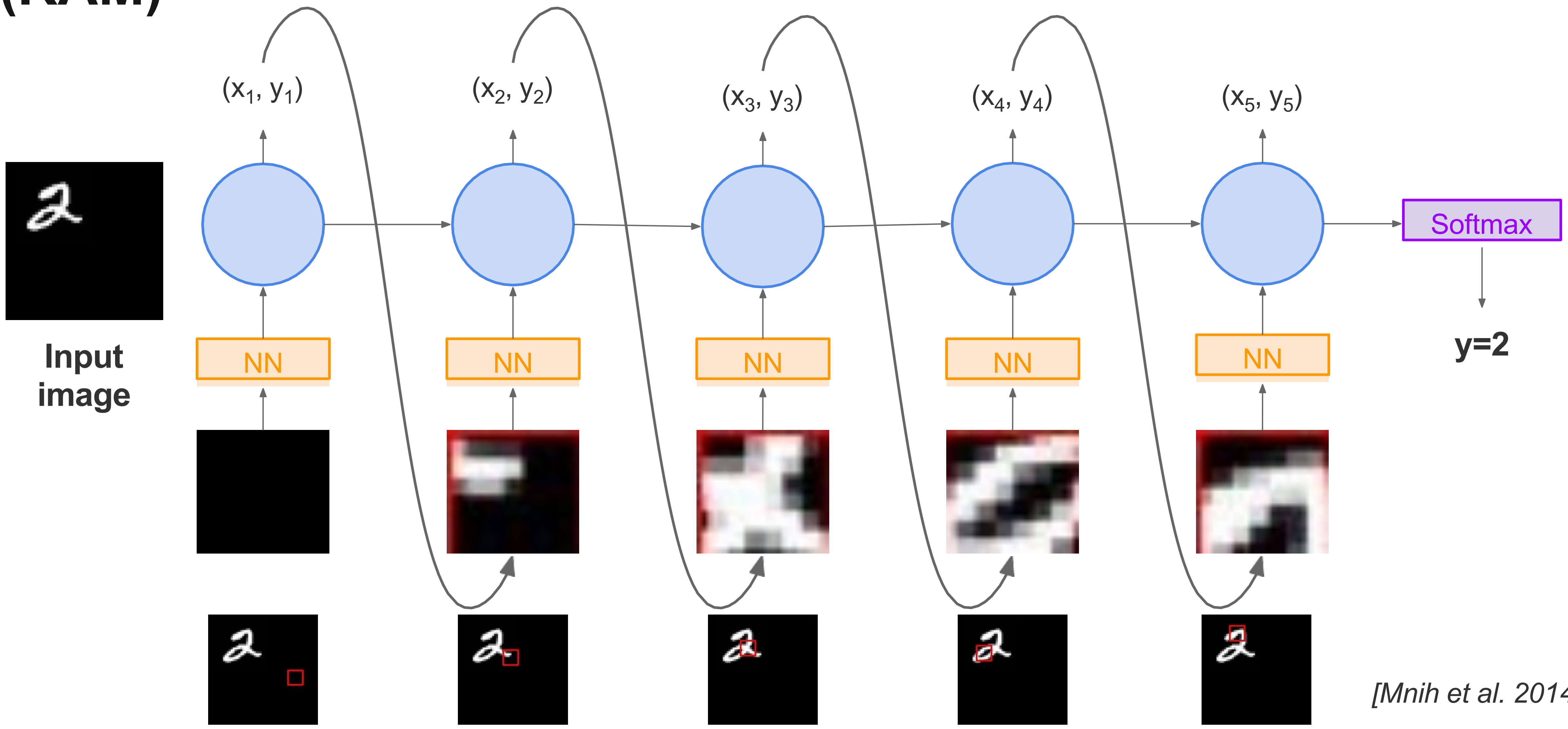
[Mnih et al. 2014]

REINFORCE in action: Recurrent Attention Model (RAM)



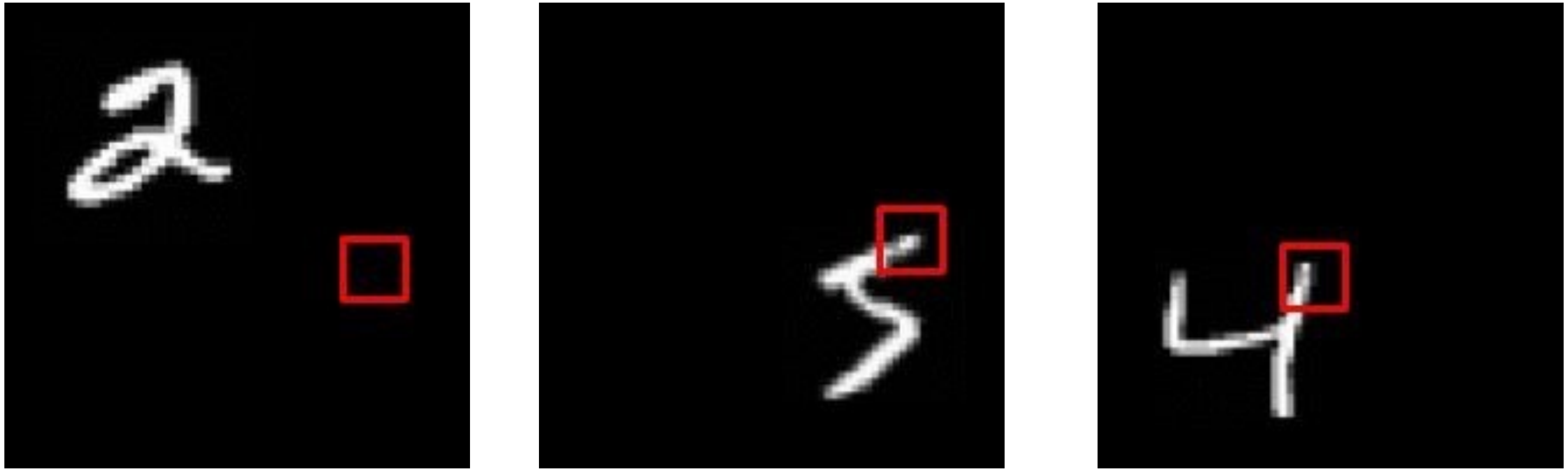
[Mnih et al. 2014]

REINFORCE in action: Recurrent Attention Model (RAM)



[Mnih et al. 2014]

REINFORCE in action: Recurrent Attention Model (RAM)

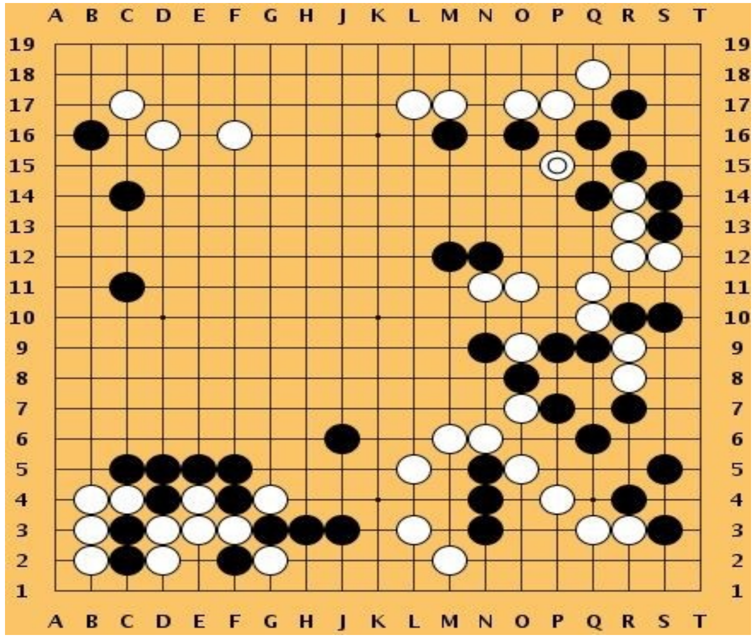


Has also been used in many other tasks including fine-grained image recognition, image captioning, and visual question-answering!

[Mnih et al. 2014]

Figures copyright Daniel Levy, 2017. Reproduced with permission.

More policy gradients: AlphaGo



Overview:

- Mix of supervised learning and reinforcement learning
- Mix of old methods (Monte Carlo Tree Search) and recent ones (deep RL)

How to beat the Go world champion:

- Featurize the board (stone color, move legality, bias, …)
- Initialize policy network with supervised training from professional go games, then continue training using policy gradient (play against itself from random previous iterations, +1 / -1 reward for winning / losing)
- Also learn value network (critic)
- Finally, combine combine policy and value networks in a Monte Carlo Tree Search algorithm to select actions by lookahead search

[Silver et al., Nature 2016]

This image is CC0 public domain

Key Takeaways

- Markov Decision Process (MDP)
- Q-learning
    - Bellman equation
    - Deep Q-learning, experience replay
- Policy gradients

- Guarantees:
    - Policy Gradients: Converges to a local minima of J(θ), often good enough!
    - Q-learning: Zero guarantees since you are approximating Bellman equation with a complicated function approximator

Questions?