Reinforcement Learning for NLP

Caiming Xiong

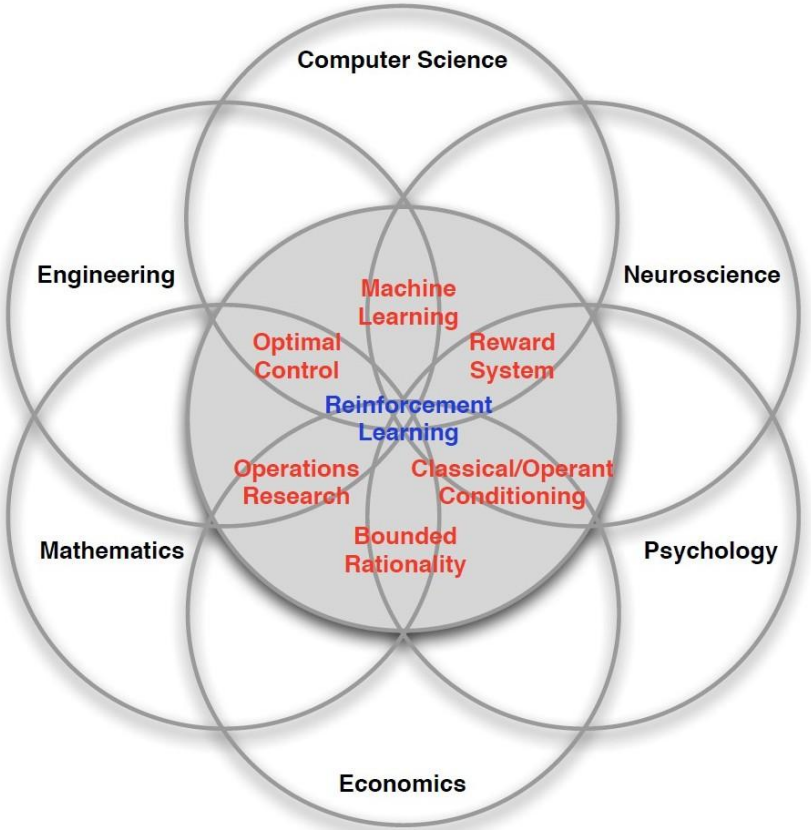
Salesforce Research

CS224N/Ling284

Outline

- Introduction to Reinforcement Learning
- Policy-based Deep RL
- Value-based
- Deep RL
- Examples of RL for NLP

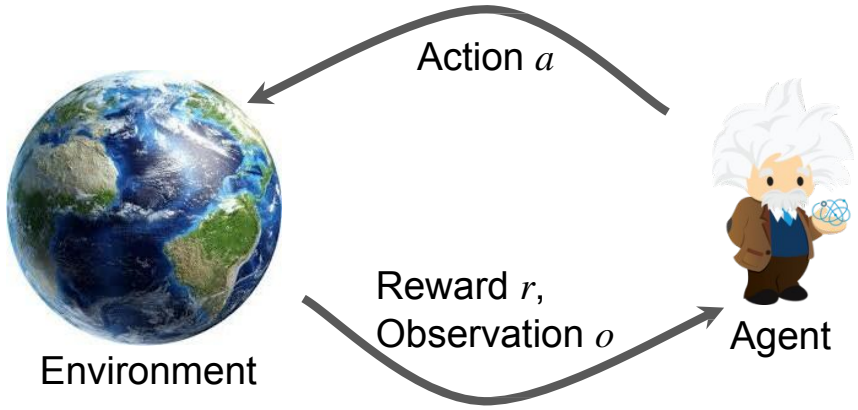
Many Faces of RL



By David Silver

What is RL?

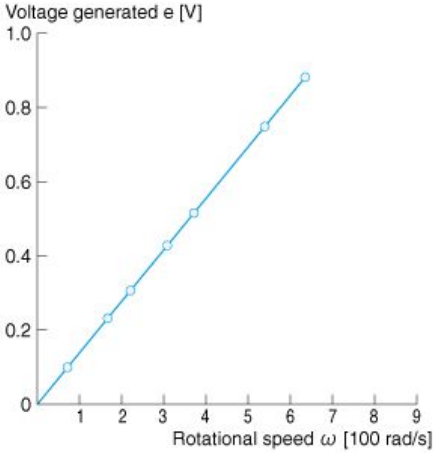
- RL is a general-purpose framework for **sequential** decision-making
- Usually describe as agent interacting with unknown environment
- Goal: select action to **maximize a future cumulative reward**



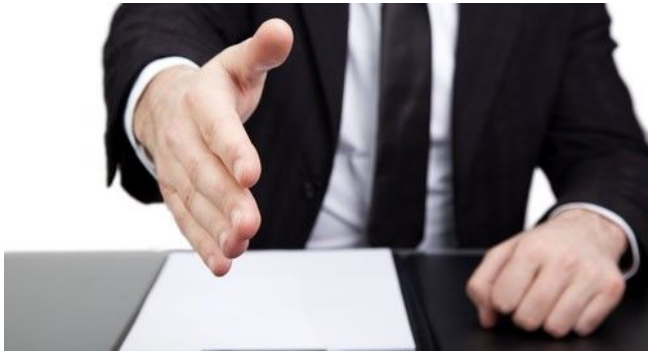
Environment

- Agent learns **mapping** from situations to actions.
  - By trial-and-error interactions with dynamic environment
- Environment must be at least **partially observable** by RL system.
- Observations may be low level, high level or “mental”.

Voltage



Accept job offer



Shift in attention

Motor Control

- Observations: images from camera, joint angle
- Actions: joint torques
- Rewards: navigate to target location, serve and protect humans

Business Management

- Observations: current inventory levels and sales history
- Actions: number of units of each product to purchase
- Rewards: future profit

Similarly, there also are resource allocation and routing problems ….

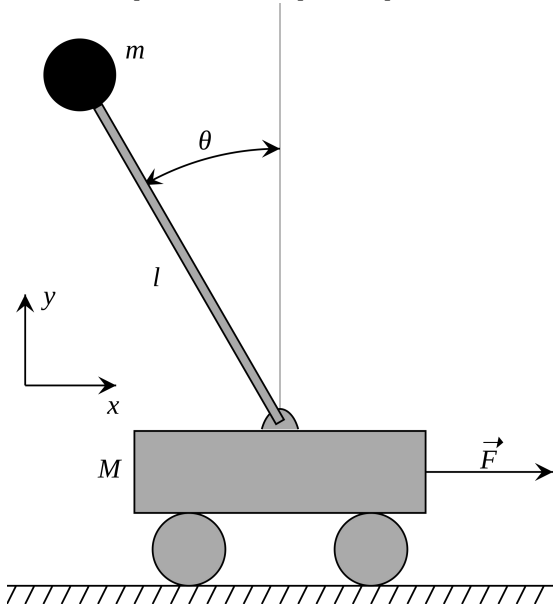
Games

Two Minute Papers

Reinforcement Function

- "Goal" of RL system
- Function of **future reinforcements(rewards)** agent seeks to maximise.
- i.e. Exists mapping from state-action pairs to reinforcements.
  - After performing an **action given a state**, agent receives a **scalar value**.

Example: Cart-pole problem



Pure **Delayed** Reward Reinforcement Function

- Goal: Balance pendulum
- State: Dynamic state of pole system
- Actions: Left or right
- Reinforcement function:
  - -1 if pole falls
  - 0 everywhere else
- Agent will learn sequence of actions to balance pole and avoid -1 reinforcement.

State

- Experience is a sequence of observations, actions, rewards

- The state is a summary of experience

$$s_t = f(o_1, r_1, a_1, \ldots, a_{t-1}, o_t, r_t)$$

RL Agent

Major components:

- Policy: agent's behavior function
  - Mapping of **states to actions**
- Value function: how good would each state and/or action be
  - **Sum of reinforcements** received from that state and **following a fixed policy** to a terminal state.
- Model: agent's prediction/representation of the environment

Policy

A function that maps from state to action:

- Deterministic policy:

$$a = \pi(s)$$

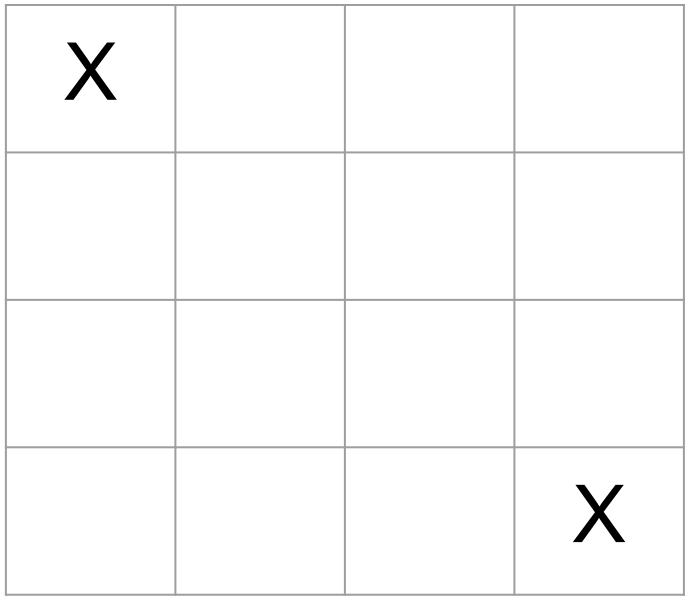
- Stochastic policy:

$$\pi(a|s) = \mathbb{P}[a|s]$$

Example of Value Function

Simple Markov Decision Process with 16 States

State space



- Each square represents a state
- Reinforcement function is -1 everywhere
  - -1 reinforcement per transition
- 4 Actions: Left, right, up, down
- Goal states: Upper left & Bottom right

Example of Value Function

Value function of **random** policy

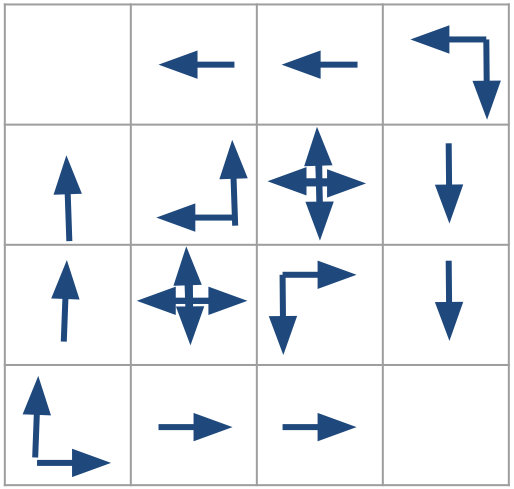
0	-14	-20	-22
-14	-18	-22	-20
-20	-22	-18	-14
-22	-20	-14	0

- For each state **randomly choose 1 action**
- Numbers in states represent **expected values of states**
- Following a random policy, when starting from lower left corner, on **average 22 transitions** before reaching terminal state

Value function of **optimal** policy

0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

Optimal value function



Optimal policy

- Value of state at lower left corner is -3
- Takes only 3 transitions to reach terminal state
- Given optimal value function, trivial to extract optimal policy

Q-Learning

- Finds mapping of **state-action pairs** to “Q-values”
  - Vs. mapping from **states** to state values (Value iteration)
- In each state, each action is associated with a Q-value, $Q(s_t, a_t)$
- Q-value - Expected future total reward
  - **Sum of the reinforcements** (possibly discounted) received when performing the **associated action** and **following the given policy** thereafter
- Optimal Q-value, $Q^{\pi}(s, a)$, is the sum of reinforcements received when performing associated action and following **optimal policy** thereafter.
- Value of state: **Maximum Q-value** in given state
- $Q(s_t, a_t) = r(s_t, a_t) + \gamma \max_{u_{t+1}} Q(s_{t+1}, a_{t+1})$ Bellman equation for Q-learning

$BQ^{\pi}(s,a) =$

Value Function

- Q-value function gives expected future total reward
  - from state and action (s, a)
  - under policy π
  - with discount factor $\gamma \in (0,1)$

$$Q^{\pi}(s,a) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s, a\right]$$

  - Show how good current policy

- Value functions can be defined using Bellman equation

$$Q^{\pi}(s,a) = \mathbb{E}_{s',a'}\left[r + \gamma Q^{\pi}(s',a') \mid s, a\right]$$

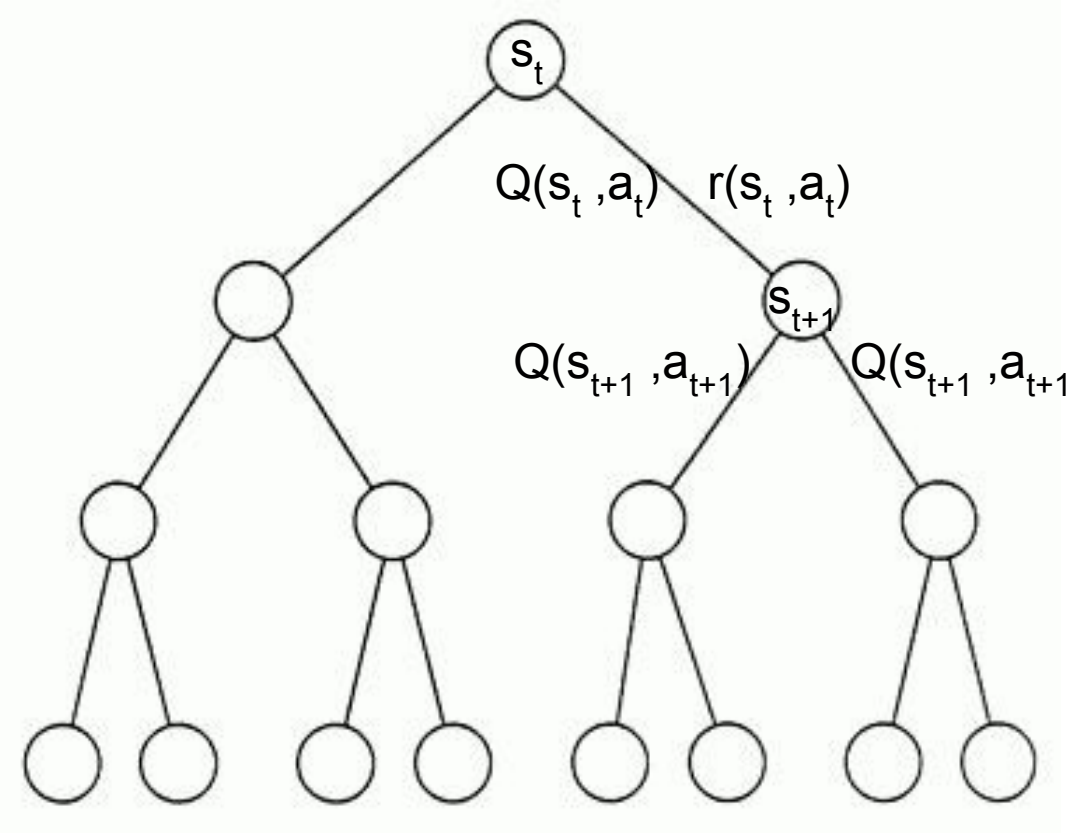
- Bellman backup operator

$$\mathrm{B}^{\pi}\mathrm{Q}(\mathrm{s},\mathrm{a}) = \mathrm{E}_{\mathrm{s}',\mathrm{a}'}[\mathrm{r} + \gamma Q^{\pi}(s',a') | s, a] \qquad Q,\ B^{\pi}Q,\ (B^{\pi})^2Q,\ (B^{\pi})^3Q,\ \cdots \rightarrow Q^{\pi}$$

Bellman Backup Operator

$$B^{\pi}Q(s,a) = E_{s',a'}[r + \gamma Q^{\pi}(s',a')|s,a]$$

$$Q,\ B^{\pi}Q,\ (B^{\pi})^2Q,\ (B^{\pi})^3Q,\ \cdots \rightarrow Q^{\pi}$$



- Lookup table to represent Q-function
- To update prediction $Q(s_t, a_t)$:
  - Perform a_t, causing transition to s_{t+1} and return $r(s_t, a_t)$.
- Only need to find **max** Q-value in **new state** to update Q-value for action just performed.
- After many updates, Q-value for a_t **converges** to **expected sum** of all reinforcements received when performing a_t and following optimal policy thereafter.

Value Function

- For optimal Q-value function $Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$, then policy function is deterministic, the Bellman equation becomes:

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

$$\mathrm{B}^{\pi}\mathrm{Q}(\mathrm{s},\mathrm{a}) = \mathrm{E}_{s'}[\mathrm{r} + \gamma \max_{a'} Q^{\pi}(s',a')|s,a]$$

$$Q,\ BQ,\ B^2Q,\ \cdots \rightarrow Q^*$$

NEXT - We will discuss about Deep Reinforcement Learning

What is Deep RL?

- Use deep neural network to approximate
  - Policy
  - Value function
  - Model
- Optimized by SGD

Approaches

- **Value-based Deep RL**
- **Policy-based Deep RL**

Value-based Deep RL

- Deep Q Learning - DQN
- Categorical DQN
- epsilon - Rainbow
    - Double DQN
    - Dueling Networks
    - Prioritized Replay
    - n-step learning
    - NoisyNets

Deep Q-Learning

- Wait before getting into Deep Q Learning
- Let’s discuss Q-Learning

Q Learning Recap

A = {1, . . . , K} set of actions
rt = reward received at time step t
st = x1, a1, x2, ..., at−1, xt
st is the state at time step t
don't be confused with xt, st
γ = discount factor per time-step [0 < **γ < 1**]

Future *discounted return* at time t

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$$

optimal action-value function Q* (s, a)

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[R_t | s_t = s, a_t = a, \pi\right]$$

The optimal action-value function Q (s, a) as the maximum expected return achievable by following any strategy, after seeing some sequence s and then taking some action a.*
where π is a policy mapping sequences to actions (or distributions over actions)

Bellman Equation

The optimal action-value function obeys an important identity known as the Bellman equation

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

The basic idea behind many reinforcement learning algorithms is to estimate the action-value function, by using the Bellman equation as an iterative update

$$Q_{i+1}(s,a) \quad = \quad \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a') | s,a\right].$$

Such value iteration algorithms converge to the optimal action-value function,
Qi → Q* as i → ∞

$$NewQ(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma max Q'(s',a') - Q(s,a)]$$

- $NewQ(s,a)$: New Q value for that state and that action
- $Q(s,a)$: Current Q value
- α: Learning Rate
- $R(s,a)$: Reward for taking that action at that state
- γ: Discount rate
- $max Q'(s',a')$: Maximum expected future reward **given the new s' and all possible actions at that new state**

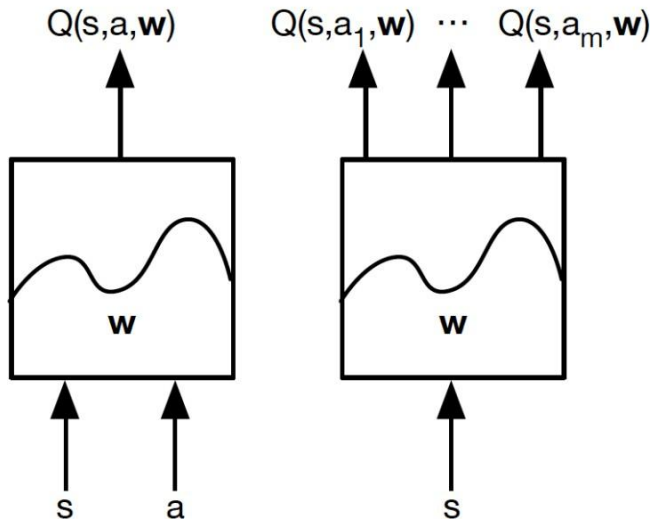
You can use different algorithm like

- Monte Carlo
- TD , TD (0), TD (lambda)

Deep Q-Learning

- Represent value function by Q-network

$$Q(s, a, \mathbf{w}) \approx Q^*(s, a)$$



Deep Q-Learning

- Optimal Q-values should obey Bellman equation

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

- Treat right-hand side as target network, given (s, a, r, s'), optimize MSE loss via SGD:

$$I = \left(r + \gamma \max_{a} Q(s',a',\mathbf{w}) - Q(s,a,\mathbf{w})\right)^2$$

Loss function

Loss: (target - predicted) , yi is target here

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

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

Gradient:

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

Target Y

notice the weights

$Q(s',a';\theta_{i-1})$

Predicted Y_hat

notice the weights

$Q(s,a;\theta_i)$

What are some problems here ?

But diverges using neural networks due to:

- Correlations between samples
- Non-stationary targets

Deep Q-Learning

Experience Replay: remove correlations, build data-set from agent's own experience

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
...
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow \quad s, a, r, s'$

- Sample experiences from data-set and apply update

$$I = \left(r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

- To deal with non-stationarity,
  - target parameters is fixed,
  - or use a seperate Q_network for target and predicted which is called **Double DQN**

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

Double DQNs

- To handle the problem of over estimation of Q-Values
- *how are we sure* that the **best action of the next state is the action with highest Q-Value**
- The solution is
  - use our DQN network to select what is the best action to take for the next state
    - (action with the highest Q-Value)
  - use our target network to calculate the target the target Q value of taking that action at the next state.

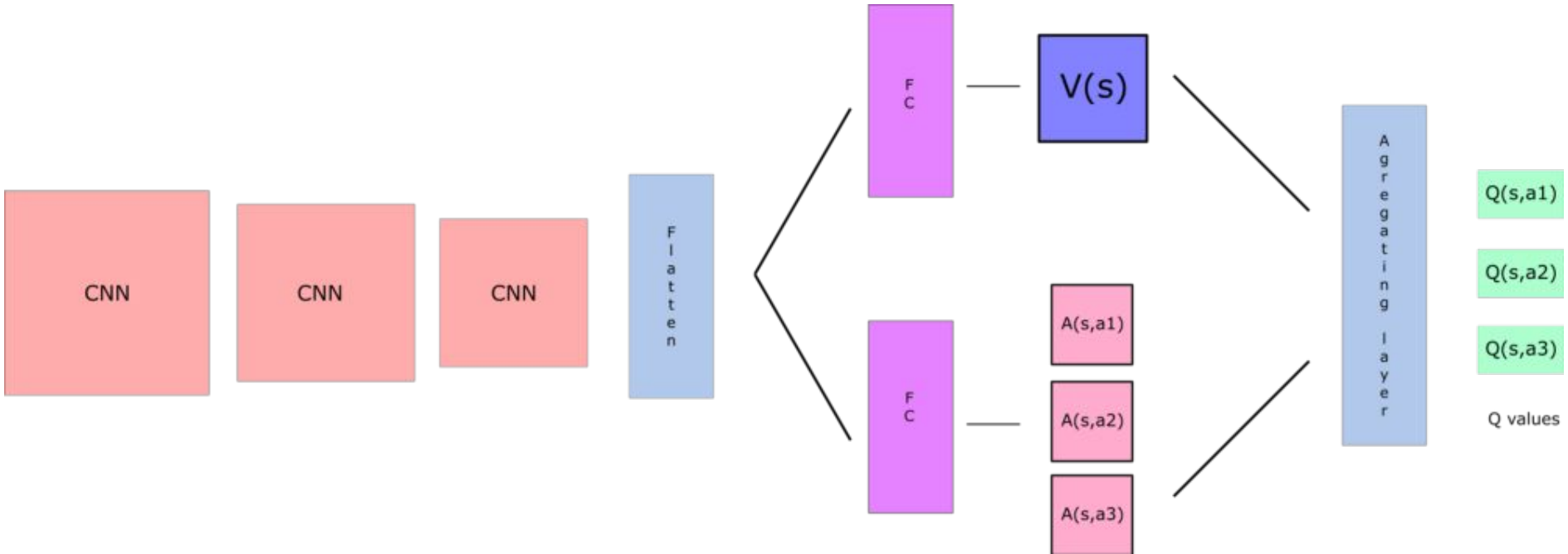
$$\underbrace{Q(s,a)}_{\text{TD target}} = r(s,a) + \gamma Q(s', \underbrace{argmax_a Q(s',a)}_{\substack{\text{DQN Network choose} \\ \text{action for next state}}})$$

Target network calculates the Q value of taking that action at that state

Dueling DQN (aka DDQN)

- Remember that Q-values correspond **to how good it is to be at that state and taking an action at that state Q(s,a).**
- So we can decompose Q(s,a) as the sum of:
  - **V(s)**: the value of being at that state
  - **A(s,a)**: the advantage of taking that action at that state (how much better is to take this action versus all other possible actions at that state).

$$Q(s,a) = A(s,a) + V(s)$$



Prioritized Experience Replay

- Prioritized Experience Replay (PER) was introduced in 2015 by Tom Schaul. The idea is that some experiences may be more important than others for our training, but might occur less frequently.
- We want to take in priority **experience where there is a big difference between our prediction and the TD target, since it means that we have a lot to learn about it.**
- We use the absolute value of the magnitude of our TD error:

$$p_t = |\delta_t| + e$$

Magnitude of our TD error

Constant assures that no experience has 0 probability to be taken.

Prioritized Experience Replay cont..

But how are we going to sample from the ReplayMemory Dataset

we can't just do greedy prioritization,
because it will lead to always training the same experiences (that have big priority),
and thus over-fitting.

Stochastic prioritization

Priority value

$$P(i) = \frac{p_i^a}{\sum_k p_k^a}$$

Normalized by all priority values in Replay Buffer

Hyperparameter used to **reintroduce some randomness in the experience selection for the replay buffer**

If **a = 0** pure uniform randomness

If **a = 1** only select the experiences with the highest priorities

Bias Correction - during weight updates

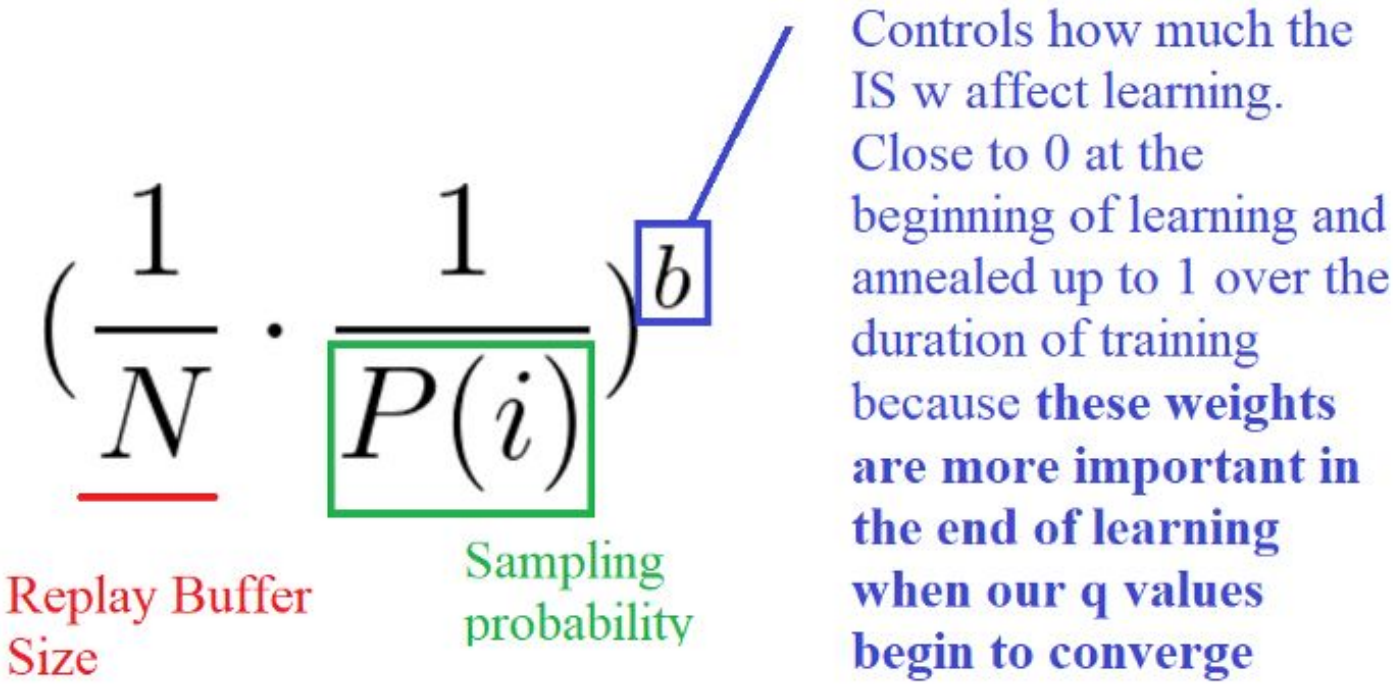
As consequence, during each time step, we will get a batch of samples with this probability distribution and train our network on it.

But, we still have a problem here. Remember that with normal Experience Replay, we use a stochastic update rule. As a consequence, the **way we sample the experiences must match the underlying distribution they came from.**

When we do have normal experience, we select our experiences in a normal distribution—simply put, we select our experiences randomly. There is no bias, because each experience has the same chance to be taken, so we can update our weights normally.

But, because we use priority sampling, purely random sampling is abandoned. As a consequence, we introduce bias toward high-priority samples (more chances to be selected).

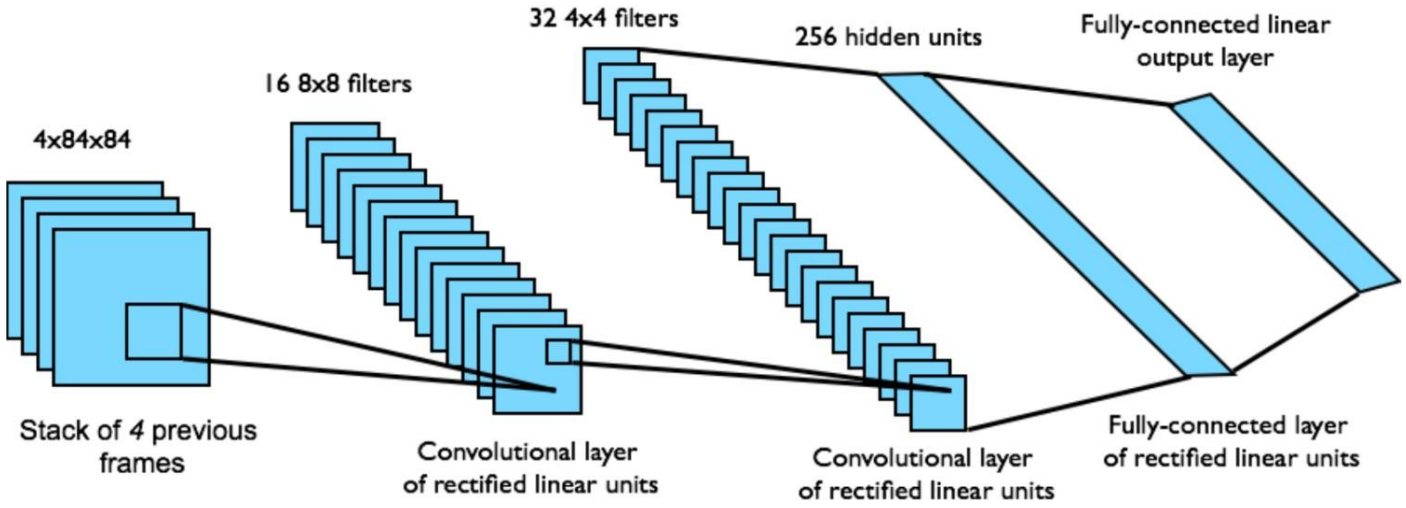
Importance sampling weights (IS)



1. The weights corresponding to high-priority samples have very little adjustment (because the network will see these experiences many times), whereas those corresponding to low-priority samples will have a full update

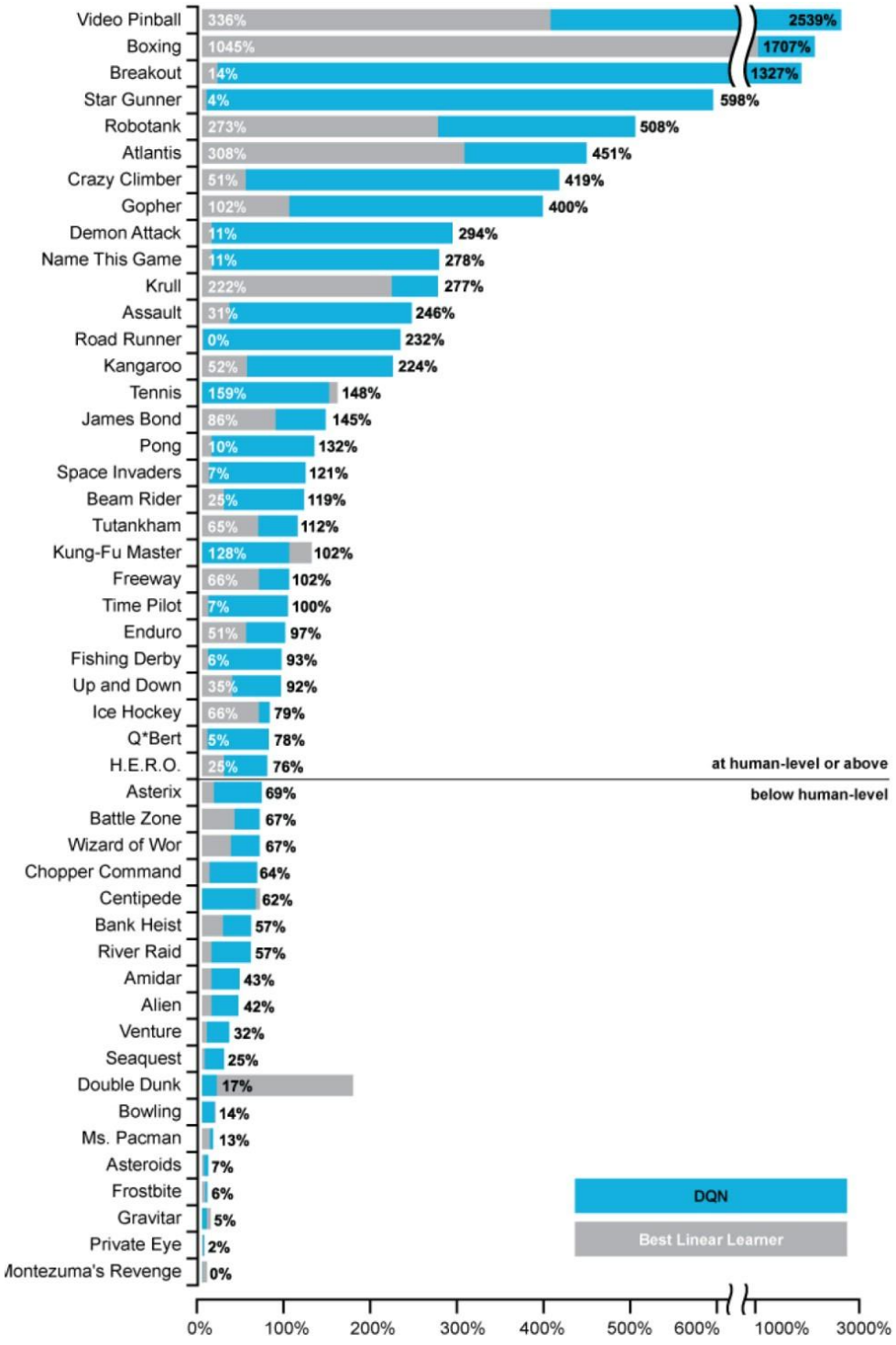
2. The role of **b** is to control how much these importance sampling weights affect learning. In practice, the b parameter is annealed up to 1 over the duration of training, because these weights are more important **in the end of learning when our q values begin to converge.**

Deep Q-Learning in Atari



Network architecture and hyperparameters fixed across all games

By David Silver



By David Silver

If you want to know more about RL, suggest to read:

Reinforcement Learning: An Introduction.
Richard S. Sutton and **Andrew G. Barto**
Second Edition, in progress
MIT Press, Cambridge, MA, 2017

Policy based - Deep RL

- Policy Gradient

Policy Gradient in a Nutshell

- In policy-based methods, instead of learning a value function that tells us what is the expected sum of rewards given a state and an action, we learn directly the policy function that maps state to action.

- It means that we directly try to optimize our policy function π without worrying about a value function. We'll directly parameterize π (select an action without a value function).

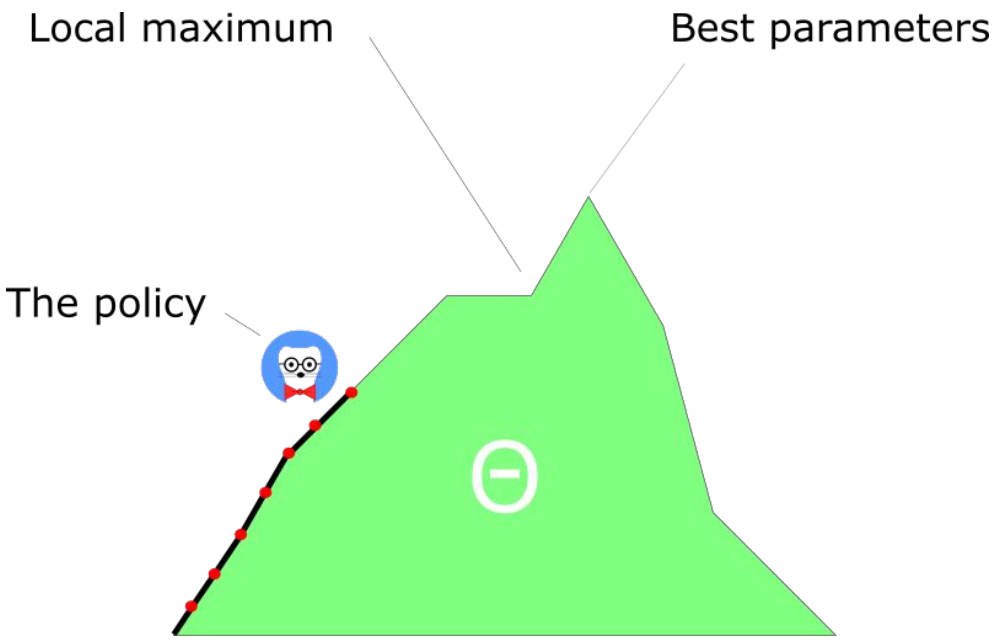
Two types of policy

A policy can be either deterministic or stochastic.

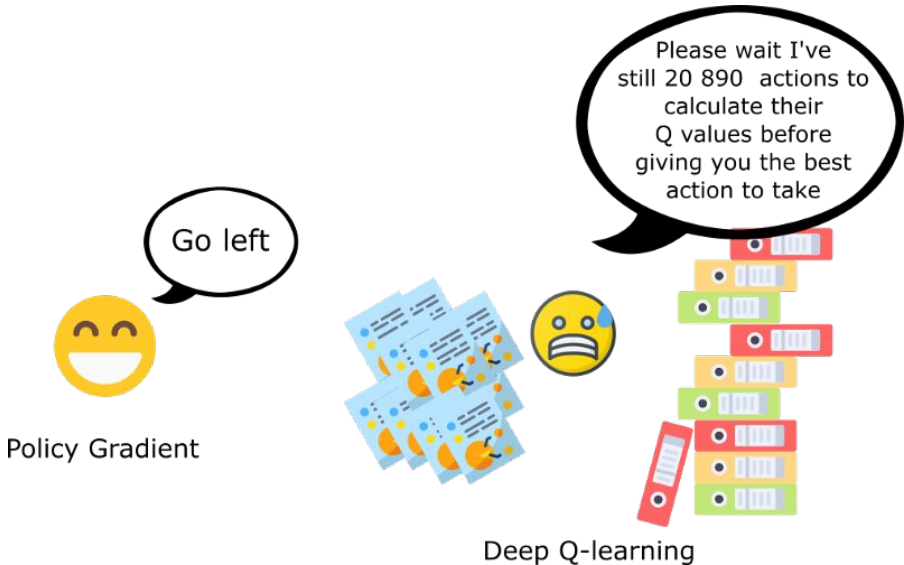
We will explore stochastic policy

Main Advantages of Policy Gradient

- **Convergence**



- **Policy gradients are more effective in high dimensional action spaces**
  - For Deep Q-Learning, what if we have an infinite possibility of actions?



Differences

1. Naturally, **Policy gradients** have one big disadvantage.
    a. A lot of the time, they **converge on a local maximum** rather than on the global optimum.
2. Instead of **Deep Q-Learning**, which always tries to reach the maximum, policy gradients **converge slower**, step by step.
    a. They can **take longer to train**.

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:

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

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

REINFORCE algorithm

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

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

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

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

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?

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.

Actor-Critic Algorithm

Initialize policy parameters θ, critic parameters ϕ

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**

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

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

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

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

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

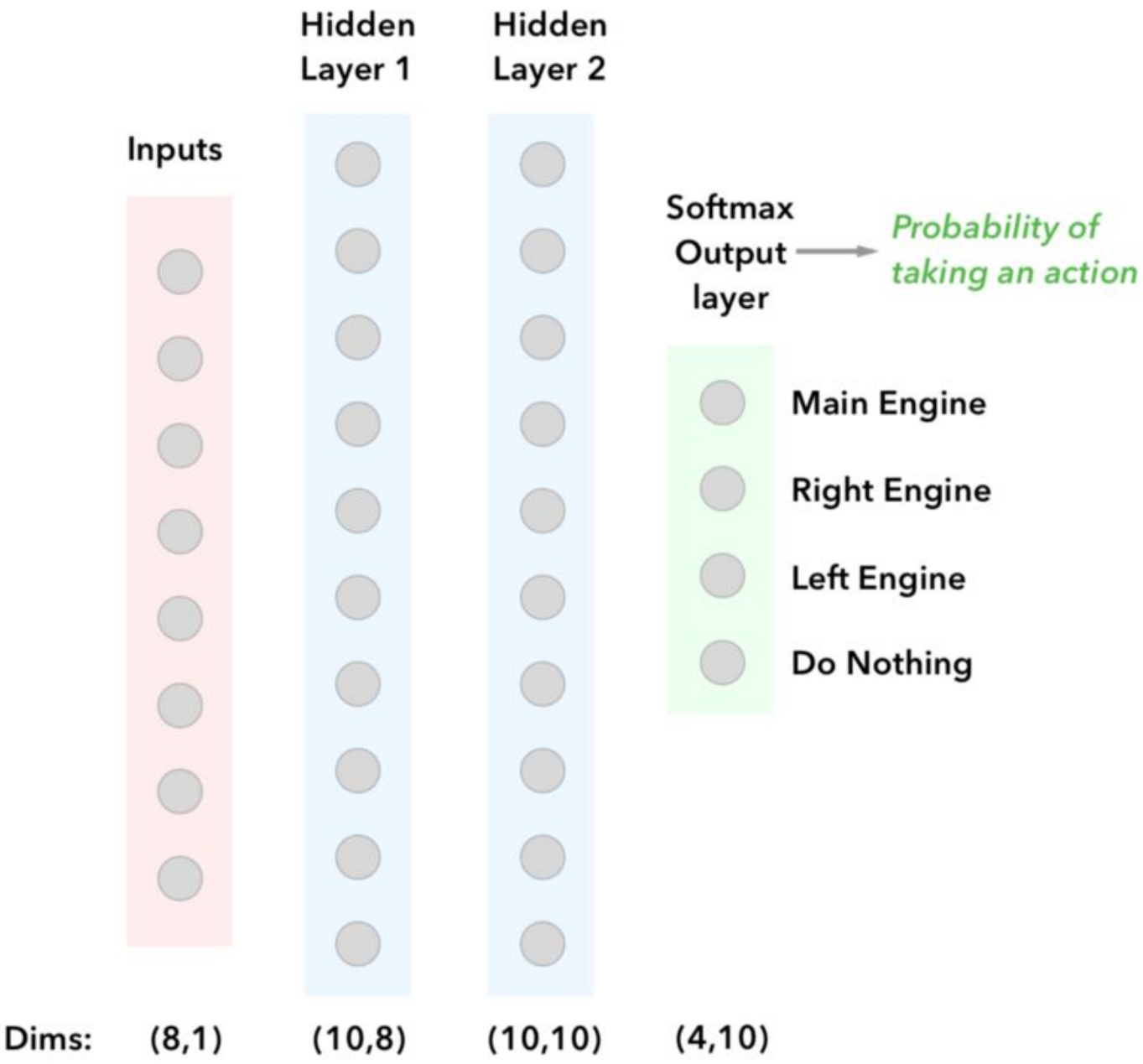
End for

Deep Policy Network

- Represent policy by deep neural network that

$$\max_{C} E_{B\sim G(B|C,H)}[r(a)|\theta, s]$$

- Ideas: given a bunch of trajectories,
  - Make the good trajectories/action more probable
  - Push the actions towards good actions


Hidden Layer 1
Hidden Layer 2
Inputs
Softmax Output layer
Probability of taking an action
Main Engine
Right Engine
Left Engine
Do Nothing
Dims:
(8,1)
(10,8)
(10,10)
(4,10)

NEXT - RL in NLP

RL in NLP

- Article summarization
- Question answering
- Dialogue generation
- Dialogue System
- Knowledge-based QA
- Machine Translation
- Text generation

RL in NLP

- **Article summarization**
- Question answering
- Dialogue generation
- Dialogue System
- Knowledge-based QA
- Machine Translation
- Text generation
  -
  -
  -

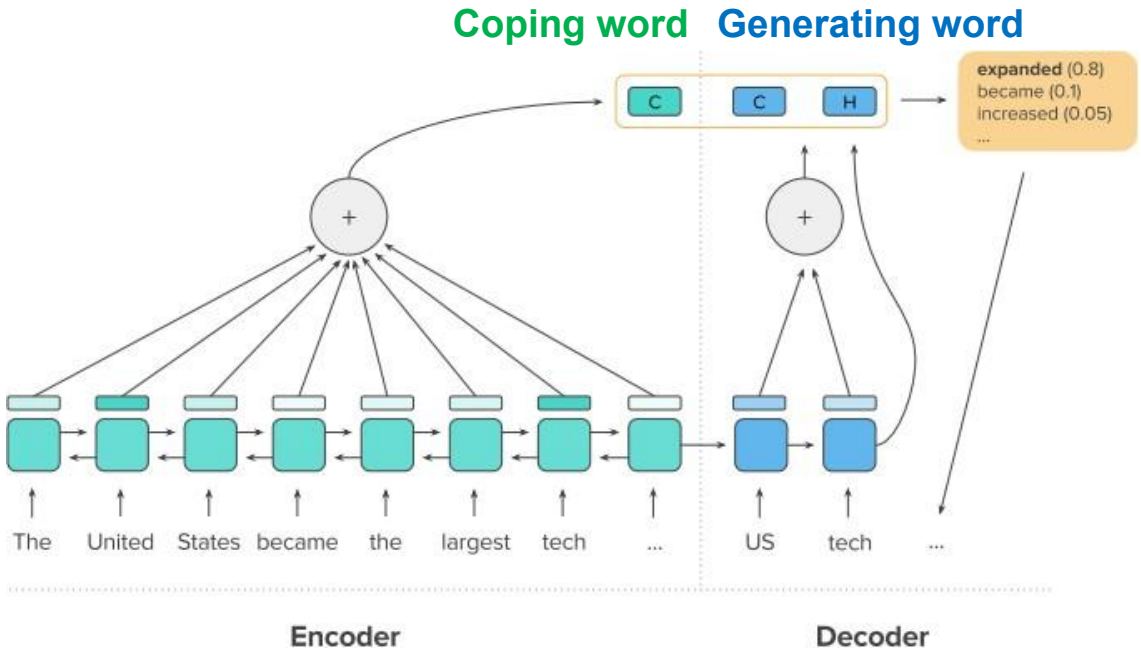
Article Summarization

Text summarization is the process of automatically generating natural language summaries from an input document while retaining the important points.

- extractive summarization
- abstractive summarization

A Deep Reinforced Model for Abstractive Summarization

Given x = $\{x_L, x_M, \cdots, x_O\}$ represents the sequence of input (article) tokens, y = $\{y_L, y_M, \cdots, y_R\}$, the sequence of output (summary) tokens



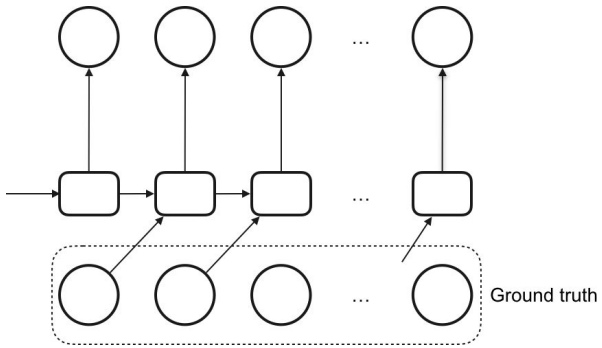
Paulus et. al.

A Deep Reinforced Model for Abstractive Summarization

The maximum-likelihood training objective:

$$L_{ml} = -\sum_{t=1}^{n'} \log p(y_t^* | y_1^*, \ldots, y_{t-1}^*, x)$$

Training with teacher forcing algorithm.



Paulus et. al.

A Deep Reinforced Model for Abstractive Summarization

There is discrepancy between training and test performance, because

- exposure bias
- potentially valid summaries
- metric difference

Paulus et. al.

A Deep Reinforced Model for Abstractive Summarization

Using reinforcement learning framework, learn a policy that maximizes a specific discrete metric.

Action: $u_t \in [copy, generate]$ and word y^H_t

State: hidden states of encoder and previous outputs

Reward: ROUGE score

$$L_{rl} = (r(\hat{y}) - r(y^s)) \sum_{t=1}^{n'} \log p(y_t^s | y_1^s, \ldots, y_{t-1}^s, x)$$

Where $p(y^H_t | y^H_1, \ldots, y^H_{t-1}, x) = p(u_t = copy) p(y^H_t | y^H_1, \ldots, y^H_{t-1}, x, u_t = copy) + p(u_t = generate) p(y^H_t | y^H_1, \ldots, y^H_{t-1}, x, u_t = generate)$

Paulus et. al.

A Deep Reinforced Model for Abstractive Summarization

Model	ROUGE-1	ROUGE-2	ROUGE-L
words-lvt2k-temp-att (Nallapati et al., 2016)	35.46	13.30	32.65
ML, no intra-attention	37.86	14.69	34.99
ML, with intra-attention	38.30	14.81	35.49
RL, with intra-attention	**41.16**	15.75	**39.08**
ML+RL, with intra-attention	39.87	**15.82**	36.90

Table 1: Quantitative results for various models on the CNN/Daily Mail test dataset

Model	ROUGE-1	ROUGE-2	ROUGE-L
ML, no intra-attention	44.26	27.43	40.41
ML, with intra-attention	43.86	27.10	40.11
RL, no intra-attention	**47.22**	30.51	**43.27**
ML+RL, no intra-attention	47.03	**30.72**	43.10

Table 2: Quantitative results for various models on the New York Times test dataset

Paulus et. al.

A Deep Reinforced Model for Abstractive Summarization

Human readability scores on a random subset of the CNN/Daily Mail test dataset

Model	Readability	Relevance
ML	6.76	7.14
RL	4.18	6.32
ML+RL	**7.04**	**7.45**

Paulus et. al.

RL in NLP

- Article summarization
- **Question answering**
- Dialogue generation
- Dialogue System
- Knowledge-based QA
- Machine Translation
- Text generation
  -
  -
  -

Text Question Answering

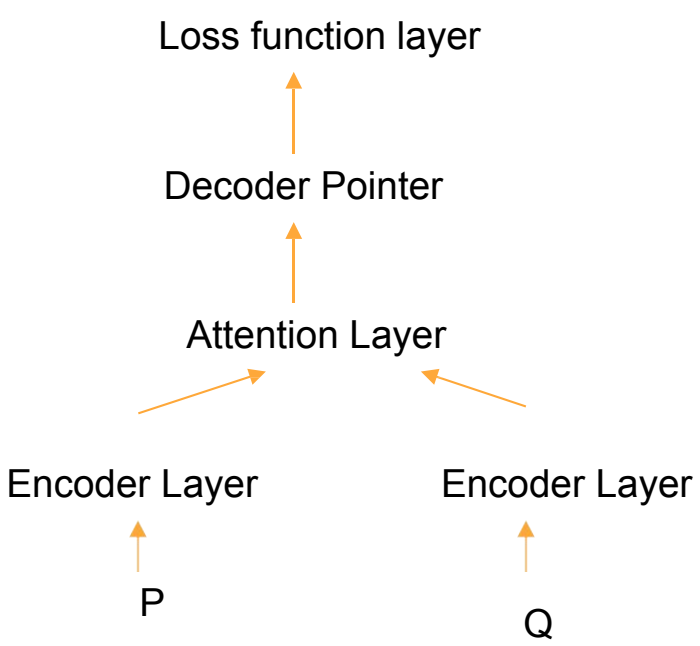
Passage: Tesla later approached Morgan to ask for more funds to build a more powerful transmitter. **When asked where all the money had gone, Tesla responded by saying that he was affected by the Panic of 1901** which he (Morgan) had caused. Morgan was shocked by the reminder of his part in the stock market crash and by Tesla's breach of contract by asking for more funds. Tesla wrote another plea to Morgan, but it was also fruitless. Morgan still owed Tesla money on the original agreement, and Tesla had been facing foreclosure even before construction of the tower began.

Question: On what did Tesla blame for the loss of the initial money?

Answer: Panic of 1901

Example from SQuaD dataset

Text Question Answering



Cross Entropy

LSTM + MLP
GRU + MLP

Self-attention
biAttention
Coattention

LSTM,
GRU

DCN+: MIXED OBJECTIVE AND DEEP RESIDUAL COATTENTION FOR QUESTION ANSWERING

Constraints of Cross-Entropy loss:

P: “Some believe **that the Golden State Warriors team of 2017** is one of the greatest teams in NBA history,…”

Q: “which team is considered to be one of the greatest teams in NBA history”

GT: **“the Golden State Warriors team of 2017”**

Ans1: “Warriors” Ans2: “history”

Xiong et. al.

DCN+: MIXED OBJECTIVE AND DEEP RESIDUAL COATTENTION FOR QUESTION ANSWERING

To address this, we introduce F1 score as extra objective combining with traditional cross entropy loss:

$$
\begin{aligned}
l_{rl}(\Theta) &= -\mathbb{E}_{\hat{\tau}\sim p_\tau}\left[R\left(s,e,\hat{s}_T,\hat{e}_T;\Theta\right)\right] \\
&\approx -\mathbb{E}_{\hat{\tau}\sim p_\tau}\left[F_1\left(\text{ans}\left(\hat{\text{s}}_\text{T},\hat{\text{e}}_\text{T}\right),\text{ans}\left(\text{s},\text{e}\right)\right)-F_1\left(\text{ans}\left(\text{s}_\text{T},\text{e}_\text{T}\right),\text{ans}\left(\text{s},\text{e}\right)\right)\right]
\end{aligned}
$$

$$
\begin{aligned}
\nabla_\Theta l_{rl}(\Theta) &= -\nabla_\Theta\left(\mathbb{E}_{\hat{\tau}\sim p_\tau}\left[R\right]\right) \\
&= -\mathbb{E}_{\hat{\tau}\sim p_\tau}\left[R\nabla_\Theta \log p_\tau\left(\tau;\Theta\right)\right]
\end{aligned}
$$

Not necessary for variable length.

Xiong et. al.

RL in NLP

- Article summarization
- Question answering
- **Dialogue generation**
- Dialogue System
- Knowledge-based QA
- Machine Translation
- Text generation

Dialogue generation

What is it?

- Generating responses for conversational agents.

A: Where are you going? (1)
B: I'm going to the restroom. (2)

A: how old are you? (1)
B: I'm 16. (2)

Dialogue generation

Training data

- OpenSubtitles dataset -- https://www.opensubtitles.org/

- A dialogue with 2 agents, p and q, will look as such:
  - p_1 , q_1 , p_2 , q_2 , … , p_i , q_i
  - Input: [p_i , q_i]
  - Output: p_{i+1}

Dialogue generation
Baseline model

- LSTM sequence-to-sequence (SEQ2SEQ)

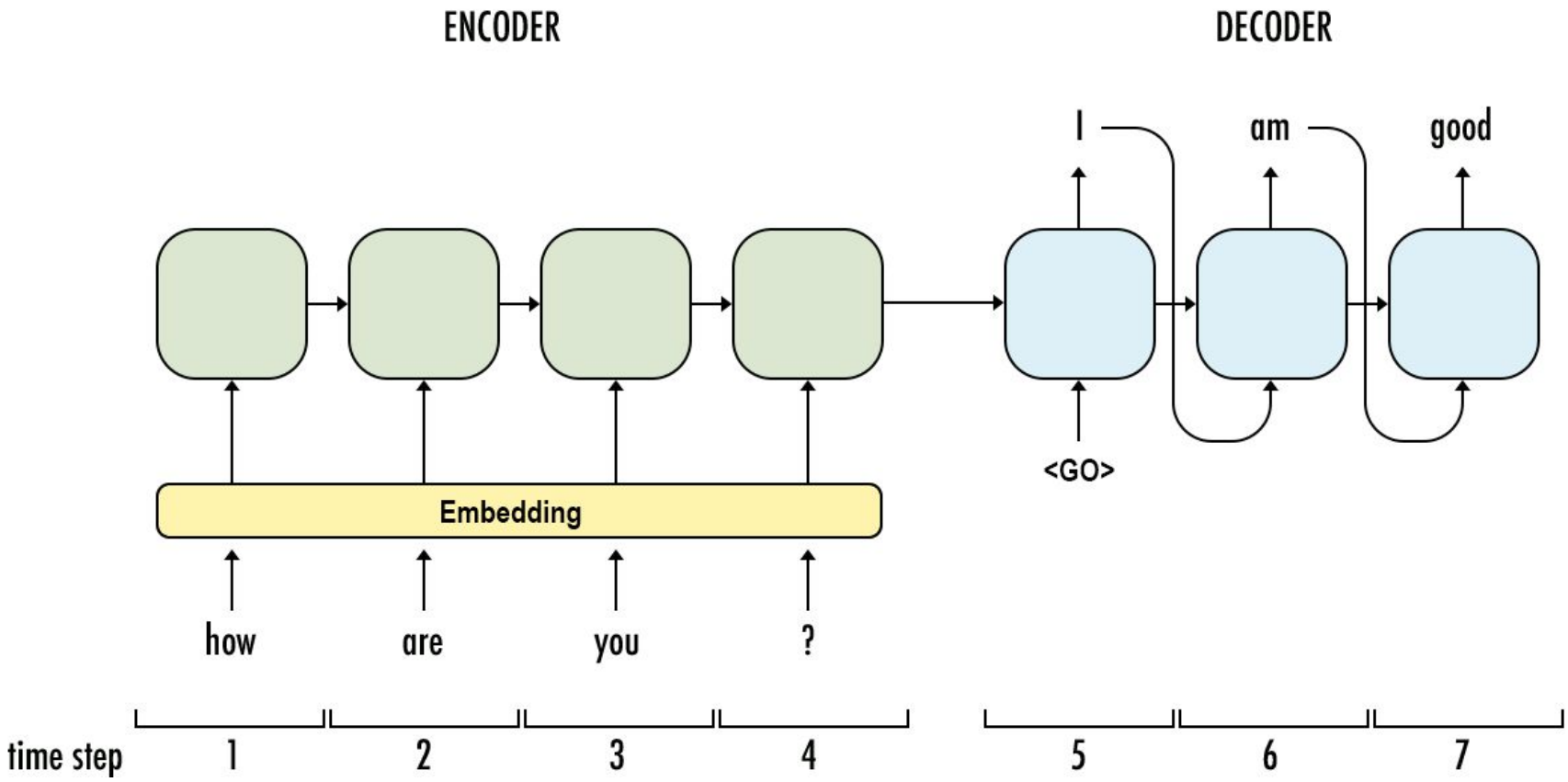


Image credits: https://towardsdatascience.com/sequence-to-sequence-model-introduction-and-concepts-44d9b41cd42d

Dialogue generation

Baseline model limitations

1) SEQ2SEQ models tend to generate highly generic responses, e.g. I don't know

2) Getting stuck in an infinite loop of repetitive responses

Baseline mutual information model (Li et al. 2015)
A: Where are you going? (1)
B: I'm going to the restroom. (2)
A: See you later. (3)
B: See you later. (4)
A: See you later. (5)
B: See you later. (6)
...
...
A: how old are you? (1)
B: I'm 16. (2)
A: 16? (3)
B: I don't know what you are talking about. (4)
A: You don't know what you are saying. (5)
B: I don't know what you are talking about . (6)
A: You don't know what you are saying. (7)
...

Dialogue generation

Baseline model limitations

1) SEQ2SEQ models tend to generate highly generic responses, e.g. I don't know

2) Getting stuck in an infinite loop of repetitive responses

To solve these, the model needs:

- Integrate developer-defined rewards that better mimic the true goal of chatbot development
- Model the long term influence of a generated response in an ongoing dialogue

Dialogue generation

RL model

- State: $[p_i , q_i]$
- Action: Sequence to be generated. The action space is infinite.
- Reward: Ease of answering, Information Flow, Semantic Coherence

- SEQ2SEQ model is similar to a policy gradients model -- it generates actions based on the state
- SEQ2SEQ optimizes parameters based on MLE
- RL further optimizes the parameters based on rewards

Dialogue generation

RL reward - Ease of answering

- A response generated by the model should be easy to respond to
- Responses which are hard to answer to -> more likely to be responded to with a ‘dull response’

- ‘Dull response’?
  - E.g. ‘I have no idea’

$$r_1 = -\frac{1}{N_{\mathbb{S}}} \sum_{s \in \mathbb{S}} \frac{1}{N_s} \log p_{\text{seq2seq}}(s|a)$$

- Higher likelihood of a dull response -> lower reward

Dialogue generation

RL reward - Information Flow

- New information should be contributed at each turn
- Penalize semantic similarity between consecutive turns from the same agent

$$r_2 = -\log\cos(h_{p_i}, h_{p_{i+1}}) = -\log\cos\frac{h_{p_i} \cdot h_{p_{i+1}}}{\|h_{p_i}\|\|h_{p_{i+1}}\|}$$

h_{pi} and h_{pi+1} are the hidden representations obtained from the encoder for 2 consecutive turns of agent p

- Higher cosine similarity -> lower reward

Dialogue generation

RL reward - Semantic coherence (Mutual Information)

- Avoid situations in which the generated replies are highly rewarded but are ungrammatical or not coherent
- Uses Mutual Information

$$r_3 = \frac{1}{N_a} \log p_{\text{seq2seq}}(a|q_i, p_i) + \frac{1}{N_{q_i}} \log p^{\text{backward}}_{\text{seq2seq}}(q_i|a)$$

Probability of generating action a, given state $[p_i, q_i]$

Backward probability of generating the previous dialogue q_i given action a

- Lower Mutual Information score -> lower reward

Dialogue generation

RL reward - Combining 3 rewards

- Final reward is weighted sum of the 3 rewards

$$r(a, [p_i, q_i]) = \lambda_1 r_1 + \lambda_2 r_2 + \lambda_3 r_3$$

Dialogue generation

Putting everything together

1) Supervised learning, training SEQ2SEQ model
   a) Not good for initializing policy model, due to generic responses

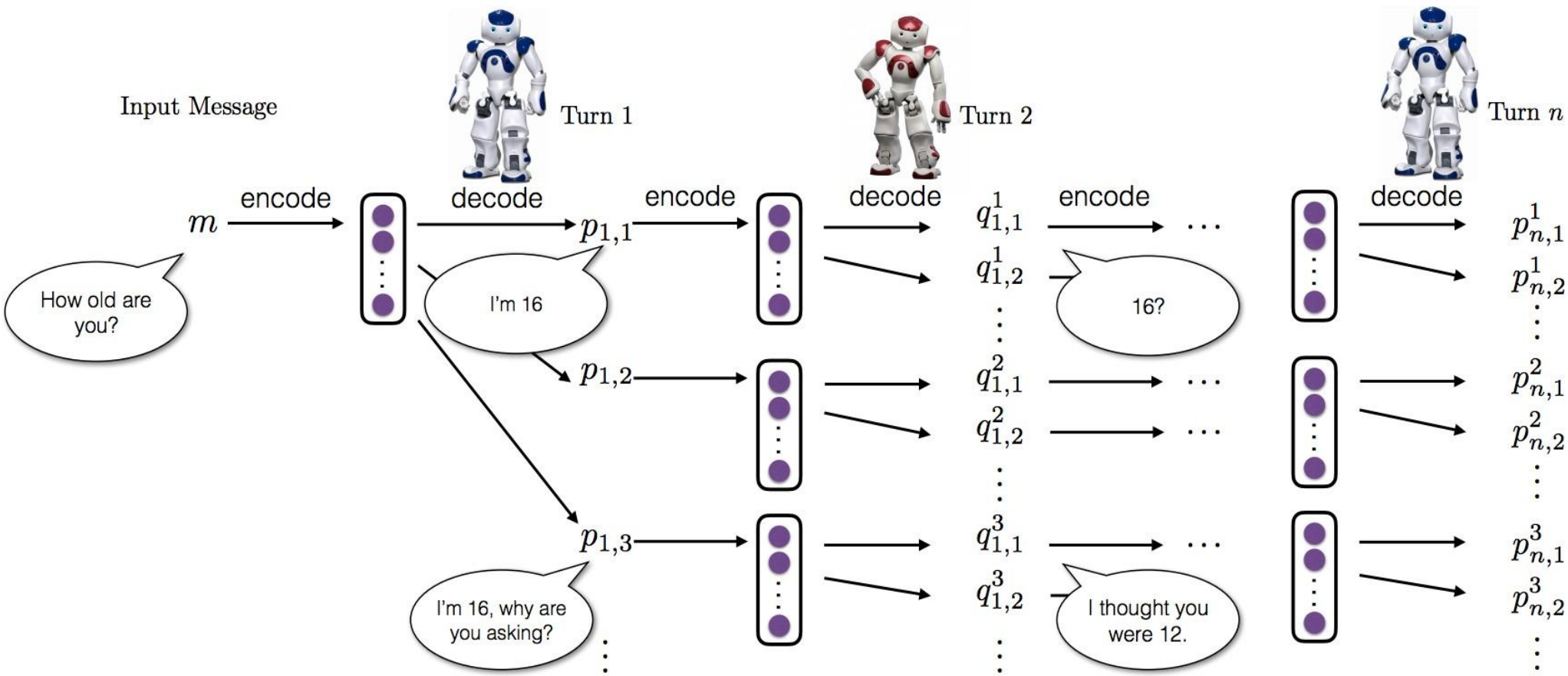
2) Mutual Information for pre-training policy model
   a) Initialize with model from (1)
   b) For each data point, there's a MLE loss, and a Mutual Information Score (reward)
   c) Use Curriculum Learning strategy, which optimizes for both

$$r_3 = \frac{1}{N_a} \log p_{\text{seq2seq}}(a|q_i, p_i) + \frac{1}{N_{q_i}} \log p^{\text{backward}}_{\text{seq2seq}}(q_i|a)$$

Dialogue generation

Dialogue Simulation between 2 agents

- Using the simulated turns and reward, maximize the expected future reward.



Dialogue generation

Results

Model	# of simulated turns
SEQ2SEQ	2.68
mutual information	3.40
RL	4.48

Setting	RL-win	RL-lose	Tie
single-turn general quality	0.40	0.36	0.24
single-turn ease to answer	0.52	0.23	0.25
multi-turn general quality	0.72	0.12	0.16

Input Message	Mutual Information Model	RL
How old are you?	I'm 16.	I'm 16, why are you asking?
What's your full name?	I have no idea.	What's yours?
I don't want to go home tonight.	Really?	Why?
Do you have any feelings for me?	I don't know what you are talking about.	I don't want to hurt your feelings.
How much time do you have here?	Not long enough. Sorry, sir.	Ten seconds.
Shall we get started?	Of course!	Yes. We've got a lot of work to do here.
Do you play football?	No, i don't	Yes. I love football!
We'd have to talk to him.	I mean, he's a good guy	About what ?
How come you never say it?	Because I don't want to hurt you.	I don't think it's a good idea to say it.

Dialogue generation

Annex 1a - Mutual Information reward derivation (part 1)

- Mutual Information for previous sequence S andresponse T

$$\log \frac{p(S,T)}{p(S)p(T)}$$

- MMI objective

$$\hat{T} = \arg\max_{T} \left\{ \log p(T|S) - \log p(T) \right\}$$

$$\hat{T} = \arg\max_{T} \left\{ \log p(T|S) - \lambda \log p(T) \right\}$$

λ : controls the penalization for generic response

Li et. al.

Dialogue generation

Annex 1b - Mutual Information reward derivation (part 2)

$$\hat{T} = \arg\max_{T} \{ \log p(T|S) - \lambda \log p(T) \}$$

$$\log p(T) = \log p(T|S) + \log p(S) - \log p(S|T)$$

$$\begin{aligned}\hat{T} &= \arg\max_{T} \{ (1-\lambda) \log p(T|S) \\ &\qquad + \lambda \log p(S|T) - \lambda \log p(S) \} \\ &= \arg\max_{T} \{ (1-\lambda) \log p(T|S) + \lambda \log p(S|T) \}\end{aligned}$$

Consider S as (q_i, p_i), T as a, we can have

$$r_3 = \frac{1}{N_a} \log p_{\text{seq2seq}}(a|q_i, p_i) + \frac{1}{N_{q_i}} \log p^{\text{backward}}_{\text{seq2seq}}(q_i|a)$$

Li et. al.

Summary

- The introduction of Reinforcement Learning
- Deep Policy Learning
- Deep Q-Learning
- Applications on NLP
  - Article summarization
  - Question answering
  - Dialogue generation