VL Deep Learning for Natural Language Processing

21. Deep Reinforcement Learning

Prof. Dr. Ralf Krestel

AG Information Profiling and Retrieval

So far… Supervised Learning

- **Data**: (x, y)
  - x is data, y is label
- **Goal**: Learn a **function** to map x -> y
- **Examples**:
  - classification,
  - regression,
  - object detection,
  - semantic segmentation,
  - Image captioning,
  - …

Cat

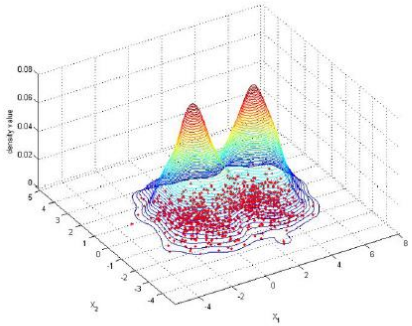
Classification

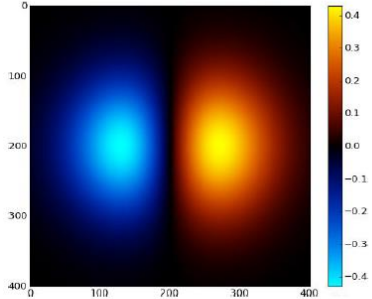
So far… Unsupervised Learning

- **Data**: x
  - Just data, no labels!
- **Goal**: Learn some underlying hidden **structure** of the data
- **Examples**:
  - Clustering,
  - dimensionality reduction,
  - feature learning,
  - density estimation,
  - …

Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation

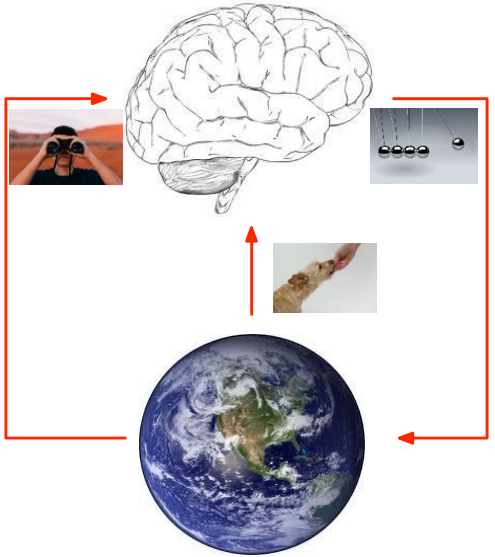




2-d density estimation

Today… Reinforcement Learning

- **Data**: (a, o, r)
  - a is an action, o is an observation, r is a reward
- **Goal**: Learn which **actions** to take to maximize **reward**
- **Examples**:
  - **Play** games:
    - Chess, Go, Atari, poker, …
  - **Explore** worlds:
    - 3D worlds, Labyrinth, ...
  - **Control** physical systems:
    - manipulate, walk, swim, ...
  - **Interact** with users:
    - recommend, optimize, personalize, ...



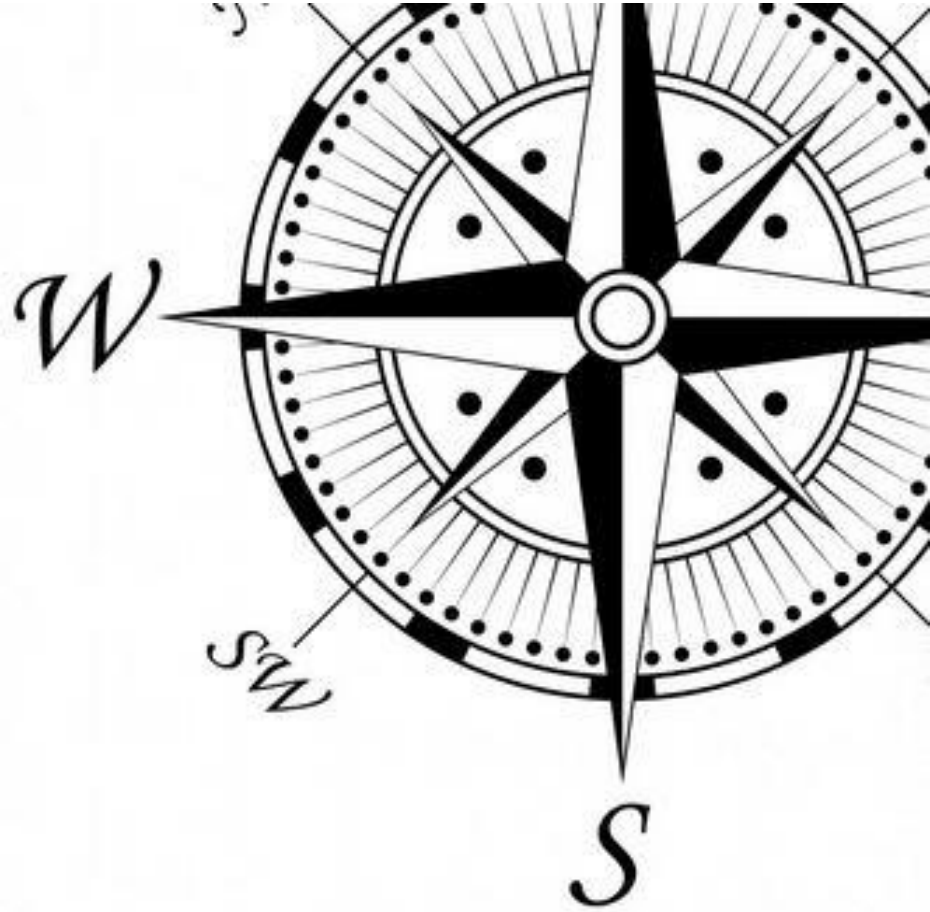
Learning Goals for this Chapter

- Understand a new type of machine learning problem and how it differs from supervised and unsupervised problems
- Explain deep reinforcement learning
- Know Q-Learning and its „deep“ brother (DQN)
- Understand how to apply DRL for NLP problems

- Related chapters:
  - MIT (2019): https://www.youtube.com/watch?v=zR11FLZ-O9M
  - S12 (2021): https://www.youtube.com/watch?v=1uMo8olr5ng

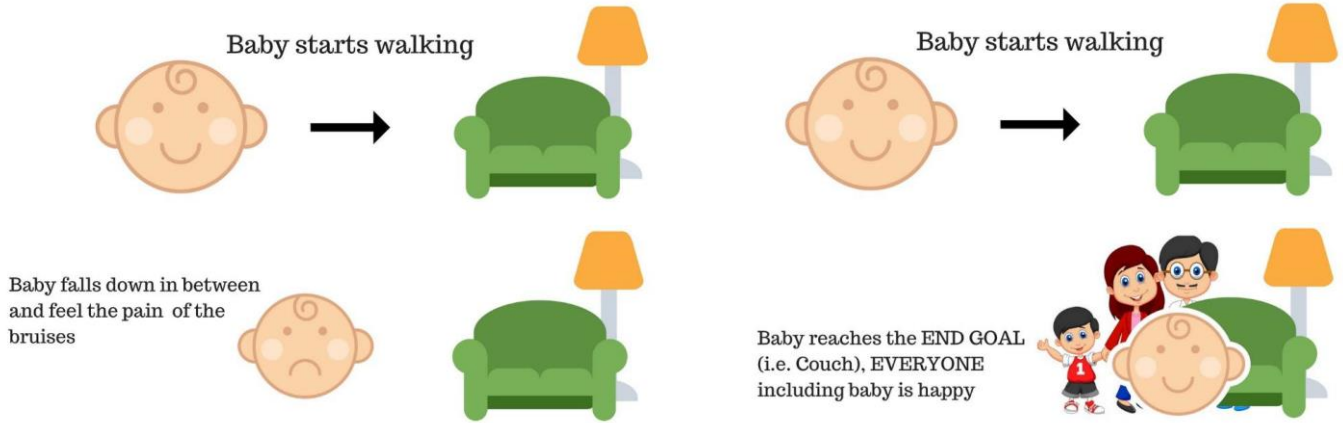
Topics Today

1. **Reinforcement Learning**
2. Deep Reinforcement Learning
3. DQN
4. DRL4NLP



Supervised vs Reinforcement Learning

- Supervised learning is “teach by **example**”: Here’s some examples, now learn patterns in these example.
- Reinforcement learning is “teach by **experience**”: Here’s a world, now learn patterns by exploring it.

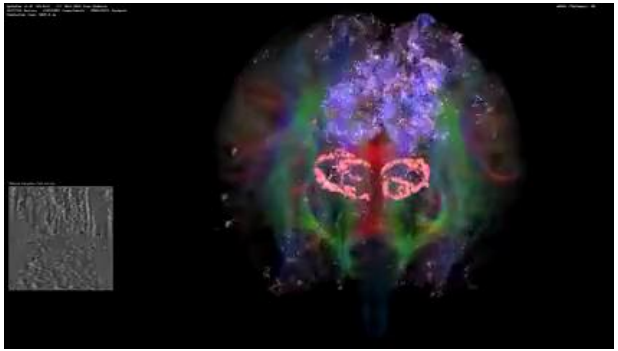


Failure **Success**

Human Learning

- Human appear to learn to walk through “very few examples” of trial and error. **How** is an open question…
- Possible answers:
  - **Training Data:** 230 million years of bipedal movement data.
  - **Imitation Learning:** Observation of other humans walking.
  - **Algorithms:** Better than backpropagation and stochastic gradient descent

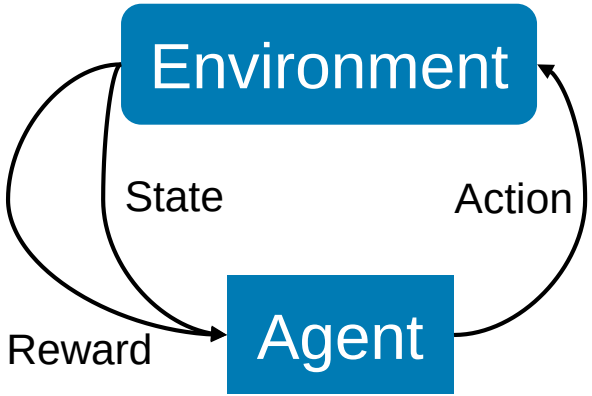
https://www.youtube.com/watch?v=B8-H4lRdGHI



https://www.youtube.com/watch?v=CKgxMu1oMOg

Reinforcement Learning Framework

- At each step, the agent:
  - Executes **action**
  - Observes new **state**
  - Receives **award**

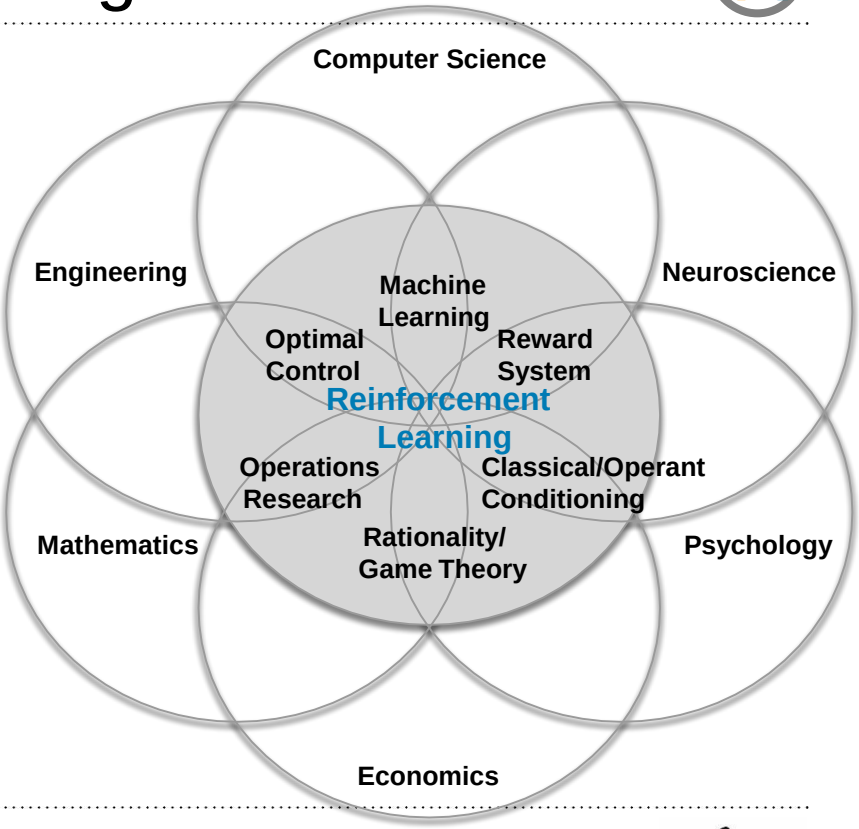


https://www.youtube.com/watch?v=Ipi19vExbzc

What is Reinforcement Learning?

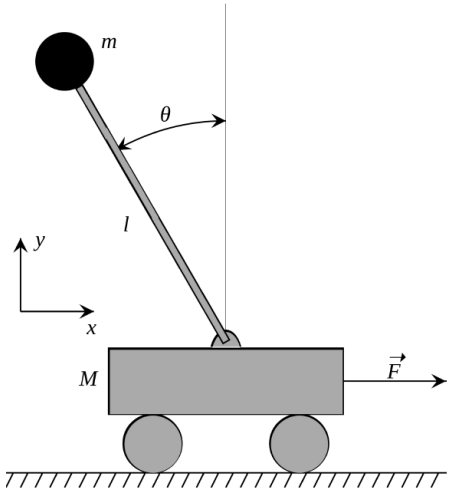
- RL is a general-purpose framework for decision-making
  - RL is for an **agent** with the capacity to **act**
  - Each **action** influences the agent's future **state** (observations)
  - Success is measured by a scalar **reward** signal

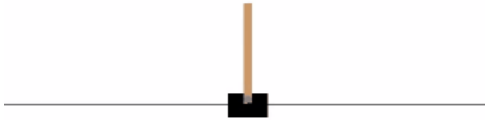
- Goal: **select actions to maximize future reward**



Cart-Pole Problem

- **Objective**: Balance a pole on top of a movable cart
- **State:** angle, angular speed, position, horizontal velocity
- **Action:** horizontal force applied on the cart
- **Reward:** 1 at each time step if the pole is upright



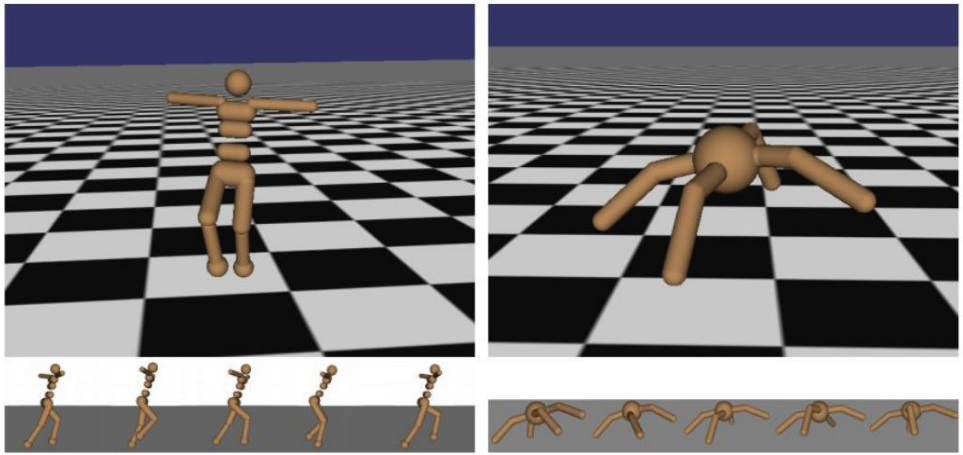


Doom

- **Objective**: Eliminate all opponents
- **State:** Raw game pixels of the game
- **Actions:** Up, Down, Left, Right, Shoot, etc.
- **Reward:** Positive when eliminating an opponent, negative when the agent is eliminated

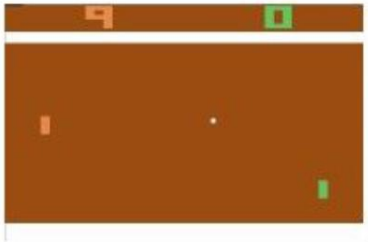
Robot Locomotion

- **Objective**: Make the robot move forward
- **State:** Angle and position of the joints
- **Action:** Torques applied on joints
- **Reward:** 1 at each time step upright + forward movement



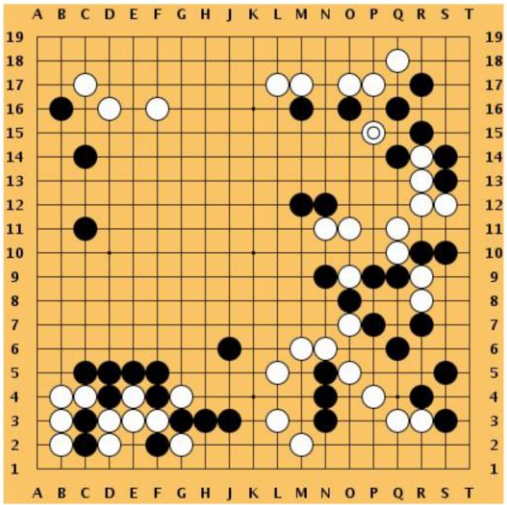
Atari Games

- **Objective**: Complete the game with the highest score
- **State:** Raw pixel inputs of the game state
- **Action:** Game controls e.g. Left, Right, Up, Down
- **Reward:** Score increase/decrease at each time step




Go

- **Objective**: Win the game!
- **State:** Position of all pieces
- **Action:** Where to put the next piece down
- **Reward:** 1 if win at the end of the game, 0 otherwise

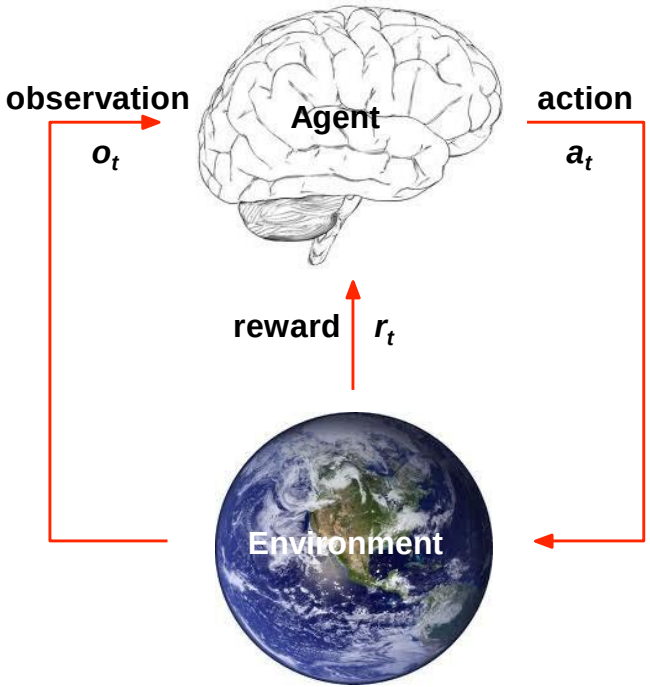


Human Life

- **Objective**: Survival? Happiness?
- **State:** Sight. Hearing. Taste. Smell. Touch.
- **Actions:** Think. Move.
- **Reward:** Homeostasis?

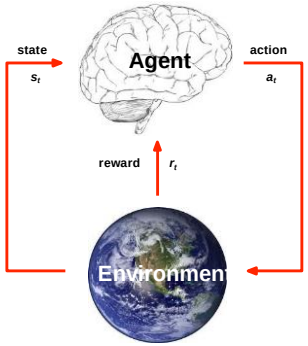
Mathematical Formalization

- At each step t the **agent**:
  - Executes action a_t
  - Receives observation o_t
  - Receives scalar reward r_t
- The **environment**:
  - Receives action a_t
  - Emits observation o_{t+1}
  - Emits scalar reward r_{t+1}



Markov Decision Process I

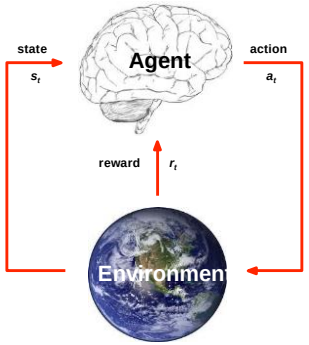
- Mathematical formulation of the RL problem
- **Markov property**: Current state completely characterizes the state of the world
- Defined by: (S, A, R, P, γ)
- S: set of possible states
- A: set of possible actions
- R: distribution of reward given (state, action) pair
- P: transition probability i.e. distribution over next state given (state, action) pair
- γ: discount factor



Markov Decision Process II

- At time step $t = 0$, environment samples initial state $s_0 \sim p(s_0)$
- Then, for $t = 0$ until done:
  - Agent selects action a_t
  - Environment samples reward $r_t \sim R(\,.\,|s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(\,.\,|s_t, a_t)$
  - Agent receives reward r_t and next state s_{t+1}

- A **policy** π a function from S to A that specifies what action to take in each state
- **Objective**: find policy π^* that maximizes cumulative discounted reward: $\sum_{t \geq 0} \gamma^t r_t$



Environment

- Fully Observable (Chess) vs Partially Observable (Poker)
- Single Agent (Atari) vs Multi Agent (DeepTraffic)
- Deterministic (Cart Pole) vs Stochastic (DeepTraffic)
- Static (Chess) vs Dynamic (DeepTraffic)
- Discrete (Chess) vs Continuous (Cart Pole)

Note: *Real-world environment might not technically be stochastic or partially-observable but might as well be treated as such due to their complexity.*

Reward

- **Future reward**: $R_t = r_t + r_{t+1} + r_{t+2} + \cdots + r_n$
- Discounted future reward: $R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-t} r_n$
- A good strategy for an agent would be to always choose an action that **maximizes the (discounted) future reward**
- Why “discounted”?
  - Math trick to help analyze convergence
  - Uncertainty due to environment stochasticity, partial observability, or that life can end at any moment:

Reward: Example

- Reward structure may have unintended consequences

Human

Deep RL Agent

- Player gets reward based on:
  1. Finishing time
  2. Finishing position
  3. Picking up “turbos”

https://openai.com/blog/faulty-reward-functions/

State

- Experience is a sequence of observations, actions, rewards

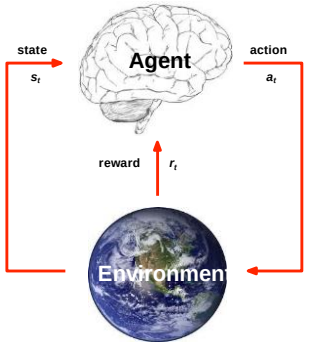
$$o_1, r_1, a_1, \ldots, a_{t-1}, o_t, r_t$$

- The **state** is a summary of experience

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

- In a fully observed environment

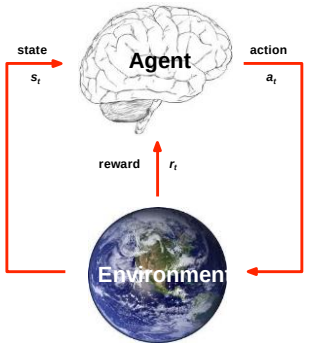
$$s_t = f(o_t)$$



Agent

Major Components of an RL Agent

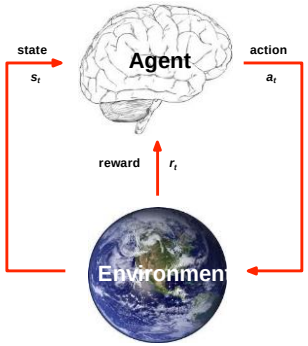
- An RL agent may include one or more of these components:
  - **Policy**: agent's behavior function
  - **Value function**: how good is each state and/or action
  - **Model**: agent's representation of the environment



- $s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_{n-1}, a_{n-1}, r_n, s_n$
  - s = state
  - a = action
  - r = reward
  - s_n = terminal state

RL Agent: Policy

- A **policy** is the agent's behavior
- It is a map from state to action:
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $\pi(a|s) = P[a|s]$



- **Optimal policy** π^* maximizes the sum of rewards

- Environment model has big impact on optimal policy
- Reward structure has big impact on optimal policy

RL Agent: Value Function

- The **value function** at state s, is the expected cumulative reward from following the policy from state s (**How good is a state?**):

$$V^{\pi}(s) = E\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$

- The **Q-value function** at state s and action a, is the expected cumulative reward from taking action a in state s and then following the policy (**How good is a state-action pair?**):
  - from state *s* and action *a*
    - under policy π
    - with discount factor γ

$$Q^{\pi}(s,a) = E\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right] = E[r_{t+1} + \gamma r_{t+2} + \gamma r_{t+3}^2 + \cdots | s, a]$$

- Value functions decompose into a Bellman equation

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

RL Agent: Optimal Value Functions

- An optimal value function is the maximum achievable value

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) = Q^{\pi^*}(s,a)$$

- Once we have Q^* we can act optimally,

$$\pi^*(s) = \underset{a}{arg\max}\, Q^*(s,a)$$

- Optimal value maximizes over all decisions. Informally:

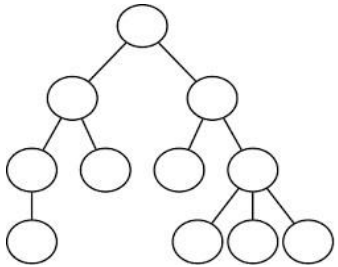
$$Q^*(s,a) = r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} r_{t+3} + \cdots$$
$$= r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})$$

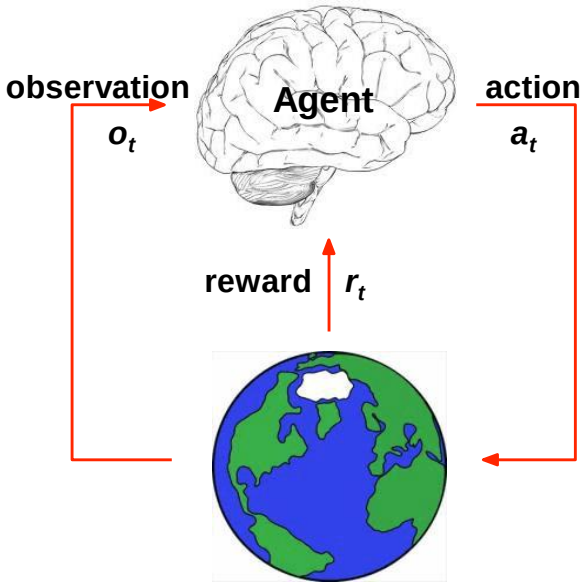
- Formally, optimal values decompose into a Bellman equation

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

RL Agent: Model

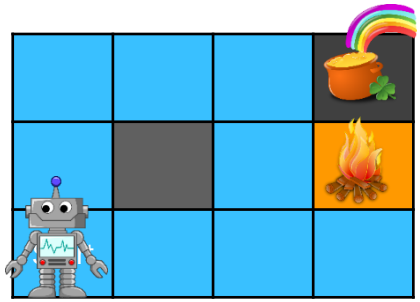
- **Model** is learnt from experience
- Acts as proxy for environment
- Planner interacts with model
- e.g. using lookahead search





Example: Robot in a Room I

- Or „Grid World“
- **Goal**: Reach a certain tile in the room with as few steps as possible
  - Optionally: avoid certain tiles
- **Actions**: UP, DOWN, RIGHT, LEFT
- **Reward**: +1 at [4,3]; -1 at [4,2];
  -0.04 for each step



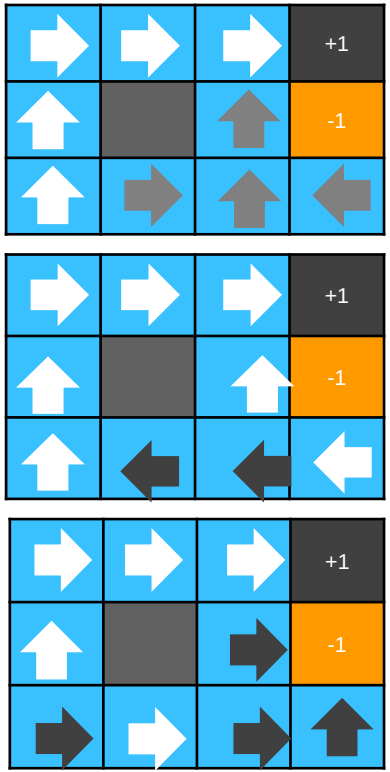
- What’s the **strategy** to achieve max reward?
  - We can learn the **model** and plan
  - We can learn the **value** of (action, state) pairs and act greed/non-greedy
  - We can learn the **policy** directly while sampling from it

Example: Robot in a Room II

- Optimal policy for a deterministic world
  - Reward: -0.04 for each step
  - Policy: shortest path

- Optimal policy for a stochastic world
  - Reward: **-0.04** for each step
  - E.g. if a=UP, result will be UP(80%), LEFT(10%), RIGHT(10%)
  - Policy: shortest path + avoid UP around -1 tile

- Optimal policy for a (another) stochastic world
  - Reward: **-2** for each step
  - E.g. if a=UP, result will be UP(80%), LEFT(10%), RIGHT(10%)
  - Policy: Shortest path



Approaches To Reinforcement Learning

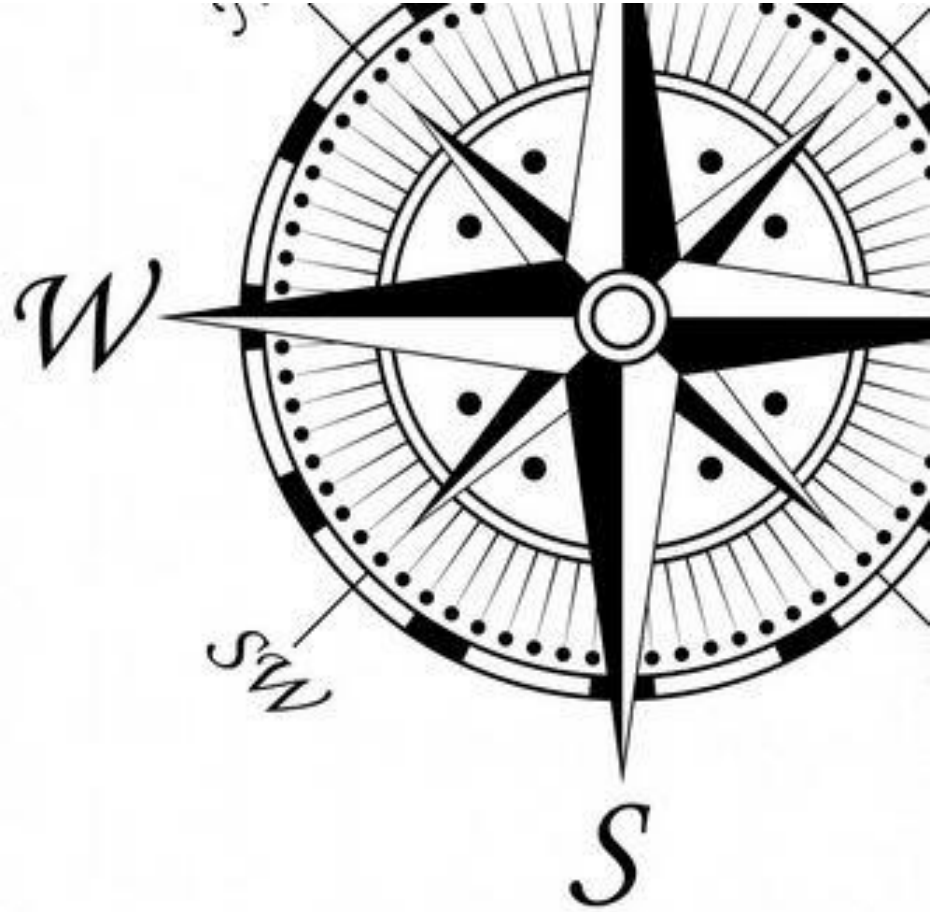
- **Value-based** RL
  - Estimate the **optimal value function** $Q^*(s, a)$
  - This is the maximum value achievable under any policy
  - Act by choosing best action in state
  - Exploration is a necessary add-on
- **Policy-based** RL
  - Search directly for the **optimal policy** π^*
  - This is the policy achieving maximum future reward
  - Act by sampling policy
  - Exploration is baked in
- **Model-based** RL
  - Build a model of the environment
  - Plan (e.g. by lookahead) using model
  - Update model often
  - Re-plan often

Model-free

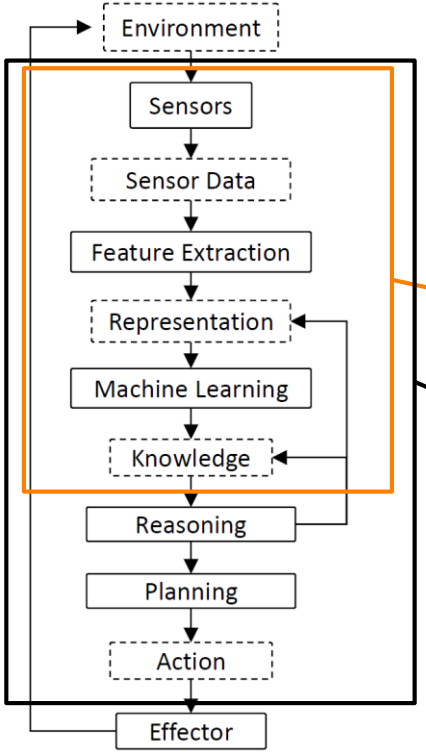
Model-based

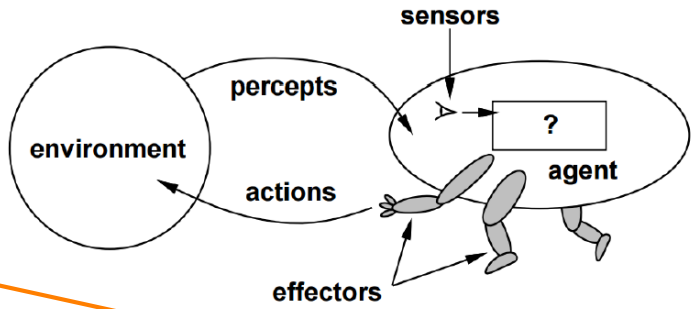
Topics Today

1. Reinforcement Learning
2. **Deep Reinforcement Learning**
3. DQN
4. DRL4NLP



What can be Learned from Data?

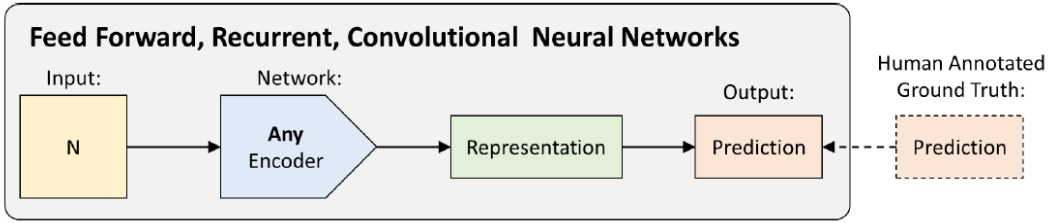




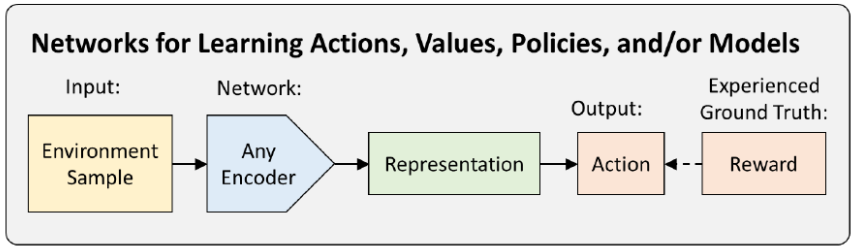
The promise of
Deep Learning

The promise of
Deep Reinforcement Learning

Deep Supervised vs DRL



- Supervised learning is “teach by **example**”: Here’s some examples, now learn patterns in these example.
- Reinforcement learning is “teach by **experience**”: Here’s a world, now learn patterns by exploring it.



Reinforcement Learning and Deep Learning

Reinforcement Learning

- RL is a general-purpose framework for decision-making
  - RL is for an **agent** with the capacity to **act**
  - Each **action** influences the agent's future **state**
  - Success is measured by a scalar **reward** signal
  - Goal: **select actions to maximise future reward**

Deep Learning

- DL is a general-purpose framework for representation learning
  - Given an **objective**
  - Learn **representation** that is required to achieve objective
  - Directly from **raw inputs**
  - Using minimal domain knowledge

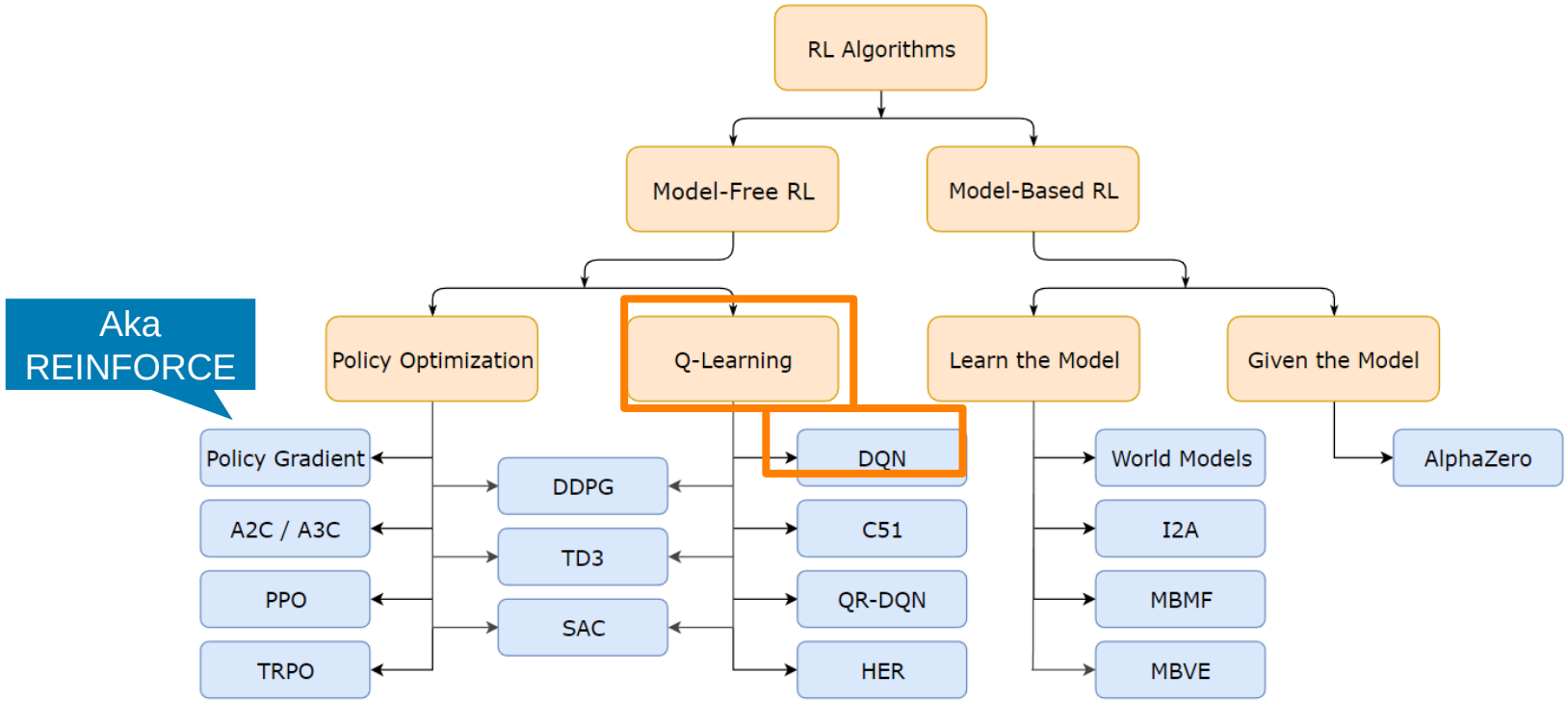
Deep Reinforcement Learning

- Train a single agent which can solve any human-level task
  - RL defines the **objective**
  - DL gives the **mechanism**
  - RL + DL = **general intelligence**

- Use deep neural networks to represent
  - **Value function**
  - **Policy**
  - **Model**

- Optimize loss function by stochastic gradient descent

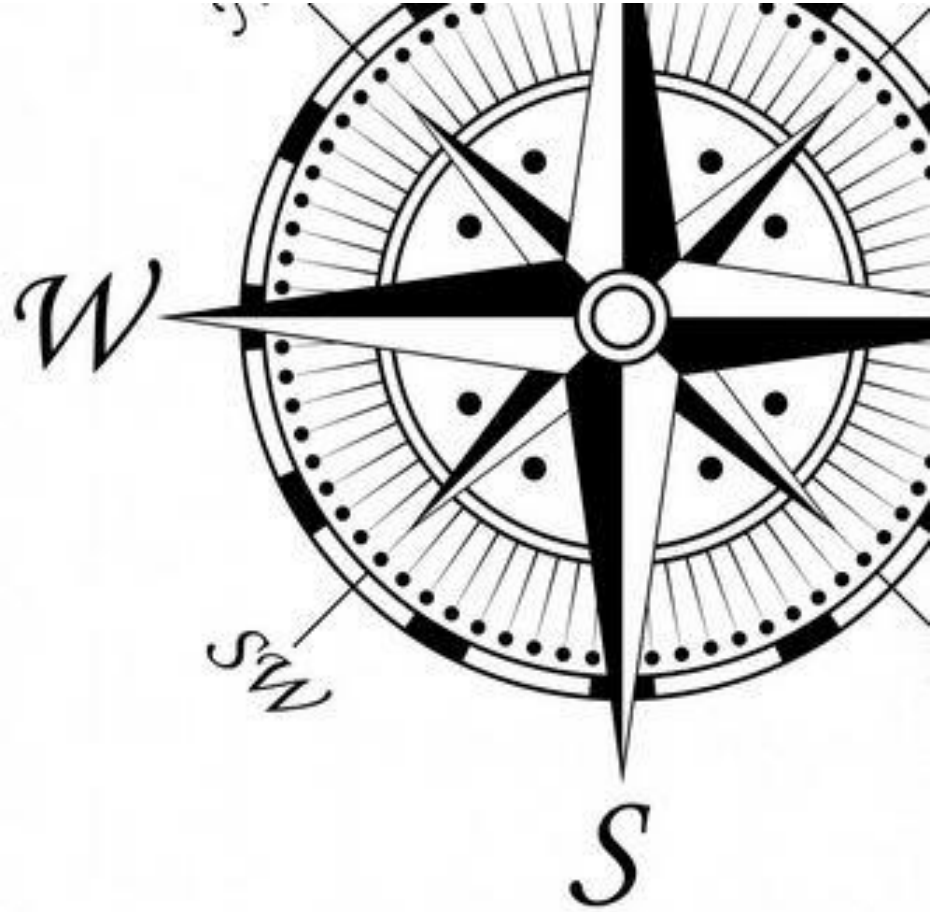
Taxonomy of RL Methods



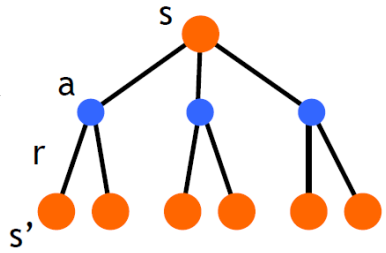
https://spinningup.openai.com

Topics Today

1. Reinforcement Learning
2. Deep Reinforcement Learning
3. **DQN**
4. DRL4NLP



Q-Learning



- State-action value function: $Q^{\pi}(s, a)$
  - Expected return when starting in *s*, performing *a*, and following π
- Q-Learning: Use **any policy** to estimate Q that maximizes future reward:
  - Q directly approximates Q* (Bellman optimality equation)
  - Independent of the policy being followed
  - Only requirement: keep updating each (s,a) pair

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left(R_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right)$$

Learning Rate

Discount Factor

New Value

Old Value

Reward

Estimate of Optimal Future Value

Exploration vs. Exploitation

- Deterministic/greedy policy won't explore all actions
  - Don't know anything about the environment at the beginning
  - Need to try all actions to find the optimal one
- **ε-greedy** policy
  - With probability 1-ε perform the optimal/greedy action, otherwise random action
  - Slowly move it towards greedy policy: ε -> 0

Q-Learning: Value Iteration

Learning Rate

Discount Factor

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left(R_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right)$$

New Value

Old Value

Reward

Estimate of Optimal Future Value

	A1	A2	A3	A4
S1	+1	+2	-1	0
S2	+2	0	+1	-2
S3	-1	+1	0	-2
S4	-2	0	+1	+1

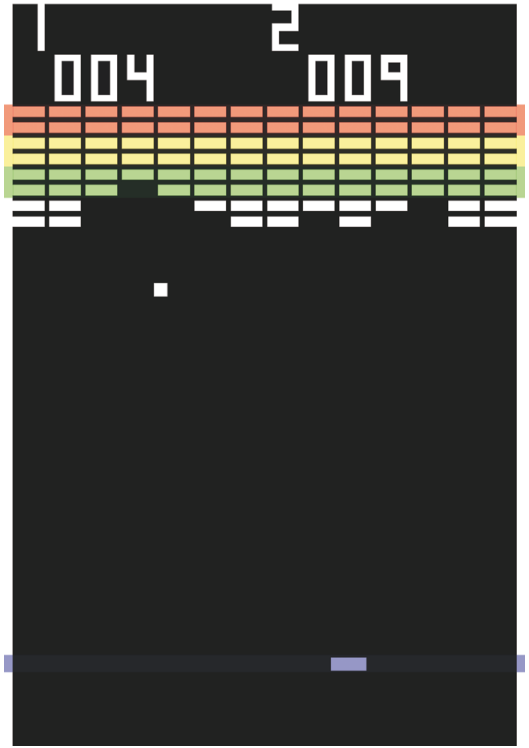
```
initialize Q[num_states,num_actions] arbitrarily
observe initial state s
repeat
      select and carry out an action a
      observe reward r and new state s'
      Q[s,a] = Q[s,a] + α(r + γ max_a' Q[s',a'] - Q[s,a])
      s = s'
until terminated
```

Q-Learning: Representation Matters

- In practice, value Iteration is impractical
  - Very limited states/actions
  - Cannot generalize to unobserved states

- Think about the **Breakout** game
  - State: screen pixels
    - Image size: $\mathbf{84 \times 84}$(resized)
    - Consecutive **4** images
    - Grayscale with **256** graylevels

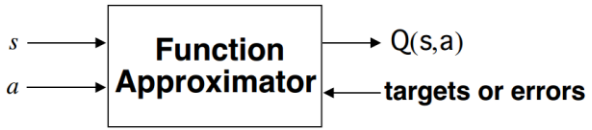
$\mathbf{256}^{\mathbf{84 \times 84 \times 4}}$ rows in the Q-table!
$= 10^{69,750} >> 10^{82}$ atoms in the universe

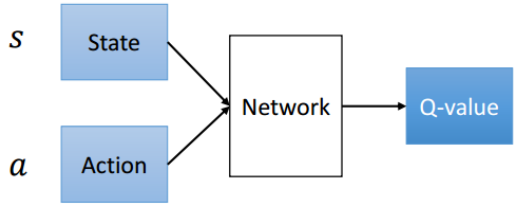


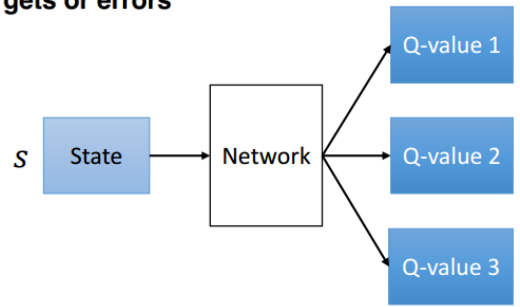
DQN: Deep Q-Network

- Use a neural network to approximate the Q-function:

$$Q(s, a; \theta) \approx Q^*(s, a)$$







- Loss Function $L = E\left[\left(r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right)^2\right]$

target prediction

DQN Tricks

- Experience Replay
  - Stores experiences (actions, state transitions, and rewards) and creates mini-batches from them for the training process
- Fixed Target Network
  - Error calculation includes the target function depends on network parameters and thus changes quickly. Updating it only every 1,000 steps increases stability of training process.

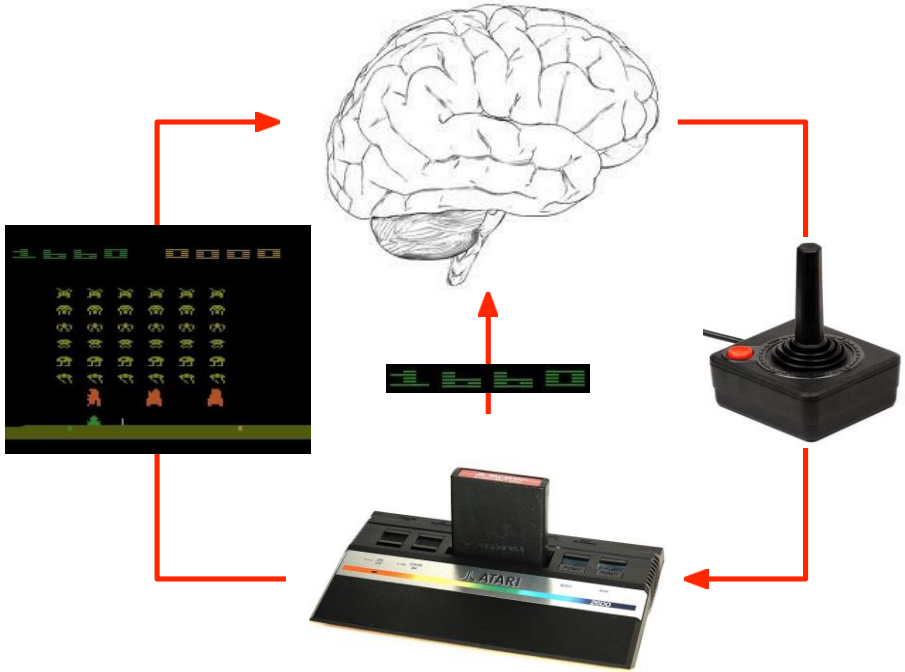
$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha \left[r_{t+1} + \gamma \max_p Q(s_{t+1}, p) - Q(s_t, a) \right]$$

target Q function in the red rectangular is fixed

Replay	○	○	×	×
Target	○	×	○	×
Breakout	**316.8**	240.7	10.2	3.2
River Raid	**7446.6**	4102.8	2867.7	1453.0
Seaquest	**2894.4**	822.6	1003.0	275.8
Space Invaders	**1088.9**	826.3	373.2	302.0

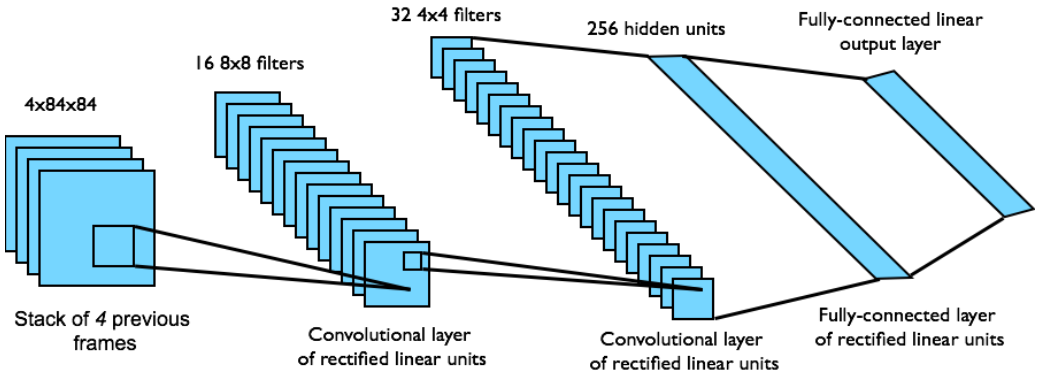
DQN Plays Breakout

DQN Plays Atari Games I

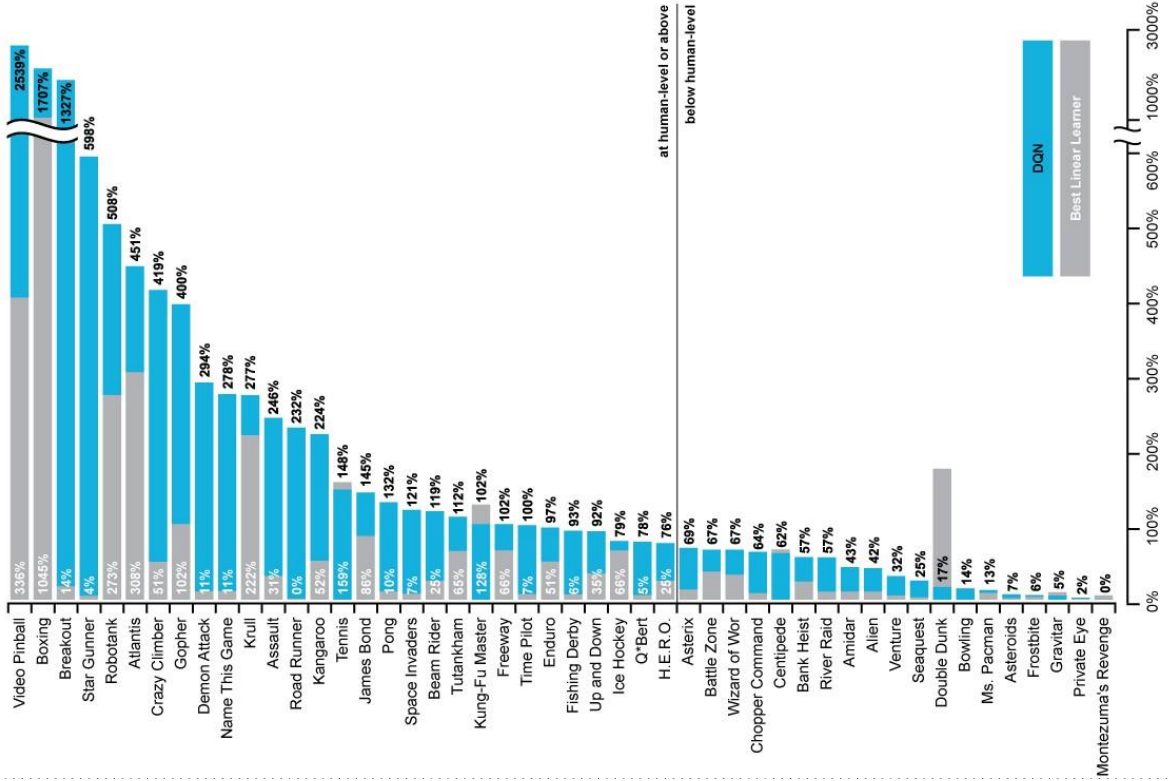


DQN Plays Atari Games II

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



DQN Plays Atari Games III

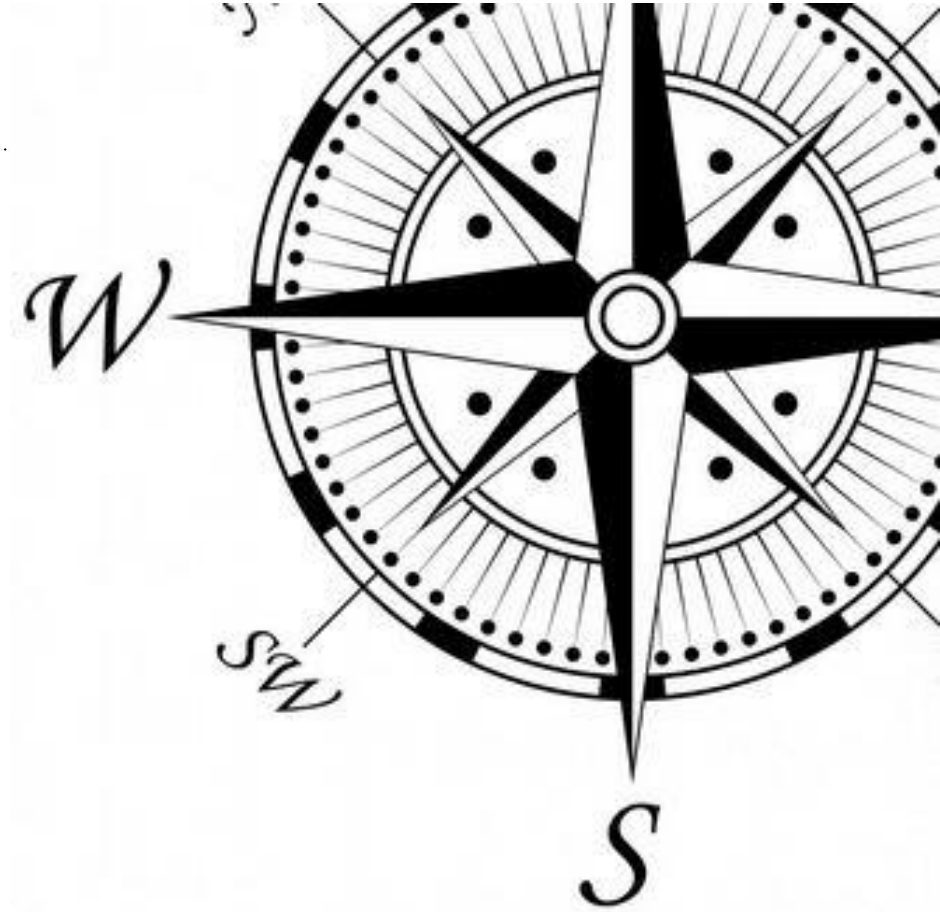


Reinforcement Learning

- Think of an application of RL in the area of Text Mining.
  - For which task involving text data, RL could be used?

Topics Today

1. Reinforcement Learning
2. Deep Reinforcement Learning
3. DQN
4. **DRL4NLP**



Top 5 NLP Applications Of RL

- **Policy Learning**
  - Train healthcare software policies to enhance patient outcomes
- **Dialogue Generation**
  - Important conversation attributes such as ease of answering, degree of semantic coherence, and the grade of informativity are maximized
- **Machine Translation**
  - Make accurate predictions about unseen, future portions of the incoming text
- **Text Summarization**
  - Attends over the input and spontaneously generates the abstract text summarization
- **Question Answering**
  - Examines several NLP-based reformulations of an initial question and aggregates the optimal answer

https://insights.daffodilsw.com/blog/top-5-nlp-applications-of-reinforcement-learning

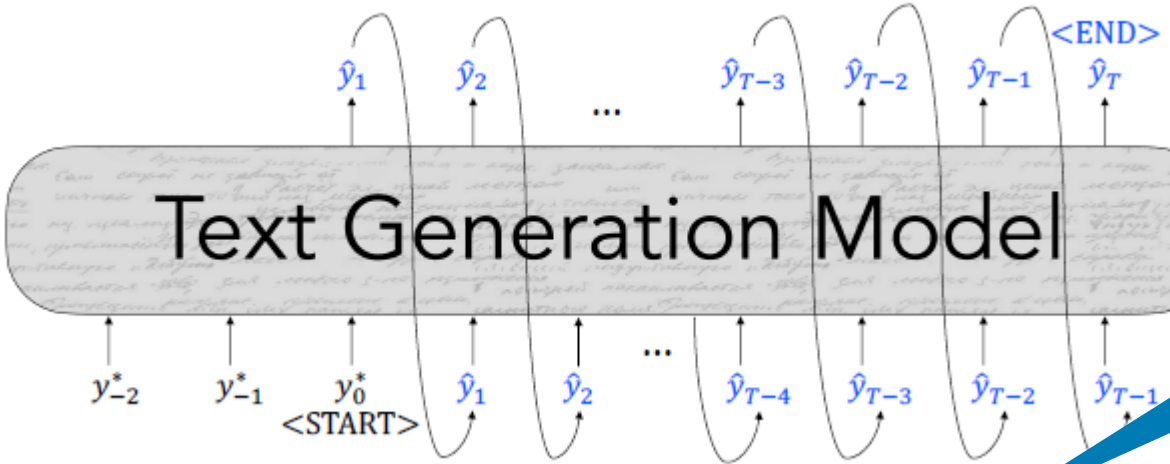
Text Generation as DRL Task

- Reinforcement Learning: cast your text generation model as a Markov decision process

- **State** *s* is the model's representation of the preceding context
- **Actions** *a* are the words that can be generated
- **Policy** π is the decoder (how to sample next token)
- **Rewards** *r* are provided by an external score
  - Learn behaviors by rewarding the model when it exhibits them

REINFORCE: Basics

- Sample a sequence from your model



$$L_{RL} = -\sum_{t=1}^{T} r(\hat{y}_t) \log P(\hat{y}_t | \{y^*\}; \{\hat{y}_t\}_{<t})$$

Next time, increase the probability of this sampled token in the same context..

…but do it more if I get a high reward from the reward function.

Reward Estimation

- How should we define a reward function? Just use your evaluation metric!
  - **BLEU** (machine translation; Ranzato et al., ICLR 2016; Wu et al., 2016)
  - **ROUGE** (summarization; Paulus et al, ICLR 2018; Celikyilmaz et al, NAACL 2018)
  - CIDEr (image captioning; Rennie et al., CVPR 2017)
  - SPIDEr (image captioning; Liu et al., ICCV 2017)

- Be careful about **optimizing for the task** as opposed to **"gaming" the reward**!
  - Evaluation metrics are merely proxies for generation quality!
  - **"even though RL refinement can achieve better BLEU scores, it barely improves the human impression of the translation quality"** – Wu et al., 2016

Reward Estimation

- What behaviors can we tie to rewards?
  - Cross-modality consistency in image captioning (Ren et al., CVPR 2017)
  - Sentence simplicity (Zhang and Lapata, EMNLP 2017)
  - Temporal Consistency (Bosselut et al., NAACL 2018)
  - Utterance Politeness (Tan et al., TACL 2018)
  - Paraphrasing (Li et al., EMNLP 2018)
  - Sentiment (Gong et al., NAACL 2019)
  - Formality (Gong et al., NAACL 2019)

- If you can formalize a behavior as a reward function (**or train a neural network to approximate it!**), you can train a text generation model to exhibit that behavior!

The Dark Side…

- Need to pretrain a model with **teacher forcing** before doing RL training
  - Your reward function probably expects coherent language inputs…

- Need to set an appropriate **baseline**:

$$L_{RL} = -\sum_{t=1}^{T} (r(\hat{y}_t) - \boldsymbol{B}) \log P(\hat{y}_t | \{y^*\}; \{\hat{y}_t\}_{<t})$$

  - Use linear regression to predict it from the state *s* (Ranzato et al., 2015)
  - Decode a second sequence and use its reward as the baseline (Rennie et al., 2017)

- Your model will learn the easiest way to exploit your reward function
  - Mitigate these shortcuts or hope that's aligned with the behavior you want!

Training DRL Takeaways

- **Teacher forcing** is still the premier algorithm for training text generation models
- **Diversity** is an issue with sequences generated from teacher forced models
  - New approaches focus on mitigating the effects of common words
- **Exposure bias** causes text generation models to **lose coherence** easily
  - Models must learn to recover from their own bad samples (e.g., scheduled sampling, DAgger)
  - Or not be allowed to generate bad text to begin with (e.g., retrieval + generation)
- Training with RL can allow models to learn behaviors that are challenging to formalize
  - Learning can be very **unstable**!

Learning Goals for this Chapter

- Understand a new type of machine learning problem and how it differs from supervised and unsupervised problems
- Explain deep reinforcement learning
- Know Q-Learning and its „deep“ brother (DQN)
- Understand how to apply DRL for NLP problems

- Related chapters:
  - MIT (2019): https://www.youtube.com/watch?v=zR11FLZ-O9M
  - S12 (2021): https://www.youtube.com/watch?v=1uMo8olr5ng

Resouces

- Lecture: Reinforcement Learning by Fei-Fei Li, Justin Johnson, Serena Yeung, Stanford, 2019
  - https://www.youtube.com/watch?v=lvoHnicueoE
  - http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture14.pdf
- Lecture: Introduction to Deep Reinforcement Learning by Lex Fridman, MIT, 2019
  - https://www.youtube.com/watch?v=zR11FLZ-O9M
  - https://www.dropbox.com/s/wekmlv45omd266o/deep_rl_intro.pdf?dl=0
- Tutorial: Deep Reinforcement Learning by David Silver, Google DeepMind, ICML, 2016
  - https://icml.cc/2016/tutorials/deep_rl_tutorial.pdf
- Coding
  - https://keon.io/deep-q-learning/
  - https://hub.packtpub.com/build-reinforcement-learning-agent-in-keras-tutorial/
  - https://github.com/keras-rl/keras-rl