Advanced Machine Learning
Lecture 19

Deep Reinforcement Learning

30.01.2017

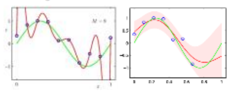
Bastian Leibe
RWTH Aachen
http://www.vision.rwth-aachen.de/

leibe@vision.rwth-aachen.de

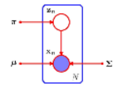
This Lecture: *Advanced Machine Learning*

$f: \mathcal{X} \rightarrow \mathbb{R}$

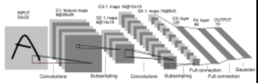
- Regression Approaches
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
  - Gaussian Processes



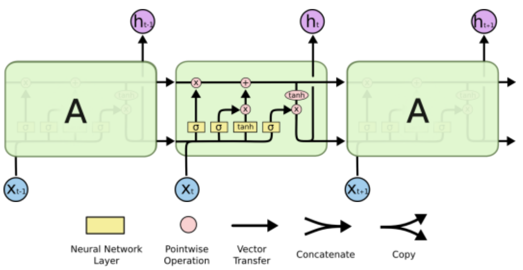
- Approximate Inference
  - Sampling Approaches
  - MCMC



- Deep Learning
  - Linear Discriminants
  - Neural Networks
  - Backpropagation & Optimization
  - CNNs, ResNets, RNNs, Deep RL, etc.



Recap: Long Short-Term Memory

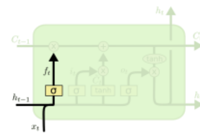


- LSTMs
  - Inspired by the design of memory cells
  - Each module has 4 layers, interacting in a special way.

Image source: Christopher Olah, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

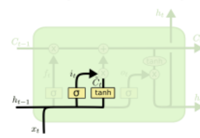
Recap: Elements of LSTMs

- Forget gate layer
  - Look at $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$ and output a number between 0 and 1 for each dimension in the cell state $\mathbf{C}_{t-1}$.
    - 0: completely delete this,
    - 1: completely keep this.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

- Update gate layer
  - Decide what information to store in the cell state.
  - Sigmoid network (input gate layer) decides which values are updated.
  - tanh layer creates a vector of new candidate values that could be added to the state.

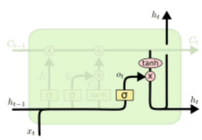


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Source: Christopher Olah, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recap: Elements of LSTMs

- Output gate layer
  - Output is a filtered version of our gate state.
  - First, apply sigmoid layer to decide what parts of the cell state to output.
  - Then, pass the cell state through a tanh (to push the values to be between -1 and 1) and multiply it with the output of the sigmoid gate.

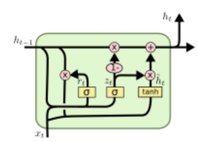


$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Source: Christopher Olah, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recap: Gated Recurrent Units (GRU)

- Simpler model than LSTM
  - Combines the forget and input gates into a single update gate z_t.
  - Similar definition for a reset gate r_t, but with different weights.
  - In both cases, merge the cell state and hidden state.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- Empirical results
  - Both LSTM and GRU can learn much longer-term dependencies than regular RNNs
  - GRU performance similar to LSTM (no clear winner yet), but fewer parameters.

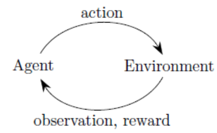
Source: Christopher Olah, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Topics of This Lecture

- Reinforcement Learning
  - Introduction
  - Key Concepts
  - Optimal policies
  - Exploration-exploitation trade-off
- Temporal Difference Learning
  - SARSA
  - Q-Learning
- Deep Reinforcement Learning
  - Value based Deep RL
  - Policy based Deep RL
  - Model based Deep RL
- Applications

Reinforcement Learning

- Motivation
  - General purpose framework for decision making.
  - Basis: Agent with the capability to interact with its environment
  - Each action influences the agent's future state.
  - Success is measured by a scalar reward signal.
  - Goal: select actions to maximize future rewards.



  - Formalized as a partially observable Markov decision process (POMDP)

Slide adapted from: David Silver, Sergey Levine

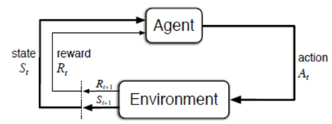
Reinforcement Learning

- Differences to other ML paradigms
  - There is no supervisor, just a reward signal
  - Feedback is delayed, not instantaneous
  - Time really matters (sequential, non i.i.d. data)
  - Agent's actions affect the subsequent data it receives

⇒ *We don't have full access to the function we're trying to optimize, but must query it through interaction.*

Slide adapted from: David Silver, Sergey Levine

The Agent-Environment Interface



- Let's formalize this
  - Agent and environment interact at discrete time steps $t = 0, 1, 2, \ldots$
  - Agent observes state at time t: $S_t \in \mathcal{S}$
  - Produces an action at time t: $A_t \in \mathcal{A}(S_t)$
  - Gets a resulting reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$
  - And a resulting next state: S_{t+1}

Slide adapted from: Sutton & Barto

Note about Rewards

- Reward
  - At each time step t, the agent receives a reward R_{t+1}
- Important note
  - We need to provide those rewards to truly indicate what we want the agent to accomplish.
  - E.g., learning to play chess:
    - The agent should only be rewarded for winning the game.
    - Not for taking the opponent's pieces or other subgoals.
    - Else, the agent might learn a way to achieve the subgoals without achieving the real goal.

⇒ *This means, non-zero rewards will typically be very rare!*

Reward vs. Return

- Objective of learning
  - We seek to maximize the expected return G_t as some function of the reward sequence $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$
  - Standard choice: expected discounted return

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

  where $0 \leq \gamma \leq 1$ is called the discount rate.

- Difficulty
  - We don't know which past actions caused the reward.

  ⇒ Temporal credit assignment problem

Markov Decision Process (MDP)

- Markov Decision Processes
  - We consider decision processes that fulfill the Markov property.
  - I.e., where the environments response at time t depends only on the state and action representation at t.
- To define an MDP, we need to specify
  - State and action sets
  - One-step dynamics defined by state transition probabilities

$$p(s'|s,a) = \Pr\{S_{t+1} = s' | S_t = s, A_t = a\} = \sum_{r \in \mathcal{R}} p(s', r | s, a)$$

  - Expected rewards for next state-action-next-state triplets

$$r(s,a,s') = \mathbb{E}[R_{t+1} | S_t = s, A_t = a, S_{t+1} = s'] = \frac{\sum_{r \in \mathcal{R}} r\, p(s', r|s,a)}{p(s'|s,a)}$$

Policy

- Definition
  - A policy determines the agent's behavior
  - Map from state to action $\pi: \mathcal{S} \rightarrow \mathcal{A}$
- Two types of policies
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $\pi(a|s) = \Pr\{A_t = a | S_t = s\}$
- Note
  - $\pi(a|s)$ denotes the probability of taking action a when in state s.

Value Function

- Idea
  - Value function is a prediction of future reward
  - Used to evaluate the goodness/badness of states
  - And thus to select between actions
- Definition
  - The value of a state s under a policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following π thereafter.

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right]$$

  - The value of taking action a in state s under a policy π, denoted $q_\pi(s,a)$, is the expected return starting from s, taking action a, and following π thereafter.

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right]$$

Bellman Equation

- Recursive Relationship
  - For any policy π and any state s, the following consistency holds

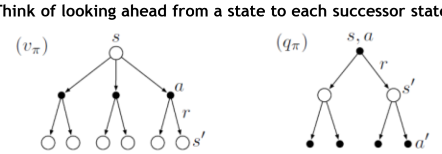
$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \middle| S_t = s\right] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s,a)\left[r + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \middle| S_{t+1} = s'\right]\right] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')], \qquad \forall s \in \mathcal{S} \end{aligned}$$

  - This is the Bellman equation for $v_\pi(s)$.

Bellman Equation

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')], \qquad \forall s \in \mathcal{S}$$

- Interpretation
  - Think of looking ahead from a state to each successor state.



  - The Bellman equation states that *the value of the start state must equal the (discounted) value of the expected next state, plus the reward expected along the way*.
  - We will use this equation in various forms to learn $v_\pi(s)$.

Optimal Value Functions

- For finite MDPs, policies can be partially ordered
  - There will always be at least one optimal policy π_*.
  - The optimal state-value function is defined as

$$v_*(s) = \max_\pi v_\pi(s)$$

  - The optimal action-value function is defined as

$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

Optimal Value Functions

- Bellman optimality equations
  - For the optimal state-value function v_*:

$$v_*(s) = \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a)$$

$$= \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]$$

  - v_* is the unique solution to this system of nonlinear equations.
  - For the optimal action-value function q_*:

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q_*(s',a')\right]$$

  - q_* is the unique solution to this system of nonlinear equations.

$\Rightarrow$ If the dynamics of the environment $p(s',r|s,a)$ are known, then in principle one can solve those equation systems.

Optimal Policies

- Why optimal state-value functions are useful
  - *Any policy that is greedy w.r.t. v_* is an optimal policy.*

$\Rightarrow$ Given v_*, one-step-ahead search produces the long-term optimal results.

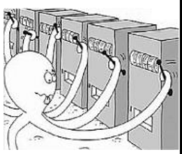
$\Rightarrow$ Given q_*, we do not even have to do one-step-ahead search

$$\pi_*(s) = \operatorname*{argmax}_{a \in \mathcal{A}(s)} q_*(s,a)$$

- Challenge
  - Many interesting problems have too many states for solving v_*.
  - Many Reinforcement Learning methods can be understood as approximately solving the Bellman optimality equations, using actually observed transitions instead of the ideal ones.

Exploration-Exploitation Trade-off

- Example: N-armed bandit problem
  - Suppose we have the choice between N actions $a_1, \ldots, a_N$.
  - If we knew their value functions $q_*(s, a_i)$, it would be trivial to choose the best.
  - However, we only have estimates based on our previous actions and their returns.



- We can now
  - Exploit our current knowledge
    - And choose the greedy action that has the highest value based on our current estimate.
  - Explore to gain additional knowledge
    - And choose a non-greedy action to improve our estimate of that action's value.

Image source: research.microsoft.com

Simple Action Selection Strategies

- ϵ-greedy
  - Select the greedy action with probability $(1-\epsilon)$ and a random one in the remaining cases.

$\Rightarrow$ In the limit, every action will be sampled infinitely often.

$\Rightarrow$ Probability of selecting the optimal action becomes $> (1-\epsilon)$.

  - But: many bad actions are chosen along the way.
- Softmax
  - Choose action a_i at time t according to the softmax function

$$\frac{e^{q_t(a_i)/\tau}}{\sum_{j=1}^{N} e^{q_t(a_j)/\tau}}$$

where τ is a temperature parameter (start high, then lower it).

  - Generalization: replace q_t by a preference function H_t that is learned by stochastic gradient ascent ("gradient bandit").

Topics of This Lecture

Temporal Difference Learning (TD-Learning)

- Policy evaluation (the prediction problem)
  - For a given policy π, compute the state-value function v_π.
- One option: Monte-Carlo methods
  - Play through a sequence of actions until a reward is reached, then backpropagate it to the states on the path.

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

Target: the actual return after time t

- Temporal Difference Learning - TD(λ)
  - Directly perform an update using the estimate $V(S_{t+\lambda+1})$.

$$V(S_t) \leftarrow V(S_t) + \alpha[\underbrace{R_{t+1} + \gamma V(S_{t+1})} - V(S_t)]$$

Target: an estimate of the return (here: TD(0))

SARSA: On-Policy TD Control

- **Idea**
  - Turn the TD idea into a control method by always updating the policy to be greedy w.r.t. the current estimate
- **Procedure**
  - Estimate $q_\pi(s,a)$ for the current policy π and for all states s and actions a.
  - TD(0) update equation

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

  - This rule is applied after every transition from a nonterminal state S_t.
  - It uses every element of the quintuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$.
  - ⇒ *Hence, the name SARSA.*

SARSA: On-Policy TD Control

- **Algorithm**

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

Q-Learning: Off-Policy TD Control

- **Idea**
  - Directly approximate the optimal action-value function q_*, independent of the policy being followed.
- **Procedure**
  - TD(0) update equation

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

  - Dramatically simplifies the analysis of the algorithm.
  - All that is required for correct convergence is that all pairs continue to be updated.

Q-Learning: Off-Policy TD Control

- **Algorithm**

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
    $s \leftarrow s'$;
  until s is terminal

Topics of This Lecture

- Reinforcement Learning
  - Introduction
  - Key Concepts
  - Optimal policies
  - Exploration-exploitation trade-off
- Temporal Difference Learning
  - SARSA
  - Q-Learning
- **Deep Reinforcement Learning**
  - Value based Deep RL
  - Policy based Deep RL
  - Model based Deep RL
- Applications

Deep Reinforcement Learning

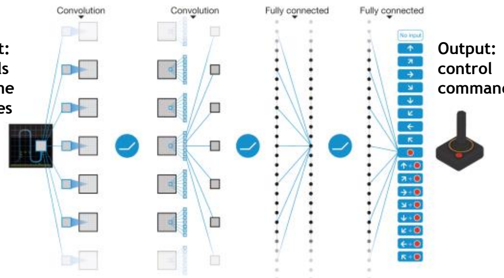
- **RL using deep neural networks to approximate functions**
  - Value functions
    - Measure goodness of states or state-action pairs
  - Policies
    - Select next action
  - Dynamics Models
    - Predict next states and rewards

Slide credit: Sergey Levine

Deep Reinforcement Learning

- Application: Learning to play Atari games

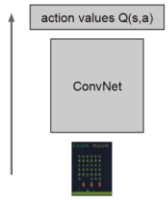
Input: pixels +game scores



Output: control commands

V. Mnih et al., Human-level control through deep reinforcement learning, Nature Vol. 518, pp. 529-533, 2015

Idea Behind the Model



- Interpretation
  - Assume finite number of actions
  - Each number here is a real-valued quantity that represents the Q function in Reinforcement Learning
- Collect experience dataset:
  - Set of tuples {(s,a,s',r), … }
  - (State, Action taken, New state, Reward received
- L2 Regression Loss

target value predicted value

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim \mathrm{U}(D)}\left[\left(\boxed{r+\gamma \max_{a'} Q(s',a';\theta_i^-)} - \boxed{Q(s,a;\theta_i)}\right)^2\right]$$

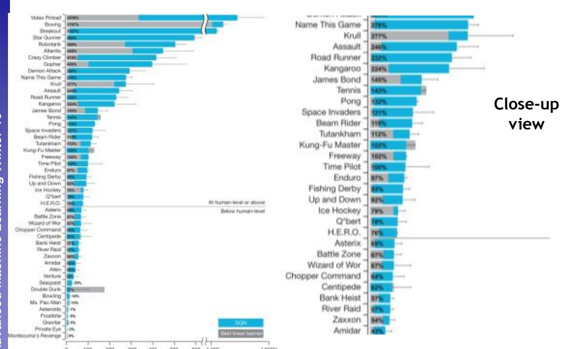
Current reward + estimate of future reward, discounted by γ

Slide credit: Andrej Karpaty

Results: Space Invaders

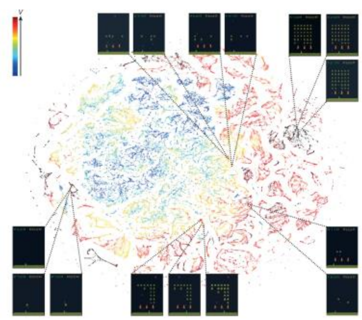
Results: Breakout

Comparison with Human Performance



Close-up view

Learned Representation



- t-SNE embedding of DQN last hidden layer (Space Inv.)

References and Further Reading

- More information on Reinforcement Learning can be found in the following book

Richard S. Sutton, Andrew G. Barto
Reinforcement Learning: An Introduction
MIT Press, 1998

- The complete text is also freely available online
  https://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html