Mini-project

Autonomously Learning Systems
708.062 KU

Anand Subramoney
Institut für Grundlagen der Informationsverarbeitung
Technische Universität Graz

Course WS 2016/17

Abstract

- Upload an abstract to courses-igi.tugraz.at
- Deadline: 16.12.2016

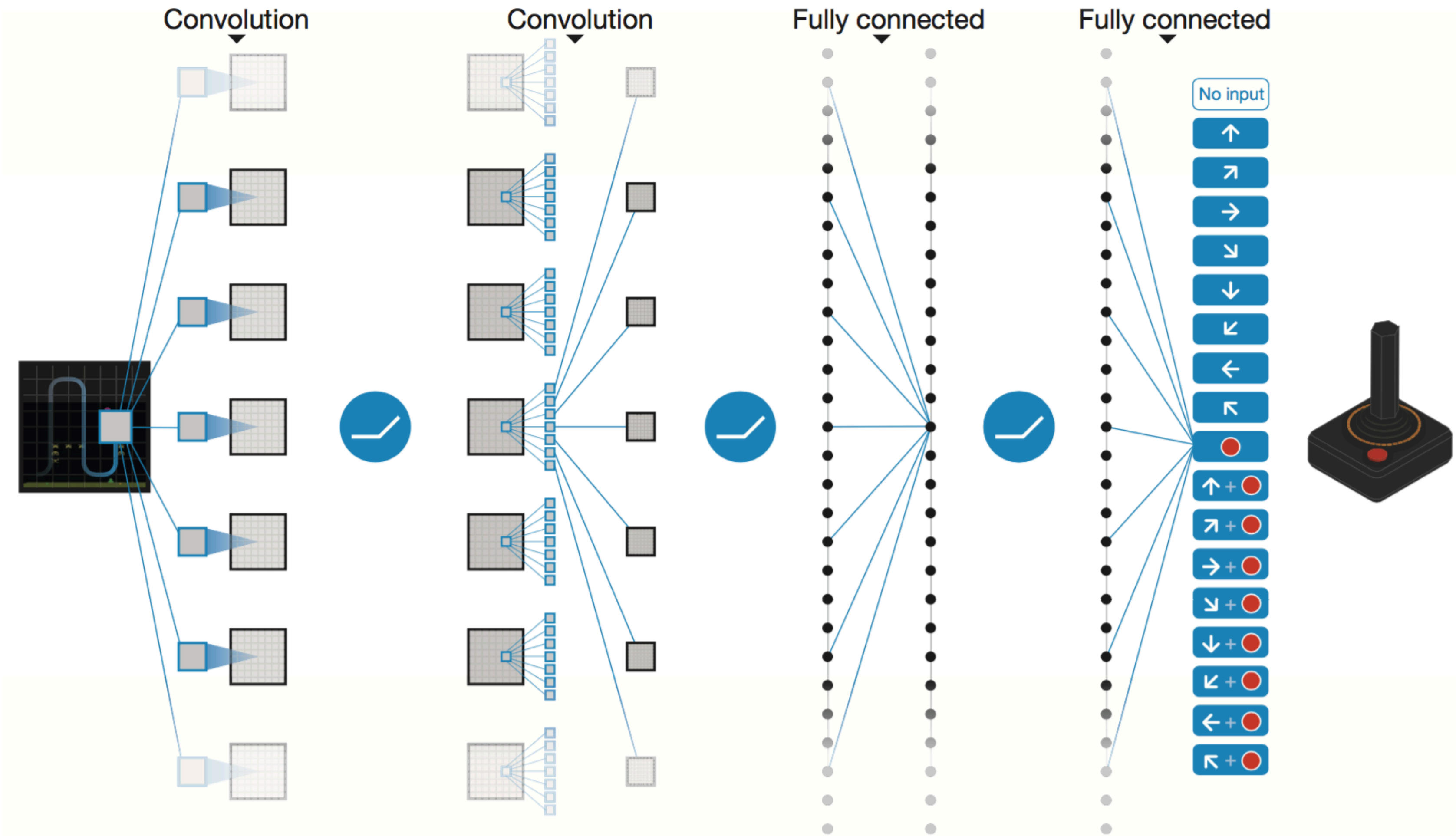
- Abstract should contain:
  - Name and Matrik. no. of team members
  - Description of the topic you plan to work on
  - Algorithm you’re going to implement
    - (can be more than one, simpler to complex versions)
  - Environment you’re going to use
    - (can be more than one, simpler to complex)
  - What is the goal of your study?

DQN

- Mnih, V. *et al.* Human-level control through deep reinforcement learning. *Nature* **518,** 529–533 (2015).

- Use a deep neural network (convolutional neural network) to estimate the Q-function
- Experience replay
- Target values $r + \gamma \max_{a'} Q(s', a'; \theta_i^-)$ calculated with an earlier version of parameters θ_i^-
  - θ_i^- is updated only every C steps

DQN schematic



Algorithm

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    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 D
    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Convolutional neural network

- Will be discussed in detail in the next class
- Some reading resources
  - http://neuralnetworksanddeeplearning.com/chap6.html
  - http://www.deeplearningbook.org/contents/convnets.html
  - https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

A3C

- Mnih, V. *et al.* Asynchronous Methods for Deep Reinforcement Learning. *arXiv:1602.01783 [cs]* (2016).

- A3C – Asynchronous Advantage Actor Critic

Asynchronous

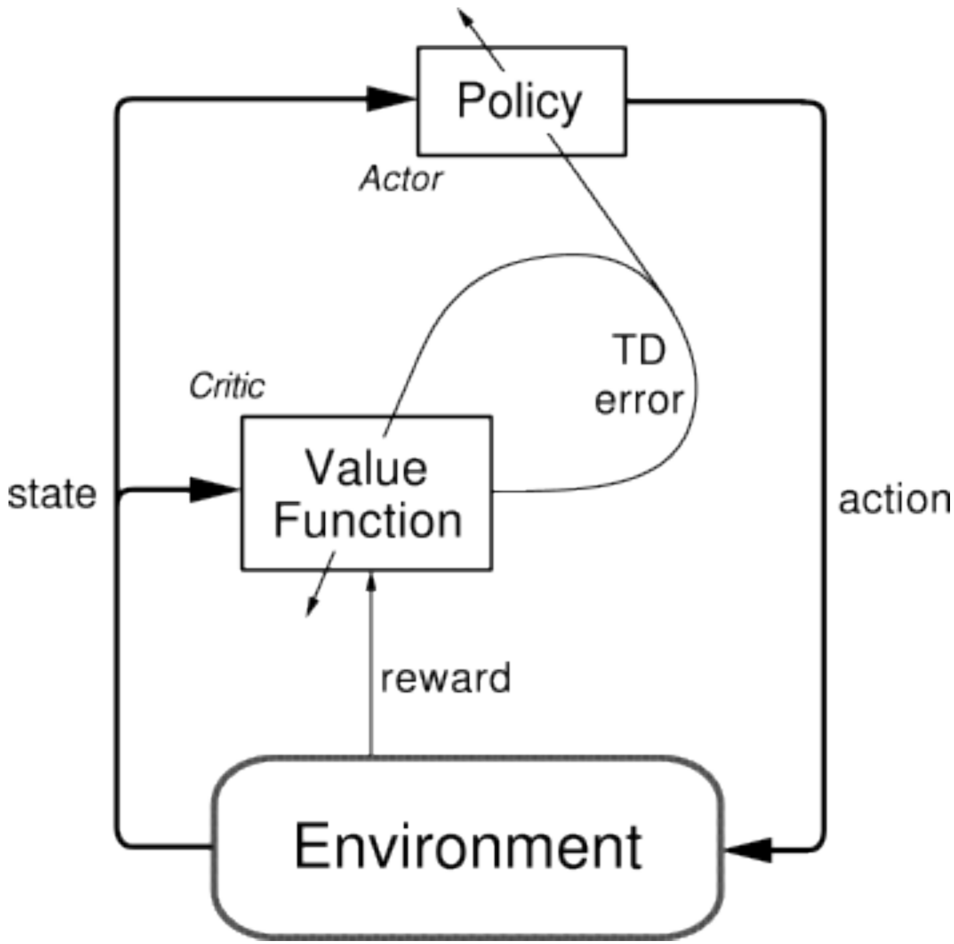
- To de-correlate updates:
  - Alternative to experience replay -- run multiple instances of the environment and agents in parallel
- The set of parameters θ is shared across agents, and updated asynchronously
- Each agent has it's own copy of parameters it uses and updates locally

Advantage

- Policy gradient with baseline
- But, baseline replaced with estimate of value function $b_t \approx V^\pi(s_t)$
- $R_t - b_t$ in this case known as the Advantage

- Advantage $A(a_t, s_t) = Q(a_t, s_t) - V(s_t)$
  - Since R_t is an estimate of $Q^\pi(a_t, s_t)$

Actor critic



Algorithm

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
    Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
    Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
    $t_{start} = t$
    Get state s_t
    repeat
        Perform a_t according to policy $\pi(a_t|s_t;\theta')$
        Receive reward r_t and new state s_{t+1}
        $t \leftarrow t + 1$
        $T \leftarrow T + 1$
    until terminal s_t **or** $t - t_{start} == t_{max}$
    $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
    for $i \in \{t-1, \ldots, t_{start}\}$ **do**
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
        Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial\left(R - V(s_i;\theta'_v)\right)^2/\partial\theta'_v$
    end for
    Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

Notes

- The *asynchronous* part and *actor-critic with advantage* part are independent

- First start with the *asynchronous* part with normal Q-learning or policy gradient
- Then use *actor-critic with advantage*

- The value function is learned in a separate network

- Multiple frames of images used for each step

Q EC

- Blundell, C. *et al.* Model-Free Episodic Control. *arXiv:1606.04460 [cs, q-bio, stat]* (2016).

- EC – Episodic Control
- Non-parameteric model that rapidly records and replays the sequence of actions that so far yielded the highest returns from a given start state

- Store $Q^{EC}(s, a)$ where each entry contains highest return ever obtained by taking action a from state s

Q EC

$$Q^{\mathrm{EC}}(s_t, a_t) \leftarrow \begin{cases} R_t & \text{if } (s_t, a_t) \notin Q^{\mathrm{EC}}, \\ \max\left\{Q^{\mathrm{EC}}(s_t, a_t), R_t\right\} & \text{otherwise,} \end{cases} \tag{1}$$

$$\widehat{Q^{\mathrm{EC}}}(s, a) = \begin{cases} \frac{1}{k}\sum_{i=1}^{k} Q^{\mathrm{EC}}(s^{(i)}, a) & \text{if } (s, a) \notin Q^{\mathrm{EC}}, \\ Q^{\mathrm{EC}}(s, a) & \text{otherwise,} \end{cases} \tag{2}$$

Algorithm

Algorithm 1 Model-Free Episodic Control.

for each episode **do**
    for $t = 1, 2, 3, \dots, T$ **do**
        Receive observation o_t from environment.
        Let $s_t = \phi(o_t)$.
        Estimate return for each action a via (2)
        Let $a_t = \arg\max_a \widehat{Q^{\mathrm{EC}}}(s_t, a)$
        Take action a_t, receive reward r_{t+1}
    end for
    for $t = T, T-1, \dots, 1$ **do**
        Update $Q^{\mathrm{EC}}(s_t, a_t)$ using R_t according to (1).
    end for
end for

Notes

- Remove least recently used updates in Q after a certain size
- Does not work in stochastic environments!
- Use a feature projection of $\phi \rightarrow \boldsymbol{Ax}$ where $A \in \mathbb{R}^{F \times D}$ and F << D, **A** is a random matrix drawn from a standard Gaussian
  - D is dimensionality of observation
  - F is dimensionality of smaller projection space

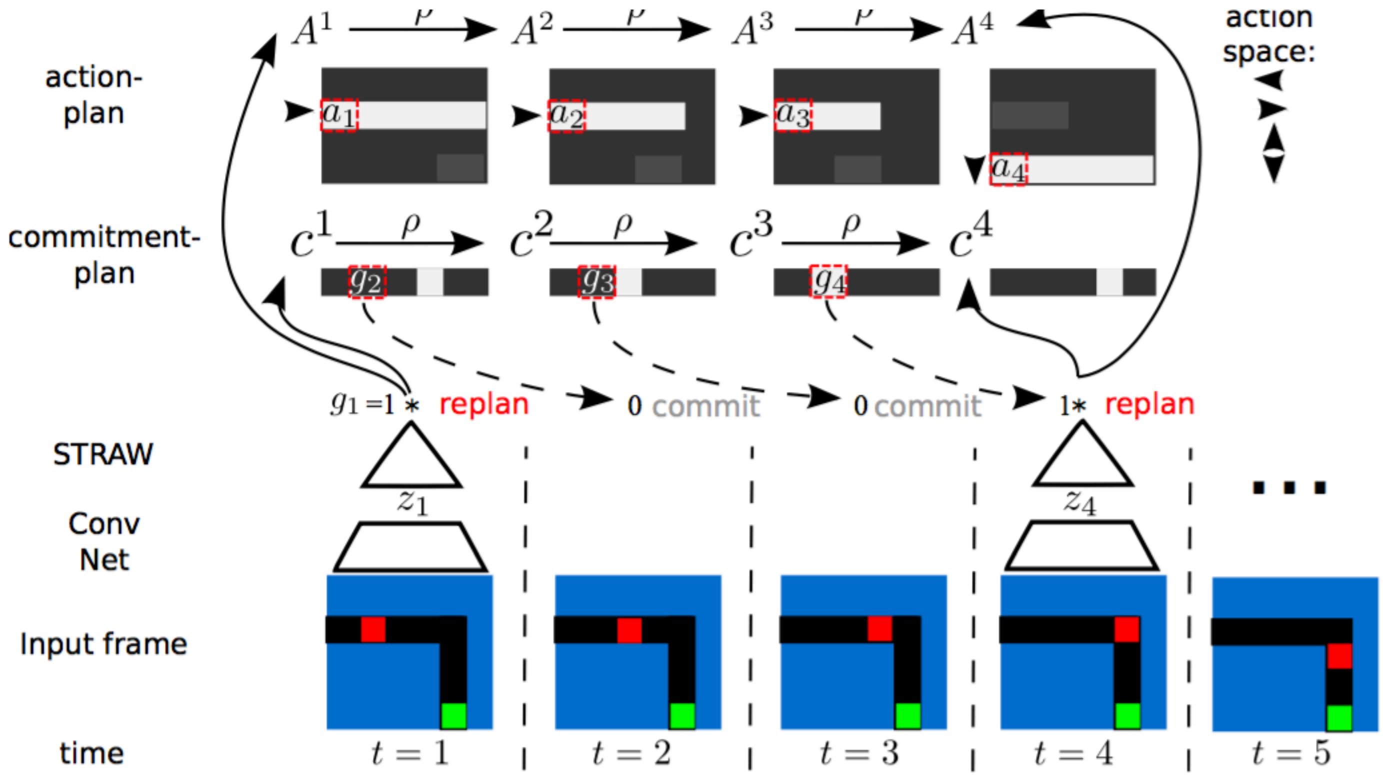
STRAW*

- Vezhnevets, A. *et al.* in *Advances in Neural Information Processing Systems 29* (eds. Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I. & Garnett, R.) 3486–3494 (Curran Associates, Inc., 2016).

- STRAW – STRategic Attentive Writer
- Learns to build implicit plans – learns macro actions of varying lengths solely from data

- Key idea
  - Output a sequence of actions at every time step and follow it till the end
  - Output when to change sequence of actions

STRAW



action-plan
commitment-plan
STRAW
Conv Net
Input frame
time
action space:
replan
0 commit
0 commit
replan

Simplified algorithm

A^t and c^t produced by deep CNN from observation x_t

For each step:

    Output A^t and c^t from DCNN
    Sample g^t from c^{t-k} (0 or 1)
    If g^t = 1
        Replace $\boldsymbol{A}$ with A^t
        Replace $\boldsymbol{c}$ with c^t
    endif
    Take action from $\mathbf{A}(:,0)$ and shift $\boldsymbol{A}$ and $\boldsymbol{c}$ left

    At end of episode (or batch):
        Train DCNN with policy gradient (or A3C)

Other

- **Talk to one of us before deciding to work on the following:**

- Model-based RL:
  - Gu,S.,Lillicrap,T.,Sutskever,I.& Levine,S. Continuous Deep Q-Learning with Model-based Acceleration. *arXiv:1603.00748 [cs]* (2016).
  - **Key idea:** use a simple linear model to learn the environment locally
- Neuroevolution
  - Stanley, K.O.& Miikkulainen, R. EvolvingNeuralNetworksthrough Augmenting Topologies. *Evolutionary Computation* **10,** 99–127 (2002).
  - **Key idea:** learn the structure of the network along with the weights with evolution
- Q(λ) forwards vs backwards view comparison on CartPole (for continuous state-space)
- Empirical study of baseline performance

General tips:

- Experiment on small and fast environments first (e.g. cartpole)
- Monitor weights and outputs of various layers
  - Esp. for CNN, make sure it works
- Don’t get stuck in parameter search!
  - You don’t have to get the best possible parameters, only one that works reasonably well
- If you’re stuck, talk to your teammates, or email us
- Have clearly defined, practical goals taking into account the limited computational resources available