ELEC-E8125 Reinforcement learning Function approximation
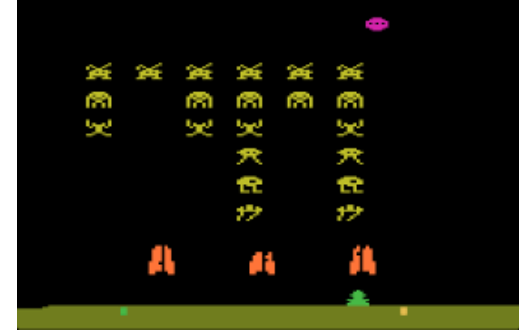
Ville Kyrki

29.9.2020

Today

- Function approximation for reinforcement learning.

Learning goals

- Understand basis and limitations of value function approximation.

- Understand incremental and batch approaches.

Motivation

- How to solve problems with large state spaces?

- For example:
  - Backgammon: ~10^{20} states.
  - Helicopter: continuous state space → infinite number of possible states.

- Value of each state can not be stored in memory.

- It is difficult to collect enough experience (too slow to learn each state independently).

Any other choices to represent V, Q?

Value function approximation
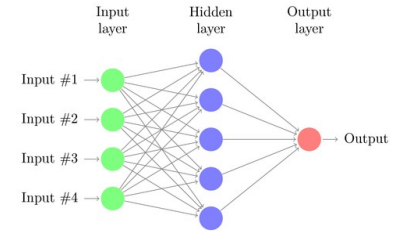
- Idea: Represent value function as a parametric approximation $\hat{V}(s, \boldsymbol{\theta})$, $\hat{Q}(s, a, \boldsymbol{\theta})$

  parameter vector

- Function approximator types:
  - Generalized linear $\hat{V}(s, \boldsymbol{\theta}) = \boldsymbol{\theta}^T \boldsymbol{\varphi}(s)$ $\hat{Q}(s, a, \boldsymbol{\theta}) = \boldsymbol{\theta}^T \boldsymbol{\varphi}(s, a)$
  - Neural network
  - Non-differentiable ones
    - e.g. decision tree, tiling

Features, for example
Radial basis function

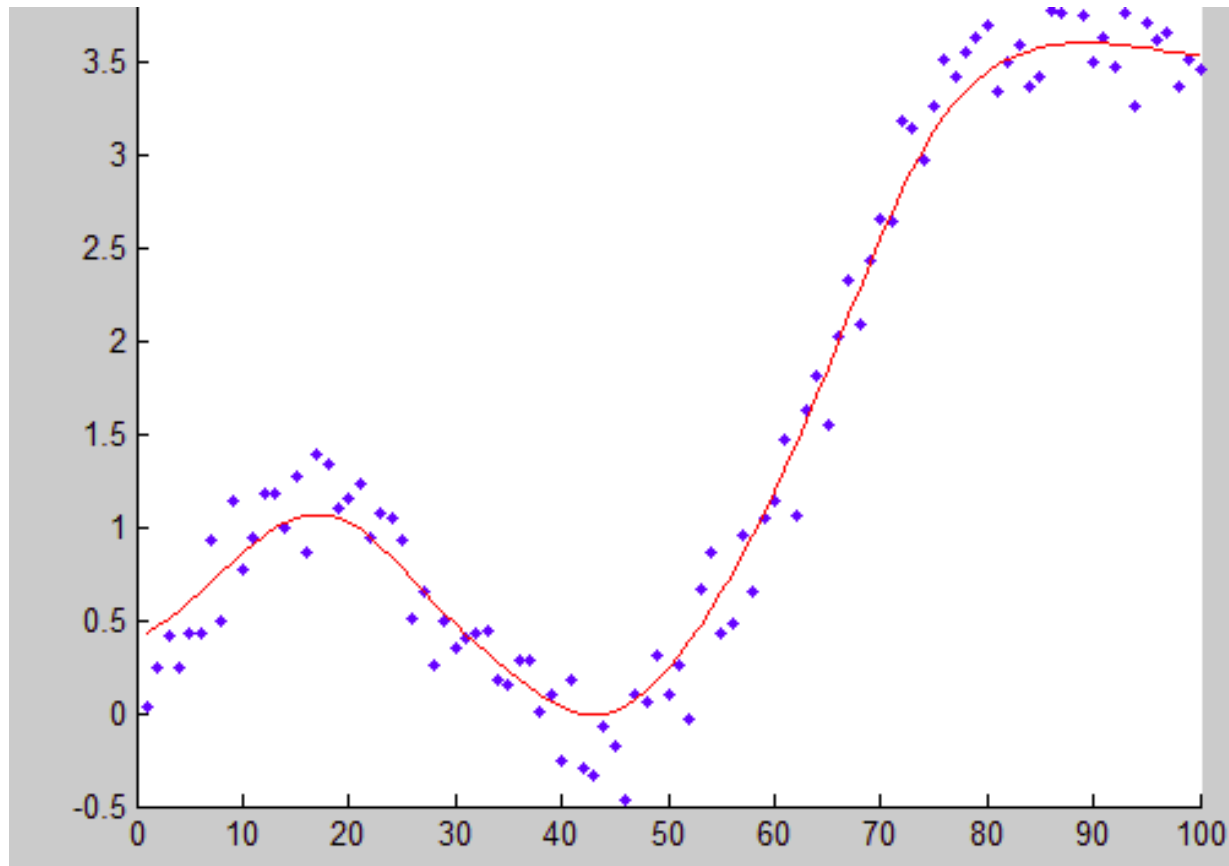$$\varphi_i(s) = e^{(s - s_i)^T \Sigma^{-1}(s - s_i)}$$

Tiling (grid)
Polynomial basis

Aalto University
School of Electrical
Engineering

How to *optimize* θ? Which criterion?
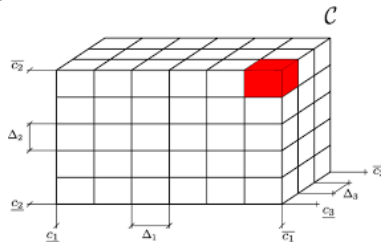
Example: Locally weighted regression

Yunyuongmok, 2014

Stochastic gradient descent

- Idea: Minimize mean-squares error in approximation

$$J(\boldsymbol{\theta}) = E\left[\left(V_\pi(s) - \hat{V}(s, \boldsymbol{\theta})\right)^2\right]$$

- Gradient descent update — Remember: $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \Delta\boldsymbol{\theta}$

$$\Delta\boldsymbol{\theta} = -\frac{1}{2}\alpha\nabla_\theta J(\boldsymbol{\theta})$$ ← Let's simplify!

- *Stochastic gradient descent* samples update

$$\Delta\boldsymbol{\theta} = \alpha\left(V_\pi(s) - \hat{V}(s, \boldsymbol{\theta})\right)\nabla_\theta \hat{V}(s, \boldsymbol{\theta})$$

Where to get the target value?

Incremental prediction

Estimate state-value function

- MC:

$$\Delta\boldsymbol{\theta} = \alpha\left(G_t - \hat{V}(s_t, \boldsymbol{\theta})\right)\nabla_\theta \hat{V}(s_t, \boldsymbol{\theta})$$

- TD(0):

Remember discrete TD(0): $V(s_t) = V(s_t) + \alpha\left(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right)$

$$\Delta\boldsymbol{\theta} = \alpha\left(r_{t+1} + \gamma\,\hat{V}(s_{t+1}, \boldsymbol{\theta}) - \hat{V}(s_t, \boldsymbol{\theta})\right)\nabla_\theta \hat{V}(s_t, \boldsymbol{\theta})$$

- TD(λ):

$$\Delta\boldsymbol{\theta} = \alpha\,\boldsymbol{E_t}\left(r_{t+1} + \gamma\,\hat{V}(s_{t+1}) - \hat{V}(s_t)\right)$$
$$\boldsymbol{E_t} = \gamma\,\lambda\,\boldsymbol{E_{t-1}} + \nabla_\theta \hat{V}(s_t, \boldsymbol{\theta})$$

vector

(Generalized) Linear function approximation

- Linear Monte-Carlo policy evaluation

$$\Delta \boldsymbol{\theta} = \alpha \left(G_t - \hat{V}(s_t, \boldsymbol{\theta}) \right) \nabla_\theta \hat{V}(s_t, \boldsymbol{\theta}) \quad \longleftarrow \text{ What is the gradient?}$$

$$= \alpha \left(G_t - \hat{V}(s_t, \boldsymbol{\theta}) \right) \boldsymbol{\varphi}(s_t)$$

  – Converges to local optimum.

- Linear TD(0)

$$\Delta \boldsymbol{\theta} = \alpha \left(r_{t+1} + \gamma \hat{V}(s_{t+1}, \boldsymbol{\theta}) - \hat{V}(s_t, \boldsymbol{\theta}) \right) \boldsymbol{\varphi}(s_t)$$

  – Converges on-policy to local optimum.

- Linear TD(λ)

$$E_t = \gamma \lambda E_{t-1} + \boldsymbol{\varphi}(s_t)$$

Convergence of prediction

	Algorithm	Discrete	Linear	Non-linear
On-policy	MC	+	+	+
	TD(0)	+	+	-
	TD(λ)	+	+	-
Off-policy	MC	+	+	+
	TD(0)	+	-	-
	TD(λ)	+	-	-

Gradient TD (non-linear GTD2, Maei 2009) converges off-policy with non-linear function approximation.

Incremental control

- Approach
  - Approximate policy evaluation for $\hat{Q}(s, a, \boldsymbol{\theta})$
  - ε-greedy policy improvement

- Policy evaluation for Q similar to V.
  - MC, TD

- SARSA and Q-learning also possible.

Approximation for action-value function

- Minimize MSE for $\hat{Q}(s, a, \boldsymbol{\theta})$.

- MC

$$\Delta\boldsymbol{\theta} = \alpha\left(G_t - \hat{Q}(s_t, a_t, \boldsymbol{\theta})\right)\nabla_\theta \hat{Q}(s_t, a_t, \boldsymbol{\theta})$$

- TD(0) / SARSA

$$\Delta\boldsymbol{\theta} = \alpha\left(r_{t+1} + \gamma\,\hat{Q}(s_{t+1}, a_{t+1}\boldsymbol{\theta}) - \hat{Q}(s_t, a_t, \boldsymbol{\theta})\right)\nabla_\theta \hat{Q}(s_t, a_t, \boldsymbol{\theta})$$

- TD(λ) / SARSA(λ)

$$\Delta\boldsymbol{\theta} = \alpha\,\boldsymbol{E_t}\left(r_{t+1} + \gamma\,\hat{Q}(s_{t+1}, a_{t+1}) - \hat{Q}(s_t, a_t)\right)$$

Convergence properties

Algorithm	Discrete	Linear	Non-linear
MC	+	(+)	-
SARSA	+	(+)	-
Q-learning	+	-	-

GQ(λ) (Maei&Sutton, 2010) convergent off-policy learning.

Batch prediction

- Sample efficiency important when few samples.

- Batch methods find single best fit for given data.

- One approach: Experience replay + stochastic gradient descent
  - Given data *D*, sample (state *s*,value *V(s)*) randomly and apply stochastic gradient descent update, repeat.
  $$\Delta\boldsymbol{\theta} = \alpha\left(V_\pi(s) - \hat{V}(s,\boldsymbol{\theta})\right)\nabla_\theta \hat{V}(s,\boldsymbol{\theta})$$
  - Converges to least-squares solution.

Not very efficient, convergence can take a long time.

Linear Least Squares for prediction

- With linear approximation, closed form solution available
- LSMC

$$E[\Delta\boldsymbol{\theta}] = \sum_{t=1}^{T} \alpha\left(G_t - \hat{V}(s_t, \boldsymbol{\theta})\right)\boldsymbol{\varphi}(s_t) = 0 \qquad \text{Solve!}$$

$$\boldsymbol{\theta} = \left(\sum_{t=1}^{T} \boldsymbol{\varphi}(s_t)\boldsymbol{\varphi}(s_t)^T\right)^{-1} \sum_{t=1}^{T} \boldsymbol{\varphi}(s_t) R_t$$

- LSTD

$$\boldsymbol{\theta} = \left(\sum_{t=1}^{T} \boldsymbol{\varphi}(s_t)\left(\boldsymbol{\varphi}(s_t) - \gamma\,\boldsymbol{\varphi}(s_{t+1})\right)^T\right)^{-1} \sum_{t=1}^{T} \boldsymbol{\varphi}(s_t) r_{t+1}$$

- LSTD(λ)

$$\boldsymbol{\theta} = \left(\sum_{t=1}^{T} \boldsymbol{E_t}\left(\boldsymbol{\varphi}(s_t) - \gamma\,\boldsymbol{\varphi}(s_{t+1})\right)^T\right)^{-1} \sum_{t=1}^{T} \boldsymbol{E_t} r_{t+1}$$

LSTDQ + LSPI

- Off-policy batch evaluation: LSTDQ

$$\theta = \left(\sum_{t=1}^{T} \varphi(s_t, a_t) \left(\varphi(s_t, a_t) - \gamma \, \varphi(s_{t+1}, \pi(s_{t+1}))\right)^T \right)^{-1} \sum_{t=1}^{T} \varphi(s_t, a_t) r_{t+1}$$

- Update policy to greedy. $\pi(s) = arg\,max_a \hat{Q}(s, a)$

- Repeat until (approximate) convergence.

Convergence of control

Algorithm	Discrete	Linear	Non-linear
MC control	+	(+)	-
SARSA	+	(+)	-
Q-learning	+	-	-
LSPI	+	(+)	

Example: Deep Q networks (Atari games, Mnih 2013, 2015)

- Learn *Q(s,a)* directly from pixels, output joystick/button position.

- Reward change in score.

- Approximate *Q* using a deep neural network.

- ε-greedy policy.

- Experience replay, optimize Q-network in LS sense using stochastic gradient descent variant.

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 $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$

    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 $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

$$
\text{Set } y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma\, \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}
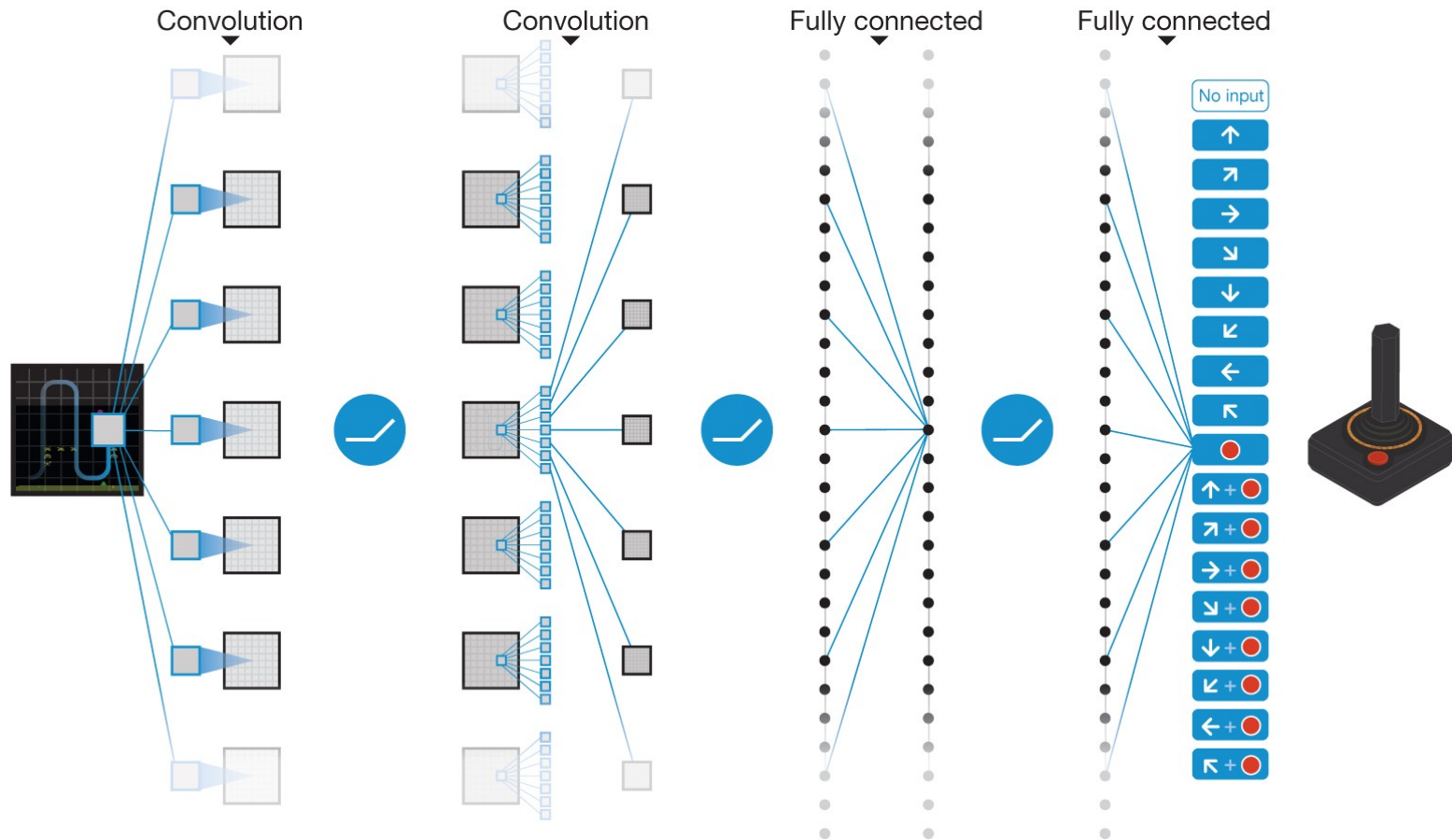$$

    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the

    network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

Schematic illustration of the convolutional neural network.

Aalto University
School of Electrical
Engineering

nature

Summary

- Value function approximation for large and continuous state-spaces.

- Convergence can be tricky especially for non-linear or off-policy cases.

Still limited action spaces. We'll later take a look at optimal control that works in continuous action space.

Next: Policy gradient and actor-critic approaches

- Do we need value functions?
  - Can we parametrize and optimize policy directly?

- Readings
  - Sutton&Barto Ch 13-13.3