Deep Learning Theory and Practice
Lecture 18
Deep Reinforcement Learning

Dr. Ted Willke
willke@pdx.edu

Monday, June 3, 2019
Review of Lecture 17

• **Learning long-term dependencies is difficult**
  - Vanishing and exploding gradient problem
  - Smaller weight given to long-term interactions

  The challenge of long-term dependencies

  - Little training success for sequences > 10-20 in length

• **Solution: Gated RNNs**
  - Control over timescale of integration of feedback
  - Eliminates repeated matrix multiplies

  Gated RNNs

  The most effective sequence models to date are gated RNNs.

  **Idea:** Create paths through time that have derivatives that neither vanish nor explode.
  - Can control time-scale of integration (forget gate)
  - Input feature computed with regular neuron
  - Accumulated into state if permitted (input gate)
  - Output of cell can be shut off (output gate)
Review of Lecture 17

• **Long Short-Term Memory**

  - Provides uninterrupted gradient flow
  - Solves the problem at the expense of more parameters

  **Long Short-Term Memory**

  Backpropagation from $c_i$ to $c_{i-1}$ does not involve matrix multiply by $W$, just element-wise multiplication by $f$.

  \[
  \begin{pmatrix}
  i \\
  f \\
  o \\
  g
  \end{pmatrix} =
  \begin{pmatrix}
  \sigma \\
  \sigma \\
  \sigma \\
  \tanh
  \end{pmatrix}
  \tanh W_i \begin{pmatrix}
  h_{i-1} \\
  h_i
  \end{pmatrix}
  \]

  \[W_i = [4n \times 2n]\]

  \[h_i = o \odot \tanh(c_i)\]

  \[c_i = f \odot c_{i-1} + i \odot g\]

  ( Hochreiter et al. 1997)

• **As revolutionary for sequential processing as CNNs were for spatial processing**

  **Challenging sequence processing tasks**

  - Toy problems: long sequence recall, long-distance interactions (math), classification and ordering of widely-separated symbols, noisy inputs, etc.
  - Real applications: natural machine translation, text-to-speech, music and handwriting generation

  Machine translation (English to French)

  Surprisingly good performance for long sentences. (Sutskever et al. 2014)
Today’s Lecture

• Deep Reinforcement Learning

(Some material drawn from Reinforcement Learning: An Introduction (2nd Ed.), by Sutton and Barto)
What is reinforcement learning?

What comes to mind when you think about how you learn?

We learn by interacting with our environment!

Learning what to do - how to map situations to actions - in order to maximize a reward signal
How does it work?

Agent

Observation, Reward

Environment

Action
What sets RL apart

Reinforcement learning is not **supervised** learning

- Generalizing from labeled data does not capture interaction
- Not practical to collect ‘correct’ examples
- **Agent must learn from its own experience**

Reinforcement learning is not **unsupervised** learning

- Not trying to find hidden structure in data
- **Agent must maximize a reward signal**
Increasingly better at complex games

Reinforcement learning with unsupervised auxiliary tasks
Jaderberg et al. (2016)

Learning objective:
\[
\arg \max_{\theta} \mathbb{E}_\pi[R_1:\infty] + \lambda_c \sum_{e \in C} \mathbb{E}_{\pi_c}[R_1:\infty] 
\]

Adjust a policy to maximize future reward

**Auxiliary tasks:**
- Maximally change pixel grid values
- Maximally activate each unit in a specific hidden layer
Learning dexterous in-hand manipulation

OpenAI et al. (2018)

Reward definition:

\[ r_t = d_t - d_{t+1}, \text{ where } d_t \text{ and } d_{t+1} \]

are desired vs current object orientation angles before and after transition

+5 for goal achievement. −20 if object dropped.
Emergence of cooperative agents

Human-level performance in 3D multiplayer games with population-based reinforcement learning
Jaderberg et al. (2019)

Agents that play on teams:

\[ J_{\text{inner}}(\pi_p|\omega_p) = \mathbb{E}_{m_p(\omega),\omega_\Omega} \mathbb{E}_{\tau_{\pi_p}} \left[ \sum_{t=1}^{T} \gamma^{t-1} w_p(\rho_{p,t}) \right] \quad \forall \pi_p \in \pi \]

\[ J_{\text{outer}}(w_p, \phi_p|\omega) = \mathbb{E}_{m_p(\omega),\omega_\Omega} \left[ \mathbb{P}_{\pi_p}^w,\phi_p \text{ wins} \right] \]

\[ \pi_p^w,\phi = \text{optimize}_{\pi_p}(J_{\text{inner}}, w, \phi) \]

Learned by RL

Learned by population-based training
The Markov decision process

**Markov decision processes** are a mathematically idealized form of the RL problem:
- Formalize sequential decision making
- Clear theory

Agent and environment interact at a sequence of discrete time steps: \( t = 0, 1, 2, 3, \ldots \)

At each time step \( t \), agent receives some environment state, \( S_t \in S \)

On this basis, agent selects an action, \( A_t \in \mathcal{A}(s) \) (potentially different action sets in different states)

One time step later, in part due to action, receives reward, \( R_{t+1} \in \mathcal{R} \subset \mathbb{R} \) and new state, \( S_{t+1} \)
The Markov decision process

The MDP and agent together give rise to a trajectory that begins as follows:

\[ S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots \]

In a finite MDP, the sets states, actions, and rewards (\( \mathcal{S}, \mathcal{A}, \mathcal{R} \)) all have a finite number of elements.

Random variables \( R_t, S_t \) will have discrete probability distributions dependent only on the preceding state and action.

That is, for a particular \( s' \in \mathcal{S}, r \in \mathcal{R} \),

\[ p(s', r \mid s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}, \text{ for all } s', s \in \mathcal{S}, r \in \mathcal{R}, \text{ and } a \in \mathcal{A}(s) \]
The Markov decision process

\[ p(s', r | s, a) \doteq \Pr\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\}, \text{ for all } s', s \in S, r \in R, \text{ and } a \in A(s) \]

This function defines the dynamics of the MDP.

Things to note:
- An ordinary deterministic function of four arguments: \( p : S \times R \times S \times A \rightarrow [0,1] \)
- \( p \) is a probability distribution for each \( s \) and \( a \): \( \sum_{s' \in S} \sum_{r \in R} p(s', r | s, a) = 1 \), for all \( s \in S, a \in A(s) \)

The Markov property:

The probability of each possible \( S_t \) and \( R_t \) depends only on \( S_{t-1} \) and \( A_{t-1} \), and not at all on earlier states and actions.
The Markov decision process

Nearly all reinforcement learning methods either assume the Markov property or use approximation methods.

We can compute a number of things from $p$:

\[ p(s'|s,a) \triangleq \Pr\{S_t = s'|S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r | s, a) \quad \text{(state transition probabilities)} \]

\[ r(s,a) \triangleq \mathbb{E}\left[ R_t | S_{t-1} = s, A_{t-1} = a \right] = \sum_{r \in \mathcal{R}} \sum_{s' \in \mathcal{S}} p(s', r | s, a) \quad \text{(expected reward for state-action)} \]

\[ r(s,a,s') \triangleq \mathbb{E}\left[ R_t | S_{t-1} = s, A_{t-1} = a, S_t = s' \right] = \sum_{r \in \mathcal{R}} r \frac{p(s', r | s, a)}{p(s'| s, a)} \quad \text{(expected reward for triples)} \]
The MDP framework

The MDP framework is abstract and flexible:

• ‘Time steps’ may refer to arbitrary successive stages of decision making

• ‘Actions’ can be low-level controls (voltage levels, etc.) or high-level decisions, like whether to learn a complex skill, like concert piano :-)

• ‘States’ can be simple sensor readings or a vector output from a complex computer vision model

• ‘States’ can be made of up memories and include complex internal agent states

• ‘Actions’ can be things the agent imagines or actually does

Anything that cannot be arbitrarily changed by the agent is considered outside of it and part of the environment.
Further reading


