Function/Image representation
Image classification
[Handwriting recognition]
Convolutional nets
Autoencoders
Visualization by dimensional reduction
Recurrent networks
Word vectors
Reinforcement learning
For more in-depth treatment, see David Silver’s course on reinforcement learning (University College London):

http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
A random walk, where the probability to go “up” is determined by the policy, and where the reward is given by the final position (ideal strategy: always go up!)

(Note: this policy does not even depend on the current state)
The simplest RL example ever

A random walk, where the probability to go “up” is determined by the policy, and where the reward is given by the final position (ideal strategy: always go up!)

(Note: this policy does not even depend on the current state)

\[
\pi_\theta(\text{up}) = \frac{1}{1 + e^{-\theta}}
\]

\[
R = x(T)
\]

RL update

\[
\Delta \theta = \eta \sum_t \left\langle R \frac{\partial \ln \pi_\theta(a_t)}{\partial \theta} \right\rangle
\]

\[
a_t = \text{up or down}
\]

\[
\frac{\partial \ln \pi_\theta(a_t)}{\partial \theta} = \pm e^{-\theta} \pi_\theta(a_t) = \pm (1 - \pi_\theta(a_t)) = 1 - \pi_\theta(\text{up}) \text{ for up}
\]

\[
-\pi_\theta(\text{up}) \text{ for down}
\]

\[
\sum_t \frac{\partial \ln \pi_\theta(a_t)}{\partial \theta} = N_{\text{up}} - N\pi_\theta(\text{up})
\]

\(N=\text{number of time steps}\)
The simplest RL example ever

**reward** \( R = x(T) = N_{\text{up}} - N_{\text{down}} = 2N_{\text{up}} - N \)

**RL update** \( \Delta \theta = \eta \sum_t \left< R \frac{\partial \ln \pi_\theta(a_t)}{\partial \theta} \right> \)

\( a_t = \text{up or down} \)

\[ \left< R \sum_t \frac{\partial \ln \pi_\theta(a_t)}{\partial \theta} \right> = 2 \left< (N_{\text{up}} - \frac{N}{2})(N_{\text{up}} - \bar{N}_{\text{up}}) \right> \]

(general analytical expression for average update, rare)

Initially, when \( \pi_\theta(\text{up}) = \frac{1}{2} \) : 

\[ \Delta \theta = 2\eta \left< (N_{\text{up}} - \frac{N}{2})^2 \right> = 2\eta \text{Var}(N_{\text{up}}) = \eta \frac{N}{2} > 0 \]

(binomial distribution!)
In general:

\[
\left\langle R \sum_t \frac{\partial \ln \pi_\theta(a_t)}{\partial \theta} \right\rangle = 2 \left\langle \left( N_{\text{up}} - \frac{N}{2} \right) (N_{\text{up}} - \bar{N}_{\text{up}}) \right\rangle \\
= 2 \left\langle \left( (N_{\text{up}} - \bar{N}_{\text{up}}) + (\bar{N}_{\text{up}} - \frac{N}{2}) \right) (N_{\text{up}} - \bar{N}_{\text{up}}) \right\rangle \\
= 2 \text{Var} N_{\text{up}} + 2(\bar{N}_{\text{up}} - \frac{N}{2}) \left\langle N_{\text{up}} - \bar{N}_{\text{up}} \right\rangle \\
= 2 \text{Var} N_{\text{up}} = 2N \pi_\theta(\text{up})(1 - \pi_\theta(\text{up}))
\]

(general analytical expression for average update, fully simplified, extremely rare)
The simplest RL example ever

probability $\pi_\theta (\text{up})$

trajectory ($=\text{training episode}$)

3 learning attempts, strong fluctuations!

(This plot for $N=100$ time steps in a trajectory; $\eta=0.001$)
Spread of the update step

\[ Y = N_{up} - \bar{N}_{up} \quad c = \bar{N}_{up} - N/2 \quad X = (Y + c)Y \]

(Note: to get \( \text{Var}(X) \), we need central moments of binomial distribution up to 4th moment)

\[ \sqrt{\text{Var}(X)} \sim N^{3/2} \]
\[ \langle X \rangle \sim N^1 \]
\[ \pi_\theta(\text{up}) \] (This plot for \( N=100 \))
Optimal baseline suppresses spread!

\[ Y = \bar{N}_{up} - \tilde{N}_{up} \quad c = \tilde{N}_{up} - N/2 \quad X = (Y + c)Y \]

with optimal baseline:

\[ X' = (Y + c - b)Y \quad b = \frac{\langle Y^2(Y + c) \rangle}{\langle Y^2 \rangle} \]

(This plot for \( N=100 \))
\[ M = \text{number of update steps} \]

\[ \Delta X = \sum_{j=1}^{M} X_j \]

\[ \langle \Delta X \rangle = M \langle X \rangle \]

\[ \sqrt{\text{Var}\Delta X} = \sqrt{M} \sqrt{\text{Var}X} \]

relative spread

\[ \frac{\sqrt{\text{Var}\Delta X}}{\langle \Delta X \rangle} \sim \frac{1}{\sqrt{M}} \]
Homework

Implement the RL update including the optimal baseline and run some stochastic learning attempts. Can you observe the improvement over the no-baseline results shown here?

Note: You do not need to simulate the individual random walk trajectories, just exploit the binomial distribution.
The second-simplest RL example

actions: move or stay

“walker”

reward = number of time steps on target

See code on website: “SimpleRL_WalkerTarget”
output = action probabilities (softmax)
\[ \pi_{\theta}(a|s) \]

a=0    a=1    a=2

input = state

categorical cross-entropy
\[ C = - \sum_a P(a) \ln \pi_{\theta}(a|s) \]
distr. from net
desired distribution

Set
\[ P(a) = R \]
for a=action that was taken

\[ P(a) = 0 \]
for all other actions a

\[ \Delta \theta = -\eta \frac{\partial C}{\partial \theta} \]
implements policy gradient
Among the major board games, “Go” was not yet played on a superhuman level by any program (very large state space on a 19x19 board!)

alpha-Go beat the world’s best player in 2017
“alpha-Go”

First: try to learn from human expert players

Sampled state-action pairs \((s, a)\), using stochastic gradient ascent to maximize the likelihood of the human move \(a\) selected in state \(s\):

\[
\Delta \sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial \sigma}
\]

We trained a 13-layer policy network, which we call the SL policy network, from 30 million positions from the KGS Go Server. The net-

Silver et al., “Mastering the game of Go with deep neural networks and tree search” (Google Deepmind team), Nature, January 2016
“alpha-Go”

Second: use policy gradient RL on games played against previous versions of the program

to the current policy. We use a reward function $r(s)$ that is zero for all non-terminal time steps $t < T$. The outcome $z_t = \pm r(s_T)$ is the terminal reward at the end of the game from the perspective of the current player at time step $t$: +1 for winning and −1 for losing. Weights are then updated at each time step $t$ by stochastic gradient ascent in the direction that maximizes expected outcome:

$$\Delta \rho \propto \frac{\partial \log p_\rho(a_t | s_t)}{\partial \rho} z_t$$

Silver et al., “Mastering the game of Go with deep neural networks and tree search” (Google Deepmind team), Nature, January 2016
Silver et al., “Mastering the game of Go with deep neural networks and tree search” (Google Deepmind team), Nature, January 2016
Silver et al., “Mastering the game of Go with deep neural networks and tree search” (Google Deepmind team), Nature, January 2016
Q-learning

An alternative to the policy gradient approach

Introduce a quality function $Q$ that predicts the future reward for a given state $s$ and a given action $a$. **Deterministic policy**: just select the action with the largest $Q$!
Q maximal

player & possible actions
Q-learning

Introduce a quality function $Q$ that predicts the future reward for a given state $s$ and a given action $a$. **Deterministic policy**: just select the action with the largest $Q$!

$$Q(s_t, a_t) = E[R_t | s_t, a_t]$$

(assuming future steps to follow the policy!)

“Discounted” future reward:

$$R_t = \sum_{t'=t}^{T} r_{t'} \gamma^{t'-t}$$

Reward at time step $t$: $r_t$

Discount factor: $0 < \gamma \leq 1$

**How do we obtain $Q$?**
Bellmann equation: (from optimal control theory)

\[ Q(s_t, a_t) = E[r_t + \gamma \max_a Q(s_{t+1}, a) | s_t, a_t] \]

In practice, we do not know the Q function yet, so we cannot directly use the Bellmann equation. However, the following update rule has the correct Q function as a fixed point:

\[ Q^{\text{new}}(s_t, a_t) = Q^{\text{old}}(s_t, a_t) + \alpha(r_t + \gamma \max_a Q^{\text{old}}(s_{t+1}, a) - Q^{\text{old}}(s_t, a_t)) \]

If we use a neural network to calculate Q, it will be trained to yield the “new” value in each step.
Initially, Q is arbitrary. It will be bad to follow this Q all the time. Therefore, introduce probability $\epsilon$ of random action ("exploration")!

Follow Q: "exploitation"

Do something random (new): "exploration"

"$\epsilon$-greedy"

Reduce this randomness later!