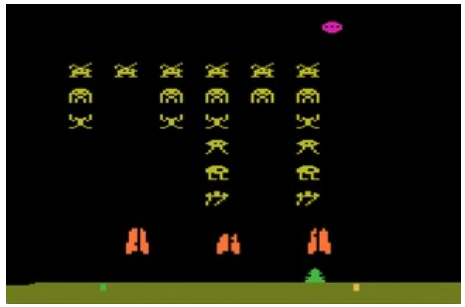
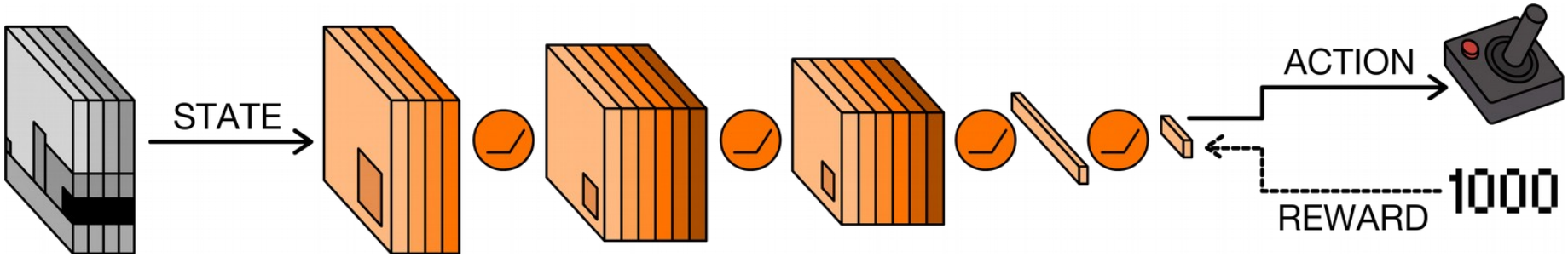
Deep Q-Learning: DQN

Learns to play ATARI 2600 games from screen images and score.
(DeepMind / Nature, 2015)

Deep feed-forward network approximating $Q(x, y)$

- input = state = 4 time-subsequent 84x84 gray-scale screens
- separate output for each $y \in Y$
- 2 convolution + 1 connected hidden layers





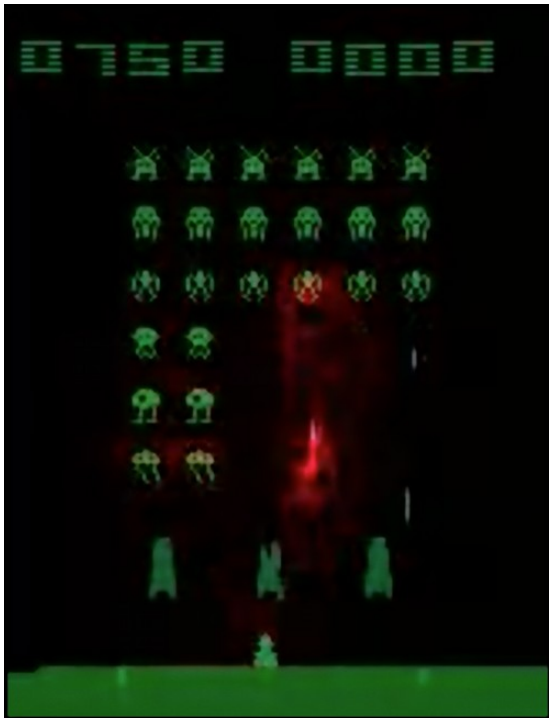
Demo

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Deep Q-Learning: DQN (cont'd)

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    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 $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to
  end for
end for



Experience replay prevents long chains of correlated training examples by sampling from a buffer of $\vec{\phi}(x), y, r, \vec{\phi}(x')$ tuples recorded in the past.

Backpropagation to the original image inputs reveals areas of 'attention'.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Policy Search

Instead of searching $\widehat{Q}$ / $\widehat{U}$ / p, search directly a good policy $\pi : X \to Y$, i.e. pose the problem as a classification problem.

A feature-based approach applicable again:

$$\pi(x) = \pi'(\phi_1(x), \ldots \phi_n(x))$$

Quality of π' is estimated as mean total rewards over repeated episodes using π.

Gradient-based search for a π' maximizing the quality not directly applicable since Y is finite (discrete), and π' thus not differentiable in its argument.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Differentiable Policy Search

Gradient-based policy search is possible with a *stochastic policy* choosing action y in state x with softmax probability

$$\frac{e^{q(\mathbf{w},\boldsymbol{\phi}(x,y))}}{\sum_{y' \in Y} e^{q(\mathbf{w},\boldsymbol{\phi}(x,y'))}}$$

where $\mathbf{w} \in \mathbb{R}^n$ are real parameters, $\boldsymbol{\phi} = \langle \phi_1, \ldots, \phi_{n'} \rangle$ are some real-valued features, and $q : \mathbb{R}^{n+n'} \to \mathbb{R}$. An example (in which $n = n'$)

$$q(\mathbf{w}, \boldsymbol{\phi}(x, y)) = \sum_{i=1}^{n} w^i \phi^i(x, y)$$

If q differentiable in all ϕ^i as above then $\mathbf{w}$ can be learned through stochastic gradient descent.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Learning a Feature-Based Environment Model

The ADP agent derives its policy from a learned model of p of the transition distribution P.

Such a model can also be feature based. So $p(x'|x, y)$ can be represeted e.g. by

$$f(\mathbf{w}, \boldsymbol{\phi}(x', x, y)) = \sum_{i=1}^{n} w^i \phi^i(x', x, y)$$

where ϕ^i are features of the x', x, y triple and w^i real parameters. This allows to use the gradient method as in the previous examples.

f need not even be normalized to the $[0; 1]$ probability range if only used in $\arg\max_y$ expressions such as (??).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Bayesian Learning of an Environment Model

Consider the following *Bayesian* approach *(not to be confused with Bayesian networks)*, which involves

- a countable (possibly infinite) set $\mathcal{M}$ of probability distributions ("model class")
- at each $k \in \mathbb{N}$, a probability distribution B_k on $\mathcal{M}$ where $B_k(p)$ ($p \in \mathcal{M}$) quantifies the belief that $P \equiv p$

The probability model at k is then assumed as

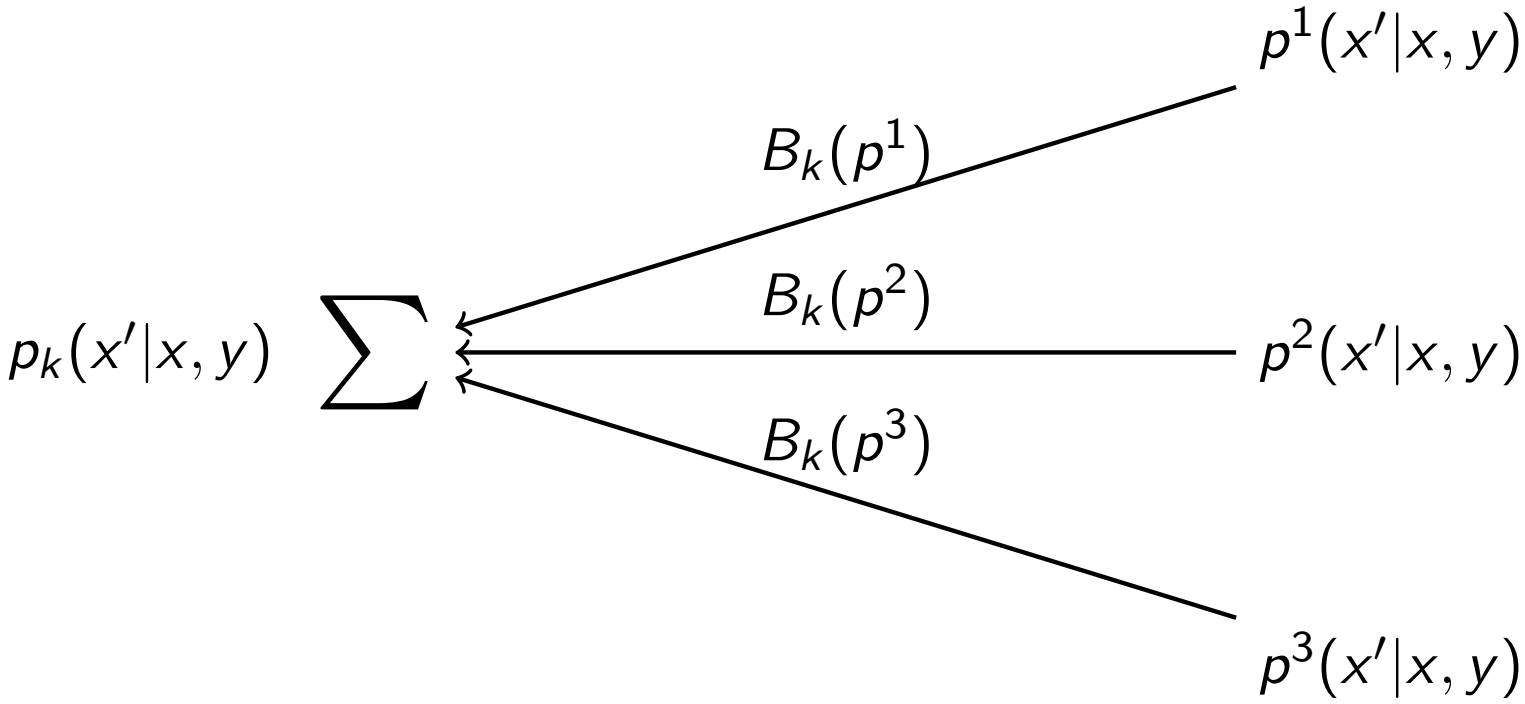
$$p_k(x'|x,y) = \sum_{p \in \mathcal{M}} p(x'|x,y) B_k(p) \tag{1}$$

i.e, a weighted sum where each distribution from $\mathcal{M}$ contributes the stronger the higher its belief.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Bayesian Learning of an Environment Model (cont'd)

Current Model *Current Belief* *Possible Environments* $\mathcal{M}$



FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Bayesian Learning of an Environment Model (cont'd)

As soon as x_{k+1} has been observed, B_k is updated by the Bayes rule to the *posterior*

$$B_{k+1}(p) = \alpha p(x_{k+1}|x_k, y_k)B_k(p) \tag{2}$$

for each $p \in \mathcal{M}$, where the normalizer α is such that

$$\sum_{p \in \mathcal{M}} B_{k+1}(p) = 1$$

Theorem 1

If $p = P$ for some $p \in \mathcal{M}$ and $B_1(p) > 0$, then $p_k \to_{k\to\infty} P$.

So also the greedy policy (??) will converge to the optimal policy. *The Bayesian approach thus avoids the exploration-exploitation dilemma* but has a new element to supply: the initial belief B_1.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Bayesian Learning - Continuous case

The Bayesian learning approach can be generalized to the continuous case where the model class $\mathcal{M}$ is uncountable.

For example, $\mathcal{M}$ may be the class of (conditional) normal distributions with all means and co-variances forming the real parameter vector $\mathbf{w}$.

Then instead of (1), we have

$$p_k(x_{k+1}|x_k, y_k) = \int_{\mathbf{w}} p(x_{k+1}|x_k, y_k, \mathbf{w}) B(\mathbf{w})$$

and the update (2) changes to

$$B_{k+1}(\mathbf{w}) = \alpha P(x_{k+1}|x_k, y_k, \mathbf{w}) B_k(\mathbf{w})$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Assumptions on Environments So Far

Recall the special assumptions we have imposed so far on the probabilistic desription (??) of the environment, which is the product of (??) and (??).

Mistake-bound learning: no assumption on observations, rewards by (??)

PAC learning: i.i.d. observations (??), rewards by (??) (or (??) for inconsistent learning)

Distribution learning (Bayes nets): i.i.d. observations (??), rewards case-specific, e.g. 0 for correct prediction of missing values, -1 otherwise

Reinforcement Learning: observations by (??) (Markov property), rewards by (??)

Let us now investigate if an agent can maximize the utility *without such assumptions*.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Universal Sequence Prediction

For the stated goal, the agent necessarily must be able to predict the percept xr_k from the history $y_{\leq k}, xr_{<k}$ so that xr_k maximizes (??).

For a start, we will disregard actions and rewards, and focus on the central problem:

Universal Sequence Prediction Problem

Given a set X and a sequence

$$x_1, x_2, \ldots x_k \in X$$

predict x_{k+1} maximizing $P(x_{k+1} \mid x_{\leq k})$ without knowing P.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Sequence Prediction

Some sequences seem obvious to extend. E.g.

$$1, 2, 3, 4, 5$$

because of the pattern $x_{k+1} = x_k + 1$.

But e.g.

$$1, 2, 3, 4, 29$$

could also be argued due to $x_k = k^4 - 10k^3 + 35k^2 - 49k + 24$.

The first pattern seems more plausible because it is *simpler*. Note that this reason is not statistical/probabilistic.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Sequence Prediction (cont'd)

Other sequences have no obvious equational pattern

$$3, 1, 4, 1, 5, 9$$

but there is still a simple extension rule: here x_k is the k's digit in the decimal expansion of the number π. So the extension 9 seems plausible here.

We need to formalize these thoughts to answer questions such as

- What exactly is meant by *pattern*?
- How to measure *complexity* of patterns and sequences?
- Are simple patterns more likely to make correct predictions?

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Computing a Sequence

The most general intepretation of sequence 'pattern' is a *program* that generates the sequence.

For simplicity, let $X = \{ 0, 1 \}$ so X^* denotes the set of all finite binary strings. Let $T : X^* \to X^*$ be a partial recursive function, i.e., there is a Turing machine computing $T(p)$ but for some $p \in X^*$ it need not halt.

The input p to the T.M. may be interpreted as a program computing $T(p)$.

Here we assume the prefix Turing machine *which as an* input tape, output tape *(both unidirectional) and any finite number of bidirectional* working tapes. *This is just for convenience, its expressive power is same as that of the vanilla Turing machine.*

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Computing a Sequence (cont'd)

Intuitively, simple strings (even infinite) are those computed by short programs p.

So e.g. the decimal expansion of π, however long, is simple because there is a short program computing it.

Denoting the length of p by $|p|$, this gives rise to the Kolmogorov complexity of strings.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Kolmogorov Complexity

The **Kolmogorov complexity** $K_T(q)$ of $q \in X^*$ with respect to T is

$$K_T(q) = \min \{ \, |p| \, ; p \in \{ \, 0, 1 \, \}^* \, , T(p) = q \, \}$$

So the complexity of a (possibly infinite) string is the length of the shortest program that generates it, i.e., the shortest binary input to T that makes it produce the string.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Kolmogorov Complexity (cont'd)

Dependence of $K_T(q)$ on T is not a serious problem as there is a *universal T.M.* U which simulates any T.M. T given the (finite!) sequential description $\langle T \rangle \in X^*$ of T as a distinguished part of its input.

$\langle T \rangle$ can be distinguished from p on the input tape e.g. by putting the string $0^{|\langle T \rangle|}1\,\langle T \rangle\, p$ on it so that the number of leading zeros indicates the length of $\langle T \rangle$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Kolmogorov Complexity (cont'd)

Consequence: given a T, for *every* $q \in X^*$:

$$K_U(q) \leq K_T(q) + \mathcal{O}(|\langle T \rangle|)$$

where the rightmost term ('translation overhead') does not depend on q and becomes negligible for large q. So we adopt K_U as the universal complexity measure and denote $K(q) = K_U(q)$.

Clearly, for every $q \in X^*$:

$$K(q) \leq |q| + c \tag{3}$$

since the program for computing q can simply contain the $|q|$ symbols of q plus some constant c number of symbols implementing the loop to print them on the output.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Kolmogorov Complexity - Examples

- $q = \underbrace{0, 0, \ldots, 0}_{n \text{ times}}$ has $K(q) = \log n + c$. Need $\log(n)$ symbols to encode the integer n plus a constant-size code to print it.
- $q =$ the first n digits in the binary expansion of π also has $K(q) = \log n + c$: $\log n$ symbols to specify n plus a constant-size code for calculating (and printing) the digits of π.
- Are there any strings q such that $K(q) \geq |q|$?
  Yes, such strings exist for any length k, as there are only $2^k - 1$ programs (binary strings) shorter than k, so there must be some string of length k for which there is no shorter program generating it. Such a string is called *incompressible* or *random* *(not in the probabilistic sense!)*.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Kolmogorov Complexity - Computability

Theorem 2

The question whether $K(q) \geq n$ $(q \in X^*, n \in \mathbb{N})$ *is* undecidable, *i.e.,* K is not finitely computable.

Proof: Assume a deciding program p exists. Consider the first (in the lexicographic order) string $q \in X^*$ such that $K(q) \geq n$.

Then q can be generated as follows: for each $q \in X^*$ in the lexicographic order, determine if $K(q) \geq n$ using p and print the first such q.

This procedure can be encoded in a program using $|p| + \log(n) + c$ symbols, which (for a sufficiently large n) is smaller than n, so $K(q) < n$, yielding a contradiction. So p does not exist.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE