CS 287: Advanced Robotics
Fall 2009

Lecture 15: LSTD, LSPI, RLSTD, imitation learning

Pieter Abbeel
UC Berkeley EECS

TD(0) with linear function approximation guarantees

- Stochastic approximation of the following operations:
  - Back-up: $(T^\pi V)(s) = \sum_{s'} T(s,\pi(s),s')\,[R(s,\pi(s),s') + \gamma V(s')]$
  - Weighted linear regression: $\min_\theta \sum_s D(s)((T^\pi V)(s) - \theta^\top \phi(s))^2$
- Batch version (for large state spaces):
  - Let {(s,a,s')} have been sampled according to D
  - Iterate:
    - Back-up for sampled (s,a,s'): $V(s) \leftarrow [R(s,a,s') + \gamma V(s')] = [R(s,a,s') + \gamma \theta^\top \phi(s')]$
    - Perform regression: $\min_\theta \sum_{(s,a,s')} (V(s) - \theta^\top \phi(s))^2$

$$\min_\theta \sum_{(s,a,s')} (R(s,a,s') + \gamma {\theta^{(\text{old})}}^\top \phi(s') - \theta^\top \phi(s))^2$$

TD(0) with linear function approximation guarantees

- Iterate:

$$\theta^{(\text{new})} = \arg\min_{\theta} \sum_{(s,a,s')} \left(R(s,a,s') + \gamma {\theta^{(\text{old})}}^\top \phi(s') - \theta^\top \phi(s)\right)^2$$

- Can we find the fixed point directly?
  - Rewrite the least squares problem in matrix notation:

$$\theta^{(\text{new})} = \arg\min_{\theta} \|R + \gamma \Phi' \theta^{(\text{old})} - \Phi\theta\|_2^2$$

  - Solution: $\theta^{(\text{new})} = (\Phi^\top \Phi)^{-1} \Phi^\top (R + \gamma \Phi' \theta^{(\text{old})})$

TD(0) with linear function approximation guarantees

- Solution: $\theta^{(\text{new})} = (\Phi^\top \Phi)^{-1} \Phi^\top (R + \gamma \Phi' \theta^{(\text{old})})$
- Fixed point?

$$\begin{aligned}
\theta &= (\Phi^\top \Phi)^{-1} \Phi^\top (R + \gamma \Phi' \theta) \\
(\Phi^\top \Phi)\theta &= \Phi^\top (R + \gamma \Phi' \theta) \\
\left(\Phi^\top \Phi - \gamma \Phi^\top \Phi'\right)\theta &= \Phi^\top R \\
\theta &= \left(\Phi^\top \Phi - \gamma \Phi^\top \Phi'\right)^{-1} \Phi^\top R
\end{aligned}$$

LSTD(0)

- Collect state-action-state triples (s_i,a_i,s'_i) according to a policy π
- Build the matrices:

$$\Phi = \begin{bmatrix} \phi(s_1)^\top \\ \phi(s_2)^\top \\ \dots \\ \phi(s_m)^\top \end{bmatrix}, \Phi' = \begin{bmatrix} \phi(s'_1)^\top \\ \phi(s'_2)^\top \\ \dots \\ \phi(s'_m)^\top \end{bmatrix}, R = \begin{bmatrix} R(s_1, a_1, s'_1) \\ R(s_2, a_2, s'_2) \\ \dots \\ R(s_m, a_m, s'_m) \end{bmatrix}$$

- Find an approximation of the value function

$$V^\pi(s) \approx \theta^\top \phi(s)$$

$$\text{for} \quad \theta = \left(\Phi^\top \Phi - \gamma \Phi^\top \Phi'\right)^{-1} \Phi^\top R$$

LSTD(0) in policy iteration

- Iterate
  - Collect state-action-state triples (s_i,a_i,s'_i) according to current policy π
  - Use LSTD(0) to compute V^π
- Tweaks:
  - Can re-use triples (s_i,a_i,s'_i) from previous policies as long as they are consistent with the current policy
  - Can redo the derivation with Q functions rather than V
  - In case of stochastic policies, can weight contribution of a triple according to Prob(a_i|s_i) under the current policy

  → Doing all three results in "***Least squares policy iteration***," (Lagoudakis and Parr, 2003).

LSTD(0) --- batch vs. incremental updates

- Collect state-action-state triples (s_i,a_i,s'_i) according to a policy π
- Build the matrices:

$$\Phi_m = \begin{bmatrix} \phi(s_1)^\top \\ \phi(s_2)^\top \\ \dots \\ \phi(s_m)^\top \end{bmatrix}, \Phi'_m = \begin{bmatrix} \phi(s'_1)^\top \\ \phi(s'_2)^\top \\ \dots \\ \phi(s'_m)^\top \end{bmatrix}, R_m = \begin{bmatrix} R(s_1, a_1, s'_1) \\ R(s_2, a_2, s'_2) \\ \dots \\ R(s_m, a_m, s'_m) \end{bmatrix}$$

- Find an approximation of the value function

$$V^\pi(s) \approx \theta_m^\top \phi(s)$$
$$\text{for } \theta_m = \left(\Phi_m^\top(\Phi_m - \gamma\Phi'_m)\right)^{-1} \Phi_m^\top R_m$$

- One more datapoint → "m+1" :

$$\theta_{m+1} = \left(\Phi_m^\top(\Phi_m - \gamma\Phi'_m) + \phi_{m+1}(\phi_m - \gamma\phi'_m)^\top\right)^{-1} \left(\Phi_m^\top R_m + \phi_{m+1} r_{m+1}\right)$$

Sherman-Morrison formula: $(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^T A^{-1}}{1 + v^T A^{-1} u}.$

RLSTD

- Recursively compute approximation of the value function by leveraging the Sherman-Morrison formula
- $$\begin{aligned} A_m^{-1} &= \left(\Phi_m^\top(\Phi_m - \gamma\Phi'_m)\right)^{-1} \\ b_m &= \Phi_m R_m \\ \theta_m &= A_m^{-1} b_m \end{aligned}$$
- One more datapoint → "m+1" :

$$\begin{aligned} A_{m+1}^{-1} &= A_m^{-1} - \frac{A_m^{-1}\phi_{m+1}(\phi_{m+1} - \gamma\phi'_{m+1})^\top A_m^{-1}}{1 + (\phi_{m+1} - \gamma\phi'_{m+1})^\top A_m^{-1}\phi_{m+1}} \\ b_{m+1} &= b_m + \phi_{m+1} r_{m+1} \end{aligned}$$

- Note: there exist orthogonal matrix techniques to do the same thing but in a numerically more stable fashion (essentially: keep track of the QR decomposition of A_m)

RLSTD: for non-linear function approximators?

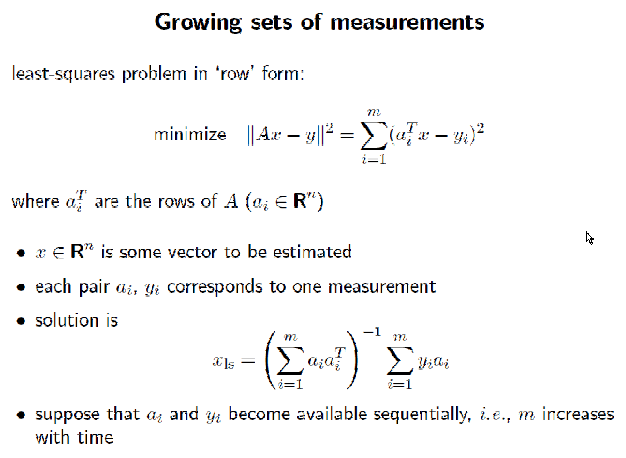
- RLSTD with linear function approximation with a Gaussian prior on \theta

  → Kalman filter

- Can be applied to non-linear setting too: simply linearize the non-linear function approximator around the current estimate of \theta; not globally optimal, but likely still better than "naïve" gradient descent

  (+prior → Extended Kalman filter)

Recursive Least Squares (1)



Growing sets of measurements

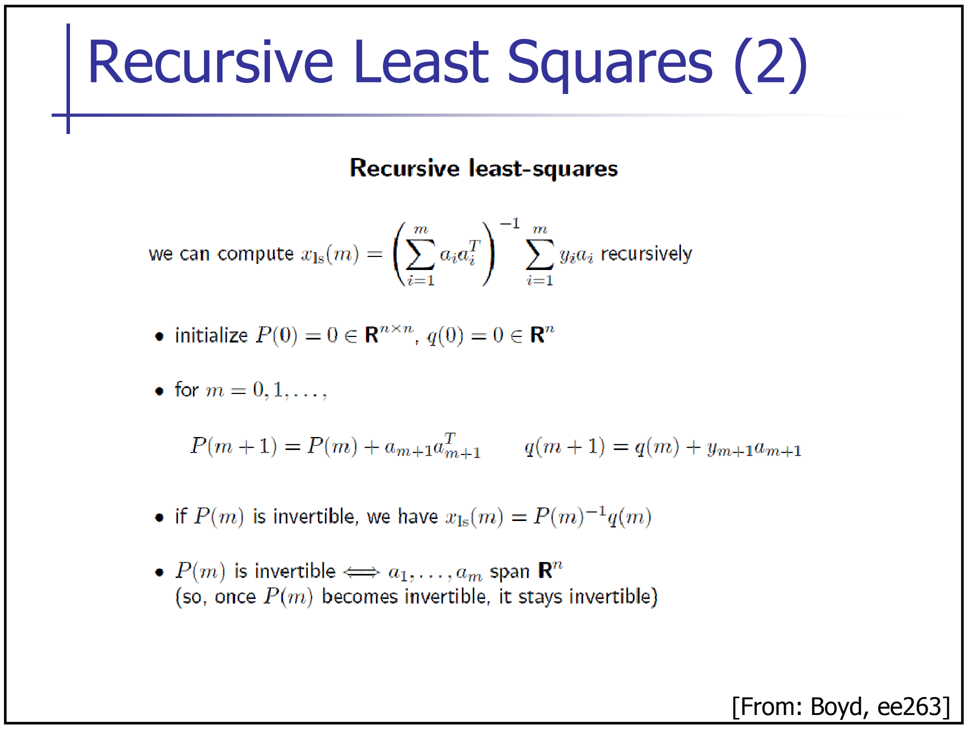
least-squares problem in 'row' form:

$$\text{minimize} \quad \|Ax - y\|^2 = \sum_{i=1}^{m} (a_i^T x - y_i)^2$$

where a_i^T are the rows of A ($a_i \in \mathbf{R}^n$)

- $x \in \mathbf{R}^n$ is some vector to be estimated
- each pair a_i, y_i corresponds to one measurement
- solution is
$$x_{\text{ls}} = \left(\sum_{i=1}^{m} a_i a_i^T\right)^{-1} \sum_{i=1}^{m} y_i a_i$$
- suppose that a_i and y_i become available sequentially, *i.e.*, m increases with time

[From: Boyd, ee263]



Recursive Least Squares (2)

Recursive least-squares

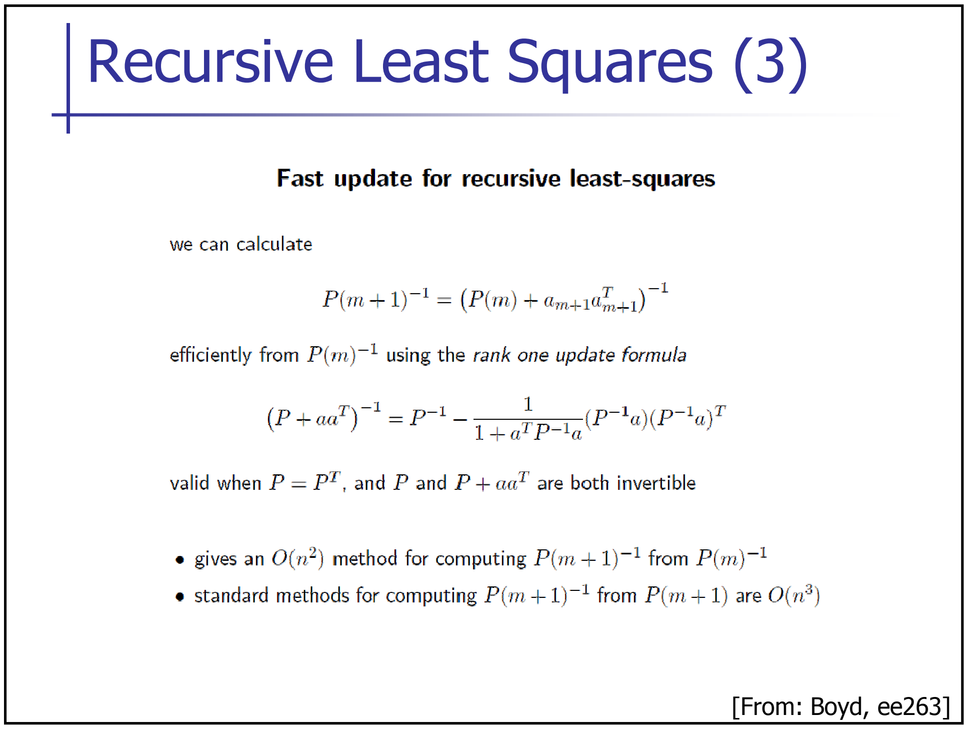
we can compute $x_{\mathrm{ls}}(m) = \left(\sum_{i=1}^{m} a_i a_i^T\right)^{-1} \sum_{i=1}^{m} y_i a_i$ recursively

- initialize $P(0) = 0 \in \mathbf{R}^{n\times n}$, $q(0) = 0 \in \mathbf{R}^n$
- for $m = 0, 1, \ldots,$

$$P(m+1) = P(m) + a_{m+1}a_{m+1}^T \qquad q(m+1) = q(m) + y_{m+1}a_{m+1}$$

- if $P(m)$ is invertible, we have $x_{\mathrm{ls}}(m) = P(m)^{-1}q(m)$
- $P(m)$ is invertible $\iff a_1, \ldots, a_m$ span $\mathbf{R}^n$
  (so, once $P(m)$ becomes invertible, it stays invertible)

[From: Boyd, ee263]



Recursive Least Squares (3)

Fast update for recursive least-squares

we can calculate

$$P(m+1)^{-1} = \left(P(m) + a_{m+1}a_{m+1}^T\right)^{-1}$$

efficiently from $P(m)^{-1}$ using the *rank one update formula*

$$\left(P + aa^T\right)^{-1} = P^{-1} - \frac{1}{1 + a^T P^{-1} a}(P^{-1}a)(P^{-1}a)^T$$

valid when $P = P^T$, and P and $P + aa^T$ are both invertible

- gives an $O(n^2)$ method for computing $P(m+1)^{-1}$ from $P(m)^{-1}$
- standard methods for computing $P(m+1)^{-1}$ from $P(m+1)$ are $O(n^3)$

[From: Boyd, ee263]

TD methods recap

- Model-free RL: learn V, Q directly from experience:
  - TD(λ), sarsa(λ): on policy updates
  - Q: off policy updates
- Large MDPs: include function Approximation
  - Some guarantees for linear function approximation
- Batch version
  - No need to tweak various constants
  - Same solution can be obtained incrementally by using recursive updates! This is generally true for least squares type systems.

Applications of TD methods

- Backgammon
- Standard RL testbeds (all in simulation)
  - Cartpole balancing
  - Acrobot swing-up
  - Gridworld --- Assignment #2
  - Bicycle riding
  - Tetris --- Assignment #2
- As part of actor-critic methods (=policy gradient + TD)
  - Fine-tuning / Learning some robotics tasks
- *Many financial institutions use some linear TD for pricing of options*

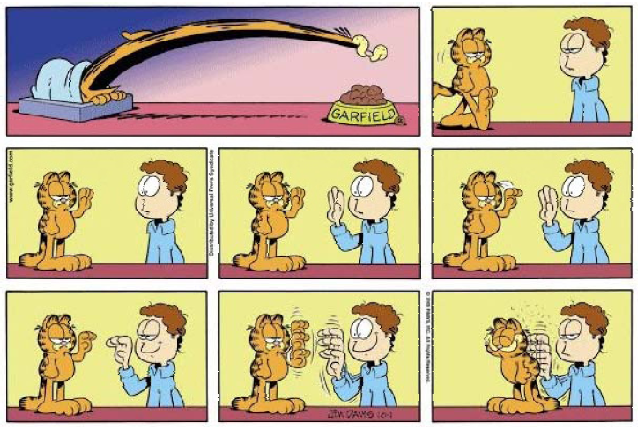
RL: our learning status

- Small MDPs: VI, PI, GPI, LP
- Large MDPs:
  - Value iteration + function approximation
    - Iterate: Bellman back-up, project, …
  - TD methods:
    - TD, sarsa, Q with function approximation
      - Simplicity, limited storage can be a convenience
    - LSTD, LSPI, RLSTD
    - Built upon in and compared to in many current RL papers
    - Main current direction: feature selection
- You should be able to read/understand many RL papers
- Which important ideas are we missing (and will I try to cover between today and the next 3-5 lectures) ?

- Imitation learning
  - Learn from observing an expert
- Linear programming w/function approximation and constraint sampling
  - Guarantees, Generally applicable idea of constraint sampling
- Policy gradient, Actor-Critic (=TD+policy gradient in one)
  - Fine tuning policies through running trials on a real system, Robotic success stories
- Partial observability
  - POMDPS
- Hierarchical methods
  - Incorporate your knowledge to enable scaling to larger systems
- Reward shaping
  - Can we choose reward functions such as to enable faster learning?
- Exploration vs. exploitation
  - How/When should we explore?
- Stochastic approximation
  - Basic intuition behind how/when sampled versions work?

Imitation learning

to program others



Imitation learning: what to learn?

- If expert available, could use expert trace s_1, a_1, s_2, a_2, s_3, a_3, … to learn "something" from the expert
  - Behavioral cloning: use supervised learning to directly learn a policy S→A.
    - No model of the system dynamics required
    - No MDP / optimal control solution algorithm required
  - Inverse reinforcement learning:
    - Learn the reward function
    - Often most compact and transferrable task description
  - Trajectory primitives:
    - Use expert trajectories as motion primitives / components for motion planning
    - Use expert trajectories as starting points for trajectory optimization

Behavioral cloning

- If expert available, could use expert trace $s_1, a_1, s_2, a_2, s_3, a_3, \ldots$ to learn the expert policy $\pi : S \rightarrow A$
- Class of policies to learn:
  - Neural net, decision tree, linear regression, logistic regression, svm, deep belief net, …
- Advantages:
  - No model of the system dynamics required
  - No MDP / optimal control solution algorithm required
- Minuses:
  - Only works if we can come up with a good policy class
    - Typically more applicable to "reactive" tasks, less so to tasks that involve planning
  - No leveraging of dynamics model if available.

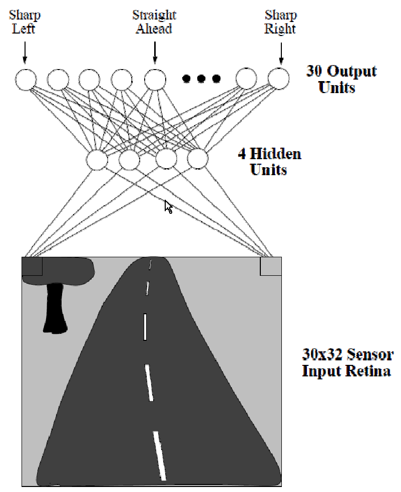
Alvinn

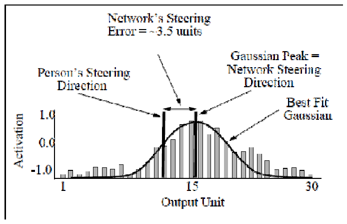
- Task: steer a vehicle

CMU Navlab Autonomous Navigation Testbed

- Input: 30x32 image.

Alvinn



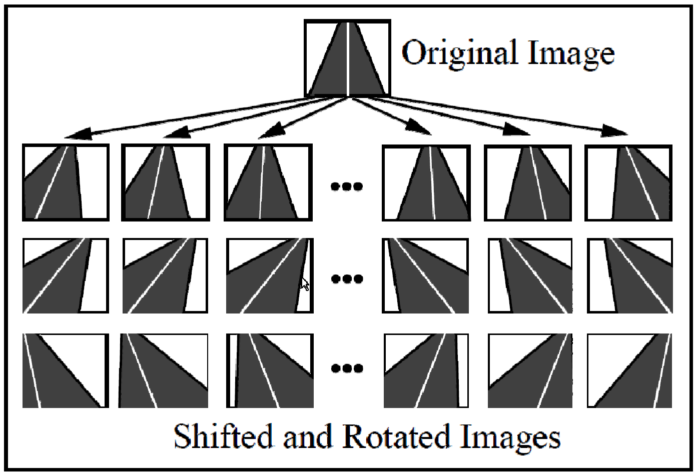


Richness of training data?

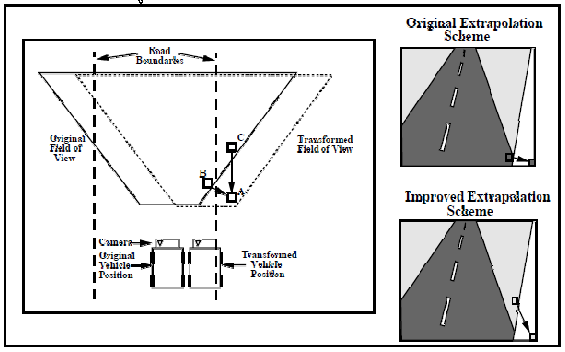
- Training data from good driver does not well represent situations from which it should be able to recover
- Might over-train on the "simple" data

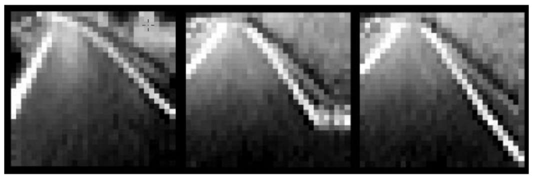
- Solution? Intentionally swerve off-center?
  - Issues:
    - Inconvenience to switch on/off the learning
    - Might require a lot of swerving (which could be especially undesirable in traffic)

Transformed images



Transformed images


Road Boundaries
Original Field of View
Transformed Field of View
C
B
A
Camera
Original Vehicle Position
Transformed Vehicle Position
Original Extrapolation Scheme
Improved Extrapolation Scheme

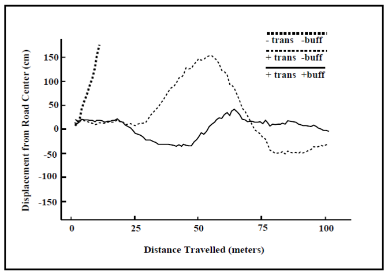


original extrap1 extrap2

Few other details

- Steering direction for transformed images:
  - “pure pursuit model” : constant steering arc will bring it back in the center at distance T
- Image buffering:
  - Keeps 200 images in buffer
  - One backpropagation pass over all images in each round of training
  - Replacement to favor neutral steering
- Road types:

Results



- Achieved 98.2% autonomous driving on a 5000 km (3000-mile) "No hands across America" trip.
- Throttle and brakes were human-controlled.

- Note: other autonomous driving projects:
  - Ernst Dickmanns
  - Darpa Grand and Urban Challenge

Sammut+al, Learning to fly (ICML1992)

- Task (in Silicon Graphics Flight Sim)
  - (crudely) Take off, fly through some waypoints, land
- Training data: 30 flights (/pilot)
- Recorded features: *on_ground, g_limit exceeded, wing_stall, twist* , *elevation, azimuth, roll_speed, elevation_speed, azimuth_speed, airspeed, climbspeed, E/W distance* from centre of runway, *altitude, N/S distance* from northern end of runway, *fuel, rollers, elevator, rudder, thrust*, *flaps*
- Data from each flight segmented into seven stages
- In each stage: Four separate decision trees (C4.5), one for each of the elevator, rollers, thrust and flaps.
- Succeeded in synthesizing control rules for a complete flight, including a safe landing. The rules **fly** the Cessna in a manner very similar to that of the pilot whose data were used to construct the rules.
- Pilots who are frugal in their use of the controls give few examples of what to do when things go wrong.

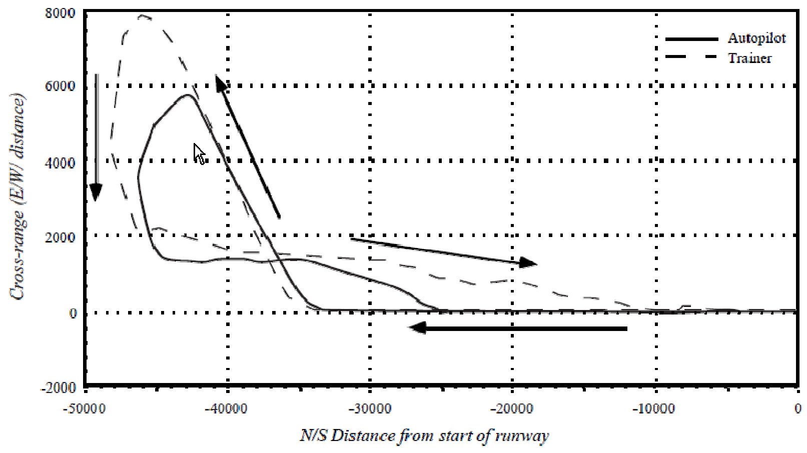
7 stages

- 1. Take off and fly to an altitude of 2,000 feet.
- 2. Level out and fly to a distance of 32,000 feet from the starting point.
- 3. Turn right to a compass heading of approximately 330°. The subjects were actually told to head toward a particular point in the scenery that corresponds to that heading.
- 4. At a North/South distance of 42,000 feet, turn left to head back towards the runway. The scenery contains grid marks on the ground. The starting point for the turn is when the last grid line was reached. This corresponds to about 42,000 feet. The turn is considered complete when the azimuth is between 140° and 180°.
- 5. Line up on the runway. The aircraft was considered to be lined up when the aircraft's azimuth is less than 5° off the heading of the runway and the twist is less that ±10° from horizontal.
- 6. Descend to the runway, keeping in line. The subjects were given the hint that they should have an ‘aiming point’ near the beginning of the runway.
- 7. Land on the runway.

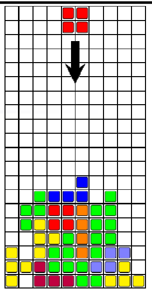
Sammut + al

- Example decision tree:
- Stage 3: Turn right to a compass heading of approximately 330°
  - twist <= -23 : left_roll_3
  - twist > -23 :
    - | azimuth <= -25 : no_roll
    - | azimuth > -25 : right_roll_2

Sammut+al



Tetris



- state: board configuration + shape of the falling piece ~2^{200} states!
- action: rotation and translation applied to the falling piece

$V(s) = \sum_{i=1}^{22} \theta_i \phi_i(s)$

- 22 features aka basis functions ϕ_i
  - Ten basis functions, 0, *. . . , 9, mapping the state to the height h[k] of each of the ten* columns.
  - Nine basis functions, 10*, . . . , 18, each mapping the state to the absolute difference* between heights of successive columns: *|h[k+1] – h[k]|, k = 1, . . . , 9.*
  - One basis function, 19, that maps state to the maximum column height: $\max_k$ *h[k]*
  - One basis function, 20, that maps state to the number of 'holes' in the board.
  - One basis function, 21, that is equal to 1 in every state.

[Bertsekas & Ioffe, 1996 (TD); Bertsekas & Tsitsiklis 1996 (TD);Kakade 2002 (policy gradient); Farias & Van Roy, 2006 (approximate LP)]

Behavioral cloning in tetris