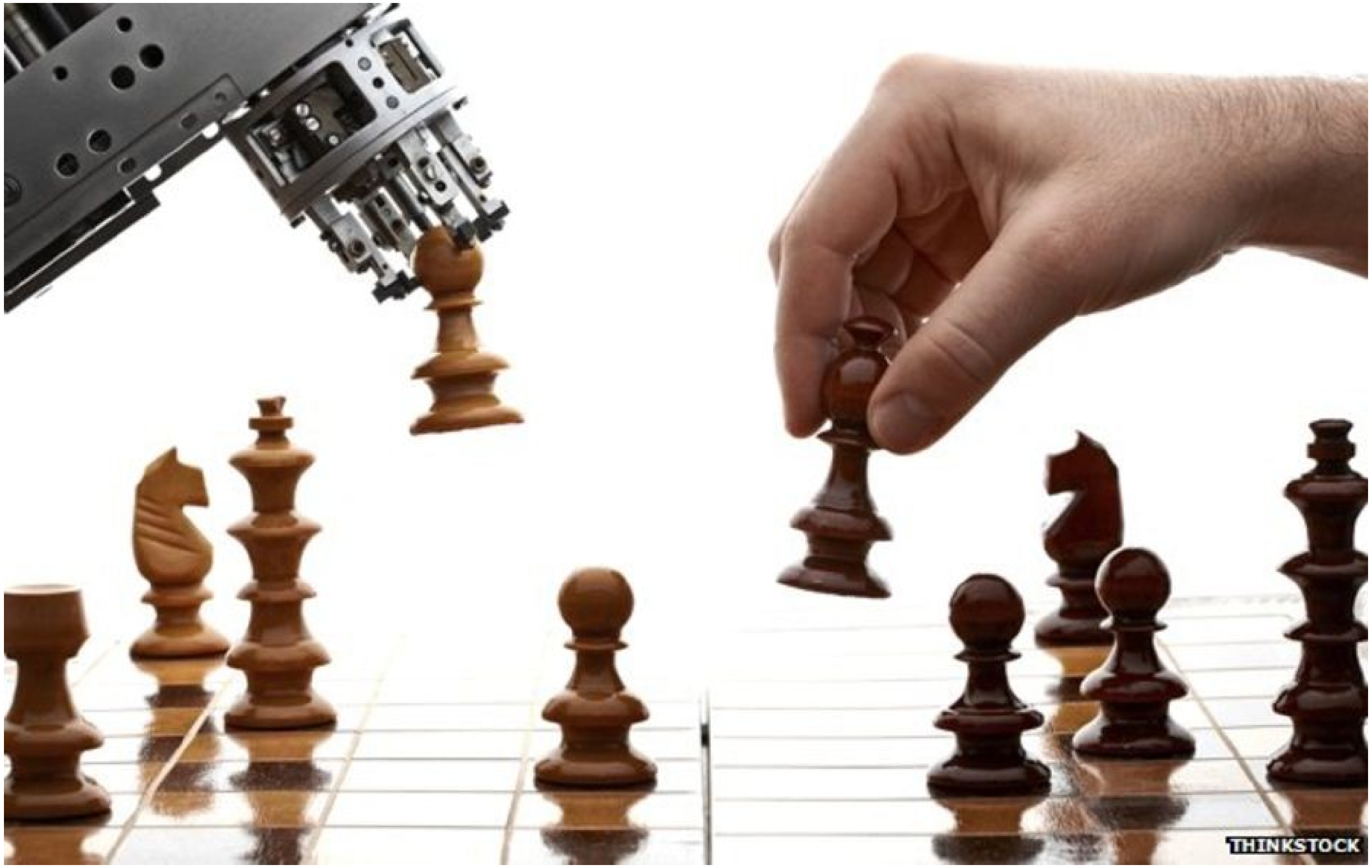


Introduction to Artificial Intelligence

COSC 4550 / COSC 5550

Professor Cheney

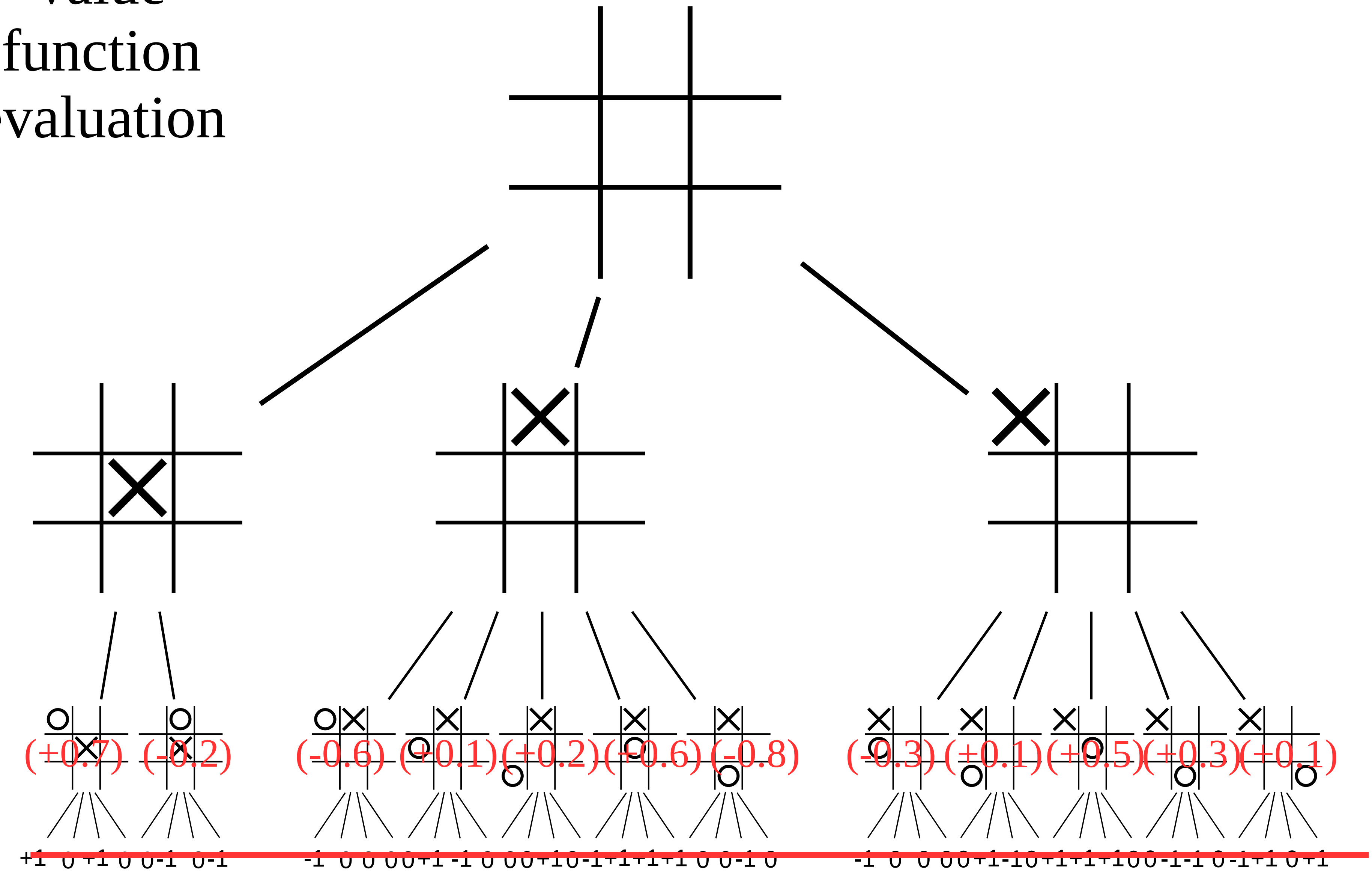
9/22/17

what pieces are on the board are “features” of that state

we often create heuristic value functions based on the features of the board (environmental) state

e.g. value of a board state = weighted sum of pieces on board:
value = #pawns * 1 + #bishops * 3 + #knights * 3 + #rooks * 5 + #queens * 9

value
function
evaluation



linear weighted sums are fast, so they're used often
(the whole point of a heuristic function is to save time!)

but this case assumes independent (non-interacting) pieces

and doesn't account for different layouts/arrangements

(you could have a different feature for each combination of piece and position)
64^7 = over 100,000,000,000 features...

how do we efficiently build flexible and robust feature sets?

optimization on non-linear function approximators

(e.g. machine learning with artificial neural networks)
(more on this later…)

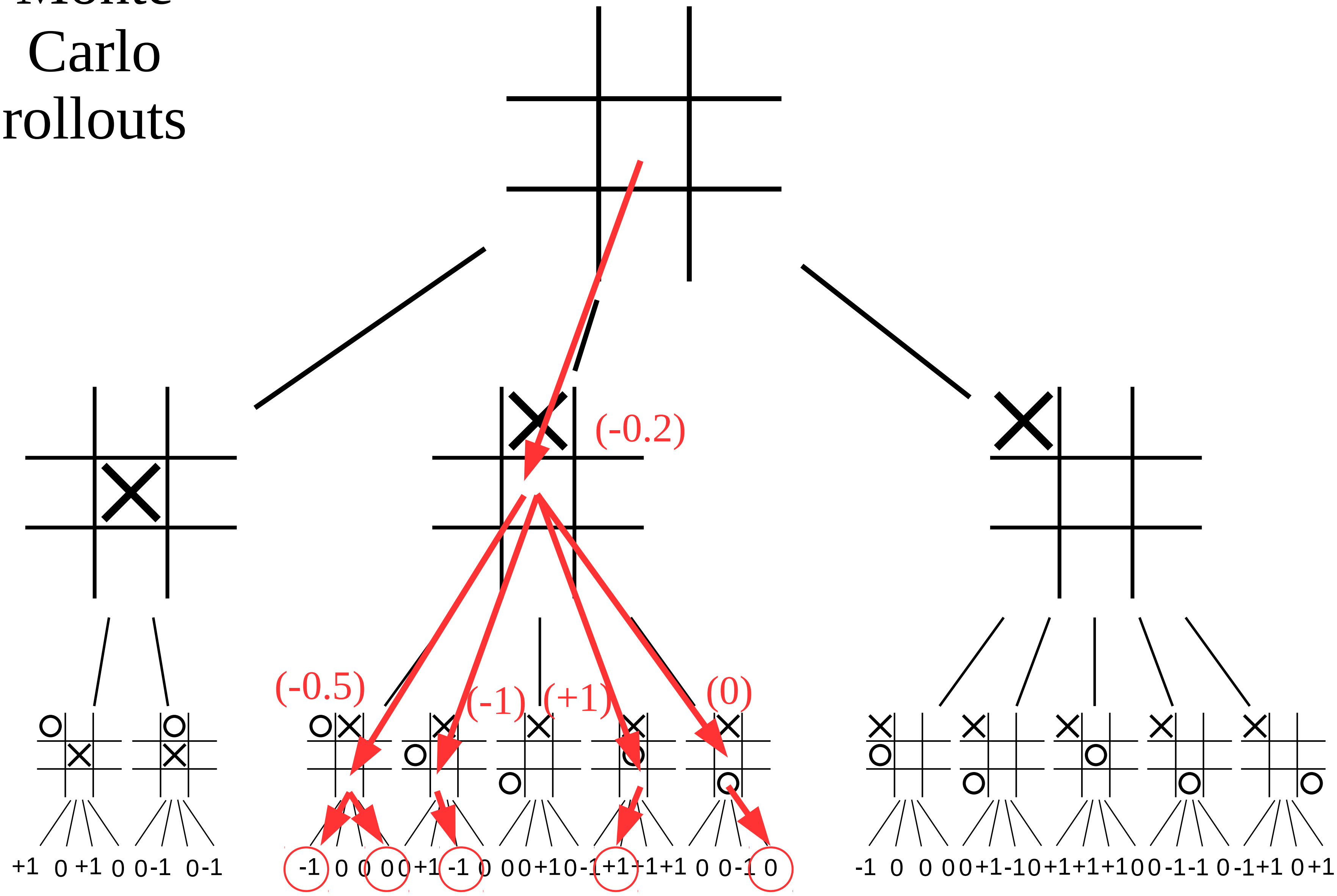
another heuristic approach to reducing tree complexity
is just to randomly sample along paths to estimate
the value of a given state or action
("Monte Carlo Tree Search")

this has the upside of seeing the true end-of-game value
(though you could also stop short and use a heuristic instead…)

but it has the downside of only sampling random paths
(i.e. unintelligently sampling)

(and as we saw from alpha-beta pruning,
many paths don't actually matter much)

Monte Carlo rollouts



how can we better (more intelligently) sample paths?

we can't use the optimal policy – because we don't know it

we can't use our current policy, because we'd
only learn the values along that path
(i.e. no exploration, only exploitation)

optimism under uncertainty!

recall from A*, that we were able to get optimal exploration by having a optimistic heurestic function

(i.e. assuming states we hadn't seen before were good, and therefore that an optimal policy would try them)

Upper Confidence-bound Tree Search (UCT search)

measure not just the mean estimated value of a state/action but keep a confidence interval (e.g. your estimated variance)

take the most optimistic value in this range of uncertainty mean value estimate + “exploration bonus”

total # of games (including paths w/o i)

(constant)

# of wins (using node i)

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln t}{n_i}}$$

# of games (using node i)

# of games (using node i)

nodes that have been sampled more
(i.e. that we are more confident about the true value of)
have a smaller exploration bonus

AlphaGo application

NATURE | VOL 529 | 28 JANUARY 2016
doi:10.1038/nature16961

Mastering the game of Go with deep neural networks and tree search

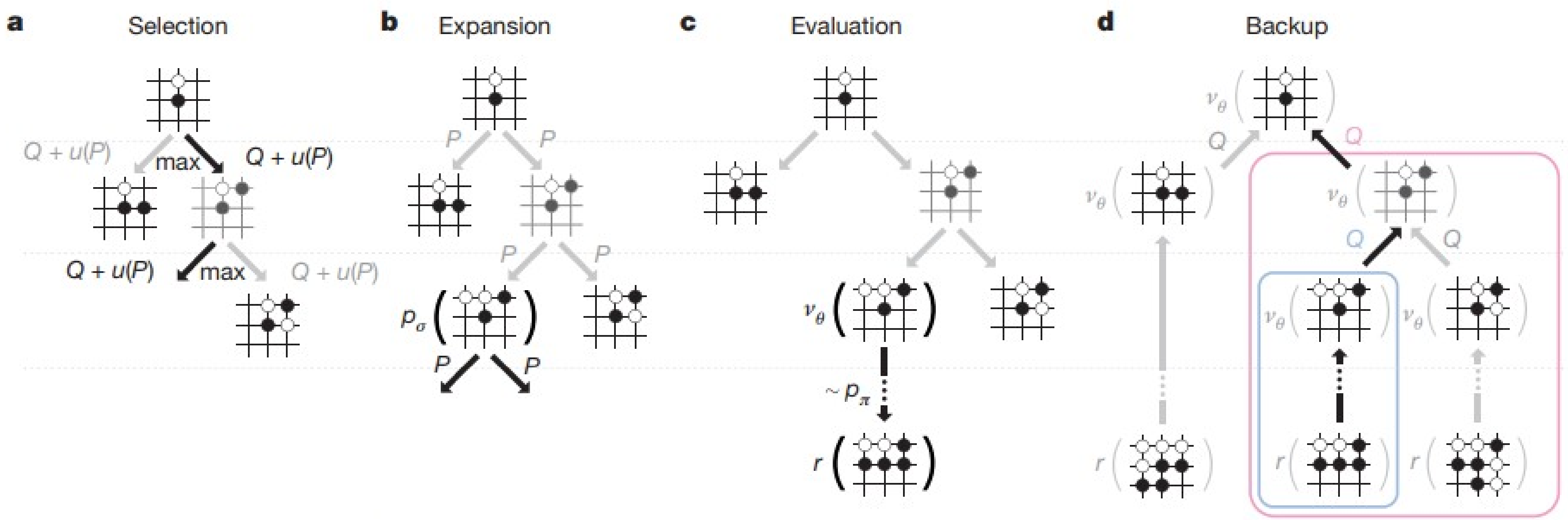


Figure 3 | Monte Carlo tree search in AlphaGo. a, Each simulation traverses the tree by selecting the edge with maximum action value Q, plus a bonus $u(P)$ that depends on a stored prior probability P for that edge. **b**, The leaf node may be expanded; the new node is processed once by the policy network p_σ and the output probabilities are stored as prior probabilities P for each action. **c**, At the end of a simulation, the leaf node is evaluated in two ways: using the value network v_θ; and by running a rollout to the end of the game with the fast rollout policy p_π, then computing the winner with function r. **d**, Action values Q are updated to track the mean value of all evaluations $r(\cdot)$ and $v_\theta(\cdot)$ in the subtree below that action.

ARTICLE

doi:10.1038/nature16961

Mastering the game of Go with deep neural networks and tree search

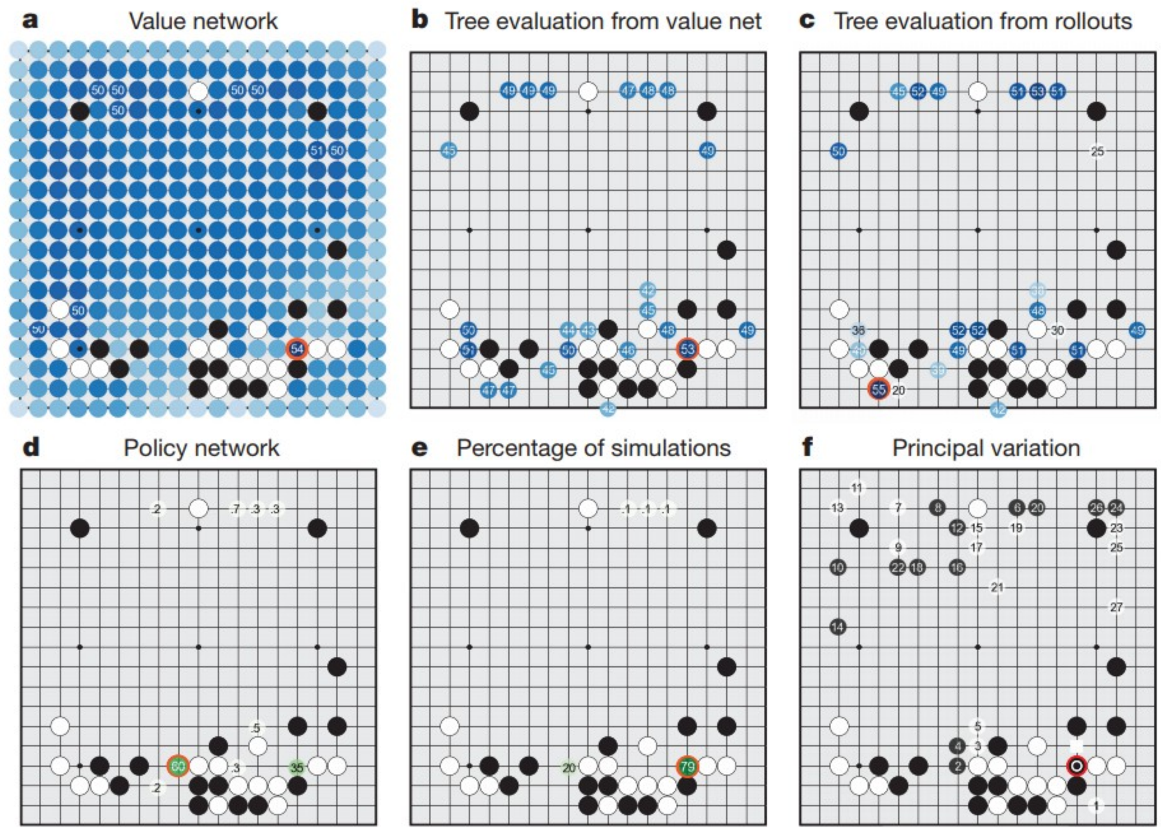
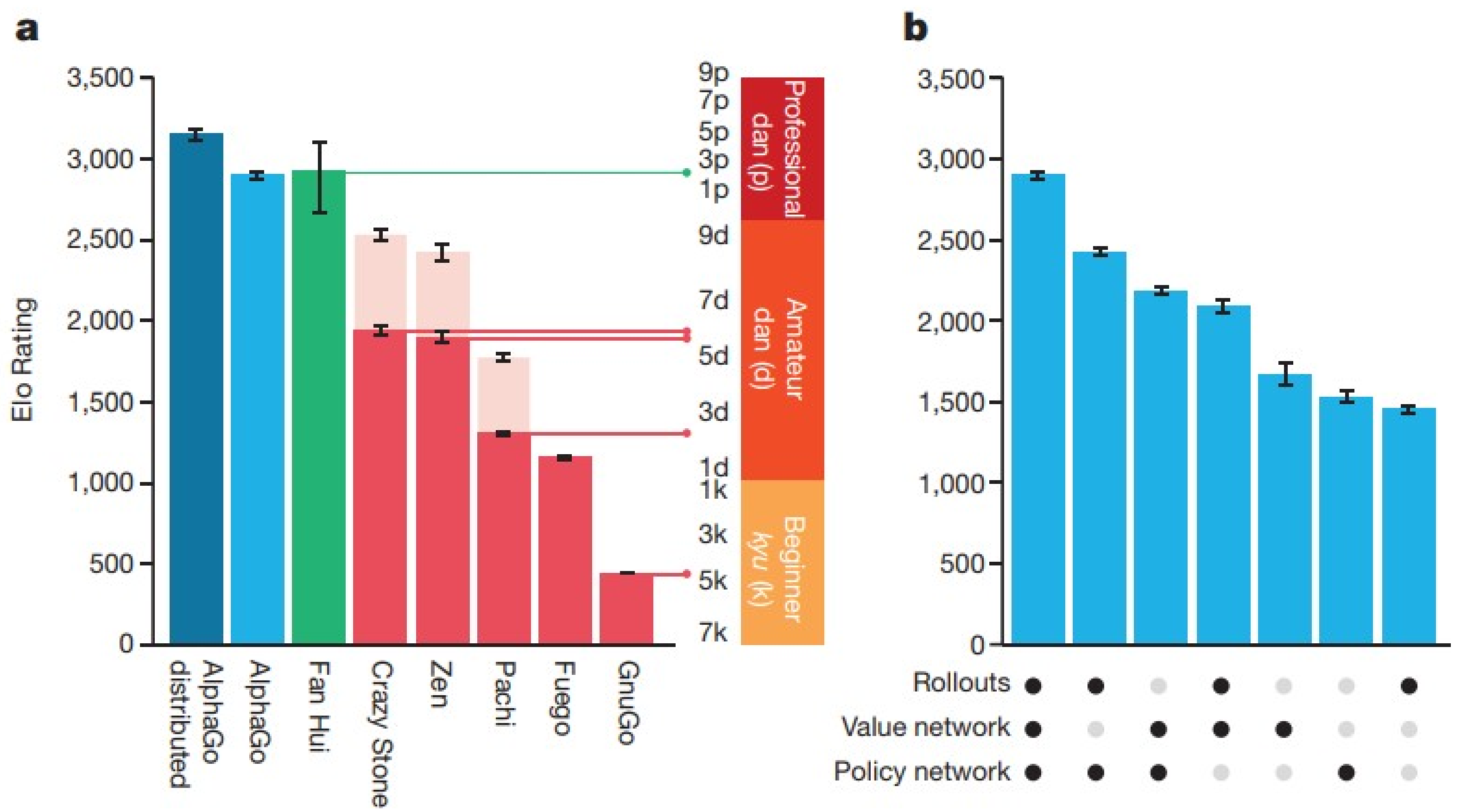


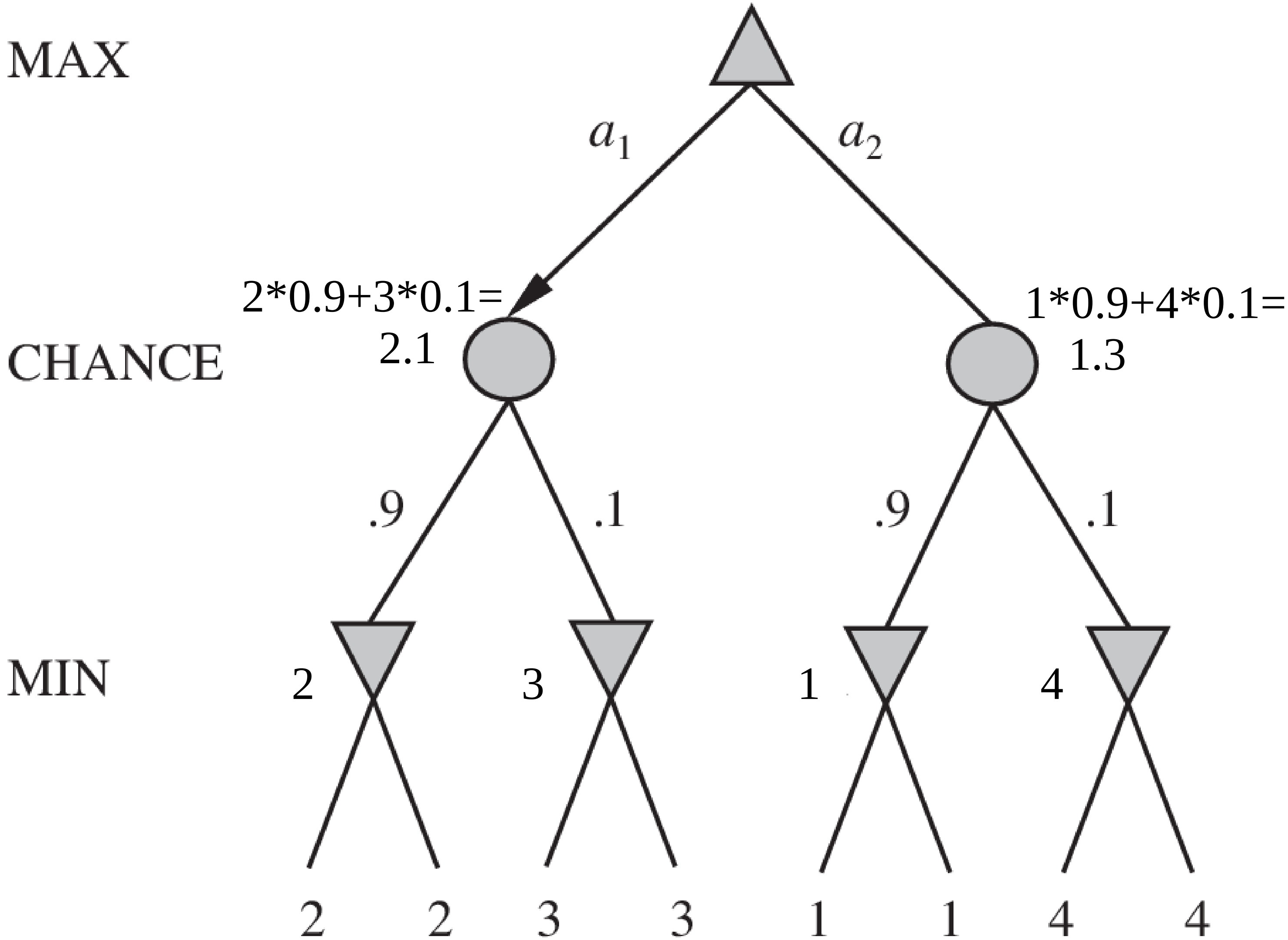
Figure 5 | How AlphaGo (black, to play) selected its move in an informal game against Fan Hui. For each of the following statistics, the location of the maximum value is indicated by an orange circle. **a**, Evaluation of all successors s' of the root position s, using the value network $v_\theta(s')$; estimated winning percentages are shown for the top evaluations. **b**, Action values $Q(s, a)$ for each edge (s, a) in the tree from root position s; averaged over value network evaluations only ($\lambda = 0$). **c**, Action values $Q(s, a)$, averaged over rollout evaluations only ($\lambda = 1$). **d**, Move probabilities directly from the SL policy network, $p_\sigma(a|s)$; reported as a percentage (if above 0.1%). **e**, Percentage frequency with which actions were selected from the root during simulations. **f**, The principal variation (path with maximum visit count) from AlphaGo's search tree. The moves are presented in a numbered sequence. AlphaGo selected the move indicated by the red circle; Fan Hui responded with the move indicated by the white square; in his post-game commentary he preferred the move (labelled 1) predicted by AlphaGo.

NATURE | VOL 529 | 28 JANUARY 2016
doi:10.1038/nature16961

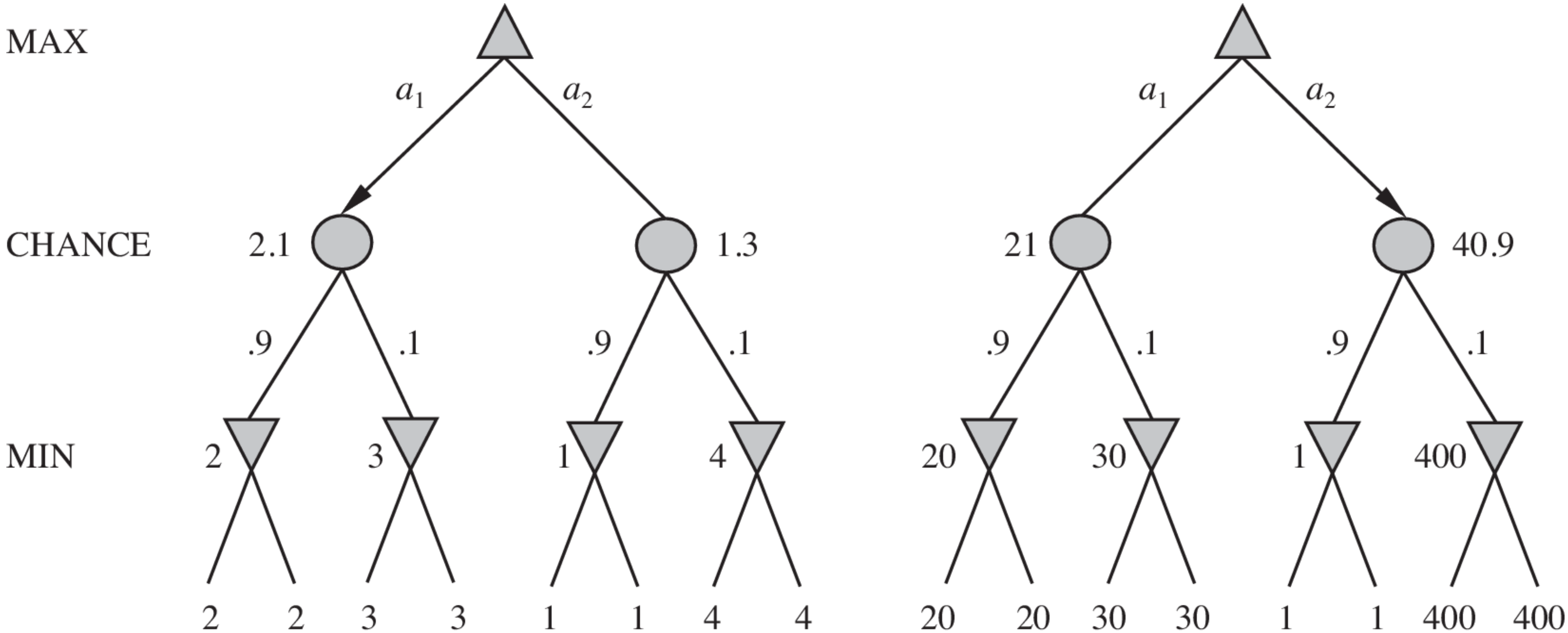
Mastering the game of Go with deep neural networks and tree search



stochastic games



use expected value through chance nodes in stochastic games



note: be careful of very small or large terminal values
when using mean expected value for intermediate nodes!
(because mean value are largely swayed by outliers)