ELEC/COMP 576:
Deep Reinforcement Learning

Ankit B. Patel
Baylor College of Medicine (Neuroscience Dept.)
Rice University (ECE Dept.)

Reinforcement Learning

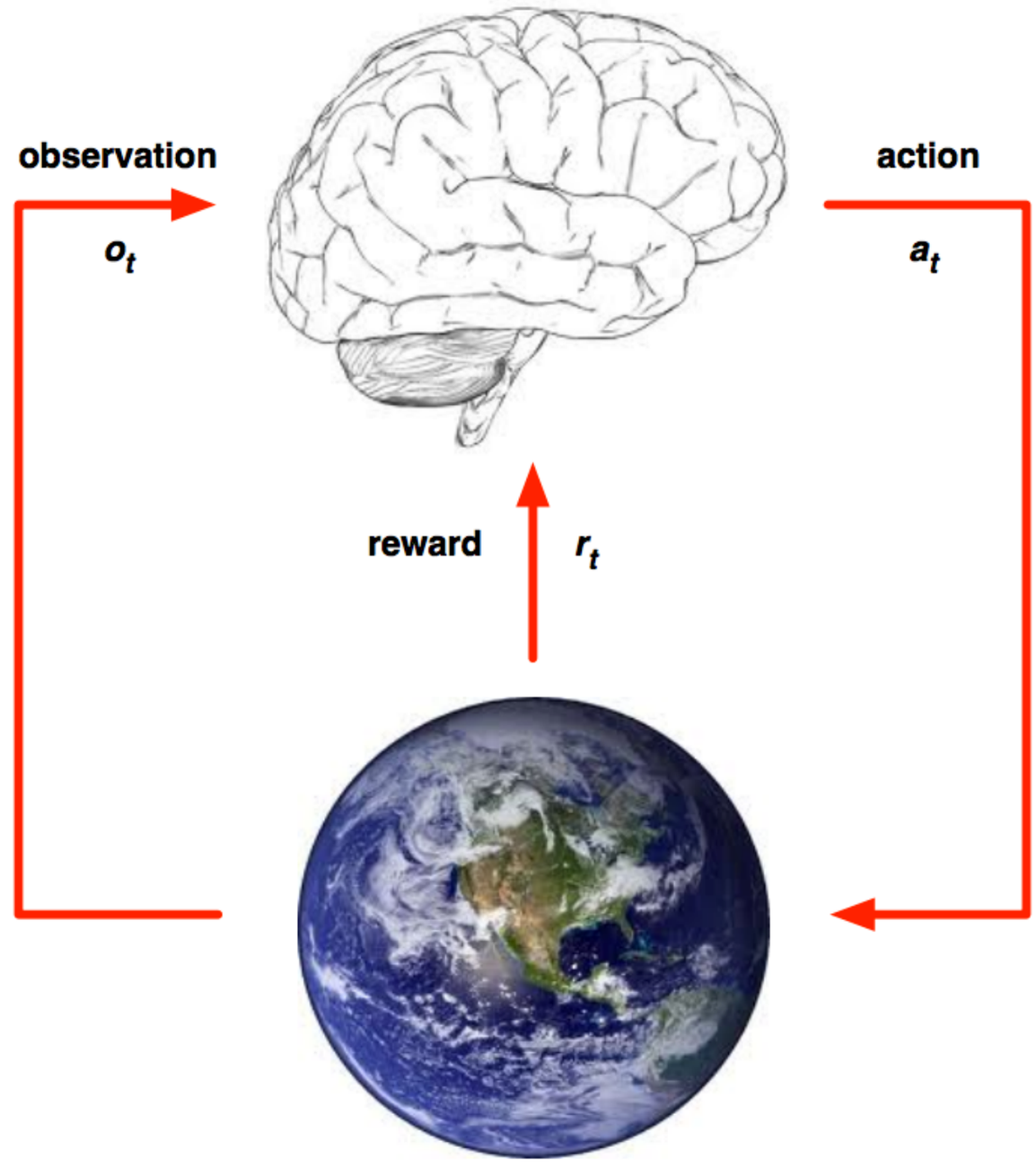
What is Reinforcement Learning

- Reinforcement Learning (RL) is a framework for decision-making

  - Have an agent with acts in an environment
  - Each action influences the agent's future state
  - Success is measured by reward
  - Find actions that maximise your future reward

Agent and Environment

- Agent
  - Executes action
  - Receives observation
  - Receives reward
- Environment
  - Receives action
  - Sends new observation
  - Sends new reward



[David Silver ICML 2016]

Defining the State

- Sequence of observations, rewards, and actions define experience

$$o_1, r_1, a_1, \ldots, a_{t-1}, o_t, r_t$$

- State is a summary of experiences

$$s_t = f(o_1, r_1, a_1, \ldots, a_{t-1}, o_t, r_t)$$

[David Silver ICML 2016]

Components of an RL Agent

- Policy
  - Agent's behavior function
- Value Function
  - Determine if each state or action is good
- Model
  - Agent's representation

Policy

- Policy is the agent's behavior
- Policy is a map from state to action
- Deterministic $a = \pi(s)$
- Stochastic $\pi(a|s) = \mathbb{P}[a|s]$

[David Silver ICML 2016]

Value Function

- Value function is a prediction of the future reward
  - What will my reward be given this action a from state s?
  - Example: Q-value function
    - State s, action a, policy π , discount factor γ

$$Q^{\pi}(s,a) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s,a\right]$$

[David Silver ICML 2016]

Optimal Value Function

- The goal is to have the maximum achievable

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) = Q^{\pi^*}(s,a)$$

- With that, can act optimally with the policy

$$\pi^*(s) = \operatorname*{argmax}_{a} Q^*(s,a)$$

- Optimal value maximises over all the decisions

$$Q^*(s,a) = r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} r_{t+3} + \ldots$$

$$= r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})$$

[David Silver ICML 2016]

Model

- The model is learned through experience
- Can act as a proxy for the environment

RL Approaches

- Value-based RL
  - Estimate the optimal value $Q^*(s, a)$
  - Maximum value achievable under any policy
- Policy-based RL
  - Search for the optimal value π^*
  - Policy achieving maximum future reward
- Model-based RL
  - Build a model of the environment
  - Plan using a model

[David Silver ICML 2016]

Deep Reinforcement Learning

Deep RL

- To have Deep RL, we need
  - RL + Deep Learning (DL) = AI
    - RL defines the objective
    - DL provides the mechanism to learn

Deep RL

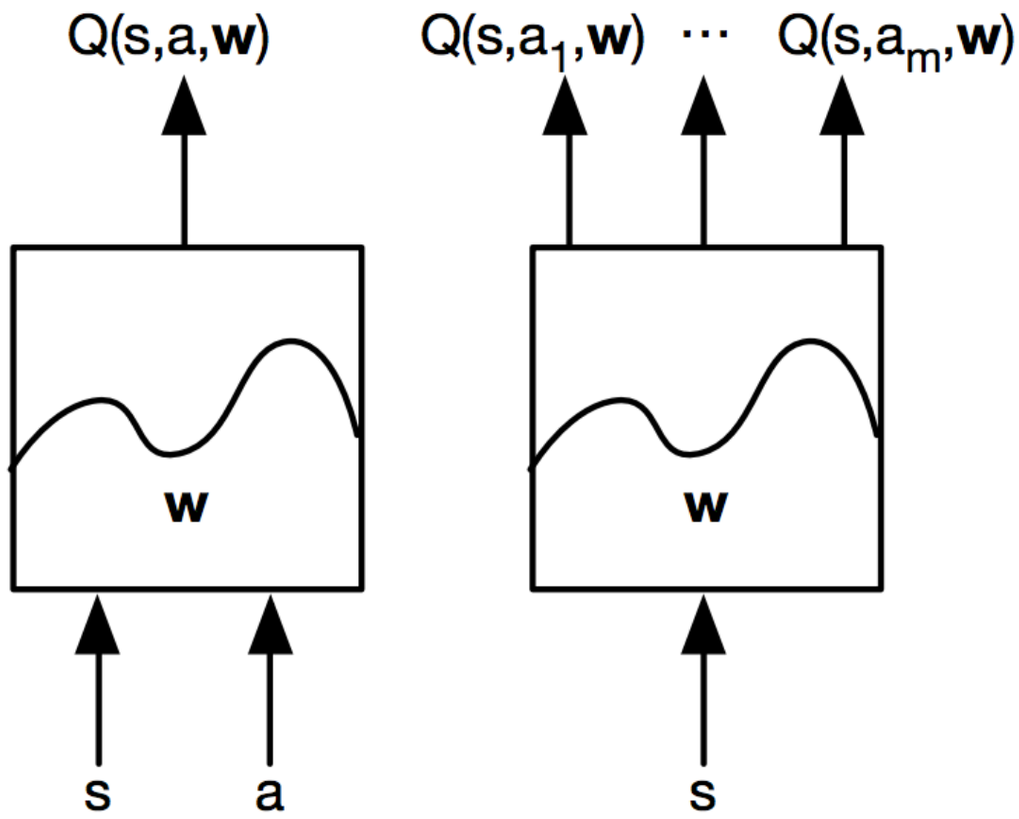
- Can use DL to represent
  - Value function
  - Policy
  - Model
- Optimise loss function via Stochastic Gradient Descent

Value-Based Deep Reinforcement Learning

Value-Based Deep RL

- The value function is represented by Q-network with weights **w**

$$Q(s, a, \mathbf{w}) \approx Q^*(s, a)$$



[David Silver ICML 2016]

Q-Learning

- Optimal Q-values obey Bellman Equation

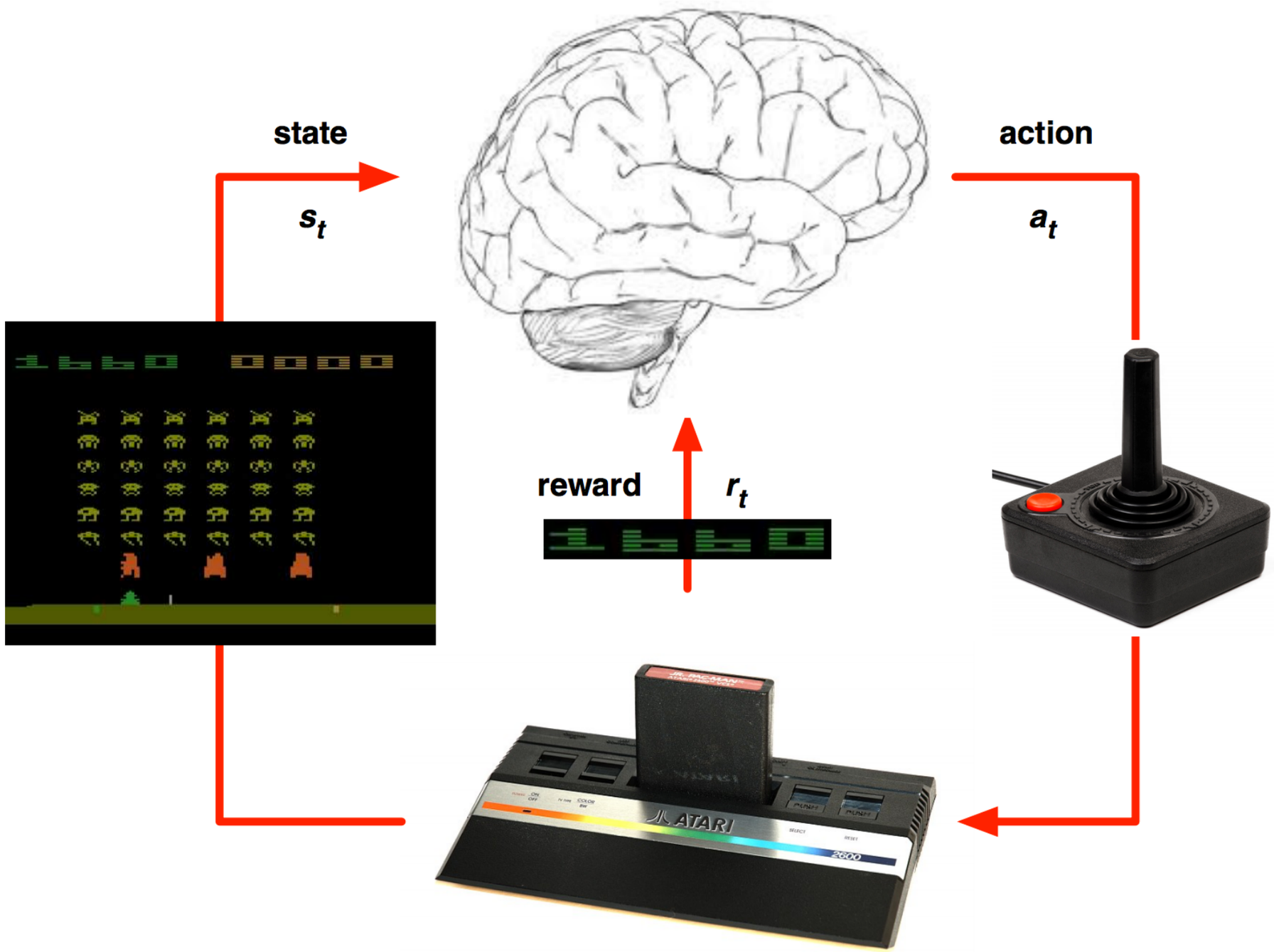
$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q(s',a')^* \mid s,a\right]$$

- Minimise MSE loss via SGD

$$I = \left(r + \gamma \max_{a} Q(s',a',\mathbf{w}) - Q(s,a,\mathbf{w})\right)^2$$

- Diverges due to
  - Correlations between samples
  - Non-stationary targets

DQN: Atari 2600



[David Silver ICML 2016]

Experience Replay

- Remove correlations
- Build dataset from agent's own experience
- Sample experiences from the dataset and apply update

$s_t, a_t, r_{t+1}, s_{t+1} \rightarrow$

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
...
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow \quad s, a, r, s'$

[David Silver ICML 2016]

Benefits of Experience Replay

- Greater data efficiency
- Breaks the correlations
- Smoothing out learning and avoiding oscillations or divergence in parameters

DQN: Atari 2600

- End-to-end learning of Q(s,a) from the frames
- Input state is stack of pixels from last 4 frames
- Output is Q(s,a) for the joystick/button positions
  - Varies with different games (~3-18)
- Reward is change in score for that step

DQN w/ Experience Replay

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For t = 1,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    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 D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

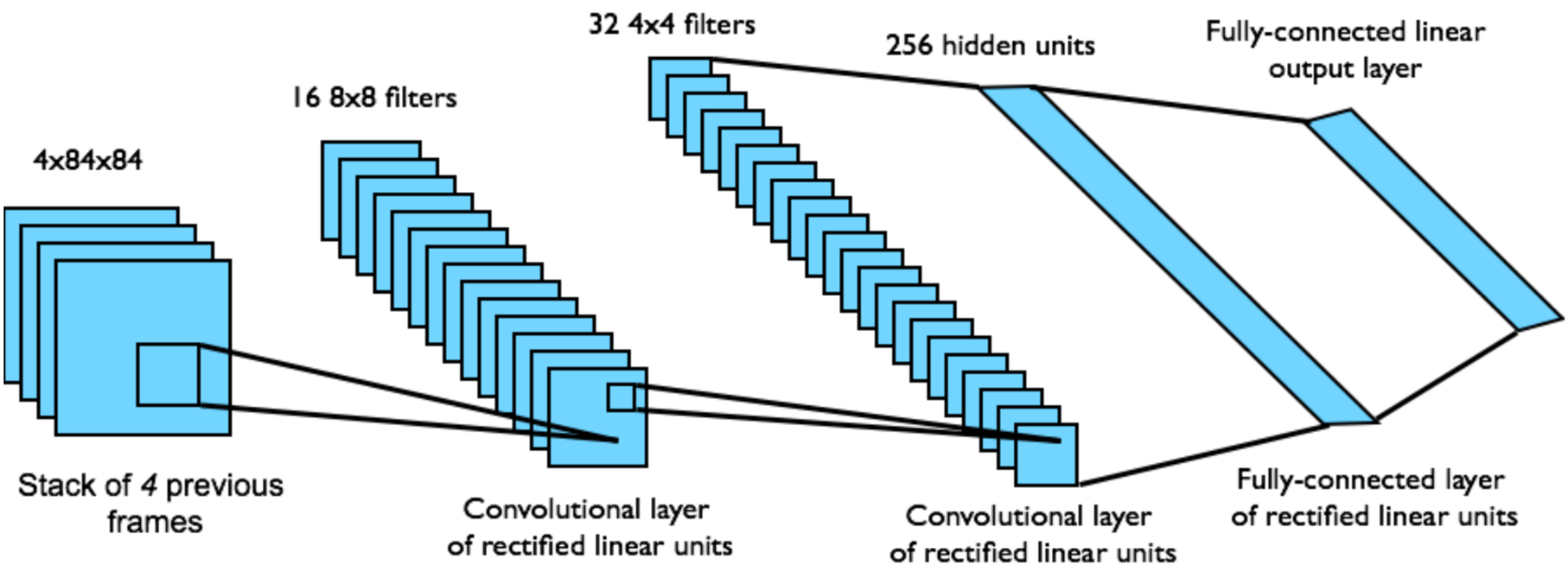
  End For

End For

[Deepmind 2014]

DQN: Atari 2600

- Network architecture and hyper parameters are fixed across all the games

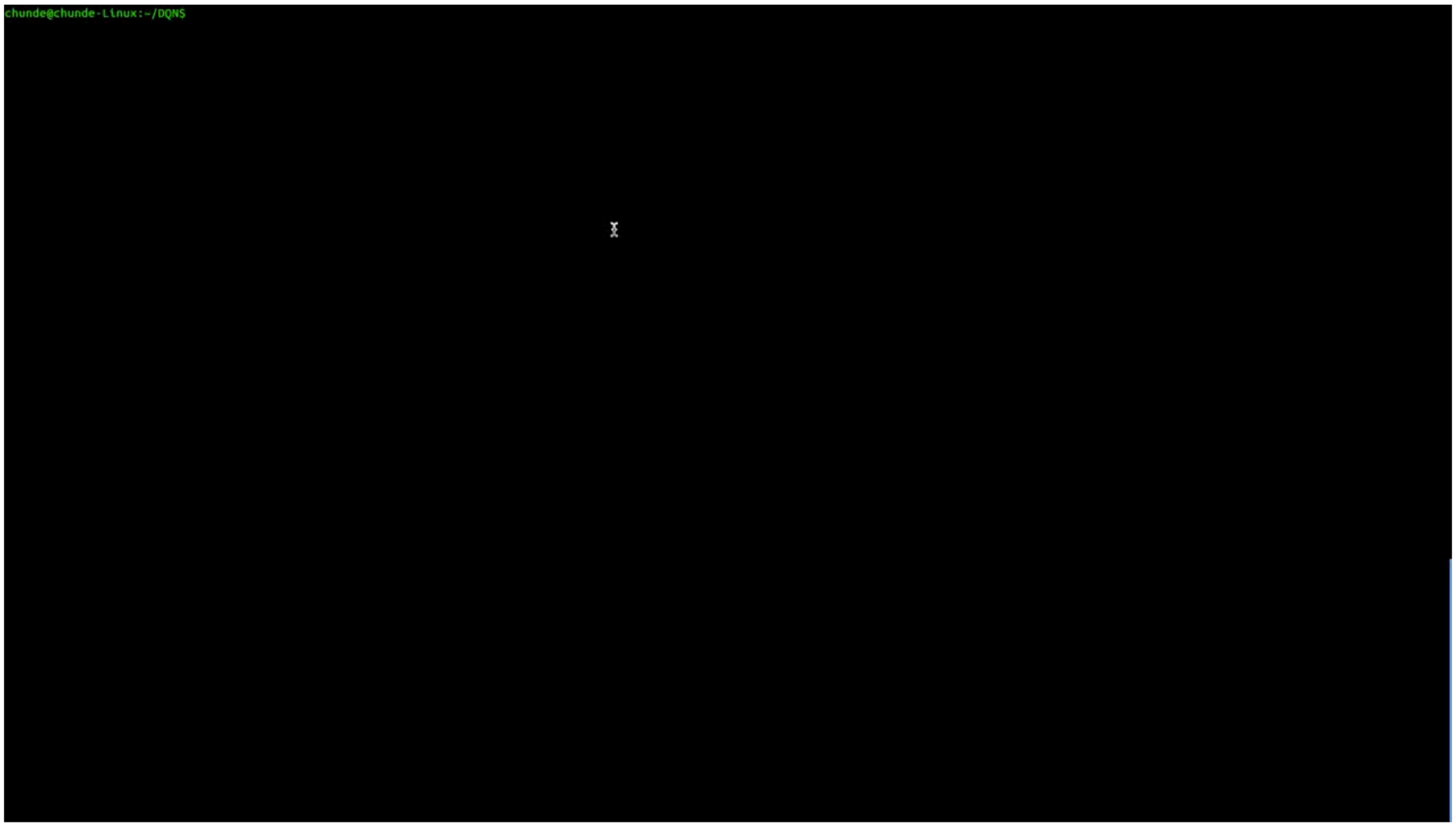


[David Silver ICML 2016]

DQN: Atari Video

[Jon Juett Youtube]

DQN: Atari Video



[PhysX Vision Youtube]

Improvements after DQN

- Double DQN
  - Current Q-network w is used to select actions
  - Older Q-network is w- is used to evaluate actions
- Prioritized reply
  - Store experience in priority queue by the DQN error

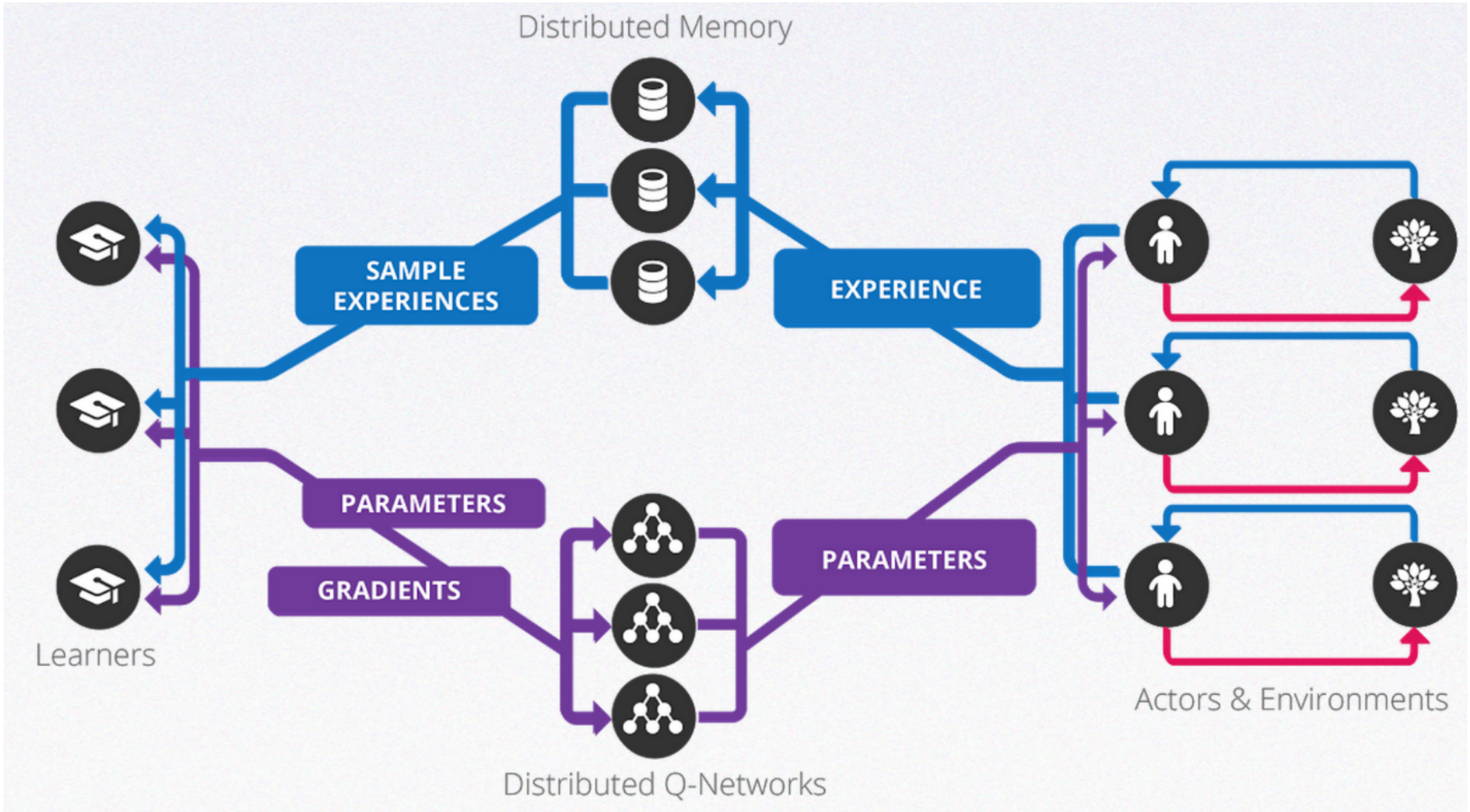
Improvements after DQN

- Duelling Network
  - Action-independent value function V(s,v)
  - Action-dependent advantage function A(s,a,**w**)
  - Q(s,a) = V(s,v) + A(s,a,**w**)

Improvements after DQN

[Marc G. Bellemare Youtube]

General Reinforcement Learning Architecture



[David Silver ICML 2016]

Policy-based Deep Reinforcement Learning

Deep Policy Network

- The policy is represented by a network with weights **u**

$$a = \pi(a|s, \mathbf{u}) \text{ or } a = \pi(s, \mathbf{u})$$

- Define objective function as total discounted reward

$$L(\mathbf{u}) = \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots \mid \pi(\cdot, \mathbf{u})\right]$$

- Optimise objectiva via SGD

- Adjust policy parameters **u** to achieve higher reward

[David Silver ICML 2016]

Policy Gradients

- Gradient of a stochastic policy

$$\frac{\partial L(\mathbf{u})}{\partial u} = \mathbb{E}\left[\frac{\partial \log \pi(a|s,\mathbf{u})}{\partial \mathbf{u}} Q^{\pi}(s,a)\right]$$

- Gradient of a deterministic policy

$$\frac{\partial L(\mathbf{u})}{\partial \mathbf{u}} = \mathbb{E}\left[\frac{\partial Q^{\pi}(s,a)}{\partial a}\frac{\partial a}{\partial \mathbf{u}}\right]$$

[David Silver ICML 2016]

Actor Critic Algorithm

- Estimate value function Q(s, a, **w**) $\approx Q^{\pi}(s, a)$
- Update policy parameters **u** via SGD

$$\frac{\partial l}{\partial \mathbf{u}} = \frac{\partial \log \pi(a|s, \mathbf{u})}{\partial \mathbf{u}} Q(s, a, \mathbf{w})$$

$$\frac{\partial l}{\partial \mathbf{u}} = \frac{\partial Q(s, a, \mathbf{w})}{\partial a} \frac{\partial a}{\partial \mathbf{u}}$$

[David Silver ICML 2016]

Asynchronous Advantage Actor-Critic: Labyrinth

[DeepMind Youtube]

A3C: Labyrinth

- End-to-end learning of softmax policy from raw pixels
- Observations are pixels of the current frame
- State is an LSTM $f(o_1, \ldots, o_t)$
- Outputs both value V(s) & softmax over actions $\pi(a|s)$
- Task is to collect apples (+1) and escape (+10)

[David Silver ICML 2016]

Improvements on the Basic DQN Algorithm

Double DQN

- Double DQN
  - Current Q-network w is used to select actions
  - Older Q-network is w- is used to evaluate actions
- Prioritized reply
  - Store experience in priority queue by the DQN error

Dueling Network

- Duelling Network
  - Action-independent value function V(s,v)
  - Action-dependent advantage function A(s,a,**w**)
  - Q(s,a) = V(s,v) + A(s,a,**w**)

Prioritized Experience Replay

Algorithm 1 Double DQN with proportional prioritization

Input: minibatch k, step-size η, replay period K and size N, exponents α and β, budget T.
Initialize replay memory $\mathcal{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$
Observe S_0 and choose $A_0 \sim \pi_\theta(S_0)$
for $t = 1$ **to** T **do**
    Observe S_t, R_t, γ_t
    Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$ with maximal priority $p_t = \max_{i<t} p_i$
    if $t \equiv 0 \mod K$ **then**
        for $j = 1$ **to** k **do**
            Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
            Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
            Compute TD-error $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
            Update transition priority $p_j \leftarrow |\delta_j|$
            Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
        end for
        Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
        From time to time copy weights into target network $\theta_{\text{target}} \leftarrow \theta$
    end if
    Choose action $A_t \sim \pi_\theta(S_t)$
end for

[Schaul et al 2016]

Prioritized Experience Replay

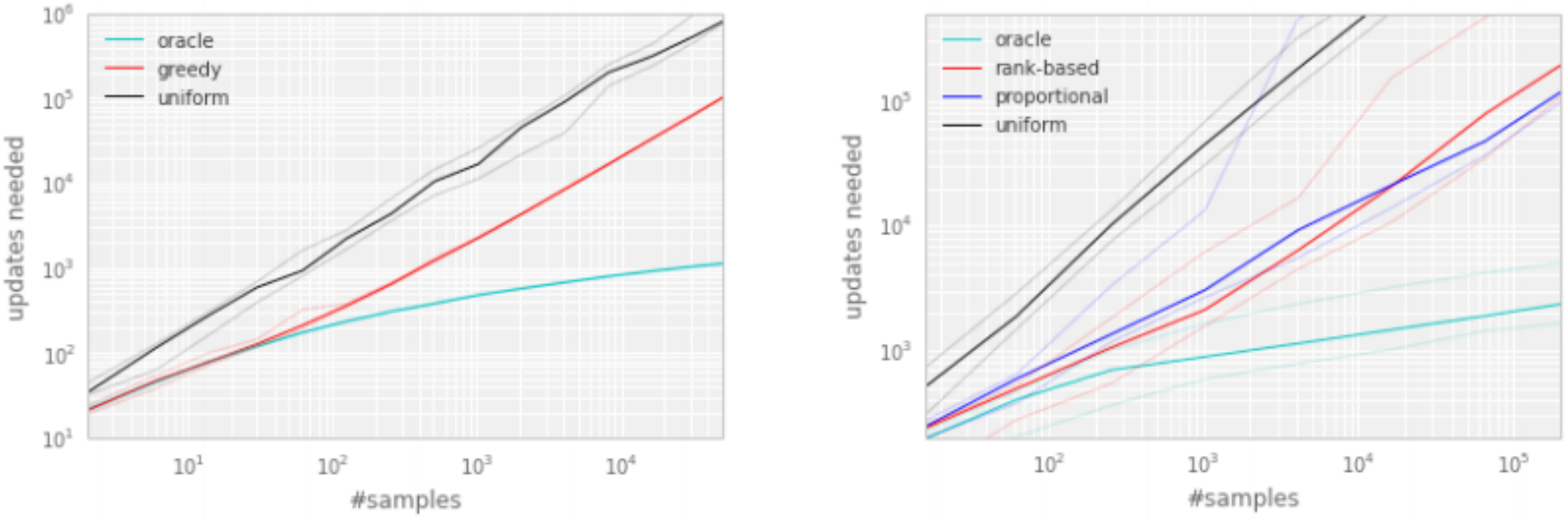


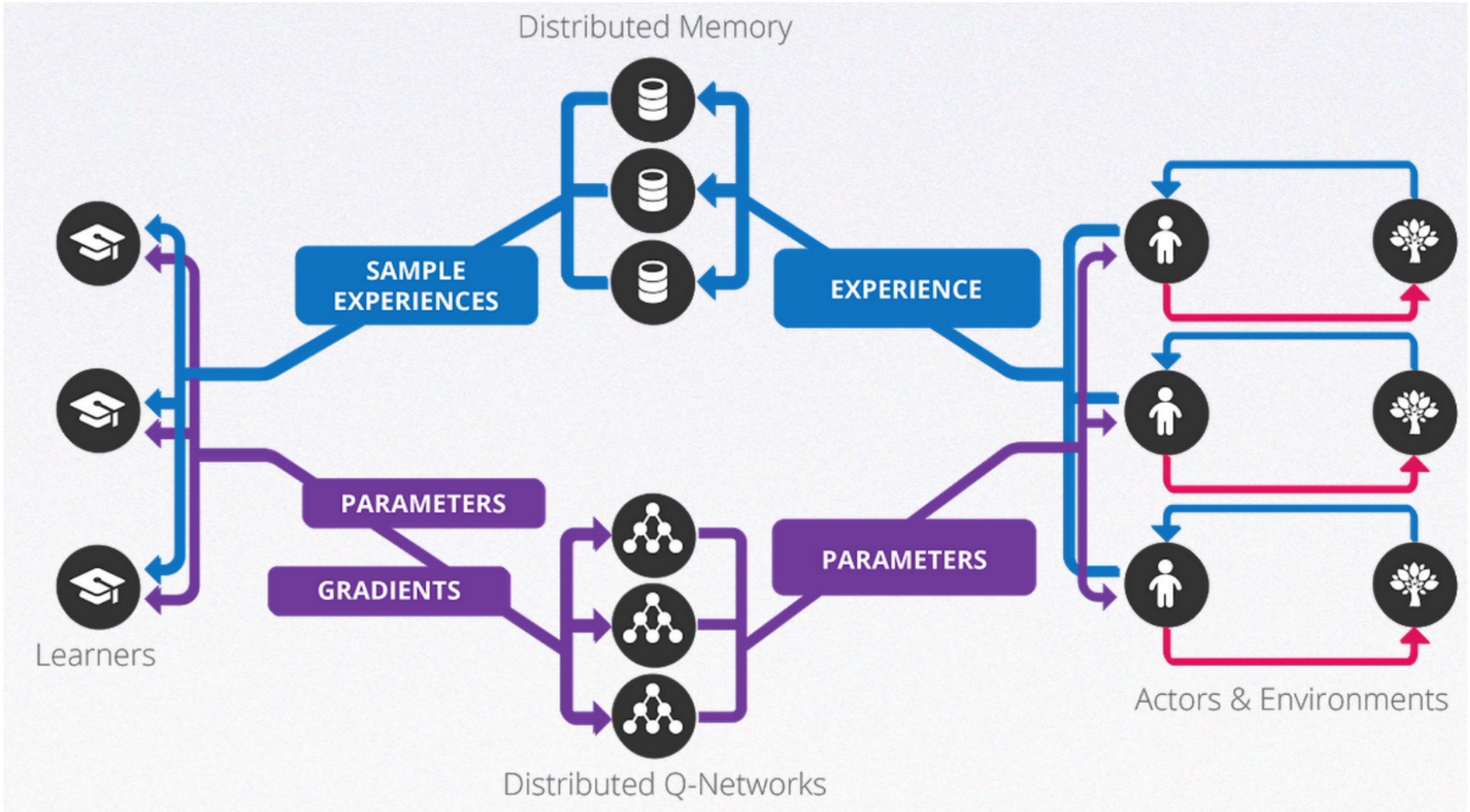
Figure 2: Median number of updates required for Q-learning to learn the value function on the Blind Cliffwalk example, as a function of the total number of transitions (only a single one of which was successful and saw the non-zero reward). Faint lines are min/max values from 10 random initializations. Black is uniform random replay, cyan uses the hindsight-oracle to select transitions, red and blue use prioritized replay (rank-based and proportional respectively). The results differ by multiple orders of magnitude, thus the need for a log-log plot. In both subplots it is evident that replaying experience in the right order makes an enormous difference to the number of updates required. See Appendix B.1 for details. **Left:** Tabular representation, greedy prioritization. **Right**: Linear function approximation, both variants of stochastic prioritization.

[Schaul et al 2016]

Improvements after DQN

[Marc G. Bellemare Youtube]

General Reinforcement Learning Architecture



[David Silver ICML 2016]

Policy-based Deep Reinforcement Learning

Deep Policy Network

- The policy is represented by a network with weights **u**

$$a = \pi(a|s, \mathbf{u}) \text{ or } a = \pi(s, \mathbf{u})$$

- Define objective function as total discounted reward

$$L(\mathbf{u}) = \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots \mid \pi(\cdot, \mathbf{u})\right]$$

- Optimise objective via SGD

- Adjust policy parameters **u** to achieve higher reward

[David Silver ICML 2016]

Policy Gradients

- Gradient of a stochastic policy

$$\frac{\partial L(\mathbf{u})}{\partial u} = \mathbb{E}\left[\frac{\partial \log \pi(a|s,\mathbf{u})}{\partial \mathbf{u}} Q^{\pi}(s,a)\right]$$

- Gradient of a deterministic policy

$$\frac{\partial L(\mathbf{u})}{\partial \mathbf{u}} = \mathbb{E}\left[\frac{\partial Q^{\pi}(s,a)}{\partial a}\frac{\partial a}{\partial \mathbf{u}}\right]$$

[David Silver ICML 2016]

Actor Critic Algorithm

- Estimate value function Q(s, a, **w**) $\approx Q^{\pi}(s, a)$
- Update policy parameters **u** via SGD

$$\frac{\partial l}{\partial \mathbf{u}} = \frac{\partial \log \pi(a|s, \mathbf{u})}{\partial \mathbf{u}} Q(s, a, \mathbf{w})$$

$$\frac{\partial l}{\partial \mathbf{u}} = \frac{\partial Q(s, a, \mathbf{w})}{\partial a} \frac{\partial a}{\partial \mathbf{u}}$$

[David Silver ICML 2016]

Asynchronous Advantage Actor-Critic: Labyrinth

[DeepMind Youtube]

A3C: Labyrinth

- End-to-end learning of softmax policy from raw pixels
- Observations are pixels of the current frame
- State is an LSTM $f(o_1, \ldots, o_t)$
- Outputs both value V(s) & softmax over actions $\pi(a|s)$
- Task is to collect apples (+1) and escape (+10)

[David Silver ICML 2016]

Model-based Deep Reinforcement Learning

Learning Models of the Environment

- Generative model of Atari 2600

- Issues
  - Errors in transition model compound over the trajectory
  - Planning trajectory differ from executed trajectories
  - Long, unusual trajectory rewards are totally wrong

[David Silver ICML 2016]

Learning Models of the Environment

[Junhyuk Oh Youtube]

Newer

Implementations

AlphaGo

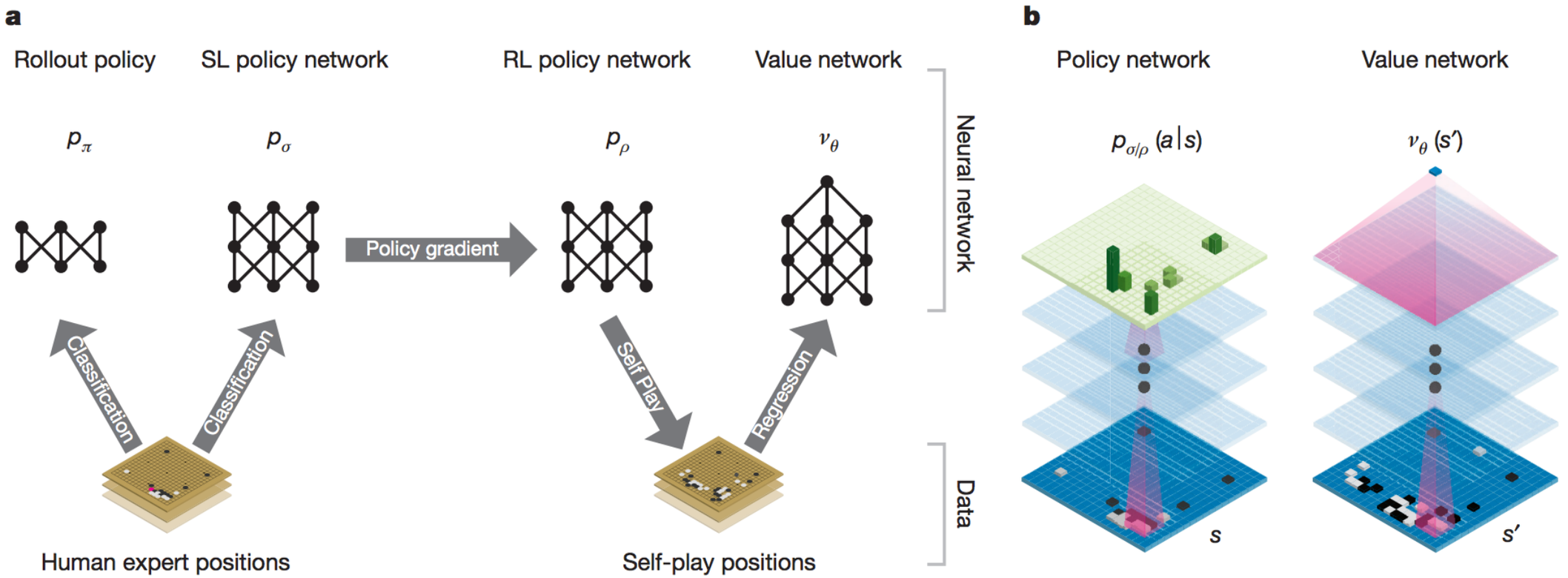


Figure 1 | Neural network training pipeline and architecture. a, A fast rollout policy p_π and supervised learning (SL) policy network p_σ are trained to predict human expert moves in a data set of positions. A reinforcement learning (RL) policy network p_ρ is initialized to the SL policy network, and is then improved by policy gradient learning to maximize the outcome (that is, winning more games) against previous versions of the policy network. A new data set is generated by playing games of self-play with the RL policy network. Finally, a value network v_θ is trained by regression to predict the expected outcome (that is, whether the current player wins) in positions from the self-play data set. **b**, Schematic representation of the neural network architecture used in AlphaGo. The policy network takes a representation of the board position s as its input, passes it through many convolutional layers with parameters σ (SL policy network) or ρ (RL policy network), and outputs a probability distribution $p_\sigma(a|s)$ or $p_\rho(a|s)$ over legal moves a, represented by a probability map over the board. The value network similarly uses many convolutional layers with parameters θ, but outputs a scalar value $v_\theta(s')$ that predicts the expected outcome in position s'.

[Deepmind Nature]

Policy and Value Networks

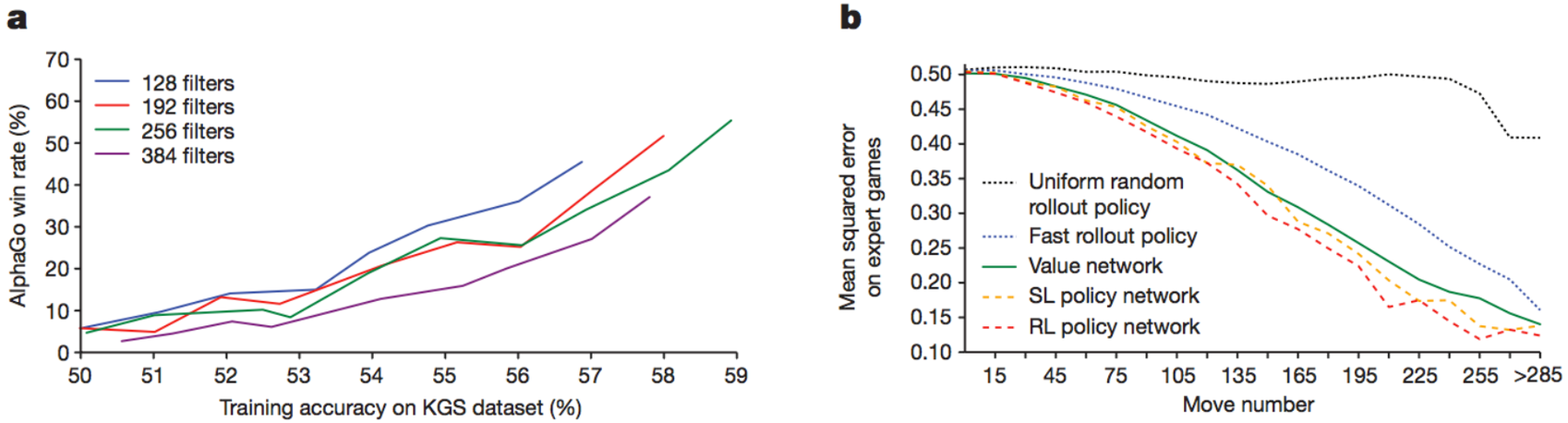


Figure 2 | Strength and accuracy of policy and value networks. **a**, Plot showing the playing strength of policy networks as a function of their training accuracy. Policy networks with 128, 192, 256 and 384 convolutional filters per layer were evaluated periodically during training; the plot shows the winning rate of AlphaGo using that policy network against the match version of AlphaGo. **b**, Comparison of evaluation accuracy between the value network and rollouts with different policies. Positions and outcomes were sampled from human expert games. Each position was evaluated by a single forward pass of the value network v_θ, or by the mean outcome of 100 rollouts, played out using either uniform random rollouts, the fast rollout policy p_π, the SL policy network p_σ or the RL policy network p_ρ. The mean squared error between the predicted value and the actual game outcome is plotted against the stage of the game (how many moves had been played in the given position).

[Deepmind Nature]

Monte Carlo Tree Search

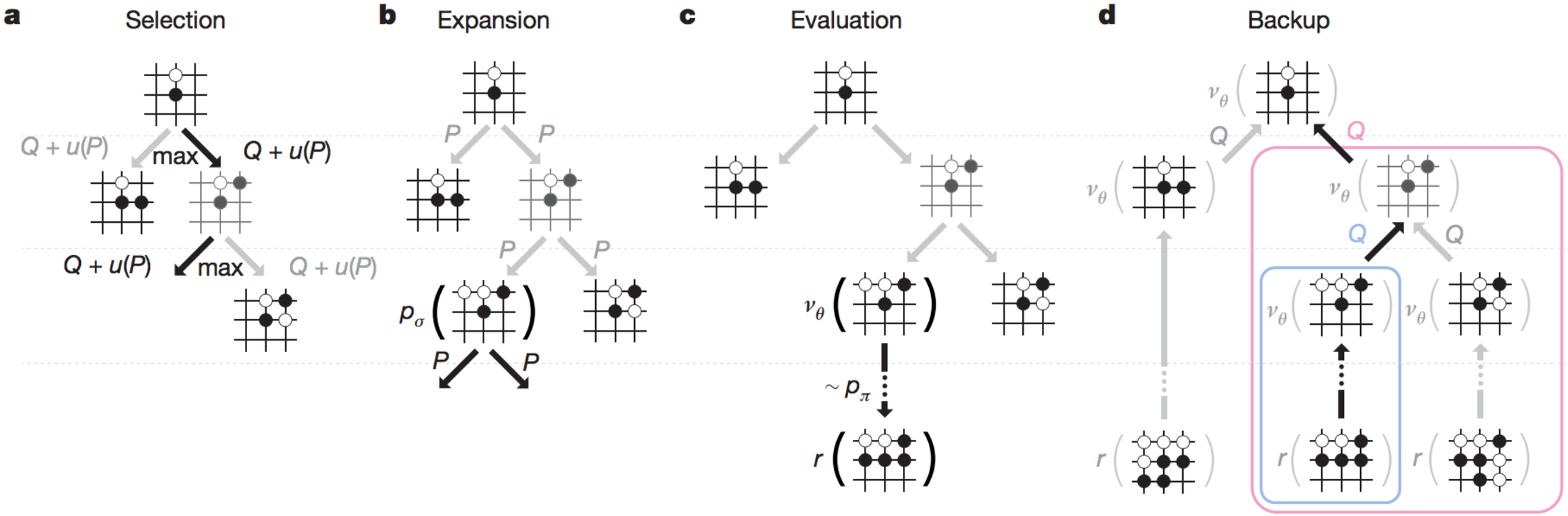


Figure 3 | Monte Carlo tree search in AlphaGo. a, Each simulation traverses the tree by selecting the edge with maximum action value Q, plus a bonus $u(P)$ that depends on a stored prior probability P for that edge. **b**, The leaf node may be expanded; the new node is processed once by the policy network 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.

[Deepmind Nature]

AlphaGo Results

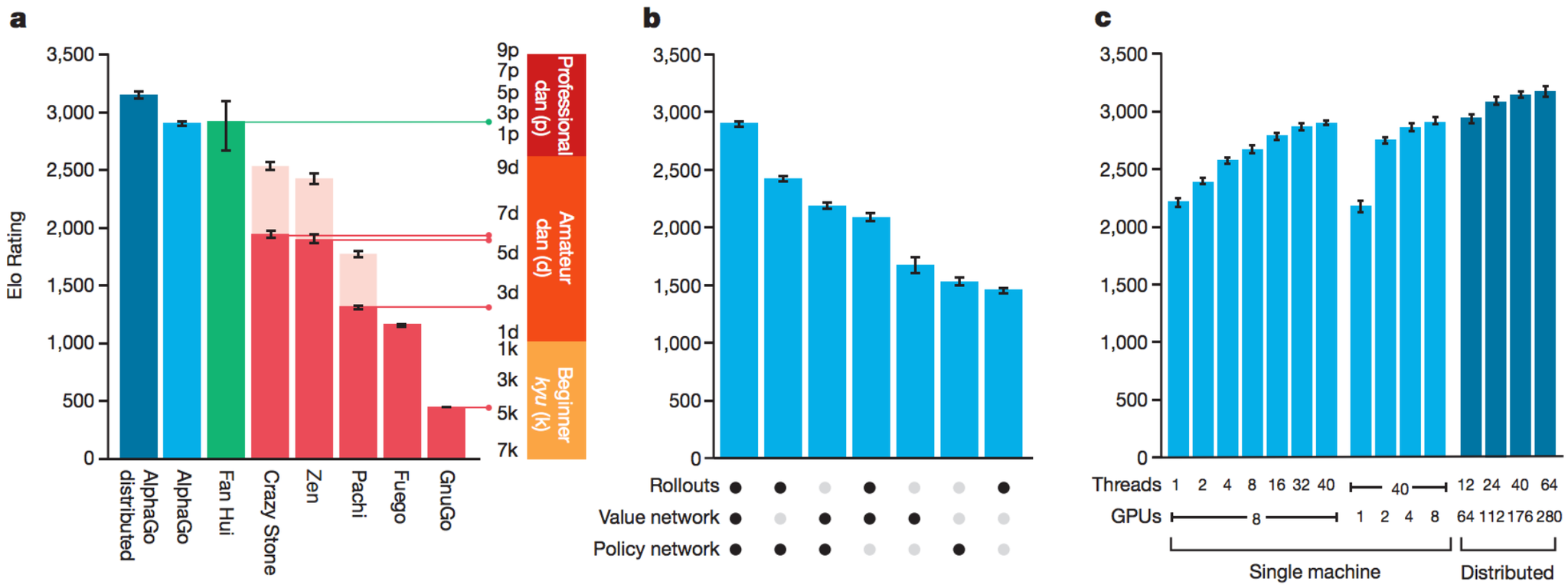


Figure 4 | Tournament evaluation of AlphaGo. a, Results of a tournament between different Go programs (see Extended Data Tables 6–11). Each program used approximately 5 s computation time per move. To provide a greater challenge to AlphaGo, some programs (pale upper bars) were given four handicap stones (that is, free moves at the start of every game) against all opponents. Programs were evaluated on an Elo scale[37]: a 230 point gap corresponds to a 79% probability of winning, which roughly corresponds to one amateur *dan* rank advantage on KGS[38]; an approximate correspondence to human ranks is also shown, horizontal lines show KGS ranks achieved online by that program. Games against the human European champion Fan Hui were also included; these games used longer time controls. 95% confidence intervals are shown. **b**, Performance of AlphaGo, on a single machine, for different combinations of components. The version solely using the policy network does not perform any search. **c**, Scalability study of MCTS in AlphaGo with search threads and GPUs, using asynchronous search (light blue) or distributed search (dark blue), for 2 s per move.

[Deepmind Nature]

Which Move to Make

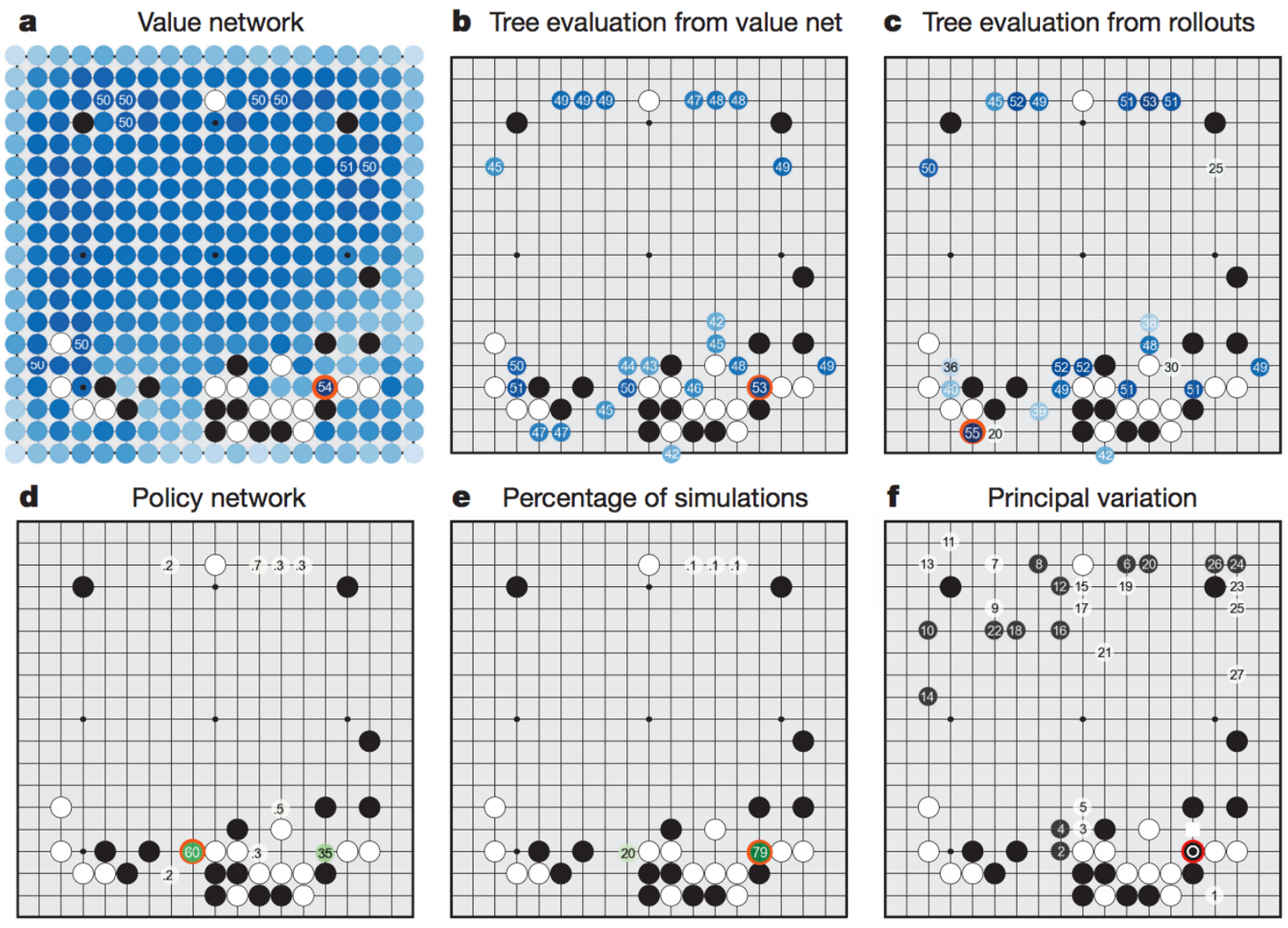
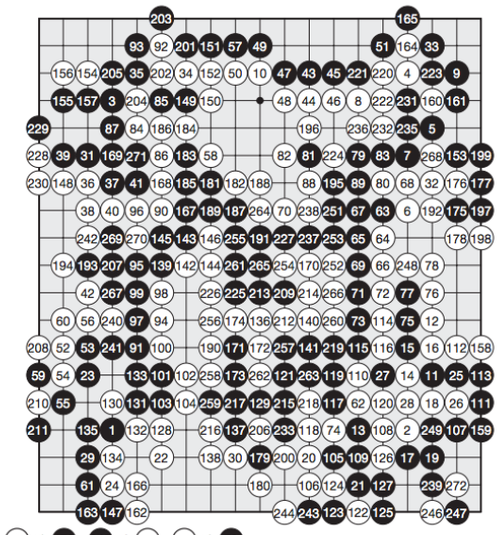


Figure 5 | How AlphaGo (black, to play) selected its move in an informal game against Fan Hui. For each of the following statistics, the location of the maximum value is indicated by an orange circle. **a**, Evaluation of all successors s' of the root position s, using the value network $v_\theta(s')$; 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.

[Deepmind Nature]

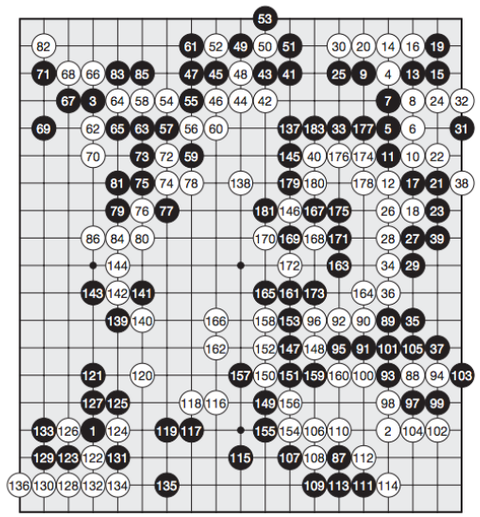
Five Matches Against Fan Hui

Game 1
Fan Hui (Black), AlphaGo (White)
AlphaGo wins by 2.5 points



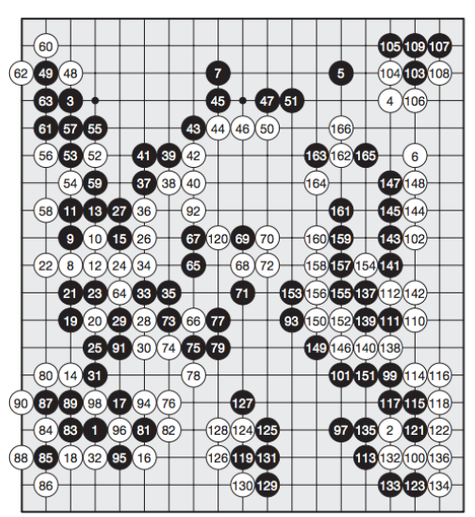
234 at 179 245 at 122 250 at 59

Game 2
AlphaGo (Black), Fan Hui (White)
AlphaGo wins by resignation

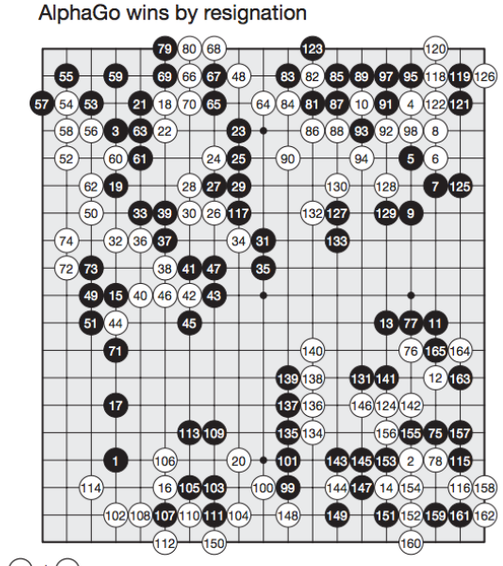


182 at 169

Game 3
Fan Hui (Black), AlphaGo (White)
AlphaGo wins by resignation

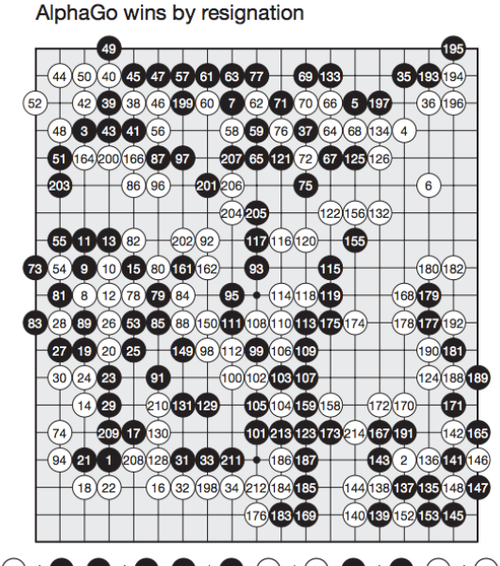


Game 4
AlphaGo (Black), Fan Hui (White)
AlphaGo wins by resignation



96 at 10

Game 5
Fan Hui (Black), AlphaGo (White)
AlphaGo wins by resignation



90 at 15 127 at 37 151 at 141 154 at 148 157 at 141 160 at 148 163 at 141

[Deepmind Nature]

Neural Architecture Search with Reinforcement Learning

- Uses an RNN to generate model descriptions of the NNs

- Trains the RNN with RL to maximize expected accuracy of generated architectures on validation set

Neural Architecture Search

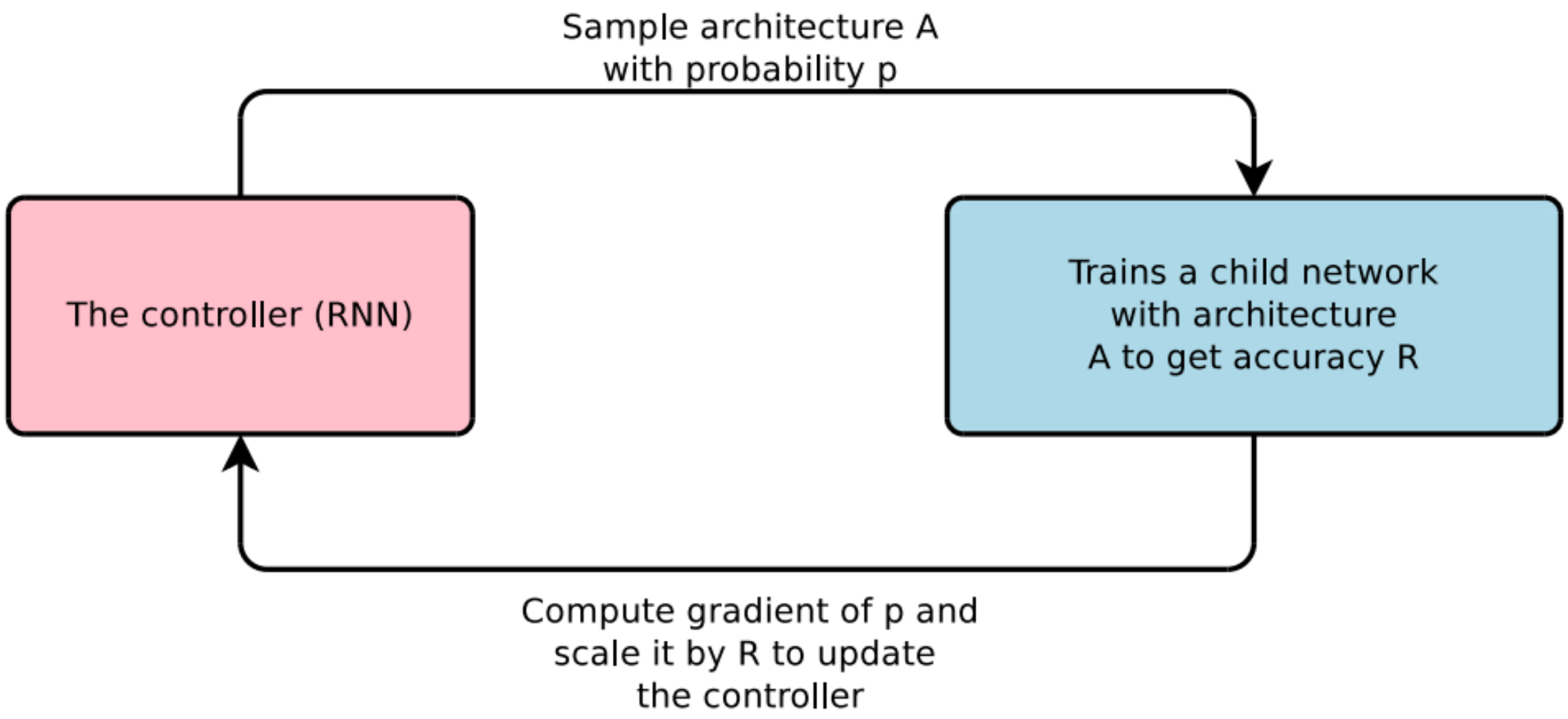


Figure 1: An overview of Neural Architecture Search.

[Zoph & Le Arxiv 2016]

RNN Controller

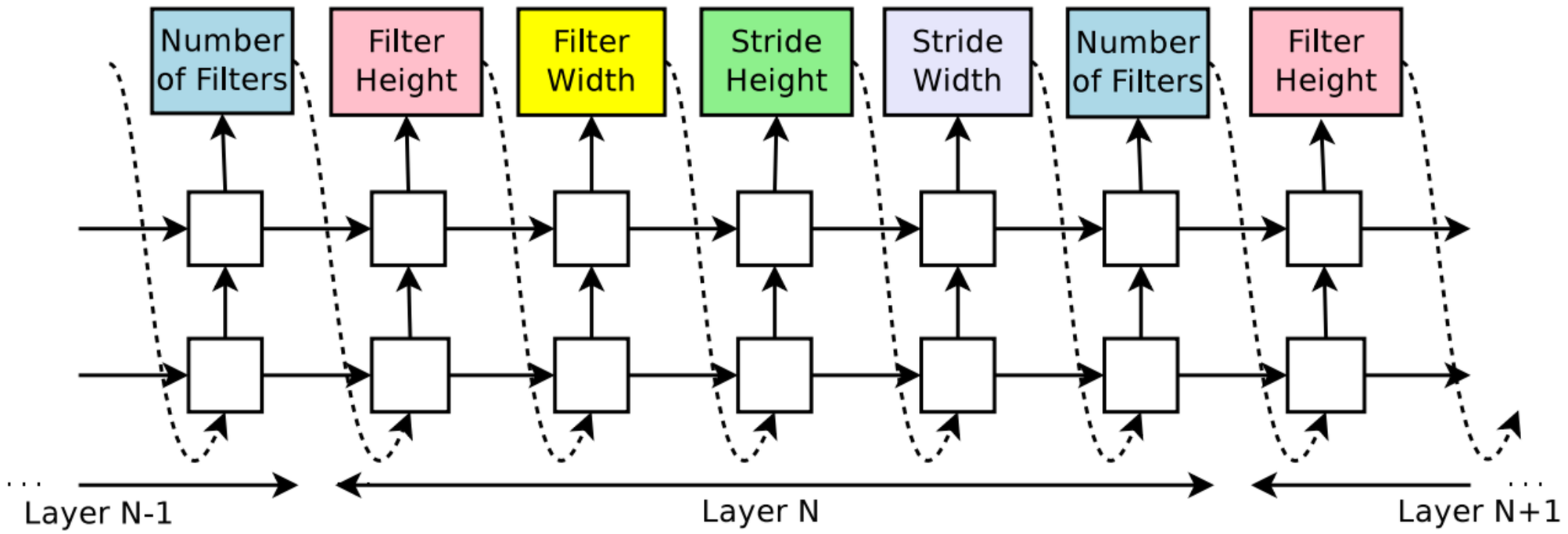


Figure 2: How our controller recurrent neural network samples a simple convolutional network. It predicts filter height, filter width, stride height, stride width, and number of filters for one layer and repeats. Every prediction is carried out by a softmax classifier and then fed into the next time step as input.

[Zoph & Le Arxiv 2016]

Training with REINFORCE

$$J(\theta_c) = E_{P(a_{1:T};\theta_c)}[R]$$

$$\nabla_{\theta_c} J(\theta_c) = \sum_{t=1}^{T} E_{P(a_{1:T};\theta_c)}\big[\nabla_{\theta_c} \log P(a_t | a_{(t-1):1}; \theta_c) R\big]$$

$$\frac{1}{m} \sum_{k=1}^{m} \sum_{t=1}^{T} \nabla_{\theta_c} \log P(a_t | a_{(t-1):1}; \theta_c) R_k$$

Where m is the number of different architectures that the controller samples in one batch and T is the number of hyperparameters our controller has to predict to design a neural network architecture.

[Zoph & Le Arxiv 2016]

Parallel Training

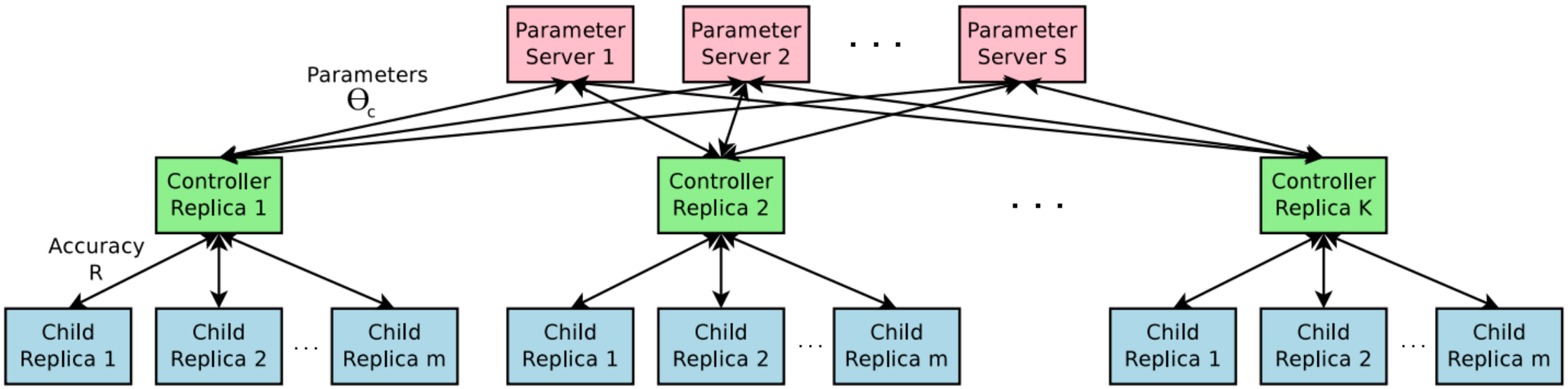


Figure 3: Distributed training for Neural Architecture Search. We use a set of S parameter servers to store and send parameters to K controller replicas. Each controller replica then samples m architectures and run the multiple child models in parallel. The accuracy of each child model is recorded to compute the gradients with respect to θ_c, which are then sent back to the parameter servers.

[Zoph & Le Arxiv 2016]

Increase Architecture Complexity

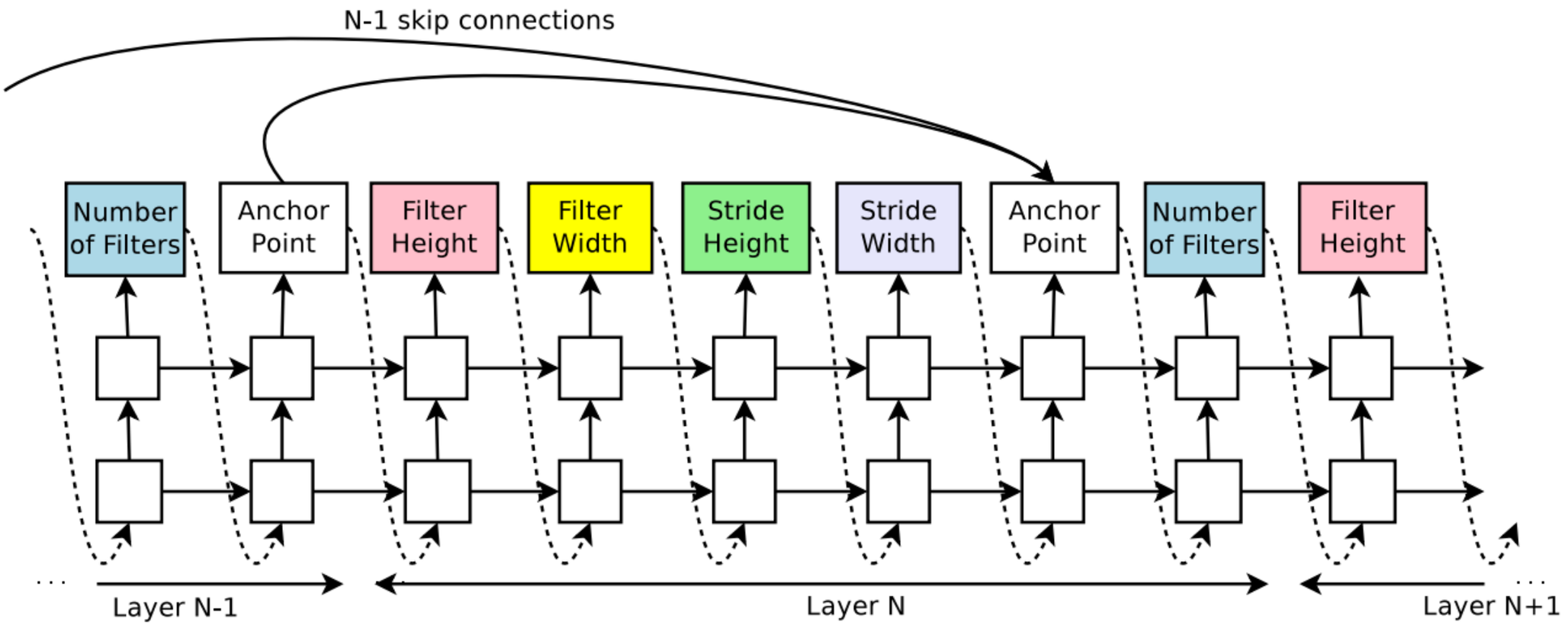


Figure 4: The controller uses anchor points, and set-selection attention to form skip connections.

[Zoph & Le Arxiv 2016]

Constructing the Network

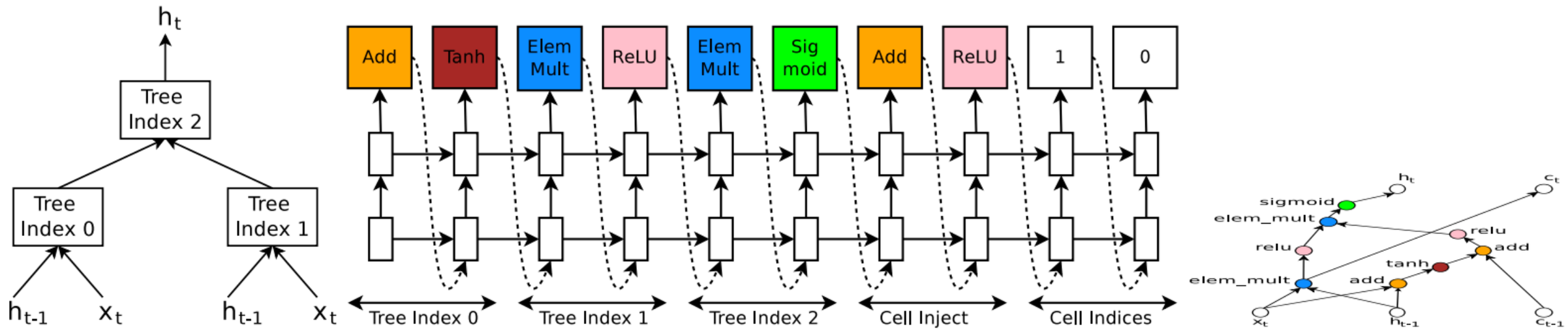


Figure 5: An example of a recurrent cell constructed from a tree that has two leaf nodes (base 2) and one internal node. Left: the tree that defines the computation steps to be predicted by controller. Center: an example set of predictions made by the controller for each computation step in the tree. Right: the computation graph of the recurrent cell constructed from example predictions of the controller.

[Zoph & Le Arxiv 2016]

Results

Model	Depth	Parameters	Error rate (%)
Network in Network (Lin et al., 2013)	-	-	8.81
All-CNN (Springenberg et al., 2014)	-	-	7.25
Deeply Supervised Net (Lee et al., 2015)	-	-	7.97
Highway Network (Srivastava et al., 2015)	-	-	7.72
Scalable Bayesian Optimization (Snoek et al., 2015)	-	-	6.37
FractalNet (Larsson et al., 2016)	21	38.6M	5.22
with Dropout/Drop-path	21	38.6M	4.60
ResNet (He et al., 2016a)	110	1.7M	6.61
ResNet (reported by Huang et al. (2016b))	110	1.7M	6.41
ResNet with Stochastic Depth (Huang et al., 2016b)	110	1.7M	5.23
	1202	10.2M	4.91
Wide ResNet (Zagoruyko & Komodakis, 2016)	16	11.0M	4.81
	28	36.5M	4.17
ResNet (pre-activation) (He et al., 2016b)	164	1.7M	5.46
	1001	10.2M	4.62
DenseNet ($L = 40, k = 12$) Huang et al. (2016a)	40	1.0M	5.24
DenseNet($L = 100, k = 12$) Huang et al. (2016a)	100	7.0M	4.10
DenseNet ($L = 100, k = 24$) Huang et al. (2016a)	100	27.2M	3.74
Neural Architecture Search v1 no stride or pooling	15	4.2M	5.50
Neural Architecture Search v2 predicting strides	20	2.5M	6.01
Neural Architecture Search v3 max pooling	39	7.1M	4.47
Neural Architecture Search v3 max pooling + more filters	39	32.0M	3.84

Table 1: Performance of Neural Architecture Search and other state-of-the-art models on CIFAR-10.

[Zoph & Le Arxiv 2016]

Reinforcement Learning with Unsupervised Auxiliary Tasks

- Train an agent that maximises other pseudo-reward functions simultaneously by RL
    - Those tasks have to develop in absence of extrinsic rewards
- Outperforms previous state-of-the-art on atari
    - Averaging 880% expert human performance

UNREAL Agent

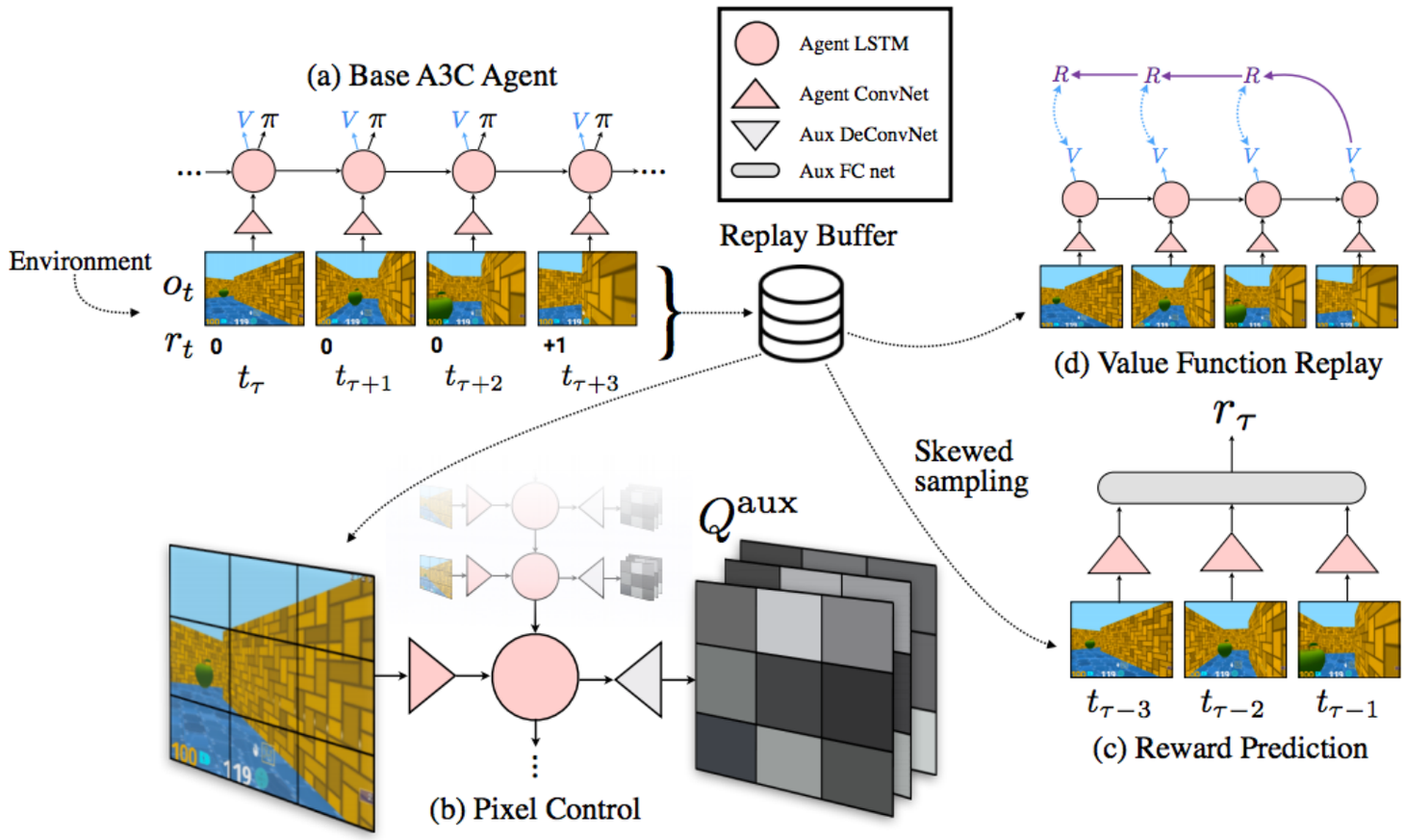


Figure 1: Overview of the *UNREAL* agent. (a) The base agent is a CNN-LSTM agent trained on-policy with the A3C loss (Mnih et al., 2016). Observations, rewards, and actions are stored in a small replay buffer which encapsulates a short history of agent experience. This experience is used by auxiliary learning tasks. (b) Pixel Control – auxiliary policies Q^{aux} are trained to maximise change in pixel intensity of different regions of the input. The agent CNN and LSTM are used for this task along with an auxiliary deconvolution network. This auxiliary control task requires the agent to learn how to control the environment. (c) Reward Prediction – given three recent frames, the network must predict the reward that will be obtained in the next unobserved timestep. This task network uses instances of the agent CNN, and is trained on reward biased sequences to remove the perceptual sparsity of rewards. (d) Value Function Replay – further training of the value function using the agent network is performed to promote faster value iteration. Further visualisation of the agent can be found in `https://youtu.be/Uz-zGYrYEjA`

[Jaderberg et. al Arxiv 2016]

Auxiliary Control Tasks

Given a set of auxiliary control tasks $\mathcal{C}$, let $\pi^{(c)}$ be the agent's policy for each auxiliary task $c \in \mathcal{C}$ and let π be the agent's policy on the base task. The overall objective is to maximise total performance across all these auxiliary tasks,

$$\arg\max_{\theta} \mathbb{E}_{\pi}[R_{1:\infty}] + \lambda_c \sum_{c \in \mathcal{C}} \mathbb{E}_{\pi_c}[R^{(c)}_{1:\infty}], \tag{1}$$

$R^{(c)}_{t:t+n} = \sum_{k=1}^{n} \gamma^k r^{(c)}_t$ is the discounted return for auxiliary reward $r^{(c)}$

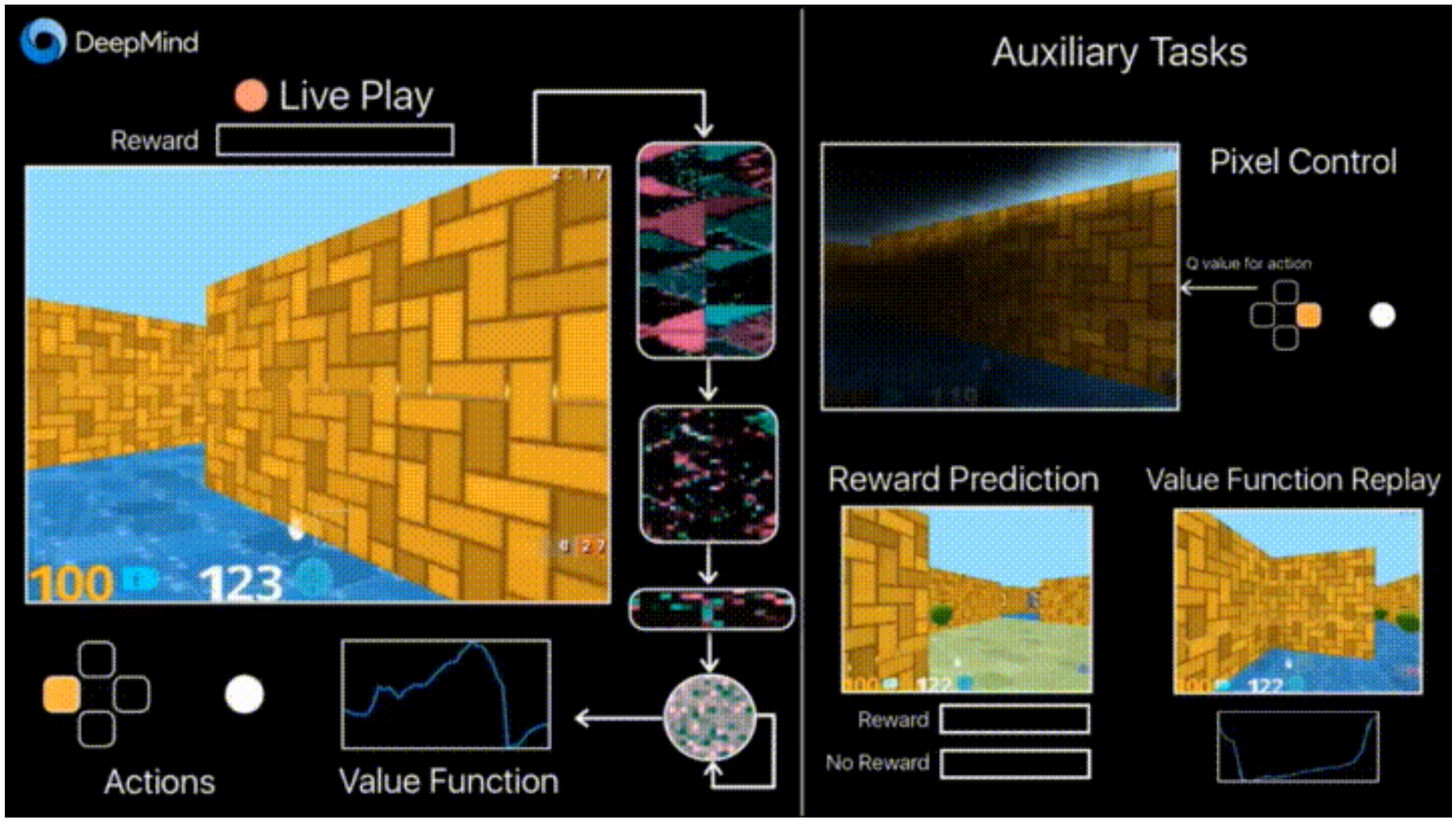
θ is the set of parameters of π and all $\pi^{(c)}$'s

n-step Q-learning loss $\mathcal{L}^{(c)}_Q =$

$$\mathbb{E}\left[\left(R_{t:t+n} + \gamma^n \max_{a'} Q^{(c)}(s', a', \theta^-) - Q^{(c)}(s, a, \theta)\right)^2\right]$$

[Jaderberg et. al Arxiv 2016]

Auxiliary Tasks



[Jaderberg et. al Arxiv 2016]

UNREAL Algorithm

$$\mathcal{L}_{UNREAL}(\theta) = \mathcal{L}_{\mathrm{A3C}} + \lambda_{\mathrm{VR}}\mathcal{L}_{\mathrm{VR}} + \lambda_{\mathrm{PC}}\sum_{c}\mathcal{L}_{Q}^{(c)} + \lambda_{\mathrm{RP}}\mathcal{L}_{\mathrm{RP}}$$

auxiliary control loss $\mathcal{L}_{\mathrm{PC}}$

auxiliary reward prediction loss $\mathcal{L}_{\mathrm{RP}}$

replayed value loss $\mathcal{L}_{\mathrm{VR}}$

A3C Loss is minimised on policy

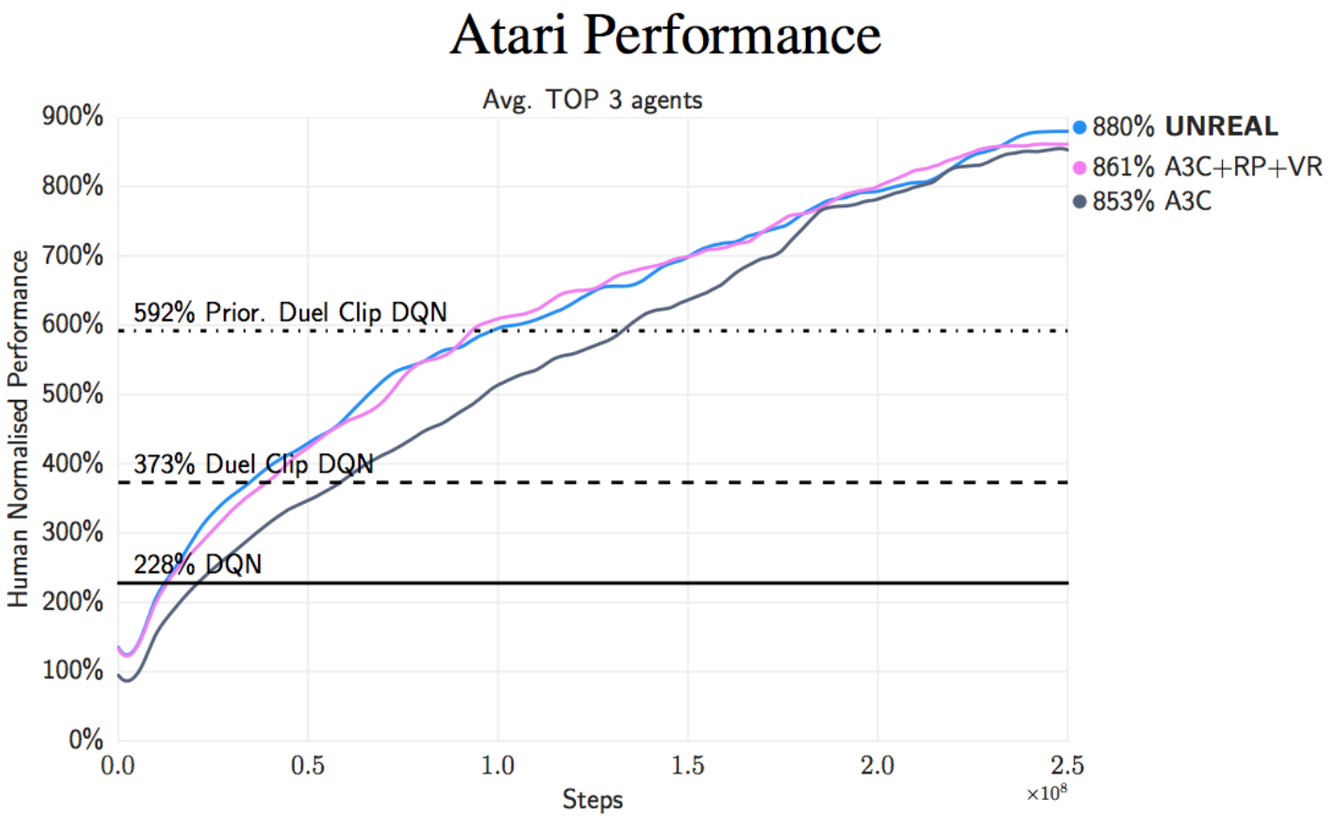
Value function is optimised from replayed data

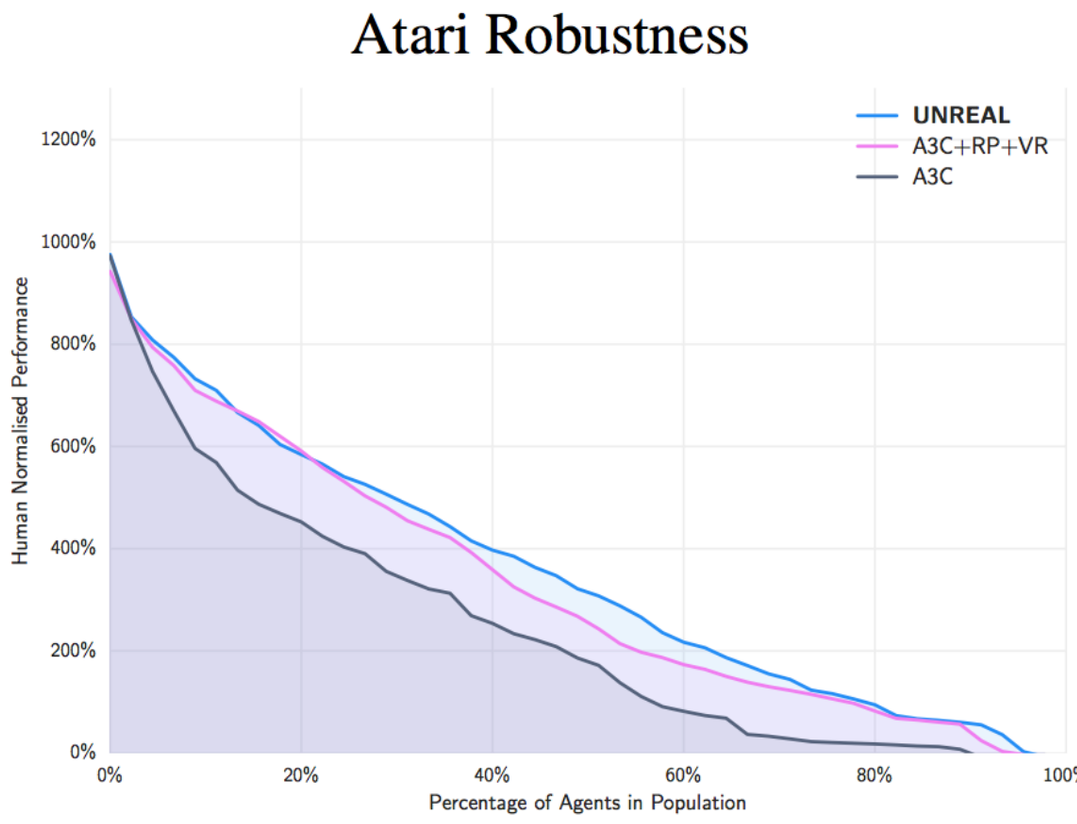
Auxiliary control loss is optimised off-policy from replied data

Reward loss is optimised from rebalanced replay data

[Jaderberg et. al Arxiv 2016]

Atari Results





[Jaderberg et. al Arxiv 2016]

Reinforcement Learning with Unsupervised Auxiliary Tasks Video

[DeepMind Youtube]

Deep Sensorimotor Learning

https://research.googleblog.com/2016/03/deep-learning-for-robots-learning-from.html

Deep Learning:
The Future

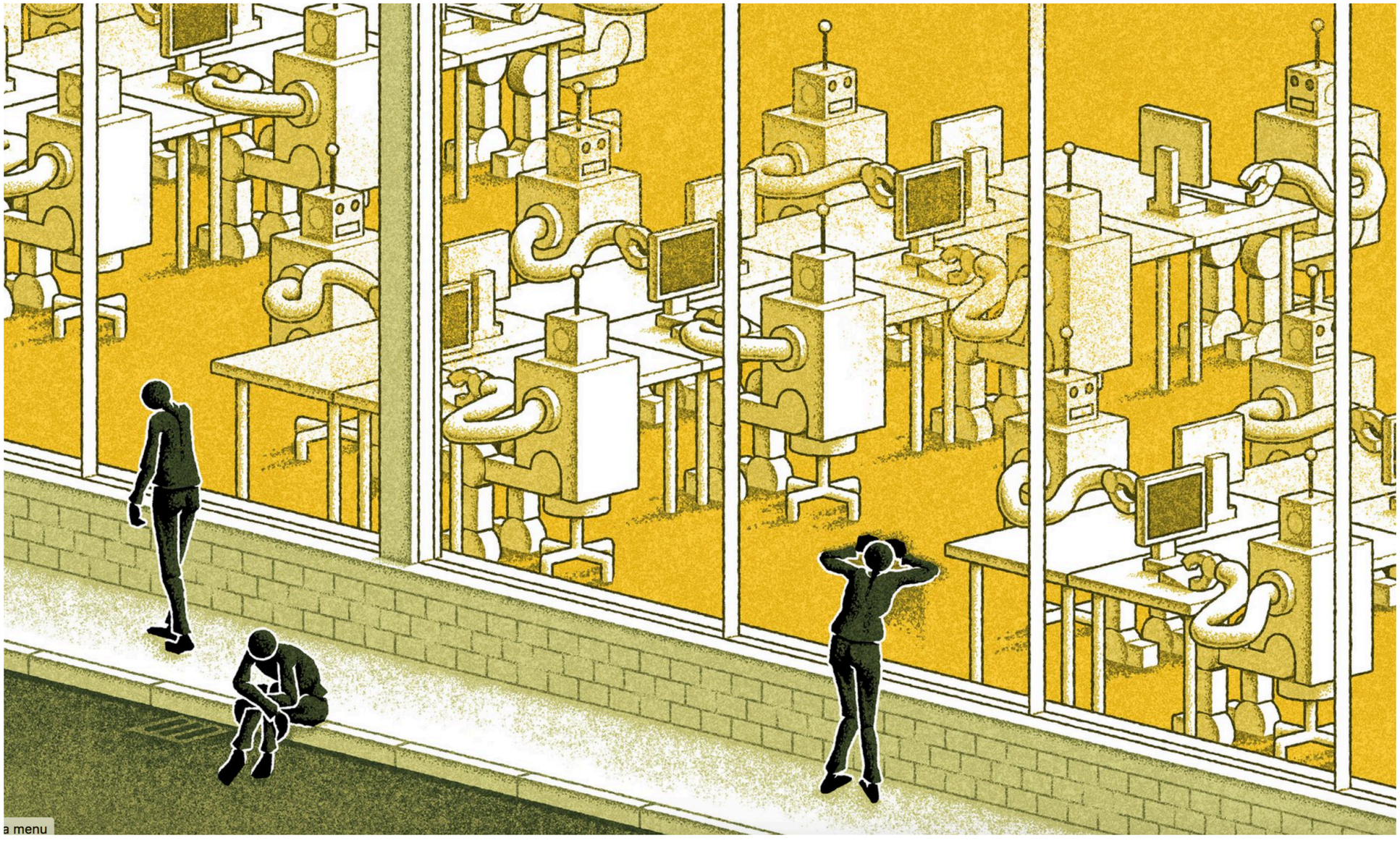
Major areas of Focus

- Semi-Supervised Learning
- Reinforcement Learning & Deep Sensorimotor Learning
- Neural Networks with Memory
- True Language Understanding (Not Just Statistical)
- Deep Learning for Hardware and Systems (Low Power)
- **Theory of Deep Learning**
- ….

Resources

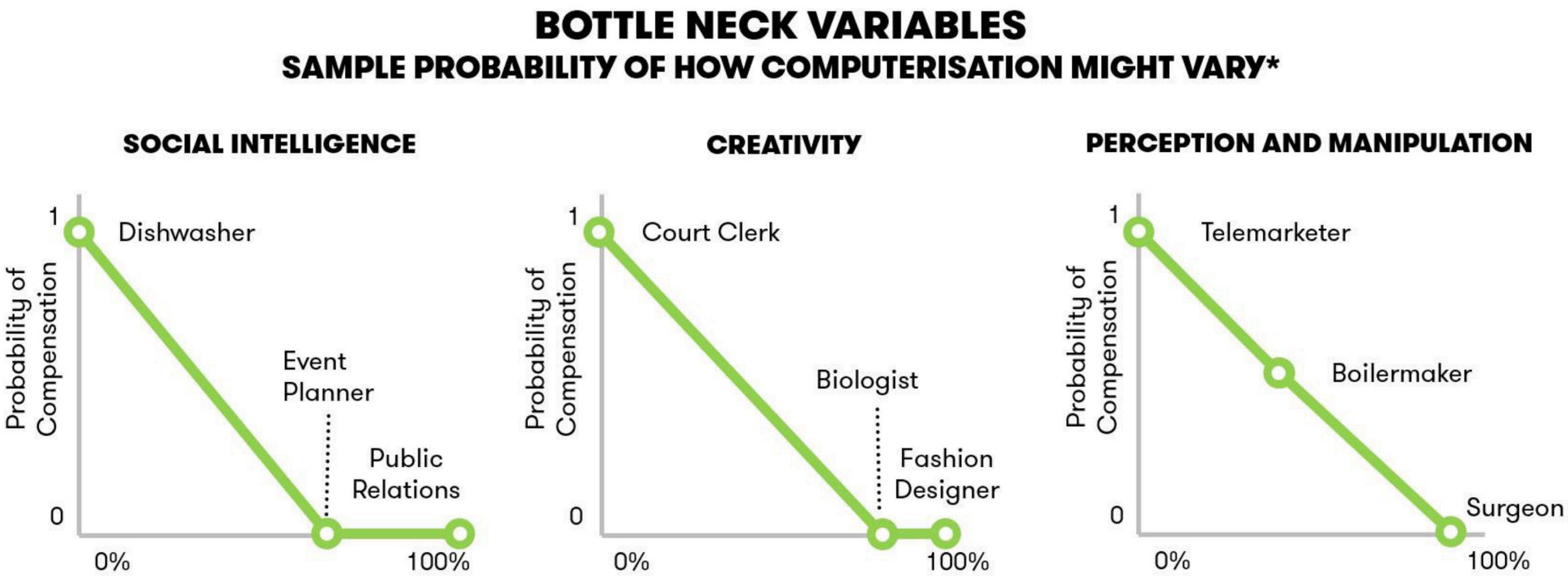
- UFLDL Course
- Chris Olah's Blog
- Google Plus Deep Learning Community
- Deep Learning First Textbook
- List of Must-Read Papers

Future of Work



The Guardian

Future of Work



* Data from the article "Which Careers are most likely to be automated?" By Peter Orr of "80,000 Hours". It gives a detailed prediction of what type of jobs will become automated and sensors that machines will not and do not possess in the near future.

Blake Irving's Blog

Myth:
Superintelligence by 2100 is inevitable

Myth:
Superintelligence by 2100 is impossible

Fact:
It may happen in decades, centuries or never: AI experts disagree & we simply don't know

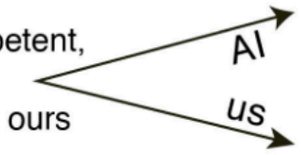
Myth:
Only Luddites worry about AI

Fact:
Many top AI researchers are concerned

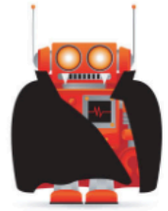
Mythical worry:
AI turning evil

Mythical worry:
AI turning conscious

Actual worry:
AI turning competent, with goals misaligned with ours



Myth:
Robots are the main concern



Fact:
Misaligned intelligence is the main concern: it needs no body, only an internet connection

Myth:
AI can't control humans

Fact:
Intelligence enables control: we control tigers by being smarter

Myth:
Machines can't have goals

Fact:
A heat-seeking missile has a goal

Mythical worry:
Superintelligence is just years away

Actual worry:
It's at least decades away, but it may take that long to make it safe

Future of Life