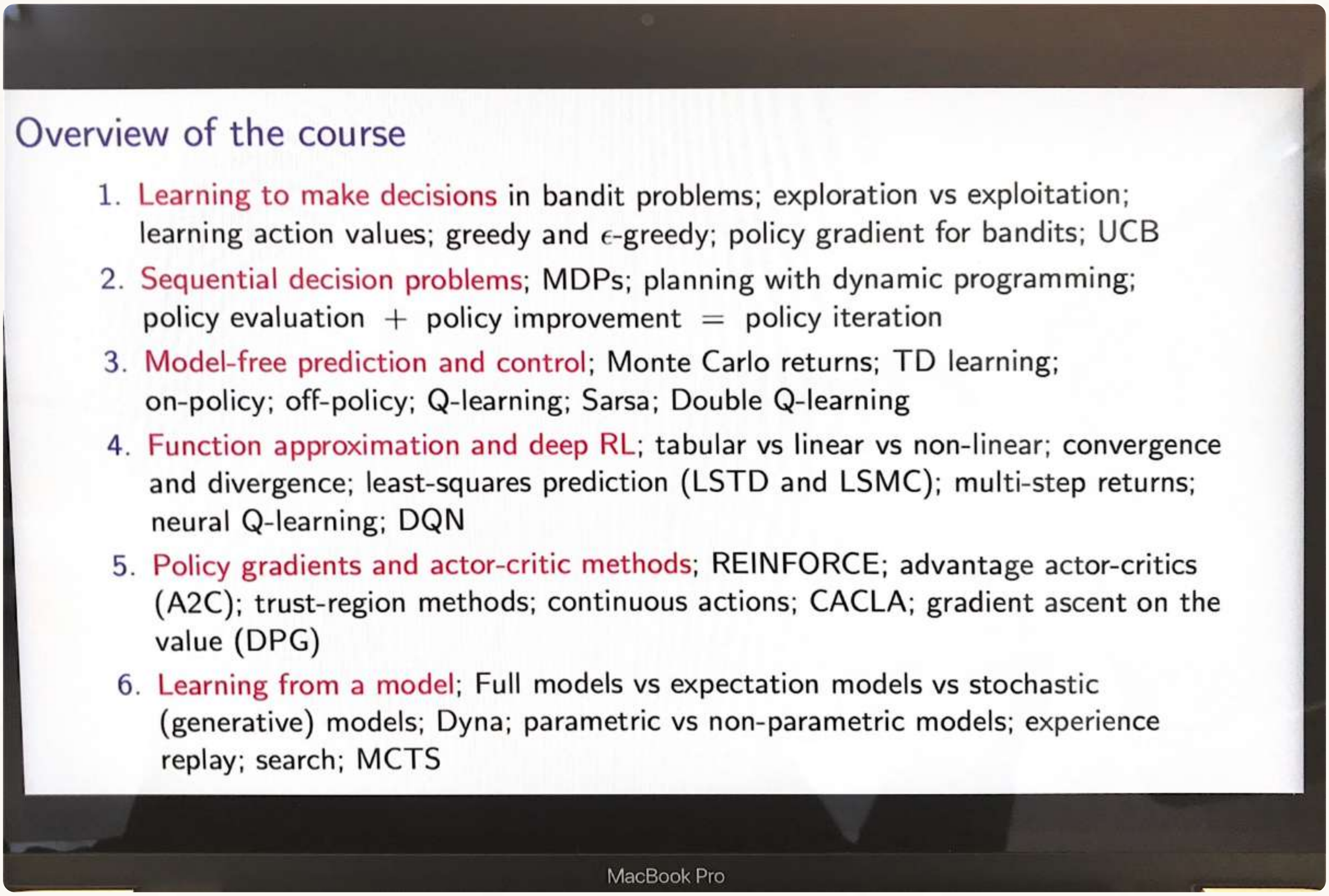
Deep RL

Multi-agent Learning

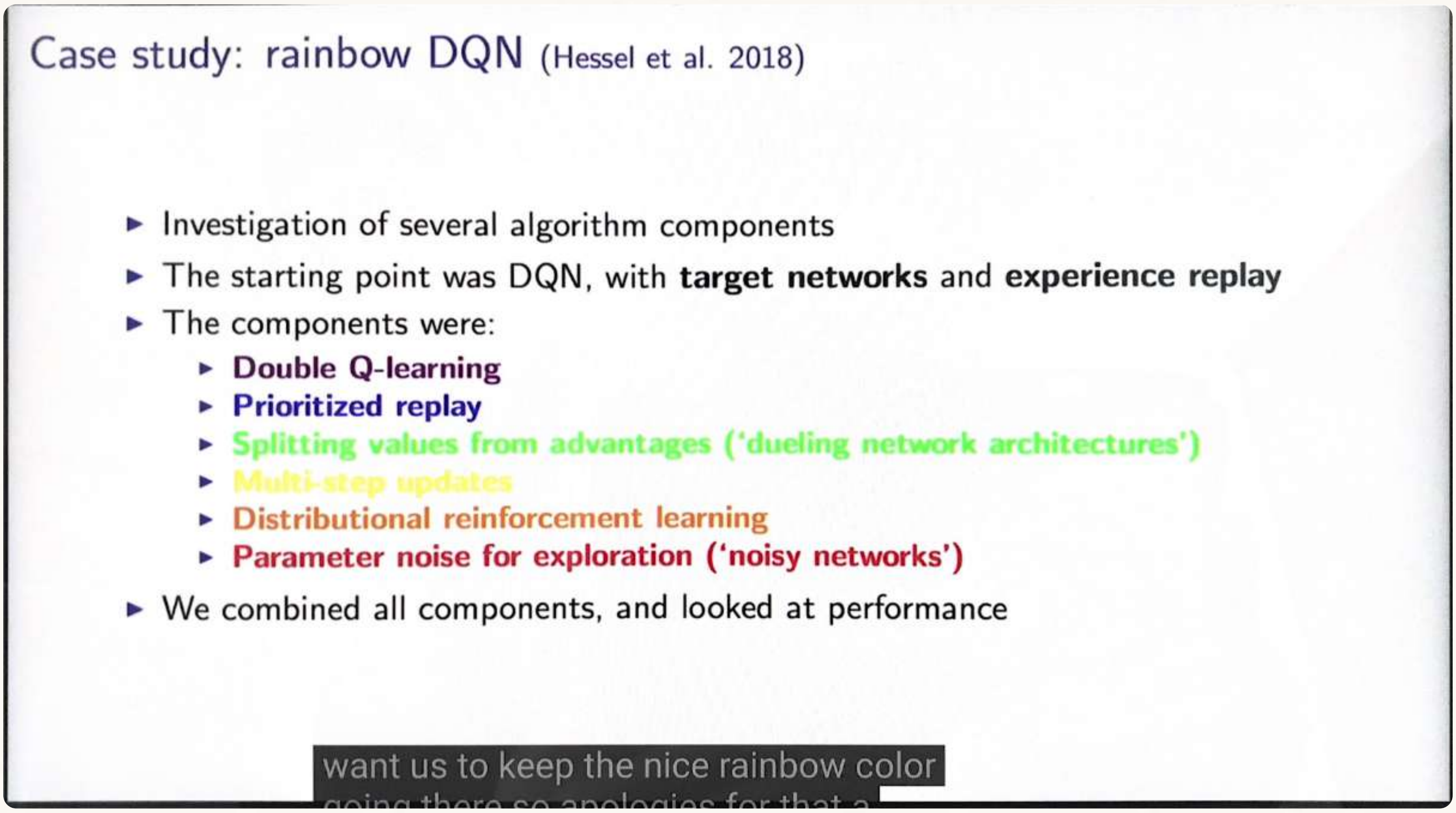


Advanced topics and active research

- The main question is: how do we maximize future rewards
- Some main sub-questions are:
  - What do we learn? (Predictions, models, policies, ...)
  - How do we learn it? (TD, planning, ...)
  - How do we represent the learnt knowledge? (deep networks, sample buffers, ...)
  - How do we use the learnt knowledge?
- Specific active research topics include:
  - Exploration in the full sequential, function approximation case
  - Credit assignment with very delayed rewards
  - Planning with partial or inaccurate models
  - Sample efficient learning
  - Appropriate generalization (e.g., fast learning in new situations)
  - Building a useful, general, and information-rich agent state

to answer
and for instance we could

Maximize future rewards!



DQN with
- target Networks
- experience replay

Double Q-learning

Domain: Arcade Learning Environment

(Bellemare 2013)

Atari games from the ALE as benchmark

"general learning algorithm"

Start Pointing: DQN

1. CNN $q_\theta : O_t \to \mathbb{R}^m$ for m actions

2. ε-greedy policy: π_t

3. reply buffer for experience reply

4. target network parameters θ^- ($\theta_0^- = \theta_0$)

5. Q-learning loss function on θ (uses replay and target network)

$$\boxed{I(\theta) = \frac{1}{2}\left(R_{i+1} + \gamma\left[\max_a q_{\theta^-}(S_{i+1}, a)\right] - q_\theta(S_i, A_i)\right)^2}$$

6. Optimizer (SGD / RMSprop / Adam)

7. Update $\theta_t^- \leftarrow \theta_t$ occasionally (e.g. every 10000 steps. $\theta_t^- = \theta_{t-1}^-$)

Double Q-learning

Loss function:

$$I(\theta) = \frac{1}{2}\Big(R_{i+1} + \gamma\big[q_{\theta^-}\big(S_{i+1}, \arg\max_a q_\theta(S_{i+1}, a)\big)\big] - q_\theta(S_i, A_i)\Big)^2$$

Prioritized Replay

DQN samples uniformly from replay

Idea: prioritize transitions on which we can learn much

$$\text{priority of sample } i = |\delta_i|$$

δ_i : TD error on the last this transition was sampled

Dueling Networks (Wang 2016)

$$q_\theta(s,a) = \underbrace{V_\xi(s)}_{} + \underbrace{A_\chi(s,a)}_{\text{advantage for taking action } a}$$

$\theta = \xi \cup \chi$

advantage for taking action a

Multi-step updates (Sutton 1988)

TD target look n steps in the future

"n-step" return:

$n=1$ (TD) $G_t^{(1)} = R_{t+1} + \gamma v(S_{t+1})$

$n=2$ $G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 v(S_{t+2})$

$\vdots$

$n=\infty$ (MC) $G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-t-1} R_T$

n-step return:

$$\boxed{G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n v(S_{t+n})}$$

n-step temporal-difference learning:

$$v(S_t) \leftarrow v(S_t) + \alpha\left(G_t^{(n)} - v(S_t)\right)$$

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n q_{\theta^-}\left(S_{t+1}, \arg\max_a q_\theta(S_{t+1}, a)\right)$$

Double Q bootstrap target

Multi-step Q-learning:

$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha\left(G_t^{(n)} - q(S_t, A_t)\right)$$

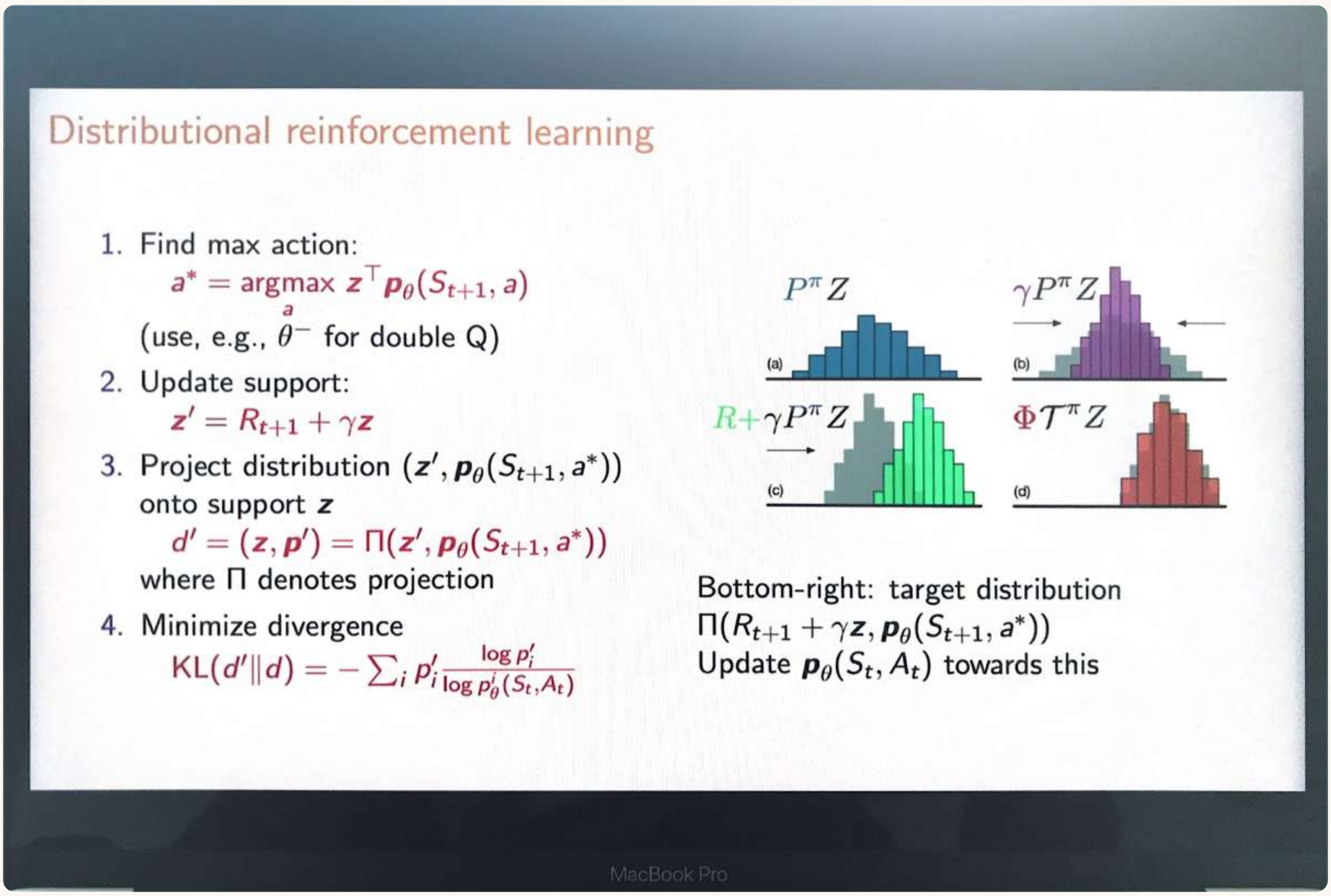
Distributional RL

expected cumulative rewards

↓

distribution of returns

Categorical DQN (Bellemare 2017)



Noisy Networks (Fortunato 2017)

DQN → ε-greedy exploration

Idea: Add noise to parameters

replace all linear operations

$$y = Wx + b$$

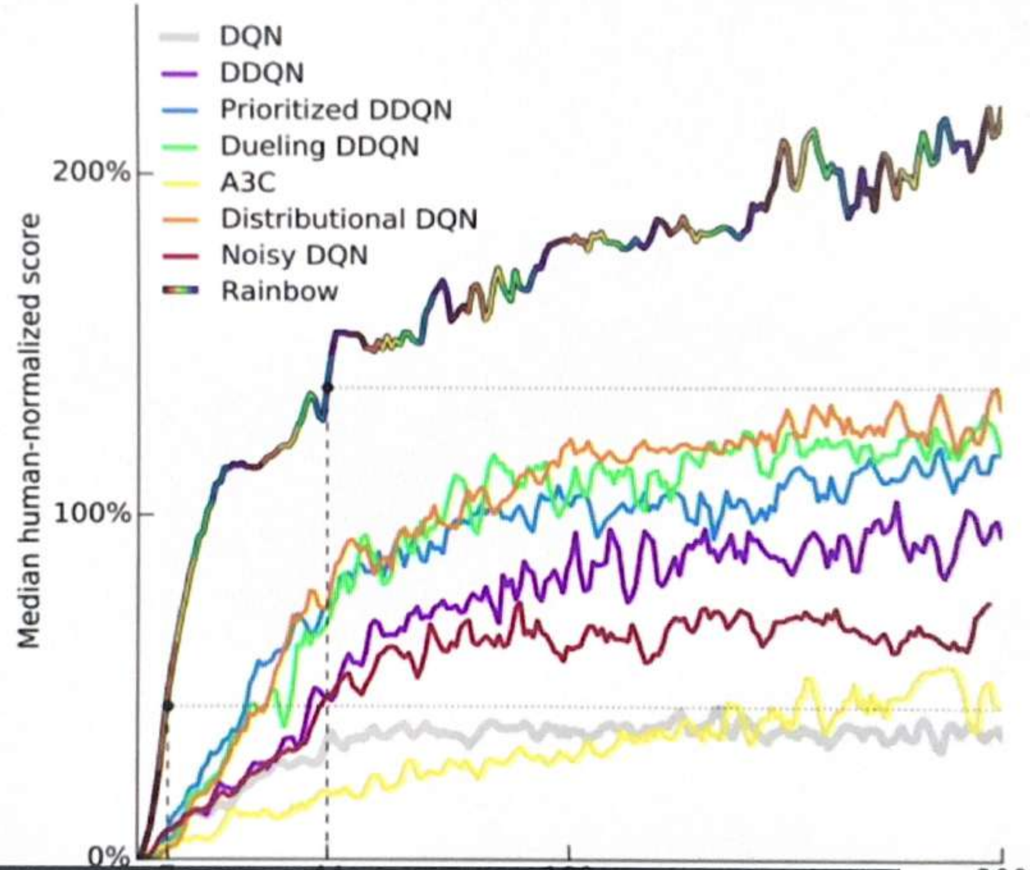
with

$$\boxed{y = Wx + b + (W' \times \varepsilon^{w}) + b' \times \varepsilon^{b}}$$

"$\times$" : element-wise product

ε^{w}, ε^{b} : random matrix / vector

Rainbow DQN: results



you've seen and y-axis is a median human
normalized score which means

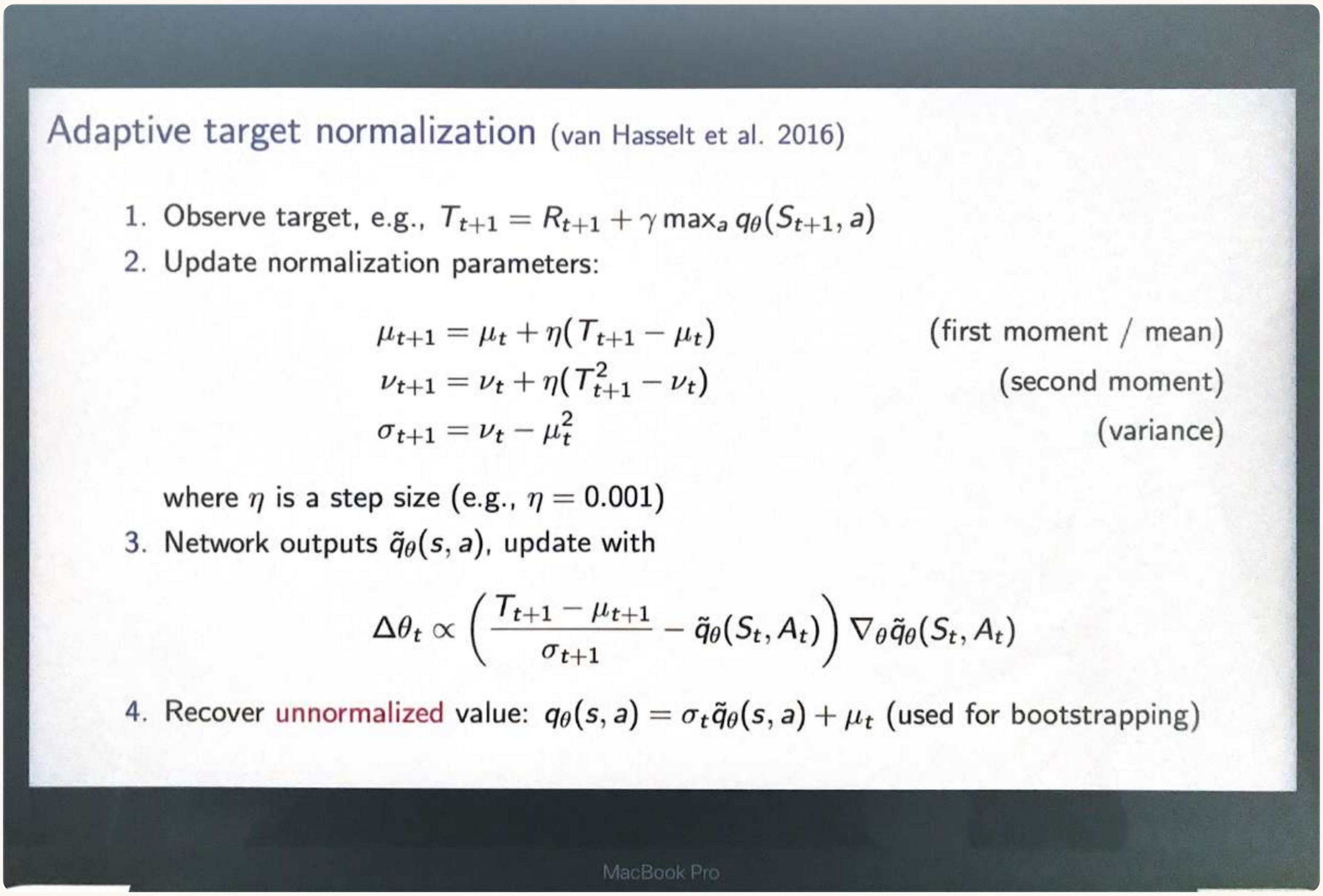
MacBook Pro

Rainbow DQN: conclusions

- Components work well together
- Most important: prioritising replay, multi-step returns
- Least important: double, dueling
- No wild overestimations due to fixed bounded support of value distribution
- But this requires knowing appropriate range...
- ...but different game have different score ranges
- This is possible due to reward clipping: in DQN rewards are clipped to $[-1, 1]$
- Makes learning easier, but changes the objective...

Adaptive normalization of updates

Normalize targets $\longrightarrow$ update



Preserve outputs

- Naive implementation changes all outputs whenever we update the normalization
- This seems bad: we should avoid updating values of unrelated states
- We can avoid this. Typically:

$$\tilde{\boldsymbol{q}}_{\boldsymbol{W},\boldsymbol{b},\theta}(s) = \boldsymbol{W}\phi_\theta(s) + \boldsymbol{b}\,.$$

- Idea: define

$$\boldsymbol{W}'_t = \frac{\sigma_t}{\sigma_{t+1}}\boldsymbol{W} \qquad \boldsymbol{b}'_t = \frac{\sigma_t \boldsymbol{b}_t + \mu_t - \mu_{t+1}}{\sigma_{t+1}}$$

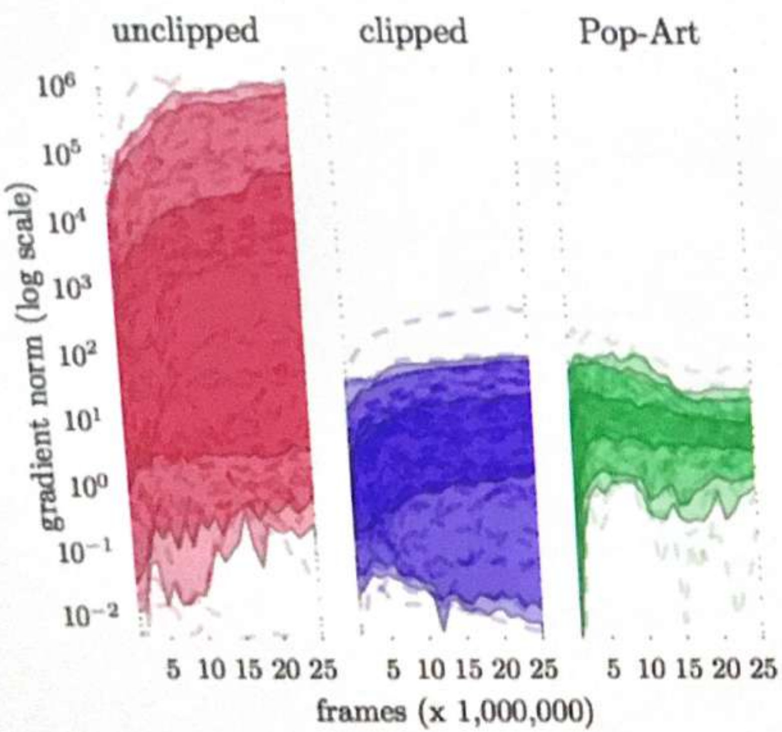
  Then

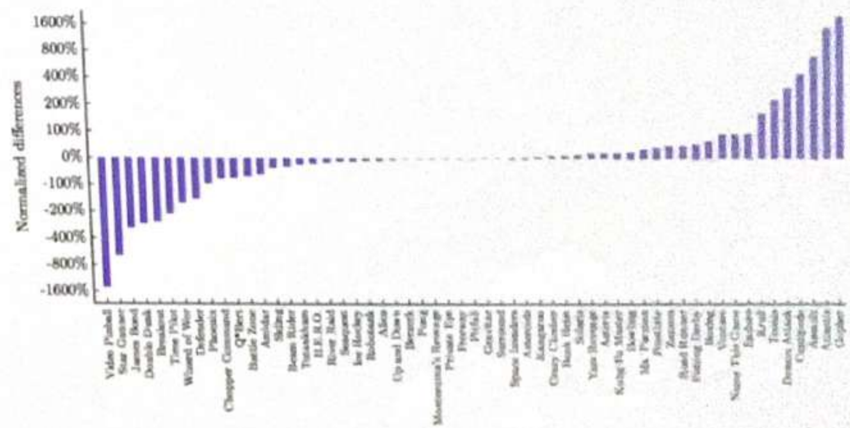
$$\sigma_{t+1}\tilde{\boldsymbol{q}}_{\boldsymbol{W}'_t,\boldsymbol{b}'_t,\theta_t}(s) + \mu_{t+1} \quad = \quad \sigma_t\tilde{\boldsymbol{q}}_{\boldsymbol{W}_t,\boldsymbol{b}_t,\theta_t}(s) + \mu_t$$

- Then update $\boldsymbol{W}'_t$, $\boldsymbol{b}'_t$ and θ_t as normal (e.g., SGD)

Preserve outputs

- **P**reserve **O**utputs **P**recisely, while **A**daptively **R**escaling **T**argets: Pop-Art





Universal value function approximation

Idea: feed a representation of (c, γ) is as input

Allows generalization across goals / tasks within an environment.

"Unicorn" (Mankowitz 2018)

GVF and models