RL: studying sequential decision-making

AI: → sequential decision making
→ learning
→ deep computational graph

RL: study sequential decision-making
(general framework)

The Deep RL Boom

TD-Gammon (Tesauro, 1989–1995)

↓

Slot car driving (Lange & Riedmiller, 2012)

↓

Arcade Learning Environment (Bellemare 2013)

↓

Deep Q-Network (2013, 2015)

↓

Trust region policy optimization (Schulman, 2015)

↓

End-to-end training (Levine 2015)

On real robots

↓

AlphaGo (2015)

RL + DNN ?

① Sparse / delayed feedback

② data distribution is non-stationary

- determined by the agent's actions

  - Most of the DNN theory no longer apply
  - Exploration vs exploitation
  - local minima.

③ Training DNN with RL was thought to be inherently unstable.

(Tsitsikis & Van Roy 1997)

DQN (Deep Q Networks)

$$\boxed{\text{CNN} \xrightarrow{\text{represent}} \text{Action-value function}(Q)}$$

(S_t, a_t, r_t, S_{t+1})
experience

ε-greedy

Tabular Q-learning:

① Start with a guess for each $Q(s,a)$
② interact with environment using policy based on Q
③ updates (based on Bellman Equations)

$$Q(S_t, a_t) \leftarrow Q(S_t, a_t) + \alpha_t \Big(\underbrace{r_t + \gamma \max_a Q(S_{t+1}, a)}_{\text{target}} - Q(S_t, a_t) \Big)$$

Problems: (DNN/CNN $\rightarrow$ Q)

① Correlation between successive updates
② Correlation between $Q(S_t, a_t)$ and the target

High-level idea:

$$\boxed{\text{Q-Learning (RL)} \xrightarrow[\text{like}]{\text{look}} \text{supervised learning}}$$

Apply Q-update on batches of past experience

instead of online

① Experience replay (Lin, 1993)

② Previously used for better data efficiency

③ Makes the data distribution more stationary

Use an old set of weights to compute the target (target network)

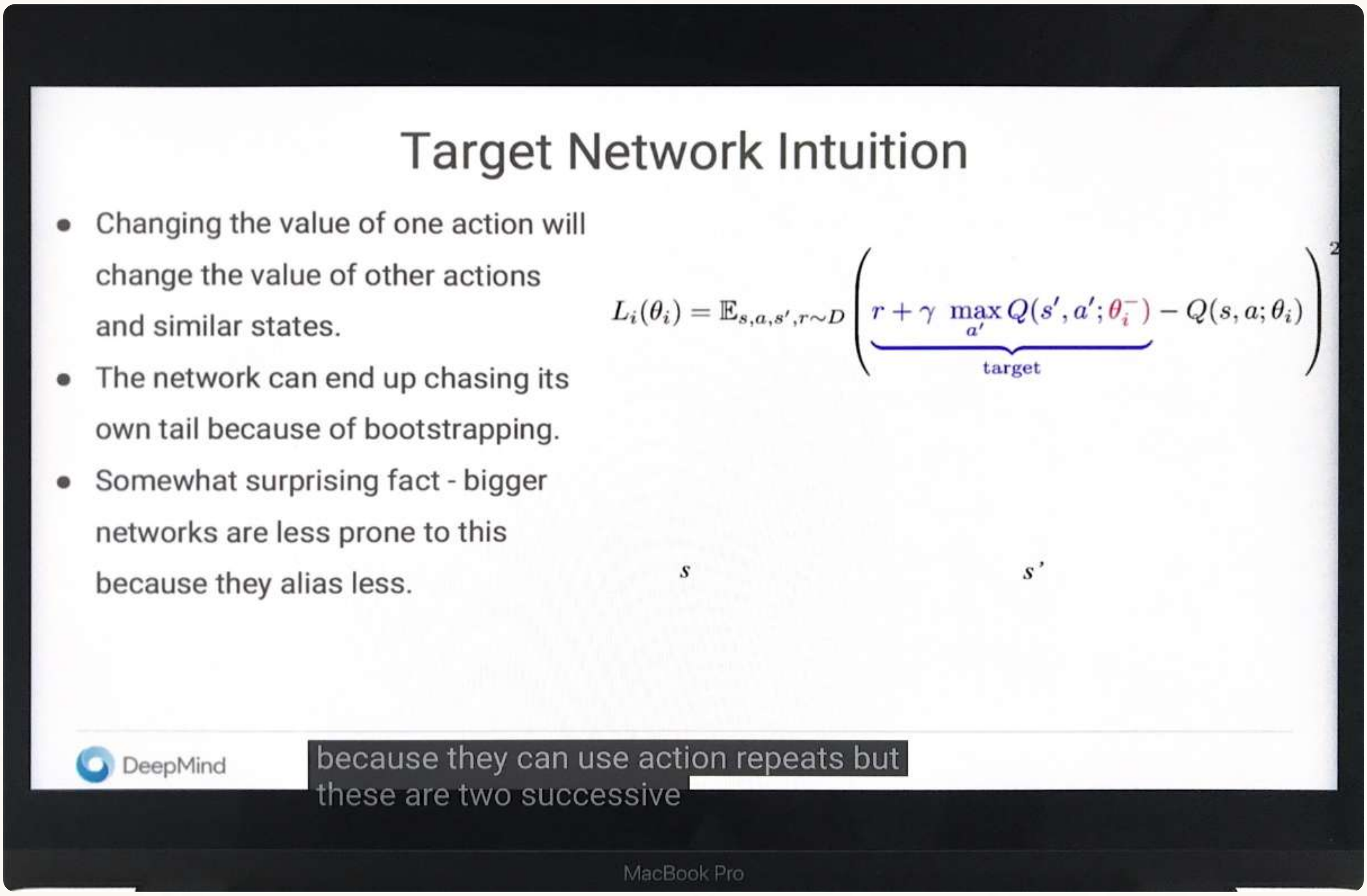
Keeps the target function from changing too quickly

target

$$\boxed{L_i(\theta_i) = E_{s,a,s',r\sim D}\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)^2}$$

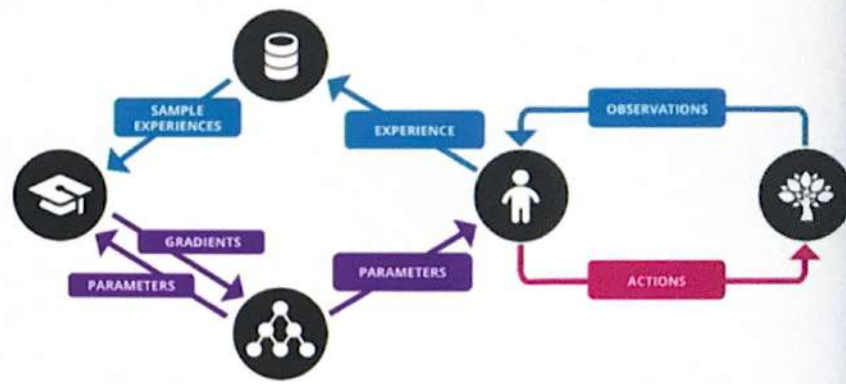
Neural Fitted Q Iteration

NFQ (Riedmiller, 2005)

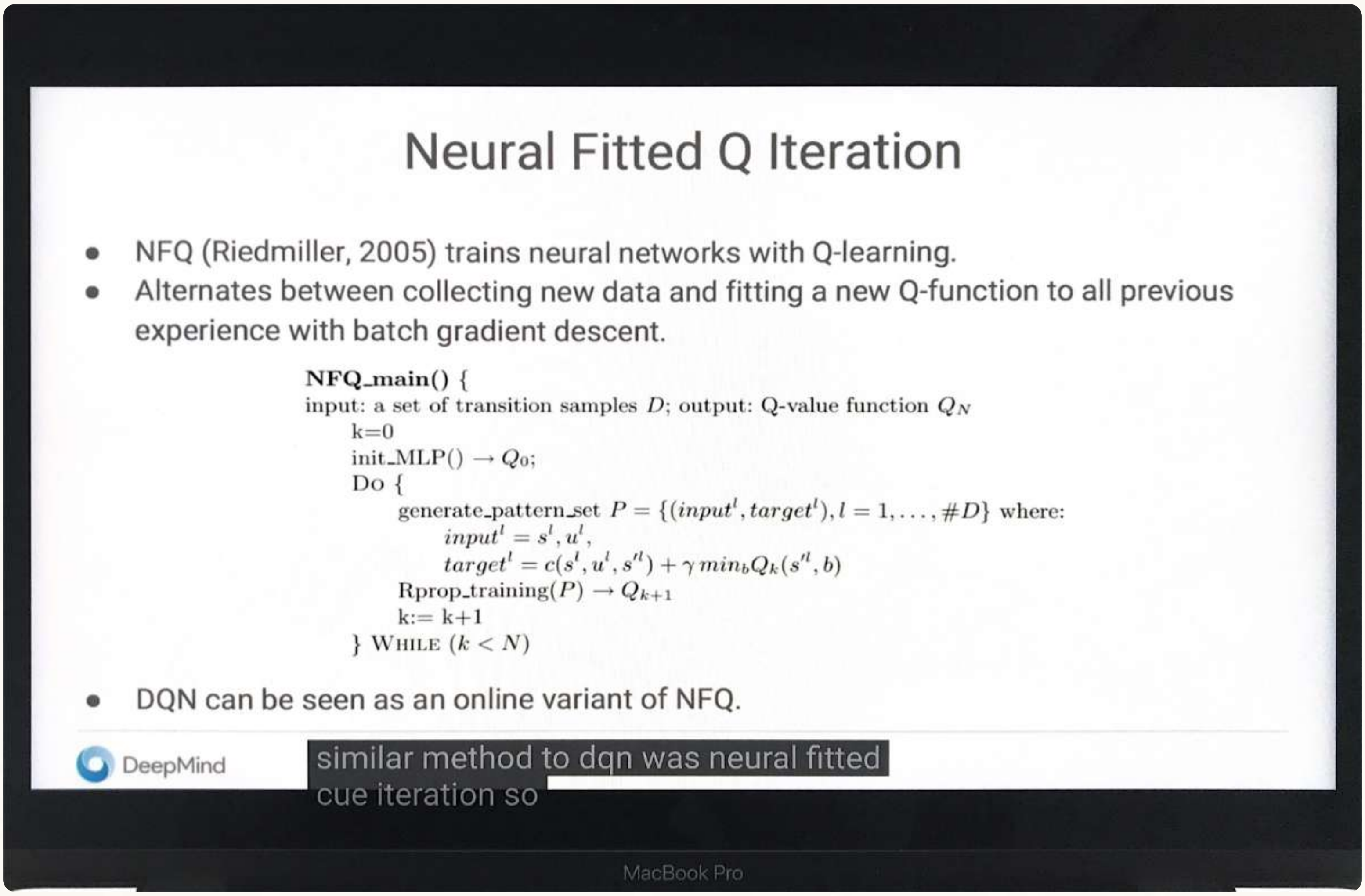


DQN Training Algorithm

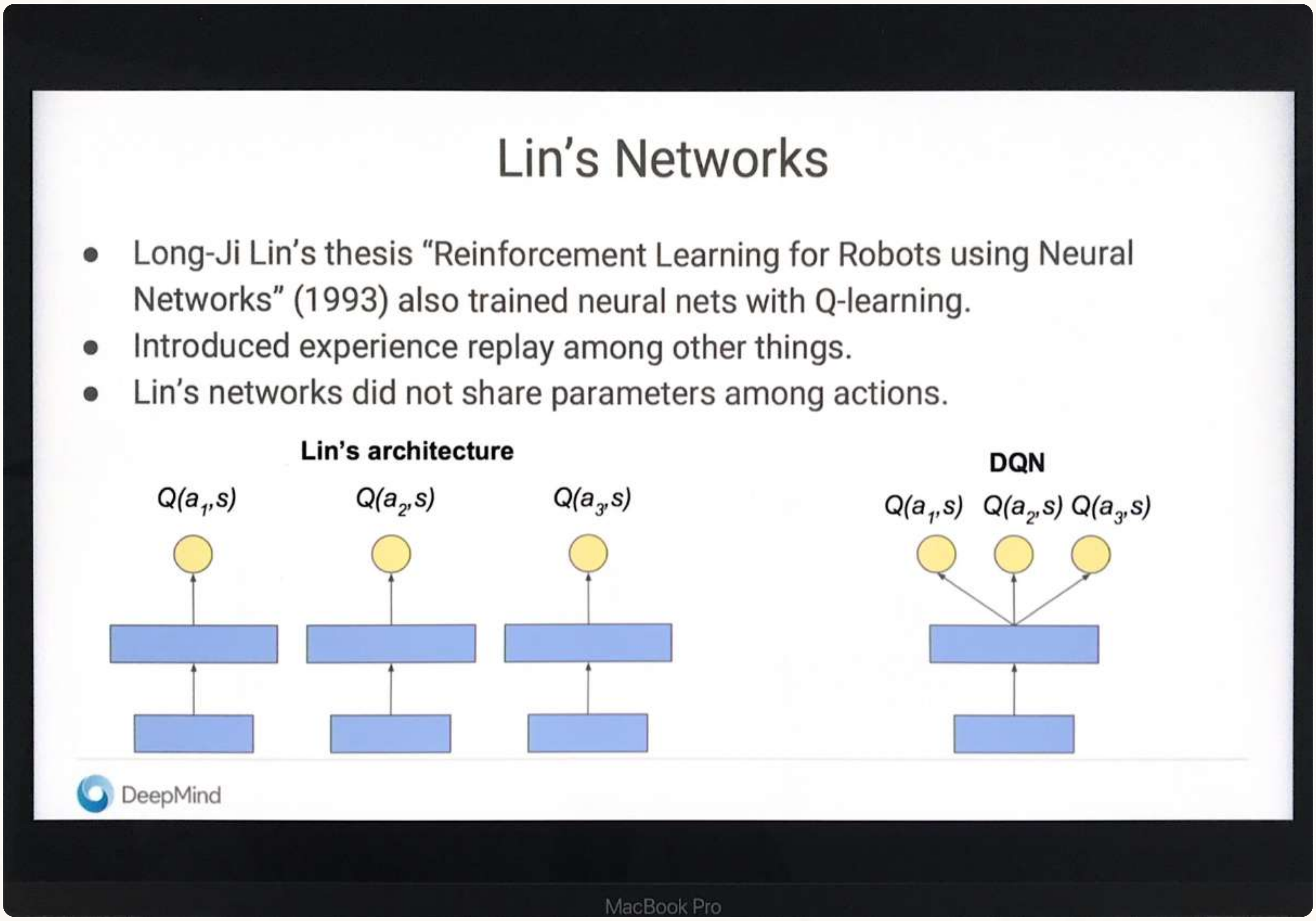
Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
    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 $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For



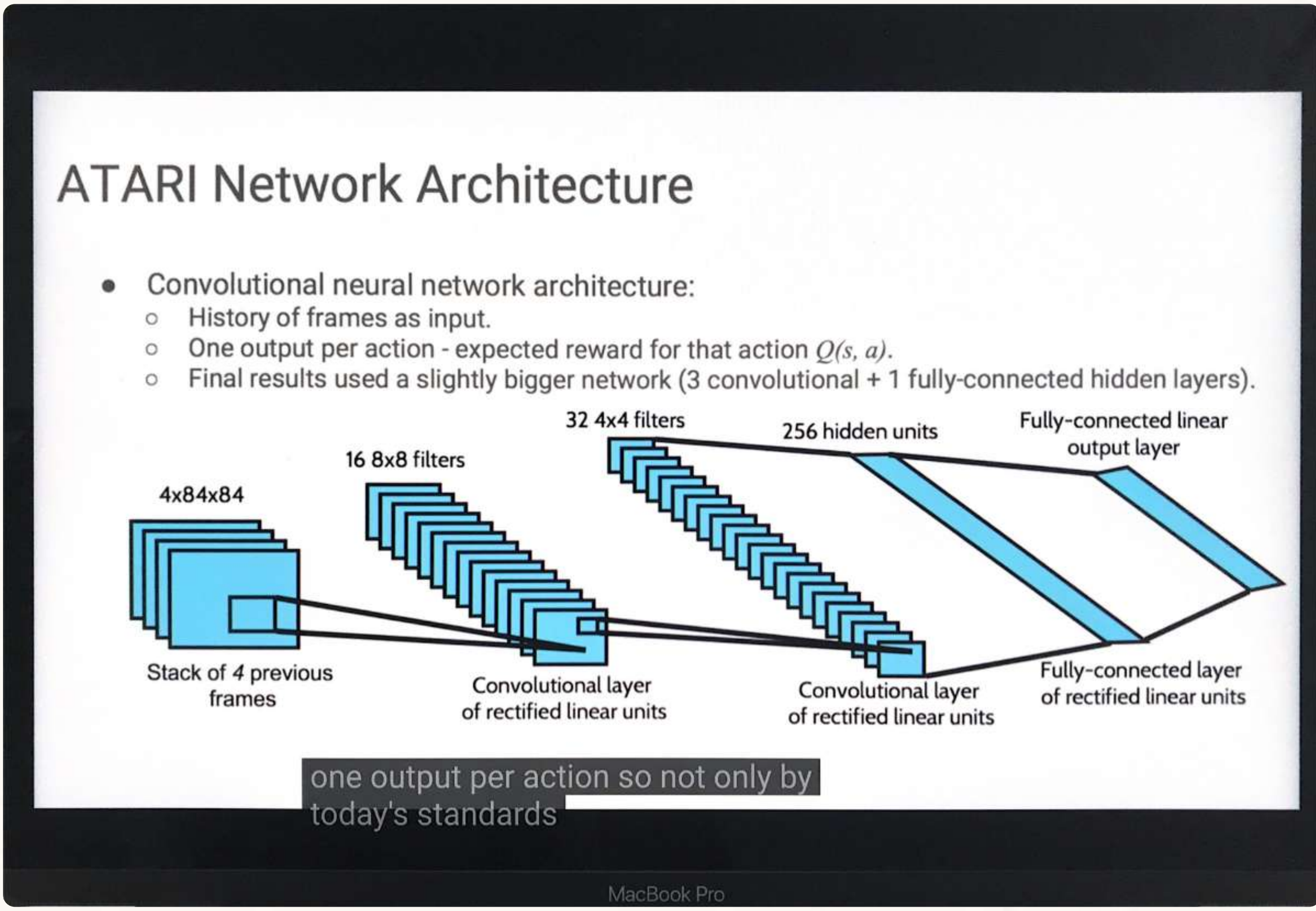
Neural Fitted Q iteration



Lin's Networks



ATARI Network Architecture



DQN / Mini-batch Q-learning

Pros

- Robust
- GPU friendly

Cons :

- Slow
- less RNN friendly (less successful in 3D environment)

Beyond DQN

① fast training

② on or off-policy methods

③ flexibility
- discrete or continuous actions
- feedforward
- recurrent models

④ Asynchronous Methods for DRL

(Mnih 2016) AsyncRL

AsyncRL

Parallel actor-learners (CPU threads)

online asynchronous updates

(Recht 2011, Lian 2015)

RL algorithm: on-policy / off-policy

value-based / policy-based

1-step Q-learning

Parallel actor-learners compute online 1-step update.

$$\boxed{\begin{aligned} y &\leftarrow r + \gamma \max_{a'} Q(s', a'; \theta^-) \\ \Delta\theta &\leftarrow \Delta\theta + \frac{\partial (y - Q(s, a; \theta))^2}{\partial \theta} \end{aligned}}$$

N-step Q-learning

Q-learning with a uniform mixture of backups of length 1 through N.

$r_t, r_{t+1}, r_{t+2}, \ldots, r_{t+n}, \quad \max X_a Q(a, S_{t+N+1})$

$$\boxed{\begin{aligned} y &\leftarrow \sum_{k=0}^{N-1} \gamma^k r_{t+k} + \gamma^N \max_{a'} Q(S_{t+N}, a'; \theta^-) \\ \Delta\theta &\leftarrow \Delta\theta + \frac{\partial (y - Q(S_t, a_t; \theta))^2}{\partial \theta} \end{aligned}}$$

Variation of "Incremental multi-step Q-learning" (Peng & Williams 1995)

Asynchronous Advantage Actor-critic (A3C)

① Agent learns
- a policy
- a state value function

② bootstrapped n-step returns

→ reduce variance over REINFORCE with a baseline

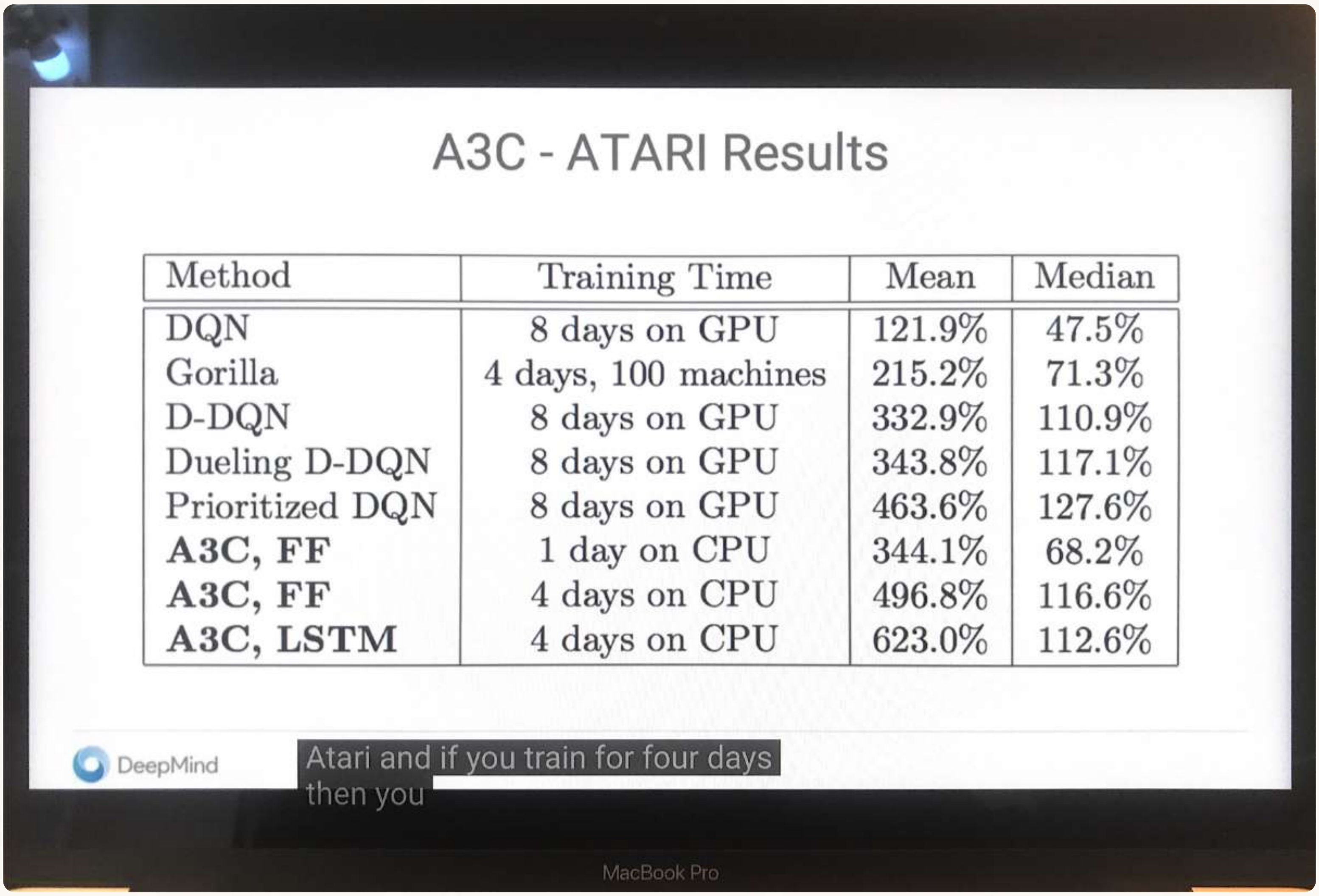
③ Policy gradient multiplied by an estimate of the advantage

$$\nabla_\theta \log \pi(a_t \mid s_t, \theta) \left(\sum_{k=0}^{N} \gamma^k r_{t+k} + \gamma^{N+1} V(S_{t+N+1}) - V(S_t) \right)$$

④ Critic/value function is trained with n-step TD learning (by minimizing the MSE)

$$\left(\sum_{k=0}^{N} \gamma^k r_{t+k} + \gamma^{N+1} V(S_{t+N+1}; \theta^-) - V(S_t; \theta) \right)^2$$

"A3C tends to dominate the value-based methods"



A3C - ATARI Results

Method	Training Time	Mean	Median
DQN	8 days on GPU	121.9%	47.5%
Gorilla	4 days, 100 machines	215.2%	71.3%
D-DQN	8 days on GPU	332.9%	110.9%
Dueling D-DQN	8 days on GPU	343.8%	117.1%
Prioritized DQN	8 days on GPU	463.6%	127.6%
A3C, FF	1 day on CPU	344.1%	68.2%
A3C, FF	4 days on CPU	496.8%	116.6%
A3C, LSTM	4 days on CPU	623.0%	112.6%

Pros of N-step methods:

① faster reward propagation

② No need for target network

③ easier training of RNN

A3C — Pros (fast , RNN friendly)
(stable , scalable)
Cons (Not GPU friendly)

IMPALA : distributed deep RL , (2018)

Scale up A3C

"V-Trace" algorithm

"DMLab-30 Task Set"
(Multi-)

DQN is more stable than A3C

off-policy harms actor-critic method

Deep RL

① Minibatch training on GPUs

② Deep ResNet and LSTMs

③ Adam / RMSProp optimizers

Deep RL

Practical Advice - Getting Started

- Start with a simple problem.
    - Something solvable in under a minute on your local machine.
    - Make it similar to the problem you really want to solve.
    - Ideally it should have knobs for controlling its difficulty.
- Plot the training curves (averaged over multiple episodes).
- Visualize the policy.
- Visualize the value function.
- Visualize everything you can think of.

has some of the properties of the real
problem you want to

Practical Advice - Neural Nets

- Doing early experiments with a small network can help iterate faster.
    - This can also backfire (DQN and target networks).
- Reasonable strategy:
    - Run a few progressively larger nets to find what's sufficient for experimenting.
    - Periodically try larger nets to max out performance and verify assumptions.
- Be careful with initialization:
    - Visualize the initial policy to make sure it gets some rewards.
- Try RMSProp and/or Adam.
- Test deep learning tricks before incorporating them: dropout, batch norm, etc.
- See John Schulman's excellent guide - http://joschu.net/docs/nuts-and-bolts.pdf