Deep RL Foundations in 6 Lectures

Lecture 2: Deep Q-Learning

Pieter Abbeel

Lecture Series

- Lecture 1: MDPs Foundations and Exact Solution Methods
- ***Lecture 2: Deep Q-Learning***
- Lecture 3: Policy Gradients, Advantage Estimation
- Lecture 4: TRPO, PPO
- Lecture 5: DDPG, SAC
- Lecture 6: Model-based RL

Quick One-Slide Recap

- Optimal Planning / Control

  =

  given an MDP (S, A, P, R, γ, H)

  find the optimal policy π^*

- Exact Methods:
  - ***Value Iteration***
  - ***Policy Iteration***

Limitations:

- Requires access to dynamics model
- Requires iteration over all states and actions, which is impractical for large MDPs

-> sampling-based approximations

-> Q/V function fitting

Outline

- Q-Learning
- Deep Learning / Neural Networks
- Deep Q Networks (DQN)

Recap Q-Values

$Q^*(s, a)$ = expected utility starting in s, taking action a, and (thereafter) acting optimally

Bellman Equation:

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*(s',a'))$$

Q-Value Iteration:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q_k(s',a'))$$

(Tabular) Q-Learning

- Q-value iteration: $Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q_k(s',a'))$
- Rewrite as expectation: $Q_{k+1} \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$
- (Tabular) Q-Learning: replace expectation by samples
  - For an state-action pair (s,a), receive: $s' \sim P(s'|s,a)$
  - Consider your old estimate: $Q_k(s,a)$
  - Consider your new sample estimate:
    $$\text{target}(s') = R(s,a,s') + \gamma \max_{a'} Q_k(s',a')$$
  - Incorporate the new estimate into a running average:
    $$Q_{k+1}(s,a) \leftarrow (1-\alpha) Q_k(s,a) + \alpha \left[\text{target}(s') \right]$$

(Tabular) Q-Learning

Algorithm:

Start with $Q_0(s, a)$ for all s, a.

Get initial state s

For k = 1, 2, … till convergence

Sample action a, get next state s'

If s' is terminal:

$$\text{target} = R(s, a, s')$$

Sample new initial state s'

else:

$$\text{target} = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$$

$$Q_{k+1}(s, a) \leftarrow (1 - \alpha) Q_k(s, a) + \alpha \, [\text{target}]$$

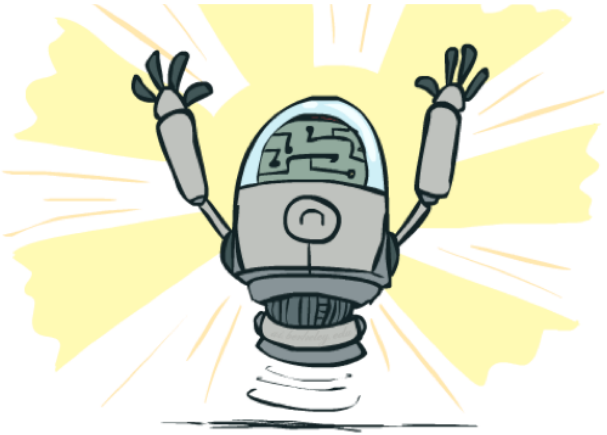
$$s \leftarrow s'$$

How to sample actions?

- Choose random actions?
- Choose action that maximizes $Q_k(s, a)$ (i.e. greedily)?
- ε-Greedy: choose random action with prob. ε, otherwise choose action greedily

Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!
- This is called off-policy learning
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - … but not decrease it too quickly



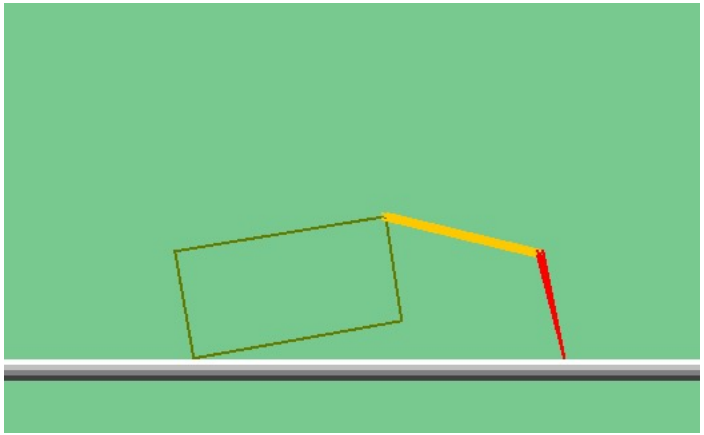
Q-Learning Properties

- Technical requirements.
  - All states and actions are visited infinitely often
    - Basically, in the limit, it doesn't matter how you select actions (!)
  - Learning rate schedule such that for all state and action pairs (s,a):

$$\sum_{t=0}^{\infty} \alpha_t(s,a) = \infty \qquad \sum_{t=0}^{\infty} \alpha_t^2(s,a) < \infty$$

For details, see Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. Neural Computation, 6(6), November 1994.

Q-Learning Demo: Crawler



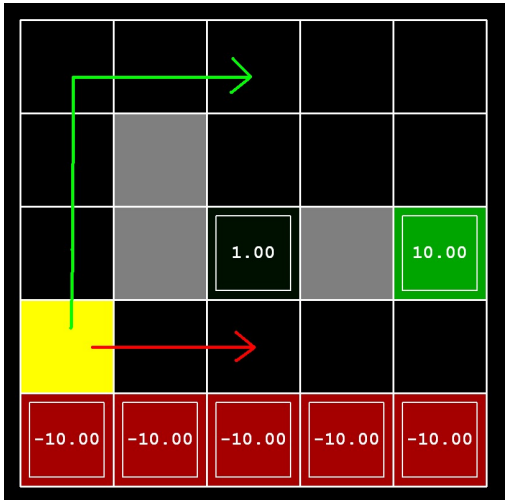
- **States: discretized value of 2d state: (arm angle, hand angle)**
- **Actions: Cartesian product of {arm up, arm down} and {hand up, hand down}**
- **Reward: speed in the forward direction**

Video of Demo Crawler Bot

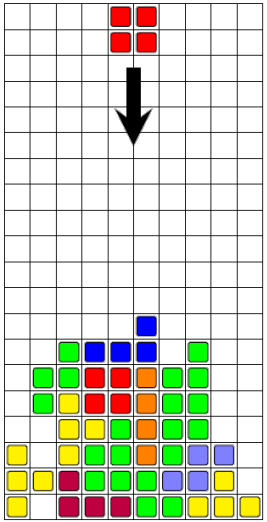
Video of Demo Q-Learning -- Crawler

Can Tabular Methods Scale?

- Discrete environments



Gridworld
10^1

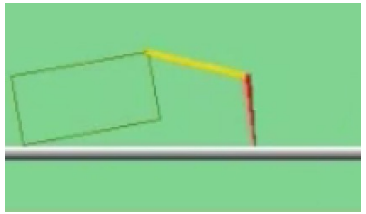


Tetris
10^60

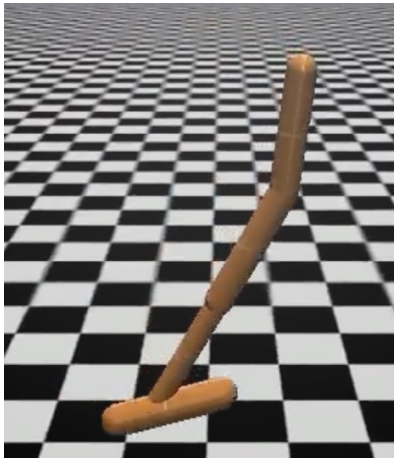
Atari
10^308 (ram) 10^16992 (pixels)

Can Tabular Methods Scale?

- Continuous environments (by crude discretization)



Crawler
10^2

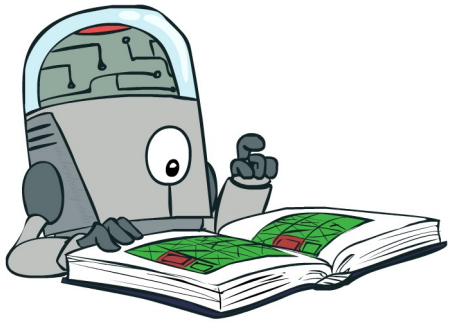


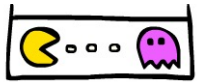
Hopper
10^10

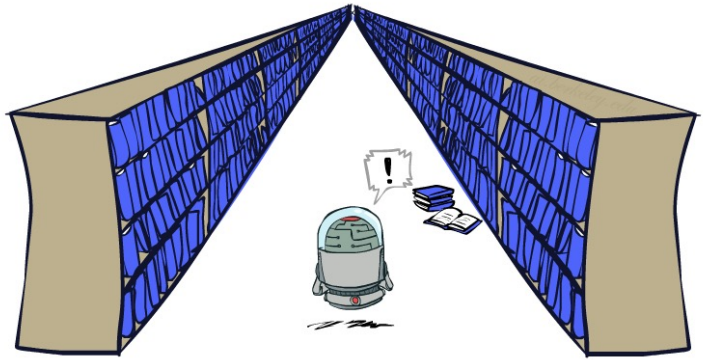
Humanoid
10^100

Generalizing Across States

- Basic Q-Learning keeps a table of all q-values
- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations
  - This is a fundamental idea in machine learning







Approximate Q-Learning

- Instead of a table, we have a ***parametrized Q function***: $Q_\theta(s, a)$
  - Can be a linear function in features:

$$Q_\theta(s, a) = \theta_0 f_0(s, a) + \theta_1 f_1(s, a) + \cdots + \theta_n f_n(s, a)$$

  - Or a neural net, decision tree, etc.

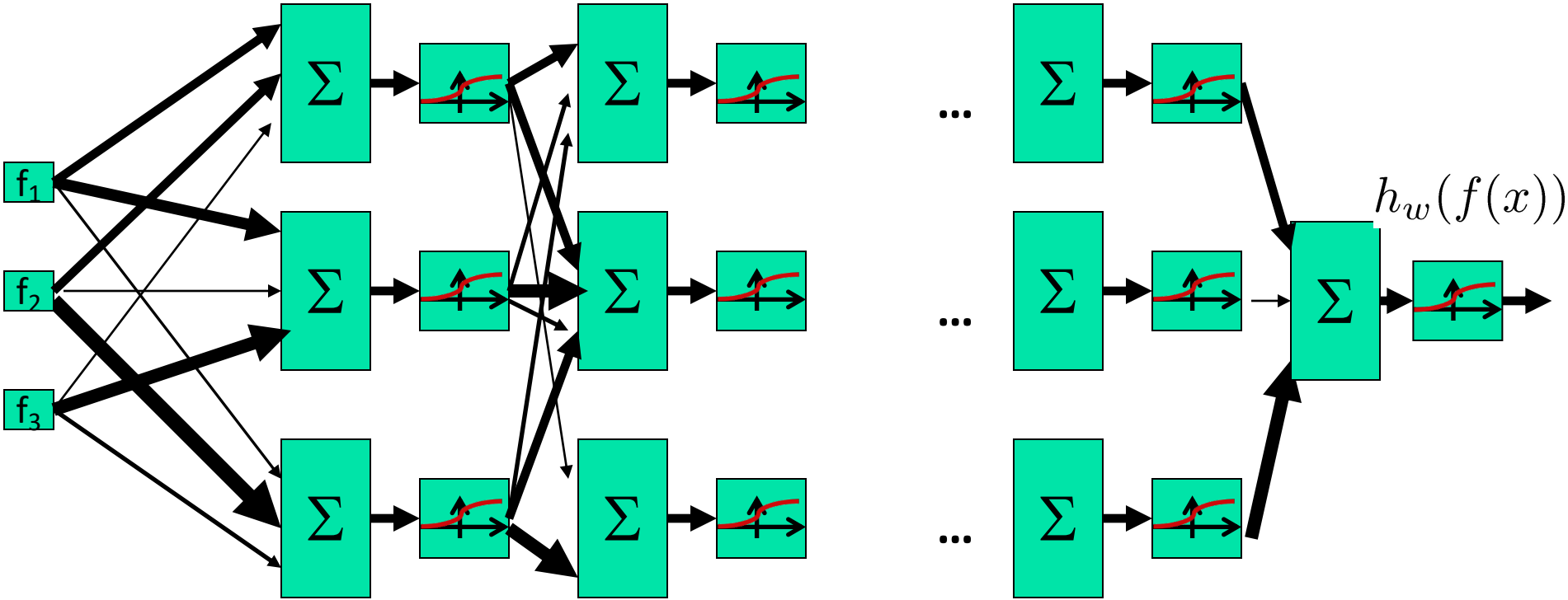
- ***Learning rule***:
  - Remember: $\text{target}(s') = R(s, a, s') + \gamma \max_{a'} Q_{\theta_k}(s', a')$
  - Update:

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \left[\frac{1}{2} (Q_\theta(s, a) - \text{target}(s'))^2 \right] \Bigg|_{\theta=\theta_k}$$

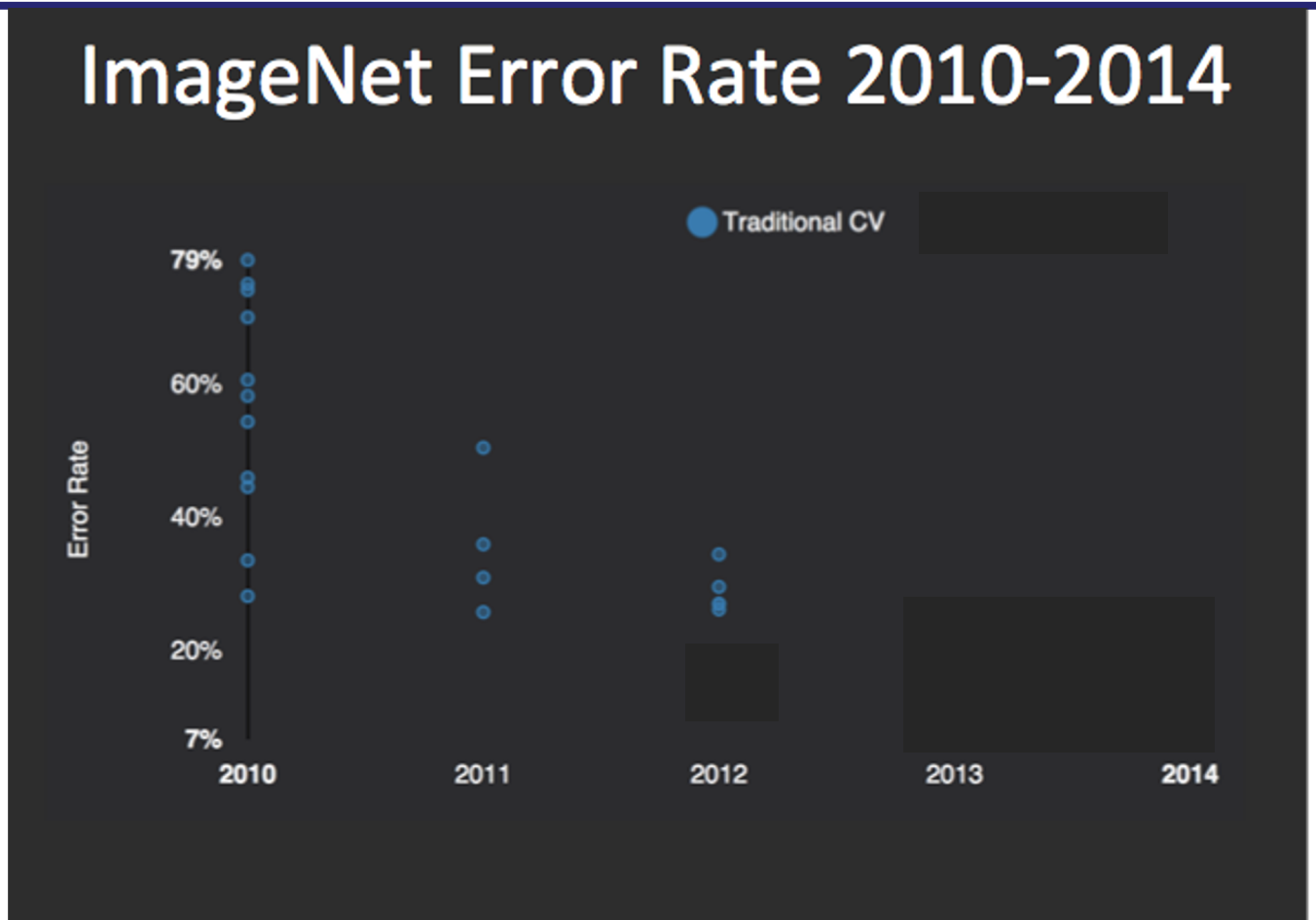
Outline

- Q-Learning
- ***Deep Learning / Neural Networks***
- Deep Q Networks (DQN)

What's a Neural Net?

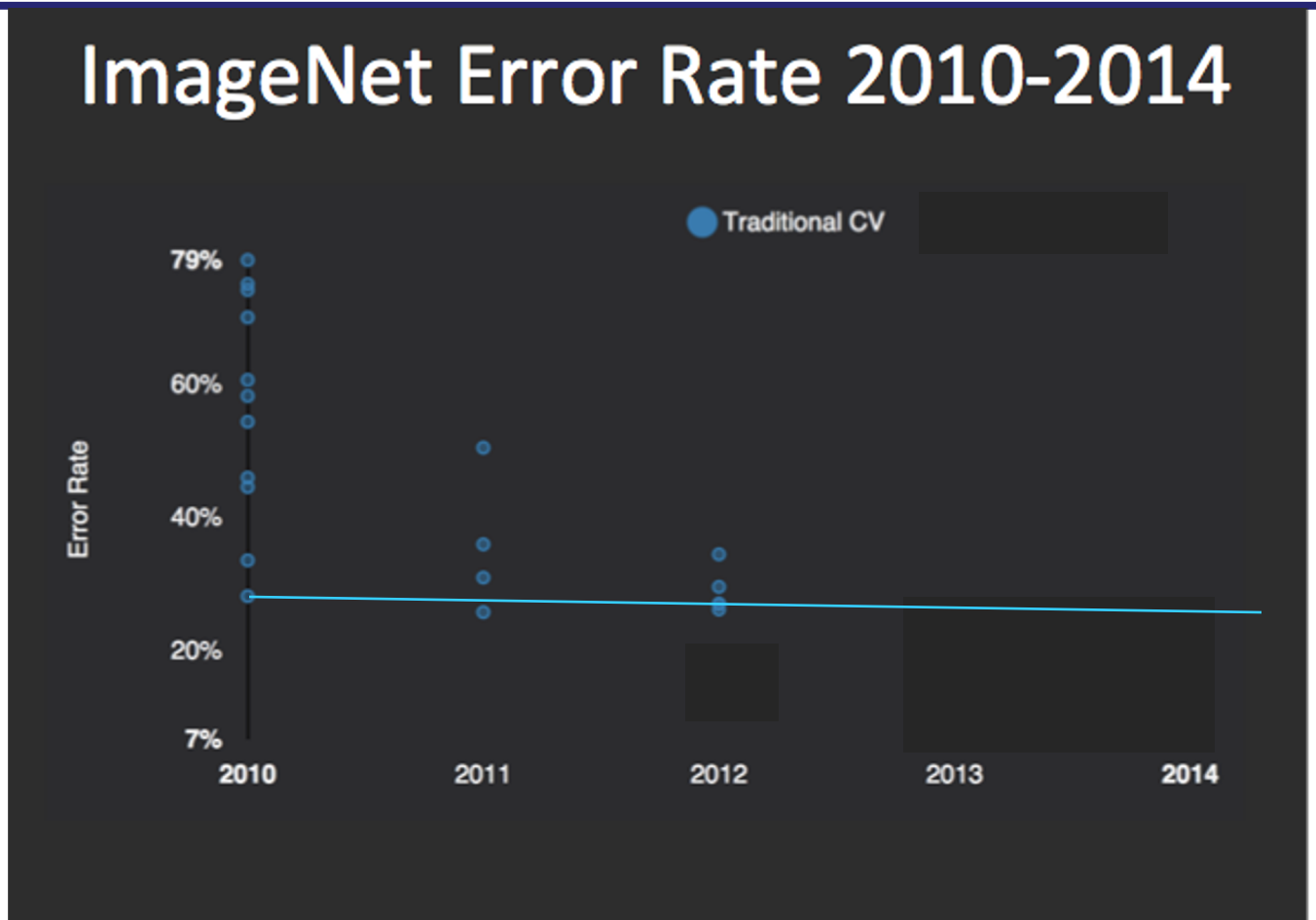


Performance



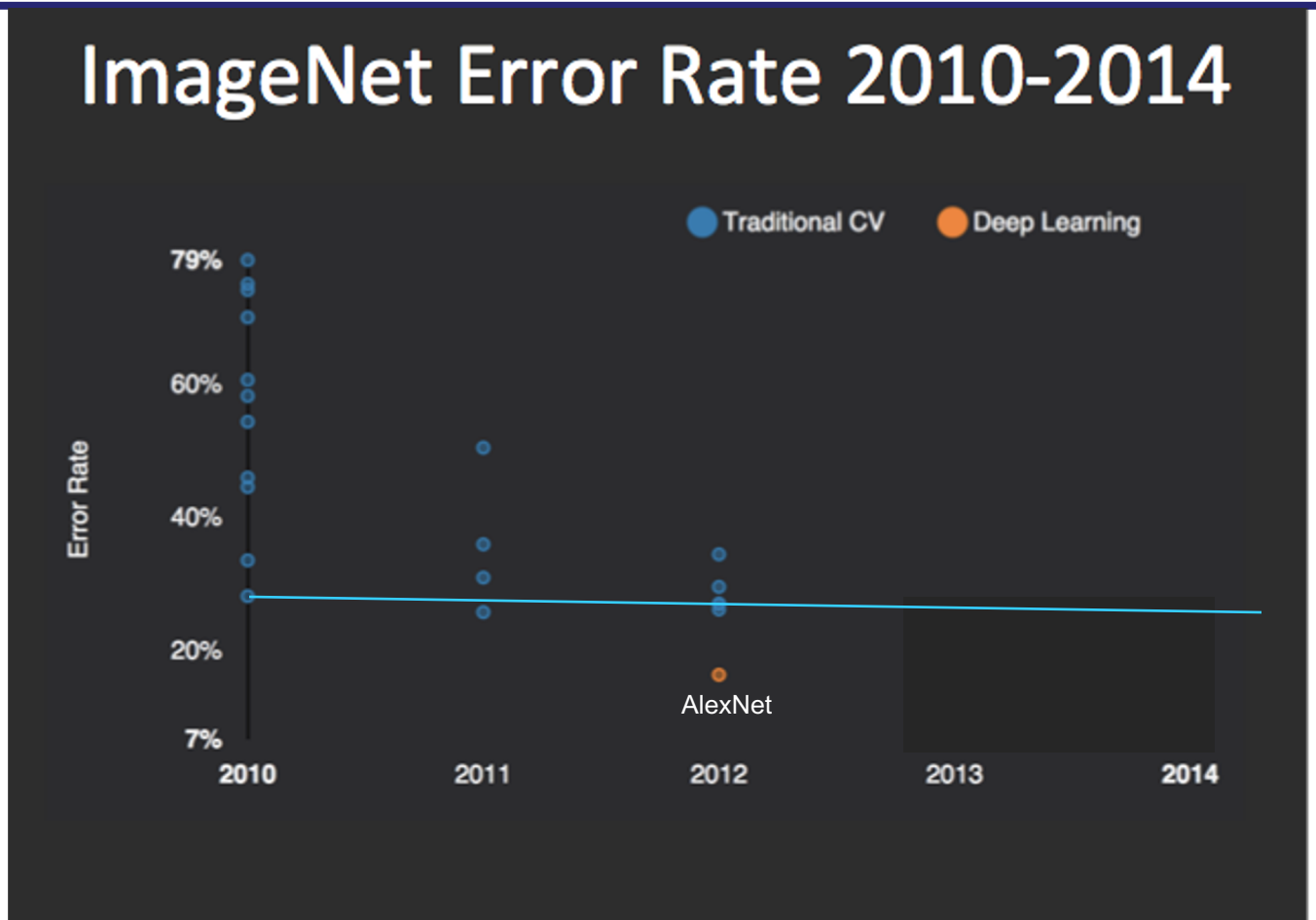
graph credit Matt Zeiler, Clarifai

Performance



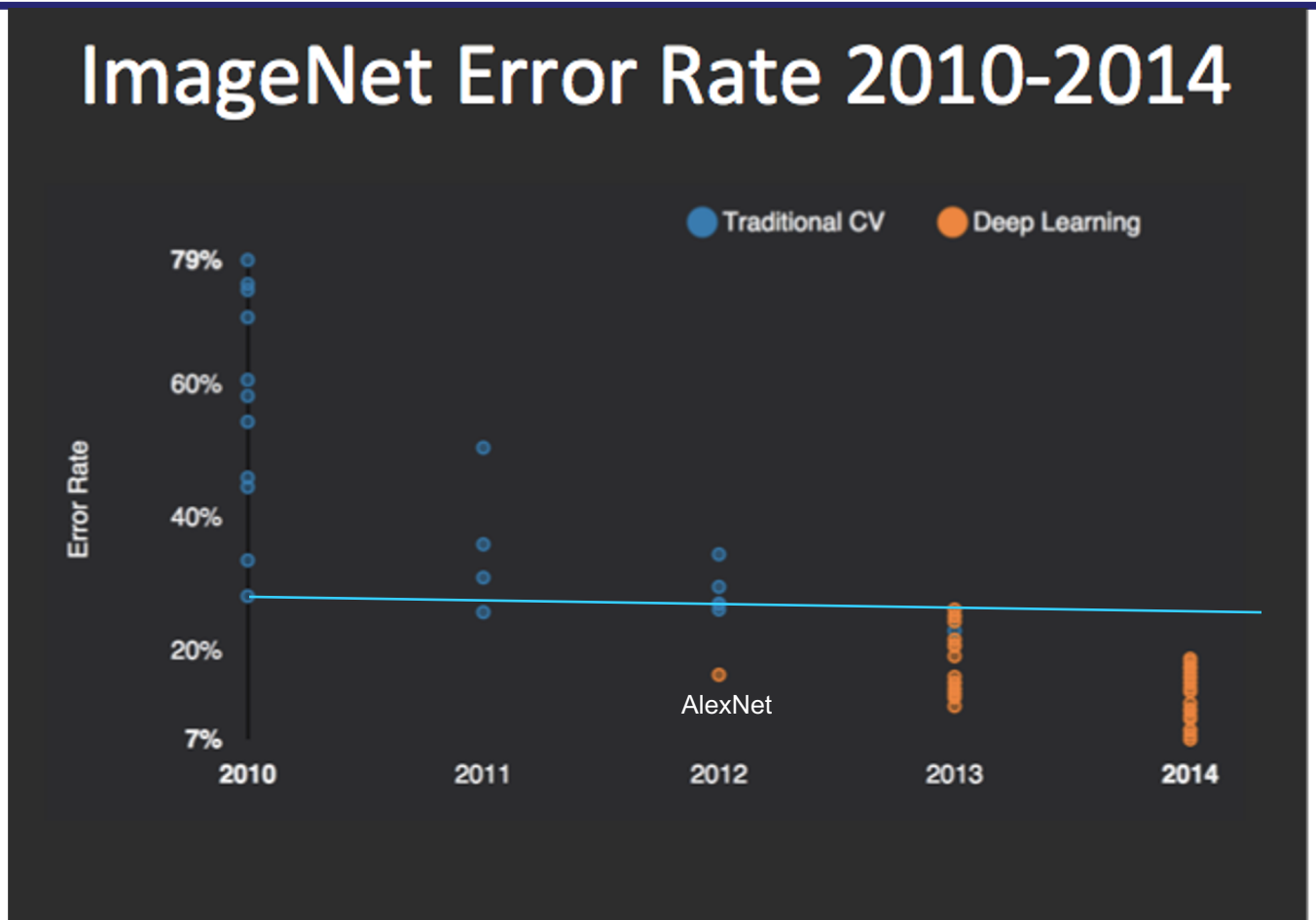
graph credit Matt Zeiler, Clarifai

Performance



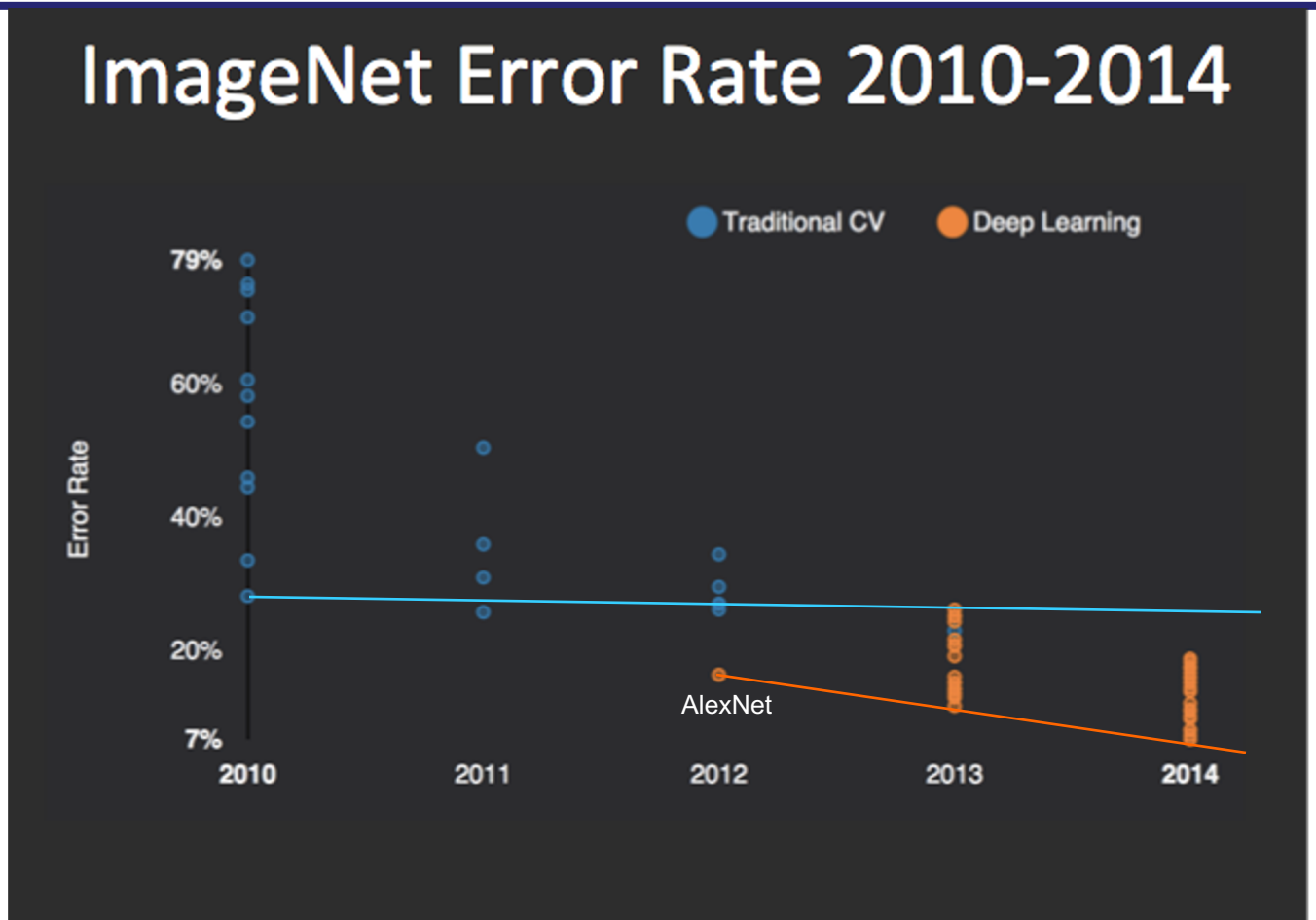
graph credit Matt Zeiler, Clarifai

Performance



graph credit Matt Zeiler, Clarifai

Performance



graph credit Matt Zeiler, Clarifai

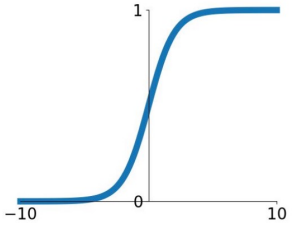
Multilayer Perceptron

- Linear function: $f(x) = Wx$
- MLP: stack linear functions and nonlinearity (activation function).
  - 2-layer Network: $f(x) = W_2 \max(0, W_0 x)$
  - 3-layer Network: $f(x) = W_3 \max(0, W_2 \max(0, W_0 x))$

Activation Functions

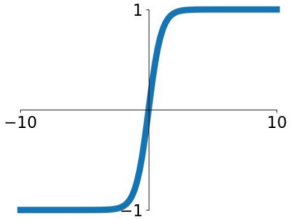
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



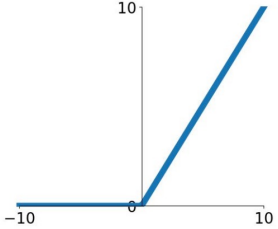
tanh

$$\tanh(x)$$



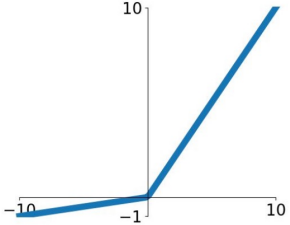
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

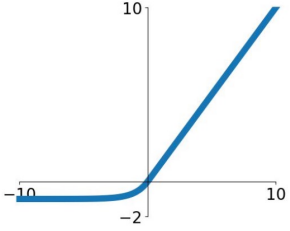


Maxout

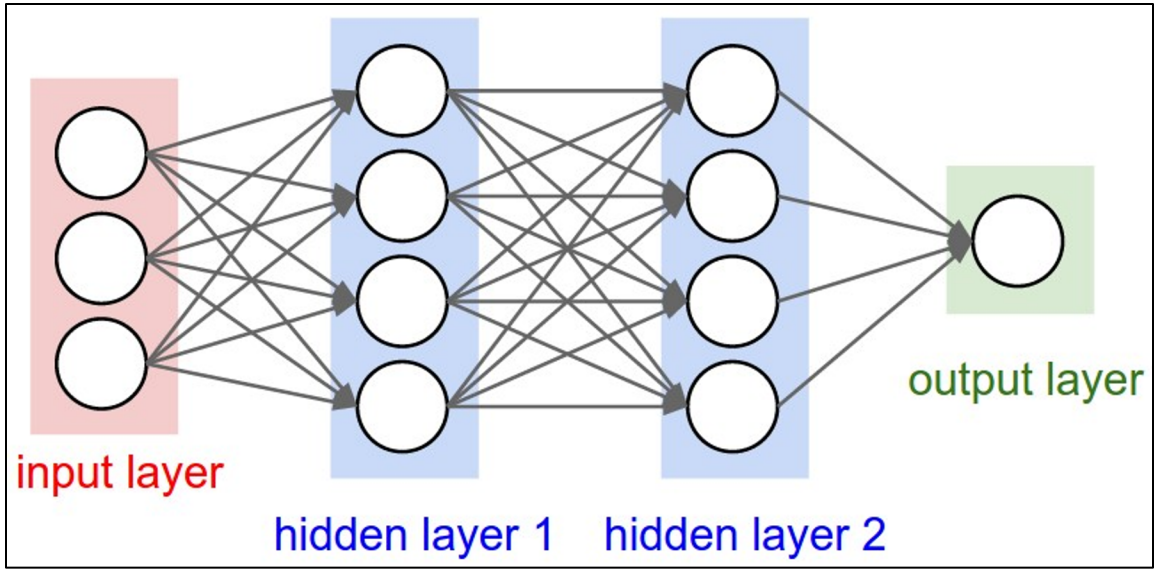
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

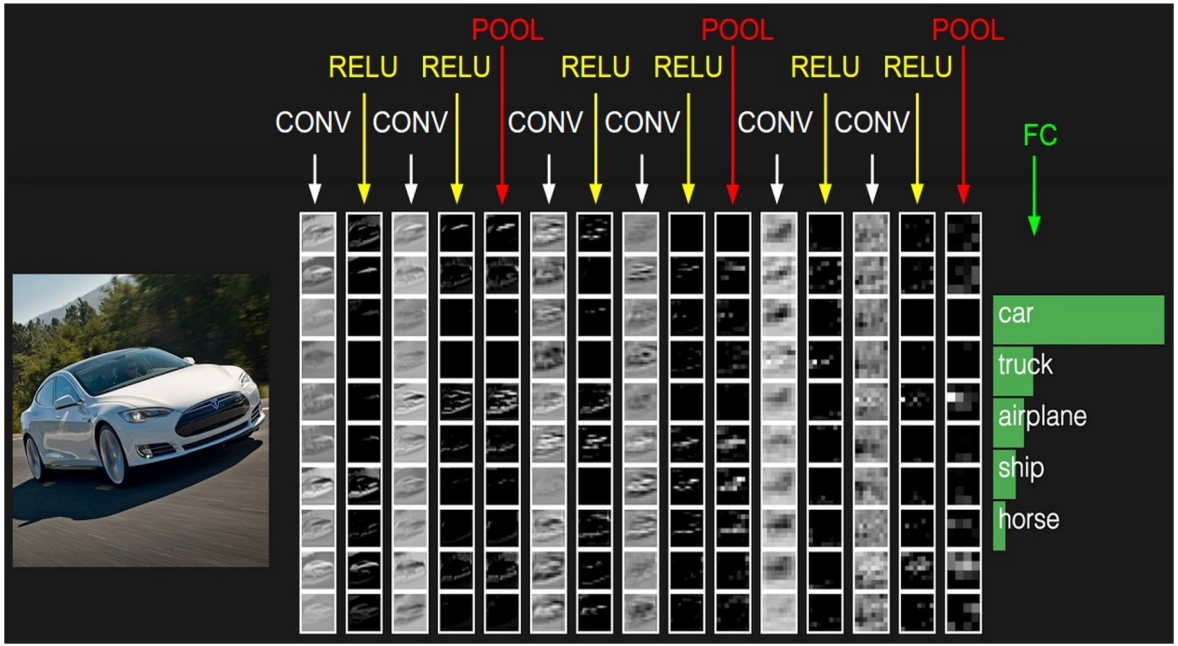


MLP



Some input vector (very few assumptions made).

Convolutional Neural Nets



Slides Credit: Andrej Karpathy

Optimization

- Non-convex problem, but
  - Gradient-based methods are surprisingly effective
  - Minibatch stochastic gradients instead of full gradient
  - Gradient calculation done by Automatic differentiation
    - Learn backpropogation
    - Use a deep learning framework: Tensorflow, Chainer, PyTorch..
  - Most common: SGD + Momentum + Preconditioning
    - Try RMSProp, Adam, Adamax…

Learn More:

- https://deeplearning.ai
- https://cs182sp21.github.io/
- http://cs231n.stanford.edu/

Outline

- Q-Learning
- Deep Learning / Neural Networks
- ***Deep Q Networks (DQN)***

Recall Approximate Q-Learning

- Instead of a table, we have a parametrized Q function
  - E.g. a neural net $Q_\theta(s, a)$
- Learning rule:
  - Compute target:

$$\text{target}(s') = R(s, a, s') + \gamma \max_{a'} Q_{\theta_k}(s', a')$$

  - Update Q-network:

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \left[\frac{1}{2} (Q_\theta(s, a) - \text{target}(s'))^2 \right] \Bigg|_{\theta=\theta_k}$$

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

    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

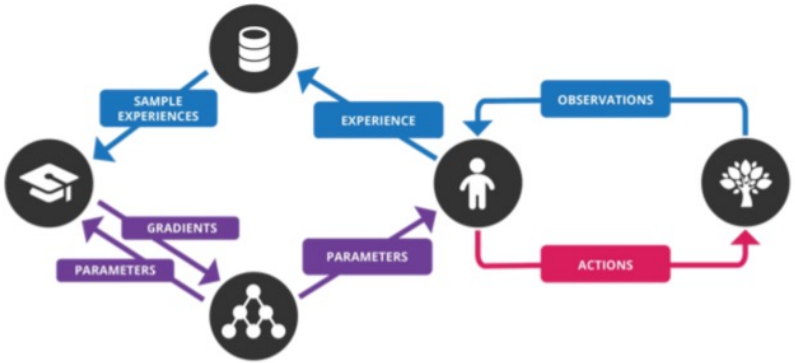
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

    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

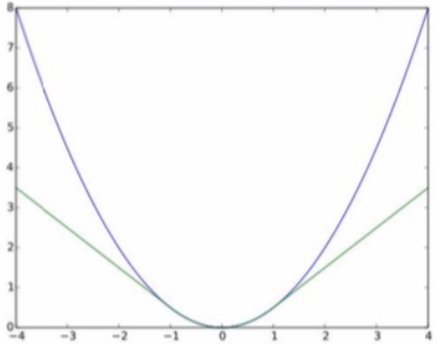
End For



DQN Details

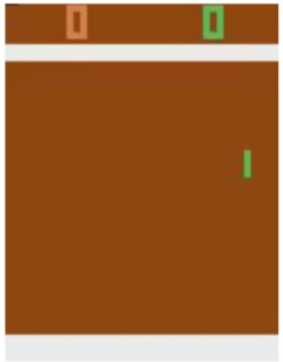
- Uses Huber loss instead of squared loss on Bellman error:

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \le \delta, \\ \delta(|a| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases}$$

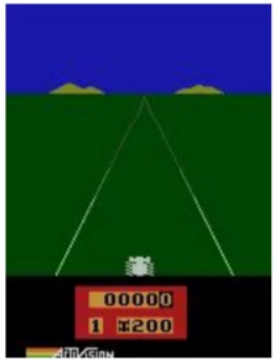


- Uses RMSProp instead of vanilla SGD.
  - Optimization in RL really matters.
- It helps to anneal the exploration rate.
  - Start ε at 1 and anneal it to 0.1 or 0.05 over the first million frames.

DQN on ATARI

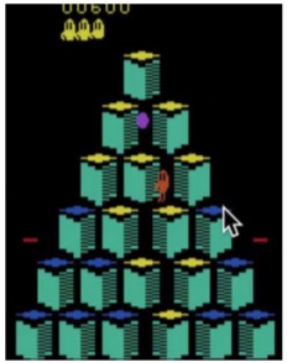


Pong



Enduro

Beamrider

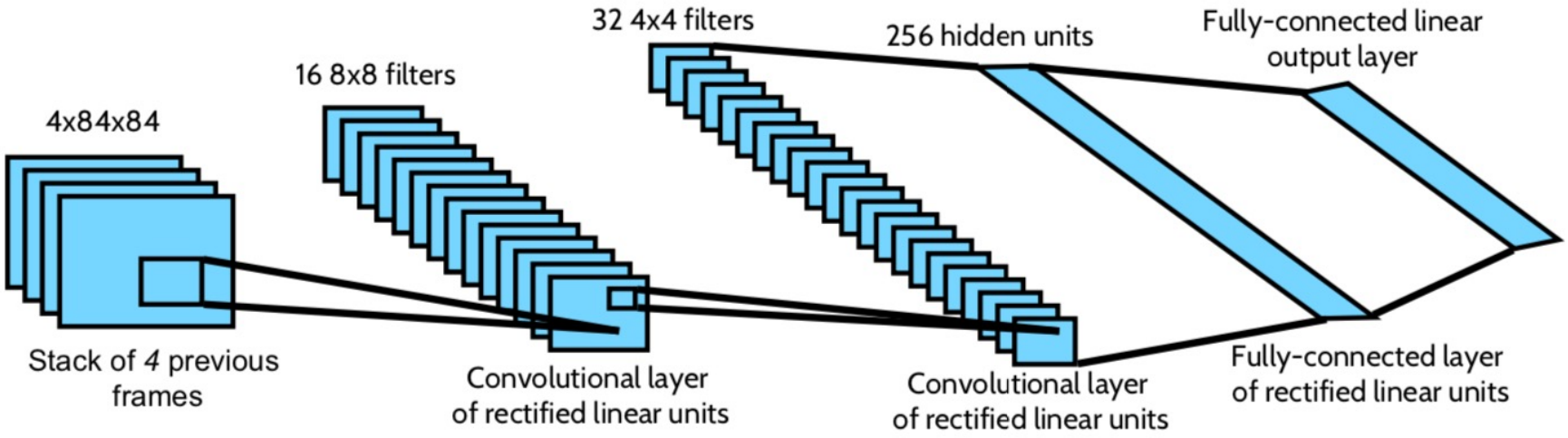


Q*bert

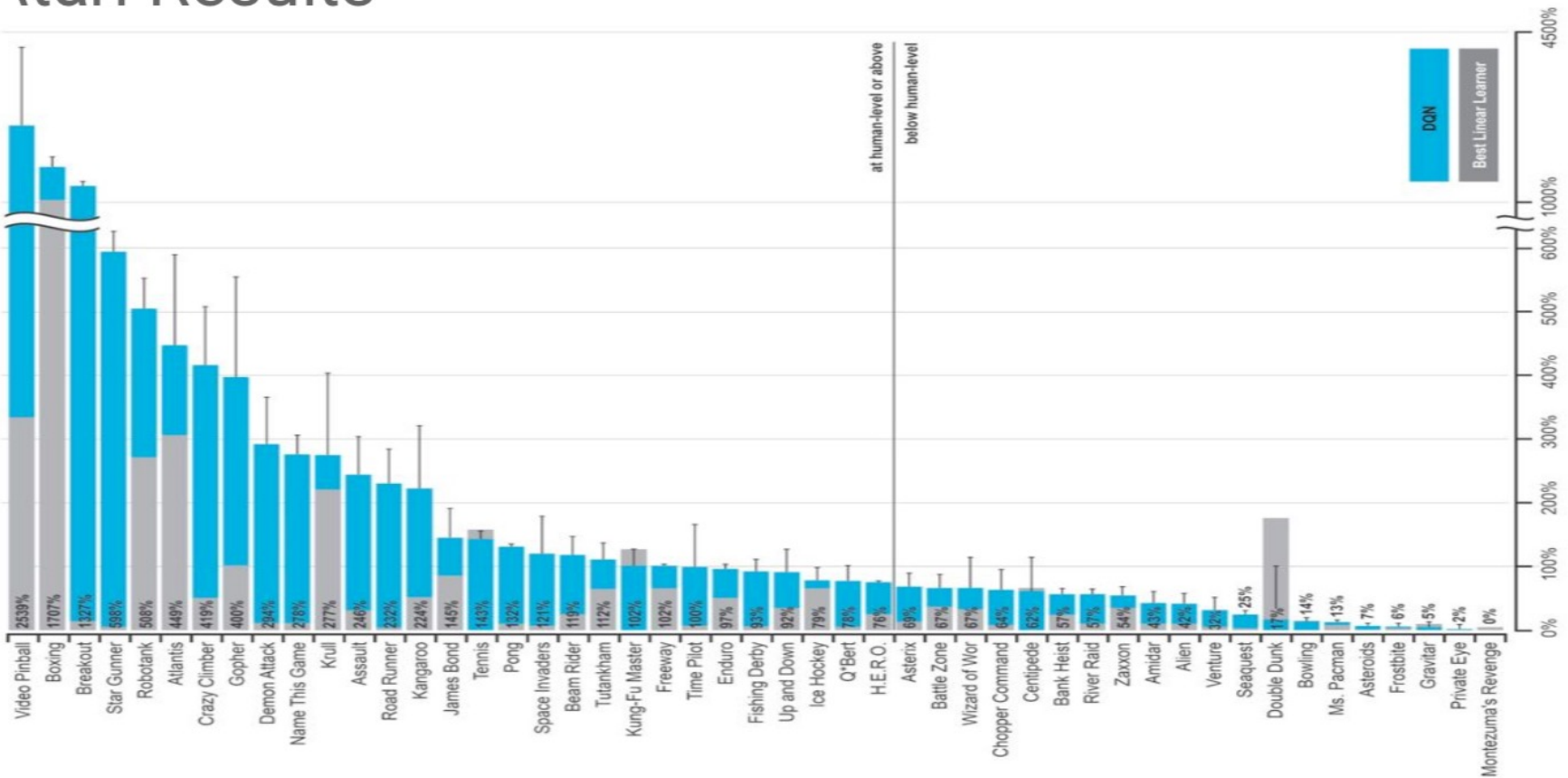
- 49 ATARI 2600 games.
- From pixels to actions.
- The change in score is the reward.
- Same algorithm.
- Same function approximator, w/ 3M free parameters.
- Same hyperparameters.
- Roughly human-level performance on 29 out of 49 games.

ATARI Network Architecture

- Convolutional neural network architecture:
  - History of frames as input.
  - One output per action - expected reward for that action $Q(s, a)$.
  - Final results used a slightly bigger network (3 convolutional + 1 fully-connected hidden layers).



Atari Results



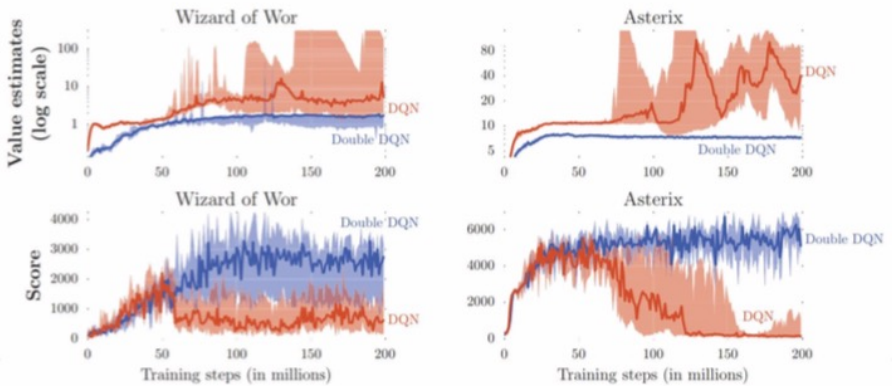
"Human-Level Control Through Deep Reinforcement Learning", Mnih, Kavukcuoglu, Silver et al. (2015)

Double DQN

- There is an upward bias in $max_a\ Q(s, a; \theta)$.
- DQN maintains two sets of weight θ and θ^-, so reduce bias by using:
  - θ for selecting the best action.
  - θ^- for evaluating the best action.
- Double DQN loss:

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r\ D}\left(r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta_i^-) - Q(s, a; \theta_i)\right)^2$$

	no ops		human starts		
	DQN	DDQN	DQN	DDQN	DDQN (tuned)
Median	93%	**115%**	47%	88%	**117%**
Mean	241%	**330%**	122%	273%	**475%**



DeepMind

"Double Reinforcement Learning with Double Q-Learning", van Hasselt et al. (2016)

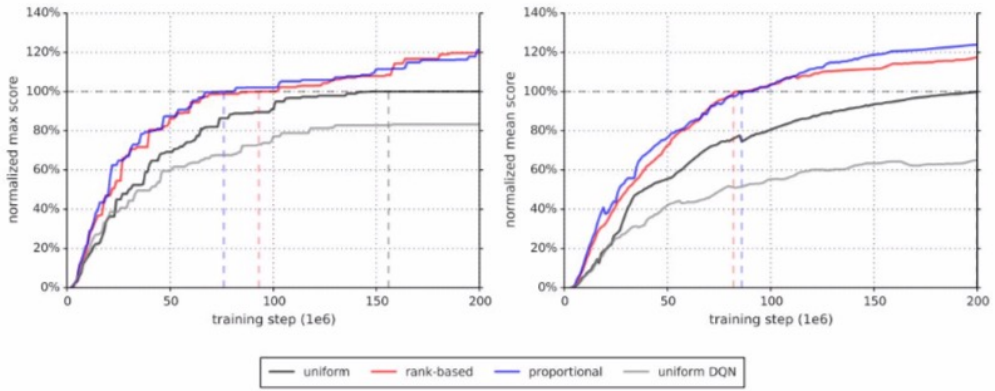
Prioritized Experience Replay

- Replaying all transitions with equal probability is highly suboptimal.
- Replay transitions in proportion to absolute Bellman error:

$$\left| r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right|$$

- Leads to much faster learning.

	DQN		Double DQN (tuned)		
	baseline	rank-based	baseline	rank-based	proportional
Median	48%	106%	111%	113%	128%
Mean	122%	355%	418%	454%	551%
> baseline	–	41	–	38	42
> human	15	25	30	33	33
# games	49	49	57	57	57



DeepMind

"Prioritized Experience Replay", Schaul et al. (2016)

See also

- “Rainbow: Combining Improvements in Deep Reinforcement Learning,” Matteo Hessel et al, 2017
  - Double DQN (DDQN)
  - Prioritized Replay DDQN
  - Dueling DQN
  - Distributional DQN
  - Noisy DQN