**Session XV: Deep Reinforcement Learning (Deep Q-Learning and OpenAI Lab)**

*Meeting date: December 9th, 2017*

We built on our coverage of Reinforcement Learning with this session. The primary focus was [Laura Graesser](https://www.linkedin.com/in/laura-graesser-60006039/) and [Wah Loon Keng](https://www.linkedin.com/in/theoriesinpractice/)'s interactive presentation (slides [here](https://github.com/the-deep-learners/study-group/blob/master/slides/2017-12-09__keng_laura__RL.pdf)) on Deep RL theory as well as their [OpenAI Lab](https://github.com/kengz/openai_lab) Python library. Secondarily, we discussed a number of Deep RL resources, including those listed in the Recommended Preparatory Work section below.

A summary blog post, replete with photos of the session, can be found [here](https://insights.untapt.com/openai-lab-for-deep-reinforcement-learning-experimentation-6287867eb611).

**Recommended Preparatory Work**

1. The first two ([one](https://www.youtube.com/watch?v=Q4kF8sfggoI&list=PLkFD6_40KJIznC9CDbVTjAF2oyt8_VAe3&index=1), [two](https://www.youtube.com/watch?v=C_LGsoe36I8&index=2&list=PLkFD6_40KJIznC9CDbVTjAF2oyt8_VAe3)) lectures from UC Berkeley CS294-112 **Fall 2017** Lectures, which are continuing to be released presently. Lecture 1 may be skimmed as it overlaps greatly with the material covered above.
2. Andrej Karpathy's [blog post](http://karpathy.github.io/2016/05/31/rl/) *Deep Reinforcement Learning: Pong from Pixels*
3. Additional resources to skim or dig your teeth into, depending on how much time you have:
   * [Deep RL](https://www.youtube.com/watch?v=lvoHnicueoE) (Lecture 14) from Stanford CS231n's Summer 2017 iteration (with Fei-Fei Li, Justin Johnson and Serena Young)
   * [Coach](https://github.com/NervanaSystems/coach)
   * [OpenAI Lab](https://github.com/kengz/openai_lab)
   * Kai Arulkumaran and colleagues' [arXiv review paper](https://arxiv.org/abs/1708.05866) *A Brief Survey of Deep Reinforcement Learning*
   * The *Reinforcement Learning* chapter (Chapter 16) from Aurélien Géron's [Hands-on Machine Learning book](http://shop.oreilly.com/product/0636920052289.do)

**Notes**

[**Deep Reinforcement Learning (Lecture 14)**](https://www.youtube.com/watch?v=lvoHnicueoE&index=14&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv)**from Stanford CS231n's Summer 2017 iteration (with Fei-Fei Li, Justin Johnson and Serena Young)**

* lecturer for this lecture is Serena Young
* <https://www.youtube.com/watch?v=lvoHnicueoE&index=14&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv>

**Broad Topic Areas**

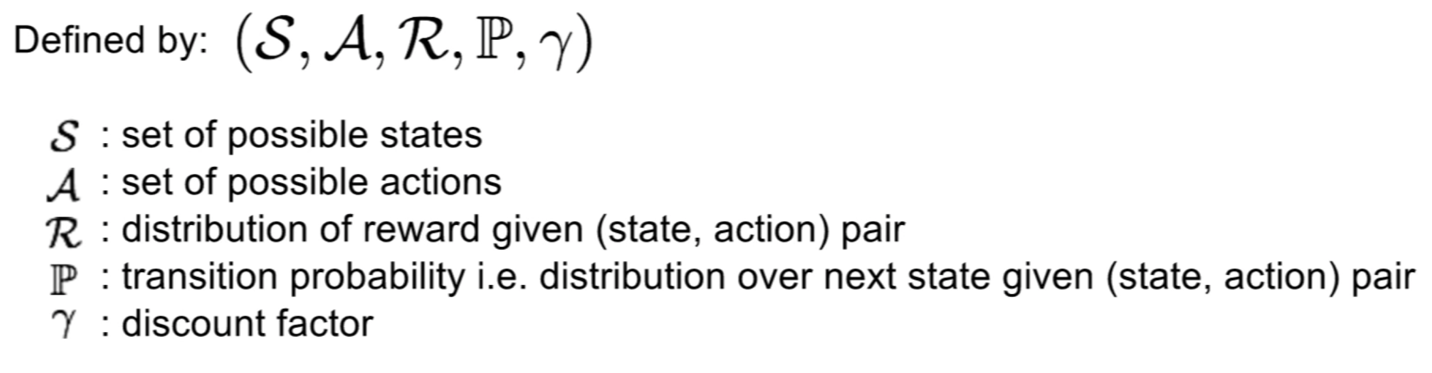
* Supervised Learning
  + have *x* and *y*
  + goal is to approximate function that predicts *y* with *x*
  + examples in CS231n:
    - classification
    - regression
    - object detection
    - semantic segmentation
    - image captioning
    - &c.
* Unsupervised Learning
  + have *x* alone without labels
  + goal is to "learn some underlying hidden *structure* of the data"
  + examples in CS231n:
    - clustering
    - dimensionality reduction
    - feature learning
    - density estimation
    - &c.
    - (for J.K.'s purposes, w2v)
* Reinforcement Learning
  + an **agent** takes action in an **environment**
    - environment returns **state** information:
      * **reward** at time *t*
      * **state** at time *t+1*
    - repeat
    - (equivalent to Emma Brunskill's slide in the previous session)
  + goal is to learn actions that maximize reward
  + the focus of this lecture, d'uh

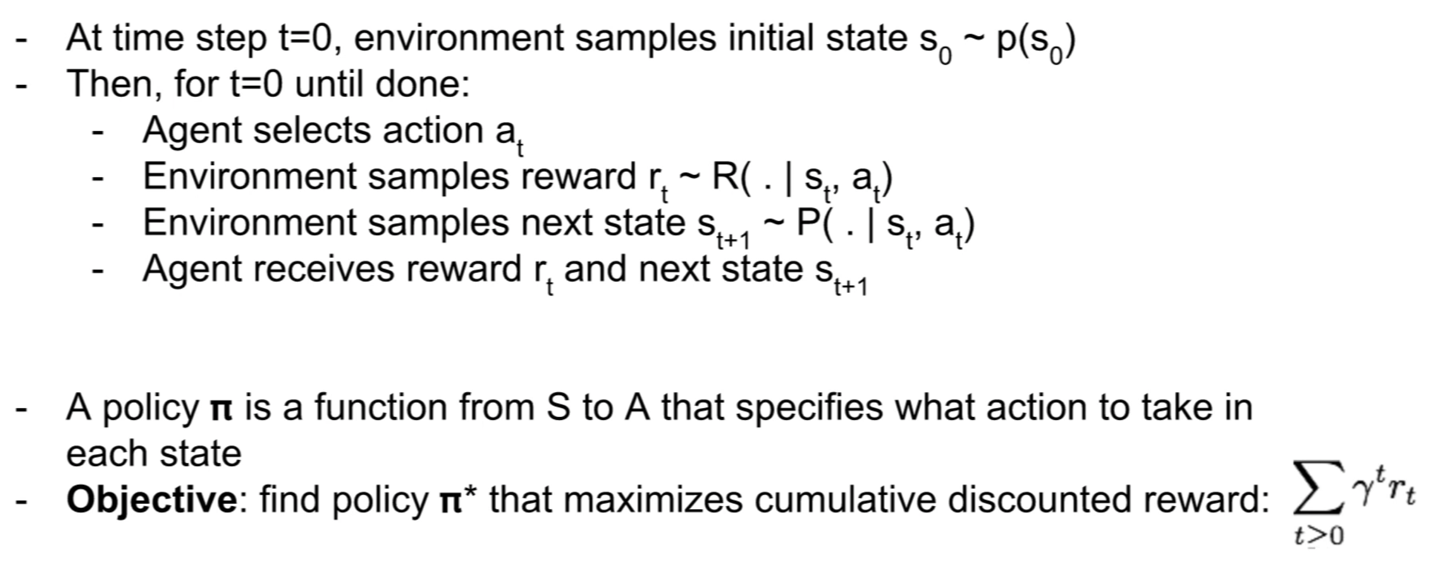
**What is RL?**

* exemplars:
  + Cart-Pole Problem
  + Robot Locomotion
  + Atari Games
  + Go

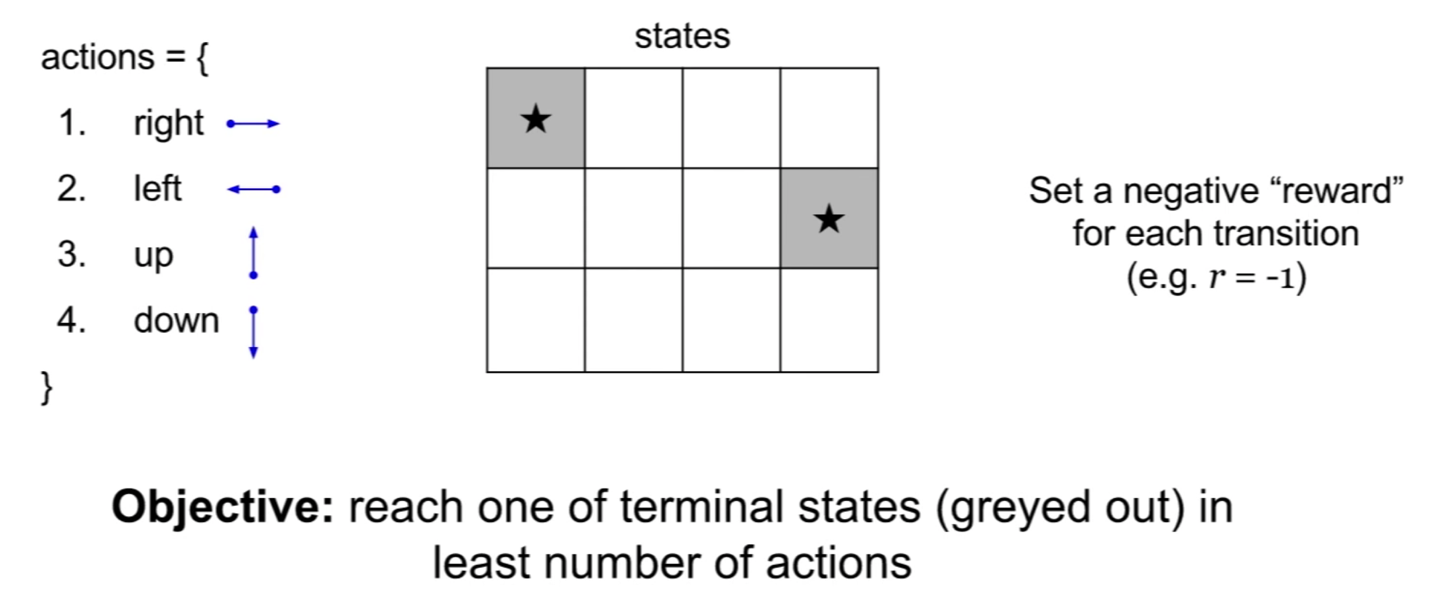
**Markov Decision Processes**

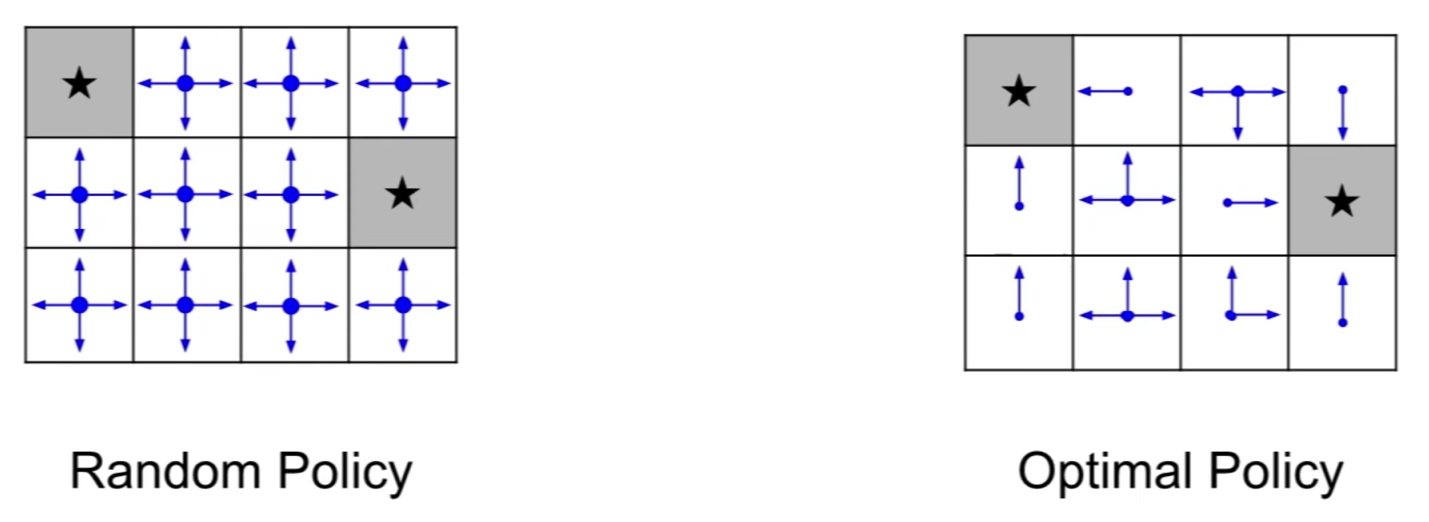
* Markov Decision Process (MDP)
  + a way to define RL mathematically (again, as in Emma's slide)
  + features the **Markov property** that the current state contains all of the relevant information from previous states, i.e., it "completely characterises the state of the world"

[](https://github.com/the-deep-learners/study-group/blob/master/weekly-work/week15/img/mdp-defn.png)

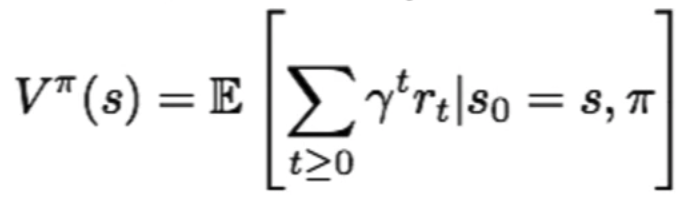
[](https://github.com/the-deep-learners/study-group/blob/master/weekly-work/week15/img/mdp-process.png)

* "Grid World" is a simple MDP:

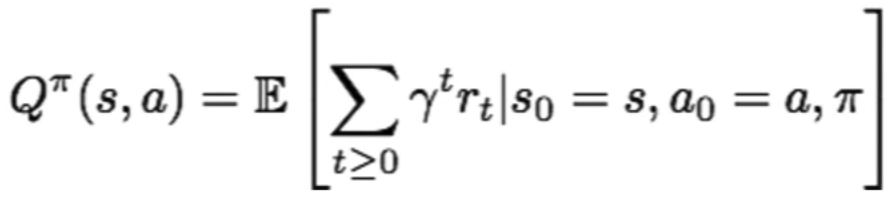
[](https://github.com/the-deep-learners/study-group/blob/master/weekly-work/week15/img/grid-world-1.png)

[](https://github.com/the-deep-learners/study-group/blob/master/weekly-work/week15/img/grid-world-2.png)

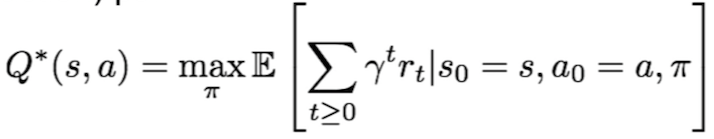
* to find the optimal **policy**, we maximize the expected sum of rewards
  + this helps us cope with randomness in the system
* first, here are a few definitions that will help us discuss theory for identifying the optimal policy:
* **value function**:
  + lets us understand how valuable a given state *s* is
  + formally, it's the "expected cumulative reward from following the policy from state *s*" (yes, this makes good sense; read it again)

[](https://github.com/the-deep-learners/study-group/blob/master/weekly-work/week15/img/value-fxn.png)

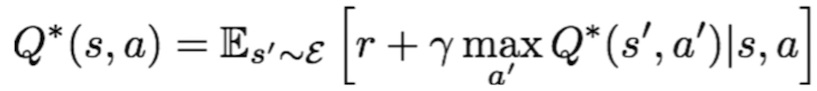
* **Q-value function**:
  + lets us understand how valuable a given pair of state *s* and action *a* are
  + formally, it's the "expected cumulative reward from taking action *a* in state *s* and then following the policy" (this makes very good sense as well!)

[](https://github.com/the-deep-learners/study-group/blob/master/weekly-work/week15/img/Qvalue-fxn.png)

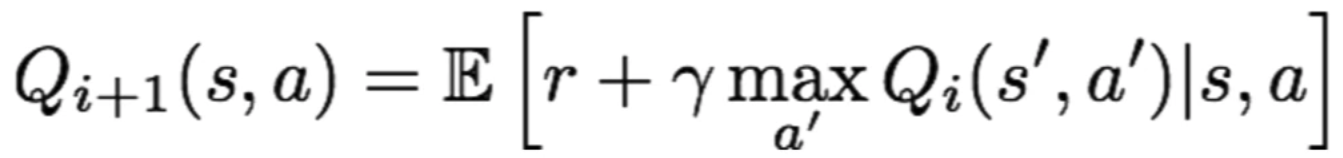
* \*\*Q\* \*\*:
  + the optimal Q-value function
  + given a (state, action) pair, this is the optimal Q-value function
  + it is defined as the "maximum expected cumulative reward achieveable"
  + it is "expected" because there's randomness in the system

[](https://github.com/the-deep-learners/study-group/blob/master/weekly-work/week15/img/Q-star.png)

* **Bellman equation**:
  + satisfied by Q\*
  + the intuition here is that it's recursive: "if the optimal state-action values for the next time-step Q\*(s',a') are known, then the optimal strategy is to take the action that maximizes the expected value of *r* + *gamma*Q(s',a')"

[](https://github.com/the-deep-learners/study-group/blob/master/weekly-work/week15/img/bellman-exn.png)

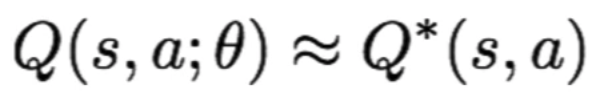
* **value iteration** algorithm:
  + update iteratively over the Bellman equation, tracing all steps
  + tracing all steps lets us approximate the best action at the current step
  + as *i* in the following equation approaches infinity, Q\_i will converge on Q\*
  + the flaw is that this is approach doesn't scale:
    - would require computing Q(s,a) for every state-action pair
    - if, e.g., the state space is the Atari game screen, this is an intractibly large space

[](https://github.com/the-deep-learners/study-group/blob/master/weekly-work/week15/img/value-itn-algo.png)

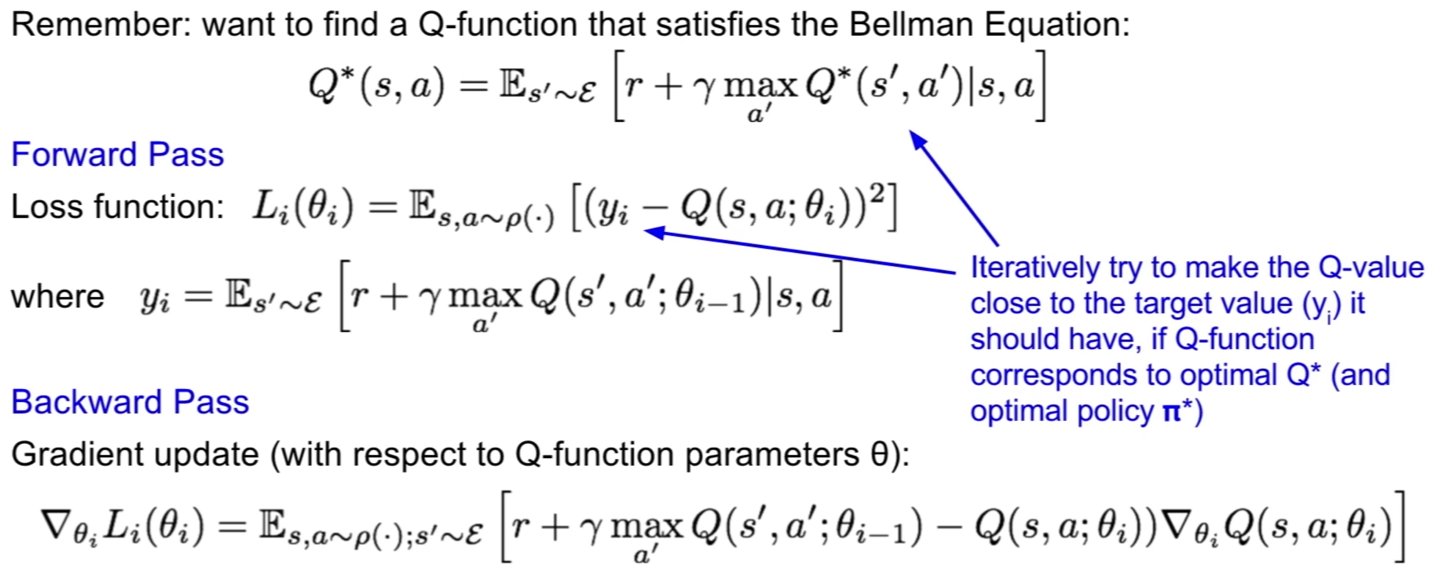
* conveniently, there is a solution: use a function approximator, e.g., an artificial neural network, to estimate Q(s,a)...

**Q-Learning**

* Q-learning is that solution!
* Q-learning is the "use of a function approximator to estimate the following action-value function", where theta is our function parameters (weights)

[](https://github.com/the-deep-learners/study-group/blob/master/weekly-work/week15/img/q-learning-fxn.png)

* **deep Q-learning**: Q-learning where the function approximator is a deep neural network
* the detail here doesn't need to be dwelt on at this time, but for reference, here are the loss function (calculated during forward pass) and the gradient updated (calculated during backward pass) when finding a Q-function that satisfies the Bellman equation with a Deep NN:

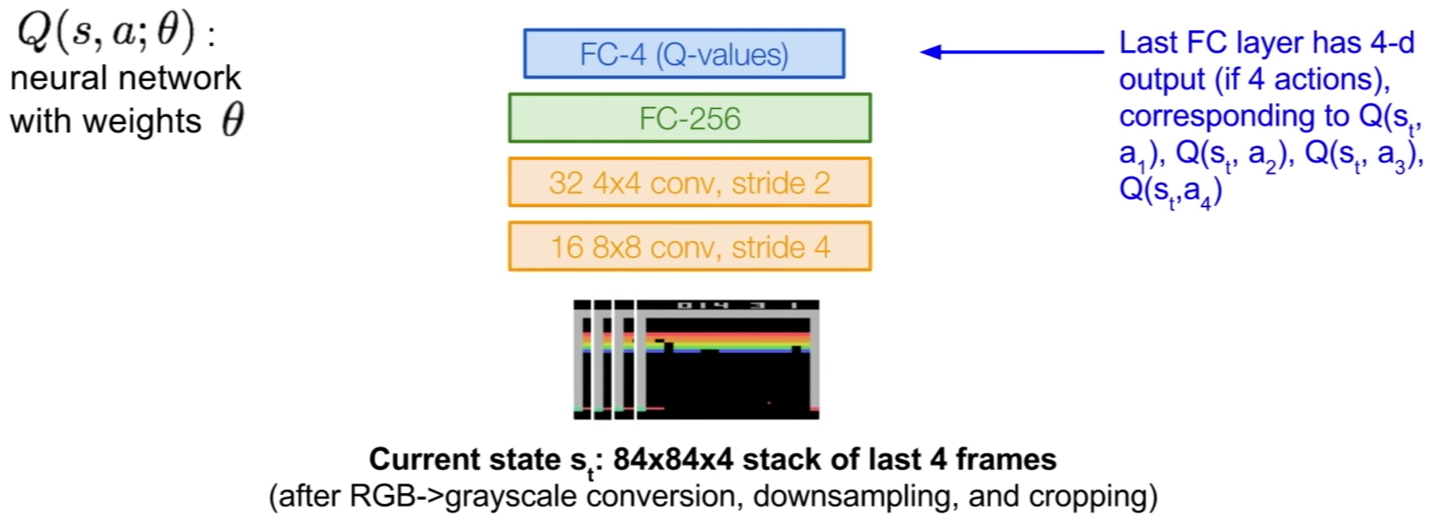
[](https://github.com/the-deep-learners/study-group/blob/master/weekly-work/week15/img/dnn-for-q-learning.png)

**Q-Learning Case Study: Playing Atari with Deep RL**

* [Mnih et al. (2013; NIPS)](https://arxiv.org/abs/1312.5602)
* [Mnih et al. (2015; Nature)](https://www.nature.com/articles/nature14236)

[](https://github.com/the-deep-learners/study-group/blob/master/weekly-work/week15/img/atari-case-study.png)

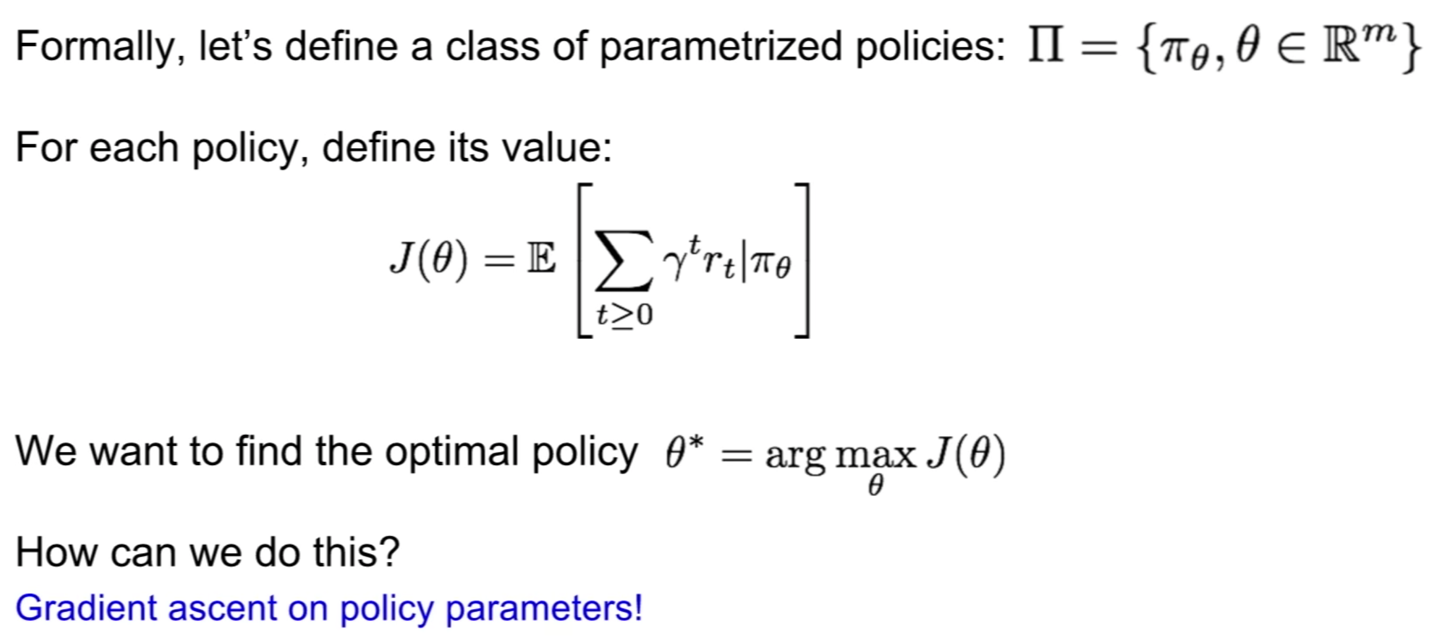
* the input (state *s\_t*) is the 84x84x4 stack
* no softmax layer because we're directly predicting Q
* the "four" actions is because it's this particular game with only four actions (up, down, left, right); Atari games have up to 18 actions, so last dense layer could be up to 18-dimensional
* because one feedforward pass computes Q-values for all actions from the current state, the Deep NN is an efficient learner

[](https://github.com/the-deep-learners/study-group/blob/master/weekly-work/week15/img/atari-case-study2.png)

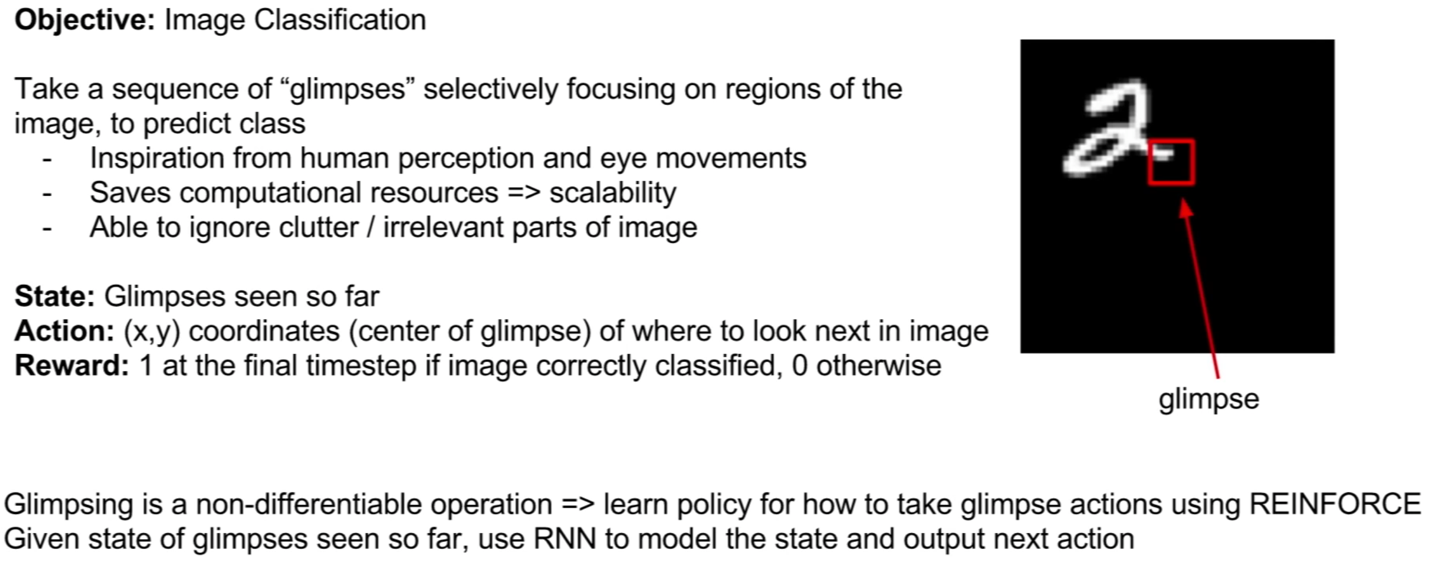
* training from consecutive samples is highly *inefficient* because consecutive samples are highly correlated
* not only is this inefficient, it can also lead to powerful, unhelpful *feedback loops* where, if, e.g. "maximizing action is to move left, training samples will be dominated by samples from left-hand" side
* both the inefficiency and feedback loop problem can be overcome by **experience replay**:
  + "continually update a **replay memory** table of transitions" as game episodes are played
  + subsequently, instead of training on consecutive samples, train the Q-network on random minibatches sampled from the replay memory
* Serena helpfully goes through every step of the Deep Q-Learning with Experience Replay in her talk
* a video of this algorithm in action learning to play breakout (the four-action game shown in screenshots above) over epochs is [available here](https://www.youtube.com/watch?v=V1eYniJ0Rnk)

**Policy Gradients**

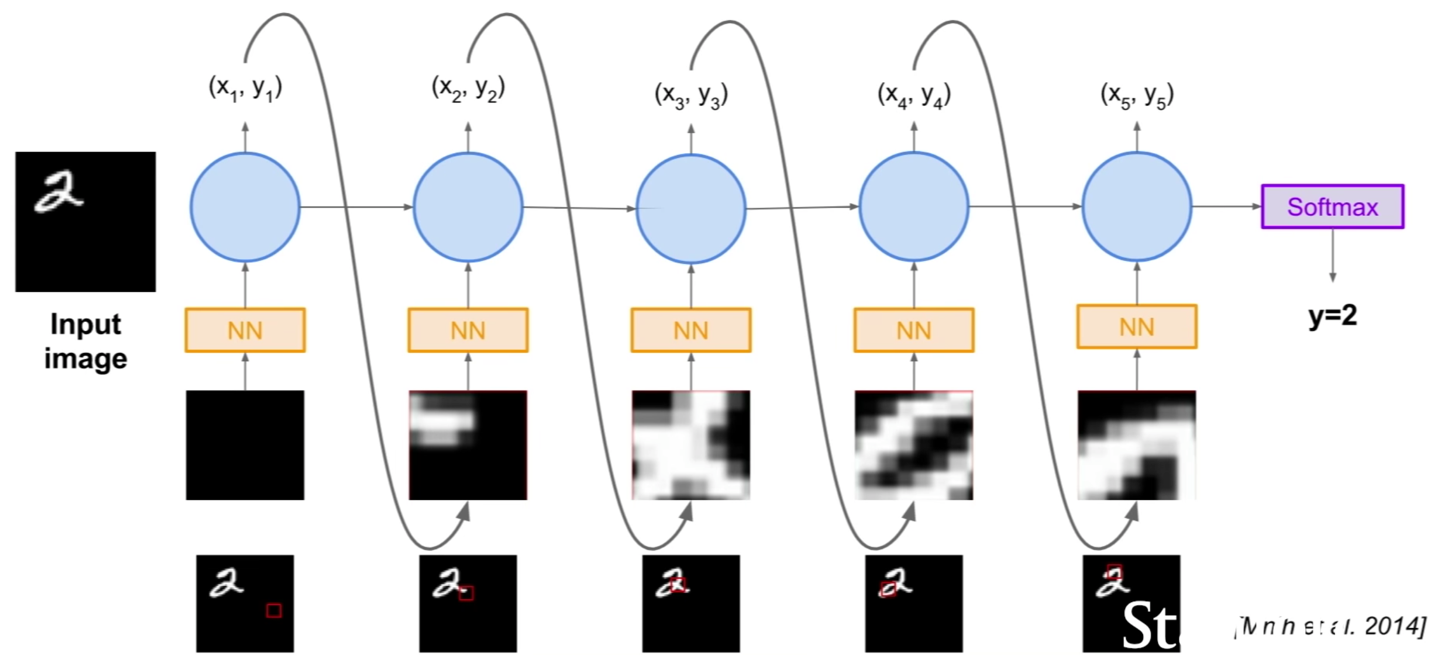
* Q-learning isn't perfect
* specifically, the "Q-function can be very complicated", e.g., if we're teaching a robot to grasp an object the exact value of every (state, action) pair is very high-dimensional (there are countless robothand positions and angles)
* the policy could be much simpler if we could learn the policy directly, e.g., identifying the optimal policy from a set of policies

[](https://github.com/the-deep-learners/study-group/blob/master/weekly-work/week15/img/policy-grad-defn.png)

* more specifically, we can use the **REINFORCE algorithm** (Serena provides detailed equations in her talk, over several slides, for deriving the policy gradients)
* **Actor-Critic Algorithm**:
  + combines Policy Gradients and Q-Learning by training an **actor** (policy) and **critic** (Q-function)
  + step-by-step:
    - the actor decides on an action to take
    - the critic evaluates the actor's action, providing feedback on how to adjust
  + can incorporate Q-learning tricks like experience replay
  + its **advantage function** provides information on how much better a given action was than was expected
* **REINFORCE** in action in [Mnih et al. (2014)](https://arxiv.org/abs/1406.6247)'s Recurrent Attention Model (RAM):

[](https://github.com/the-deep-learners/study-group/blob/master/weekly-work/week15/img/reinforce-in-axn.png)

* instead of passing a convolutional filter over the entire image, the REINFORCE approach is far more computationally efficient because it attends to important details in the image while ignoring irrelevant parts:

[](https://github.com/the-deep-learners/study-group/blob/master/weekly-work/week15/img/reinforce-in-axn2.png)

* Serena wraps up the lecture by detailing AlphaGo's algorithm, which incorporates many of the above concepts

**Summary**

* **policy gradients**:
  + general, but "suffer from high variance so require a lot of samples", i.e., its shortcoming is *sample-inefficiency*
  + *guaranteed* to converge to a local minimum of J(theta), which is often good enough
* **Q-learning**:
  + doesn't always apply, but typically more sample-efficient when it does, i.e., its shortcoming is *exploration-insufficiency*
  + *no guarantees* because it involves approximating the Bellman equation with a complex function approximator