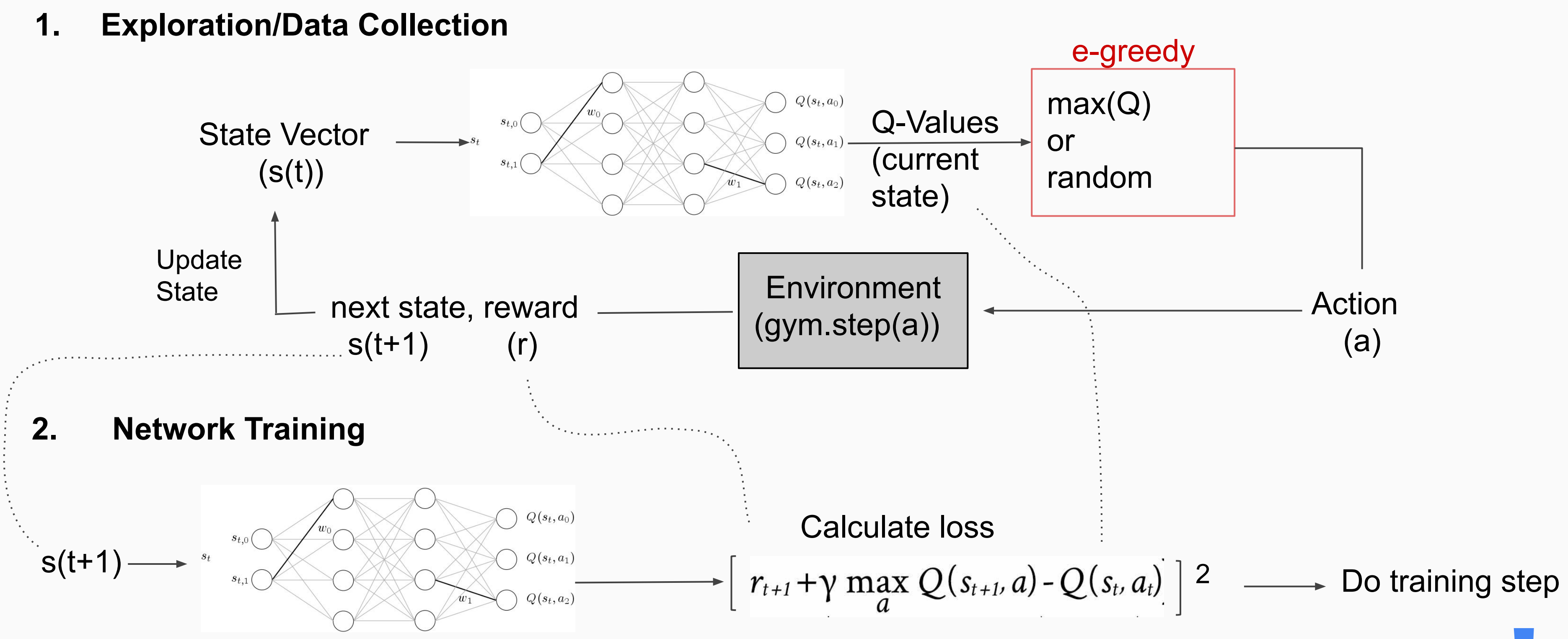
StarAi: Deep Reinforcement Learning

Lesson 4: Neural Q-Learning

Overall NQL Algorithm

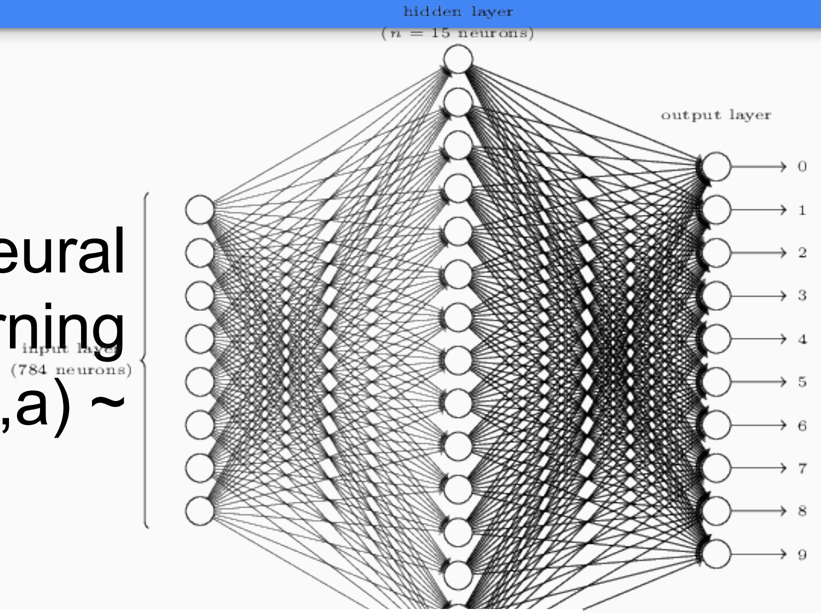


NQL vs DQN

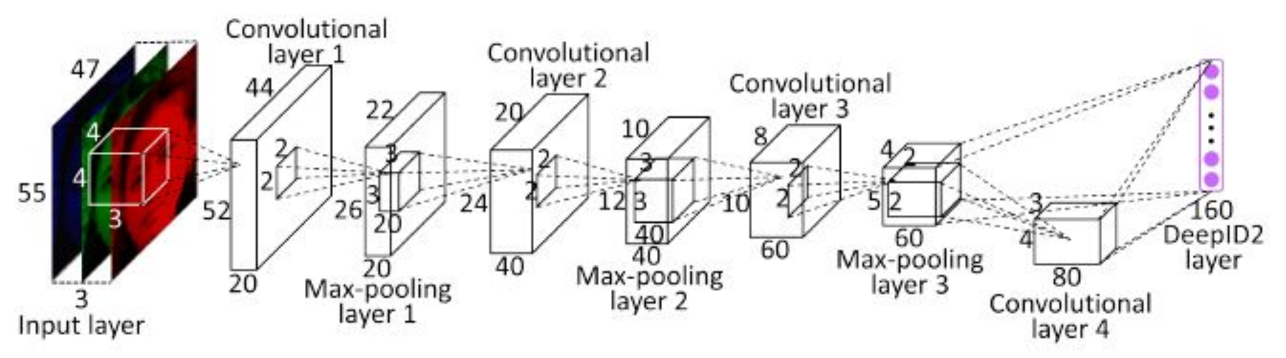
- **NQL** is JUST the use of a Neural Network to approximate the Q-function. NO TRICKS.

- **DQN** is ambiguous - but usually refers to the Minh 2015 algorithm (ConvNet + tricks to make it work)

Neural Q-Learning
Q(s,a) ~



DQN
Q(s,a) ~



Where are we right now?

The learning rate, i.e. that extent to which new information overrides old information. This is a number between 0 and 1.

The Q function we are updating, based on state **s** and action **a** at time *t*

The reward earned when transitioning from time *t* to the next next turn, time *t*+1.

The value of the action that is estimated to return the largest (i.e. maximum) total future reward, based on the all possible actions that can be made in the next state.

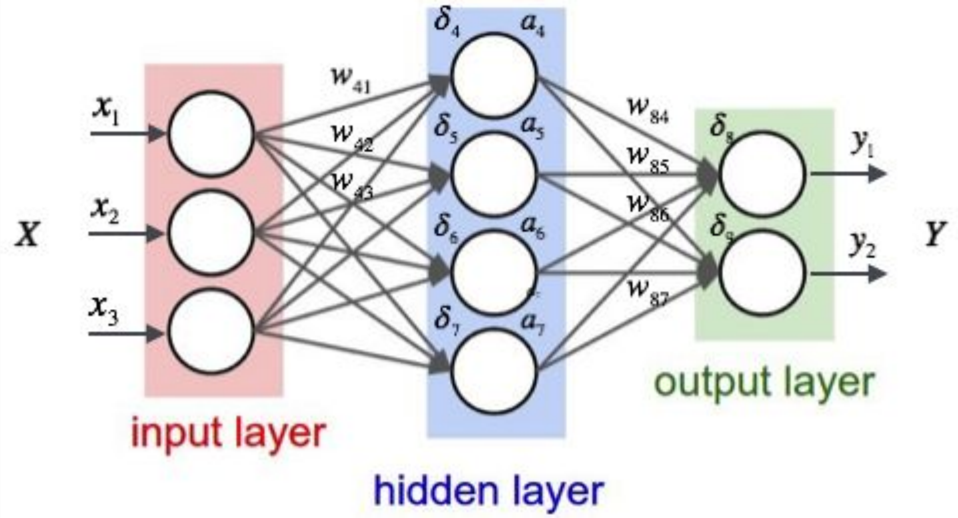
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

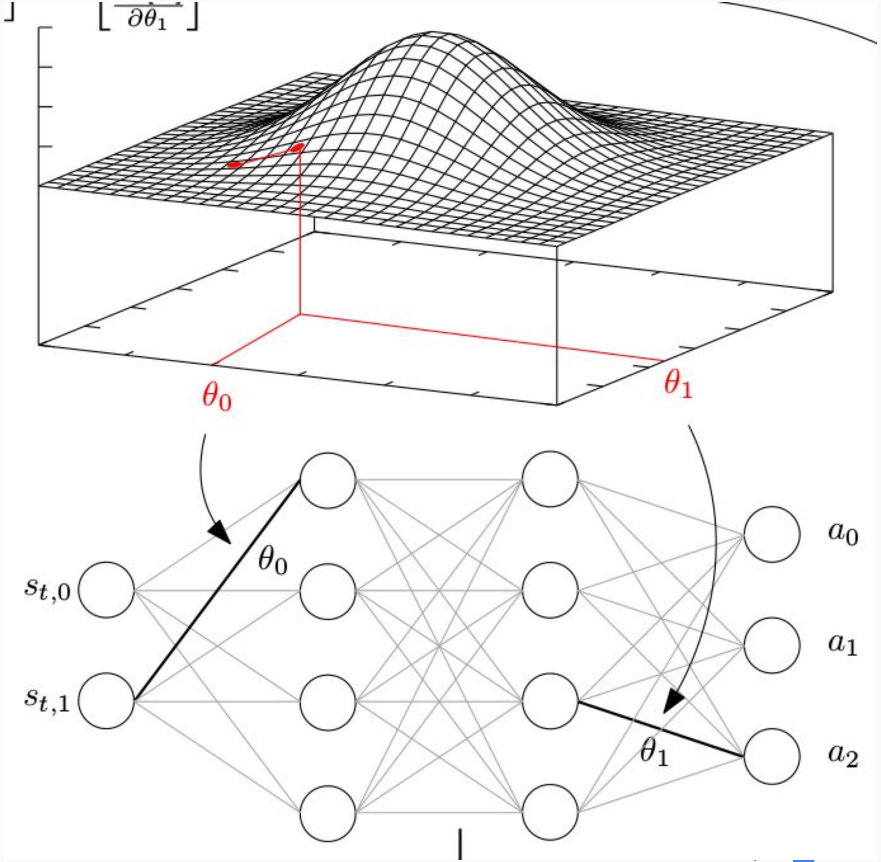
The arrow operator means update the Q function to the left. This is saying, add the stuff to the right (i.e. the difference between the old and the new estimated future reward) to the existing Q value. This is equivalent in programming to A = A+B.

The discount rate. Determines how much future rewards are worth, compared to the value of immediate rewards. This is a number between 0 and 1

The existing estimate of the **Q** function, (a.k.a. current the action-value).

Quick Review: Neural Network Maths

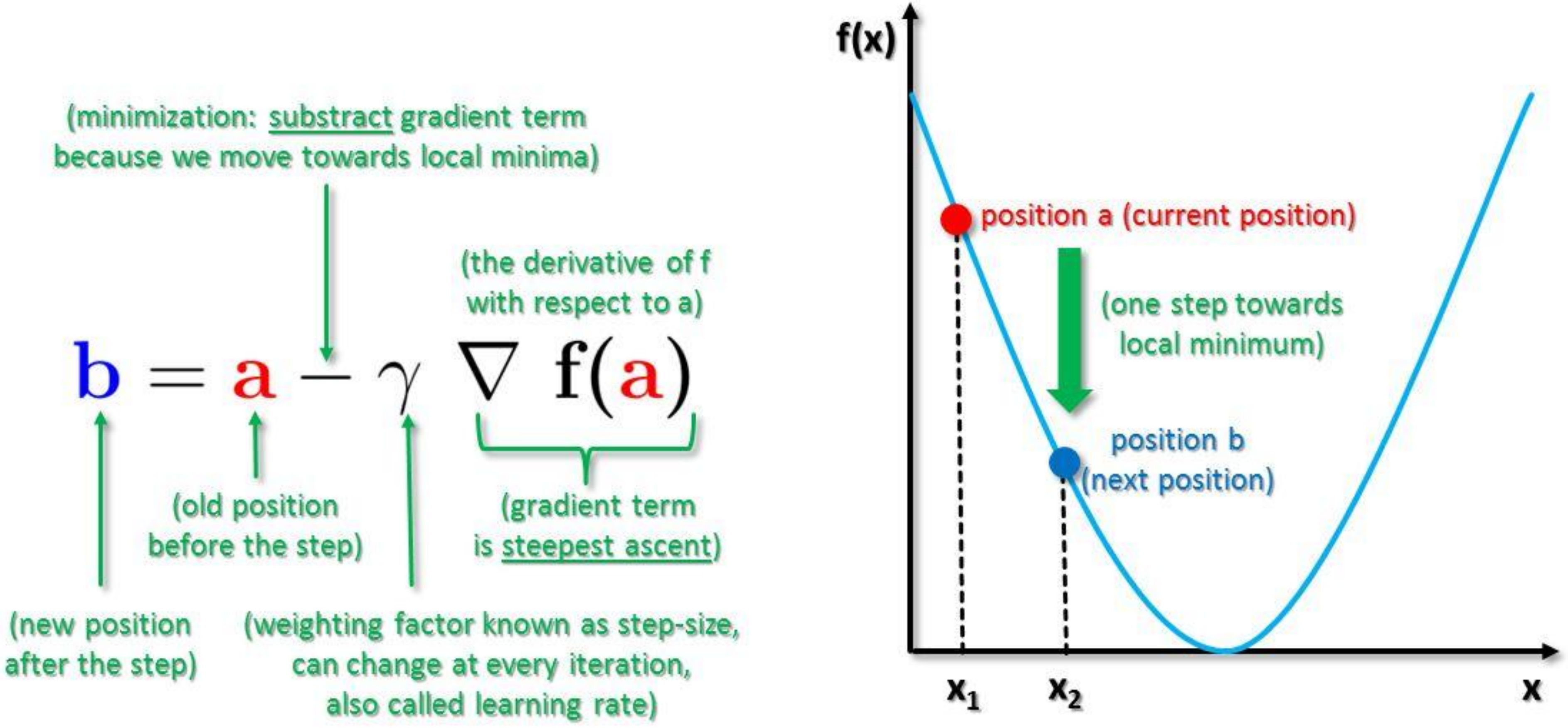




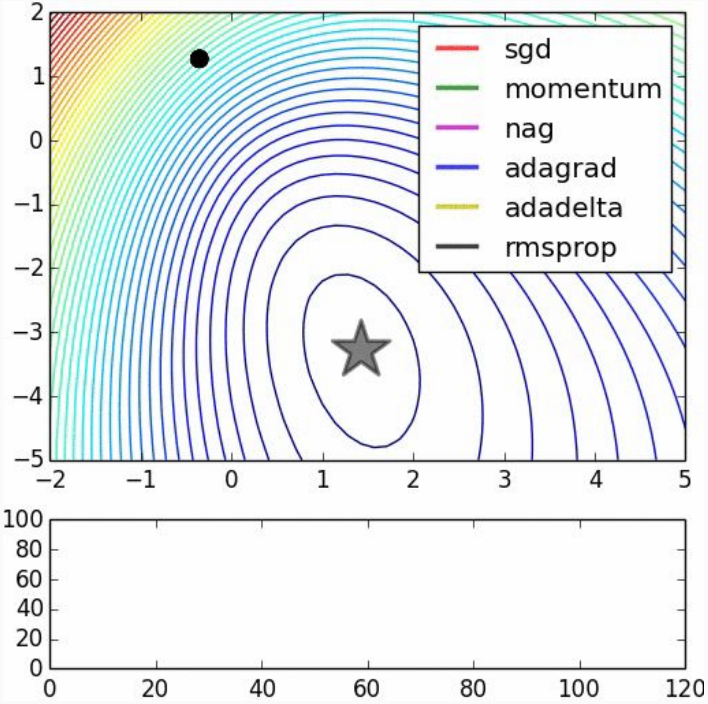
Loss = $\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$

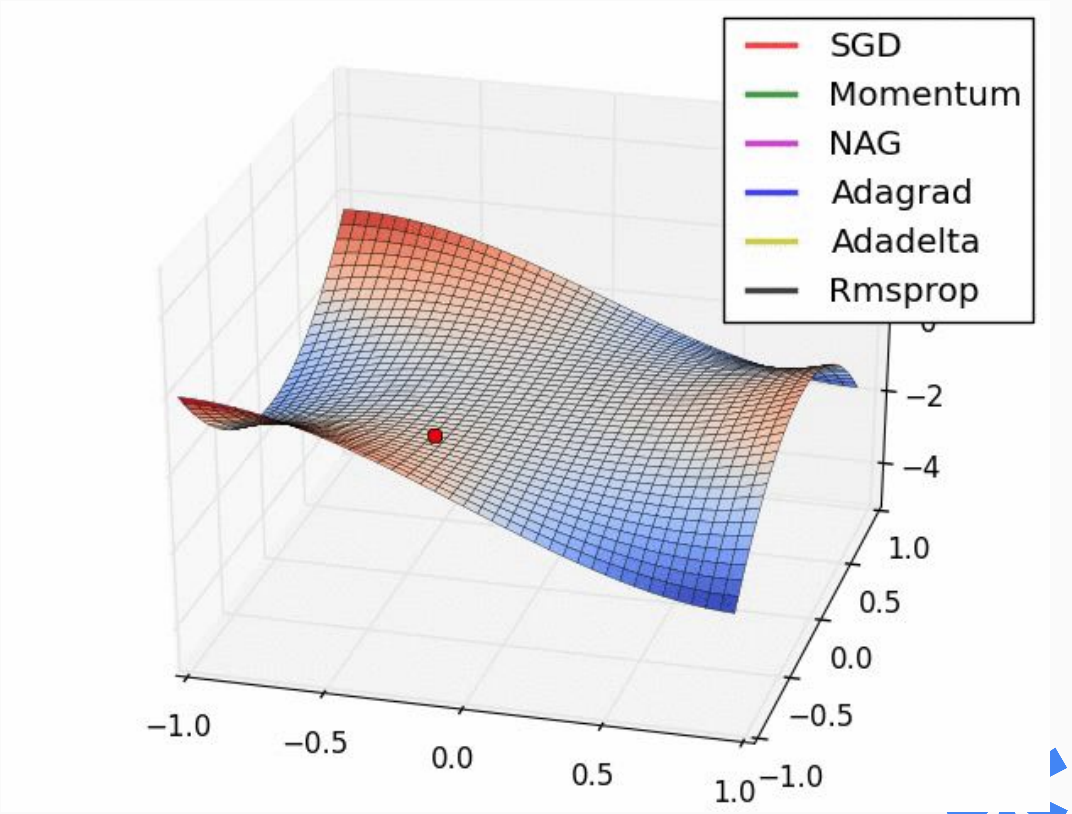
- Goal is find the parameters that result in the smallest loss across all samples
- This is just an optimization problem
- Can solve with GD, evolution, Simulated Annealing etc.

Quick Review: Gradient Descent



Gradient Descent: Visualization





Hang on that looks familiar….

Gradient Descent: $\mathbf{b} = \mathbf{a} - \gamma \nabla \mathbf{f}(\mathbf{a})$

Hang on that looks familiar….

Gradient Descent: $\mathbf{b} = \mathbf{a} - \gamma \underbrace{\nabla f(\mathbf{a})}$

Both just a gradient

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[\underbrace{\boxed{r_{t+1} + \gamma \max_a Q(s_{t+1}, a)} - \boxed{Q(s_t, a_t)}} \right]$$

What we want - What we have

Looks like gradient descent!

Note: gamma’s represent different things here

But wait...

Gradient Descent: $\mathbf{b} = \mathbf{a} - \gamma \underbrace{\nabla \mathbf{f}(\mathbf{a})}$ Here - ▼ f(a) is the gradient of the loss

Both just a gradient

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[\boxed{r_{t+1} + \gamma \max_a Q(s_{t+1}, a)} - \boxed{Q(s_t, a_t)} \right]$$

What we want - What we have

This is not the gradient of our loss

- It IS our loss

Looks like gradient descent! Surprise! No it's not!

$$Q(s_t, a_t) \leftarrow (1 - \alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

NQL loss function

Note: Details from here on are *implementation specific* - could use different losses and different network structure.

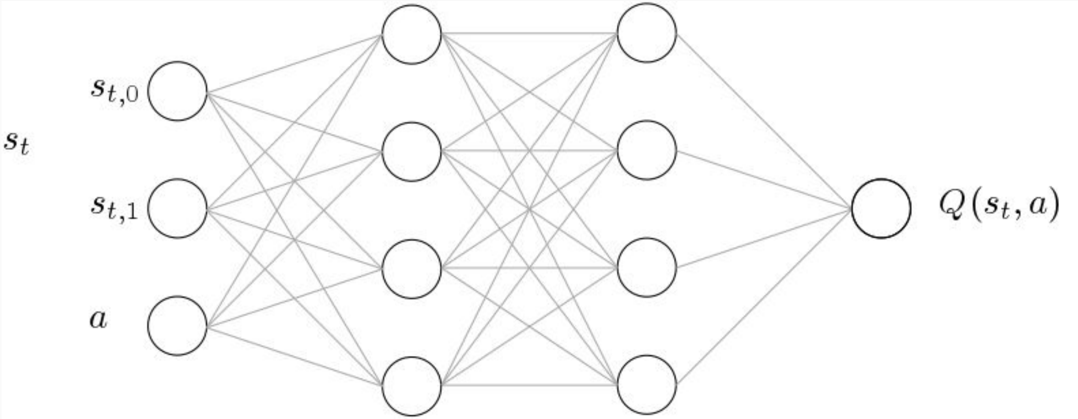
Loss = (target value - current value)²

$$\text{Loss} = \left[\underbrace{r_{t+1} + \gamma \max_a Q(s_{t+1}, a)} - \underbrace{Q(s_t, a_t)} \right]^2$$

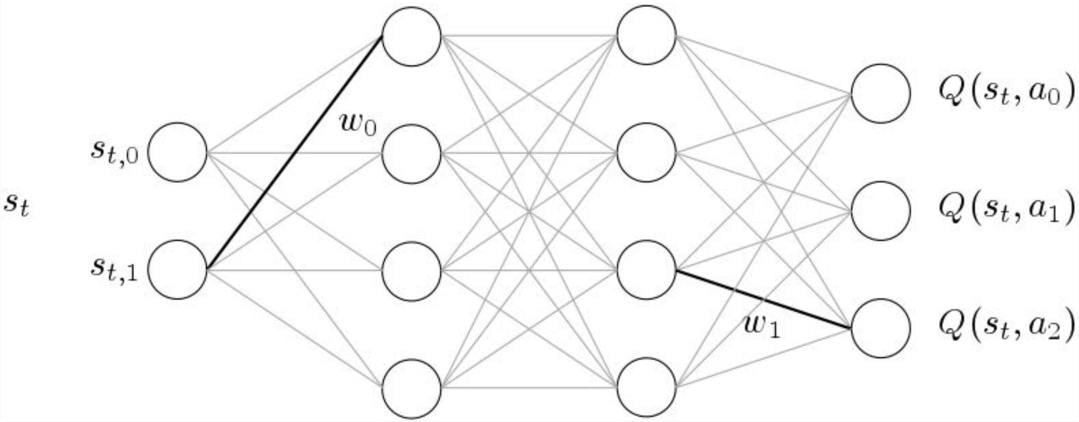
If we use and automatic differentiation all we have to do is define this loss and the optimization is done for us

Network Structure

Dumb way:



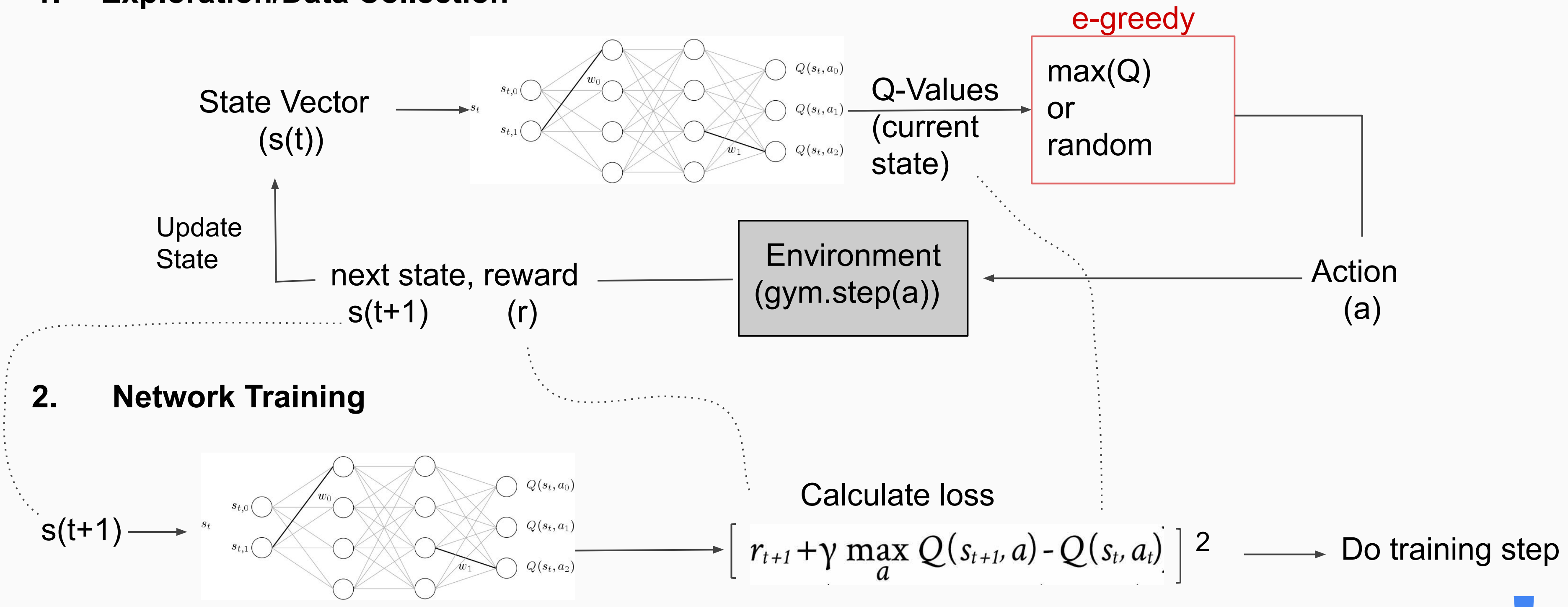
Smart way:



- Might as well calculate all Q-values at once since we need to do that max_a step.
- Fewer inference calls generally quicker
- Use ReLU activations.
- **Question:**
  Do we need activation functions on the final layer?

Overall NQL Algorithm

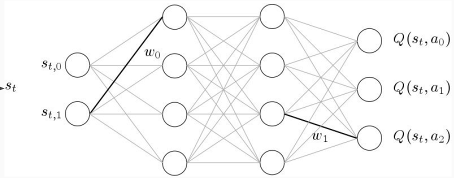
1. Exploration/Data Collection



Overall NQL Algorithm

1. Exploration/Data Collection

State Vector
(s(t))



Q-Values
(current
state)

e-greedy

max(Q)
or
random

Action
(a)

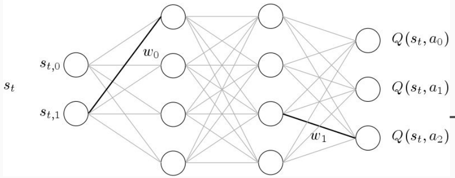
Environment
(gym.step(a))

next state, reward
s(t+1) (r)

Update
State

2. Network Training

s(t+1)



Calculate loss

$$\left[r_{t+1} + \gamma \max_{a} Q(s_{t+1}, a) - Q(s_t, a_t) \right]^2$$

Do training step

https://github.com/AlexanderJLong/RL-Notebooks

Overall

Algorithm 1: Online-Q

Initialize the value function q

while *not converged* **do**

  Get the initial state s

  while *s is not the terminal state* **do**

    Select an action a according to a ϵ-greedy policy derived from q

    Execute the action a, get the reward r and the next state s'

    Update the value function q with (s, a, r, s') following the Q-learning update rule

    $s = s'$

  end

end