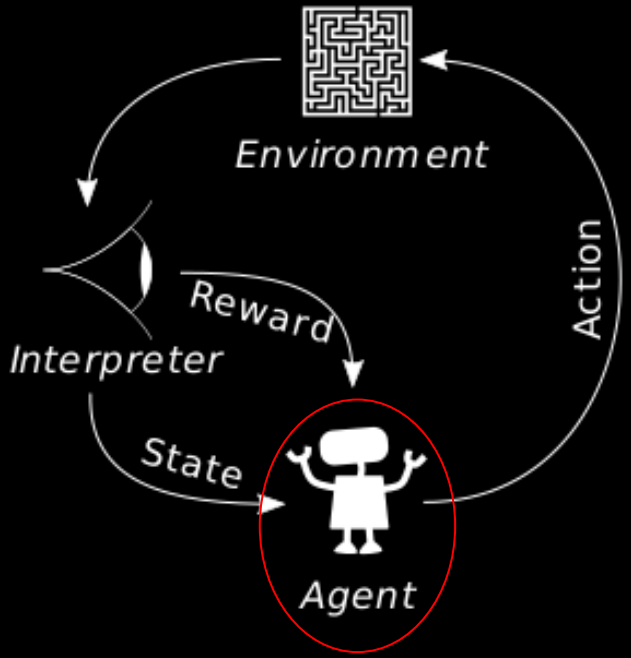
T-Rex Runner II

Seoul AI Meetup

Martin Kersner, 2017/05/27

Reinforcement learning



The goal of the agent is to interact with the environment by selecting actions in a way that maximises future rewards.

Reinforcement Learning with Deep Q Learning[1]

- Experience replay
  - $e_t = (s_t, a_t, r_t, s_{t+1})$
  - $D = e_1, ..., e_N$
- Each step of experience is potentially (**past experiences are selected randomly**) used in many weight updates, which allows for greater data efficiency.
- Learning directly from consecutive samples is inefficient, due to the strong correlations between the samples.

[1] https://arxiv.org/pdf/1312.5602v1.pdf

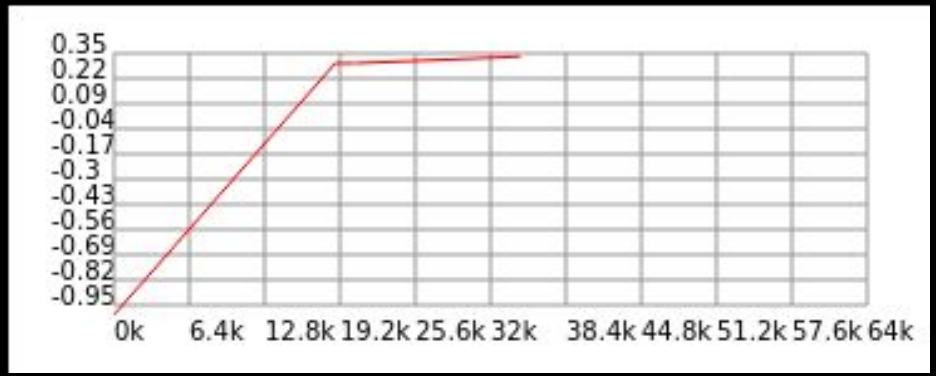
Applying Deep Q-learning I

- ConvNetJS[1]
- Game state
  - `[Speed, Obstacle_distance, Obstacle_height, Obstacle_width, Obstacle_type]`
- Reward
  - **Positive** when jumped over cactus
  - **Negative** when died
- `random_distribution = [0.1, 0.9]` (Jump, Idle)
- Always look at what is happening during training!
  - Average reward
  - Average loss
  - Weights

[1] http://cs.stanford.edu/people/karpathy/convnetjs/demo/rldemo.html

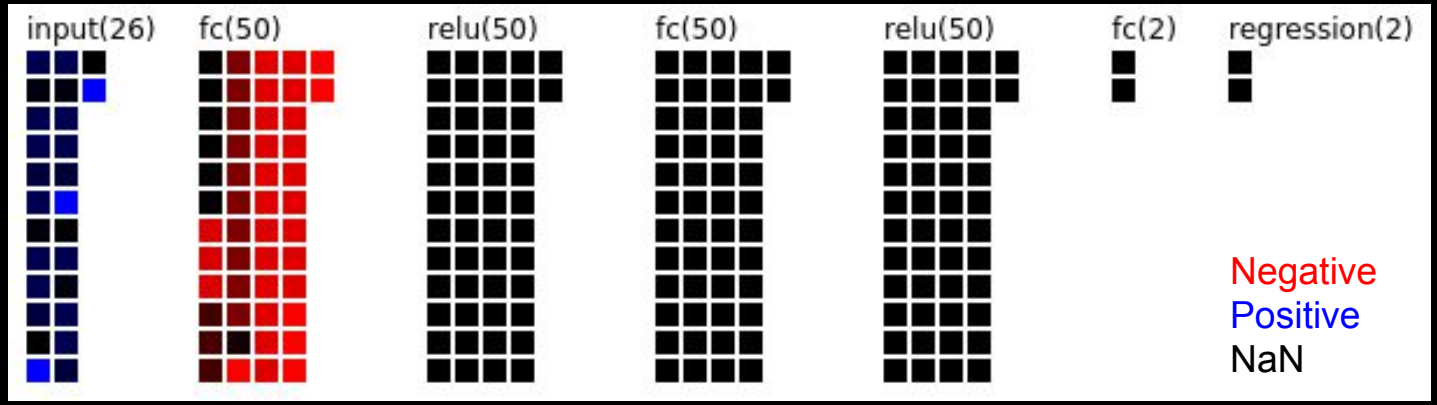
Applying Deep Q-learning II

- Displayed parameters
  - **experience replay size:** 17533
  - **exploration epsilon:** 0.926228426395939
  - **age:** 17533
  - **average Q-learning loss:** 0.029755900302159716
  - **smooth-ish reward:** 0.0054000000000000856



Average reward

Dead Network



fc fully convolutional layer
relu rectified linear unit

Even though there wouldn't be any NaN values, training of network with such initialization couldn't lead to well trained model (loss couldn't be optimized due to negative values in layer before RELU).

Fixing dead network I

- Smaller input layer.
  - Temporal window 5 → 3, 1
- Simplify network.
  - fc_relu(20) → fc_regression(3)
  - fc_relu(20) → fc_regression(2)
- Normalize input values[1].
- Decrease learning rate[2].
- Momentum update[2].

$$z = \frac{x - \mu}{\sigma}$$

$$\mu = \frac{1}{N}\sum_{i=1}^{N}(x_i)$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

$$v = \gamma v + \alpha \nabla_\theta J(\theta; x^{(i)}, y^{(i)})$$
$$\theta = \theta - v$$

[1] http://sebastianraschka.com/Articles/2014_about_feature_scaling.html
[2] http://cs231n.github.io/neural-networks-3/#sgd

Fixing dead network II

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

- Normalized input values processed by sigmoid function.
  - S(x) = (0, 1)
- Modified rewards.
  - Half successful jump 5 → **0.4**
  - Successful jump 10 → **0.6**
  - Death -10 → **0.0**
- Different values of `start_learn_threshold`.
- Tried to utilize javascript files which are used for demo at official ConvNetJS site.

Fixing dead network III

- `brain.value_net.layers[6].out_act.dw === [0, NaN]`
- `brain.experience[0].reward0 === undefined`
- `undefined`
  - `undefined + 3.14 == NaN`
  - `undefined * 42   == NaN`

```
2       var avcost = 0.0;
3       for(var k=0;k < this.tdtrainer.batch_size;k++) {
4         var re = convnetjs.randi(0, this.experience.length);
5         var e = this.experience[re];
6         var x = new convnetjs.Vol(1, 1, this.net_inputs);
7         x.w = e.state0;
8         var maxact = this.policy(e.state1);
9         var r = e.reward0 + this.gamma * maxact.value;
10        var ystruct = {dim: e.action0, val: r};
11        var loss = this.tdtrainer.train(x, ystruct);
12        avcost += loss.loss;
13      }
```

Plan

- Run multiple games at the same time and input game state to one brain ⇛ speed up training.
- Add reward for only surviving.
- Successfully train dino to get at least to situations where he has to duck.
- Train with 3 actions (idle, jump and **duck**).

https://github.com/martinkersner/dino-ai

Input network size

```
var network_size = num_inputs*temporal_window +
                   num_actions*temporal_window + num_inputs;
// num_inputs = 5
// temporal_window = 3
// num_actions = 2
// network_size = 5*3 + 2*3 + 5 = 26
```