Playing multiple games using deep reinforcement learning

Michal Chovanec, Ondrej Šuch

Abstract In this paper we presents deep Q network (DQN) playing multiple games with the same weights - six ATARI games. We trained network by randomly changing games after several interations, using deep reinforcement learning. For comparison, we also trained common DQN. Instead of using 8x8 or 4x4 kernels as in (cite Atari DQN and Doom paper) we used only 3x3 kernels, and much lower of feature maps in layers. This network is much more parameter efficient. As results we present training score progress and network feature maps usability for each game.

1 Introduction

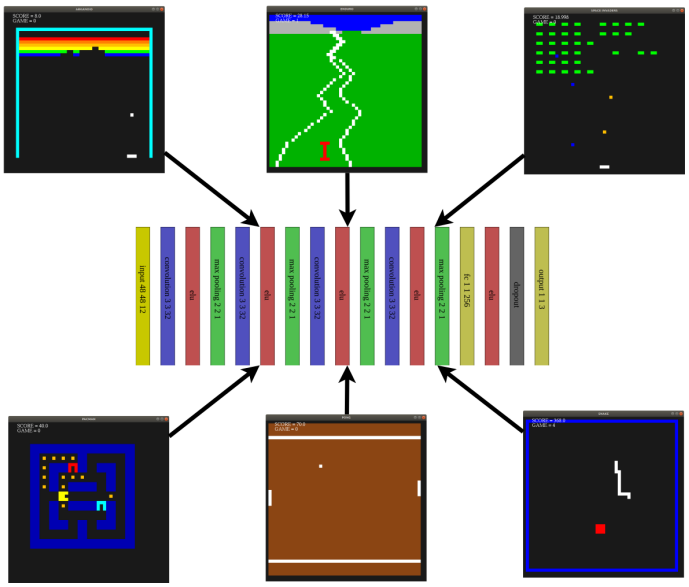


Figure 1: Multi DQN can play six games with the same weights

2 Experiment setup

Network input: Last 4 frames, width = 48, height = 48.

Network architecture: Following state of the art (cite VGG Net, ResNet), we used only 3x3 kernels, instead of (cite Atari DQN, Doom, with 8x8 kernels). As activation function the ELU is choosen, with $\alpha = 0.1$. After four convolution and pooling layers, the full connected layer with 256 neurons is used. On the last layer the linear activation is used. We also tried network without 256 FC layer - to reduce amount of parameters, but the training was much slower.

Proposed network architecture is :

$IN48\times48\times12-C3\times3\times32-P2\times2-C3\times3\times32-P2\times2-C3\times3\times32-P2\times2-C3\times3\times32-P2\times2-FC256-FC_{actions}$

Hyperparmaters of network were choosen as :

- learning rate $\eta = 0.001$
- L1 and L2 regularization $\lambda_1 = \lambda_2 = 0.0000001$
- dropout 0.02 (only after FC256 layer)
- minibatch size 32 for single DQN, 128 for multi DQN
- gradient clip $\langle -10, 10 \rangle$

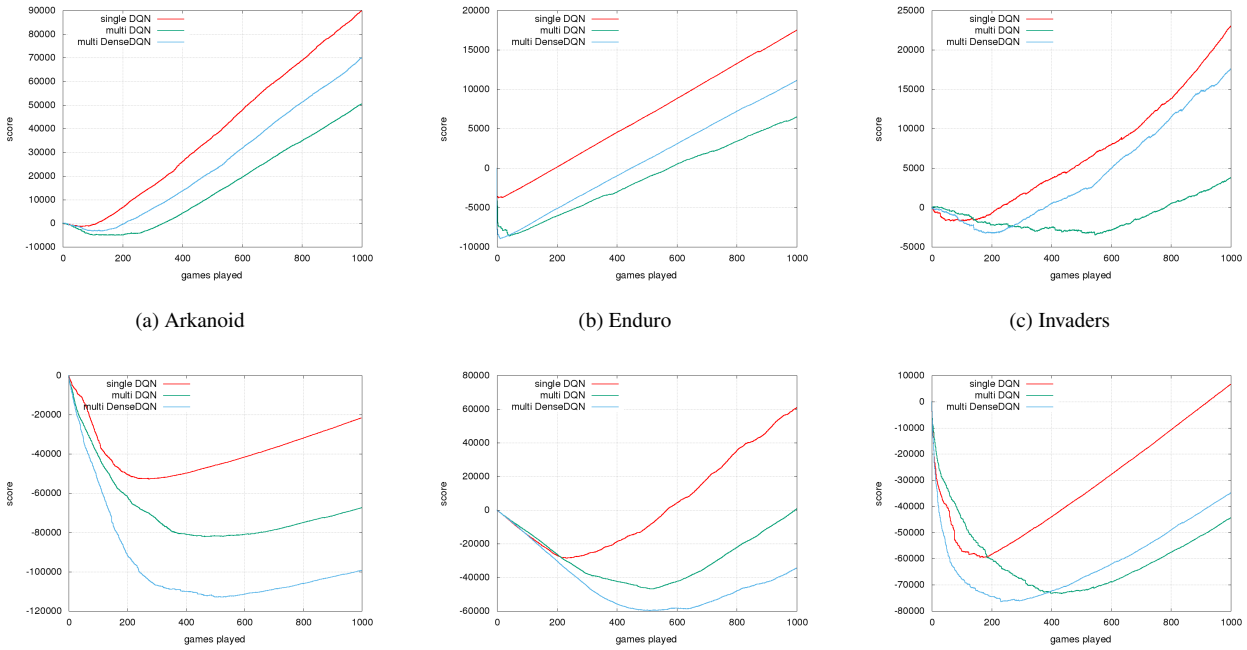
3 Results

First we presents training score progress. The games rewards has been chosen to be impossible achieve positive score only by random playing. On figures 2 we can see typical RL training progress - first is agent playing randomly, taking negative rewards. After few hundred games the agent finds good strategy and is able to achieve more and more positive rewards.

As second part, we focused on learned features visualisation. We processed the maximum kernel response, for all networks and all kernels. The inputs for maximum kernel response are presented on figure 3. Those images are obtained by maximizing kernel response using gradient backpropagation, starting with random input (cite). We visualised only convolutional layers.

To compare trained networks for learning similar features we computed correlations between networks (TODO more specify how). Considering networks features correlation (Table 1) we assume that multi_net_0 is learning different features. But we can see strongest correlation with most complicated games - Pacman and Snake, and the lowest correlation with the simplest game - Pong [1].

This results leads us to visualise activation heatmap - to understand what are different networks focusing on.



(a) Arkanoid (b) Enduro (c) Invaders

(d) Pacman (e) Pong (f) Snake

Figure 2: Training score result for first 1000 games

network	arkanoid	enduro	invaders	pacman	pong	snake	multi_net_0	summary
arkanoid	1.0	0.152	0.184	0.177	0.14	0.172	0.171	**1.9951**
enduro	0.18	1.0	0.241	0.252	0.103	0.169	0.186	**2.1308**
invaders	0.189	0.205	1.0	0.203	0.112	0.164	0.16	**2.0330**
pacman	0.194	0.232	0.243	1.0	0.14	0.17	0.227	**2.2048**
pong	0.118	0.076	0.086	0.116	1.0	0.092	0.114	**1.6013**
snake	0.223	0.239	0.208	0.222	0.154	1.0	0.268	**2.3137**
multi_net_0	0.166	0.171	0.155	0.207	0.127	0.202	1.0	**2.0283**

Table 1: Networks correlation

[1] most/less complicated game - in meaning of input complexity

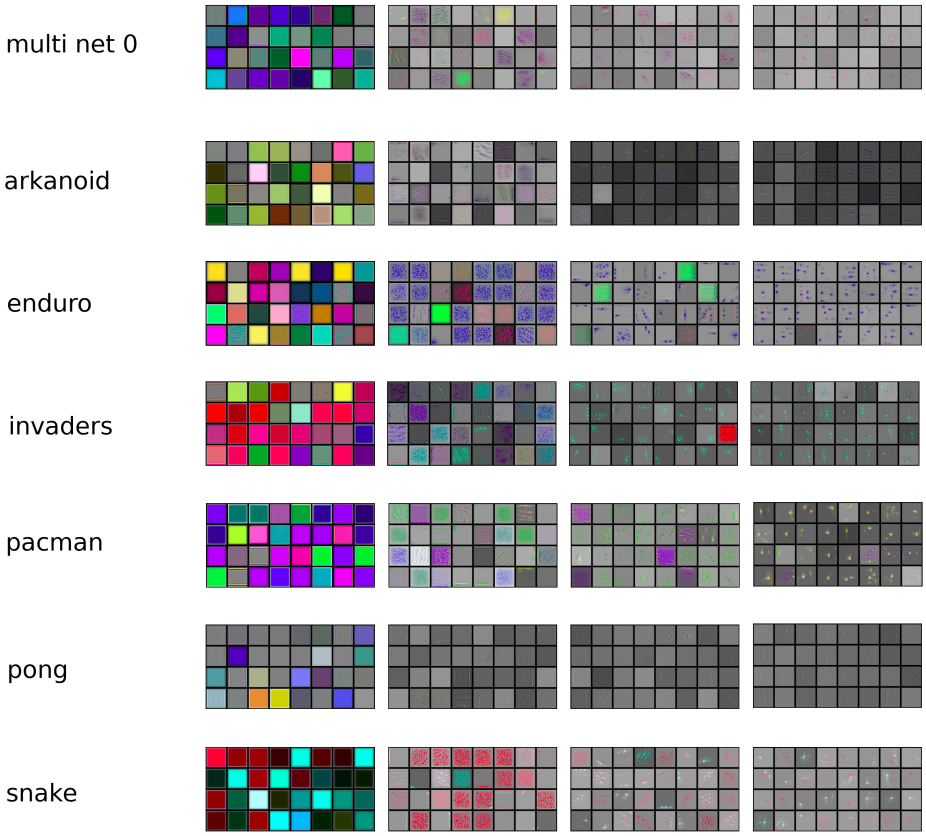


Figure 3: Kernel maximum response inputs visualisation for different networks