Playing multiple games using deep reinforcement learning, or Spatial Attention network can play (beat) simultaneously multiple games in reinforcement learning

Michal Chovanec, Katarína Jasenčáková, Katarína Bachratá, Hynek Bachratý

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 (one network per game). Our networks architectures are : basic convolutional neural network (CNN), DenseNet and Spatial Attention CNN. Spatial Attention Network was able to outperform both (basic and DenseNet, with much fewer parameters than DenseNet). Instead of using 8x8 or 4x4 kernels as in [1] and [2] we used only 3x3 kernels, and much lower of feature maps in layers. Presented networks are much more parameter and computation cost effective. As results we present training score progress, network feature kernels maps for each game and attention heatmaps.

1 Introduction

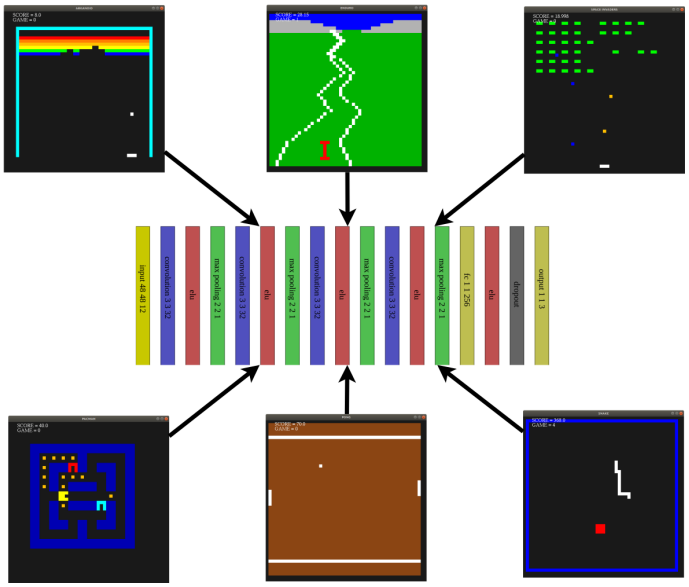


Figure 1: Multi DQN can play six games with the same weights

2 Experiment setup

We choosen 6 Atari games, in 48x48 grid : Arkanoid, Enduro, Invaders, Pacman, Pong and Snake. For relevant results, we first trained 6 basic CNN, each for one game. We choosen 1000 games for training and 100 games for testing the final score. Then we trained three different architectures (named multi_net_0 ... 2). Those networks have different architectures (basic CNN, DenseNet and Spatial Attention CNN). During training, the games were randomly changed (after 8192 iterations). Training is finished when there are played 1000 games from all games,. As input (game state) into neural network we used raw frame stacking. Our input are last four RGB frames, with size 48x48 pixels. This leads to input tensor dimensions of 48x48x12 or 48x48x3x4 if 4D tensors are used. Outputs are linear mapped Q values on last network layer. For action selecting we choosen ϵ-greedy strategy. Starting on $\epsilon = 1$ (total random strategy) exponentialy decreasing to value $\epsilon = 0.1$.

2.1 Network architecture

Following state of the art [3] and [4], we used only 3x3 kernels with stride 1, instead of 8x8 kernels as in [1]. As activation function the ELU is choosen, with $\alpha = 0.1$. After convolution and pooling layer we used one full connected with 256 units and second full connected corresponding to actions count in game. For multi_nets the amount of units on last layer depends on game with maximum possible actions.

Basic network architecture: multi_net_0 is basic deep Q network - the same as individual games playing networks. There are four convolutional layers with 32 filters, followed by ELU activation, 2x2 pooling and one 256 units full connected layer. The last layer with linear activation is directly mapping outputs to Q values. Number of units on this layer is equal to actions count (respectively maximum required actions - in our case it is 5). We also tried network without 256 FC layer - to reduce amount of parameters, but the training was much slower [1].

DenseNet network architecture: Next tested architecture is DenseNet [5]. We used four dense blocks. Each dense block consist of four 3x3x32 convolutional layers, one 1x1x32 layer and 2x2 pooling layer. The main idea is to use state of the art network, which have much bigger capacity and probaly can outperform basic DQN. The main problem can be a lot of parameters of this network. Thanks dense connection, the much deeper architecture can be used, and much more complex features can be extracted.

Spatial Attention CNN network architecture: The last architecture we tested is Spatial Attention CNN, inspired by [6]. The main idea of attention in CNN is to "amplify" some selected featured and present them to next layer. Our presented attention module is illustrated on figure 2. Input tensor with dimensions $width \times height \times depth$ is processed in convolution block (with kernel size $3 \times 3 \times depth$). Instead of input features-depth pooling - which is useful in classification attention networks, we are using full features inputs into convolution module (layer). After convolution the sigmoid non-linear activation is applied on feature maps. Next block is gating unit, which is using point-wise (Hadamard) product. The last step is addition block, which adds original input and attention features - this can be intepreted as ResNet idea. Full network architectures can be seen in table 1. The formal definition can be written in following term

$$y = x + x \odot f(x; w, b) \tag{1}$$

where
x is input tensor
y is output tensor
w, b are kernel weights and bias respectively
$f(x; w)$ is convolution between x and w followed by sigmoid nonliearity.

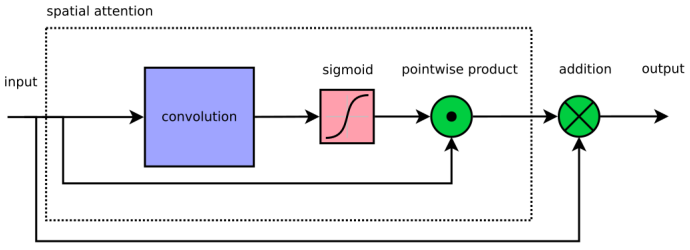


Figure 2: Designed spatial attention module

We choosen the some hyperparameters for all networks. Only difference is minibatch size. For single game playing net we choosen size 32, for multi-nets 128 - to chieve better training stability. Summary hyperparameters setup are :

- learning rate $\eta = 0.001$
- L1 and L2 regularization $\lambda_1 = \lambda_2 = 0.0000001$
- dropout 0.02 (only after FC256 layer)

[1] We are not presenting results, but this require more research - how to reduce number of parameters in DQN and make training faster

- minibatch size 32 for single DQN, 128 for multi DQN
- gradient clip $\langle -10, 10 \rangle$
- Q-learning discount factor $\gamma = 0.99$

layer	multi_net_0	multi_net_1	multi_net_2
0	C3x3x32	4 x dense C3x3x32	C3x3x32
1	P2x2	C1x1x32	attention 3x3
2	C3x3x32	P2x2	P2x2
3	P2x2	4 x dense C3x3x32	C3x3x32
4	C3x3x32	C1x1x32	attention 3x3
5	P2x2	P2x2	P2x2
6	C3x3x32	4 x dense C3x3x32	C3x3x32
7	P2x2	C1x1x32	attention 3x3
8	FC 256	P2x2	P2x2
9	FC actions	4 x dense C3x3x32	C3x3x32
10		C1x1x32	attention 3x3
11		P2x2	P2x2
12		FC 256	FC 256
13		FC actions	FC actions

Table 1: Tested networks architectures

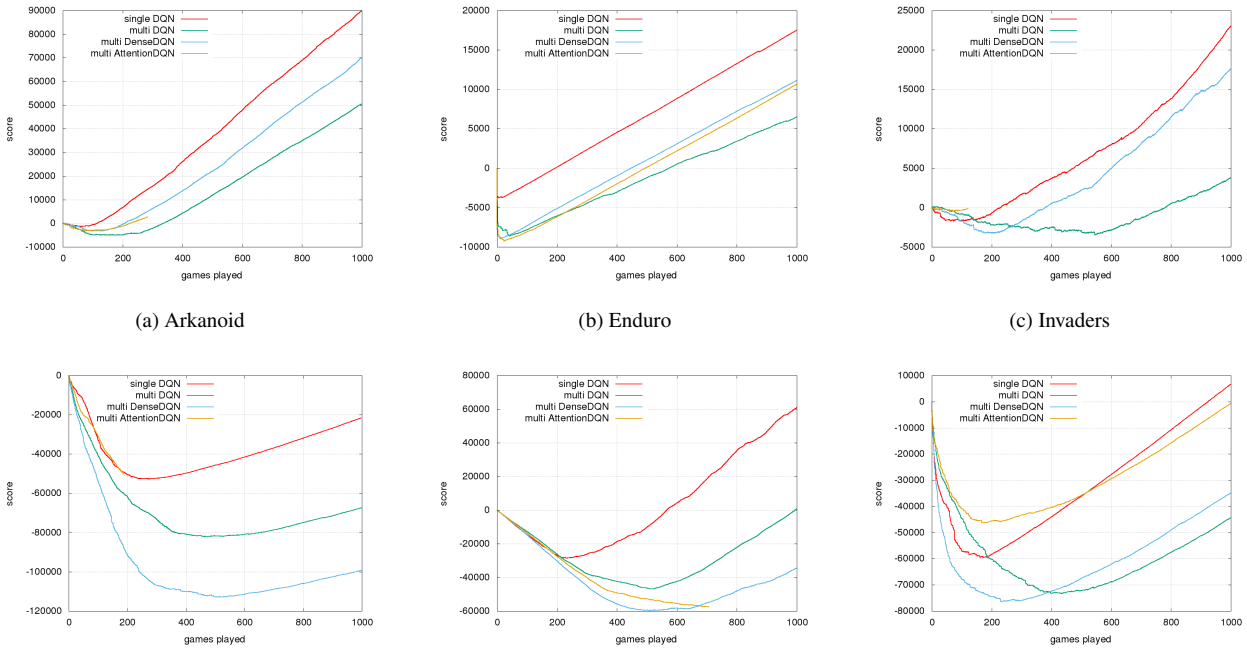
3 Results

First we presents training score progress. The games rewards has been chosen to be impossible achieve positive score only by random playing. On figures 3 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 4. 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 2) 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 [2].

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 3: 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 2: Networks correlation

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

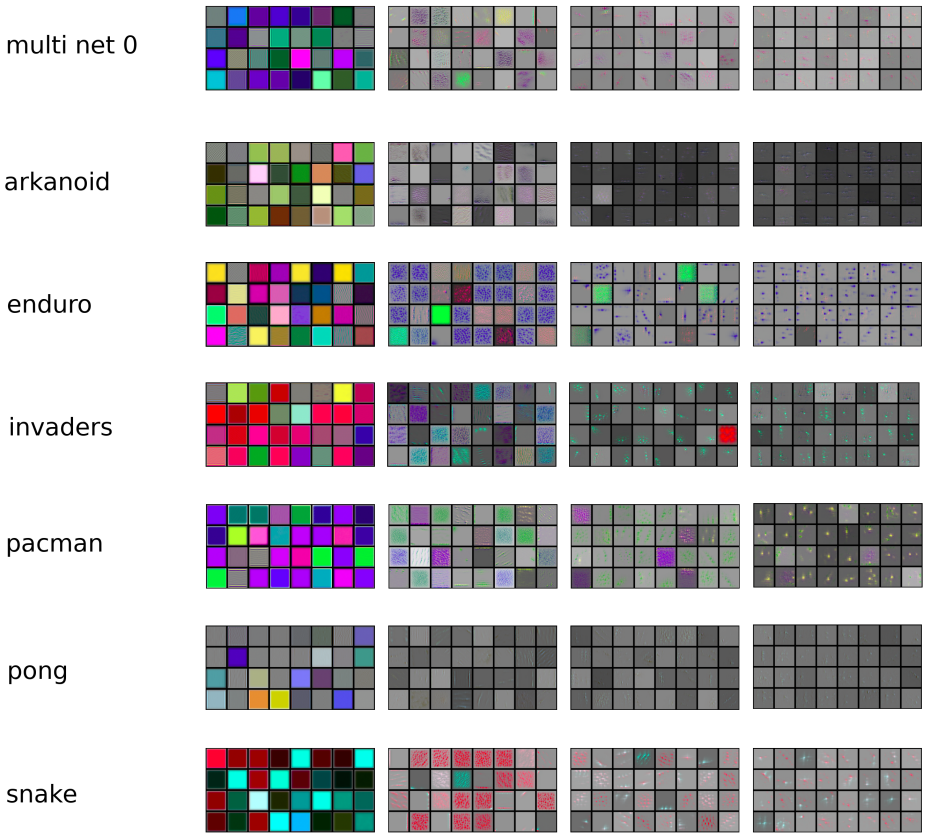


Figure 4: Kernel maximum response inputs visualisation for different networks

References

[1] Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis AntonoglouDaan Wierstra Martin Riedmiller. Playing Atari with Deep Reinforcement Learning 2013, https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf

[2] Guillaume Lample, Devendra Singh Chaplot. Playing FPS Games with Deep Reinforcement Learning, 2017, https://arxiv.org/abs/1609.05521

[3] Very Deep Convolutional Networks for Large-Scale Image Recognition. Karen Simonyan, Andrew Zisserman, 2015, https://arxiv.org/abs/1409.1556

[4] Deep Residual Learning for Image Recognition. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, 2015, https://arxiv.org/abs/1512.03385

[5] Densely Connected Convolutional Networks Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger, 2017, https://arxiv.org/abs/1608.06993

[6] Woo, Sanghyun and Park, Jongchan and Lee, Joon-Young and So Kweon, In. CBAM: Convolutional Block Attention Module 2018, https://eccv2018.org/openaccess/content_ECCV_2018/papers/Sanghyun_Woo_Convolutional_Block_Attention_ECCV_2018_paper.pdf