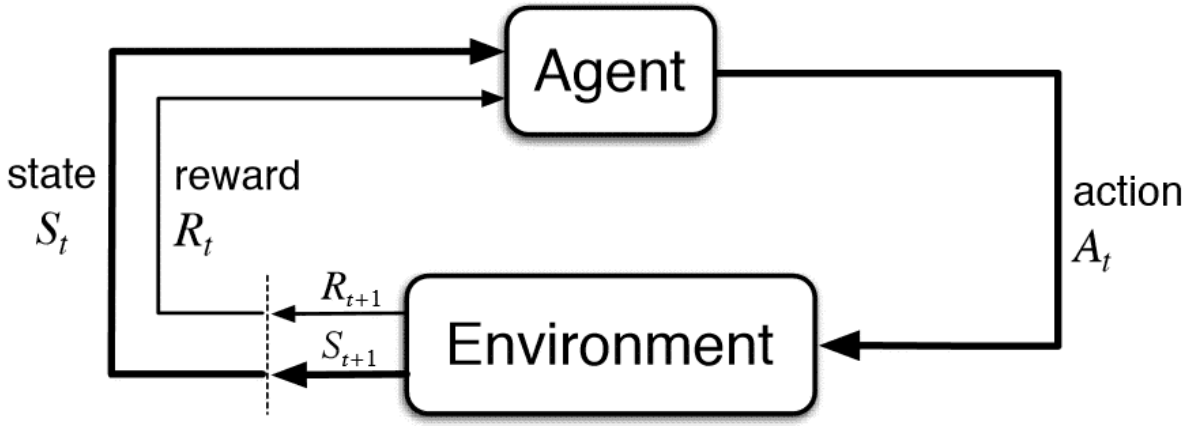
Deepmac

Utilizing deep Q-learning to create agent playing Pac-Man

- Jakub Skalski
- Paweł Szmergała

What is Q-learning?



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

New Q-value estimation

Former Q-value estimation

Learning Rate

Immediate Reward

Discounted Estimate optimal Q-value of next state

Former Q-value estimation

TD Target

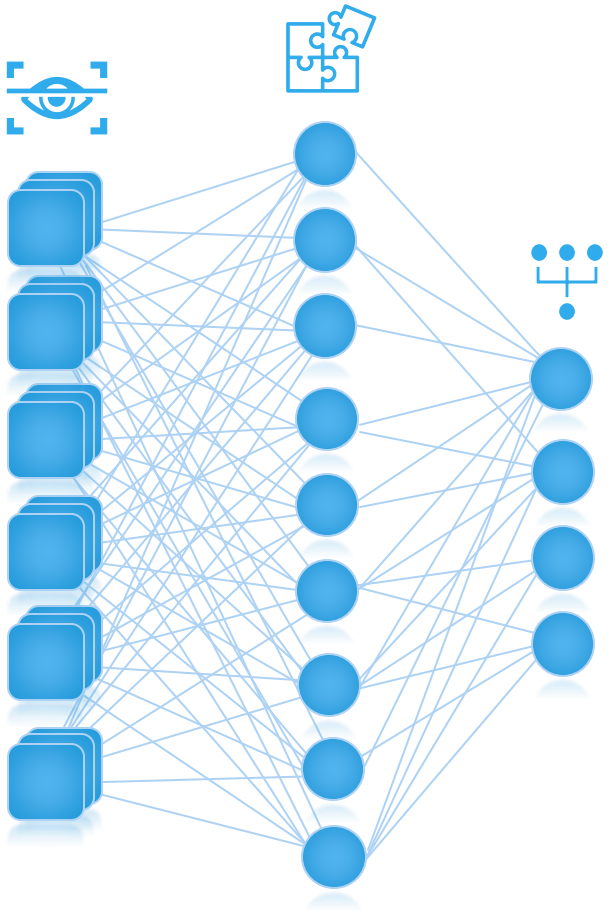
TD Error

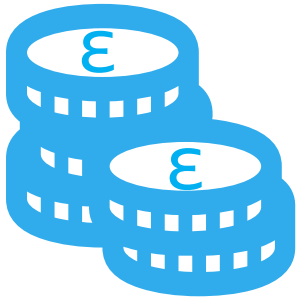
Models and data

- Data is generated through gameplay and stored in memory in form of [state, action, reward, next state] transitions.
- Model consists of two neural networks both with the same architecture (double Q-Learning).
- Training agent takes action according to the ε-greedy strategy.
- Input is read from an image and processed before being fed to the convolutional neural network.

Architecture

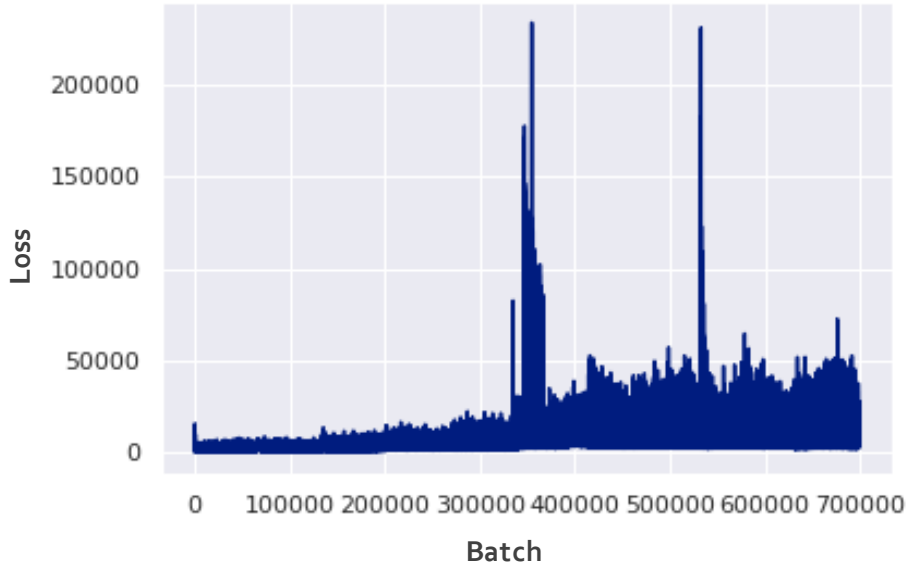
Transition batch
(states, actions, rewards, new states)

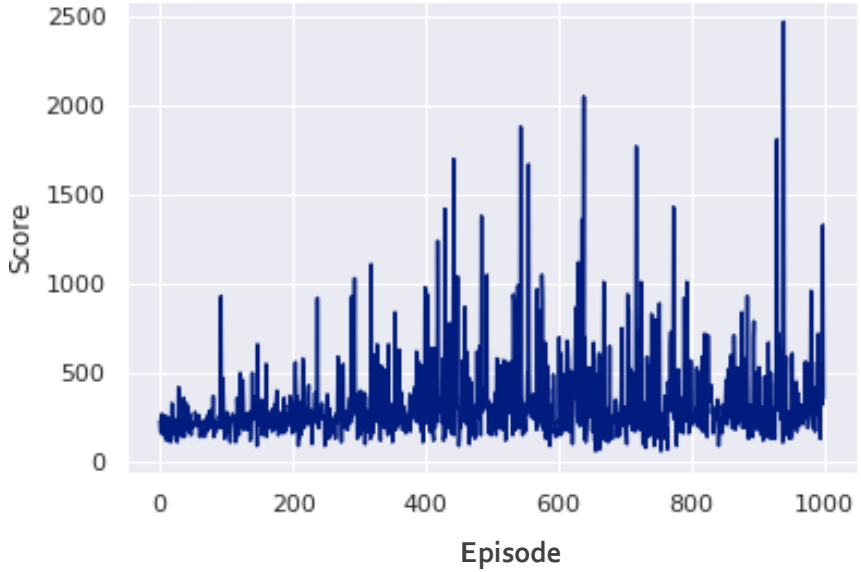


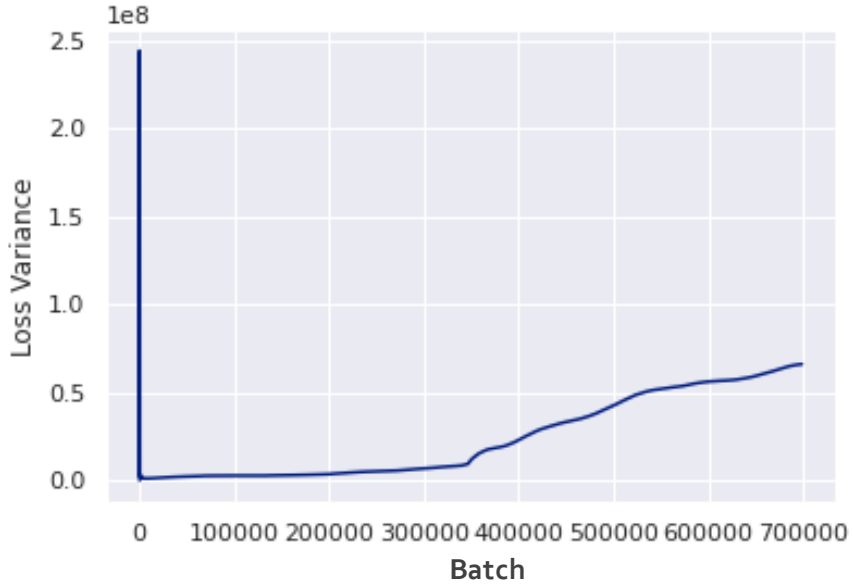


Epsilon greedy
(choose random action with epsilon probability instead)

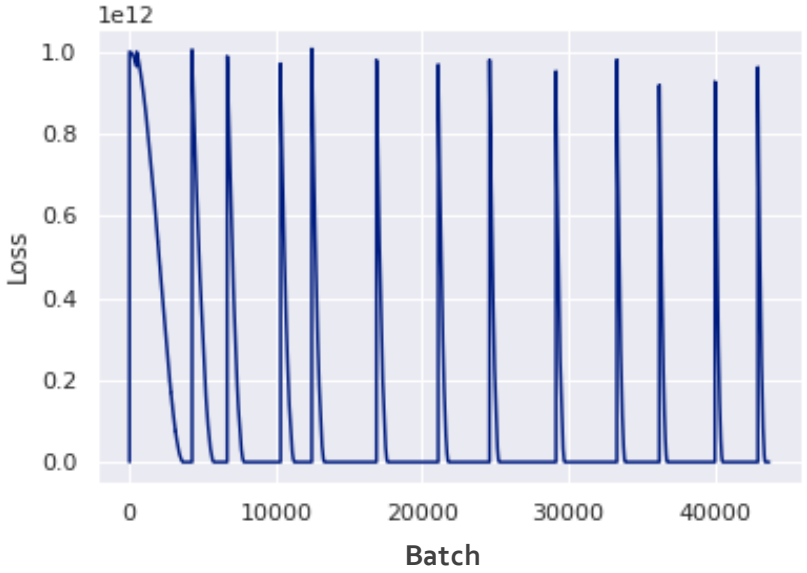
Successful training process

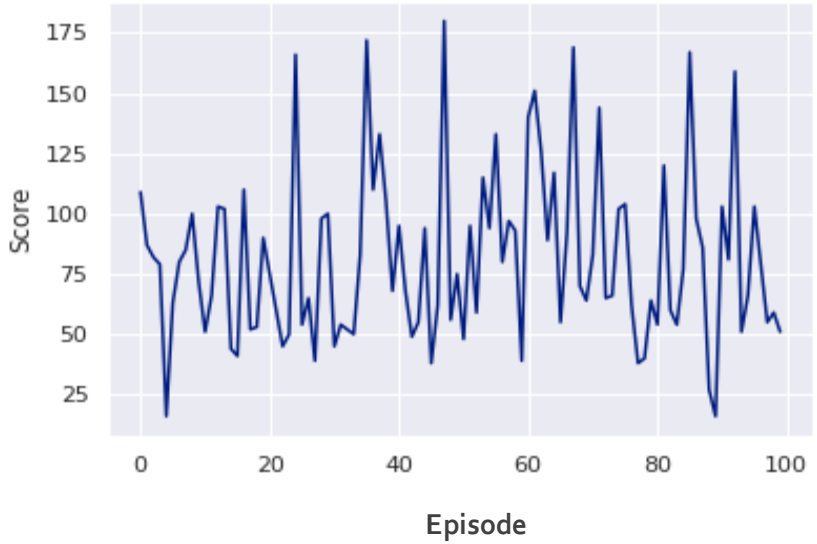


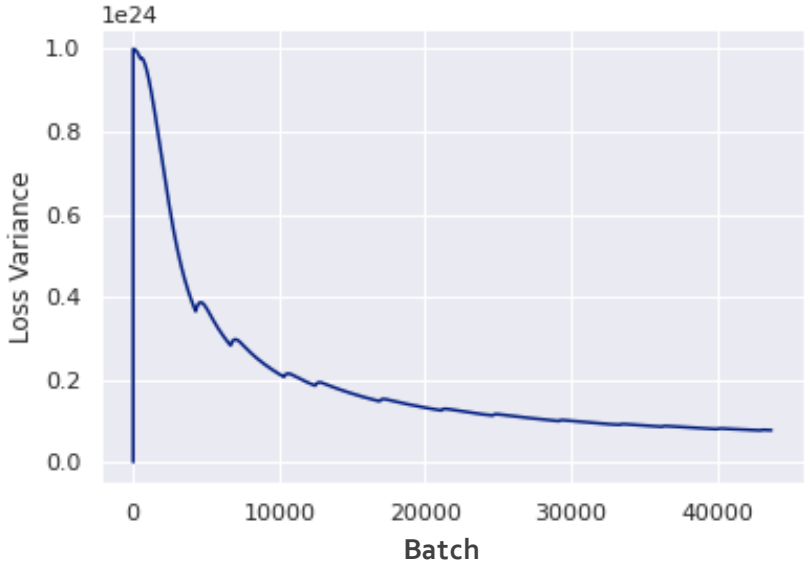




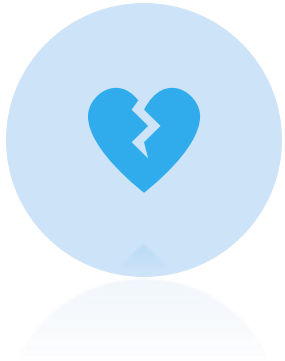
Unsuccessful training process



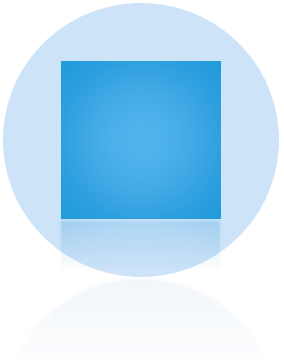




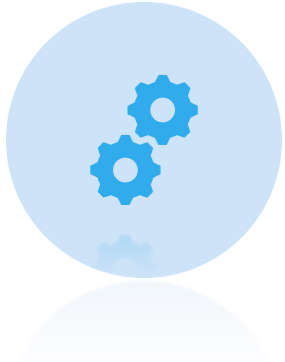
Conclusions



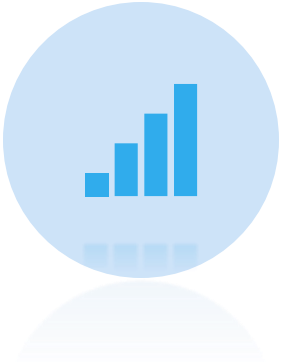
Deep reinforcement learning is hard :)



Input format matters way more than expected



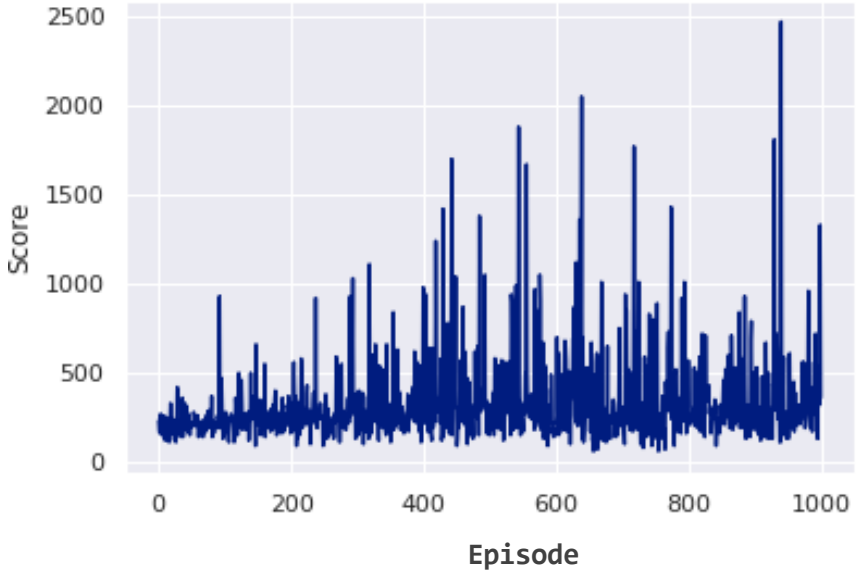
Lots of hyperparameters to manage

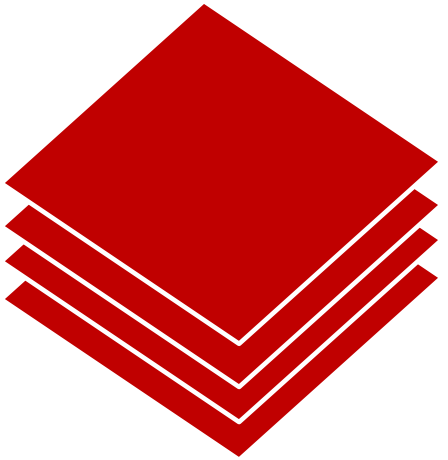


Ambiguos diagnostics

Results

- Agent shows signs of learning.
- Results are way better than those of a random playing agent.
- We could not make the agent learn from non-image input format of the game which was disappointing.
- Agent looks promising yet training is very computationally expensive.





Semi one-hot encoded game state

Sauce code