AI: Project 2

Abgabe 09.09.2022, 23:55 Uhr

Marcel Bauer 5570300, Roman Svetlitckii 5124010

Explaining (also included in the notebook as commentary):

Some used links:
https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939
https://machinelearningknowledge.ai/pytorch-conv2d-explained-with-examples/
https://datascience.stackexchange.com/questions/64278/what-is-a-channel-in-a-cnn
https://arxiv.org/pdf/1603.07285v2.pdf
https://pytorch.org/tutorials/beginner/basics/optimization_tutorial.html

A DQN can be built with CNN, which will consist in our case of 4 layers:
1) Convolutional(Conv2d) x2
2) Pooling(MaxPool2d)
3) Fully connected(Linear)

CNN1:
in_channel = 1, because the input frames after preprocessing are stored in uint8 format.
kernel size is 8 because it is enough to capture a single grid entry(size ~5 pixel) and a little bit of area around it. Choosing a value larger than 5 would also decrease the size of output feature map.
stride is chosen to be 4 because it allows the 50% overlapping of kernels when convolution is conducted, which could theoretically improve feature detection.

pool:
basically exist to reduce the size of input and the amount of calculations.

CNN2:
parameters are adjusted to fit the output from previous layer.

The ReLU activation function is used because it is simple and pretty common.

For the loss we used the mean squared error. According to the project description: "we need to use a gradient descent based optimization method", so we used the Adam, which is a stochastic gradient descent method. The learning rate is 0.001 because it is the most common value in different online sources.

'states' has the following shape: 64x1x4x1x105, so we either have to transform it to the batch(e.g. extracting single frames into a list) or apply policy step-wise. Step-wise method was chosen because it seems easier and no significant improvement was seen with the batch method.

Epsilon(second parameter in 'np.random.binomial' below) will be adjusted in each step.
A CNN is is not able to predict actions in the beginning, so at first actions will be chosen mostly randomly. After some specified 'eps_amplifier' value will be equal to 'steps' the epsilon becomes constant = self.eps, so mostly the network will predict actions and only seldom will they be chosen randomly.

Bernoulli distribution is used to select either exploitation(p = self.eps) or exploration(p = 1-self.eps)
A tensor is 'compressed' to 1 dimension and after that the greatest index is chosen, that will be a predicted action.

Unfortunately the training could not be conducted properly, because the GPU was not available on own devices and the Collab has thrown some errors(e.g. Environment MsPacmanDeterministic doesn't exist), which could not be solved. So there is no information about convergence.