Neuroevolution of Self-Interpretable Agents

September 13, 2022

Gianmarco Ursini

Abstract

The neural network's attention mechanism enhances some parts of the input data while diminishing other parts. In Reinforcement Learning, in particular, agents capable of seeing only a portion of the pixel inputs from the environment not only can solve the tasks with simpler models, but also achieve better generalization. Further investigations over this idea, as also the implementation of a self-attention agent, will be carried out during the present project. All the code is publicly accessible GitHub.

1. Introduction

Once dealing with perception tasks, selective attention typically results in inattentional blindness which enforce the focus over the most important and relevant (w.r.t. the completion of the assigned task) features of the environment we are in, relaxing, on the other hand, the attention devoted to all the irrelevant details. In order to deal with a classical reiforcement learning task (i.e. the OpenAi Gym game `CarRacing-v2`), and inspired by (Tang et al., 2020), an artificial agent constrained to perceive the gaming environment through a self-attention bottleneck will be developed.

2. Methods

2.1. Environment

`CarRacing-v2` (in its action-space discrete version) is the selected OpenAI Gym game used to develop the project. In this environment the t_{th} observation $O_t \in \mathbb{R}^{96\times96\times3}$, where 96$\times$ 96 $\times$ 3 recalls the observation image size. Its action space size (in the discrete version of the game) allows 5 different actions (i.e. do nothing, steer left, steer right, gas, brake). The assigned reward equals -0.1 for every game frame and $\frac{+1000}{N}$ for every track tile visited, where N recalls the total number of tiles throughout the entirety of the track. The run is considered successful if the agent achieves a total score $\geq$ 900.

2.2. Models

The adopted model has been fully inspired to (Tang et al., 2020), and it's visualized in figure 1. The model is fed with RGB observation images: unlike in (Tang et al., 2020), where they used those raw images, an earlier preprocessing step is executed. It foresee the normalization of pixel values in the range $[0, 1]$, the observation's grayscaling as also the stacking of 3 time consecutive grayscale observations. Better performances were achieved w.r.t. the ones obtained via the setup in (Tang et al., 2020). A patch-extractor module then transforms the pre-processed input into overlapping deep (i.e. containing all the 3 time-channels) 529 patches extracted with a window of size 8 $\times$ 8 and a stride of size 4. The patches are subsequently flattened, and attention scores are computed across their linear projections obtained via 2 distinct learnable linear maps $W_Q, W_K : \mathbb{R}^{147} \mapsto \mathbb{R}^4$. The K (K=10 in our experiments) patches having the greatest attention scores (computed via a row-wise sum) are then selected and fed to the feature extractor. It is worth to notice that the adoption of such a patch-extractor-and-selection mechanism results into a fully convolutional model (end-to-end) capable to process inputs of arbitrary dimensions whithout the need to scale in size. On the other hand, it also allows to build much smaller agents, since they will just have to process a small (and most relevant) part of the input. Subsequently, 3 different top-patch feature extraction mechanism have been implemented: the reference one foresee the extraction of the top-patches location (i.e. their X and Y coordinates in the observation reference system) and their subsequent flattening (e.g. if K=10 the output of the feature-extractor will be of size 1 $\times$ 20). In the second mechanism, the aim is the extraction of features from the top-patches via the convolution of learnable filters. The features are then fed to an LSTM layer having 16 hidden neurons, headed by a fully connected classifier mapping LSTM output to the action space. Since this second approach suffers from the constrain over the size of the model imposed by the training algorithm (i.e. CMA-ES is efficient only if the size of the

Email: Gianmarco Ursini <ursini.1635956@studenti.uniroma1.it>.

Deep Learning and Applied AI 2022, Sapienza University of Rome, 2nd semester a.y. 2021/2022.

parameter space is restrained), a third mechanism is implemented. If foresee the training of a deterministic autoencoder over an L_2 loss with a naive gradient descent. The AE will push the top-scored patches towards a bottleneck, trying to reconstruct them from their latent representations. Once trained, the AE will be freezed and extract features (i.e. the latent codes) from the top patches, passing them straight to a linear classifier mapping each patch feature to the action space. Lastly, the final action to perform is chosen via majority voting across the patches batch. Since it is key to show to the AE the greatest amount and variability of top patches possible, its training will be executed in parallel to 30 episodes run of our pre-trained model equipped with the top-patches location extractor.

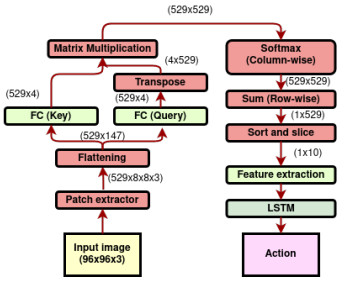

Figure 1. Visualization of the Self-attention agent.

2.3. Training algorithms

Since few of the data manipulation executed (i.e. the patch sorting and slicing) are non-gradient friendly, Evolutionary and gradient-free algorithms had to be exploited to train the model: in particular, the CMA-ES (Hanses) library was ready to be used for our training purposes. As a negative value function, we naively opted for the negative value discounted cumulative expected reward $R(\tau)$ that (given a policy τ, a discount rate γ and the reward r_i obtained at the i_{th} step) is defined as $R(\tau) = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$. In the 3 setups, CMA-ES has been run for respectively 600, 100 and 150 generations.

3. Results

The best values of discounted cumulative expected reward are reported in table 1 for each of the attempted feature-extraction mechanism.

4. Discussion and conclusion

As clear from table 1, all of the attempted approaches failed to reach a satisfactory score value. Since even the model equipped with the top-patches location feature extractor (fully inspired to (Tang et al., 2020)) fails in the task, we reasonably face the evidence that a better and more efficient choice of the value function has been reasonably made by the authors of the original paper. The convolutional feature extractor completely fails to extract meaningful features: the main problem faced was the obligation to keep the model size small. Indeed, even if convolutional layers are probably among the most efficient ways to extract feature from images thanks to their equivariance and multi-scale-activation properties, having adopted a convolutional feature-extractor with just $\approx$ 3000 parameters didn't allow the expression of such a power [1]. Indeed, if on one hand the adoption of the attention scoring and slicing mechanism allows to build small agents (w.r.t. the agents processing the whole input) capable to deal with complicated tasks, it also obliges the choice of a gradient-free optimization method, often unstable if the size of the model is too large. Because of that, the development of small models under the adoption of non-differentiable data manipulations looks like a constrained (and not a design) choice. The AE, on the other hand, allows to see that even if an efficient convolutional back-end is leveraged, the model still fails in the task. It highlights that, despite their simplicity, features expressing the location of the top-patches seems to be the key to solve `CarRacing`, and such an information can not be preserved by any convolutional extractor due to their innate translational equivariance.

Features	# Params	Fitness value
Top-patches location	4'061	712.67 $\pm$192.44
Convolutional	6'243	-52.06 $\pm$33.37
Autoencoder	2'829	71,72 $\pm$79.42

Table 1. Fitness values achieved by the models (computed across 30 different episodes) reported with their sizes.

5. Further works

The implementation of a stable large scale optimization method could overcome issues induced by the constraint imposed over the size of the model, possibly improving the results of the present work. The naive execution of more evolutionary generations for the AE architecture should also be tested to assess if a delayed convergence finally occurs. If the top-patches are considered as nodes of an undirected graph (in which the car occupies the central node) and the attention scores as their feature, the implementation of a Graph AE could also provide a powerful way to extract features from the top-patches preserving the information about their locations.

[1] Several kernel crashes have been observed increasing the model size over the value of 7000. Recall the complexity of computing the Variance-Covariance matrix foreseen by the CMA-ES.

References

Hanses, N. Covariance matrix adaptation evolution strategy. https://github.com/CMA-ES/pycma.

Tang, Y., Nguyen, D., and Ha, D. Neuroevolution of self-interpretable agents. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2020. URL https://attentionagent.github.io.