Neuroevolution of Self-Interpretable Agents

Yujin Tang
Google Brain, Tokyo
yujintang@google.com

Duong Nguyen
Google Japan
duongnt@google.com

David Ha
Google Brain, Tokyo
hadavid@google.com

ABSTRACT

Inattentional blindness is the psychological phenomenon that causes one to miss things in plain sight. It is a consequence of the selective attention in perception that lets us remain focused on important parts of our world without distraction from irrelevant details. Motivated by selective attention, we study the properties of artificial agents that perceive the world through the lens of a *self-attention* bottleneck. By constraining access to only a small fraction of the visual input, we show that their policies are directly interpretable in pixel space. We find neuroevolution ideal for training self-attention architectures for vision-based reinforcement learning (RL) tasks, allowing us to incorporate modules that can include discrete, non-differentiable operations which are useful for our agent. We argue that self-attention has similar properties as *indirect encoding*, in the sense that large implicit weight matrices are generated from a small number of key-query parameters, thus enabling our agent to solve challenging vision based tasks with at least 1000x fewer parameters than existing methods. Since our agent attends to only task critical visual hints, they are able to generalize to environments where task irrelevant elements are modified while conventional methods fail.[1]

ACM Reference Format:
Yujin Tang, Duong Nguyen, and David Ha. 2020. Neuroevolution of Self-Interpretable Agents. In *Genetic and Evolutionary Computation Conference (GECCO '20), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3377930.3389847

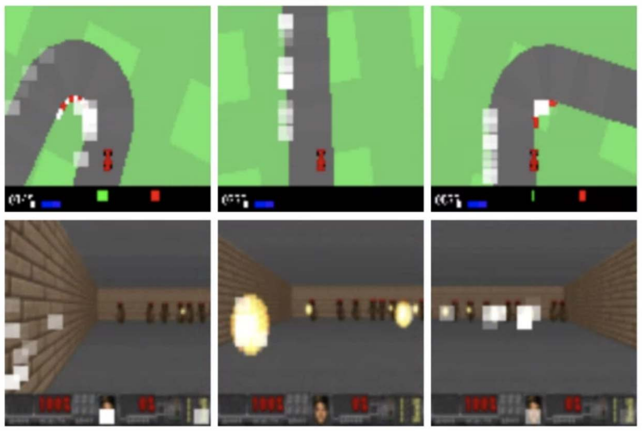


Figure 1: In this work, we evolve agents that attend to a small fraction of its visual input critical for its survival, allowing for interpretable agents that are not only compact, but also more generalizable. Here, we show examples of our agent's attention highlighted in white patches. Note that our agent's decision-making controller receives *only* the *positions* of these white patches as visualized here, *not* their contents. In CarRacing (top), our agent mostly attends to the road borders, but shifts its focus to the turns before it changes heading directions. In DoomTakeCover (bottom), the agent is able to focus on fireballs and monsters, consistent with our intuitions.

1 INTRODUCTION

While visual inputs contain rich information, humans are able to quickly locate several task related spots, extract key information and reason about them to take actions [1, 88]. In the field of machine learning (ML), existing literature has demonstrated successful applications of deep reinforcement learning (RL) to challenging tasks with visual inputs, however, it is unclear whether these agents reason similarly as we do. This lack of interpretability is one of the major concerns that caused debates in the wider adoption of ML in safety and security prioritized applications. To get to know what these agents are "thinking," some methods relied on dedicated network architectures and/or carefully designed training schemes [2, 67, 87]. Is it possible to build a simple mechanism and design an agent that behaves similarly to the way humans do?

One possible solution is to design agents that can encode and process abstract concepts. Recent works have demonstrated the importance of learning abstract information from visual inputs. For example, the *world models* line of work [26, 37, 39, 51] learns compact representations of input image sequences, and have shown that agents trained to use these representations do better in various vision-based tasks. However, not all elements of an input image are equally important. As is pointed out recently [29, 84], an agent does not have to learn representations that are capable of reconstructing the full observation—it is sufficient if the representation allows predicting quantities that are directly relevant for planning. Addressing this, recent works by Risi and Stanley [76, 77] have demonstrated that neuroevolution can be leveraged to train an agent's world model alongside its controller together, end-to-end, even when there are millions of model parameters in the agent's architecture. They also found that the abstract representations learned end-to-end are more task-specific. All these suggests a design of agents that focuses more on and learns from task critical areas of the input.

Most current methods used to train neural networks, whether with gradient descent or evolution strategies, aim to solve for the value of each individual weight parameter of a given neural network. We refer to these methods as *direct encoding* methods. *Indirect encoding* [82, 91], on the other hand, offers a radically different approach. These methods optimize instead for a *small* set of rules

[1]Videos of our results and source code available at **https://attentionagent.github.io/**

or operations, referred to as the *genotype*, that specify how the (much larger) neural network (the *phenotype*) should be generated.

Before the popularity of Deep RL, indirect encoding methods in the neuroevolution literature have also been a promising approach for vision-based RL problems. For example, earlier works demonstrated that large neural networks can be encoded into much smaller, genotype solutions, that are capable of playing Atari [43] (when it was still considered challenging in 2012) or car racing directly from pixel-only inputs [56], hinting at its potential power.

By encoding the weights of a large model with a small number of parameters, we can substantially reduce the search space of the solution, at the expense of restricting our solution to a small subspace of all possible solutions offered by direct encoding methods. This constraint naturally incorporates into our agent an *inductive bias* that determines what it does well at [28, 42, 102], and this bias is dependent on the choice of our indirect encoding method. For instance, HyperNEAT [90] has been successful at robotic gait control [14, 15, 75], suggesting CPPNs [89] to be effective at representing modular and symmetric properties suitable for locomotion. But are better there other encoding methods for vision-based tasks?

In this work, we establish that *self-attention* can be viewed as a form of indirect encoding, which enables us to construct highly parameter-efficient agents. We investigate the performance and generalization properties of these agents for vision-based RL tasks. Self-attention has been popularized by Transformer [97] models that have been successfully applied in domains such as natural language processing [21, 73] and vision [6, 18, 48, 72]. As we will explain, self-attention offers a simple yet powerful approach for parameterizing a large weight matrix of size $O(n^2)$ using only $O(d)$ number of parameter values, where n is the size of the visual input, d is the dimension of some transformed space and $n \gg d$. Furthermore, such a parameterization enforces an inductive bias to encourage our agent to attend to only a small fraction of its visual input, and as such naturally makes the agent more interpretable.

As we will show, neuroevolution is an ideal method for training self-attention agents, because not only can we remove unnecessary complexity required for gradient-based methods, resulting in much simpler architectures, we can also incorporate modules that enhance the effectiveness of self-attention that need not be differentiable. We showcase self-attention agents trained with neuroevolution that require 1000x fewer parameters than conventional methods and yet is able to solve challenging vision-based RL tasks. Specifically, with less than 4K parameters, our self-attention agents can reach average scores of 914 over 100 consecutive trials in a 2D car racing task [9] and 1125 in a 3D VizDoom task [100] (the tasks are considered solved for scores > 900 and > 750), comparable with existing state-of-the-art results [37, 76, 77]. Moreover, our agent learns to attend to only task critical visual spots and is therefore able to generalize to environments where task irrelevant elements are modified whereas conventional methods fail.

The goal of this work is to showcase self-attention as a powerful tool for the neuroevolution toolbox, and we will open-source code for reproducing our experiments. We hope our results will encourage further investigation into the neuroevolution of self-attention models, and also revitalize interest in indirect encoding methods.

2 RELATED WORK

Our work has connections to work in various areas:

Neuroscience Although the human visual processing mechanisms are not yet completely understood, recent findings from anatomical and physiological studies in monkeys suggest that visual signals are fed into processing systems to extract high level concepts such as shape, color and spatial organization [1, 27]. Research in consciousness suggests that our brains interpret our surrounding environment in a "language of thought" that is abstract enough to interface with decision making mechanisms [20]. On the other hand, while recent works in deep RL for vision-based tasks have thrived [65, 66], in most cases it is not clear why they work or fail.

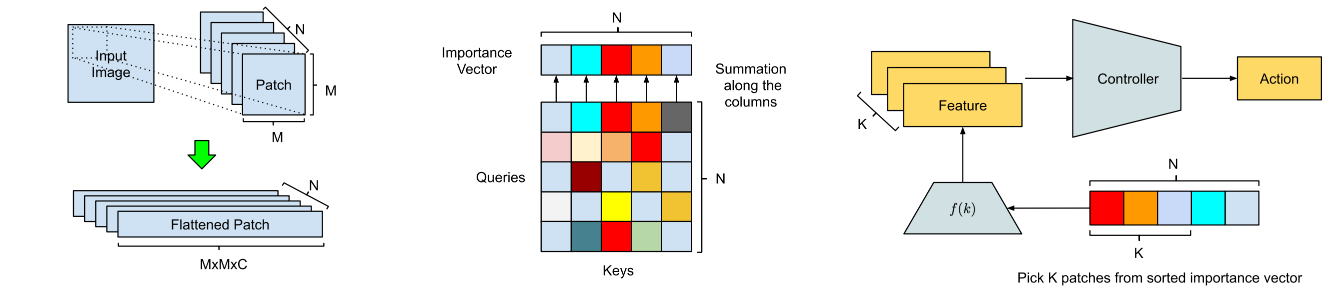
A line of work that narrows the gap is *world models* [37, 39], where the controller's inputs are abstract representations of the visual and temporal information, provided by encouraging a probabilistic "world model" to compress and predict potential future experiences. Their agent excels in challenging vision-based RL tasks, and by projecting the abstract representation back to the pixel space, it is possible to get some insights into the agent's mind. [29, 84] suggest that not all details in the visual input are equally important, specifically they pointed out that rather than learning abstract representations that are capable of reconstructing the full observation, it is sufficient if the representation allows predicting quantities that are directly relevant for planning.

While we do not fully understand the mechanisms of how our brains develop abstract representations of the world, it is believed that attention is the unconscious mechanism by which we can only attend to a few selected senses at a time, allowing our consciousness to condense sensory information into a synthetic code that is compact enough to be carried forward in time for decision making [20, 61, 99]. In this work, in place of a probabilistic world model, we investigate the use of self-attention to distill an agent's visual input into small synthetic features as inputs for a small controller.

Neuroevolution-based methods for tackling challenging RL tasks have recently gained popularity due to their simplicity and competitiveness to Deep RL methods, even on vision-based RL benchmark tasks [33, 35, 62, 80, 93]. More recent work [76, 77] demonstrated that evolution can even train RL agents with millions of weight parameters, such as the aforementioned world models-based agents. Because these approaches do not require gradient-based computation, they offer more flexibility such as discrete latent codes, being able to optimize directly for the total reward across multiple rollouts, or ease of scaling computation across machines.

It is worthy to note that even before the popularity of deep RL-based approaches for vision-based tasks, indirect encoding methods from the neuroevolution literature have been used to tackle challenging vision-based tasks such as Atari domain [43] and car navigation from pixels [56]. Indirect encoding methods are inspired by biological genotype–phenotype representations, and aim to represent a large but expressive neural network with a small *genotype* code, reducing a high dimensional optimization problem to a more manageable one that can be solved with gradient-free methods.

Indirect encoding methods are not confined to neuroevolution. Inspired by earlier works [81, 90], *hypernetworks* [36] are recurrent neural networks (RNNs) whose weight matrices can change over time, depending on the RNN's input and state. It uses an outer



Input Transformation
After some basic image pre-processing, we extract patches from an input image using a sliding window of size MxM and stride S. The resulted shape of the output is (N, M, M, C), where N is the number of patches, M is the height / width of each patch and C is the number of channels in the input image. We then flatten the data.

Importance Voting via Self-Attention
We project the flattened patches into Keys and Queries and calculate their attention matrix, the shape of which is (N, N). Softmax is applied on each row. We then sum the matrix along the columns to get a patch importance vector.

Patch Selection and Feature Retrieval
We sort the patch importance vector, extract the indices of the top K patches that have the highest importance. Then we map the patch indices to the corresponding features and input to the controller to get the action.

Figure 2: Method overview. Illustration of data processing flow in our proposed method.

product projection of an embedding vector, allowing a large weight matrix to be modified via a small genotype embedding. As we will show in the next section, self-attention also relies on taking an outer product of input and other parameter vectors to produce a much larger weight matrix. Transformers [97] demonstrated that this type of modified self-attention matrix is a tremendously powerful prior for various language modeling tasks. Here, we investigate the use of self-attention as an indirect encoding mechanism for training agents to perform vision-based RL tasks using neuroevolution.

Attention-based RL Inspired by biological vision systems, earlier works formulated the problem of visual attention as an RL problem [5, 12, 64, 83, 92]. Recent work [103] incorporated multi-head self-attention to learn representations that encode relational information between feature entities, with these features the learned agent is able to solve a novel navigation and planning task and achieve SOTA results in six out of seven StarCraft II tasks. Because the agent learned relations between entities, it can also generalize to unseen settings during training. In order to capture the interactions in a system that affects the dynamics, [31] proposed to use a group of modified RNNs. Self-attention is used to combine their hidden states and inputs. Each member competes for attention at each step, and only the winners can access the input and also other members' states. This modular mechanism improved generalization on Atari.

Attention is also explicitly targeted for interpretability in RL. In [87], the authors incorporated soft and hard attention mechanism into the deep recurrent Q-network, and they were able to outperform Deep Q-network [65] in a subset of the Atari games. Most recently, [67] used a soft, top-down attention mechanism to force the agent to focus on task-relevant information by sequentially querying its view of the environment. Their agents achieved competitive performance on Atari while being more interpretable.

The high dimensionality the visual input makes it computationally prohibitive to apply attention directly to individual pixels, and we rather operate on image *patches* (which have lower dimensions) instead. Although not in the context of self-attention, previous work (e.g. [10, 13, 22, 24, 30, 94]) segments the visual input and attend to the patches rather than individual pixels. Our work is similar to [13], where the input to the controller their is a vector of patch features weighted by attentions, the dimension of which grows linearly as we increase the number of patches. However, as our method do not rely on gradient-based learning, we can simply restrict the input to be only the K patches with highest importance. Ordinal measures have been shown to be robust and used in various feature detectors and descriptors [78]. Using gradient-free methods are more desirable in the case of non-differentiable operations because these ordinal measures can be implemented as *argmax* or top K patch selection, critical for our self-attention agent.

3 BACKGROUND ON SELF-ATTENTION

3.1 Self-Attention

We now give a brief overview of self-attention. Here, we describe a simpler subset of the full Transformer [97] architecture used in this work. In particular, we omit *Value* matrices, *positional encoding*, *multi-head attention* from our method, and opt for the simplest variation that complements neuroevolution methods for our purpose. We refer to [8] for an in-depth overview of the Transformer model.

Let $X \in \mathcal{R}^{n \times d_{in}}$ be an input sequence of n elements (e.g. number of words in a sentence, pixels in an image), each of dimensions d_{in} (e.g. word embedding size, RGB intensities). Self-attention module calculates an *attention score matrix* and a weighted output:

$$A = \text{softmax}\big(\frac{1}{\sqrt{d_{in}}}(XW_k)(XW_q)^\intercal\big) \tag{1}$$

$$Y = AX \tag{2}$$

where $W_k, W_q \in \mathcal{R}^{d_{in} \times d}$ are matrices that map the input to components called *Key* and *Query* (Key $= XW_k$, Query $= XW_q$), d is the dimension of the transformed space and is usually a small integer. Since the average value of the dot product grows with the vector's dimension, each entry in the Key and Query matrices can be disproportionally too large if d_{in} is large. To counter this, the factor $\frac{1}{\sqrt{d_{in}}}$ is used to normalize the inputs. Applying the softmax[2] operation along the rows of the matrix product in Equation 1, we get the attention matrix $A \in \mathcal{R}^{n \times n}$, where each row vector of A sums to 1. Thus, each row of output $Y \in \mathcal{R}^{n \times d_{in}}$ can be interpreted as a weighted average of the input X by each row of the matrix.

Self-attention lets us map arbitrary input X to target output Y, and this mapping is determined by an attention matrix A parameterized by much smaller Key and Query parameters, which can be trained using machine learning techniques. The self-attention mechanism is at the heart of recent SOTA methods for translation and language modeling [21, 73], and has now become a common place method for natural language processing domain.

[2] $\text{softmax}(x_i) = \exp(x_i)/\sum_k \exp(x_k)$

3.2 Self-Attention for Images

Although self-attention is broadly applied to sequential data, it is straightforward to adapt it to images. For images, the input is a tensor $X \in \mathcal{R}^{H \times W \times C}$ where H and W are the height and width of the image, C is the number of image channels (e.g., 3 for RGB, 1 for gray-scale). If we reshape the image so that it becomes $X \in \mathcal{R}^{n \times d_{in}}$ where $n = H \times W$ and $d_{in} = C$, all the operations defined in Section 3.1 are valid and can be readily applied. In the reshaped X, each row represents a pixel and the attentions are between pixels. Notice that the complexity of Equation 1 grows quadratically with the number of rows in X due to matrix multiplication, it therefore becomes computationally prohibitive when the input image is large. While down-sampling the image before applying self-attention is a quick fix, it is accompanied with performance trade-off ([18, 72] propose methods to partially overcome this trade-off).

Instead of applying operations on individual pixels of the entire input, a popular method for image recognition is to organize the image into patches and take them as inputs as described in previous work (e.g. [10, 22, 24, 30]). In our approach, our agent attends to patches of the input rather than individual pixels, and we use a sliding window to crop the input image in our input transformations. Conceptually, our approach is similar to *Spatial Softmax* [25, 57, 95], which compresses visual inputs into a set of 2D keypoints that are relevant to the task. This has been shown to work on robot perception tasks, where the keypoints are spatially interpretable.

3.3 Self-Attention as Indirect Encoding

Indirect encoding methods represent the weights of a neural network, the *phenotype*, with a smaller set of *genotype* parameters. How a genotype encodes a larger solution space is defined by the indirect encoding algorithm. HyperNEAT [90] encodes the weights of a large network via a coordinate-based CPPN-NEAT [89] network, while Compressed Network Search [56] uses discrete cosine transform to compress the weights of a large weight matrix into a small number of DCT coefficients, similar to JPEG compression.

Due to compression, the space of possible weights an indirect encoding scheme can produce is only a small subspace of all possible combination of weights. The constraint on the solution space resulting from indirect encoding enforces an inductive bias into the phenotype. While this inductive bias determines the types of tasks that the network is *naturally* suited at, it also restricts the network to a subset of all possible tasks that an unconstrained phenotype can (in theory) perform. More recent works have proposed ways to broaden its task domain of indirect encoding. [74] proposed adapting part of the indirect encoding algorithm itself to the task environment. Hypernetworks [36] suggested making the phenotype directly dependent on the inputs, thus tailoring the weights of the phenotype to the specific inputs of the network. By incorporating information from the input into the weight-generating process, it has been shown [11, 23, 69, 98] that the phenotype can be highly expressive as the weights can adapt to the inputs for the task at hand, while static indirect encoding methods cannot.

Similarly, self-attention enforces a structure on the attention weight matrix A in Equation 1 that makes it also input-dependent. If we remove the Key and Query terms, the outer product XX^T[3] defines an association matrix where the elements are large when two distinct input terms are in agreement. This type of structure enforced in A has been shown to be suited for associative tasks where the downstream agent has to learn the relationship between unrelated items. For example, they are used in the Hebbian learning [44] rule inspired by *neurons that fire together wire together*, shown to be useful for associative learning [4, 63]. Matrix factorization applied to weights has been proposed in the deep learning literature [32, 79], and are also present in recommender systems [55] to represent relationships between different inputs.

[3] XX^T are also known as Gram matrices, and are key to classical statistical learning.

As the outer product XX^T so far has no free parameters, the corresponding matrix A will not be suitable for arbitrary tasks beyond association. The role of the small Key and Query matrices in Equation 1 allow A to be modified for the task at hand. $W_k, W_q \in \mathcal{R}^{d_{in} \times d}$ are the matrices that contain the free parameters, d_{in} is a constant with image inputs (3 for RGB images and 1 for gray scale images), therefore the number of free parameters in self-attention is in the order of $O(d)$. As we explained previously, when applying self-attention to images n can be the number of pixels in an input the magnitude of which is often tens of thousands for moderately sized images. On the other hand, d is the dimension of the transformed space in which the Key and Query matrices reside and is often much smaller than n ($d = 4$ in our experiments). This form of indirect encoding enables us to represent the phenotype, the attention matrix A, of size $O(n^2)$ using only $O(d)$ number of genotype parameters. In our experiments, we show that our attention matrix A can be represented using only $\sim$ 1200 trainable genotype parameters.

Furthermore, we demonstrate that features from this attention matrix is especially useful to a downstream decision-making controller. We find that even if we restrict the size of our controller to only $\sim$ 2500 parameters, it can still solve challenging vision-based tasks by leveraging the information provided by self-attention.

4 PROPOSED METHOD

Encouraged by the success of early Deep RL works such as [65], most succeeding works adopted variants of earlier models rooted from a similar design. For instance, convolution layers for image down-sampling and feature extraction (the visual feature extraction module) are first used to extract features that are then fed to fully-connected or recurrent layers to produce value estimations or control signals (the controller module). In such a design, the visual feature extraction module regards each of the elements in the entire input image of equal importance and relies on the training signal to direct its learning so that a small fraction of the weights learn to emphasize task related factors while the others deal with nuances.

Our proposed method is based on a different premise: when the brain is involved in effort-demanding tasks, it assigns most of its attention capacity only to task relevant elements and is temporarily blind to other signals [50, 61]. In this vein, our agent is designed to focus on only task critical regions in the input image and ignore the others, Figure 2 depicts the overview of our proposed method. To be concrete, given an observation our agent first resizes it into an input image of shape $L \times L$, the agent then segments the image into N patches and regard each patch as a potential region to attend to. To decide which patches are appropriate, the agent passes the patches to the self-attention module to get a vector representing each patch's importance, based on which it selects K patches of the highest importance. It then uses the index (k) of each of the K patches to fetch relevant features with a function $f(k)$ (described in Sec. 4.3), which can be either a learned module or a pre-defined function that incorporates domain knowledge. Finally, the agent inputs the features to its controller and generates the action corresponding to the given observation.

To gain a better sense of the magnitudes involved, we summarize the hyper-parameters used in this work in Table 1. Some of the parameters are explained in the following sections.

Table 1: Hyper-parameters in this paper. Left: Parameters for input transformation. After resizing the observation into an image of shape $L\times L$, we use a sliding window of specified size and stride to segment the image into $N = (\lfloor \frac{L-M}{S} + 1 \rfloor)^2 = (\lfloor \frac{96-7}{4} + 1 \rfloor)^2 = 529$ patches. Right: Parameters for self-attention. Since the attention is between patches and each patch is RGB, we therefore have $d_{in} = M^2 \times C = 7^2 \times 3 = 147$.

Parameter	Value
Input size (L)	96
Window size (M)	7
Stride (S)	4
# of Patches (N and n)	529

Parameter	Value
d_{in}	147
d	4
K	10
# of LSTM neurons	16

4.1 Input Transformation

Our agent does some basic image processing and then segments an input image into multiple patches. For all the experiments in this paper, our agent receives RGB images as its input, therefore we simply divide each pixel by 255 to normalize the data, but it should be straightforward to integrate other data preprocessing procedures. Similarly, while there can be various methods for image segmentation, we find a simple sliding window strategy to be sufficient for the tasks in this paper. To be concrete, when the window size M and stride S are specified, our agent chops an input of shape (H, W, C) into a batch of N patches of shape (M, M, C), where H and W are the height and width of the input image and C is the number of channels. We then reshape the processed data into a matrix of shape $(N, M \times M \times C)$ before feeding it to the self-attention module. M and S are hyper-parameters to our model that determine how large each patch is and whether patches overlap. In the extreme case when $M = S = 1$ this becomes self-attention on each individual pixel in the image.

4.2 Importance Voting via Self-Attention

Upon receiving the transformed data in $\mathcal{R}^{n\times d_{in}}$ where $n = N$ and $d_{in} = M \times M \times C$, the self-attention module follows Equation 1 to get the attention matrix of shape (N, N). To keep the agent as simple as possible, we do not use positional encoding in this work.

By applying softmax, each row in the attention matrix sums to one, so the attention matrix can be viewed as the results from a voting mechanism between the patches. To be specific, if each patch can distribute fractions of a total of 1 vote to other patches (including itself), row i thus shows how patch i has voted and column j gives the votes that patch j acquired from others. In this interpretation, entry (i, j) in the attention matrix is then regarded as how important patch j is from patch i's perspective. Taking sums along the columns of the attention matrix results in a vector that summarizes the total votes acquired by each patch, and we call this vector the *patch importance vector*. Unlike conventional self-attention operations, we rely solely on the patch importance vector and do not calculate a weighted output with Equation 2.

4.3 Patch Selection and Feature Retrieval

Based on the patch importance vector, our agent picks the K patches with the highest importance. We pass in the index of these K patches (denoted as index k to reference the k^{th} patch) into a *feature retrieval operation* $f(k)$ to query the for their features. $f(k)$ can be static mappings or learnable modules, and it returns the features related to the image region centered at patch k's position. The following list gives examples of possible features:

- **Patch center position.** $f(k) : \mathcal{R} \mapsto \mathcal{R}^2$ where the output contains the row and column indices of patch k's center.
- **Patch's image histogram.** $f(k) : \mathcal{R} \mapsto \mathcal{R}^b$ where the output is the image histogram calculated from patch k and b is the number of bins.
- **Convolution layers' output.** $f(k) : \mathcal{R} \mapsto \mathcal{R}^{s\times s\times m}$ is a stack of convolution layers (learnable or fixed with pre-trained weights). It takes the image region centered at patch k as input and outputs a tensor of shape $s \times s \times m$.

In this work, we use the first example for its simplicity. These design choices give us control over our agent's capabilities and computational efficiency, and will also affect its interpretability.

By discarding patches of low importance the agent becomes temporarily blind to other signals, this is built upon our premise and effectively creates a bottleneck that forces the agent to focus on patches only if they are critical to the task. Once learned, we can visualize the K patches and see directly what the agent is attending to (see Figure 1). Although this mechanism introduces K as a hyper-parameter, we find it easy to tune (along with M and S). In principle we can also let neuroevolution decide on the number of patches, and we will leave this for future work.

Pruning less important patches also leads to the reduction of input features, so the agent is more efficient by solving tasks with fewer weights. Furthermore, correlating the feature retrieval operation $f(k)$ with individual patches can also lower the computational cost. For instance, if some local features are known to be useful for the task yet computationally expensive, K acts as a budget cap and only compute features from the most promising regions. Notice however, this does not imply we permit only local features, as $f(k)$ also has the flexibility to incorporate global features. In this paper, $f(k)$ is a simple mapping from patch index to patch position in the image and is a local feature. But $f(k)$ can also be a stack of convolution layers whose receptive fields are centered at patch k. If the receptive fields are large, $f(k)$ can provide global features.

4.4 Controller

Temporal information between steps is important to most RL tasks, but single RGB images as our input at each time step do not provide this information. One option is to stack multiple input frames like what is done in [66], but we find this inelegant approach unsatisfactory because the time window we care about can vary for different tasks. Another option is to incorporate the temporal information as a hidden state inside our controller. Previous work [19] has demonstrated that with a good representation of the input image, even a small RNN controller with only 6–18 neurons is sufficient to perform well at several Atari games using only visual inputs.

In our experiments, we use Long short-term memory (LSTM) [47] network as our RNN controller so that its hidden state can capture temporal information across multiple input image frames. We find that an LSTM with only 16 neurons is sufficient to solve challenging tasks when combined with features extracted from self-attention.

4.5 Neuroevolution of the Agent

Operators such as importance sorting and patch pruning in our proposed methods are not gradient friendly. It is not straightforward to apply back-propagation in the learning phase. Besides, restricting to gradient based learning methods can prohibit the adoption of learnable feature retrieval functions $f(k)$ that consist of discrete operations or produce discrete features. We therefore turn to evolution algorithms to train our agent. While it is possible to train our agent using any evolution strategy or genetic algorithms (GA), empirically we find the performance of Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [40] stable on a set of tasks [34, 96]. Despite its power, the computation of the full covariance matrix is non-trivial, and because of this CMA-ES is rarely applied to problems in high-dimensional space [68] such as the tasks dealing with visual inputs. As our agent contains significantly fewer parameters than conventional methods, we are therefore able to train it with an off-the-shelf implementation of CMA-ES [41].

5 EXPERIMENTS

Our goal in this section is to answer the following questions via experiments and analysis:

(1) Is our agent able to solve challenging vision-based RL tasks? If so, what are the advantages over other methods?
(2) How robust is the learned agent? If the agent is focusing on task critical factors, does it generalize to the environments with modifications that are irrelevant to the core mission?

We evaluate our method in two challenging vision-based RL tasks: CarRacing [54] and DoomTakeCover [53, 71]. See the first column of Figure 3 for sample screenshots from these tasks. Refer to [38] for an in-depth description of these tasks. For comparison purposes, we also include World Models [37], GA [76], DIP [77] and Proximal Policy Optimization (PPO) [60, 85] as baselines.

5.1 Agent Settings

Our network architecture and related parameters are shown in Figure 4. We resize the input images to 96 × 96 and use the same architecture for both CarRacing and DoomTakeCover (except for the output dimensions). We use a sliding window of size $M = 7$ and stride $S = 4$ to segment the input image, this gives us $N = 529$ patches. After reshaping, we get an input matrix X of shape $(n = 529, d_{in} = 147)$. We project the input matrix to Key and Query with $d = 4$, after self-attention is applied we extract features from the $K = 10$ most importance patches and input to the single layer LSTM controller (#hidden=16) to get the action. Table 2 summarizes the number of parameters in our agent, we have also included models from some existing works for the purpose of comparison . For

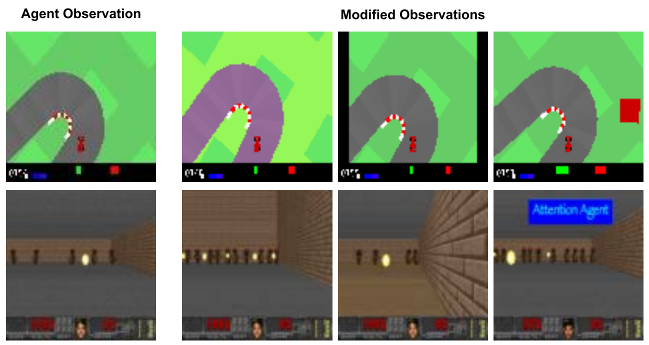


Figure 3: CarRacing and DoomTakeCover. *Left:* Original tasks. The observations are resized to 96x96px and presented to the agent. *Right:* Modified CarRacing environments: Color Perturbation, Vertical Frames, Background Blob. Modified DoomTakeCover environments: Higher Walls, Different Floor Texture, Hovering Text.

feature retrieval function $f(k)$, we use a simple mapping from patch index to patch center position in the input image. We normalize the positions by dividing the largest possible value so that each coordinate is between 0 and 1.

Table 2: Learnable parameters. GA, DIP share the same world model architecture. Fully connected (FC) layers include bias term.

Method	Component	#Params	Total
World model [37] GA [76] DIP [77]	VAE	4,348,547	4,733,485
	MD-RNN	384,071	
	Controller	867	
PPO [60]	Conv Stack	393,848	445,955
	FC Stack	52,107	
Ours	FC (Query)	592	3,667
	FC (Key)	592	
	LSTM (#hidden=16)	2,483	

We use pycma [41], an off-the-shelf implementation of CMA-ES [40] to train our agent. We use a population size of 256, set the initial standard deviation to 0.1 and keep all other parameters at default values. To accurately evaluate the fitness of each individual in the population, we take the mean score over 16 rollouts in CarRacing and 5 rollouts in DoomTakeCover as the its fitness.

5.2 Results

Not only is our agent able to solve both tasks, it also outperformed the existing methods, Table 3 summarizes our agent's scores. In addition to the SOTA scores, the attention patches visualized in pixel space also make it easier for humans to understand the decisions made by our agent. In Figure. 1, we plot the top K important patches elected by the self-attention module on top of the input image and see directly what the agent is attending to (see the accompanying videos for more results). The opacity indicates the importance, the whiter the more important.

From the figures, we notice that most of the patches the agent attends to are consistent with humans intuition. For example in CarRacing, the agent's attention is on the border of the road but shifts its focus to the turns before the car needs to change its heading direction. Notice the attentions are mostly on the left side of the road. This makes sense from a statistical point of view considering that the racing lane forms a closed loop and the car is always

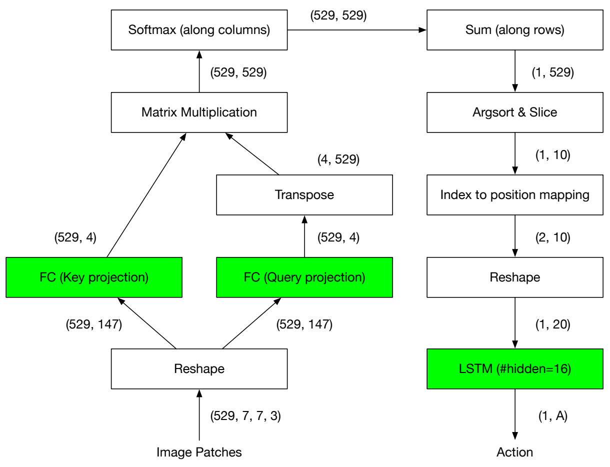


Figure 4: Agent network architecture. Numbers next to each arrow indicate data shape after the operation. A is the action dimension and is task dependent. Learnable parameters in green.

running in a counter-clockwise direction. In DoomTakeCover, the agent is able to focus its attention on fireballs. When the agent is near the corner of the room, it is also able to detect the wall and change its dodging strategy instead of stuck into the dead end. Notice the agent also distributes its attention on the panel at the bottom, especially on the profile photo in the middle. We suspect this is because the controller is using patch positions as its input, and it learned to use these points as anchors to estimate its distances to the fireballs. We also notice that the scores from all methods have large variance in DoomTakeCover. This seems to be caused by the environment's design: some fireballs might be out of the agent's sight but are actually approaching. The agent can still be hit by them when it's dodging other fireballs that are in the vision.

Table 3: Scores from CarRacing and DoomTakeCover. We report the average score over 100 consecutive tests with standard deviations. For reference, the required scores above which the tasks are considered solved are also included. Best scores are highlighted.

Method	**CarRacing**	**DoomTakeCover**
Required score	900	750
World model [37]	906 ± 21	1092 ± 556
PPO [60]	865 ± 159	-
GA [76]	903 ± 73	-
DIP [77]	-	824 ± 492
Ours	**914** ± 15	**1125** ± 589

Through these tasks, we are able to give a positive answer to the first question in Section 5. Our agent is indeed able to solve these vision-based RL challenges, and it is efficient in terms of being able to reach higher scores with significantly fewer parameters.

5.3 Region of Interest to Importance Mapping

As our feature retrieval function $f(k)$ is a mapping from patch index to normalized patch center positions, it provides location information, but discards the content in the patches. On first thought it is surprising to see that the agent is able to solve tasks with the position information alone, but a closer look at Figure 5 reveals the agent has learned not only *where* but also *what* to attend to.

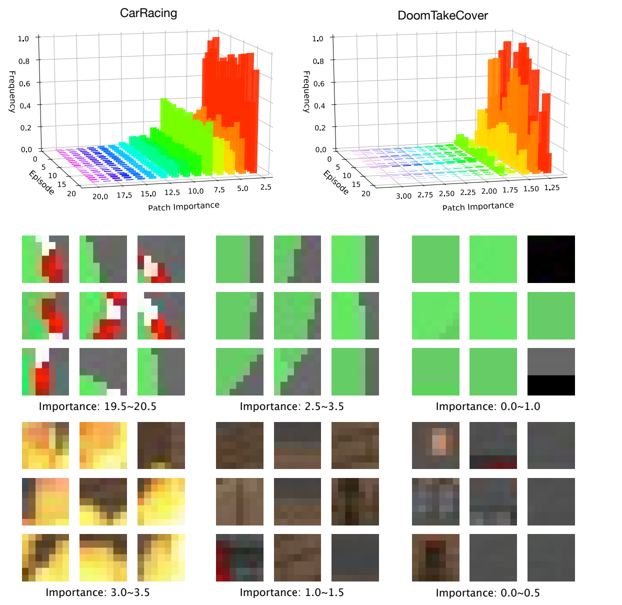


Figure 5: Region of interest to patch importance mapping. Importance voting mechanism via self-attention is able to identify a small minority of patches that are important for the task at hand. The histogram shows the importance distribution of patches from 20 test episodes by patch importance scores (top). Example patches sampled from specified importance ranges (bottom).

In Figure 5, we plot the histogram of patch importance that are in the top 5% quantile from 20 test episodes. When sampling and plotting patches whose importance are in the specified ranges, we find that the agent is able to map regions of interest (ROI) to higher importance values. The patches of the highest importance are those critical to the core mission. These are the patches containing the red and white markers at the turns in CarRacing and the patches having fires in DoomTakeCover (patches on the left). Shifting to the range that is around the 5% quantile, the patch samples are not as interesting as before but still contains useful information such as the border of the road in CarRacing and the texture of walls in DoomTakeCover. If we take an extreme and look at the patches with close to zero importance (patches on the right), those patches are mostly featureless and indeed have little information. By mapping ROIs to importance values, the agent is able to segment and discriminate the input to its controller and learn what the objects are it is attending to.

5.4 Generalize to Modified Environments

To test our agent's robustness and its ability to generalize to novel states, we test pre-trained agents in modified CarRacing and DoomTakeCover *without* re-training or fine-tuning. While there are infinitely many ways to modify an environment, our modifications respect one important principle: the modifications should not cause changes of the core mission or critical information loss. With this design principle in mind, we present the following modifications:

- **CarRacing - Color Perturbation** We randomly perturb the background color. At the beginning of each episode, we sample two scalar perturbations uniformly from the interval

$[-0.2, 0.2]$ and add respectively to the lane and grass field RGB vectors. Once the perturbation is added, the colors remain constant throughout the episode.

- **CarRacing - Vertical Bars** We add black vertical bars to both sides of the screen. The window size of CarRacing is 800px × 1000px, and we add two vertical bars of width 75px on the two sides of the window.
- **CarRacing - Background Blob** We add a red blob at a fixed position relative to the car. In CarRacing, as the lane is a closed loop and the car is designed to run in the counter clock-wise direction, the blob is placed to the north east of the car to reduce lane occlusion.
- **DoomTakeCover - Higher Walls** We make the wall higher and keep all other settings the same.
- **DoomTakeCover - Different Floor Texture** We change the texture of the floor and keep all other settings the same.
- **DoomTakeCover - Hovering Text** We place a blue blob containing text on top part of the screen. The blob is placed to make sure no task critical visual information is occluded.

For the purpose of comparison, we used the released code (and pre-trained models, if available) from [37, 76] as baselines. While our reproduced numbers do not exactly match the reported scores, they are within error bounds, and close enough for the purpose of testing for generalization. For each modification, we test a trained agent for 100 consecutive episodes and report its scores in Table 4.

Table 4: Generalization tests. We train agents in the original task and test in the modified environments without re-training. For comparison, we also include the performance in the unmodified tasks. Results with significant performance drop highlighted.

CarRacing	Original	Color Perturb	Vertical Bars	Blob
WM [37]	901 ± 37	655 ± 353	**166 ± 137**	**446 ± 299**
GA [76]	859 ± 79	**442 ± 362**	675 ± 254	833 ± 135
PPO [60]	865 ± 159	505 ± 464	615 ± 217	855 ± 172
Ours	914 ± 15	866 ± 112	900 ± 35	898 ± 53

TakeCover	Original	Higher Walls	Floor Texture	Text
WM [37]	959 ± 564	**243 ± 104**	**218 ± 69**	**240 ± 63**
Ours	1125 ± 589	934 ± 560	1120 ± 613	1035 ± 627

Our agent generalizes well to all modifications while the baselines fail. While world model (WM) does not suffer a large performance drop in color perturbations in CarRacing, it is sensitive to all changes. Specifically, we observe > 75% score drops in Vertical Frames, Higher Walls, Floor Texture, Hovering Text and a > 50% score drop in Background Blob from its performances in the unmodified tasks. Since WM's controller used as input the abstract representations it learned from reconstructing the input images, without much regularization, it is likely that the learned representations will encode visual information that is crucial to image reconstruction but not task critical. If this visual information to be encoded is modified in the input space, the model produces misleading representations for the controller and we see performance drop.

In contrast, GA and PPO performed better at generalization tests. The end-to-end training may have resulted in better task-specific representations learned compared to World model, which uses an unsupervised representation based data collected from random policies. Both GA and PPO can fine-tune their perception layers to assign greater importance to particular regions via weight learning.

Through these tests, we are able to answer the second question in Section 5: The small change in performance shows that our agent is robust to modifications. Unlike baseline methods that are subject to visual distractions, our agent focuses only on task critical positions, and simply relies on the coordinates of small patches of its visual input identified via self-attention, and is still able to keep similar performance in the modified tasks without any re-training. By learning to ignore parts of the visual input that it deems irrelevant to the task, it can naturally still perform its task even when irrelevant parts of its environment are modified.

6 DISCUSSION

The paper demonstrated that self-attention is a powerful module for creating RL agents that is capable of solving challenging vision-based tasks. Our agent achieves competitive results on CarRacing and DoomTakeCover with significantly fewer parameters than conventional methods, and is easily interpretable in pixel space. Trained with neuroevolution, the agent learned to devote most of its attention to visual hints that are task critical and is therefore able to generalize to environments where task irrelevant elements are modified while conventional methods fail.

Yet, our agent is nowhere close to generalization capabilities of humans. The modifications to the environments in our experiments are catered to attention-based methods. In particular, we have not modified properties of objects of interest, where our method may perform as poorly (or worse) than methods that do not require sparse attention in pixel space. We believe this work complements other approaches (e.g. [3, 31, 45, 46, 52, 58, 70, 86, 101, 105]) that approach the generalization problem, and future work will continue to develop on these ideas to push the generalization abilities proposed in more general domains [7, 16, 17, 49, 59, 70, 104]. For interesting failure cases, please visit https://attentionagents.github.io/.

Neuroevolution is a powerful toolbox for training intelligent agents, yet its adoption in RL is limited because its effectiveness when applied to large deep models was not clear until only recently [76, 93]. We find neuroevolution to be ideal for learning agents with self-attention. It allows us to produce a much smaller model by removing unnecessary complexity needed for gradient-based method. In addition, it also enables the agent to incorporate modules that include discrete and non-differentiable operations that are helpful for the tasks. With such small yet capable models, it is exciting to see how neuroevolution trained agents would perform in vision-based tasks that are currently dominated by Deep RL algorithms in the existing literature.

In this work, we also establish the connections between indirect encoding methods and self-attention. Specifically, we show that self-attention can be viewed as a form of indirect encoding. Another interesting direction for future works is therefore to explore more specific forms of indirect encoding that, when combined with neuroevolution, can produce RL agents with useful innate behaviors.

Acknowledgments *The authors would like to thank Yingtao Tian, Lana Sinapayen, Shixin Luo, Krzysztof Choromanski, Sherjil Ozair, Ben Poole, Kai Arulkumaran, Eric Jang, Brian Cheung, Kory Mathewson, Ankur Handa, and Jeff Dean for valuable discussions.*

REFERENCES

[1] 2012. *Vision: Processing Information.* Retrieved January 10, 2020 from https://www.brainfacts.org/thinking-sensing-and-behaving/vision/2012/vision-processing-information

[2] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. 2018. Sanity checks for saliency maps. In *Advances in Neural Information Processing Systems*. 9505–9515. https://arxiv.org/abs/1810.03292

[3] Rishabh Agarwal, Chen Liang, Dale Schuurmans, and Mohammad Norouzi. 2019. Learning to generalize from sparse and underspecified rewards. *arXiv preprint arXiv:1902.07198* (2019). https://arxiv.org/abs/1902.07198

[4] Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. 2016. Using fast weights to attend to the recent past. In *Advances in Neural Information Processing Systems*. 4331–4339. https://arxiv.org/abs/1610.06258

[5] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. 2014. Multiple object recognition with visual attention. *arXiv preprint arXiv:1412.7755* (2014). https://arxiv.org/abs/1412.7755

[6] Irwan Bello, Barret Zoph, Ashish Vaswani, Jonathon Shlens, and Quoc V Le. 2019. Attention augmented convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*. 3286–3295. https://arxiv.org/abs/1904.09925

[7] Benjamin Beyret, José Hernández-Orallo, Lucy Cheke, Marta Halina, Murray Shanahan, and Matthew Crosby. 2019. The Animal-AI Environment: Training and Testing Animal-Like Artificial Cognition. *arXiv preprint arXiv:1909.07483* (2019). https://arxiv.org/abs/1909.07483

[8] Peter Bloem. 2019. Transformers from Scratch. *http://www.peterbloem.nl/* (2019). http://www.peterbloem.nl/blog/transformers

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. http://arxiv.org/abs/1606.01540 cite arxiv:1606.01540.

[10] Yuning Chai. 2019. Patchwork: A Patch-wise Attention Network for Efficient Object Detection and Segmentation in Video Streams. *CoRR* abs/1904.01784 (2019). arXiv:1904.01784 http://arxiv.org/abs/1904.01784

[11] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. 2018. Neural ordinary differential equations. In *Advances in neural information processing systems*. 6571–6583. https://arxiv.org/abs/1806.07366

[12] Brian Cheung, Eric Weiss, and Bruno Olshausen. 2016. Emergence of foveal image sampling from learning to attend in visual scenes. *arXiv preprint arXiv:1611.09430* (2016). https://arxiv.org/abs/1611.09430

[13] Jinyoung Choi, Beom-Jin Lee, and Byoung-Tak Zhang. 2017. Multi-Focus Attention Network for Efficient Deep Reinforcement Learning. In *The Workshops of the The Thirty-First AAAI Conference on Artificial Intelligence, Saturday, February 4-9, 2017, San Francisco, California, USA.* http://aaai.org/ocs/index.php/WS/AAAIW17/paper/view/15100

[14] Jeff Clune, Benjamin E Beckmann, Charles Ofria, and Robert T Pennock. 2009. Evolving coordinated quadruped gaits with the HyperNEAT generative encoding. In *2009 iEEE congress on evolutionary computation*. IEEE, 2764–2771. https://bit.ly/2SqUrNJ

[15] Jeff Clune, Kenneth O Stanley, Robert T Pennock, and Charles Ofria. 2011. On the performance of indirect encoding across the continuum of regularity. *IEEE Transactions on Evolutionary Computation* 15, 3 (2011), 346–367. https://bit.ly/2V8g3QG

[16] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. 2019. Leveraging Procedural Generation to Benchmark Reinforcement Learning. *arXiv preprint arXiv:1912.01588* (2019). https://arxiv.org/abs/1912.01588

[17] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. 2018. Quantifying generalization in reinforcement learning. *arXiv preprint arXiv:1812.02341* (2018). https://arxiv.org/abs/1812.02341

[18] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. 2019. On the Relationship between Self-Attention and Convolutional Layers. *CoRR* abs/1911.03584 (2019). arXiv:1911.03584 http://arxiv.org/abs/1911.03584

[19] Giuseppe Cuccu, Julian Togelius, and Philippe Cudré-Mauroux. 2019. Playing atari with six neurons. In *Proceedings of the 18th international conference on autonomous agents and multiagent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 998–1006.

[20] Stanislas Dehaene. 2014. *Consciousness and the brain: Deciphering how the brain codes our thoughts*. Penguin. https://en.wikipedia.org/wiki/Consciousness_and_the_Brain

[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[22] Lei Ding, Hao Tang, and Lorenzo Bruzzone. 2019. Improving Semantic Segmentation of Aerial Images Using Patch-based Attention. *ArXiv* (2019). https://arxiv.org/abs/1911.08877

[23] Vincent Dumoulin, Ethan Perez, Nathan Schucher, Florian Strub, Harm de Vries, Aaron Courville, and Yoshua Bengio. 2018. Feature-wise transformations. *Distill* (2018). https://doi.org/10.23915/distill.00011

[24] Gamaleldin Elsayed, Simon Kornblith, and Quoc V Le. 2019. Saccader: Improving Accuracy of Hard Attention Models for Vision. In *Advances in Neural Information Processing Systems*. 700–712. https://arxiv.org/abs/1908.07644

[25] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. 2016. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 512–519. https://arxiv.org/abs/1509.06113

[26] Daniel Freeman, David Ha, and Luke Metz. 2019. Learning to Predict Without Looking Ahead: World Models Without Forward Prediction. In *Advances in Neural Information Processing Systems*. 5380–5391. https://learningtopredict.github.io/

[27] Jeremy Freeman and Eero P Simoncelli. 2011. Metamers of the ventral stream. *Nature neuroscience* 14, 9 (2011), 1195. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3164938/

[28] Adam Gaier and David Ha. 2019. Weight agnostic neural networks. In *Advances in Neural Information Processing Systems*. 5365–5379. https://weightagnostic.github.io

[29] Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G Bellemare. 2019. Deepmdp: Learning continuous latent space models for representation learning. *arXiv preprint arXiv:1906.02736* (2019). https://arxiv.org/abs/1906.02736

[30] Nils Gessert, Thilo Sentker, Frederic Madesta, Rudiger Schmitz, Helge Kniep, Ivo Baltruschat, Rene Werner, and Alexander Schlaefer. 2019. Skin Lesion Classification Using CNNs with Patch-Based Attention and Diagnosis-Guided Loss Weighting. *IEEE Transactions on Biomedical Engineering* (2019). https://arxiv.org/abs/1905.02793

[31] Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. 2019. Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893* (2019). https://arxiv.org/abs/1909.10893

[32] Roger Grosse and James Martens. 2016. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*. 573–582. http://www.jmlr.org/proceedings/papers/v48/grosse16.pdf

[33] D. Ha. 2017. Evolving Stable Strategies. *http://blog.otoro.net/* (2017). http://blog.otoro.net/2017/11/12/evolving-stable-strategies/

[34] David Ha. 2017. A Visual Guide to Evolution Strategies. *http://blog.otoro.net* (2017). http://blog.otoro.net/2017/10/29/visual-evolution-strategies/

[35] David Ha. 2018. Reinforcement Learning for Improving Agent Design. *arXiv:1810.03779* (2018). https://designrl.github.io

[36] David Ha, Andrew Dai, and Quoc V Le. 2017. Hypernetworks. In *Fifth International Conference on Learning Representations (ICLR 2017)*. https://openreview.net/forum?id=rkpACe1lx

[37] David Ha and Jürgen Schmidhuber. 2018. Recurrent World Models Facilitate Policy Evolution. In *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc., 2451–2463. https://worldmodels.github.io

[38] David Ha and Jürgen Schmidhuber. 2018. World models. *arXiv preprint arXiv:1803.10122* (2018). https://worldmodels.github.io/

[39] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. 2018. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551* (2018). https://planetrl.github.io/

[40] Nikolaus Hansen. 2006. *The CMA Evolution Strategy: A Comparing Review*. Springer Berlin Heidelberg, Berlin, Heidelberg, 75–102. https://doi.org/10.1007/3-540-32494-1_4

[41] Nikolaus Hansen, Youhei Akimoto, and Petr Baudis. 2019. CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634. https://doi.org/10.5281/zenodo.2559634

[42] Uri Hasson, Samuel A Nastase, and Ariel Goldstein. 2020. Direct Fit to Nature: An Evolutionary Perspective on Biological and Artificial Neural Networks. *Neuron* 105, 3 (2020), 416–434. https://www.biorxiv.org/content/10.1101/764258v2.full

[43] Matthew Hausknecht, Piyush Khandelwal, Risto Miikkulainen, and Peter Stone. 2012. HyperNEAT-GGP: A HyperNEAT-based Atari general game player. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. 217–224. http://nn.cs.utexas.edu/downloads/papers/hausknecht.gecco12.pdf

[44] Donald O Hebb. 1949. *The organization of behavior*. na. https://en.wikipedia.org/wiki/Hebbian_theory

[45] Irina Higgins, Arka Pal, Andrei Rusu, Loic Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. 2017. Darla: Improving zero-shot transfer in reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1480–1490. https://arxiv.org/abs/1707.08475

[46] Felix Hill, Andrew Lampinen, Rosalia Schneider, Stephen Clark, Matthew Botvinick, James L McClelland, and Adam Santoro. 2019. Emergent systematic generalization in a situated agent. *arXiv preprint arXiv:1910.00571* (2019). https://arxiv.org/abs/1910.00571

[47] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[48] Han Hu, Zheng Zhang, Zhenda Xie, and Stephen Lin. 2019. Local relation networks for image recognition. In *Proceedings of the IEEE International Conference on Computer Vision*. 3464–3473. https://arxiv.org/abs/1904.11491

[49] Arthur Juliani, Ahmed Khalifa, Vincent-Pierre Berges, Jonathan Harper, Ervin Teng, Hunter Henry, Adam Crespi, Julian Togelius, and Danny Lange. 2019. Obstacle tower: A generalization challenge in vision, control, and planning. *arXiv preprint arXiv:1902.01378* (2019). https://arxiv.org/abs/1902.01378

[50] Daniel Kahneman. 2011. *Thinking, fast and slow*. Farrar, Straus and Giroux, New York. https://en.wikipedia.org/wiki/Thinking,_Fast_and_Slow

[51] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. 2019. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374* (2019).

[52] Ken Kansky, Tom Silver, David A Mély, Mohamed Eldawy, Miguel Lázaro-Gredilla, Xinghua Lou, Nimrod Dorfman, Szymon Sidor, Scott Phoenix, and Dileep George. 2017. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1809–1818. https://arxiv.org/abs/1706.04317

[53] Michal Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaskowski. 2016. ViZDoom: A Doom-based AI research platform for visual reinforcement learning. In *IEEE Conference on Computational Intelligence and Games, CIG 2016, Santorini, Greece, September 20-23, 2016*. 1–8. https://doi.org/10.1109/CIG.2016.7860433

[54] Oleg Klimov. 2016. *CarRacing-v0*. Retrieved January 17, 2020 from https://gym.openai.com/envs/CarRacing-v0/

[55] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37. https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf

[56] Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez. 2013. Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. 1061–1068. http://people.idsia.ch/~juergen/compressednetworksearch.html

[57] Tejas D Kulkarni, Ankush Gupta, Catalin Ionescu, Sebastian Borgeaud, Malcolm Reynolds, Andrew Zisserman, and Volodymyr Mnih. 2019. Unsupervised learning of object keypoints for perception and control. In *Advances in Neural Information Processing Systems*. 10723–10733. https://bit.ly/2wLiEFZ

[58] Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. 2020. Network Randomization: A Simple Technique for Generalization in Deep Reinforcement Learning. In *International Conference on Learning Representations*. https://openreview.net/forum?id=HJgcvJBFvB

[59] Joel Z Leibo, Cyprien de Masson d'Autume, Daniel Zoran, David Amos, Charles Beattie, Keith Anderson, Antonio García Castañeda, Manuel Sanchez, Simon Green, Audrunas Gruslys, et al. 2018. Psychlab: a psychology laboratory for deep reinforcement learning agents. *arXiv preprint arXiv:1801.08116* (2018). https://arxiv.org/abs/1801.08116

[60] Xiaoteng Ma. 2019. *Car Racing with PyTorch*. Retrieved March 6, 2020 from https://github.com/xtma/pytorch_car_caring

[61] Arien Mack, Irvin Rock, et al. 1998. *Inattentional blindness*. MIT press. https://en.wikipedia.org/wiki/Inattentional_blindness

[62] Horia Mania, Aurelia Guy, and Benjamin Recht. 2018. Simple random search of static linear policies is competitive for reinforcement learning. In *Advances in Neural Information Processing Systems*. 1800–1809. https://bit.ly/38sMEEn

[63] Thomas Miconi, Jeff Clune, and Kenneth O Stanley. 2018. Differentiable plasticity: training plastic neural networks with backpropagation. *arXiv preprint arXiv:1804.02464* (2018). https://arxiv.org/abs/1804.02464

[64] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. 2014. Recurrent models of visual attention. In *Advances in neural information processing systems*. 2204–2212. https://arxiv.org/abs/1406.6247

[65] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013). https://arxiv.org/abs/1312.5602

[66] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533. https://daiwk.github.io/assets/dqn.pdf

[67] Alexander Mott, Daniel Zoran, Mike Chrzanowski, Daan Wierstra, and Danilo Jimenez Rezende. 2019. Towards Interpretable Reinforcement Learning Using Attention Augmented Agents. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*. 12329–12338. https://bit.ly/2Ul97za

[68] Nils Müller and Tobias Glasmachers. 2018. Challenges in High-Dimensional Reinforcement Learning with Evolution Strategies. In *Parallel Problem Solving from Nature – PPSN XV*, Anne Auger, Carlos M. Fonseca, Nuno Lourenço, Penousal Machado, Luís Paquete, and Darrell Whitley (Eds.). Springer International Publishing, Cham, 411–423.

[69] Tsendsuren Munkhdalai and Hong Yu. 2017. Meta networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2554–2563. https://arxiv.org/abs/1703.00837

[70] Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. 2018. Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282* (2018). https://arxiv.org/abs/1810.12282

[71] Philip Paquette. 2017. *DoomTakeCover-v0*. Retrieved January 17, 2020 from https://gym.openai.com/envs/DoomTakeCover-v0/

[72] Niki Parmar, Prajit Ramachandran, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. 2019. Stand-Alone Self-Attention in Vision Models. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*. 68–80. http://papers.nips.cc/paper/8302-stand-alone-self-attention-in-vision-models

[73] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. (2019). https://bit.ly/31PMViq

[74] Sebastian Risi and Kenneth O Stanley. 2012. An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons. *Artificial Life* 18, 4 (2012), 331–363. https://eplex.cs.ucf.edu/papers/risi_alife12.pdf

[75] Sebastian Risi and Kenneth O Stanley. 2013. Confronting the challenge of learning a flexible neural controller for a diversity of morphologies. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. 255–262. https://eplex.cs.ucf.edu/papers/risi_gecco13b.pdf

[76] Sebastian Risi and Kenneth O. Stanley. 2019. Deep neuroevolution of recurrent and discrete world models. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*. 456–462. https://doi.org/10.1145/3321707.3321817

[77] Sebastian Risi and Kenneth O. Stanley. 2020. Improving Deep Neuroevolution via Deep Innovation Protection. *CoRR* abs/2001.01683 (2020). arXiv:2001.01683 http://arxiv.org/abs/2001.01683

[78] Edward Rosten, Gerhard Reitmayr, and Tom Drummond. 2005. Real-time video annotations for augmented reality. In *International Symposium on Visual Computing*. Springer, 294–302. http://www.edrosten.com/work/rosten_2005_annotations.pdf

[79] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. 2013. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 6655–6659. https://bit.ly/39ZEF26

[80] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. 2017. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *Preprint arXiv:1703.03864* (2017). https://arxiv.org/abs/1703.03864

[81] Juergen Schmidhuber. 1993. A 'self-referential' weight matrix. In *International Conference on Artificial Neural Networks*. Springer, 446–450. https://mediatum.ub.tum.de/doc/814784/file.pdf

[82] Juergen Schmidhuber. 1997. Discovering neural nets with low Kolmogorov complexity and high generalization capability. *Neural Networks* 10, 5 (1997), 857–873. ftp://ftp.idsia.ch/pub/juergen/loconet.pdf

[83] Juergen Schmidhuber and Rudolf Huber. 1991. Learning to generate artificial fovea trajectories for target detection. *International Journal of Neural Systems* 2, 01n02 (1991), 125–134. http://people.idsia.ch/~juergen/attentive.html

[84] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. 2019. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *CoRR* abs/1911.08265 (2019). arXiv:1911.08265 http://arxiv.org/abs/1911.08265

[85] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017). https://arxiv.org/abs/1707.06347

[86] Xingyou Song, Yiding Jiang, Stephen Tu, Yilun Du, and Behnam Neyshabur. 2020. Observational Overfitting in Reinforcement Learning. In *International Conference on Learning Representations*. https://openreview.net/forum?id=HJli2hNKDH

[87] Ivan Sorokin, Alexey Seleznev, Mikhail Pavlov, Aleksandr Fedorov, and Anastasiia Ignateva. 2015. Deep Attention Recurrent Q-Network. *ArXiv* abs/1512.01693 (2015). https://arxiv.org/abs/1512.01693

[88] E. S Spelke and K. D. Kinzler. 2007. Core knowledge. *Developmental Science* 10 (2007), 89–96. http://www.wjh.harvard.edu/~lds/pdfs/SpelkeKinzler07.pdf

[89] Kenneth O Stanley. 2007. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines* 8, 2 (2007), 131–162. https://eplex.cs.ucf.edu/papers/stanley_gpem07.pdf

[90] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. 2009. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life* 15, 2 (2009), 185–212. http://eplex.cs.ucf.edu/hyperNEATpage/

[91] Kenneth O Stanley and Risto Miikkulainen. 2003. A taxonomy for artificial embryogeny. *Artificial Life* 9, 2 (2003), 93–130. http://nn.cs.utexas.edu/?stanley:

alife03

[92] Marijn F Stollenga, Jonathan Masci, Faustino Gomez, and Jürgen Schmidhuber. 2014. Deep networks with internal selective attention through feedback connections. In *Advances in neural information processing systems*. 3545–3553. https://arxiv.org/abs/1407.3068

[93] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. 2017. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567* (2017). https://arxiv.org/abs/1712.06567

[94] Gencer Sumbul and Begüm Demir. 2019. A CNN-RNN Framework with a Novel Patch-Based Multi-Attention Mechanism for Multi-Label Image Classification in Remote Sensing. *arXiv preprint arXiv:1902.11274* (2019).

[95] Supasorn Suwajanakorn, Noah Snavely, Jonathan J Tompson, and Mohammad Norouzi. 2018. Discovery of latent 3d keypoints via end-to-end geometric reasoning. In *Advances in Neural Information Processing Systems*. 2059–2070. https://keypointnet.github.io/

[96] Yujin Tang and David Ha. 2019. How to run evolution strategies on Google Kubernetes Engine. *https://cloud.google.com/blog* (2019). https://cloud.google.com/blog/products/ai-machine-learning/how-to-run-evolution-strategies-on-google-kubernetes-engine

[97] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 5998–6008. http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

[98] Johannes von Oswald, Christian Henning, João Sacramento, and Benjamin F. Grewe. 2020. Continual learning with hypernetworks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SJgwNerKvB

[99] Edward Vul, Deborah Hanus, and Nancy Kanwisher. 2009. Attention as inference: selection is probabilistic; responses are all-or-none samples. *Journal of Experimental Psychology: General* 138, 4 (2009), 546. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2822457/

[100] Marek Wydmuch, Michal Kempka, and Wojciech Jaskowski. 2019. ViZDoom Competitions: Playing Doom From Pixels. *IEEE Trans. Games* 11, 3 (2019), 248–259. https://doi.org/10.1109/TG.2018.2877047

[101] Chang Ye, Ahmed Khalifa, Philip Bontrager, and Julian Togelius. 2020. Rotation, Translation, and Cropping for Zero-Shot Generalization. *arXiv preprint arXiv:2001.09908* (2020). https://arxiv.org/abs/2001.09908

[102] Anthony M Zador. 2019. A critique of pure learning and what artificial neural networks can learn from animal brains. *Nature communications* 10, 1 (2019), 1–7. https://www.nature.com/articles/s41467-019-11786-6

[103] Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew Botvinick, Oriol Vinyals, and Peter Battaglia. 2019. Deep reinforcement learning with relational inductive biases. In *International Conference on Learning Representations*. https://openreview.net/forum?id=HkxaFoC9KQ

[104] Amy Zhang, Yuxin Wu, and Joelle Pineau. 2018. Natural environment benchmarks for reinforcement learning. *arXiv preprint arXiv:1811.06032* (2018). https://arxiv.org/abs/1811.06032

[105] Chenyang Zhao, Olivier Siguad, Freek Stulp, and Timothy M Hospedales. 2019. Investigating generalisation in continuous deep reinforcement learning. *arXiv preprint arXiv:1902.07015* (2019). https://arxiv.org/abs/1902.07015