Learning One Abstract Bit at a Time Through Self-invented Experiments Encoded as Neural Networks

Vincent Herrmann[1(✉)], Louis Kirsch[1], and Jürgen Schmidhuber[1,2]

[1] IDSIA/USI/SUPSI, Lugano, Switzerland
{vincent.herrmann,louis.kirsch,juergen}@idsia.ch
[2] AI Initiate, King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia

Abstract. There are two important things in science: (A) Finding answers to given questions, and (B) Coming up with good questions. Our artificial scientists not only learn to answer given questions, but also continually invent new questions, by proposing hypotheses to be verified or falsified through potentially complex and time-consuming experiments, including thought experiments akin to those of mathematicians. While an artificial scientist expands its knowledge, it remains biased towards the simplest, least costly experiments that still have surprising outcomes, until they become boring. We present an empirical analysis of the automatic generation of interesting experiments. In the first setting, we investigate self-invented experiments in a reinforcement-providing environment and show that they lead to effective exploration. In the second setting, pure thought experiments are implemented as the weights of recurrent neural networks generated by a neural experiment generator. Initially interesting thought experiments may become boring over time.

Keywords: Reinforcement Learning · Exploration

1 Introduction and Previous Work

It has been pointed out that there are two important things in science: (A) Finding answers to given questions, and (B) Coming up with good questions, e.g., [2,30,31,42,60,63,65,68]. (A) is arguably just the standard problem of computer science. But how to implement the creative part (B) in artificial systems through reinforcement learning (RL), gradient-based artificial neural networks (NNs), and other machine learning methods?

For at least three decades, work on artificial scientists equipped with artificial curiosity and creativity has been published that addresses this question, e.g., [33, 38,40,42,48,53,57,60,70,72,73]. One early such work is the intrinsic motivation-based **adversarial system** from 1990 [38,42]. It is an artificial Q&A system designed to invent and answer questions. For that, it uses two artificial NNs. The first NN is called the controller C. C probabilistically generates outputs that

C. L. Buckley et al. (Eds.): IWAI 2023, CCIS 1915, pp. 254–274, 2024.
https://doi.org/10.1007/978-3-031-47958-8_16

may influence an environment. The second NN is called the world model M. It predicts the environmental reactions to C's outputs. Using gradient descent, M minimizes its error, thus becoming a better predictor. But in a zero-sum game, the reward-maximizing C tries to find sequences of output actions that maximize the error of M. M's loss is the gain of C (like in the later application of artificial curiosity called GANs [10,64], but also for the more general cases of sequential data and RL [20,74,80]).

C is asking questions through its action sequences: What happens if I do that? M is learning to answer those questions. C is motivated to come up with questions where M does not yet know the answer and loses interest in questions with known answers.

This type of Q&A system helps to understand the world, which is necessary for planning [38,39,42] and may boost external reward [2,31,40,50,52,58]. Clearly, the adversarial approach makes for a fine exploration strategy in many deterministic environments. **In stochastic environments, however, it might fail.** C might learn to focus on those parts of the environment where M can always get high prediction errors due to randomness, or due to computational limitations of M. For example, an agent controlled by C might get stuck in front of a TV screen showing highly unpredictable white noise, e.g., [2,57]. Therefore, in stochastic environments, C's reward should not be the errors of M, but (an approximation of) the *first derivative* of M's errors across subsequent training iterations, that is, M's **learning progress or improvements** [40,54]. As a consequence, despite M's high errors in front of a noisy TV screen, C won't get rewarded for getting stuck there, simply because M's errors won't improve. Both the totally predictable and the fundamentally unpredictable will get boring.

This simple insight led to lots of follow-up work [57]. For example, one particular RL approach for artificial curiosity in stochastic environments was published in 1995 [72]. A simple M learned to predict or estimate the probabilities of the environment's possible responses, given C's actions. After each interaction with the environment, C's intrinsic reward was the KL-Divergence [25] between M's estimated probability distributions before and after the resulting new experience—the **information gain** [72]. This was later also called *Bayesian Surprise* [19]. Compare earlier work on information gain [66] and its maximization *without* RL & NNs [6].

In the general RL setting where the environment is only partially observable [61, Sec. 6], C and M may greatly profit from a memory of previous events [38,39,43]. Towards this end, both C and M can be implemented as LSTMs [7,12,16,61] or Transformers [28,75].

The better the predictions of M, the fewer bits are required to encode the history H of observations because short codes can be used for observations that M considers highly probable [17,83]. That is, the learning progress of M has a lot to do with the concept of *compression progress* [53,55–57]. But it's not quite the same thing. In particular, it does not take into account the bits of information needed to specify M. A more general approach is based on algorithmic information theory, e.g., [22,26,51,69,78,79]. Here C's intrinsic reward is indeed

based on **algorithmic compression progress** [53,55–57] based on some coding scheme for the weights of the model network, e.g., [8,15,23,24,46,47,71], and also a coding scheme for the history of all observations so far, given the model [15,17,34,53,78,83]. Note that the history of science is a history of compression progress through incremental discovery of simple laws that govern seemingly complex observation sequences [53,55–57].

In early systems, the questions asked by C were restricted in the sense that they always referred to all the details of future inputs, e.g., pixels [38,42]. That's why in 1997, a more general adversarial RL machine was built that could ignore many or all of these details and ask **arbitrary abstract questions** with computable answers [48–50]. Example question: if we run this policy (or program) for a while until it executes a special interrupt action, will the internal storage cell number 15 contain the value 5, or not? Again there are two learning, reward-maximising adversaries playing a zero-sum game, occasionally betting on different yes/no outcomes of such computational experiments. The winner of such a bet gets a reward of 1, the loser –1. So each adversary is motivated to come up with questions whose answers surprise the other. And both are motivated to avoid seemingly trivial questions where both already agree on the outcome, or seemingly hard questions that none of them can reliably answer for now. This is the approach closest to what we will present in the following sections.

All the systems above (now often called CM systems [62]) actually maximize the sum of the standard external rewards (for achieving user-given goals) and the intrinsic rewards. **Does this distort the basic RL problem?**

It turns out not so much. Unlike the external reward for eating three times a day, the curiosity reward in the systems above is ephemeral, because once something is known, there is no additional intrinsic reward for discovering it again. That is, the external reward tends to dominate the total reward. In totally learnable environments, in the long run, the intrinsic reward even *vanishes* next to the external reward. Which is nice, because in most RL applications we care only for the external reward.

RL Q&A systems of the 1990s did not **explicitly, formally enumerate their questions.** But the more recent PowerPlay framework (2011) [60,70] does. Let us step back for a moment. What is the set of all formalisable questions? How to decide whether a given question has been answered by a learning machine? To define a question, we need a computational procedure that takes a solution candidate (possibly proposed by a policy) and decides whether it is an answer to the question or not. PowerPlay essentially enumerates the set of all such procedures (or some user-defined subset thereof), thus enumerating all possible questions or problems. **It searches for the simplest question that the current policy cannot yet answer but can quickly *learn* to answer *without* forgetting the answers to previously answered questions.** What is the simplest such Q&A to be added to the repertoire? It is the cheapest one—the one that is found first. Then the next trial starts, where new Q&As may build on previous Q&As.

In our empirical investigation of Sect. 3, we will revisit the above-mentioned concepts of complex computational experiments with yes/no outcomes, focusing on two settings: (1) The generation of experiments driven by model prediction error in a deterministic reinforcement-providing environment, and (2) An approach where C (driven by information gain) generates pure thought experiments in form of weight matrices of RNNs.

2 Self-invented Experiments Encoded as Neural Networks

We present a CM system where C can design essentially arbitrary computational experiments (including thought experiments) with binary yes/no outcomes. Experiments may run for several time steps. However, C will prefer simple experiments whose outcomes still surprise M, until they become boring.

In general, both the controller C and the model M can be implemented as (potentially multi-dimensional) LSTMs [11]. At each time step $t = 1, 2, \ldots$, C's input includes the current sensory input vector $in(t)$, the external reward vector $R_e(t)$, and the intrinsic curiosity reward $R_i(t)$. C may or may not interact directly with the environment through action outputs. How does C ask questions and propose experiments? C has an output unit called the START unit. Once it becomes active (>0.5), C uses a set of extra output units for producing the *weight matrix or program* θ of a separate RNN or LSTM called E (for Experiment), in fast weight programmer style [4,9,18,21,36,37,41,44,45].

E takes sensory inputs from the environment and produces actions as outputs. It also has two additional output units, the HALT unit [59] and the RESULT unit. Once the weights θ are generated at time step t', E is tested in a trial, interacting with some environment. Once E's HALT unit exceeds 0.5 in a later time step t'', the current experiment ends. That is, the experiment computes its own runtime [59]. The experimental outcome $r(t'')$ is 1 if the activation $result(t'')$ of E's RESULT unit exceeds 0.5, and 0 otherwise. At time t', so before the experiment is being executed, M has to compute its output $pr(t') \in [0, 1]$ from θ (and the history of C's inputs and actions up to t', which includes all previous experiments their outcomes). Here, $pr(t')$ models M's (un)certainty that the final binary outcome of the experiment will be 1 (YES) or 0 (NO). Then the experiment is run.

In short, C is proposing an experimental question in form of θ that will yield a binary answer (unless some time limit is reached). M is trying to predict this answer before the experiment is executed. Since E is an RNN and thus a general computer whose weight matrix can implement any program executable on a traditional computer [67], any computable experiment with a binary outcome can be implemented in its weight matrix (ignoring storage limitations of finite RNNs or other computers). That is, by generating an appropriate weight matrix θ, C can ask any scientific question with a computable solution. In other words, C can propose any scientific hypothesis that is experimentally verifiable or falsifiable.

At t'', M's previous prediction $pr(t')$ is compared to the later observed outcome $r(t'')$ of C's experiment (which spans $t''-t'$ time steps), and C's intrinsic curiosity reward $R_i(t'')$ is proportional to M's surprise. To calculate it, we interpret $pr(t')$ as M's estimated probability of $r(t'')$, given the history of observations so far. Then we train M by gradient descent (with regularization to avoid overfitting) for a fixed amount of time to improve all of its previous predictions including the most recent one. This yields an updated version of M called M^*.

In general, M^* will compute a different prediction $PR(t')$ of $r(t'')$, given the history up to $t'-1$. At time t'', the contribution $R_{IG}(t'')$ to C's curiosity reward is proportional to the apparent resulting information gain, the KL-divergence

$$R_{IG}(t'') \sim D_{KL}\big(PR(t')||pr(t')\big).$$

If M had a confident belief in a particular experimental outcome, but this belief gets shattered in the wake of C's experiment, there will be a major surprise and a big insight for M, as well as lots of intrinsic curiosity reward for C. On the other hand, if M was quite unsure about the experimental outcome, and remains quite unsure afterwards, then C's experiment can hardly surprise M and C will fail to profit much. C is motivated to propose *interesting* hypotheses or experiments that violate M's current deep beliefs and expand its horizon. An alternative intrinsic curiosity reward would be based on compression progress [53,55–57].

Note that the entire experimental protocol is the responsibility of θ. Through θ, E must initialize the experiment (e.g., by resetting the environment or moving the agent to some start position if that is important to obtain reliable results), then run the experiment by executing a sequence of computational steps or actions, and translate the incoming data sequence into some final abstract binary outcome YES or NO.

C is motivated to design experimental protocols θ that surprise M. C will get bored by experiments whose outcomes are predicted by M with little confidence (recall the noisy TV), as well as by experiments whose outcomes are correctly predicted by M with high confidence. *C will get rewarded for surprising experiments whose outcomes are incorrectly predicted by M with high confidence.*

A negative reward per time step encourages C to be efficient and lazy and come up with simple and fast still surprising experiments. If physical actions in the environment cost much more energy (resulting in immediate negative reward) than E's internal computations per time step, C is motivated to propose a θ defining a "thought experiment" requiring only internal computations, without executing physical actions in the (typically non-differentiable) environment. In fact, due to C's bias towards the computationally cheapest and least costly experiments that are still surprising to M, most of C's initial experiments may be thought experiments. Hence, since C, E and M are differentiable, not only M but also C may be often trainable by backpropagation [4] rather than the generally slower policy gradient methods [1,29,77,81]. Of course, this is only true if the reward function is also differentiable with respect to C's parameters.

3 Experimental Evaluation

Here we present initial studies of the automatic generation of interesting experiments encoded as NNs. We evaluate these systems empirically and discuss the associated challenges. This includes two setups: (1) Adversarial intrinsic reward encourages experiments executed in a differentiable environment through sequences of continuous control actions. We demonstrate that these experiments aid the discovery of goal states in a sparse reward setting. (2) Pure thought experiments encoded as RNNs (without any environmental interactions) are guided by an information gain reward.

Together, these two setups cover the important aspects discussed in Sect. 2: the use of abstract experiments with binary outcomes as a method for curious exploration, and the creation of interesting pure thought experiments encoded as RNNs. We leave the integration of both setups into a single system (as described in Sect. 2) for future work.

3.1 Generating Experiments in a Differentiable Environment

Reinforcement learning (RL) usually involves exploration in an environment with non-differentiable dynamics. This requires RL methods such as policy gradients [82]. To simplify our investigation and focus solely on the generation of self-invented experiments, we introduce a fully differentiable environment that allows for computing analytical policy gradients via backpropagation. This does not limit the generality of our approach, as standard RL methods can be used instead.

Our continuous force field environment is depicted in Fig. 1. The agent has to navigate through a 2D environment with a fixed external force field. This force field can have different levels of complexity. The states in this environment are the position and velocity of the agent. The agent's actions are real-valued force vectors applied to itself. To encourage laziness and a bias towards simple experiments, each time step is associated with a small negative reward (-0.1). A sparse large reward (100) is given whenever the agent gets very close to the goal state. We operate in the single life setting without episodic resets. Additional information about the force field environment can be found in Appendix A. Since the environment is deterministic, it is sufficient for C to generate experiments whose results the current M cannot predict.

Method. Algorithm 1 and Fig. 2 summarize the process for generating a sequence of interesting abstract experiments with binary outcomes. The goal is to test the following three hypotheses:

- Generated experiments implement exploratory behavior, facilitating the reaching of goal states.
- If there are negative rewards in proportion to the runtime of experiments, then the average runtime will increase over time, as the controller will find it harder and harder to come up with new short experiments whose outcomes the model cannot yet predict.

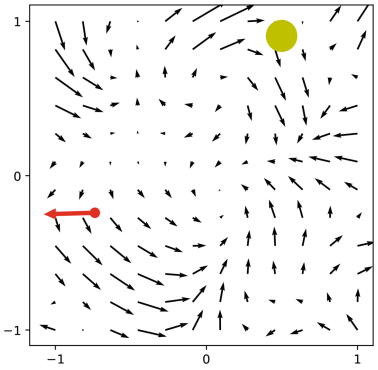


Fig. 1. A differentiable force field environment. The agent (red) has to navigate to the goal state (yellow) while the external force field exerts forces on the agent. (Color figure online)

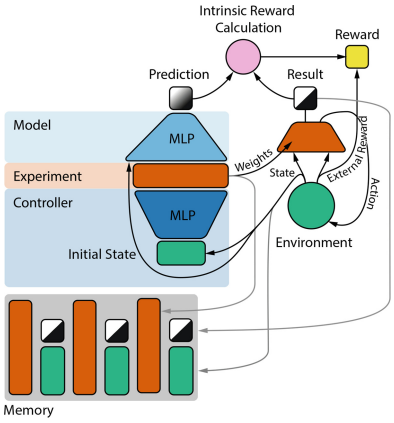


Fig. 2. Generating self-invented experiments in a differentiable environment. A controller C_ϕ is motivated to generate experiments E_θ that still surprise the model $M_{\mathbf{w}}$. After execution in the environment, the experiments and their binary results are stored in memory. The model is trained on the history of previous experiments.

– As the model learns to predict the yes/no results of more and more experiments, it becomes harder for the controller to create experiments whose outcomes surprise the model.

The generated experiments have the form $E_\psi(s) = (a, \hat{r})$, where E_ψ is a linear feedforward network with parameters ψ, s is the environment state, a are the actions and $\hat{r} \in [0, 1]$ is the experimental result. Both s and a are real-valued vectors.

Instead of a HALT unit, a single scalar $\tau \in \mathbb{R}^+$ determines the number of steps for which an experiment will run. To further simplify the setup, the experiment network is a feedforward NN without recurrence. To make the exper-

imental result differentiable with respect to the runtime parameter, τ predicts the mean of a Gaussian distribution with fixed variance over the number of steps. The actual result $\tilde{r}$ is the expectation of the result unit $\hat{r}$ over the distribution defined by τ (more details on this can be found in Appendix A.1). The binarized result r has the value 1 if $\tilde{r} > 0.5$, and 0 otherwise. The parameters θ of the experiment are the network parameters ψ together with the runtime parameter τ, i.e. $\theta := (\psi, \tau)$.

For a given starting state s, the controller C_ϕ generates experiments: $C_\phi(s) = \theta$. C_ϕ is a multi-layer perceptron (MLP) with parameters ϕ, and θ denotes the parameters of the generated experiment. The model $M_\mathbf{w}$ is an MLP with parameters $\mathbf{w}$. It makes a prediction $M_\mathbf{w}(s, \theta) = \hat{o}$, with $\hat{o} \in [0, 1]$, for an experiment defined by the starting state s and the parameters θ.

During each iteration of the algorithm, C_ϕ generates an experiment based on the current state s of the environment. This experiment is executed until the cumulative halting probability defined by the generated τ exceeds a certain threshold (e.g., 99%). The starting state s, experiment parameters θ and binary result r are saved in a memory buffer $\mathcal{D}$ of experiments. Every state encountered during the experiment is saved to the state memory buffer $\mathcal{B}$.

After the experiment execution, the model $M_\mathbf{w}$ is trained for a fixed number of steps of stochastic gradient descent (SGD) to minimize the loss

$$\mathcal{L}_M = \mathbb{E}_{(s,\theta,r)\sim\mathcal{D}}[\mathrm{bce}(M_\mathbf{w}(s, \theta), r)], \tag{1}$$

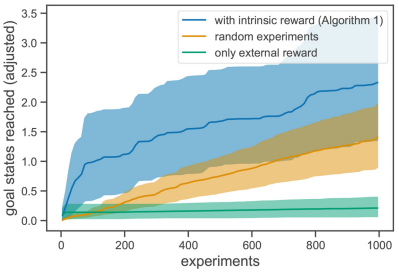
where bce is the binary cross-entropy loss function.

The third and last part of each iteration is the training of the controller C_ϕ. The loss that is being minimized via SGD is

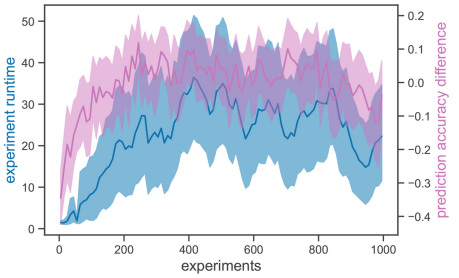
$$\mathcal{L}_C = \mathbb{E}_{s\sim\mathcal{B}}[-\mathrm{bce}\big(M_\mathbf{w}(s, C_\phi(s)), \tilde{r}(C_\phi(s), s)\big) - R_e(C_\phi(s), s)]. \tag{2}$$

The function $\tilde{r}$ maps the experiment parameters and starting state to the continuous result of the experiment. The function R_e maps the experiment parameters and starting state to the external reward. Note that gradient information will flow back from $\tilde{r}$ and R to ϕ through the execution of the experiment in the differentiable environment. The first term corresponds to the intrinsic reward for the controller, which encourages it to generate experiments whose outcomes $M_\mathbf{w}$ cannot predict. The second term is the external reward from the environment, which punishes long experiments. Since the reward for reaching the goal is sparse and not differentiable with respect to the experiment's actions, no information about the goal state reaches C_ϕ through the gradient.

Results and Discussion. To investigate our first hypothesis, Fig. 3a shows the cumulative number of times a goal state was reached during an experiment, adjusted by the number of environment interactions of each experiment. Specifically, it shows $h(j) = \sum_{k=1}^{j} \frac{g_k}{n_k}$, where $j = 1, 2, \ldots$ is the index of the generated experiment, g_k is 1 if the goal state was reached during the kth experiment and 0 otherwise, and n_k is the runtime of the kth experiment. Our method,



(a) Number of times the goal state was reached, adjusted by the number of environment interactions. Experiments generated with adversarial intrinsic reward benefit exploration more than random experiments. Without intrinsic motivation, the agent usually fails to reach any goal states in the sparse reward setting. Mean with bootstrapped 95% confidence intervals across 30 seeds.



(b) Blue: the average runtime of experiments generated by C_ϕ. Purple: the difference between result prediction accuracy of the current $M_\mathbf{w}$ for the generated experiment and the average prediction accuracy of the current $M_\mathbf{w}$ for random experiments. As it gets harder for C_ϕ to generate experiments θ that are surprising to $M_\mathbf{w}$ (hard to predict), the runtime increases and the experiments tend to be harder to predict than the average randomly drawn experiment. Mean with bootstrapped 95% confidence intervals across 30 seeds.

Fig. 3. Experiments in the differentiable force field environment

as described above and in Algorithm 1, reaches the most goal states per environment interaction. Purely random experiments also discover goal states, but less frequently. Note that such random exploration in parameter space has been shown to be a powerful exploration strategy [32,35,76]. The average runtime of the random experiments is 50 steps, compared to 22.9 for the experiments generated by C_ϕ. To rule out a potential unfair bias due to different runtimes, Fig. 6 in the Appendix shows an additional baseline of random experiments with an average runtime of 20 steps, leading to results very similar to those of longer running random experiments. If we remove the intrinsic adversarial reward, the controller is left only with the external reward. This means that there is no bce term in Eq. 2. It is not surprising that in this setting, C_ϕ fails to generate experiments that discover goal states, since the gradient of $\mathcal{L}_C$ contains no information about the sparse goal reward.

Figure 3b addresses our second and third hypotheses. C_ϕ indeed tends to prolong experiments as $M_\mathbf{w}$ has been trained on more experiments, even though experiments with long runtimes are discouraged through the punitive external reward. Our explanation for this is that it becomes harder with time for C_ϕ to come up with short experiments for which $M_\mathbf{w}$ cannot yet accurately predict the correct results. This is supported by the fact that the prediction accuracy of $M_\mathbf{w}$ for newly generated experiments goes up. Specifically, Fig. 3b shows the difference between prediction accuracy of the current $M_\mathbf{w}$ for the newly generated experiment and the expected prediction accuracy of the current $M_\mathbf{w}$ for experiments randomly sampled from a simple prior. This accounts for the general gain of $M_\mathbf{w}$'s prediction accuracy over the course of training. It can be seen that in the beginning, C_ϕ is successful at creating adversarial experiments that surprise

$M_{\mathbf{w}}$. With time, however, it fails to continue doing so and is forced to create longer experiments to challenge $M_{\mathbf{w}}$.

3.2 Pure RNN Thought Experiments

The previous experimental setup uses feedforward NNs as experiments and an intrinsic reward function that is differentiable with respect to the controller's weights. This section investigates a complementary setup: interesting pure thought experiments (with no environment interactions) are generated in the form of RNNs without any inputs, driven by an intrinsic curiosity reward based on information gain which we treat as non-differentiable.

Algorithm 1. Adversarial yes/no experiments in a differentiable environment

Input: Randomly initialized differentiable Controller $C_\phi : \mathrm{S} \to \Theta$, randomly initialized differentiable Model $M_{\mathbf{w}} : \mathrm{S} \times \Theta \to \mathbb{R}$, empty experiment memory $\mathcal{D}$, empty state memory $\mathcal{B}$, set of random initial experiments $\mathcal{E}_{\text{init}}$, Differentiable environment
Output: An experiment memory populated with (formerly) interesting experiments

for $\theta \in \mathcal{E}_{\text{init}}$ **do**
  $s \leftarrow$ current environment state
  Execute the experiment parametrized by θ in the environment, obtain binary result r
  Save the tuple (s, θ, r) to $\mathcal{D}$
  Save all encountered states during the experiment to $\mathcal{B}$
end for
repeat
  $s \leftarrow$ current environment state
  $\theta \leftarrow C_\phi(s)$
  Execute the experiment parametrized by θ in the environment, obtain binary result r
  Save tuple (s, θ, r) to $\mathcal{D}$
  $\hat{s} \leftarrow$ current environment state
  for some steps **do**
    Sample tuple (s, θ, r) from $\mathcal{D}$
    Update the model using SGD: $\nabla_{\mathbf{w}} \mathrm{bce}(M_{\mathbf{w}}(s, \theta), r)$
  end for
  for some steps **do**
    Sample starting state s from $\mathcal{B}$
    Set environment to state s
    Execute the experiment parametrized by $C_\phi(s)$ in the environment, obtain continuous result $\tilde{r}$ and external reward R_e
    Update the controller using SGD: $\nabla_\phi \big(- \mathrm{bce}(M_{\mathbf{w}}(s, C_\phi(s)), \tilde{r}) - R_e \big)$
  end for
  Set environment to state $\hat{s}$
until no more interesting experiments are found

Method. In many ways, this new setup (depicted in Fig. 4 and described in Algorithm 2 in the Appendix) is similar to the one presented in Sect. 3.1. In what follows, we highlight the important differences.

An experiment E_θ is an RNN of the form $(h_{t+1}, r_{t+1}, \gamma_{t+1}) = E_\theta(h_t)$, where h_t is the hidden state vector, $r_t \in \{0, 1\}$ is the binary result at experiment time step t, and $\gamma_t \in [0, 1]$ is the HALT unit. The result r of E_θ is the r_t for the experiment step t where γ_t first is larger than 0.5. Since there is no external environment and the experiments are independent of each other, the model $M_\mathbf{w}$ is again a simple MLP with parameters $\mathbf{w}$. It takes only the experiment parameters θ as input and makes a result prediction $\hat{o} = M_\mathbf{w}(\theta), \hat{o} \in [0, 1]$.

As mentioned above, here we treat the intrinsic reward signal as non-differentiable. This means that—in contrast to the method presented in Sect. 3.1—the controller cannot receive information about $M_\mathbf{w}$ from gradients that are backpropagated through the model. Instead, it has to infer the learning behavior of $M_\mathbf{w}$ from the history ω of previous experiments and intrinsic rewards to come up with new surprising experiments. The controller C_ϕ is now an LSTM that is trained by DDPG [27] and generates new experiments solely based on the history of past experiments: $C_\phi(\omega) = \theta$. The history ω is a sequence of tuples (θ_i, r_i, R_i), where $i = 1, 2, \ldots$ is the index of the experiment. It contains experiments up to the last one that has been executed. More details on the training of $M_\mathbf{w}$ and the algorithm can be found in Appendix B.

For these pure thought experiments, we use a reward based on information gain. Let $\mathbf{w}$ be M's weights at certain point in time. Then a new experiment with parameters θ is generated, executed and saved to the buffer. On this buffer $\mathcal{D}$, which now includes θ, M is trained for a fixed number of SGD steps to obtain new weights $\mathbf{w}^*$. Then, the information gain reward associated with experiment θ is

$$R_{IG}(\theta, \mathbf{w}, \mathbf{w}^*) = \frac{1}{|\mathcal{D}|} \sum_{\tilde{\theta} \in \mathcal{D}} D_{KL}(M_{\mathbf{w}^*}(\tilde{\theta}) || M_\mathbf{w}(\tilde{\theta})), \tag{3}$$

where we interpret the output of the model as a Bernoulli distribution.

Results and Discussion. Figure 5 shows the information gain reward associated with each new experiment that C_ϕ generates. We observe that, after a short initial phase, the intrinsic information gain reward steadily declines. This is similar to what we observe for the prediction accuracy in Sect. 3.1: it becomes harder for the controller to generate experiments that surprise the model. It should be mentioned that this is a natural effect, since—as the model is trained on more and more experiments—every new additional experiment contributes on average less to the model's change during training, and thus is associated with less information gain reward. An interesting, albeit minor, effect shown in Fig. 5 is that also in this setup, the average runtime of the generated experiments increases slightly over time, even though there is no negative reward for longer thought experiments. For shorter experiments, however, it is apparently easier for the model to learn to predict the results. Hence, at least in the beginning,

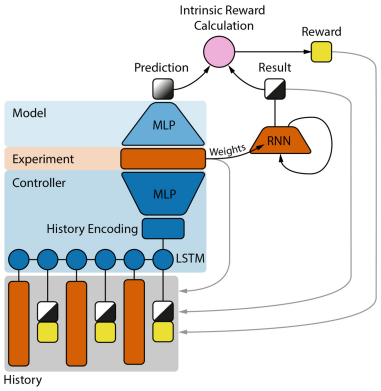


Fig. 4. Generating abstract thought experiments encoded as RNNs. The model is trained to predict the results of previous experiments. The controller generates new interesting thought experiments (without environment interactions) based on the history of previous experiments, their results and rewards.

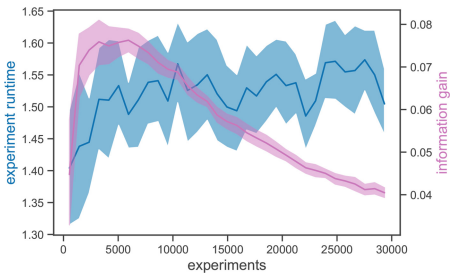


Fig. 5. Empirical results for pure thought experiments encoded as RNNs. Blue: the average runtime of each experiment generated by C_ϕ. Purple: information gain reward (Eq. 3) for C_ϕ associated with each experiment. Mean with bootstrapped 95% confidence intervals across 20 seeds.

they yield more learning progress and more information gain. Later, however, longer experiments become more interesting.

In comparison to the experiments generated in Sect. 3.1, the present ones have a much shorter runtime. This is a side-effect of the experiments being RNNs with a HALT unit; for randomly initialized experiments, the average runtime is approximately 1.6 steps.

4 Conclusion and Future Work

We extended the neural Controller-Model (CM) framework through the notion of arbitrary self-invented computational experiments with binary outcomes: experimental protocols are essentially programs interacting with the environment, encoded as the weight matrices of RNNs generated by the controller. The model has to predict the outcome of an experiment based solely on the experiment's parameters. By creating experiments whose outcomes surprise the model, the controller curiously explores its environment and what can be done in it. Such a system is analogous to a scientist who designs experiments to gain insights about the physical world. However, an experiment does not necessarily involve actions taken in the environment: it may be a pure thought experiment akin to those of mathematicians.

We provide an empirical evaluation of two simple instances of such systems, focusing on different and complementary aspects of the idea. In the first setup,

we show that self-invented abstract experiments encoded as feedforward networks interacting with a continuous control environment facilitate the discovery of rewarding goal states. Furthermore, we see that over time the controller is forced to create longer experiments (even though this is associated with a larger negative external reward) as short experiments start failing to surprise the model. In the second setup, the controller generates pure abstract thought experiments in the form of RNNs. We observe that over time, newly generated experiments result in less intrinsic information gain reward. Again, later experiments tend to have slightly longer runtime. We hypothesize that this is because simple experiments initially lead to a lot of information gain per time interval, but later do not provide much insight anymore.

These two empirical setups should be seen as initial steps towards more capable systems such as the one proposed in Sect. 2. Scaling these methods to more complex environments and the generation of more sophisticated experiments, however, is not without challenges. Direct generation and interpretation of NN weights may not be very effective for large and deep networks. Previous work [3] already combined hypernetworks [13] and policy fingerprinting [5,14] to generate and evaluate policies. Similar innovations will facilitate the generation of abstract self-invented experiments beyond the small scale setups presented in this paper.

A Experiments in the Force Field Environment

The force field of the environment is based on a 2D grid of randomly sampled force vectors. To get a continuous force field, bicubic interpolation between the vectors of the grid is used. Hence, the resolution of the grid influences the complexity of the force field (higher resolution $\rightarrow$ more intricate force field). In all experiments, the grid resolution is sampled uniformly from $\{(3,3),(5,5),(7,7)\}$. The random seed of each run affects both the force field and the position of the goal state. This means that every run has its own unique environment.

A.1 Experiment Execution

Let $\hat{r}_t \in [0,1]$ be the value of the result node at step t of the experiment whose runtime is determined by the parameter $\tau \in [0,100]$. The maximum runtime is fixed to 100 steps. A distribution over experiment steps t is defined by τ as follows: $p_\tau(t) = \frac{\exp(-0.5(t-\tau)^2)}{\sum_{u=1}^{100}\exp(-0.5(u-\tau)^2)}$.

The continuous result of the experiment is the expectation of the result unit over this distribution: $\tilde{r} = \mathbb{E}_{t\sim p_\tau}\hat{r}_t$. The binary result of the experiment r is the boolean value $\tilde{r} > 0.5$.

A.2 Hyperparameters for the Force Field Experiments

Table 1 shows the hyperparameters for Algorithm 1. The output nodes of C_ϕ that generate the parameters ψ of the experiment network have a tanh output

nonlinearity and are then scaled to the predefined range. The output node that generates τ is clipped to the range $[0, 100]$.

The experiment parameters for random baselines are generated as $\psi = 2\tanh(v)$, where $v \sim \mathcal{N}(0, 4I)$. The runtime parameter τ is sampled uniformly from the allowed range. The hyperparameters for the model are the same as in Table 1. The baseline with only external reward also uses the hyperparameters of Table 1. The difference is that in this setting, the loss of the C_ϕ is simply $\mathcal{L}_C = \mathbb{E}_{s\sim\mathcal{B}}[-R(C_\phi(s), s)]$ instead of the one defined in Eq. 2.

Table 1. Hyperparameters for Algorithm 1

Hyperparameter	Value
hidden layers $M_{\mathbf{w}}$	[128, 128, 128, 128]
hidden layers C_ϕ	[128, 128, 128, 128]
training steps per iteration $M_{\mathbf{w}}$	100
training steps per iteration C_ϕ	100
learning rate $M_{\mathbf{w}}$	0.0003
learning rate C_ϕ	0.0003
weight decay $M_{\mathbf{w}}$	0.01
weight decay C_ϕ	0.01
experiment parameter range	[–2, 2]
noise input nodes C_ϕ	8
environment grid resolutions	[(3, 3), (5, 5), (7, 7)]
number of iterations	1000
number of initial experiments in $\mathcal{E}_{\text{init}}$	100

A.3 Additional Results

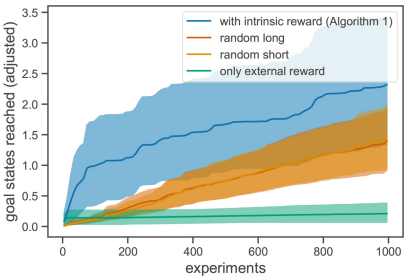


Fig. 6. Similar to Fig. 3a, but with an additional baseline of short random experiments with an average runtime of 20 steps.

To account for a potential bias due to experimental runtime, Fig. 6 shows the adjusted number of goal states for a baseline of shorter random experiments.

B Pure Thought Experiments

Algorithm 2 summarizes the method described in Sect. 3.2. In this setup, the model $M_{\mathbf{w}}$ is trained to minimize the following loss:

$$\mathcal{L}_M = \mathbb{E}_{(\theta,r)\sim\mathcal{D}}[\mathrm{bce}(M_{\mathbf{w}}(\theta), r)]. \tag{4}$$

Efficient approximation of the policy gradients for the controller is achieved through an actor-critic method, specifically DDPG [27]. The controller C_ϕ has an additional LSTM encoder that generates a vector-sized representation of the history ω of previous experiments, their results and the reward associated with them. The actor is an MLP that receives as input the history representation created by the LSTM and generates the weights of an experiment RNN, whereas the critic receives both a history representation and experiment weights as input, and outputs a scalar reward estimation. Actor and critic share the same LSTM history encoder and take alternating gradient descent steps during training. The input to the LSTM history encoder is the sequence ω of the last 1000 that have been executed.

The experiment RNNs E_θ used in this empirical evaluation have 3 hidden units and no inputs. The initial hidden state h_0 is treated as part of the parameters θ and is thus also generated by C_ϕ. Random experiments are sampled the same way as described in Sect. A.2. All other hyperparameters are listed in Table 2.

Table 2. Hyperparameters for Algorithm 2

Hyperparameter	Value
hidden layers $M_{\mathbf{w}}$	[128, 128, 128, 128]
hidden layers C_ϕ LSTM	[64]
hidden layers C_ϕ MLP	[128, 128, 128, 128]
training steps per iteration $M_{\mathbf{w}}$	50
training steps per iteration C_ϕ	10
learning rate $M_{\mathbf{w}}$	0.0001
learning rate C_ϕ	0.0001
weight decay $M_{\mathbf{w}}$	0.01
weight decay C_ϕ	0.01
experiment parameter range	[–3, 3]
number of iterations	30000
number of initial experiments in $\mathcal{E}_{\text{init}}$	100

Algorithm 2. Pure thought experiments encoded by RNNs

Input: Randomly initialized differentiable Controller $C_\phi : \Omega \to \Theta$, where Ω is the set of sequences of the form $(\theta_i, r_i, R_i, \theta_{i+1}, r_{i+1}, R_{i+1}, \ldots)$, randomly initialized differentiable Model $M_{\mathbf{w}} : \Theta \to \mathbb{R}$, empty sequential experiment memory $\mathcal{D}$, set of random initial experiments $\mathcal{E}_{\text{init}}$

Output: An experiment memory populated with (formerly) interesting pure thought experiments

for $\theta \in \mathcal{E}_{\text{init}}$ **do**
  Execute the RNN thought experiment parametrized by θ, obtain binary result r
  Save the tuple (θ, r) to $\mathcal{D}$
  Train $M_{\mathbf{w}}$ on data from $\mathcal{D}$ for a fixed number of steps minimizing Equation 4 to obtain updated weights $\mathbf{w}^*$
  Calculate the intrinsic reward $R_i = R_{IG}(\theta, \mathbf{w}, \mathbf{w}^*)$ (Equation 3)
  $\mathbf{w} \leftarrow \mathbf{w}^*$
  Save R_i to $\mathcal{D}$
end for
repeat
  $\omega \leftarrow$ sequence of the last experiments from $\mathcal{D}$
  $\theta \leftarrow C_\phi(\omega)$
  Execute the RNN thought experiment parametrized by θ, obtain binary result r
  Train $M_{\mathbf{w}}$ on data from $\mathcal{D}$ for a fixed number of steps to obtain updated weights $\mathbf{w}^*$
  Calculate the intrinsic reward $R_i = R_{IG}(\theta, \mathbf{w}, \mathbf{w}^*)$
  $\mathbf{w} \leftarrow \mathbf{w}^*$
  Save R_i to $\mathcal{D}$
  Train C_ϕ for a fixed number of steps with DDPG to maximize the expected intrinsic reward
until no more interesting experiments are found

References

1. Berner, C., et al.: Dota 2 with large scale deep reinforcement learning. CoRR abs/1912.06680 (2019). http://arxiv.org/abs/1912.06680
2. Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., Efros, A.A.: Large-scale study of curiosity-driven learning. Preprint arXiv:1808.04355 (2018)
3. Faccio, F., Herrmann, V., Ramesh, A., Kirsch, L., Schmidhuber, J.: Goal-conditioned generators of deep policies. arXiv preprint arXiv:2207.01570 (2022)
4. Faccio, F., Kirsch, L., Schmidhuber, J.: Parameter-based value functions. Preprint arXiv:2006.09226 (2020)
5. Faccio, F., Ramesh, A., Herrmann, V., Harb, J., Schmidhuber, J.: General policy evaluation and improvement by learning to identify few but crucial states. arXiv preprint arXiv:2207.01566 (2022)
6. Fedorov, V.V.: Theory of Optimal Experiments. Academic Press, Cambridge (1972)
7. Gers, F.A., Schmidhuber, J., Cummins, F.: Learning to forget: continual prediction with LSTM. Neural Comput. **12**(10), 2451–2471 (2000)
8. Gomez, F., Koutník, J., Schmidhuber, J.: Compressed network complexity search. In: Coello, C.A.C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012. LNCS, vol. 7491, pp. 316–326. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32937-1_32
9. Gomez, F., Schmidhuber, J.: Evolving modular fast-weight networks for control. In: Duch, W., Kacprzyk, J., Oja, E., Zadrożny, S. (eds.) ICANN 2005. LNCS, vol. 3697, pp. 383–389. Springer, Heidelberg (2005). https://doi.org/10.1007/11550907_61
10. Goodfellow, I., et al.: Generative adversarial nets. In: Advances in Neural Information Processing Systems (NIPS), pp. 2672–2680, December 2014
11. Graves, A., Fernández, S., Schmidhuber, J.: Multi-dimensional recurrent neural networks. In: Proceedings of the 17th International Conference on Artificial Neural Networks, September 2007
12. Graves, A., Liwicki, M., Fernandez, S., Bertolami, R., Bunke, H., Schmidhuber, J.: A novel connectionist system for improved unconstrained handwriting recognition. IEEE Trans. Pattern Anal. Mach. Intell. **31**(5) (2009)
13. Ha, D., Dai, A., Le, Q.V.: Hypernetworks. arXiv preprint arXiv:1609.09106 (2016)
14. Harb, J., Schaul, T., Precup, D., Bacon, P.L.: Policy evaluation networks. arXiv preprint arXiv:2002.11833 (2020)
15. Hochreiter, S., Schmidhuber, J.: Flat minima. Neural Comput. **9**(1), 1–42 (1997)
16. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997). based on TR FKI-207-95, TUM (1995)
17. Huffman, D.A.: A method for construction of minimum-redundancy codes. Proc. IRE **40**, 1098–1101 (1952)
18. Irie, K., Schlag, I., Csordás, R., Schmidhuber, J.: Going beyond linear transformers with recurrent fast weight programmers. Adv. Neural Inf. Process. Syst. **34**, 7703–7717 (2021)
19. Itti, L., Baldi, P.F.: Bayesian surprise attracts human attention. In: Advances in Neural Information Processing Systems (NIPS), vol. 19, pp. 547–554. MIT Press, Cambridge, MA (2005)
20. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: a survey. J. AI Res. **4**, 237–285 (1996)
21. Kirsch, L., Schmidhuber, J.: Meta learning backpropagation and improving it. Adv. Neural Inf. Process. Syst. **34**, 14122–14134 (2021)

22. Kolmogorov, A.N.: Three approaches to the quantitative definition of information. Probl. Inf. Transm. **1**, 1–11 (1965)
23. Koutník, J., Gomez, F., Schmidhuber, J.: Evolving neural networks in compressed weight space. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, pp. 619–626 (2010)
24. Koutník, J., Cuccu, G., Schmidhuber, J., Gomez, F.: Evolving large-scale neural networks for vision-based reinforcement learning. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 1061–1068. ACM, Amsterdam, July 2013
25. Kullback, S., Leibler, R.A.: On information and sufficiency. Ann. Math. Stat. 79–86 (1951)
26. Li, M., Vitányi, P.M.B.: An Introduction to Kolmogorov Complexity and its Applications (2nd edition). Springer, New York (1997). https://doi.org/10.1007/978-1-4757-2606-0
27. Lillicrap, T.P., et al.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015)
28. Micheli, V., Alonso, E., Fleuret, F.: Transformers are sample efficient world models. arXiv preprint arXiv:2209.00588 (2022)
29. Andrychowicz, O.M., et al.: Learning dexterous in-hand manipulation. Int. J. Robot. Res. **39**(1), 3–20 (2020)
30. Oudeyer, P.-Y., Baranes, A., Kaplan, F.: Intrinsically motivated learning of real-world sensorimotor skills with developmental constraints. In: Baldassarre, G., Mirolli, M. (eds.) Intrinsically Motivated Learning in Natural and Artificial Systems, pp. 303–365. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-32375-1_13
31. Pathak, D., Agrawal, P., Efros, A.A., Darrell, T.: Curiosity-driven exploration by self-supervised prediction. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 16–17 (2017)
32. Plappert, M., et al.: Parameter space noise for exploration. arXiv preprint arXiv:1706.01905 (2017)
33. Ramesh, A., Kirsch, L., van Steenkiste, S., Schmidhuber, J.: Exploring through random curiosity with general value functions. In: Oh, A.H., Agarwal, A., Belgrave, D., Cho, K. (eds.) Advances in Neural Information Processing Systems (2022)
34. Rissanen, J.: Modeling by shortest data description. Automatica **14**, 465–471 (1978)
35. Rückstieß, T., Felder, M., Schmidhuber, J.: State-dependent exploration for policy gradient methods. In: Daelemans, W., Goethals, B., Morik, K. (eds.) ECML PKDD 2008. LNCS (LNAI), vol. 5212, pp. 234–249. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-87481-2_16
36. Schlag, I., Irie, K., Schmidhuber, J.: Linear transformers are secretly fast weight programmers. In: International Conference on Machine Learning, pp. 9355–9366. PMLR (2021)
37. Schlag, I., Schmidhuber, J.: Learning to reason with third order tensor products. In: Advances in Neural Information Processing Systems (NIPS), pp. 9981–9993 (2018)
38. Schmidhuber, J.: Making the world differentiable: On using fully recurrent self-supervised neural networks for dynamic reinforcement learning and planning in non-stationary environments. Technical report FKI-126-90 (1990). http://people.idsia.ch/~juergen/FKI-126-90_(revised)bw_ocr.pdf, Tech. Univ. Munich

39. Schmidhuber, J.: An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. In: Proceedings of the IEEE/INNS International Joint Conference on Neural Networks, San Diego, vol. 2, pp. 253–258 (1990)
40. Schmidhuber, J.: Curious model-building control systems. In: Proceedings of the International Joint Conference on Neural Networks, Singapore, vol. 2, pp. 1458–1463. IEEE press (1991)
41. Schmidhuber, J.: Learning temporary variable binding with dynamic links. In: Proceedings of the International Joint Conference on Neural Networks, Singapore, vol. 3, pp. 2075–2079. IEEE (1991)
42. Schmidhuber, J.: A possibility for implementing curiosity and boredom in model-building neural controllers. In: Meyer, J.A., Wilson, S.W. (eds.) Proc. of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats, pp. 222–227. MIT Press/Bradford Books (1991)
43. Schmidhuber, J.: Reinforcement learning in Markovian and non-Markovian environments. In: Lippman, D.S., Moody, J.E., Touretzky, D.S. (eds.) Advances in Neural Information Processing Systems, vol. 3 (NIPS 3), pp. 500–506. Morgan Kaufmann (1991)
44. Schmidhuber, J.: Learning to control fast-weight memories: an alternative to recurrent nets. Neural Comput. **4**(1), 131–139 (1992)
45. Schmidhuber, J.: On decreasing the ratio between learning complexity and number of time-varying variables in fully recurrent nets. In: Proceedings of the International Conference on Artificial Neural Networks, Amsterdam, pp. 460–463. Springer (1993)
46. Schmidhuber, J.: Discovering solutions with low Kolmogorov complexity and high generalization capability. In: Prieditis, A., Russell, S. (eds.) Machine Learning: Proceedings of the Twelfth International Conference, pp. 488–496. Morgan Kaufmann Publishers, San Francisco, CA (1995)
47. Schmidhuber, J.: Discovering neural nets with low Kolmogorov complexity and high generalization capability. Neural Netw. **10**(5), 857–873 (1997)
48. Schmidhuber, J.: What's interesting? Technical report IDSIA-35-97, IDSIA (1997). ftp://ftp.idsia.ch/pub/juergen/interest.ps.gz; extended abstract in Proc. Snowbird'98, Utah, 1998; see also [50]
49. Schmidhuber, J.: Artificial curiosity based on discovering novel algorithmic predictability through coevolution. In: Angeline, P., Michalewicz, Z., Schoenauer, M., Yao, X., Zalzala, Z. (eds.) Congress on Evolutionary Computation, pp. 1612–1618. IEEE Press (1999)
50. Schmidhuber, J.: Exploring the predictable. In: Ghosh, A., Tsuitsui, S. (eds.) Advances in Evolutionary Computing, pp. 579–612. Springer, Berlin, Heidelberg (2003). https://doi.org/10.1007/978-3-642-18965-4_23
51. Schmidhuber, J.: Hierarchies of generalized Kolmogorov complexities and nonenumerable universal measures computable in the limit. Int. J. Found. Comput. Sci. **13**(4), 587–612 (2002)
52. Schmidhuber, J.: Overview of artificial curiosity and active exploration, with links to publications since 1990 (2004). http://www.idsia.ch/~juergen/interest.html
53. Schmidhuber, J.: Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. Connect. Sci. **18**(2), 173–187 (2006)
54. Schmidhuber, J.: Simple algorithmic principles of discovery, subjective beauty, selective attention, curiosity & creativity. In: Corruble, V., Takeda, M., Suzuki, E. (eds.) DS 2007. LNCS (LNAI), vol. 4755, pp. 26–38. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75488-6_3

55. Schmidhuber, J.: Compression progress: the algorithmic principle behind curiosity and creativity (with applications of the theory of humor) (2009). 40 min video of invited talk at Singularity Summit 2009, New York City: http://www.vimeo.com/7441291. 10 min excerpts: http://www.youtube.com/watch?v=Ipomu0MLFaI
56. Schmidhuber, J.: Driven by compression progress: a simple principle explains essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes. In: Pezzulo, G., Butz, M.V., Sigaud, O., Baldassarre, G. (eds.) ABiALS 2008. LNCS (LNAI), vol. 5499, pp. 48–76. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02565-5_4
57. Schmidhuber, J.: Formal theory of creativity, fun, and intrinsic motivation (1990–2010). IEEE Trans. Auton. Ment. Dev. **2**(3), 230–247 (2010). https://doi.org/10.1109/TAMD.2010.2056368
58. Schmidhuber, J.: Overviews of artificial curiosity/creativity and active exploration (with links to publications since 1990) (2012). http://www.idsia.ch/~juergen/interest.html, http://www.idsia.ch/~juergen/creativity.html
59. Schmidhuber, J.: Self-delimiting neural networks. Technical report. IDSIA-08-12, arXiv:1210.0118v1 [cs.NE], The Swiss AI Lab IDSIA (2012)
60. Schmidhuber, J.: PowerPlay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem. Front. Psychol. (2013). https://doi.org/10.3389/fpsyg.2013.00313, (Based on arXiv:1112.5309v1 [cs.AI], 2011)
61. Schmidhuber, J.: Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015). https://doi.org/10.1016/j.neunet.2014.09.003, published online 2014; 888 references; based on TR arXiv:1404.7828 [cs.NE]
62. Schmidhuber, J.: On learning to think: algorithmic information theory for novel combinations of reinforcement learning controllers and recurrent neural world models. Preprint arXiv:1511.09249 (2015)
63. Schmidhuber, J.: Artificial Curiosity & Creativity Since 1990–91. https://people.idsia.ch/~juergen/artificial-curiosity-since-1990.html (AI Blog, 2021), https://people.idsia.ch/~juergen/artificial-curiosity-since-1990.html
64. Schmidhuber, J.: Generative adversarial networks are special cases of artificial curiosity (1990) and also closely related to predictability minimization (1991). Neural Networks (2020)
65. Schmidhuber, J.: Learning one abstract bit at a time through self-invented experiments. Unpublished Tech Report, IDSIA & NNAISENSE (2020)
66. Shannon, C.E.: A mathematical theory of communication (parts I and II). Bell Syst. Tech. J. **XXVII**, 379–423 (1948)
67. Siegelmann, H.T., Sontag, E.D.: Turing computability with neural nets. Appl. Math. Lett. **4**(6), 77–80 (1991)
68. Singh, S., Barto, A.G., Chentanez, N.: Intrinsically motivated reinforcement learning. In: Advances in Neural Information Processing Systems (NIPS), vol. 17. MIT Press, Cambridge, MA (2005)
69. Solomonoff, R.J.: A formal theory of inductive inference. Part I. Inf. Control **7**, 1–22 (1964)
70. Srivastava, R.K., Steunebrink, B.R., Schmidhuber, J.: First experiments with PowerPlay. Neural Netw. **41**, 130–136 (2013). https://doi.org/10.1016/j.neunet.2013.01.022, http://www.sciencedirect.com/science/article/pii/S0893608013000373, special Issue on Autonomous Learning

71. van Steenkiste, S., Koutník, J., Driessens, K., Schmidhuber, J.: A wavelet-based encoding for neuroevolution. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016, pp. 517–524. GECCO '16, ACM, New York, NY, USA (2016)
72. Storck, J., Hochreiter, S., Schmidhuber, J.: Reinforcement driven information acquisition in non-deterministic environments. In: Proceedings of the International Conference on Artificial Neural Networks, Paris, vol. 2, pp. 159–164. EC2 & Cie (1995)
73. Sun, Y., Gomez, F., Schmidhuber, J.: Planning to be surprised: optimal Bayesian exploration in dynamic environments. In: Proceedings of the Fourth Conference on Artificial General Intelligence (AGI), Google, Mountain View, CA (2011)
74. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998)
75. Vaswani, A., et al.: Attention is all you need. Adv. Neural Inf. Process. Syst. 5998–6008 (2017)
76. Vemula, A., Sun, W., Bagnell, J.: Contrasting exploration in parameter and action space: a zeroth-order optimization perspective. In: The 22nd International Conference on Artificial Intelligence and Statistics, pp. 2926–2935. PMLR (2019)
77. Vinyals, O., et al.: Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature **575**(7782), 350–354 (2019)
78. Wallace, C.S., Boulton, D.M.: An information theoretic measure for classification. Comput. J. **11**(2), 185–194 (1968)
79. Wallace, C.S., Freeman, P.R.: Estimation and inference by compact coding. J. R. Stat. Soc. Ser. B **49**(3), 240–265 (1987)
80. Wiering, M., van Otterlo, M.: Reinforcement Learning. Springer, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27645-3
81. Wierstra, D., Foerster, A., Peters, J., Schmidhuber, J.: Recurrent policy gradients. Log. J. IGPL **18**(2), 620–634 (2010)
82. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach. Learn. **8**, 229–256 (1992)
83. Witten, I.H., Neal, R.M., Cleary, J.G.: Arithmetic coding for data compression. Commun. ACM **30**(6), 520–540 (1987)