Autonomous learning and chaining of motor primitives using the Free Energy Principle

Louis Annabi, Alexandre Pitti, Mathias Quoy

Autonomous learning and chaining of motor primitives using the Free Energy Principle

Louis Annabi
ETIS UMR 8051
CY University, ENSEA, CNRS
F-95000, Cergy-Pontoise, France
louis.annabi@ensea.fr

Alexandre Pitti
ETIS UMR 8051
CY University, ENSEA, CNRS
F-95000, Cergy-Pontoise, France
alexandre.pitti@ensea.fr

Mathias Quoy
ETIS UMR 8051
CY University, ENSEA, CNRS
F-95000, Cergy-Pontoise, France
mathias.quoy@ensea.fr

***Abstract*—In this article, we apply the Free-Energy Principle to the question of motor primitives learning. An echo-state network is used to generate motor trajectories. We combine this network with a perception module and a controller that can influence its dynamics.**

This new compound network permits the autonomous learning of a repertoire of motor trajectories. To evaluate the repertoires built with our method, we exploit them in a handwriting task where primitives are chained to produce long-range sequences.

***Index Terms*—Unsupervised learning, self-organizing feature maps, inference algorithms, predictive coding, intelligent control, nonlinear dynamical systems**

I. Introduction

We consider the problem of building a repertoire of motor primitives from an open-ended, task agnostic interaction with the environment. We suggest that a suitable repertoire of motor primitives should enable the agent to reach a set of states that covers best its state space. Based on this hypothesis, we train an agent to learn a discrete representation of its state space, as well as motor primitives driving the agent in the learned discrete states. In a fully observable environment, a clustering algorithm such as Kohonen self-organising maps [1] applied to the agent's sensory observations make it possible to learn a set of discrete states that covers well the agent's state space. Using this set of discrete states as goals, an agent can learn policies that drive it towards those goals, thus building for itself a repertoire of motor primitives. Our main contribution is to address this twofold learning problem in terms of free energy minimisation.

The Free Energy Principle [2] (FEP) suggests that the computing mechanisms in the brain accounting for perception, action and learning can be explained as a process of minimisation of an upper bound on surprise called free energy. On the one hand, FEP applied to perception [3] translates the inference on the causes of sensory observations into a gradient descent on free energy, and aligns nicely with the predictive coding [4] and Bayesian brain hypotheses [5]. On the other hand, FEP applied to action, or active inference [6], [7], can explain motor control and decision making as an optimisation of free energy constrained by prior beliefs. In this work, we present a variational formulation of our problem that allows us to translate motor primitives learning into a free energy minimisation problem.

This work was funded by the Cergy-Paris University Foundation (Facebook grant) and partially by Labex MME-DII, France (ANR11-LBX-0023-01).

In previous works, we applied the principle of free energy minimisation in a spiking recurrent neural network for the generation of long range sequences [8] but not associated to sensorimotor control. The presented model was able to generate long range sequences minimising free energy functions corresponding to several random goals. We used a randomly connected recurrent neural network in order to generate trajectories, and combined it with a second population of neurons in charge of driving its activation into directions minimising the distance towards a randomly sampled goal.

Using randomly connected recurrent neural networks to generate sequences is at the core of reservoir computing (RC) techniques [9], [10]. In particular, there is work using RC for the generation of motor trajectories, see for instance [11], [12]. In the RC framework, inputs are mapped to a high-dimensional space by a recurrent neural network called reservoir, and decoded by an output layer. The reservoir weights are fixed and the readout weights are regressed, usually with gradient descent. In [8], we proposed to alter the learning problem by fixing the readout weights to random values as well, and by optimising the input of the reservoir network instead.

In this article, we propose to combine the ideas developed in our previous work with a perception module in order to learn a repertoire of motor primitives. We train and evaluate our model in an environment designed for handwriting, where the agent controls a 2 degrees of freedom arm. The agent randomly explores its environment by drawing random trajectories on the canvas. By clustering its visual observations of the resulting trajectories, the agent learns a set of prototype observations which it will sample as goals to achieve during its future drawing episodes. This learning method is interesting from a developmental point of view since it implements goal-babbling. Developmental psychology tells us that learning sensorimotor contingencies [13] plays a key role in the development of young infants. In [14], the authors present a review about sensorimotor contingencies in the fields of developmental psychology and developmental robotics, in which they propose a very general model on how a learning agent should organise its exploration of the environment to develop

its sensorimotor skills. They suggest that the agent should continuously sample goals from its state space and practice achieving these goals. The work we propose in this article aligns nicely with their suggestion, as our agent randomly samples goals from a discrete state space, and optimises the motor sequences leading to these discrete states.

In the following we will first present the model, then the results on motor primitives learning. In a third part, we will evaluate the learned repertoires on a task of motor primitive chaining to draw complex trajectories.

II. Methods

A. Model for motor primitives learning

Our neural network architecture, represented in figure 1, can be segmented into three substructures. On the sensory pathway (top), a Kohonen map is used to cluster the observations. On the motor pathway (bottom), a reservoir network and a controller are used to model the generation and optimisation of motor sequences.

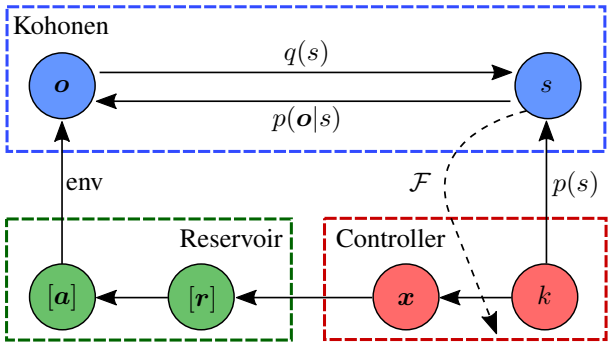


Fig. 1: Model for motor primitives learning. $\boldsymbol{x}$ denotes the activation signal. $[\boldsymbol{r}]$ denotes the sequence of activations $\{\boldsymbol{r}_t\}_{t=1..T}$ of the reservoir network. $[\boldsymbol{a}]$ denotes the sequence of atomic motor commands $\{\boldsymbol{a}_t\}_{t=1..T}$ decoded from the reservoir dynamics. $\boldsymbol{o}$ denotes visual observation. s denotes the hidden state. k denotes the primitive index on which depend the activation signal $\boldsymbol{x}$ and the prior probability over hidden state $p(s)$. $\mathcal{F}$ denotes the free energy, it is used as an optimisation signal for the controller.

The activation signal $\boldsymbol{x}$ stimulates the random recurrent neural network (RNN), that exhibits a self-sustained activity $\boldsymbol{r}$ during T time steps. T atomic motor commands $\boldsymbol{a}$ are readout from the T activations of the recurrent network. The environment provides an observation $\boldsymbol{o}$ after being impacted by the sequence of actions. This observation is then categorised in $s \in S = \{s_i\}_{i<n}$ by a Kohonen network.

1) Reservoir network: Motor primitives are sequences of atomic motor commands read out from the dynamics of the reservoir network. To optimise these sequences according to some criteria, one can either optimise the readout weights, the dynamics of the reservoir or control its activity (our case). Updating the recurrent weights of the RNN can have the undesirable effect of limiting its dynamics. Thus in RC, the focus is usually put on the learning of appropriate readout weights. However we showed in a previous work [8] that it is possible to control the RNN dynamics by optimising its initial state. In this article this is the strategy we use in order to learn motor primitives.

Our implementation of the reservoir network is based on an existing model for handwriting trajectory generation using RC [15], using the following equations :

$$\boldsymbol{u}(t) = (1 - \frac{1}{\tau}) \cdot \boldsymbol{u}(t-1) + \frac{1}{\tau}(\boldsymbol{W_r} \cdot \boldsymbol{r}(t-1)) \quad (1)$$

$$\boldsymbol{r}(t) = \tanh(\boldsymbol{u}(t)) \quad (2)$$

$$\boldsymbol{a}(t) = \tanh(\boldsymbol{W_o} \cdot \boldsymbol{r}(t)) \quad (3)$$

where $\boldsymbol{u}$ and $\boldsymbol{r}$ denote the network activation respectively before and after application of the non-linearity. We denote by τ the time constant of the network.

For the recurrent network to have a self-sustained activity, we referred to the weights initialisation used in [15]. The recurrent weights matrix $\boldsymbol{W_r}$ is made sparse with each coefficient having a probability p_r of being non-null. When non-null, the coefficients are sampled from a normal distribution $\mathcal{N}(0, \frac{\sigma_r^2}{n_r})$ with a variance scaled according to the network size n_r. The readout weights $\boldsymbol{W_o}$ are sampled from a normal distribution $\mathcal{N}(0, \sigma_o^2)$.

2) Kohonen map: The Kohonen map [1] takes as input a 64x64 gray scale image. Each filter learned by the Kohonen map has to correspond to a distinct motor primitive. Since we expected to learn motor primitives corresponding mainly to movement orientations, we used a Kohonen map topology with only one cyclic dimension. We also ran experiments with 2-d and 3-d topologies with and without cyclic dimensions. We chose to stick with the 1-d cyclic topology because it presented a fast learning and a balanced use of all the learned filters. Here are the equations of the Kohonen network:

$$i_w(t) = \underset{i}{\operatorname{argmin}}(\|\boldsymbol{W_k}[i,:](t) - \boldsymbol{o}(t)\|_2^2) \quad (4)$$

$$\boldsymbol{W_k}(t+1) = (1 - \lambda_k) \cdot \boldsymbol{W_k}(t) + \lambda_k \cdot N(i_w(t)) \odot \boldsymbol{o}(t) \quad (5)$$

where $\boldsymbol{W_k}$ denotes the Kohonen weights, each row $\boldsymbol{W_k}[i,:]$ corresponding to the filter associated with neuron of index i. i_w denotes the winner neuron index, i.e. the index of the Kohonen neuron whose associated filter is closest to the input stimulus $\boldsymbol{o}$. The neighbourhood function $N(i_w(t))$ depends on the chosen topology. It is maximum in $i_w(t)$ and decreases exponentially according to the distance with regard to the winner neuron index $i_w(t)$. This exponential decay is parameterised by a neighbourhood width σ_k^2. The operator $\odot$ denotes the element-wise product.

3) Free energy derivations: Our goal is to learn the right activation signals $\boldsymbol{x}$ that stimulate the random recurrent network in a way that leads to desired categorisations of the observation $\boldsymbol{o}$ by the Kohonen network. Let n be the number of motor primitives that we want to learn. We set the size of the Kohonen network as well as the number of activation signals to learn to n. For the trajectory of index k, we want to learn the stimulus $\boldsymbol{x_k^*}$ that better activates the corresponding category in the Kohonen network (i.e. such that $p(\boldsymbol{o}|s = s_k) \approx 1$). We can

observe that the optimal activation signal $\boldsymbol{x_k^*}$ depends on the weights of the Kohonen map. Because of this dependence, it would be easier to learn and fix the Kohonen map before optimisation of the controller. However, it is more realistic from a developmental point of view to train these two structures at the same time. For this reason, we learn both networks in parallel but control their learning parameters over time, to be able to favor the learning of one network compared to the other.

We use free-energy minimisation as the strategy to train the controller. What follows is a formalisation of our model using a variational approach:

- $p(s)$ is the prior probability over states. Here we propose using a softmax, parameterised by $\beta > 0$, around the index k of the current primitive.

$$p(s = s_i) = \frac{\exp(-\beta \cdot |k - i|)}{\sum_j \exp(-\beta \cdot |k - j|)} \tag{6}$$

- $p(\boldsymbol{o}|s)$ is the state observation mapping. The observations are images of size d. For simplicity, we make the approximation of considering all pixel values as independent. We choose to use Bernoulli distributions for all pixel values $o_{l<d} \in \{0, 1\}$. Since all pixel values are considered independent, the probability distribution over the whole observation can be factorised as:

$$p(\boldsymbol{o}|s = s_i) = \prod_{l<d} \boldsymbol{W_k}[i, l]^{o_l} \cdot (1 - \boldsymbol{W_k}[i, l])^{1-o_l} \tag{7}$$

  where $\boldsymbol{W_k}[i, l]$ is the value of pixel l of the filter i of the Kohonen map.
- $q(s)$ is the approximate posterior probability over states knowing the observation $\boldsymbol{o}$. Here we define $q(s)$ to be the one-hot distribution over states such that $q(s = s_i) = \delta_{i_w,i}$, where i_w is the index of the Kohonen neuron with the highest activation (i.e. whose filter is the closest to the observation): $i_w = \text{argmin}_j(\|\boldsymbol{W_k}[j, :] - \boldsymbol{o}\|_2^2)$.

We can now derive the free-energy computations using this model:

$$\mathcal{F}_1(\boldsymbol{o}) = \text{KL}(q(s)||p(s)) - \sum_{i<n} q(s_i) \log(p(\boldsymbol{o}|s_i)) \tag{8}$$

$$= \sum_{i<n} q(s_i) \log \frac{q(s_i)}{p(s_i)} - \sum_{i<n} q(s_i) \log(p(\boldsymbol{o}|s_i)) \tag{9}$$

$$= -\log(p(s_{i_w})) - \log(p(\boldsymbol{o}|s_{i_w})) \tag{10}$$

The first term of the free energy in eq. (8) is a quantity called complexity. It scores how complex the approximate posterior is compared to the prior. It decreases when $q(s)$ and $p(s)$ are close. In our case, it is minimal when $i_w = k$, meaning that the category chosen by the Kohonen map is the one with the highest prior probability. Minimising complexity thus induces the network to generate trajectories that activate the right Kohonen category.

The second term is the opposite of the quantity called accuracy. Accuracy measures how good the approximate posterior probability $q(s)$ is at predicting the observation $\boldsymbol{o}$. Here, it increases when the Kohonen filter of the winner neuron is close to the observation. Maximising accuracy induces the network to generate trajectories that are as close as possible to one of the Kohonen filter. For simplicity, we will call this quantity inaccuracy instead of opposite of accuracy.

Summing those two quantities, minimising free energy would result in observations that are close to one of the Kohonen filter, and in this Kohonen filter being the one with the highest prior probability.

4) Optimisation method: Our optimisation problem is the following. For each primitive k, we want to find an activation signal $\boldsymbol{x_k}$ that generates an observation $\boldsymbol{o}$ resulting in a low free energy $\mathcal{F}_1(\boldsymbol{o})$. To use gradient based methods, we would need to have a diiferentiable model of how the activation signals $\boldsymbol{x_k}$ impacts the resulting free energy $\mathcal{F}_1(\boldsymbol{o})$. Since we do not have a model of how the environment produces observations, the whole $\boldsymbol{x} \rightarrow \boldsymbol{r} \rightarrow \boldsymbol{a} \rightarrow \boldsymbol{o} \rightarrow \mathcal{F}_1(\boldsymbol{o})$ chain is not differentiable. To solve our problem, we instead use a random search optimisation method, detailed by the following algorithm.

{Random initialisation of the controller}
for $k < n$ **do**
  $\boldsymbol{x}_k \leftarrow \mathcal{N}(0, 1)$
end for
{Training}
for $e < E$ **do**
  $k \leftarrow \mathcal{U}(n)$
  $\boldsymbol{\delta x} \sim \mathcal{N}(0, \sigma^2(e))$
  $u_+ \leftarrow \boldsymbol{x_k} + \boldsymbol{\delta x}$
  $u_- \leftarrow \boldsymbol{x_k} - \boldsymbol{\delta x}$
  $[\boldsymbol{a}_1, \ldots, \boldsymbol{a}_T]_+ \leftarrow$ simulate_action(u_+)
  $[\boldsymbol{a}_1, \ldots, \boldsymbol{a}_T]_- \leftarrow$ simulate_action(u_-)
  $\boldsymbol{o}_+ \leftarrow$ env($[\boldsymbol{a}_1, \ldots, \boldsymbol{a}_T]_+$)
  $\boldsymbol{o}_- \leftarrow$ env($[\boldsymbol{a}_1, \ldots, \boldsymbol{a}_T]_-$)
  $i_{w,+} \leftarrow$ simulate_kohonen($\boldsymbol{o}_+$)
  $i_{w,-} \leftarrow$ simulate_kohonen($\boldsymbol{o}_-$)
  $f_+ \leftarrow$ free_energy($i_{w,+}, \boldsymbol{o}_+$)
  $f_- \leftarrow$ free_energy($i_{w,-}, \boldsymbol{o}_-$)
  $\boldsymbol{x}_k \leftarrow \boldsymbol{x}_k - \lambda \cdot (f_+ - f_-) \cdot \boldsymbol{\delta x}$
end for

The parameter e in the search standard deviation $\sigma^2(e)$ indicates that this coefficient can depend on the training episode e. The "simulate_action" function used in the code above corresponds to the iterative application of equations (1), (2), (3) for the duration T of the motor primitives. The "env" function corresponds to the generation of an observation by the environment after being impacted by the sequence of actions. This computation is performed by the environment and unknown to the agent. The "simulate_kohonen" function corresponds to the application of equations (4) and (5). The "free_energy" function corresponds to the application of equation (10).

B. Experimental setup

1) Environment: The environment is an initially blank canvas on which the agent can draw. The initial position of the pen is at the center of the canvas. The agent can act on

the environment via 2D actions. The actions are the angle velocities of a 2 degrees of freedom arm as represented in figure 2.

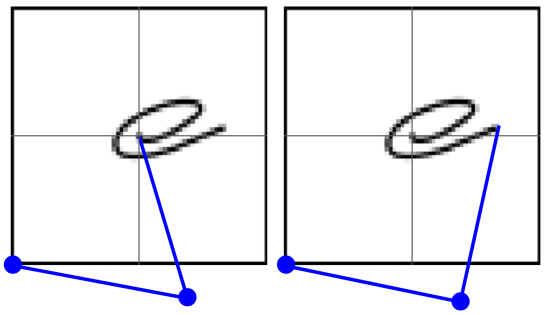


Fig. 2: Initial (left) and final (right) arm position for a trajectory taken from a data set of handwriting trajectories.

2) Training: We start from a random initialisation of the Kohonen map. Over time, the Kohonen map self-organises when being presented with the trajectories generated by the random search algorithm. Simultaneously, the random search algorithms learns to generate motor trajectories that lead to the different Kohonen prototype observations.

Training was performed using the following set of parameters for the different components described in the previous section:

- RNN: $n_r = 100$, $\tau = 10$, $p_r = 0.1$, $\sigma_r^2 = 1, 5$.
- Readout layer: $n_o = 2$, $\sigma_o^2 = 1$.
- Kohonen map: $n = 50$, $\lambda_k = 0.01$, Kohonen width $\sigma_k^2(e)$ varies over time, see figure 3.
- Free energy: $\beta \in \{2^{-5}, 2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 1, 2, 4, 8, 16\}$.
- Random optimisation: $\lambda = 0.01$, $\sigma^2(e)$ varies over time, see figure 3.

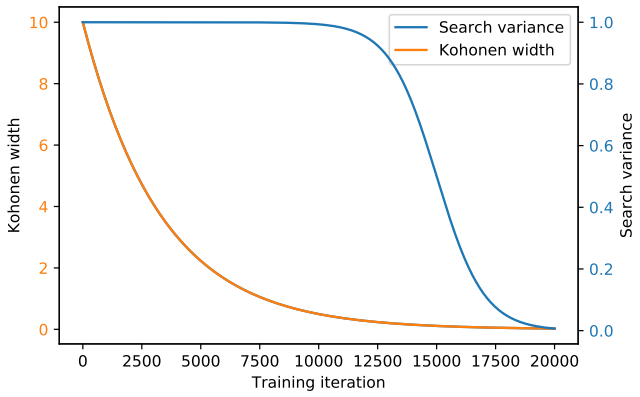


Fig. 3: Search variance $\sigma^2(e)$ and Kohonen width $\sigma_k^2(e)$ according to training iteration e.

C. Results

We trained our model for $E = 20000$ iterations on $n = 50$ primitives. At each iteration, we uniformly sample k from $[1, n]$. We train on the k^{th} primitive by adjusting the prior probability as in (6) and optimising $\boldsymbol{x_k}$. On average, each activation signal $\boldsymbol{x_k}$ is trained on 400 iterations. Figure 3 displays the evolution of the random optimisation search variance and of the Kohonen width over the 20000 iterations. We discuss this choice in the light of the following results.

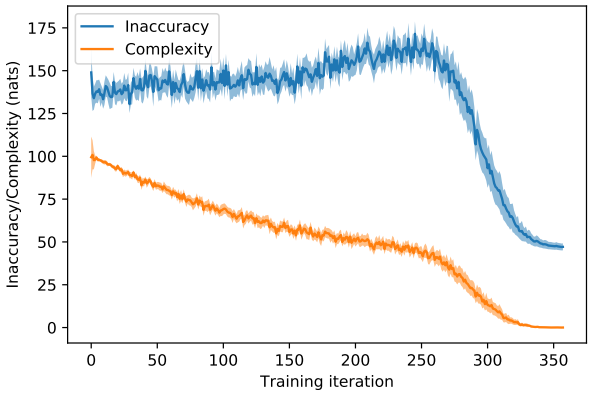


Fig. 4: Inaccuracy and complexity averaged on the number of primitives $n = 50$, with $\beta = 8$. Each primitive has been sampled on at least 360 iterations, and on average on 400 iterations.

Figure 4 displays the evolution of inaccuracy and complexity during training.

During the first phase, when $e < 12500$, the random search has a very high variance. Consequently, the trajectories generated and fed to the Kohonen map are very diverse and this allows the Kohonen map to self-organise. Inaccuracy does not seem to decrease in this early phase. This is because the Kohonen filters, initially very broad, are becoming more precise. The high variance in the random search allows for a diminution of complexity but still generates trajectories that are too noisy to accurately fit the more precise Kohonen filters.

During the second phase, we decrease the variance of the random search. The system can now converge more precisely and this causes a faster decrease of both inaccuracy and complexity.

We can question whether it is necessary for the random search variance to remain high for such a long time, since it slows down learning. We observed that if we reduce the duration of the first phase, the Kohonen does not have the time to self organise and this results in an entangled topology. Because of the complexity term in the free energy computations, the topology of the Kohonen has an influence over the learning. For instance, with an entangled topology, a search direction for $\boldsymbol{x_k}$ that activates a neuron closer to k might not make the actual trajectory closer to the ones recognised by the k^{th} Kohonen neuron. In other words, having a proper topology smooths the loss function.

We also notice that the inaccuracy cannot decrease below a certain value. At first, we could think that this is because the

optimisation strategy is stuck in a local optimum. However, we obtained the same lower bound on inaccuracy over different training sessions. Since the optimisation strategy relies on random sampling, there is no evident reason to encounter the same local minimum. Our explanation is that this lower bound is imposed by the Kohonen network neighbourhood function. Because the Kohonen width does not reach 0, the Kohonen centroids are still attracting each other and this prevents them from completely fitting the presented observations. In consequence, the filters are always partly mixed with their neighbours and this causes the inaccuracy to plateau at a value that depends on σ_k^2.

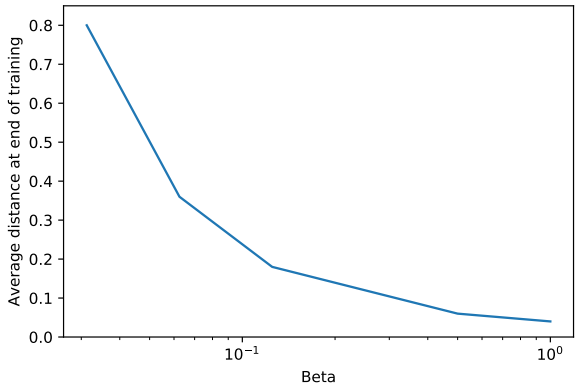


Fig. 5: Average distance $|i_w - k|$ between the activated neuron in the Kohonen i_w and the primitive index k, according to β.

Figure 5 shows the impact of the parameter β over the convergence. Looking at the equations (6) and (10) we can see that this parameter directly scales the overall complexity. For low values of β, the random search is more likely to be stuck in local minima of free-energy, when activating a Kohonen neuron closer to k corresponds to an increase in inaccuracy that exceeds the decrease in complexity. We measured the average distance between the activated neuron in the Kohonen and the primitive index k at the end of training for $\beta \in \{2^{-5}, 2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^{0}\}$. The results, presented in figure 5, confirm that the final states obtained with higher values of β correspond to a more precise mapping between i_w (winner index of the Kohonen map) and k (state index enforced by the prior probability).

Figure 6 displays some of the learned motor primitives. The blue component of the image corresponds to the trajectory that is actually being generated by the reservoir network for the activation signal $\boldsymbol{x_k}$. The red component of the image corresponds to the Kohonen filter of index k. This figures allows visual confirmation of several points. First, the inaccuracy at the end of training seems to indeed come from the blurriness of the Kohonen filters. Second, the filters and motor primitive trajectories seem to follow a topology: the index of the primitive seems highly correlated with the orientation of the route taken by the arm end effector. Finally, every trajectory seems to be in the center of the corresponding Kohonen filter, which suggests

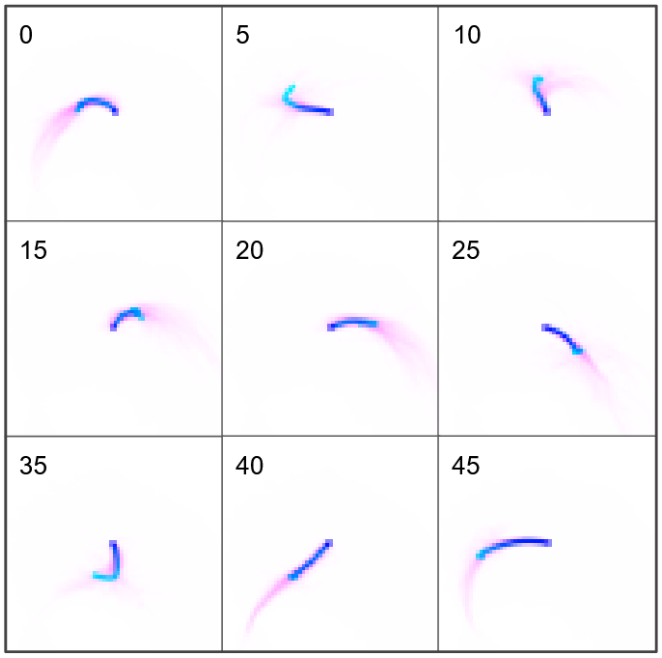


Fig. 6: 9 of the 50 learned motor primitives and corresponding Kohonen filters: $k \in \{0, 5, 10, 15, 20, 25, 35, 40, 45\}$. The blue component of the image corresponds to the trajectory that is actually being generated by the network for the activation signal $\boldsymbol{x_k}$. The red component of the image corresponds to the Kohonen filter of index k.

that the minimisation of complexity successfully enforced the mapping between i_w and k.

III. Chaining of motor primitives

To validate our approach, we still need to show that this repertoire of motor primitives is efficient at constructing more complex movements. To perform this evaluation, we propose to extend the model presented in section II.

First, we define a new perception module meant to classify sensory observations into states corresponding to more complex trajectories. To avoid confusion, we will denote this new hidden state σ.

Second, we enable in this revisited model the chaining of motor primitives. In the previous model, each drawing episode corresponds to one motor trajectory of length T being generated. Here in each drawing episode the agent will draw a trajectory corresponding to M motor primitives of length T chained together.

Since we are simply trying to evaluate the primitives learned in the first model, we won't address the training of the network used to classify sensory observations. Instead, our focus will be on the decision making process, i.e. the selection of the primitives $\{k_1, \ldots, k_M\}$ to chain in order to reach a certain desired state σ^*.

A. Model

The model for motor primitives chaining is presented in figure 7.

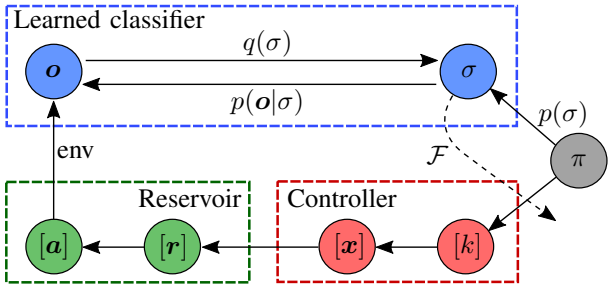


Fig. 7: Model for motor primitives chaining. $\pi = p(\sigma)$ denotes the prior probability over states σ. $[k]$ denotes the sequence of motor primitives indices $\{k_m\}_{m=1..M}$. $[\boldsymbol{x}]$ denotes the sequence of corresponding activation signals $\{\boldsymbol{x}_m\}_{m=1..M}$.$[\boldsymbol{r}]$ denotes the resulting sequence of activations $\{\boldsymbol{r}_{m,t}\}_{m=1..M,t=1..T}$ of the reservoir network. $[\boldsymbol{a}]$ denotes the sequence of atomic motor commands $\{\boldsymbol{a}_{m,t}\}_{m=1..M,t=1..T}$ decoded from the reservoir dynamics. $\boldsymbol{o}$ denotes the visual observation provided by the environment after being modified by the sequence of atomic motor commands. σ denotes the hidden state, it is different from the hidden state of the previous section. $\mathcal{F}$ denotes the expected free energy, it is used as an optimisation signal for the choice of sequence of motor primitives indices $\{k_m\}_{m=1..M}$.

1) New hidden state: The hidden state σ corresponds to new categories representing complex motor trajectories. We used the Character Trajectories Data Set from [16] composed of approximately 70 trajectories for each letter of the alphabet that can be drawn without lifting the pen. The trajectories are provided as sequences of pen positions. We drew these trajectories using our drawing environment and used the resulting observations to build the new state observation mapping.

2) Expected free energy derivations: The model uses active inference for decision making. It selects actions that minimise a free energy function constrained by prior beliefs over hidden states. According to active inference, constraining the prior probability over states to infer a state distribution that favors a target state σ^* will force the agent to perform actions that fulfill this prediction. In this sense, the prior probability over states can be compared to the definition of a reward in reinforcement learning. For instance, a rewarding state would be a state that is more likely under the prior probability over states.

Here is the formalisation of this model in the variational framework:

- Prior probability over states $p(\sigma)$ acts similarly to a reward function in reinforcement learning. The prior beliefs (or prior preferences) over σ will be set manually to different values during testing to guide the agent into a desired state.
- The other probability distribution over states is one that the agent has control on. By choosing one primitive in the learned repertoire, the agent selects one resulting distribution over states $q_k(\sigma)$ modeling how the choice of motor primitive k will influence the state. This probability distribution over hidden states corresponds to the output of the learned classifier when fed with the observation resulting from the application of the k^{th} motor primitive.
- As in the primitive learning model presented in section II, the state observation mapping $p(\boldsymbol{o}|\sigma)$ is built using equation (7). The filters used for each category correspond to the average of the observations belonging to this class obtained from the data set.

We can now derive the expected free-energy using this model:

$$\mathbb{E}[\mathcal{F}_2(k)] = \mathrm{KL}(q_k(\sigma)||p(\sigma)) - \sum_{i<n} q_k(\sigma_i)\mathcal{H}(p(\boldsymbol{o}|\sigma_i)) \quad (11)$$

where $\mathcal{H}(p(\boldsymbol{o}|\sigma_i))$ denotes the entropy of the state observation mapping for state i.

When the agent has to make a decision about which motor primitive to use, it computes its expected future free energy $\mathbb{E}[\mathcal{F}_2(k)]$ for each possible primitive and selects the primitive with the minimum expected free energy.

This time again, expected free energy can be segmented as complexity and inaccuracy. Minimising the first term of equation (11) will result in choosing motor primitives that lead to hidden states matching our prior preferences. This can be seen as directly optimising reward. Minimising inaccuracy will result in choosing actions that lead to hidden states with high precision (low entropy) in the state observation mapping. This connects to the drive of surprise avoidance inherent to the free energy principle.

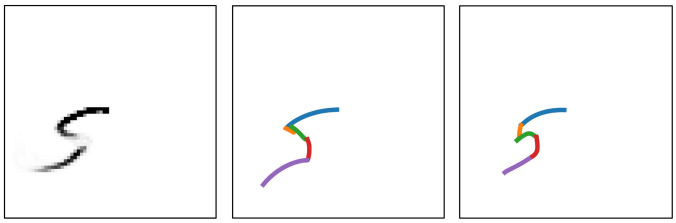
3) Action selection: The following algorithm details the action selection process on a trajectory composed of M motor primitives using free energy minimisation :

$p(\sigma) \leftarrow$ init()
for $m < M$ **do**
  for $k < n$ **do**
    $u \leftarrow \boldsymbol{x}_k$
    $[\boldsymbol{a}_1, \ldots, \boldsymbol{a}_T] \leftarrow$ simulate_action(u)
    $\boldsymbol{o} \leftarrow$ env_model$([\boldsymbol{a}_1, \ldots, \boldsymbol{a}_T])$
    $q_k(\sigma) \leftarrow$ classifier$(\boldsymbol{o})$
    $f_k \leftarrow$ expected_free_energy$(q_k(\sigma), p(\sigma), \boldsymbol{o})$
  end for
  $k^*_m \leftarrow \mathrm{argmin}_k(f_k)$
  $u^* \leftarrow \boldsymbol{x}_{k^*_m}$
  $[\boldsymbol{a}_1, \ldots, \boldsymbol{a}_T] \leftarrow$ simulate_action(u)
  env$([\boldsymbol{a}_1, \ldots, \boldsymbol{a}_T])$
end for

The function "env_model" corresponds to a learned forward model that allows estimating the future observations for any sequence of actions. In our experiments, we simply simulated the actions and rewound the environment, which (in a deterministic environment) corresponds to having a perfect forward model (env = env_model). The function "classifier" corresponds to the classification of the observation by a learned classifier. It outputs a probability distribution over hidden states σ. Finally, the function

"expected_free_energy" corresponds to the application of eq. (11).

B. Results



(a) (b) (c)

Fig. 8: Example of filter and produced trajectories. Left: Filter for the category 's'. Middle and right: trajectories produced by chaining five motor primitives belonging to two different learned repertoires.

In our tests, the hidden state σ was build using five classes corresponding to the letters 'c', 'h', 'i', 's', 'r'. We chose these letters because the trajectories inside each category were relatively close and this allowed for filters of low entropy.

Figure 8 displays one filter of the learned classifier and two trajectories generated by chaining five motor primitives from two different primitive repertoires learned with our model. The trajectories were obtained by setting the prior preferences to 0.96 for the category 's' and 0.01 for the four other categories.

To verify that our model learns a valuable repertoire of motor primitives, we compare the quality of the constructed complex trajectories (as in 8b and 8c) with our model and with random repertoires.

Random repertoires are built using the same RNN and readout layer initialisations. They differ from the learned repertoires in the fact that we do not optimise the activation signals $\boldsymbol{x_k}$ of the reservoir. The initial states of the reservoir used to generate the primitives are taken as $\boldsymbol{x}_k \sim \mathcal{N}(0, 1)$. In other words, they are equivalent to the repertoires our model would provide before learning.

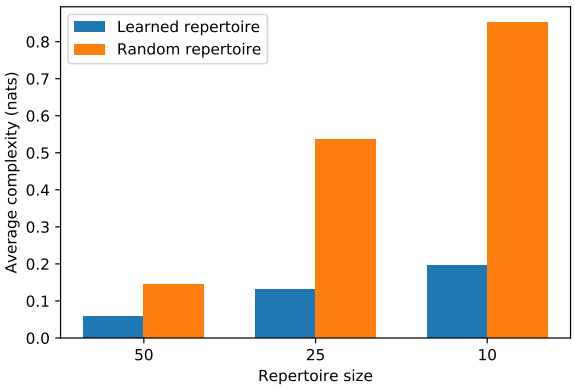


Fig. 9: Average complexity for learned and random repertoires of different sizes.

Figure 9 displays the average complexities measured at the end of the episode. For each episode, we set the prior preferences of one of the letter category to 0.96 and the others to 0.01. The values are averaged over the different letters and for 5 different repertoires, learned or random.

The complexity scores how close the recognition probability $q(\sigma)$ (provided by the learned classifier) is to the prior preferences $p(\sigma)$ and thus constitutes a suitable indicator for comparison. For low complexities, the constructed images are close to the filter, as in 8.

We observe that the average complexity tends to be lower for repertoires of larger sizes, independently of the type of repertoire. Having a larger repertoire of primitives indeed should be an asset in order to reconstruct more complex trajectories. For every repertoire size, we measure a lower complexity with repertoires learned using the model described in section II.

IV. Conclusion

The results displayed in section III show that our model is able to learn repertoires of motor primitives that are efficient at building more complex trajectories when combined.

To further validate our approach, it would be interesting to compare our results with other strategies for motor primitive learning. On the one hand, there is existing work in developmental robotics prescribing guidelines to build repertoires of motor primitives [14], [17], but they don't provide neural network implementation to be used for comparison.

On the other hand, the option discovery literature in hierarchical reinforcement learning provides practical methods to build repertoires of relevant options. Options were introduced in [18] as a candidate solution to address the issue of temporal scaling in reinforcement learning. An option is defined as a temporally extended action, and thus is conceptually similar to a motor primitive. It would be interesting to measure how our approach compares with current state of the art techniques for unsupervised option learning such as [19].

Acknowledgment

This work was funded by the Cergy-Paris University Foundation (Facebook grant) and Labex MME-DII, France (ANR11-LBX-0023-01).

References

[1] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43, pp. 59–69, 1982.

[2] K. Friston and J. Kilner, "A free energy principle for the brain," *J. Physiol. Paris*, vol. 100, pp. 70–87, 2006.

[3] K. Friston, "Hierarchical models in the brain," *PLOS Computational Biology*, vol. 4, no. 11, pp. 1–24, 11 2008.

[4] R. Rao and D. Ballard, "Predictive coding in the visual cortex a functional interpretation of some extra-classical receptive-field effects," *Nat Neurosci*, vol. 2, pp. 79–87, 1999.

[5] P. Dayan, G. E. Hinton, R. M. Neal, and R. S. Zemel, "The helmholtz machine," *Neural Computation*, vol. 7, pp. 889–904, 1995.

[6] K. Friston, J. Daunizeau, and S. Kiebel, "Reinforcement learning or active inference?" *PLoS ONE*, vol. 4, no. 7, p. e6421, 2009.

[7] K. Friston, T. FitzGerald, F. Rigoli, P. Schwartenbeck, J. ODoherty, and G. Pezzulo, "Active inference and learning," *Neuroscience & Biobehavioral Reviews*, vol. 68, pp. 862–879, 2016.

[8] A. Pitti, P. Gaussier, and M. Quoy, "Iterative free-energy optimization for recurrent neural networks (inferno)," *PLoS ONE*, vol. 12, no. 3, p. e0173684, 2017.
[9] D. Verstraeten, B. Schrauwen, M. DHaene, and D. Stroobandt, "An experimental unification of reservoir computing methods," *Neural Network*, vol. 20, pp. 391–403, 2007.
[10] M. Lukoeviius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127 – 149, 2009.
[11] J. Namikawa and J. Tani, "Learning to imitate stochastic time series in a compositional way by chaos," *Neural Networks*, vol. 23, no. 5, pp. 625 – 638, 2010.
[12] F. Mannella and G. Baldassare, "Selection of cortical dynamics for motor behaviour by the basal ganglia," *Biological Cybernetics*, vol. 109, pp. 575–595, 2015.
[13] J. K. O'Regan and A. No, "A sensorimotor account of vision and visual consciousness," *Behavioral and Brain Sciences*, vol. 24, no. 5, p. 939973, 2001.
[14] L. Jacquey, G. Baldassare, V. Santucci, and O. J.K., "Sensorimotor contingencies as a key drive of development: From babies to robots," *Frontiers in NeuroRobotics*, vol. 13, no. 98, 2019.
[15] R. Laje and D. Buonomano, "Robust timing and motor patterns by taming chaos in recurrent neural networks," *Nature Neuroscience*, vol. 16, no. 7, pp. 925–935, 2013.
[16] D. Dua and C. Graff, "Uci machine learning repository," *[http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science*, 2019.
[17] E. Ugur, E. Sahin, and E. Öztop, "Self-discovery of motor primitives and learning grasp affordances," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3260–3267, 2012.
[18] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1, pp. 181 – 211, 1999.
[19] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine, "Diversity is all you need: Learning skills without a reward function," *CoRR*, vol. abs/1802.06070, 2018.