PAPER

Scaling up liquid state machines to predict over address events from dynamic vision sensors

To cite this article: Jacques Kaiser *et al* 2017 *Bioinspir. Biomim.* **12** 055001

Bioinspiration & Biomimetics

RECEIVED
16 December 2016

REVISED
16 May 2017

ACCEPTED FOR PUBLICATION
1 June 2017

PUBLISHED
1 September 2017

PAPER

Scaling up liquid state machines to predict over address events from dynamic vision sensors

Jacques Kaiser[1,3], **Rainer Stal**[1], **Anand Subramoney**[2], **Arne Roennau**[1] **and Rüdiger Dillmann**[1]

[1] FZI Research Center for Information Technology, 76131 Karlsruhe, Germany
[2] Institute for Theoretical Computer Science, Graz University of Technology, A-8010 Graz, Austria
[3] Author to whom any correspondence should be addressed

E-mail: jkaiser@fzi.de, stal@fzi.de, roennau@fzi.de, dillmann@fzi.de and anand.subramoney@tugraz.at

Keywords: liquid state machine, dynamic vision sensor, event-based vision, spiking neural network, motion prediction

Abstract

Short-term visual prediction is important both in biology and robotics. It allows us to anticipate upcoming states of the environment and therefore plan more efficiently. In theoretical neuroscience, liquid state machines have been proposed as a biologically inspired method to perform asynchronous prediction without a model. However, they have so far only been demonstrated in simulation or small scale pre-processed camera images. In this paper, we use a liquid state machine to predict over the whole 128 × 128 event stream provided by a real dynamic vision sensor (DVS, or silicon retina). Thanks to the event-based nature of the DVS, the liquid is constantly fed with data when an object is in motion, fully embracing the asynchronicity of spiking neural networks. We propose a smooth continuous representation of the event stream for the short-term visual prediction task. Moreover, compared to previous works (2002 *Neural Comput.* **2525** 282–93 and Burgsteiner H *et al* 2007 *Appl. Intell.* **26** 99–109), we scale the input dimensionality that the liquid operates on by two order of magnitudes. We also expose the current limits of our method by running experiments in a challenging environment where multiple objects are in motion. This paper is a step towards integrating biologically inspired algorithms derived in theoretical neuroscience to real world robotic setups. We believe that liquid state machines could complement current prediction algorithms used in robotics, especially when dealing with asynchronous sensors.

1. Introduction

The human visual system can make short-term predictions over the state of the surrounding environment. It is essential when dealing with objects moving faster than our own body. One such example is a tennis game. The player needs to know in advance where and when will the ball arrive so that he/she can be in the right place of the court and position his/her racket properly. As the ball moves faster than the tennis player, the player can not simply follow the current position of the ball. Rather, the player needs to anticipate the position of the ball in the near future, and position him/herself with respect to it.

In robotics, predictions are often used to anticipate upcoming states of the environment, for more efficient planning. It became a crucial step in many problems such as state estimation [3], visual servoing [4, 5], obstacle avoidance [6, 7] and more recently imitation learning [8, 9]. They can also be used to evaluate our belief about an environment by comparing a predicted state with the actual future state.

Most of the current approaches for prediction in robotics work in a synchronous fashion where new data is sensed at discrete time intervals. However, recent bio-inspired sensors such as the dynamic vision sensor (DVS, or silicon retina [10]) deliver data asynchronously upon local changes in the environment. The event-based nature of these sensors has many important advantages concerning robotics: low information redundancy, reduced processing latency and bandwidth, high dynamic range, lightweight data storage and power consumption.

Recent model-based methods have been developed to perform asynchronous predictions over address events [6, 11]. They fit the observed data to a known kinematic model, which is then used to predict upcoming observations. These techniques require assumptions over the observed motion, such as slowly varying velocity or acceleration.

In theoretical neuroscience, liquid state machines have been presented as a biologically realistic general purpose learning framework, asynchronous by nature and well suited to temporal processing. They belong to the family of reservoir computing and are implemented with spiking neural networks. The liquid can be seen as a kernel method performing a time-dependent projection to a high dimensional space. It allows the training of a linear memoryless readout neurons to approximate non-linear time-dependent functions. They have already been used to perform short-term visual predictions, from simulated [1] or real [2] frame-based sensors. Coupled with event-based sensors, liquid state machines have the potential to operate more efficiently than with classical frame-based sensors. Indeed, event-based sensors only perceive changes in the environment, while the liquid can hold a full representation of it thanks to its memory. Echo state networks, the non-spiking counterparts of liquid state machines, were already considered as feature extractors for event-based data [12]. However, purely spiking liquid state machines have not yet been evaluated in a realistic robotic setup with asynchronous full-size visual input.

In this paper, we implement and evaluate a liquid state machine predicting future visual input from address events provided by a DVS. Compared to previous works [1, 2], we substantially scale up the input dimensionality by two orders of magnitudes by demonstrating the method on the whole 128 × 128 pixel array. The main contribution of this work is the evaluation of liquid state machines for short-term prediction on real-world full-size event streams. This required a proper tuning of the liquid hyperparameters (tables 1–3), a suitable representation of the event stream (equation (2)) and adequate error metrics (section 4). This paper is a step towards integrating biologically inspired algorithms derived in theoretical neuroscience to real world robotic setups.

After a review of the related work in section 2, we describe the biologically inspired architecture of the liquid in section 3.1. The training of the readout neurons is based on a simple linear regression described in section 3.2. The method is evaluated both in simulation against different ball motions, and with a real DVS in section 4. In section 5, we discuss how the method could be improved and used in closed-loop experiments.

2. Related work

Reservoir computing can be defined as an approach to design, train, and analyze recurrent neural networks. It is inherently suited to process temporal sequences. Recurrent neural networks are powerful, yet hard to train. The principle of reservoir computing is that solely readout weights are trained, and not the recurrent part of the network. Liquid state machines and echo state networks [13] are two implementations of reservoir computing that were independently invented. They usually consist of spiking neurons for the former and analog neurons for the later. Liquid state machines are biologically inspired models for real-time computations on continuous streams of data.

Each neuron within the liquid creates its own non-linear transformation of the input. A layer of readout neurons is connected to some of the neurons in the liquid. The liquid state is defined by the post-synaptic activity of the liquid neurons that are connected to the readouts. The training consists of finding the appropriate weights mapping the liquid state to the desired signal. It is usually performed with a simple supervised linear regression.

That ability of separation and transformation of the liquid states into the given target output is called echo state property. This property depends directly on the complexity of the liquid [14]. It is connected with the fading memory property: the previous input has influence on the current liquid state, but this influence fades away.

An interesting study of the computational capability of liquid state machines is presented in [15] where a bucket of water was used as the physical liquid and input streams were injected with two motors. Video images of the surface of the water were used as states for pattern recognition. The experiment demonstrated how a single perceptron which performs a linear classification can solve a XOR problem if the input is pre-processed with the water medium. Subsequently, liquid state machines have been demonstrated in applications ranging from decoding actual brain activity [16] to control robots [17–19]. Usually trained in a supervised fashion, target output signals must be known for a given training set. In robot control, a classical robot controller is often derived first, so that the liquid learns the mapping between input and target signals specific to this controller [17–19]. The focus can be on embodiment [17] or spiking network controllers [18, 19].

In this work, we focus on visual motion prediction, as in the previous works [1, 2] and, to some extent, [12]. In [1], a liquid state machine is used to predict over a 8 × 8 simulated sensor array. The model of the liquid was constructed to reflect the empirical data from microcircuits in the somatosensory cortex of a rat. The simulated sensor encodes the proportion of the ball occupying the cell at a given instant. The liquid states were sampled every 5 ms. Using the same liquid, the method successfully trained different readout neurons to predict inputs 50 ms in future, where the object will exit the frame, and classify two different stimuli (ball or bar). The dataset consisted of 1500 samples of linear trajectories of the object that was moving with various constant speeds and random orientations.

A similar approach was presented in [2] where the liquid takes as input real data coming from a frame-based video camera that was mounted on a robot. The goal was to predict movement of the ball in the video stream of the robots camera. The sensor itself was a color camera with a resolution of 320 × 240 pixels. However, the frames were pre-processed and down-

Table 1. Hyper-parameters of the liquid. We chose them experimentally for the liquid to run in a good regime [14, 32]. They are kept constant across all evaluation scenarios.

Parameter	Variable	Value
Number of excitatory liquid neurons	n_{exc}	1000
Number of inhibitory liquid neurons	n_{inh}	250
Number of recorded liquid neurons	n_{rec}	500
Number of synapses between input and liquid	n_{inpsyn}	102 400

Table 2. NEST Parameters of the leaky integrate-and-fire neuron model with exponential shaped post-synaptic currents (*iaf_psc_exp*). Inspired by [33].

Parameter	Variable	Value
Resting membrane potential (mV)	E_L	0.0
Capacity of the membrane (pF)	C_m	30.0
Membrane time constant (ms)	τ_m	30.0
Duration of refractory period ($V_m = V_{reset}$) (ms)	t_{ref}	2.0
Membrane potential (mV)	V_m	10.0
Spike threshold (mV)	V_{th}	15.0
Reset membrane potential after a spike (mV)	V_{reset}	13.8
Time constant of post-synaptic excitatory currents (ms)	τ_{syn}^{ex}	3.0
Time constant of post-synaptic inhibitory currents (ms)	τ_{syn}^{inh}	2.0

Table 3. NEST Parameters of the distributions used for synaptic delays, weights and noise. Inspired by [33].

Parameter	Distribution	Mean	Deviation
Strength of the EE connection (pA)	Normal	5.0	$0.7 \cdot 5.0$
Strength of the EI connection (pA)	Normal	25.0	$0.7 \cdot 25.0$
Strength of the II connection (pA)	Normal	-20.0	$0.7 \cdot (-20.0)$
Strength of the IE connection (pA)	Normal	-20.0	$0.7 \cdot (-20.0)$
Delay of all the connections (ms)	Normal clipped [3.0,200.0]	10.0	20.0
Strength of the input current (pA)	Log-normal clipped [0.0,14.9]	2.65	0.025
Strength of the noise (pA)	Normal	1.0	$0.7 \cdot 1.0$
Delay of the noise (ms)	Normal clipped [3.0,200.0]	10.0	20.0

scaled before being fed to the liquid. A color blob detection algorithm was used to detect the ball in the image, which was presented to the 8×6 sensor field of the liquid. In the training scenarios, the ball was rolling with different velocities and directions across the field. The video sequences had different lengths and contained images sampled every 50 ms. The prediction took place over multiple time steps for 100 ms, 200 ms and 300 ms. It has been stated that it was possible to reliably predict ball movement up to 200 ms ahead, but that it was not possible to visually identify the ball position for the prediction time of 300 ms.

Echo state networks have recently been evaluated on event-based data in [12]. The goal was to learn spatiotemporal features for event-based data. Many feature-specific echo state networks are trained simultaneously to predict over receptive fields of the event stream. Address events are sampled in intervals and converted with an exponential filter before being fed to the echo state networks. This exponential filter is similar to the one we describe in equation (2), which we use to represent target signals but not to pre-process input events. In the conducted experiments, an echo state network predicts a single motion over 17×17 input cells, at most. Moreover, the predictions are provided for one timestep in the future. Unlike [12], we take in this paper a global approach where only one liquid state machine is trained to predict over the whole 128×128 pixel array. Additionally, we present results on how the prediction time Δ_{pred} affects the performance.

In this paper, we implement and evaluate a liquid state machine predicting future visual input from address events provided by a DVS. Compared to previous works [1, 2], we substantially scale up the input dimensionality by two orders of magnitudes by demonstrating the method on the whole 128×128 pixel array, without pre-processing. Moreover, the event-based nature of the DVS brings several advantages. Firtly, there is no issue of motion blur and we can sample the liquid state at high frequency, since the DVS continuously delivers address events. Secondly, the representation of the environment by the liquid is more efficient since the DVS only perceives changes. We propose a smooth continuous representation to generate target signals for the readouts (see equation (2) and figure 2) to deal with the task of predicting visual input from event streams. To the best of our knowledge, liquid state machines have not yet been evaluated in a realistic robotic setup with asynchronous full-size visual input.

3. The liquid state machine

In this paper, we propose a liquid state machine to predict visual input from address events using a DVS, see figure 1. This section describes the architecture of the liquid, along with the generation of the targets for the supervised training procedure.

3.1. Architecture

The liquid state machine consists of three distinct parts: an input layer, a liquid and an output layer. The input layer consists of spike generator neurons having the same dimensionality as the input. The address events emitted by the DVS are represented as spikes in

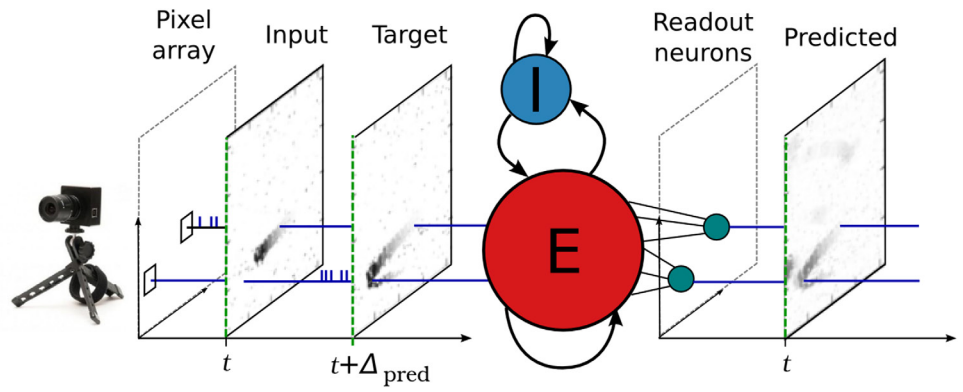


Figure 1. Our method to perform short-term visual predictions from address events provided by a DVS using a liquid state machine. Events from the DVS are converted to spikes and directly fed to the excitatory pool of neurons in the liquid. We use the function b_i (see equation (2)) to sample all input neurons i into target images used only for training. A layer of readout neurons of the same dimensionality as the input is connected to n_{rec} excitatory neurons of the liquid. Those connections are trained at time t with the target image sampled at time $t + \Delta_{\text{pred}}$. In the test phase, unseen events are streamed to the liquid and the activity of the readout neurons encodes the predicted future input. The input image is just shown for visualization purposes, only address events are streamed to the liquid. Reproduced with permission from iniVation AG.

the respective input neurons. Input neurons connect randomly in a feedforward fashion to excitatory neurons in the liquid. We use a constant number of synapses n_{inpsyn} between the input and the liquid.

The liquid consist of randomly connected leaky integrate-and-fire neurons, 80% of which are excitatory and 20% inhibitory. Such proportion is biologically inspired [1]. The liquid neuron parameters are described in table 2. We use dynamic synapses [20] for the connections within the liquid in order to stabilize the activity regime [21]. Furthermore, decorrelation between liquid neurons is introduced by drawing synaptic delays, current bias and connection strengths from a probability distribution. Parameters of these distributions are described in table 3.

We connect n_{rec} excitatory liquid neurons all-to-all to perfect linear readout neurons in a feedforward fashion. Only these connections are trained, with a procedure described in section 3.2. The readout neurons map the liquid activity to the desired output. Note that $n_{\text{rec}} = n_{\text{exc}}$ (i.e. all-to-all connection between internal excitatory liquid neurons and readouts) is a common choice [2, 19]. In this paper, we only map a fraction of the excitatory neurons to the output, which lowered memory consumption and did not impact the performance.

The connection strengths between input and liquid neurons, as well as within the liquid can be tuned to make the liquid memory driven or input driven. For example, by increasing the connection strengths between input and liquid neurons, the activity regime of the liquid will depend more on the inputs than its memory.

3.2. Training

The training part of a liquid state machine consists of finding the proper weights of the connections between the liquid and the readout neurons. Let $\boldsymbol{w_i} \in R^{n_{\text{rec}}}$ be the weights between a readout neuron i and the set of recorded liquid neurons. Training consists of finding the proper weights for all readout neurons covering the output pixel array $\boldsymbol{W} = [\boldsymbol{w_1}, ..., \boldsymbol{w_p}]$, with $p = \text{rows} \times \text{columns}$ the number of pixels.

The liquid state $\boldsymbol{x}(t) \in \mathbb{R}^{n_{\text{rec}}}$ is a vector containing the post-synaptic potentials of the n_{rec} recorded liquid neurons at time t. We sample the liquid state at discrete timesteps of Δ_{sample} for $t \in [0, t_{\text{train}}]$. They are accumulated into a matrix $\boldsymbol{X} \in \mathbb{R}^{n_{\text{rec}} \times n_{\text{samples}}}$, with $n_{\text{samples}} = t_{\text{train}}/\Delta_{\text{sample}}$. The matrix $\boldsymbol{X}$ contains all the liquid states encountered during training. We therefore have the linear system for all readout neurons i:

$$\boldsymbol{X} \cdot \boldsymbol{w_i} = \boldsymbol{b_i}, \tag{1}$$

with $\boldsymbol{b_i} \in \mathbb{R}^{n_{\text{samples}}}$ the sampled target signals of readout neuron i. In other words, each readout neuron solves a linear regression to map the accumulated liquid states to accumulated target signals for this neuron. To reduce overfitting, we use a regularized linear regression which penalizes strong synaptic weights. The regularization parameter λ is selected with respect to a cross-validation dataset.

Since we are using the liquid to predict future input, the target signals are built from input spiketrains shifted by a duration of Δ_{pred}. The duration Δ_{pred} is an hyperparameter and determines how long in the future we want to predict.

A proper representation of the visual information for the target signals is essential for robust learning. Indeed, naively trying to output binary values representing the presence of a spike in the time interval does not work well due to the instantaneous character of spikes. Instead, we generate target signals by converting an input spiketrain to continuous values between 0 and 1 depending on recent spiking activity, see figure 2. Specifically, we define the target signal b_i for neuron i:

$$\begin{cases} f_i(t) &= \exp(-\dfrac{t - t_i^{\text{spike}}}{\tau}) \\ b_i(t) &= f_i(t + \Delta_{\text{pred}}), \end{cases} \tag{2}$$

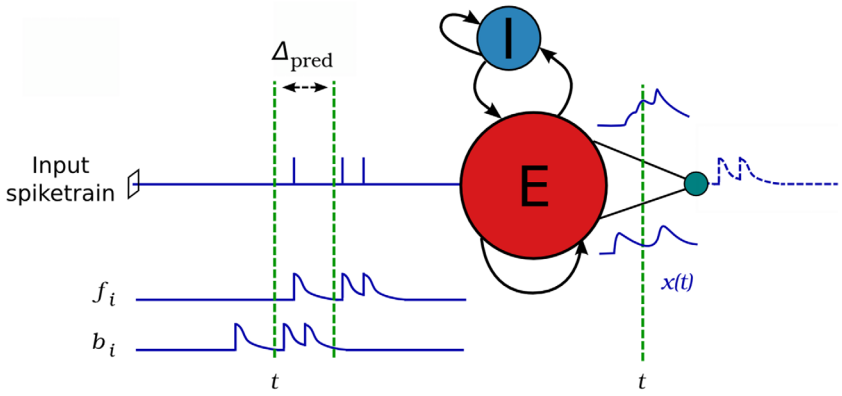


Figure 2. Visualization of the training of readout neuron i. To generate a target signal, we first apply f_i to map the input spiketrain to continuous values. It is then shifted by Δ_{pred} with the function b_i, see equation (2). During training, the readout neuron i learns a mapping between the liquid state $\boldsymbol{x}(t)$ to $b_i(t)$ for all times t.

with $t_i^{\text{spike}} \in [0, t]$ the last spike time of input neuron i, and τ a global fading term. the function f_i maps a spiketrain of a neuron i to a continuous signal, while the function b_i shifts this signal in time by Δ_{pred}, since we are performing a prediction task. Both f_i and b_i are visualized in figure 2.

It is important to note that, unlike classical frame-based methods, the samples taken for training do not coincide with the frame-rate of the sensor. Since both the DVS and the liquid output continuous streams, the parameter Δ_{sample} can be tuned without modifying the method or changing the sensor.

4. Evaluation

The liquid state machine is implemented using the NEST neural network simulator [22]. We evaluate the method both in simulation and with a real DVS. The DVS simulation is taken from [23]. It renders, subtracts and then thresholds consecutive frames of a classic simulated camera, and output the results as address events. With a sufficient simulated frame-rate, individual pixels can be emitted independently, just like a real DVS. This simulation is not as complex as the one presented in [24] but is more convenient since it can be used from the robotic simulator Gazebo [25] directly.

4.1. Experimental setup

The architecture and parameters of the liquid are described in table 1 and kept constant whether the events come from a simulated or a real DVS.

We use a reduced input array of 32×32 for the simulated DVS. The number of synapses between input layer and the liquid is kept constant regardless of the input dimensionality. That is, an input neuron in a 32×32 scenario will have more synapses with the liquid than an input neuron in a 128×128 scenario. This allows us to scale up and down input dimensionality without perturbing the regime of the liquid.

We evaluate the method in three scenarios. In the first scenario, a simulated DVS observes a ball performing distinct projectile trajectories overlapping with each other. With this scenario, we evaluate the ability of our method to predict different motions, based on its memory. In the second scenario, a real DVS observes a ball rolling down a structure. This scenario proves that the method can scale up to a real DVS in simple setups. In the third scenario, a real DVS observes a person juggling with three balls. This is an evaluation whether the method can be used on complex scenes with multiple objects in motion. The three scenarios can be visualized in figure 3.

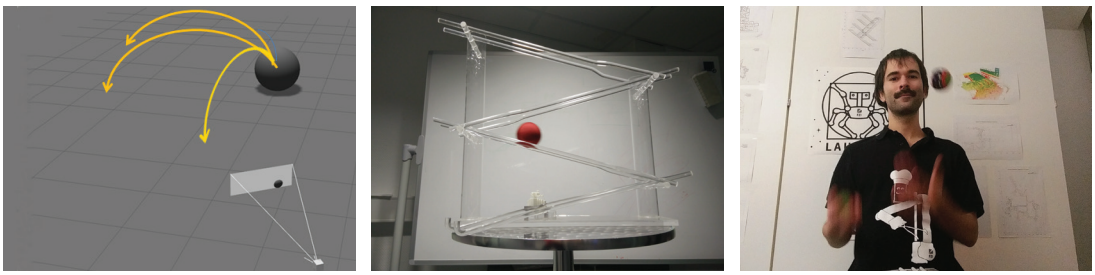
We cross-validate the regularization parameter λ two times: once for the simulated 32×32 DVS and once for the real 128×128 DVS. All others parameters are kept constant and described in tables 1–3.

In each scenario, we first present the emitted address events for a time t_{train} to the liquid and record the liquid states. We then train the readout weights with the procedure described in section 3.2. Finally, the liquid is presented new address events it has never seen before for a time t_{test}. We set the sampling time interval of the liquid to $\Delta_{\text{sample}} = 10$ ms. Predictions are generated by reading from the readout neurons at the same sample rate $1/\Delta_{\text{sample}}$ used for training.

We evaluate the performance of the method with respect to the predictions generated by the liquid. We define two metrics to evaluate the predictions. The first metric is general and consists of computing the normalized error for all predictions:

$$e_1(W) = \frac{1}{n_{\text{samples}}^{\text{test}} \cdot p} \cdot \sum_{i=1}^{p} \| X^{\text{test}} \cdot w_i - \boldsymbol{b}_i \|, \quad (3)$$

with $\boldsymbol{X}^{\text{test}}$ the accumulated liquid states during the test period, and $n_{\text{samples}}^{\text{test}}$ the number of samples in the test set. The residual error is the one we implicitly minimize when solving the linear equations for all readout neurons, see equation (1). This metric is not ideal for our problem since it does not take into account the 2D spatial structure of our pixel array. Indeed, if we predict high activation at a given pixel location, but the activated pixel turns out to be the neighboring one, we would like to penalize this less than if the activation was predicted at the other side of the image.



(a) (b) (c)

Figure 3. The three scenarios against which we validate our method. (a) The first scenario. A 32 × 32 simulated DVS observes a ball jumping from left to right and from right to left of the pixel array. (b) The second scenario. A real DVS observes a ball rolling down a structure. This image depicts the point of view of the DVS. (c) The third scenario. A real DVS observes a person juggling with three balls. This image depicts the point of view of the DVS.

The second metric we propose takes the spatial structure into account but is meaningful only when a single object is in motion. We therefore use this metric only for scenario 1 and 2, but not scenario 3 which has many objects in motion. The second metric consists of computing the distance between the centroid positions of the prediction and the target image. We define the centroid of an image as the average position of the activation:

$$\boldsymbol{c}(I) = \begin{bmatrix} r_c \\ c_c \end{bmatrix} = \frac{1}{\sum_{r,c} I(r,c)} \cdot \begin{bmatrix} \sum_{r,c} I(r,c) \cdot r \\ \sum_{r,c} I(r,c) \cdot c \end{bmatrix}, \quad (4)$$

with $\boldsymbol{I}$ an input image. The second error metric can therefore be expressed as:

$$\begin{aligned} e_2(W) &= \frac{1}{n_{\text{samples}}^{\text{test}}} \cdot \sum_{t \in \boldsymbol{t}^{\text{test}}} \|\boldsymbol{c}(\text{predicted}) - \boldsymbol{c}(\text{target})\| \\ &= \frac{1}{n_{\text{samples}}^{\text{test}}} \cdot \sum_{t \in \boldsymbol{t}^{\text{test}}} \|\boldsymbol{c}(\boldsymbol{x}(t) \cdot W) - \boldsymbol{c}(\boldsymbol{b}(t))\| \end{aligned}, \quad (5)$$

with $\boldsymbol{t}^{\text{test}}$ the sampled times during test, and $\boldsymbol{b}(t)$ the target signals for all readouts at time t. Here, we assume that both the prediction $\boldsymbol{x}(t) \cdot \boldsymbol{W}$ and the target signals $\boldsymbol{b}(t)$ are reshaped to images of size rows × columns before the computation of the centroid. In practice, when evaluating our prediction with this metric in the following scenarios, we also threshold low activations from the readouts before computing the centroid position. This has proved to greatly reduce the impact of the ambient noise inherent from the method.

4.2. Results

We first present the results obtained for the cross-validation of the regularization parameter λ. Then, we present the results obtained on the different scenarios using the selected regularization parameters. For each scenario, we show two representative prediction samples (figures 5, 8 and 11). *Input* and *Target* denotes the encoded input spiketrain at a given time t and $t + \Delta_{\text{pred}}$, see equation (2). *Predicted* is the output from the liquid at time t and *Error* refers to the residual error (equation (3)), which is the difference between *Predicted* and *Target*. In the quantitative evaluations, both error metrics (equation (3), solid blue line and equation (5), dashed red line) are presented with varying amount of training data and increasing prediction times Δ_{pred}.

4.3. Cross-validation

We first cross-validate the regularization parameter λ both for the simulated 32 × 32 and real 128 × 128 DVS. We follow the same setup as described in section 4.1 with constant prediction time $\Delta_{\text{pred}} = 200$ ms and λ varying on a log scale. The dataset used for cross-validation was held separate from training and testing sets.

As can be seen in figure 4(a), both metrics are minimal for $\lambda = 10^4$ for the simulated DVS. For the real DVS (figure 4(b)), the centroid error remains roughly constant for $\lambda \leqslant 10^3$ and then starts increasing. Meanwhile, the residual error is decreasing for $\lambda \geqslant 10^1$. This is due to the fact that the generated targets for the predictions have a sparse activation on the 128 × 128 pixel array (only few pixels are active at a time). Indeed, high values of λ will force the weights $\boldsymbol{W}$ to be small, hence the prediction activation to be weak. Therefore, a naive prediction algorithm returning null activations for all readouts at any time would have a low error according to the residual error metric. This is a common pitfall in statistics for problems containing a relatively small amount of positive samples. In such cases, precision and recall metrics are often preferred. However, it is not trivial to adapt these metrics to our spatio-temporal problem of prediction. We therefore pick $\lambda = 10^3$ for the real DVS scenarios.

4.4. First scenario: simulated projectile trajectories

The training set consists of 10 ball trajectories. In half of those, the ball jumps from left to right, and in the other half, from right to left. The total duration of the training set is 27.6 s. The test set consists of a single jump from left to right lasting 1.9 s.

Since the training set contains both left to right and right to left ball motions, the liquid has to rely on its memory to properly predict the test motion.

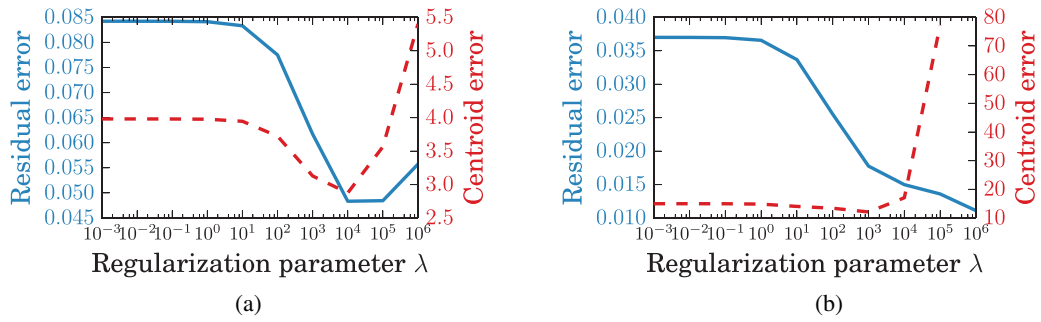


Figure 4. Cross-validation of the regularization parameter λ with the 32×32 simulated DVS and the 128×128 real DVS. Both metrics presented in equations (3) and (5) are shown. For the simulated DVS, the cross validation set consists of a ball jumping from right to left. For the real DVS, the cross validation set consists of a ball rolling down the structure. (a) Residual and centroidal error for the DVS simulation. Both error metrics admit a minimum at $\lambda = 10^4$. (b) Residual and centroidal error for the real DVS. The positional error is roughly constant for $\lambda \leqslant 10^3$, and then starts increasing. On the other hand, the residual error decreases for $\lambda \geqslant 10^1$. Indeed, high values of λ will decrease overall readout activation, and since the target activation is sparse on the large 128×128 pixel array, this metric decreases. We therefore consider the centroidal error metric and select $\lambda = 10^3$ for the real DVS scenarios.

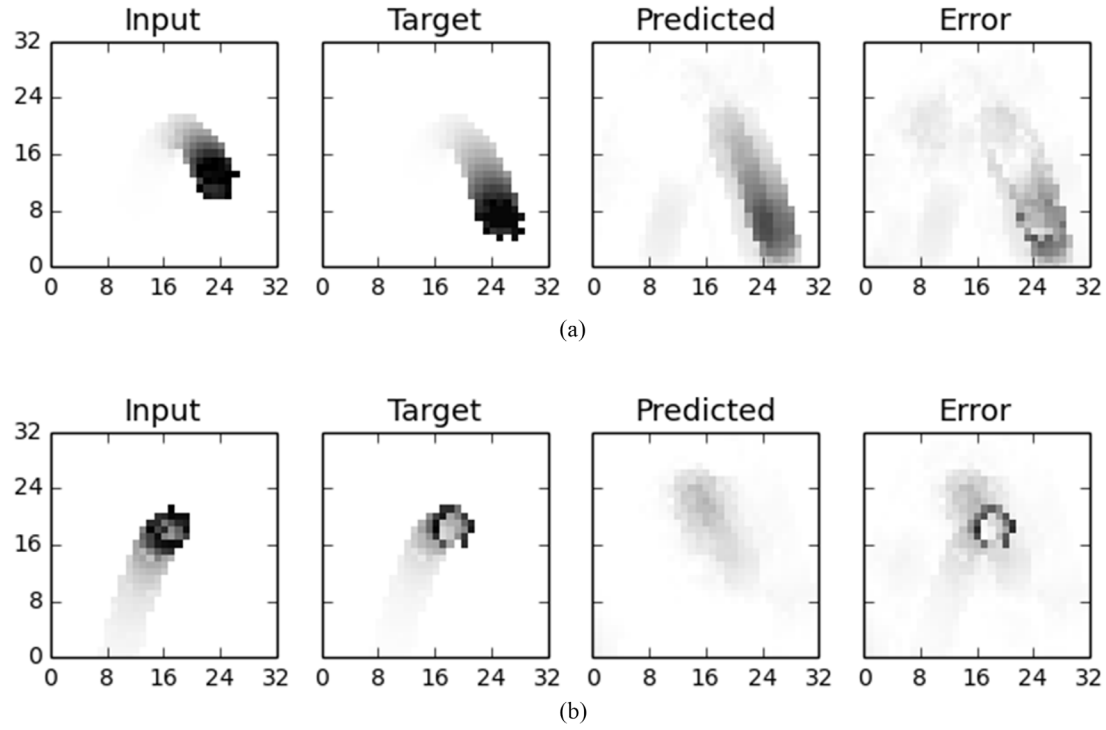


Figure 5. Selected test samples from the first scenario at prediction time $\Delta_{\text{pred}} = 200$ ms. (a) The ball starts descending after reaching its highest point, the liquid predicts the fall. (b) The ball is reaching its highest point where only a few address events are emitted. The prediction is weak, suggesting that our liquid is probably too input driven.

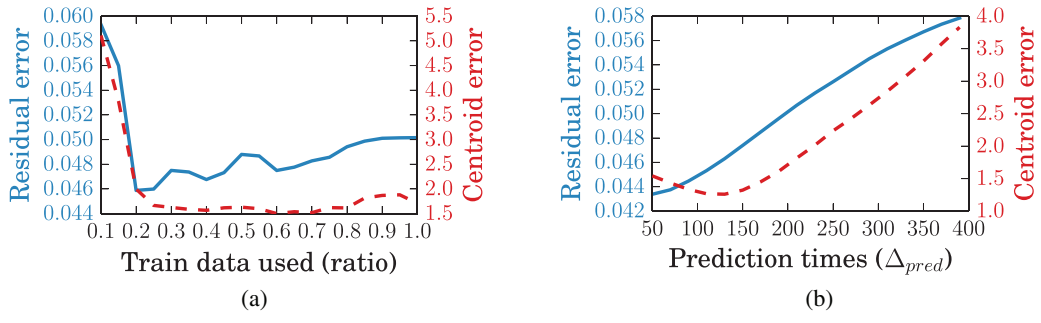


Figure 6. Residual error and centroidal error obtained in the first scenario: simulated ball jumping left and right. (a) Error with respect to the amount of training data used with $\Delta_{\text{pred}} = 200$ ms. (b) Error with respect to prediction time Δ_{pred}.

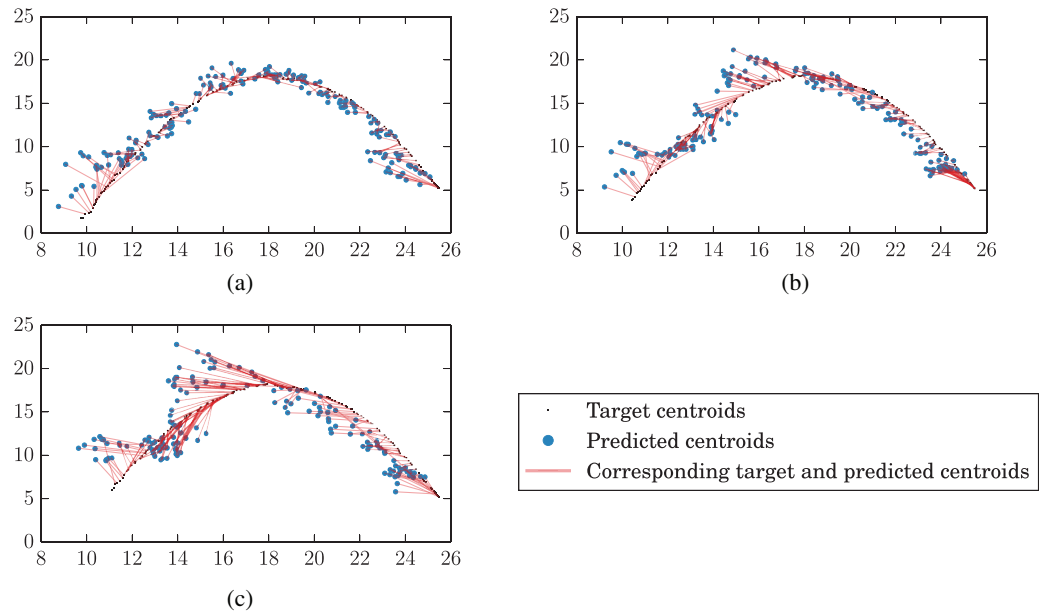


Figure 7. Target and predicted centroids for the first scenario across the whole test motion for varying prediction times. (a) Prediction time $\Delta_{\text{pred}} = 100$ ms. (b) Prediction time $\Delta_{\text{pred}} = 200$ ms. (c) Prediction time $\Delta_{\text{pred}} = 300$ ms.

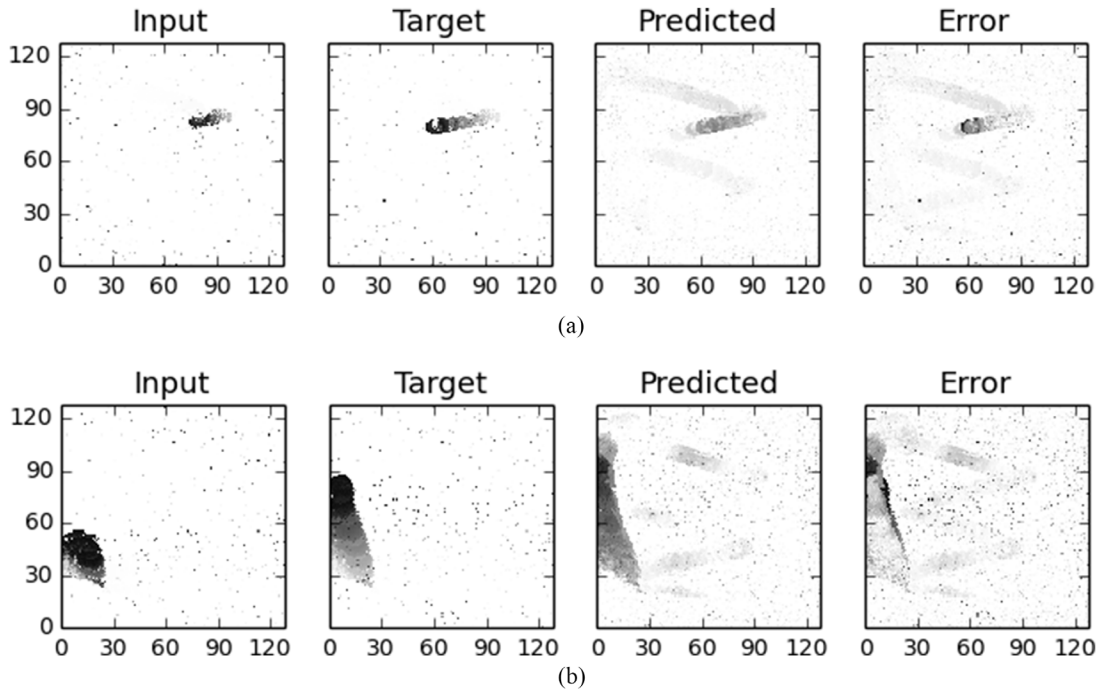


Figure 8. Selected test samples from the second scenario at prediction time $\Delta_{\text{pred}} = 200$ ms. (a) The ball is rolling on the structure, the liquid predicts its path. (b) The ball has reached the end of the structure, the liquid predicts the hand that will lift it up back to the top of the structure.

As seen in the samples (figure 5), the liquid can rely on its memory to properly predict the motion. Indeed, both ball motions (from left to right and from right to left) seen during training are covering the same pixels of the image but in different order. To provide successful predictions, the liquid is therefore remembering previous address events and not only the current ones.

The method needs to be trained with at least 20% of the training set before the yielding good predictions with respect to both our metrics (figure 6(a)). Indeed, since the training contains 10 complete motions of two classes, a portion below 20% does not contain the two motions entirely. This means that the method should be trained with similar motions as the one it tries to predict. For ratios bigger than 20% the residual error slightly increases but not the positional error. Therefore, it is not required that the training set contains multiple times the same motions (e.g. ball jumping left to right).

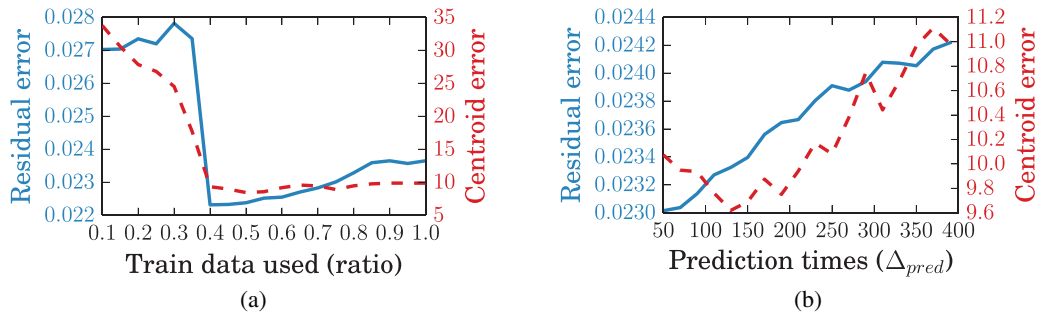


Figure 9. Residual error and centroidal error obtained in the second scenario: ball rolling on a structure. (a) Error with respect to the amount of training data used with $\Delta_{\text{pred}} = 200$ ms. (b) Error with respect to prediction time Δ_{pred}.

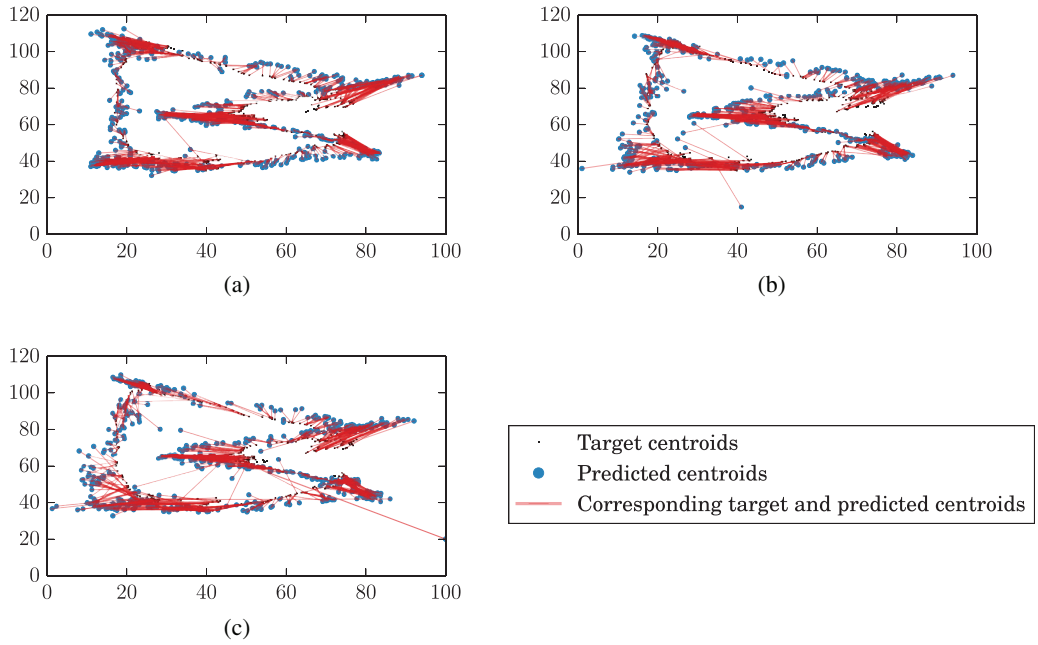


Figure 10. Target and predicted centroids for the second scenario across the whole test motion for varying prediction times. (a) Prediction time $\Delta_{\text{pred}} = 100$ ms. (b) Prediction time $\Delta_{\text{pred}} = 200$ ms. (c) Prediction time $\Delta_{\text{pred}} = 300$ ms.

Both residual and centroidal error metrics increase with Δ_{pred}, the amount of time predicted in the future (figure 6(b)). Before computing the centroid of the prediction, we nulled out the readout neurons with activation below 0.05, mostly caused by ambient noise. The target and predicted centroids are displayed for various Δ_{pred} in figure 7. The method successfully managed to learn the trajectory of the ball.

4.5. Second scenario: rolling ball on structure

The training set consists of the ball rolling two and a half times down the structure. When the ball arrives to the end of the structure, it is manually replaced on top of it. The total duration of the training set is 15 s. The test set consists of the ball rolling down the structure a single time lasting 6.8 s.

The method performs successful predictions for $\Delta \leqslant 200$ ms, as can be seen in figures 8 and 10. This proves that our method can scale up from 32×32 simulated DVS to the whole 128×128 real DVS without tuning any hyperparameter of the liquid. Before computing the centroid of the predictions, we nulled out the readout neurons with activation below 0.15. This threshold is slightly higher than for the first scenario because the regularization parameter $\lambda = 10^3$ is lower, yielding the active readouts to have stronger activations.

It seems that the liquid performs better predictions when the input is strong, such as the hand entering the image in figure 8(b).

Both error metrics exhibit the same behavior as the first scenario, showing the liquid learning capability (figure 9). The same conclusions can be drawn from the first scenario with regard to the content of the training set, see figure 9(a). This time, the method does not achieve good results before being trained with at least 40% of the training set, which coincides with the first full run of the ball on the structure. At 80% of the training set, the results did not improve, showing that it is not necessary to have multiple times the same motions in the training set. We note that the residual

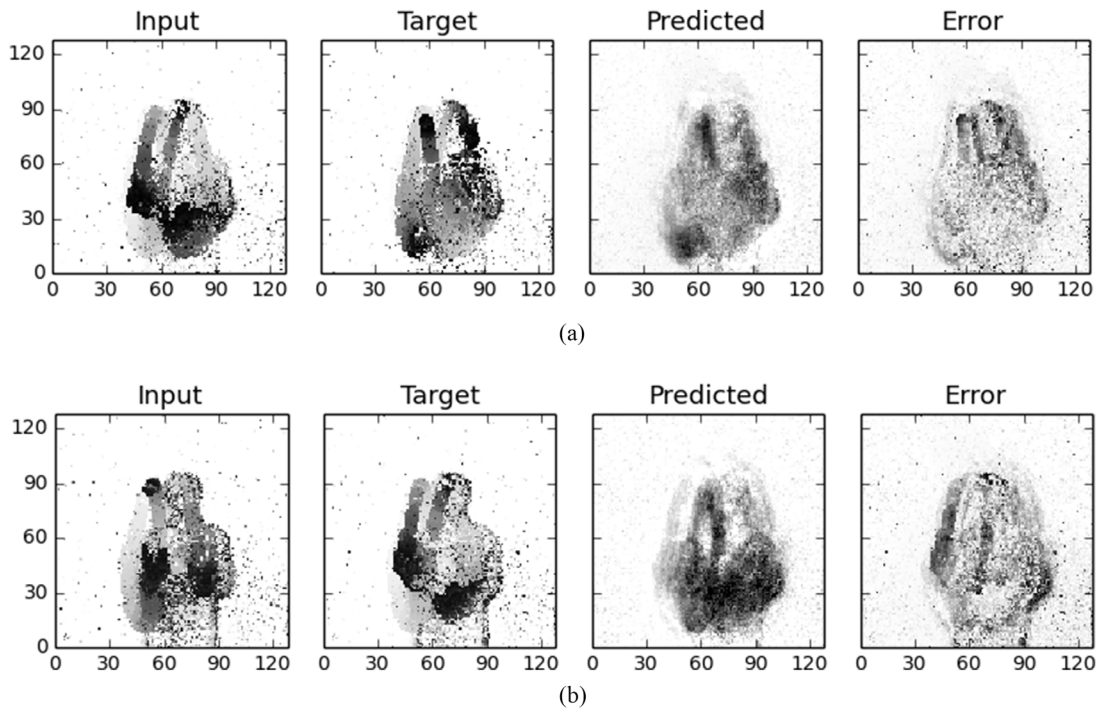


Figure 11. Selected test samples from the third scenario at prediction time $\Delta_{\text{pred}} = 200$ ms. (a) The hand on the left of the image has thrown a ball and is currently catching one. In the target image (200 ms later), the hand on the right just threw a third ball and is receiving the one seen on the input image. In the liquid output, we can recognize the position of the hands of the juggler and the two balls present in the target image. (b) This sample is the opposite of the previous one: a ball is thrown from the right and caught on the left of the image. The liquid predicted the ball trajectory but is confused with respect to the position of the hands.

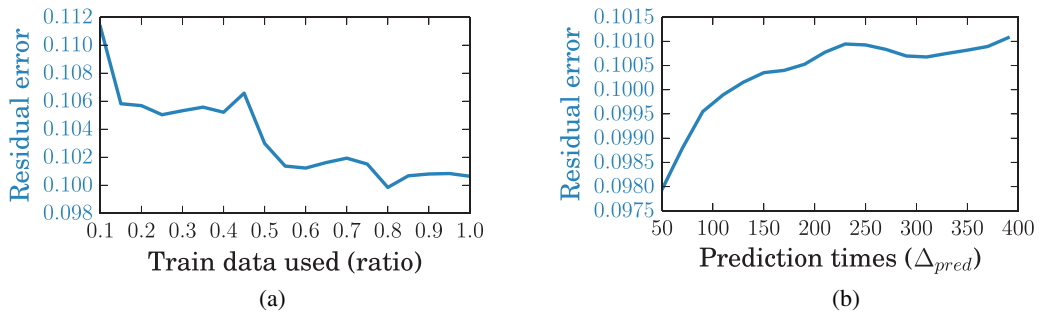


Figure 12. Residual error error obtained in the third scenario: juggling with three balls. The centroidal error is not relevant in this case: since the juggler himself does not move across the image, the centroid of activation is static. (a) Error with respect to the amount of training data used with $\Delta_{\text{pred}} = 200$ ms. (b) Error with respect to prediction time Δ_{pred}.

error is smaller with the real 128×128 DVS than with the simulated 32×32 DVS. This is because the residual error is normalized by the number of readouts (see equation (3)), and that the activation is sparser for the second scenario than the first one.

4.6. Third scenario: juggling with three balls

The training set consists of a juggler juggling with three balls for 29.7 s. The test set consists of the same juggler juggling for another 5.5 s.

Since there is more than a single moving object in the third scenario, the activation is more hectic. It also prevents us to use the centroid error metric, as in this case the centroid is the center of the juggler and remains roughly static over time. We can still recognize the juggler and the silhouettes of the balls in the liquid's predictions, see figure 11. However, these predictions are probably not sufficiently clear to be used in an actual robotic application. Unlike the previous scenarios, increasing learning data might help as the error keeps decreasing, as can be seen in figure 12(a). The residual error is much higher than in the other scenarios due to the motion covering more pixels.

This suggests that the input data should be abstracted before being fed to the liquid for predictions. Operating on abstracted visual input, the activation would be sparser, hence more similar to previous scenarios. Moreover, with convolutional layers, one could also make the method more translation invariant. However, the visual predictions would be made in the latent space instead of the input space. One should therefore prefer an abstraction method which can also reconstruct the input from the latent space, such as auto-encoders or restricted boltzmann machines [26].

Depending on the application, one might also want to segment the visual input stream. In this scenario, the liquid was not trained to predict what motions depend on each other. If the juggler would throw a single ball from a side to another, the liquid would predict another one coming from the other side, since it has been trained with a three ball juggling pattern.

5. Discussion and future work

In this paper, we presented a liquid state machine approach to perform short-term visual predictions from address events provided by a DVS. Based on spiking neural network, this method provides a framework for learning which handles the event-based nature of the DVS by design. Moreover, thanks to the asynchronous regime of the liquid, the predictions can be requested on demand at anytime, instead of being provided upon new sensor data. These advantages make liquid state machines good candidates to complement the prediction step of many robotic algorithms, especially when dealing with asynchronous sensors.

This paper is a step towards integrating liquid state machines, which are biological models derived in neuroscience, to real world robotic setups. The main contribution of this work is the evaluation of liquid state machines for short-term prediction on real-world full-size event streams. Properly representing visual target signal was a crucial step for the method to provide relevant predictions. By evaluating our approach on various scenarios of increasing complexity, we have shown that the method was able to learn different motions on the full scale 128×128 pixel arrays of the DVS, without any knowledge about its environment or physical laws. It is therefore well suited to fast robotic tasks where no knowledge about the environment is available. However, we identified in the third experiment scenario that the liquid was not able to deal with multiple objects in motion at the same time. Indeed, in a real robotic application, raw DVS events should be abstracted or segmented before being fed to the liquid. Unlike [2] which downscales the input image to a 8×6 sensor field, one could extract high-level features from the complete event stream with a convolutional spiking network [26, 27, 28]. Similar approaches already yielded promising results with analog neural networks [29].

The most interesting aspect of the liquid state machine resides in its genericity and the simple learning procedure. In the future, we could add input neurons to stream different information to the liquid. For instance, an inertial stream provided by an IMU could be combined with the DVS stream. A pool of readout neurons could then be trained to provide absolute scale self velocity. The liquid and the readouts would therefore perform a sensor fusion internally, without any change in the current architecture.

However, the simple learning procedure also admits drawbacks. Since it is based on a supervised off-line linear regression, data has to be collected and target signals have to be known. In literature, different approaches have been presented to overcome these constraints. In [21], a recursive least-squares algorithm is used, so that the liquid can be trained on-line while performing a task. Alternatively, several reward-modulated STDP rules [30, 31] have been suggested so that the method could learn in a reinforcement learning fashion, without known target signals. With an online learning rule to train the readout neurons, no training phase would be required and the robot would be able to learn new motions while operating.

Acknowledgments

The research leading to these results has received funding from the European Union Horizon 2020 Programme under grant agreement n.720270 (Human Brain Project SGA1).

References

[1] Maass W, Legenstein R and Markram H 2002 A new approach towards vision suggested by biologically realistic neural microcircuit models *Int. Workshop on Biologically Motivated Computer Vision* (Berlin: Springer) pp 282–93
[2] Burgsteiner H *et al* 2007 Movement prediction from real-world images using a liquid state machine *Appl. Intell.* **26** 99–109
[3] Kalman R E 1960 A new approach to linear filtering and prediction problems *J. Basic Eng.* **82** 35–45
[4] Kim S *et al* 2014 Catching objects in flight *IEEE Trans. Robot.* **30** 1049–65
[5] Allibert G and Courtial E 2009 What can prediction bring to image-based visual servoing? *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (St Louis, MO, USA, 10–15 October 2009)* pp 5210–5
[6] Mueggler E *et al* 2015 Towards evasive maneuvers with quadrotors using dynamic vision sensors *European Conf. on Mobile Robots (Lincoln, UK, 2–4 September 2015)* (https://doi.org/10.1109/ECMR.2015.7324048)
[7] Foka A F and Trahanias P E 2002 Predictive autonomous robot navigation *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems* vol 1 pp 490–5
[8] Tow A *et al* 2017 What would you do? Acting by learning to predict *IROS* in preparation (arXiv:1703.02658)
[9] Copete J L *et al* 2016 Motor development facilitates the prediction of others' actions through sensorimotor predictive learning *Joint IEEE Int. Conf. on Development and Learning and Epigenetic Robotics (Cergy-Pontoise, France, 19–22 September 2016)* pp 223–9
[10] Lichtsteiner P *et al* 2008 A 128 × 128 120 db 15 μs latency asynchronous temporal contrast vision sensor *IEEE J. Solid State Circuits* **43** 566–76
[11] Mueggler E *et al* 2015 Lifetime estimation of events from dynamic vision sensors *Int. Conf. on Robotics and Automation* pp 4874–81
[12] Lagorce X *et al* 2015 Spatiotemporal features for asynchronous event-based data *Frontiers Neurosci.* **9** 46
[13] Lukoševičius M and Jaeger H 2009 Reservoir computing approaches to recurrent neural network training *Comput. Sci. Rev.* **3** 127–49
[14] Grzyb B *et al* 2009 Which model to use for the liquid state machine? *Int. Joint Conf. on Neural Networks (Atlanta, GA, USA, 14–19 June 2009)* pp 1018–24
[15] Fernando C and Sojakka S 2003 Pattern recognition in a bucket *European Conf. on Artificial Life* (Berlin: Springer) pp 588–97
[16] Nikolić D *et al* 2009 Distributed fading memory for stimulus properties in the primary visual cortex *PloS Biol.* **7** e1000260

[17] Urbain G *et al* 2017 Morphological properties of mass spring networks for optimal locomotion learning *Frontiers Neurorobot.* **11** 16
[18] Burgsteiner H 2005 Training networks of biological realistic spiking neurons for real-time robot control *Proc. of the 9th Int. Conf. on Engineering Applications of Neural Networks (Lille, France) pp* 129–36
[19] Probst D *et al* 2012 Liquid computing in a simplified model of cortical layer iv: learning to balance a ball *Int. Conf. on Artificial Neural Networks* ed A E P Villa *et al* (Berlin: Springer) pp 209–16
[20] Markram H *et al* 1998 Differential signaling via the same axon of neocortical pyramidal neurons *Proc. Natl Acad. Sci.* **95** 5323–8
[21] Sussillo D and Abbott L F 2009 Generating coherent patterns of activity from chaotic neural networks *Neuron* **63** 544–7
[22] Gewaltig M-O and Diesmann M 2007 NEST (neural simulation tool) *Scholarpedia* **2** 1430
[23] Kaiser J *et al* 2016 Towards a framework for end-to-end control of a simulated vehicle with spiking neural networks *IEEE Int. Conf. on Simulation, Modeling, and Programming for Autonomous Robots (San Francisco, CA, USA, 13–16 December 2016)* pp 127–34
[24] Mueggler E *et al* 2016 The event-camera dataset and simulator: event-based data for pose estimation, visual odometry and SLAM (arXiv:1610.08336)
[25] Koenig N and Howard A 2004 Design and use paradigms for gazebo, an open-source multi-robot simulator *Int. Conf. on Intelligent Robots and Systems (Sendai, Japan, 28 Sept.-2 Oct. 2004)* vol 3 (IEEE) pp 2149–54
[26] Kaiser J *et al* 2017 Spiking convolutional deep belief networks *Int. Conf. on Artificial Neural Networks* submitted
[27] Kheradpisheh S R *et al* 2016 STDP-based spiking deep neural networks for object recognition (arXiv:1611.01421)
[28] Hunsberger E and Eliasmith C 2015 Spiking deep networks with LIF neurons 1–9 (arXiv:1510.08829)
[29] Finn C and Levine S 2016 Deep visual foresight for planning robot motion (arXiv:1610.00696)
[30] Kappel D *et al* 2017 Reward-based stochastic self-configuration of neural circuits (arXiv:1011.1669v3)
[31] Florian R V 2005 A reinforcement learning algorithm for spiking neural networks *Seventh Int. Symp. on Symbolic and Numeric Algorithms for Scientific Computing* (Synasc) (*Timisoara, Romania, 25–29 September 2005*) p 8
[32] Legenstein R and Maass W 2007 Edge of chaos and prediction of computational performance for neural circuit models *Neural Netw.* **20** 323–4
[33] Principles of Brain Computation, TU Graz 2016 http://igi.tugraz.at/lehre/PrinciplesOfBrainComputation/SS16/ (accessed: 08 October 2016)