Online Fitting of Computational Cost to Environmental Complexity: Predictive Coding with the ϵ-network

Lana Sinapayen[1] and Takashi Ikegami[1]

[1]The University of Tokyo, Japan
lana@sacral.c.u-tokyo.ac.jp

Abstract

We propose the Epsilon Network (ϵ-network), a network that automatically adjusts its size to the complexity of a stream of data while performing online learning. The network optimises its topology during training, simultaneously adding and removing neurons and weights: it adds neurons where they can raise performance, and removes redundant neurons while preserving performance. The network is a neural realisation of the ϵ-machine devised by Crutchfield and al. (Crutchfield and Young (1989)). In this paper our network is trained to predict video frames; we evaluate it on simple, complex, and noisy videos and show that the final number of neurons is a good indicator of the complexity and predictability of the data stream.

Introduction

In this paper, we propose a new neural network that fits the number of its neurons to the complexity of the input data. The network uses the concepts of surprise, short term memory and attention to optimise its topology. We answer this question: How can a network's experience influence neural generation and neural pruning, to reach optimal size in a given environment?

We think of the ϵ-network as a first step towards an implementation of decision making in embodied robotic agents. This is why the focus on online learning and computational cost is important in this work. A review by Attik et al. (2005) identified families of network topology optimisation methods (growing, pruning, regularisation) that we can also divide according to the timing of optimisation. Topology is usually optimised before training, especially in Deep Learning. The architecture of the network is decided before training, using parameter values known in the field for giving good results, generally obtained by trial and error. Topology can be optimised post-training: post-training optimisation uses pruning techniques aiming to reduce the network size while preserving the performance level. Finally, Genetic Algorithms are used to find optimal topology during training but are unsuited to online learning.

By contrast, our network optimises its topology during training, simultaneously adding and removing neurons and weights: it adds neurons where they can raise performance, and removes redundant neurons while preserving performance.

Predictive coding, the idea that intelligence and learning come from attempts to predict the world, has recently been gathering interest in the field of Artificial Intelligence thanks to the theoretical framework laid by Friston (Friston (2005); Friston and Kiebel (2009); Friston (2009)). Friston builds a framework aiming to explain how prediction and inference can explain not only learning in the brain, but also an agent's actions. The core idea is that agents strive to minimise "surprise" (unpredicted events). It is a theoretically appealing framework, but its extreme computational cost and the complexity of the core concepts makes it difficult to grasp, let alone implement in a computer, despite its having been used in a very simplified version as inspiration for implementing a deep learning predictive network (Lotter et al. (2015, 2016)).

Around the same time as Friston published his first papers on predictive coding, Hawkins published his own ideas on prediction: the Memory-Prediction Framework (Hawkins and Blakeslee (2004)). This theory aims to explain how the neocortex works; it does not have the advantage of a sound mathematical framework as Friston's work does, but Hawkins' work resulted in a fair number of applications and publications: for example Garalevicius (2007) shows a model capable of image recognition; Bundzel and Hashimoto (2010) are also able to do image recognition, in a moving robot; Rawlinson and Kowadlo (2012) coupled the framework to reinforcement learning to produce behaviour.

The concept of surprise is core in approaches where prediction reflects the performance of the network. Baldi and Itti (2010) define the concept of Bayesian surprise as the distance between the prior probability distribution and the posterior distribution, a definition closer to the human experience than the Shannon surprise (the occurrence of rare events). In our work, surprise itself is used to decide how to update the prior, therefore we use a different definition than both the Shannon and Bayesian definition of surprise. Here surprise is defined as the unpredicted activation of a neuron.

Carole Knibbe et al, eds., Proceedings of the ECAL 2017, Lyon, France, 4-8 September 2017,
(Cambridge, MA: The MIT Press, ©2017 Massachusetts Institute of Technology).

Computational cost is an issue often overlooked in Deep Learning, as we build ever more powerful computers. Yet even with these powerful machines, we still need hours or days to learn big datasets, and online learning is virtually impossible with a single computer. Biological organisms have solved this issue. We take the issue of adaptation, spending the right amount of energy in the right kind of setting during the lifetime of an organism, as the central topic of this paper.

We propose a model where the number of connections and the number of neurons can vary with time, to minimise both the surprise and the computational cost of learning, processing and storing the network.

Our network implements a concept that is deemed biologically plausible by Hawkins and Blakeslee (2004): hierarchically ordered "name cells", which are cells that act as pattern detectors. Yet we do not aim to directly emulate biology, but to tackle issues that biological organisms face while also being backed up from a statistical point of view by the work of Crutchfield and Young (1989). In that paper, Crutchfield proposes ϵ-machines as a measure of complexity for phenomena that fall on the spectrum between periodic and random behaviour. Complexity is, in his words, "the information contained in the minimum number of equivalence classes obtained by reducing a data set modulo a symmetry". Said symmetry is "time translation invariance, the symmetry appropriate to forecasting": complexity is predictability. ϵ-machines model phenomena through automata built from a stream of data. First, we build a hierarchical tree where nodes represent the data and edges represent the probability of transition between nodes. The nodes of the tree are then pruned to eliminate redundancy: two nodes with the same probabilities on successive edges are considered equivalent. The graph complexity of an ϵ-machine, in Crutchfield's words, "measures the computational resources required to reproduce a data stream". Our ϵ-network, like the ϵ-machine, results in a graph that can reproduce a data steam and adapt the computational resources to the complexity of the input data steam. The neurons are equivalent to nodes, the weights are equivalent to edges, and the pruning mechanism is functionally equivalent to Crutchfield's pruning methods. The main significant differences are that our ϵ-network has a growing mechanism based on the notion of surprise, several nodes can be active simultaneously as we deal with multivariate data streams, and we do not yet deal with the concept of indeterminacy of transitions.

As we calculate probabilities on events, it is easier to treat events as binary values "happened" or "did not happen". Therefore the activation of neurons is a binary value. The network is not completely binary, as the weights have real values. Binary Neural networks are typically less costly to train and run than real valued or spiking networks, but they also have worse performance. The main specificities of our approach compared to existing networks is that we propose online topology optimisation, and in our network the count of neurons reflects the complexity of the data. A disadvantage is that learning is not trivial (we do not use Genetic Algorithms or backpropagation), and it is not easy to compare our results with existing methods.

We present the results of the network on a clean versus a noisy dataset, and a simple vs a complex dataset.

Implementation

The ϵ-network learns transition probabilities between two successive inputs. Two kinds of neurons and two kinds of weights are used. The difference between the prediction computed by the network and the actual input at $t + 1$ is used to update the weight values. The number of neurons is increased to minimise the value of the surprise; simultaneously, redundancy is calculated by finding equivalent output probabilities on neurons, and redundant neurons are pruned. We detail the procedure in this section.

Network architecture

The ϵ-network's architecture and update rules are fundamentally different from what can be found in existing approaches. The network itself is composed of 2 types of neurons (regular neurons RN, and pattern neurons PaN) and 2 types of weights (prediction weights PrW and instantaneous weights IW). These elements are equivalent to a boolean circuit with OR gates (RN), AND gates (PaN) and wires (IW). The XOR function, that would grant to this network a complete set of boolean operators, cannot be formally computed but in practice there are some equivalences. During training, there is no null input to the network: not a (respectively not b) is exactly equivalent to another set of symbols X (respectively Y) being all TRUE. Using a XOR b = (a OR b) AND (X OR Y), the network does forms a complete set of boolean operators $\{$AND, XOR$\}$. The PrW, together with Short Term Memory and the attention mechanism, are an extra layer of information that allows to optimise the network. We define the network's components as:

- RN have a discrete activation value of 0 (not activated) or 1 (activated). A RN becomes activated when **one or more** input IW are activated.
- PaN have a discrete activation of 0 or 1. A PaN becomes activated when **all** input IW are activated. As a result PaN are only activated when their input pattern is present in the network.
- IW have one input neuron, one output neuron and a discrete activation of 0 or 1. An IW is activated or inactivated simultaneously with its input neuron, with no time delay, hence the name of Instantaneous Weight. It does not carry information on predictions.
- PrW have one input neuron and one output neuron. They have a discrete activation of 0 or 1, a continuous predic-

Carole Knibbe et al, eds., Proceedings of the ECAL 2017, Lyon, France, 4-8 September 2017,
(Cambridge, MA: The MIT Press, ©2017 Massachusetts Institute of Technology).

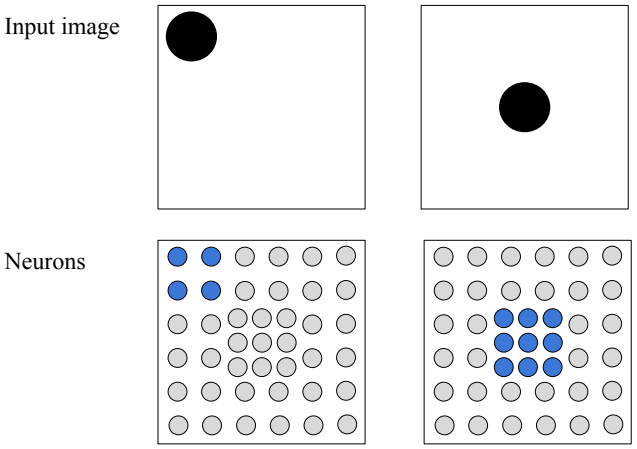

Figure 1: Before adding weights and neurons. The greyscale input is discretised so that the area has 6 sensors respectively activated by one of 6 shades of grey. Here we show the receptive field of neurons sensitive to black.

tion probability value between 0 and 1, and a discrete maturity value of 0 or 1. A PrW becomes activated when its input neuron is activated, but is inactivated with a delay of one timestep after the input neuron was inactivated. They carry information on the probability that the output neuron will become activated at the next time step. This information is used only for optimisation: PrW do not influence the activation of the output neuron. The rules to calculate the prediction probability are explained later in this paper. The maturity value determines whether this weight can still be updated or not, and is important when performing pruning of the network.

The network takes an array of external sensors as input (here, visual sensors). At the initial state, each RN has a receptive field as shown on Fig. 1. In our experiments, there is a higher concentration of RN in the middle of the image (1 sensor for an area of 2x2 pixels) than in the periphery (1 sensor for 5x5 pixels). There are 6 layers of sensors, each layer sensitive to one shade of grey.

Initially, there are no weights in the network. During one timestep: the predicted activation is calculated; activation is propagated through the network via the IW, then compared to the predicted activation to update PrW values. This loop is repeated at the next timestep (Alg. 1).

The probability values of PrW are updated as on Fig. 2 and Alg. 2.

The age variable of PrW counts the number of times that the input neuron was activated, while the $value$ variable counts the times when the output neuron was activated 1 timestep after the input neuron. p is therefore the exact probability for a neuron to be activated $t+1$ after another neuron. We can interpret this in terms of ϵ-machine: it is the proba-

```
for each sensor do
    if sensor.activation == 1 then
        for each output IW do
            IW.activation = 1;
            IW.RN.activation = 1;
            //recursively activate IW.RN.IW etc
        end
    end
end
for each PaN do
    if all input IW.activation == 1 then
        PaN.activation = 1;
    end
end
for each PrW do
    if PrW.activation == 1 then
        update PrW.p;
        PrW.activation=0;
    end
end
for each element in RN, PaN do
    if neuron.activation==1 then
        for each neuron.PrW do
            PrW.activation=1;
        end
    end
end
for each element in IW, RN, PaN do
    element.activation=0;
end
```

Algorithm 1: Activation and weights update

```
/* Initialized when PrW is created:
   */
value=1;
age=1;
/* Each timestep                    */
while running do
    if PrW.activation == 1 &
    PrW.outputNeuron.activation == 1 then
        value=value+1;
    end
    if PrW.inputNeuron.activation==1 then
        age=age+1;
    end
    /* prediction value             */
    p = value/age;
end
```

Algorithm 2: Prediction probability update

Carole Knibbe et al, eds., Proceedings of the ECAL 2017, Lyon, France, 4-8 September 2017,
(Cambridge, MA: The MIT Press, ©2017 Massachusetts Institute of Technology).

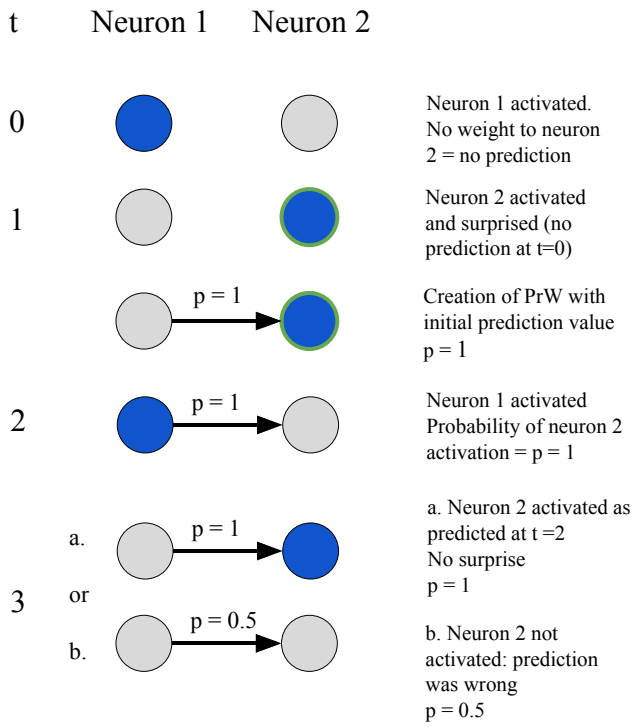

Figure 2: Rules of creation and update of a prediction weight. The weight is created if a neuron is surprised (the neuron is activated unexpectedly). The prediction value p is used to predict whether the output neuron will be activated at the next timestep, and updated during learning.

bility of going from one state to another state, each state here being a neuron. p is therefore a prediction value calculated from the past experience of 2 neurons; it is reminiscent of Hebbian rules, except that we get exact probabilities. The network takes multiple inputs and therefore, several neurons can be activated at a given timestep.

The ensemble of all the neurons having an input PrW satisfying $p > x$, $x = 0.8$ defines the predicted activation at $t+1$. The predicted input is $a = 1$ for the neurons with high probability of activation and $a = 0$ for all other neurons. The maturity value is defined as 0 if $age < M$, 1 otherwise. $M = 20$ is a constant defining how many times the probability value of a PrW can be updated. It avoids pruning neurons that are still actively in a learning phase.

Updating the number of neurons and weights

Increasing the number of PrW At the initial state, there are no PrW in the network. We need to create these weights from the neurons that were activated at $t-1$ to the neurons activated at t. For this we introduce the concept of Short Term Memory (STM). The STM simply stores the ID of the neurons that were activated a $t-1$. At t, PrW are created from all neurons in the STM to all currently activated neurons. PrW are not created if an older PrW already existed between an input neuron and an output neuron.

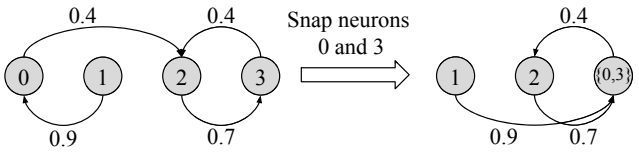

Figure 3: Example of snapping procedure: two neurons are fused together if they have the same output values; their input weights are reported to the fused neuron.

Decreasing the number of neurons and weights Decreasing the number of neurons is referred as the "snapping" procedure; this procedure is conducted periodically after a predefined number of timesteps (every 50 timesteps in this paper). If 2 neurons or more are "similar", these neurons are considered redundant and they are fused together (Fig. 3). The concept of "similarity" here is defined as: PrW are similar if they have the same output neuron and the same prediction value. Neurons are similar if all their output PrW are similar and none is immature. It is the implementation of the idea that if two events lead to the exact same consequences, these two events must be equivalent. Concretely in our paper, it means that a visual pattern or object that was represented by many neurons will now be represented by only one neuron; if two objects have the same motion dynamics they will be considered as the same object. It also fits nicely with the concept of "name cell" in the work of Hawkins, where the same "pattern" (a visual object, a song...) ultimately always activates the same dedicated "name cell". When 2 neurons are fused into one, all input IW and PrW are redirected to one neuron; the redundant weights are deleted, so the global number of weights is less or equal than before snapping. This falls outside the focus of this paper, but different choices could be made to report the input weights, leading to different consequences in terms of generalisation of inferences.

Increasing the number of PaN and IW So far predictions are made neuronwise (from one neuron to one other neuron), but some events require to know the state of several neurons to be correctly predicted (for example, an AND boolean operation). These events can be predicted by PaN. To detect whether a PaN is necessary, we introduce the notion of surprise, close to the idea of Friston and to Shannon surprise. Surprise is defined as a prediction error: the number of neurons which have an activation value of 1 while their predicted activation was 0. This is only one of several choices as a visual indicative of performance, all not optimal; but it is a good proxy for what is happening in the network as surprise is used to update the structure of the network. Surprise is therefore one measure of the performance of the network. When a neuron is surprised by an unpre-

Carole Knibbe et al, eds., Proceedings of the ECAL 2017, Lyon, France, 4-8 September 2017,
(Cambridge, MA: The MIT Press, ©2017 Massachusetts Institute of Technology).

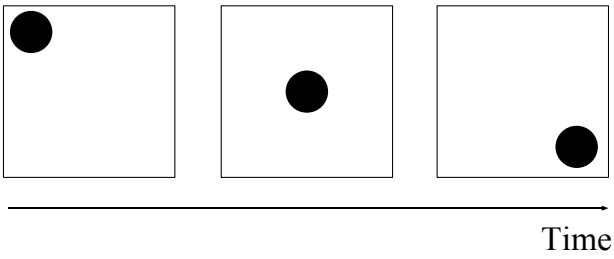

Figure 4: Simple sequence: ball falling

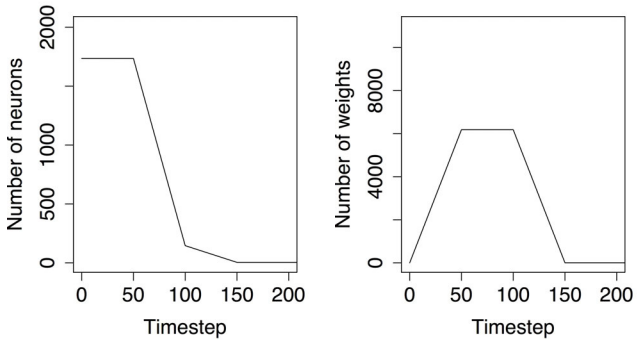

Figure 5: Evolution of the number of neurons (left) and of probability weights (right). After a few snapping procedures, the number of neurons falls to 4 (one neuron for each position of the ball and a 4th neuron gathering all unused inputs like the grayscale values absent from the dataset); after the initial increase, the final number of PrW is 3.

dicted activation event, a new PaN is created with input IW from all neurons in the STM, and one output PrW to the surprised neuron. This is repeated for all currently surprised neurons. Only PaN can be added to the network. Since the initial structure with one RN per sensor allows all possible AND operations between sensors, and PrW can be added between PaN, this choice does not introduce limitations on the network functionality.

Short Term Memory and Attention Mechanism In this paper we only consider predictions 1 timestep in the future, so the STM only stores neurons that were activated at the previous timestep. In addition, not all activated neurons can enter the STM. If a PaN is activated, the input neurons that caused its activation do not enter the STM. It makes sense as a pattern neuron "stands for" all the neurons that constitute this pattern, and therefore we can create input weights from the one PaN rather that from each one of the neurons that constitute this pattern. This is quite similar to attentional mechanisms in humans that allow us to perceive a face as a whole rather than as a group of separate parts.

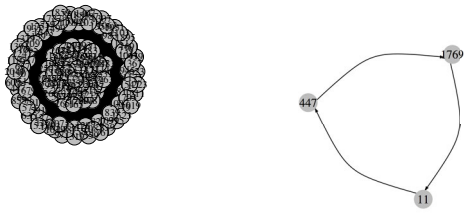

Figure 6: Evolution of the network's topology: $t = 50$ and $t = 200$. The final topology perfectly represents the automaton of the time series, with each neurons corresponding to a state and each weight to a transition with probability 1. The 4th neuron, with no output weights, is omitted.

Results

A. Minimal automaton

To show that the network does work as expected, we perform a simple modelling task where the data can be analysed by the naked eye. The time series consists of 3 frames (50x50 pixels) of a ball falling (Fig. 4). The network is trained by looping on these images.

The results are summarised Fig. 5. First, we can see that the number of neurons decreases until there are only 4 neurons left. The number of weights increases at first, then decreases proportionally with the number of neurons. At the end of the task, the network (Fig. 6) perfectly represents the automaton describing the time series: 1 neuron for each state of the automaton (equivalent to the position of the ball, see Fig. 7), with weight transitions of value 1 between the 3 states. The 4th neuron is the result of all unused neurons having been snapped together. In this dataset, for example there are no grey pixels: all neurons for those have no output weights and are detected as redundant. They are snapped into one single neuron.

Next we try a time series that requires the creation of PaN. We use a similar setting as before, with a variation for half of the trials (Fig. 8): one time out of two, a cue appears at the 2nd image and the 3rd image shows the ball in the upper right corner of the image. The results are shown on Fig. 9. This time 5 neurons are left, and 11 weights. The 5 neurons are the 3 same as the previous experiment, plus 2 pattern neurons for the two ambiguous situations ($t = 2$ and $t = 4$).

These are simple cases; in the next experiments, we perform modelling of more complex videos.

B. Clean or noisy environments

For this experiment, we use videos from the KITTI dataset (Geiger et al. (2013), Fig. 10), which are videos taken from a moving car's perspective. We resize the set $2011\_09\_26/image\_00$ (83 frames) to the size of 79x51 pixels. In the first experiment we use the images without further modifications; in the second experiment, we add

Carole Knibbe et al, eds., Proceedings of the ECAL 2017, Lyon, France, 4-8 September 2017,
(Cambridge, MA: The MIT Press, ©2017 Massachusetts Institute of Technology).

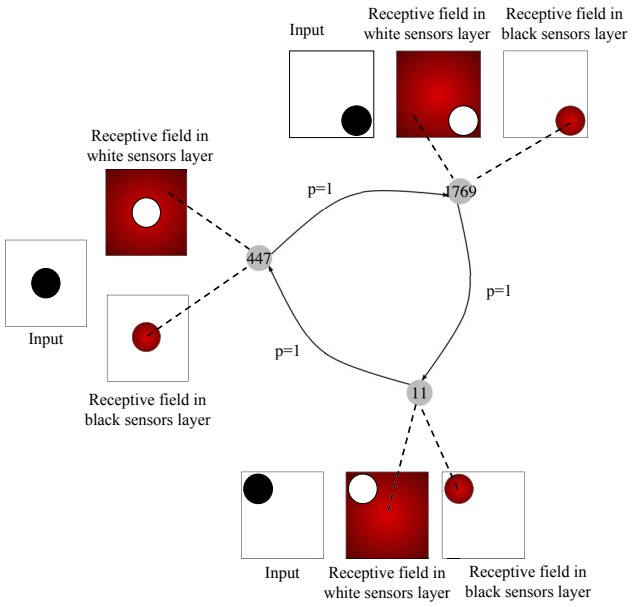

Figure 7: Detail of the final receptive field of each neuron. As the result of the snapping procedure, each neuron can now be activated by any of the sensors in large receptive fields, as activation of any of these sensors is sufficient to predict the next frame accurately. The receptive fields span the width and height of the image but also several greyscale values. From the neurons activation, we can still deduce which image the network is seeing.

random noise to the image at each timestep: 10% of the pixels are selected at random and to the greyscale value ($0 <= g <= 255$) we add noise from the uniform distribution on the interval $[-50, 50]$). Fig. 11 presents the results of both experiments: the final number of weights and neurons is smaller without noise, and bigger with noise. The network correctly represents the noisy dataset as having higher complexity. We can also note that the numbers continue to slightly increase in the noisy experiment, due to the unpredictable nature of the noise. To solve this issue we could decrease the threshold for surprise, or use the "graph indeterminacy" method of Crutchfield, a measure of the ambiguity of transitions between states. This is discussed at the end of this paper. Finally, Fig. 12 shows the evolution of the surprise value for the first 100 timesteps. Both networks end up with the same performance, despite the noisy experiment requiring more neurons and weights. Initially the network is "naive" and the surprise is very big until the 5th frame. The value slowly decreases as the networks learn the data. At t=82 (black line), the video sequence starts a new loop which is a sudden and highly surprising event for the network. This peak of surprise comes at the end of each loop during the whole experiment.

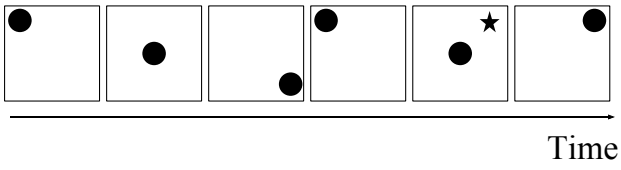

Figure 8: Sequence with a cue: we introduce a variation in the ball position on the 3rd and 6th frames.

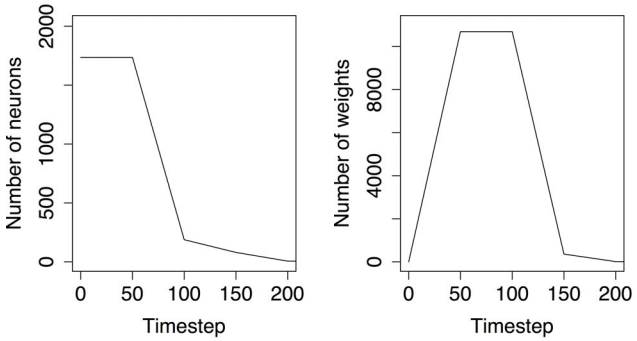

Figure 9: Evolution of the number of neurons (left) and of probability weights (right) for the modified sequence. The final number of neurons is 6; the number of probability weights rises more sharply than for the simple sequence, but then decreases from more than 9,000 to only 11 PrW.

C. Complex data stream and simple data stream

For this experiment, we use sequences from the public domain 1932 cartoon "Oswald the Lucky Rabbit: Mechanical Man" (Lantz and Nolan (1932)). We sample the video at 5 frames/s and resize it to 50x50 pixels. We use two sequences of 40 frames. In the first experiment, we use a sequence where the background almost does not change, and in the foreground two characters do an almost periodic motion. In the second experiment, we use an action packed sequence where the background changes, the action in the foreground has no repetitive elements, and there is one change of scene in the middle of the sequence (Fig. 13).

The results are summarised in Fig. 14: although both networks start with the same number of neurons, for the simple sequence the final number of neurons is almost half that of the complex sequence. The number of connections is also much lower. Another interesting result is that it takes less than a 1 complete loop (40 frames, black line) for the surprise to decrease in either network (Fig. 15), which shows that the network is not simply memorising the whole video sequence. Instead, early experience is generalised to later frames. In addition, the prediction performance (surprise) is the same for both tasks, which means that it has reached optimal value despite the differences in the two tasks. This optimal value is not 0, as prediction cannot be perfect if we use only very short term dependency (our network only has

Carole Knibbe et al, eds., Proceedings of the ECAL 2017, Lyon, France, 4-8 September 2017,
(Cambridge, MA: The MIT Press, ©2017 Massachusetts Institute of Technology).

Figure 10: Examples of frames from the KITTI dataset

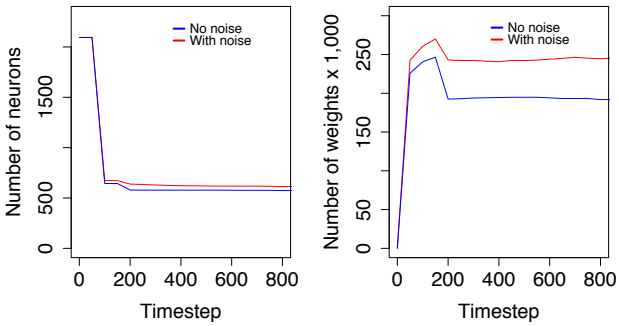

Figure 11: Evolution of the number of neurons (left) and of probability weights (right) on a clean video (blue lines) and a noisy version of the same video (red lines). The final results capture nicely the complexity induced by trying to predict a noisy dataset.

a 1 timestep memory). This performance stays stable even as the size of the networks decrease drastically.

Discussion

We introduced the ϵ-network, a network that automatically adjusts its size to the complexity of a dataset while learning to predict its inputs. The work of Crutchfield on ϵ-machines and our own results on the "falling ball" dataset suggest that this network could be able to find the optimal number of neurons and weights for a given task. Once optimised for parallel computing, our approach could have interesting advantages over classical Deep Learning methods: besides the ability to perform online learning and the bypassing of hand tuning hundreds of parameters (number of layers, number of neurons), the reduced size of the network should make computation less costly, while the growth procedure would insure that we still have enough neurons to tackle the task at hand.

This is relatively important for computers, but even more important for living organisms. Although we do not suggest that such drastic changes in brain density happen in animals, this network has a similarity with famous cases of hydrocephaly (water in the brain) where humans retain complete abilities while only a thin outer layer of brain cells remains alive.

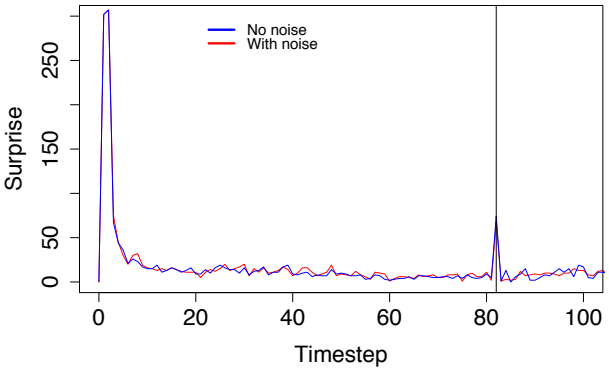

Figure 12: Evolution of the surprise in the network with and without noise. The value slowly decreases as the networks evolves. At $t = 82$ (black line), the video sequence starts a new loop which is a sudden and highly surprising event for the network.

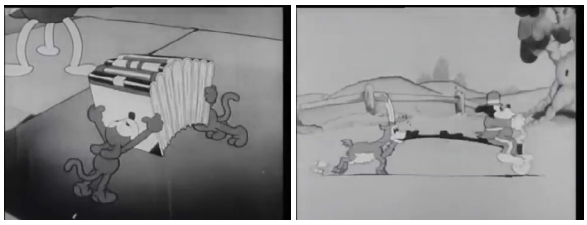

Figure 13: Frames from a simple sequence (2 characters play the accordion in almost periodic motion, static background) and a complex sequence (a character rides a bicycle, the background changes)

To improve the ϵ-network, we could increase the capacity of the short term memory. Although computationally costly at the beginning of tasks, this might lead to even smaller networks at the final stage of the learning task, as deeper temporal relationships are captured. For example, the modified falling ball task could theoretically be solved with only 3 neurons instead of 6 with a bigger short term memory. We are also in the process of implementing actions and proprioception in the network; early results suggest that this reduces the final size of the network for complex datasets. It is also of prime importance to introduce in the network the notion of indeterminacy of value of sets of weights at defined by Crutchfield. This is what allows ϵ-machines to represent simple noise with simple graphs: random noise is not predictable, but it is statistically simple: it is unsurprising in the Bayesian definition. In other words, our network should not seek to predict the unpredictable, but instead recognise some classes of events as unpredictable and unsurprising (e.g. white noise on a TV screen becomes unsurprising to

Carole Knibbe et al, eds., Proceedings of the ECAL 2017, Lyon, France, 4-8 September 2017,
(Cambridge, MA: The MIT Press, ©2017 Massachusetts Institute of Technology).

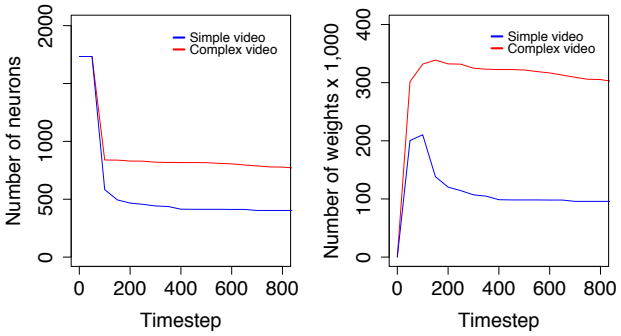

Figure 14: Evolution of the number of neurons (left) and of weights (right) for the two experiments. Although both videos have the same number of frames, the number of neurons and connections is much higher for the complex video sequence.

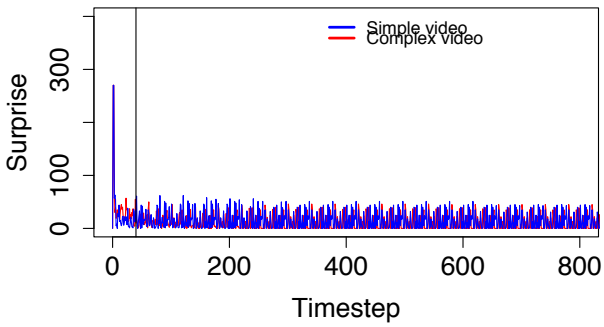

Figure 15: Evolution of the surprise in the simple and complex tasks. The surprise decreases sharply at the beginning of either task, then stays stable at the network optimises its size.

human beings past the first minute, although the exact visual patterns cannot be predicted).

References

Attik, M., Bougrain, L., and Alexandre, F. (2005). Neural network topology optimization. *Artificial Neural Networks: Formal Models and Their Applications–ICANN 2005*, pages 748–748.

Baldi, P. and Itti, L. (2010). Of bits and wows: a bayesian theory of surprise with applications to attention. *Neural Networks*, 23(5):649–666.

Bundzel, M. and Hashimoto, S. (2010). Object identification in dynamic images based on the memory-prediction theory of brain function. *Journal of Intelligent Learning Systems and Applications*, 2(04):212.

Crutchfield, J. P. and Young, K. (1989). Inferring statistical complexity. *Physical Review Letters*, 63(2):105–108.

Friston, K. (2005). A theory of cortical responses. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 360(1456):815–836.

Friston, K. (2009). The free-energy principle: a rough guide to the brain? *Trends in cognitive sciences*, 13(7):293–301.

Friston, K. and Kiebel, S. (2009). Predictive coding under the free-energy principle. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 364(1521):1211–1221.

Garalevicius, S. J. (2007). Memory-prediction framework for pattern recognition: Performance and suitability of the bayesian model of visual cortex. In *FLAIRS Conference*, pages 92–97.

Geiger, A., Lenz, P., Stiller, C., and Urtasun, R. (2013). Vision meets robotics: The KITTI dataset. *International Journal of Robotics Research*.

Hawkins, J. and Blakeslee, S. (2004). *On Intelligence*. Times Books.

Lantz, W. and Nolan, B. (1932). Mechanical man, oswald the lucky rabbit series. https://www.youtube.com/watch?v=gOkXItHRnsg. Accessed: 2017-04-12.

Lotter, W., Kreiman, G., and Cox, D. (2015). Unsupervised learning of visual structure using predictive generative networks. *arXiv preprint arXiv:1511.06380*.

Lotter, W., Kreiman, G., and Cox, D. (2016). Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104*.

Rawlinson, D. and Kowadlo, G. (2012). Generating adaptive behaviour within a memory-prediction framework. *PloS one*, 7(1):e29264.

Carole Knibbe et al, eds., Proceedings of the ECAL 2017, Lyon, France, 4-8 September 2017,
(Cambridge, MA: The MIT Press, ©2017 Massachusetts Institute of Technology).