SOM-based System for Sequence Chunking and Planning

Martin Takac[1,2], Alistair Knott[1,3], and Mark Sagar[1,4]

[1] Soul Machines, Ltd., Auckland, New Zealand
[2] Comenius University in Bratislava, Slovakia
[3] University of Otago, New Zealand
[4] University of Auckland, New Zealand
{martin.takac, alistair.knott, mark.sagar}@soulmachines.com
https://www.soulmachines.com

Abstract. In this paper we present a connectionist architecture called C-block combining several powerful and cognitively relevant features. It can learn sequential dependencies in incoming data and predict probability distributions over possible next inputs, notice repeatedly occurring sequences, automatically detect sequence boundaries (based on surprise in prediction) and represent sequences declaratively as chunks/plans for future execution or replay. It can associate plans with reward, and also with their effects on the system state. It also supports plan inference from an observed sequence of behaviours: it can recognize possible plans, along with their likely intended effects and expected reward, and can revise these inferences as the sequence unfolds. Finally, it implements goal-driven behaviour, by finding and executing a plan that most effectively reduces the difference between the current system state and the agent's desired state (goal). C-block is based on modified self-organizing maps that allow fast learning, approximate queries and Bayesian inference.

Keywords: sequence learning; self-organizing maps, planning

1 Introduction

In many domains of human action, it is important to learn *sequences* of items, and to represent these sequences declaratively. Sometimes these sequences are actions produced by the agent; in this case, the agent needs to find sequences that are useful, or rewarding. Sometimes, they are sequences of stimuli perceived in the world; in this case, the agent needs to identify frequently occurring stimuli. In either case, the items being sequenced can be of many different types: the sequences can be of motor actions, or perceived movements, or spoken words, or whole events.

There is good evidence that the brain makes use of a *general-purpose* circuit for learning to recognise, perform, and represent sequences, which is *replicated* for different cognitive faculties, that sequence different types of item. This idea is at the heart of many current general models of cognition, e.g. [1, 3, 7].

In this paper we present a connectionist model of such a circuit. We name it the ‘C-block’ circuit, where the ‘C’ stands for **chunking**. The process of chunking is the process of learning a *declarative* representation of a temporal sequence of items. The representation of a whole chunk can serve for *identification* (inferring what chunk is being generated after seeing its first few elements), or *generation* (where the chunk serves as a motor schema: a high-level representation of a sequence of actions, that guides generation of this sequence).

In many contexts, the elements of a temporal sequence can be thought of as actions that have an effect on the global state of the system, whatever that state is. Learned chunks then correspond to plans that take the system from one state to another. The planning component of C-block learns associations between chunks and their effects (potentially including an externally provided reward signal). This allows the C-block to operate in a goal-driven mode, wherein it selects a plan supposed to reduce as effectively as possible the differences between the system’s current state and a desired goal state, or a plan most likely to bring about an expected reward. Plan selection is dynamic: each time a plan completes (or fails), the new current state is used to recompute a new difference between the current state and the goal state, and the plan which most effectively reduces this new difference is selected.

We have been successfully using C-block in BabyX [9]—a hyperrealistic virtual simulation of a human baby combining models of the facial motor system and models of basic neural systems involved in interactive behaviour and learning. Different C-block instances are used for sequencing of the baby’s own motor movements, and for sequencing a wide variety of events she can perceive in the world, from low-level events involved in the production of facial expressions, to high-level events associated with utterances in a dialogue. In this paper we illustrate the operation of C-block on a simple artificial example of chunking sequences of letters into words.

2 The Architecture

The C-block consists of two components: a sequencer and a planner. The sequencer learns to predict the next element in a sequence, and in the process, learns declarative representations of whole sequences: that is, chunks. Chunk boundaries can be identified through two separate methods. One method implements the idea that chunk boundaries are associated with surprise [6]. The other method implements the idea that chunks are associated with reward states [4]. (We implement this by setting a ‘chunk boundary’ signal whenever a reward signal occurs.) At a chunk boundary, however it is identified, the planner learns a declarative representation of the preceding sequence, and associates this representation with the effect achieved by the whole sequence on a multidimensional system state, and with a reward signal (if one was provided).

The C-block circuit can operate in three different modes. In *observation mode* it receives a sequence of items, together with information about the global state and/or reward, and learns to represent chunks and their effects. At each moment, the circuit generates a Bayesian prediction about the most likely next item, and makes inferences about the plan that likely generated the sequence thus far, and the likely effect of this inferred plan. In *generation mode*, the C-block's predictions are used to actively influence the global state of the world, through motor actions.[5] Generation is goal-driven: the C-block begins with an active goal, then selects a plan that is expected to achieve this goal, then predicts a sequence of items that implement this plan. The third *collaboration mode* is a combination of observation and generation. The C-block begins by observing a fragment of a sequence, and makes inferences about the likely plan (and goal) that produced it, as usual. If there is a pause in the observed sequence, and the C-block is confident in its inference, it can adopt the inferred goal itself, and actively produce the rest of the sequence. This kind of cooperation is seen in infants at an early age.

2.1 Architecture of the Sequencer

At the core of the **sequencer** is a neural network trained to predict the next item in the incoming sequence. Because the next item is often a function of the recent items, neural networks for sequence learning usually use recurrent connections that enrich the immediate input with a *context*—an exponentially decaying encoding of the history of preceding elements. A seminal solution for this task is the simple recurrent network (SRN) [2], trained with back-propagation of error, and using a softmaxed output layer. As long as the output representation is localist, i.e. there is one neuron for each possible next element, the softmaxed output can be interpreted as a probability distribution and standard measures such as entropy or KL-divergence can be applied to it. The SRN trained on the next element prediction task is known to learn transition probabilities between elements [2]. A SRN can be augmented with a tonic input that drives the prediction and biases it towards a particular declaratively represented sequence. This tactic was used, for instance, in Reynolds *et al.*'s [8] model of event segmentation, where the tonic signal represented an event and significantly helped to stabilise the prediction of the event's elements.

In the C-block, we use a recurrent version of the **self-organising map (SOM)** [5] as the sequence-learning engine, rather than a SRN. The reason for that is twofold. First, backpropagation is slow and it takes many training epochs before the prediction reflects transition probabilities implicit in the training data. Second, the SRN operates in one direction only: it predicts the next element of the sequence from the immediate input, the recurrent context and the declarative representation of the chunk. We would like the system to also predict the likely chunk based on the fragment of the sequence seen so far.

The recurrent SOM implementing the sequencer network is shown in Figure 1. A SOM basically learn declarative representations of common patterns on its input media. Unlike networks trained with backprop, a trained SOM can

[5] This happens outside the C-block proper. 'Reafferent' representations of executed actions are passed back to the system as perceptual inputs at the next time step.

operate with partial inputs, and reconstruct the missing ones. (Note that unlike a SRN, the sequencing SOM takes the 'next' item as one of its inputs.) A SOM basically implements an auto-associative memory over its inputs, by retrieving a best-matching stored weight vector for any partial or noisy input. We have significantly refined this ability with two modifications to the standard SOM:

1. Instead of using a simple Euclidean distance for finding the best match, we use a weighted Euclidean distance where differences in some parts of the vector can be emphasized, deemphasized, or even ignored (by setting the mixing weight to zero).
2. Instead of returning the weights of the best-matching neuron, we reconstruct the input as a combination of all weight vectors with mixing coefficients proportional to their degree of match. This effectively turns the SOM into a Bayesian probabilistic device computing the expected value of the input based on posterior probabilities that the input falls into categories represented by the neurons in the map (see the Appendix for details).

The SOM in Figure 1 thus learns to associate the next item (Next) with the most recent item (Recent), the context representing the history of previous elements (Context) and the Tonic—declarative representation of the whole sequence. It can be queried to predict the next item based on the Tonic, Content and Recent, or to estimate Tonic based on the remaining parts of the input vector.

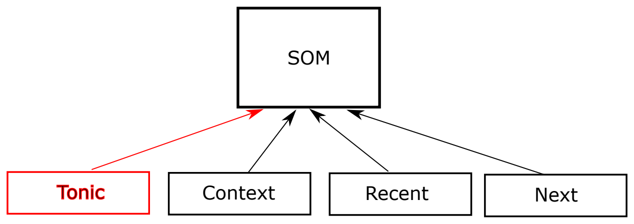

Fig. 1. The sequencing SOM: the C-block's sequence-learning device.

The Context is computed as an overlay/sum of localist representations of previous inputs (preceding the most recent item) exponentially decaying into the past. This implements a variant of the **merge SOM** [10].

To illustrate how this merge SOM-like recurrent architecture works, assume it is receiving a stream of letters, that spell people's names. Say the SOM has been trained on the word JAMES. When the first three letters (JAM) have been presented to the SOM, and the fourth letter E is then presented, the SOM's *Next* field will hold a localist representation of E. The *Recent* field will hold M (the previous input), and the *Context* field will hold representations of A and J, with the most recent of these most strongly represented.[6] When the letter E arrives, the SOM is trained on the input fields as just described. Then its plasticity

[6] More formally, we can write this $\mathrm{A}+c\cdot\mathrm{J}+c^2\cdot\mathrm{prev}$, where 'prev' is whatever preceded J, and $c<1$ is a decay coefficient, usually between 0.5–0.9.

is turned off, and the input fields are updated. Then three things happen: the *Context* field is multiplied by c, and the contents of the *Recent* field are added to it; the letter in the *Next* field (E) is copied to the *Recent* field; and the mixing weight of the *Next* field in the Euclidean distance is set to zero, so that the network starts predicting the next element from its other inputs. Thanks to the localist representation in the next layer and the SOM reconstructing the input from its full activity (instead of just the best-matching neuron), the result can be interpreted as a probability distribution over possible next letters. In this case, if it is already trained, it will predict S, with some confidence.

Besides this basic next element prediction functionality, the SOM also has an input field where a representation of chunks can be learned: the *Tonic* input, shown in red in Figure 1. As the name suggests, the contents of this field should have tonic, or persistent, activity, during sequential presentation of all elements of the chunk. In the letter sequencing example, chunks are names. Crucially, the circuit must have the capacity to *infer* possible chunks from an incomplete letter sequence. Since there can be several possible chunks consistent with the items presented so far, our *Tonic* input field holds an intermediate representation with *some* tonic properties, that is *underspecified* between alternative possible chunks.

In fact, our underspecified *Tonic* representation is the same kind of representation as that maintained in *Context* field: an overlaid combination of the items presented so far, but with the emphasis on the earliest, not the latest items. At the chunk boundary (based on an external signal or a mismatch in prediction), the chunk representation is reset and the *Tonic* field is simply set to hold a copy of the *Recent* field (which is the first item in a new chunk). If the chunk boundary occurs again, the *Tonic* will be filled in with the new *Recent*; otherwise it will be retained and the new *Recent* (multiplied with a decaying constant) will be overlaid on it.[7] In this way, the tonic and the context layers form complementary representations decaying in opposite directions.

We will illustrate the above scheme with an example of two words that start with the same subsequence: JOHN and JOSEPH. After both chunk representations have been successfully learned, they will both contain J as the most active item, O as the second most active and then decaying HN, or SEPH respectively. When a J is observed as the onset of a new sequence, the tonic representation will only contain J at that moment and will be consistent with both JOHN and JOSEPH (and any other chunk starting with J). The identity of the chunk will be progressively specified in a natural way as more items arrive. In a strict sense, the chunk representation is not tonic, because it is evolving in time, but because of its nature, there should be a high similarity between J, JO, JOH and JOHN.

The progressively specified *Tonic* representation supports the inference of fully-specified chunk representations in the other part of the C-block, the planner, which we will now describe.

[7] A similar approach to encoding chunks is used in [8], with the difference that they use a SRN instead of an SOM and their tonic representation follows a more complex update rule and is more noisy.

2.2 Architecture of the Planner

The complete C-block circuit is shown in Figure 2, with the sequencer (Sequencing SOM) on the right, and the planner (Plan SOM) on the left. The planner

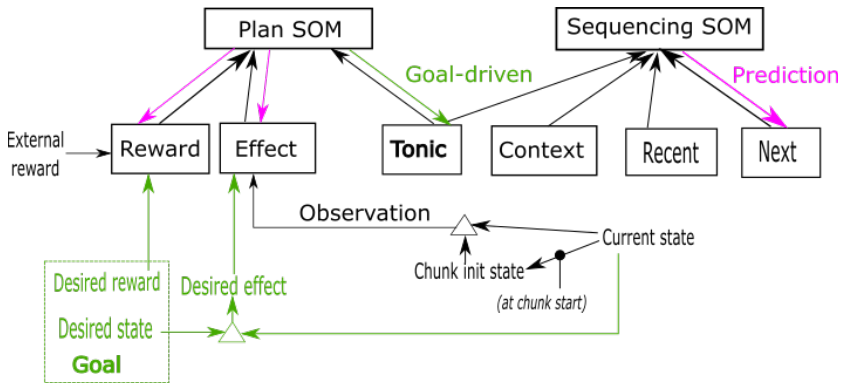

Fig. 2. The complete C-block, including the planner. During observation (black arrows), the system is trained to recognize the sequence and its outcome (reward and effect) from a fragment and predict its next element (purple arrows). In goal-driven mode (green arrows) it selects the best plan to achieve the goal and conditions the sequence generation on the tonic retrieved top-down from the planner.

is implemented by the same kind of modified SOM as the sequencer (with the weighted Euclidean distance for different parts of the input and option to either reconstruct from the best matching unit or from the full SOM activity). Its main function is to learn *declarative* representations of the chunks identified by the *Tonic* field of the sequencer, and to use these for goal-driven behaviour. Recall that representations of chunks are *gradually* constructed, item by item, in the sequencing SOM's *Tonic* field. The planner, by contrast, only learns its representations of chunks *at chunk boundaries*, when the representation of a chunk in the *Tonic* field is complete. Because of its autoassociative property, the trained plan SOM is able to reconstruct a full chunk representation from a fragment of an evolving sequence (either the best matching one or a mixture of several chunk representations proportional to their similarity with the fragment). In the letter sequencing example given above, if JO is presented, the activity in the plan SOM will be high in areas representing the names JOHN and JOSEPH and the retrieved tonic will reflect their mixture. But if the letter S is then presented, the plan SOM will update to a distribution highlighting only JOSEPH.

Associating Plans with Reward Values The planner is more than just autoassociative memory for tonic chunk representations. It can also learn associations between chunks and their effects/outcomes. The simplest outcome is a scalar reward state, encoded in the *Reward* input field of the plan SOM. When a chunk boundary is reached, the plan SOM not only learns a declarative representation of the chunk encoded in the *Tonic* input field, but also learns *an* *association between this chunk and the current reward.* After training of this type, the plan SOM can be queried with a high reward value, to retrieve plans associated with high reward, and then select one of these to execute. We will describe this process in more detail below.

Plans as State Update Operations The plan SOM can also represent chunks more semantically, as plans that have a particular effect *on the state of the world*, as represented by the agent. In the same way that the plan SOM associates a completed chunk with a reward value, it can also associate a completed chunk with a *state update* representation, in the *Effect* input field of the plan SOM.

In the C-block, our representation of 'state' is very general: it is an n-dimensional vector. We will discuss our conception of state in more detail below. We will begin by illustrating how plans can be associated with state updates.

Assume a simple state space with six dimensions, each occupied by either 1 or 0. Say the agent starts off in state [000011], and then performs a sequence of actions associated with a chunk $C1$, that leave it in the state [110011]. The plan SOM can learn to associate the chunk with a **state update operation**, that represents the *change in state* the chunk brings about. The change in state is just the difference ('delta') between the two states: in this case, [110000].

The utility in associating plans with state *updates*, rather than directly with states, is that updates *generalise* over the elements of the state which a plan leaves unchanged, and *focus* on the elements that need to be changed. Say the agent has the goal of achieving state [110000], and it is currently in state [000000]. The C-block computes a *goal state update* (in this case, [110000]), and then presents this goal update in the *Effect* input field to the plan SOM, as a query. Even though during training, chunk $C1$ resulted in the state [110011]—which is different from the current goal state—we are querying the plan SOM with a desired *change in state*, and the plan SOM can retrieve chunk $C1$ from this query. The overall paradigm here is that at the end of each chunk, the C-block computes a new goal state update, by taking the difference between the goal state and the current state, and then presents this goal state update as a query to the plan SOM. Note that the 'full' goal state update might decompose into several *separate* state updates, which are associated with distinct chunks. In this case, we have the equivalent of a partially ordered plan at the higher level, where some actions can be taken in an arbitrary order. For instance, say the plan SOM has learned two chunks, associated with updates [110000] and [001100]. If the agent is currently in state [000000], and desires to be in state [111100], both chunks will be (somewhat) activated, and can be performed in either order.

Attentional Modulation of Effect/State Inputs to the Plan SOM Not every aspect of the current state is equally relevant to the agent at any given time. It needs to *attend* to different aspects of the state vector at different times. For instance, if it is thirsty, it might pay more attention to a state dimension encoding the presence/absence of water.

These attentional emphases can be neatly encoded by setting the coefficients in the weighted Euclidean distance in the SOM calculations on each individual dimension of the *Effect* input field to the plan SOM, that encodes a desired state update. Dimensions with a zero coefficient will be ignored in the selection of plans, while dimensions with high values will have a significant influence in selecting the next plan. The same mechanism can also be used to weight the contribution of the *Reward* input field (which is entirely neutral to state) against the more content-focussed contribution of the *Effect* field.

2.3 Operation of the C-block in Observation and Generation Mode

We have already mentioned the difference between *observation* and goal-driven *generation* modes of operation of the C-block. More formally, the difference is charted in the arrows in Figure 2 that depict the direction of information flow. The crucial issue is how the *Tonic* input shared by the plan SOM and sequencing SOM is activated.

Observation mode In observation mode (black arrows in Figure 2), the tonic representation of the whole sequence is gradually constructed from the incoming inputs, as their overlay decaying into the future, and at each point is passed as a bottom-up input to the plan SOM for plan recognition. This SOM works as an autoassociator, retrieving the distribution over plans most likely to have produced the sequence seen so far.[8] This in turn is used by the sequencer alongside the context and the most recent element to predict the next element.

Generation mode In generation mode (green arrows in Figure 2), the plan SOM is first queried with a goal state update, and reward, appropriately weighted by attentional coefficients, to evoke an activity pattern representing the probability distribution over plans. The winning plan is then selected from this distribution, and used to reconstruct a representation in the *Tonic* input. This is then used as a tonically active input to the sequencing SOM to produce a sequence of inputs.[9]

The plan terminates in one of the following ways:

– Surprise: plan terminates unexpectedly because of a mismatch in prediction.
– Plan completion: the ‘end-of-sequence’ signal is predicted.
– Goal reached: a good match is achieved between the goal (desired effect and/or reward along with their attention coefficients) and the actual effect and reward.

[8] The distribution is encoded by the SOM activity. We can pick the most active plan, or reconstruct its expected value based on the whole distribution (see Appendix for details).

[9] Actually, what is produced is a sequence of *probability distributions* over possible inputs, from each of which a 1-hot input is selected.

An inhibition-of-return (IOR) scheme operates on plans, to encourage selection of different plans. When a plan is selected, it receives a small amount of inhibition, that decays slowly over time. If two plans are equally activated by a given goal/reward state, this inhibition means that if one plan is picked, and fails, the other plan is likely to be picked next.

3 Experiments

3.1 Training Set

We demonstrate the C-block operation on sequences of letters that form words as chunks. In order to demonstrate fast learning, the network was exposed to each letter sequence only once. At the end of each sequence the system got an external chunk boundary signal (EoS) together with high reward value $r = 1$: JASON(EoS)ANDREW(EoS)LIAM(EoS)CAROLINE(EoS)KHURRAM(EoS) MARTIN(EoS)ALI(EoS)MARK(EoS)DAVID(EoS)NICOLAS(EoS)TIM(EoS) PAULA(EoS)PAULINE(EoS)ROB(EoS)SIMON(EoS)OLEG(EoS)XUEYUAN (EoS)RACHEL(EoS)GREG(EoS)KIERAN. We also maintained a representation of the 'current state' after each letter input. To simulate the effects brought about by whole chunks, we updated this state representation at the end of each word, to create an 'effect' input for the network, as described in Section 3.2.

3.2 Network Parameters

The sequencing SOM had 50x50 neurons. Tonic, Context, Recent, Next fields had 100 neurons each. Incoming letters in the Recent and Next parts were coded 1-hot in their first 26 components.[10] The plan SOM had 30x30 neurons. The Tonic and Effect parts of its input had 100 components each; the reward was scalar (1 component). The Effect input was computed as a difference between the system's state at the beginning and end of the chunk. We represented the effect of generating the words listed above with localist coding (counters) in the first 20 components of the system's state. Each time a generated sequence happened to correspond to one of the words, the respective counter was incremented. For example, generating JASON incremented the first component, ANDREW the second, etc. Both SOMs had a high learning rate (0.9) and a small Gaussian neighbourhood size ($\sigma = 0.6$). The decay coefficient was 0.6 for both Context and Tonic and 0.4 for plan IOR.

3.3 Sequence Prediction and Plan Recognition

We began by testing the trained system in 'observation mode'. We randomly shuffled the words in the training set and played them to the C-block's input

[10] The remaining 74 components were zero all the time. This was because we used the same parameters and network sizes across several applications, in some of which we needed more neurons to encode the input.

letter by letter. We then measured the match between the most strongly predicted letter and the actual next letter.[11] The prediction was correct in 94.1% cases (80 out of 85). All 5 errors occurred at places with genuine ambiguity. For instance, when presented with the sequence MAR, which appeared in training both in MARK and MARTIN, it predicted K most strongly. If the test word happened to be MARTIN, then after seeing the next letter T, the C-block revised its hypothesis about the chunk, and correctly predicted I as the next letter. Given the Bayesian nature of our SOMs, both hypotheses K and T should have high probability in the soft-reconstructed Next field. To verify this, we also computed the Kullback-Leibler divergence (KLD) between the prediction in the Next field and the actual input in the next step. Because the actual input is always 1-hot, the KLD reduces to the negative log surprise $-\log_{26}(p)$ where p is the predicted activity at the position of the actual next letter. If the system's predictions about the next letter are accurate, we expect the mean value of surprise for all but the first position to be close to zero.[12] Indeed, mean KLD excepting first letters of words was 0.06 (SD=0.22).

Table 1. Success rate and the negative log surprise for predictions of the 'effect' of a chunk after seeing its first N elements.

N	1	2	3	4	5+
Ratio	15/20	18/20	18/20	16/17	28/28
Proportion	75.0%	90.0%	90.0%	94.1%	100%
Surprise mean (SD)	0.34 (0.40)	0.14 (0.29)	0.14 (0.29)	0.12 (0.25)	0.10 (0.21)

We also explored whether the C-block can accurately identify whole words (chunks) while their letters are arriving, and accurately predict the 'effect' of the currently arriving word on the global state. To do this, we inspected the system's predicted 'effect' field after each letter of each arriving word. We counted how often this prediction was correct (see Table 1), and we computed negative log surprise for the prediction after each letter (i.e. KLD with the effect that will actually occur). On all these measures, we found good performance from the earliest letters of words. Errors were again only found for ambiguous words. This shows the C-block is able to correctly identify chunks and predict their effects while they are still arriving.

3.4 Sequence Generation to Achieve Desired States

We then tested the model's ability to autonomously choose and generate sequences for a composite desired effect. To do this, we configured the C-block to 'want' to bring about a global state in which the effects associated with each word are all true, by setting the first 20 components in the 'Desired State' input to 1, and setting the attention coefficients for Effect to 1, and for Reward and

[11] We did this for all but the first position in a chunk, because C-block has no means to predict the continuation of a chunk before it has started.

[12] Predicting the winner with probability $p = 1$ leads to KLD=0. Flat predicted distribution $p = 1/26$ leads to KLD=1.

Tonic to 0. We then ran the C-block in goal-driven generation mode. Because the C-block updates the difference between the desired state and the actual state at each chunk boundary, it was then able to execute all the plans without repetition (in random order). All words were generated with all letters at correct positions.

3.5 Sequence Generation to Achieve Reward States

We also tested the model when configured to ‘replay rewarding plans’. To do this, we set the attention coefficients for Reward to 1, and for Effect and Tonic to 0 and set the Desired reward to 1; then we again ran the C-block in goal-driven mode. In this mode, the C-block first selected and generated the word most strongly associated with reward. On completion of this word, its inhibition-of-return mechanism led it to select a different word, and cycle through a varied sequence of words. All letters were again generated at correct positions.

3.6 Collaboration mode

Finally, we tested the model in ‘collaboration mode’, by presenting it with the first two letters of a word and allowing it time to complete the intended word itself. The C-block did so correctly in all cases (except for ambiguous cases MARK/MARTIN and PAULA/PAULINE where it completed the first option).

4 Summary

The C-block is a network that can learn sequences, and learn to group particular sequences into chunks. In this paper, we have shown how the C-block can use these abilities both in perception, to predict sequences and upcoming states, and in action, to generate sequences based on higher-level goals. While most sequencing models in current AI use backprop-trained deep networks, SOM-based models have some interesting strengths of their own, which we have tried to emphasise here. For one thing, they can be configured to learn very rapidly, as in the experiments we report. They also have great versatility: because a SOM can operate flexibly with different patterns of input and output, the same network can be co-opted for a variety of sequence-related functions, both in perception and motor control.

A Bayesian Inference in the SOM

In our version of SOM, the activity A_i of each unit is computed as

$$A_i = \frac{a_i}{\sum_{j=1}^{N} a_j}, \text{ where } a_i = exp\left(-c \cdot d^2(\boldsymbol{x}, \boldsymbol{w}_i)\right) \cdot m_i \ . \tag{1}$$

$d^2(\boldsymbol{x}, \boldsymbol{w}_i)$ is the squared Euclidean distance between the input $\boldsymbol{x}$ and the weight vector $\boldsymbol{w}_i$, a_i is the (unnormalized) activity of the i-th unit, m_i is the activation mask for the i-th unit. Activities A_i are normalized to sum to 1.

Comparing Equation 1 with the standard Bayes' rule

$$p(h_i|d) = \frac{p(d|h_i) \cdot p(h_i)}{p(d)} = \frac{p(d|h_i) \cdot p(h_i)}{\sum_{j=1}^{N} p(d|h_j) \cdot p(h_j)} \tag{2}$$

we can interpret the activity of each unit as the posterior probability $p(h_i|d)$ of the *hypothesis* that the current SOM input (*data*) belongs to the class represented by the unit i. The Gaussian[13] term $\exp\left(-c \cdot d^2(\boldsymbol{w}_i, \boldsymbol{x})\right)$ corresponds to the likelihood $p(d|h_i)$. The mask m_i corresponds to the prior probability of the i-th hypothesis $p(h_i)$. The denominator $\sum_{j=1}^{N} a_j$ in the formula for normalized activities A_i is a total response of the map to the current input and corresponds to $\sum_{j=1}^{p}(d|h_j) \cdot p(h_j) = p(d)$, which is just the probability of the data itself. A very low total activity in the map indicates strange (or novel) input data.

By specifying coefficients m_i, we can choose different prior bias on the SOM, for example relative frequency of how often the i-th neuron became the best-matching unit in the past. All m_i equal to the same value would effectively mean a uniform prior and will have no influence.

Normalized activity of the whole SOM corresponds to the posterior probability distribution over all the hypotheses/neurons given the current input/data. We can reconstruct the most likely input either as the weights of the winner ('hard' output) or as an activity-weighted combination of the weights of all the neurons ('soft' output): $\boldsymbol{y} = \sum_{j=1}^{N} A_j \cdot \boldsymbol{w}_j$, which corresponds to the expected value of the input given the distribution.

References

1. Bar, M.: Predictions: A universal principle in the operation of the human brain. Philosophical Transactions of the Royal Society, Series B **364**(1521), 1181–1182 (2009)
2. Elman, J.: Finding structure in time. Cognitive Science **14**, 179–211 (1990)
3. Friston, K.: The free-energy principle: a unified brain theory? Nature Reviews Neuroscience **11**, 127–138 (2010)
4. Graybiel, A.: The basal ganglia and chunking of action repertoire. Neurobiology of Learning and Memory **70**(1–2), 119–136 (1998)
5. Kohonen, T.: Self-organized formation of topologically correct feature maps. Biological Cybernetics **43**, 59–69 (1982)
6. Kurby, C., Zacks, J.: Segmentation in the perception and memory of events. Trends in Cognitive Sciences **12**(2), 72–79 (2007)
7. Parr, T., Friston, K.: The anatomy of inference: Generative models and brain structure. Frontiers in Computational Neuroscience **12**, 90 (2018)
8. Reynolds, J., Zacks, J., Braver, T.: A computational model of event segmentation from perceptual prediction. Cognitive Science **31**, 613–643 (2007)
9. Sagar, M., Seymour, M., Henderson, A.: Creating connection with autonomous facial animation. Communications of the ACM **59**(12), 82–91 (2016)
10. Strickert, M., Hammer, B.: Merge SOM for temporal data. Neurocomputing **64**, 39–71 (2005)

[13] c regulates the width of the Gaussian. We used $c = 30$ for sequencing SOM and varied $c = \{15, 25, 2\}$ for plan SOM predicting from Tonic, Reward and Effect respectively.