SYDE 556/750
Simulating Neurobiological Systems
Lecture 11: The Semantic Pointer Architecture

Andreas Stöckel

Based on lecture notes by
Chris Eliasmith and Terrence C. Stewart

March 24 & 26, 2020

Accompanying Readings: Chapter 3, 5, 6 and 7 of “How to Build a Brain”

Contents

1 Introduction

Note: In the previous lecture, we discussed Vector Symbolic Architectures (VSAs) in combination with circular convolution as a way to represent symbols as vectors and to "combine" concepts into a new one. However, VSAs alone are not a "cognitive architecture". It is unclear how to build "cognitive agents" that receive information from the environment, turn this information into symbol vectors, and finally produce *behaviour*. The semantic pointer architecture (SPA) is an attempt at providing a cognitive architecture based on VSAs.

As we have seen, we can use Vector Symbolic Architectures in conjunction with circular convolution to solve some relatively challenging cognitive problems, such as Raven's progressive matrices. However, so far, we have presumed that a certain vocabulary of random symbol vectors exists and that there is some kind of system in place that extracts these symbol vectors from the environment.

In this lecture we are going to be a little more specific. We are going to describe an architecture based on symbol vectors that hypothesises how sensory information could be turned into symbol vectors, how these vectors a processed, and, lastly, how symbol vectors can be turned into motor commands. This architecture is called the "Semantic Pointer Architecture" (SPA).

At a smaller scale, the SPA can be used to build neural models of cognitive phenomena such as working memory or language association tasks. At a larger scale, the SPA has been used to build one of the largest functional brain models to date, the Semantic Pointer Architecture Unified Network (SPAUN) [1, 2]. SPAUN is able to solve a variety of cognitive tasks, using the same underlying neural network and without being "reprogrammed". This ability is a major strength of the SPA in contrast to other models of cognition.

In this lecture we will first have a broader look at the SPA, including the concepts of "shallow and deep semantics". Then, we have quick look at SPAUN, and in particular a specific model of human serial working memory, followed by a discussion of cognitive control and action selection in particular.

2 The Semantic Pointer Architecture

Note: The content of this section is to a large degree based on Chapter 3 of "How to Build a Brain" [3]. Read the chapter for a more thorough discussion of semantic pointers.

In a nutshell, the Semantic Pointer Architecture combines the idea of vector symbolic architectures (VSAs) with the NEF as a "computational substrate". In addition, the SPA proposes, first, a specific cognitive architecture—the basal ganglia cortex control loop.

Second, symbol vectors in the SPA are no longer random, but thought to be the result of a reversible *compression* process. Just as a pointers in most programming languages can be "dereferenced" to access the original content, symbol vectors $\mathbf{x}$ constructed in this way can be *decompressed* to access the information—"deep semantics"—about the underlying represented object. Hence, symbol vectors in the SPA are called *semantic pointers*.

2.1 Shallow and Deep Semantics

In order to gain a better understanding of semantic pointers, we must first distinguish between "shallow" and "deep" semantics. The difference between the two can best be explained by what artificial intelligence researchers refer to as the "symbol grounding problem".

In general—and without going into any philosophical details—a *symbol* is an arbitrary mapping from a short piece of information (the representation of the symbol itself) onto a richer semantic concept.

For example, consider the symbol TREE. As a human being seeing this symbol, we tend to immediately map it onto a certain class of tall plants, along with the corresponding visual imagery, smells, sounds, and textures. All this cross-modal information establishes the *deep semantics* of the symbol, i.e., our "subjective experience" of the concept.

Now consider an early 1980s artificial intelligence researcher building a symbolic reasoning system. In such a system the symbol TREE might just be represented as a short sequence of bytes (i.e., TREE is `0x54 0x52 0x45 0x45` in ASCII), along with a set of similarly encoded information linking this symbol to other symbols, for example

$$\forall x \mathbf{is\_a}(x, \texttt{PINE}) \rightarrow \mathbf{is\_a}(x, \texttt{TREE}) \wedge \mathbf{has}(x, \texttt{NEEDLES}) \wedge \mathbf{is}(x, \texttt{EVERGREEN}),$$
$$\forall x \mathbf{is\_a}(x, \texttt{TREE}) \rightarrow \mathbf{is\_a}(x, \texttt{PLANT}), \quad \forall x \mathbf{is\_a}(x, \texttt{PLANT}) \rightarrow \mathbf{is}(x, \texttt{ALIVE}).$$

The problem of "symbol grounding" is the observation that even though we could imagine building a database with all important information there is to know about a concept, the symbols we use in this relational description (the *shallow semantics*) are not really linked or related to any sensory information or "subjective experience".

Aside: A project that aims to build exactly such a database is Douglas Lenat's Cyc project, started in 1984. The database is still under active development. (`https://www.cyc.com/`)

The random symbol vectors we discussed in the last lecture are as arbitrary a mapping onto concepts as the relationship between the byte sequence `0x54 0x52 0x45 0x45` and an actual tree. Such vectors will thus at most be useful for *shallow* semantics. This is similarly true for some non-random methods deriving symbol vectors.

One example of such methods are *word embeddings*. Here, the general idea is to perform some statistical analysis of how often certain words appear closely together in large corpora of text, usually resulting in a term-frequency matrix. One can then apply a dimensionality reduction method (such as PCA or auto-encoders) to this data, mapping each word onto a vector with a few hundred dimensions.

Interestingly, such mappings capture relational semantics of the represented concepts. For example, using the "word2vec" embedding [4], we can perform the following arithmetic[1]:

$$\texttt{WOMAN} + (\texttt{KING} - \texttt{MAN}) \approx \texttt{QUEEN},$$
$$\texttt{SPACE} + (\texttt{SHIPS} - \texttt{WATER}) \approx \texttt{SPACESHIP}.$$

While these embeddings are great as part of VSAs, they do not capture any "deep semantics".

[1] See `http://turbomaze.github.io/word2vecjson/` for an online demo. Ignore concepts in the output that are already present in your input.

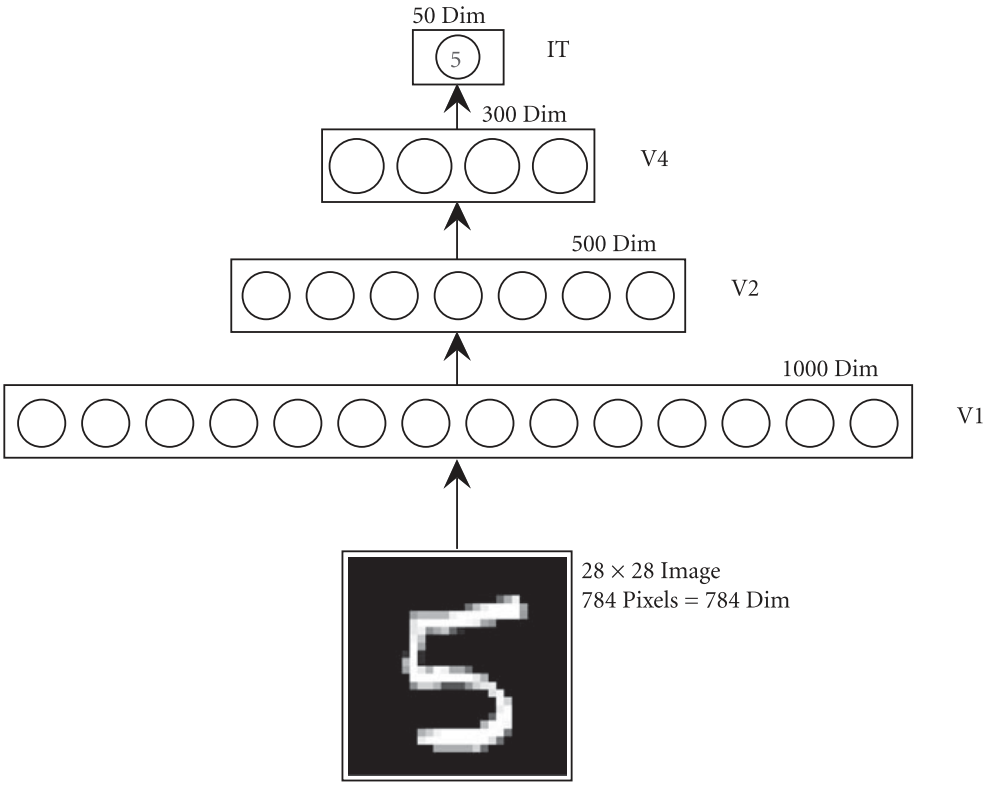

Figure 1: Visual processing hierarchy. The representation in the uppermost layer corresponds to a semantic pointer. Notice that there is a direct relationship between the input and the representation. Changing the visual stimulus (i.e., different styles of "fives") will be captured in the final representation. Figure copied from Eliasmith, "How to build a brain", 2013, Figure 3.4.

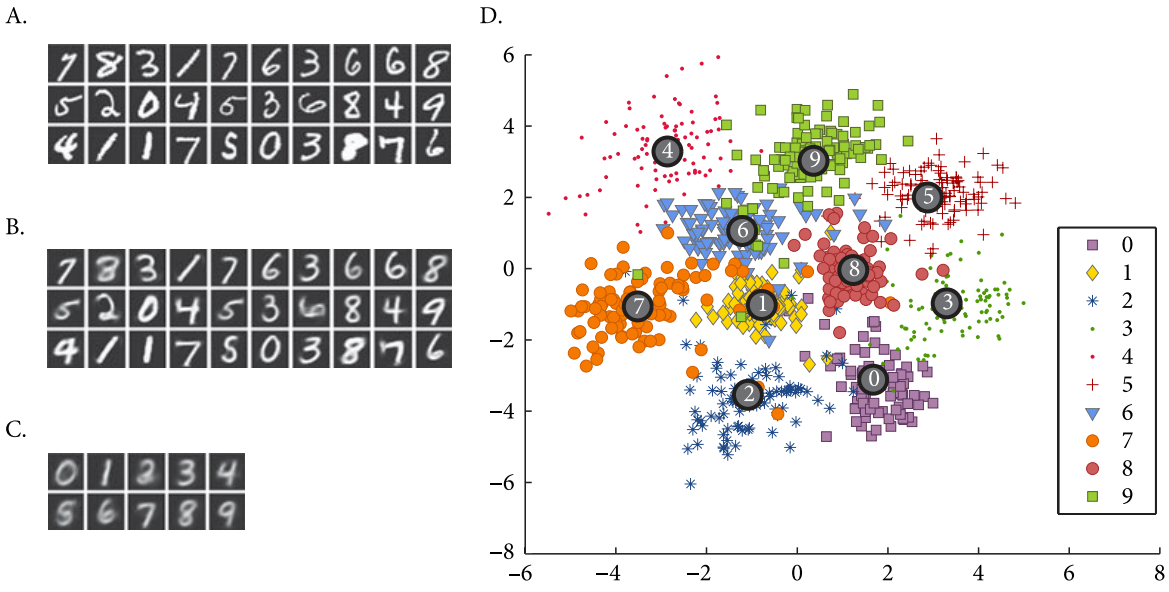

Figure 2: Dereferencing semantic pointers. **A.** Original input images. **B.** "Dereferenced" semantic pointers. **C.** Dereferenced average semantic pointer. **D.** Dimensionality reduction of the 50-dimensional semantic pointer space. Figure copied from Eliasmith, "How to build a brain", 2013, Figure 3.7.

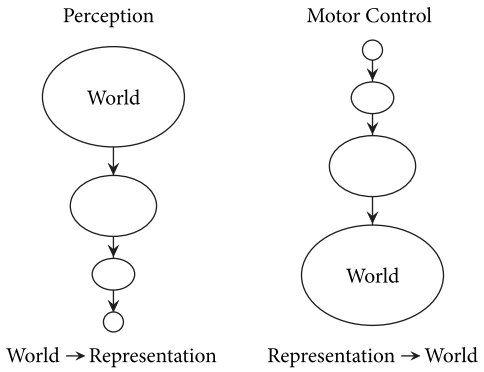

Figure 3: Perception and action can be described as similar, but reversed processes. Perception compresses a high-dimensional signal from the environment into a low-dimensional description, a semantic pointer. Motor-control decompresses a semantic pointer into a high-dimensional motor plan (i.e., muscle tensions over time). Figure copied from Eliasmith, "How to build a brain", 2013, Figure 3.9.

2.2 Deep Semantics for Perception

Now, how are semantic pointers supposed to capture parts of the "deep semantics" of objects? As mentioned above, the answer is that semantic pointers are thought to be generated by repeated compression of stimuli, i.e., the symbol is directly related to the stimulus it represents.

As an example, take vision: we know that visual stimuli are processed in multiple "layers" of visual cortex, where the complexity of the represented concepts increases with each layer (V1, V2, V4, IT). In other words, each layer is performing dimensionality reduction (or compression) of the information in the previous layer, trying to extract as much information as possible from the previous layer (fig. 1).

The representation in the "final" layer could be interpreted as a *semantic pointer*. Notice that this vectorial representation is directly influenced by the input to the system. Changes to the visual stimulus will result in slightly different representations. Crucially, we can "dereference" or "decompress" the visual representation. That is, we can "clamp" the final layer to a specific semantic pointer and compute in "backwards direction" which neural activity is most likely to evoke a certain semantic pointer (fig. 2). In the context of visual processing, this would correspond to "mental imagery".

2.3 Deep Semantics for Action

Similar processing hierarchies do not only exist for visual perception, or, perception in general. For example, there is evidence that high-level motor commands are successively being translated into more detailed control signals. In a sense, this is exactly the opposite problem of turning a high-dimensional perceptual signal into a low-dimensional semantic pointer.

We can treat these high-level control signals as "semantic pointers" that are being decompressed into a high-dimensional control signal (i.e., all possible muscle tensions in the body over time). Correspondingly, a semantic pointer for motor control might be something such

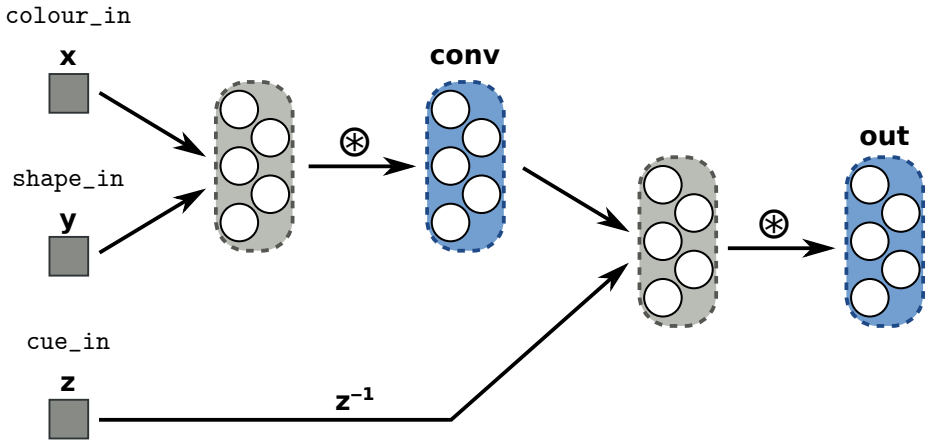

Figure 4: Network diagram of the SPA example network performing "question answering" described below. Grey boxes correspond to input nodes ("`spa.Transcode`" objects), large boxes to neuron populations. Grey neuron populations are implicitly generated by `nengo_spa` in order to compute the circular convolution—as we have seen before, when computing multiplication (which is part of the circular convolution operator), both operands need to be represented in a common pre-population.

as "reach the bottle" (represented in cortical areas such as the pre-motor area "PM" and the supplemental motor area "SMA"), which is then turned into a detailed motor plan (in cortical areas such as M1, in conjunction with non-cortical areas such as S1, S2 and the cerebellum).

Note: A model that implements optimal motor control on top of this framework is the "NOCH" model by Travis Dewolf [5]. This model is also described in Chapter 3 of "How to Build a Brain".

2.4 Using the Semantic Pointer Architecture in Nengo

In order to use semantic pointers in Nengo models, we can use the `nengo_spa` library. While this library does not implement the perception and action systems we discussed above (i.e., per default, semantic vectors are just random symbol vectors), it does provide an implementation of the action selection system we discuss below.

In a nutshell, the `nengo_spa` library provides `spa.Transcode` in lieu of `nengo.Node` objects whenever a symbolic stimulus is received from the environment. Furthermore, `nengo.Ensemble` instances can be replaced with `spa.State` objects to represent symbol vectors in a neural population.

Consider the "question asking" example we discussed in the last lecture, where we saw that the pseudo-inverse of a symbol vector $\mathbf{x}$ can be used to reconstruct which other concept $\mathbf{x}$ has been bound to. For example, $(\mathrm{RED} \circledast \mathrm{SQUARE}) \circledast \mathrm{SQUARE}^{-1} \approx \mathrm{RED}$, i.e., we are asking the question "what property does the square have"?

The following code example (⌨ *Code*) shows how to implement a network that queries different properties from a dynamically changing set of inputs. A diagram giving a better overview of what we are implementing here is given in fig. 4.

```python
import nengo
import nengo_spa as spa

# Alternate between RED and BLUE as a color
def colour_input(t):
    if (t // 0.5) % 2 == 0:
        return "RED"
    else:
        return "BLUE"

# Alternate between CIRCLE AND SQUARE
def shape_input(t):
    if (t // 0.5) % 2 == 0:
        return "CIRCLE"
    else:
        return "SQUARE"

# Alternate between any of the four properties as a question/cue
def cue_input(t):
    sequence = ["0", "CIRCLE", "RED", "0", "SQUARE", "BLUE"]
    idx = int((t // (1. / len(sequence))) % len(sequence))
    return sequence[idx]

with model:
    colour_in = spa.Transcode(colour_input, output_vocab=dimensions)
    shape_in = spa.Transcode(shape_input, output_vocab=dimensions)
    cue = spa.Transcode(cue_input, output_vocab=dimensions)

    conv = spa.State(dimensions)
    out = spa.State(dimensions)

    # Connect the buffers using overloaded Python operators
    colour_in * shape_in >> conv
    conv * ~cue >> out
```

Note that the strings returned in the callback functions passed to `spa.Transcode` are automatically translated into unique, randomly generated symbol vectors, the so called "vocabulary" of the model. Figure 5 shows the result of running the above code. Since it is hard to visualise higher-dimensional vectors, the plots show the similarity of any vector currently represented in a population to the vectors in the vocabulary.

Note: While the `nengo_spa` library offers useful abstractions for building SPA models, keep in mind that the resulting network is still a low-level NEF network; even though you use the `spa.Transcode` or `spa.State` objects, the network is consisting of `nengo.Node` objects for anything input/output related and `nengo.Ensemble` objects for everything else.

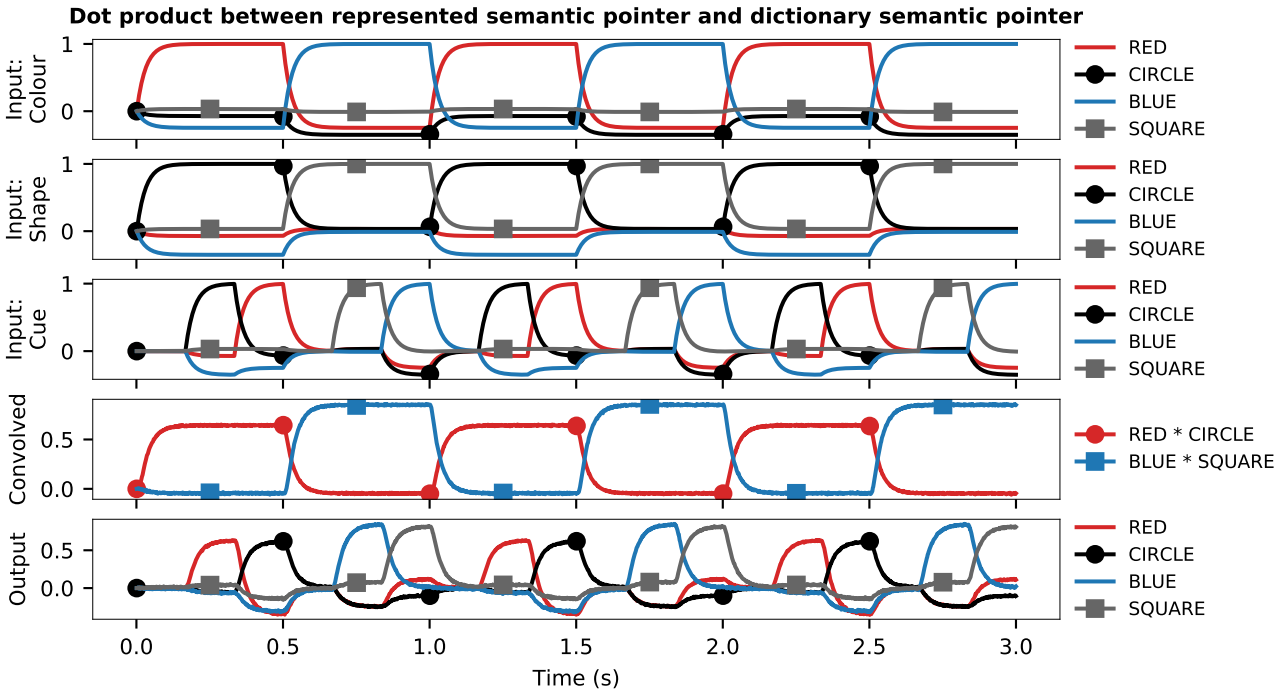

Figure 5: Output of the nengo_spa "question asking" example detailed above. As can be seen in the output plot, the network correctly identifies the concept that is bound to any given cue. ⌨ *Code*

3 SPAUN

As mentioned above, SPAUN is one of the largest functional brain models to date [1]. SPAUN implements the ideas discussed above. Visual input is directly translated into semantic pointers, and semantic pointers representing motor actions are used to drive a motor system (fig. 6).

Processing—apart from visual input and motor output—consists of a comparably small set of building blocks information is routed to according to an action selection system. The action selection system itself is driven by the state of a working memory and a reward evaluation model. All these building blocks are coarsely modelled after individual brain regions, as detailed in fig. 7. Although SPAUN consists of relatively few modules, it is able to flexibly use these components to solve a variety of tasks—without the experimenter having to intervene and reconfigure the model (see here for some videos showing SPAUN in action: 📺 digit recognition, 📺 copy drawing, 📺 addition by counting, 📺 pattern completion).

For a detailed description of the model, refer to "How to Build a Brain" [3]. In the following, we will focus on two import sub-components: the working memory model used in SPAUN, as well as the Basal Ganglia/Thalamus model that is used for action selection and that lies at the heart of the Semantic Pointer Architecture.

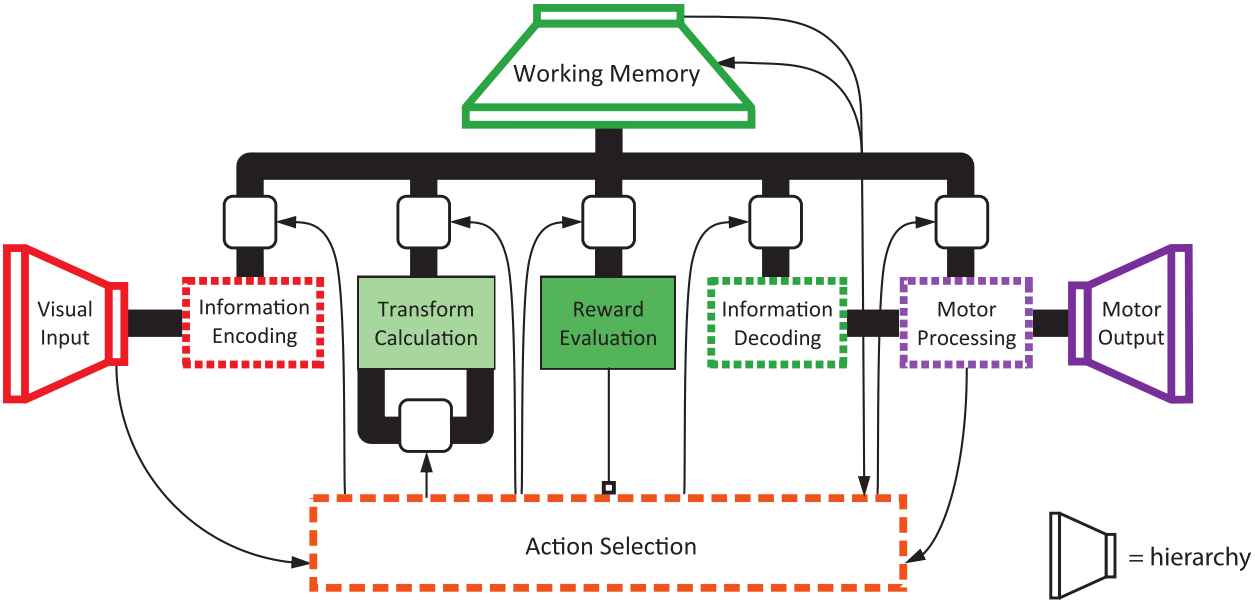

Figure 6: SPAUN's architecture. Visual input and motor output translate visual stimuli into semantic pointers and vice versa. Information is routed to different subsystems according to an action selection system, which, turn, is controlled by the content of the working memory and a reward system. Figure copied from [1].

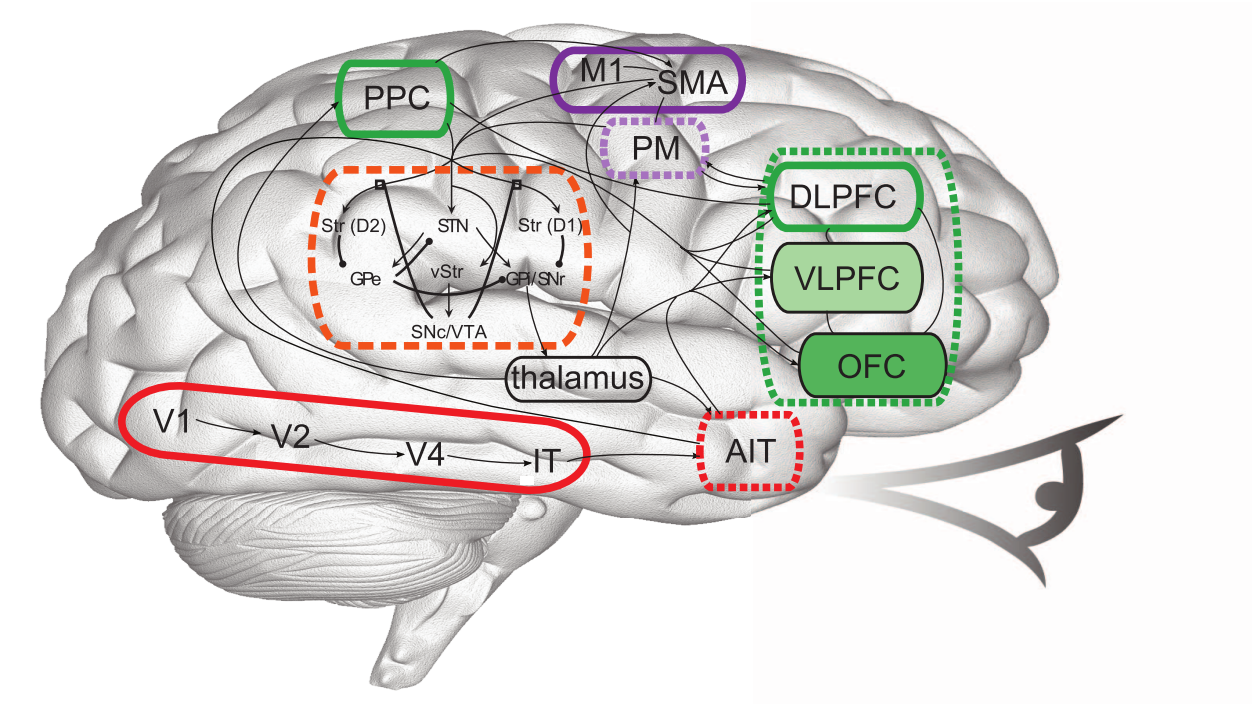

Figure 7: Brain areas modelled by SPAUN. Colours correspond to those found in fig. 6. The orange dashed line corresponds to the Basal Ganglia. Figure copied from [1].

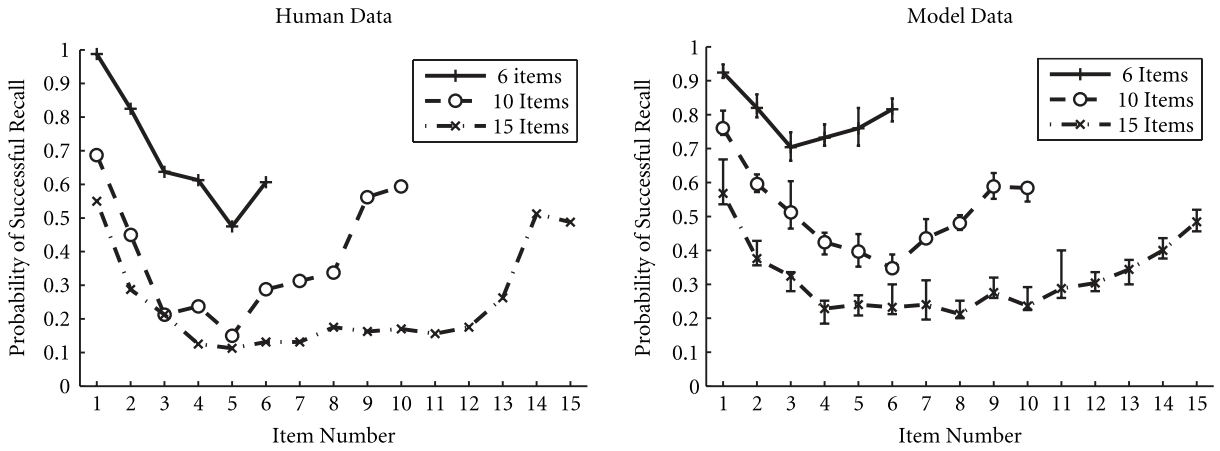

Figure 8: Human experimental data on serial recall, as well as the corresponding model-generated data. Figure copied from Eliasmith, "How to build a brain", 2013, Figure 6.2. Human data from Jahnke, *Delayed recall and the serial-position effect of short-term memory*, 1986 [6].

3.1 Serial Working Memory

Working memory describes the ability of animals to remember a limited amount of information for immediate processing (colloquially, "keeping something in mind"). Correspondingly, working memory is typically assumed to span time spans of a couple of seconds, and, is as you might know from personal experience, quite flawed.

Working memory (or, here, synonymously *short-term memory*), has been extensively studied by psychologists in various experiments. One such experiment is delayed serial recall. Participants are presented with a list of random words, one at a time, and are then asked to reproduce the list in order.

When repeating this experiment multiple times with lists of different lengths, one will find that participants are relatively good at remembering the first and last words in the list, but can often not remember the words in between. The ability to remember the initial words is called *primacy effect* (i.e., the first (primal) observations of a new sequence of events are remembered), the ability to remember the last words is called *recency effect* (i.e., the most recent events are still in memory) (fig. 8).

As we have discussed before, in the context of the NEF, we think about working memory content as being encoded in neural activity patterns and not in neural connection weights. Furthermore, we discussed neural integrators as a kind of working memory. Instead of just representing one dimension, we can also implement higher-dimensional neural integrators that "remember" entire semantic pointers $\mathbf{x}$ (this requires careful tuning of the neural encoders).

When presenting a sequence of symbol vectors $\texttt{ITEM}_A, \texttt{ITEM}_B, \ldots, \texttt{ITEM}_X$, a perfect integrator will automatically remember the sum of these symbol vectors, i.e., the state $\mathbf{x}$ after all items have been presented is given as a weighted sum of the individual symbol vectors:

$$\mathbf{x} = \texttt{ITEM}_A + \texttt{ITEM}_B + \ldots + \texttt{ITEM}_X\,.$$

Of course, when building a model of the above experiment, we also need to store the position

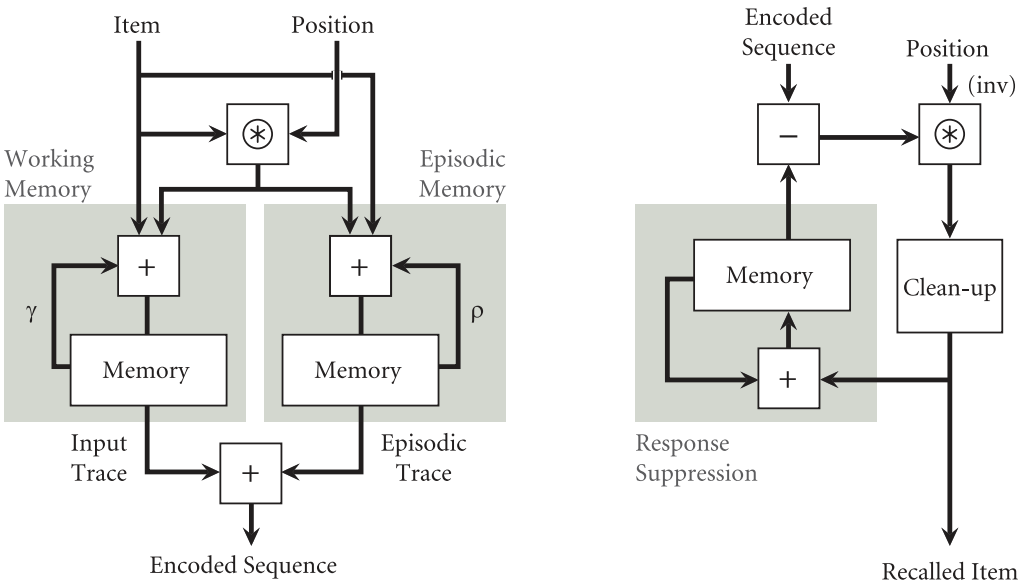

Figure 9: Ordinal serial encoding model. The left portion of the diagram shows the two sub-memories, responsible for the "recency" and "primacy" effects. Figure copied from Eliasmith, "How to build a brain", 2013, Figure 6.3.

of the items within the list. As we have discussed in the last lecture, we can use unitary symbol vectors to create an arbitrary long sequence of symbol vectors encoding integer positions

$$\mathrm{POS}_i = \underbrace{\mathrm{POS} \circledast \mathrm{POS} \circledast \ldots \circledast \mathrm{POS}}_{i \text{ times}}, \qquad \text{where } \mathrm{POS} \text{ is a unitary vector, i.e. } \|\mathbf{x} \circledast \mathrm{POS}\| = \|\mathbf{x}\|.$$

We can thus create these position vectors "on the fly" whenever a new item is presented, by just binding POS to the previous position vector.

Depending on how exactly the integrator is tuned (i.e., whether it is implemented as a leaky integrator or not, what the strength of the feedback connection is, radius of the population, etc.), a neural integrator implemented using NEF principle three can either show recency or primacy effects. Combining a primacy and a recency memory system thus allows us to implement a model of human serial working memory. Having two memory systems maps well onto the human brain, where we know that prefrontal cerebral cortex (PFC) is responsible for representing events in the immediate past, whereas Hippocampus is involved in remembering events in the more distant past (this type of memory is also referred to as "episodic memory"). See the left half of fig. 9 for an overview diagram of this memory model.

Of course, in order to recall items from this memory in series, we need a circuit capable of decoding the sequence. Naïvely, we can just apply the binding operator to the pseudo-inverse of the position we would like to extract. In order to improve the performance of the system, we can additionally remember which items were already recalled, and subtract those from the symbol vector encoding the sequence. This system is depicted in the right half of fig. 9.

An experiment comparing human performance to the performance of this memory system is depicted in the right half of fig. 8. As visible, the model data matches the human performance quite well, both qualitatively and quantitatively.

Note: *Repeating experiments.* When evaluating NEF models, one should always repeat the experiment several times with different random seeds. This will cause the x-intercepts, maximum firing rates and encoders to be different in each trial. Each of these experimental trials can be interpreted as a different "participant". One should then look at the distribution of the results, for example by visualising the mean, median, standard deviation, and the 10-/90-percentiles in a box plot (see Wikipedia and the Matplotlib documentation for more information).

3.2 Action Selection

- *Talk about SPAUN, need for action selection, need for routing of information between modules*

What is an action?

- Physical movements (⇒ control theory)
- Moving attention
- Changing the contents of working memory
- Recalling items from long-term memory

⇒ Routing information

A simple example

- A critter is trying to survive in a harsh environment
- Possible actions
  - **A** Go home
  - **B** Move randomly
  - **C** Go towards food
  - **D** Go away from predator
- Which one do we pick?

⇒ Reinforcement learning

Reinforcement Learning

- Learn function $Q(\mathbf{s}, \boldsymbol{\alpha})$ assigning a *utility* to the current state
- Policy: choose action $\boldsymbol{\alpha}$ with the highest utility $Q(\mathbf{s}, \boldsymbol{\alpha})$ in a given state $\mathbf{s}$

Action Selection Example

- Use lateral/mutual inhibition
- Hard to tune, especially in more complicated cases (higher dimensional $\mathbf{q}$, $\mathbf{s}$)
- How does the brain do it? ⇒ Basal Ganglia

Basal Ganglia Notes, do not write down!

- All of cortex connects to this
- Output goes to Thalamus, central routing system
- Clinical evidence
  - Parkinson's disease
    - Neurons in the substantia nigra die off
    - Extremely difficult to trigger actions to start
    - Usually physical actions; as disease progresses and more of the SNc is gone, can get cognitive effects too
  - Huntington's disease
    - Neurons in the striatum die off
    - Actions are triggered inappropriately (disinhibition)
    - Small uncontrollable movements
    - Trouble sequencing cognitive actions too
- Neurophysiological Evidence that BG are involved in RL, dopamine levels in BG correspond to reward prediction error
- Thalamus disinhibits other brain regions
- I.e., outputs $[1, 1, 0, 1]$ if action $\mathbf{C}$ is chosen
- Use inhibitory connections to suppress information from being transmitted

References

[1] Chris Eliasmith et al. "A Large-Scale Model of the Functioning Brain". In: *Science* 338 (2012), pp. 1202–1205. DOI: 10.1126/science.1225266. URL: http://www.sciencemag.org/cgi/content/full/338/6111/1202?ijkey=y5vph.jw5AgRQ&keytype=ref&siteid=sci.

[2] Xuan Choo. "Spaun 2.0: Extending the Worlds Largest Functional Brain Model". PhD thesis. University of Waterloo / UWSpace, 2018. URL: http://hdl.handle.net/10012/13308.

[3] Chris Eliasmith. *How to Build a Brain: A Neural Architecture for Biological Cognition*. Oxford Series on Cognitive Models and Architectures. New York, New York: Oxford University Press, 2013. 456 pp. ISBN: 978-0-19-026212-9.

[4] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. eprint: arXiv:1301.3781.

[5] Travis DeWolf. "NOCH: A Framework for Biologically Plausible Models of Neural Motor Control". Masters Thesis. Waterloo, ON: University of Waterloo, 2010. URL: http://hdl.handle.net/10012/4949.

[6] John C. Jahnke. "Delayed Recall and the Serial-Position Effect of Short-Term Memory." In: *Journal of Experimental Psychology* 76 (4, Pt.1 1968), pp. 618–622. ISSN: 0022-1015(Print). DOI: 10.1037/h0025692.