MPSiP

Nengo – Exercises 1

Iwo Błądek

Exercises and most of the text is based on or directly taken from the "How to Build a Brain" book by Chris Eliasmith.

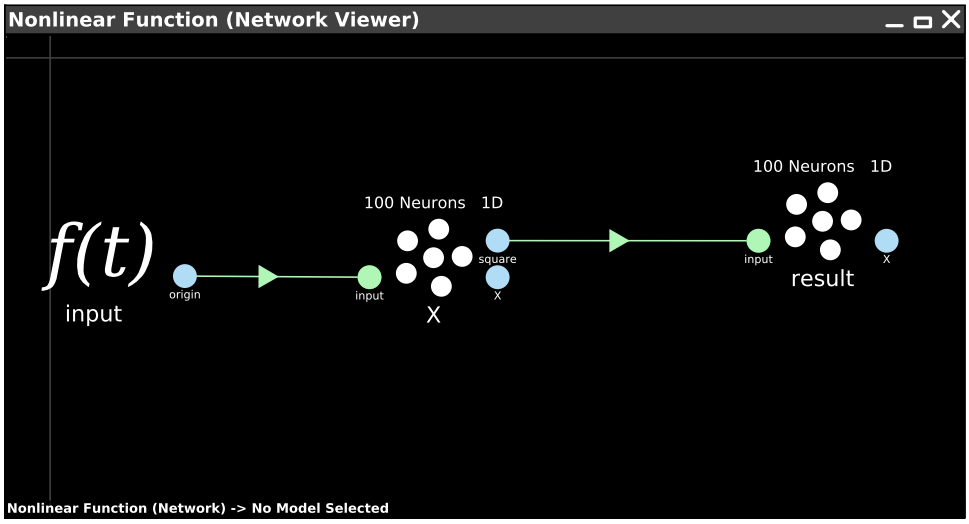
1 Nonlinear transformations

We already covered linear transformations in the first lesson about Nengo. They can be simply achieved by connecting populations of neurons and setting weights of those connections. In this exercise we will learn how to realize nonlinear transformations, for example square of the number (x^2).

1.1 Construction of the model

1. Create a new network called 'Nonlinear Function'.
2. Inside that network, create an ensemble and name it 'X' (it should have 100 neurons representing 1 dimension with a radius of 1.0).
3. Create a constant function input for this ensemble and connect it to the ensemble (remember to create a decoded termination with a connection weight of 1.0 on the 'X' ensemble and a PSTC of 0.02).
4. Create an ensemble named 'result'.
5. Create a termination on the 'result' ensemble (weight of 1.0, PSTC 0.02).
6. Drag an Origin from the template bar onto the 'X' ensemble.
7. In the dialog box that appears, name the origin 'square', set Output Dimensions to 1, and click Set Functions.
8. Select User-defined Function from the drop-down list of functions and click Set. The 'User-defined Function' dialog box that is now displayed allows you to enter arbitrary functions for the ensemble to approximate. The 'Expression' line can contain the variables represented by the ensemble, mathematical operators, and any functions specified by the 'Registered Functions' drop-down list.
9. In the Expression field, type 'x0*x0'. 'x0' refers to the scalar value represented by the 'X' ensemble. In an ensemble representing more than one dimension, the second variable is 'x1', the third is 'x2', and so forth. Returning to the preceding expression, it should be clear that it is multiplying the value in the 'X' ensemble by itself, thus computing the squared value.
10. Click OK to close dialogs until the decoded origin is created.
11. Drag from the 'square' origin to the 'input' termination to create a projection between the 'X' and 'result' ensembles.

    The projection that has just been made completes the set up for computing a nonlinear transformation. The 'X' and 'result' ensembles have been connected together and Nengo automatically sets connection weights appropriately to match the specified function.

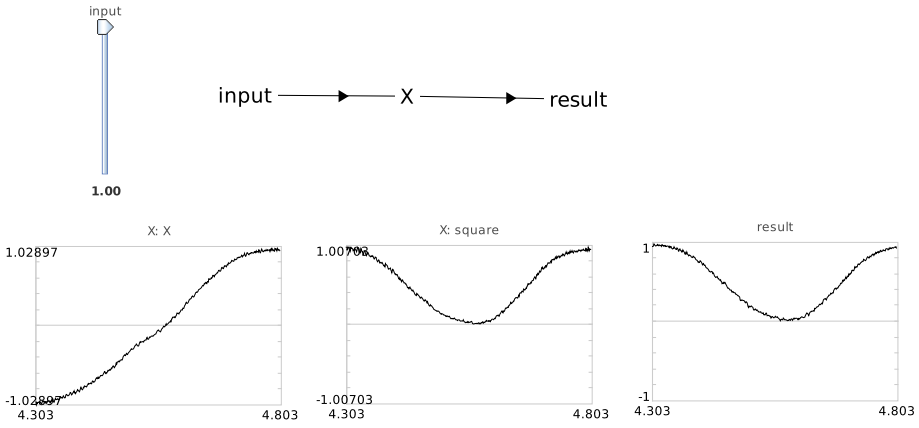


1.2 Testing the model

1. Open the Interactive Plots window.
2. Right-click 'result' and display its value.
3. Right-click your input and display its control.
4. Right-click 'X' and select X→value.
5. Right-click 'X' and select square→value.

   Note that, unlike the 'result' ensemble, the 'X' ensemble doesn't have just one value that can be displayed. Instead, it has the representational decoding of 'X', as well as any transformational decoding we have specified, in this case 'square'.

6. Run the simulation. To demonstrate it is computing the square, move the slider as smoothly as possible between -1 and 1. You should see a line in the 'X' value and a parabola in the 'square' and 'result' values.



Remember that these decoded values are not actually available to the neurons, but are ways for us to visualize their activities. However, if we only examine the relationship between the input and the spike responses in the output, we will see that the firing rates of the output neurons are in an approximate 'squaring' relation (or its inverse for 'off' neurons) to the input values. This can be seen by examining the spike raster from the 'result' population while moving the slider smoothly between -1 and 1.

7. Right-click ‘X’ and display its ‘spike raster’.
8. Right-click ‘result’ and display its ‘spike raster’.
9. Observe the behavior of spike activity of neurons while changing input.

2 Structured Representations

“Semantic pointers themselves are high-dimensional vector representations. When we think of semantic pointers as acting like symbols, we can name them to keep track of different semantic pointers in the same high-dimensional space. The collection of named vectors in a space forms a kind of ‘vocabulary’ of known semantic pointers in that space. Structured representations can then be built out of this vocabulary.”

In the example below:

$$\begin{pmatrix}1.69\\1.69\\1.73\\1.77\end{pmatrix} = \begin{pmatrix}0.2\\0.6\\0.6\\0.1\end{pmatrix} \circledast \begin{pmatrix}0.3\\0.9\\0.1\\0.5\end{pmatrix} + \cdots + \begin{pmatrix}0.4\\0.2\\0.3\\0.2\end{pmatrix} \circledast \begin{pmatrix}0.6\\1.0\\0.7\\0.4\end{pmatrix}$$

$$\mathbf{P} = \mathbf{agent} \circledast \mathbf{student} + \mathbf{verb} \circledast \mathbf{learn} + \mathbf{what} \circledast \mathbf{kogni}$$

semantic pointers are **P**, **agent**, **student**, etc. For simplicity of presentation they are in 4 dimensions, which by no account aspires to “high-dimensional vector representations”. ’⊛’ is a symbol of circular convolution (pol. *splot cykliczny*), and is used to bind together some vectors to create a single vector being somewhat similar to its parents. In the previous lesson it was described how exactly to compute it. We won’t be doing this manually (this is what computers are for), and it was introduced at this level of detail so that we have some idea what really lies behind this binding. ’+’ on the other hand is a standard vector addition operator.

2.1 Construction of the model

1. Open a blank Nengo workspace and create a network named ‘Structured Representation’.
2. Create an ensemble in the network, name it ‘A’, give it 300 neurons, and make it 20 dimensions.

   This ensemble is representing a vector space just as in the simpler low-dimensional case we considered earlier, only this time it is 20D. To use semantic pointers, we need to work in higher-dimensional spaces.
3. Open the network with Interactive Plots. Right-click the ‘A’ population and click ‘value’. Right-click the population again and click ‘semantic pointer’. You should now have two graphs.
4. Run the simulation for about 1 second.

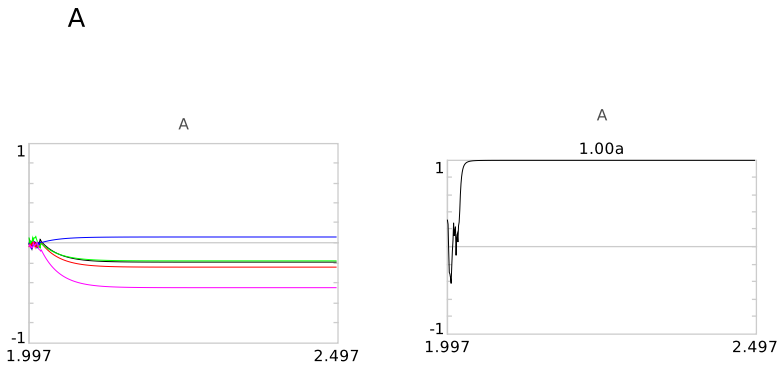
   Displaying the ‘value’ graph in Interactive Plots shows the value of individual components of this vector (20 components total).[1] The ‘semantic pointer’ graph compares the vector represented by the ensemble to all of the elements of the vocabulary, and displays their similarity. Initially, the vocabulary contains no vectors and thus nothing is plotted in the semantic pointer graph. As well, since there is no input to the population, the value graph shows noise centered around zero for all components.

[1] By default, the plot shows five components – to change this you can right-click the ‘value’ graph and choose ‘select all’ but be warned that plotting all twenty components may cause the simulation to be slow. Individual components can be toggled from the display by checking or unchecking the items labeled ‘v[0]’ to ‘v[19]’ in the right-click menu.

5. Right-click the semantic pointer graph and select ‘set value’. Enter ‘a’ into the dialog that appears and press OK.

6. Run the simulation again.

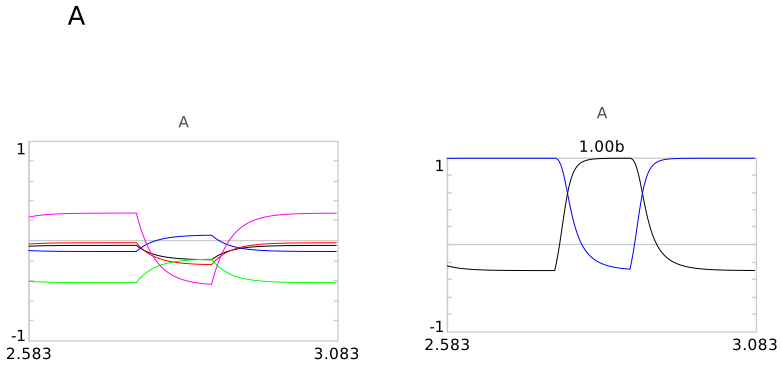
   Using the ‘set value’ option of the semantic pointer graph does two things: 1) if there is a named vector in the set value dialog that is not part of the vocabulary, it adds a randomly chosen vector to the vocabulary of the network and associates the given name (e.g., ‘a’) with it; and 2) it makes the represented value of the ensemble match the named vector. The result is that the ensemble essentially acts as a constant function that outputs the named vocabulary vector. As a result, when the simulation is run, the semantic pointer graph plots a 1, because the representation of the ensemble is exactly similar (i.e., equal) to the ‘a’ vector.



7. Right-click the ‘semantic pointer’ graph and select ‘set value’. Enter ‘b’ into the ‘Set semantic pointer’ dialog and press OK.

8. Run the simulation.

9. Switch between setting ‘a’ and setting ‘b’ on the semantic pointer graph while the simulation is running.



   Setting the value of the semantic pointer to ‘b’ adds a second, random 20-dimensional vector to the vocabulary. The ‘value’ plot reflects this by showing that the neural ensemble is driven to a new vector. Switching between the ‘a’ and ‘b’ vectors, it should be clear that although the vectors were randomly chosen initially, each vector is fixed once it enters the vocabulary. The semantic pointer graph changes to reflect which vocabulary item is most similar to the current representation in the ensemble.

   These are the vocabulary items that can be combined to form structured representations. To experiment with the binding (i.e., convolution) and conjoin (i.e., sum) operations, we require additional ensembles.

10. In the ‘Structured Representation’ network, add an ensemble named ‘B’ with 300 neurons and 20 dimensions.

11. Add a third ensemble named ‘Sum’ with the same parameters.

12. Add a decoded termination to the ‘Sum’ ensemble. Name the termination ‘A’ and set the number of input dimensions to 20, and PSTC to 0.02.

13. The coupling matrix of the termination should be set to an identity matrix by default (a grid of zeros except for a series of ones along the diagonal). Click Set Weights to make sure, and keep this default. Click OK twice.

14. Add a second decoded termination to the ‘Sum’ ensemble. Name the termination ‘B’ and set all other parameters as for the ‘A’ termination.

15. Add projections from ensembles ‘A’ and ‘B’ to the ‘Sum’ ensemble.

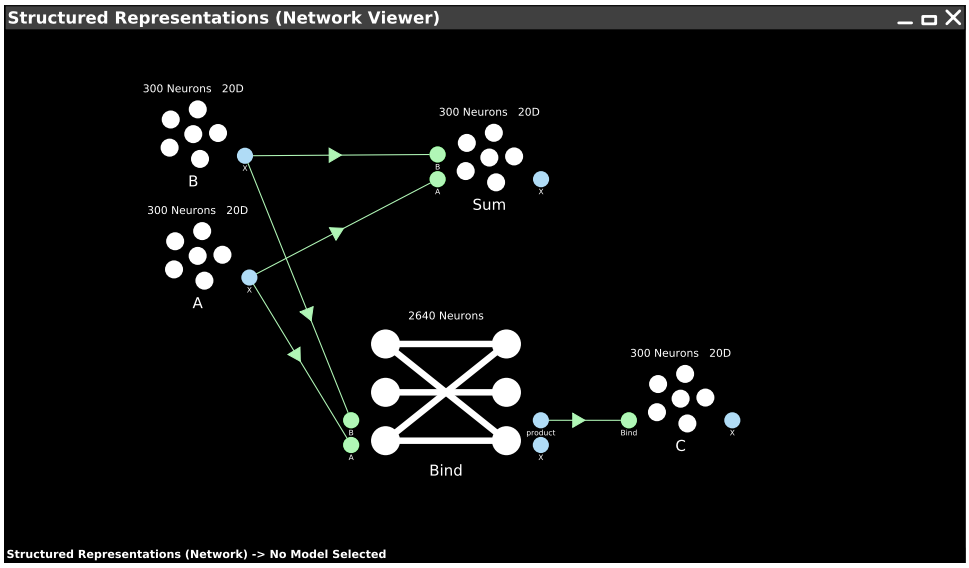
16. Add a fourth ensemble to the network named ‘C’ with the same parameters as all of the previous ensembles.

17. Drag the icon for the ‘Binding’ template from the bar on the left side of the screen into the network viewer for the ‘Structured Representation’ network.

18. In the dialog box that is opened, enter the name ‘Bind’, use ‘C’ as the name of the output ensemble, and set 60 neurons per dimension. The two inversion options should not be checked.

19. Project the ‘A’ and ‘B’ ensembles to the ‘Bind’ network that was created.

    The ‘Bind’ network is a component that computes the circular convolution of its two inputs using nonlinear transformations as discussed in the preceding tutorial. To look at the subnetwork elements you can double-click it. You cannot see the connections into and out of the subnetwork when you open it.

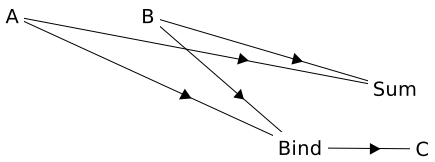


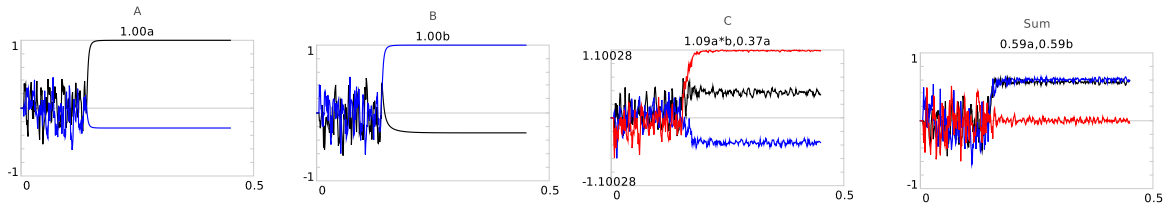
2.2 Testing the model

1. Open the Interactive Plots viewer. Right-click the background and select the ‘A’, ‘B’, ’C’, ‘Bind’ and ‘Sum’ nodes if they aren’t already displayed.

2. Show the semantic pointer graphs for the ‘A’, ‘B’, ‘C’, and ‘Sum’ nodes by right-clicking the nodes and selecting ‘semantic pointer’.

3. Set the value of the ‘A’ ensemble to ‘a’ and the value of the ‘B’ ensemble to ‘b’, by right-clicking the relevant semantic pointer graphs.
4. Right-click the semantic pointer graph of ensemble ‘C’ and select ‘show pairs’.
5. Right-click the semantic pointer graph of ensemble ‘C’ and select ‘a*b’.
6. Repeat the previous two steps for the ‘Sum’ ensemble, so that both ‘show pairs’ and ‘a*b’ are checked.
7. Run the simulation for about 1 second, and then pause it.





Scheme of colors assigned to subsequent vocabulary elements:

1. **Black**
2. **Blue**
3. **Red**
4. **Green**
5. **Magenta**

The ‘C’ population represents the output of a neurally-computed circular convolution (i.e., binding) of the ‘A’ and ‘B’ input vectors, while the ‘Sum’ population represents the sum of the same two input vectors. The label above each semantic pointer graph displays the name of the vocabulary vectors that are most similar to the vector represented by that neural ensemble. The number preceding the vector name is the value of the normalized dot product between the two vectors (i.e., the similarity of the vectors).

In this simulation, the ‘most similar’ vocabulary vector for the ‘C’ ensemble is ‘a*b’. The ‘a*b’ vector is the analytically-calculated circular convolution of the ‘a’ and ‘b’ vocabulary vectors. This result is expected, of course. Also of note is that the similarity of the ‘a’ and ‘b’ vectors alone is significantly lower. Both of the original input vectors should have a low degree of similarity to the result of the binding operation. Toggling the ‘show pairs’ option of the graph makes the difference even clearer. The ‘show pairs’ option controls whether bound pairs of vocabulary vectors are included in the graph.

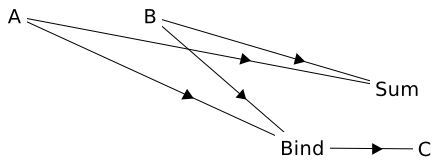
In contrast, the ‘a’ and ‘b’ vectors have a relatively high (and roughly equal) similarity to the vector stored in the ‘Sum’ ensemble. The sum operation preserves features of both original vectors, so the sum is similar to both. Clearly, the conjoin and bind operations have quite different properties.

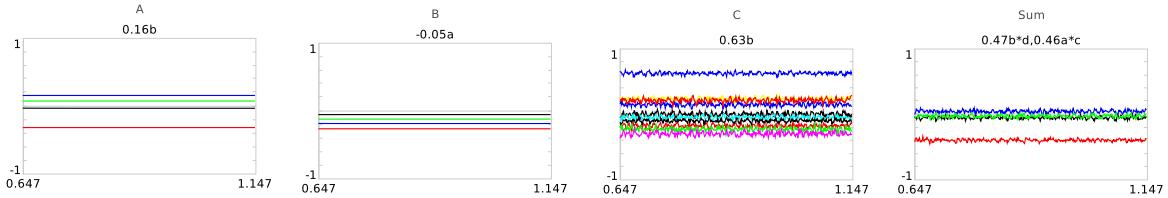
2.3 Extracting information from structured representations

We have now seen how we can implement the two functions needed to create structured representations in a spiking network. However, to process this information, we must also be able to extract the information stored in these conjoined and bound representations. This is possible through use of an inverse operation.

1. Set the value of the 'A' semantic pointer to 'a*c+b*d'.
2. Set the value of the 'B' semantic pointer to '~ d'.
3. Hide the 'Sum' and 'C' semantic pointer graphs.
4. Display the 'C' semantic pointer graph again by right-clicking 'C' and selecting 'semantic pointer' (hiding and displaying the graph allows it to adjust to the newly added 'c' and 'd' vocabulary items).
5. Right-click the graph of 'C' and select 'select all' (the letters 'a' through 'd' should be checked).
6. Run the simulation for about 1 second.

   In this simulation, we first set the input to a known, structured semantic pointer. The value 'a*c+b*d' is the semantic pointer that results from binding 'a' with 'c' and 'b' with 'd' then summing the resulting vectors. This vector is calculated analytically as an input, but it could be computed with neurons if need be. The second input, '~ d', represents the pseudo-inverse of 'd'. The pseudo-inverse vector is a shifted version of the original vector that approximately reverses the binding operation. Binding the pseudo-inverse of 'd' to 'a*c+b*d' yields a vector similar to 'b' because the 'b*d' operation is inverted while the 'a*c' component is bound with 'd' to form a new vector that is dissimilar to anything in the vocabulary and so is ignored as noise.
7. Experiment with different inverse values as inputs to the 'B' ensemble ('~ a', '~ b', or '~ c'). Note the results at the 'C' ensemble for different inputs.
8. You can also try changing the input 'statement' given to the 'A' ensemble by binding different pairs of vectors together and summing all the pairs.
9. Binding more pairs of vectors together will degrade the performance of the unbinding operation. If 'B' is set to '~ a', then setting 'A' to 'a*b+c*d+e*f+g*h' will not produce as clean an estimate of 'b' as when 'A' is set only to 'a*b'.





Scheme of colors assigned to subsequent vocabulary elements:

1. **Black**
2. **Blue**
3. **Red**
4. **Green**
5. **Magenta**

Instead of naming the vectors ‘a’, ‘b’ and so on, you can name them ‘dog’, ‘subject’, and so on. Doing so makes it clear how this processing can be mapped naturally to the various kinds of structure processing considered earlier in this chapter. Note that the name ‘I’ is reserved for the identity vector. The result of binding any vector with the identity vector is the vector itself. The lowercase ‘i’ does not refer to the identity vector.

There are two competing constraints that determine the accuracy of information storage and retrieval in this network. First, the number of dimensions. With a small number of dimensions, there is a danger that randomly chosen vectors will share some similarity due to the small size of the vector space. So it would be desirable to have very high dimensional spaces. However, the second constraint is the number of neurons. A larger number of dimensions requires a larger number of neurons to accurately represent and manipulate vectors. There is, of course, only a limited number of neurons in the brain. So, there is an unsurprising trade-off between structure processing power, and the number of neurons. The network presented in this tutorial is using a rather low number of dimensions and neurons for its semantic pointer representations for the sake of computational performance.