Team 77

- Tobias Pucher (S5751659)
- Matthias Heiden (S5751616)

Questions

Q1:Exploring weights

Question 1a: why do we get a 10x10 matrix? What do the numbers represent?

- The dimensions are due to the fact that the connection is a matrix of weights. Both the pre- and post-learning ensambles or "layers" have 10 neurons and each neuron is connected to each other. Each column of the matrix stores the weights from that pre-neuron to all possible post-learning neurons. We have 10 which gives 10 columns with 10 rows each because post has 10 neurons. In total 100 weights are present. For more dimensions the matrix dimensions would increase.
- The numbers represent the multiplicative "synaptic weight" factor of each individual input neuron to each individual output neuron. They are initialized randomly in a range around zero. Because of the mutliplication, each weight "scales" the signal (spike) and changes the "sign" as well.

Question 1b: open plots for the values of pre_learning and post_learning, and run this model.

Reset the model by clicking the reverse button. Run the model again. Can you see a relation between the values of both ensembles for different runs? Explain why there is a relation or not.

No relation can be spotted. This is due to the fact that the initialization of the connection weights happens in the range of [-1, 1] using random values ([0. . . 1] * 2 - 1 => [-1. . . 1]). Therefore the values are different for each run.

Question 1c: run the model. Explain why post_learning does not represent the same value as pre_learning, even though the neurons are directly connected.

This is because the connection (transformation) of the neurons has no effect on how the indiviudal neurons of the ensembles encode and decode the signal values. The LIF neurons used by these ensembles all have different response and tuning curves. These vary not only between ensambles but also between runs. This is particularly a problem because the nonlinearity of both reponse and tuning curves. Each neuron is tuned differently which adds to the problem.

Another reason is that on-/off-behaviour of linked neurons are not matched and therefore introduce further differences.

Question 1d: what would be the requirement for the above to work? Try out if this works by setting the seed argument of both ensembles (the seed argument sets the seed of the random number generator).

The requirement of the above to work would be that for each neuron in the ordered list of neurons that for each postition in the two lists it holds that: - same on/off behaviour - same response curve threshold - same response curve saturation point - same tuning curve threshold - same tuning curve saturation point

Q2: Learning weights

Question 2a: What do you think the connection weights will be in this case? You can check using the code above.

1x10 matrix of zeros. This is appearently the default behaviour of nengo which assumes that these 10 weights will be used for all 10 pre-neurons. According to the documentation the transform is a linear transform mapping the pre function output to the post input. And because the post input has a dimension of 10 and a scalar 0 is given it is used to form the 1x10 matrix of zeros.

Question 2b: I set the learning rate to a rather low value, so you can observe the learning (the default value is 1e-4). Try what happens if you set the learning rate to .01 or .1. Can you think of an explanation?

These learning rates are too high and the error and ouput start to oscillate. This is because the weights get adjusted too much and therefore the output overshoots and so does the error, therefore correcting in the opposite direction where it again overshoots creating a feedback loop that oscillates. All this happens with a faster frequency than the given sinus stimuli because of the noise in the stimuli that gets amplified by the weights and not the sinus itself. The sinus can be observed as the mean of the oscillating ouput.

Question 2c: We now developed a model that learns the connection weights between two ensembles. However, to do so, we had to specify the error: basically reintroducing pre-calculated connection weights. Think about how this would work when doing, for example, alphabet-arithmetic. Where does the error signal come from in that case? What are therefore the requirements for using supervised learning?

- Supervised learning requires a defined error metric with which the model can be evaluated. It should quantify the difference between the predicted result and the actual correct result from the labelled training data. This will then be used to calculate the error signal.

- In alphabet-arithmetic the error would come from the difference between the predicted result and the actual correct result. This requires the model to do backpropagation, updating the weights to minimize the error.
- Alternatively: Use symbol vectors: tuning curves of neurons are then dependant on vector representations instead of a single number.
- Use a Semantic Pointer Architecture: Semantic pointers can move around and can be combine with other semantic pointers.
  - Semantic pointers can also be manipulated (e.g. alphabet-arithmetic)