Contents

[What is Sparse Coding? 2](#_Toc185029714)

[Summary on Spiking Neural Networks (SNN) and Back-propagation-Free Algorithms (BFA) 2](#_Toc185029715)

[Relating NGC-Learn to Sparse Coding and SNN BFA 3](#_Toc185029716)

[Experimental Analysis and Documentation: 4](#_Toc185029717)

[Configuring Ngcsimlib in NGC-Learn 5](#_Toc185029718)

[Content 3 5](#_Toc185029719)

[Content 4 6](#_Toc185029720)

**Name: Melkamu Tesema**

**Date: 12/14/2024**

### ****What is Sparse Coding?****

* Sparse coding is a representation learning technique inspired by biological neural networks. It encodes input data into a sparse representation where only a small subset of neurons (features) is active for any given input.
* It is biologically motivated.
* It has application in different fields including image compression, machine learning.

### Summary on Spiking Neural Networks (SNN) and Back-propagation-Free Algorithms (BFA)

#### SNN

* information is transmitted through discrete spikes rather continues activation.
* Spikes are processed only when events occur
* Traditional back propagation does not work well for SNNs due to the discrete and temporal nature of spikes. Instead, training involves Spike-Timing Dependent Plasticity (STDP ) and Surrogate Gradients

#### BFA

* this is the method of training a model in which it doesn’t involve back propagation rather uses local learning with principle of **Hebbian Learning and use** Predictive Coding
* Compared with back propagation it reduces it reduces computational overhead
* Sparse coding can complement SNNs by providing efficient input representations.

### Relating NGC-Learn to Sparse Coding and SNN BFA

#### **NGC-Learn Overview**

* NGC-Learn is a library designed to implement and simulate biologically inspired models, particularly those aligned with the predictive coding framework.
* It supports constructing and experimenting with hierarchical networks that learn through local error correction instead of global back-propagation. This aligns closely with sparse coding and SNN BFA principles

#### **Sparse Coding in NGC-Learn:**

* NGC-Learn allow building predictive coding networks (PCNs), which use sparse representations to encode data efficiently.

#### **Local Learning Rules:**

* NGC-Learn replace global back-propagation with local error correction mechanisms, aligning with the BFA philosophy. This makes the training process biologically plausible and computationally efficient.
* Predictive coding, the basis of NGC-Learn, is a form of BFA where the network dynamically adjusts weights to minimize prediction errors without global gradients. This mirrors Hebbian learning and other BFA paradigms.

### Experimental Analysis and Documentation:

I used pc\_discrim for this

#### Initial Performance:

* + **Development Set Accuracy (Dev Acc):** 11.33%
  + **Training Set Accuracy (Tr Acc):** 11.18%
  + **Negative Log-Likelihood (NLL):** 6.978
  + **Energy-Free Error (EFE):** Not computed initially (--).

Interpretation: The initial performance is near-random, indicating that the model starts with untrained weights and no prior optimization.

#### Final Performance:

* + **Development Set Accuracy (Dev Acc):** 94.79%
  + **Training Set Accuracy (Tr Acc):** 94.59%
  + **Negative Log-Likelihood (NLL):** 0.282
  + **Energy-Free Error (EFE):** -25.02 (computed).

Interpretation: After training, the model achieves high accuracy and low NLL, demonstrating its ability to generalize effectively on unseen data. The negative EFE reflects reduced prediction errors, consistent with the principles of sparse coding and efficient representations.

#### Training Time:

* + Total simulation time: **0.081 hours (292.84 seconds).**

Interpretation: The training process was computationally efficient, leveraging sparse activations for faster convergence.

I had to decrease the iteration from 100 to 5 because it was taking longer to compute since I don’t have GPU.

### Configuring Ngcsimlib in NGC-Learn

Ngcsimlib uses a config.json file for global configurations like module paths and logging. Key configurations include:

1. **Modules:** Set the module\_path to point to your experiment-level modules.json.
2. **Logging:** Define logging\_level, logging\_file, and hide\_console to control verbosity and log output.
3. **Usage:** Access configurations via get\_config (as a dictionary) or provide\_namespace (as an object) for flexibility in experiments.

The modules.json file in **ngclearn** centralizes module imports and associated class definitions, enhancing reusability and simplifying model building. By default, it is located at json\_files/modules.json, but this path can be customized in the configuration.

The file mirrors Python import statements as JSON objects. For example:  
**Python:**

**from ngclearn.commands import AdvanceState as advance**

**JSON:**

{

"absolute\_path": "ngclearn.commands",

"attributes": [

{

"name": "AdvanceState",

"keywords": ["advance"]s

}

]

}

This schema allows multiple keywords for a single attribute and supports human-readable formats for class and module details. It is case-insensitive and supports importing multiple attributes from a single module.

### Content 3

Build simple model with 2 simple grade cells connected with synaptic cable

* We wire the 2 nodes together by synaptic cable wab
* A is stateless with a time constant tau\_m = 0, meaning it instantaneously outputs the clamped input value without retaining any history.
* B on the other hand, has a time constant tau\_m = 20, allowing it to accumulate inputs over time. This makes **b** a simple integrator of the weighted outputs from **a**.
* At each time b new = b old+aoutput/tm ( insthis case 20 )

### Content 4

* This one continues from content 3 and in this one b is not only dependent on a rather Hebbian learning is used b learns from previous interactions or inputs.
* If there is no signal the output remains the same for input 1 synaptic weight evolves proportionally.
* Hebbian learning allows the system to learn and adapt its synaptic weight dynamically, instead of being static.
* The output of b now reflects both the input from and the learned changes in the synapse (WAB).
* The system becomes plastic and capable of adapting to patterns in the input signal.