Acta Scientific COMPUTER SCIENCES

Volume 4 Issue 12 December 2022

Research Article

Brainopy: A Biologically Relevant SQLite-Based Artificial Neural Network Library

Jensen ZH Tan, Nicholas TF Tan and Maurice HT Ling*

School of Applied Sciences, Temasek Polytechnic, Singapore

***Corresponding Author:** Maurice HT Ling, School of Applied Sciences, Temasek Polytechnic, Singapore.

Received: October 27, 2022

Published: November 15, 2022

Abstract

Artificial neural network (ANN) is a computing system inspired by biological neural networks but recently, there is a move towards studying biological neural networks using neuronal simulations. Hence, ANN can be a tool to study biological neural networks. However, most ANN libraries only cater to one signal (equivalent to one neurotransmitter) and generally requires neurons to be organized into layers, which may not have direct biological equivalence. Here, we present Brainopy as a biologically relevant Python-based ANN library as it enables multiple neurotransmitters and allow each neuron to connect to any other neurons. The constructed neural network is persisted as an SQLite database file. Despite focusing on biological relevancy over computational efficiency, we built and simulated neural networks of up to 15000 neurons (within the neuronal complexity of *Caenorhabditis elegans*, a well-studied organism in neuroscience) using a retail laptop.

Keywords: Artificial Neural Network; Python Library; Biologically Relevant Representation; SQLite Persistence

Introduction

Artificial neural network (ANN) is a computing system inspired by information processing of biological neural networks that constitutes a biological brain [1], which had been used in a wide variety of applications [2-6]. On the other hand; systems neuroscience, which has its roots in systems theory such as cybernetics [7] and functional systems theory [8], is focused on how particular circuits and large brain networks modulate brain states and behaviour [9]. For example, how organismal behaviours can be explained by its corresponding neuronal connections [10]. This may have implications to psychiatry and addiction studies [11-15] as neuronal connectivity has been implicated.

This requires the ability to construct large simulations of neural networks with biological relevance as these simulations may be used as a model for the biological brain [16]. This suggests that ANNs can be a tool to study neurobiology [17]. Recent studies by Ito., *et al*. [18] and several other groups [19-22] support such a possibility. However, most current ANN libraries are focused on problem solving; such as, clustering and classification; and ease of use [23]. As a result, biological relevancy is deficient in three major ways. Firstly, most ANNs cater to only one signal type, which enables easier implementation and higher computational efficiency using matrix operations [24]. Single signal type corresponds to one neurotransmitter; yet, more than 100 neurotransmitters or neuroactive peptides with neurophysiological roles had been identified [25,26]. Secondly, most ANNs are based on algorithmic processing across layers of neurons [27,28], which may not have direct biological equivalence [29]. Finally, the concept of neuron is abstracted into a series of matrices and vectors for computational efficiency [24], which resulted in the loss of equivalent biological structure of a neuron.

In this study, we present Brainopy as a Python-based ANN library that addresses the deficiencies raised above. Brainopy enables multiple neurotransmitters and uses a biologically

relevant neuron representation. Each neuron is not placed in a layer but free to connect to other neurons using its dendrites and axon via synapses as the major components biological neuron; such as, dendrites, neuron body, axon, and synapses; are preserved. In addition, resulting neural network is persisted as an SQLite database file. Hence, the focus of Brainopy is on biological relevancy over computational efficiency. Despite so, our test suggests that Brainopy can be used to simulate a neural network of up to 15 thousand neurons using a retail laptop.

Architecture and implementation

Structure of brainopy

Brainopy is implemented as a single file, brainopy.py, to be imported as a module into another Python code file for use. Brainopy imports three modules from Python Standard Library (https://docs.python.org/3/library/); namely, random, sqlite3 and UUID. The random module is used for random selection of neurons, synapses, and axons. The sqlite3 module bridges SQLite database engine (www.sqlite.com) with Python. The UUID (Universal Unique Identifier) module generates unique identifiers (IDs) for unique identification of objects; such as, neurons, axons, and synapses; in Brainopy. All the functions / methods of Brainopy are encapsulated into brainopy class; hence, the usage file needs to import brainopy class from within brainopy.py module.

Biologically, a neuron usually consists of one or more dendrites and an axon attached a cell body [30]. Dendrite(s) and axon are responsible for signal input into the neuron and signal output from the neuron, respectively. In terms of implementation, each part of a neuron is represented by a set of neurotransmitter states. Hence, a neuron can be represented by one or more dendritic states (neurotransmitter states for a dendrite), one neuron state, and one axonal state (Figure 1A). Synapse is also represented as a set of neurotransmitter states, known as synaptic state, as synapse is the junction between two neurons; more specifically, the junction between the preceding axon and the proceeding dendrite (Figure 1B).

As neurons and synapses are represented by neurotransmitter states, data processing is the updating of proceeding neurotransmitter states from preceding neurotransmitter states by transfer functions. A total of five transfer functions were defined (Table 1: Transfer): (a) tfSynapseDendrite updates dendritic neurotransmitter state from the preceding synaptic state; (b) tfDendriteNeuron updates neuronal state from its preceding one or more dendritic state(s); (c) tfNeuronAxon updates axonal state from the preceding neuronal state; (d) tfAxonSynapse updates synaptic state from the preceding axonal, which mimics neurotransmitter release [31]; and (e) tfSynapseAxon updates the preceding axonal state from the proceeding synapstic state, which mimics synaptic re-uptake [32]. With these transfer functions, neurotransmitters can transfer from synapses to the neuron body through dendrite(s) before transferring to the next synapse through the axon (Figure 1B).

Besides transfer functions, two other classes of functions; namely, modulator functions (Table 1: Modulator), and maintenance functions (Table 1: Maintenance); were defined. The modulator functions are mfDendrite, mfNeuron, mfAxon, and mfSynapse; which modulates dendritic states, neuronal states, axonal states, and synaptic states, respectively. Current implementation of modulator functions varies each of the neurotransmitter values in their respective states within 0.5% of its original values. Five unimplemented placeholder maintenance functions; namely, mtNeuronGrowth for adding new neuron(s) during execution, mtSynapseGrowth for adding new synapse(s) during execution, mtNeuronPrune for removing neuron(s) during execution, mtSynapsePrune for removing synapse(s) during execution, and mtGlobal as a catchall for all other unspecified operations; were defined.

Figure 1: Schematics of modulating functions and transfer functions within and across neurons.
Panel A shows a single neuron with modulating functions.
Panel B shows the
connections between neurons, together with the transfer functions within and across neurons.

Class	Name	Description
Modulator	Dendrite Modulating Function (DMF)	Adjustments to dendritic states. Implemented as brainopy.mfDendrite() method. DM(DS) ➔ DS'
	Neuron Modulating Function (NMF)	Adjustments to neuronal states. Implemented as brainopy.mfNeuron() method. NM(NS) ➔ NS'
	Axon Modulating Function (AMF)	Adjustments to axonal states. Implemented as brainopy.mfAxon() method. AM(AS) ➔ AS'
	Synapse Modulating Function (SMF)	Adjustments to synaptic states. Implemented as brainopy.mfSynapse() method. SM(SS) ➔ SS'
Transfer	Synapse to Dendrite Transfer Function (SDTF)	Equivalent to synaptic uptake [32]. Moves synaptic states into dendritic states. Implemented as brainopy.tfSynapseDendrite() method. SDTF(SS, DS) ➔ (SS', DS')
	Dendrite to Neuron Transfer Function (DNTF)	Moves dendritic states into neuronal states. Implemented as brainopy.tfDendriteNeuron() method. DNTF(DS, NS) ➔ (DS', NS')
	Neuron to Axon Transfer Function (NATF)	Moves neuronal states into axonal states. Implemented as brainopy.tfNeuronAxon() method. NATF(NS, AS) ➔ (NS', AS')
	Axon to Synapse Transfer Function (ASTF)	Equivalent to neurotransmitter release [31]. Moves axonal states into synaptic states. Implemented as brainopy.tfAxonSynapse() method. ASTF(AS, SS) ➔ (AS', SS')
	Synapse to Axon Transfer Function (SATF)	Equivalent to synaptic re-uptake [32]. Moves synaptic states into axonal states. Placeholder without implementation as brainopy.tfSynapseAxon() method. SATF(SS, AS) ➔ (SS', AS')
Maintenance	Neuronal Growth Function (NGF)	Addition of neuron(s). Placeholder without implementation as brainopy.mtNeuronGrowth() method. NGF(N) ➔ N'
	Synaptic Growth Function (SGF)	Addition of synapse(s). Placeholder without implementation as brainopy.mtSynapseGrowth() method. SGF(S) ➔ S'
	Neuronal Prune Function (NPF)	Removal of neuron(s). Placeholder without implementation as brainopy.mtNeuronPrune() method. NPF(N, S) ➔ (N', S')
	Synaptic Prune Function (SPF)	Removal of synapse(s). Placeholder without implementation as brainopy.mtSynapsePrune() method. SPF(S) ➔ S'
	Global Maintenance Function (GMF)	A catch-all function to performs all other unlisted operations. Placeholder without implementation as brainopy.mtGlobal() method. GMF(N, S, DS, NS, AS, SS) ➔ (N', S', DS', NS', AS', SS')

Table 1: Functions governing the operations within and across neurons.

Operation of brainopy

The operations of Brainopy are carried out using the functions listed in Table 2. Using brainopy_example.py as an example, the first step is to initialize brainopy object after import requires a path to an SQLite database, referred to as Brainopy database. If the given path points to a non-existent Brainopy database, it will be created with the 12 required data tables (Figure 2). Database logging is default to false. Secondly, a set of neurotransmitters is defined using addNeurotransmitters function, followed by adding the neurons and synapses using addNeuron and addSynapse functions, respectively. Each added neuron will have only one dendrite, representing a bipolar neuron [33], but the number of dendrites can be increased

using addDendrite function to represent a multipolar neuron [34]. Lastly, links between known axons and synapses can be established using linkAxonSynapse function or links between random axons and synapses can be established using linkRandomAxonSynapse function. Similarly, links between known synapses and dendrites can be established using linkSynapseDendrite function or links between random synapses and dendrites can be established using linkRandomSynapseDendrite function.

Figure 2: Entity-relationships of the 12 tables in Brainopy database.

Figure 3: Order of function execution during a cycle of brain simulation.

Class	Function/Parameters/Description
Database connection	• connectBrain(Path to database file): Connects to the brain database specified by the brainDB, which is a SQLite database. If the brain database does not exist, the database will be created. • disconnectBrain(): Disconnects and closes the brain database file.
Labelling	• nameID(ID, label, description): Method to add a name label (with corresponding description) to an ID (which can be neuron ID, neuron state ID, dendrite state ID, axon state ID, and synapse state ID). • getStateIDFromNeuronID(neuron ID, type of ID needed): Get dendrite state ID, neuron state ID or axon state ID from neuron ID / neuron body ID. • getStateIDFromNeuronName(neuron name, type of ID needed): Get dendrite state ID, neuron state ID or axon state ID from neuron name label.
Entity addition	• addNeurotransmitters(neurotransmitters): Add / register neurotransmitters. Neurotransmitters is given as a dictionary of {<neurotransmitter>: <description>}; for example, {“Ach”: “acetylcholine”, “DA”: “dopamine”, “GLU”: “glutamate”, “NE”: “norepinephrine”, “5HT”: “serotonin”, “GABA”: “gamma-Aminobutyric acid”}. This method does not backpatch new neurotransmitters to existing neurons and synapses, which may cause errors in processing; hence, all neurotransmitters must be confirmed and registered before initializig neurons and synapses. • addNamedNeuron(name, description): Add a labelled neuron. The added neuron consists of one dendrite (which may be increased using addDendrite() method), one neuron body, and one axon. Hence, one neuron minimally consists of one dendrite state, one neuron state, and one axon state. One state is represented by the values of a set of registered neurotransmitters. • addNeuron(number of neurons): Add neuron(s). Each added neuron consists of one dendrite (which may be increased using addDendrite() method), one neuron body, and one axon. Hence, one neuron minimally consists of one dendrite state, one neuron state, and one axon state. One state is represented by the values of a set of registered neurotransmitters. • addNamedSynapse(name, description): Add a labelled synapse. The added synapse is represented by one synapse state, which is represented by the values of a set of registered neurotransmitters. • addSynapse(number of synapses): Add synapse(s). Each synapse is represented by one synapse state, which is represented by the values of a set of registered neurotransmitters. • addDendrite(neuron ID): Add a new dendrite to an existing neuron (represented by neuron_ID). The added dendrite is represented by one synapse state, which is represented by the values of a set of registered neurotransmitters.

Neuron linking	• stapleNeurons(from_neuron, to_neuron, identifier_type): Link up 2 neurons (staple 2 neurons together) using a pair of neuron body IDs or neuron names. There is an underlying assumption that each neuron has only one dendrite and one axon. However, multiple axons can link to one dendrite, and one axon can link to multiple dendrites. During linking process, a synapse will be created. This results in (from_neuron_ID)--axon_state_ID--> synapse_state_ID >--dendrite_state_ID--(to_neuron_ID). • linkAxonSynapse(axon_state_ID, synapse_state_ID): Register a connection between an existing axon (represented by axon_state_ID) and an existing synapse (represented by synapse_state_ID). • linkRandomAxonSynapse(number of connections): Register one or more random connection(s) between an existing axon (represented by axon_state_ID) and an existing synapse (represented by synapse_state_ID). • linkSynapseDendrite(synapse_state_ID, dendrite_state_ID): Register a connection between an existing dendrite (represented by dendrite_state_ID) and an existing synapse (represented by synapse_state_ID). • linkRandomSynapseDendrite(number of connections): Register one or more random connection(s) between an existing dendrite (represented by dendrite_state_ID) and an existing synapse (represented by synapse_state_ID).
Simulation	• inputSignal(synapse_state_ID, signal_state): Update a synapse state (represented by synapse_state_ID) from an exogenous state (represented by signal_state). This represents the input of signal into the brain. The signal_state is a dictionary of {<neurotransmitter>: <value>}; for example, {“Ach”: 0.11, “DA”: 0.15, “GLU”: 0.21, “NE”: 0.25, “5HT”: 0.31, “GABA”: 0.35}. • runBrain(neuronList, synapseList): Execute / run the entire brain or part of the brain. If a list of neuron_IDs (represented by neuronList) and list of synapse IDs (represented by synapseList) are not given, the entire brain will be executed / ran. To execute / run part of the brain, neurons (represented by neuron_IDs in neuronList) and/or synapses (represented by synapse_state_IDs in synpaseList) for the part of the brain must be given as neuronList. • readNeurotransmitters(identifier, type of identifier): Read neurotransmitter values using an identifier (ID or name label tagged by nameID method). This can be used to read dendrite state, neuron state, axon state, or synapse state. If the identifier is a neuron body ID or name label of a neuron body, the neuron state of the neuron body will be returned.
Others	• getIDs(database table): Get IDs registerd in a table, which is one of [“axon_state”, “dendrite_state”, “neuron_body”, “neuron_state”, “synapse_state”]. • getNeurotransmitters(): Get list of registered neurotransmitters.

Table 2: Simulation and Utility Functions.

To simulate neural network, a set of input neurotransmitter states must be inserted as input using inputSignal function. Actual simulation is performed using runBrain function, which can execute the entire or parts of the neural network for a single cycle. The runBrain function (Figure 3) will first trigger the neuron processes in each neuron, which is a sequence of tfSynapseDendrite (SDTF), mfDendrite (DMF), tfDendriteNeuron (DNTF), mfNeuron (DNTF), tfNeuronAxon (NATF), mfAxon (AMF), and tfAxonSynapse (ASTF) functions. Secondly, the mfSynapse (SMF) function will be executed for each synapse. Thirdly, the tfSynapseAxon (SATF) function will be executed for each neuron. Finally, the brain maintenance processes; consisting of the mtNeuronGrowth (NGF), mtNeuronPrune (NPF), mtSynapseGrowth (SGF), mtSynapsePrune (SPF), and mtGlobal (GMF) functions; will be executed. After one cycle of simulation, the updated neurotransmitter states can be read using readNeurotransmitters function.

A number of simplification steps has been implemented and illustrated in brainopy_example2.py. Rather than using IDs for identification, it may be more convenient to use descriptive labels. Hence, any ID can be labelled using nameID function. To further ease operations, new neurons and synapses can be labelled using addNamedNeuron and addNamedSynapse respectively at the time of addition. In addition, it is also possible to establish neurons without the required synapses. In this case, a synapse is automatically created when two neurons are linked using stapleNeurons function.

Testing brainopy

Two tests – small-scale and larger-scale – were conducted on Brainopy. The small-scale test was to ensure that the table connectivity was correct, and the defined neural network can perform computation. The larger-scale test was to evaluate the time taken to setup and execute a neural network of more than 1000 neurons, as well as the corresponding database file size.

Small-Scale: Testing with 10 neurons

A small-scale test was performed using 10 neurons and 10 synapses (file = brainopy_example.py), with one dendrite per neuron. Thirty random axon-synapse links and thirty random synapse-dendrite links were defined. Five neurotransmitters were defined. Ten cycles of the neural network were executed after inputting neurotransmitter values into each of the synapses (equivalent to Cycle 0 in Figure 4A). Our results show that there is a change in neurotransmitter levels at a specific synapse across the 10 simulation cycles (Figure 4A) and neurotransmitter levels vary in each synapse (Figure 4B). An examination of the database tables shows that the ID links within the four link tables (Figure 5); namely, (a) neuron dendrite table, (b) neuron body table, (c) synapse dendrite link table, and (d) axon synapse link table; from the four corresponding state ID tables (Dendrite State Table, Neuron State Table, Axon State Table, and Synapse State Table) were correct. This suggests that the simulation is working, which implies that Brainopy is likely to be functional.

Figure 4: Levels of 2 of the 5 defined neurotransmitters – serotonin and gamma-aminobutyric acid (GABA). Panel A shows the neurotransmitter levels from cycle 0 (initial inputted values) to cycle 10 of simulation. Panel B shows the neurotransmitter levels of 5 of the 10 neurons after cycle 10 of simulation.

Figure 5: Connections between various IDs in Brainopy database tables.

Larger-Scale: Testing with 1000-15000 neurons

A larger-scale test was performed (file = brainopy_stresstest.py) using 1000 to 15000 neurons where each neuron has three dendrites with five neurotransmitters were defined. Number of synapses as 200% the number of neurons. Number of random axon-to-synapse links as 50% the number of neurons. Number of random synapse-to-dendrite links as 300% the number of random axon-to-synapse links, which is 150% the number of neurons. The number of neurons tested was in the range of 1000 and 15000, which is within the neuronal complexity of *Caenorhabditis elegans*, a well-studied organism in neuroscience [35-37], consisting of 306 neurons and about 7500 synapses [38,39]. This test was performed on a retail available MacBook Pro running MacOS Big Sur version 11.6.4 with 1.4 GHz quad-core Intel Core i5 processor, 8 GB 2133 MHz DDR3 RAM, and 512 GB solid-state drive.

Our results (Table 3) show time required to setup Brainopy (Figure 6A) and executing one cycle of simulation (Figure 6B) with or without logging is directly proportional to the number of neurons, where logging takes substantially more time to setup and execute. In addition, the resulting database size is directly proportional to the number of neurons where logging results in substantially larger database (Table 3, Figure 7). This suggests that Brainopy is capable to simulate at least 15000 neurons with 30000 synapses on a retail laptop.

	Number				Time (Seconds)		Database Size (MB)
	Neuron	Synapse	Axon-Synapse	Synapse-Dendrite	Setup	Cycle	
With Logging	1000	2000	500	1500	27	54.4 (2.11)	119.8
	3000	6000	1500	4500	77	191.5 (11.21)	360.1
	5000	10000	2500	7500	130	343.6 (8.06)	602.5
	7000	14000	3500	10500	198	529.1 (19.25)	844.6
	9000	18000	4500	13500	246	773.8 (61.44)	1090.0
	11000	22000	5500	16500	381	1026.5 (14.03)	1330.0
	13000	26000	6500	19500	455	1299.7 (60.41)	1570.0
	15000	30000	7500	22500	421	1450.8 (46.59)	1810.0
Without Logging	1000	2000	500	1500	12	6.3 (0.46)	7.2
	3000	6000	1500	4500	36	33.8 (1.08)	21.4
	5000	10000	2500	7500	60	74.8 (1.72)	35.8
	7000	14000	3500	10500	85	143.8 (2.48)	50.0
	9000	18000	4500	13500	107	210.6 (16.2)	64.5
	11000	22000	5500	16500	134	289.5 (16.13)	78.6
	13000	26000	6500	19500	162	405.0 (9.18)	93.1
	15000	30000	7500	22500	187	507.7 (9.82)	107.6

Table 3: Time taken and database size up to 15000 neurons. Time taken for each cycle was calculated as average time using 10 replicates and standard deviation was given in brackets.

Figure 6: Time taken to setup and to execute one cycle of simulation. Panel A shows the setup time required. Panel B shows the average time needed to execute one cycle. Error bars denote standard deviation.

Figure 7: Database file size after 10 cycles.

Conclusion

Brainopy as a Python-based ANN library for building biologically relevant neural networks as it enables multiple neurotransmitters and each neuron to connect to any other neurons via synapses. The major components of a biological neuron; such as, dendrites, neuron body, and axon; are preserved. The constructed neural network is persisted as an SQLite database file.

Supplementary Materials

Brainopy has been deposited in Bactome repository (https://github.com/mauriceling/bactome) under GNU General Public License version 3 for academic or not-for-profit use only, and can be downloaded at https://bit.ly/brainopy_1.

Acknowledgement

This work is supported by Temasek Polytechnic School of Applied Science under the Student Project Fund (TP_PR1199).

Conflict of Interest

The authors declare no conflict of interest.

Bibliography

1. Basheer IA and Hajmeet M. "Artificial Neural Networks: Fundamentals, Computing, Design, and Application". *Journal of Microbiological Methods* 43 (2000): 3-31.

2. Bhagya Raj GVS and Dash KK. "Comprehensive Study on Applications of Artificial Neural Network in Food Process Modeling". *Critical Reviews in Food Science and Nutrition* 62.10 (2022): 2756-2783.

3. Feng F., *et al*. "Artificial Neural Networks for Microwave Computer-Aided Design: The State of the Art". *IEEE Transactions on Microwave Theory and Techniques* (2022): 1-23.

4. Nagy B., *et al*. "Application of Artificial Neural Networks in the Process Analytical Technology of Pharmaceutical Manufacturing - A Review". *The AAPS Journal* 24.4 (2022): 74.

5. Mumali F. "Artificial Neural Network-Based Decision Support Systems in Manufacturing Processes: A Systematic Literature Review". *Computers and Industrial Engineering* 165 (2022): 107964.

6. Padma KR and Don KR. "Artificial Neural Network Applications in Analysis of Forensic Science". *Cyber Security and Digital Forensics*, eds Ghonge MM, Pramanik S, Mangrulkar R, Le D (Wiley), 1st Ed (2022): 59-72.

7. Wiener N. Cybernetics: or, Control and Communication in the Animal and the Machine (MIT Press, Cambridge, Massachusetts), 2nd Ed (1965).

8. Anokhin PK. "Systems Analysis of the Integrative Activity of the Neuron (1974)". *The Pavlovian Journal of Biological Science* 19.2 (1974): 43-101.

9. Shen Y., *et al*. "The Emergence of Molecular Systems Neuroscience". *Molecular Brain* 15.1 (2022): 7.

10. Dennis EJ., *et al*. "Systems Neuroscience of Natural Behaviors in Rodents". *The Journal of Neuroscience* 41.5 (2021): 911-919.

11. Tolomeo S and Yu R. "Brain Network Dysfunctions in Addiction: A Meta-Analysis of Resting-State Functional Connectivity". *Translational Psychiatry* 12.1 (2022): 41.

12. Brennan AR and Arnsten AFT. "Neuronal Mechanisms Underlying Attention Deficit Hyperactivity Disorder: The Influence of Arousal on Prefrontal Cortical Function". *Annals of the New York Academy of Sciences* 1129 (2008): 236-245.

13. Bi B., *et al*. "Neural Network of Bipolar Disorder: Toward Integration of Neuroimaging and Neurocircuit-Based Treatment Strategies". *Translational Psychiatry* 12.1 (2022): 143.

14. Lee TY., *et al*. "Distinct Neural Networks Associated with Obsession and Delusion: A Connectome-Wide Association Study". *Psychological Medicine* 51.8 (2021): 1320-1328.

15. Yankouskaya A., *et al*. "Neural Connectivity Underlying Reward and Emotion-Related Processing: Evidence From a Large-Scale Network Analysis". *Frontiers in Systems Neuroscience* 16 (2022): 833625.

16. Glaser JI., *et al*. "The Roles of Supervised Machine Learning in Systems Neuroscience". *Progress in Neurobiology* 175 (2019): 126-137.

17. Savage N. "How AI and Neuroscience Drive Each Other Forwards". *Nature* 571.7766 (2019): S15-S17.

18. Ito T., *et al*. "Constructing Neural Network Models from Brain Data Reveals Representational Transformations Linked to Adaptive Behavior". *Nature Communications* 13.1 (2022): 673.

19. Güçlü U., *et al*. "Modeling the Dynamics of Human Brain Activity with Recurrent Neural Networks". *Frontiers in Computational Neuroscience* 11 (2017).

20. Anand A., *et al*. "Quantifying the Brain Predictivity of Artificial Neural Networks With Nonlinear Response Mapping". *Frontiers in Computational Neuroscience* 15 (2021): 609721.

21. Fakhar K and Hilgetag CC. "Systematic Perturbation of an Artificial Neural Network: A Step Towards Quantifying Causal Contributions in the Brain". *PLoS Computational Biology* 18.6 (2022): e1010250.

22. van Nifterick AM., *et al*. "A Multiscale Brain Network Model Links Alzheimer's Disease-Mediated Neuronal Hyperactivity to Large-Scale Oscillatory Slowing". *Alzheimer's Research and Therapy* 14.1 (2022): 101.

23. Gruzenkin DV., *et al*. "Neural Networks to Solve Modern Artificial Intelligence Tasks". *Journal of Physics: Conference Series* 1399.3 (2019): 033058.

24. Wiedemann S., *et al*. "Compact and Computationally Efficient Representation of Deep Neural Networks". *IEEE Transactions on Neural Networks and Learning Systems* 31.3 (2020): 772-785.

25. Cuevas J. "Neurotransmitters and Their Life Cycle". *Reference Module in Biomedical Sciences* (Elsevier) (2019): B9780128012383113000.

26. Snyder SH. "A Life of Neurotransmitters". *Annual Review of Pharmacology and Toxicology* 57.1 (2017): 1-11.

27. Syms S., *et al*. "Survey on Neural Network Architectures with Deep Learning". *Journal of Soft Computing Paradigm* 2.3 (2020): 186-194.

28. Elsken T., *et al*. "Neural Architecture Search: A Survey". *Journal of Machine Learning Research* 20.1 (2019): 1997-2017.

29. Suhaimi A., *et al*. "Representation Learning in the Artificial and Biological Neural Networks Underlying Sensorimotor Integration". *Science Advances* 8.22 (2022): eabn0984.

30. González C and Couve A. "The Axonal Endoplasmic Reticulum and Protein Trafficking: Cellular Bootlegging South of the Soma". *Seminars in Cell and Developmental Biology* 27 (2014): 23-31.

31. Gormal RS and Meunier FA. "Nanoscale Organization of the Pre-Synapse: Tracking the Neurotransmitter Release Machinery". *Current Opinion in Neurobiology* 75 (2022): 102576.

32. Bruinsma TJ., *et al*. "The Relationship Between Dopamine Neurotransmitter Dynamics and the Blood-Oxygen-Level-Dependent (BOLD) Signal: A Review of Pharmacological Functional Magnetic Resonance Imaging". *Frontiers in Neuroscience* 12 (2018): 238.

33. Gomis Perez C., *et al*. "Rapid Propagation of Membrane Tension at Retinal Bipolar Neuron Presynaptic Terminals". *Science Advances* 8.1 (2022): eabl4411.

34. Wang X., *et al*. "The Krüppel-Like Factor Dar1 Determines Multipolar Neuron Morphology". *The Journal of Neuroscience* 35.42 (2015): 14251-14259.

35. Santos J and Shlizerman E. "Closing the Loop: Optimal Stimulation of *C. elegans* Neuronal Network via Adaptive Control to Exhibit Full Body Movements". *BMC Neuroscience* 16.S1 (2015): O14.

36. Sengupta P and Samuel ADT. "Caenorhabditis elegans: A Model System for Systems Neuroscience". *Current Opinion in Neurobiology* 19.6 (2009): 637-643.

37. Rapti G. "A Perspective on *C. elegans* Neurodevelopment: From Early Visionaries to a Booming Neuroscience Research". *Journal of Neurogenetics* 34.3-4 (2020): 259-272.

38. Kaiser M and Hilgetag CC. "Nonoptimal Component Placement, But Short Processing Paths, Due to Long-Distance Projections in Neural Systems". *PLoS Computational Biology* 2.7 (2006): e95.

39. Cook SJ., *et al*. "Whole-Animal Connectomes of Both Caenorhabditis elegans Sexes". *Nature* 571.7763 (2019): 63-71.