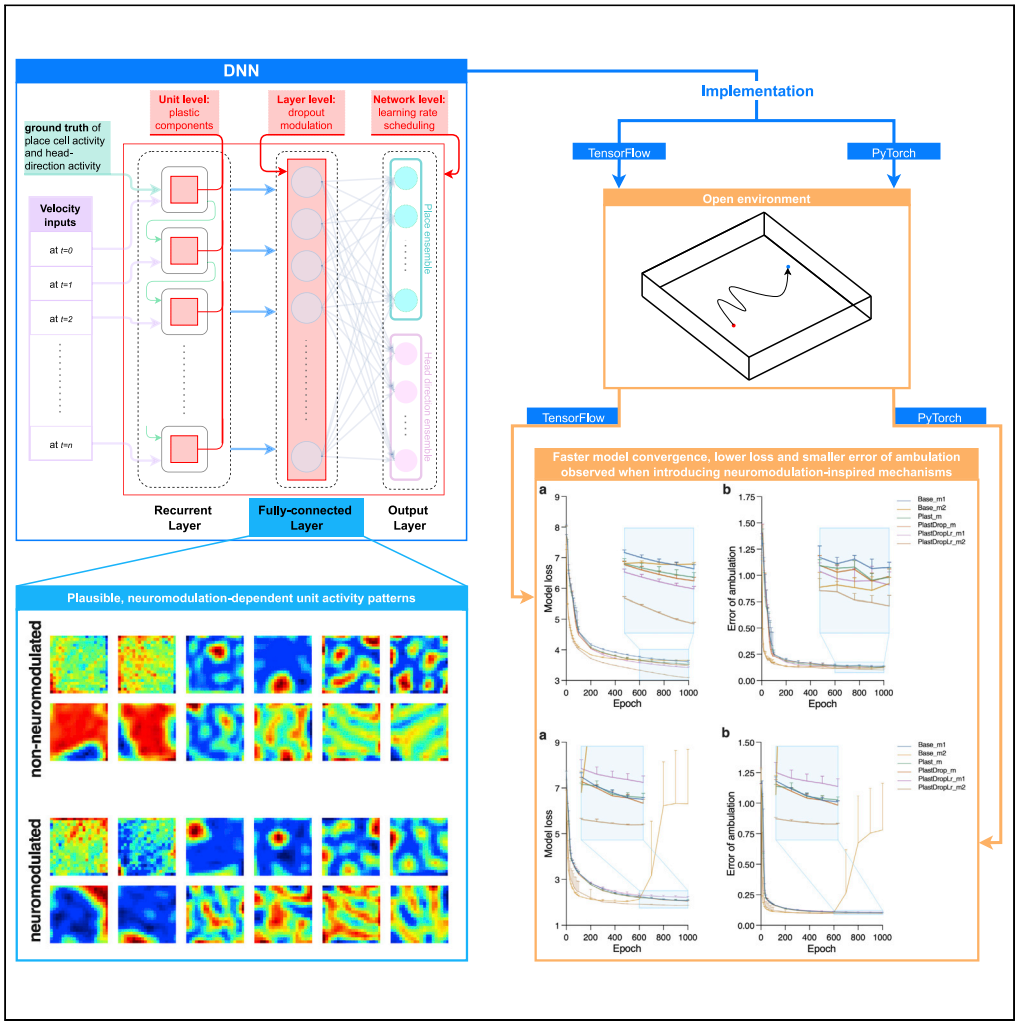
iScience

CellPress
OPEN ACCESS

Article

Effects of neuromodulation-inspired mechanisms on the performance of deep neural networks in a spatial learning task



Jie Mei, Rouzbeh Meshkinnejad, Yalda Mohsenzadeh

majorjiemei@gmail.com (J.M.)
ymohsenz@uwo.ca (Y.M.)

Highlights

Integration of multiscale neuromodulation-inspired mechanisms into a neural network

These mechanisms led to faster learning in a spatial learning task

Smaller errors of ambulation were observed when adding neuromodulatory components

Neuromodulatory components affect single unit activity in the fully connected layer

Mei et al., iScience 26, 106026
February 17, 2023
https://doi.org/10.1016/j.isci.2023.106026

iScience

CellPress
OPEN ACCESS

Article

Effects of neuromodulation-inspired mechanisms on the performance of deep neural networks in a spatial learning task

Jie Mei,[1,2,4,5,*] Rouzbeh Meshkinnejad,[2,3] and Yalda Mohsenzadeh[1,2,3,*]

SUMMARY

In recent years, the biological underpinnings of adaptive learning have been modeled, leading to faster model convergence and various behavioral benefits in tasks including spatial navigation and cue-reward association. Furthermore, studies have investigated how the neuromodulatory system, a major driver of synaptic plasticity and state-dependent changes in the brain neuronal activities, plays a role in training deep neural networks (DNNs). In this study, we extended previous studies on neuromodulation-inspired DNNs and explored the effects of neuromodulatory components on learning and single unit activities in a spatial learning task. Under the multiscale neuromodulatory framework, plastic components, dropout probability modulation, and learning rate decay were added to the single unit, layer, and whole network levels of DNN models, respectively. We observed behavioral benefits including faster learning and smaller error of ambulation. We then concluded that neuromodulatory components can affect learning trajectories, outcomes, and single unit activities, in a component- and hyperparameter-dependent manner.

INTRODUCTION

In the past decades, the field of artificial intelligence has significantly advanced, showing promising results in pattern recognition, image restoration and reconstruction, and medical imaging analysis thanks to the fast development of deep neural networks (DNNs), availability of large datasets, and increasing computing power. DNNs are loosely inspired by the biological nervous system, and are composed of multiple hidden layers, allowing for tuning of the weights to minimize the differences between the predicted and actual outcomes.

To further understand how biologically plausible mechanisms may shed light on DNNs and optimization methods, implementations at the dendritic, single-neuron, or microcircuitry levels have been increasingly realized.[1–5] Thus far, deep learning and neurorobotics studies[6–13] have examined whether biological neuromodulation may lead to behavioral benefits. In these studies, neuromodulation was commonly defined as a mechanism that self-reconfigures network hyperparameters and connectivity based on environmental or/and behavioral states of the neural network (Table 1).

More recently, neuromodulation-inspired learning rules have been employed in DNNs.[6–9] The neuromodulatory system consists of the serotonergic (5-HT), dopaminergic (DA), noradrenergic (NA), and cholinergic systems (ACh), which modulate a spectrum of physiological and cognitive processes through highly region- and target-specific projections originating from midbrain, hindbrain, or forebrain areas.[3,10] Through neuromodulation-inspired learning, DNNs that employ adaptive learning rules including synaptic plasticity and feedback-based hyperparameter tuning were validated in tasks including spatial learning, cue-reward association, and image classification and recognition (Table 1). As an example,[9] Vecoven et al. implemented and tested a DNN that allowed for the use of contextual information about the current task and environment by adjusting the slope and bias of the activation functions in the DNN (Table 1). Others have proposed to modulate synaptic plasticity to overcome catastrophic forgetting, or introduce neuromodulatory neurons and network-computed modulatory signals to achieve more efficient learning (Table 1).[7,8]

In contrast to the field of deep learning, application of neuromodulation-inspired learning in neurorobotics followed a more modular approach. That is, systems-level properties of neuromodulators are individually

[1]Western Institute for Neuroscience, University of Western Ontario, London, ON N6A 5B7, Canada

[2]Department of Computer Science, University of Western Ontario, London, ON N6A 5B7, Canada

[3]Vector Institute for Artificial Intelligence, Toronto, ON M5G 1M1, Canada

[4]Present address: International Research Center for Neurointelligence, The University of Tokyo, Bunkyo, Tokyo, 113–0033, Japan

[5]Lead contact

*Correspondence: majorjiemei@gmail.com (J.M.), ymohsenz@uwo.ca (Y.M.)

https://doi.org/10.1016/j.isci.2023.106026

Table 1. An overview of deep learning studies that applied neuromodulation-inspired mechanisms to DNNs

Hypothesis	Task	Neural network	Artificial neuromodulatory mechanism	Findings	Study
Modularity in neural networks alleviates catastrophic forgetting and improves skill learning.	In an environment, to eat all nutritious food while avoiding poisonous food to achieve maximum fitness.	A 5-layer, feedforward neural network with 10 input neurons. The three hidden layers have 10, 4, and 2 neurons, respectively.	Both non-modulatory and modulatory neurons were used. Inputs to each neuron consist of modulatory and non-modulatory connections. The sum of modulatory inputs to downstream neurons determines weight modifications of non-modulatory connections, and weight change is governed by a regular Hebbian learning term.	Neuromodulation leads to greater improvement in model performance in a neural network of higher modularity, and promotes evolution of the neural network against catastrophic forgetting.	Ellefsen et al.[7]
Deep neural network architectures inspired by cellular neuromodulation learn adaptive behaviors.	Navigating toward one (navigation problem 1) or two targets (navigation problem 2) in a 2D space with noisy movements.	A 6-layer feedforward RNN of 100, 75, 45, 30, 10, and 1 unit(s) in each layer, respectively.	Introduction of a neuromodulatory network that tunes the slope and bias of activation functions of the main network, then replacing activation functions in the main network with the neuromodulatory capable version.	The neuromodulated neural network learns faster (as quantified through the number of episodes), and obtains higher rewards. It also demonstrates adaptation abilities, and greater robustness against random seeds and network architectures.	Vecoven et al.[9]
Neuromodulated plasticity improves model performance in different tasks, e.g., reinforcement learning and supervised learning.	A cue-reward association task (task 1), a maze navigation task (task 2), and a language modeling task (task 3).	Task 1: An RNN with 200 neurons in the hidden layer. Task 2: An RNN with 100 neurons in the hidden layer. Task 3: Two LSTM models with 4.8 and 24.2 million parameters, respectively.	Use of (1) a Hebbian plastic component in addition to a fixed weight in each connection, (2) a network-computed, time-varying neuromodulatory signal, and (3) a dopamine activity-inspired eligibility trace.	Neuromodulated plasticity improves performance in three tasks of different nature, as indicated by higher rewards (tasks 1 and 2), or lower test perplexity (task 3). Lower test perplexity is observed in both LSTM models in task 3.	Miconi et al.[8]
Within local regions of a neural network, neuromodulators act as synaptic amplifiers or dampeners and support different, state-dependent behaviors.	Modified Go-NoGo tasks, where the agent is trained to give zero output ("NoGo") for the positive stimulus and negative output ("AntiGo") for the null stimulus, in addition to behavior sets in the classic Go-NoGo task.	An RNN with 200 neurons, 80% excitatory and 20% inhibitory. Each neuron can be connected to any other neuron with a certain probability (initialized at 0.8).	Scaling of weights of target neurons through a neuromodulatory factor for the whole network (whole network neuromodulation, i.e., applying to all neurons) and subpopulation neuromodulation (i.e., applying to randomly chosen, selected, overlapping or non-overlapping neuronal subpopulations).	Through modulation of synaptic weights, neuromodulators can enable distinct synaptic memory regions within a single neural network, and effects could be observed at single neuron, cluster, and global network activity levels.	Tsuda et al.[14]

Table 2. Loss before and after training, lowest loss during training, and change in loss over 1,000 epochs in TensorFlow models

	Training loss, epoch 0	Test loss, epoch 0	Training loss, epoch 1,000	Test loss, epoch 1,000	Training loss, lowest	Test loss, lowest	$Loss_{t=1000} - Loss_{t=0}$, training	$Loss_{l=1000} - Loss_{l=0}$, test
Base_m1	8.052 (0.015)	8.053 (0.016)	3.606 (0.028)	3.608 (0.039)	3.606	3.608	4.446	4.445
Base_m2	8.034 (0.006)	8.036 (0.005)	3.631 (0.023)	3.655 (0.014)	3.628	3.646	4.403	4.381
Plast_m	8.038 (0.010)	8.025 (0.006)	3.512 (0.043)	3.527 (0.046)	3.512	3.527	4.526	4.498
Plast Drop_m	8.029 (0.003)	8.034 (0.005)	3.416 (0.036)	3.494 (0.033)	3.416	3.494	4.613	4.540
Plast DropLr_m1	8.029 (0.005)	8.029 (0.009)	3.355 (0.021)	3.417 (0.023)	3.355	3.417	4.674	4.612
Plast DropLr_m2	8.037 (0.010)	8.032 (0.009)	2.911 (0.010)	3.085 (0.016)	2.911	3.085	5.126	4.947

Data are presented as mean (SD).

modeled based on experimental findings to enable high-level behavioral and cognitive effects, such as attention to salient objects, harm aversion, and risk taking.[11–13,15,16] For example, detection of objects triggers DA or 5-HT neurons, depending on the level of novelty or danger embodied.[11,17] In other studies, ACh and NA were modeled as filters that gate attention and distills events by adjusting weights from neurons that correspond to sensory events from ACh or NA neurons.[11,12]

In studies using neuromodulation-inspired DNNs, behavioral benefits were observed at multiple spatio-temporal levels, which could be attributed to the emergence of intrinsic neuronal properties under varying intensity of neuromodulation.[18] Despite the recent effort to introduce adaptive learning inspired by neuromodulators, the effects of modulatory processes at multiple spatial scales on model learning, as well as the interactions between neuromodulatory components and model hyperparameters, remain largely unexplored. To further previous lines of research, instead of one neuromodulatory component that governs the properties of DNN by adaptively adjusting connectivity or hyperparameters, we propose a multiscale framework, where neuromodulatory processes are realized at various spatial scales, including the whole network, single layer, and single units (Figure 12). This way, we individually add neuromodulatory components and investigate how each alters the dynamics, learning trajectory, and single unit activities of the DNN. In addition, we may study interactions across these components, for example, how effects induced by neuromodulatory components may only be observed when certain hyperparameter values are used.

To sum up, in the present study, we investigate and report how multiscale neuromodulation of (1) connectivity (e.g., plastic components for weight updates), (2) layer-specific properties (e.g., modulation of dropout probability), and (3) neural network hyperparameters (e.g., learning rate scheduling) may affect the learning trajectories, learning outcomes, and unit activity patterns of a DNN in an open field spatial learning task. Moreover, we highlight improvements in model convergence and learning efficiency, as well as variant-specific single unit activity patterns, when neuromodulatory components were added to the DNN. Finally, we summarize findings and limitations, discuss future research directions, and highlight how the neuromodulation-inspired framework could be further validated in DNNs to improve adaptive behavior and learning.

RESULTS

TensorFlow implementation

Spatial learning in non-neuromodulated and neuromodulated DNNs

Custom model loss. Prior to model training, both training and test loss were in the range of 8.025–8.053 and were comparable (Table 2). After 1,000 epochs, the training and test loss of **Base_m1** was 3.606 and 3.608, respectively (Table 2). Furthermore, adding plastic components to **Base_m1** marginally improved model convergence as indicated by a greater decrease in custom loss during training within the same number of epochs (Figure 1A), leading to a final training loss of 3.512 and test loss of 3.527 (**Plast_m**; Table 2).

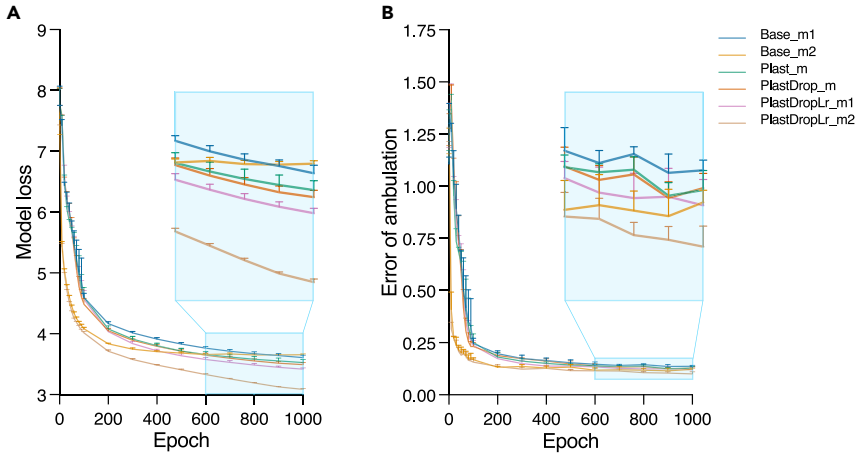


Figure 1. Performance of TensorFlow models

(A) custom model loss, (B) error of ambulation. Results shown were obtained from the test set and were the average of 5 trials. Data are presented as mean (SD).

Instead of a static dropout probability of 0.5 (as in **Base_m1** and **Plast_m**), introducing a modulated dropout probability to the model speeded up model convergence, resulting in a training custom loss of 3.416 and a test loss of 3.494 after 1,000 epochs (**PlastDrop_m**; Table 2). Finally, compared to a static learning rate, a lower test loss was observed when learning rate decay was applied (**PlastDrop_m** vs **PlastDropLr_m1**: 3.494 vs 3.417).

For a base model with a learning rate of 1e-4, neuromodulation-inspired mechanisms delayed the early saturation in custom model loss and decreased test loss from 3.655 to 3.085 after 1,000 epochs (**Base_m2** vs **PlastDropLr_m2**).

Error of ambulation. Error of ambulation of both training and test sets before and after training was summarized in Table 3. In the base model **Base_m1** and all its variants, errors of ambulation obtained from training sets were comparable to those obtained from test sets (Table 3). In **Base_m1**, before training, error of ambulation of the test set was 1.108m, which decreased to 0.137m after 1,000 epochs (Table 3; Figure 1B). When a plastic component was added to **Base_m1**, error of ambulation of the test set was 1.119 and 0.128m before and after training, respectively (**Plast_m**). Despite the lack of behavioral improvement when only dropout probability modulation was added to **Plast_m** (pre- and post-training error of ambulation of **PlastDrop_m**: 1.150 and 0.129m), enabling learning rate decay during model training, as in **PlastDropLr_m1**, decreased the error of ambulation to 0.120m after 1,000 epochs.

Table 3. Error of ambulation before and after training, lowest error of ambulation, and change in error of ambulation over 1,000 epochs in TensorFlow models

	Training error of ambulation (m), epoch 0	Test error of ambulation (m), epoch 0	Training error of ambulation (m), epoch 1,000	Test error of ambulation (m), epoch 1,000	Training error of ambulation (m), lowest	Test error of ambulation (m), lowest	$\varepsilon_{l=1000} - \varepsilon_{l=0}$, training	$\varepsilon_{l=1000} - \varepsilon_{l=0}$, test
Base_m1	1.135 (0.027)	1.108 (0.033)	0.135 (0.005)	0.137 (0.005)	0.133	0.133	1	0.971
Base_m2	1.128 (0.030)	1.120 (0.036)	0.126 (0.007)	0.122 (0.006)	0.120	0.109	1.002	0.998
Plast_m	1.143 (0.050)	1.119 (0.052)	0.131 (0.008)	0.128 (0.009)	0.126	0.124	1.012	0.991
Plast Drop_m	1.130 (0.045)	1.150 (0.044)	0.126 (0.010)	0.129 (0.007)	0.125	0.121	1.004	1.021
Plast DropLr_m1	1.160 (0.048)	1.153 (0.033)	0.127 (0.009)	0.120 (0.012)	0.124	0.119	1.033	1.033
Plast DropLr_m2	1.152 (0.078)	1.171 (0.044)	0.098 (0.005)	0.101 (0.010)	0.098	0.099	1.054	1.07

Data are presented as mean (SD).

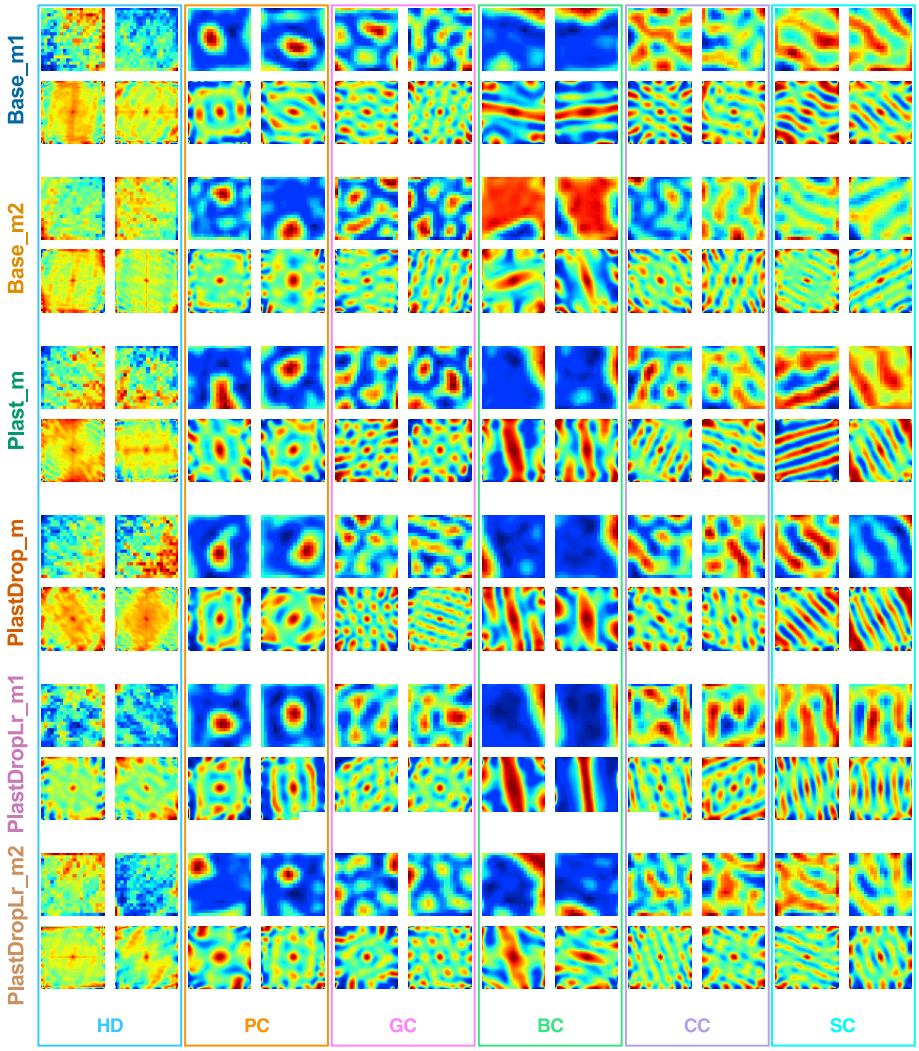


Figure 2. Unit activity patterns and autocorrelograms of fully connected layer units in DNNs implemented using TensorFlow, at 1,000 epochs

HD, head direction cell-like units; PC, place cell-like units; GC, grid cells-like units; BC, border cell-like units; CC, conjunctive cell-like units; SC, stripe cell-like units. Upper rows represent unit activity maps and lower rows represent corresponding autocorrelograms.

Moreover, adding neuromodulation-inspired components to a model with a fixed learning rate of 1e-4 (**Base_m2**) led to a reduced error of ambulation at 1,000 epochs (**Base_m2** vs **PlastDropLr_m2**: 0.122 and 0.101m).

Effects of neuromodulatory components on grid cell-like cell activity patterns in the fully connected layer. Biologically plausible activity patterns were observed in the fully connected layer in all model variants (Figure 2). In accordance with the drastic decrease in model loss shortly after the start of model training, biologically plausible unit activity patterns developed significantly during the first 200 epochs (Figure S1). Specifically, in all models, we were able to identify place cell-like, grid cell-like, head direction cell-like, conjunctive cell-like, stripe cell-like, and border cell-like units (Figure 2). However, in **Base_m2**, only the inverse of border cell-like activity patterns present, where increased activity levels were seen across the whole open arena but the border (Figure 2).

Base model learning rate (Base_m1 vs Base_m2). Increasing the learning rate of the base model from 1e-5 (**Base_m1**) to 1e-4 (**Base_m2**) led to an increase in the number of grid cell-like units (28.2 (4.6) vs 37.6

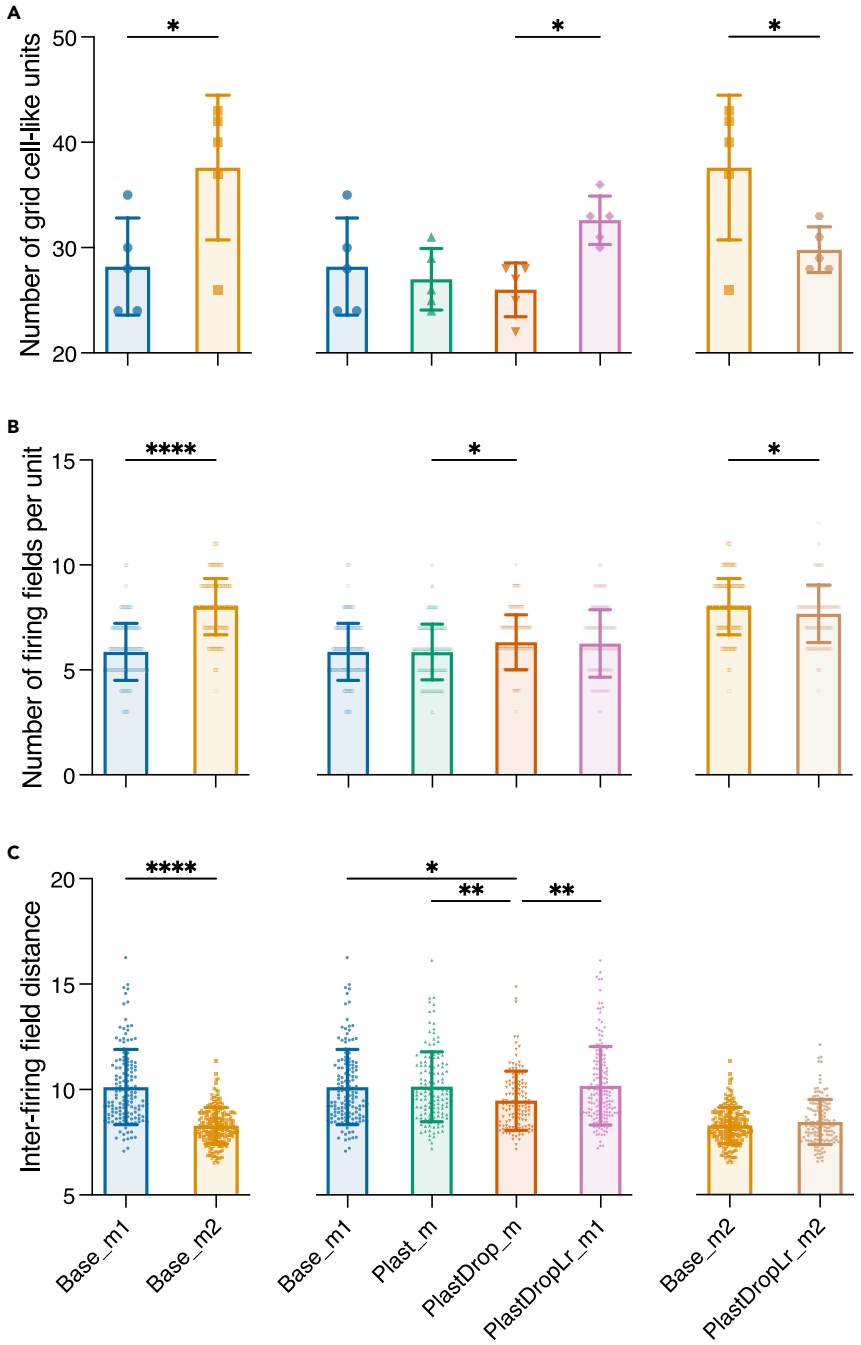


Figure 3. Effects of learning rate and neuromodulatory components on grid cell-like units and their general properties in TensorFlow models

(A) number of grid cell-like units, (B) number of firing fields in each unit, and (C) inter-field distance. *: $p < 0.05$, **: $p < 0.01$, ****: $p < 0.0001$. Data are presented as mean (SD).

(6.9); $p = 0.0109$; Figure 3A) and the number of firing fields per unit (5.9 (1.4) vs 8.0 (1.3); $p < 0.0001$; Figure 3B), as well as a decrease in the inter-field distance (10.1 (1.8) vs 8.3 (0.9); $p < 0.0001$; Figure 3C). Increased base model learning rate is also associated with a decrease in the average size, width, and height of firing fields (13.8 (4.7) vs 9.5 (2.2), 6.7 (1.4) vs 5.5 (1.0), and 6.6 (1.3) vs 5.4 (1.0), respectively; $p < 0.0001$; Figure 4), and the maximum width (11.4 (2.6) vs 10.8 (2.5); $p = 0.0429$; Figure 4B). In the meantime, the level

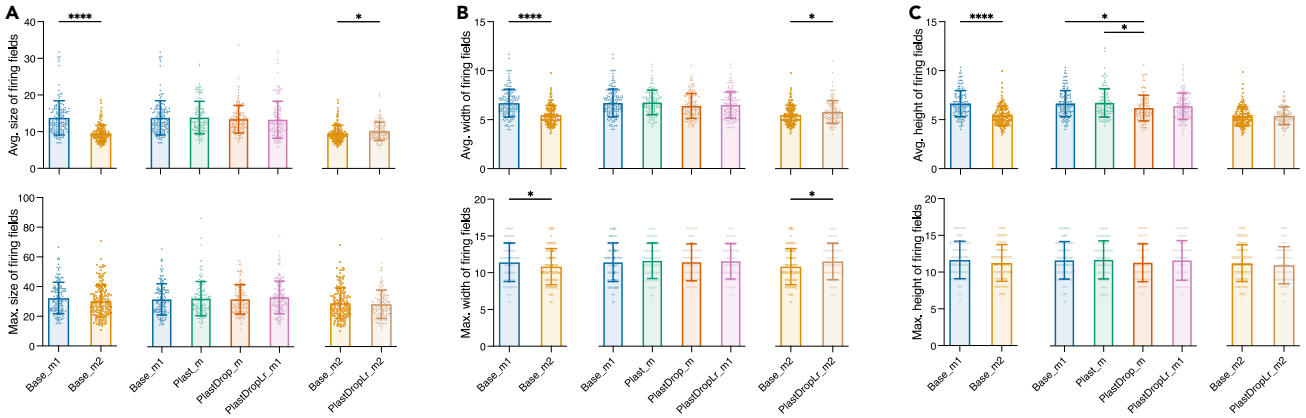


Figure 4. Effects of learning rate and neuromodulatory components on the size of grid cell-like unit firing fields in TensorFlow models

(A) average and maximum size of firing fields, (B) average and maximum width of firing fields, and (C) average and maximum height of firing fields.*: $p < 0.05$, ****: $p < 0.0001$. Data are presented as mean (SD).

of firing field activity became significantly lower (avg. level of activity: 84.8 (33.7) vs 20.5 (5.0), max. level of activity: 141.0 (61.8) vs 40.5 (13.3); $p < 0.0001$; Figure 5).

Effects of single unit level neuromodulation on unit activities (Base_m1 vs Plast_m). Adding plastic components to **Base_m1** did not alter the number of grid cell-like units (Figure 3A), number of firing fields per unit (Figure 3B), inter-field distance (Figure 3C), the average and maximum size, width, and height of firing fields (Figure 4), and the average and maximum levels of firing field activity (Figure 5).

Effects of layer level neuromodulation on unit activities (Plast_m vs PlastDrop_m). Incorporating dropout probability modulation into **Plast_m** led to a lower inter-field distance (10.1 (1.7) vs 9.5 (1.4); $p = 0.0036$; Figure 3C) and a higher number of firing fields per unit (5.9 (1.3) vs 6.3 (1.3); $p = 0.0492$; Figure 3B). The average height of firing fields decreased from 6.7 (1.4) to 6.2 (1.3) ($p = 0.0182$; Figure 4C). Moreover, both the average and maximum level of firing field activity decreased when dropout probability modulation was used (avg. level of activity: 85.1 (26.1) vs 47.4 (21.7), max. level of activity: 131.0 (41.5) vs 78.6 (36.2); $p < 0.0001$; Figure 5). Nonetheless, no significant change was observed in the number of grid cells (Figure 3A).

Effects of network level neuromodulation on unit activities (PlastDrop_m vs PlastDropLr_m1). Applying a learning rate decay to a DNN of a fixed learning rate of 1e-5 increased the number of grid cell-like units (Figure 3A) as well as the inter-field distance (9.5 (1.4) vs 10.2 (1.9); $p = 0.0061$; Figure 3C), and decreased the average and maximum levels of firing field activity (avg. level of activity: 47.4 (21.7) vs 21.6 (16.9), max. level of activity: 78.6 (36.2) vs 39.9 (31.5); $p < 0.0001$; Figure 5). Using a decaying instead of fixed learning rate did not affect the number of firing fields (Figure 3B), and the average and maximum size, width, and height of firing fields (Figure 4).

Learning rate-dependent effects of neuromodulation (Base_m1 – PlastDropLr_m1 vs Base_m2 – PlastDropLr_m2). Although introducing neuromodulatory components to **Base_m1** gave rise to a non-significant increase in the number of grid cell-like units (**Base_m1** vs **PlastDropLr_m1**: 28.2 (4.6) vs 32.6 (2.3); $p = 0.1769$; Figure 3A), a neuromodulated version of **Base_m2** yielded a lower number of grid cell-like units (**Base_m2** vs **PlastDropLr_m2**: 37.6 (6.9) vs 29.8 (2.2); $p = 0.0466$; Figure 3A). Similar effects were seen in the number of firing fields, where the neuromodulated version of **Base_m1** had a marginally higher number of firing fields on average than its neuromodulated version (**Base_m1** vs **PlastDropLr_m1**: 5.9 (1.4) vs 6.3 (1.6); $p = 0.1454$; Figure 3B), while the neuromodulated version of **Base_m2** had fewer firing fields (**Base_m2** vs **PlastDropLr_m2**: 8.0 (1.3) vs 7.7 (1.4); $p = 0.0164$; Figure 3B).

Incorporating neuromodulatory components into **Base_m1** led to no change in average size or width of firing fields; however, the neuromodulated version of **Base_m2** had firing fields of greater size and width (**Base_m2** vs **PlastDropLr_m2**: 9.5 (2.2) vs 10.2 (2.5) and 5.5 (1.0) vs 5.8 (1.2), $p = 0.0168$ and 0.0104,

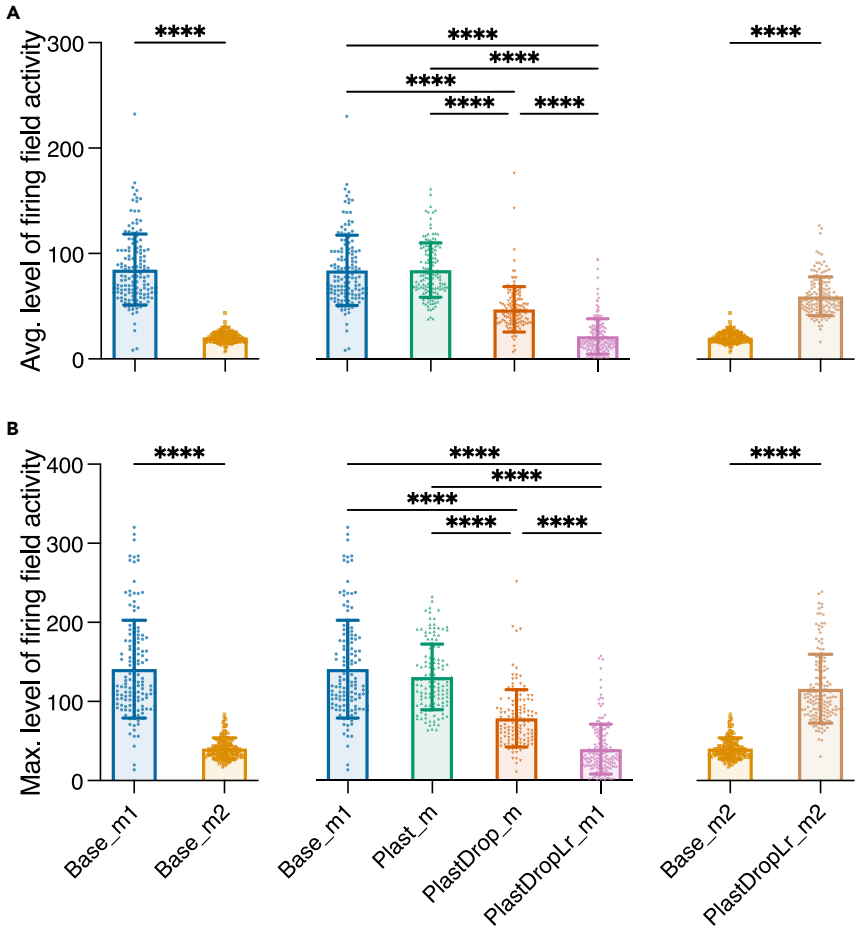


Figure 5. Effects of learning rate and neuromodulatory components on firing field activities in TensorFlow models

(A) average level of activity, (B) maximum level of activity. ****: $p < 0.0001$. Data are presented as mean (SD).

respectively; Figure 4). While neuromodulatory components did not alter the maximum width of firing fields of grid cell-like units in **Base_m1**, adding these components to **Base_m2** caused an overall increase in the maximum width (**Base_m2** vs **PlastDropLr_m2**: 10.8 (2.5) vs 11.5 (2.5); $p = 0.0112$; Figure 4B).

Overall, compared to **Base_m1**, the neuromodulated version **PlastDropLr_m1** had lower levels of activity (**Base_m1** vs **PlastDropLr_m1**, avg. level of activity: 84.8 (33.7) vs 21.6 (16.9), max. level of activity: 141.0 (61.8) vs 39.9 (31.5); $p < 0.0001$; Figure 5). On the contrary, a base model of a learning rate of 1e-4 showed a higher level of activity when neuromodulatory components were incorporated (**Base_m2** vs **PlastDropLr_m2**, avg. level of activity: 20.5 (5.0) vs 59.6 (18.4), max. level of activity: 40.5 (13.3) vs 116.0 (43.5); $p < 0.0001$; Figure 5).

Overall, we have observed faster model convergence and lower training and test losses when multilevel neuromodulation-inspired mechanisms were incorporated into the base model. In the meantime, these mechanisms affected activity patterns of fully connected layer grid cell-like units, as indicated by their firing field properties.

PyTorch implementation

To replicate all experiments on another framework where platform-dependent implementation and optimization are different, and to compare whether the effects of neuromodulatory components on learning and unit activities in one platform (TensorFlow) will persist on the other (PyTorch), we evaluated model learning and unit activity patterns using comparable procedures using PyTorch models.

Table 4. Loss before and after training, lowest loss during training, and change in loss over 1,000 epochs in PyTorch models

	Training loss, epoch 0	Test loss, epoch 0	Training loss, epoch 1,000	Test loss, epoch 1,000	Training loss, lowest	Test loss, lowest	$Loss_{l=1000} - Loss_{l=0}$, training	$Loss_{l=1000} - Loss_{l=0}$, test
Base_m1	8.036 (0.0007)	8.036 (0.0004)	2.069 (0.190)	2.077 (0.192)	2.045	2.050	5.967	5.959
Base_m2	8.029 (0.0007)	8.029 (0.0003)	2.029 (0.021)[a]	2.037 (0.008)[a]	1.988	1.998	6	5.992
Plast_m	8.036 (0.001)	8.036 (0.001)	2.069 (0.197)	2.084 (0.197)	2.041	2.048	5.967	5.952
PlastDrop_m	8.035 (0.001)	8.035 (0.000)	2.036 (0.191)	2.012 (0.187)	2.028	2.036	5.999	6.023
PlastDropLr_m1	8.029 (0.001)	8.029 (0.000)	2.181 (0.227)	2.159 (0.224)	2.203	2.212	5.848	5.87
PlastDropLr_m2	8.033 (0.000)	8.033 (0.000)	1.852 (0.174)	1.852 (0.173)	1.846	1.861	6.181	6.181

Data are presented as mean (SD)
[a]Training and test loss for **Base_m2** at epoch 500 are reported.

Spatial learning in non-neuromodulated and neuromodulated DNNs

Custom model loss. All model variants demonstrated a test loss of approximately 8.0 before training (Table 4; Figure 6A). Adding Hebbian-based plastic components resulted in a test loss of 2.084 in **Plast_m**, comparable to the test loss of 2.077 of the base model (**Base_m1**). Increasing the learning rate of **Base_m1** to 1e-4 (**Base_m2**) resulted in a diverging loss after around 500 epochs (Figures 6A and S2). The test loss of **Base_m2** at 500 epochs was 2.037, lower than **Base_m1** and **Plast_m**. Modulation of dropout probability in **PlastDrop_m** resulted in a lower final test loss of 2.012 compared to **Base_m1**, **Base_m2**, and **Plast_m**. While scheduling learning rate on log scale from 1e-5 to 1e-6 as in **PlastDropLr_m1**, the final test loss was 2.159, higher than all other model variants. Scheduling the learning rate on log scale from 1e-4 to 1e-5 in the **PlastDropLr_m2** variant, however, resulted in a final test loss of 1.852, showcasing lowest loss among all variants (Figure 6A; Table 4).

Error of ambulation. The ambulation error of all model variants was around 1.2m prior to training (Table 5; Figure 6B). After training for 1,000 epochs, similar to the TensorFlow version, the ambulation error obtained from training and test sets was comparable (Table 5). The base model **Base_m1** demonstrated an error of ambulation of 0.102. Adding plastic components as in the **Plast_m** variant led to an error of ambulation of 0.103, comparable to **Base_m1**. As lowering the learning rate of **Base_m1** to 1e-4 in the **Base_m2** variant resulted in diverging loss after 500 epochs, the ambulation error of 0.101 at 500 epochs was considered instead. Compared to **Base_m1** and **Plast_m**, the error of ambulation of **Base_m2** was marginally lower. Dynamically modulating dropout probability in **PlastDrop_m** resulted in an error of ambulation of 0.101, comparable to the **Base_m2** variant and lower than **Base_m1** and **Plast_m**. Scheduling learning rate from 1e-5 to 1e-6 on log scale in **PlastDropLr_m1** gives rise to an ambulation error of 0.107, higher than all other PyTorch variants. Scheduling the learning rate from 1e-4 to 1e-5 instead in the **PlastDropLr_m2** variant, however, resulted in an ambulation error of 0.095, the lowest among all model variants (Table 5; Figure 6B).

Effects of optimizer, learning rate, and gradient clipping. When using the Stochastic Gradient Descent (SGD) optimizer, very slow model convergence and decrease in loss were observed when the learning rate was set to 1e-4 or 1e-5 (Figure S3), resulting in a final test loss of 4.157 (0.009) or 7.754 (0.004). We then increased the learning rate to 1e-3 and obtained a higher final test loss compared to the base model with the RMSprop optimizer (**Base_SGD** vs **Base_m1**: 2.263 vs 2.077). Experimenting with lower learning rates (learning rate = 1e-6, **Base_loLR**), stricter gradient clipping (gradient clipping = 1e-6, **Base_SGC**), or both (**Base_loLR_SGC**) did not lead to hexagonal grid cell patterns (Figure 7).

Effects of neuromodulatory components on grid-like cell activity patterns in the fully connected layer. The PyTorch models rarely had any place cell-like or border cell-like activity patterns (Figure 8). Place and head direction cell-like unit activities were observed in **PlastDropLr_m1**, but not other model variants. Firing fields in units with grid cell-like activities were organized into square-like patterns, and a

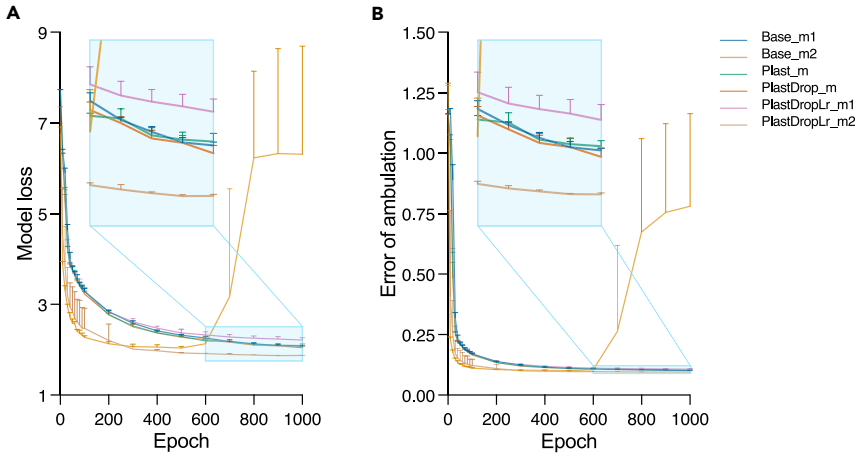


Figure 6. Performance of PyTorch models

(A) custom model loss, (B) error of ambulation. Data shown were obtained from the test set. For each model, results were the average of 5 trials. Data are presented as mean (SD).

larger number of units with stripe-cell or conjunctive cell-like activity patterns were identified. Head direction cell-like unit activities were present in the PyTorch implementation (Figure 8). Biologically plausible unit activity patterns emerged early during model training (Figure S4).

Base model learning rate (Base_m1 vs Base_m2). Changing the learning rate of **Base_m1** to 1e-4 caused a significant decrease in the number of grid cells (**Base_m1** vs **Base_m2**, 33.00 (6.245) vs 7.800 (3.962), $p < 0.0001$; Figure 9A), average level of activity (**Base_m1** vs **Base_m2**, 16.29 (4.344) vs 2.861 (1.004), $p < 0.0001$; Figure 11A), and maximum level of activity (28.91 (7.662) vs 5.210 (1.735), $p < 0.0001$; Figure 11B). No significant change was observed in the number of firing fields per unit (Figure 9B), inter-field distance (Figure 9C), or size/width/height of firing fields (Figure 10).

Effects of single unit level neuromodulation on unit activities (Base_m1 vs Plast_m). Adding a plastic component resulted in significant decrease in the number of grid-like cells (**Base_m1** vs **Plast_m**, 33.00 (6.245) vs 29.40 (6.580), $p = 0.0118$; Figure 9A) and significant increase in the average level of firing field activity (16.29 (4.344) vs 18.39 (4.832), $p = 0.0147$; Figure 11A) metrics. However, it did not cause significant changes in maximum firing field activity (Figure 11B), number of firing fields per unit (Figure 9B), inter-field distance (Figure 9C), average and maximum size of firing fields (Figure 10A), or max/average width/height of firing fields metrics (Figures 10B and 10C).

Table 5. Error of ambulation before and after training, lowest error of ambulation, and change in error of ambulation over 1,000 epochs in PyTorch models

	Training error of ambulation (m), epoch 0	Test error of ambulation (m), epoch 0	Training error of ambulation (m), epoch 1,000	Test error of ambulation (m), epoch 1,000	Training error of ambulation (m), lowest	Test error of ambulation (m), lowest	$\varepsilon_{l=1000} - \varepsilon_{l=0}$, training	$\varepsilon_{l=1000} - \varepsilon_{l=0}$, test
Base_m1	1.174 (0.557)	1.175 (0.557)	0.102 (0.059)	0.102 (0.059)	0.101	0.101	1.072	1.073
Base_m2	1.184 (0.560)	1.185 (0.561)	0.101 (0.061)[a]	0.101 (0.061)[a]	0.100	0.100	1.083	1.084
Plast_m	1.201 (0.568)	1.202 (0.569)	0.103 (0.060)	0.103 (0.060)	0.101	0.102	1.098	1.099
Plast Drop_m	1.245 (0.586)	1.244 (0.586)	0.101 (0.059)	0.101 (0.059)	0.101	0.101	1.144	1.143
PlastDrop Lr_m1	1.213 (0.573)	1.212 (0.573)	0.107 (0.073)	0.107 (0.072)	0.107	0.107	1.106	1.105
PlastDrop Lr_m2	1.196 (0.564)	1.197 (0.564)	0.095 (0.055)	0.095 (0.056)	0.094	0.095	1.101	1.102

Data are presented as mean (SD)

[a]Training and test error of ambulation for **Base_m2** at epoch 500 are reported.

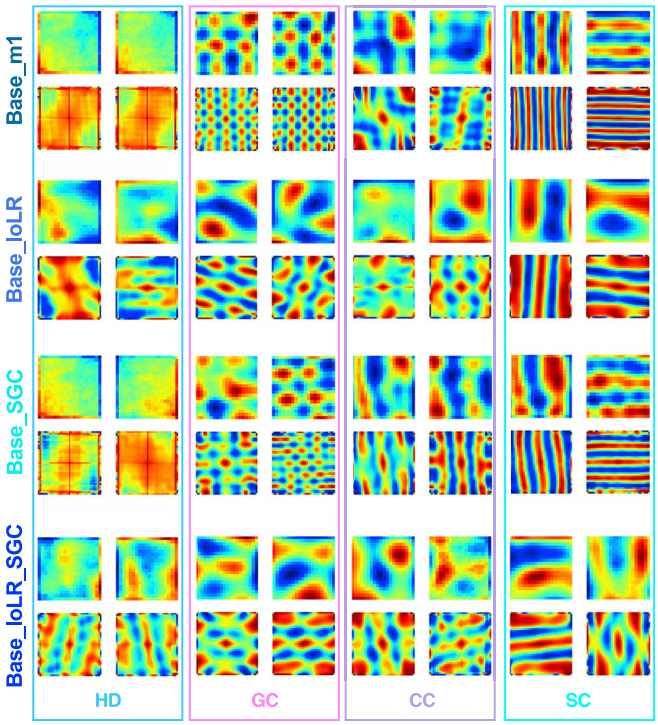


Figure 7. Effects of lower learning rate (Base_loLR), stricter gradient clipping (Base_SGC), or both (Base_loLR_SGC) on unit activities compared to the base model (Base_m1)

Examples of units exhibiting head direction cell-like (HD), grid cell-like (GC), conjunctive cell-like (CC), and stripe cell-like (SC) unit activities are presented.

Effects of layer level neuromodulation on unit activities (Plast_m vs PlastDrop_m). Modulation of dropout probability in the fully connected layer of the DNN caused significant decrease in the average and maximum level of firing field activities (18.39 (4.832) vs 13.94 (3.6), $p < 0.0001$; Figures 11A and 32.08 (8.428) vs 24.83 (6.526), $p < 0.0001$; Figure 11B respectively). The number of grid-like cells (Figure 9A), number of firing fields (Figure 9B), inter-field distance Figure 9C), and max/average size/width/height of firing fields (Figure 10) did not see significant changes.

Effects of network level neuromodulation on unit activities (PlastDrop_m vs PlastDropLR_m1). The addition of learning rate scheduling resulted in a significant increase in the number of grid cell-like units (**PlastDrop_m** vs **PlastDropLr_m1**, 21.20 (6.611) vs 31.20 (0.8367), $p < 0.0430$; Figure 9A), inter-field distance (10.29 (2.027) vs 12.93 (2.030), $p < 0.0001$; Figure 9C), average size of firing fields (16.80 (5.543) vs 24.21 (7.703), $p < 0.0001$; Figure 10A), average width of firing fields (4.684 (1.192) vs 5.480 (1.338), $p < 0.0001$; Figure 10B), and maximum and average height of firing fields (7.566 (1.966) vs 8.833 (2.146), $p < 0.0001$ and 4.419 (1.

090) vs 5.509 (1.361), $p < 0.0001$ respectively; Figure 10C). The addition of learning rate scheduling also caused a significant decrease in the number of firing fields (6.019 (1.852) vs 4.179 (1.322), $p < 0.0001$; Figure 9B). No significant changes were observed in the average and maximum levels of firing field activity (Figure 11).

Learning rate-dependent effects of neuromodulation (Base_m1 – PlastDropLr_m1 vs Base_m2 – PlastDropLr_m2). The addition of neuromodulatory components had differing effects based on the learning rate employed. While using a learning rate of 1e-5, adding neuromodulatory components (**Base_m1** vs **PlastDropLr_m1**) resulted in a significant increase in the inter-field distance (10.67 (1.758) vs 12.93 (2.030), $p < 0.0001$; Figure 9C), average size of firing fields (17.77 (6.005) vs 24.21 (7.703), $p < 0.0001$; Figure 10A), maximum size of firing fields (37.33 (11.55) vs 46.99 (14.68), $p < 0.0001$; Figure 10A), average width of firing fields (4.865 (1.241) vs 5.480 (1.338), $p = 0.0006$; Figure 10B), average height of firing

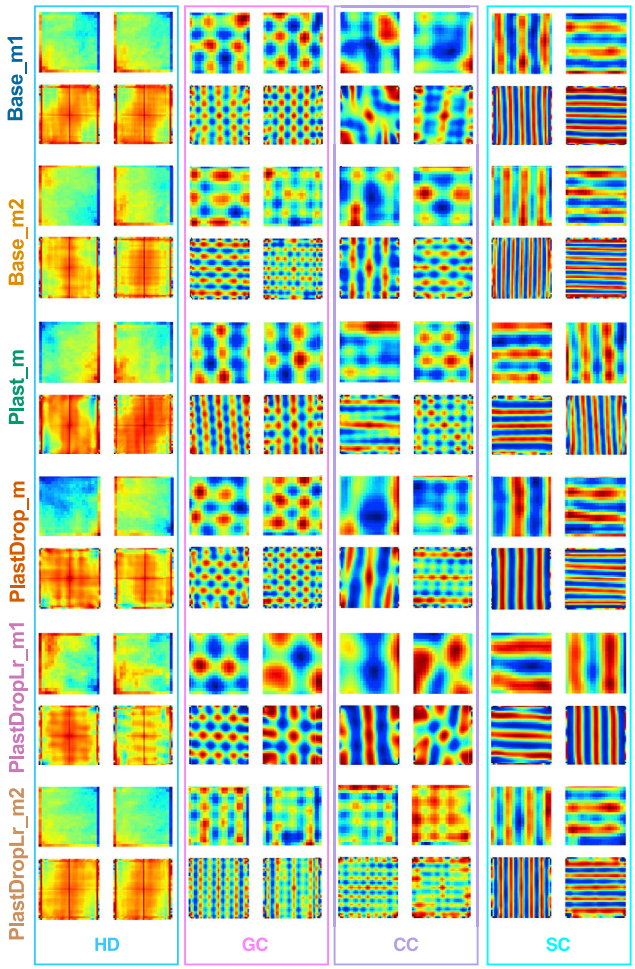


Figure 8. Unit activity patterns and autocorrelograms of fully connected layer units in PyTorch models after training

HD, head direction cell-like units; GC, grid cell-like units; CC, conjunctive cell-like units; SC, stripe cell-like units. Upper rows represent unit activity maps and lower rows represent corresponding autocorrelograms.

fields (4.643 (1.155) vs 5.509 (1.361), $p < 0.0001$; Figure 10C), and maximum height of firing fields (8.085 (2.117) vs 8.833 (2.146), $p = 0.0319$; Figure 10C). Additionally, a significant decrease in the number of firing fields (5.709 (1.711) vs 4.179 (1.322), $p < 0.0001$; Figure 9B) was observed in this setting. Adding neuromodulatory components when using a learning rate of 1e-4 (**Base_m2** vs **PlastDropLr_m2**), however, caused a significant increase in another set of metrics: the average (2.861 (1.004) vs 12.90 (4.204), $p < 0.0001$; Figure 11A) and maximum (5.210(1.735) vs 24.70(7.962), $p < 0.0001$; Figure 11B) levels of activity.

DISCUSSION

In the present study, we introduce a multiscale self-modulated framework to a DNN for a spatial learning task, where neuromodulation-inspired components were added at the single unit, layer, and network levels. We used a DNN with path integration abilities thus allowing the DNN to determine and update its current location without use of landmarks or explicit training signals such as deviations from the actual trajectory. Instead, the DNN was trained through minimizing the difference between predicted and actual place and head direction cell activations, that is, spatial learning performance was improved based on activities of neurons.

To examine the effects of neuromodulatory mechanisms on spatial learning and fully connected layer unit activities, and to enable cross-platform comparisons, we implemented a base model and five variants using

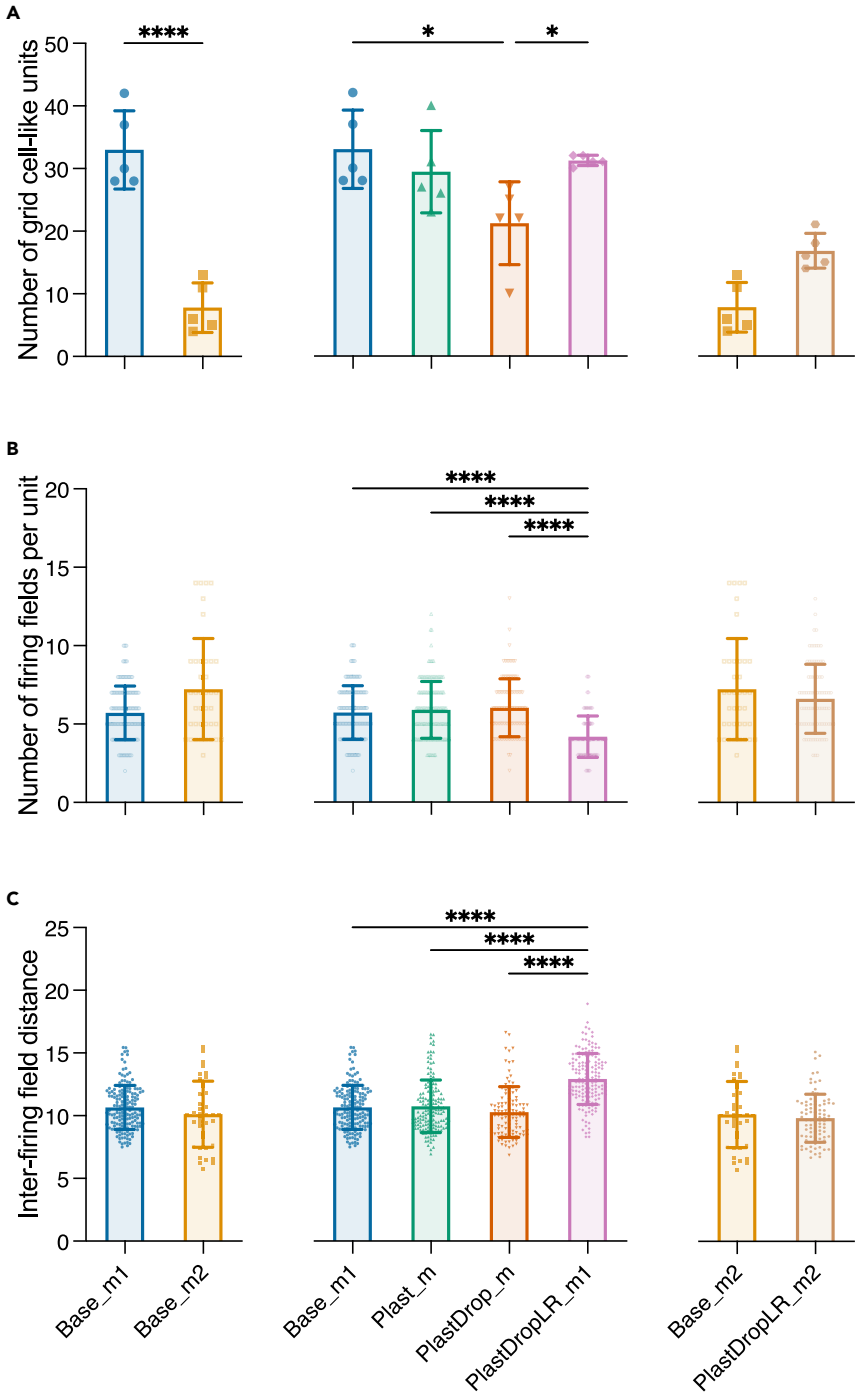


Figure 9. Effects of learning rate and neuromodulatory components on grid cell-like units and their general properties in PyTorch models

(A) number of grid cell-like units, (B) number of firing fields in each unit, and (C) inter-field distance. *: $p < 0.05$, ****: $p < 0.0001$. Data are presented as mean (SD).

both TensorFlow and PyTorch. To our knowledge, the PyTorch models in this study are the first PyTorch implementation of the TensorFlow models in the study by Banino et al.[19] that demonstrate biologically plausible unit activities. Overall, neuromodulatory components facilitated path integration in an open field and led to faster model convergence.

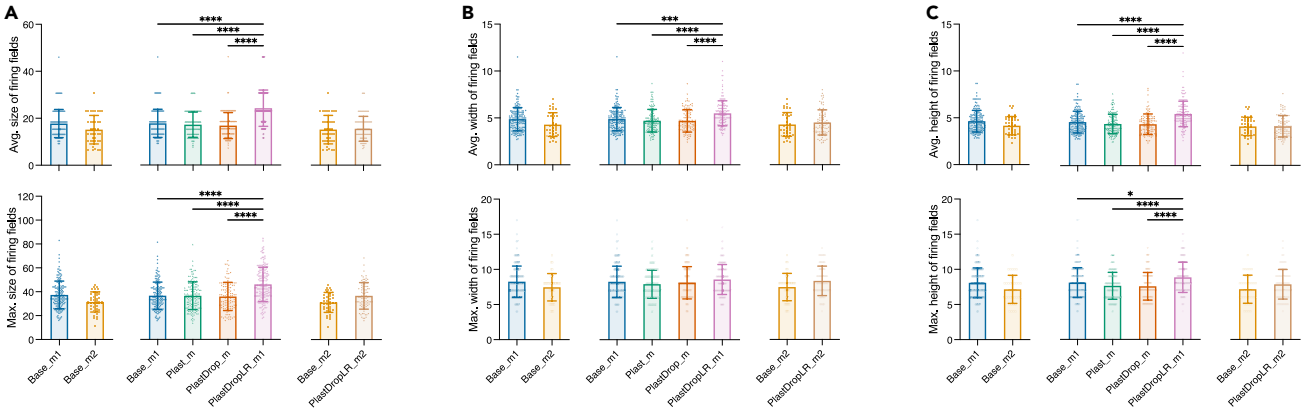


Figure 10. Effects of learning rate and neuromodulatory components on the size of grid cell-like unit firing fields in PyTorch models

(A) average and maximum size of firing fields, (B) average and maximum width of firing fields, and (C) average and maximum height of firing fields. *: $p < 0.05$, ***: $p < 0.001$, ****: $p < 0.0001$. Data are presented as mean (SD).

More specifically, in the neuromodulated model variants, we observed (1) faster model convergence during training, (2) lower model loss during and after training, (3) lower error of ambulation during and after training, (4) changes in fully connected layer grid cell-like unit activity patterns, and (5) hyperparameter-dependent neuromodulatory effects. Moreover, there were marked differences in unit activity patterns,

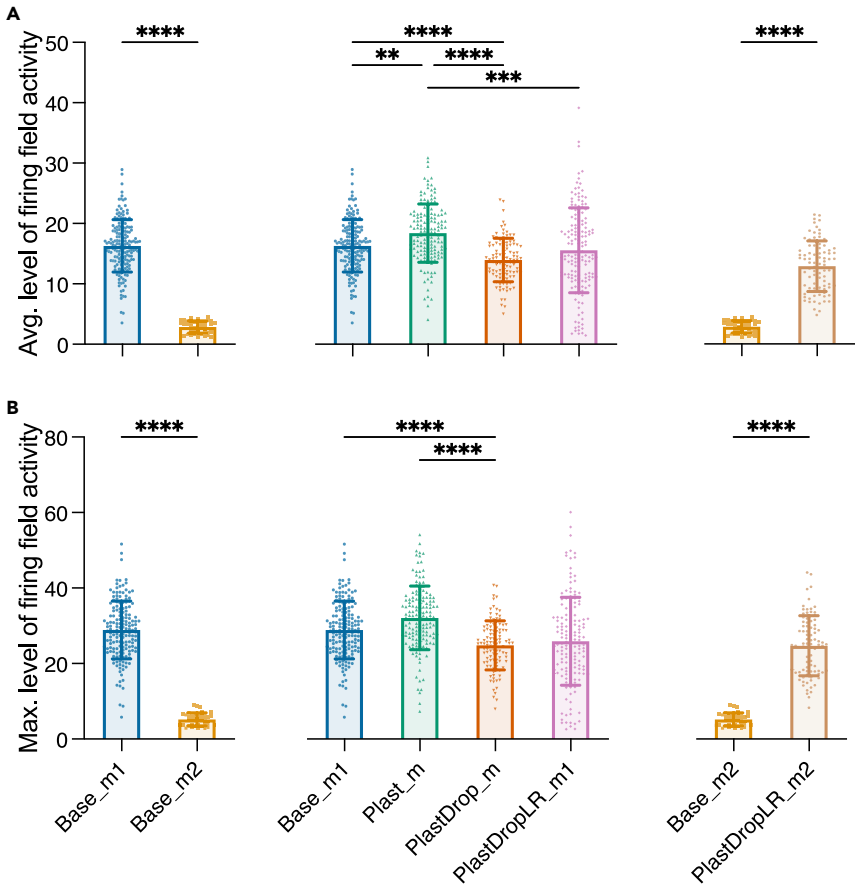


Figure 11. Effects of learning rate and neuromodulatory components on firing field activities in PyTorch models

(A) average level of activity, (B) maximum level of activity. **: $p < 0.01$, ***: $p < 0.001$, ****: $p < 0.0001$. Data are presented as mean (SD).

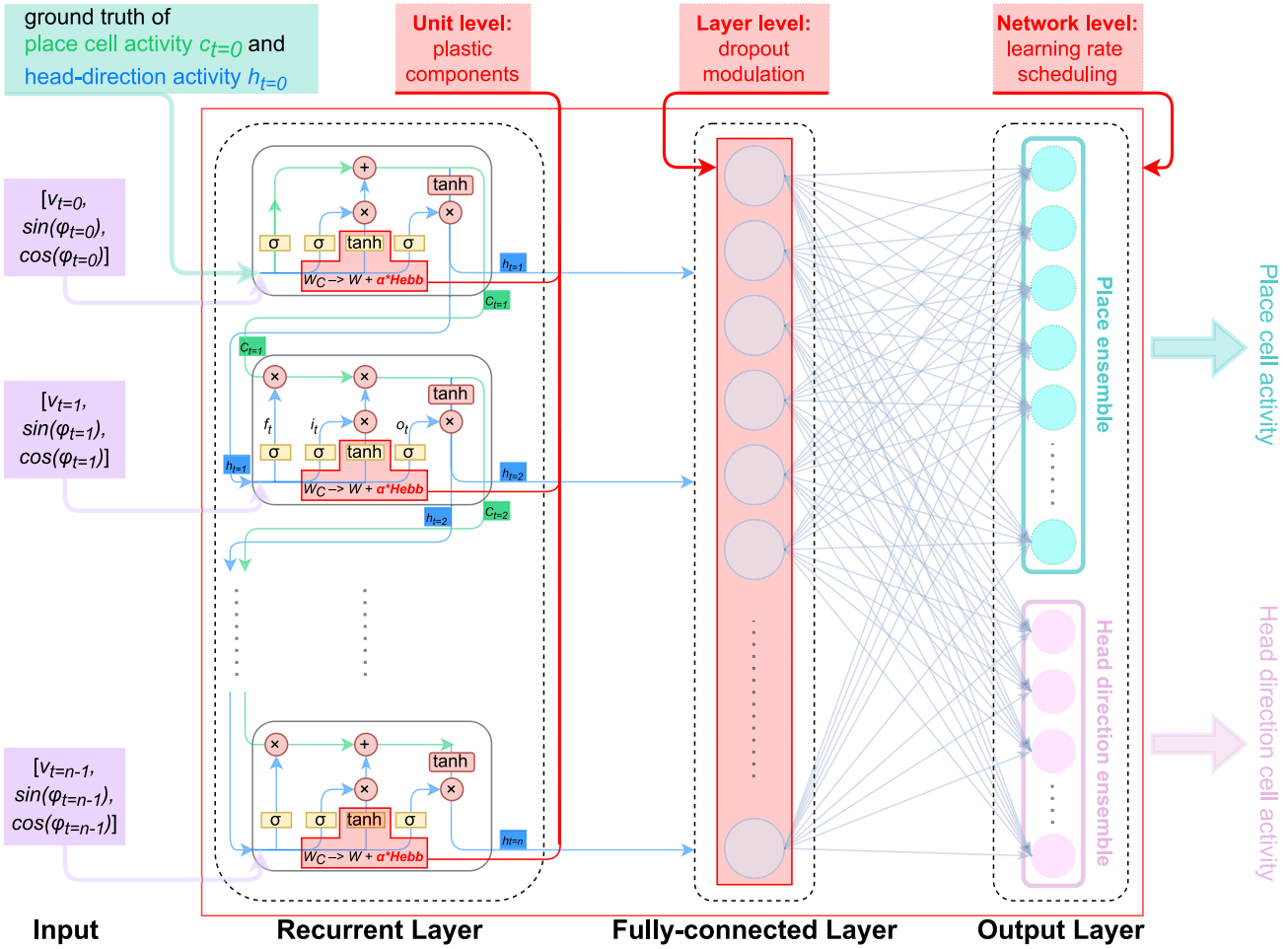


Figure 12. Architecture of the deep neural network with neuromodulation-inspired mechanisms

Red, neuromodulatory components introduced at the single unit (LSTM units of the recurrent layer), layer (adaptive dropout probability of the fully connected layer), and network levels (learning rate scheduling).

and in how neuromodulatory components affect unit activities in the TensorFlow and PyTorch implementations (Table S2). These observations have thus allowed us to conclude that incorporating neuromodulation-inspired mechanisms at different spatial scales may facilitate faster model convergence and smaller ambulation errors, and in the meantime, the modulatory effects of these mechanisms on single unit activities can be affected by the platform.

Neuromodulatory components enabled and speeded up model training

In the PyTorch base model with a learning rate of 1e-4, both train and test loss decreased to approximately 2.0 at 500 epochs and then rebounded to approximately 8.0 (Figure S2). This behavior usually indicates unstable training due to a learning rate set too high.[20] We also hypothesized that the diverging loss was caused by a low epsilon value in the RMSProp optimizer: In RMSProp, the calculation of effective learning rate includes a step that divides the scheduled learning rate by the square root of the moving average of squared gradients:

$$\frac{\lambda}{\sqrt{v}+\varepsilon}$$

Where λ is the scheduled learning rate, v is the moving average of gradients, and ε is used to prevent explosion of effective learning rate when v is too small.

To test this hypothesis, a set of experiments were conducted, with higher epsilon values of 1e-6 and 1e-4 compared to the TensorFlow version's 1e-10 for the RMSProp optimizer. Nevertheless, models still failed to

converge (Figure S5). However, when incorporating neuromodulation-inspired mechanisms such as learning rate decay (**PlastDropLr_m2**), even with a higher learning rate of 1e-4, the model loss decreased steadily (Figure 6A), demonstrating the effectiveness of neuromodulatory components in enabling model training.

Furthermore, neuromodulated DNNs had faster model convergence and reached comparable loss in fewer epochs when compared with the base model (Figures 1 and 6). More specifically, neuromodulated model variants had lower losses at an earlier time point compared to the base model: the loss of **Plast_m**, **PlastDrop_m**, and **PlastDropLr_m1** was 3.582, 3.597, and 3.578 at 800, 700, and 600 epochs, respectively, while the loss of **Base_m1** was 3.608 at 1,000 epochs. Moreover, the model loss of **Base_m2** at 1,000 epochs was 3.655, comparable to the model loss of **PlastDropLr_m2** at 300 epochs (3.576). Similar trend was seen in error of ambulation, with the error of ambulation of **Base_m1** being 0.137 at 1,000 epochs, while **Plast_m**, **PlastDrop_m**, and **PlastDropLr_m1** reached a lower error of ambulation at 700, 700, and 600 epochs (0.136, 0.133, and 0.134). The error of ambulation of **Base_m2** at 1,000 epochs was comparable to that of **PlastDropLr_m2** at 500 epochs (0.122 vs 0.117). PyTorch models also yielded faster convergence, smaller model loss, and error of ambulation upon incorporating neuromodulation (Figure 6).

Unit activity patterns in TensorFlow and PyTorch models

Different cell types, activity patterns, and firing field properties in TensorFlow and PyTorch models

Despite similar learning trajectories observed in the TensorFlow and PyTorch implementations, the underlying cell types and grid cell-like unit activity patterns were different. In the fully connected layer of TensorFlow models, we were able to identify grid cell-like units, place cell-like units, head direction cell-like units, and border cell-like units (Figure 2). However, in the PyTorch implementation, place cell- and border cell-like activity patterns emerged only in **PlastDropLr_m1** (Figure 8). Firing fields of grid cell-like units in TensorFlow models organized into a hexagonal pattern comparable to biological grid cells. In contrast, in models implemented using PyTorch, firing fields showed square-like patterns (Figure 8).

Although firing patterns of head direction cell-like units in the two implementations were comparable, stripy patterns in grid cell-like units in the PyTorch implementation were more prominent. Interestingly, consistent with the study by Krupic et al,[21] stripy patterns as well as square-like grid cell activities were observed in square environments. Moreover, grid cell-like units of models implemented on the two platforms had activity patterns that differed fundamentally: In TensorFlow models, firing fields of grid cell-like units organized into a hexagonal pattern, while in PyTorch models, square-like patterns were observed (Figures 2 and 8).

Understanding square-like grid patterns in PyTorch models

To further understand this divergence, we first examined the consistency between hyperparameters, and model initialization and optimization procedures. Given the identical model hyperparameters used, we concluded that a possible cause was the implementation of the RMSProp optimizer in PyTorch and TensorFlow. Therefore, we replaced the RMSProp optimizer of the PyTorch version of the base model with the SGD optimizer while keeping all other model hyperparameters unchanged. The difference in unit activity patterns, however, persisted. For experiments with the SGD optimizer, learning rate was set to 1e-3, in contrast to RMSProp version's 1e-5. This choice was made due to slow convergence of SGD variants where learning rate was set to 1e-5 or 1e-4 (Figure S3).

A recent study using the base model in the study by Banino et al.[19] reported square-like grid activity when using TensorFlow.[22] It revealed square-like grid activity when a higher learning rate of 1e-3 and loose gradient clipping of 1 were being employed. However, they observed hexagonal grid activity when learning rate and gradient clipping were set to 1e-5 as in the study by Banino et al.[19] They also reported lower final loss and faster convergence under a higher learning rate and loose gradient clipping, and concluded hexagonal grid activity patterns emerge only in low learning rate and strict gradient clipping settings. To test whether lower learning rate and stricter gradient clipping would lead to hexagonal grid patterns in the PyTorch version, **Base_loLR**, **Base_SGC**, and **Base_loLR_SGC** variants were implemented trained. The results, however, contradicted the findings in the study by Songlin et al.[22] This supports the hypothesis that unit activity patterns may depend on the deep learning framework being used.

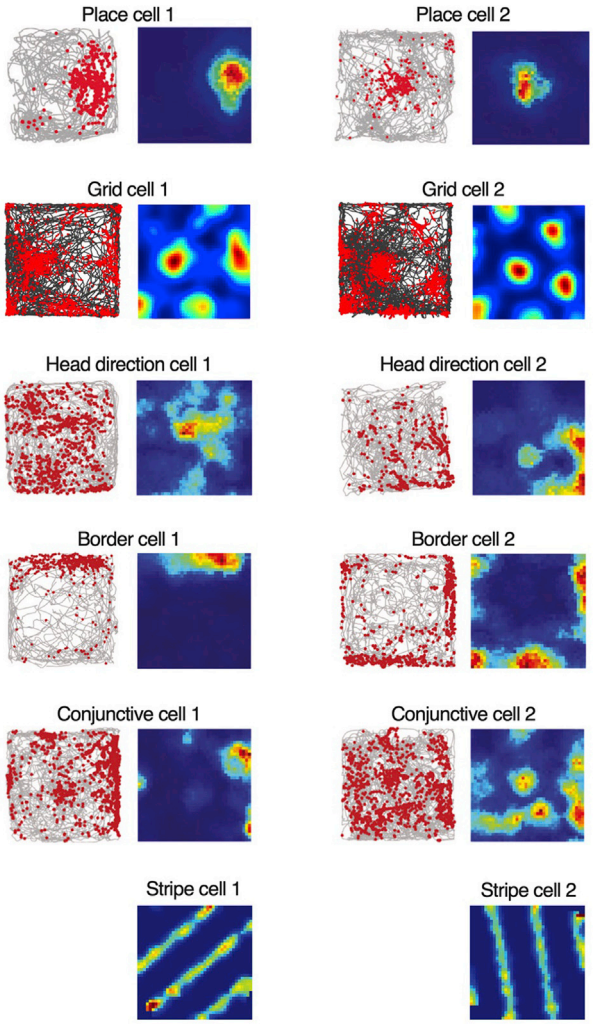


Figure 13. Trajectories (left) and firing activity maps (right) of place cells, grid cells, head direction cells, border cells, conjunctive cells, and stripe cells recorded from rodents or obtained from simulations

Lines, trajectories of ambulation; red dots, superimposed spike locations. Experimental recordings of place cells, head direction cells, border cells, and conjunctive cells were performed by Long and Zhang.[24] For grid cells, ratemaps were acquired by Gerlei et al.[25] Stripe cell activity patterns were originally obtained through simulations in a study by Pilly and Grossberg.[26]

The difference in grid cell-like unit activity patterns arising from the two platforms can also be attributed to the fact that there were no symmetry-breaking constraints in the bottleneck layers. As suggested by Sorscher et al.,[23] there are multiple solutions to the optimization problem our models were solving. One possibility can be that the PyTorch and TensorFlow versions converged to different solutions and if an additional constraint was introduced, they may have converged to the same result, namely the conventional hexagonal grid firing patterns.

Biological plausibility of fully connected layer unit activity

Biologically plausible activity patterns were observed in the fully connected layer of both TensorFlow and PyTorch models, resembling head direction cells, place cells, grid cells, border cells, stripe cells, and conjunctive cells (Figure 13 vs Figures 2 and 13 vs Figure 8). When neuromodulation-inspired mechanisms were introduced to the base models, these cell types were still identifiable based on their patterns of activity. In addition, we were able to characterize an interaction between neuromodulatory components and grid cell-like unit properties.

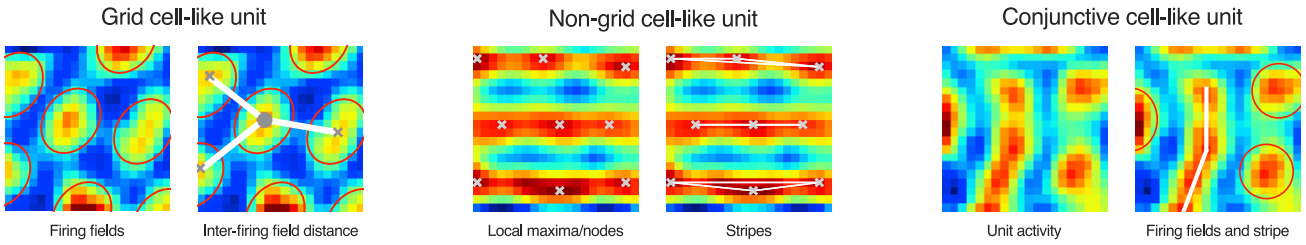


Figure 14. Identification of grid cell-like units and exclusion of non-grid cell-like units

Left: Identifying firing fields and determining the inter-firing field distance of grid cell-like units. An area that has a higher level of activity when compared with its proximity was defined as a firing field (red circles). The inter-firing field distance represents the average of distances (white lines) from the center of one firing field (gray dot) to the centers of the 3 closest firing fields (gray crosses). Middle: Units with stripe-like patterns of activities were regarded as non-grid cell-like units. Per definition, the stripe count of the ratemap is 6. Right: A unit that exhibits conjunctive properties, showing both firing fields and stripes.

Although activity patterns of some fully connected layer units may not look biologically plausible as they had non-grid cell-like, stripy, or square-like grid patterns, their activities resemble some of the spatially periodic cells (SPC) reported in the study by Krupic et al.[27] The SPCs observed in the study by Krupic et al.[27] had a different number of significant Fourier components between one and four. Grid cells mostly had three significant Fourier components resulting in hexagonal firing patterns, but while there were grid cells with two significant components, their firing patterns were not generally square-like as in the PyTorch model since the significant components were not orthogonal to each other, except very few instances. Moreover, among non-grid SPCs, there were a significant number of cells with one significant Fourier component, referred to as cells with a band-like firing pattern by Krupic et al.[27] For these cells, there is also computational evidence suggesting that combining multiple of them can lead to grid-like activities.[26,28,29]

Furthermore, a large number of SPCs in the study by Krupic et al.[27] seemed to be cells with conjunctive properties. In alignment with computational and experimental findings,[30,31] a significant number of units in both TensorFlow and PyTorch versions had similar activity patterns to conjunctive cells. Overall, similar to the population of cells responsible for spatial learning in the brain, ratemaps obtained from models in this study showcase a diversity of activity patterns.

Limitations of the study

In the present study, classification of grid cell-like units was performed using an automated pipeline. Although all units were inspected visually to ensure that there are no false positives (i.e., assuming a unit is grid cell-like while it is not), there may have been some false negatives in the attempt to exclude stripe cells in the PyTorch implementation. Furthermore, the study did not consider units with conjunctive properties, from which irregular activity patterns, instead of grid-like or regular patterns, could be observed.[24] Thus, for the analysis of grid cell-like unit activity patterns, units displaying conjunctive cell-like activities were excluded. For example, a unit with 6 firing fields organized in a grid-like pattern but having 2 fields connected with each other thus showing a stripe-like pattern was excluded.

Importantly, for the statistical analysis of the PyTorch implementation, data of grid cell-like units from the **Base_m2** variant were acquired at 500 epochs, in contrast to other variants where unit activities at 1,000 epochs were analyzed. The main reason was the divergence of model loss after 500 epochs in **Base_m2**. Given this approach may have biased model comparisons that involved **Base_m2**, we examined the evolution of ratemaps of all model variants to ensure that early development of unit activities occurred earlier than epoch 500 (Figure S6). Moreover, as shown in Figure S6, the difference between ratemaps extracted at epoch 500 and epoch 1,000 was not significant. Nevertheless, given the differences in the optimization procedure of the two platforms, and the early stopping in **Base_m2**, it was challenging to directly assess and understand the differences in the effects that neuromodulatory components had on model learning and unit activities.

It is worth noting that DNNs in the present study receive angular and linear velocities, rather than employing sensory inputs such as visual[32–35] and auditory[36–41] signals, or even information such as terrain slant.[42] Essentially, based on a functional equivalence view, different sensory inputs can result in corresponding spatial representations, exploratory behavior, and navigation performance.[42,43] In fact, studies

demonstrating the formation of place cells prior to grid cells[44,45] and retention of place cell activities after disrupting grid cell activities[46] suggest that place cells receive inputs from various types of cells in parallel.[47] The emergence of some of these types of non-grid cells may have been interfered by the use of precise velocities as an input, thus leading to unit activities that diverge from biologically plausible patterns.

Overall, the proposed framework was loosely inspired by the multiscale nature of neuromodulatory processes in the brain. Therefore, the spatiotemporal dynamics of neuromodulation, the effects of chemical neuromodulation on processes at the cellular level, and the detailed morphological settings of individual neurons are yet to be realized.[48]

Outlook

Based on its aims, scope, and limitations, we may consider the following directions for furthering the present study: First, we did not perform a thorough parameter space search to fully probe the interplay between model hyperparameters and neuromodulatory components. Instead, given the limited time and computational resources, we primarily used a learning rate of 1e-5 and 1e-4. Therefore, we may consider performing a parameter space search to test combinations of hyperparameters and neuromodulatory components exhaustively, to investigate how hyperparameter-dependent effects of neuromodulation emerged. Secondly, in the present study, we performed classification of fully connected layer units based on similarity between artificial and biological neuronal activity patterns, without access to information of the actual cell type (i.e., labels). To enable more efficient and accurate classification of cell types using convolutional neural networks, more data on firing patterns and properties of biologically plausible artificial neurons, especially conjunctive cells, should be acquired and made available. Thirdly, individual neuromodulatory components could be further developed to allow for higher complexity and biological plausibility. One example is the plastic component, where a more complex mechanism using retroactive neuromodulation and eligibility traces could be enabled.[8] Finally, the proposed multiscale neuromodulatory framework could be added to other neural network architectures or DNNs of greater complexity, and tested in various behavioral contexts, for examining its robustness.

To sum up, neuromodulation-inspired mechanisms can speed up model training and lead to smaller errors in a spatial learning task. More importantly, the effects of these mechanisms may depend on model hyperparameters and the platform of implementation, further reinforcing the importance of transparency in reporting experimental details and model training protocols, in studies that use biologically plausible DNNs. Therefore, for future studies on neuromodulation-inspired learning in DNNs, extra care should be taken during model implementation, statistical analysis, interpretation of results, and reporting of experimental procedures, to allow for maximal reproducibility and comparability across studies.

STAR★METHODS

Detailed methods are provided in the online version of this paper and include the following:

- KEY RESOURCES TABLE
- RESOURCE AVAILABILITY
  - Lead contact
  - Materials availability
  - Data and code availability
- METHOD DETAILS
  - Dataset
  - Neural network architecture and artificial neuromodulation
  - Model variants, implementation, and training
  - Classification of neurons
  - Evaluation metrics
- QUANTIFICATION AND STATISTICAL ANALYSIS

SUPPLEMENTAL INFORMATION

Supplemental information can be found online at https://doi.org/10.1016/j.isci.2023.106026.

ACKNOWLEDGMENTS

J.M. acknowledges support from a BrainsCAN Postdoctoral Fellowship Award through Canada First Research Excellence Fund (CFREF). R.M. acknowledges support from the Vector Scholarship in Artificial Intelligence, provided through the Vector Institute, and Western University via Western Graduate Research Program (WGRS). Y.M. acknowledges the support from BrainsCAN at Western University through the Canada First Research Excellence Fund (CFREF).

AUTHOR CONTRIBUTIONS

Conceptualization: J.M. and Y.M.; Methodology: J.M.; Software: J.M. and R.M.; Validation: J.M. and R.M.; Formal analysis: J.M. and R.M.; Writing-original draft: J.M. and R.M.; Writing-review & editing: J.M., R.M., and Y.M.; Visualization: J.M. and R.M.; Supervision and project administration: J.M. and Y.M.; Funding acquisition: J.M. and Y.M.

DECLARATION OF INTERESTS

The authors declare no competing interests.

Received: September 9, 2022
Revised: November 18, 2022
Accepted: January 19, 2023
Published: January 23, 2023

REFERENCES

1. Guerguiev, J., Lillicrap, T.P., and Richards, B.A. (2017). Towards deep learning with segregated dendrites. Elife *6*, e22901. https://doi.org/10.7554/eLife.22901.

2. Lindsey, J., and Litwin-Kumar, A. (2020). Learning to learn with feedback and local plasticity. Preprint at arXiv. https://doi.org/10.48550/arXiv.2006.09549.

3. Mei, J., Muller, E., and Ramaswamy, S. (2022). Informing deep neural networks by multiscale principles of neuromodulatory systems. Trends Neurosci. *45*, 237–250. https://doi.org/10.1016/j.tins.2021.12.008.

4. Mesnard, T., Vignoud, G., Sacramento, J., Senn, W., and Bengio, Y. (2019). Ghost units yield biologically plausible backprop in deep neural networks. Preprint at arXiv. https://doi.org/10.48550/arXiv.1911.08585.

5. Wu, X., Liu, X., Li, W., and Wu, Q. (2018). Improved expressivity through dendritic neural networks. In Advances in Neural Information Processing Systems (Curran Associates, Inc.).

6. Beaulieu, S., Frati, L., Miconi, T., Lehman, J., Stanley, K.O., Clune, J., and Cheney, N. (2020). Learning to continually learn. Preprint at arXiv. https://doi.org/10.48550/arXiv.2002.09571.

7. Ellefsen, K.O., Mouret, J.-B., and Clune, J. (2015). Neural modularity helps organisms evolve to learn new skills without forgetting old skills. PLoS Comput. Biol. *11*, e1004128. https://doi.org/10.1371/journal.pcbi.1004128.

8. Miconi, T., Rawal, A., Clune, J., and Stanley, K.O. (2020). Backpropamine: training self-modifying neural networks with differentiable neuromodulated plasticity. Preprint at arXiv. https://doi.org/10.48550/arXiv.2002.10585.

9. Vecoven, N., Ernst, D., Wehenkel, A., and Drion, G. (2020). Introducing neuromodulation in deep neural networks to learn adaptive behaviours. PLoS One *15*, e0227922. https://doi.org/10.1371/journal.pone.0227922.

10. Avery, M.C., and Krichmar, J.L. (2017). Neuromodulatory systems and their interactions: a review of models, theories, and experiments. Front. Neural Circ. *11*, 108.

11. Krichmar, J.L. (2013). A neurorobotic platform to test the influence of neuromodulatory signaling on anxious and curious behavior. Front. Neurorob. *7*, 1. https://doi.org/10.3389/fnbot.2013.00001.

12. Krichmar, J.L. (2012). A biologically inspired action selection algorithm based on principles of neuromodulation. In The 2012 International Joint Conference on Neural Networks (IJCNN)), pp. 1–8. https://doi.org/10.1109/IJCNN.2012.6252633.

13. Redgrave, P., Prescott, T.J., and Gurney, K. (1999). Is the short-latency dopamine response too short to signal reward error? Trends Neurosci. *22*, 146–151. https://doi.org/10.1016/s0166-2236(98)01373-3.

14. Tsuda, B., Pate, S.C., Tye, K.M., Siegelmann, H.T., and Sejnowski, T.J. (2021). Neuromodulators enable overlapping synaptic memory regimes and nonlinear transition dynamics in recurrent neural networks. Preprint at bioRxiv. https://doi.org/10.1101/2021.05.31.446462.

15. Sporns, O., and Alexander, W.H. (2002). Neuromodulation and plasticity in an autonomous robot. Neural Network. *15*, 761–774. https://doi.org/10.1016/s0893-6080(02)00062-x.

16. Xing, J., Zou, X., and Krichmar, J.L. (2020). Neuromodulated patience for robot and self-driving vehicle navigation. In 2020 International Joint Conference on Neural Networks (IJCNN)), pp. 1–8. https://doi.org/10.1109/IJCNN48605.2020.9206642.

17. Fiore, V.G., Sperati, V., Mannella, F., Mirolli, M., Gurney, K., Friston, K., Dolan, R.J., and Baldassarre, G. (2014). Keep focussing: striatal dopamine multiple functions resolved in a single mechanism tested in a simulated humanoid robot. Front. Psychol. *5*, 124. https://doi.org/10.3389/fpsyg.2014.00124.

18. Kietzmann, T.C., McClure, P., and Kriegeskorte, N. (2019). Deep Neural Networks in Computational Neuroscience (Oxford Research Encyclopedia of Neuroscience). https://doi.org/10.1093/acrefore/9780190264086.013.46.

19. Banino, A., Barry, C., Uria, B., Blundell, C., Lillicrap, T., Mirowski, P., Pritzel, A., Chadwick, M.J., Degris, T., Modayil, J., et al. (2018). Vector-based navigation using grid-like representations in artificial agents. Nature *557*, 429–433. https://doi.org/10.1038/s41586-018-0102-6.

20. LeCun, Y.A., Bottou, L., Orr, G.B., and Müller, K.-R. (2012). Efficient BackProp. In Neural Networks: Tricks of the Trade: Second Edition Lecture Notes in Computer Science, G. Montavon, G.B. Orr, and K.-R. Müller, eds. (Springer), pp. 9–48. https://doi.org/10.1007/978-3-642-35289-8_3.

21. Krupic, J., Bauza, M., Burton, S., Barry, C., and O'Keefe, J. (2015). Grid cell symmetry is shaped by environmental geometry. Nature *518*, 232–235. https://doi.org/10.1038/nature14153.

22. Songlin, L., Yangdong, D., and Zhihua, W. (2020). Grid cells are ubiquitous in neural

networks. Preprint at arXiv. https://doi.org/10.48550/arXiv.2003.03482.

23. Sorscher, B., Mel, G.C., Ocko, S.A., Giocomo, L., and Ganguli, S. (2020). A unified theory for the computational and mechanistic origins of grid cells. Preprint at bioRxiv. https://doi.org/10.1101/2020.12.29.424583.

24. Long, X., and Zhang, S.-J. (2021). A novel somatosensory spatial navigation system outside the hippocampal formation. Cell Res. *31*, 649–663. https://doi.org/10.1038/s41422-020-00448-8.

25. Gerlei, K., Passlack, J., Hawes, I., Vandrey, B., Stevens, H., Papastathopoulos, I., and Nolan, M.F. (2020). Grid cells are modulated by local head direction. Nat. Commun. *11*, 4228. https://doi.org/10.1038/s41467-020-17500-1.

26. Pilly, P.K., and Grossberg, S. (2013). Spiking neurons in a hierarchical self-organizing map model can learn to develop spatial and temporal properties of entorhinal grid cells and hippocampal place cells. PLoS One *8*, e60599. https://doi.org/10.1371/journal.pone.0060599.

27. Krupic, J., Burgess, N., and O'Keefe, J. (2012). Neural representations of location composed of spatially periodic bands. Science *337*, 853–857. https://doi.org/10.1126/science.1222403.

28. Mhatre, H., Gorchetchnikov, A., and Grossberg, S. (2012). Grid cell hexagonal patterns formed by fast self-organized learning within entorhinal cortex. Hippocampus *22*, 320–334. https://doi.org/10.1002/hipo.20901.

29. Burgess, N., Barry, C., and O'Keefe, J. (2007). An oscillatory interference model of grid cell firing. Hippocampus *17*, 801–812. https://doi.org/10.1002/hipo.20327.

30. Hardcastle, K., Maheswaranathan, N., Ganguli, S., and Giocomo, L.M. (2017). A multiplexed, heterogeneous, and adaptive code for navigation in medial entorhinal cortex. Neuron *94*, 375–387.e7. https://doi.org/10.1016/j.neuron.2017.03.025.

31. Grieves, R.M., and Jeffery, K.J. (2017). The representation of space in the brain. Behav. Process. *135*, 113–131. https://doi.org/10.1016/j.beproc.2016.12.012.

32. Bourboulou, R., Marti, G., Michon, F.-X., El Feghaly, E., Nouguier, M., Robbe, D., Koenig, J., and Epsztein, J. (2019). Dynamic control of hippocampal spatial coding resolution by local visual cues. Elife *8*, e44487. https://doi.org/10.7554/eLife.44487.

33. Chen, X., McNamara, T.P., Kelly, J.W., and Wolbers, T. (2017). Cue combination in human spatial navigation. Cognit. Psychol. *95*, 105–144. https://doi.org/10.1016/j.cogpsych.2017.04.003.

34. Kinkhabwala, A.A., Gu, Y., Aronov, D., and Tank, D.W. (2020). Visual cue-related activity of cells in the medial entorhinal cortex during navigation in virtual reality. Elife *9*, e43140. https://doi.org/10.7554/eLife.43140.

35. Nardini, M., Jones, P., Bedford, R., and Braddick, O. (2008). Development of cue integration in human navigation. Curr. Biol. *18*, 689–693. https://doi.org/10.1016/j.cub.2008.04.021.

36. Klatzky, R.L., Marston, J.R., Giudice, N.A., Golledge, R.G., and Loomis, J.M. (2006). Cognitive load of navigating without vision when guided by virtual sound versus spatial language. J. Exp. Psychol. Appl. *12*, 223–232. https://doi.org/10.1037/1076-898X.12.4.223.

37. Loomis, J.M., Klatzky, R.L., McHugh, B., and Giudice, N.A. (2012). Spatial working memory for locations specified by vision and audition: testing the amodality hypothesis. Atten. Percept. Psychophys. *74*, 1260–1267. https://doi.org/10.3758/s13414-012-0311-2.

38. Nardi, D., Carpenter, S.E., Johnson, S.R., Gilliland, G.A., Melo, V.L., Pugliese, R., Coppola, V.J., and Kelly, D.M. (2022). Spatial reorientation with a geometric array of auditory cues. Q. J. Exp. Psychol. *75*, 362–373. https://doi.org/10.1177/1747021820913295.

39. Nardi, D., Anzures, B.J., Clark, J.M., and Griffith, B.V. (2019). Spatial reorientation with non-visual cues: failure to spontaneously use auditory information. Q. J. Exp. Psychol. *72*, 1141–1154. https://doi.org/10.1177/1747021818780715.

40. Viaud-Delmon, I., and Warusfel, O. (2014). From ear to body: the auditory-motor loop in spatial cognition. Front. Neurosci. *8*, 283. https://doi.org/10.3389/fnins.2014.00283.

41. Watanabe, S., and Yoshida, M. (2007). Auditory cued spatial learning in mice. Physiol. Behav. *92*, 906–910. https://doi.org/10.1016/j.physbeh.2007.06.019.

42. Nardi, D., Singer, K.J., Price, K.M., Carpenter, S.E., Bryant, J.A., Hatheway, M.A., Johnson, J.N., Pairitz, A.K., Young, K.L., and Newcombe, N.S. (2021). Navigating without vision: spontaneous use of terrain slant in outdoor place learning. Spatial Cognit. Comput. *21*, 1–21. https://doi.org/10.1080/13875868.2021.1916504.

43. Loomis, J.M., Klatzky, R.L., Avraamides, M., Lippa, Y., and Golledge, R.G. (2007). Functional equivalence of spatial images produced by perception and spatial language. In Spatial Processing in Navigation, Imagery and Perception, F. Mast and L. Jäncke, eds. (Springer US)), pp. 29–48. https://doi.org/10.1007/978-0-387-71978-8_3.

44. Langston, R.F., Ainge, J.A., Couey, J.J., Canto, C.B., Bjerknes, T.L., Witter, M.P., Moser, E.I., and Moser, M.-B. (2010). Development of the spatial representation system in the rat. Science *328*, 1576–1580. https://doi.org/10.1126/science.1188210.

45. Wills, T.J., Cacucci, F., Burgess, N., and O'Keefe, J. (2010). Development of the hippocampal cognitive map in preweanling rats. Science *328*, 1573–1576. https://doi.org/10.1126/science.1188224.

46. Koenig, J., Linder, A.N., Leutgeb, J.K., and Leutgeb, S. (2011). The spatial periodicity of grid cells is not sustained during reduced theta oscillations. Science *332*, 592–595. https://doi.org/10.1126/science.1201685.

47. Brandon, M.P., Koenig, J., and Leutgeb, S. (2014). Parallel and convergent processing in grid cell, head-direction cell, boundary cell, and place cell networks. Wiley Interdiscip. Rev. Cogn. Sci. *5*, 207–219. https://doi.org/10.1002/wcs.1272.

48. Shine, J.M., Müller, E.J., Munn, B., Cabral, J., Moran, R.J., and Breakspear, M. (2021). Computational models link cellular mechanisms of neuromodulation to large-scale neural dynamics. Nat. Neurosci. *24*, 765–776. https://doi.org/10.1038/s41593-021-00824-6.

49. Van Rossum, G., and Drake, F.L. (2009). Python 3 Reference Manual (CreateSpace).

50. Hunter, J.D. (2007). Matplotlib: a 2D graphics environment. Comput. Sci. Eng. *9*, 90–95. https://doi.org/10.1109/MCSE.2007.55.

51. Waskom, M. (2021). seaborn: statistical data visualization. J. Open Source Softw. *6*, 3021. https://doi.org/10.21105/joss.03021.

52. Bradski, G. (2000). The OpenCV library. Dr. Dobb's J. Softw. Tools Prof. Program. *25*.

53. Raudies, F., and Hasselmo, M.E. (2012). Modeling boundary vector cell firing given optic flow as a cue. PLoS Comput. Biol. *8*, e1002553. https://doi.org/10.1371/journal.pcbi.1002553.

54. Etienne, A.S., and Jeffery, K.J. (2004). Path integration in mammals. Hippocampus *14*, 180–192. https://doi.org/10.1002/hipo.10173.

55. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). {TensorFlow}: A System for {Large-Scale} Machine Learning, pp. 265–283.

56. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). PyTorch: an imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems (Curran Associates, Inc.).

57. Miconi, T. (2016). Backpropagation of Hebbian plasticity for continual learning. In NIPS Workshop on Continual Learning.

58. Miconi, T., Clune, J., and Stanley, K.O. (2018). Differentiable plasticity: training plastic neural networks with backpropagation. Preprint at arXiv. https://doi.org/10.48550/arXiv.1804.02464.

59. Gal, Y., and Ghahramani, Z. (2016). A theoretically grounded application of dropout in recurrent neural networks. In Advances in Neural Information Processing Systems (Curran Associates, Inc.).

60. Baldi, P., and Sadowski, P.J. (2013). Understanding dropout. In Advances in Neural Information Processing Systems (Curran Associates, Inc.).

61. Gal, Y., and Ghahramani, Z. (2016). Dropout as a Bayesian approximation: representing model uncertainty in deep learning. In Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 ICML'16 (JMLR.org), pp. 1050–1059.

62. Kiryk, A., Janusz, A., Zglinicki, B., Turkes, E., Knapska, E., Konopka, W., Lipp, H.-P., and Kaczmarek, L. (2020). IntelliCage as a tool for measuring mouse behavior - 20 years perspective. Behav. Brain Res. *388*, 112620. https://doi.org/10.1016/j.bbr.2020.112620.

63. Cieślak, P.E., Llamosas, N., Kos, T., Ugedo, L., Jastrzębska, K., Torrecilla, M., and Rodriguez Parkitna, J. (2017). The role of NMDA receptor-dependent activity of noradrenergic neurons in attention, impulsivity and exploratory behaviors. Gene Brain Behav. *16*, 812–822. https://doi.org/10.1111/gbb.12383.

64. Takeuchi, H., Iba, M., Inoue, H., Higuchi, M., Takao, K., Tsukita, K., Karatsu, Y., Iwamoto, Y., Miyakawa, T., Suhara, T., et al. (2011). P301S mutant human tau transgenic mice manifest early symptoms of human tauopathies with dementia and altered sensorimotor gating. PLoS One *6*, e21050. https://doi.org/10.1371/journal.pone.0021050.

65. Reynolds, M., Barth-Maron, G., Besse, F., de Las Casas, D., Fidjeland, A., Green, T., Puigdomènech, A., Racanière, S., Rae, J., and Viola, F. (2017). Open sourcing Sonnet-a new library for constructing neural networks. DeepMind. https://deepmind.com/blog/open-sourcing-sonnet/.

66. Bicanski, A., and Burgess, N. (2020). Neuronal vector coding in spatial cognition. Nat. Rev. Neurosci. *21*, 453–470. https://doi.org/10.1038/s41583-020-0336-9.

67. Madl, T., Chen, K., Montaldi, D., and Trappl, R. (2015). Computational cognitive models of spatial memory in navigation space: a review. Neural Network. *65*, 18–43. https://doi.org/10.1016/j.neunet.2015.01.002.

68. Giocomo, L.M. (2016). Environmental boundaries as a mechanism for correcting and anchoring spatial maps. J. Physiol. *594*, 6501–6511. https://doi.org/10.1113/JP270624.

69. Sanders, H., Rennó-Costa, C., Idiart, M., and Lisman, J. (2015). Grid cells and place cells: an integrated view of their navigational and memory function. Trends Neurosci. *38*, 763–775. https://doi.org/10.1016/j.tins.2015.10.004.

70. Burgess, N. (2008). Spatial cognition and the brain. Ann. N. Y. Acad. Sci. *1124*, 77–97. https://doi.org/10.1196/annals.1440.002.

71. Moser, M.-B., Rowland, D.C., and Moser, E.I. (2015). Place cells, grid cells, and memory. Cold Spring Harbor Perspect. Biol. *7*, a021808. https://doi.org/10.1101/cshperspect.a021808.

72. Solstad, T., Moser, E.I., and Einevoll, G.T. (2006). From grid cells to place cells: a mathematical model. Hippocampus *16*, 1026–1031. https://doi.org/10.1002/hipo.20244.

73. Geva-Sagiv, M., Las, L., Yovel, Y., and Ulanovsky, N. (2015). Spatial cognition in bats and rats: from sensory acquisition to multiscale maps and navigation. Nat. Rev. Neurosci. *16*, 94–108. https://doi.org/10.1038/nrn3888.

74. Lever, C., Burton, S., Jeewajee, A., O'Keefe, J., and Burgess, N. (2009). Boundary vector cells in the subiculum of the hippocampal formation. J. Neurosci. *29*, 9771–9777. https://doi.org/10.1523/JNEUROSCI.1319-09.2009.

75. Solstad, T., Boccara, C.N., Kropff, E., Moser, M.-B., and Moser, E.I. (2008). Representation of geometric borders in the entorhinal cortex. Science *322*, 1865–1868. https://doi.org/10.1126/science.1166466.

76. Giocomo, L.M., Stensola, T., Bonnevie, T., Van Cauter, T., Moser, M.-B., and Moser, E.I. (2014). Topography of head direction cells in medial entorhinal cortex. Curr. Biol. *24*, 252–262. https://doi.org/10.1016/j.cub.2013.12.002.

77. Taube, J.S. (1995). Head direction cells recorded in the anterior thalamic nuclei of freely moving rats. J. Neurosci. *15*, 70–86. https://doi.org/10.1523/JNEUROSCI.15-01-00070.1995.

78. Wills, T.J., Barry, C., and Cacucci, F. (2012). The abrupt development of adult-like grid cell firing in the medial entorhinal cortex. Front. Neural Circ. *6*, 21.

79. Burgess, N. (2008). Grid cells and theta as oscillatory interference: theory and predictions. Hippocampus *18*, 1157–1174. https://doi.org/10.1002/hipo.20518.

80. Sargolini, F., Fyhn, M., Hafting, T., McNaughton, B.L., Witter, M.P., Moser, M.-B., and Moser, E.I. (2006). Conjunctive representation of position, direction, and velocity in entorhinal cortex. Science *312*, 758–762. https://doi.org/10.1126/science.1125572.

81. Kropff, E., and Treves, A. (2008). The emergence of grid cells: intelligent design or just adaptation? Hippocampus *18*, 1256–1269. https://doi.org/10.1002/hipo.20520.

82. Weber, S.N., and Sprekeler, H. (2019). A local measure of symmetry and orientation for individual spikes of grid cells. PLoS Comput. Biol. *15*, e1006804. https://doi.org/10.1371/journal.pcbi.1006804.

83. van der Walt, S., Schönberger, J.L., Nunez-Iglesias, J., Boulogne, F., Warner, J.D., Yager, N., Gouillart, E., and Yu, T.; scikit-image contributors (2014). scikit-image: image processing in Python. PeerJ *2*, e453. https://doi.org/10.7717/peerj.453.

84. Ghasemi, A., and Zahediasl, S. (2012). Normality tests for statistical analysis: a guide for non-statisticians. Int. J. Endocrinol. Metabol. *10*, 486–489. https://doi.org/10.5812/ijem.3505.

85. Thode, H.C. (2002). Testing for Normality (CRC Press). https://doi.org/10.1201/9780203910894.

86. Yap, B.W., and Sim, C.H. (2011). Comparisons of various types of normality tests. J. Stat. Comput. Simulat. *81*, 2141–2155. https://doi.org/10.1080/00949655.2010.520163.

87. Dinno, A. (2015). Nonparametric pairwise multiple comparisons in independent groups using Dunn's test. STATA J. *15*, 292–300. https://doi.org/10.1177/1536867X1501500117.

STAR★METHODS

KEY RESOURCES TABLE

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Deposited data		
GitHub repositories with all code used in the present study (https://github.com/CNAILab)	This study	https://github.com/CNAILab
Software and algorithms		
GraphPad Prism	GraphPad Software, San Diego, California USA	https://www.graphpad.com
Python	Van Rossum and Drake, 2009[49]	https://www.python.org
Matplotlib	Hunter, 2007[50]	https://matplotlib.org
Seaborn	Waskom, 2021[51]	https://seaborn.pydata.org
OpenCV	Bradski, 2000[52]	https://opencv.org
Diagrams.net	Diagrams.net	https://www.diagrams.net

RESOURCE AVAILABILITY

Lead contact

Further information and requests for materials may be directed to and will be fulfilled by the lead contact, Dr. Jie Mei (majorjiemei@gmail.com).

Materials availability

This study did not generate new unique reagents.

Data and code availability

- All data reported in this paper will be shared by the lead contact upon request.
- All original code has been deposited on GitHub and is publicly available as of the date of publication. DOIs are listed in the key resources table.
- Any additional information required to reanalyze the data reported in this paper is available from the lead contact upon request.

METHOD DETAILS

Dataset

The dataset used in this study was introduced by[19] and consisted of simulated rat ambulation trajectories generated for a total of 100 timesteps, in a square-like open environment with dimensions of 2.2 × 2.2m. To allow direct comparison between our results and,[19] the generated trajectories were obtained from a rat-like motion simulator and uniformly covered the environment.[53] All DNNs implemented in the present study were trained using this dataset to perform path integration through linear and angular velocity inputs across the current and past timesteps.

A total of 100 generated ambulation trajectories were made available in the format of TFRecord and can be obtained from https://console.cloud.google.com/storage/browser/grid-cells-datasets.

Neural network architecture and artificial neuromodulation

To investigate the effects of neuromodulation in a biologically plausible setting, we chose to add neuromodulatory components to a DNN with the ability to path integrate (i.e., determining and updating one's location through cues generated by movements and monitoring one's trajectory in relation to a starting location, without using landmarks).[54] The DNNs implemented and tested were inspired by the

base model used in an open field navigation task in.[19] This model is a DNN with biologically plausible unit activities, demonstrating unit activity patterns which resemble that of biological neurons in the hippocampus and medial entorhinal cortex, e.g., grid cells and place cells.

We aim to explore how multi-level modulation of DNN properties, including hyperparameter auto-tuning and adaptive, plastic modulation of weights, may affect (1) number of different types of neurons (e.g., place cell-like units, grid cell-like units, and border cell-like units), and (2) unit activity patterns. In attempt to (1) examine whether experimental results could be replicated using another platform, (2) enable cross-platform comparisons of the effects of neuromodulatory components on learning and unit activities, and (3) provide a base code for future model development that facilitates implementation and testing of differing neuromodulatory components, we performed all experiments using both TensorFlow[55] and PyTorch.[56]

PyTorch is a more flexible framework which allows additional modifications of deep learning models and their training processes, benefiting the present study as well as future research. The PyTorch implementation enabled analysis of reproducibility of results where platform-dependent implementation and optimizations were different.

The base model

The base model consists of a single recurrent layer of long short-term memory (LSTM) units, followed by a linear layer of 256 units and an output layer (Figure 12). The recurrent layer extends across 100 timesteps, receiving a vector $[\vartheta_t, \sin(\varphi t), \cos(\varphi t)]$ that encodes velocity of the agent, with ϑ_t and φt representing the linear and angular velocities at time t. The output of the recurrent layer is passed to the fully-connected bottleneck layer which is regularized by a dropout probability of 0.5. Network outputs are generated by projecting the bottleneck activations to a layer representing place and head direction cell ensembles.

For a given spatial location in the open environment and facing angle, the output layer presents corresponding activations of place and head direction units, simulated by the posterior probability of each component of a mixture of two-dimensional isotropic Gaussians and von Mises distributions, respectively.[19] The base model is trained to predict the activations of the place and head direction cell ensembles by optimizing a custom loss $LOSS = LOSS_{PC} + LOSS_{HD}$, where $LOSS_{PC}$ represents the cross entropy between the predicted and actual place cell activities, and $LOSS_{HD}$ represents the cross entropy between the predicted and actual head direction cell activities.

Multi-scale neuromodulation

Neuromodulation-inspired components were realized at three levels, i.e., single unit level, layer level and network level, through plastic components, adaptive dropout probability, and learning rate scheduling, respectively (Figure 12).

Single unit level: Plastic components

A neural network's weights can be coupled with a plastic component $\alpha_{i,j}Hebb_{i,j}(t)$.[57,58] Moreover, parameters associated with the plasticity component can be learned through gradient descent:

$$x_j(t) = \sigma\left\{\sum_{i \in inputstoj} \left(w_{i,j} + \alpha_{i,j}\mathrm{Hebb}_{i,j}(t)\right)x_i(t-1)\right\} \quad \text{(Equation 1)}$$

$$\mathrm{Hebb}_{i,j}(t+1) = \mathrm{Clip}\left(\mathrm{Hebb}_{i,j}(t) + M(t)x_i(t-1)x_j(t)\right) \quad \text{(Equation 2)}$$

Where $x_i(t)$ represents output of neuron i at time l, σ is a non-linear transformation, $\omega_{i,j}$ is the weight between neurons i and j, and $\alpha_{i,j}$ is the plasticity coefficient. $\mathrm{Hebb}_{i,j}$ represents the plastic component, and $M(t)$ is the time-varying neuromodulatory signal computed by the DNN that modulates plasticity of each connection.[8]

Layer level: Modulation of dropout probability.

At the layer level, instead of using a fixed dropout probability of 0.5 throughout training,[19] we modulated dropout probability based on $\Delta_{l_1 - l_2}Loss$, i.e., the change in model loss between epochs l_1 and l_2. This allows us to use the model loss as a feedback signal, and to take into account the model performance, when

adjusting the number of neurons to deactivate during learning. Modulation of dropout probability was applied every 10 epochs, thus, at epoch $l = l_n + 10$:

$$max\_step\_size = min\left(\left|upperbound - keep\_prob_{l=0}\right|, \left|lowerbound - keep\_prob_{l=0}\right|\right) / \left(total\_epochs / frequency\_of\_update\right) \quad \text{(Equation 3)}$$

$$scaling\_factor\_list = \left[1, 1\cdot10^{-1}, 1\cdot10^{-2}, 1\cdot10^{-3}, 1\cdot10^{-4}, 1\cdot10^{-5}, 1\cdot10^{-6}\right] \quad \text{(Equation 4)}$$

$$scaling_{factor} = argmin\left(\left|x - \left|max\_step\_size / \left(Loss_{l=l_n} - Loss_{l=l_{n-10}}\right)\right|\right) for\ x\ scaling\_factor\_list \quad \text{(Equation 5)}$$

$$\Delta keep\_prob = \left(Loss_{l=l_n} - Loss_{l=l_{n-10}}\right)\cdot scaling\_factor \quad \text{(Equation 6)}$$

$$keep\_prob_{l=l_n+10} = keep_{prob_{l=l_n}} - \Delta keep_{prob} \quad \text{(Equation 7)}$$

$$dropout_{l=l_n+10} = 1 - keep\_prob_{l=l_n+10} \quad \text{(Equation 8)}$$

where the upper bound and lower bound of dropout probability was set to 0.8 and 0.2, respectively.[59–61]

Network level: Learning rate decay

At the network level, we applied a decay scheme to learning rate η. From a behavioral perspective, decrease in learning rate is partly comparable to animal ambulation and learning patterns in novel environments: When an animal is first introduced to a new environment, it may exhibit intense exploratory activities, followed by decreased ambulation and exploration when the environment becomes more familiar.[62–64] According to experimental studies, highest amount of daily and hourly exploration and ambulation may be recorded in the animals' first day in a novel environment, before they perform fine-tuned exploration for the purpose of gaining access to food and water.[62] Under this inspiration, at epoch 0, learning rate was initialized to:

$$\eta_{l=0} = n \text{ for } n\ \varepsilon\ \left\{1\cdot10^{-4}, 1\cdot10^{-5}\right\} \quad \text{(Equation 9)}$$

And decayed every 10 epochs with evenly spaced decrements on a logarithmic scale of base e, ending at:

$$\eta_{l=1000} = \eta_{l=0}\cdot10^{-1} \quad \text{(Equation 10)}$$

Model variants, implementation, and training

Model variants

In the present study, the base model and five variants were implemented using Tensorflow and PyTorch:

1. **Base_m1**: The base model with a learning rate of 1e-5. All hyperparameters, model training protocols and optimization procedures were kept the same as in.[19]
2. **Base_m2**: Base_m1 with a learning rate of 1e-4.
3. **Plast_m**: Base_m1 with plastic components incorporated (Equations 1 and 2).
4. **PlastDrop_m**: Plast_m with dropout probability modulation (Equations 3–8).
5. **PlastDropLr_m1**: PlastDrop_m with learning rate scheduling. Learning rate was initialized at 1e-5 and gradually decayed to 1e-6 on a log scale (Equations 9 and 10).
6. **PlastDropLr_m2**: Base_m2 with plasticity, dropout probability modulation and learning rate scheduling. Learning rate was initialized at 1e-4 and decayed to 1e-5 on a log scale (Equations 9 and 10).

Although it is commonly observed that learning rate scheduling improves model learning and convergence, we did not implement it in all base models for the following reasons: First, as a surrogate for network-level neuromodulation effects, adding it as an individual component allow direct comparison with other model variants, as well as the base model in.[19] Moreover, it was not trivial that improvement

in performance (e.g., loss) and convergence would lead to a considerably lower error of ambulation. As a result, we decided to present the addition of learning rate scheduling as a separate configuration.

We have observed differences in the underlying implementation of the Root Mean Square Propagation algorithm (RMSprop) optimizer in TensorFlow and PyTorch. Moreover, previous work suggested that lower learning rates and stricter gradient clipping values may lead to hexagonal unit activities.[22] To test whether lower learning rate, higher gradient clipping values, and/or altered optimizer implementations affect unit activity patterns, two more variants were implemented using PyTorch:

1. **Base_SGD**: Base_m1 with a Stochastic Gradient Descent (SGD) optimizer instead of RMSProp. Experiments were conducted to examine whether different optimizers would lead to diverging behavior in the development of learning and unit activity patterns. The learning rate was set to 1e-3 since lower learning rates were demonstrating a very slow decrease in loss during the first 100 epochs, while the momentum was kept at 0.9 to keep hyperparameters consistent with other model variants.
2. **Base_loLR, Base_SGC, Base_loLR_SGC**: Base_m1 with a learning rate of 1e-6, stricter gradient clipping (1e-6), or both. Three sets of experiments, each of 2 trials, were performed to analyze model performance and of unit activities when (a) the learning rate was lowered to 1e-6, or (b) when gradient clipping was stricter by being set to 1e-6, or (c) both a learning rate of 1e-6 and a gradient clipping of 1e-6 were applied at same time.

Model implementation

All models were implemented using TensorFlow and PyTorch and are made available (https://github.com/CNAILab). The TensorFlow implementation of the base model was partly based on,[19] and all model components implemented on the Sonnet platform[65] were replaced with a native TensorFlow version. The PyTorch implementation of the base model is partly inspired by a previous model created by Lucas Pompe (https://github.com/LPompe/gridtorch).

Model training

All models and variants in the present study were trained to minimize a custom loss defined as $Loss = Loss_{PC} + Loss_{HD}$, where $Loss_{PC}$ represents the cross entropy between the predicted and actual place unit activities and $Loss_{HD}$ represents the cross entropy between the predicted and actual head direction unit activities. To make our study comparable to,[19] we used the same protocol for model training and evaluation. Models and variants were trained with a batch size of 10 for a total of 1,000 epochs using an RMSProp optimizer (momentum = 0.9). For each model, training and test were performed 5 times, and all results reported in the present study were the average of these 5 trials. A train-test split of 90:10 was used, with ambulatory trajectories number 0 to number 89 as the training set and ambulatory trajectories number 90 to number 99 as the test dataset. No hyperparameter tuning was performed using a validation dataset, and the test set was kept separate and unused during model training.

There were 1,000 training steps per epoch, thus, not all training samples were used in one epoch. Instead, approximately a total of $training\_steps \times batch\_size$ training samples were used. Similar to the training process, model evaluation was performed through 400 evaluation steps. In each of these steps, model output was evaluated using the test set. The batch size for model evaluation was 10, same as for model training. To allow for comparison, all model hyperparameters used were the same as the implementation in[19] unless otherwise mentioned.

Classification of neurons

We quantified the development of fully-connected layer unit activity maps by (1) identifying the different types of cells in this layer, and (2) evaluating grid cell-like unit activities. Unit activity maps are ratemaps that illustrate single unit activity over different spatial locations, obtained from the fully-connected layer activations. Classification of the type of fully-connected layer units was based on the resemblance of their activity patterns to that of biological neurons involved in spatial learning and navigation, including place cells, grid cells,[66,67] head direction cells[24] and border cells:[68,69]

1. **Place cells** are a class of hippocampal neurons that activate strongly when an animal enters a specific location in an environment (the place field; Figure 13). Their activities are largely independent from the animal's orientation in the environment.[70]
2. Investigation of cells in a brain area upstream of the hippocampus, that is, the medial entorhinal cortex (MEC), led to the discovery of **grid cells.** Studies suggested that grid cells in the MEC show increased activities at multiple regular locations in an environment, following a triangular/hexagonal grid pattern (Figure 13). It has been hypothesized that a weighted combination of grid cell inputs may lead to the emergence of place cell firing fields.[67,71,72]
3. **Border cells** only show a higher firing rate in proximity to one or several salient boundaries of an environment (Figure 13) and may encode the environmental geometry.[68,73] Border cells have been found in regions within the hippocampal formation, including the subiculum and entorhinal cortex.[74,75]
4. **Head direction cells** have been identified in multiple brain areas, including the entorhinal cortex and thalamus.[76,77] Head direction cells are not responsible for location-specific environmental cues, rather, they show increased firing whenever an animal's head is pointing at a certain direction (i.e., the cell's preferred direction). Therefore, head direction cells do not show location-based increase in activities (Figure 13).
5. **Conjunctive cells** are cells that respond selectively to two or more types of information, e.g., position, direction, and/or speed. As a result, their ratemaps may look like a combination of grid, head direction, or place cells (Figure 13).[24,78] Moreover, a recent study suggested that the majority of neurons in the MEC are of conjunctive properties, rather than selecting and responding to only one type of input.[30] The same study reports that the response of many cells in the MEC that encode position or head direction changes with speed, suggesting adaptive change of neural code based on sensory input.
6. It has been suggested that grid cells can be shaped by combining a set of **stripe cells** of various orientations.[26,28] Stripe cells have also been referred to as band cells in oscillatory-interference models.[79] Each cell has a periodic firing pattern resembling stripes or bands (Figure 13) and tracks the distance traveled in the direction of stripes by moving the activity bump around as the agent moves along the stripe direction.[26] Although limited, experimental support has also been provided by studies that located cells with periodic band-like activity patterns in layers III, V, and VI of the dorsal segment of MEC,[80] and in layers II and III of MEC and the adjacent pre- and para-subiculum.[27]

Cell type classification of the TensorFlow model

For classification of cell types, we first obtained unit activity maps within the environment for all fully-connected layer units, then trained a k-nearest neighbors (k-NN) classifier using the concatenation of the following features:

1. Averaged level of activity, for identifying units with lower levels of activity on average (e.g., place cell-like units and border cell-like units)
2. Standard deviation of activity levels, for identifying units with lower variance in activity across different locations (i.e., head direction cell-like units)
3. Percentage of area associated with higher-than-average levels of activity, for identifying units with higher levels of activity over a large portion of the environment (e.g., inverse of place cell-like units and inverse of border cell-like units) and
4. Grid score calculated as in,[19] for discriminating between grid cell-like units and non-grid cell-like units

A visual inspection was performed following classification by k-NN to ensure all fully-connected layer units were correctly classified.

For the analysis of grid cell-like unit activities, we selected these units using a 3-step approach:

- Step 1: We compute the mean and standard deviation of unit activity across the whole environment, and apply a lower bound and an upper bound (Table S1) for both to exclude units with (a) high or low

level of activity across the environment, (b) head direction cell-like activities or/and (c) high levels of activity in one certain area of the environment.[52, 55 and 56] Non-grid cell-like units are also excluded at this step by applying a threshold to only include units with a grid_score >0.[80–82]

- Step 2: For all remaining units, we identify the local maxima of unit activity (i.e., an area associated with a higher level of activity compared to its proximity, Figure 14) using the scikit-image peak_local_max() function[83] and define them and their surrounding areas as firing fields. Then we compute (1) the standard deviation of activity level across all firing fields, as well as (2) the difference between activity levels at the global maxima and at the 3 local maxima with the lowest level of activity and (3) apply a lower bound and an upper bound (Table S1) for both parameters to exclude units with (a) one local maximum with significantly higher level of activity or (b) units with activity patterns resembling stripes.
- Step 3: We further exclude units with stripe-like activity patterns (i.e., stripe cell-like units and conjunctive cell-like units; Figure 14) by setting a threshold for the maximal height/width of firing fields. That is, if the height or width of a firing field exceeds this threshold, the unit was excluded as a non-grid cell-like unit.

Cell type classification of the PyTorch model

The process for classifying grid cell-like units in the PyTorch implementation consists of the following steps:

- Step 1: The standard deviation of unit activity across the open environment is computed for each unit. Head direction cell-like unit activities were excluded based on the small standard deviation in unit activities.
- Step 2: Average size of firing fields was computed for each unit to exclude place cell- and border cell-like units, given their large firing fields.
- Step 3: Viewing firing fields as nodes of a graph, we used a Depth First Search (DFS) algorithm so that an edge between two nodes (i.e., firing fields) was drawn if there was a path (defined as a sequence of adjacent pixels, each with a value above a user-defined quantile of firing activity) between these two nodes. The number of distinct longest paths with a length of more than 2 nodes (i.e., stripe count) was used as a metric to identify then count stripe-like patterns for each unit. Units with greater-than-zero stripe-like patterns were excluded (Figure 14).
- Step 4: Finally, activity patterns of all remaining units were visually inspected to identify grid cell-like units.

Evaluation metrics

Model loss and error of ambulation

We use (1) the custom model loss $Loss = Loss_{PC} + Loss_{HD}$ and (2) error of ambulation ε to assess model learning and performance. Error of ambulation was defined as the Euclidean distance between the predicted ambulatory trace and the actual ambulatory trace, and was measured in meters (m).

Activity patterns of fully-connected layer grid cell-like units

To examine the interplay between neuromodulatory components and grid cell-like cell activity patterns, the following metrics were defined and used:

For each model variant:

1. Number of grid cell-like units in the fully-connected layer, averaged across 5 trials

For each grid cell-like unit:

1. Number of firing fields, calculated using a center-based method or a contour-based method.
   a. For the center-based method, each local maximum was identified using the scikit-image peak_local_max() function. This function dilates the original image, merges neighboring local maxima

closer than the dilation size, and returns coordinates of pixels that have the same value in the original and dilated image.

b. For the contour-based method, we set all pixels of the activation map with a value below 0.65 quantile to 0, and set all other pixels to 255. We then use the OpenCV[52] findContours() function to identify contours and count the number of contours for each unit.

2. Inter-firing field distance. To approximate the distance between neighboring firing fields, firing fields were first identified. Given the small number of firing fields in some units, for each firing field, only the 3 closest firing fields were located and the mean distance to these 3 firing fields were returned as the inter-firing field distance (Figure 14). This approach also represents a common approximation for inter-firing field distance in situations such as hexagonal, square-like, or stripy firing patterns, where there usually are 3 neighboring firing fields.
3. Average size of firing fields. After identifying all contours, their sizes were determined using the OpenCV contourArea() function. For each unit, the average size of firing fields was calculated by averaging across all contour sizes.
4. Maximum size of firing fields, i.e., size of the largest firing field. For each unit, the largest contour size was returned.
5. Average level of activity of firing fields.
6. Maximum level of activity, i.e., activity level of the firing field with the highest activity level.
7. Average width and height of firing fields. These were calculated using the OpenCV boundingRect() function for all contours, and then averaged per unit.
8. Maximum width/height of firing fields, i.e., the largest height/width identified for each unit.

When analyzing the effects of neuromodulatory components on unit activity patterns, 5 scenarios were defined for the interpretation of experimental results:

1. **Base_m1** vs **Base_m2**, for examining the effect of learning rate;
2. **Base_m1** vs **Plast_m**, for examining the effects of plastic component;
3. **Plast_m** vs **PlastDrop_m**, for studying the effects of dropout probability modulation;
4. **PlastDrop_m** vs **PlastDropLr_m1**, for studying the effects of learning rate decay;
5. **Base_m1** – **PlastDropLr_m1** vs **Base_m2** – **PlastDropLr_m2**, for understanding learning rate-dependent effects of neuromodulatory components.

QUANTIFICATION AND STATISTICAL ANALYSIS

The Shapiro-Wilk test was used to examine the normality of data.[84–86] To compare between two model variants, T test was used when data samples followed a normal distribution, and Mann Whitney U test was used when data samples deviated from a normal distribution. To compare across three or more model variants, for normally distributed data of equal sample sizes, one-way ANOVA with Tukey's procedure to correct for multiple comparisons was used. For normally distributed data of unequal sample sizes, Brown-Forsythe and Welch ANOVA tests were conducted, followed by the Dunnett's T3 test. To compare non-normal data collected from three or more model variants, Kruskal-Wallis test with Dunn's post-hoc test for multiple comparisons was used.[87] Data are shown as mean (SD). p-values <0.05 were considered statistically significant.

Statistical analyses were performed using GraphPad Prism (version 9.4.1, GraphPad Software, San Diego, California USA) and data were visualized using GraphPad Prism (version 9.4.1, GraphPad Software, San Diego, California USA), Python (version 3.8.3),[49] Matplotlib (version 3.5.0),[50] and Seaborn (version 0.11.2).[51] Diagrams were created using a free flowchart maker and online diagram software (Diagrams.net).