Learning in Biologically Plausible Neural Networks

Draco Xu
Department of Mathematics, Computer Science
University of Rochester

Abstract

Biologically Plausible Neural Networks (BPNNs) have attracted significant attention in recent years, mainly due to their ability to bridge the gap between artificial neural networks and the biological processes that underlie them. In this paper, I present an exhaustive literature review of learning algorithms for three specific types of BPNNs: 1) Constrained Deep Neural Networks (CDNNs), 2) Spiking Neural Networks (SNNs), and 3) Reduced Spiking Neural Networks (Rate Models, RSNNs). Through case studies, I demonstrate the implementation and training of CDNNs and introduce a novel learning method for RSNNs, which we also implement. Furthermore, I propose an innovative approach to compare Spiking Neural Networks and Constrained Deep Neural Networks. As future work, I plan to expand my investigation of learning algorithms for SNNs, an endeavor that will further enhance our understanding of biologically inspired neural network models.

1 Introduction

The success of ChatGPT has sparked public interest in deep neural networks and fostered positive expectations for the rapid development of super-human artificial intelligence. Some researchers attribute the advancements in large language models to their biologically plausible architectures, which are based on neural network structures. However, others challenge this claim, arguing that engineering perspectives, rather than biologically plausible architectures, have driven this success. They maintain that designing biologically plausible architectures is not essential for achieving general artificial intelligence. Despite the controversy, both sides concur that the earliest artificial neural networks were inspired by findings in neuroscience. Over the past 50 years, the field of neural networks has been extensively studied, resulting in two primary categories: deep neural networks (DNNs) and spiking neural networks (SNNs). DNNs underpin the large language models seen today, such as AlphaGo, which defeated the best human players in Go. In contrast, SNNs are considered more biologically plausible and energy-efficient.

Deep neural network success can be attributed to advancements in hardware and learning approaches based on backpropagation. Historically, SNNs have been challenging to train because backpropagation cannot be directly applied due to the lack of a proper gradient at spiking times. However, recent discoveries of effective gradient estimation functions for spiking times have renewed interest in SNN learning within both academia and industry. Companies such as Intel and IBM have invested significantly in the development of algorithms and hardware for neuromorphic computing.

In this study, I will present a thorough review of learning processes in biologically plausible neural networks, with an emphasis on spiking neural networks. Additionally, I will introduce two innovative learning methods for these networks and illustrate their applications in neuroscience and neuromorphic computing.

2 Related Work in Spiking Neural networks

In this section, we review the literature related to learning algorithms for spiking neural networks (SNNs). Spiking Neural Networks (SNNs) are a type of artificial neural networks that are inspired by the behavior of biological neurons. SNNs employ spiking neurons, which communicate through discrete events called spikes or action potentials, to process and transmit information. Due to their event-driven nature, SNNs are capable of processing temporal information and are computationally efficient.

1. **Neuron Model:**
   There are various mathematical models to describe the behavior of spiking neurons. One of the most widely used models is the Leaky Integrate-and-Fire (LIF) neuron model. The LIF neuron model can be described by the following differential equation (Ghosh-Dastidar and Adeli, 2009):

   $$\tau_m \frac{dV_m}{dt} = -(V_m - V_{rest}) + R_m I(t) \tag{1}$$

   where V_m is the membrane potential, V_{rest} is the resting potential, R_m is the membrane resistance, τ_m is the membrane time constant, and $I(t)$ is the input current. When the membrane potential reaches a threshold value V_{th}, the neuron emits a spike and resets its membrane potential to the reset potential V_{reset}.

2. **Synaptic Dynamics (Taherkhani et al., 2020):**
   The synaptic connections between neurons in an SNN determine how the spikes are transmitted and processed. Synaptic dynamics can be modeled using various approaches, such as exponential or alpha synapses. For instance, the dynamics of an exponential synapse can be described by the following equation:

   $$\tau_s \frac{dx}{dt} = -x + w \sum_f \delta(t - t^f) \tag{2}$$

   where x is the synaptic variable, τ_s is the synaptic time constant, w is the synaptic weight, and t^f represents the spike times of the presynaptic neuron.

3. **Spike Propagation:**
   In an SNN, the propagation of spikes through the network is determined by the connectivity and the synaptic weights. The output spikes of a neuron are transmitted to its postsynaptic neurons through the synapses. The postsynaptic neurons then update their membrane potentials based on the input spikes and the synaptic weights, as described by the neuron model.

4. **Learning and Plasticity (Caporale and Dan, 2008):**
   SNNs can learn and adapt their behavior through various learning algorithms, such as Spike-Timing-Dependent Plasticity (STDP), which adjust the synaptic weights based on the timing of pre- and post-synaptic spikes. The general STDP learning rule can be expressed as:

   $$\Delta w_{ij} = \eta(f_+(t_j - t_i) - f_-(t_j - t_i)) \tag{3}$$

   where Δw_{ij} is the change in synaptic weight between neuron i and neuron j, η is the learning rate, t_i and t_j are the spike times of pre- and post-synaptic neurons, and f_+ and f_- are the potentiation and depression functions that depend on the spike timing.

2.1 Supervised Learning in SNNs

Supervised learning in spiking neural networks (SNNs) focuses on adjusting synaptic weights based on available input-output pairs to minimize a predefined loss function. The aim is to enable the network to generalize and produce correct outputs for new, unseen inputs. This section delves deeper into supervised learning algorithms for SNNs and discusses their principles, advantages, and limitations.

SpikeProp SpikeProp, introduced by Bohte et al. (2000), was one of the first supervised learning algorithms for SNNs. It extended the error backpropagation method used in traditional artificial neural networks (ANNs) to single-layer SNNs. The algorithm relied on precise spike timing to compute the error gradient and adjust the synaptic weights. While SpikeProp demonstrated the potential of supervised learning in SNNs, it was limited to single-layer networks and lacked the ability to model complex tasks.

Let $t_j^{(l)}$ be the spike time of the j-th neuron in layer l. We define the error between the desired spike time $t_j^{(L)}$ for the output neuron j and the actual spike time as:

1. Compute the error for each output neuron j by comparing the actual spike time $t_j^{(L)}$ with the desired spike time $t_j^{(L)}$desired:
   $E_j = \frac{1}{2}(t_j^{(L)} - t_j^{(L)_{\text{desired}}})^2$
2. Calculate the total network error by summing the errors for all output neurons:
   $E = \sum_{j=1}^{N_L} E_j$
3. Find the derivative of the error with respect to the spike time for output neuron j:
   $\frac{\partial E_j}{\partial t_j^{(L)}} = t_j^{(L)} - t_j^{(L)_{\text{desired}}}$
4. Compute the derivative of the spike time with respect to the membrane potential for neuron j in layer l:
   $\frac{\partial t_j^{(l)}}{\partial V_j^{(l)}} = \frac{1}{V_j^{(l)'} - V_j^{(l)}}$
5. Calculate the derivative of the membrane potential with respect to the synaptic weight between neuron i in layer $(l-1)$ and neuron j in layer l:
   $\frac{\partial V_j^{(l)}}{\partial w_{ji}^{(l)}} = \sum_{k \in P_j^{(l-1)}} w_{kj}^{(l-1)} \cdot \rho(t_j^{(l)} - t_k^{(l-1)})$
6. Using the chain rule, find the gradient of the error with respect to the synaptic weight between neuron i in layer $(l-1)$ and neuron j in layer l:
   $\frac{\partial E_j}{\partial w_{ji}^{(l)}} = \frac{\partial E_j}{\partial t_j^{(L)}} \cdot \frac{\partial t_j^{(l)}}{\partial V_j^{(l)}} \cdot \frac{\partial V_j^{(l)}}{\partial w_{ji}^{(l)}}$
7. Update the synaptic weights using gradient descent to minimize the error:
   $w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} - \eta \frac{\partial E_j}{\partial w_{ji}^{(l)}}$

ReSuMe The Remote Supervised Method (ReSuMe) proposed by Kasinski and Ponulak (2005) combined spike-timing-dependent plasticity (STDP), a biologically inspired unsupervised learning rule, with gradient descent-based error backpropagation. ReSuMe could be applied to multi-layer SNNs and allowed for more flexibility in terms of spike timing. The algorithm minimized the difference between the desired and actual spike times to achieve learning, but faced challenges related to the non-differentiable nature of spike events and the inherent complexity of the algorithm.

1. Initialize the synaptic weights w_{ij} and eligibility trace e_{ij} for each synapse between presynaptic neuron i and postsynaptic neuron j.
2. For each input spike, update the membrane potential V_j of the postsynaptic neuron j using the following equation:
   $V_j(t) = V_j(t) + w_{ij} \cdot \delta(t - t_i)$
   where t_i is the spike time of the presynaptic neuron i, and δ is the Dirac delta function.
3. When the membrane potential V_j of the postsynaptic neuron j reaches the threshold, generate an output spike and reset the membrane potential.
4. Calculate the spike timing error Δt_{jk} for each output neuron j as the difference between the actual spike time t_j and the desired spike time t_j^{desired}:
   $\Delta t_{jk} = t_j - t_j^{\text{desired}}$
5. Update the eligibility trace e_{ij} for each synapse using the following equation:
   $e_{ij}(t) = \alpha e_{ij}(t) + \delta(t - t_i)$
   where α is the trace decay factor.

6. Update the synaptic weights w_{ij} using the following learning rule:
   $w_{ij} \leftarrow w_{ij} - \eta \cdot \Delta t_{jk} \cdot e_{ij}(t)$
   where η is the learning rate.

ANN-to-SNN Conversion In order to harness the power of deep learning techniques, Diehl et al. (2016) proposed a method to convert pre-trained ANNs to SNNs. This approach involved training a deep ANN using conventional backpropagation, followed by converting the trained network to an equivalent SNN with the same architecture. To achieve this, activation functions were replaced by spike generation mechanisms, and the continuous weights were discretized to match the binary nature of SNNs. While this method enabled the application of deep learning techniques to SNNs, it did not fully exploit the temporal dynamics and unique features of spiking neurons.

1. Train an Artificial Neural Network (ANN) with a deep learning algorithm such as backpropagation. The ANN should have synaptic weights w_{ij}^{ANN}, biases b_j^{ANN}, and activation functions $\phi(\cdot)$ for each neuron.
2. Normalize the synaptic weights in the ANN. For each layer l, compute the maximum absolute weight $W_{\text{max}}^{(l)}$ and divide each weight by it:
   $w_{ij}^{\text{normalized}} = \frac{w_{ij}^{\text{ANN}}}{W_{\text{max}}^{(l)}}$
3. Convert the ANN activation functions to Integrate-and-Fire (IF) neuron models in the Spiking Neural Network (SNN). For each neuron j in the SNN, the membrane potential $V_j^{\text{SNN}}(t)$ is updated as follows:
   $V_j^{\text{SNN}}(t) = V_j^{\text{SNN}}(t) + \sum_{i=1}^{N} w_{ij}^{\text{normalized}} \cdot \delta(t - t_i)$
   where N is the number of input neurons, t_i is the spike time of the presynaptic neuron i, and δ is the Dirac delta function.
4. Set the firing threshold θ_j for each neuron in the SNN based on the ANN biases and normalized weights:
   $\theta_j = b_j^{\text{ANN}} \cdot W_{\text{max}}^{(l)}$
5. For each input spike, update the membrane potential $V_j^{\text{SNN}}(t)$ of the postsynaptic neuron j. When the membrane potential reaches the firing threshold θ_j, generate an output spike and reset the membrane potential.
6. Optionally, apply a refractory period or spike frequency adaptation mechanism to the SNN neurons to fine-tune their spiking behavior and match the dynamics of the ANN more closely.

Spike-Based Backpropagation To address the challenges posed by the non-differentiable nature of spike events, researchers have developed spike-based backpropagation algorithms. One such method, introduced by Lee et al. (2020) , utilized surrogate gradients to approximate the true gradient of the loss function with respect to spike timings. By replacing the non-differentiable spike function with a differentiable surrogate, this approach enabled gradient-based learning in multi-layer SNNs.

1. Initialize the synaptic weights w_{ij}, biases b_j, and spike times $t_j^{(l)}$ for each neuron j in layer l of the Spiking Neural Network (SNN).

2. For each input spike, update the membrane potential $V_j^{(l)}(t)$ of neuron j in layer l using the following equation:
   $V_j^{(l)}(t) = V_j^{(l)}(t) + \sum_{i=1}^{N} w_{ij}^{(l)} \cdot \delta(t - t_i^{(l-1)})$
   where N is the number of neurons in layer $(l-1)$, and $t_i^{(l-1)}$ is the spike time of neuron i in layer $(l-1)$.

3. When the membrane potential $V_j^{(l)}$ of a neuron j in layer l reaches the firing threshold, generate an output spike and reset the membrane potential.

4. Calculate the error for each output neuron j by comparing the actual spike time $t_j^{(L)}$ with the desired spike time $t_j^{(L)}{}_{\text{desired}}$:
   $E_j = \frac{1}{2}(t_j^{(L)} - t_j^{(L)_{\text{desired}}})^2$

5. Calculate the total network error by summing the errors for all output neurons:
   $E = \sum_{j=1}^{N_L} E_j$
6. Compute the gradient of the error with respect to the synaptic weights $w_{ij}^{(l)}$ and biases $b_j^{(l)}$ for each neuron j in layer l using spike timing-based backpropagation rules.
7. Update the synaptic weights and biases using gradient descent to minimize the error:
   $w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta \frac{\partial E_j}{\partial w_{ij}^{(l)}}$
   $b_j^{(l)} \leftarrow b_j^{(l)} - \eta \frac{\partial E_j}{\partial b_j^{(l)}}$
   where η is the learning rate.

SuperSpike Another spike-based backpropagation method, known as SuperSpike (Zenke and Ganguli, 2018), used the principle of the feedback alignment algorithm to train SNNs. It also employed a surrogate gradient to overcome the non-differentiability of spike events, and introduced a novel synaptic plasticity rule that promoted learning in deep SNNs. While SuperSpike showed promising results, it still faced challenges in training very deep networks and handling complex tasks.

Temporal Coding Methods Temporal coding methods in supervised learning for SNNs focus on leveraging the precise timing of spikes to encode and process information. Spike Timing Dependent Plasticity with Error Backpropagation (STDP-EB) (Tavanaei and Maida, 2019) combined the principles of STDP and error backpropagation to train SNNs using precise spike timing. By minimizing the spike timing difference between the desired and actual output, STDP-EB enabled learning in deep SNNs. However, the algorithm's reliance on precise spike timing made it sensitive to input noise and temporal jitter.

2.2 Unsupervised Learning in SNNs

Unsupervised learning methods have been developed to exploit the advantages of SNNs in tasks without explicit target outputs. One such method is STDP (Caporale and Dan, 2008), a biologically inspired learning rule that modifies synaptic weights based on the relative timing of pre- and post-synaptic spikes. Another approach, Spike Timing Dependent Plasticity with Adaptive Normalization (STDP-AN) (Tegner and Kepecs, 2002), extended STDP to incorporate weight normalization, which improved the stability of unsupervised learning in SNNs.

2.3 Reinforcement Learning in SNNs

Reinforcement learning (RL) techniques have also been applied to SNNs, allowing them to learn from interaction with their environment. One notable example is Tempotron (Gütig and Sompolinsky, 2006), which used a spike-based RL algorithm to train a single-layer SNN for time-sensitive tasks. More recently, Frémaux et al. (2013) proposed a biologically plausible SNN model that combined STDP and RL principles, enabling the network to learn complex tasks in a spiking neural environment.

Reinforcement Learning (RL) in Spiking Neural Networks (SNNs) is an area of research that seeks to develop learning algorithms for SNNs that can adapt their behavior based on the feedback received from the environment. The goal is to enable SNNs to learn complex tasks and make decisions autonomously. Reinforcement learning in SNNs combines the computational efficiency and biological plausibility of spiking neurons with the powerful learning mechanisms of RL.

1. **RL Framework in SNNs:**
   Reinforcement learning in SNNs follows the standard RL framework consisting of an agent, environment, states, actions, and rewards. In the context of SNNs, the agent is the SNN itself, which interacts with the environment by processing sensory inputs (in the form of spike trains) and generating actions (also represented as spike trains). The agent receives feedback from the environment in the form of rewards or punishments, which are used to guide the learning process.
2. **Neuronal Reward Signals:**
   To implement RL in SNNs, one of the main challenges is to develop a mechanism to convey the reward signals to individual neurons, which can then modulate their synaptic weights based on the feedback. One approach is to use global reward signals, which are broadcasted

to all neurons in the network. Another approach is to use local reward signals, which are computed and delivered to specific neurons or groups of neurons based on their activity and contribution to the overall network performance.

3. **Learning Rules for RL in SNNs:**
   A variety of learning rules have been proposed for implementing RL in SNNs. Some of the most prominent ones include:
   - **Reward-modulated STDP (Legenstein et al., 2008)**:
     Spike-Timing-Dependent Plasticity (STDP) with reward-modulated learning adjusts the synaptic weights based on the timing of pre- and post-synaptic spikes as well as the received reward signals. The learning rule can be expressed as:
     $\Delta w_{ij} = \eta(r - \bar{r}) \cdot (f_+(t_j - t_i) - f_-(t_j - t_i))$
     where Δw_{ij} is the change in synaptic weight between neuron i and neuron j, η is the learning rate, r is the received reward, $\bar{r}$ is the baseline reward, t_i and t_j are the spike times of pre- and post-synaptic neurons, and f_+ and f_- are the potentiation and depression functions that depend on the spike timing.
   - **Temporal Difference (TD) Learning:** (Jeong, 2018)
     TD learning in SNNs involves updating the synaptic weights based on the difference between the predicted and actual rewards. The learning rule can be expressed as:
     $\Delta w_{ij} = \eta \cdot \delta \cdot x_i(t)$
     where Δw_{ij} is the change in synaptic weight between neuron i and neuron j, η is the learning rate, δ is the temporal difference error given by $\delta = r + \gamma \cdot V(s') - V(s)$, $x_i(t)$ is the eligibility trace of neuron i, and $V(s)$ and $V(s')$ are the values of the current state s and the next state s', respectively. The discount factor γ determines the importance of future rewards.
   - **Policy Gradient Methods (Huh and Sejnowski, 2018):**
     Policy gradient methods in SNNs involve directly optimizing the network's policy, which maps states to actions, by updating the synaptic weights based on the gradient of the expected cumulative reward. The learning rule can be expressed as:
     $\Delta w_{ij} = \eta \cdot \nabla_w J(w)$
     where Δw_{ij} is the change in synaptic weight between neuron i and neuron j, η is the learning rate, $J(w)$ is the expected cumulative reward, and $\nabla_w J(w)$ is the gradient of the expected cumulative reward with respect to the synaptic weights. The gradient can be estimated using various techniques, such as the REINFORCE algorithm or the advantage actor-critic method.

2.4 Hybrid Learning Approaches

Some research has focused on developing hybrid learning methods that combine elements of supervised, unsupervised, and reinforcement learning. One such example is Deep Belief Networks for SNNs (DBN-SNN) (Campbell et al., 2020), which utilized a combination of unsupervised pre-training and supervised fine-tuning to train deep SNNs. Another approach, proposed by Kulkarni and Rajendran (2018), combined STDP with backpropagation to create a hybrid learning algorithm that leveraged the advantages of both techniques.

Hybrid Learning Approaches in the context of Spiking Neural Networks (SNNs) refer to the combination of different learning paradigms to exploit their strengths and mitigate their limitations. By integrating supervised, unsupervised, and reinforcement learning methods, hybrid learning approaches aim to create more robust and efficient learning mechanisms for SNNs, which can handle complex tasks and adapt to diverse environmental conditions.

1. **Supervised-Unsupervised Hybrid Learning:**
   In supervised-unsupervised hybrid learning, SNNs combine the use of labeled data for learning specific tasks with the extraction of features from unlabeled data. This approach can enhance the network's ability to generalize to new, unseen data and improve its robustness against noisy inputs. Some popular methods for supervised-unsupervised hybrid learning in SNNs include:
   - Autoencoders: Train SNNs to learn efficient, low-dimensional representations of high-dimensional input data in an unsupervised manner, which can then be used as input for supervised learning tasks.

    - Unsupervised pre-training: Train an SNN in an unsupervised manner to learn features from the data and then fine-tune the network using supervised learning for the specific task.

2. **Supervised-Reinforcement Hybrid Learning:**
   In supervised-reinforcement hybrid learning, SNNs combine the use of labeled data for learning specific tasks with the optimization of the network's behavior based on the feedback received from the environment. This approach can enhance the network's ability to adapt to changing environmental conditions and improve its overall performance. Some popular methods for supervised-reinforcement hybrid learning in SNNs include:
    - Imitation learning: Train SNNs to mimic the behavior of an expert or another neural network using supervised learning and then refine the network's behavior using reinforcement learning to improve its performance.
    - Auxiliary tasks: Train SNNs to learn multiple tasks simultaneously, with some tasks being supervised and others being reinforcement-based, to leverage the shared knowledge and improve the network's overall performance.

3. **Unsupervised-Reinforcement Hybrid Learning:**
   In unsupervised-reinforcement hybrid learning, SNNs combine the extraction of features from unlabeled data with the optimization of the network's behavior based on the feedback received from the environment. This approach can enhance the network's ability to adapt to diverse environmental conditions and improve its robustness against noisy inputs. Some popular methods for unsupervised-reinforcement hybrid learning in SNNs include:
    - Intrinsic motivation: Train SNNs to learn features from the data in an unsupervised manner and then optimize the network's behavior using reinforcement learning based on intrinsic rewards, which are generated based on the network's internal states and activities.
    - Exploration-exploitation trade-off: Train SNNs to learn features from the data in an unsupervised manner and then optimize the network's behavior using reinforcement learning, balancing the need to explore new states and actions with the need to exploit the knowledge already acquired.

3 Introduction to Rate Model

Mean field theory (MFT) is a powerful mathematical framework widely used to analyze the collective behavior of large-scale systems, such as spiking neural networks (SNNs) (Hasegawa, 2003). In the context of neuroscience, MFT provides an effective approach to describe the average behavior of neurons within a network while reducing the complexity associated with simulating individual interactions. In this section, we introduce the fundamental concepts of MFT as applied to SNNs and present the mathematical details necessary for a comprehensive understanding.

The primary idea behind MFT is to approximate the behavior of a neuron in the network by the average behavior of all neurons, under the assumption that the interactions between individual neurons are sufficiently weak. Let us denote the membrane potential of neuron i at time t as $V_i(t)$ and the spike train of neuron i as $S_i(t)$. The dynamics of $V_i(t)$ can be described by the following differential equation:

$$\tau_m \frac{dV_i(t)}{dt} = -V_i(t) + \sum_{j=1}^{N} w_{ij} S_j(t) + I_i(t), \tag{4}$$

where τ_m is the membrane time constant, w_{ij} represents the synaptic weight from neuron j to neuron i, and $I_i(t)$ is the external input to neuron i. The spike train $S_j(t)$ is given by:

$$S_j(t) = \sum_{k} \delta(t - t_{j,k}), \tag{5}$$

where $\delta(t)$ is the Dirac delta function and $t_{j,k}$ represents the time of the k-th spike emitted by neuron j.

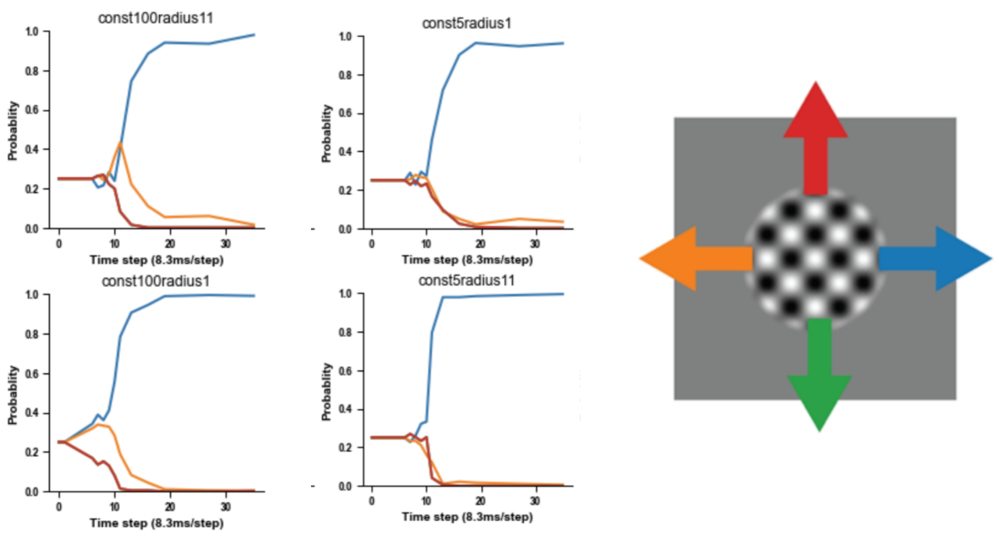


Abbildung 1: Behavioral Data

In order to apply MFT, we introduce the population-averaged membrane potential $\bar{V}(t)$ and population-averaged firing rate $\bar{r}(t)$, defined as:

$$\bar{V}(t) = \frac{1}{N}\sum_{i=1}^{N} V_i(t), \tag{6}$$

$$\bar{r}(t) = \frac{1}{N}\sum_{i=1}^{N}\sum_{k} \delta(t - t_{i,k}). \tag{7}$$

By replacing the individual membrane potentials and spike trains with their respective population averages in the differential equation for $V_i(t)$, we obtain a mean-field approximation for the dynamics of the network:

$$\tau_m \frac{d\bar{V}(t)}{dt} = -\bar{V}(t) + w_{avg}\bar{r}(t) + \bar{I}(t), \tag{8}$$

where w_{avg} is the average synaptic weight and $\bar{I}(t)$ is the population-averaged external input.

The mean-field approximation greatly simplifies the analysis of SNNs, as it reduces the complexity of simulating individual neuronal interactions to describing the average behavior of the network. This approximation is particularly useful for studying the emergent properties and dynamics of large-scale networks, which can provide insights into the underlying principles governing neural computation and information processing.

In naturalistic motion perception, even though a stimulus may barely occupy a receptive field for a limited time, observers must nonetheless extract accurate estimations of both self and object motion. Much of the earlier motion research has employed longer stimuli (Newsome and Pare, 1988), such as those lasting hundreds of milliseconds or longer, which require observers to integrate weak signals over time to determine the sensitivity of observers to analogously brief stimuli. Recently, there is a dataset based on an psychophysics experiment. It was four-alternative forced-choice visual motion discrimination task, employing both short and long stimuli.

As shown in the Figure 1, It's basically a traditional psychophysics experiment. The stimulus is the plaid motion shown here. It could move in four directions, up, down, left, and right. And the subjects are going to make the decision on what direction it's moving. You could imagine that we only show the stimulus to subjects for a short time, for example, 10 ms. Then subjects are basically making a

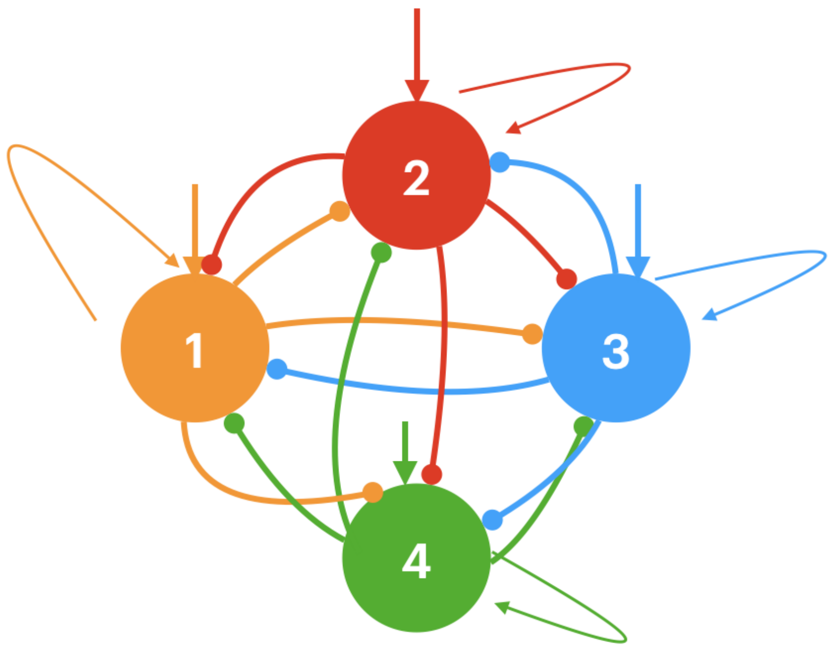


Abbildung 2: simplified four-variable version of a biophysically realistic cortical network model of decision making

random guess. But if we show for a long time; then there is a high chance they could make the right decision. Here we have four conditions. The contrast could be high or low, which means you could see white or black in the stimulus or just gray. And the size of the ball could be big or small. If we suppose the right direction is the correct one, then you could see the results from these four figures. The x is the time, and y are the probability of the four directions the subjects made. It makes sense that with the limited time, they are all just 0.25, which is a random guess. And the correct one will converge to 1 when time goes up. However, what interests us is that in the high contrast condition, you could see the left, which is the opposite direction, going up and down. And in low contrast, there is no such thing. We call it the axis extraction since there seems to be a temporary preference for the axis in the high contrast condition; I mean, the right and the left are on the same axis. The other interesting finding is that in the big size here, there will be an interval of time when the opposite direction is even higher. The project is trying to explain the axis extraction with the help of neural network.

4 Results

4.1 Reduced Spiking Neural Network (RSNN)

4.1.1 Network Definition

Xiaojing-Wang Wang (2002) proposed a biophysically realistic cortical network model for a binary visual discrimination experiment. It's a common way to reduce the model by treating the net input to a neuron in a large homogeneous population as a Gaussian random process (Wong and Wang, 2006). Therefore, for a binary visual decision making task, the mean activity of a (homogeneous) population (left/right) can be represented by a single unit. Based on the well-developed simplified two-variable version of a biophysically realistic cortical network model of decision making (Wong and Wang, 2006), I extended it to a four-variable version (See Figure 2). The 4 units stand for 4 directions, where 1/2/3/4 indicate 180/90/0/270 deg. There is a self-to-self excitation for each unit, and each unit will send inhibition to all the other three units.

Basically, let r be the firing rate of a leaky integrate-and-fire (LIF) neuron receiving a noisy input current. Then, the firing rate could be described by the equation 9 (Amit and Tsodyks, 1991; Renart et al., 2003; Ricciardi, 2013) shown below.

$$r = \Phi(\mathcal{I}_{syn}) = (\tau_{ref} + \tau_m \sqrt{\phi} \int_{\frac{V_{reset}-V_{ss}}{\sigma_V}}^{\frac{V_{th}-V_{ss}}{\sigma_V}} e^{x^2}(1 + erf(x))dx)^{-1} \tag{9}$$

where Φ is a function of the total synaptic input current I_{syn}. τ_m is the membrane time constant. V_{th} is the spiking threshold for the membrane voltage, V_{reset} is the reset voltage, τ_{ref} is the refractory period, σ_V is the membrane potential SD, and $Vss = VL + \frac{I_{syn}}{g_L}$. Instead of using the equation 9, the two-variable version used Abbott and Chance function (Abbott and Chance, 2005) for $\Phi(I_{syn})$. Then, they could use several first-order dynamical equations to model the firing rate of the model. Here, I used the same simplification but for four variables. The model considers four excitatory neural assemblies, populations 1, 2, 3 and 4, standing for left, up, right and down, that compete with each other through a shared pool of inhibitory neurons. The firing rate is shown in the equation 10.

$$r_i = F(I_i) = \frac{aI_i - b}{1 - exp(-d(aI_i - b))},\ i = 1, 2, 3, 4 \tag{10}$$

In the equation 10, I let r_1, r_2, r_3, and r_4 be firing rates of E and I populations, and the total synaptic input current I_i and the resulting firing rate r_i of the neural population i obey the following input-output relationship ($F - I$ curve).It captures the current-frequency function of a leaky integrate-and-fire neuron.
Moreover, the synapic drive is defined as below

$$\frac{dS_1}{dt} = F(I_1)\gamma(1 - S_1) - S_1/\tau_s \tag{11}$$

$$\frac{dS_2}{dt} = F(I_2)\gamma(1 - S_2) - S_2/\tau_s \tag{12}$$

$$\frac{dS_3}{dt} = F(I_3)\gamma(1 - S_3) - S_3/\tau_s \tag{13}$$

$$\frac{dS_4}{dt} = F(I_4)\gamma(1 - S_4) - S_4/\tau_s \tag{14}$$

The net current into each population directions 1/2/3/4 indicate 180/90/0/270 deg:

$$I_1 = J_E S_1 + J_{Iothg} S_2 + J_{Iopst} S_3 + J_{Iothg} S_4 + I_b + J_{ext}\mu_1 \tag{15}$$

$$I_2 = J_E S_2 + J_{Iothg} S_1 + J_{Iopst} S_4 + J_{Iothg} S_3 + I_b + J_{ext}\mu_2 \tag{16}$$

$$I_3 = J_E S_3 + J_{Iothg} S_2 + J_{Iopst} S_1 + J_{Iothg} S_4 + I_b + J_{ext}\mu_3 \tag{17}$$

$$I_4 = J_E S_3 + J_{Iothg} S_1 + J_{Iopst} S_2 + J_{Iothg} S_3 + I_b + J_{ext}\mu_4 \tag{18}$$

There are three critical parameters in this model, which are J_E, J_{Iothg} and J_{Iopst}, where is the connection weight for the self-to-self excitatory connection weight, J_{Iopst} is the self-to-opposite inhibitory connection weight, and J_{Iothg} is the self-to-orthogonal inhibitory connection weight. Since unit 1 (180^o) to unit 3 (0^o) are opposite direction, and 1 (180^o) to both unit 2 (90^o) and unit 4 (270^o) are orthogonal direction, the J_E, J_{Iothg} and J_{Iopst} are set accordingly in the equation 15.

An important contribution here I made is designing the input of the RSNN. Basically, I computed the motion energy of the input stimulus. After using the Naka-Rushton function (Schauder et al., 2017) to fit the motion energy, I design the input based on the motion energy.

4.1.2 RSNN Training Proposed

The firing rate r_i of the unit for the i-th choice is shown as follows: $r_i = F(I_i) = \frac{aI_i - b}{1 - exp(-d(aI_i - b))}$, $i = 1, \cdots, K$, where I_i (Input current) at time t is define as $I_i = \Sigma_i^j J_{ji} \cdot S_j + I_{bi} + J_{ext} \cdot \mu_i$. Here, the S_i (the synaptic drive) is defined as $\frac{dS_i}{dt} = F(I_i)\gamma(1 - S_i) - S_1/\tau_i$. J_{ji} is the connection weight from the unit j to the unit i, I_{bi} is the background input, and μ_i is the e_t^{gi}(input as we mentioned before) constrained by a uniform weight J_{ext}. Except for μ_i, all the other variables with parameters we don't introduce in detail is trainable. The output of i-th unit at time t (trial g) is defined as $e_t^{gi} = \sigma(\mathfrak{I})\,i$, where σ is SoftMax and $\mathfrak{I} = (r_1, r_2, \cdots, r_K)$. That said,

$f(x)^g_t = (e^{g1}_t, e^{g2}_t, \cdots, e^{gK}_t)$. We define the objective function O is consisting of two parts, one is the hypothesis reading from the data and the other one is the weighted 2-normal distance between the model outputs and the labels. Therefore, I proposed the training for the RSNN as an optimization problem, the objective function is shown below:

$$O(f(x), y)) = \Psi_y(f(x)) + \mathcal{M}(f(x), y)$$

$$\mathcal{M}(f(x), y) = E_t[\Sigma_i^k \frac{\Sigma_i^k |y^{gi}_t| + |e^{gi}_t|}{|y^{gi}_t| + |e^{gi}_t|} \left(e^{gi}_t - y^{gi}_t\right)^2$$

$$\Psi_y = K \cdot sgn(E_{|\frac{dr_1}{dt}\frac{dr_3}{dt}|>\epsilon}(r_3 - r_1)) \cdot E_{c,|\frac{dr_1}{dt}\frac{dr_3}{dt}|>\epsilon, \frac{dr_1}{dt}>0, \frac{dr_3}{dt}>0} e^{c}(\frac{dr_3}{dt} - \frac{dr_1}{dt})$$

Here H and C are constants. Here the hypothesis reading encoded are the two key characteristics of motion perceptual choice: (1) the axes effect—observers first perceive the motion axes (i.e., horizontal/vertical) then delineate the exact direction (i.e., left/right) within the axes of the high contrast motion stimuli; (2) the opposite-direction effect—Brief Suppression. We develop a derivation of the adjoint method(Errico, 1997) to compute the gradients of the parameters numerically, then take Adam(Kingma and Ba, 2015) as the learning method. The results show that our training method can reproduce the behavior data well. The results, only considering hypothesis reading is hard to capture the temporal pattern.

Here since we have only three parameters to be trained, J_E, J_{Iothg} and J_{Iopst}, we could use numerical differentiation to do the SGD for this optimization.

4.1.3 input to RSNN

The input to the RSNN is the motion energy based on the spatiotemporal energy models (Adelson and Bergen, 1985). It's a classical method in capturing the perception of motion, based on the outputs of quadrature pairs of filters. The first step is to compute the Fourier Transform of the stimulus, then numerically fitting the distribution of the frequencies across the time.

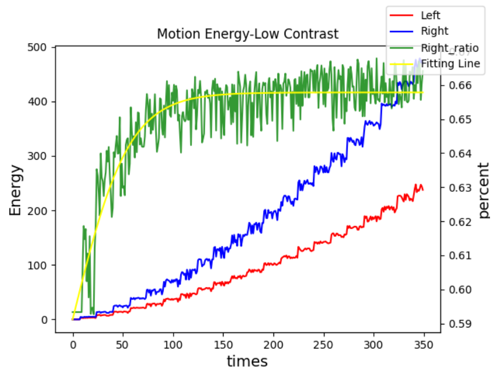


Abbildung 3: Low Contrast Motion Enengry

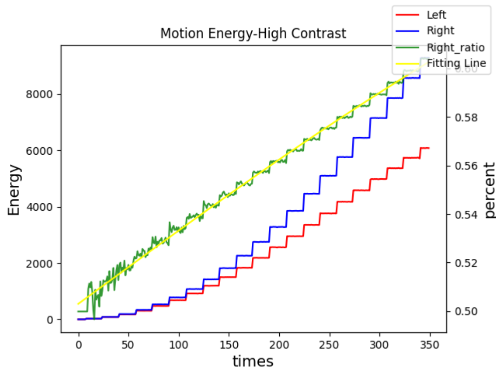


Abbildung 4: High Contrast Motion Enengy

As shown in the Figure 3 4 for low-contrast stimuli, the ratio difference immediately emerges at very short durations. But for high-contrast stimuli, the ratio difference gradually increases. This may explain why we see the two directions get tangled only for high-contrast stimuli.

4.2 Training of RNN

The training of the RNN is based on a framework proposed for modeling a wide range of neural activity patterns and behavior (Song et al., 2016a). The implementation was done with Neurogym. The input is a 1×44 vector, representing the stimulus's radius and contrast. We have 4 conditions: low contrast-high contrast, low contrast-low contrast, high contrast-low contrast, high contrast-high contrast. Then there is a hidden layer that has 256 nodes. The input connects the hidden layer with a 44×256 matrix with a 1×256 bias matrix. Every unit in the hidden layer connects with all the other units in the hidden layer, so there is a 256×256 hidden to hidden matrix. It is an Excitation-Inhibition network because we define some of the connections must be positive and some of them must be negative. To be specific, 204 units are designed to be excitatory, and 52 are designed to be inhibitory. Then, we link the 204 excitatory units to a 1×4 output vector. Therefore, there is a 4×204 output matrix. The output vector represents the behaviour, e.g. 1-left, 2-up.

4.2.1 RSNN Results

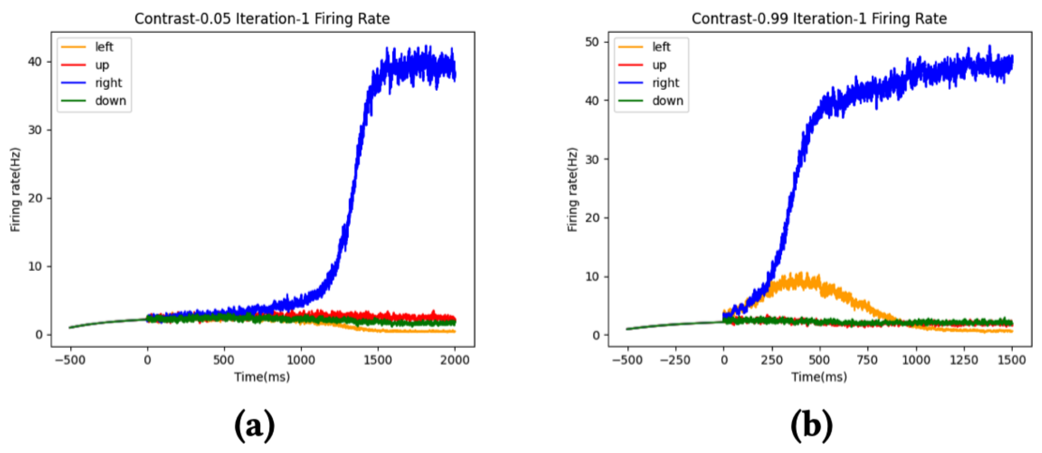
After training the model accordingly, we could obtain the firing rate for low (contrast=0.05) and high (contrast=0.99) contrast conditions, as shown in Figure **??** and Figure 5. The parameters are listed as follows: J_E = 0.3103, J_{Iothg} = -0.007, J_{Iopst} = -0.048. There is a stronger self-to-opposite inhibitory connection weight. As we can see, both in the low and high contrast condition, there is a competition among the four groups of neurons at the beginning, and then, following with the wining of the right direction. However, only in the high contrast condition could we observe the stronger opposite direction (left) comparing to the orthogonal directions (up/down). In terms of the decision making, we could put the firing rate to a softmax function to derive the psychophysical curve shown in Figure **??** and Figure **??**. That's a good match to the behavioral data regarding the axis extraction.

4.3 Excitatory-Inhibitory Recurrent Neural Network

Excitatory-Inhibitory Recurrent Neural Network has been proven to be powerful in uncovering the dynamical and computational principles governing population responses when optimized to perform the same tasks as behaving animals Song et al. (2016b). Here, we used the Excitatory-Inhibitory Recurrent Neural Network to model the 4AFC task.

4.3.1 Network Architecture

As seen in the Figure 6, there is a single hidden layer in the network, which is made up of 256 units. They are fully connected by a 256×256 weight matrix across the time. As seen in 6, 80% of the units are designed to be excitatory, that is the weights across time are forced to be positive, while the left 20% are inhibitory whose weights across time are forced to be negative.



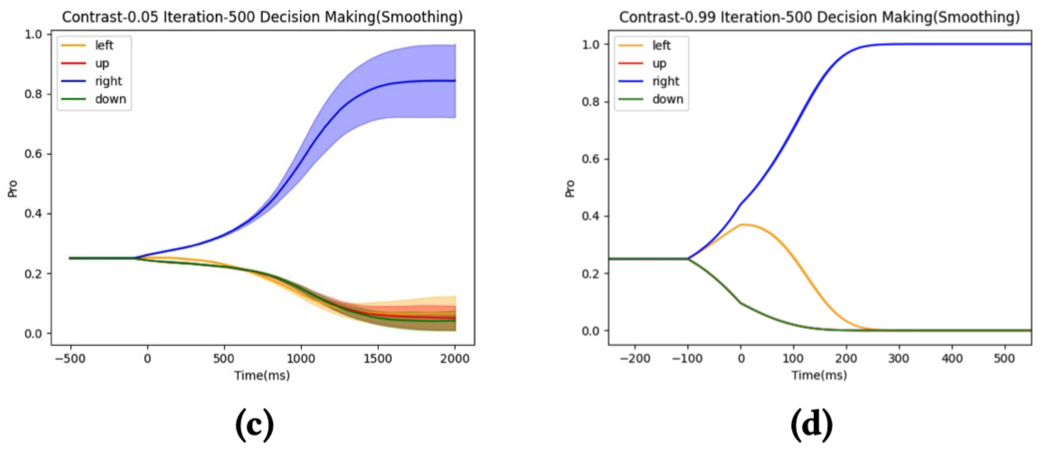


Abbildung 5: RSNN Results: (a) Firing Rate for Low Contrast (b) Firing Rate for High Contrast (c) Decision-Making for Low Contrast (d) Decision-Making for High Contrast

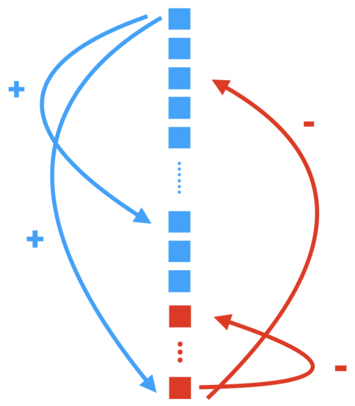


Abbildung 6: Hidden Layer

4.3.2 Training Results

After training, we could have the output of the network shown in the Figure 7. It's not surprising to see the good fitting ability based on the nature of RNN.

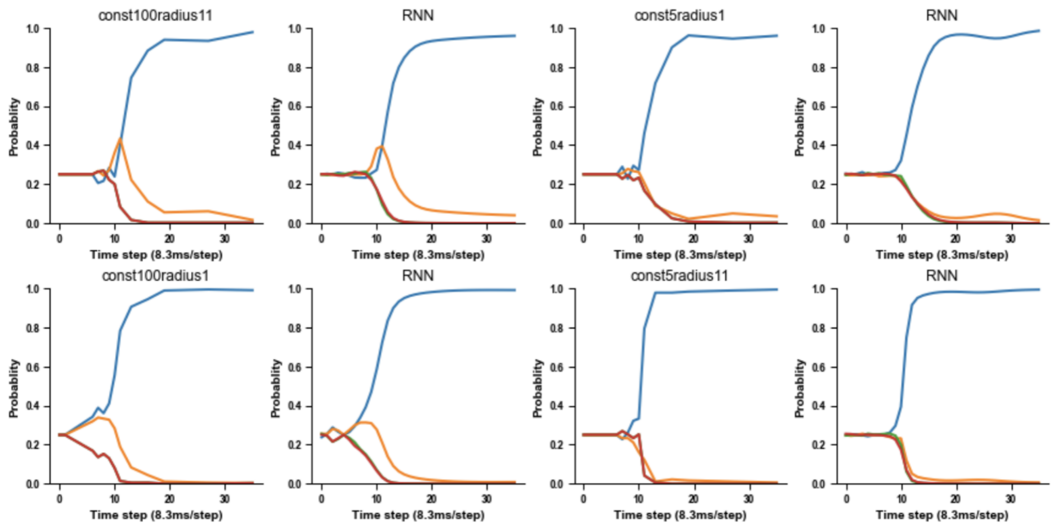


Abbildung 7: RNN Results

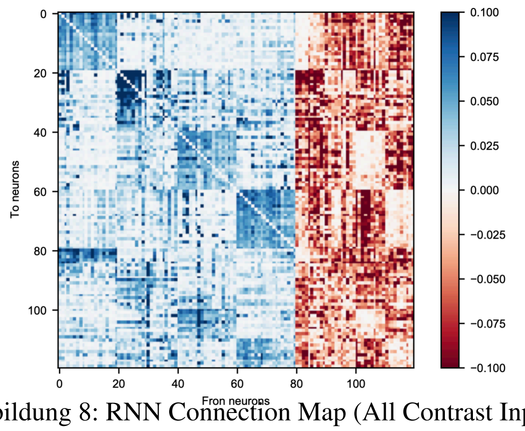


Abbildung 8: RNN Connection Map (All Contrast Inputs)

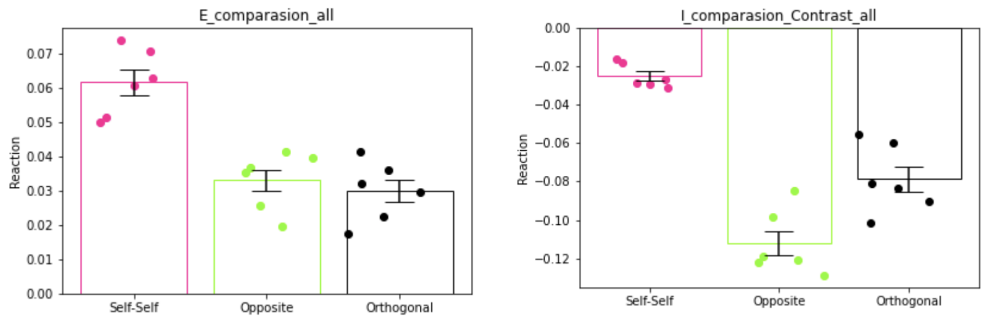


Abbildung 9: RNN Weight Connection

If we look at the the selective units in the 256×256 weight matrix, then we could compare the connection strength for different directions. As shown in the Figure 9, there is a strong self-to-self excitatory connection. Moreover, self-to-opposite inhibitory connection is stronger to self-to-orthogonal inhibitory connection. The results are consistent with what we see in the RSNN, regrading the J_E, J_{Iothg} and J_{Iopst}.

4.3.3 Single RNN units don't explain the activity of single RSNN neurons

I used a metric referred to "alignment score" (Higgins et al., 2021), which measures the extent to which variance in each neuron's firing rate can be explained by a single RNN unit. The alignment score of the four RSNN neurons are 0.0128, 0.027, 0.104, and 0.096, which is relatively low. However, we notice that the alignment score of the right direction is the highest. Overall, it shows that single RNN units don't explain the activity of single RSNN neurons. It raises the need for a more disentangling neural network, which should be a part of the future work.

5 Reinforcement Learning Approach

```
$Q$-learning: Learn function $Q : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$
Require:
  States $\mathcal{X} = \{s_1, \dots, s_{n_x}\}$
  Actions $\mathcal{A} = \{a_1, \dots, s_{n_a}\}$,    $A : \mathcal{X} \Rightarrow \mathcal{A}$
  Reward function $R : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$
  Black-box (probabilistic) transition function $T : \mathcal{X} \times \mathcal{A} \to \mathcal{X}$
  Learning rate $\alpha \in [0, 1]$, typically $\alpha = 0.1$
  Discounting factor $\gamma \in [0, 1]$
  procedure QLEARNING($\mathcal{X}$, $A$, $R$, $T$, $\alpha$, $\gamma$)
    Initialize $Q : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ arbitrarily
    while $Q$ is not converged do
      Start in state $s \in \mathcal{X}$
      while $s$ is not terminal do
        Calculate $\pi$ according to Q and exploration strategy (e.g. $\pi(x) \leftarrow \arg\max_a Q(x, a)$)
        $a \leftarrow \pi(s)$
        $r \leftarrow R(s, a)$                                  ▷ Receive the reward
        $s' \leftarrow T(s, a)$                                 ▷ Receive the new state
        $Q(s', a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s', a'))$
        $s \leftarrow s'$
    return $Q$
```

We define $S_d = \{0 \pm k * \beta \mid k = 0, 1, \cdots, N\}$, then the three trainable parameters J_E, $J_{Iothg}, J_{Iopst} \in S_d$ (the space size $n_x = (2N + 1)^3$).

For each state s, we have to decide the possible actions, here we set the $\mathcal{A} = \{[0, 0, 0], [\beta, 0, 0], [-\beta, 0, 0], [0, \beta, 0], [0, -\beta, 0], [0, 0, \beta], [0, 0, -\beta]\}$, which is of size n_a. We first initialize a Q table, here we need to create a table of size $n_x \times n_a$. Here we need to design the R, we propose that $r(s, a)$ is how well the model based on s' fits the data. We let $\{e_t\}$ denotes the simulated data, and $\{y_t\}$ denotes the observed data. For each condition (contrast / size), we have the objective function O_c below:

$$O_c(s) = E_t[\Sigma_i^4 \frac{\Sigma_i^4 \left|y_t^i\right| + \left|e_t^i\right|}{\left|y_t^i\right| + \left|e_t^i\right|} \left(e_t^i - y_t^i\right)^2]$$

, where $e = e(s)$. Then we define the R to be $-\Sigma_c O_c$.

The problem I see here is the q-learning approach would be able to find us in finding the optimal $s*$. The steps we need for finding the $s*$ first time is largely dependent on the initial Q table, which we should have no control.

6 Forward-Forward Non-BP SNN training

Ongoing

7 DISCUSSIONS

The proposed a biophysically realistic cortical network model provides a good explanation the quadruple visual discrimination experiment. Turning the spiking neural network could be converted to an optimization problem when the number of parameters is limited. Therefore, the training strategy for machine learning could be useful to train the reduced spiking neural network based on the numerical differentiation.

Both the RSNN and RNN show that the strong self-to-self excitatory connection and stronger self-to-opposite inhibitory connection comparing with self-to-orthogonal inhibitory connection is critical to the axis extraction. However, since single RNN units don't explain the activity of single RSNN neurons, we may need to use a more disentangling neural network in the future. It could be the disentangled sequential autoencoder (Li and Mandt, 2018), and the idea is to train it with the natural video dataset, then compare it with the RSNN, bring a more normative explanation for the axis extraction.

However, the RSNN doesn't consider the size effect at present, thus not able to fitting the time-dependent suppression of the opposite direction as seen in the Figure 1. I will work on the temporal properties of center–surround interactions in motion (Tadin et al., 2006) to include the size as a factor.

Biologically Plausible Neural Networks (BPNNs) have attracted significant attention in recent years, mainly due to their ability to bridge the gap between artificial neural networks and the biological processes that underlie them. In this paper, I present an exhaustive literature review of learning algorithms for three specific types of BPNNs: 1) Constrained Deep Neural Networks (CDNNs), 2) Spiking Neural Networks (SNNs), and 3) Reduced Spiking Neural Networks (Rate Models, RSNNs). Through case studies, I demonstrate the implementation and training of CDNNs and introduce a novel learning method for RSNNs, which we also implement. Furthermore, I propose an innovative approach to compare Spiking Neural Networks and Constrained Deep Neural Networks. As future work, I plan to expand my investigation of learning algorithms for SNNs, an endeavor that will further enhance our understanding of biologically inspired neural network models.

Literatur

Larry F Abbott and Frances S Chance. 2005. Drivers and modulators from push-pull and balanced synaptic input. *Progress in brain research* 149 (2005), 147–155.

Edward H Adelson and James R Bergen. 1985. Spatiotemporal energy models for the perception of motion. *Josa a* 2, 2 (1985), 284–299.

Daniel J Amit and MV Tsodyks. 1991. Quantitative study of attractor neural network retrieving at low spike rates. I. Substrate-spikes, rates and neuronal gain. *Network: Computation in neural systems* 2, 3 (1991), 259.

Sander M Bohte, Joost N Kok, and Johannes A La Poutré. 2000. SpikeProp: backpropagation for networks of spiking neurons.. In *ESANN*, Vol. 48. Bruges, 419–424.

Sawyer D Campbell, Ronald P Jenkins, Philip J O'Connor, and Douglas Werner. 2020. The explosion of artificial intelligence in antennas and propagation: How deep learning is advancing our state of the art. *IEEE Antennas and Propagation Magazine* 63, 3 (2020), 16–27.

Natalia Caporale and Yang Dan. 2008. Spike timing–dependent plasticity: a Hebbian learning rule. *Annu. Rev. Neurosci.* 31 (2008), 25–46.

Peter U Diehl, Guido Zarrella, Andrew Cassidy, Bruno U Pedroni, and Emre Neftci. 2016. Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 1–8.

Nicolas Frémaux, Henning Sprekeler, and Wulfram Gerstner. 2013. Reinforcement learning using a continuous time actor-critic framework with spiking neurons. *PLoS computational biology* 9, 4 (2013), e1003024.

Samanwoy Ghosh-Dastidar and Hojjat Adeli. 2009. Spiking neural networks. *International journal of neural systems* 19, 04 (2009), 295–308.

Robert Gütig and Haim Sompolinsky. 2006. The tempotron: a neuron that learns spike timing–based decisions. *Nature neuroscience* 9, 3 (2006), 420–428.

Hideo Hasegawa. 2003. Dynamical mean-field theory of noisy spiking neuron ensembles: Application to the Hodgkin-Huxley model. *Physical Review E* 68, 4 (2003), 041909.

Irina Higgins, Le Chang, Victoria Langston, Demis Hassabis, Christopher Summerfield, Doris Tsao, and Matthew Botvinick. 2021. Unsupervised deep learning identifies semantic disentanglement in single inferotemporal face patch neurons. *Nature communications* 12, 1 (2021), 1–14.

Dongsung Huh and Terrence J Sejnowski. 2018. Gradient descent for spiking neural networks. *Advances in neural information processing systems* 31 (2018).

Doo Seok Jeong. 2018. Tutorial: Neuromorphic spiking neural networks for temporal learning. *Journal of Applied Physics* 124, 15 (2018), 152002.

Andrzej Kasinski and Filip Ponulak. 2005. Experimental demonstration of learning properties of a new supervised learning method for the spiking neural networks. *ICANN (1)* 3696 (2005), 145–152.

Shruti R Kulkarni and Bipin Rajendran. 2018. Spiking neural networks for handwritten digit recognition—Supervised learning and network optimization. *Neural Networks* 103 (2018), 118–127.

Chankyu Lee, Syed Shakib Sarwar, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. 2020. Enabling spike-based backpropagation for training deep neural network architectures. *Frontiers in neuroscience* (2020), 119.

Robert Legenstein, Dejan Pecevski, and Wolfgang Maass. 2008. A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS computational biology* 4, 10 (2008), e1000180.

Yingzhen Li and Stephan Mandt. 2018. Disentangled sequential autoencoder. *arXiv preprint arXiv:1803.02991* (2018).

William T Newsome and Edmond B Pare. 1988. A selective impairment of motion perception following lesions of the middle temporal visual area (MT). *Journal of Neuroscience* 8, 6 (1988), 2201–2211.

Alfonso Renart, Nicolas Brunel, and Xiao-Jing Wang. 2003. Mean-field theory of recurrent cortical networks: working memory circuits with irregularly spiking neurons. *Computational neuroscience: A comprehensive approach* (2003), 432–490.

Luigi M Ricciardi. 2013. *Diffusion processes and related topics in biology*. Vol. 14. Springer Science & Business Media.

Kimberly B Schauder, Woon Ju Park, Duje Tadin, and Loisa Bennetto. 2017. Larger receptive field size as a mechanism underlying atypical motion perception in autism spectrum disorder. *Clinical Psychological Science* 5, 5 (2017), 827–842.

H Francis Song, Guangyu R Yang, and Xiao-Jing Wang. 2016a. Training excitatory-inhibitory recurrent neural networks for cognitive tasks: a simple and flexible framework. *PLoS computational biology* 12, 2 (2016), e1004792.

H. Francis Song, Guangyu R. Yang, and Xiao-Jing Wang. 2016b. Training Excitatory-Inhibitory Recurrent Neural Networks for Cognitive Tasks: A Simple and Flexible Framework. *PLOS Computational Biology* 12, 2 (02 2016), 1–30. https://doi.org/10.1371/journal.pcbi.1004792

Duje Tadin, Joseph S. Lappin, and Randolph Blake. 2006. Fine Temporal Properties of Center–Surround Interactions in Motion Revealed by Reverse Correlation. *Journal of Neuroscience* 26, 10 (2006), 2614–2622. https://doi.org/10.1523/JNEUROSCI.4253-05.2006 arXiv:https://www.jneurosci.org/content/26/10/2614.full.pdf

Aboozar Taherkhani, Ammar Belatreche, Yuhua Li, Georgina Cosma, Liam P Maguire, and T Martin McGinnity. 2020. A review of learning in biologically plausible spiking neural networks. *Neural Networks* 122 (2020), 253–272.

Amirhossein Tavanaei and Anthony Maida. 2019. BP-STDP: Approximating backpropagation using spike timing dependent plasticity. *Neurocomputing* 330 (2019), 39–47.

Jesper Tegner and Ádám Kepecs. 2002. An adaptive spike-timing-dependent plasticity rule. *Neurocomputing* 44 (2002), 189–194.

Xiao-Jing Wang. 2002. Probabilistic decision making by slow reverberation in cortical circuits. *Neuron* 36, 5 (2002), 955–968.

Kong-Fatt Wong and Xiao-Jing Wang. 2006. A Recurrent Network Mechanism of Time Integration in Perceptual Decisions. *Journal of Neuroscience* 26, 4 (2006), 1314–1328. https://doi.org/10.1523/JNEUROSCI.3733-05.2006 arXiv:https://www.jneurosci.org/content/26/4/1314.full.pdf

Friedemann Zenke and Surya Ganguli. 2018. Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation* 30, 6 (2018), 1514–1541.