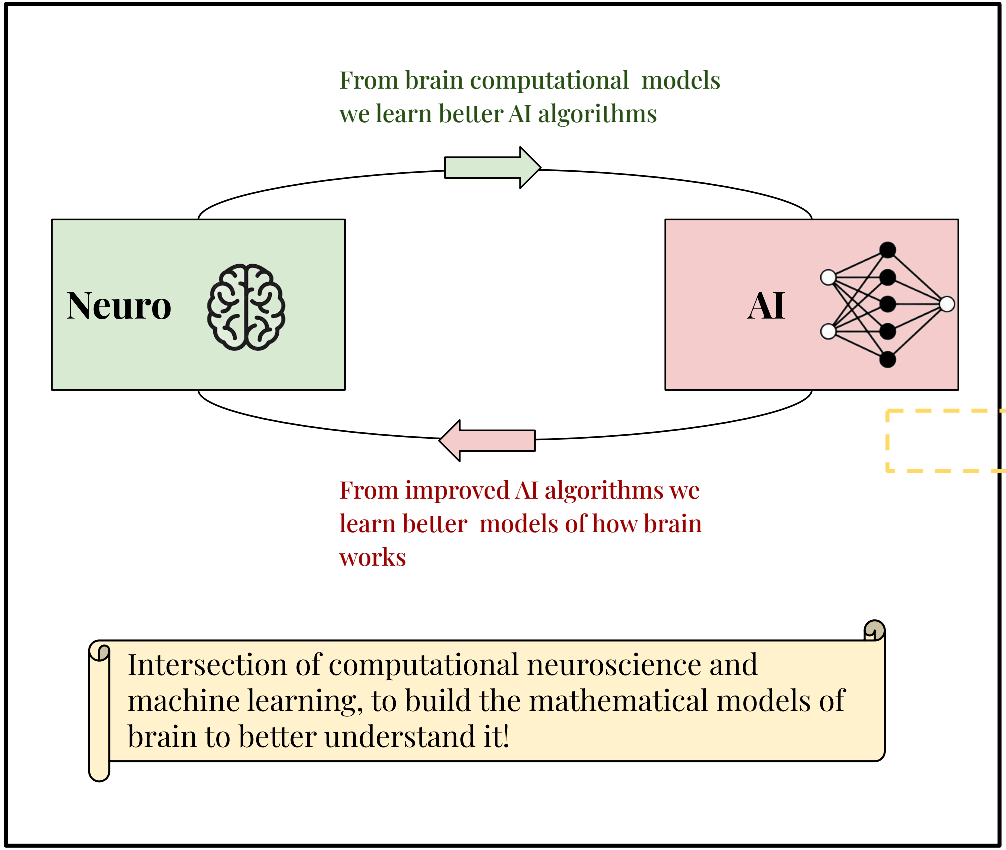
BTP-I PRESENTATION

Reinforced Liquid State Machine

by Devashri Deulkar (21CS10021)
under the guidance of Prof. Saumik Bhattacharya

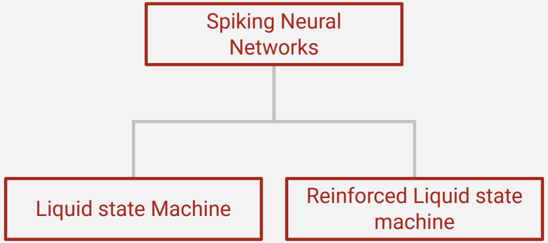
Table of Contents

Introduction and Motivation:



<u>A Novel Spiking Neural Network method inspired from neuroscience theory and analytical investigation</u>

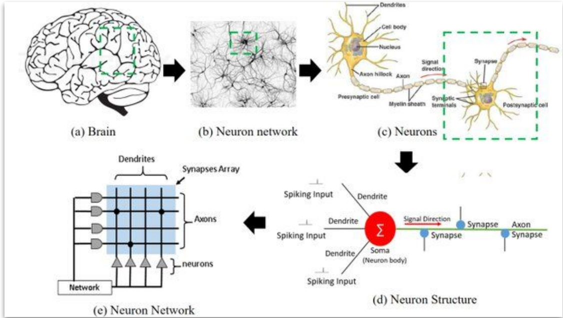
- Liquid state Machine is still unexplored potential area of interest



Inspiration

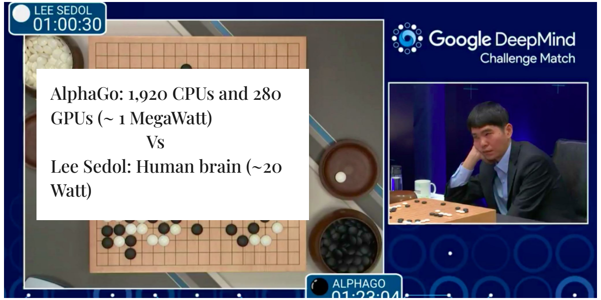
1. Neuroscience:

SNNs are what the brain does. If we want to fully understand the brain we need to understand SNNs.



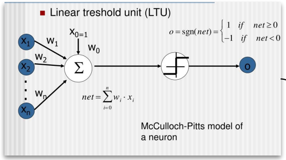
2. Energy Consumption:

Neuromorphic hardware. Low power consumption



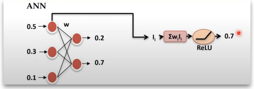
3. Different Level of Bio plausibility:

Perceptron: 1st generation

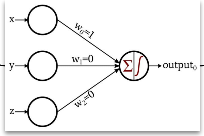


Simple classification-
Threshold non-linearity

LOW

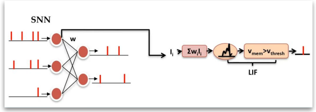


ANNs: 2nd generation

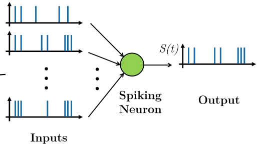


Multiclass classification-
sigmoid, Tanh, Relu

> LOW



SNN: 3rd generation



Neuromorphic computing
Integrate and fire Non-linearity

BETTER

Spiking Neural Networks

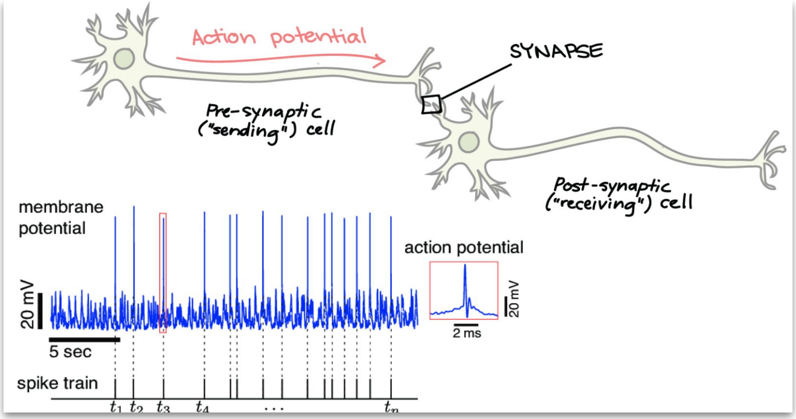
What is SNNs?

LIF neuron Model

Synapse Model

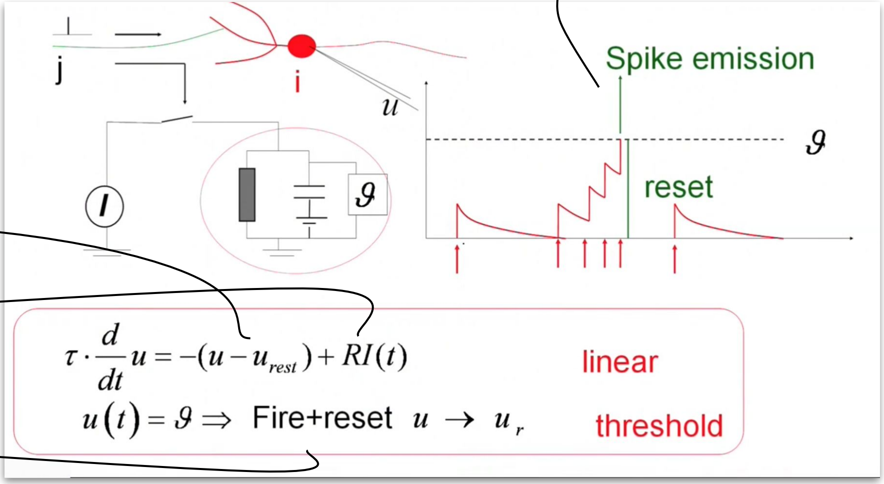
Leaky-Integrate and Fire Model

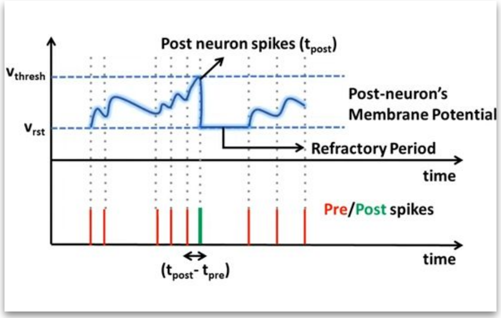
A single spiking neuron is modeled as LIF (Leaky-Integrate and Fire) Model:



We model membrane potential is modeled as capacitor, with Ion channels as pathway with resistance Ri and its equilibrium potential Ei.
When given spike ↠ Iext ↠ LIF

Non-Differential!





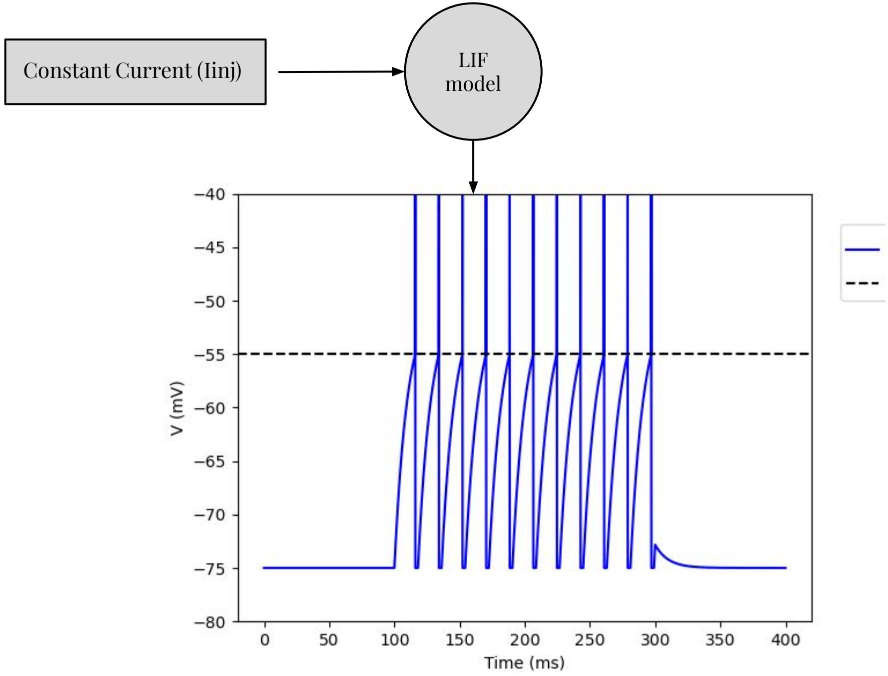
leaky

Integrate

Fire

```python
def simulate(self, Iinj, stop=False):
  # Initialize voltage
  v = np.zeros(self.Lt)
  v[0] = self.V_init

  # Set current time course
  Iinj = Iinj * np.ones(self.Lt)

  # If current pulse, set beginning and end to 0
  if stop:
    Iinj[:int(len(Iinj) / 2) - 1000] = 0
    Iinj[int(len(Iinj) / 2) + 1000:] = 0

  # Loop over time
  rec_spikes = []  # record spike times
  spike_train = np.zeros(self.Lt) # spike train
  tr = 0.  # the count for refractory duration
  count = 0

  for it in range(self.Lt - 1):

    if tr > 0:  # check if in refractory period
      v[it] = self.V_reset  # set voltage to reset
      tr = tr - 1 # reduce running counter of refractory period

    elif v[it] >= self.V_th:  # if voltage over threshold
      rec_spikes.append(it)  # record spike event
      spike_train[it] = 1    # record spike
      count+=1
      v[it] = self.V_reset   # reset voltage
      tr = self.tref / self.dt  # set refractory time



    # Calculate the increment of the membrane potential
    dv = (-(v[it]-self.E_L) + Iinj[it] / self.g_L)*self.dt/self.tau_m


    # Update the membrane potential
    v[it + 1] = v[it] + dv

  # Get spike times in ms
  rec_spikes = np.array(rec_spikes) * self.dt
  #print("rec_spikes", rec_spikes)
  spike_train = spike_train.astype(int)

  return v, rec_spikes, spike_train
```

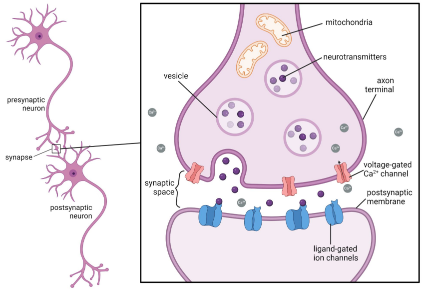
Neuron Model: LIF



(My results)

synapse model

MSSM (Modified Stochastic Synaptic Model)

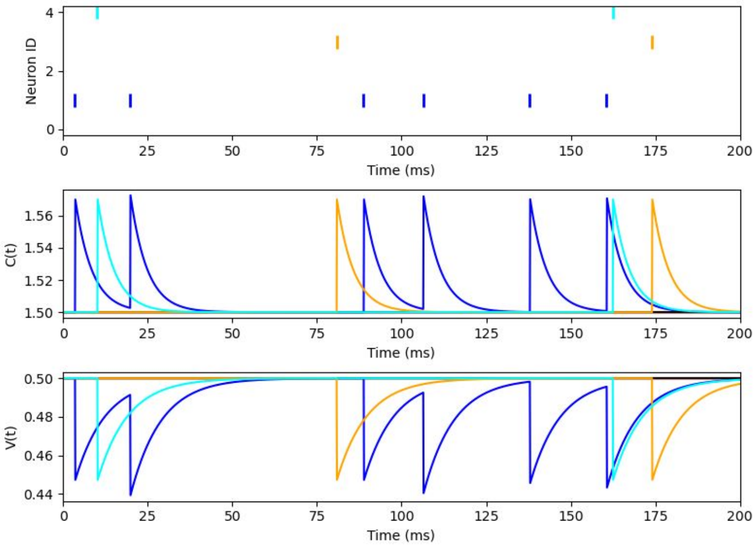


$$
\begin{aligned}
w &= C \cdot V \cdot N_t & (0)\\
P(t) &= 1 - \exp\left(-w\right), & (1)\\
\frac{dC}{dt} &= \frac{(C_0 - C)}{\tau_C} + \alpha \cdot \sum_i \delta(t - t_i), & (2)\\
\frac{dV}{dt} &= \frac{(V_0 - V)}{\tau_V} - P(t) \cdot \sum_i \delta(t - t_i), & (3)\\
\frac{dN_t}{dt} &= \max\left(0, -\frac{dV}{dt}\right) + \frac{(N_{t0} - N_t)}{\tau_{N_t}}, & (4)\\
\tau_{E_{psp}} \frac{dE_{psp}}{dt} &= -E_{psp} + k_{epsp} \cdot N_t. & (5)
\end{aligned}
$$

Simulate liquid synapses when given spike trains as input. Working as shown below:

```
synapse = MSSM(pars)
C, V, E_final, W, Iinj, P = synapse(spike_train, is_inh)
```

`synapse` is an instance, `C, V, E_final, W, Iinj` are Ca , Vesicle concentration, Potential, Weights, current



(My results)

Spiking Neural Network

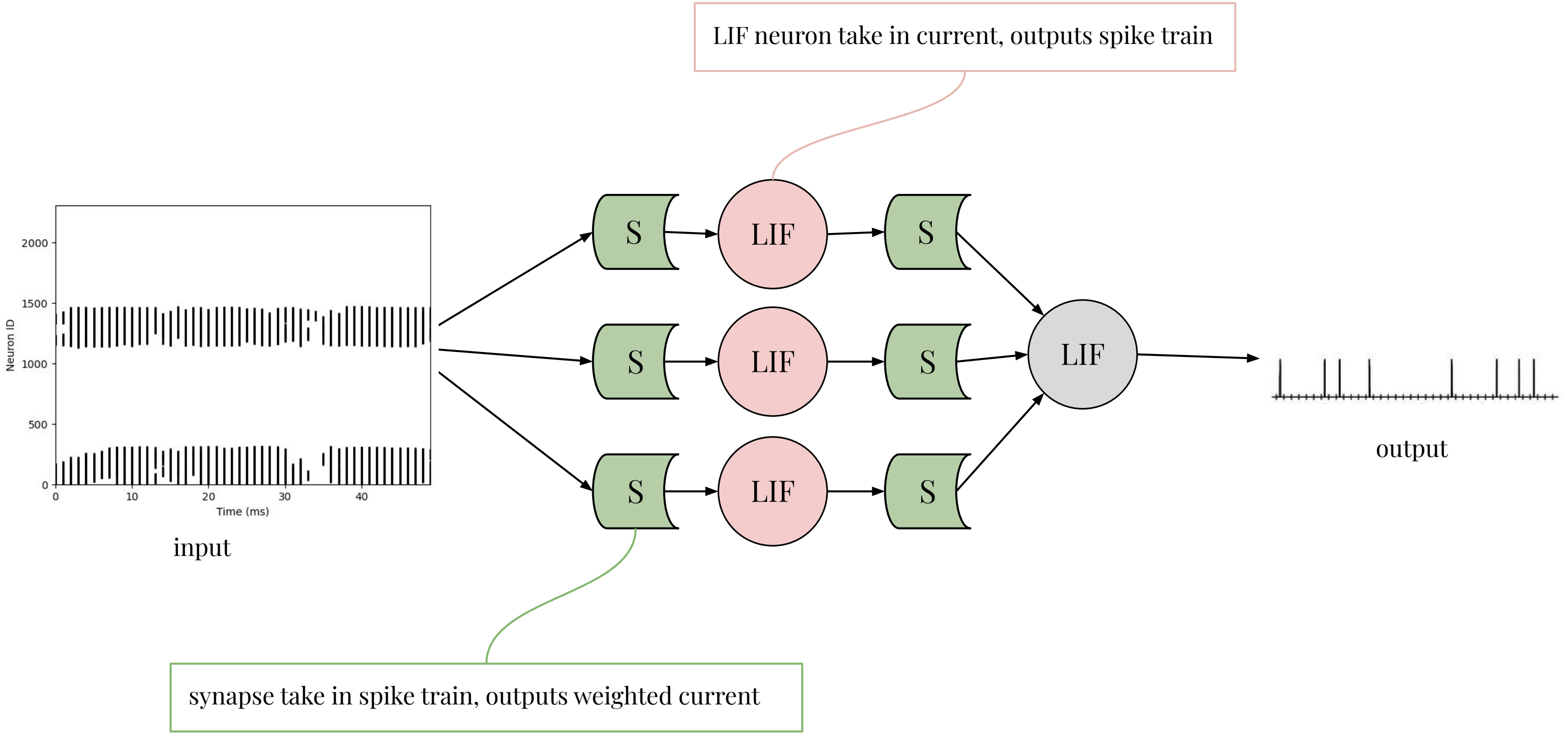
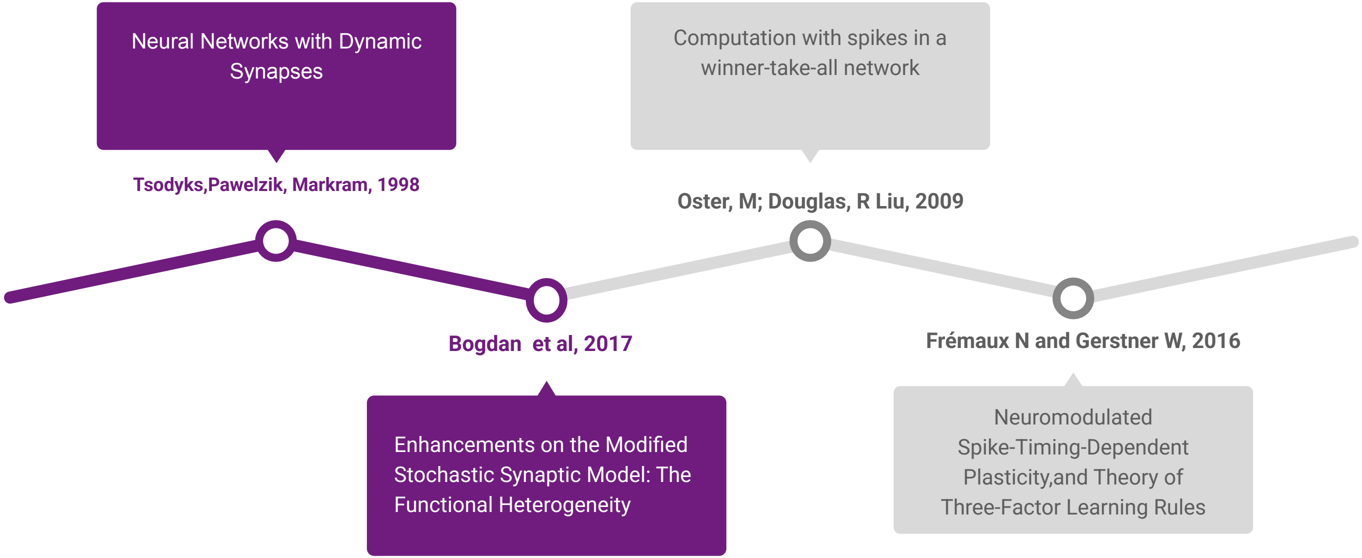


Fig. Simple Spiking Neural Network

Literature review:



Problem statement

- <u>Applying Reinforcement learning(RL) in Recurrent Spiking Neural Network (SNN)</u>
    - Predictive coding is prevalent in neural microcircuits, minimizing the prediction error through hierarchical feedback loops.
    - R-STDP reward modulated Spike time-dependant plasticity for adaptive, reward-based synaptic updates.
    - Our goal is to make model that performs well with complex tasks such as activity detection real-time videos, real-time fmri dataset etc.

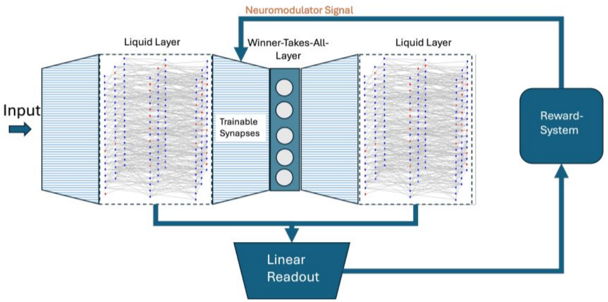


Fig. 1: The architecture of the Reinforced Liquid State Machine.

Ref. RLSM

Liquid State Machine (LSM)

Inspiration

Components

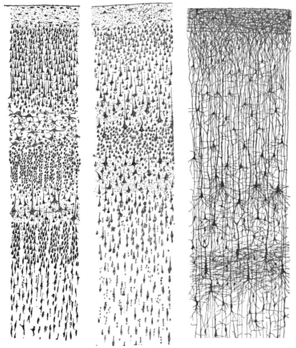
What’s LSM?

Inspiration

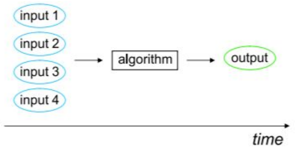
1. Almost 80% synapses are used in multiple **recurrent loops**, which is a challenge for traditional ANN.
2. Neocortex works well with rapidly changing spatiotemporal inputs and it is **high dimensional dynamical** system.

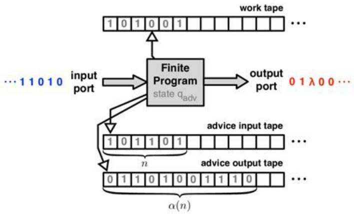
3. Issue with Turing Machine

Cortical Microcircuits:

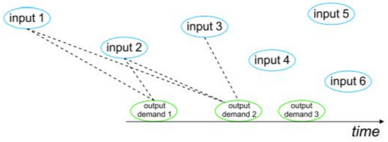


Offline computation:





Online computation:



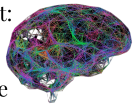
[More detail]

computation theory and algorithm design have focused for several decades on offline computations

But our cortical microcircuit does real-time computation (online computation)

4. Properties of Cortical microcircuits vs ANNs:

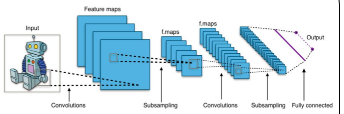
Cortical microcircuit:



- Fix Architecture
- Multiplexing
- Continuously changing spatiotemporal inputs
- Energy Efficient

VS/

ANNs:



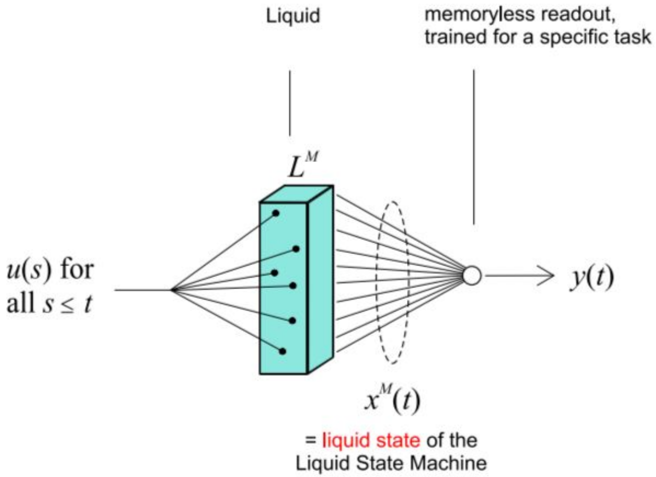
- Different Architecture
- Single task
- Turing Machine
- Energy demanding

LSM tries to Mediate the differences...

Liquid state machine: Overview

Input: encoded spike trains	Liquid: It is recurrent SNN, with few special properties.	Readout: trained output layer

Readout neurons play a crucial role of identifying salient features of high dimensional dynamic systems, with transient internal state.



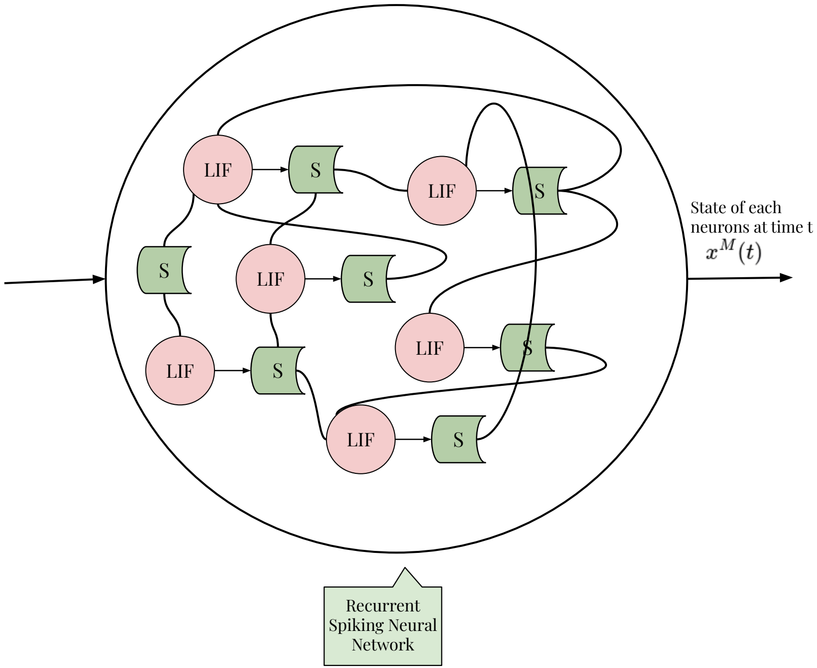
L^M liquid mapping: input u(·) → high-dimensional state $x^M(t)$

$$x^M(t) = (L^M u)(t)$$

$$y(t) = f^M(x^M(t))$$

f^M = readout (trained) → output y(t)

Liquid/reservoir:



Empirically/theoretically random reservoir/liquid follows SP and Fading memory.

- Recurrent SNN
- Encodes input into rich high dimension

$$x^M(t) = (L_M u)(t)$$

$$x^M(t) = \{\, x_1^M(t),\, x_2^M(t),\, \ldots,\, x_n^M(t) \,\}$$

- Untrained
  - Already provides High dimensional temporal space
  - Training readout layer is suffice
- Follows Separation Property (SP)

$$u_1 \neq u_2 \;\Rightarrow\; (L_M u_1)(t) \neq (L_M u_2)(t)$$

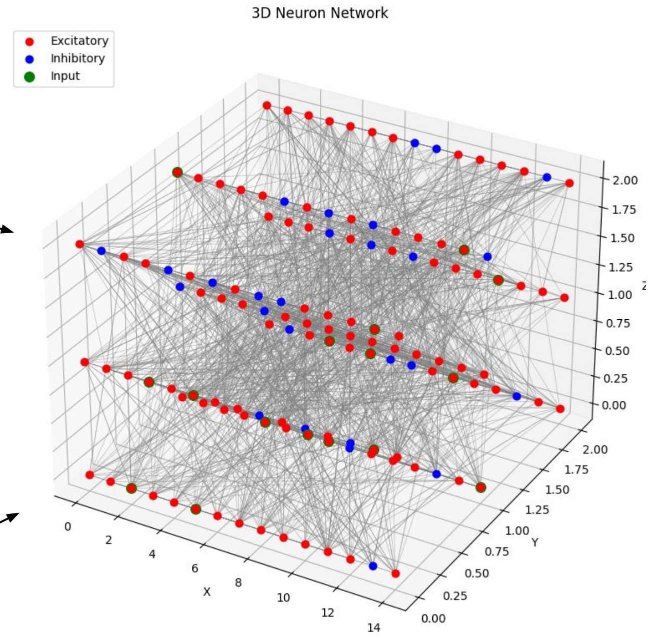
- Follows Fading memory

$$\|(L_M u)(t) - (L_M v)(t)\| \to 0 \quad \text{if} \quad u(s) \approx v(s) \text{ for } s < t$$

Liquid Architecture:

```python
# setting seed
np.random.seed(1821)
rng = np.random.default_rng(seed=1821)

# parameters
pars = default_pars()

# Liquid
rsnn = SRN(pars)

# temp synthetic data
pre_spike_train_ex = Poisson_generator(pars, rate=1500, n=27, myseed=2020) #1, 3, 4 have spikes

m, e, i, inpt = rsnn.architecture()
liquid_state = rsnn.simulate(pre_spike_train_ex)

# plot
my_neuronNet_plot(m, e, i, inpt)
```

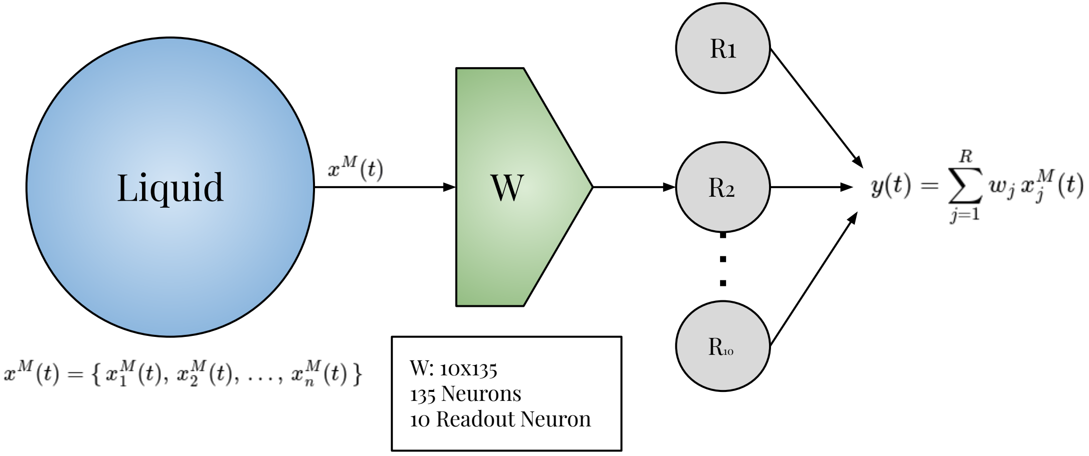
```
Number of neurons:  135
Number of excitatory neurons:  108
Number of Input neurons:  16
Index of Input neurons: [ 44  43  65  33  35 108  20 102  18  59  51   9 109 126  80  45]
Neurons in position: 41th neuron is in [6 2 1] position



Created 1454 MSSM synapses.
synapse[0]: (0, 15)
```



(My results)

Readout Layer:

The readout layer is where all task-specific learning happens



A linear readout is enough to extract the desired pattern.

- Normal Layer (AN or SN)
- Linear mapping of high dimension liquid state to output
- Trained
  - W matrix is updated by supervised learning
  - No backprop through time
- Follows Approximation Property (AP)
  - If the liquid satisfies SP and FM, then a linear readout can approximate any causal function of the input (Maass et al., 2002).

$$y(t) = f_M(x^M(t)) = f_M((L_M u)(t))$$

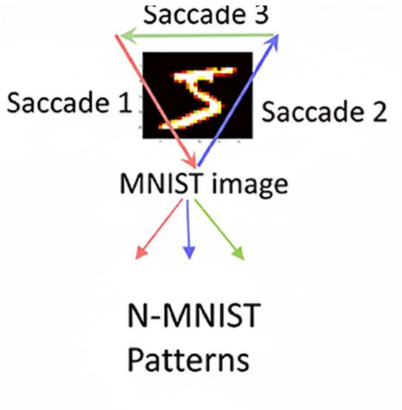
```python
#@title Readout layer
class Net(nn.Module):
    def __init__(self):
      super(Net, self).__init__()
      self.fc1 = nn.Linear(135,10)

    def forward(self, x):
      x = self.fc1(x)
      x = F.relu(x)
      return x



class ReadoutLayer:
      def init(self, pars):
          self.pars = pars
          self.net = Net()
          self.P = 10

      def simulate(self, liquid_state):
            # liquid state N neurons getting added to 10 readout neurons
            input = np.mean(liquid_state, axis=1)
            input = torch.from_numpy(input).float().unsqueeze(0) # (1, 135)
            output = self.net(input)  # (1, 10)

            return output

pars = default_pars()
readout = ReadoutLayer()
readout.init(pars)
output = readout.simulate(liquid_state)
print("train:", output.shape)

```

Readout Layer

Input: Spike Encoding + Dataset

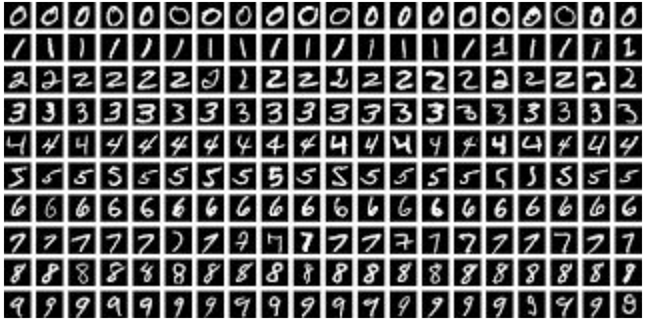
Way to converting input signals to spike trains (NMNIST)

- Neuromorphic-MNIST (NMNIST) is a spiking version of the classic MNIST dataset
- Event-based encoding :
  Generates a spike only when the pixel intensity changes
- It mimics biological eye-movements
  (Neurons emit spikes only when input signals change)

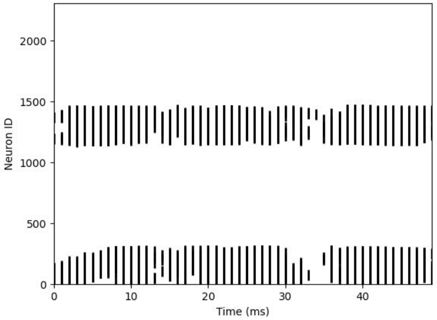


Dataset:

- N-MNIST converts the static 28×28 MNIST digits into spike trains by **physically moving a camera (ATIS)** in three small **microsaccades** while viewing the images on a monitor
- Each pixel emits an ON/OFF spike on intensity change due to motion
  ON = brightness ↑, OFF = brightness ↓

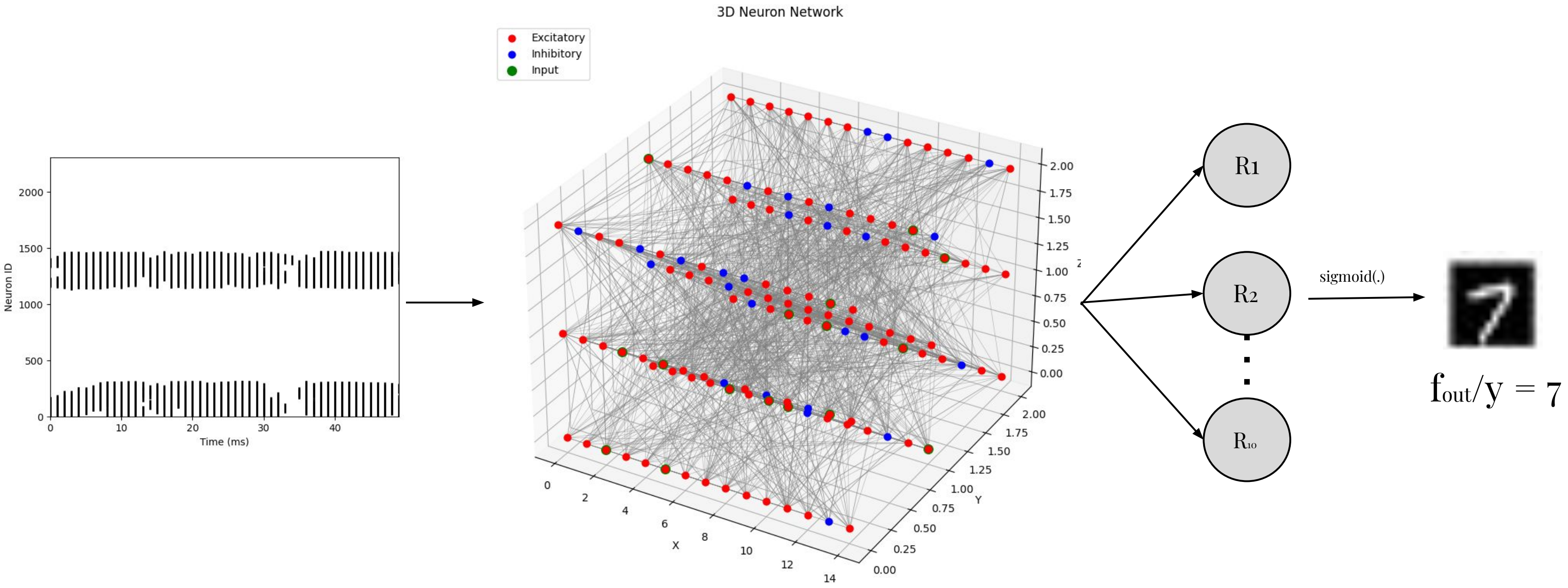


Raster graph from the N-MNIST dataset:



From Local code, 2312 neuron spike trains for digit 3

Putting all together: LSM



Input: encoded spike trains	Liquid: It is recurrent SNN, with few special properties.	Readout: trained output layer

Next Thing to do: RLSM pipeline

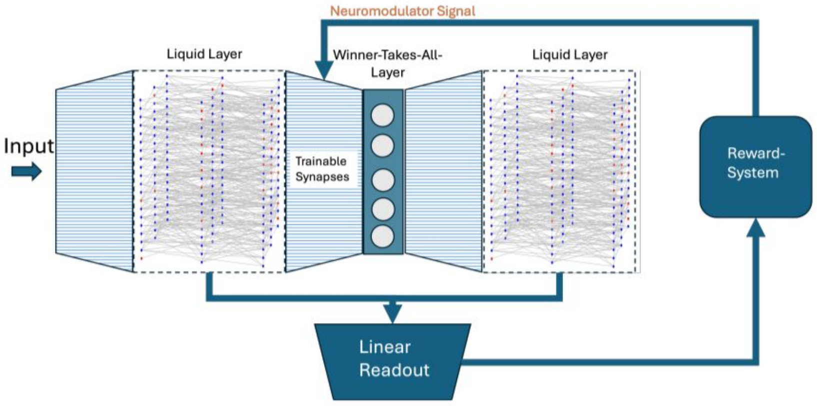


Fig. 1: The architecture of the Reinforced Liquid State Machine.

Predictive coding: Through hierarchical feedback loop, minimising predictive error.

Features:

- We use multiple liquid layer, Deep Liquid State Machine
- How do we connect liquid layer?
  - ➢ Winner-Takes-All layer, with STDP (spike-time dependant plasticity) synapse model.
  - ➢ Trainable synapses
- Reward-system: reward modulated stdp, again mimicking neurobiology of neuromodulators

Upgraded Synaptic Model: STDP

STDP is governed by the relative timing of these spikes, leading to either long-term potentiation (LTP) or long-term depression (LTD) of the synapse

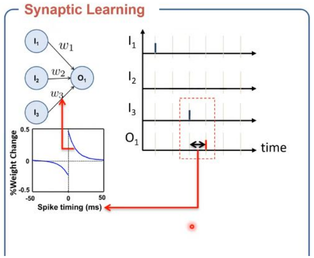
Pre-synaptic trace:

$$\frac{dx_{\mathrm{pre}}}{dt} = -\frac{x_{\mathrm{pre}}}{\tau_{\mathrm{pre}}} + \sum_f \delta(t - t^f)$$

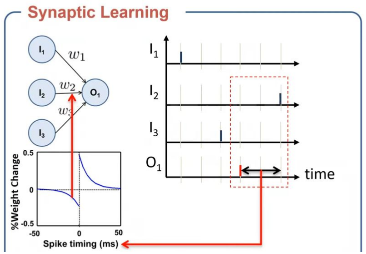
$$\Delta w_{\mathrm{local}} = \eta\left(x_{\mathrm{pre}} - x_{\mathrm{tar}}\right)\left(w_{\mathrm{max}} - w\right)^{\mu}$$

$$w_{\mathrm{local}} = \frac{w_{i,j}}{\sum_{j=1}^{N} w_{i,j}} \cdot \alpha$$

Strength of synapse (W) depends on ISI between spikes



O1 (Postsynaptic) spike just after I3 (Presynaptic)
correlated

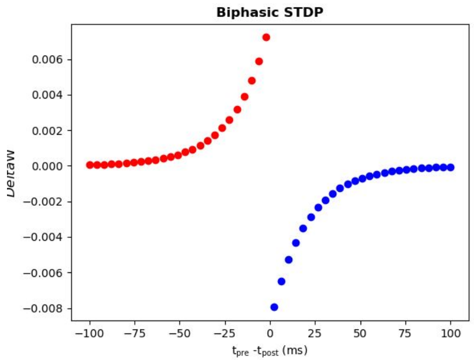


I2 (Presynaptic) spike after O1 (Postsynaptic)
uncorrelated

Weight update happens according to above rule, unsupervised

There is also concept Homeostasis, which I will not cover here...

Biphasic STDP: (My work)



WTA- Layer (R-STDP)

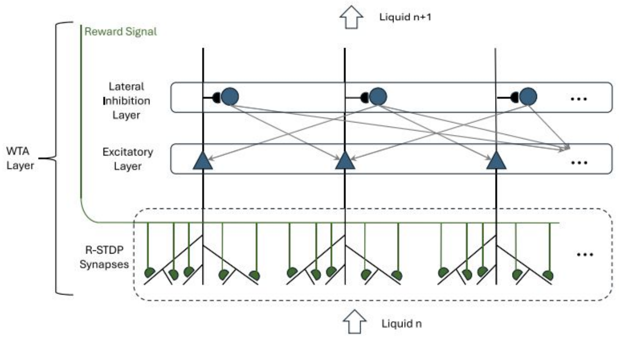


FIGURE 2
Winner-Takes-All layer with R-STDP synapses. Triangle neurons represent the winner neurons, receiving input from the preceding liquid layer. These neurons inhibit each other via round inhibitory neurons while providing their aggregated spatio temporal information to the next liquid layer. Their input synapses are governed by R-STDP and the Reward Signal is coming from the overall performance of the network.

```
    # Initialization
    N, M = W.shape
    dt    = pars['dt']
    V_th  = pars['V_wth']
    V_rst = pars['V_reset']
    spikes = np.zeros(M)

    # 1. Excitatory drive from liquid → WTA
    I_exc = ls @ W                                          # (M,)

    # 2. Top-k selection
    winners = np.argpartition(I_exc, -k)[-k:]

    # 3. Global inhibition current (shared)
    I_inh = g_inh * np.sum(I_exc[winners])

    # 4. Apply inhibition: winners keep full drive, others get clamped down
    I_exc_post = I_exc - I_inh
    I_exc_post[winners] = I_exc[winners]
    I_exc_post = np.clip(I_exc_post, 0.0, None)  # no negative currents

    # 5. Single non-leaky LIF integration step: (C=1) V(0)=0
    V = I_exc_post * dt                                     # integrate over one dt

    widx = V >= V_th                                     # which neurons cross threshold?
    #print("widx:",widx)

    spikes[widx] = 1.0                                   # spike if above threshold
    # 6. Reset all spiking neurons
    V[widx] = V_rst                                      # reset to V_reset

    return winners, spikes
```

$$\frac{de_{\text{trace}}}{dt} = -\frac{e_{\text{trace}}}{\tau_{\text{trace}}} + \Delta w_{\text{local}}$$

$$\Delta w_{\text{feedback}} = r \cdot e_{\text{trace}}$$

1. **Eligibility trace** acts as a short-term memory — it decays over time but is refreshed whenever a local synaptic change occurs.

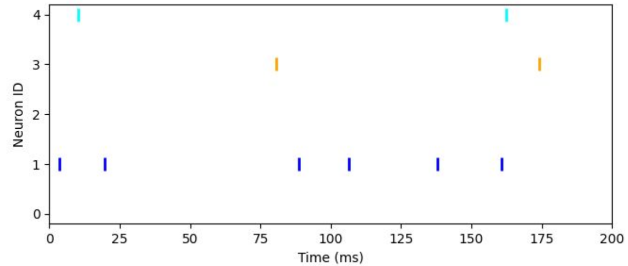
2. This trace stores **when and how much** a synapse was active, waiting for a later reward signal.

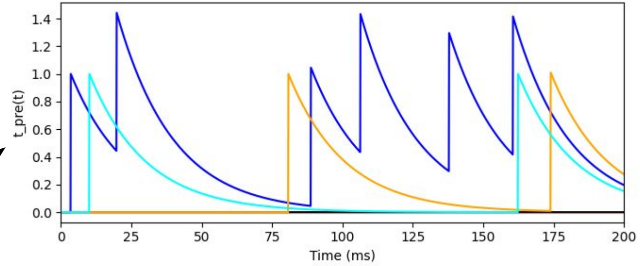
3. When a **reward** (r) arrives, the actual weight change is computed as linking past activity to current reward — the essence of **reward-modulated plasticity**.

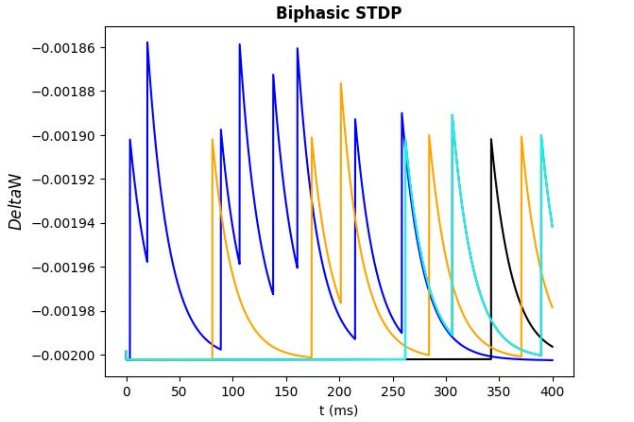
Section 5: WTA and R-STDP

1. Presynaptic trace created by incoming presynaptic spikes: $\frac{dx_{\text{pre}}}{dt} = \frac{x_{\text{pre}}}{\tau_{\text{pre}}} + \sum_f \delta(t - t^f)$
2. When the postsynaptic neuron fires, the weight of the synapse is updated based on x_{pre} and x_{tar} :
   $\Delta w_{\text{local}} = \eta(x_{\text{pre}} - x_{\text{tar}})(w_{\text{max}} \cdot w)^{\mu}$
3. Synaptic scaling: $w_{\text{local}} = \frac{w_{ij}}{\sum_j^N w_{ij}} \cdot \alpha$
4. To enable faster learning by crediting actions that contributed to future rewards, a eligibility trace etrace is introduced. This trace acts as a bridge linking the local synaptic updates: $\frac{de_{\text{trace}}}{dt} = \frac{e_{\text{trace}}}{\tau_{\text{trace}}} + \Delta w_{\text{local}}$
5. Overall weight update: $\Delta w_{\text{feedback}} = r \cdot e_{\text{trace}}$

```
pre_spike_train_ex = Poisson_generator(pars, rate=10, n=5, myseed=2020)
t_pre = generate_presynaptic_trace(pars, pre_spike_train_ex)
my_example_trace(pre_spike_train_ex, pars, t_pre)

dW,W = Delta_W(pars, t_pre)
```





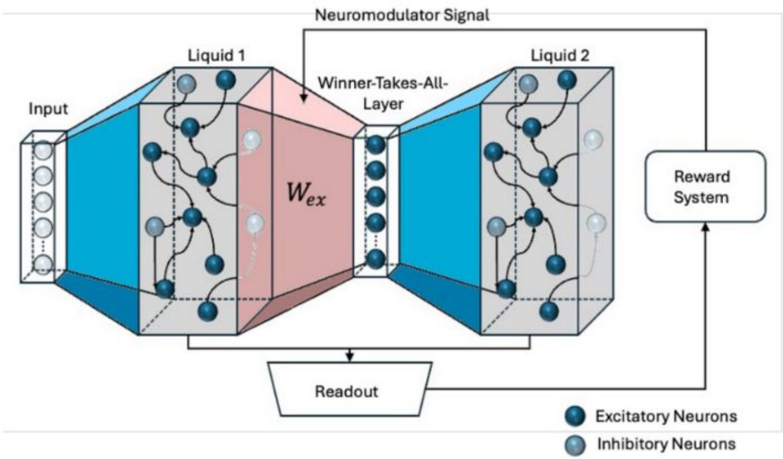


New Features Summary:

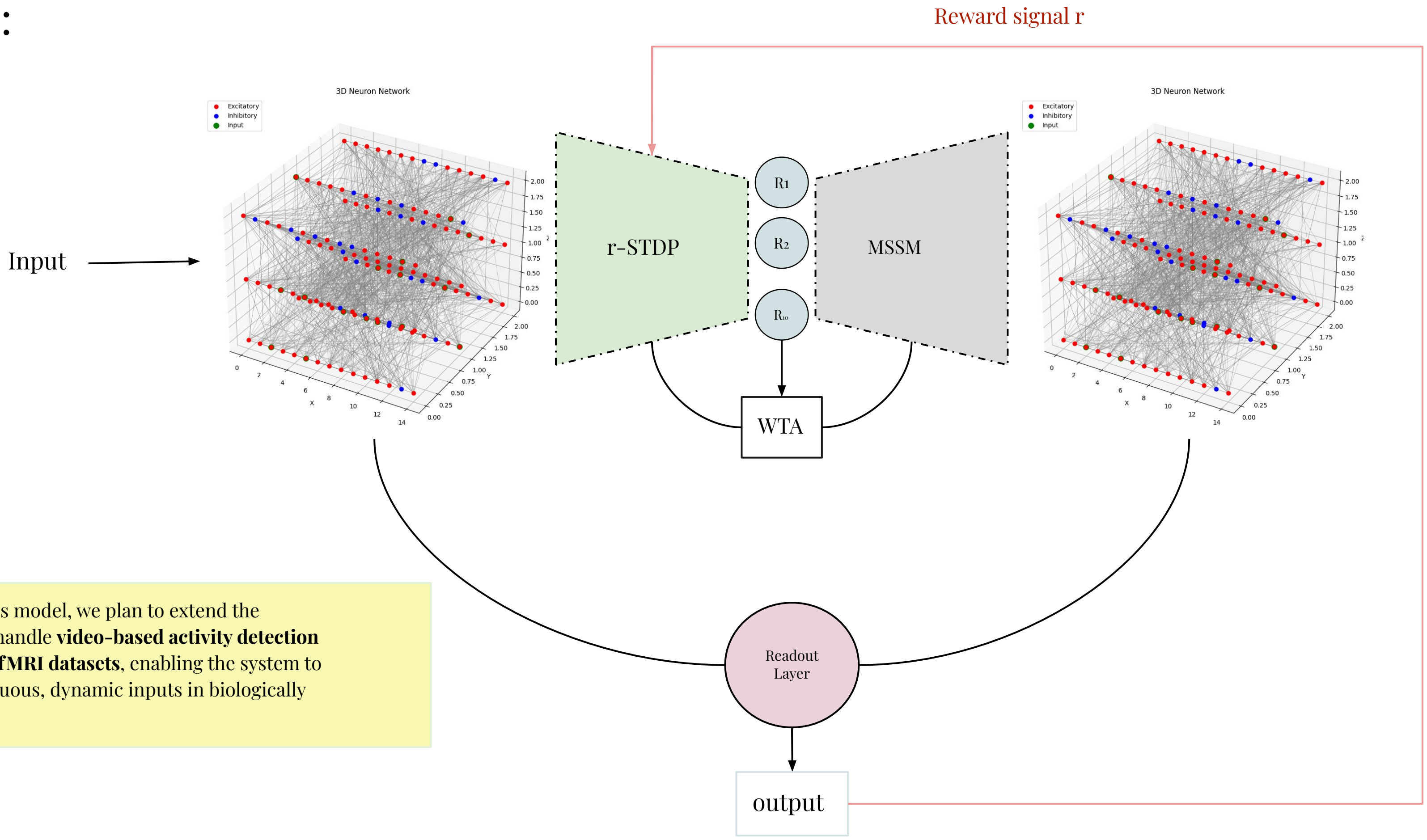
1. **Local temporal learning:**
 STDP captures fine-grained spike-timing correlations between liquid (pre-) and WTA (post-) neurons — enabling each WTA neuron to becom selective to specific temporal input patterns.

2. **Global performance feedback:**
 The reward signal adds a third factor that reinforces learning— linking local plasticity to task outcome.

3. **Competition and specialization:**
The WTA mechanism ensures only the K fastest (strongest) neurons fire; R-STDP helps to specialize on distinct input features.



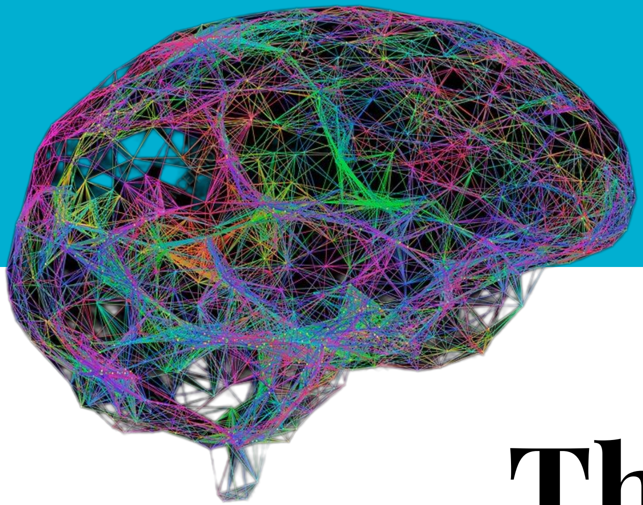
RLSM:



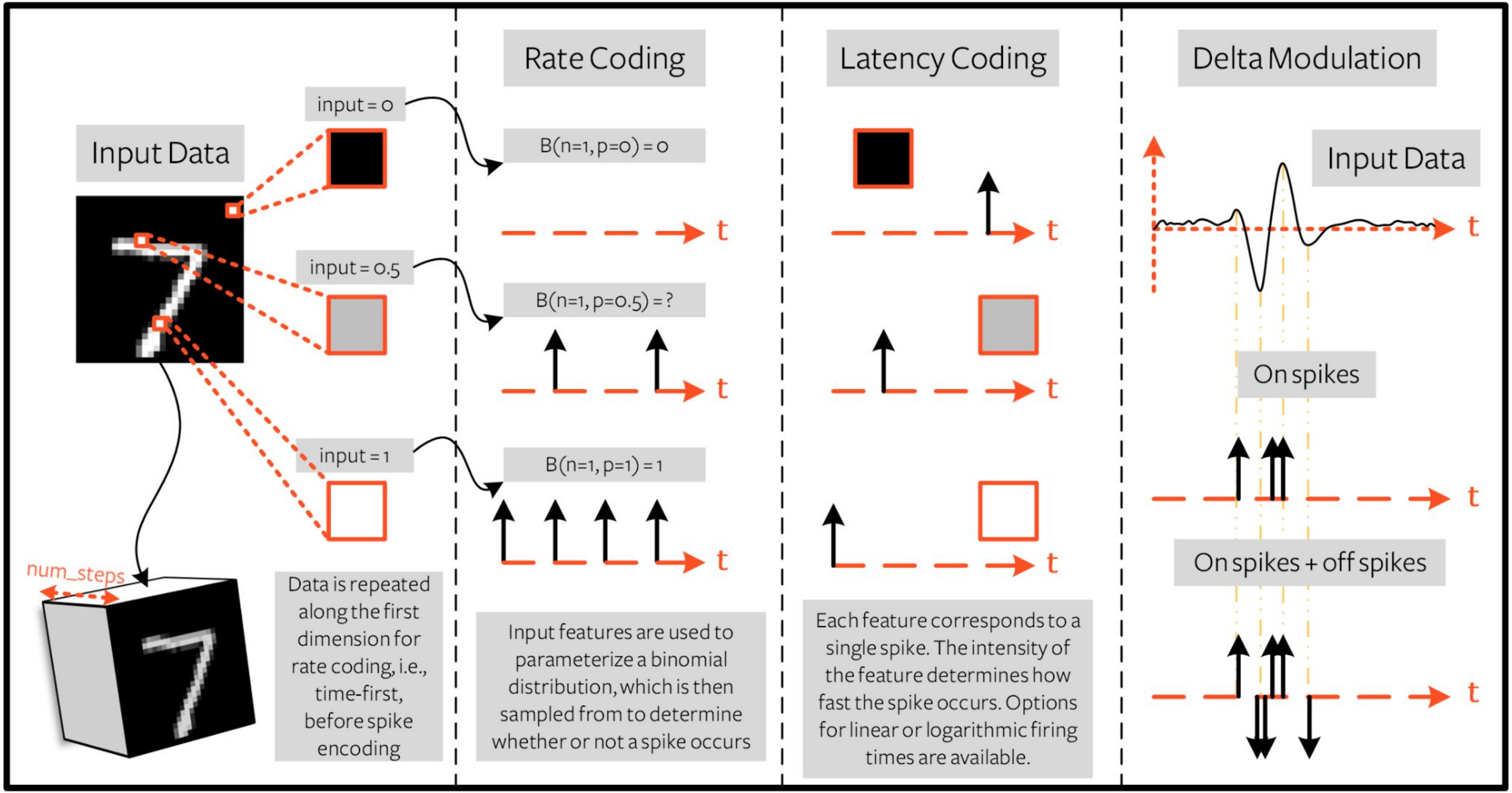
Building on this model, we plan to extend the framework to handle **video-based activity detection** and **real-time fMRI datasets**, enabling the system to process continuous, dynamic inputs in biologically realistic ways.

references:

1. Krenzer, D., and Bogdan, M. (2025). Reinforced Liquid State Machines—New Training Strategies for Spiking Neural Networks Based on Reinforcements. Frontiers in Computational Neuroscience, 19, 1569374. https://doi.org/10.3389/fncom.2025.1569374
2. Krenzer, D., and Bogdan, M. (2025). The Reinforced Liquid State Machine: A New Training Architecture for Spiking Neural Networks. ESANN 2025 Proceedings, 699–704. Bruges, Belgium.
3. Maass, W., Natschläger, T., and Markram, H. (2002). Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations. Neural Computation, 14(11), 2531–2560. https://doi.org/10.1162/089976602760407955
4. Soures, N., and Kudithipudi, D. (2019). Deep Liquid State Machines with Neural Plasticity for Video Activity Recognition. Frontiers in Neuroscience, 13, 686.
5. Frémaux, N., and Gerstner, W. (2016). Neuromodulated Spike-Timing-Dependent Plasticity, and Theory of Three-Factor Learning Rules. Frontiers in Neural Circuits, 9, 85.
6. El-Laithy, K., and Bogdan, M. (2010). Predicting Spike Timing of a Thalamic Neuron Using a Stochastic Synapse. European Symposium on Artificial Neural Networks (ESANN), 358–363.
7. Ellatihy, K., and Bogdan, M. (2017). Enhancements on the Modified Stochastic Synaptic Model: The Functional Heterogeneity. ICANN 2017, LNCS 10613, 389–396.
8. Oster, M., Douglas, R., and Liu, S.-C. (2009). Computation with spikes in a winner-take-all network. Neural Comput. 21, 2437–2465. https://doi.org/10.1162/neco.2009.07-08-829
9. Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. Neural Netw. 10, 1659–1671. doi: 10.1016/S0893-6080(97)00011-7
10. Maass, W. (2011). Liquid state machines: motivation, theory, and applications. Comput. Context 2011, 275–296. doi: 10.1142/9781848162778_0008
11. Spike times make sense VanRullen, Rufin et al. Trends in Neurosciences, Volume 28, Issue 1, 1 - 4 https://doi.org/10.1016/j.tins.2004.10.010



Thank You!


Rate Coding
Latency Coding
Delta Modulation
Input Data
input = 0
B(n=1, p=0) = 0
input = 0.5
B(n=1, p=0.5) = ?
input = 1
B(n=1, p=1) = 1
t
num_steps
Data is repeated along the first dimension for rate coding, i.e., time-first, before spike encoding
Input features are used to parameterize a binomial distribution, which is then sampled from to determine whether or not a spike occurs
Each feature corresponds to a single spike. The intensity of the feature determines how fast the spike occurs. Options for linear or logarithmic firing times are available.
Input Data
On spikes
On spikes + off spikes

back

NMNIST

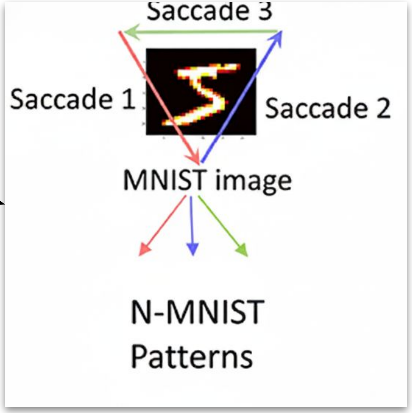
Generates a spike only when the pixel intensity changes i.e., when there is new information.

> NMNIST is created from event-based spike encoding

FIGURE 2 | (A) A picture of the ATIS mounted on the pan tilt unit used in the conversion system. **(B)** The ATIS placed viewing the LCD monitor.

Method:

1. **Event based Camera**
   ATIS (Asynchronous Time-based Image Sensor)
   -The camera/sensor was mounted on a **pan-tilt** unit facing a monitor displaying static MNIST digits.
2. **Monitor**
   -screen displaying one MNIST digit at given time
3. **Motion/ saccades**
   - performs three tiny saccades while looking at a static MNIST digit on a monitor
4. **Duration**
   -each digit recorded for ≈ **300 ms** (three saccades of 100 ms each)



“Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades,” Orchard et al., 2015