Continuous Attractor Neural Networks

刘潇 2021.08.12

Want to Build a Brain?

Some Bad News:

- We're still in the early days of neural computation.
- Our theories of brain function are vague and wrong.

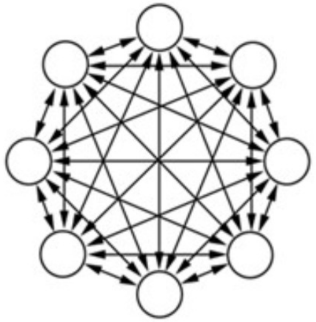
The Misunderstood Brain

- We know a lot about what makes neurons fire. We know a good deal about wiring patterns.
- We know only a little about how information is represented in neural tissue.
- Where are the “noun phrase” cells in the brain?
- We know almost nothing about how information is processed.
- This course explores what we do know. There is progress every month.
- It's an exciting time to be a computational neuroscientist.

Attractor Neural Networks

- Networks of various types/structures, formed by large numbers of neurons, are the substrate of brain functions.
- The brain carries out computation by updating network states in response to external inputs.
- The stationary states, i.e., attractors, of networks encode the stimulus information.

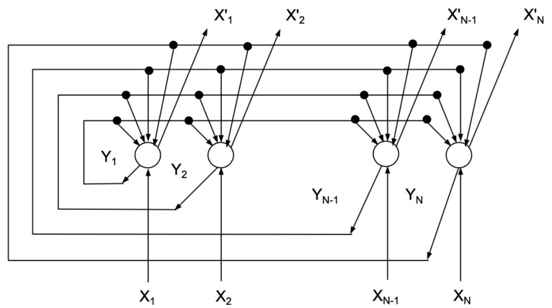
Hopfield Model



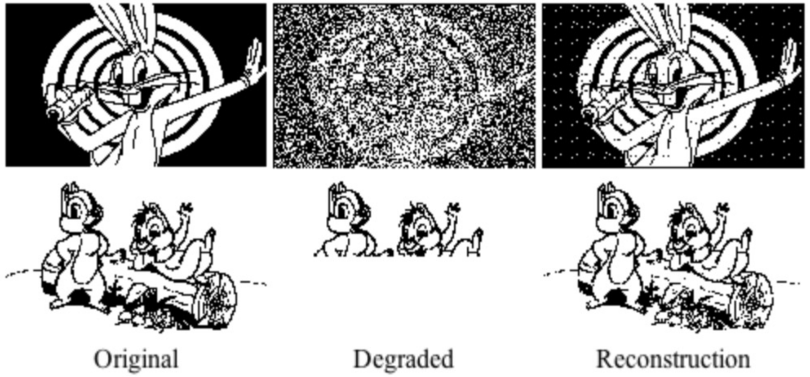
Hopfield Model (S-I Amari. 1972; John J. Hopfield; 1982)

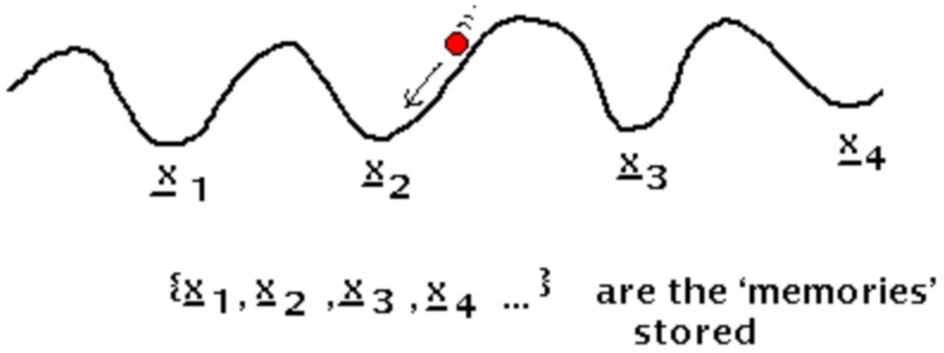
- An attractor model
- The simplest model captures the computation of a network
- Recurrent network & Hebb learning

    Cells that fire together, wire together

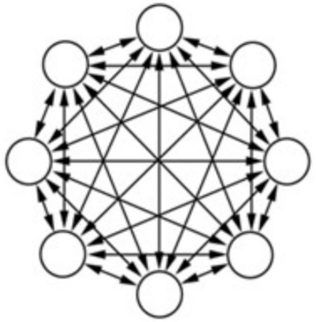


- Associative memory



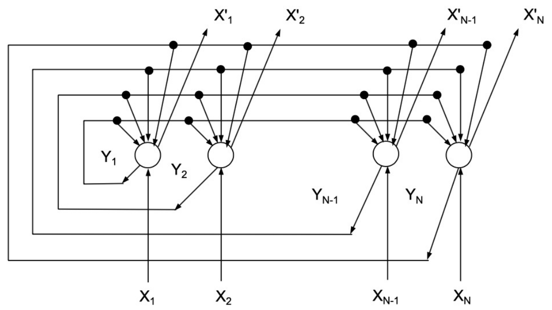


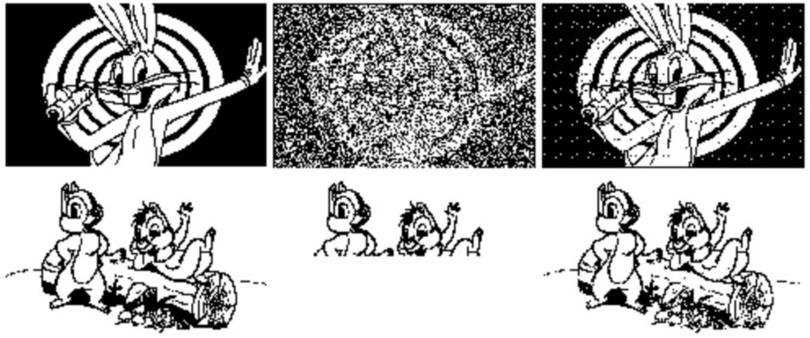
Hopfield Model



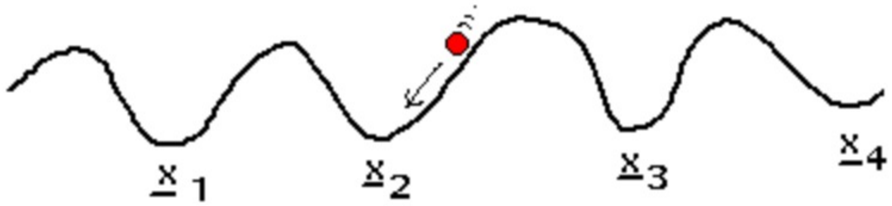
Hopfield Model (S-I Amari. 1972; John J. Hopfield; 1982)

- Recurrent network composed of a set of N neurons x_i
- **Network state** → the states of all the neurons
- Every neuron is connected with all others w_{ij}
- Connections are symmetric, i.e., for all I and j, $w_{ij} = w_{ji}$
- A set of p patterns stored in it



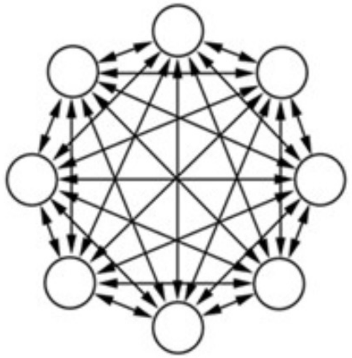


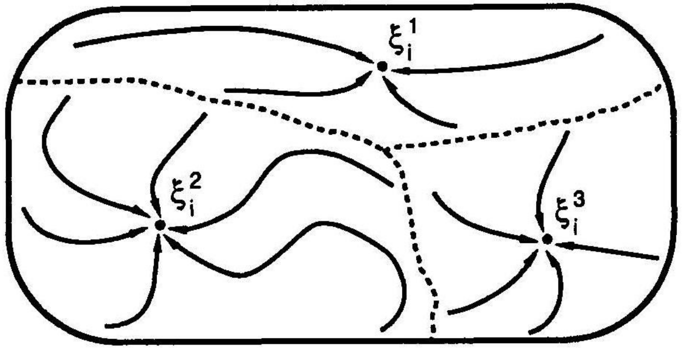
Original Degraded Reconstruction



$\{\underline{x}_1, \underline{x}_2, \underline{x}_3, \underline{x}_4 \ldots\}$ are the 'memories' stored

Associative Memories





- Network state for N neurons

$$x_i, \forall i \in \{1,2,\dots,N\}, x_i = \pm 1$$

- Memory pattern

$$\xi_i^\mu, \forall i \in \{1,2,\dots,N\}, \mu \in \{1,2,\dots,P\} \; and \; \xi_i^\mu = \pm 1$$

- Hebbian learning rule

$$w_{i,j} = \begin{cases} \frac{1}{P}\sum_{\mu=1}^{P} \xi_i^\mu \xi_j^\mu, & i \neq j; \\ 0, & i = j. \end{cases}$$

- Network Dynamic

$$h_i(t+1) = 1/N \sum_{j=1}^{N} w_{i,j} x_j(t)$$

$$x_i(t+1) = \text{sign}\left(h_i(t+1)\right)$$

- Update rule
  Synchronous or asynchronous

Implement the Hopfield

```python
# load img as pattern
img = rgb2gray(mpimg.imread('./hopfield.jpg'))
size_w, size_h = img.shape
xi = (img.reshape([size_w * size_h, 1]) - 0.5) * 2

show_img(xi,size_w,size_h)
```

Image size 407*406

Memory pattern ξ^0 → vector (407*406, 1)

** TIPS ** $\xi_i^0 = \pm 1$

Number of neuro N = 406*407

Implement the Hopfield

Weights

$$w_{i,j} = \begin{cases} \frac{1}{P}\xi_i^0 \xi_j^0, & i \neq j; \\ 0, & i = j. \end{cases}$$

```
# generate weight using Hebb
weight = np.dot(xi, xi.T)
weight = weight - np.diag(weight)
```

Image size 407*406

Memory pattern $\xi^0 \rightarrow$ vector (407*406, 1)

** TIPS ** $\xi_i^0 = \pm 1$

Number of neuro N = 406*407

Implement the Hopfield

Weights

$$w_{i,j} = \begin{cases} \frac{1}{P}\xi_i^0\xi_j^0, & i \neq j; \\ 0, & i = j. \end{cases}$$

Network Dynamic

$$h_i(t+1) = 1/N \sum_{j=1}^{N} w_{i,j} x_j(t)$$

$$x_i(t+1) = \mathrm{sign}\big(h_i(t+1)\big)$$

Image size 407*406

Memory pattern $\xi^0 \rightarrow$ vector (407*406, 1)

** TIPS ** $\xi_i^0 = \pm 1$

Number of neuro N = 406*407

```
### synchronous update
for t in range(2):
    h = np.dot(weight, x)
    x = np.sign(h)
    show_img(x, size_w, size_h)
```

Associative Memories

Weights

$$w_{i,j} = \begin{cases} \frac{1}{P}\xi_i^0 \xi_j^0, & i \neq j; \\ 0, & i = j. \end{cases}$$

Network Dynamic

$$h_i(t+1) = 1/N \sum_{j=1}^{N} w_{i,j} x_j(t)$$

$$x_i(t+1) = \mathrm{sign}\left(h_i(t+1)\right)$$

Input $\mathrm{x}_0 = \xi^0 + \text{noise}$

Image size 407*406

Memory pattern $\xi^0 \rightarrow$ vector (407*406, 1)

** TIPS ** $\xi_i^0 = \pm 1$

Number of neuro N = 406*407

Associative Memories

Weights

$$w_{i,j} = \begin{cases} \frac{1}{P}\xi_i^0 \xi_j^0, & i \neq j; \\ 0, & i = j. \end{cases}$$

Network Dynamic

$$h_i(t+1) = 1/N \sum_{j=1}^{N} w_{i,j} x_j(t)$$

$$x_i(t+1) = \text{sign}\left(h_i(t+1)\right)$$

Input $x_0 = \xi^0$ + noise, zeros, crop ξ^0

Image size 407*406

Memory pattern $\xi^0 \rightarrow$ vector (407*406, 1)

** TIPS ** $\xi_i^0 = \pm 1$

Number of neuro N = 406*407

Implement the Hopfield

Weights

$$w_{i,j} = \begin{cases} \frac{1}{P}\xi_i^0 \xi_j^0, & i \neq j; \\ 0, & i = j. \end{cases}$$

Network Dynamic

$$h_i(t+1) = 1/N \sum_{j=1}^{N} w_{i,j} x_j(t)$$

$$x_i(t+1) = \mathrm{sign}\left(h_i(t+1)\right)$$

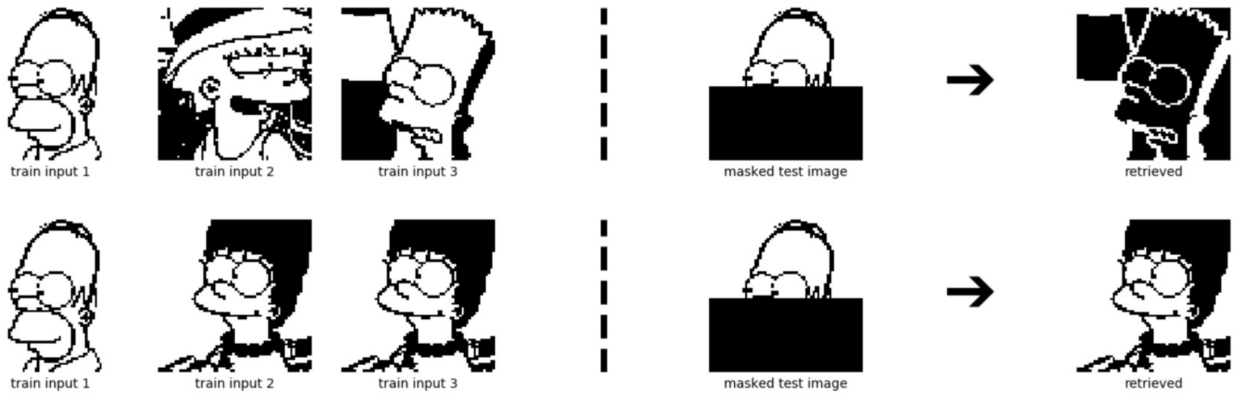
Input $x_0 = \xi^0 +$ noise, zeros, crop ξ^0

Image size 407*406

Memory pattern $\xi^0 \rightarrow$ vector (407*406, 1)

** TIPS ** $\xi_i^0 = \pm 1$

Number of neuro N = 406*407



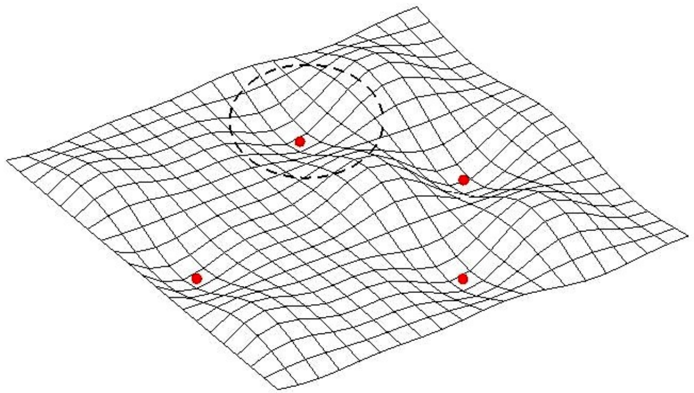
The Energy Function

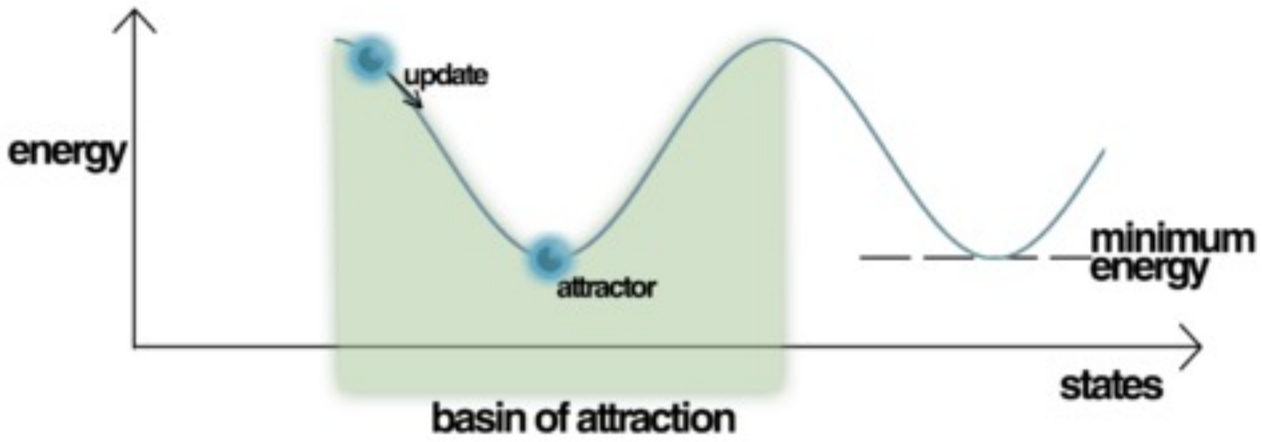
$$E = -\frac{1}{2}\boldsymbol{x}^T\boldsymbol{W}\boldsymbol{x} = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}x_ix_j$$

$$\Delta E = -1/2\sum_{ij} \Delta w_{uij} x_{ui} x_{uj}$$

$$= -1/2\sum_{ij} x_{ui} x_{uj} x_{ui} x_{uj}$$

$$= -1/2N^2$$





What’s more ?

- Storage capacity
- Correlated patterns
- Multi-layer Hopfield
- …

What’s more ?

- Storage capacity
- Correlated patterns
- Multi-layer Hopfield
- …

- Modern Hopfield Networks (aka Dense Associative Memories)

Large Associative Memory Problem in Neurobiology and Machine Learning

Dmitry Krotov
MIT-IBM Watson AI Lab
IBM Research
krotov@ibm.com

John Hopfield
Princeton Neuroscience Institute
Princeton University
hopfield@princeton.edu

Abstract

Dense Associative Memories or modern Hopfield networks permit storage and reliable retrieval of an exponentially large (in the dimension of feature space) number of memories. At the same time, their naive implementation is non-biological, since it seemingly requires the existence of many-body synaptic junctions between the neurons. We show that these models are effective descriptions of a more microscopic (written in terms of biological degrees of freedom) theory that has additional (hidden) neurons and only requires two-body interactions between them. For this reason our proposed microscopic theory is a valid model of large associative memory with a degree of biological plausibility. The dynamics of our network and its reduced dimensional equivalent both minimize energy (Lyapunov) functions. When certain dynamical variables (hidden neurons) are integrated out from our microscopic theory, one can recover many of the models that were previously discussed in the literature, e.g. the model presented in “Hopfield Networks is All You Need” paper. We also provide an alternative derivation of the energy function and the update rule proposed in the aforementioned paper and clarify the relationships between various models of this class.

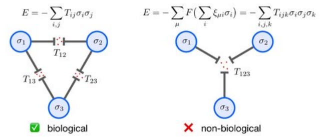


Figure 1: Two binary networks consisting of three neurons $\sigma_1, \sigma_2, \sigma_3 = \{\pm 1\}$. On the left is the classical Hopfield network [1] with the matrix $T_{ij} = \sum_\mu \xi_{\mu i}\xi_{\mu j}$ being the outer product of memory vectors (see section 2 for the definitions of notations). In this case the matrix T_{ij} is interpreted as a matrix of synaptic connections between cells i and j. On the right is a Dense Associative Memory network of [2] with cubic interaction term $F(x) = x^3$. In this case the corresponding tensor $T_{ijk} = \sum_\mu \xi_{\mu i}\xi_{\mu j}\xi_{\mu k}$ has three indices, thus cannot be interpreted as a biological synapse, which can only connect two cells.

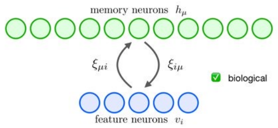
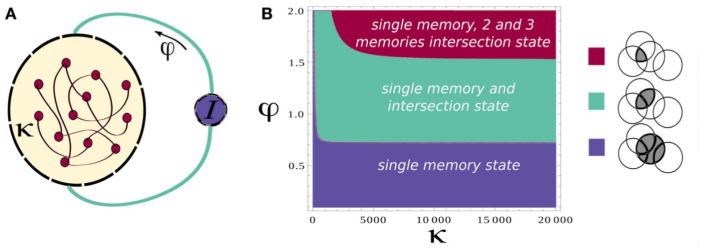


Figure 2: Proposed microscopic theory in which feature neurons are coupled to memory (hidden) neurons. By solving for the activity of the hidden neurons, one can obtain an effective theory that reduces to some of the models previously discussed in the literature. For example Dense Associative Memories [2,3] or modern Hopfield networks of [6].

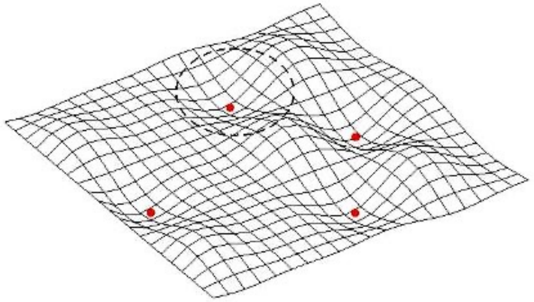
https://ml-jku.github.io/hopfield-layers/

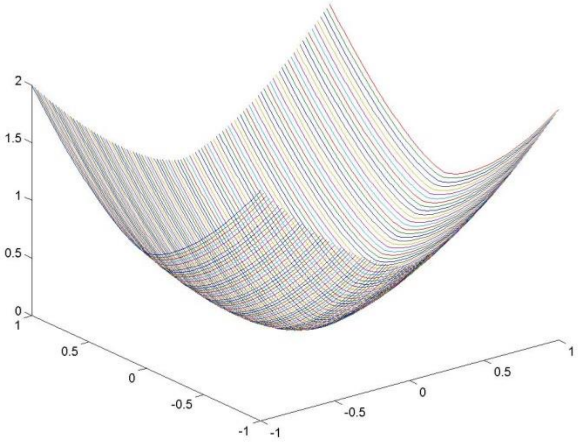
- Transition between items are determined by similarities in their long-term memory representations

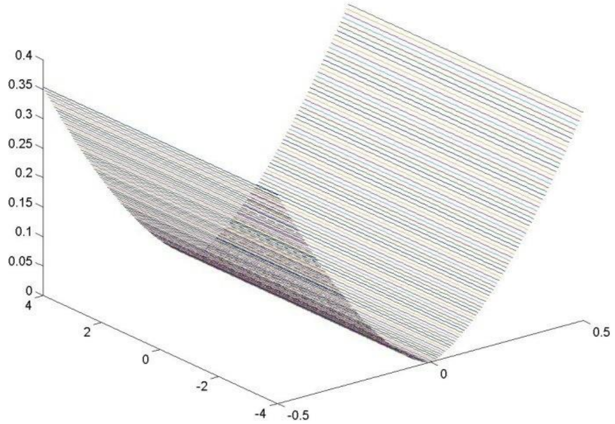


Recanatesi et al. 2015

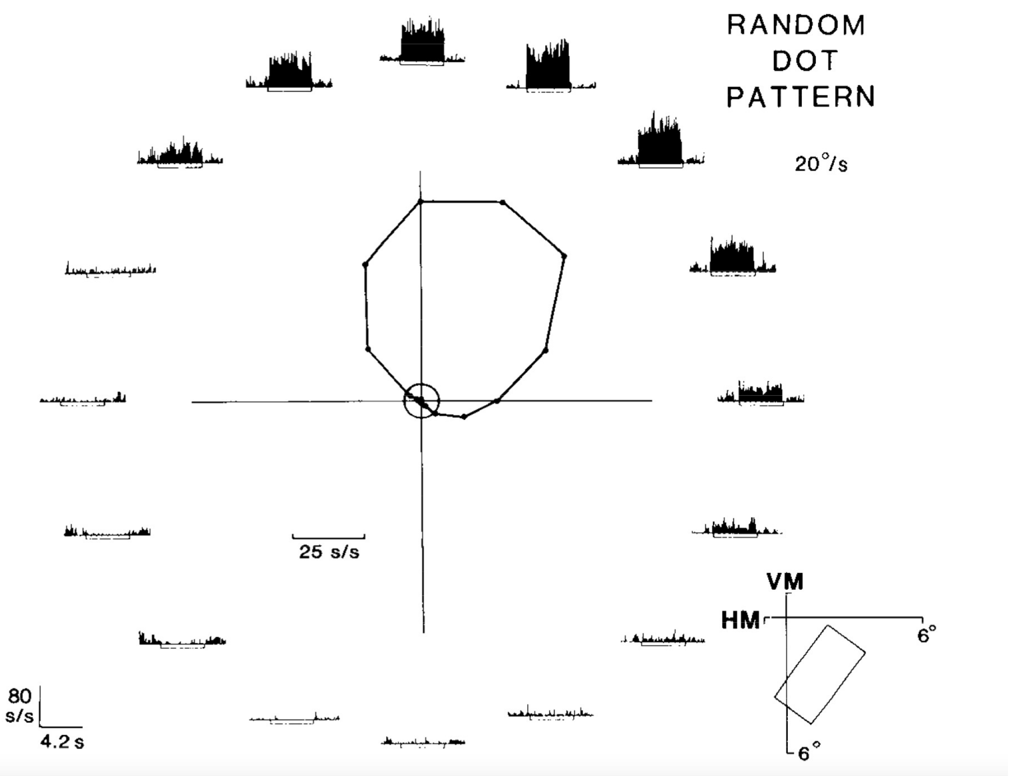
Continuous Attractor Neural Networks





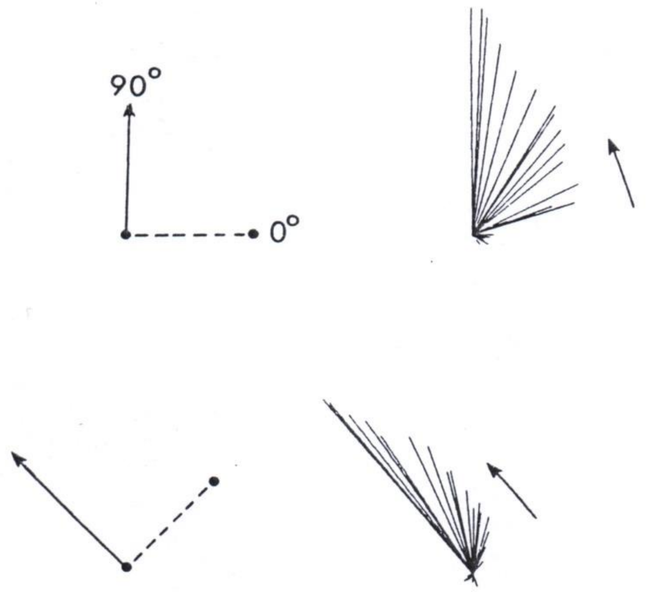


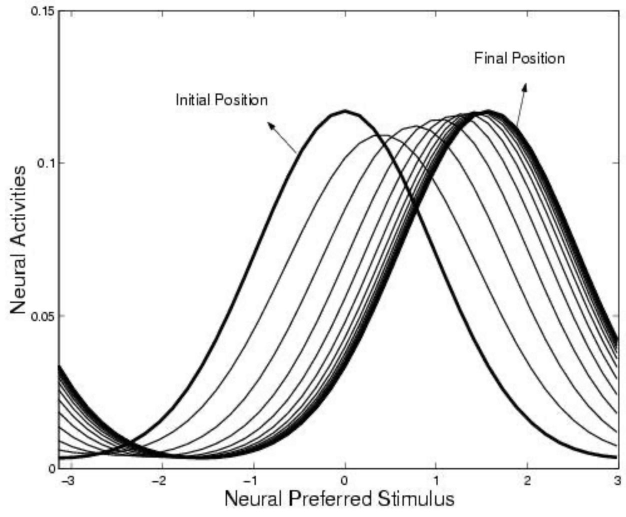
Neural Encoding of Motion Direction



Example of direction tuning of a typical MT neuron.

Mental rotation in the premotor cortex



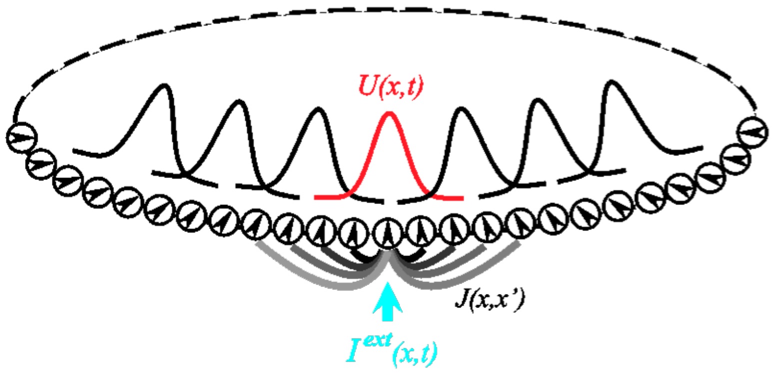


A. Georgopoulos et al., science, 1993

Continuous Attractor Neural Network (CANN)

Key Structure:

- Bell-shaped recurrent connection strength
- Translation-invariant connection pattern
- Global divisive normalization



Key Mathematic Properties:

- Recurrent pos..ve-feedback generates attractor, retaining input information
- Divisive normalization avoids exploration
- Translation-invariance ensures many attractors

References: 1. Amari, 1977, 2. Ben-Yishai et al., 1995, 3. Zhang, 1996, 4. Seung, 1996, 5. Deneve et al, 1999, 6. Wu et al, 2002, 2005, 2008, 2010, 2012

The mathematical formulation

$$\tau \frac{\partial U(x,t)}{\partial t} = -U(x,t) + \rho \int d\,x' J(x-x') r(x',t) + I^{ext}(x,t)$$

$$r(x,t) = \frac{U(x,t)^2}{1 + k\rho \int d\,x' U(x',t)^2}$$

$$\tau \frac{du(x,t)}{dt} = -u(x,t) + \rho \sum_{x'} J(x,x') r(x',t) dx' + I_{ext}$$

$$r(x,t) = \frac{u(x,t)^2}{1 + k\rho \sum_{x'} u(x',t)^2 dx'}$$

$$J(x-x') = \frac{J}{\sqrt{2\pi a}} \exp\left[-\frac{(x-x')^2}{2a^2}\right]$$

$$I_{ext} = A \exp\left[-\frac{\left|x - z(t)\right|^2}{4a^2}\right]$$

- u(x, t) : the synaptic input at time t of the neuron that preferred stimulus at location x
- r(x, t) : the corresponding firing rate
- ρ: the neural density
- τ: the synaptic time constant
- $x \in (-\pi,\ \pi)$

Implement the CANN with BrainPy

Three tasks:

- Population coding
- Template matching
- Smooth tracking

Implement the CANN with BrainPy

```python
class CANN1D(bp.NeuGroup):
    target_backend = ['numpy', 'numba']

    def __init__(self, num, **kwargs):
        super(CANN1D, self).__init__(size=num, **kwargs)

        # parameters
        self.tau = 1.  # The synaptic time constant
        self.k = 8.1  # Degree of the rescaled inhibition
        self.a = 0.5  # Half-width of the range of excitatory connections
        self.A = 10.  # Magnitude of the external input
        self.J0 = 4.  # maximum connection value
        self.z_range = 2 * np.pi  # feature space

        # variables
        self.u = np.zeros(num)  # variable u
        self.input = np.zeros(num)  # external input
        self.x = np.linspace(-np.pi, np.pi, num)  # The encoded features

        # The connection matrix
        self.conn_mat = self.make_conn(self.x)
```

Necessary parameters

Function for the distance in the ring

```python
    def dist(self, d):
        d = np.remainder(d, self.z_range)
        d = np.where(d > 0.5 * self.z_range, d - self.z_range, d)
        return d
```

Function for the connection

$$J(x,x') = \frac{1}{\sqrt{2\pi}a}\exp(-\frac{|x-x'|^2}{2a^2})$$

```python
    def make_conn(self, x):
        assert np.ndim(x) == 1
        x_left = np.reshape(x, (-1, 1))
        d = self.dist(x_left - x)
        Jxx = self.J0 * np.exp(-0.5 * np.square(d / self.a)) / \
              (np.sqrt(2 * np.pi) * self.a)
        return Jxx
```

Function for stimulus

```python
    def get_stimulus_by_pos(self, pos):
        return self.A * np.exp(-0.25 * np.square(self.dist(self.x - pos) / self.a))
```

$$I_{ext} = A\exp\left[-\frac{|x-z(t)|^2}{4a^2}\right]$$

Update Function

```
@staticmethod
@bp.odeint(method='rk4', dt=0.05)
def int_u(u, t, conn, k, tau, Iext):
    r1 = np.square(u)
    r2 = 1.0 + k * np.sum(r1)
    r = r1 / r2
    Irec = np.dot(conn, r)
    du = (-u + Irec + Iext) / tau
    return du

def update(self, _t):
    self.u = self.int_u(self.u, _t, self.conn_mat, self.k, self.tau, self.input)
    self.input[:] = 0.
```

$$r(x,t) = \frac{u(x,t)^2}{1 + k\rho \sum_{x'} u(x',t)^2 dx'}$$

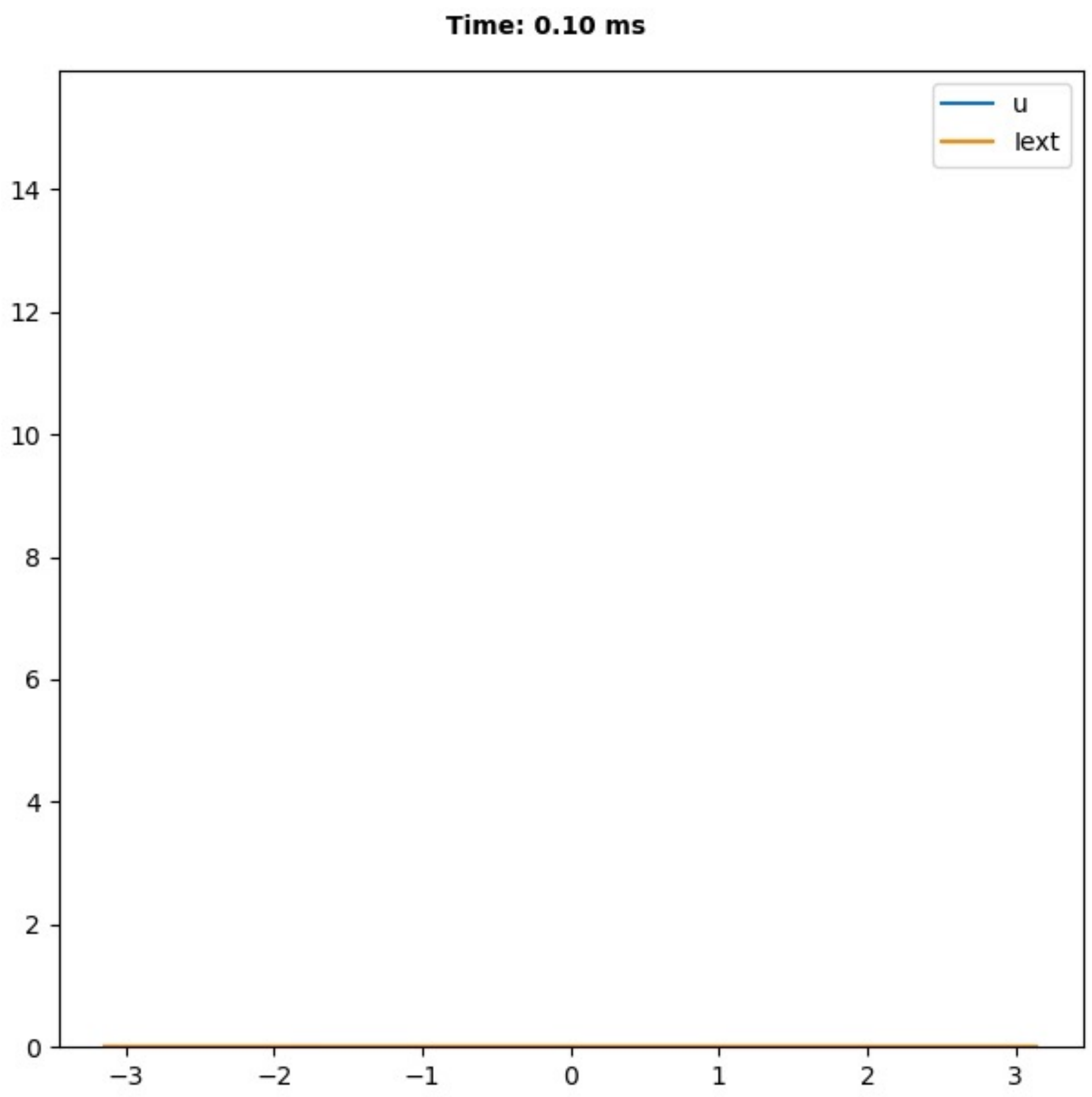
$$\tau \frac{du(x,t)}{dt} = -u(x,t) + \rho \sum_{x'} J(x,x') r(x',t) dx' + I_{ext}$$

Task 1: population coding

```
cann = CANN1D(num=512, monitors=['u'])
cann.k = 0.1

I1 = cann.get_stimulus_by_pos(0.)
Iext, duration = bp.inputs.constant_input([(0., 1.), (I1, 8.), (0., 8.)])
cann.run(duration=duration, inputs=('input', Iext))

bp.visualize.animate_1D(
    dynamical_vars=[{'ys': cann.mon.u, 'xs': cann.x, 'legend': 'u'},
                    {'ys': Iext, 'xs': cann.x, 'legend': 'Iext'}],
    frame_step=1,
    frame_delay=100,
    show=True,
    # save_path='../../images/CANN-encoding.gif'
)
```

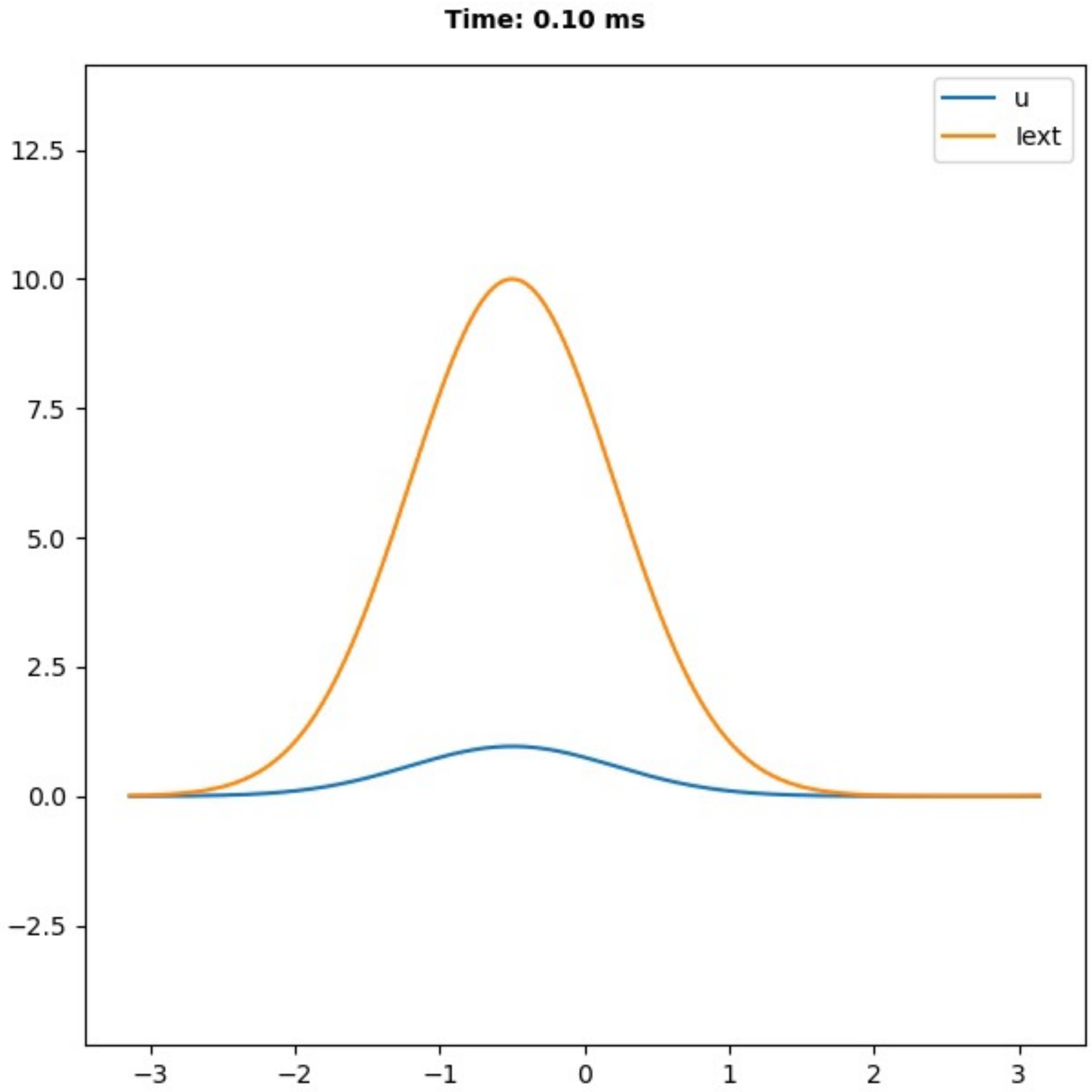

Time: 0.10 ms
u
Iext
14
12
10
8
6
4
2
0
−3
−2
−1
0
1
2
3

Task 2: template matching

```python
cann = CANN1D(num=512, monitors=['u'])
cann.k = 8.1

dur1, dur2, dur3 = 10., 30., 0.
num1 = int(dur1 / bp.backend.get_dt())
num2 = int(dur2 / bp.backend.get_dt())
num3 = int(dur3 / bp.backend.get_dt())
Iext = np.zeros((num1 + num2 + num3,) + cann.size)
Iext[:num1] = cann.get_stimulus_by_pos(0.5)
Iext[num1:num1 + num2] = cann.get_stimulus_by_pos(0.)
Iext[num1:num1 + num2] += 0.1 * cann.A * np.random.randn(num2, *cann.size)
cann.run(duration=dur1 + dur2 + dur3, inputs=('input', Iext))

bp.visualize.animate_1D(
    dynamical_vars=[{'ys': cann.mon.u, 'xs': cann.x, 'legend': 'u'},
                    {'ys': Iext, 'xs': cann.x, 'legend': 'Iext'}],
    frame_step=5,
    frame_delay=50,
    show=True,
    # save_path='../../images/CANN-decoding.gif'
)
```

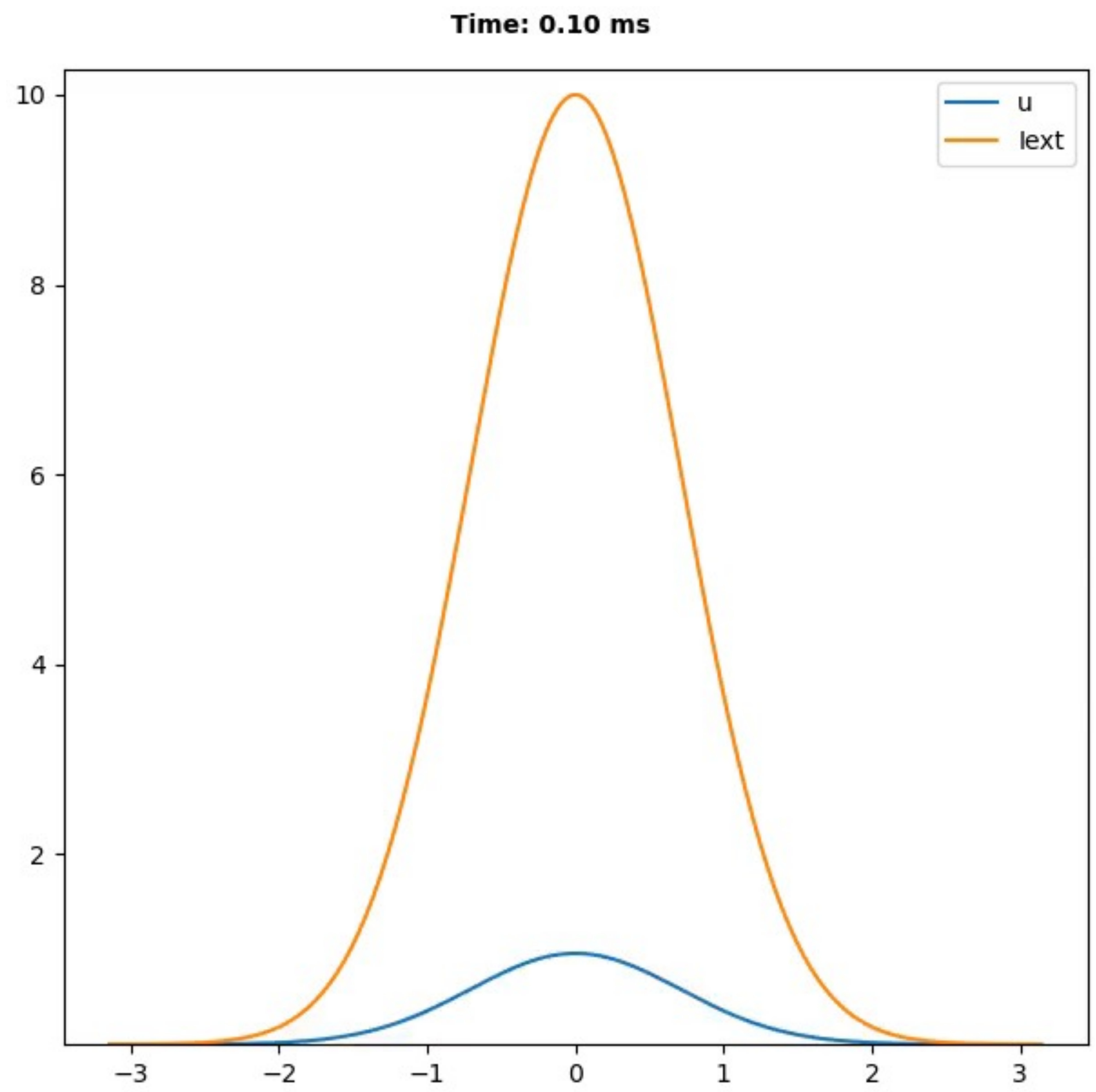

Time: 0.10 ms
u
Iext
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−3
−2
−1
0
1
2
3

Task 3: smooth tracking

```
cann = CANN1D(num=512, monitors=['u'])
cann.k = 8.1

dur1, dur2, dur3 = 20., 20., 20.
num1 = int(dur1 / bp.backend.get_dt())
num2 = int(dur2 / bp.backend.get_dt())
num3 = int(dur3 / bp.backend.get_dt())
position = np.zeros(num1 + num2 + num3)
position[num1: num1 + num2] = np.linspace(0., 12., num2)
position[num1 + num2:] = 12.
position = position.reshape((-1, 1))
Iext = cann.get_stimulus_by_pos(position)
cann.run(duration=dur1 + dur2 + dur3, inputs=('input', Iext))

bp.visualize.animate_1D(
    dynamical_vars=[{'ys': cann.mon.u, 'xs': cann.x, 'legend': 'u'},
                    {'ys': Iext, 'xs': cann.x, 'legend': 'Iext'}],
    frame_step=5,
    frame_delay=50,
    show=True,
    # save_path='../../images/CANN-tracking.gif'
)
```


Time: 0.10 ms
u
Iext
10
8
6
4
2
−3
−2
−1
0
1
2
3