BindsNET: An ML-oriented spiking networks library built with PyTorch

BindsNET
https://github.com/Hananel-Hazan/bindsnet

Daniel J. Saunders, Hananel Hazan, Hassaan Khan, Hava T. Siegelmann, Robert Kozma

What is BindsNET?

- Clock-driven *spiking neural networks* (SNN) simulator
- Oriented towards ML + RL
- User-friendly syntax + fast prototyping
- *Functional* (rather than *exact*) dynamics
- Run on CPUs, GPUs, or both
- Inherits performance + functionality of PyTorch

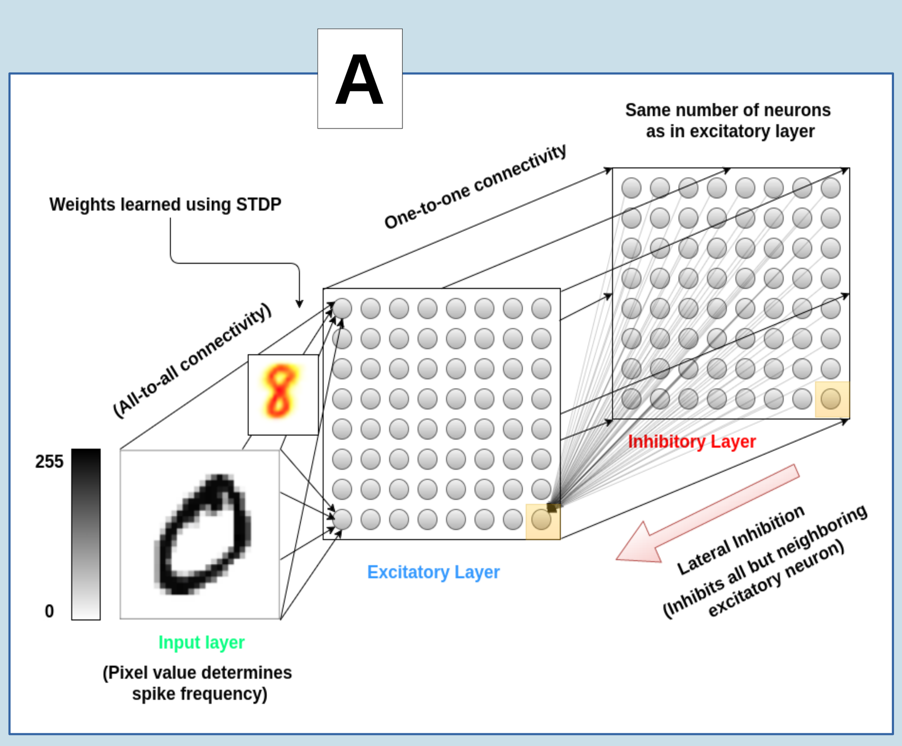
Why spiking neurons?

- More *biologically plausible* than ANN neurons
- Useful for modeling neuronal circuits + brains
- Speedup + power reduction on *neuromorphic chips*
- Naturally incorporates time by integrating input
- Weight updates *as needed;* ANN updates every step

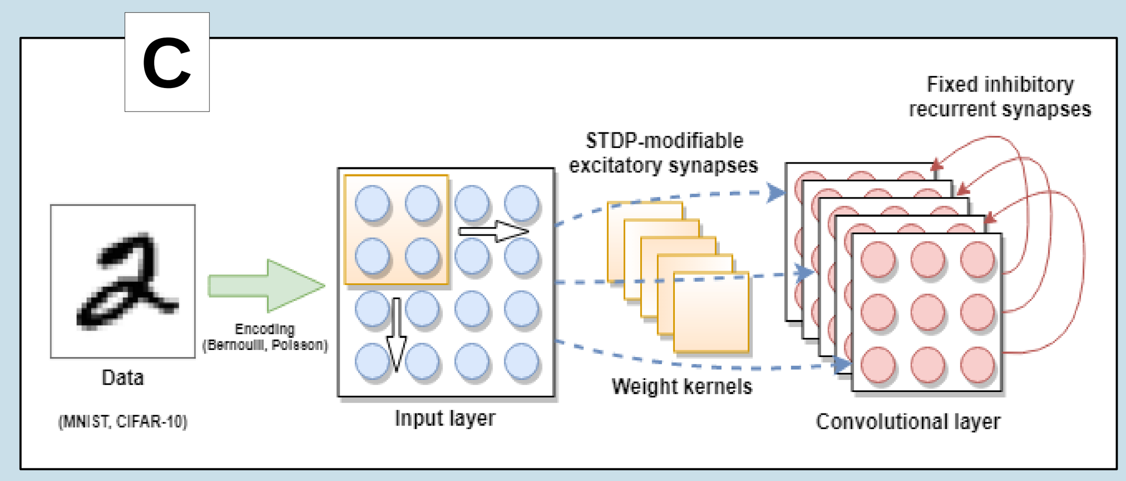
What's in the library?

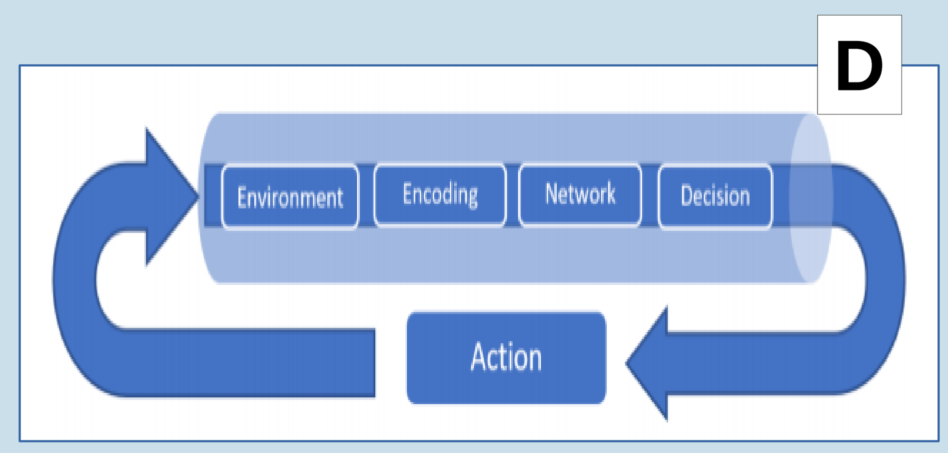
- **Network**: Central data structure; handles simulation of various SNN components
- **Learning** rules: Hebbian learning, STDP, reward-modulation; local updates
- **Pipeline**: Coordinates network + environment
- **ML datasets + RL environments**

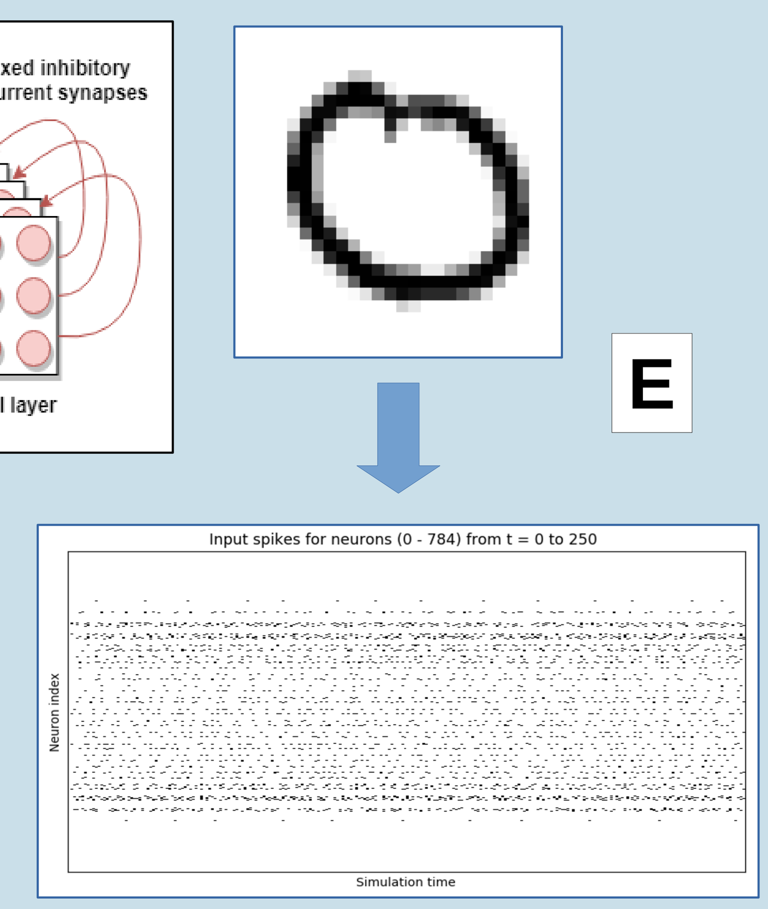
BindsNET	Description
network	Network object + network components + saving / loading functions
nodes	Groups of neurons of arbitrary size and dimensionality
topology	Different types of connectivity between groups of neurons
monitors	Record time-varying state variables of network components (spikes, voltage, …)
environment	Reinforcement learning environments (OpenAI gym and dataset wrappers)
datasets	Downloading, pre-processing, and iteration over popular machine learning datasets
encoding	Conversion of arbitrary data into binary spikes for SNN input
learning	Methods for learning the parameters of connection (topology) objects
pipeline	Contains Pipeline object for coord. of network + environment + action + encoding
action	Functions for mapping network activity to actions in an environment
evaluation	Evaluation of spiking neural networks as machine learning models
analysis	Tools for assessing state and evolution of network component variables
plotting	Online (during simulation) plotting functions (spikes, voltages, weights, …)
visualization	Offline (after simulation) plotting functions (spikes, voltages, weights, …)
models	Network architectures from the spiking neural networks literature




```python
import torch

from bindsnet.network import Network
from bindsnet.network import nodes, topology, monitors

n_input, n_output, time = 100, 100, 1000

# Create network object.
network = Network()

# Create input and output groups of neurons.
input_group = nodes.Input(n=n_input)  # 100 input nodes.
output_group = nodes.LIFNodes(n=n_output)  # 100 output nodes.
network.add_layer(input_group, name='input')
network.add_layer(output_group, name='output')

# Input -> output connection.
# Unit Gaussian feed-forward weights.
w = torch.randn(n_input, n_output)
forward_conn = topology.Connection(input_group, output_group, w=w)

# Output -> output connection.
# Random, inhibitory recurrent weights.
w = torch.bernoulli(torch.rand(n_output, n_output)) - torch.diag(torch.ones(n_output))
recurrent_conn = topology.Connection(output_group, output_group, w=w)
network.add_connection(forward_conn, source='input', target='output')
network.add_connection(recurrent_conn, source='output', target='output')

# Monitor input and output spikes during the simulation.
for l in network.layers:
    monitor = monitors.Monitor(network.layers[l], state_vars=['s'], time=time)
    network.add_monitor(monitor, name=l)

# Create input ~ Bernoulli(0.05) for 1,000 timesteps.
inpts = {'input': torch.bernoulli(0.05 * torch.rand(time, n_input))}

# Run network simulation for 1,000 timesteps and retrieve spikes.
network.run(inpts=inpts, time=time)
spikes = {l: network.monitors[l].get('s') for l in network.layers}
```
B







A: Example SNN architecture; **B**: Example network building + simulation script; **C**: Two-layer convolutional SNN; **D**: Schematic of Pipeline object; **E**: Poisson encoding of MNIST digit for 250 timesteps

How is PyTorch used?

- **torch.Tensor** object: Linear algebra + tensor ops
- **torch.nn** module: Advanced network operations
- **torch.distributions** module: Generating spike data
- **torch.save, load**: Save / load params to / from disk
- **torchvision.datasets**: Planned integration!

ML + RL approach

- **Unsupervised**: Hebbian / associational rules
- **Supervised**: Force certain neurons to spike
- **RL**: Reward signal modulates learning rules
- *Competitive* inhibitory connections
- *Cooperative* excitatory connections

DARPA
UMASS CS SCHOOL OF COMPUTER SCIENCE
BINDS BIOLOGICALLY INSPIRED NEURAL AND DYNAMICAL SYSTEMS LAB